Foxtable 2019 - 人人都能开发管理软件

Table of Contents

1. 关于 Foxtable 15
1.1 关于 Foxtable 15
1.2 关于帮助文件 15
1.3 关于网络支持 16
1.4 关于易表 16
1.5 功能概览 16
1.5.1 关于功能概览 16
1.5.2 轻松录入 17
1.5.3 轻松统计 26
1.5.4 轻松报表 40
1.5.5 轻松查询 45
1.5.6 轻松开发 53
1.5.7 移动开发 60
1.5.8 其他特色 64
2. 关于视频教程 68
3. 使用指南 68
3.1 关于菜单 68
3.2 关于插图 70
3.3 关于示例文件 71
3.4 项目相关 72
3.4.1 新建项目 72
3.4.2 压缩项目 73
3.4.3 备份项目 74
3.4.4 锁定项目 74
3.4.5 项目信息 75
3.4.6 项目属性 75
3.4.7 表标题设置 78
3.4.8 初始化项目 80
3.5 表相关 81
3.5.1 基本操作 81
3.5.2 表样式 95
3.6 列相关 100
3.6.1 基本操作 100
3.6.2 列属性 111
3.6.3 扩展列类型 145
3.6.4 使用列窗口 163
3.7 关联表 171
3.7.1 本章示例文件 171
3.7.2 关于表间关联 172
3.7.3 设置表间关联 172
3.7.4 父表和子表 174

3.7.5 自动合计子表 175
3.7.6 关联选项 176
3.7.7 关联表生成模式 177
3.7.8 双向生成的优势 180
3.7.9 双向生成的劣势 181
3.7.10 多级关联 182
3.7.11 同表关联 182
3.7.12 多个关联列 182
3.7.13 在关联表增加行 183
3.7.14 关联表默认位置 184
3.7.15 设置关联表窗口 184
3.7.16 打开关联表窗口 186
3.7.17 关联表的列位置 186
3.8 表达式 186
3.8.1 本章示例文件 186
3.8.2 增加表达式列 186
3.8.3 修改表达式 188
3.8.4 运算符与函数 188
3.8.5 表达式不计算？ 191
3.8.6 一定要注意列类型 192
3.8.7 关于方括号 192
3.8.8 关于空值 193
3.8.9 聚合函数 193
3.8.10 统计子表数据 194
3.8.11 引用父表数据 194
3.8.12 表达式的综合示例一 195
3.8.13 表达式的综合示例二 196
3.8.14 时段列与表达式 198
3.8.15 有条件统计子表数据 198
3.8.16 不重复编号的生成 199
3.8.17 自动编号与关联 200
3.8.18 锁定标志列 201
3.8.19 数据转换 201
3.8.20 如何实现四舍五入 202
3.8.21 如何计算年月日 202
3.8.22 计算功能太弱? 203
3.8.23 数据无法保存? 203
3.8.24 美化自动编号 204
3.8.25 表达式列和数据列的相互转换 205
3.9 行相关 206
3.9.1 增加行 206
3.9.2 插入行 206
3.9.3 克隆行 207
3.9.4 删除行 207

3.9.5 锁定行 208
3.9.6 排序行 208
3.9.7 调整行位置 209
3.9.8 筛选行 210
3.9.9 行视图 229
3.9.10 同步数据 230
3.9.11 行高设置 230
3.9.12 行号的特殊意义 232
3.9.13 快捷菜单中的行操作命令 232
3.10 编辑数据 233
3.10.1 在单元格中输入数据 233
3.10.2 列表项目 234
3.10.3 目录树列表 234
3.10.4 输入日期 235
3.10.5 输入数值 236
3.10.6 代码项目 236
3.10.7 查找 239
3.10.8 替换 240
3.10.9 数据填充 241
3.10.10 快速清零 242
3.10.11 插入符号 243
3.10.12 长文本的编辑 243
3.10.13 在单元格换行输入 244
3.10.14 记录窗口 244
3.10.15 计算器 247
3.10.16 高级功能 248
3.10.17 快速移动 252
3.10.18 中文输入 252
3.10.19 回车键的移动方向 252
3.10.20 Tab 键的移动方向 253
3.10.21 编辑模式 253
3.10.22 选定数据或单元格 253
3.11 统计数据 254
3.11.1 本章示例文件 254
3.11.2 汇总模式 255
3.11.3 合计模式 272
3.11.4 合并模式 273
3.11.5 分组统计 276
3.11.6 交叉统计 298
3.11.7 快速统计 320
3.11.8 统计精度 321
3.11.9 统计条件 322
3.11.10 统计后台数据 323
3.11.11 自动计算 324

3.11.12 筛选树与数据统计 325
3.11.13 图示统计数据 326
3.12 打印 343
3.12.1 打印表 343
3.12.2 批量打印条形码 348
3.12.3 中国移动的二维码应用 349
3.13 Excel 报表 350
3.13.1 关于 Excel 报表 350
3.13.2 本章示例文件 351
3.13.3 Excel 制表快速入门 352
3.13.4 细节区和数据引用 353
3.13.5 设计报表模板 354
3.13.6 添加报表模板 355
3.13.7 生成 Excel 报表 356
3.13.8 符号要半角的 357
3.13.9 多行细节区 358
3.13.10 指定有效区域 360
3.13.11 设置打印范围 361
3.13.12 关于日期列 362
3.13.13 换页控制 363
3.13.14 引用图片列 363
3.13.15 按比例缩放图片 364
3.13.16 使用表达式 365
3.13.17 设置单元格公式 367
3.13.18 打印关联表 368
3.13.19 统计关联表 369
3.13.20 打印多个关联表 370
3.13.21 自动添加空行 370
3.13.22 设置打印顺序 371
3.13.23 根据多列排序 372
3.13.24 指定页首行 372
3.13.25 页尾行与分页汇总 373
3.13.26 汇总已经打印的行 374
3.13.27 分组统计报表 374
3.13.28 多个分组列 377
3.13.29 引用父表数据 377
3.13.30 直接引用数据 378
3.13.31 直接统计 379
3.13.32 引用顺序号 380
3.13.33 又一个关联表打印示例 380
3.13.34 打印工资条 382
3.13.35 标签打印 382
3.13.36 自动合并单元格 384
3.13.37 引用图片文件 385

3.13.38 再谈表达式计算 386
3.13.39 金额的按位打印 387
3.13.40 再谈有效区域的设置 388
3.13.41 使用条形码字体 388
3.13.42 分割打印 389
3.14 Word 报表 389
3.14.1 关于 Word 报表 389
3.14.2 数据引用说明 390
3.14.3 设计报表模版 392
3.14.4 添加报表模版 393
3.14.5 生成 Word 报表 394
3.14.6 自动换页控制 395
3.14.7 Word 文章套打 396
3.14.8 打印关联子表 397
3.14.9 插入关联子表 399
3.14.10 常用函数介绍 401
3.14.11 符号要半角的 403
3.14.12 报表与表的关系 404
3.14.13 表达式中的运算符冲突 405
3.14.14 Word 报表还是 Excel 报表? 405
3.15 导入导出 408
3.15.1 导出 408
3.15.2 导入 410
3.15.3 合并 413
3.16 用户管理 415
3.16.1 用户管理 415
3.16.2 用户级别 417
3.16.3 默认用户 419
3.16.4 切换用户 420
3.16.5 扩展用户属性 420
3.16.6 关于权限管理 421
3.17 网络与数据源 421
3.17.1 关于数据源 421
3.17.2 关于主键 422
3.17.3 连接外部数据源 424
3.17.4 加载外部数据表 468
3.17.5 使用 Oracle 作为数据源 482
3.17.6 无法连接数据源时 486
3.17.7 隐藏共享文件夹 487
3.17.8 数据源的转换 487
3.17.9 数据类型对照 490
3.17.10 外部数据源与用户管理 491
3.17.11 关于三层架构 493
3.18 动态加载数据 493

3.18.1 控制初始加载量 493
3.18.2 加载树 495
3.18.3 分时段加载 498
3.18.4 加载树可以多选 499
3.18.5 分页加载 499
3.18.6 选择分页列 501
3.18.7 改变加载顺序 501
3.18.8 自动显示加载树 502
3.18.9 后台统计 503
3.18.10 后台筛选 504
3.18.11 分页加载与数据统计 505
3.19 使用查询表 507
3.19.1 增加查询表 507
3.19.2 Select 语句 508
3.19.3 单表查询 509
3.19.4 多表查询 510
3.19.5 指定别名 511
3.19.6 表达式列 512
3.19.7 分组统计 512
3.19.8 刷新结果 514
3.19.9 连接类型 515
3.19.10 选择系统列 516
3.19.11 选择所有列 516
3.19.12 限制加载行数 517
3.19.13 查询表与加载树 518
3.19.14 查询表的替代者 520
3.19.15 用数据表还是查询表? 523
3.20 表视图 523
3.21 网络环境下的数据同步 525
3.22 我还需要继续吗? 525
4. 开发指南 526
4.1 写给初学者 526
4.2 最重要的两章 526
4.3 编程基础 527
4.3.1 灾难恢复 527
4.3.2 编程语言 527
4.3.3 运行示例代码 527
4.3.4 基本概念 528
4.3.5 运算符 545
4.3.6 流程控制语句 548
4.3.7 字符串(String) 558
4.3.8 日期(Date) 561
4.3.9 时段(TimeSpan) 564
4.3.10 常用对话框 566

4.3.11 函数参考 574
4.3.12 数据转换 582
4.3.13 杂项 585
4.4 Foxtable 编程 596
4.4.1 基本类型 596
4.4.2 事件编程 667
4.4.3 导入导出 791
4.4.4 计划管理 802
4.4.5 自定义函数 803
4.4.6 杂谈 806
4.5 统计与查询 839
4.5.1 设置合并模式 839
4.5.2 设置汇总模式 841
4.5.3 设置合计模式 846
4.5.4 分组统计 848
4.5.5 交叉统计 862
4.5.6 直接统计后台数据 875
4.5.7 高级后台统计工具 877
4.5.8 查询表生成器 902
4.5.9 合并多个查询结果 909
4.5.10 指定连接类型 911
4.5.11 综合使用 SQL 语句和统计工具 911
4.5.12 SQLJoinTableBuilder 和统计工具 913
4.5.13 记得设置数据源名称 914
4.5.14 组合统计结果 914
4.5.15 临时表生成器 920
4.5.16 不显示生成的表 922
4.5.17 自动显示合计行 922
4.5.18 数据填充器 923
4.5.19 手工编码填充数据 925
4.5.20 杂谈 925
4.6 Excel 报表 945
4.6.1 本章示例文件 945
4.6.2 基础应用 945
4.6.3 报表编程 983
4.6.4 类型介绍 1002
4.6.5 打印设置 1017
4.6.6 扩展 Excel 报表的功能 1020
4.6.7 标记与事件 1020
4.7 Word 报表 1030
4.7.1 基础应用 1030
4.7.2 编程指南 1048
4.8 菜单设计 1054
4.8.1 主菜单组成 1054

4.8.2 主菜单组件 1055
4.8.3 如何返回系统菜单？ 1059
4.8.4 快速入门 1060
4.8.5 组件参考 1079
4.8.6 Gallery 1109
4.8.7 实战演练 1116
4.8.8 一点改进 1127
4.8.9 关于 TopToolBar 1127
4.8.10 关于 BottomToolBar 1127
4.8.11 关于 TabGroup 1128
4.8.12 菜单和代码编辑器 1129
4.8.13 设计传统菜单 1131
4.9 窗口设计 1133
4.9.1 窗口基础 1133
4.9.2 控件参考 1180
4.9.3 窗口中的 Table 1359
4.9.4 窗口中的菜单 1380
4.9.5 动态增加控件 1401
4.9.6 自定义录入界面 1406
4.9.7 其它 1424
4.10 票据设计 1452
4.10.1 用窗口设计票据 1452
4.10.2 和打印相关的控件属性 1453
4.10.3 页面设置 1454
4.10.4 其它设置 1455
4.10.5 DataFormat 事件和票据打印 1456
4.10.6 实战演练一 1456
4.10.7 实战演练二 1459
4.10.8 实战演练三 1461
4.10.9 统一录入和打印窗口 1461
4.10.10 关于边框打印 1462
4.10.11 隐藏行号列 1462
4.10.12 背景图片和 DPI 1463
4.10.13 打印指定容器中的控件 1463
4.10.14 用复选框模拟单选框 1464
4.11 生成图表 1465
4.11.1 图表基础 1465
4.11.2 图表类型 1478
4.11.3 编程参考 1491
4.11.4 横向生成图表 1515
4.11.5 无法绑定的情况 1516
4.11.6 甘特图 1518
4.11.7 ChartBuilder 1519
4.11.8 为 Excel 报表增加图表功能 1521

4.12 ListView 1522
4.12.1 快速入门 1522
4.12.2 类型参考 1570
4.13 条形码 1588
4.13.1 关于条形码 1588
4.13.2 批量打印条形码 1590
4.13.3 基本属性 1591
4.13.4 常用二维码 1595
4.13.5 BarCode(控件) 1599
4.13.6 BarCodeBuilder 1601
4.13.7 自动生成代码 1602
4.13.8 为 Excel 报表增加条形码功能 1603
4.14 动态加载 1604
4.14.1 动态加载数据 1604
4.14.2 处理后台数据 1632
4.14.3 表的动态加载与卸载 1651
4.14.4 SQL 表达式的语法差异 1656
4.15 SQL 相关 1657
4.15.1 SQL 语言入门 1657
4.15.2 SQL 函数参考 1666
4.15.3 SQLCommad 1676
4.15.4 SQL 应用实例 1687
4.15.5 编写通用代码 1711
4.15.6 关于表达式列 1712
4.15.7 再谈 INNER JOIN 1713
4.15.8 记得设置链接名称 1713
4.15.9 后台数据的更新方式 1713
4.15.10 容易混淆表达式语法 1716
4.16 消息推送 1717
4.16.1 关于消息推送 1717
4.16.2 OpenQQ 1718
4.16.3 UDPClient 1786
4.17 权限管理 1795
4.17.1 关于权限管理 1795
4.17.2 数据管理权限 1796
4.17.3 文件管理权限 1839
4.18 二进制列 1849
4.18.1 关于二进制列 1849
4.18.2 SQLSetValue 1850
4.18.3 SQLGetValue 1851
4.18.4 SQLInsertFile 1851
4.18.5 SQLLoadFile 1852
4.18.6 SQlLoadImage 1853
4.18.7 使用二进制列的简单示例 1853

4.18.8 多文件类型与本地缓存 1857
4.18.9 二进制列的多文件管理 1858
4.19 数据上报 1862
4.19.1 远程数据上报 1862
4.19.2 通过手机上报 1863
4.19.3 用 FTP 上报数据 1868
4.19.4 通过邮件上报数据 1869
4.20 工作流 1870
4.20.1 关于工作流 1870
4.20.2 审批流程的实现 1871
4.20.3 主动通知下一个流程 1872
4.20.4 自定义流程通知窗口 1873
4.20.5 有关知识点 1876
4.20.6 传统编辑方式 1877
4.20.7 独占式编辑 1880
4.20.8 用户间通讯 1884
4.20.9 数据的自动流转之一 1887
4.20.10 数据的自动流转之二 1892
4.20.11 工作流和并发冲突 1901
4.20.12 OpenQQ、工作流与并发冲突 1907
4.21 单元格绘图 1912
4.21.1 在单元格中直接绘图 1912
4.21.2 在单元格绘制按钮 1914
4.21.3 在单元格绘制甘特图 1916
4.22 网络相关 1919
4.22.1 MailSender 1919
4.22.2 使用 QQ 邮箱发送邮件 1922
4.22.3 FTPClient 1923
4.22.4 Network 1932
4.22.5 TryConnectHost 1935
4.22.6 网络状态监测事件 1935
4.22.7 串口通讯 1935
4.23 软件加密 1943
4.23.1 关于配套的 UKey 1943
4.23.2 UKey 参考 1943
4.23.3 启动 UKey 1944
4.23.4 固化加密函数 1944
4.23.5 用户算法函数 1946
4.23.6 软件加密示例 1947
4.23.7 获取 UKey 编号 1948
4.23.8 身份验证示例 1949
4.23.9 设置存储器写密码 1954
4.23.10 存储器的数据读写 1955
4.23.11 软件加密与授权示例 1956

4.23.12 另一种软件加密方式 1959
4.24 协同开发 1961
4.24.1 多人协同开发 1961
4.24.2 指定责任设计者 1961
4.24.3 谁创建谁设计 1961
4.24.4 让开发者退居幕后 1962
4.24.5 只加载自己负责的表 1962
4.24.6 拆分成多个项目设计 1962
4.25 源码保护 1964
4.25.1 保护整个项目 1964
4.25.2 保护关键代码 1967
4.25.3 更改授权母锁 1968
4.25.4 代码授权模式 1969
4.25.5 发布后的授权 1970
4.26 杂项 1970
4.26.1 语音朗读 1970
4.26.2 语音校对 1972
4.26.3 自定义帮助文件 1973
4.26.4 容易混淆表达式语法 1973
4.26.5 远程自动升级的实现 1974
4.26.6 定向数据表 1979
4.26.7 动态修改表结构 1981
4.26.8 创建操作日志 1984
4.26.9 创建系统列 1986
4.26.10 农历日期 1986
4.26.11 系统信息 1987
4.26.12 表达式求值 1988
4.26.13 模拟按键动作 1989
4.26.14 递归函数使用实例 1992
4.26.15 用递归函数生成会计科目目录树 1994
4.26.16 文件监视器 1998
4.26.17 使用剪贴板 2000
4.26.18 压缩和解压文件 2002
4.26.19 判断文件是否已经打开 2006
4.26.20 垂直表和水平表的转换 2006
4.26.21 获取特殊目录的路径 2010
4.26.22 ShowAppWindow 方法 2011
4.26.23 Image 类 2011
4.26.24 StringBuilder 类 2012
4.26.25 网络与启动速度 2013
4.26.26 修复 MDAC 2.8 2014
4.26.27 在代码中拆分和合并语句 2014
4.26.28 捕获异常错误 2015
4.26.29 数据库事务 2017

4.26.30 暂停执行事件代码 2018
4.26.31 常见效率问题 2019
4.26.32 高级话题 2030
5. 专业报表 2037
5.1 关于专业报表 2037
5.2 报表基础 2038
5.2.1 一行文本 2038
5.2.2 样式设置 2038
5.2.3 排列方式 2048
5.2.4 默认尺寸 2049
5.2.5 位置与尺寸 2050
5.2.6 打印图片 2052
5.2.7 复杂图文 2056
5.2.8 换页控制 2057
5.2.9 容器(RenderArea) 2059
5.2.10 页面设置 2062
5.2.11 分栏打印 2068
5.2.12 强制换页 2071
5.2.13 页眉页脚 2073
5.2.14 水印 2074
5.2.15 自动生成目录 2075
5.2.16 指定打印机 2076
5.2.17 禁止自动旋转 2077
5.2.18 禁止打印 2077
5.2.19 无需记忆任何东西 2077
5.3 表格基础 2081
5.3.1 创建表格 2081
5.3.2 网格线 2082
5.3.3 默认尺寸 2083
5.3.4 设定尺寸 2084
5.3.5 指定行数和列数 2085
5.3.6 指定行高和列宽 2086
5.3.7 插入行和列 2089
5.3.8 单元格内距 2090
5.3.9 行列样式 2091
5.3.10 单元格样式 2092
5.3.11 打印图片 2093
5.3.12 快速设置打印对象 2094
5.3.13 统一设置属性 2095
5.3.14 一个例外 2096
5.3.15 给表格加上标题 2097
5.3.16 合并单元格 2098
5.3.17 再谈合并单元格 2099
5.3.18 丢失的表头 2101

5.3.19 行组与表头 2102
5.3.20 复杂的表头 2104
5.3.21 每页都显示标题和表头 2105
5.3.22 列组 2106
5.3.23 表尾 2108
5.3.24 用表格设计标签 2108
5.3.25 单元格的容器 2109
5.3.26 表格与页眉页脚 2110
5.4 其他对象 2111
5.4.1 线条 2111
5.4.2 条形码 2115
5.4.3 弧形 2116
5.4.4 椭圆形 2117
5.4.5 饼形 2118
5.4.6 四边形 2119
5.4.7 空对象 2119
5.4.8 RenderGraphics 2120
5.5 实战演练 2121
5.5.1 打印表 2121
5.5.2 打印单据 2138
5.6 更专业的条形码功能 2153
5.7 增加图表功能 2154
5.8 保存报表 2154
5.9 数据绑定 2156
5.9.1 绑定基础 2156
5.9.2 分组报表 2160
6. 发布项目 2175
6.1 关于发布 2175
6.2 发布项目 2175
6.3 制作安装程序 2177
6.4 发布后程序的远程更新 2184
6.5 自动生成远程升级文件 2190
6.6 自动消失的 Foxtable 痕迹 2191
6.7 自定义主程序和描述 2192
6.8 如何保护开发成果 2193
7. 高级开发指南 2196
7.1 关于《高级开发指南》 2196
7.2 关于移动开发 2197
8. 外接设备 2204
8.1 短信收发 2204
8.1.1 关于配套的短信猫 2204
8.1.2 注册短信猫组件 2204
8.1.3 设置短信猫事件 2204
8.1.4 确定端口号 2205

8.1.5 打开和关闭短信猫 2206
8.1.6 判断短信猫状态 2207
8.1.7 设置短信中心号码 2207
8.1.8 发送短信 2207
8.1.9 读取短信 2209
8.1.10 记录来电号码 2210
8.1.11 一个完整的例子 2210
8.1.12 应用实例 2211
8.1.13 编程参考 2217
8.1.14 短信与发布 2220
8.1.15 另一种发短信的方式 2222
8.2 指纹识别 2223
8.2.1 关于指纹识别 2223
8.2.2 FOXFinger 参考 2224
8.2.3 设置指纹事件代码 2224
8.2.4 基本概念与步骤 2225
8.2.5 一个简单的例子 2227
8.2.6 一些 UI 图片 2229
8.2.7 设计指纹登录系统 2229
8.2.8 设计指纹审核系统 2231
8.3 使用身份证阅读器 2232
8.4 来电显示与录音 2235
官方网站为本节内容提供了示例，下载地址：http://www.foxtable.com/samples/caller.rar 2235
8.5 智能 IC 卡读写示例 2239
官方网站为本节内容提供了示例，下载地址：http://www.foxtable.com/samples/iccard.rar 2239
9. 附录 2245
9.1 系统命令大全 2245
9.2 中英文颜色对照表 2252
9.3 纸张编号与类型表 2255
9.4 Keys 枚举 2259
9.5 VBScript 参考 2265
9.6 Excel 与 VBA 2274
9.7 Word 和 VBA 2309

1. 关于 Foxtable

1.1 关于 Foxtable

oxtable 将 Excel、Access、Foxpro 以及易表的优势融合在一起，无论是数据录入、查询、统计，还是报表生成，都前所未有的强大和易用，使得普通用户无需编写任何代码，即可轻松完成复杂的数据管理工作，真正做到拿来即用。同时 Foxtable 又是一个高效的.net 平台开发工具，专门针对数据管理软件的开发作了大量的优化，使得用户在开发过程中只需关注商业逻辑，无需纠缠于具体功能的实现，这样 Foxtable 不仅开发效率十倍于其他专业开发工具，而且更加易用，几乎人人都能掌握。让普通人开发出专业水准的软件，以前是一个不可思议的想法，而 Foxtable 的出现，让这成为可能。

Foxtable 不仅内建数据库，还支持 Access、SQL Server、Oracle 等主流数据库作为数据源，并提供了数据动态加载、后台统计等功能，使得相距千里的不同电脑协同处理数据，以及海量数据管理，非专业人士也能轻松应付，从而让普通人也能快速开发出各种基于互联网的管理系统，如进销存、ERP、OA、CRM、SCM、MIS、MRP 等等。此外 FoxTable 同时具备 B/S 和 C/S 架构的优势，可以像 B/S 软件一样易于部署，同时又具备 C/S 软件良好的用户体验和交互性。

1.2 关于帮助文件

Foxtable 首先是一个优秀的应用软件，其菜单提供的录入、统计、查询、报表功能非常强大，而且使用简单，你不需要编写任何代码，即可高效完成日常数据管理工作。
Foxtable 同时又是一个轻松、高效、强大的.NET 平台开发工具，实际上你所看到的 Foxtable 本身，也正是基于 Foxtable 开发的。
所以 Foxtable 的帮助文件主要分为两部分，分别为《使用指南》和《开发指南》。
对于相当多的人来说，仅仅掌握帮助文件中的《使用指南》，也相当于掌握了一个前所未有的数据管理利器，因为目前还没有任何一个通用软件能在数据管理方面达到同样或接近的水准。
《 使用指南》同时也是《开发指南》的基础， 因为一些基本概念，例如表达式和外部数据源的连接，就只在《使用指南》中讲述，所以不管你今后是否使用 Foxtable 进行开发，都必须细读 《使用指南》。
我们见过不少用户也许认为《使用指南》过于简单，简单浏览后就匆匆进入开发指南，结果事倍功半。
我们也提供了本帮助文件的在线版本，这样大家就可以利用手机或其它手持设备随时随地学习 Foxtable：

http://help.foxtable.com/
本帮助文件并不包括移动开发部分，如果你需要基于 Foxtable 开发手机端的管理系统，可以访问我们的移动开发指南：

http://www.foxtable.com/mobilehelp/

1.3 关于网络支持

Foxtable 内置有数据源，这也是默认的数据源，这使得 Foxtable 和 Excel 一样，可以拿来即用。
同时 Foxtable 也支持 Access、SQL Server、Oracle 等外部数据源，而且可以直接在 Foxtable 中创建外部数据表，或修改外部数据表的结构。
外部数据源可以来自本机，也可以来自于局域网或互联网，且无需安装服务端软件或进行任何设置，即时连接，即时使用。
不管你使用内部数据源，还是外部数据源，都支持网络环境下多人同时协同工作，且日常操作和使用都完全一致，无任何差别。
同时 Foxtable 还提供了迁移功能，可以将基于内部数据源设计的系统，快速迁移到 SQl Server，而用户要做的只是单击几次鼠标。
上述特性，使得普通用户也能轻松设计基于局域网或互联网的管理软件。

1.4 关于易表

除 Foxtable 外，我公司还有另一个产品：易表。
易表是一个非常经典的数据管理软件，目前国内有上 10 万的企事业单位正在使用着易表。
易表小巧精干，易于上手，虽然功能不如 Foxtable 全面，且不支持外部数据源，但也足以满足很多中小企业日常数据管理的需求。
如果您对功能和网络要求不高，需要一个更容易的数据管理软件，易表可以作为您的选择之一。
易表主页：
www.egrid2000.com

1.5 功能概览

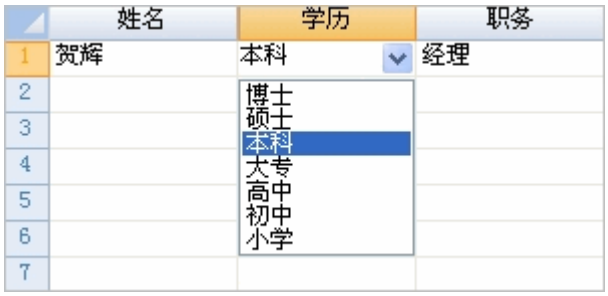
1.5.1 关于功能概览

功能概览并非 Foxtable 的使用指导，只是为了让你对 Foxtable 的基本功能有一个大概的了解。
Foxtable 的功能很多，这里只能简单介绍其中很少的一部分，只有通过对使用指南和开发指南的学习，才能真正了解和掌握 Foxtable 的全部功能。
你可以跳过功能概览，直接进入后面的使用指南开始学习。
作为一个应用软件，Foxtable 是非常简单的，任何人在帮助的指导下，都可以在短时间内快速掌握。

1.5.2 轻松录入

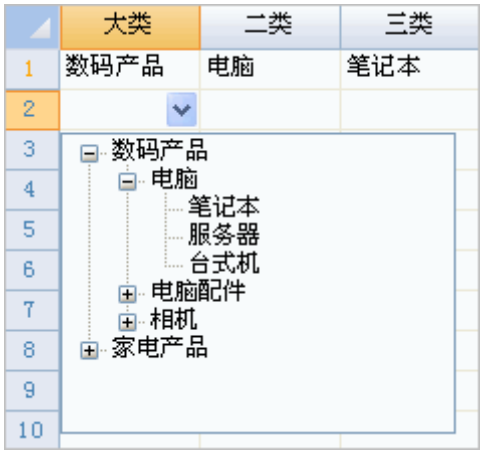
1.5.2.1 列表项目

通过设置列表项目，可以提高某些数据的录入速度，并能减少录入错误。例如在人事管理系统中，可以将学历列的列表项目设为“博士|硕士|本科|大专|高中|初中|小学”，这样一旦选择学历列，将出现一个下拉列表框，可以从中选择学历：

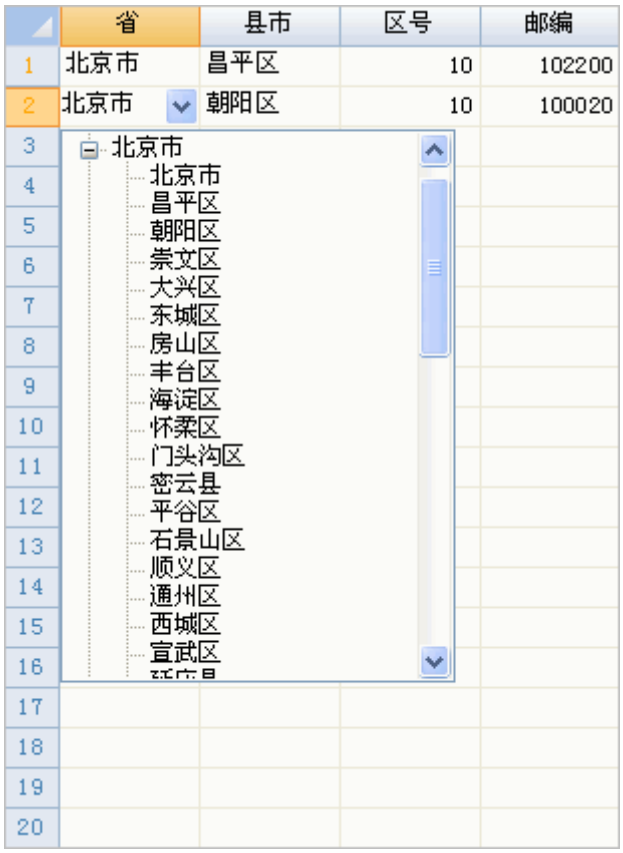


1.5.2.2 目录列表

目录列表能够更加高效地输入数据，例如下图的大类列就设置了目录列表，在大类列单击下拉按钮，会出现一个目录树，单击目录树中的某个节点，能够同时输入大类、二类、三类这三列的数据：



再例如下面的目录列表，只需在省列的目录树选择某个县市，即可同时输入省、县市、邮编、区号四列的内容：



1.5.2.3 多值字段

Foxtable 提供了多值字段，多值字段可以同时选择多个值输入，例如下面的负责人列：



1.5.2.4 备注图片

Foxtable 提供了备注列和图片列，而且可以单独打开窗口进行编辑：

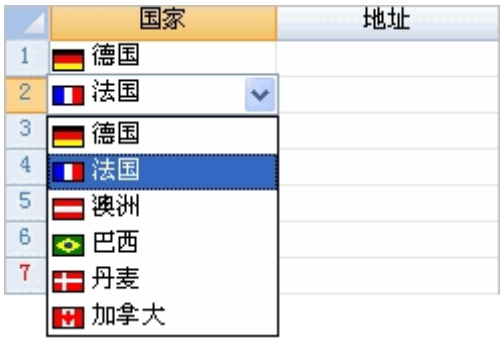


1.5.2.5 图形字典

通过设置图形字典，可以用图形这种更直观的形式来表示数据，例如下面的国家列就设置了图形字典:

	国家	地址
1	德国	
2	法国	
3	澳洲	
4	巴西	
5	丹麦	
6	加拿大	

图形字典也可以下拉输入：



1.5.2.6 记录窗口

如果一个表有很多列，我们将不得不左右滚动才能查看全部数据，以至经常顾此失彼，而 Foxtable 的记录窗口可以轻松解决这个问题。记录窗口以垂直方式显示数据，每次只显示一条记录，如下图所示:



1.5.2.7 内置输入器

对于数值列和日期列，可以选择使用内置输入器。

例如下面的入职日期列使用了内置输入器，单击其下拉按钮，会出现一个日历，用于选择输入日期：



下面的月薪列也是用了内置输入器，单击下拉按钮，会出现一个计算器，用于简单的计算和输入：



1.5.2.8 代码项目

通过设置代码项目，可以用代码快速输入数据，例如下面的岗位列，输入 1，会被替换为资深专家，输入 2，会被替换为高级专家，输入 4，会被替换为业务经理。



在候选值比较多的时候，用代码项目能更有效地输入，例如你还可以用 bj 表示北京，用 gz 表示广州等等。

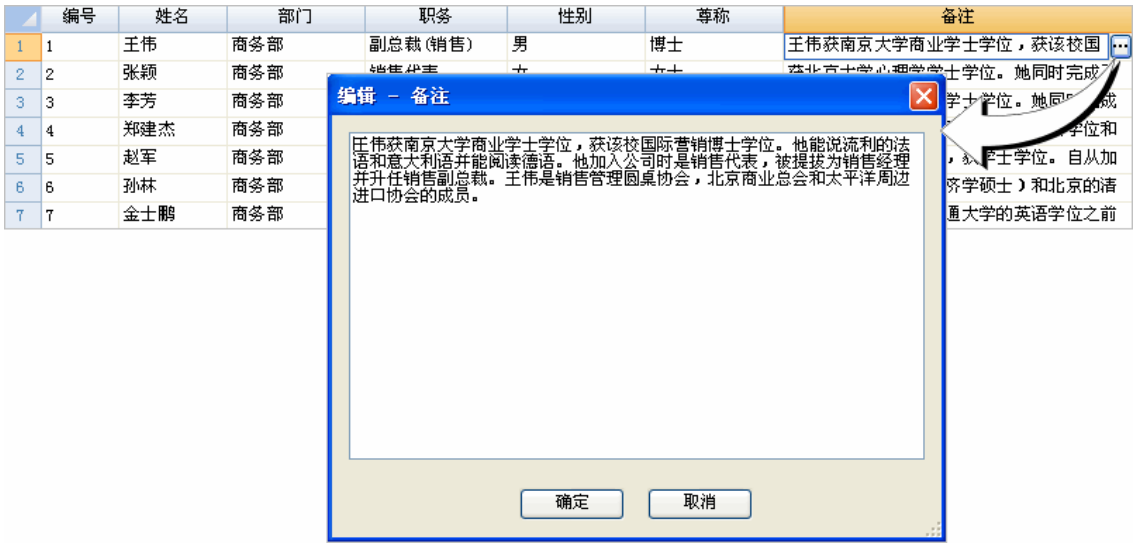
1.5.2.9 数据字典

下面的协调人列设置了数据字典，单击下拉按钮，会出现一个表格用于选择人员：



1.5.2.10 长文本编辑

Foxtable 可以直接在单元格右边显示一个按钮，单击此按钮，即可编辑和显示长文本：

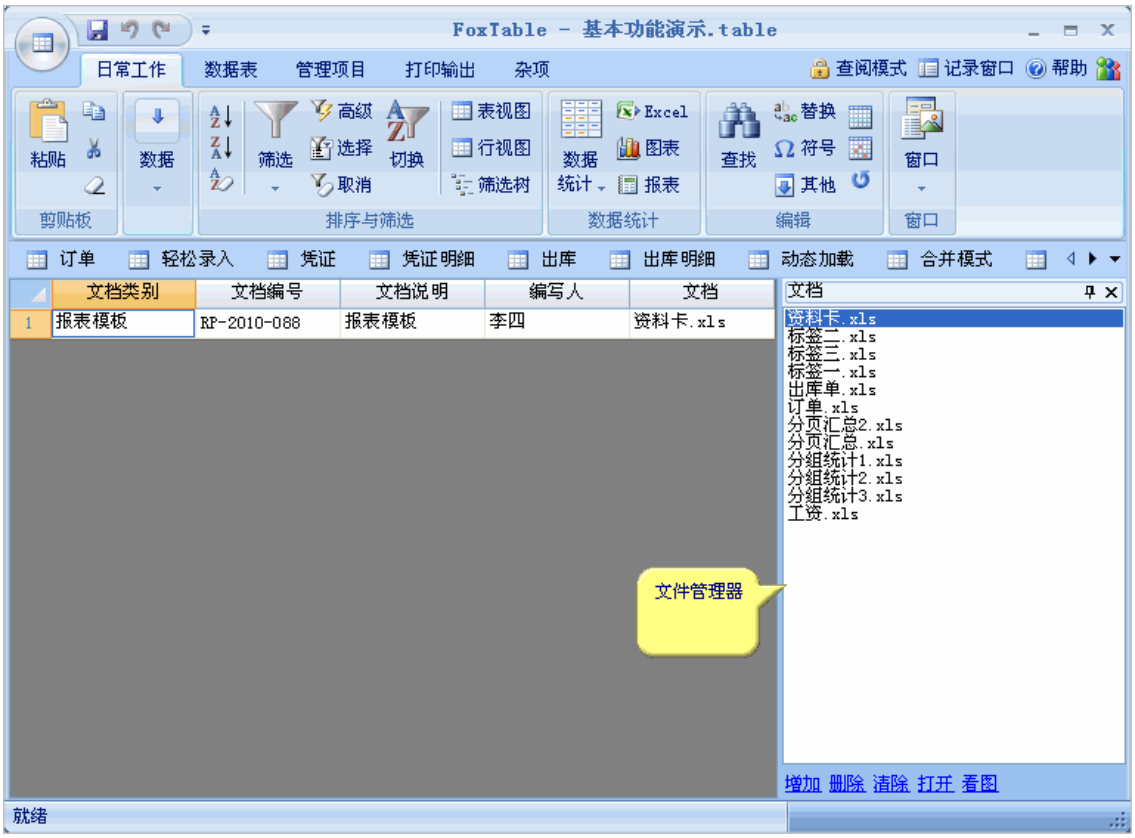


此外，还可以以备注列的形式，单独开一个停靠窗口，用来显示和编辑长文本：



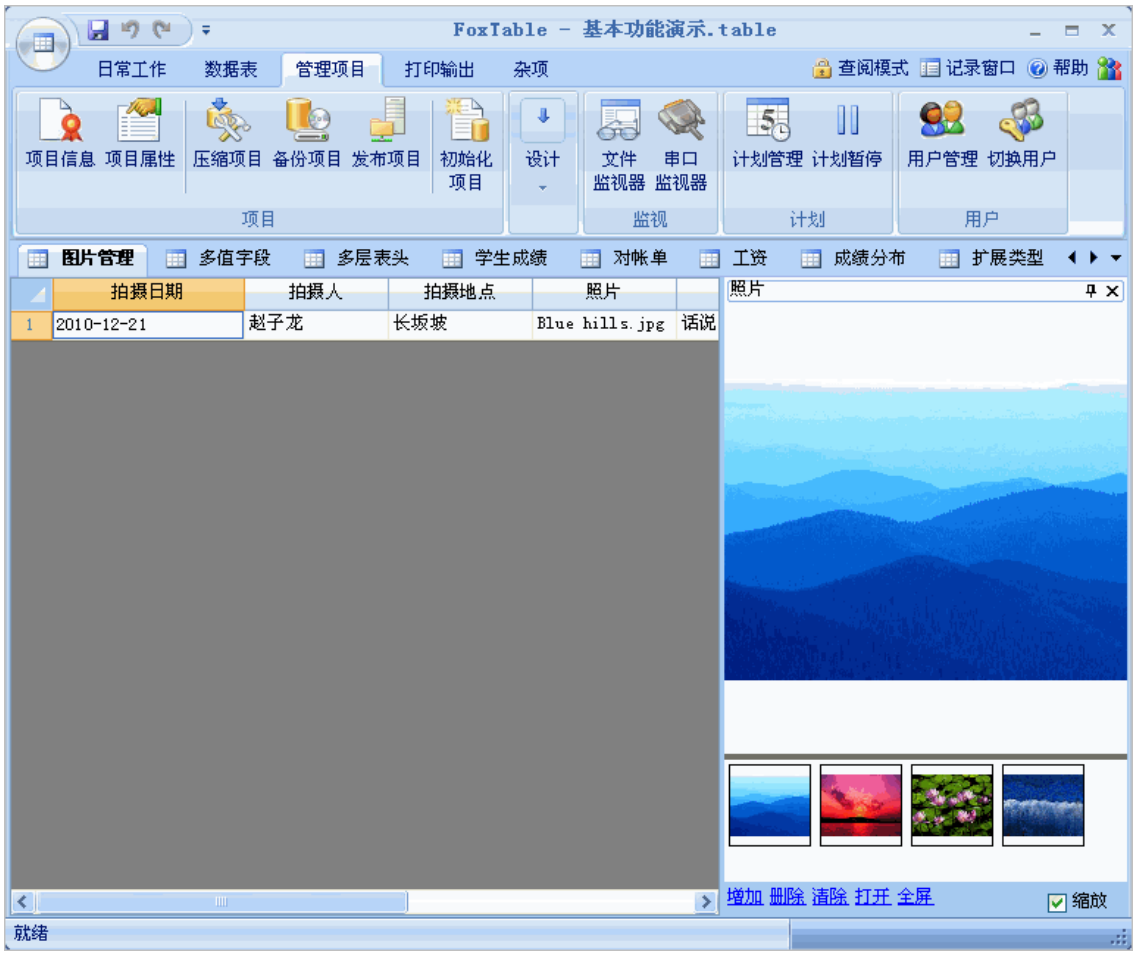
1.5.2.11 文件管理器

Foxtable 提供的多文件型列，可以高效便捷地管理各种类型的文档：
文件管理器：



1.5.2.12 图片管理器

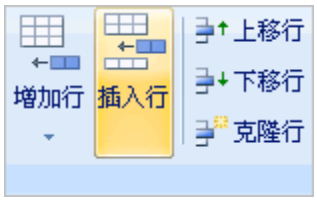
Foxtable 提供的图片型列，可以高效便捷地管理多个图片。
图片管理器：



1.5.2.13 插入行

Foxtable 作为一个纯数据库软件，却提供了电子表格的插入行、移动行功能，更加贴近普通用户的需要。

即使使用 SQL Server、Oracle、Access 作为 Foxtable 的数据源，也同样可以启用插入行、移动行功能。



1.5.2.14 修改标记

Foxtable 具备修改标记功能，能够以红底白字标出那些单元格已经被修改过。

例如订单表原来为：

	产品	客户	雇员	单价	折扣	数量	日期
1	PD05	CS03	EP04	17	0.1	650	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	1999-01-03
3	PD05	CS04	EP02	17	0	320	1999-01-03
4	PD01	CS03	EP04	18	0.15	80	1999-01-04
5	PD03	CS03	EP04	10	0	490	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	1999-01-07
7	PD05	CS03	EP04	17	0	160	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1999-01-08

修改订单数据后，按下菜单中的标记按钮，即可以红底白字标出所有已经修改过的单元格：

	产品	客户	雇员	单价	折扣	数量	日期
1	PD05	CS03	EP04	17	0.1	650	1999-01-02
2	PD02	CS01	EP01	21	0.1	400	1999-01-03
3	PD05	CS04	EP02	17	0	320	1999-01-03
4	PD01	CS03	EP04	18	0.2	80	1999-01-04
5	PD03	CS05	EP04	10	0	490	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	1999-01-07
7	PD05	CS03	EP04	17	0	160	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1999-01-08

而且你可以单独选择任何一个已经修改过的单元格，将其值还原到修改前。

1.5.3 轻松统计

1.5.3.1 汇总模式

1.5.3.1.1 汇总模式

利用 Foxtable 的汇总模式，你只需单击几次鼠标，即可得到各种各样的统计结果：

	产品	客户	雇员	数量	单价	折扣	金额	日期
194	PD03	CS05	EP01	485	13.5	0	6547.50	2010-06-04
195	PD03	CS05	EP01	706	12.5	0.05	8383.75	2010-07-03
196	PD03	CS05	EP01	793	12.5	0	9912.50	2010-07-23
197	PD03	CS05	EP01	5	12.5	0	62.50	2010-11-09
	PD03 小计			4460			56816.75	
198	PD04	CS05	EP01	52	22	0	1144.00	2010-01-07
199	PD04	CS05	EP01	881	21.5	0	18941.50	2010-01-22
200	PD04	CS05	EP01	325	21	0	6825.00	2010-04-02
201	PD04	CS05	EP01	383	21	0	8043.00	2010-04-02
202	PD04	CS05	EP01	2	23	0	46.00	2010-06-14
203	PD04	CS05	EP01	768	22.5	0.15	14688.00	2010-08-06
204	PD04	CS05	EP01	357	20.5	0	7318.50	2010-09-30
205	PD04	CS05	EP01	555	22.5	0	12487.50	2010-10-19
206	PD04	CS05	EP01	446	23	0	10258.00	2010-10-29
	PD04 小计			3769			79751.50	
207	PD05	CS05	EP01	971	31	0	30101.00	2010-06-27
208	PD05	CS05	EP01	926	29.5	0.1	24585.30	2010-07-13
209	PD05	CS05	EP01	762	30.5	0	23241.00	2010-07-23
210	PD05	CS05	EP01	799	30	0.1	21573.00	2010-10-26
	PD05 小计			3458			99500.30	
	CS05 合计			21634			448811.16	
	总计			104540			2191159.41	

1.5.3.1.2 分段统计

汇总模式能够根据日期分段统计，例如按天、按周、按月、按季、按年分组统计。

按月汇总：

	产品	客户	雇员	数量	单价	折扣	金额	日期
590	PD05	CS04	EP03	663	30.5	0.1	18199.35	2010-11-07
591	PD05	CS04	EP03	684	30.5	0	20862.00	2010-11-07
592	PD05	CS02	EP01	856	31	0	26536.00	2010-11-11
593	PD05	CS05	EP03	77	31	0	2387.00	2010-11-12
594	PD05	CS03	EP03	831	31.5	0.05	24867.68	2010-11-13
595	PD05	CS05	EP02	202	29.5	0	5959.00	2010-11-15
596	PD05	CS04	EP03	117	29.5	0	3451.50	2010-11-21
597	PD05	CS02	EP03	157	30	0	4710.00	2010-11-28
	11月 小计			4226			123560.97	
598	PD05	CS04	EP02	232	30.5	0	7076.00	2010-12-01
599	PD05	CS01	EP03	569	29	0	16501.00	2010-12-02
600	PD05	CS02	EP02	6	31	0	186.00	2010-12-02
601	PD05	CS03	EP03	930	31	0.02	28253.40	2010-12-03
602	PD05	CS02	EP01	133	31.5	0	4189.50	2010-12-06
603	PD05	CS03	EP02	262	31.5	0	8253.00	2010-12-11
604	PD05	CS03	EP02	483	29	0	14007.00	2010-12-13
605	PD05	CS02	EP03	390	28.5	0	11115.00	2010-12-20
606	PD05	CS01	EP03	26	29	0	754.00	2010-12-23
607	PD05	CS01	EP01	549	30	0	16470.00	2010-12-29
	12月 小计			3580			106804.90	
	PD05 合计			53411			1553986.23	
	总计			303731			6294008.57	

按季汇总：

	产品	客户	雇员	数量	单价	折扣	金额	日期
56	PD01	CS01	EP02	132	17.5	0	2310.00	2010-09-09
57	PD01	CS01	EP02	789	17.5	0.02	13531.35	2010-09-18
	3季度 小计			3290			57999.35	
58	PD01	CS01	EP03	781	18.5	0	14448.50	2010-10-01
59	PD01	CS01	EP01	76	17.5	0	1330.00	2010-10-05
60	PD01	CS01	EP05	739	18.5	0.12	12030.92	2010-10-10
61	PD01	CS01	EP03	108	18.5	0	1998.00	2010-10-12
62	PD01	CS01	EP01	983	18.5	0	18185.50	2010-10-12
63	PD01	CS01	EP02	291	17.5	0	5092.50	2010-10-21
64	PD01	CS01	EP04	922	19	0.1	15766.20	2010-10-23
65	PD01	CS01	EP04	510	19	0.1	8721.00	2010-10-24
66	PD01	CS01	EP04	703	18	0	12654.00	2010-11-03
67	PD01	CS01	EP02	240	19	0	4560.00	2010-11-04
68	PD01	CS01	EP01	443	19	0	8417.00	2010-11-16
69	PD01	CS01	EP02	976	18.5	0	18056.00	2010-11-28
70	PD01	CS01	EP05	11	17.5	0	192.50	2010-12-08
71	PD01	CS01	EP05	828	19	0.1	14158.80	2010-12-22
72	PD01	CS01	EP03	589	18.5	0.12	9588.92	2010-12-24
73	PD01	CS01	EP01	186	19	0	3534.00	2010-12-26
	4季度 小计			8386			148733.84	
	2010年 合计			18610			326034.20	
	总计			39079			688846.61	

1.5.3.1.3 截止统计

汇总模式不仅能够按时间分组统计，而且可以统计截止到某个时间的累计数量。
例如下面不仅统计出了每个月的销量和金额，还统计出了截止到每个月为止的累计销量和金额：

	产品	客户	雇员	数量	单价	折扣	金额	日期
211	PD01	CS01	EP02	976	18.5	0	18056.00	2010-11-28
	小计 11月			6683			127670.53	
	截止 11月			65599			1351219.44	
212	PD05	CS01	EP03	569	29	0	16501.00	2010-12-02
213	PD03	CS01	EP01	938	13.5	0.05	12029.85	2010-12-04
214	PD04	CS01	EP05	577	23	0.05	12607.45	2010-12-04
215	PD05	CS01	EP05	433	30.5	0	13206.50	2010-12-08
216	PD03	CS01	EP02	981	13	0.05	12115.35	2010-12-16
217	PD02	CS01	EP01	914	28	0	25592.00	2010-12-16
218	PD02	CS01	EP04	743	27	0.12	17653.68	2010-12-20
219	PD01	CS01	EP05	828	19	0.1	14158.80	2010-12-22
220	PD03	CS01	EP02	871	13	0	11323.00	2010-12-22
221	PD02	CS01	EP01	955	26	0.05	23588.50	2010-12-22
222	PD02	CS01	EP01	998	27	0	26946.00	2010-12-23
223	PD02	CS01	EP04	833	25.5	0	21241.50	2010-12-23
224	PD01	CS01	EP03	589	18.5	0.12	9588.92	2010-12-24
225	PD03	CS01	EP05	884	12.5	0.02	10829.00	2010-12-26
226	PD05	CS01	EP01	549	30	0	16470.00	2010-12-29
	小计 12月			11662			243851.55	
	截止 12月			77261			1595070.99	
	2010年 合计			77261			1595070.99	
	总销量			156392			3387806.44	

1.5.3.1.4 多种统计

汇总模式可以对同一列进行多种统计，例如下面的汇总模式，同时统计出了各个科目的最低分，最高分和平均分：

	班级	姓名	语文	数学	英语	化学	物理	总分
30	3班	谢杰立	114	105	105	96	82	502.0000
31	3班	谢晓霞	116	100	100	99	83	498.0000
32	3班	杨景亮	118	95	95	100	84	492.0000
33	3班	张李华	120	90	90	92	85	477.0000
	3班 最低分			90	90	75	75	477.0000
	3班 最高分			120	140	100	85	510.0000
	3班 平均分			102	115	89	80	495.9091
34	4班	陈成进	110	78	106	98	36	428.0000
35	4班	陈诗淇	113	123	117	78	78	509.0000
36	4班	陈思达	113	75	92	46	55	381.0000
37	4班	程康文	116	120	132	83	78	529.0000
38	4班	邓玉婷	118	109	90	78	77	472.0000
39	4班	林景阳	119	133	95	61	57	465.0000
40	4班	苏　捷	124	118	124	88	83	537.0000
41	4班	谢杰立	129	118	139	80	88	554.0000
42	4班	谢晓霞	134	113	132	87	85	551.0000
43	4班	杨景亮	132	136	146	94	93	601.0000
44	4班	张李华	133	125	140	75	71	544.0000
	4班 最低分			75	90	46	36	381.0000
	4班 最高分			136	146	98	93	601.0000
	4班 平均分			113	119	79	73	506.4545
	最低分(总)			58	74	34	36	381.0000
	最高分(总)			136	146	100	93	601.0000
	平均分(总)			110	118	79	76	502.1818

1.5.3.1.5 框架模式

在设置汇总模式的时候，可以在表格左侧显示一个目录树，通过这个目录树，我们可以用类似于Windows 资源管理器的方式来显示和管理表中数据：

1 2 *	产品	客户	雇员	数量	单价	折扣	金额	日期
	总计			113208			2368133.29	
⊟	EP01 合计			24996			502680.85	
⊟	PD01 小计			5597			102528.11	
1	PD01	CS05	EP01	194	18	0	3492.00	2010-01-04
2	PD01	CS01	EP01	898	19	0	17062.00	2010-01-16
3	PD01	CS05	EP01	406	18.5	0	7511.00	2010-01-31
4	PD01	CS02	EP01	525	18	0	9450.00	2010-02-28
5	PD01	CS05	EP01	418	18	0	7524.00	2010-03-06
6	PD01	CS02	EP01	597	18.5	0.02	10823.61	2010-03-13
7	PD01	CS02	EP01	938	18	0	16884.00	2010-03-17
8	PD01	CS01	EP01	465	19	0	8835.00	2010-03-20
9	PD01	CS04	EP01	419	18	0	7542.00	2010-03-24
10	PD01	CS03	EP01	507	18.5	0	9379.50	2010-03-28
11	PD01	CS05	EP01	230	17.5	0	4025.00	2010-03-28
⊞	PD02 小计			4106			106980.90	
⊞	PD03 小计			7736			95941.50	
⊞	PD04 小计			2791			59508.00	
⊞	PD05 小计			4766			137722.34	
⊞	EP02 合计			26380			530785.48	
⊞	EP03 合计			22623			470854.85	
⊞	EP04 合计			21154			487684.53	
⊞	EP05 合计			18055			376127.59	

1.5.3.1.6 自定义汇总

Foxtable 可以自定义汇总模式，用以满足一些特殊的要求，如下图所示的汇总模式：

	产品	客户	雇员	数量	单价	折扣	金额	日期
43	PD04	CS01	EP05	175	20.5	0	3587.50	2010-02-25
44	PD04	CS03	EP03	953	22	0	20966.00	2010-02-26
45	PD04	CS04	EP02	444	22.5	0	9990.00	2010-02-27
46	PD04	CS02	EP04	877	21	0	18417.00	2010-02-28
	产品:PD04 订单数:14 订购数量:6328 订购金额:134292.08							
47	PD05	CS05	EP03	332	31.5	0	10458.00	2010-02-03
48	PD05	CS04	EP04	478	28.5	0	13623.00	2010-02-03
49	PD05	CS03	EP01	492	29.5	0	14514.00	2010-02-06
50	PD05	CS04	EP01	333	30	0	9990.00	2010-02-06
51	PD05	CS02	EP03	38	29	0	1102.00	2010-02-06
52	PD05	CS03	EP02	108	31	0	3348.00	2010-02-07
53	PD05	CS01	EP04	620	31	0.02	18835.60	2010-02-09
54	PD05	CS04	EP05	481	29	0	13949.00	2010-02-10
55	PD05	CS02	EP01	246	30	0	7380.00	2010-02-13
56	PD05	CS02	EP03	252	29	0	7308.00	2010-02-15
57	PD05	CS03	EP04	86	30	0	2580.00	2010-02-18
58	PD05	CS03	EP01	859	30	0.12	22677.60	2010-02-22
59	PD05	CS02	EP03	780	30.5	0	23790.00	2010-02-22
60	PD05	CS03	EP03	14	29.5	0	413.00	2010-02-24
61	PD05	CS02	EP04	928	31.5	0	29232.00	2010-02-25
62	PD05	CS03	EP04	456	30.5	0	13908.00	2010-02-27
	产品:PD05 订单数:16 订购数量:6503 订购金额:193108.2							

1.5.3.2 分组统计

1.5.3.2.1 分组统计

利用分组统计，只需单击几次鼠标，即可得到任何你想要的统计结果。
和汇总模式不同，分组统计单独生成一个统计表来显示统计结果，而且统计能力更强大。
不同产品的销售数量和金额：

	产品	数量	金额
1	PD01	193943	3401923.65
2	PD02	200510	5235463.285
3	PD03	194018	2432593.06
4	PD04	195189	4132876.15
5	PD05	206828	5975820.585
6	合计	990488	21178676.73

各月销售数量和金额：

	年	月	数量	金额
1	2009	1	41426	889732.04
2	2009	2	44734	984384.355
3	2009	3	42236	910206.395
4	2009	4	43971	916265.93
5	2009	5	39369	851883.78
6	2009	6	39201	849708.06
7	2009	7	39295	868029.02
8	2009	8	51104	1106204.245
9	2009	9	51236	1074093.76
10	2009	10	33347	688785.595
11	2009	11	41818	906485.145
12	2009	12	37859	870163.005
13	2010	1	44005	902920.705
14	2010	2	29402	669362.07
15	2010	3	39801	795850.51
16	2010	4	41544	914346.175
17	2010	5	42099	879864.425
18	2010	6	34300	

1.5.3.2.2 多种统计

分组统计可以对同一列进行多种统计，例如下图同时统计出了各个科目的最低分，最高分和平均分:

	班级	语文			数学			英语			化学			物理			总分		
		平均分	最高分	最低分	平均分	最高分	最低分	平均分	最高分	最低分	平均分	最高分	最低分	平均分	最高分	最低分	平均分	最高分	最低分
1	1班	122.09	133	110	111.45	136	58	117.91	146	74	72.82	94	34	75.73	93	57	500	601	389
2	2班	122.55	134	110	113.45	136	75	119.36	146	90	77.27	94	46	73.73	93	36	506.36	601	391
3	3班	110	120	100	102.27	120	90	115	140	90	88.64	100	75	80	85	75	495.91	510	477
4	4班	121.91	134	110	113.45	136	75	119.36	146	90	78.91	98	46	72.82	93	36	506.45	601	381

1.5.3.2.3 环比分析

分组统计可以进行环比分析，例如下图不仅统计出了各月的销售数量，还计算出了相对上月的环比增长率：

	年	月	数量	环比增长
1	2009	1	41426	
2	2009	2	44734	7.99%
3	2009	3	42236	-5.58%
4	2009	4	43971	4.11%
5	2009	5	39369	-10.47%
6	2009	6	39201	-0.43%
7	2009	7	39295	0.24%
8	2009	8	51104	30.05%
9	2009	9	51236	0.26%
10	2009	10	33347	-34.91%
11	2009	11	41818	25.40%
12	2009	12	37859	-9.47%
13	2010	1	44005	16.23%
14	2010	2	29402	-33.18%
15	2010	3	39801	35.37%
16	2010	4	41544	4.38%
17	2010	5	42099	1.34%
18	2010	6	309	-18.53%

按周、按月，按季、按年都可以进行环比分析，也可以同时对多列进行环比分析，例如下图按季度统计销售数量和金额 ，并各自计算了环比增长率：

	年	季	销量		金额	
			值	环比	值	环比
1	2009	1	128396		2784322.79	
2	2009	2	122541	-4.56%	2617857.77	-5.98%
3	2009	3	141635	15.58%	3048327.025	16.44%
4	2009	4	113024	-20.20%	2465433.745	-19.12%
5	2010	1	113208	0.16%	2368133.285	-3.95%
6	2010	2	117943	4.18%	2476147.245	4.56%
7	2010	3	127816	8.37%	2751678.125	11.13%
8	2010	4	125925	-1.48%	2666776.745	-3.09%

1.5.3.2.4 同比分析

分组统计可以进行同比分析，例如下图不仅统计出了各月的销售数量，还计算出了相对去年同期的同比增长率：

	年	月	数量	同比增长
1	2009	1	41426	
2	2009	2	44734	
3	2009	3	42236	
4	2009	4	43971	
5	2009	5	39369	
6	2009	6	39201	
7	2009	7	39295	
8	2009	8	51104	
9	2009	9	51236	
10	2009	10	33347	
11	2009	11	41818	
12	2009	12	37859	
13	2010	1	44005	6.23%
14	2010	2	29402	-34.27%
15	2010	3	39801	-5.77%
16	2010	4	41544	-5.52%
17	2010	5	42099	6.93%
18	2010	6	34300	-12.50%
19	2010	7	43653	11.09%
20	2010	8	46453	-9.10%
21	2010	9	37710	-26.40%
22	2010	10	47441	42.26%
23	2010	11	38726	-7.39%
24	2010	12	39758	5.02%

按周、按月，按季、按年都可以进行同比分析，也可以同时对多列进行同比分析，例如下图按季度统计销售数量和金额 ，并各自计算了同比增长率：

	年	季	销量		金额	
			值	同比	值	同比
1	2009	1	128396		2784322.79	
2	2009	2	122541		2617857.77	
3	2009	3	141635		3048327.025	
4	2009	4	113024		2465433.745	
5	2010	1	113208	-11.83%	2368133.285	-14.95%
6	2010	2	117943	-3.75%	2476147.245	-5.41%
7	2010	3	127816	-9.76%	2751678.125	-9.73%
8	2010	4	125925	11.41%	2666776.745	8.17%

1.5.3.2.5 占比分析

分组统计可以进行占比分析，例如下图不仅统计出了各月的销量，还计算出了每月销量占全年销量的比例：

	月	数量	占比
1	1	44005	9.08%
2	2	29402	6.06%
3	3	39801	8.21%
4	4	41544	8.57%
5	5	42099	8.68%
6	6	34300	7.07%
7	7	43653	9.00%
8	8	46453	9.58%
9	9	37710	7.78%
10	10	47441	9.78%
11	11	38726	7.99%
12	12	39758	8.20%

1.5.3.2.6 综合分析

分组统计可以同时进行环比、同比、占比分析，这样我们就可以进行更为复杂的统计分析工作。例如下图按月统计销量和金额，并计算各月销量和金额的环比和同比增长率：

	年	月	销量			金额		
			值	同比	环比	值	同比	环比
1	2009	1	41426			889732.04		
2	2009	2	44734		7.99%	984384.355		10.64%
3	2009	3	42236		-5.58%	910206.395		-7.54%
4	2009	4	43971		4.11%	916265.93		0.67%
5	2009	5	39369		-10.47%	851883.78		-7.03%
6	2009	6	39201		-0.43%	849708.06		-0.26%
7	2009	7	39295		0.24%	868029.02		2.16%
8	2009	8	51104		30.05%	1106204.245		27.44%
9	2009	9	51236		0.26%	1074093.76		-2.90%
10	2009	10	33347		-34.91%	688785.595		-35.87%
11	2009	11	41818		25.40%	906485.145		31.61%
12	2009	12	37859		-9.47%	870163.005		-4.01%
13	2010	1	44005	6.23%	16.23%	902920.705	1.48%	3.76%
14	2010	2	29402	-34.27%	-33.18%	669362.07	-32.00%	-25.87%
15	2010	3	39801	-5.77%	35.37%	795850.51	-12.56%	18.90%
16	2010	4	41544	-5.52%	4.38%	914346.175	-0.21%	14.89%
17	2010	5	42099	6.93%	1.34%	879864.425	3.28%	-3.77%
18	2010	6	34300	-12.50%	-18.53%	681936.645	-19.74%	-22.50%
19	2010	7	43653	11.09%	27.27%	908663.645	4.68%	33.25%
20	2010	8	46453	-9.10%	6.41%	1026431.4	-7.21%	12.96%
21	2010	9	37710	-26.40%	-18.82%	816583.08	-23.97%	-20.44%
22	2010	10	47441	42.26%	25.80%	979121.4	42.15%	19.90%
23	2010	11	38726	-7.39%	-18.37%	845087.095	-6.77%	-13.69%
24	2010	12	39758	5.02%	2.66%	842568.25	-3.17%	-0.30%

1.5.3.2.7 截止统计

分组统计也可以进行截止统计，例如下图统计出各月销量，以及截止到某月为止的累计销量：

	月	数量	截止
1	1	44005	44005
2	2	29402	73407
3	3	39801	113208
4	4	41544	154752
5	5	42099	196851
6	6	34300	231151
7	7	43653	274804
8	8	46453	321257
9	9	37710	358967
10	10	47441	406408
11	11	38726	445134
12	12	39758	484892

截止统计还可以进行占比分析，例如下图统计出了各季度销量，以及截止到此季度的累计销量；并计算出该季度销量在全年销量中所占的比例，以及截止到该季度为止的累计销量在全年销量中所占的比例：

	季度	本季度		截止本季度	
		销量	占比	销量	占比
1	1	113208	23.35%	113208	23.35%
2	2	117943	24.32%	231151	47.67%
3	3	127816	26.36%	358967	74.03%
4	4	125925	25.97%	484892	100.00%

1.5.3.3 交叉统计

1.5.3.3.1 交叉统计

交叉统计可以同时设置水平和垂直分组列，是一种更为高级的分组统计。
下图是通过交叉统计得到的不同客户订购不同产品的数量：

	客户	产品					合计
		PD01	PD02	PD03	PD04	PD05	
1	CS01	39079	40043	38037	35028	37219	189406
2	CS02	43104	41905	38021	33746	38260	195036
3	CS03	43441	38956	41461	37262	38636	199756
4	CS04	31026	38628	33159	48428	46304	197545
5	CS05	37293	40978	43340	40725	46409	208745
6	合计	193943	200510	194018	195189	206828	990488

1.5.3.3.2 日期统计

交叉统计也可以根据日期分段统计。
下图统计的是每种产品在各个季度的销量，日期列作为垂直分组：

	产品	2009年				2010年				合计
		1季度	2季度	3季度	4季度	1季度	2季度	3季度	4季度	
1	PD01	24787	21332	24321	19471	25842	26816	23496	27878	193943
2	PD02	33477	20128	25219	22057	18309	24878	30353	26089	200510
3	PD03	22776	21801	24774	21416	24973	25477	27506	25295	194018
4	PD04	21717	32629	38292	21181	22471	17877	18662	22360	195189
5	PD05	25639	26651	29029	28899	21613	22895	27799	24303	206828
6	合计	128396	122541	141635	113024	113208	117943	127816	125925	990488

下图是每个产品在各月的销量，日期列作为水平分组：

	年	月	产品					合计
			PD01	PD02	PD03	PD04	PD05	
1	2009	1	5569	12720	8638	7325	7174	41426
2	2009	2	9685	10004	6029	8633	10383	44734
3	2009	3	9533	10753	8109	5759	8082	42236
4	2009	4	9026	5337	7644	13885	8079	43971
5	2009	5	4751	7840	7768	8903	10107	39369
6	2009	6	7555	6951	6389	9841	8465	39201
7	2009	7	5204	6807	6088	11864	9332	39295
8	2009	8	8103	9755	8579	13565	11102	51104
9	2009	9	11014	8657	10107	12863	8595	51236
10	2009	10	6001	6504	7644	7302	5896	33347
11	2009	11	7346	10433	8843	6165	9031	41818
12	2009	12	6124	5120	4929	7714	13972	37859
13	2010	1	8548	6409	11585	9142	8321	44005
14	2010	2	5449	7677	3445	6328	6503	29402
15	2010	3	11845	4223	9943	7001	6789	39801
16	2010	4	8228	11208	7213	6194	8701	41544
17	2010	5	10069	9154	8709	6439	7728	42099
18	2010	6	8519	4516	9555	5244	6466	34300
19	2010	7	6861	12682	12709	3821	7580	43653
20	2010	8	6830	8774	9137	7926	13786	46453
21	2010	9	9805	8897	5660	6915	6433	37710
22	2010	10	13623	7286	8444	9471	8617	47441
23	2010	11	6448	10598	7529	6231	7920	38726
24	2010	12	7807	8205	9322	6658	7766	39758
25	合计		193943	200510	194018	195189	206828	990488

1.5.3.3.3 多个分组

不管是是交叉统计的水平分组，还是垂直分组，都可以设置多个分组列。

例如下图统计的是不同客户在每年订购各种产品的数量，有两个垂直分组，分别是年和产品：

	客户	2009年					2010年					合计
		PD01	PD02	PD03	PD04	PD05	PD01	PD02	PD03	PD04	PD05	
1	CS01	20469	17479	11803	21422	23593	18610	22564	26234	13606	13626	189406
2	CS02	19747	21382	21476	18974	17170	23357	20523	16545	14772	21090	195036
3	CS03	17917	19409	19327	21310	19333	25524	19547	22134	15952	19303	199756
4	CS04	14186	21041	18451	27304	25130	16840	17587	14708	21124	21174	197545
5	CS05	17592	21570	19710	24809	24992	19701	19408	23630	15916	21417	208745
6	合计	89911	100881	90767	113819	110218	104032	99629	103251	81370	96610	990488

1.5.3.3.4 多列统计

交叉统计也可以同时对多列进行统计，例如下图按季度统计不同产品的销售数量和金额：

	年	季度	PD01		PD02		PD03		PD04		PD05		合计	
			数量	金额	数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	2009	1	24787	429218.8	33477	871260.3	22776	282122.485	21717	465153.835	25639	736567.37	128396	2784322.79
2	2009	2	21332	380654.09	20128	518920.96	21801	272221.295	32629	684919.46	26651	761141.965	122541	2617857.77
3	2009	3	24321	425556.36	25219	665477.9	24774	315138.885	38292	801836.815	29029	840317.065	141635	3048327.025
4	2009	4	19471	341100.14	22057	575852.935	21416	270669.615	21181	446552.72	28899	831258.335	113024	2465433.745
5	2010	1	25842	458767.985	18309	482494.65	24973	313476.745	22471	472405.485	21613	640988.42	113208	2368133.285
6	2010	2	26816	460248.52	24878	651306.425	25477	316584.61	17877	391172.275	22895	656835.415	117943	2476147.245
7	2010	3	23496	414215.54	30353	790808.18	27506	347216.255	18662	393203.545	27799	806234.605	127816	2751678.125
8	2010	4	27878	492162.215	26089	679341.935	25295	315163.17	22360	477632.015	24303	702477.41	125925	2666776.745
9	合计		193943	3401923.65	200510	5235463.285	194018	2432593.06	195189	4132876.15	206828	5975820.585	990488	21178676.73

下图统计的是不同客户订购各种产品的数量和金额：

	客户	PD01		PD02		PD03		PD04		PD05		合计	
		数量	金额	数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	CS01	39079	688846.61	40043	1031933.6	38037	473554.56	35028	746472.685	37219	1078435.34	189406	4019242.795
2	CS02	43104	744678.85	41905	1095777.2	38021	472682.595	33746	709516.135	38260	1112161.605	195036	4134816.385
3	CS03	43441	762825.83	38956	1013186.905	41461	520570.865	37262	787108.25	38636	1120150.64	199756	4203842.49
4	CS04	31026	545352.215	38628	999286.375	33159	419556.365	48428	1026135.57	46304	1335021.67	197545	4325352.195
5	CS05	37293	660220.145	40978	1095279.205	43340	546228.675	40725	863643.51	46409	1330051.33	208745	4495422.865
6	合计	193943	3401923.65	200510	5235463.285	194018	2432593.06	195189	4132876.15	206828	5975820.585	990488	21178676.73

1.5.3.3.5 数据分析

交叉统计也有很强的数据分析功能。
各客户订购不同产品的数量，以及每个产品在该客户总订购量中所占的比例：

	产品	CS01		CS02		CS03		CS04		CS05		合计
		数量	占比	数量	占比	数量	占比	数量	占比	数量	占比	
1	PD01	39079	20.63%	43104	22.10%	43441	21.75%	31026	15.71%	37293	17.87%	193943
2	PD02	40043	21.14%	41905	21.49%	38956	19.50%	38628	19.55%	40978	19.63%	200510
3	PD03	38037	20.08%	38021	19.49%	41461	20.76%	33159	16.79%	43340	20.76%	194018
4	PD04	35028	18.49%	33746	17.30%	37262	18.65%	48428	24.51%	40725	19.51%	195189
5	PD05	37219	19.65%	38260	19.62%	38636	19.34%	46304	23.44%	46409	22.23%	206828
6	合计	189406	100.00%	195036	100.00%	199756	100.00%	197545	100.00%	208745	100.00%	990488

按月统计不同产品的销售数量，以及某月某产品的销量在该产品的年销量中所占的比例：

	年	月	PD01		PD02		PD03		PD04		PD05		合计
			销量	占比	销量	占比	销量	占比	销量	占比	销量	占比	
1	2009	1	5569	6.19%	12720	12.61%	8638	9.52%	7325	6.44%	7174	6.51%	41426
2	2009	2	9685	10.77%	10004	9.92%	6029	6.64%	8633	7.58%	10383	9.42%	44734
3	2009	3	9533	10.60%	10753	10.66%	8109	8.93%	5759	5.06%	8082	7.33%	42236
4	2009	4	9026	10.04%	5337	5.29%	7644	8.42%	13885	12.20%	8079	7.33%	43971
5	2009	5	4751	5.28%	7840	7.77%	7768	8.56%	8903	7.82%	10107	9.17%	39369
6	2009	6	7555	8.40%	6951	6.89%	6389	7.04%	9841	8.65%	8465	7.68%	39201
7	2009	7	5204	5.79%	6807	6.75%	6088	6.71%	11864	10.42%	9332	8.47%	39295
8	2009	8	8103	9.01%	9755	9.67%	8579	9.45%	13565	11.92%	11102	10.07%	51104
9	2009	9	11014	12.25%	8657	8.58%	10107	11.14%	12863	11.30%	8595	7.80%	51236
10	2009	10	6001	6.67%	6504	6.45%	7644	8.42%	7302	6.42%	5896	5.35%	33347
11	2009	11	7346	8.17%	10433	10.34%	8843	9.74%	6165	5.42%	9031	8.19%	41818
12	2009	12	6124	6.81%	5120	5.08%	4929	5.43%	7714	6.78%	13972	12.68%	37859
13	2010	1	8548	8.22%	6409	6.43%	11585	11.22%	9142	11.24%	8321	8.61%	44005
14	2010	2	5449	5.24%	7677	7.71%	3445	3.34%	6328	7.78%	6503	6.73%	29402
15	2010	3	11845	11.39%	4223	4.24%	9943	9.63%	7001	8.60%	6789	7.03%	39801
16	2010	4	8228	7.91%	11208	11.25%	7213	6.99%	6194	7.61%	8701	9.01%	41544
17	2010	5	10069	9.68%	9154	9.19%	8709	8.43%	6439	7.91%	7728	8.00%	42099
18	2010	6	8519	8.19%	4516	4.53%	9555	9.25%	5244	6.44%	6466	6.69%	34300
19	2010	7	6861	6.60%	12682	12.73%	12709	12.31%	3821	4.70%	7580	7.85%	43653
20	2010	8	6830	6.57%	8774	8.81%	9137	8.85%	7926	9.74%	13786	14.27%	46453
21	2010	9	9805	9.42%	8897	8.93%	5660	5.48%	6915	8.50%	6433	6.66%	37710
22	2010	10	13623	13.10%	7286	7.31%	8444	8.18%	9471	11.64%	8617	8.92%	47441
23	2010	11	6448	6.20%	10598	10.64%	7529	7.29%	6231	7.66%	7920	8.20%	38726
24	2010	12	7807	7.50%	8205	8.24%	9322	9.03%	6658	8.18%	7766	8.04%	39758
25	合计		193943		200510		194018		195189		206828		990488

按季度统计不同产品的销售数量，以及每个产品在该季度总销量中所占的比例：

	产品	2009年								2010年								合计
		1季度		2季度		3季度		4季度		1季度		2季度		3季度		4季度		
		销量	占比	销量	占比	销量	占比	销量	占比	销量	占比	销量	占比	销量	占比	销量	占比	
1	PD01	24787	19.31%	21332	17.41%	24321	17.17%	19471	17.23%	25842	22.83%	26816	22.74%	23496	18.38%	27878	22.14%	193943
2	PD02	33477	26.07%	20128	16.43%	25219	17.81%	22057	19.52%	18309	16.17%	24878	21.09%	30353	23.75%	26089	20.72%	200510
3	PD03	22776	17.74%	21801	17.79%	24774	17.49%	21416	18.95%	24973	22.06%	25477	21.60%	27506	21.52%	25295	20.09%	194018
4	PD04	21717	16.91%	32629	26.63%	38292	27.04%	21181	18.74%	22471	19.85%	17877	15.16%	18662	14.60%	22360	17.76%	195189
5	PD05	25639	19.97%	26651	21.75%	29029	20.50%	28899	25.57%	21613	19.09%	22895	19.41%	27799	21.75%	24303	19.30%	206828
6	合计	128396	100.00%	122541	100.00%	141635	100.00%	113024	100.00%	113208	100.00%	117943	100.00%	127816	100.00%	125925	100.00%	990488

1.5.3.4 表达式列

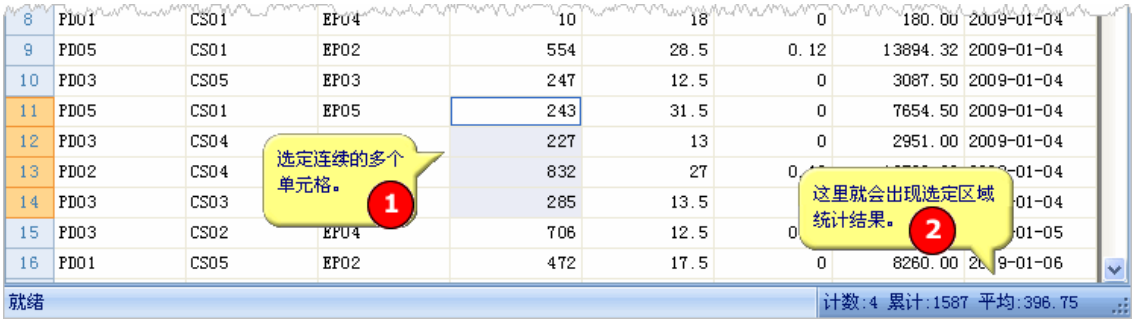
除常规的数据列外，你还可以增加表达式列，表达式列可以设置计算公式，其内容由其它列计算得出。

例如下面的订单表，其金额列就是一个表达式列，其计算公式为：数量 * 单价 * (1 - 折扣)，这样只要输入数量、单价和折扣，金额即可自动计算得出：

	产品	客户	雇员	数量	单价	折扣	金额	日期
1	PD05	CS03	EP04	650	30	0.05	18525.00	2009-01-02
2	PD03	CS04	EP01	610	13.5	0	8235.00	2009-01-02
3	PD02	CS04	EP04	342	25.5	0	8721.00	2009-01-02
4	PD01	CS01	EP05	968	18	0.05	16552.80	2009-01-03
5	PD02	CS02	EP04	733	27.5	0	20157.50	2009-01-03
6	PD03	CS03	EP04	595	12.5	0	7437.50	2009-01-03
7	PD05	CS02	EP02	698	28.5	0.12	17505.84	2009-01-04
8	PD01	CS01	EP04	10	18	0	180.00	2009-01-04
9	PD05	CS01	EP02	554	28.5	0.12	13894.32	2009-01-04
10	PD03	CS05	EP03	247	12.5	0	3087.50	2009-01-04
11	PD05	CS01	EP05	243	31.5	0	7654.50	2009-01-04
12	PD03	CS04	EP02	227	13	0	2951.00	2009-01-04

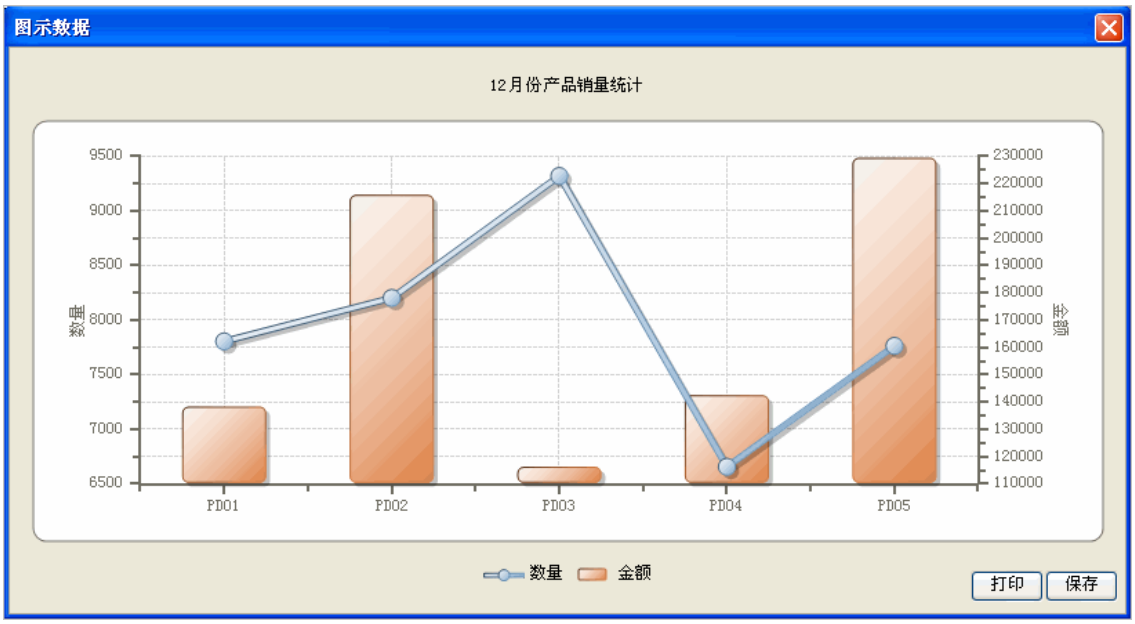
1.5.3.5 自动计算

在默认情况下，选定连续的多个数值单元格，状态栏右侧会出现选定单元格的数量、累计值和平均值：



1.5.3.6 生成图表

Foxtable 提供了功能强大、使用简单的图表生成功能，生成的图表还可以插入到报表中，这样就可以生成图文并茂的报表。



1.5.4 轻松报表

1.5.4.1 票据设计

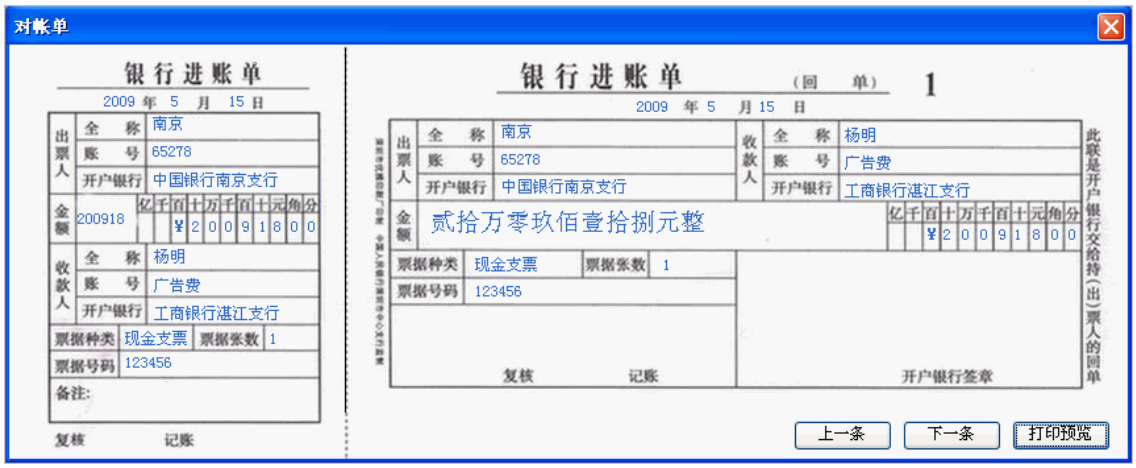
Foxtable 提供了可视化的票据设计功能，任何人都可以轻松设计出各种票据：





1.5.4.2 票据套打

Foxtable 可以很轻松地设计出各种票据套打程序，下面是两个例子。
进账单套打：



支票套打：



1.5.4.3 Excel 报表

Foxtable 可以直接利用 Excel 生成报表，你只需根据指定的规则设计好一个模板文件即可；Excel 报表功能灵活强大，几乎可以设计出任意复杂的报表，而且使用简单、便捷，即使不会 Excel 的用户，也能熟练掌握并自如应用。

例如你要打印工资条，只需利用 Excel 定义一个工资条模板：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															<END>
2	<工资,7>	单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴	
3		[单位]	[姓名]	[职务工资]	[级别工资]	[基础工资]	[工龄工资]	[奖金津贴]	[教龄津贴]	[教护工资]	[特殊津贴]	[物价补贴]	[技术津贴]	[山区津贴]	
4		适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资		
5		[适当补贴]	[保健津贴]	[从优待警]	[警衔津贴]	[补差工资]	[补发工资]	[年终奖金]	[应发工资]	[扣所得税]	[养老保险]	[扣发合计]	[实发工资]		
6	<工资>														
7	<END>														

Foxtable 即可根据工资条模板自动批量生成和打印工资条：

单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
县委办	韦国	440	408	230	31					52	10	120
适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
57							1348	24.8		24.8	1323.2	

单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
县委办	沈九	315	281	230	23					52	10	120
适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
55							1086	11.7		11.7	1074.3	

单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
县委办	黄伟	275	231	230	17					52	10	120
适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
55							990	6.9		6.9	983.1	

单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
县府办	罗永	180	158	230	7					52	8	95
适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
52							782				782	

单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
县府办	覃恒	213	231	230	28					52	8	80
适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
50							892	2		2	890	

更多例子：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>			湛江佳信公司出库单		在Excel中定义的出库单模板		
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:[%出库明细,Sum(数量)]			
9	<END>							



示例二

在Excel中定义的资料卡模板

	C	D	E	F	G
1					<End>
2	<员工, 2>	员工资料卡			
3				<"编号:" & [编号]>	
4					
5	姓名	[姓名]	出生日期	[出生日期]	
6	部门	[部门]	雇用日期	[雇佣日期]	
7	性别	[性别]	职务	[职务]	[&照片, 3, 3, 150, 159]
8	地址	[地址]			
9	家庭电话	[家庭电话]	办公电话	[办公电话]	
10	[备注]				
11					
12	<员工>				
13	<End>				

根据上面的模板生成的资料卡

员工资料卡

编号:1

姓名	王伟	出生日期	1962-2-19
部门	商务部	雇用日期	1992-8-14
性别	男	职务	副总裁(销售)
地址	罗马花园 890 号		
家庭电话	(010) 65559482	办公电话	3457



王伟获南京大学商业学士学位，获该校国际营销博士学位。他能说流利的法语和意大利语并能阅读德语。他加入公司时是销售代表，被提拔为销售经理并升任销售副总裁。王伟是销售管理圆桌协会，北京商业总会和太平洋周边进口协会的成员。

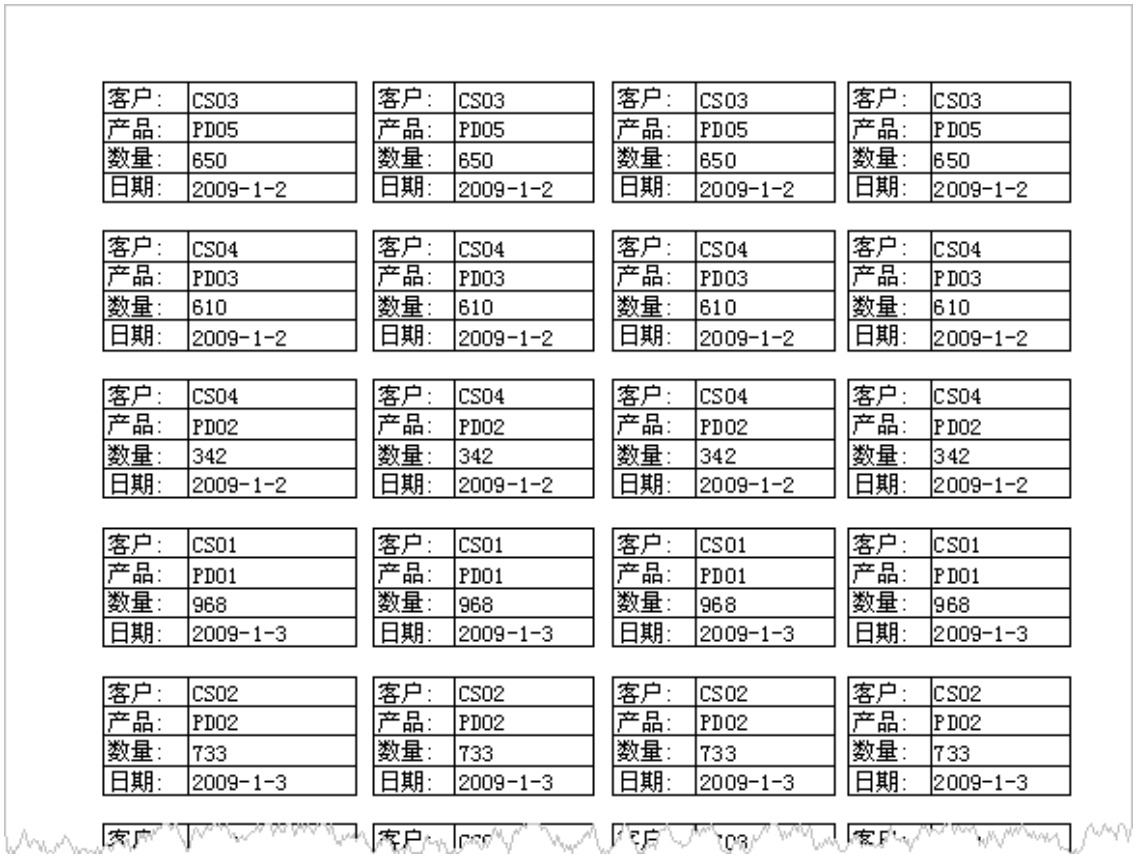
1.5.4.4 标签打印

Excel 报表还可以很方便地批量打印标签。

例如只需按下图定义一个模板：

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													<END>
2	<订单, 11>	客户:	[客户]		客户:	[客户]		客户:	[客户]		客户:	[客户]	
3		产品:	[产品]		产品:	[产品]		产品:	[产品]		产品:	[产品]	
4		数量:	[数量]		数量:	[数量]		数量:	[数量]		数量:	[数量]	
5		日期:	[日期]		日期:	[日期]		日期:	[日期]		日期:	[日期]	
6	<订单>												
7	<END>												

即可批量打印标签：



客户:	CS03
产品:	PD05
数量:	650
日期:	2009-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	2009-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	2009-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	2009-1-2

客户:	CS04
产品:	PD03
数量:	610
日期:	2009-1-2

客户:	CS04
产品:	PD03
数量:	610
日期:	2009-1-2

客户:	CS04
产品:	PD03
数量:	610
日期:	2009-1-2

客户:	CS04
产品:	PD03
数量:	610
日期:	2009-1-2

客户:	CS04
产品:	PD02
数量:	342
日期:	2009-1-2

客户:	CS04
产品:	PD02
数量:	342
日期:	2009-1-2

客户:	CS04
产品:	PD02
数量:	342
日期:	2009-1-2

客户:	CS04
产品:	PD02
数量:	342
日期:	2009-1-2

客户:	CS01
产品:	PD01
数量:	968
日期:	2009-1-3

客户:	CS01
产品:	PD01
数量:	968
日期:	2009-1-3

客户:	CS01
产品:	PD01
数量:	968
日期:	2009-1-3

客户:	CS01
产品:	PD01
数量:	968
日期:	2009-1-3

客户:	CS02
产品:	PD02
数量:	733
日期:	2009-1-3

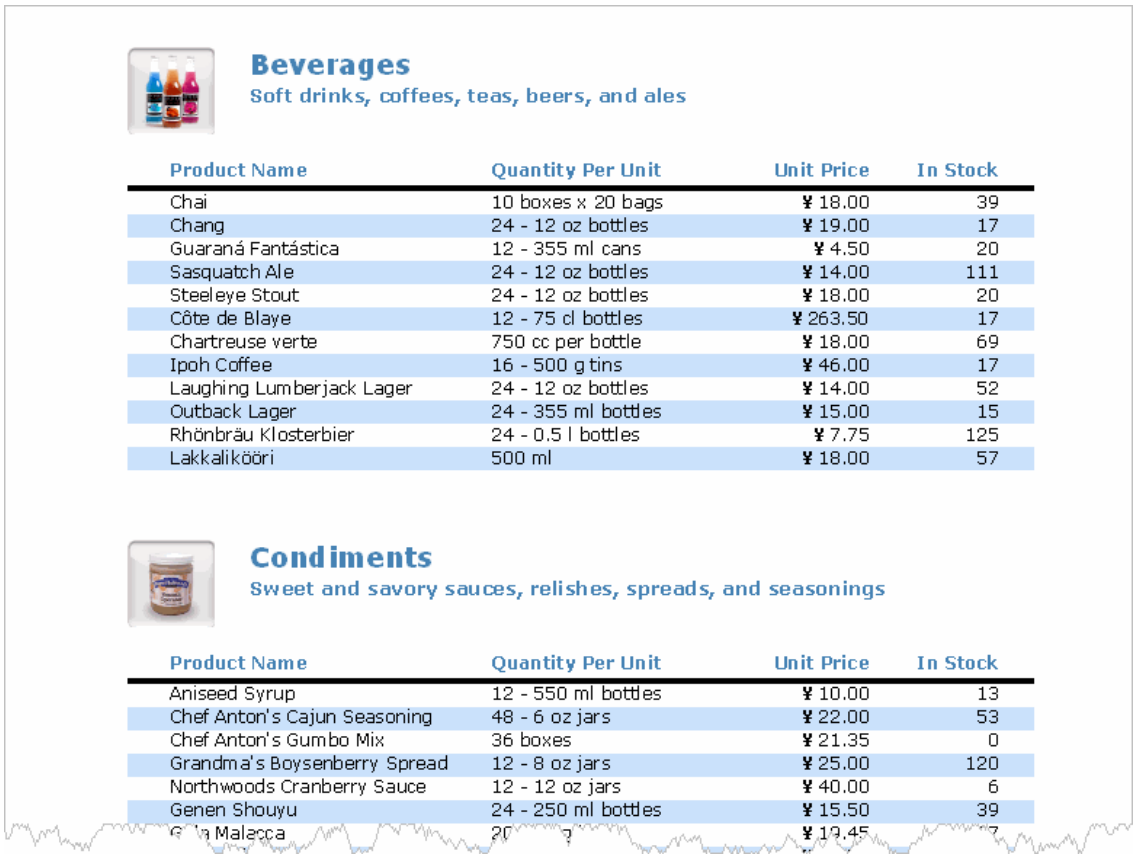
客户:	CS02
产品:	PD02
数量:	733
日期:	2009-1-3

客户:	CS02
产品:	PD02
数量:	733
日期:	2009-1-3

客户:	CS02
产品:	PD02
数量:	733
日期:	2009-1-3

1.5.4.5 专业报表

Foxtable 还提供了一个专业报表工具，这是一个功能非常强大的报表开发工具，如果能够掌握的话，我们设计出的报表，可以达到很多专业报表开发软件的水准：



Beverages
Soft drinks, coffees, teas, beers, and ales

Product Name	Quantity Per Unit	Unit Price	In Stock
Chai	10 boxes x 20 bags	¥ 18.00	39
Chang	24 - 12 oz bottles	¥ 19.00	17
Guaraná Fantástica	12 - 355 ml cans	¥ 4.50	20
Sasquatch Ale	24 - 12 oz bottles	¥ 14.00	111
Steeleye Stout	24 - 12 oz bottles	¥ 18.00	20
Côte de Blaye	12 - 75 cl bottles	¥ 263.50	17
Chartreuse verte	750 cc per bottle	¥ 18.00	69
Ipoh Coffee	16 - 500 g tins	¥ 46.00	17
Laughing Lumberjack Lager	24 - 12 oz bottles	¥ 14.00	52
Outback Lager	24 - 355 ml bottles	¥ 15.00	15
Rhönbräu Klosterbier	24 - 0.5 l bottles	¥ 7.75	125
Lakkalikööri	500 ml	¥ 18.00	57

Condiments
Sweet and savory sauces, relishes, spreads, and seasonings

Product Name	Quantity Per Unit	Unit Price	In Stock
Aniseed Syrup	12 - 550 ml bottles	¥ 10.00	13
Chef Anton's Cajun Seasoning	48 - 6 oz jars	¥ 22.00	53
Chef Anton's Gumbo Mix	36 boxes	¥ 21.35	0
Grandma's Boysenberry Spread	12 - 8 oz jars	¥ 25.00	120
Northwoods Cranberry Sauce	12 - 12 oz jars	¥ 40.00	6
Genen Shouyu	24 - 250 ml bottles	¥ 15.50	39
G[illegible] Malacca	20 [illegible]	¥ 19.45	7

1.5.5 轻松查询

1.5.5.1 筛选树

Foxtable 可以根据任意列的内容自动生成筛选树，只需单击筛选树的某个节点，即可自动筛选出对应的行：

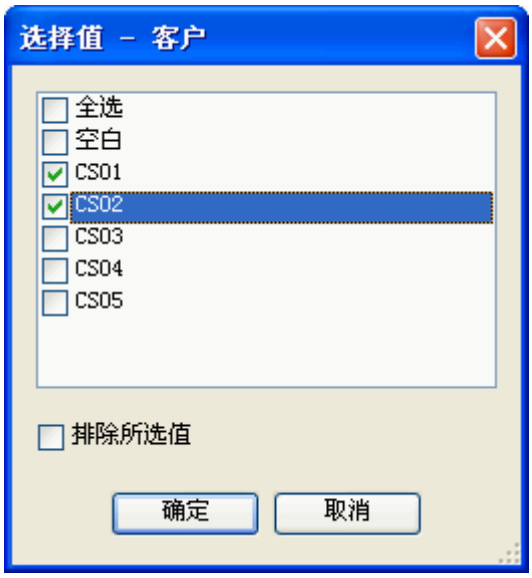


	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
2	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
3	PD01	CS02	EP05	14.4	0.15	150	1836	1999-02-22
4	PD01	CS02	EP03	14.4	0	200	2880	1999-02-26
5	PD01	CS02	EP02	18	0	100	1800	1999-03-04
6	PD01	CS02	EP01	14.4	0	700	10080	1999-03-17
7	PD01	CS02	EP03	18	0	400	7200	1999-03-19
8	PD01	CS02	EP02	14.4	0	60	864	1999-04-03
9	PD01	CS02	EP03	18	0.25	600	8100	1999-04-15
10	PD01	CS02	EP01	14.4	0.05	160	2188.8	1999-04-17
11	PD01	CS02	EP01	18	0	100	1800	1999-05-07
12	PD01	CS02	EP03	14.4	0.2	450	5184	1999-05-28
13	PD01	CS02	EP01	18	0	300	5400	1999-06-30

1.5.5.2 按值筛选

在任何一列，执行菜单中的“选择”命令，即可自动列出该列所有值供选择筛选。

例如选择订单表的客户列，执行选择命令，会列出所有客户供选择，如果选择客户 CS01 和 CS02:



即可自动筛选出 CS01 和 CS02 两个客户的订单：

	产品	客户	雇员	数量	单价	折扣	金额	日期
1	PD01	CS01	EP05	968	18	0.05	16552.80	2009-01-03
2	PD02	CS02	EP04	733	27.5	0	20157.50	2009-01-03
3	PD05	CS02	EP02	698	28.5	0.12	17505.84	2009-01-04
4	PD01	CS01	EP04	10	18	0	180.00	2009-01-04
5	PD05	CS01	EP02	554	28.5	0.12	13894.32	2009-01-04
6	PD05	CS01	EP05	243	31.5	0	7654.50	2009-01-04
7	PD03	CS02	EP04	706	12.5	0.12	7766.00	2009-01-05
8	PD02	CS01	EP02	176	27.5	0	4840.00	2009-01-08
9	PD01	CS02	EP03	121	17.5	0	2117.50	2009-01-11

1.5.5.3 自动筛选

1.5.5.3.1 字符列筛选

假定我们在客户列中，选择一个单元格，单元格的内容为“CS03”，单击“筛选”按钮，出现的下拉菜单为：



1、如果你要筛选出“CS03”客户的所有订单，那么单击“等于 CS03”即可。
2、如果你要筛选出非“CS03”客户的所有订单，那么单击“不等于 CS03”即可。
3、指向“文本筛选”，会列出很多命令，例如等于、包含、类似等等，几乎囊括了任何可能的文本筛选命令。
4、例如你要筛选的是客户名称中包括“联想”二字的记录，可以单击“包含”命令，出现下图所示的窗口，输入“联想”，单击“确定”即可：



1.5.5.3.2 日期列筛选

假定我们在日期列中，选择一个单元格，单元格的内容为“1999-01-04”。
1、单击“筛选”按钮，首先出现的是一些和选定单元格的日期进行比较的命令，假如你要筛选出1999 年 1 月 4 日的所有订单，单击命令“等于 1999-01-04”即可。



2、指向“指定时段”命令，会出现下图所示的菜单，利用这些命令，你可以筛选出指定月份或季度的数据，例如你要筛选出 3 月份的所有订单，只需单击命令“三月”即可：



3、指向“相同时段”命令，会出现下图所示的菜单，利用这些命令，你可以筛选出和选定行处于同一时段范围内的所有行，例如你要筛选出和 1999-01-04 这一天属于同一周的订单，单击命令“同周”即可。



4、指向“日期筛选”，会列出很多命令，例如等于、早于、晚于等等，囊括了所有可能的日期筛选命令：

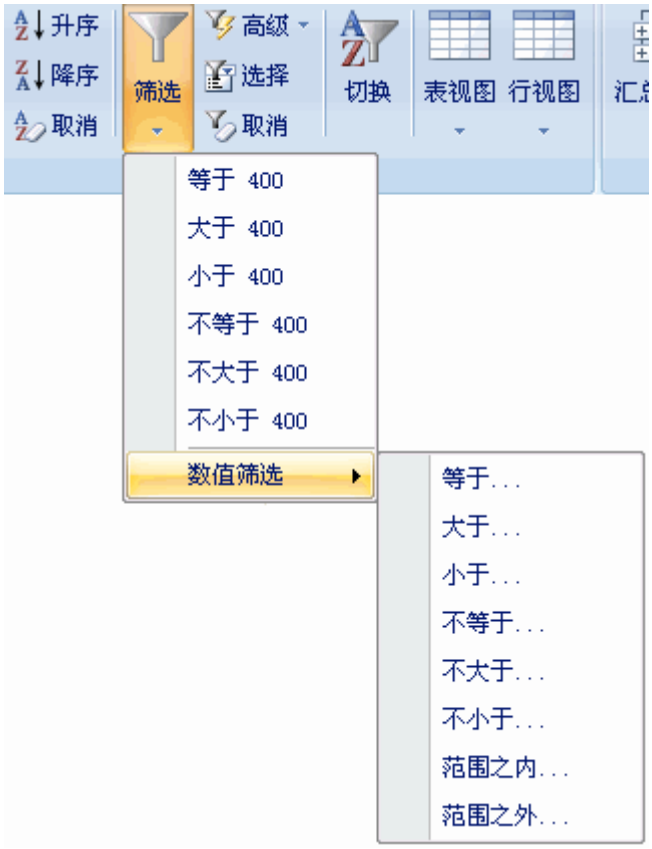


例如你要筛选 1999-02-01 到 1999-02-10 这一日期范围内的订单，只需单击“范围之内”命令，输入起始日期和终止日期，单击“确定”即可：



1.5.5.3.3 数值列筛选

假定我们在数量列中，选择一个单元格，单元格的内容为 400，单击“筛选”按钮，出现的下拉菜单为：



1、首先是一些和选定单元格的值进行比较的命令，例如你要筛选出订货数量超过 400 的订单，单击命令“大于 400”即可。

2、指向数值筛选，会列出很多命令，例如等于、小于、大于等等，几乎囊括了所有可能的数值筛选命令。

3、 例如你要筛选出订货数量在 600 到 800 之间的订单，可以单击命令“范围之内”，出现下图所示的窗口，分别输入起始值和终止值，单击“确定”即可：



1.5.5.4 高级筛选

Foxtable 提供了高级筛选功能，可以同时输入多个筛选条件，例如在订单表筛选客户 CS01 订购产品 PD01 的订单：



1.5.5.5 重复值筛选

在输入数据的时候，难免会因为某些原因出现重复输入的数据。
Foxtable 的菜单提供三个重复值筛选命令，可以快速找出重复值或冗余值：

- **显示重复值**
  如果当前列中某一个单元格的内容是唯一的，那么该单元格所在行将被隐藏。
  也就是说单元格的内容至少在当前列中出现两次，该单元格所在的行才会显示。

- **排除重复值**
  如果当前列中有多个单元格具备某一相同的值，将只显示其中一个单元格所在的行。

- **显示冗余值**
  如果当前列中有 n(n>1)个单元格具备某一相同的值，将只显示其中 n-1 行。

例如表的初始状态：

	第一列	第二列
1	1	A
2	2	B
3	1	C
4	2	D
5	1	E
6	2	F
7	3	G
8	4	H

显示第一列的重复值：

	第一列 ↓	第二列
1	1	A
2	1	C
3	1	E
4	2	B
5	2	D
6	2	F

排除第一列的重复值：

	第一列 ↓	第二列
1	1	A
2	2	B
3	3	G
4	4	H

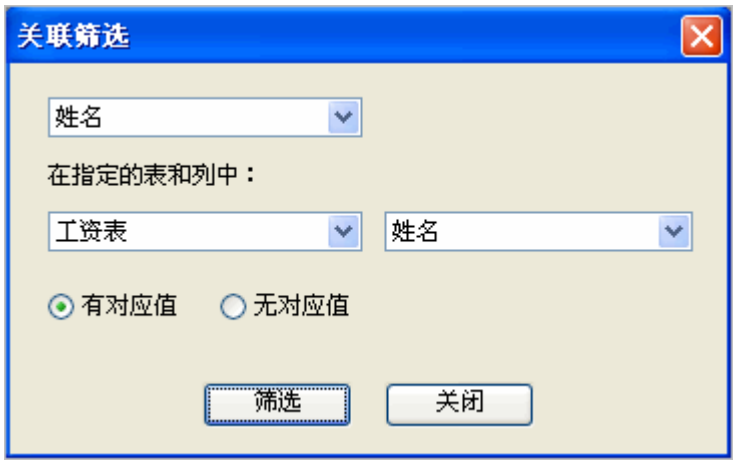
显示第一列的冗余值：

	第一列 ↓	第二列
1	1	C
2	1	E
3	2	D
4	2	F

1.5.5.6 关联筛选

有的时候需要在两个表之间进行关联性的筛选，例如在一个人事管理系统中，希望找出在基本信息表中有记录而在工资表中没有记录的员工。

Foxtable 提供了关联筛选功能，可以轻松完成这样的筛选任务，你可以在基本信息表中执行关联筛选，按下图所示设置即可：



1.5.5.7 其它筛选

- 手工隐藏行
  Foxtable 提供了直接隐藏选定行(或未选定行)的功能，这样可以完成一些并无明确逻辑关系的筛选任务。

- 锁定状态筛选
  Foxtable 可以筛选出已经被锁定的行，或筛选出未被锁定的行。

- 表达式筛选
  你也可以直接输入一个条件表达式进行筛选，来完成一个逻辑复杂，无法使用上述方式完成的筛选任务。

1.5.6 轻松开发

1.5.6.1 高效开发

Foxtable 不仅是一个优秀的应用软件，同时又是一个高效的.net 平台开发工具，专门针对数据管理软件的开发作了大量的优化，使得用户在开发过程中只需关注商业逻辑， 无需纠缠于具体功能的实现，这样 Foxtable 不仅开发效率十倍于其他专业开发工具，而且更加易用，几乎人人都能掌握。让普通人开发出专业水准的软件，以前是一个不可思议的想法，而 Foxtable 的出现，让这成为可能。
下面我们用一个简单的例子来说明 Foxtable 是如何高效的。
假定我们要根据订单表得到每月销售数量和金额，并计算出销量和金额环比增长率和同比增长率，这样的统计任务，即使是资深的程序员，也需要花相当长的时间才能完成编码工作，而且必须具备深厚的 SQL 语言基础：

	年	月	销量			金额		
			值	同比	环比	值	同比	环比
1	2009	1	41426			889732.04		
2	2009	2	44734		7.99%	984384.355		10.64%
3	2009	3	42236		-5.58%	910206.395		-7.54%
4	2009	4	43971		4.11%	916265.93		0.67%
5	2009	5	39369		-10.47%	851883.78		-7.03%
6	2009	6	39201		-0.43%	849708.06		-0.26%
7	2009	7	39295		0.24%	868029.02		2.16%
8	2009	8	51104		30.05%	1106204.245		27.44%
9	2009	9	51236		0.26%	1074093.76		-2.90%
10	2009	10	33347		-34.91%	688785.595		-35.87%
11	2009	11	41818		25.40%	906485.145		31.61%
12	2009	12	37859		-9.47%	870163.005		-4.01%
13	2010	1	44005	6.23%	16.23%	902920.705	1.48%	3.76%
14	2010	2	29402	-34.27%	-33.18%	669362.07	-32.00%	-25.87%
15	2010	3	39801	-5.77%	35.37%	795850.51	-12.56%	18.90%
16	2010	4	41544	-5.52%	4.38%	914346.175	-0.21%	14.89%
17	2010	5	42099	6.93%	1.34%	879864.425	3.28%	-3.77%
18	2010	6	34300	-12.50%	-18.53%	681936.645	-19.74%	-22.50%
19	2010	7	43653	11.09%	27.27%	908663.645	4.68%	33.25%
20	2010	8	46453	-9.10%	6.41%	1026431.4	-7.21%	12.96%
21	2010	9	37710	-26.40%	-18.82%	816583.08	-23.97%	-20.44%
22	2010	10	47441	42.26%	25.80%	979121.4	42.15%	19.90%
23	2010	11	38726	-7.39%	-18.37%	845087.095	-6.77%	-13.69%
24	2010	12	39758	5.02%	2.66%	842568.25	-3.17%	-0.30%

可是在 Foxtable 中，这样的任务是再简单不过了，只需简单几行代码即可：

```
Dim g As New GroupTableBuilder("统计表 2", DataTables("订单"))
g.Groups.AddDef("日期", DateGroupEnum.Year, "年")
g.Groups.AddDef("日期", "月")
g.Totals.AddDef("数量", "", "数量_值")
g.Totals.AddDef("金额", "", "金额_值")
g.SamePeriodGrowth = True
g.CircleGrowth = True
g.Build()
```

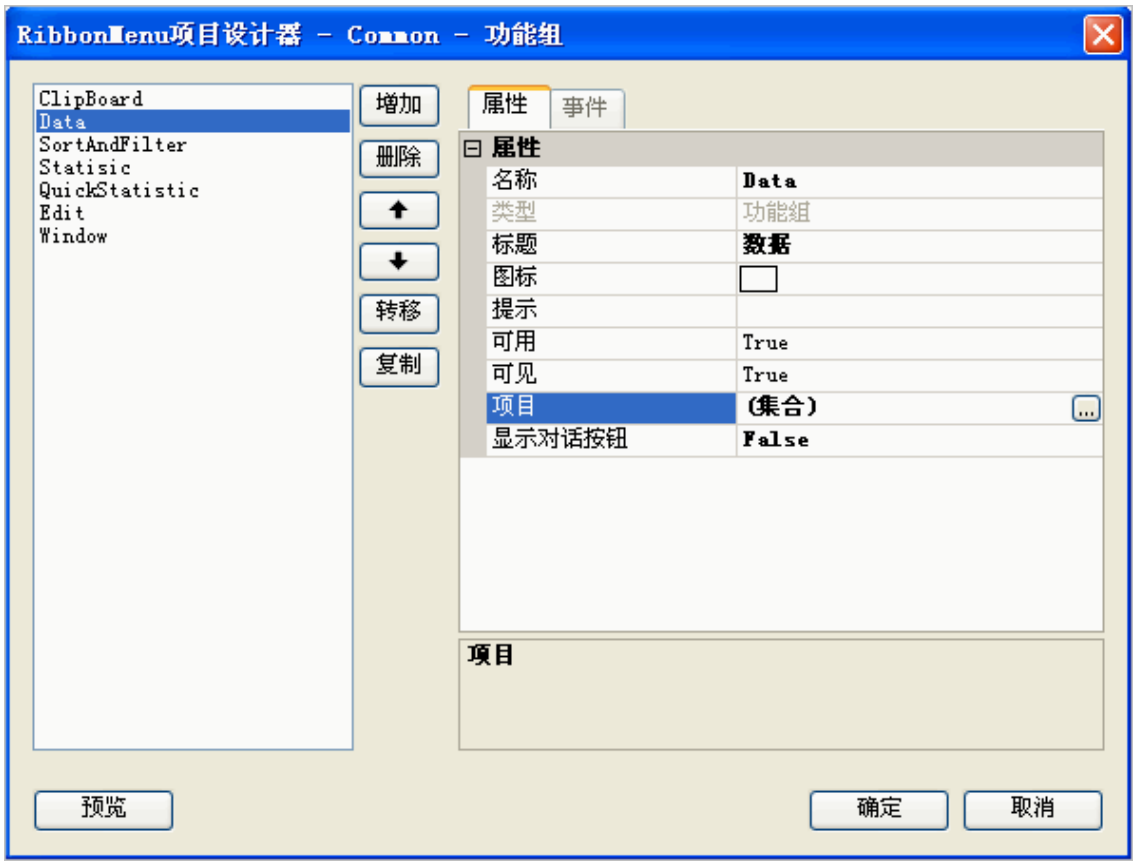
上面的代码不仅简单，而且逻辑清晰，任何人在帮助的指导下，都可以在几分钟内掌握，不需要任何专业背景。

你甚至不需要编写一行代码，只需通过菜单操作，单击几次鼠标，即可得到上面这种统计结果。

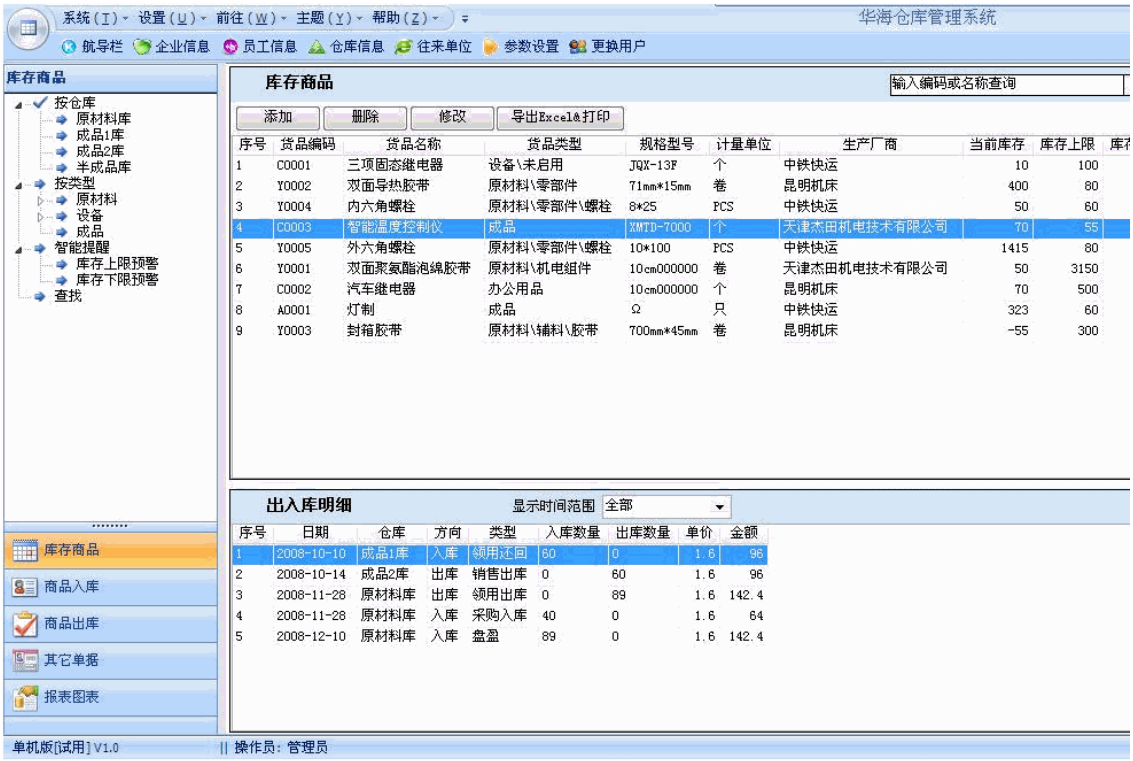
你还可以利用菜单自动生成代码，总之 Foxtable 在方方面面都为用户考虑周到，让普通用户也能设计出让专业程序员目瞪口呆的软件来。

1.5.6.2 菜单设计

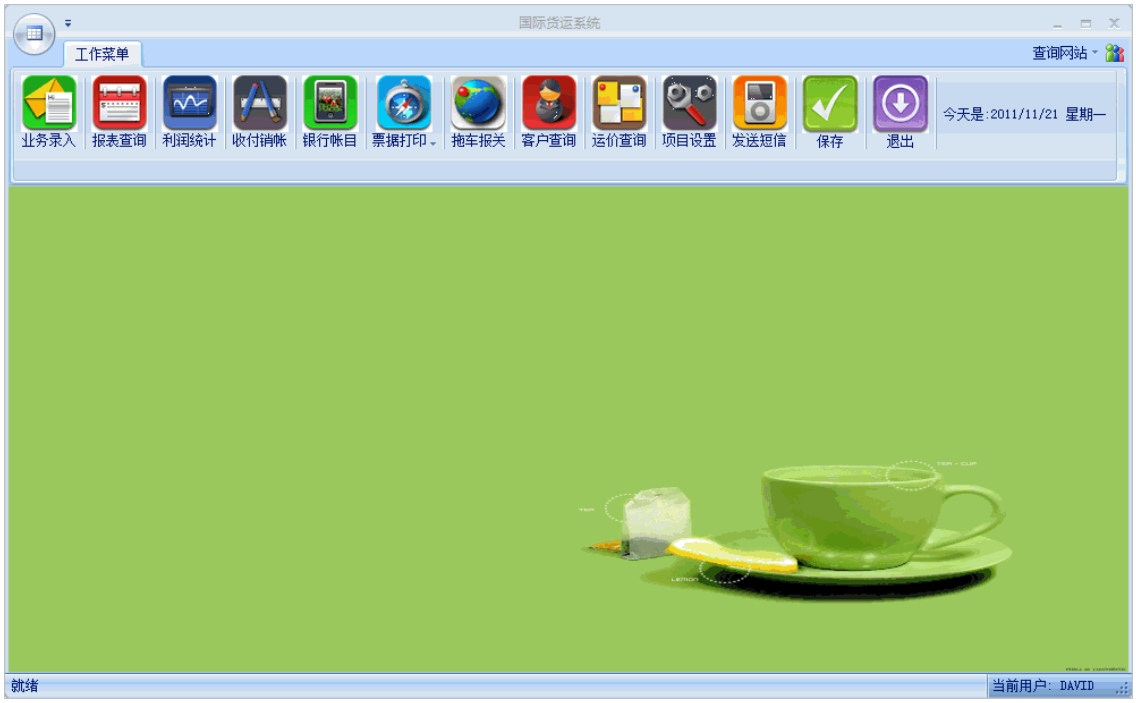
Foxtable 提供了一个功能强大，使用简单的菜单设计器：



Foxtable 的菜单设计很灵活，既可以设计传统风格的菜单：



也可以设计 Ribbon 风格的菜单：

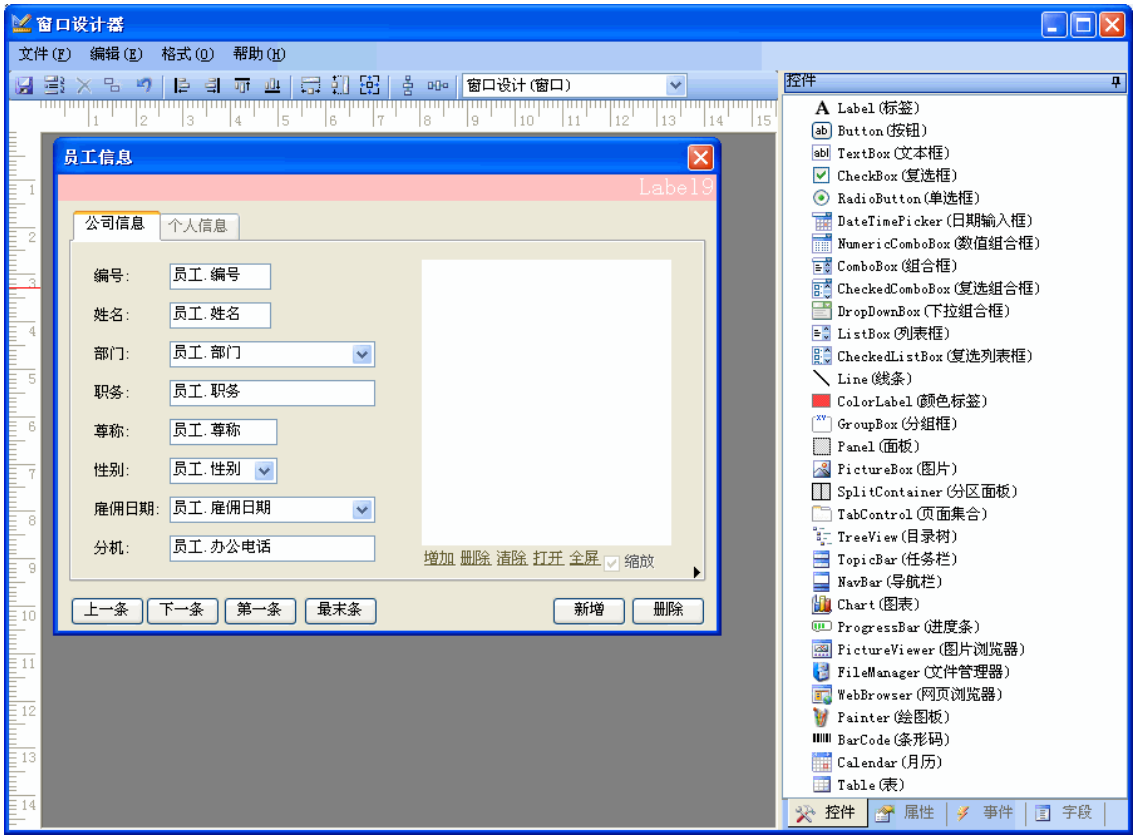


实际上，Foxtable 的系统菜单也是使用菜单设计器设计出来的，这也间接说明了这个菜单设计器是非常强大的 。

1.5.6.3 窗口设计

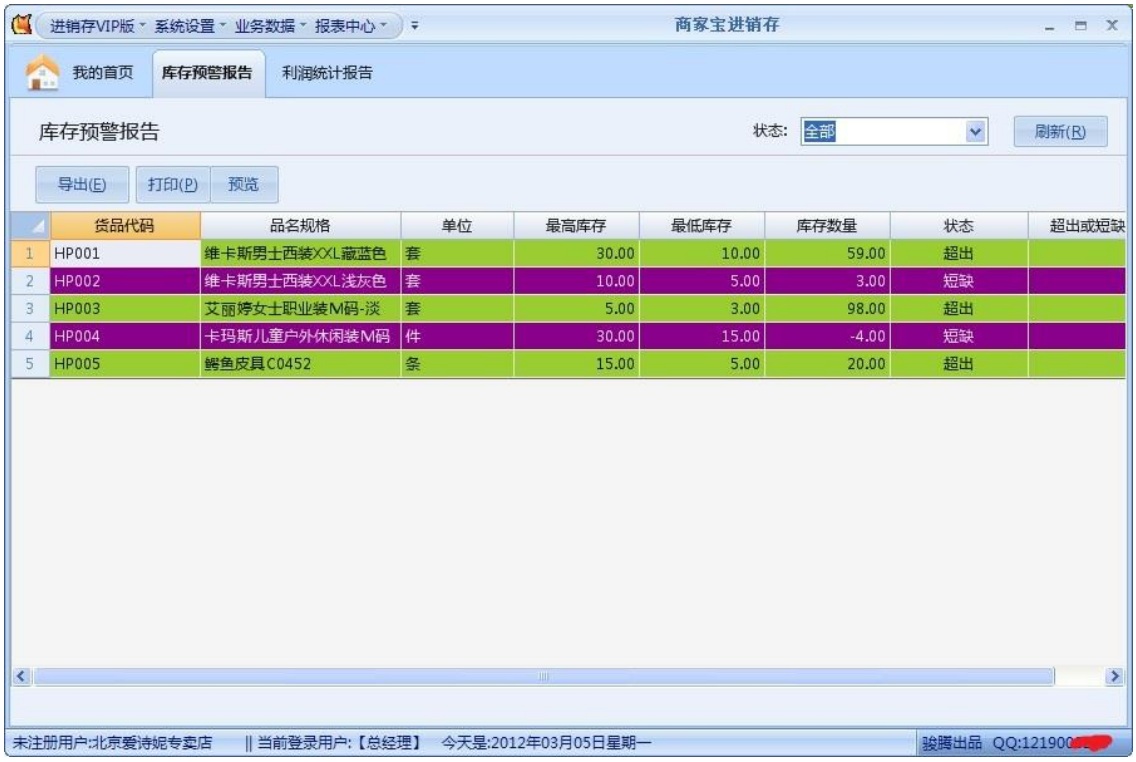
Foxtable 提供了一个功能强大的窗口设计器，有了 35 个控件，不少控件是专门针对数据管理而研发的。

这个窗口设计器使用很简单，大多数时候，只需将字段或者控件直接拖入窗口合适位置，并适当设置其属性即可：



有了窗口设计器，普通用户，可以轻松设计出各种专业的界面，例如下面这些窗口就是用窗口设计器设计出来的：







1.5.6.4 海量数据

Foxtable 不仅内建数据库，同时还支持 Access、SQL Server、Oracle 等主流数据库。

同一个管理项目，支持多个数据源，数据源可以预先通过菜单设置好，也可以在运行过程中动态创建和切换。

此外 Foxtable 还支持数据动态加载和分页加载，可以在运行过程中根据需要加载数据。

不管是分组统计还是交叉统计，不仅可以统计已经加载的数据，还可以直接统计后台数据。

这些特性使得 Foxtable 可以轻松应付海量数据的管理。

下图是 Foxtable 自动生成的一个加载树，单击某个节点，即可加载对应的数据，同时利用目录树下方的导航按钮，可以实现分页加载：



	产品	客户	雇员	单价	数量	日期	折扣
1	PD01	CS01	EP05	18	968	2009-01-03	0.05
2	PD01	CS01	EP04	18	10	2009-01-04	0
3	PD01	CS05	EP02	17.5	472	2009-01-06	0
4	PD01	CS02	EP03	17.5	121	2009-01-11	0
5	PD01	CS04	EP05	17.5	914	2009-01-12	0
6	PD01	CS02	EP04	19	600	2009-01-13	0.02
7	PD01	CS02	EP01	18.5	315	2009-01-15	0
8	PD01	CS01	EP02	18	492	2009-01-18	0
9	PD01	CS02	EP03	19	870	2009-01-23	0.12
10	PD01	CS03	EP01	18.5	27	2009-01-25	0
11	PD01	CS01	EP01	18	687	2009-01-26	0
12	PD01	CS02	EP05	18	93	2009-01-27	0
13	PD01	CS03	EP05	18.5	249	2009-02-01	0
14	PD01	CS04	EP03	17.5	814	2009-02-04	0
15	PD01	CS03	EP03	18	912	2009-02-05	0.15
16	PD01	CS05	EP03	17.5	128	2009-02-05	0
17	PD01	CS02	EP03	19	539	2009-02-06	0.12

上述这些特性，不仅可以让 Foxtable 管理海量数据，还可以设计基于互联网的应用，使得相距千里之外的电脑，也能协同处理数据。

1.5.6.5 无限扩展

Foxtable 可以添加 DLL 引用，包括第三方的 DLL，还可以定义命名空间。

Foxtable 可以定义 API 函数。

Foxtable 支持通过 VBA 直接操控 Excel，这样资深的 Excel 用户可以充分结合利用 Foxtable 和 Excel 的各自优势。

Foxtable 支持直接运行 SQL 语句。

所有这些，可以让你无限扩展 Foxtable 的功能。

1.5.6.6 系统迁移

Foxtable 提供了系统迁移工具，你可以将设计好的管理项目，随时在 Access、SQL Server、Oracle 和 Foxtable 内置数据库之间迁移，而你要做的只是单击几次鼠标那么简单。

1.5.6.7 发布项目

Foxtable 的开发版提供了发布功能，设计好的程序经过发布之后，可以生成可执行文件，脱离 Foxtable，直接复制到其它电脑运行。

下面是一个基于 Foxtable 开发的仓管软件，已经编译成可执行文件，并做成了安装程序，已经没有任何 Foxtable 的痕迹了：

http://www.foxtable.com/download/huahai.zip

1.5.6.8 短信收发

Foxtable 内置短信收发功能，可以方便地通过短信实现远程数据的采集和查询，以及远程审批功能 ，还可以直接以网页形式将查询结果返回到用户手机上。

1.5.6.9 权限管理和工作流

Foxtable 的权限管理非常细致，不仅可以控制某个表、某一行、某一列或某个单元格的内容能被那些用户编辑或查看；还可以伪装数据，使某单元格的数据在某些用户前面可以真实显示，例如 0.8，而另一些用户看到的却是伪装后的数据，例如 0.5，或者空白；不仅如此，任何一个菜单命令，任何一个按钮，任何一项操作，小到移动行，调整列宽这些细微的操作，都可以设置为必须经过授权才能执行；可以说，在 Foxtable 中， 没有一项操作是可以脱离授权控制的。

不仅如此，Foxtable 还可以随心所欲地定义工作流，远比那些所谓的工作流软件更为简洁和灵活，对于内部通讯的需求和并发冲突的处理，也能轻松应付。
我们见到不少用户，以为权限管理、工作流、远程上报有多复杂，多专业，需要花费大价钱购买专业软件才能实现，其实这些都是很基本的功能，有了 Foxtable，任何用户都可以轻松解决。

1.5.6.10 融合 B/S 和 C/S 架构优势

我们知道，B/S 结构的软件易于部署，而 C/S 结构的软件则能提供更好的交互性。

虽然鱼和熊掌不可兼得，但 Foxtable 却较好地融合了二者的优势， 既可以开发传统的 C/S 管理系统，也可以开发 B/S 管理系统，并提供了手机网页生成以及微信接入功能，可以非常快速地搭建手机端的管理系统。

同时对于开发好的 C/S 管理系统内置自动升级功能，完全无需人工维护。

1.5.7 移动开发

1.5.7.1 Web 服务

有了 Foxtable，你可以用非常简单的代码搭建一个 Web 服务，无需任何专业知识。
Foxtable 只需 3 行代码，即可搭建一个 Web 服务器：

```
HttpServer.Prefixes.Add("http://*/")
HttpServer.WebPath = "d:\web"
HttpServer.Start()
```

1.5.7.2 手机网页

Foxtable 内置手机网页自动生成功能，你可以完全不懂网页设计，也能开发出非常专业的基于网页的手机管理系统：





1.5.7.3 JSON 解析

Foxtable 用非常方便的方法实现 JSON 和 XML 的解析与生成。
在命令窗口执行：

```
Dim json As String = "{'name':'李云龙','age':'36'}"
Dim jo As JObject = JObject.Parse(json)
Output.Show(jo("name"))
Output.Show(jo("age"))
```

显示的结果是：

```
李云龙
36
```

在命令窗口执行：

```
Dim xml As String = "<xml><to>George</to><from>John</from><content>Don't forget the meeting!</content></xml>"
Dim xo As XObject = XObject.Parse(xml)
Output.Show(xo("to"))
Output.Show(xo("from"))
Output.Show(xo("content"))
```

显示的结果为：

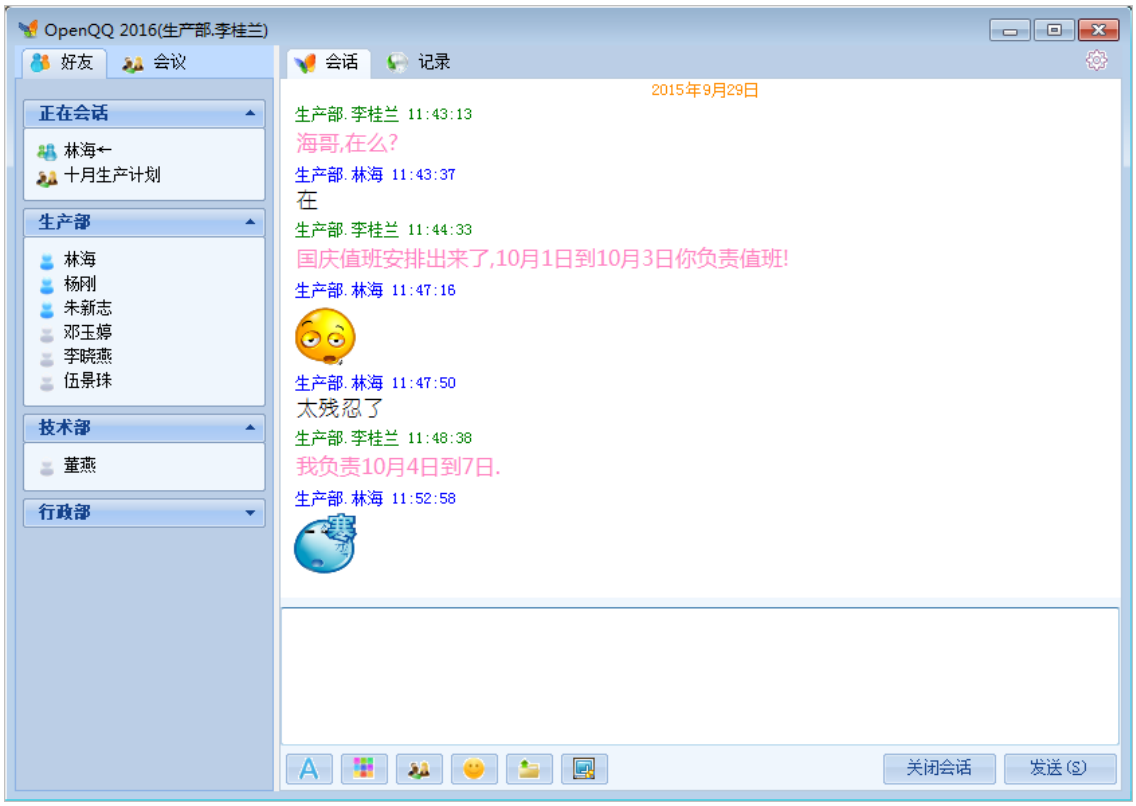
```
George
John
Don't forget the meeting!
```

1.5.7.4 微信接入

Foxtable 内置了微信开发功能，不需要专业专业知识，任何人都可以轻松地将微信整合到自己的管理系统中，可以通过微信收发信息，通过微信输入、查询、统计和审核数据。

1.5.7.5 企业 IM

Foxtable 从 2016 版开始，内置了和 QQ 一样的即时通讯功能，称之为 OpenQQ。
OpenQQ 不仅可以像 QQ 一样，用于常规消息、图片和文件的收发，更重要的是，它可以和现有的管理系统紧密地整合在一起，可以用代码实现消息、图片和文件的自动收发。
在企业内部的管理系统中，整合一个和 QQ 一样的即时通讯系统，是很多 Foxtable 用户的梦想，OpenQQ 的出现，让您轻松实现。
从此，服务器和客户端之间，以及客户端和客户端之间，都可以进行消息的主动推送和接收了。



1.5.8 其他特色

1.5.8.1 合并模式

合并模式是一种清晰诱人的数据显示方式，在普通的电子表格软件中，也许要经过无数次合并单元格操作才能得到类似的效果，而在 Foxtable 中，这一切都是自动的。

你只需单击几次鼠标，即可任意选择要合并的列，随意进入和退出合并模式。

下图就是一个处在合并模式的表格：

	国家	金属	类型	数量
1	美国	白银	出口	70
2			进口	321
3		黄金	出口	78
4			进口	789
5	日本	白银	出口	123
6			进口	167
7		黄金	出口	567
8			进口	400
9	中国	白银	出口	70
10			进口	120
11		黄金	出口	70
12			进口	130

1.5.8.2 多层表头

对于一些结构复杂的数据，传统的数据库软件往往束手无策，而用普通的电子表格软件，却又会失去很多灵活性；而 Foxtable 作为一个纯粹的数据库软件，其表头可以像电子表格一样分层显示，从而让您鱼与熊掌兼得。

即使你采用 Access、Sql Server、Oracle 作为 Foxtable 的数据源，表头同样可以分多层显示，而且不需要对原来的表结构做任何的修改，只需在 Foxtable 中为各列重新指定一个列标题即可。

	产品	东部				南部			
		一季度	二季度	三季度	四季度	一季度	二季度	三季度	四季度
1	PD01	7177	6378	6390	6849	7870	3035	3959	5488
2	PD02	5181	4730	6159	6299	5152	4128	4314	3601
3	PD03	4473	4497	6976	6536	4022	7582	5293	5042
4	PD04	6801	4298	5696	7039	7342	6809	5253	7409
5	PD05	3441	5016	6861	4359	4090	4394	6748	7250
6	PD06	3660	3779	6240	7613	3000	4018	3078	4237
7	PD07	7044	6240	7701	6291	6633	6204	6194	5136
8	PD08	6526	7981	5729	5638	6279	6777	5457	7634
9	PD09	3000	4105	4232	5939	6218	7940	7939	4605
10	PD10	5597	7121	5747	4695	6093	5200	5402	6860

1.5.8.3 表间关联

Foxtable 可以在表与表之间建立关联，而且具有一些其他数据管理软件所不具备的特性。

例如在订单管理系统中，通常包括客户、产品、订单三个表，显然订单表中的每一个订单，都分别在客户表和产品表中有对应的行，也就是说，每一个订单都会对应有一个客户和一个产品；同样，客户表或者产品表中每一行，会在订单表中对应有若干行(也就是订单)，而且订单表中的单价列是需要从产品表中提取的；如果我们能够在客户表与订单表、产品表与订单表之间建立关联，那么这些表就可以相互引用数据进行计算， 或快速查阅关联数据。

例如：

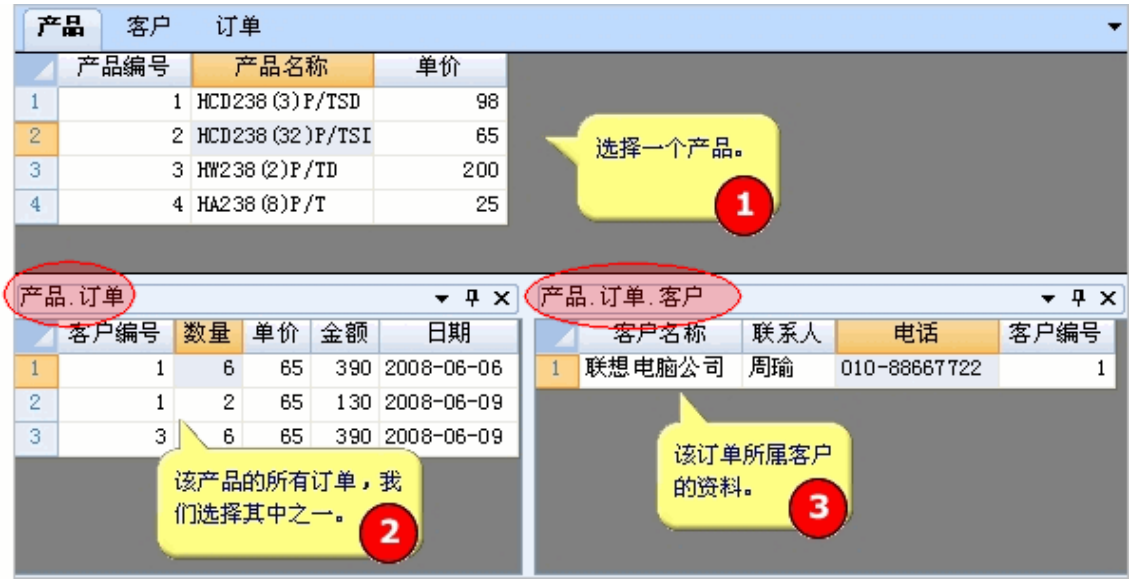
1、订单表可以引用产品表中的单价数据来计算金额。

2、在订单表中选择一个订单，即可用关联表的形式列出该订单所属的产品和客户。

3、在产品表中选择一个产品，即可用关联表的形式列出该产品所有的订单，而在该关联表中选择一个订单，会用另一个关联表列出该订单所属客户的资料。

4、在客户表中选择一个客户，即可用关联表的形式列出该客户所有的订单，而在该关联表中选择一个订单，会用另一个关联表列出该订单所订购产品的资料。

通过关联表，可以让所有相关数据唾手可得：



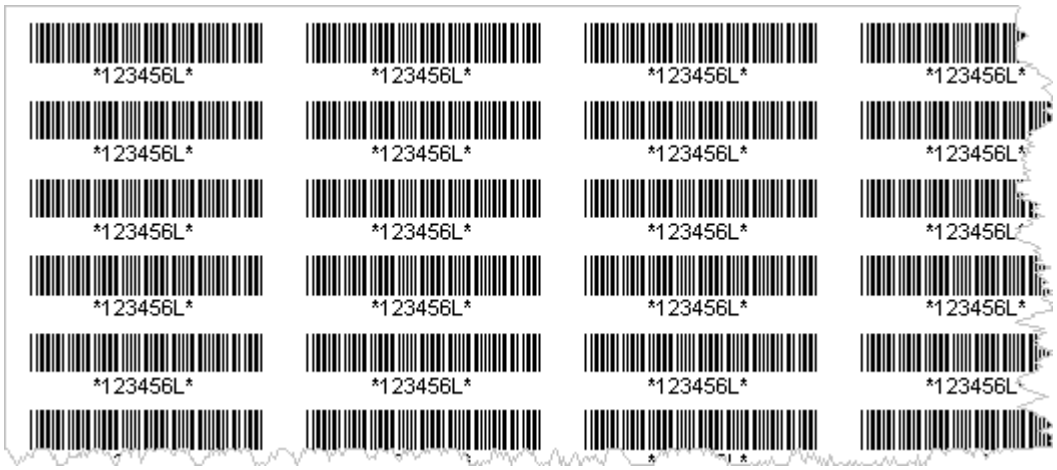
1.5.8.4 数据标记

Foxtable 提供了强大的自动数据标记功能，例如下图的成绩表，对于单科成绩低于 60 分的（语文数学为 90 分），会以红底白字标出，对于总分低于 400 分的，会以蓝底白字标出：

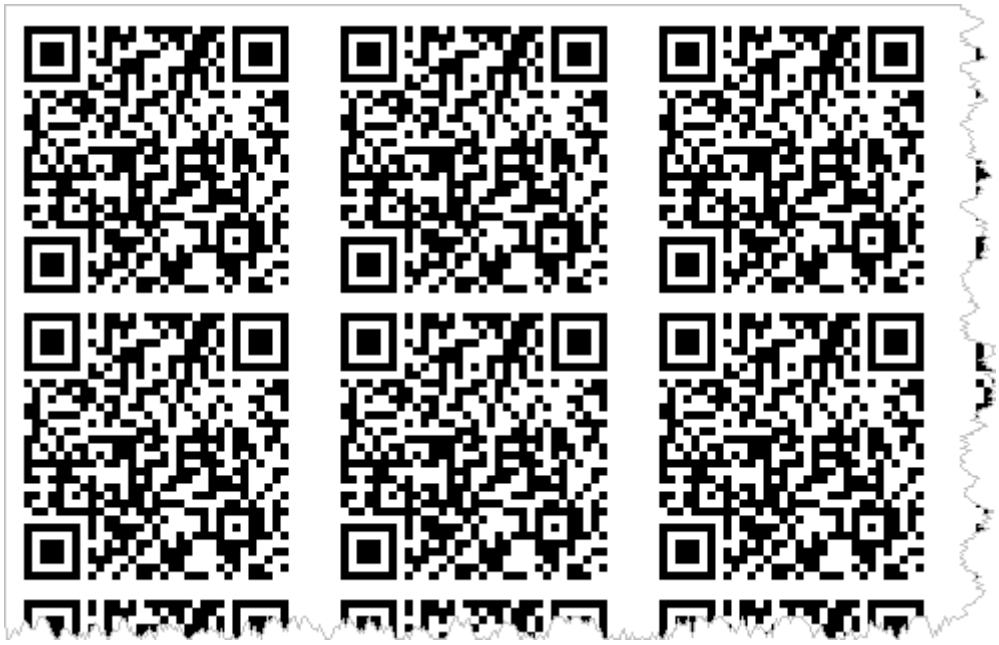
	班级	姓名	语文	数学	英语	化学	物理	总分
3	1班	陈思达	123	95	98	66	79	461.0000
4	1班	程康文	110	123	117	78	78	506.0000
5	1班	邓玉婷	118	58	74	62	77	389.0000
6	1班	林景阳	129	120	139	80	88	556.0000
7	1班	苏　捷	132	136	146	94	93	601.0000
8	1班	谢杰立	119	133	95	61	57	465.0000
9	1班	谢晓霞	129	109	126	80	66	510.0000
10	1班	杨景亮	116	120	132	83	78	529.0000
11	1班	张李华	124	118	124	88	83	537.0000
12	2班	陈成进	110	78	106	80	36	410.0000
13	2班	陈诗淇	113	123	117	78	78	509.0000
14	2班	陈思达	113	75	92	46	65	391.0000
15	2班	程康文	116	120	132	83	78	529.0000
16	2班	邓玉婷	125	109	90	78	77	479.0000
17	2班	林景阳	119	133	95	61	57	465.0000
18	2班	苏　捷	124	118	124	88	83	537.0000

1.5.8.5 条码输出

Foxtable 提供了非常强大的条形码功能，可以生成 91 种条形码，囊括了所有类型的条形码。
Foxtable 还可以生成各种二维码，例如 QRCode、DataMatrix、PDF417 等等。
Foxtable 可以生成中国移动格式的二维码图片，供手机直接扫描使用。
Foxtable 不仅可以在窗口、单据、报表中打印条形码，还提供了条形码批量打印工具。
批量打印的条形码：



批量打印的二维码：



1.5.8.6 扩展列类型

Foxtable 可以将字符型列扩展为其他类型，例如邮件、网址、文件、颜色等等。
例如下图的"个人主页"列已经扩展为网址型，你可以像平时一样输入内容，也可以单击单元格右侧的按钮打开网址：

	姓名	电子邮件	个人主页	照片
1	王伟	ww@21cn.net	www.egrid2000.com	EP2.bmp
2	张颖	zy@21cn.net	www.foxtable.com	EP1.bmp
3	李芳	lf@21cn.net	www.china.com	EP3.bmp
4	郑建杰	zjj@21cn.net	www.sohu.com	EP4.bmp
5	赵军	zhj@21cn.net	www.21cn.com	EP5.bmp
6	孙林	sl@21cn.net	www.it168.com	EP6.bmp
7	金士鹏	jsp@21cn.net	www.yinsha.com	EP7.bmp
8	刘英玫	lym@21cn.net	www.microsoft.com	EP8.bmp

2. 关于视频教程

我们并不建议您看视频来学习 Foxtable，我们在编写帮助的时候，是将用户当作零基础来看待的，一般用户，只需踏踏实实按顺序学习帮助文件都能掌握。

如果你在学习和使用过程遇到问题，可以在我们的论坛发帖提问，我们的工程师会非常乐意为您服务：

http://www.foxtable.com/support/

但确实有部分用户看帮助学习有困难，所以我们也提供了视频教程：

http://www.foxtable.com/video/

必须强调一下，任何开发工具，都没有办法通过视频完全掌握，视频只能带您入门，最好的学习方式就是四个字：啃透帮助。

Foxtable 已经将管理软件开发的门槛降低到人人都能掌握的程度，已经让大量的普通企业管理人员开发出很专业的管理系统，甚至让一些资深程序员也目瞪口呆，但我们依然没有办法让用户看看视频就能完全掌握。

3. 使用指南

3.1 关于菜单

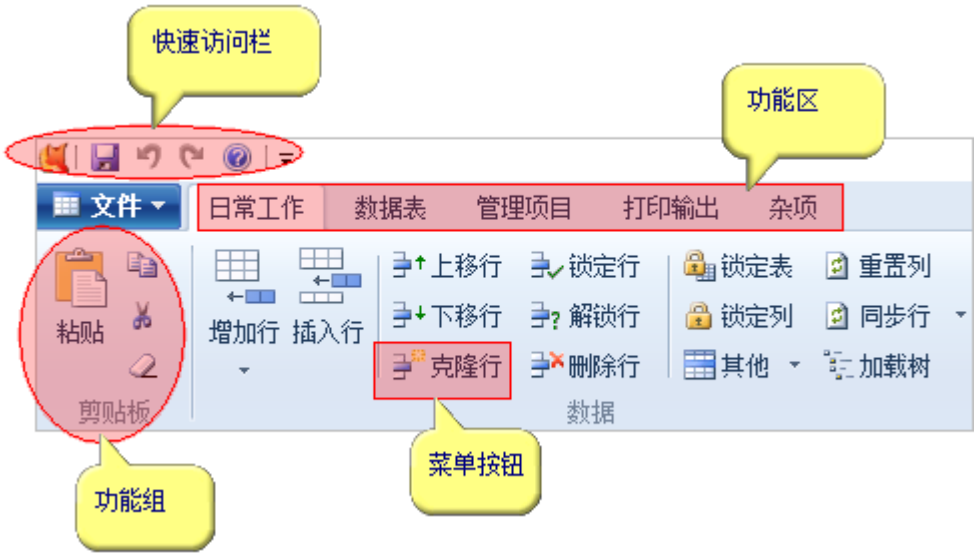
绝大多数的日常操作都是通过菜单来完成的，所以我们首先介绍一下 Foxtable 的菜单。

一、菜单的构成

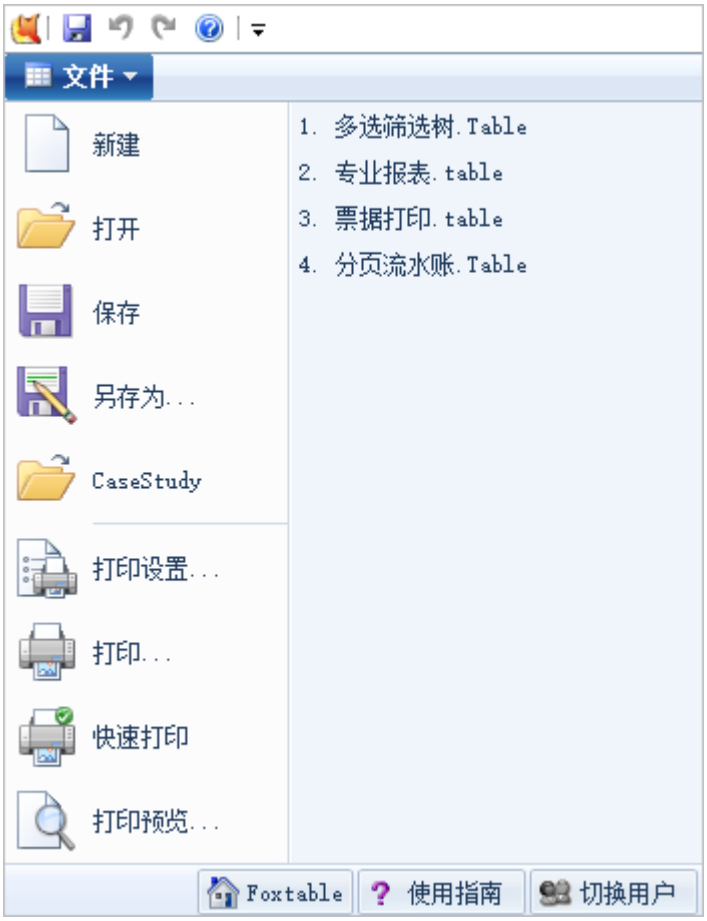
Foxtable 的菜单和传统的菜单有所不同，它是由功能区组成的。

系统菜单默认包括日常工作、数据表、管理项目、打印输出、杂项等五个功能区；而每一个功能区又是由功能组构成的，例如日常工作这个功能区，包含剪贴板、数据、排序与筛选、 数据统计、编辑、窗口等功能组；通过功能区和功能组，我们可以将各种按钮按照功能分类，有序地组合起来。

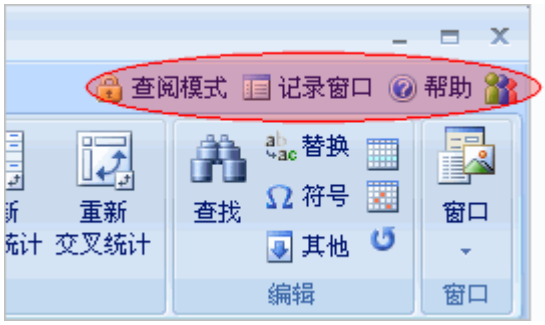
除了功能区外，菜单还包括程序菜单，快速访问栏和配置栏，如下图所示：



单击"文件"按钮会出现程序菜单，程序菜单包括与文件和打印相关的命令：



配置栏位于窗口的右上角：



初次使用，您可能会对这种风格的菜单布局感到不适应，其实这种风格是一种趋势，越来越多的软件会采用；相信你习惯之后，肯定会感到这样的布局其实是非常方便的 ；例如当你完成一个管理项目的设计工作后，之后录入数据、查询、筛选、统计、打印报表等几乎所有的日常工作，都可以在菜单的日常工作区完成，无需像传统的菜单那样，需要反复选择不同的菜单。

二、按钮的类型

按钮分为三种类型：

1、标准按钮

绝大部分按钮都是标准按钮，单击标准按钮即可执行某一项操作，例如复制、剪切、粘贴等按钮。

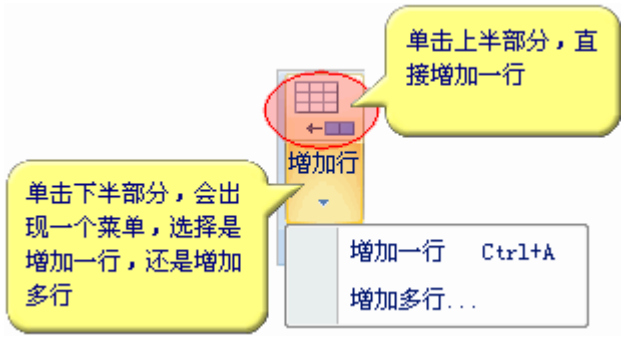
2、下拉按钮

单击下拉按钮将出现一个下拉菜单，用以选择具体的操作，例如功能组“排序与筛选”中的“表视图”按钮。

3、组合按钮

组合按钮其实就是标准按钮和下拉按钮的组合，组合按钮分成两部分，单击上半部分会立即执行某项操作，而单击下半部分，将出现一个下拉菜单，用以选择更多的类似操作。

例如“数据”功能组中的“增加行”按钮，就是一个组合按钮，单击按钮的上半部分，将直接增加一行，而单击按钮的下半部分，将出现一个下拉菜单，用以选择是增加一行还是增加多行，如下图所示：



之所以专门介绍一下按钮的种类，是因为不少用户会忽略组合按钮所包含的下拉菜单，而这些下拉菜单中有不少功能是相当重要的。

3.2 关于插图

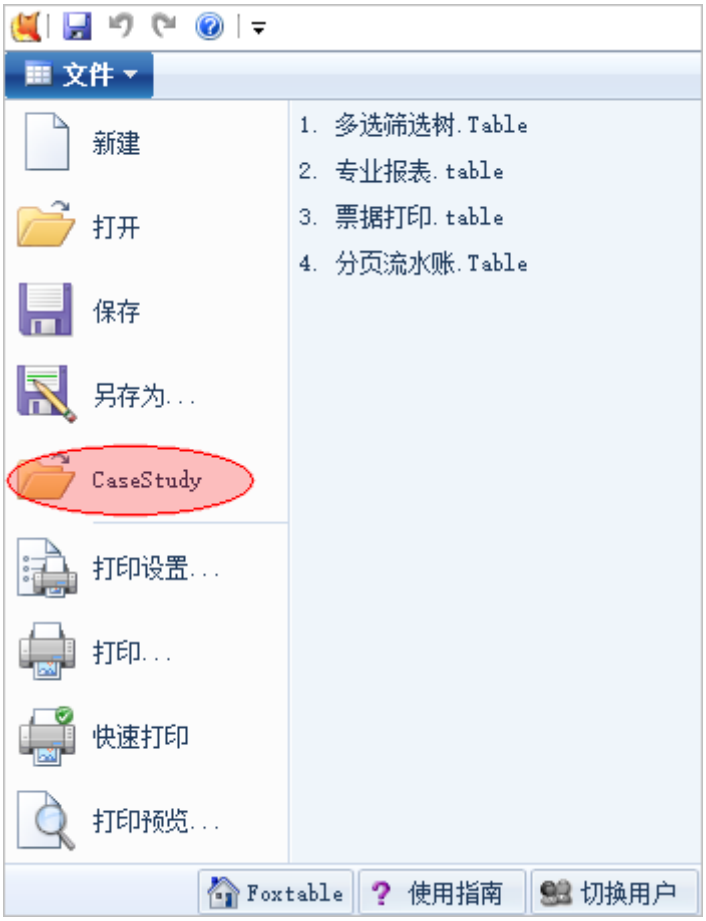
本帮助文件有超过一千幅的插图，可以让大家用更直观的方式了解 Foxtable 的使用方法。

由于 Foxtable 本身在不断的改进，所以少数插图和软件本身会有细节上的出入，不过这种出入不会影响正常的学习和理解。

3.3 关于示例文件

Foxtable 提供了 40 多个示例文件，用于大家配合本帮助文件学习使用。
这些示例文件中的相当部分，仅仅是一些数据而已，如果脱离了帮助，并无实际的意义。
大家在学习的过程中，可以使用这些示例文件，对于帮助介绍的各种操作或代码，进行实际的操作和验证，以加深印象，排除困惑。
帮助文件在讲述特定章节的时候，会提示你打开对应的示例文件。
例如在讲述数据统计之前，会首先提示：本章课程使用的示例文件为“统计演示.Table”，请在学习本章内容之前，先打开此文件。

示例文件位于 Foxtable 安装目录的 CaseStudy 子目录下。
为了方便大家快速打开示例文件，Foxtable 的程序菜单提供了一个“CaseStudy”按钮，单击此按钮即可选择打开 CaseStudy 目录下的示例文件：

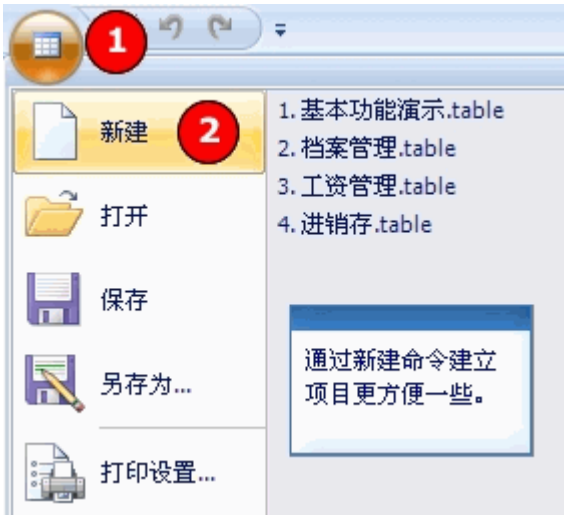


3.4 项目相关

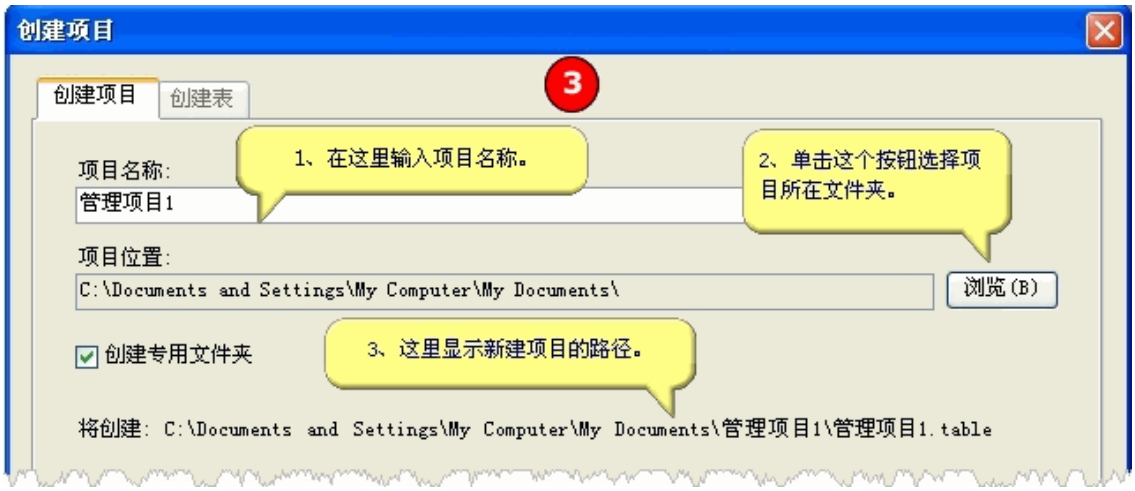
3.4.1 新建项目

启动 Foxtable 之后，会自动建立一个空白的管理项目，您可以在此基础上进行修改。
但更多的时候，我们是通过直接单击程序菜单中的"新建"(如下图所示)命令来新建一个项目，因为这样更为方便。

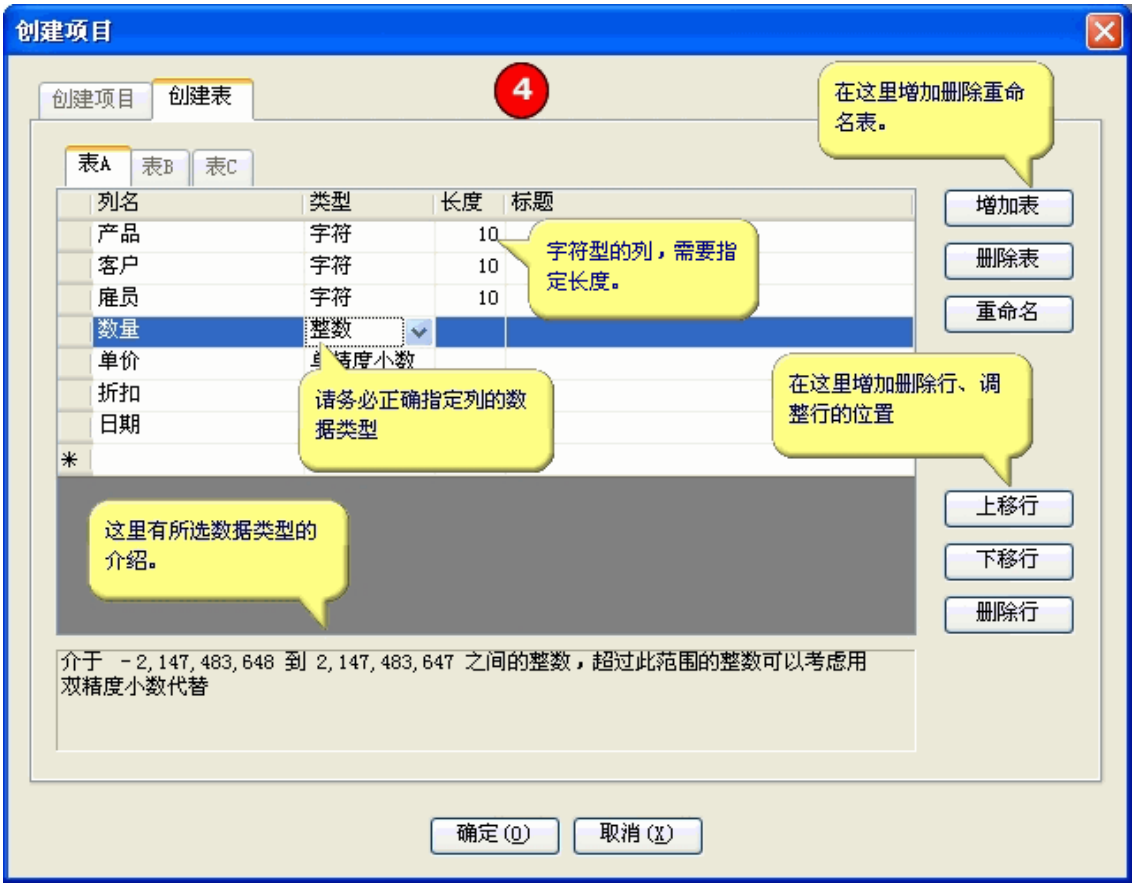
新建项目步骤



Foxtable 中的每一个管理项目，都有一个同名的专用独立文件夹，这会为今后的管理带来很多方便；当您输入项目名称，选择项目位置后，会自动显示新建项目的路径和名称，如下图所示：



指定项目名称和位置后，单击"创建表"页签，可以在这里增加、删除、设计表：



根据需要增加所有的表和列，设置好列的类型和名称，单击“确定”按钮，即可新建一个项目。

小提示：

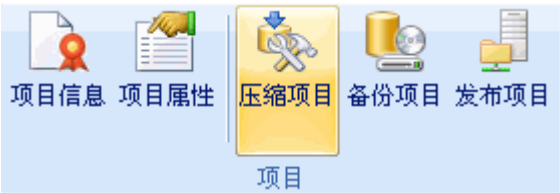
1、如果某一列的值由其它列计算得出，也就是我们常说的表达式列，是不能在这里增加的，后续章节会详细介绍如何增加表达式列。

2、表结构不可能一开始就能完全建好的，在设计过程中，肯定还需要不断地调整，所以 Foxtable 还提供了增加表、删除表，修改表结构的功能，后续章节会对此进行介绍。

3.4.2 压缩项目

Foxtable 虽然界面类似电子表格，但实际上是个纯数据库软件。在 Foxtable 中，被删除的行仍然存在数据库中，占据存储空间，直到执行“压缩项目”的命令，这些行才会真正被删除。

压缩项目命令位于菜单的“管理项目”功能区：



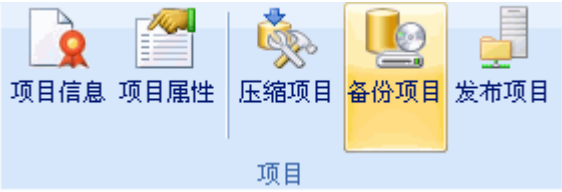
如果在删除行后，继续增加行，那么新增行会使用被删除行的存储空间，而不会占用新的存储空间，所以“压缩项目”命令通常无需频繁地执行。

压缩项目不仅能够节省存储空间，还能提高查询性能，所以定期执行此命令是有好处的。

需要注意的是，压缩项目不会对外部数据表起作用，如果你有数据表来自于外部数据源，例如 SQL Server、Access、Oracle，那么请直接在这些软件中执行压缩命令，Access 对应的命令是“压缩与修复”，SQL Server 对应的命令是“收缩”。

3.4.3 备份项目

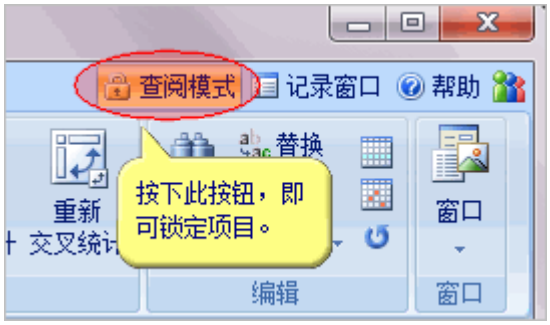
为便于用户快速完成日常的手工备份工作，Foxtable 提供了一个备份功能。
在菜单的“管理项目”功能区中，单击“备份项目”按钮，选择备份文件路径，输入备份文件名，单击“确定”按钮即可完成备份。



备份文件的格式为 zip，备份文件名由当前日期和时间组成，例如在 2012 年 6 月 23 日上午 11 点 25 分备份，那么默认的备份文件名将是“项目名称_201206231125.zip”。
备份文件内容包括项目文件本身，及其所有附件，这样需要恢复项目的时候，只需直接将备份文件解压即可，而且所有的数据、设计和附件都不会丢失。
提示：
1、最好按日期顺序保留多个备份，因为并不是所有意外都能被及时发现的；如果只保留最近的一个备份，那么已经有问题的备份就会覆盖之前好的备份，导致项目无法安全恢复。
2、定期将备份文件复制到多个存储介质中，避免因为存储介质的损坏而丢失数据；如果只在本机备份，一旦硬盘损坏，项目文件和备份文件就可能“全军覆没”，带来灾难性的后果。
3、备份内容不包括外部数据表，如果你有数据表来自于外部数据源，例如 SQL Server、Oracle、Access，那么请另行做好备份工作。

3.4.4 锁定项目

所谓锁定项目，就是使项目处于只读状态，禁止编辑任何表中的内容。
按下配置栏中的“查阅模式”按钮，即可锁定项目；弹起该按钮，即可回到正常状态。



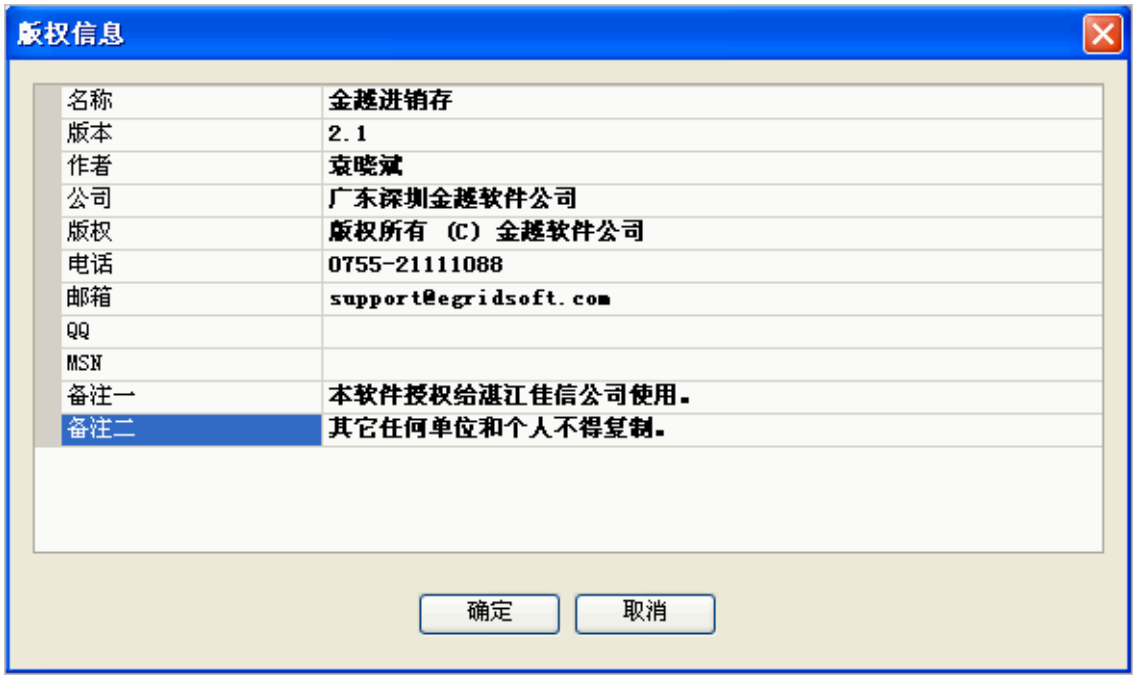
如果你突然无法在表中输入数据，请检查是否无意中按下了“查阅模式”按钮。

3.4.5 项目信息

在“项目管理”功能区中，单击下图中的“项目信息”按钮，可以输入一些和版权相关的信息：



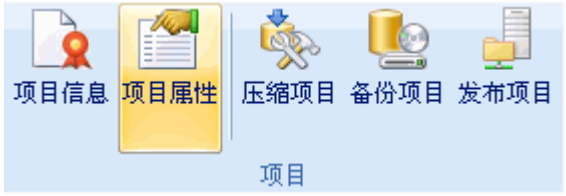
下面是项目信息输入窗口：



项目信息不是必须的，但是我公司在提供某些服务的时候，会查看项目信息，所以输入准确的项目信息，有助于保护您的开发成果。

3.4.6 项目属性

菜单的“管理项目”功能区，单击“项目属性”按钮：



会出现项目属性设置窗口：



大部分的项目属性是逻辑属性，逻辑属性只有两个选项：True(是)和 False(否)。

备份

以下项目属性用于设置自动备份：

- 备份目录
  用于指定默认的备份文件存放目录。

- 备份周期
  此属性用于指定自动备份的周期，默认为“无”，也就是不自动备份。

- 包括附件
  指定备份的时候，是否包括附件，如果设为 False，将只备份项目文件本身。

不能因为设置了自动备份，就忽视了手工备份，因为自动备份是没有办法将备份文件复制到多个存储介质上的。

下面的属性很少使用，普通用户可以忽略。

界面

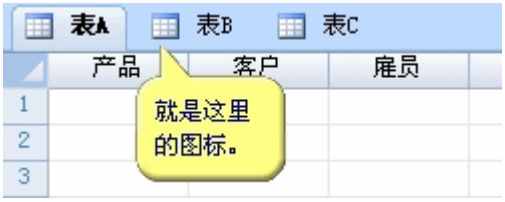
以下项目属性用于设置 Foxtable 的界面：

- 界面风格
  用于指定项目的界面风格，包括默认的蓝色(Office2007Blue)、银灰(Office2007Silver)、黑色(Office2007Black)。

- 显示标题
  是否在菜单的下面，列出所有的表名，用于单击选择不同的表。
  如果将此属性设为 False，将隐藏表的标题，用户无法通过单击表名来选择不同的表，只能通过代码来切换表。

- 显示快选按钮
  是否在表标题右侧显示快速选择表的下拉按钮。

- 显示图标
  是否在表标题处显示图标，设为 True 显示，False 不显示，默认为 False，也就是不显示。



表的图标是可以自定义的，在表属性设置中，有一个名为“自定义图标”的属性，利用该属性即可为表选择个性化的图标，图标的大小最好是 16*16 个像素。

- 登录提示
  设置用户登录窗口的提示信息。
  提示：默认不会出现登录窗口，除非增加用户或者修改了开发者和管理员的密码。
  有关用户管理，会有专门的章节讲述。

- 加载提示
  设置在打开项目的时候，Foxtable 加载表的提示信息，用“{0}”表示正在加载的表的表名。

窗口

以下项目属性用于设置 Foxtable 的主窗口，这些属性不是即时生效的，只有重新打开项目才会生效：

- 最大化
  启动 Foxtable 后，窗口是否最大化。

在最大化属性为 False 的时候，可以通过以下五个属性对窗口进行进一步的设置：

- 宽度
  设置窗口的宽度，单位为像素。

- 高度
  设置窗口的高度，单位为像素。

- 最大化按钮
  是否在窗口标题栏显示最大化按钮。

- 最小化按钮
  是否在窗口标题栏显示最小化按钮。

- 允许调整大小
  是否允许用户在运行过程中调整主窗口大小。

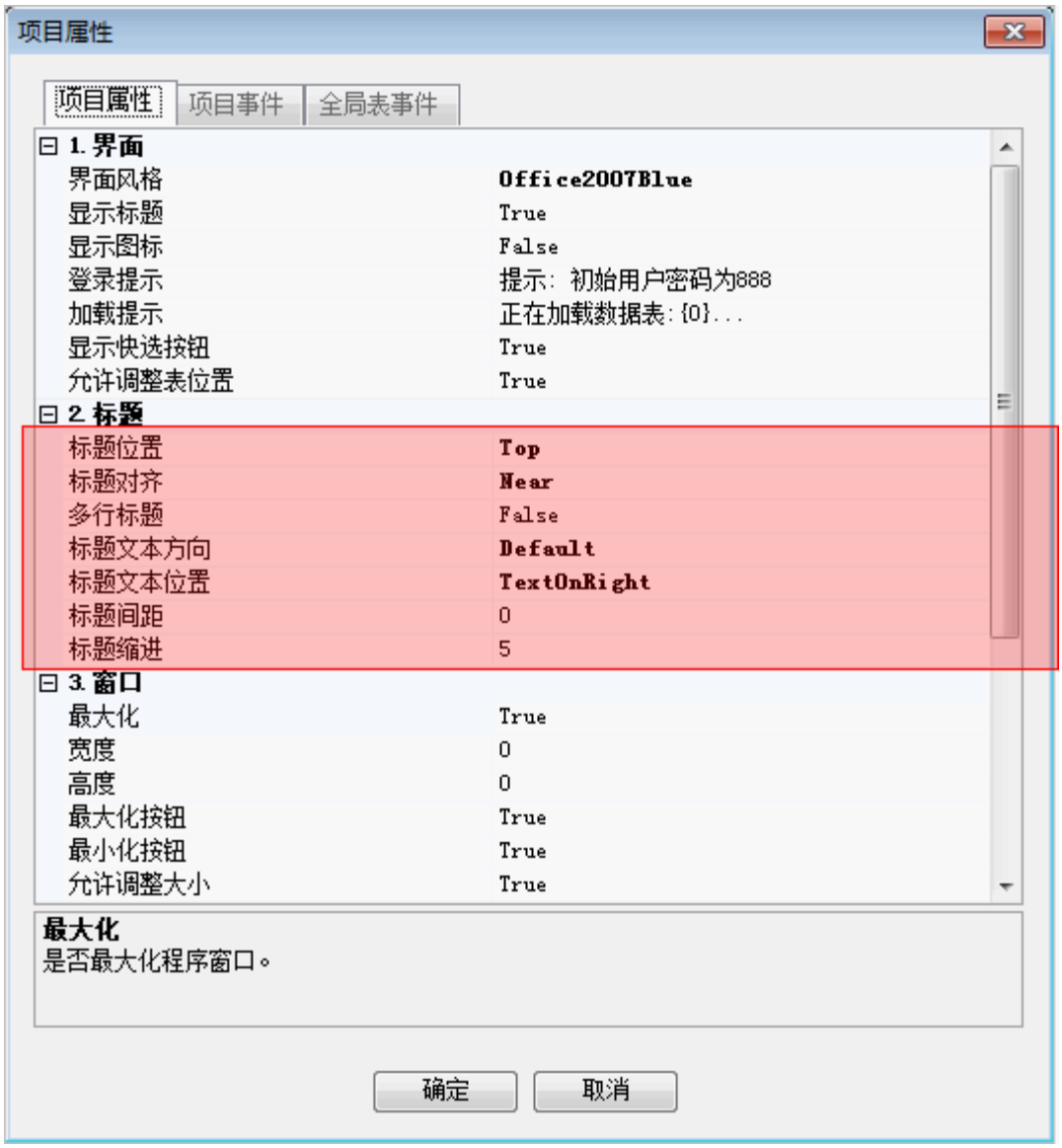
开发设置

- 关闭开发功能
  此属性默认为 False，如果设为 True，则除开发者之外，其他任何人均无法使用 Foxtable 的设计功能。
  如果设计好的项目需要提供给第三方使用，那么利用此属性，可以有效保护您的开发成果 。

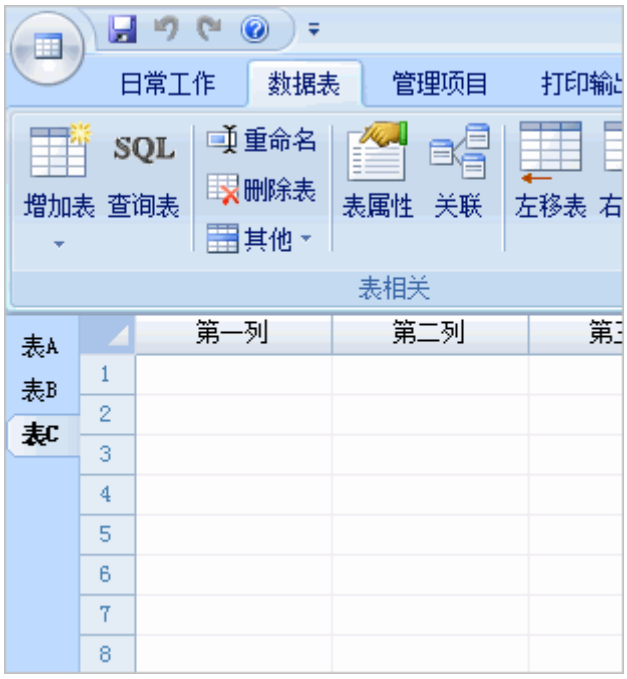
- 允许设置数据源
  如果打开项目的时候，数据源无效，是否允许管理员级别的用户重新设置数据源的链接字符串。

3.4.7 表标题设置

项目属性提供了一组属性，用于对表标题进行细节设置。



例如我们将“标题位置”属性设置为“Left”，标题文本方向设置为“Horizontal”，表标题将显示表的左边位置：

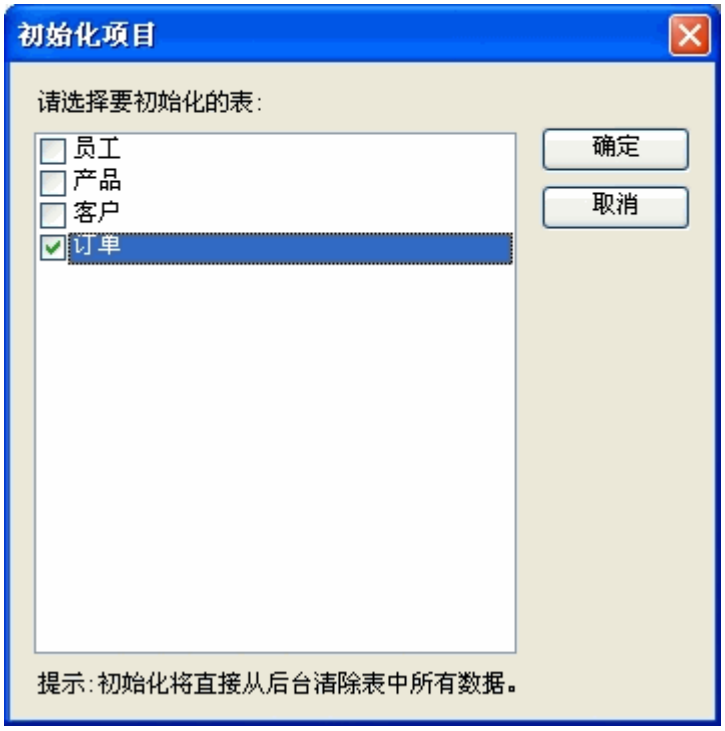


3.4.8 初始化项目

所谓初始化项目，就是清空数据，回到初始状态；“初始化项目”命令位于“管理项目”功能区中：



在初始化窗口，选择要初始化的表，只有选中的表，数据才会被清空：



选择要初始化的表，单击“确定”按钮，即可完成初始化的工作，并重新打开项目。
初始化会直接从后台清除表的所有数据，不管这些数据是否已经加载，所以在初始化一个表之前，一定要慎重考虑。
如果希望禁止某个表被初始化，可以在表属性设置中，将其“允许初始化”属性设为 False(否)。

3.5 表相关

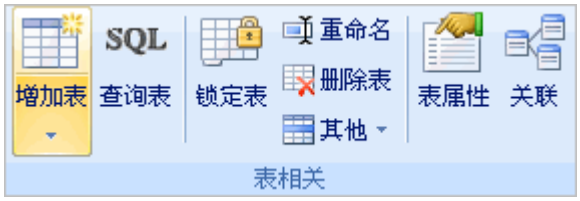
3.5.1 基本操作

3.5.1.1 表的类型

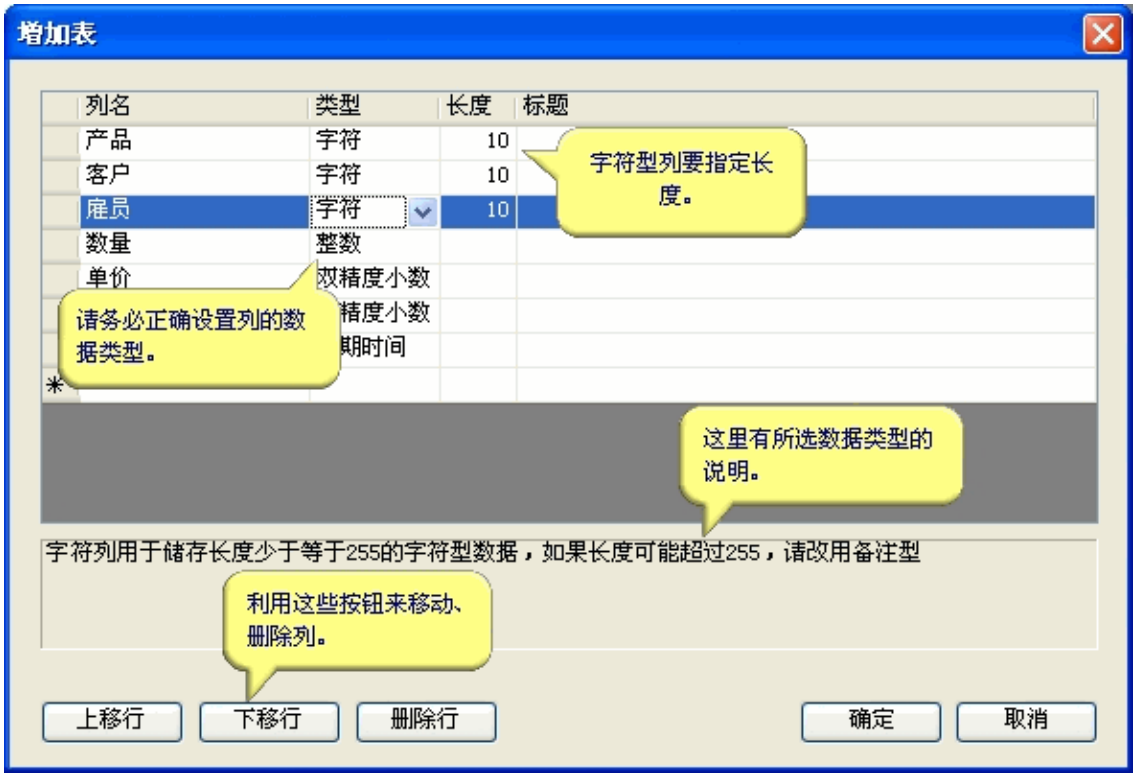
Foxtable 包括以下表类型：
内部数据表
外部数据表
查询表
临时表
实际应用的时候，使用最多的是内部数据表和外部数据表，为方便起见，统称为“数据表”。
现在我们不需要去深究每种表的差别，随着学习的深入，我们自然就会了解的。

3.5.1.2 增加表

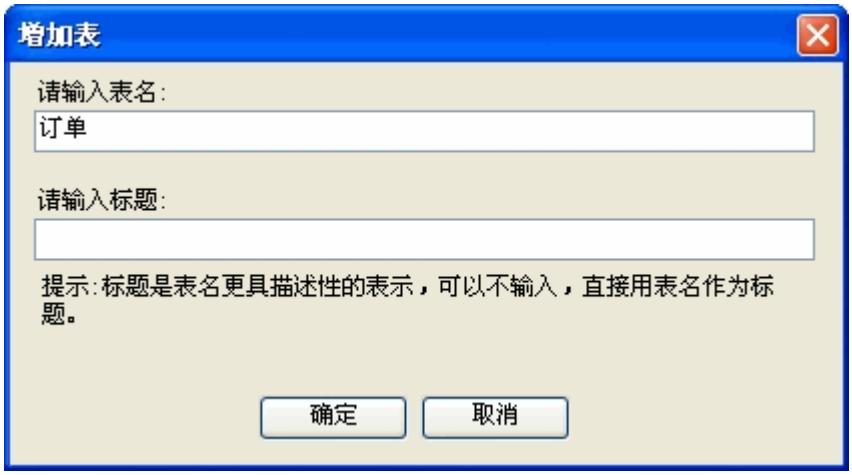
“增加表”的命令位于“数据表”功能区中：



下面是增加表窗口：



设置好各列的名称、类型、长度后，单击确定按钮，会弹出一个窗口，要求输入表的名称和标题，其中名称是必须输入的，而标题则可以不输入：



在增加一个表之前，应该仔细想一想，这个表应该包括哪些列（栏目），每一列的类型是什么，哪些列是由其他列计算得出的，这是一个管理项目的基础；特别是列的数据类型，更是决定计算、统计、筛选能否正确执行的基础。当然你可以在启用项目之后，再修改列的数据类型，但是此时操作 可能会带来一些额外的麻烦，例如某个表已经通过某一列和其他表建立了关联，那么在修改该列的数据类型之前，您必须先删除有关的关联，而删除关联，又会影响相关表达式的计算，并

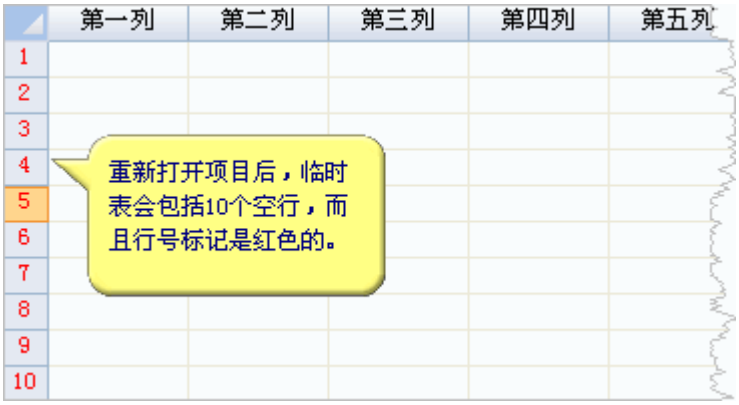
导致关联设置的丢失。
此外，字符列需要指定长度，请合理设置长度，太长则浪费空间，太短则恐怕以后不够用；例如对于公司名称，绝大多数公司的名称会在 15 个字符以内，考虑到特殊单位名称的出现，可以将公司名称列设为 20 个字符，而姓名列，中国人的姓名绝大多数不会超过 4 个字符，所以将姓名列设为 5 个字符足够了，如果涉及到少数民族或者外籍人员，那么姓名列的长度应该设为多少，就得仔细衡量一下了。
关于列数据类型，可以参考：数据类型详解
提示：如果某一列的值由其它列计算得出，也就是我们常说的表达式列，是不能在这里增加的，后续章节会详细介绍如何增加表达式列。

3.5.1.3 关于临时表

临时表的数据只会在运行的时候存在，并不会保存，通常用于在运行过程中显示动态的数据，例如统计结果。
临时表的使用在开发篇才会接触到，目前我们只需了解如何增加临时表。
下图是增加临时表的命令：



我们遇到过一些用户，在增加数据表的时候，误单击了“增加临时表”命令，导致在这个表中输入的数据无法保存，却始终不明白是怎么回事。
下次如果你遇到整个表不能保存的问题，请首先检查一下这个表是不是临时表。
判断一个表是否是临时表的方法很简单，重新打开项目，如果某个表包括 10 个空行，而且左边的行号是红色，那么基本就可以判定这个表是临时表了：



3.5.1.4 调整表结构

虽然我们建议你在设计表的时候，尽可能完成所有的表结构设计工作，以避免后期因为调整表结构而带来麻烦。

但是实际上，百密必有一疏，后期完全不调整表结构是不可能的。
通过“数据表”功能区的下述按钮，可以很方便地调整表结构：



上图中的“更改列”按钮，用于重命名列或更改列的数据类型。
表结构的调整是即刻生效的，且不可撤销，所以在调整表结构，特别是删除列之前，务必要慎重考虑。

3.5.1.5 删除表

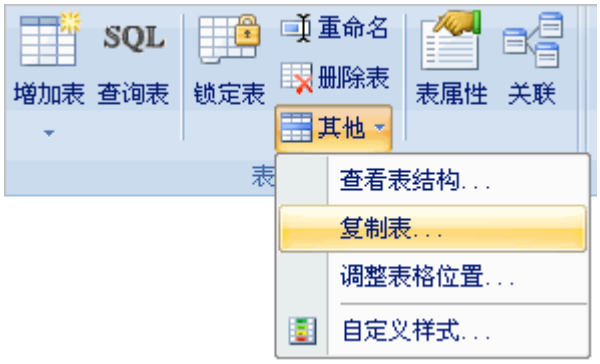
在菜单的“数据表”功能区，单击下面的按钮，可以删除表：



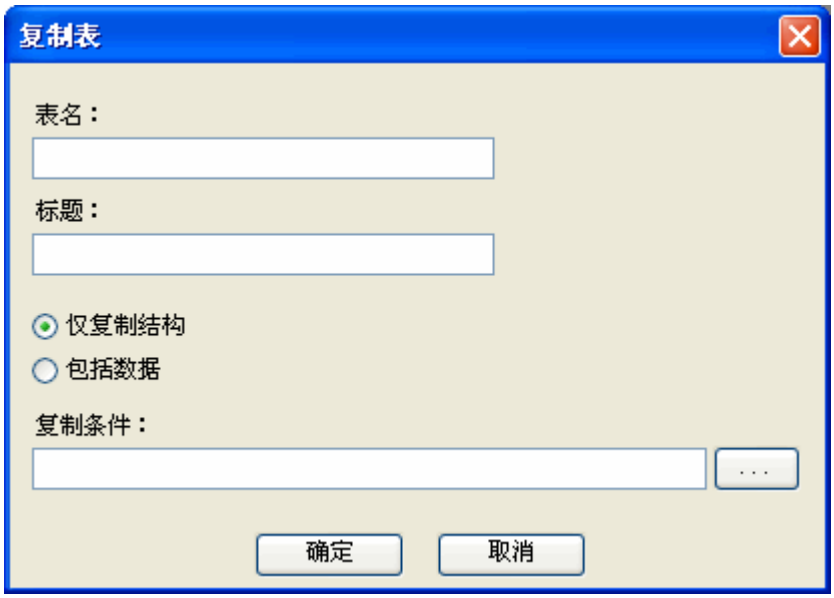
删除表是即刻生效、且不可逆转的，附属于此表的窗口也将被删除，所以删除表之前务必要慎重考虑。
如果被删除的是内部数据表，且包括有大量的数据，那么删除后最好执行一次压缩项目的命令。

3.5.1.6 复制表

在菜单的“数据表”功能区，单击下面的按钮，可以复制表：



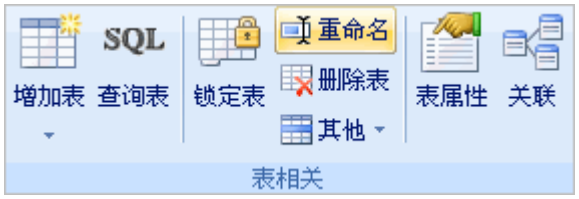
这是复制表窗口：



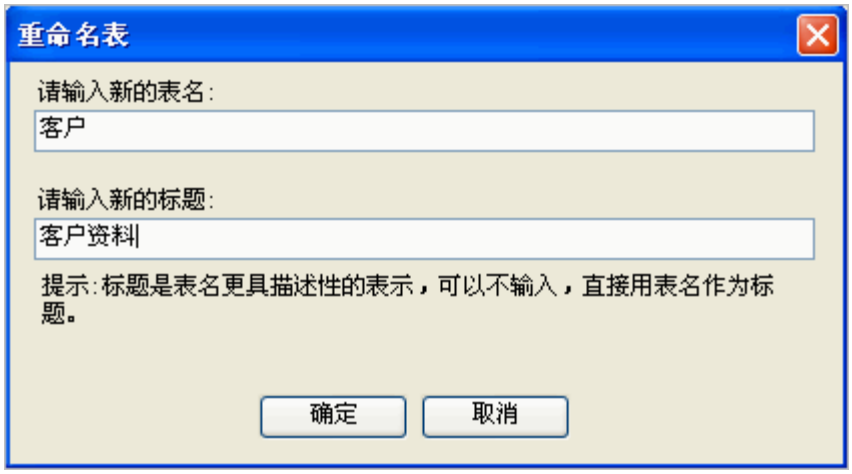
你可以仅复制结构，也可以连同数据一起复制。
可以设置数据复制条件，复制条件是一个逻辑表达式，只有该表达式成立的行才会复制，有关表达式的知识将在后续章节讲述。
如果不设置复制条件，将复制所有数据。
需要注意的是，复制表只复制数据列，不复制表达式列。

3.5.1.7 重命名表

单击下面的按钮，即可指定新的表名和标题：

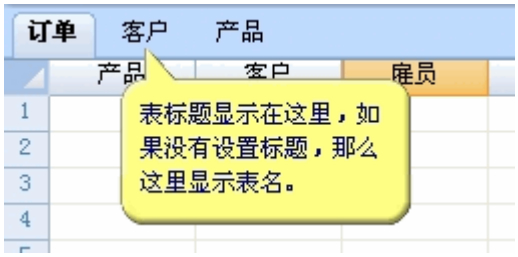


如果有公式或代码使用了这个表的表名，那么重命名表可能会导致这些公式和代码无法运行。
所以关于表名，我们应该把握两个原则：
1、事先就要慎重考虑，为每个表设置一个合适的表名，项目一旦启用，不要轻易去修改表名。
2、如果一定要重命名某个表，而且有大量的公式和代码牵涉到这个表，那么应该考虑用修改表标题来代替，例如你需要将“客户”表重命名为“客户资料”，那么可以将该表的标题改为“客户资料”，而表名保持不变，这样就不会影响项目的正常运行了。



3.5.1.8 表名与表标题

我们已经知道，在新建表和重命名表的时候，都可以同时设置表名和表标题。
表标题是表名更具描述性的表示，我们通常无需设置表标题，这时直接用表名作为标题：



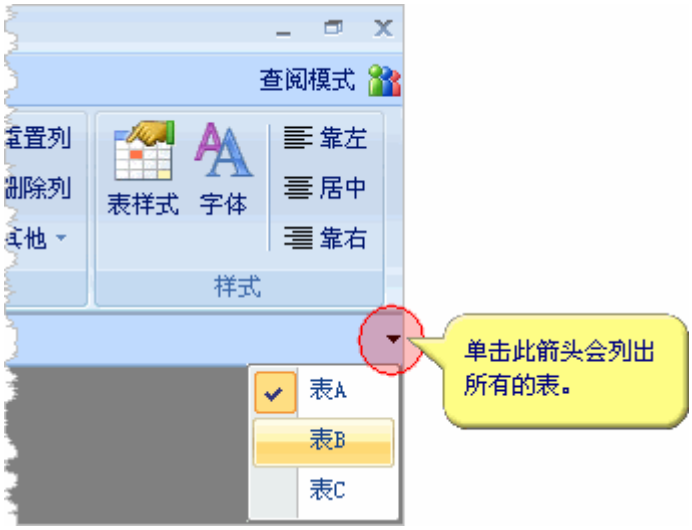
如果你希望有一个简洁的表名，以便于编写公式和代码，同时希望呈现给用户的是一个更具描述性、容易理解的名称，那么表标题就派上用场了。

例如你可以将某一个表的名称设为“emp”，而标题设为“员工档案”，这样在公式和代码中可以使用表名“emp”，而用户看到的是比较容易理解的“员工档案”这个标题。 之后，如果觉得用“员工资料”代替“员工档案”较为合适，那么只需将表标题改为“员工档案”即可，因为表名没有发生变化，还是“emp”，所以不会对系统的正常运行有任何的影响。

如果表名比较简洁，而且容易理解，之后也没有更改的必要，那么就没有必要设置表标题，直接用表名作为标题即可。
表名不能以数字开头，更不能全是数字，也不能包括空格和各种符号(下划线(_)除外)，但是表标题没有任何限制。

3.5.1.9 快速选择表

前面已经看到，表标题以页签的形式出现在表格的上方，用于选择不同的表；如果你有很多个表，例如几十个，那么这种方式无法一次显示所有的表标题供选择；此时你可以通过下图所示的方法快速选择表：



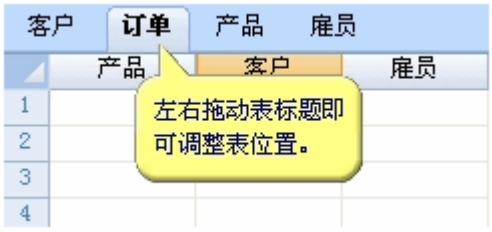
如果不希望出现此按钮，可以在项目属性设置中，将“显示快选按钮”属性设置为 False。

3.5.1.10 调整表位置

用菜单的“数据表功能”区，单击下图所示的按钮，即可调整表的位置。



或者用鼠标左键按住表标题，左右拖动也可调整表位置：



如果在项目属性设置中，将“允许调整表位置”属性设置为 False，将禁止用户通过鼠标拖动调整表位置。

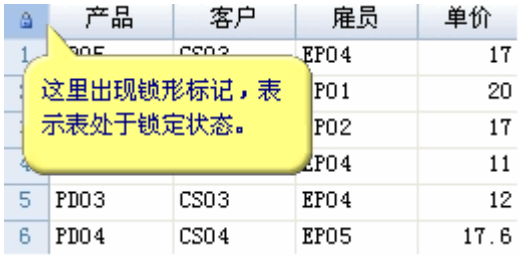
3.5.1.11 锁定表

所谓锁定表，就是将表设为只读，禁止编辑表中的内容。
有两种方法锁定表：
1、如果要永久锁定表，那么在表属性窗口，将允许编辑属性设为 False(否)。
2、如果要临时锁定表，可在菜单的“日常工作”功能区，单击下图所示的“锁定表”按钮：



上图的“锁定表”按钮就像一个开关，反复单击此按钮，可以使得表在锁定和非锁定两个状态之间切换。

被锁定的表，其左上角会有一个锁形标志，表示表处于锁定状态：



同样有两种方法来取消锁定表：

1、如果要永久取消锁定表，那么在表属性窗口，将允许编辑属性设为 True(是)

2、如果要临时锁定表，那么就单击前面提到的“锁定表”按钮，这个按钮同时具备锁定表和取消锁定表的功能。

无法取消锁定表？

如果锁定项目了， 那么即使取消锁定表，表还是会处于锁定状态，无法回到正常状态，直到取消锁定项目。

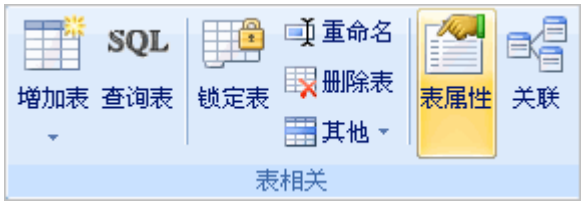
所以如果你发现没有办法取消锁定表，那么就应该检查一下是不是项目处于锁定状态了 。

如果窗口右上角的查阅模式按钮处于压下状态，那么就说明项目处于锁定状态：

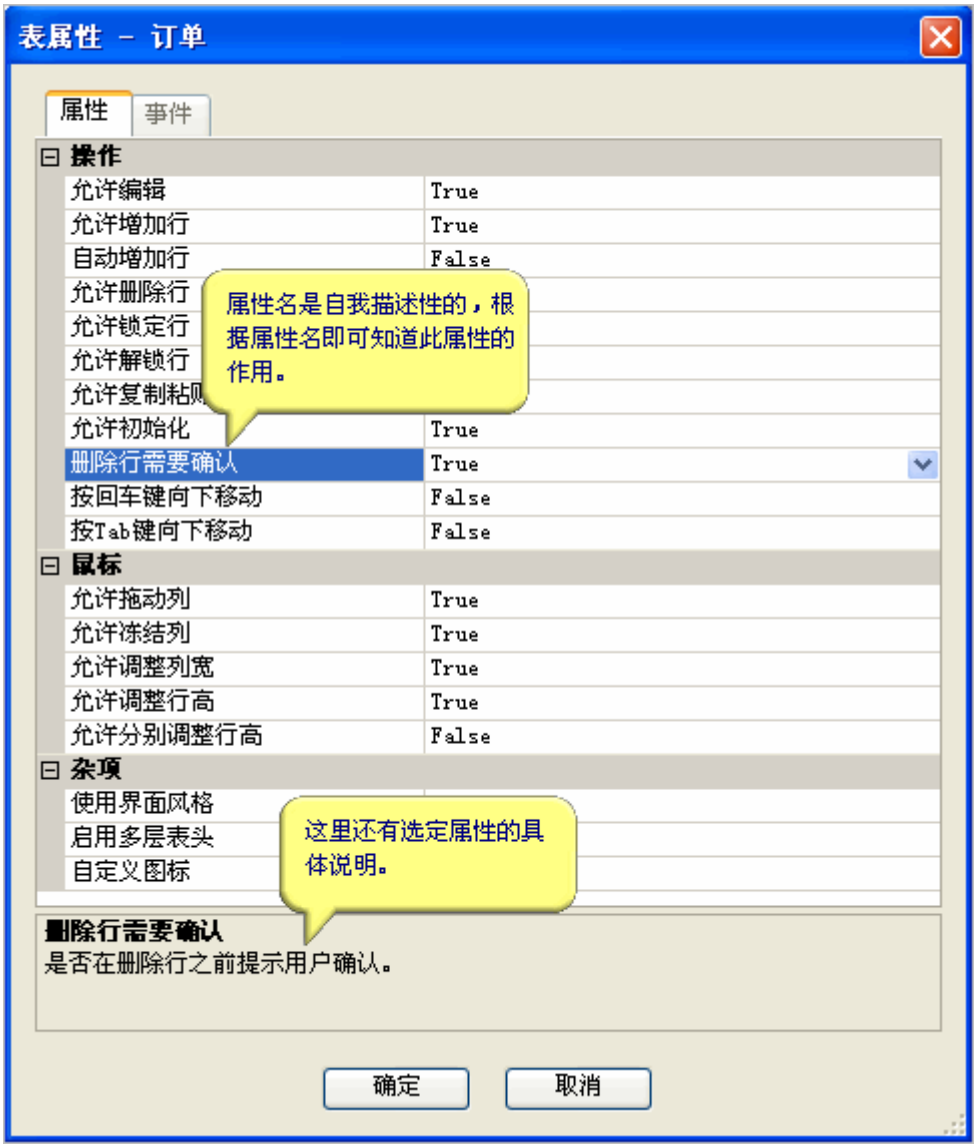


3.5.1.12 表属性

在菜单的“数据表”功能区，单击下面的按钮，即可设置表的属性：



这是表属性设置窗口：



每一个表属性都是很好理解的，而且选择一个属性，窗口下方还会出现该属性的具体说明。

例如你要禁止编辑某个表，也就是锁定该表，只需将其“允许编辑”属性设为 False(否)。

由于属性名称都是自我描述性的，而且非常容易理解，所以我们就不一一介绍这些属性了。

3.5.1.13 多层表头的实现

Foxtable 的表头可以分多层显示，利用这个特性，用户可设计出结构任意复杂的表，如下图：

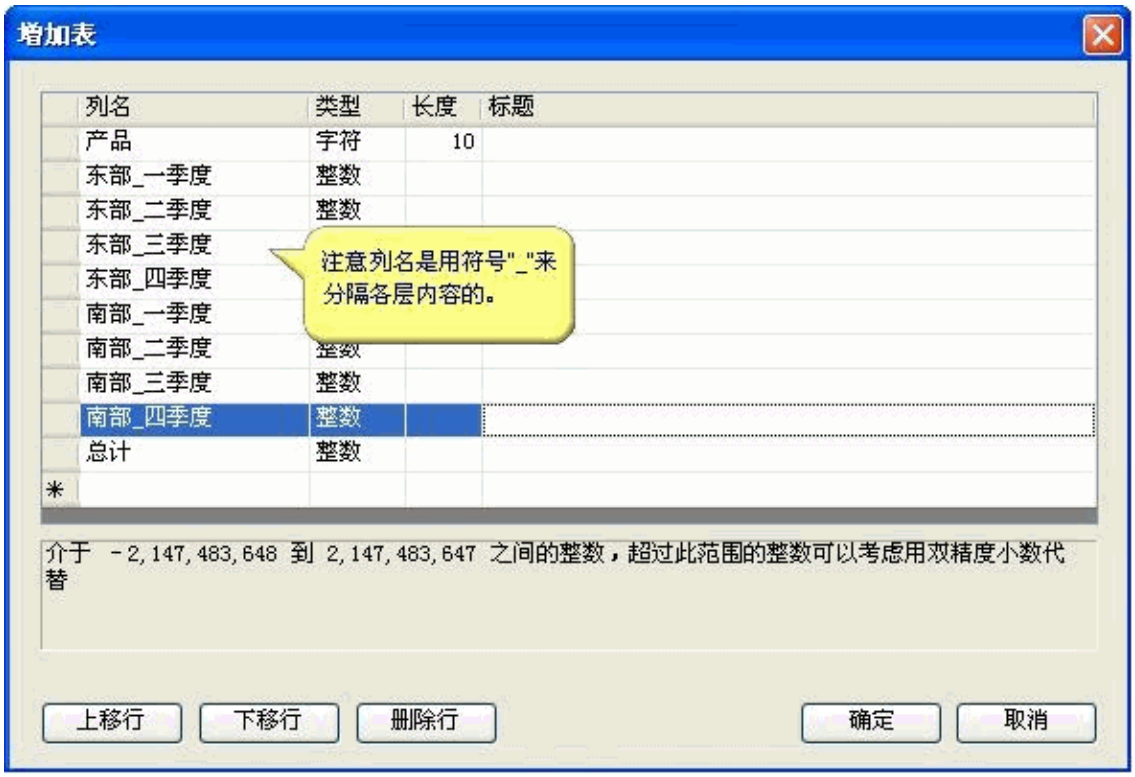
	产品	东部				南部				总计
		一季度	二季度	三季度	四季度	一季度	二季度	三季度	四季度	
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

有两种方法可实现多层表头 ：

一、用列名实现

利用列名可以非常轻松地实现多层表头，每一层的内容用下划线“_”隔开即可，例如“东部_第一季度”。

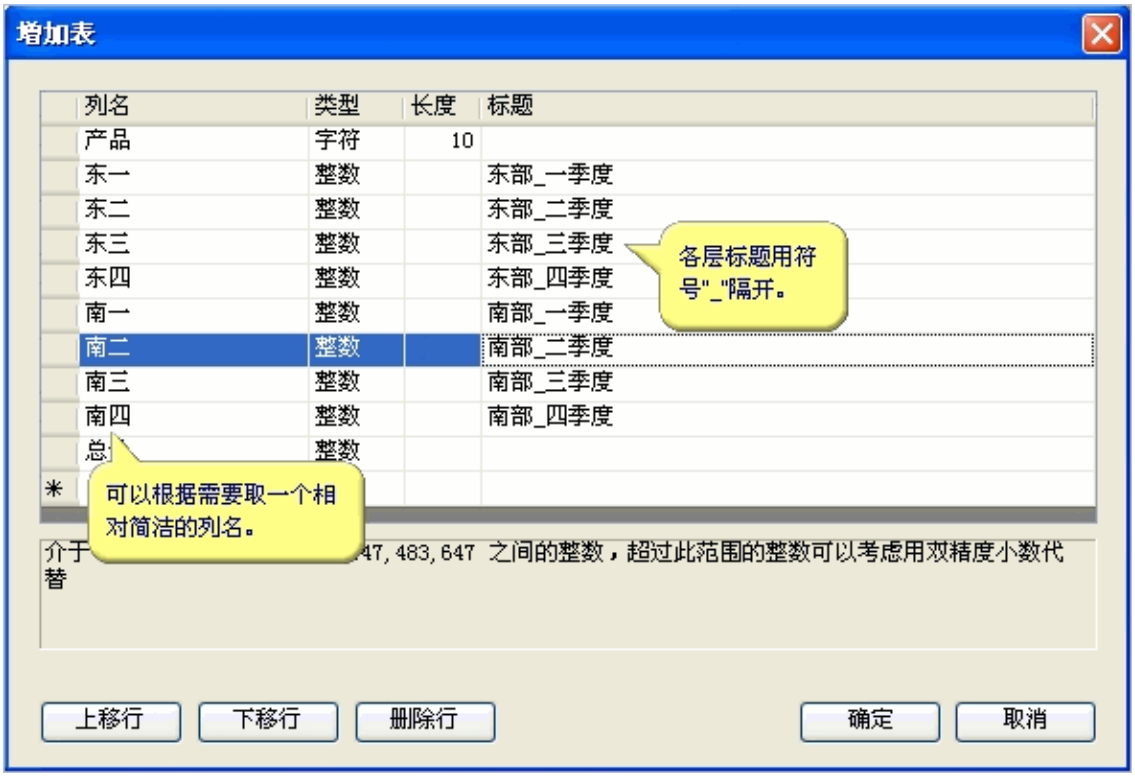
例如要实现上图所示的多层表头，按下图所示设计即可：



二、用列标题实现

另一种方法，也是更为常用的方法，是利用列标题来实现，每一层的列标题同样用下划线“_”隔开。

上面的多层表头如果改用列标题来实现，可以按下图所示设计：



三、现有表实现多层表头

对于现有的表，如果要实现多层表头，可以利用以下按钮来更改列名和标题，以及调整列的位置：

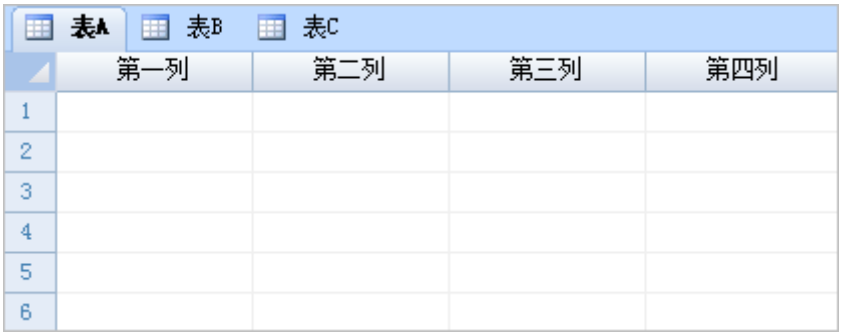


四、关闭多层表头

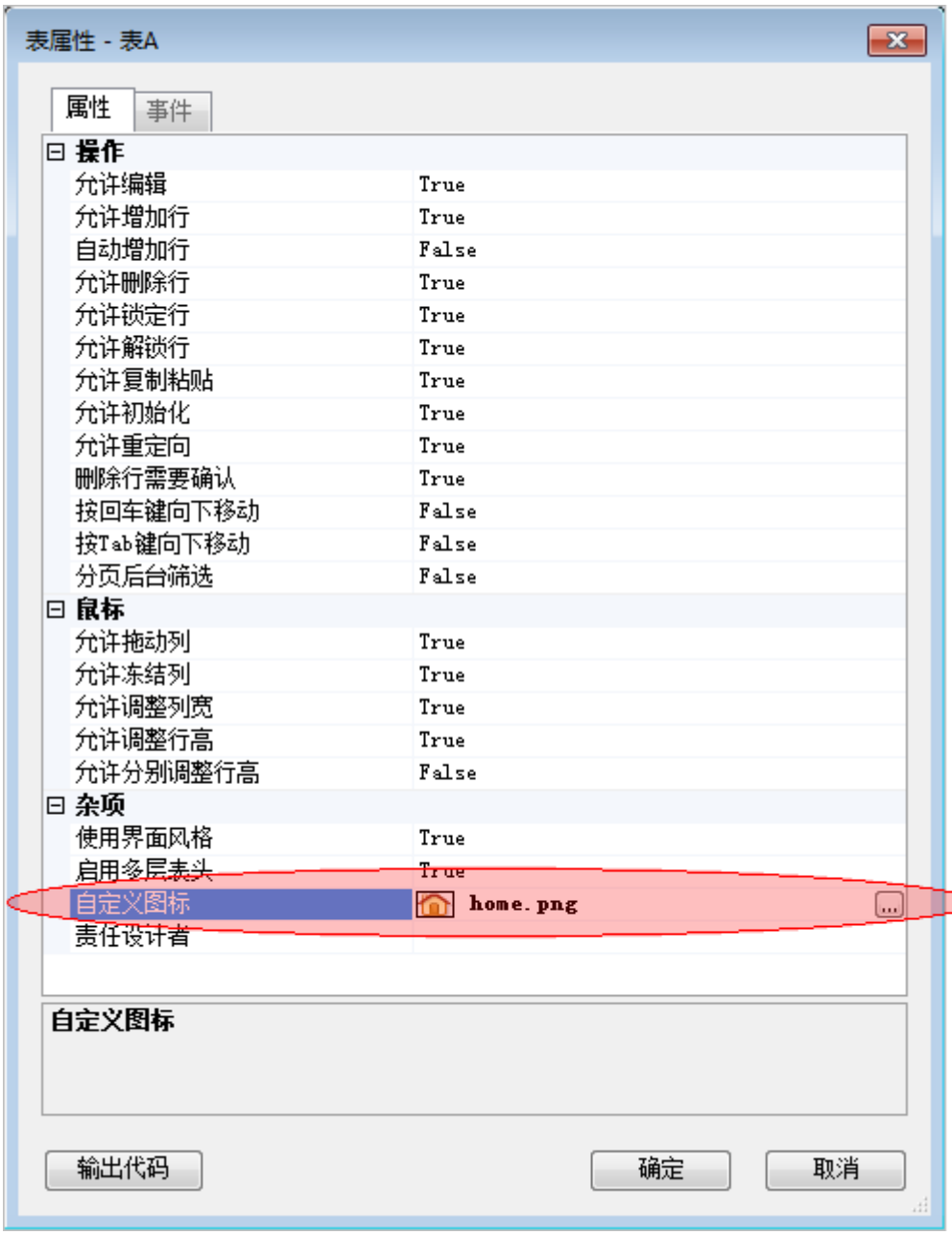
如果不希望显示多层表头，只希望将符号“_”作为普通的字符处理，可以在表属性设置中，将“启用多层表头”属性设为 False(否)。

3.5.1.14 自定义表图标

我们知道，如果将项目属性中的“显示图标”属性设为 True(是)时，会在表的标题处显示图标 ：



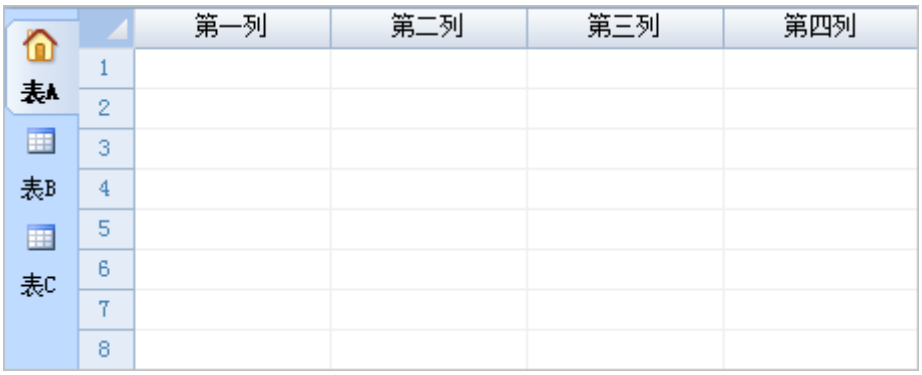
表的图标默认都是一样的，我们可以在表属性中，给表指定一个自定义的图标：



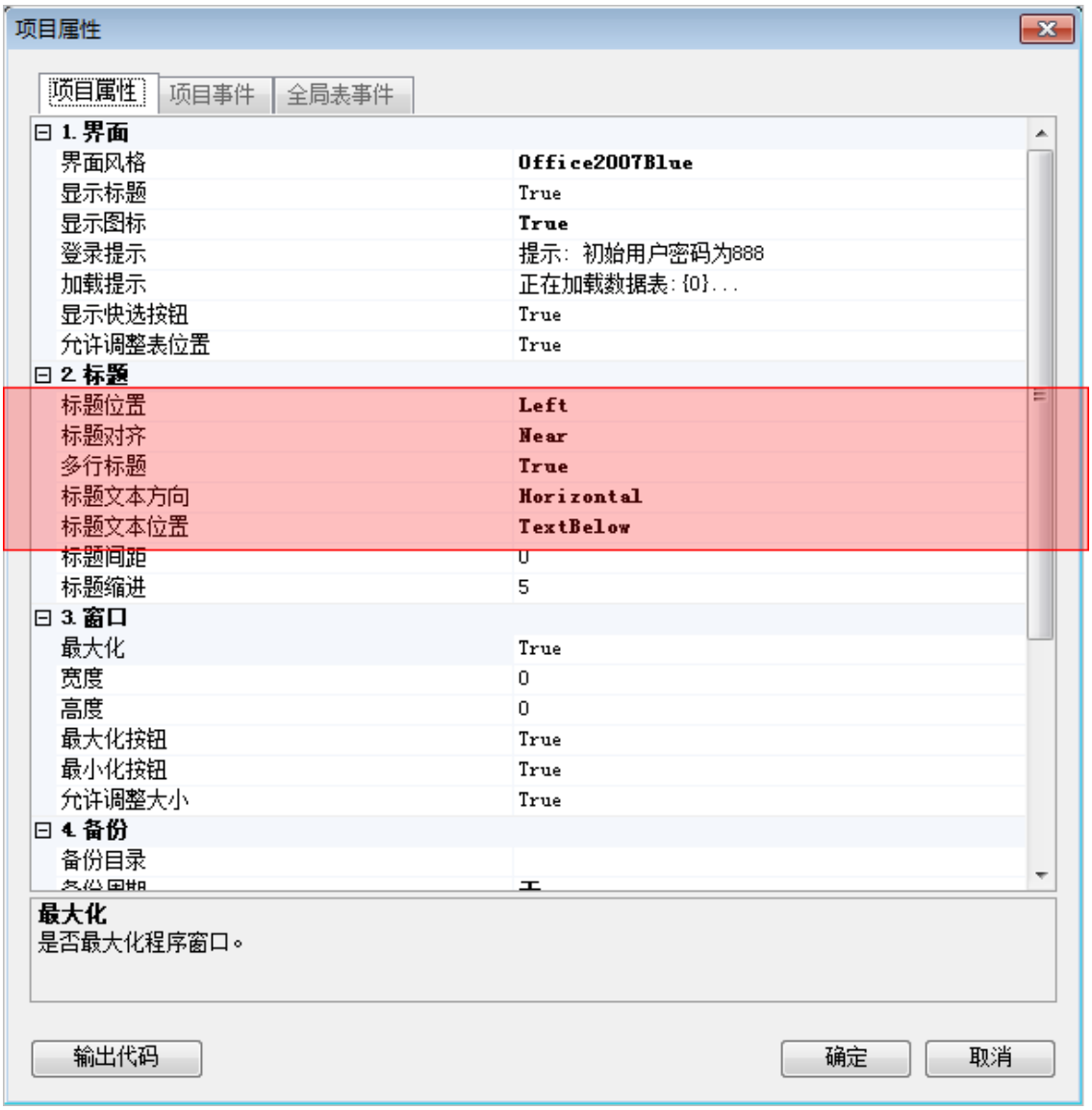
显示效果：



项目属性中有一组用于设置表标题的属性， 例如希望将表的标题显示在表的左边：



可以按下图所示设置项目属性：

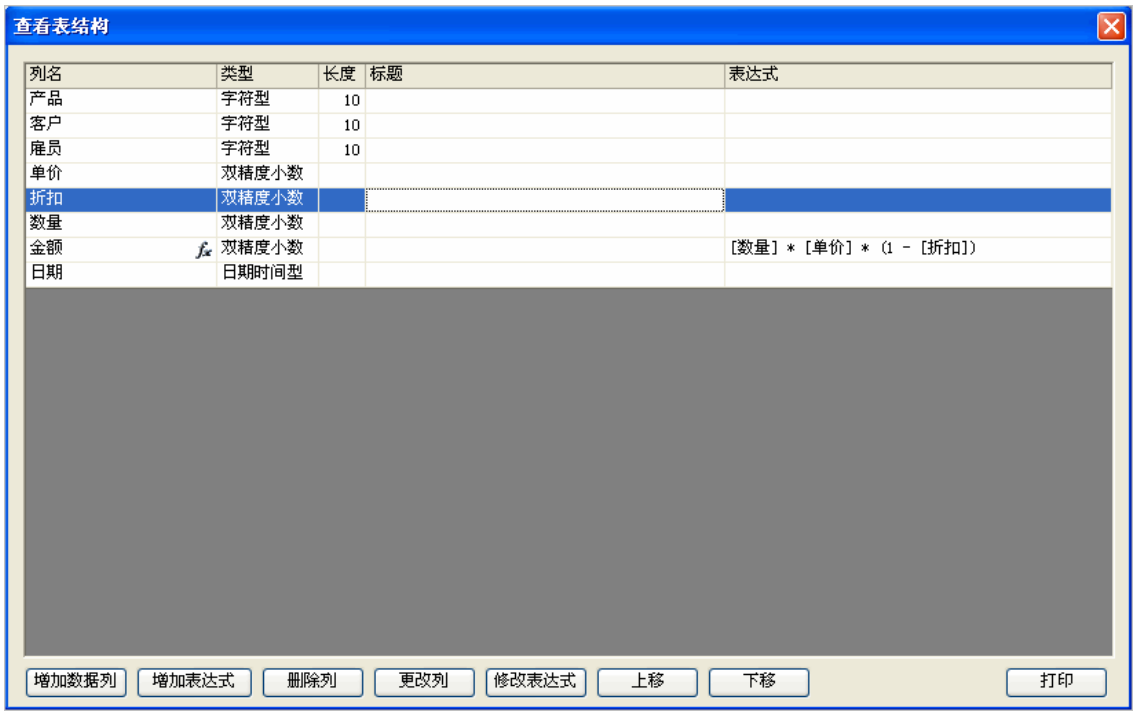


3.5.1.15 查看表结构

如果需要查看表的完整结构，可以在菜单的“数据表”功能区，单击下面的按钮：



下面是结构查看窗口，你不仅可以在这里查看表结构，还可以在这里集中修改表结构：



3.5.1.16 输出表结构

在“杂项”功能区，单击下面的按钮，可以生成并打开一个 RTF 格式的文件，该文件会列出所有表的结构，包括关联的构成：



3.5.2 表样式

3.5.2.1 行号的特殊意义

在默认的情况下，表的左边会显示行号，行号的颜色是有特殊意义的，不同的颜色表示不同的行状态。

例如在下图中，灰色的行号(2、3 行)表示行被锁定，无法编辑；橙色的行号（4、5 行），表示行的内容被修改了；红色的行号（8、9 行），表示这是新增加的行：

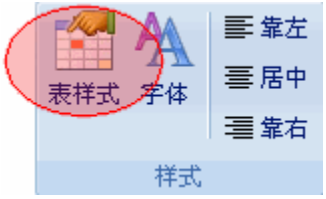
	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03
4				11	0.15	80	748	1999-01-04
5				12	0	490	5880	1999-01-04
6				17.6	0.12	240	3717.12	1999-01-07
7	PD05	CS03	EP04	17	0	160	2720	1999-01-07
8								
9								

不同状态的行，行号的颜色是不同的。

3.5.2.2 设置表样式

表样式决定了表的外观，默认的表样式符合大多数人的习惯，通常我们无须去更改表样式。

在菜单的数据表功能区，单击下图所示的按钮，可以设置表样式：



下面是表样式设置窗口，包括“样式设置”和“配色方案”两个页面，可以控制表格外观的方方面面：



样式设置窗口中每一个属性的名称，都是自我描述性的，而且很容易理解，例如你要更改新增行的行号颜色，只需在“行号颜色”这组属性中，选择“新增行”这一项，然后修改其颜色即可。

如果对某一个属性的具体意义不是很清楚，可以更改该属性的值，然后单击“应用”按钮，即可看出更改该属性所带来的变化。

3.5.2.3 锁定行标记

表样式中有一个“显示锁定行标记”的属性，在默认的情况下，该属性设置不管是 True(是)，还是 False(否)，锁定行标记都是不会显示的，这是因为锁定行是通过行号的颜色来标记的；要显示锁定行标记，必须在表样式设置窗口中，如下设置属性值：

显示行号：False(否)

显示锁定行标记：True(是)

也就是说，只有关闭行号显示，锁定行标记才会显示的。

锁定行标记符是一个红色的勾，例如下图的 2、3 行就是处于锁定状态：

	产品	客户	雇员	单价	折扣	数量	金额	日期
	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02
✓	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
✓	PD05	CS04	EP02	17	0	320	5440	1999-01-03
	PD01	CS03	EP04	11	0.15	80	748	1999-01-04
	PD03	CS03	EP04	12	0	490	5880	1999-01-04
	PD04	CS04	EP05	17.6	0.12	240	3717.12	1999-01-07
	PD05	CS03	EP04	17	0	160	2720	1999-01-07

我个人觉得用行号更为方便。

3.5.2.4 其它标记

除了锁定行外，被锁定的表、列以及排序列，都会用相应的符号标记出来的，如下图所示：



在表样式设置窗口，可以关闭这些标记，可是我们实在想不出有什么理由要去关闭标记显示功能。

3.5.2.5 配色方案

表样式还提供了配色方案功能，利用配色方案，你可以控制表中所有区域的背景颜色和字体颜色，设置颜色后，单击“应用”按钮即可看到设置效果，请随心所欲地去设置，乱了也没关系，单击“还原为默认值”按钮，即可回到初始状态。

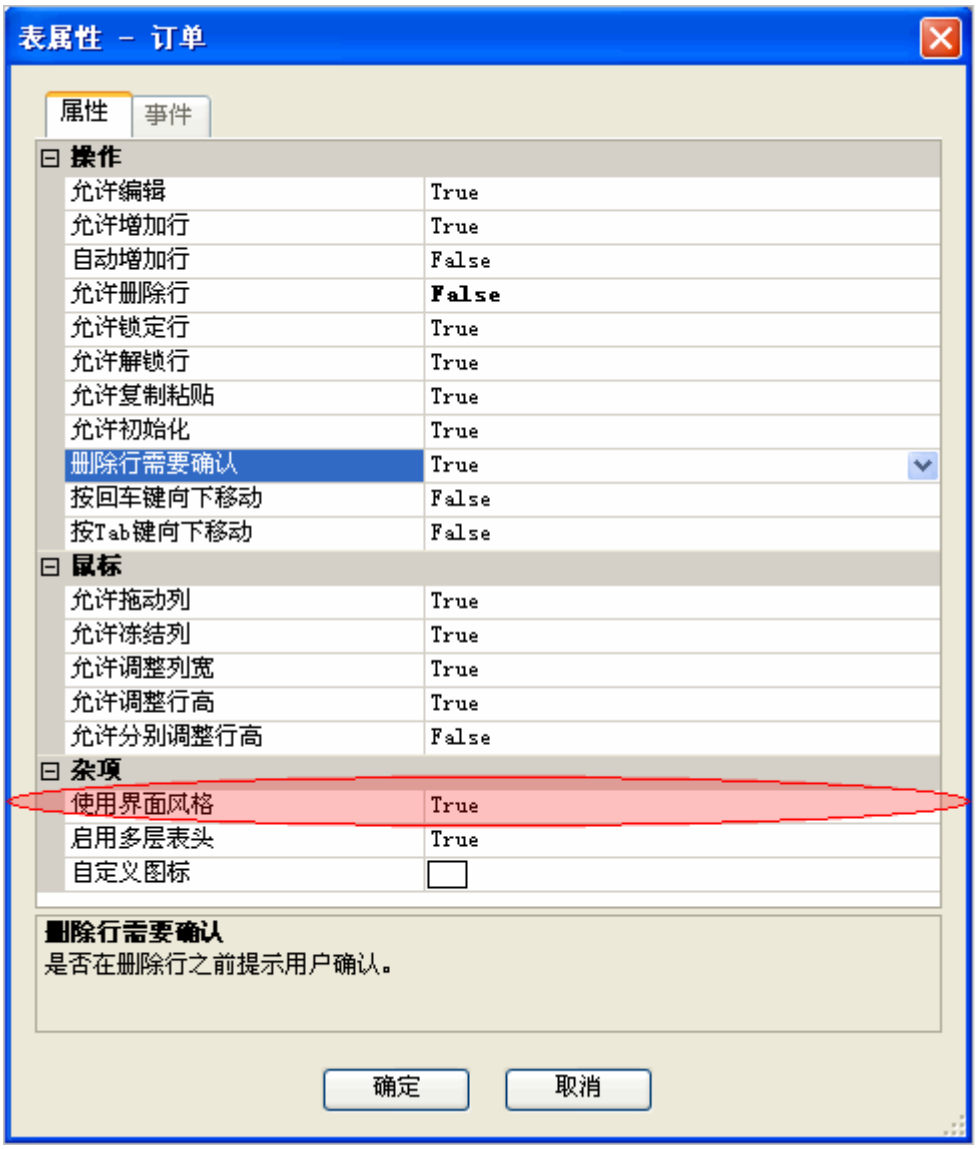


需要注意的是：

1、当前行的字体颜色和背景颜色设置，只有在整行突出显示时才有效。
2、标题区和选定区的背景颜色设置，只有在表属性中的“使用界面风格”设置为 False 的时候，才会生效。

3.5.2.6 配色方案与界面风格

我们已经知道，在默认情况下，标题区和选定区的背景颜色是由界面风格决定的，所以在配色方案中对此作出的修改并不会生效。
如果需要改变上述区域的背景颜色，可以在表属性设置中，将“使用界面风格”属性设置为 False:

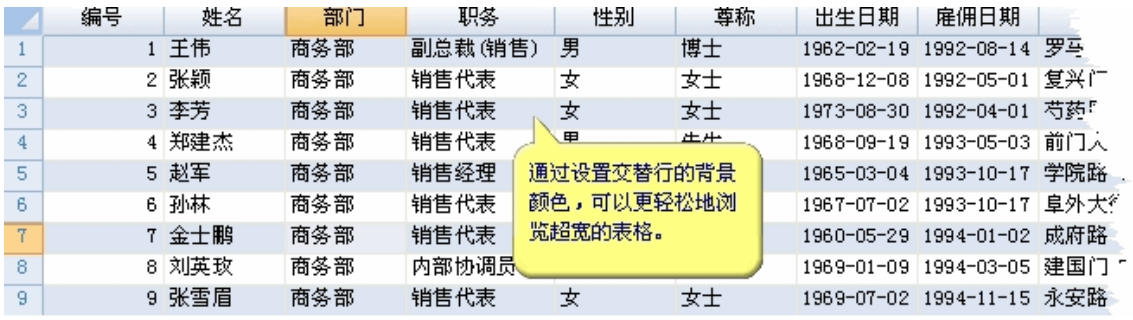


这样就可以不受界面风格的影响，自由设置任何区域的颜色：



3.5.2.7 交替行背景颜色

需要更改系统默认配色方案的时候并不多，唯独交替行的背景颜色较为常用。
如果一个表很宽，浏览数据的时候，容易造成上下行交错，设置交替行的背景颜色，浏览起来会更轻松一些。



一些用户可能觉得，设置交替行背景颜色使表格显得凌乱了，如果这样，可以考虑使用整行突出功能。

3.5.2.8 网格线不见了？

默认情况下，网格线是浅灰色，如果打印机的颜色精度不是很高，例如针式打印机，那么网格线可能无法打印出来；为解决这个问题，可以将网格线设置为较深的颜色，例如深灰色、甚至黑色。
同样，在一些质量较差或设置不当的显示器上，网格线可能无法正常显示出来，此时也可以考虑将网格线的颜色设置得深一点。
网格线的颜色通过表样式设置。

3.5.2.9 不能设置单元格样式？

没错，通过菜单是不能设置单个单元格的样式的，这是否意味着 Foxtable 的样式设置功能很弱呢？
错，Foxtable 具有超强的单元格样式控制功能，可以根据各种逻辑关系自动设置单元格的样式，标记出不同类型或各种特殊的数据，只是这个功能放在开发篇中讲述，现在不是时候。
例如下图，不同的成绩，用不同的颜色标出来了，而且是自动标记的，无须手工设置：

	期数	班级	学号	姓名	语文	数学	英语	物理	化学	总分
1	2	1	00101	陈成进	↑16 (42/58)	←0 (37/37)	↑9 (34/43)	↓25 (44/19)	↑18 (38/56)	↑12 (47/59)
2	2	1	00102	谢杰立	↑47 (9/56)	↑22 (38/60)	↓42 (45/3)	↑19 (29/48)	↑13 (22/35)	↑32 (24/56)
3	2	1	00103	陈思达	↓1 (53/52)	↓57 (65/8)	↓2 (30/28)	↑14 (2/16)	↑10 (24/34)	↓32 (50/18)
4	2	1	00104	张李华	↑8 (7/15)	↑39 (24/63)	↑17 (17/34)	↑9 (22/31)	↑10 (17/27)	↑28 (4/32)
5	2	1	00105	谢晓霞	↓17 (20/3)	↓19 (64/45)	↑3 (61/64)	↓2 (54/52)	↓2 (8/6)	↓8 (56/48)
6	2	1	00106	杨景亮	↓58 (62/4)	↓1 (51/50)	↓20 (65/45)	↓42 (47/5)	↓8 (23/15)	↓51 (65/14)
7	2	1	00107	苏　捷	↑12 (34/46)	↑1 (22/23)	↓41 (57/16)	↑5 (2/7)	←0 (52/52)	↓14 (38/24)
8	2	1	00108	程康文	↓36 (60/24)	↓48 (55/7)	↑54 (1/55)	↑1 (12/13)	↑55 (5/60)	↑1 (29/30)
9	2	1	00109	陈诗淇	↑46 (15/61)	↑37 (10/47)	↓13 (55/42)	↓16 (52/36)	↑5 (51/56)	↑26 (37/63)

标记单元格的功能，将在开发篇详细讲述。

3.6 列相关

3.6.1 基本操作

3.6.1.1 列的类型

有两种类型的列，分别是：

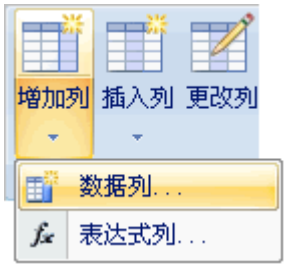
- **数据列**
  数据列用于直接输入数据，数据列的内容将保存在数据库文件中。

- **表达式列**
  顾名思义，表达式列的内容是由一个表达式计算得出的。
  表达式列的内容并不会保存在数据文件中，而是在运行的时候动态生成。
  例如订单表中的金额列，它的内容是通过数量、单价、折扣这三列计算得出的，表达式为：

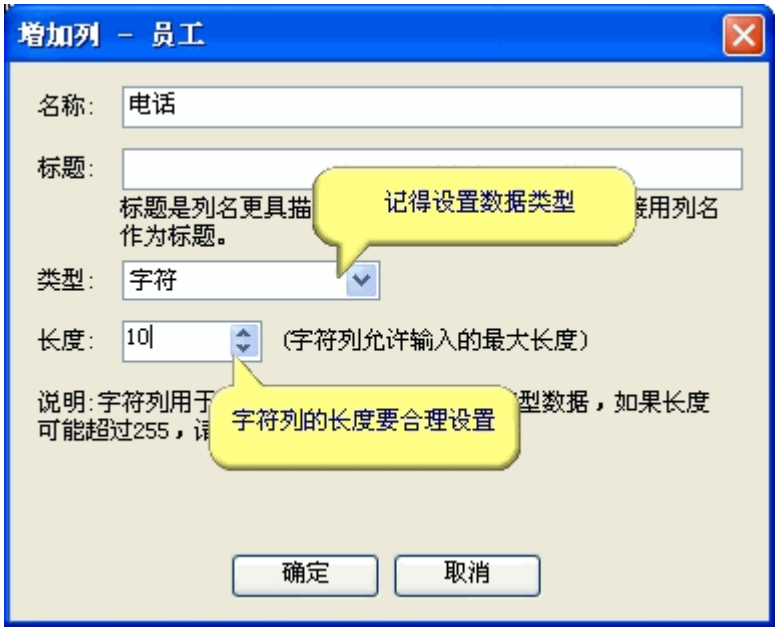
  [数量] * [单价] * (1 - [折扣])

3.6.1.2 增加数据列

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以增加数据列：



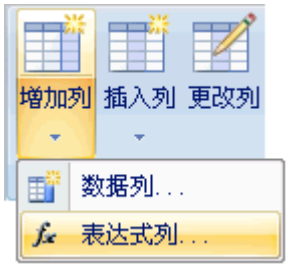
增加列按钮是一个组合按钮，单击按钮的上半部分可直接增加一个数据列，单击按钮的下半部分，会出现一个菜单，用以选择是增加数据列还是表达式列。
增加数据列窗口：



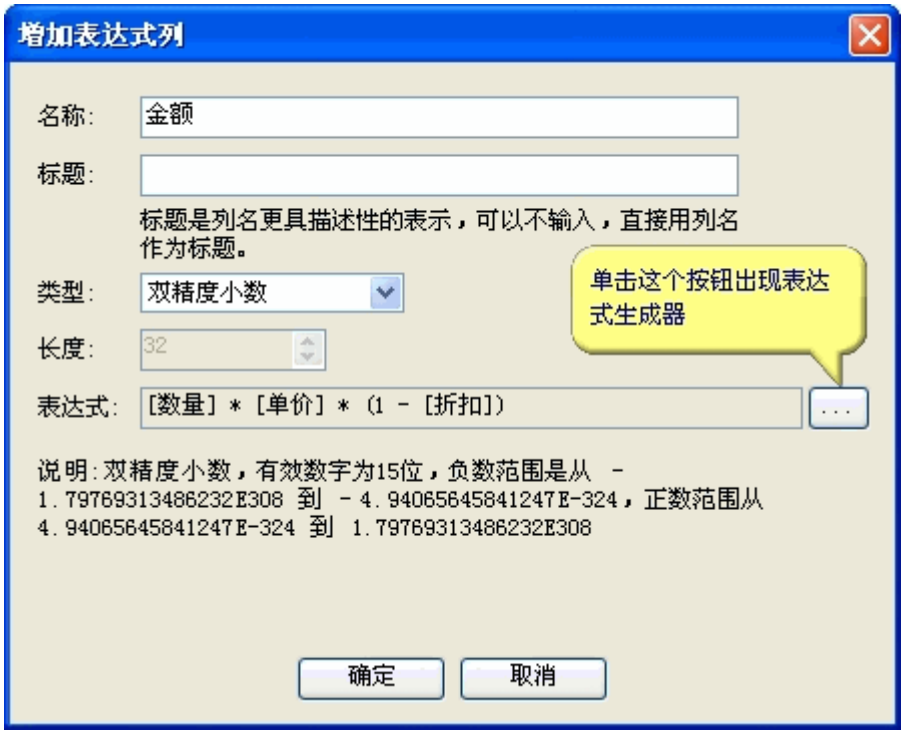
列的数据类型，是决定计算、统计、筛选能否正确执行的基础，务必要正确设置。虽然增加列之后，列的数据类型是可以修改的，但是此时修改可能会带来一些额外的麻烦，例如某个表已经通过某一列和其他表建立了关联，那么在修改该列的数据类型之前，您必须先删除有关的关联，而删除关联，又会影响相关表达式的计算，并导致关联设置的丢失。

3.6.1.3 增加表达式列

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以增加表达式列：



注意增加列是一个组合按钮，你需要单击按钮的下半部分，才会出现下拉菜单。
这是增加表达式列的窗口：



这是表达式生成器：



关于表达式的知识，我们会有专门的一章讲述，现在你不需要对此作过多的研究。

3.6.1.4 修改表达式

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以修改选定列的表达式：



如果选定列不是表达式列，会提示你是否新建一个表达式列。

3.6.1.5 插入列

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以在选定列的位置插入一列：



这是一个组合按钮，直接单击按钮的上半部分，会插入一个数据列，单击按钮的下半部分，会出现一个下拉菜单，用以选择要插入数据列，还是要插入表达式列。

3.6.1.6 删除列

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以删除选定列：



删除列操作即刻生效，且不可撤销，所以在删除一列之前，一定要慎重考虑。
如果选定列参与了表达式的计算，则不能被删除。
假定金额列的计算表达式为：
[数量] * [单价] * (1 - [折扣])
那么数量、单价、折扣这些列是不能被删除的，除非你先删除金额列的表达式。
此外，如果数据表通过某列和其它表建立了关联，那么删除此列之前，必须先删除有关的关联。

3.6.1.7 更改列

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以重命名选定列、设置新的标题、或更改数据类型：

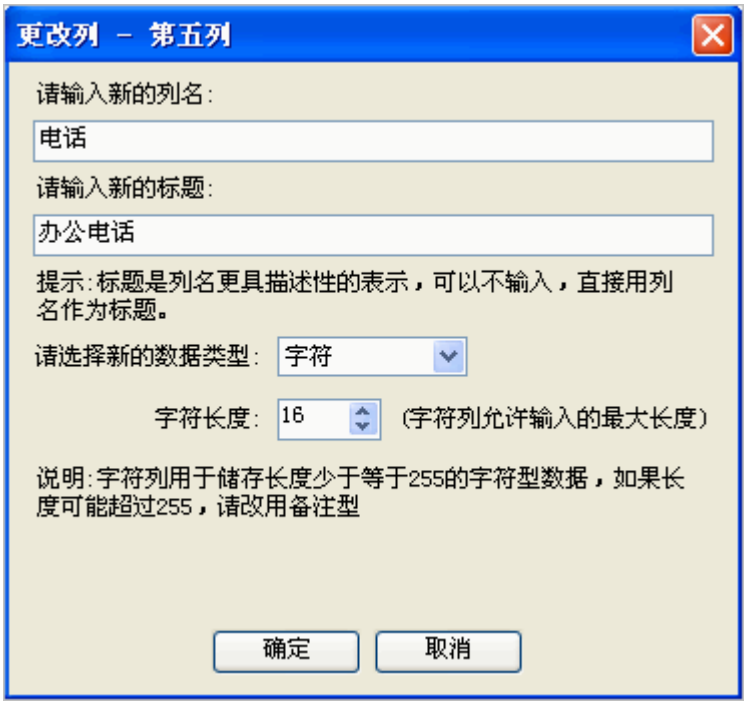


外部数据表中的列，不可以重命名，但可以设置标题。

如果有公式或代码引用了这一列，那么重命名列可能会导致这些公式和代码无法运行。
所以关于列名，我们应该把握几个原则：

- 事先就要慎重考虑，为每个列设置一个合适的列名，项目一旦启用，不要轻易去修改列名。

- 如果一定要重命名某列，而且有大量的公式和代码牵涉到这一列，那么应该考虑用修改列标题来代替，例如你需要将“电话”列重命名为“办公电话”，那么可以将该列的标题改为“办公电话”，而列名保持不变，这样就不会影响项目的正常运行了。



- 多层表头应该尽量使用列标题来实现。

3.6.1.8 数据类型详解

数据类型	说明
字符	字符列用于储存长度少于等于 255 的字符型数据，实际可存储长度，由用户在设计表的时候指定；如果长度可能超过 255，请改用备注型；单个字符，不管是中文字符还是英文字符，在计算长度的时候，均为 1。
备注	用于输入超过 255 字符的文本内容，备注型数据还可以扩展为文件型或者图片型，请在列属性设置中，选择相应的"扩展类型"即可。
日期时间	日期时间列默认只允许输入日期，如果要输入时间型数据，请在列属性设置中，将"日期时间格式"属性设置为"Time"或"LongTime"
逻辑型	逻辑列只有两个值，分别为 True(是)和 False(否)
微整数	介于 0 到 255 之间的微整数

短整数	介于 -32768 到 32767 之间的短整数
整数	介于 -2147483648 到 2147483647 之间的整数，超过此范围的整数可以考虑用双精度小数代替
单精度小数	单精度小数，有效数字为 7 位
双精度小数	双精度小数，有效数字为 15 位
高精度小数	高精度小数，有高达 28 位的有效数字，适用于要求使用大量有效的整数及小数位数并且没有舍入错误的财务计算，可存储的小数位数默认为 4，可根据需要调整。

请合理设置字符列的长度，太长浪费空间，太短则恐怕以后不够用；例如对于公司名称，绝大多数公司的名称会在 15 个字符以内，考虑到特殊单位名称的出现，可以将公司名称列设为 20 个字符，而姓名列，中国人的姓名绝大多数不会超过 4 个字符，所以将姓名列设为 5 个字符足够了，如果涉及到少数民族或者外籍人员，那么姓名列的长度应该设为多少，就得仔细衡量一下了。

数值型有六种，分别是微整数、短整数、整数、单精度小数、双精度小数、高精度小数，除了整数和小数的差别外，主要是范围和精度的差别，范围越大、精度(有效数字)越高，那么占据的存储空间就越大，处理速度就越慢，特别是高精度小数，除非确有需要，否则不要选用。

前面提到有效数字，那么什么是有效数字呢？简单地说，就是从左边第一个不是 0 的数字算起，直到最后一个数字，就是一个数值的有效数字，例如 1.324 的有效数字是 4 位(1、3、2、4)，1.3240 的有效数字是 5 位(1、3、2、4、0)，而 0.024 的有效数字是两位(2、4)。明白了有效数字的概念，有助于我们选择合理的小数类型 。例如对于销售管理中的折扣列，范围在 0 到 1 之间,精度通常不会超过 3 个有效数字，那么选用单精度类型，完全符合要求；而对于金额列，单精度小数区区 7 个有效数字，显然已经不符合要求了，这时就需要选用双精度小数了。

关于数值型列如何选择数据类型，如果不想分析，那么记住一点，没有小数的列选择整数型，有小数的列选择双精度小数型，这会符合绝大多数场合的需要；唯一需要注意的是，整数的范围并不大，在–2147483648 到 2147483647 之间，如超出此范围，请用双精度小数代替。

3.6.1.9 列名与列标题

列标题是列名更具描述性的表示，通常无需设置列标题，此时用列名作为列标题使用。

	产品	客户	雇员	单价	数量	金额	日期
1	PD05	CS03	EP04	30	650	19500	1999-01-02
2	PD02	CS01	EP02			5114.5	2000-05-05
3	PD02	CS02	EP01			2193	2000-10-08
4	PD03	CS02	EP04			8736	2000-02-19
5	PD01	CS04	EP04			9196	1999-06-21
6	PD05	CS05	EP02	29.5	202	5959	2000-11-15
7	PD04	CS01	EP01	21.5	827	17780.5	1999-03-22
8	PD02	CS04	EP03	26.5	794	21041	2000-02-26
9	PD05	CS05	EP01	30	618	18540	1999-03-09

列标题显示在这里，如果没有设置列标题，那么显示列名。

例如，你可以将某一列的名称设为"Q1"，将其标题设为"一季度"，这样在表中，显示的不是列名

“Q1”，而是容易理解的标题“一季度”，但在表达式和代码中仍然是使用"Q1"这个名称；如果项目启用后，您觉得应该用"春季"代替"一季度"，那么只需将列标题改为"春季"即可，您不需要再去调整计算公式和代码，因为列的名称仍然是 Q1。
列名不能以数字开头，更不能全是数字，也不能包括空格和各种符号(下划线除外)，但是列标题没有任何限制，只要你愿意，不同的列甚至可以有相同的标题。

如果列名比较简洁，而且容易理解，之后也没有更改的必要，那么就没有必要设置列标题了。

3.6.1.10 调整列位置

调整列位置有两种方法：

1、单击某列的列标题，选定此列，然后重新用鼠标左键按住该列的列标题，左右拖动，拖动过程会动态显示目标位置，到达目标位置后，松开鼠标即可。

2、在“数据表”功能区，单击“列相关”功能组中的左移列、右移列按钮，即可调整选定列的位置。



提示：

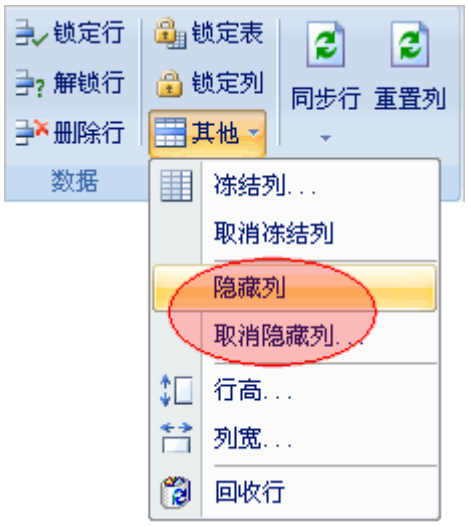
1、如果要同时调整多列的位置，那么在调整列位置之前，选定多列即可。

2、在表属性中，有一个名为“允许拖动列”的属性，将该属性设为 False(否)，将禁止用户通过拖动列标题来调整列位置。

3、调整好列位置后，单击菜单中的保存按钮，重新打开项目后，列位置会保持不变。

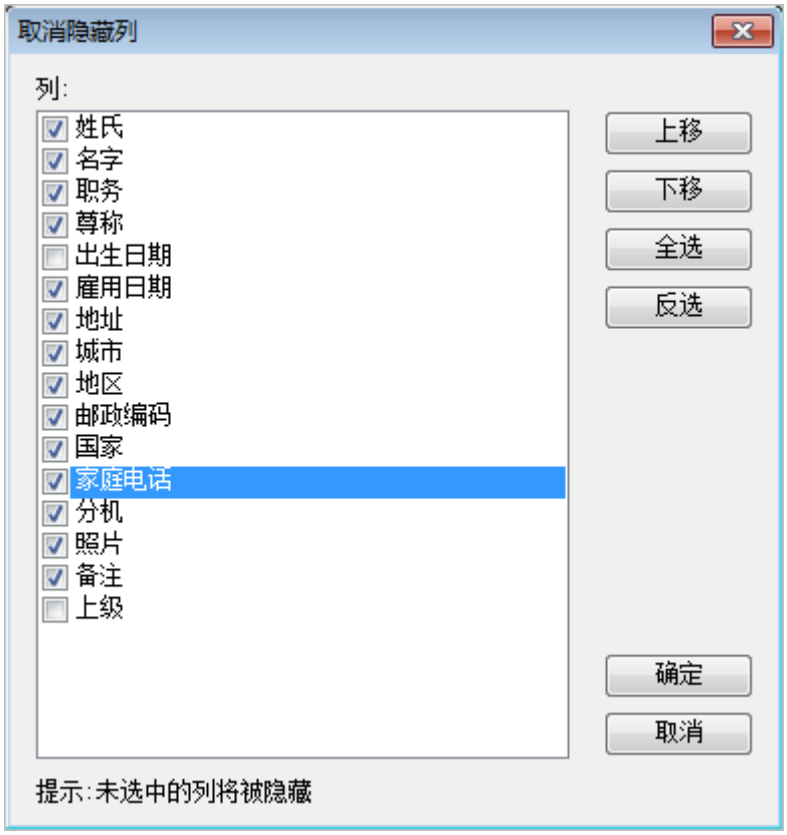
3.6.1.11 隐藏列

在“日常工作”功能区的“数据”功能组，单击“其他”按钮，出现的下拉菜单中，有隐藏列和取消隐藏列的命令：



单击隐藏列命令，将隐藏选定列。

单击取消隐藏列，将出现一个窗口，可以选择要显示的列：



可以看到，这个窗口不仅可以选择要显示的列，还可以调整列的显示位置。

显然，如果要隐藏多列，而且是非连续的，用取消隐藏列窗口更方便。

3.6.1.12 冻结列

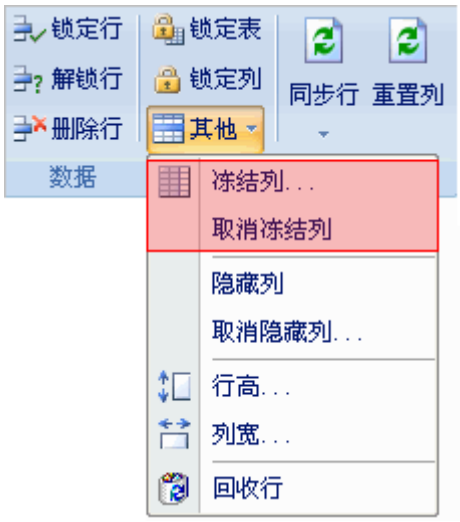
对于一个有很多列的表，可以冻结左边的部分列，左右滚动表时，被冻结的列不会随其它列滚动，而是一直显示在表的最左边，如下图所示:



有两种方式来设置或取消冻结列。

一、利用菜单命令

在“日常工作”功能区的“数据”功能组，单击“其他”按钮，出现的下拉菜单中，有冻结列和取消冻结列的命令：



二、用鼠标直接拖动

1、冻结列

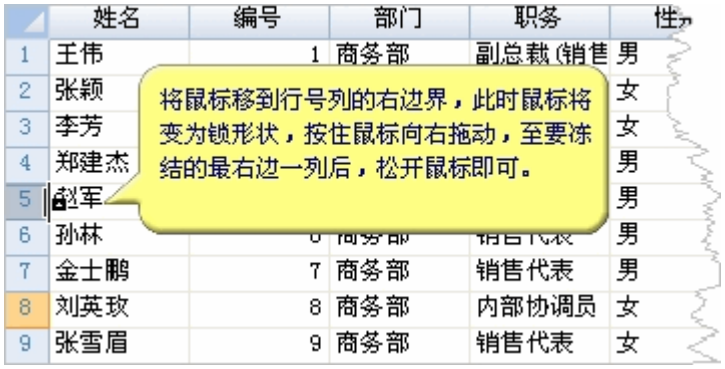
将鼠标移到行号列的右边界，此时鼠标将变为锁形状，按住鼠标向右拖动，至要冻结的最右边一列后，松开鼠标即可。

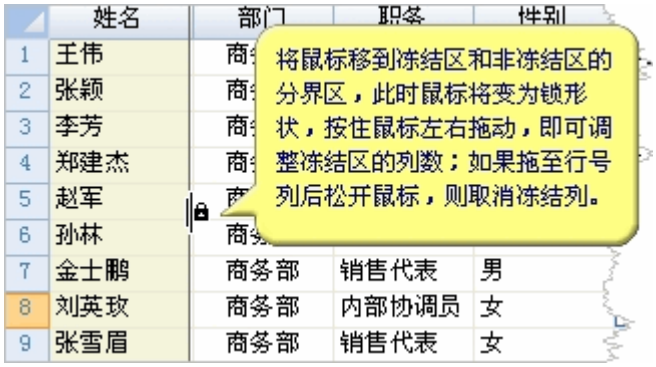
2、调整冻结列数

将鼠标移到冻结区和非冻结区的分界区，此时鼠标将变为锁形状，按住鼠标左右拖动，即可调整冻结区的列数。

3、取消冻结列

将鼠标移到冻结区和非冻结区的分界区，此时鼠标将变为锁形状，按住鼠标向左拖动，到达行号列后松开鼠标，即可取消冻结列。





提示：

1、在表属性中，有一个名为“允许冻结列”的属性，将该属性设为 False(否)，将禁止用户通过拖动来冻结或取消冻结列。

2、通过表样式的配色方案设置，可以改变冻结区的背景颜色和字体颜色 。

3.6.1.13 锁定列

所谓锁定列，就是将列设为只读状态，禁止编辑其中的内容。

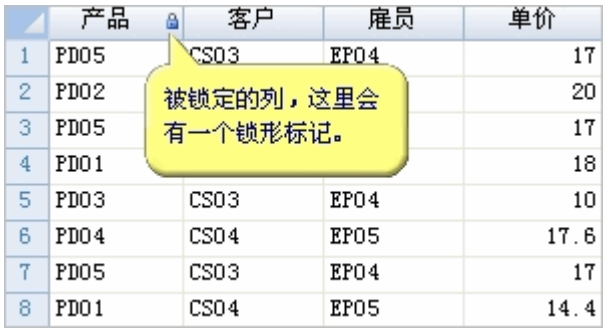
有两种方法可以锁定或取消锁定列：

1、在“日常工作”功能区的“数据”功能组，单击按钮“锁定列”，可以临时性地锁定或取消锁定列：



2、在列属性设置中，将“允许编辑”属性设为 False(否)，可以锁定列，而设为 True(是)，则取消锁定列 ，这种操作是永久性的。

被锁定的列，在列标题处会有一个锁形标记：



如果不希望出现此标记，可以在表样式设置中，将“显示锁定列标记”设为 False(否)。

3.6.1.14 列宽设置

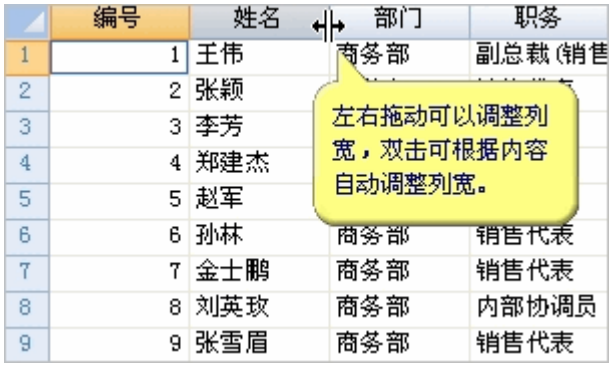
1、调整列宽

拖动列标题的右边界，直到达到所需列宽后松开鼠标。

如果要同时调整多列的宽度，请选定多列，然后再拖动其中一列的右边界。

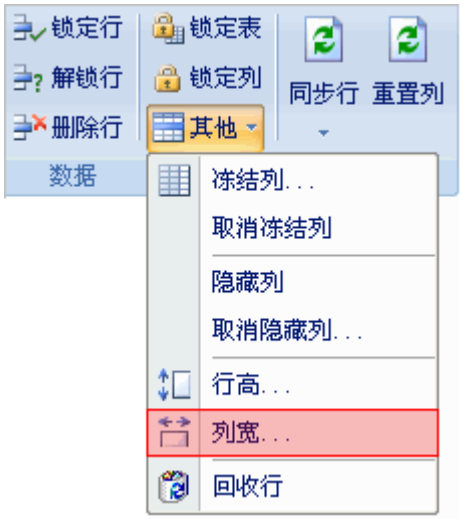
2、自动列宽

双击列标题右边界，即可自动根据列中内容调整列宽。



3、精确设置列宽

在“日常工作”功能区的“数据”功能组，单击“其他”按钮，出现的下拉菜单中，有精确设置列宽的命令：



提示:

调整好列宽后，单击菜单中的保存按钮，重新打开项目后，列宽会保持不变。

3.6.1.15 快捷菜单中的列操作命令

右击表中任何一个单元格，在弹出的快捷菜单中有一个“列相关”菜单，此菜单会列出大部分和列相关的命令：



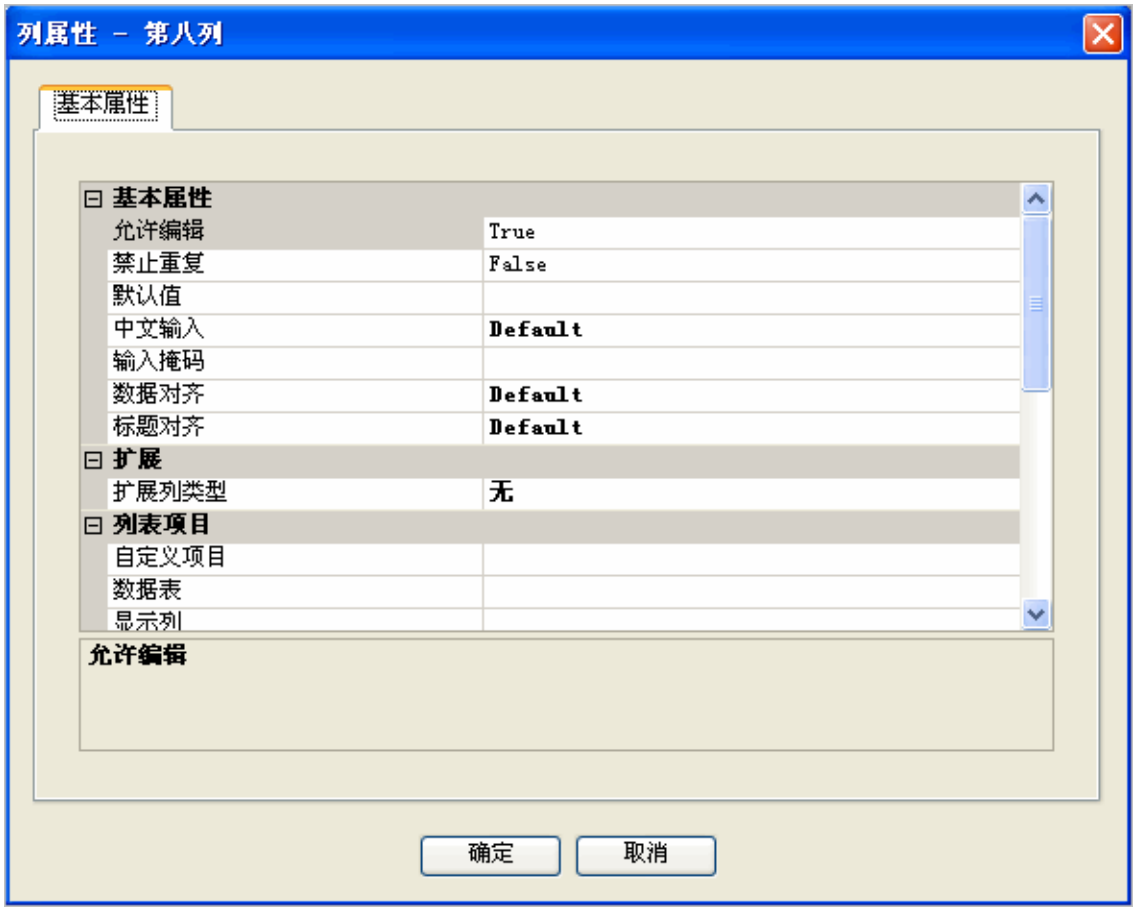
3.6.2 列属性

3.6.2.1 设置列属性

在“数据表”功能区的“列相关”功能组，单击“列属性”按钮，即可设置列属性：



很多非常重要的功能都是通过列属性实现的，接下来我们会逐一讲解每个属性的用途。



3.6.2.2 基本属性

3.6.2.2.1 允许编辑

此属性默认为 True，如果设为 False，将锁定该列，禁止在其中输入数据。
有关锁定列，可以参考：
锁定列

3.6.2.2.2 禁止重复

此属性默认为 False(否)，如果设为 True(是)，将禁止在列中输入重复内容。
一些编号性的列，例如订单编号、身份证号、员工编号，应该将此属性设为 True(是)，以避免出现重复的号码。
如果你输入重复的内容，系统会出现提示，在输入正确内容前，不能离开正在输入的单元格。

3.6.2.2.3 中文输入

通过此属性，你可以控制选择某一列时，是自动打开或关闭中文输入法，还是保持当前状态不变。

该属性有三个可选值：
Default：保持当前输入法状态不变
True：　自动打开中文输入法
False：　自动关闭中文输入法
例如对于姓名列，我们应该将此属性设为 True，因为中国人的姓名自然是中文的；而对于产品型号列，应该将此属性设为 False，因为型号一般由字母和数字组成。
有了此属性，我们可以避免手工切换中英文输入，而由系统根据输入列的设置自动选择。
此属性相当于模拟 Shift 按键。
提示：在数值列输入数字的时候，有的时候第一个输入的数字会无效，只是使得单元格进入编辑状态，如果发生这种情况，请将该列的中文输入属性设为 False。

3.6.2.2.4 输入掩码

一般用户可以忽略本节内容。
输入掩码可以规范和校验用户所输入的数据。
可作为输入掩码使用的字符有：

字符	说明
0	数字（0 到 9，必需项，不允许使用加号 [+] 与减号 [－]）
9	数字或空格（可选项，不允许使用加号和减号）
#	数字或空格（可选项，允许使用加号和减号）
L	字母（A 到 Z，必需项）
?	字母（A 到 Z，可选项）
A	字母或数字（必需项）
a	字母或数字（可选项）
&	任一字符或空格（必需项）
C	任一字符或空格（可选项）。
. , : ; - /	小数点占位符及千位、日期与时间的分隔符
<	将所有字符转换为小写
>	将所有字符转换为大写
\	使接下来的字符以表意字符显示（例如，\A 只显示为 A）

例如，产品型号列的格式是三个字母、一个横线、两个数字组成，例如“KLU-73”，可以将该列的输入掩码设为“AAA-00”，如果要求三个字母必须为大写，那么掩码可以设为“>AAA-00”。

3.6.2.2.5 数据对齐

在默认情况下，数值型数据靠右，其它类型数据靠左。
通过设置数据对齐属性，可以根据需要改变这种默认的对齐方式。
该属性有三个可选值：
Default：默认

Left： 靠左
Center： 居中
Right： 靠右
此外，通过菜单“数据表”功能区“样式”功能组中的三个按钮，可以用可视化的方式设置数据对齐方式：



3.6.2.2.6 标题对齐

在默认情况下，列标题都是居中对齐的。
通过设置标题对齐属性，你可以根据需要控制列标题的对齐方式。
该属性有三个可选值：
Default：默认
Left： 靠左
Center： 居中
Right： 靠右

3.6.2.2.7 日期时间格式

Date： 标准日期格式，如"2008-12-31"
DateTime： 短日期时间格式，包括日期和时分，如"2008-12-31 9:45"
DateLongTime： 长日期时间格式，包括日期和时分秒，如"2008-12-31 9:45:30"
Time： 短时间格式，包括时分，如“12:59”
LongTime： 长时间格式，包括时分秒，如“12:59:21”

3.6.2.2.8 设置数值格式

- **最大小数位数**
  只有小数型的列才会出现此属性。
  此属性决定该列允许输入的最大小数位数，修改该属性，不会影响现有数据的值，例如列中某单元格内容为 1.234，如果将最大小数位数修改为 2，该单元格将显示 1.23，但是它的值还是 1.234；对于新输入的数据，程序会自动按照所设置的最大小数位数来进行四舍五入，同样假定最大小数位数为 2，如果输入 1.234，那么单元格存储和显示的值都是 1.23，如果输入 1.236，那么该列存储和显示的值是 1.24

- **固定小数位数**
  只有小数型的列才会出现此属性。
  该属性默认为 False(否)，如果设为 True(是)，值的实际小数位数小于最大小数位数时，会自动

在后面补 0，凑够至最大小数位数；例如某单元格的值是 1.23，假定最大小数位数为 3，那么该单元格显示的内容是 1.230，而不是 1.23

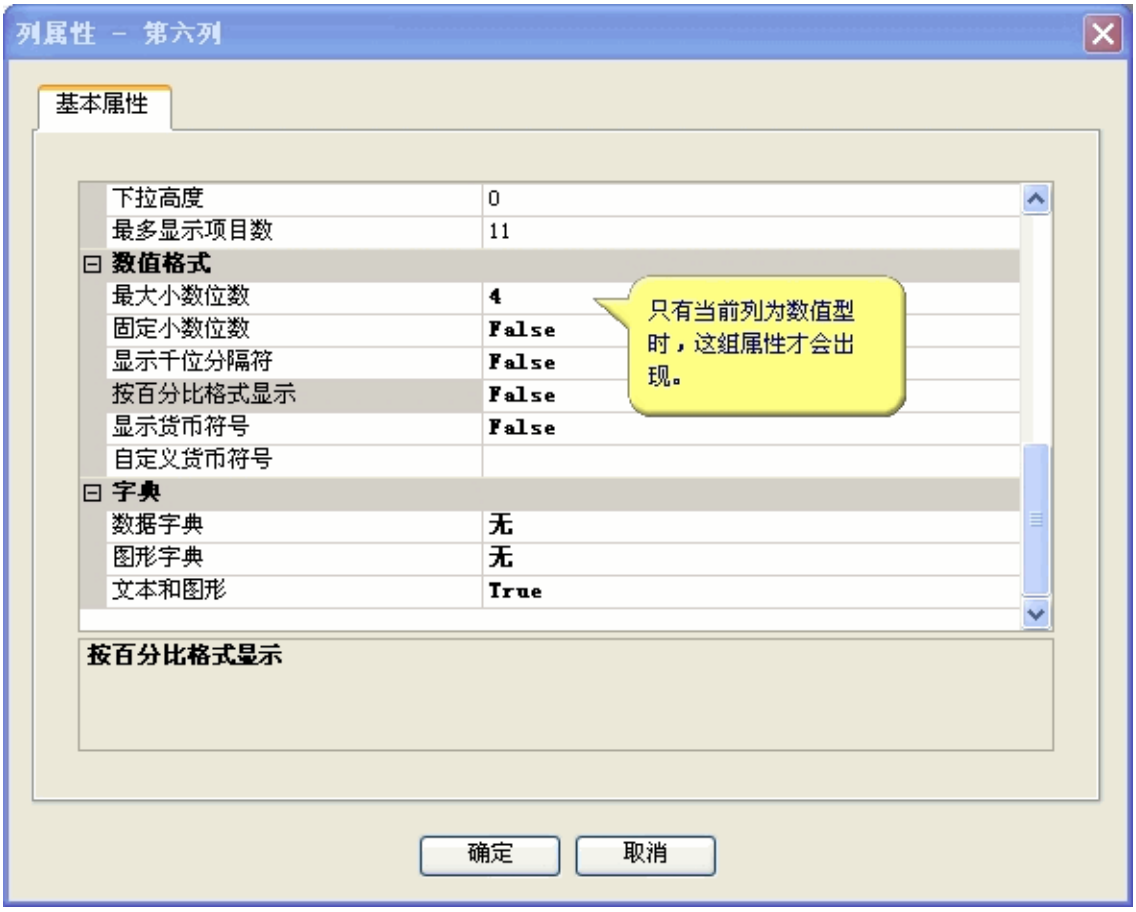
- **固定整数位数**
  只有整数型的列才会出现此属性。
  设置在小数点左边固定显示的整数位数，例如此属性设为 3，数值 1 将显示为“001”。
  默认值为 0，表示不固定。

- **显示千位分隔符**
  如果此属性设为 True(是)，那么每三位整数中间插入一个千位分隔符（逗号），例如：1,234

- **按百分比格式显示**
  此属性设为 True(是)的话，会自动将数值乘以一百，然后加上百分比符号显示，例如 0.231，显示为 23.1%

- **显示货币符号**
  是否在数值前显示货币符号，例如：￥234

- **自定义货币符号**
  默认的货币符号为￥，利用此属性，你可以修改为您需要的货币符号，例如：$



3.6.2.2.9 逻辑格式

如果选定列是逻辑列，列属性窗口会出现一个“逻辑格式”属性。
在默认情况下，逻辑列以检查框的形式显示，检查框被选中表示 True，否则表示 False。
如果你希望用字符型形式表示逻辑值，例如开/关、高/低等等，可以通过设置逻辑格式属性来实现。例如你希望单元格值为 True 的时候显示“开”，单元格内容为 False 的时候显示“关”，只需将逻辑格式属性设为“开;关”即可，注意字符之间是用分号(;)分割的。
对于设置了格式的逻辑列，可以通过双击单元格或按空格键来改变单元格内容。



折扣	数量	金额	日期	结账
0.1	650	9945	1999-01-02	☑
0.1	400	7200	1999-01-03	☑
0	320	1920	1999-01-04	☑
0.15	80	544	1999-01-05	☐
0	490	3430	1999-01-06	☐
0	240	4224	1999-01-07	☑
0	160	2720	1999-01-07	☐
0	200	2880	1999-01-08	☑
0.1	200	1440	1999-01-08	☑
0.2	800	11520	1999-01-10	☐

折扣	数量	金额	日期	结账
0.1	650	9945	1999-01-02	是
0.1	400	7200	1999-01-03	是
0	320	1920	1999-01-04	是
0.15	80	544	1999-01-05	否
0	490	3430	1999-01-06	否
0	240	4224	1999-01-07	是
0	160	2720	1999-01-07	否
0	200	2880	1999-01-08	是
0.1	200	1440	1999-01-08	是
0.2	800	11520	1999-01-10	否

3.6.2.2.10 使用内置输入器

当列数据类型为日期或者数值型时，此属性才会出现在列属性设置窗口。
此属性默认为 False，设为 True 的话，在输入数据的时候，单元格会自动出现一个下拉按钮，单击该按钮，会出现一个日历或者计算器，用于输入数据。

数值输入器：



日期输入器：



3.6.2.2.11 默认值

默认值属性是一个表达式，当新增一行时，会自动计算该表达式，并将计算结果填入到列中。

默认值采用 VBScript 语法，例如我们希望新增一行时，某列能够自动填入该列的日期，只需将该列的默认值属性设为：

Date()

再例如，有一个字符型的结算方式列，默认的结算方式为现金，可以将该列的默认值属性设为:

"现金"

注意字符型的默认值一定要加上双引号，因为默认值是一个表达式。

如果是数值，则不需要加双引号。

默认值采用 VBScript 语法。

请参考：

VBScript 参考

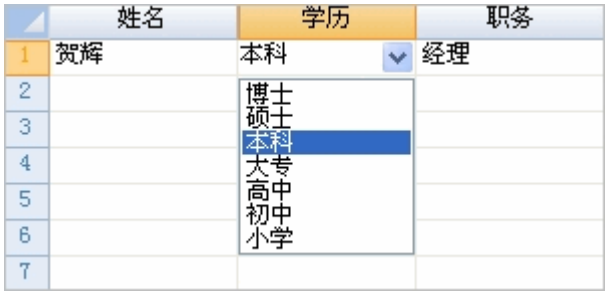
3.6.2.2.12 列属性与列类型

前面已经知道，不同数据类型的列，列属性的条目是不同的，例如只有单精度小数或双精度小数列，才会有"最大小数位数"和"固定小数位数"这样的列属性。
正确设置列的数据类型，是一个管理项目正常运转的基础，初学者一定要注意。
例如一个订单表，客户列和产品列是字符型，订购日期列是日期时间型，数量列是整数型，折扣列和金额列则是双精度小数型。
如果要检查或更改某一列的数据类型，请参考：更改列
如果要检查某个表所有列的数据类型，请参考：查看表结构
要详细了解各种数据类型，请参考：数据类型详解

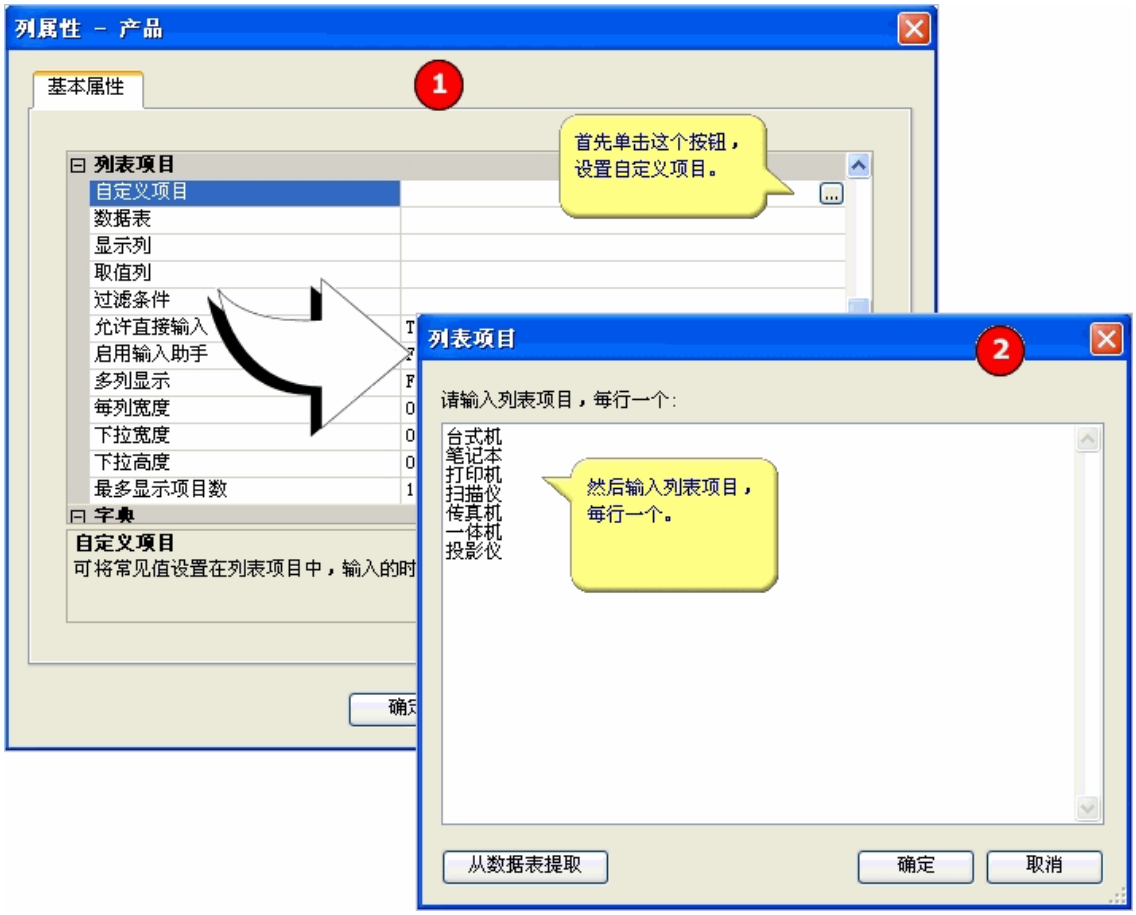
3.6.2.3 列表项目

3.6.2.3.1 设置自定义项目

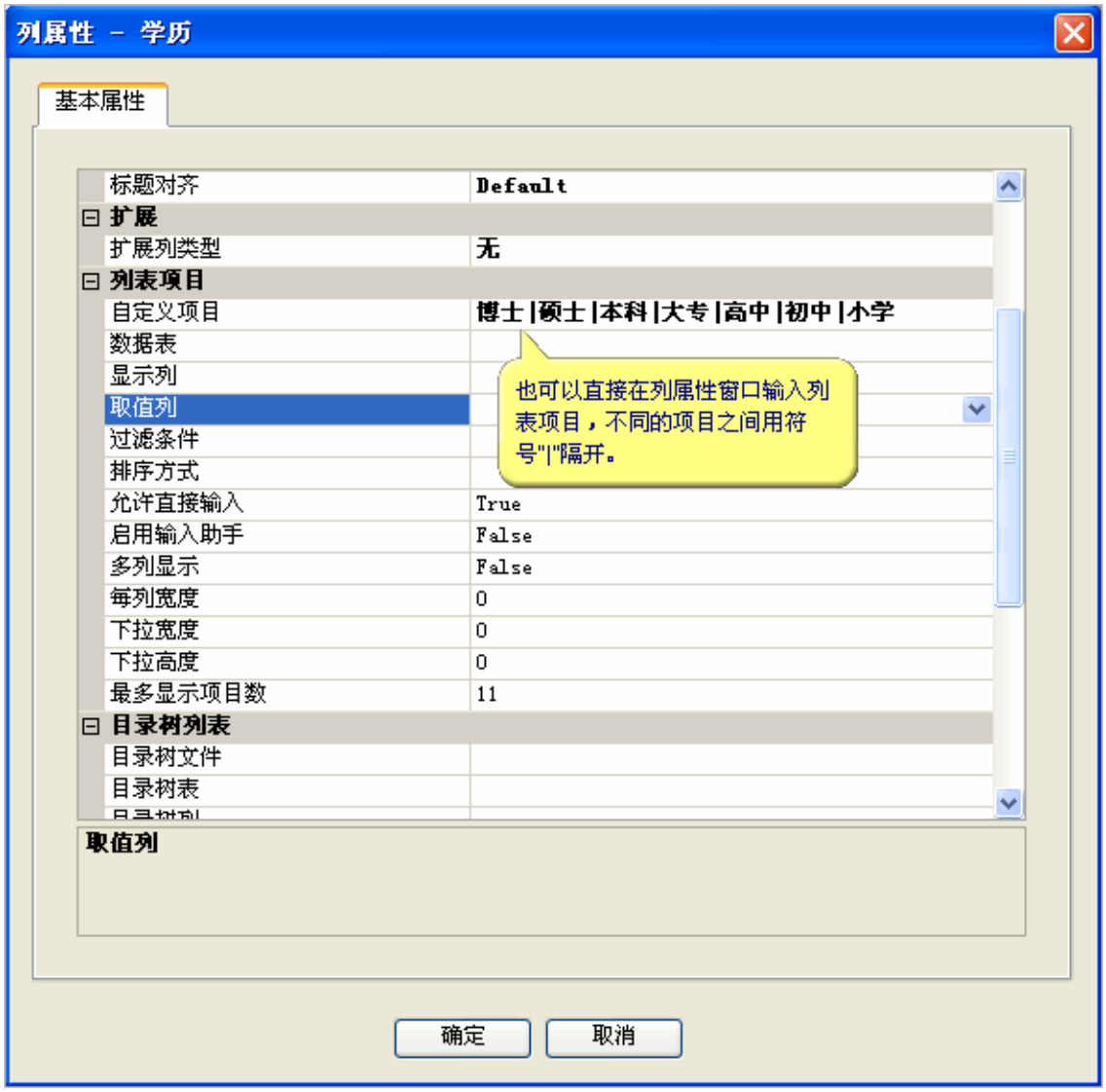
通过设置列表项目，可以提高某些数据的录入速度，并能减少录入错误。例如在人事管理系统中，可以将学历列的列表项目设为“博士|硕士|本科|大专|高中|初中|小学”，这样一旦选择学历列，将出现一个下拉列表框，可以从中选择学历：



设置列表项目的方法很简单，首先选定此列，打开列属性设置窗口，参考下图所示设置:



也可以直接在列属性窗口输入列表项目，不同的项目之间用符号"|"隔开：



3.6.2.3.2 禁止直接输入

在默认情况下，用户不仅可以从列表项目中选择内容，还可以直接输入不在列表项目中的内容。如果你要求用户只能从列表项目中选择内容，不能直接输入，可以在列属性设置窗口中，将属性“允许直接输入”设为 False(否)。

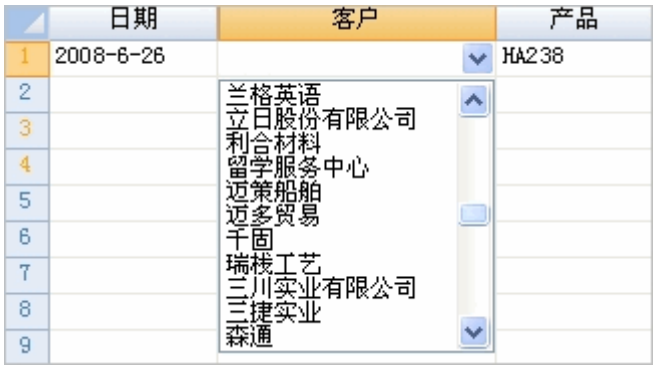
3.6.2.3.3 用键盘选择列表项目

大多数时候，我们是通过鼠标从下拉列表中选择要输入的项目。
不过也可以通过键盘来操作，选择一个空白单元格，按空格键进入编辑状态，再次按空格键会自

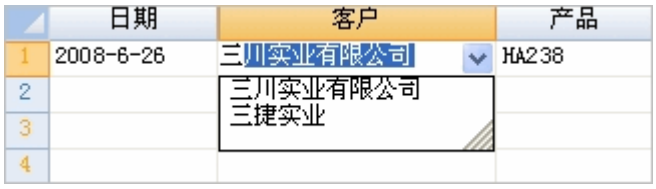
动展开下拉列表，按上下箭头选择项目，按空格键或者回车键确认选择。
此特性可以让习惯使用键盘的用户，也能通过列表项目快速输入数据。

3.6.2.3.4 启用输入助手

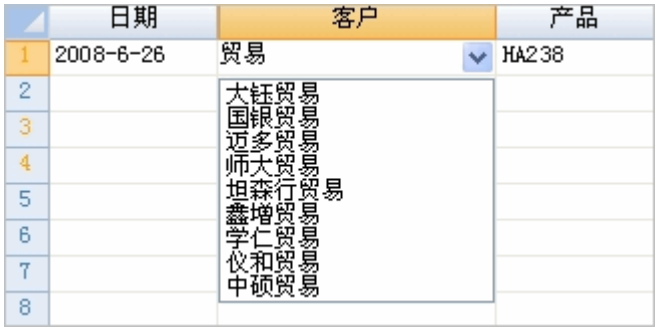
如果列表项目上百个，甚至更多，那么从中找出自己需要的也不容易。
例如下图的客户列，下拉列表中有数百个客户名称供选择，从中找出自己想输入的客户名称，可能还不如直接输入来得快：



可是直接输入的话，又难免会出现输入错误，对于电脑来说，“北京市公安局”和“北京公安局”可是完全不同的两个单位。
为解决这个问题，可以在列属性设置窗口中，将属性“启用输入助手”设为 True(是)，这将启动两个辅助性的输入功能。
1、自动完成输入
输入一个字符，系统会自动输入第一个以该字符开始的项目，同时筛选出所有以该字符开始的项目供选择；你可以按 Tab 键接受自动输入的项目，也可以选择筛选出来的候选项目；随着输入字符的增多，会更精确地定位出您可能要输入的项目。
例如在下图中，用户只是输入了一个字符“三”，但是程序却自动输入了第一个以“三”开头的项目“三川实业有限公司”，同时列出了所有以“三”开头的项目供选择，你可以按 Tab 键接受自动输入的内容，也可以从列出的候选项目中选择一个。

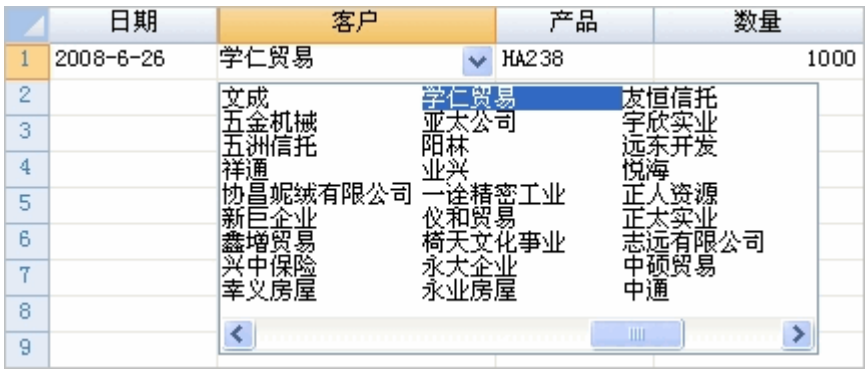


2、高级筛选功能
前面提到的自动筛选功能，只能筛选出以输入字符开始的项目。
如果需要筛选出包括输入字符的项目，可以按“Ctrl + Enter”键。
例如下图中，我们输入“贸易”二字，然后按“Ctrl + Enter”键，即可自动列出所有包括“贸易”二字的项目供选择。



3.6.2.3.5 分多列显示下拉列表

1、将“多列显示”属性设为 True(是)。
2、设置“每列宽度”属性，单位为像素。
3、设置“下拉宽度”属性，下拉宽度应该等于：每列宽度 * N，N 为你希望显示的列数。
例如下图，每列宽度等于 100，下拉宽度等于 300，下拉列表分三列显示：



3.6.2.3.6 控制下拉列表框大小

以下属性用于精确控制下拉列表的大小，其实使用这些属性的机会不多。

最多显示项目数

下拉列表默认最多显示 11 项，其余的需要滚动才能显示，可以用此属性修改最多显示的项目数。

下拉宽度

此属性用于设置下拉列表的宽度，如果设为 0，则等于列宽。

下拉高度

此属性用于设置下拉列表的高度，如果设为 0，高度由“最多显示项目数”决定。
如果所设置的高度，不足以显示“最多显示项目数”指定的项目数，将自动加大高度。

3.6.2.3.7 列表项目来自于列

列表项目不仅可以自定义，还可以直接用某一列中已经输入的内容作为列表项目。
例如客户表中已经输入了客户名称，订单表中也有客户名称一列，为了直接使用客户表中已经输入的客户名称作为列表项目，首先选择订单表中的客户名称列，然后如下设置该列的列属性：
1、将“数据表”属性设为“客户”。

2、将“显示列”和“取值列”属性都设为“客户名称”。

经过这样的设置，在订单表中输入客户名称时，会自动列出客户表中已经输入的客户名称供选择。



3.6.2.3.8 显示列和取值列

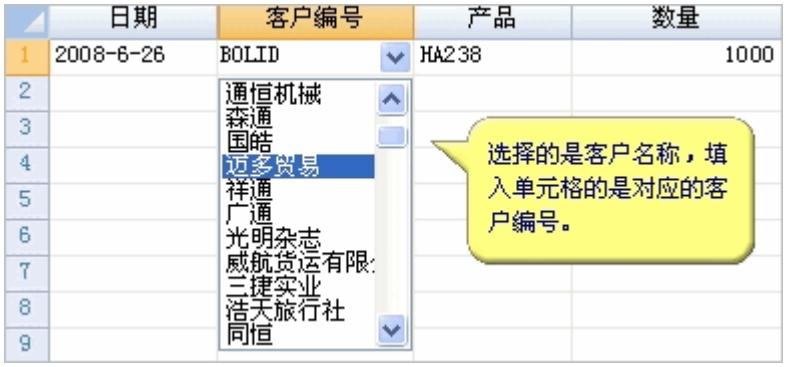
列表项目来自于列时，“显示列”和“取值列”属性有着不同的意义。

展开下拉列表时，显示哪一列的内容，由“显示列”属性决定。

单击某个列表项目时，将哪一列的内容填入到单元格中，由“取值列”属性决定。

例如订单表中有一个客户编号列，其列表项目来自于客户表的客户编号列，如果“显示列”和“取值列”属性都设为客户编号列，那么从下拉列表中选择正确的客户编号可不容易，你也许知道“威航货运有限公司”的客户编号是“CACTU”，但是你能知道所有客户的编号吗？

最好的办法是将“显示列”属性设为“客户名称”，“取值列”属性设为“客户编号”，这样下拉列表显示的是客户名称，而单击某一客户名称，单元格即可自动填入该客户对应的客户编号。



3.6.2.3.9 排序方式

如果列表项目来自于某一列，那么列表项目默认按照取值列的内容来排序。
通过设置排序方式属性，可以改变这种默认的排序。
例如列表项目的取值列为客户编号，显示列为客户名称，下拉列表中显示的客户名称会根据客户编号来排序，如果你希望下拉列表能够直接按照客户名称来排序，只需将排序方式属性设为：客户名称 。
默认的排序是升序，如果希望降序排序，可以在排序列名后面加上 DESC，例如：日期 DESC

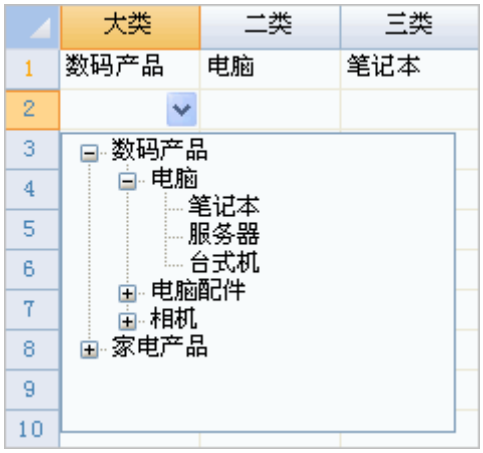
3.6.2.3.10 过滤条件

列表项目来自于列的时候，你可以用“过滤条件”属性设置选择哪些行用于列表项目。
例如来自于客户表的时候，你希望只列出华北地区的客户，你可以设置条件为：
[地区] = '华北'
有关表达式的知识，我们会在后续章节讲述，现在你只需记住：表达式中的字符是用单引号括起来的。

3.6.2.4 目录树列表

3.6.2.4.1 关于目录树列表

目录树列表能够更加高效地输入数据 ，例如下图的大类列就设置了目录树列表，在大类列单击下拉按钮，会出现一个目录树，单击目录树中的某个节点，能够同时输入大类、二类、三类这三列的数据：



可以看出，相对于列表项目，目录树列表具备两点优势：
1、可以一次输入多列内容
2、可以对候选项目进行分组，便于快速定位到要输入的项目。

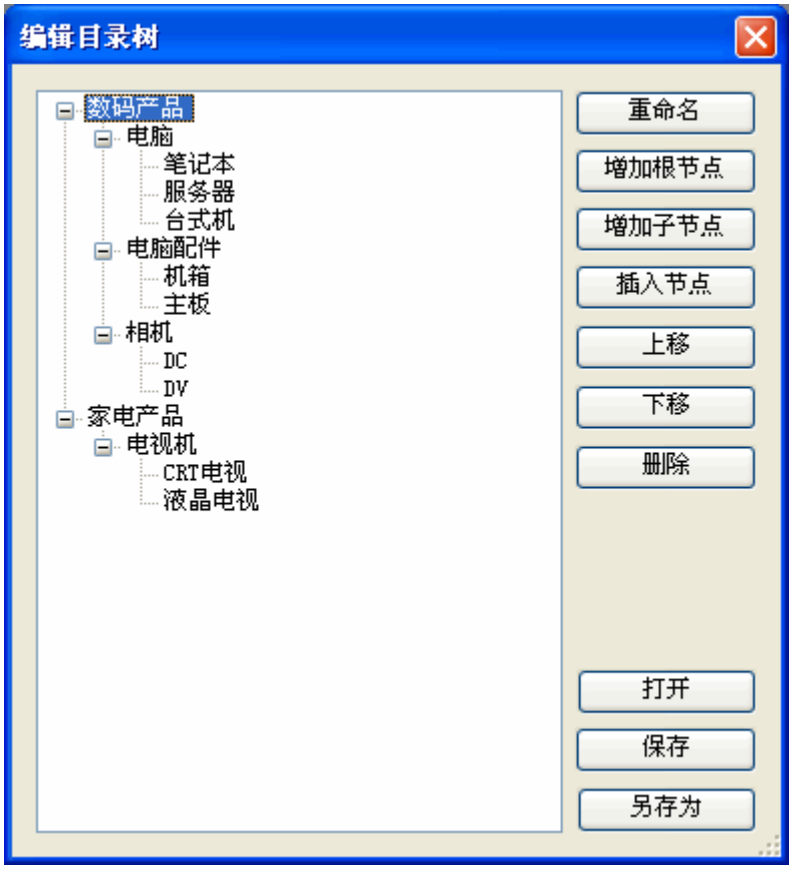
3.6.2.4.2 文件型目录树

3.6.2.4.2.1 创建目录树文件

使用目录树列表之前，通常需要先创建一个目录树文件。
在菜单的“杂项”功能区，单击下面的按钮：



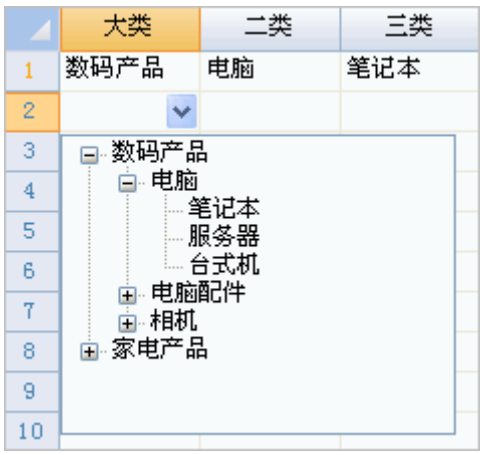
可以打开目录树编辑窗口，可以在这里创建目录树文件，也可以打开一个现有的目录树文件修改：



提示：目录树文件最好保存在项目的 Attachments 文件夹中，以便于系统调用。

3.6.2.4.2.2 使用目录树文件

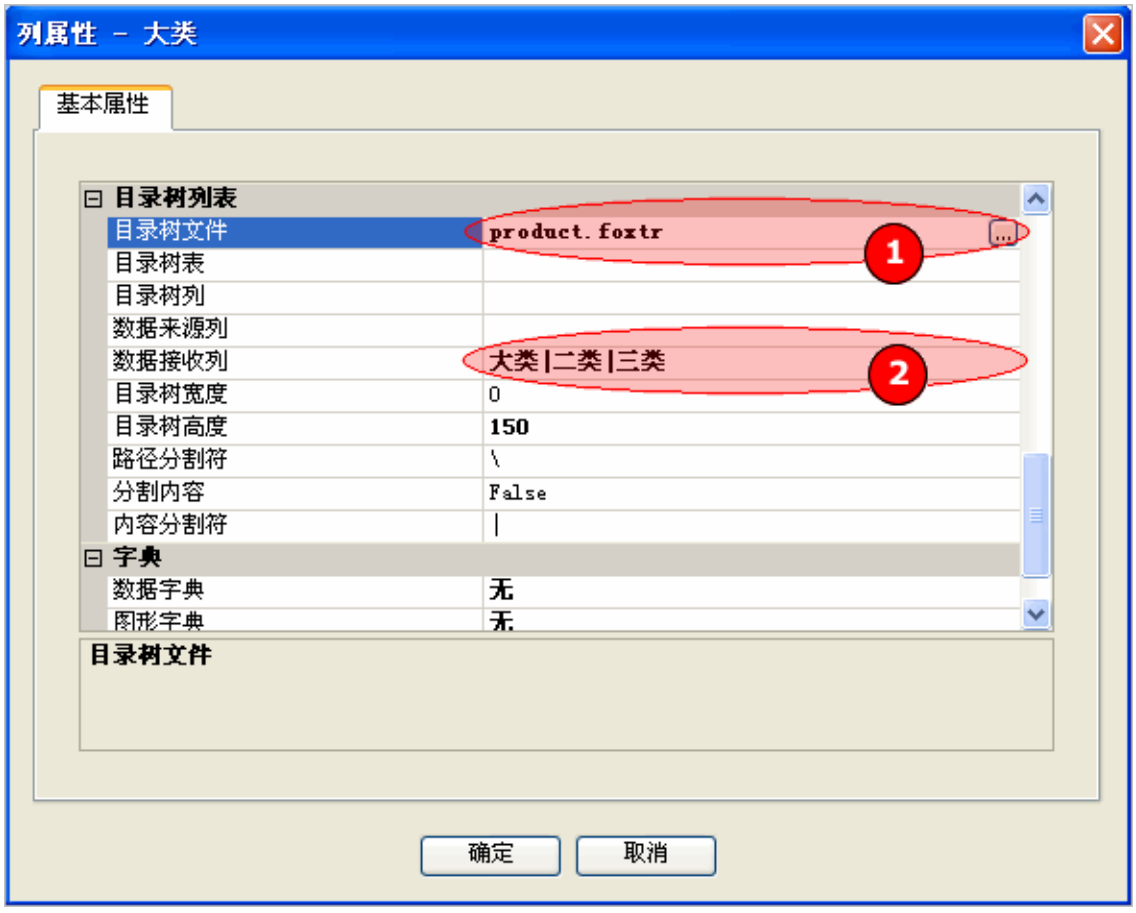
本节内容可以参考 CaseStudy 目录下的文件：文件型目录树.Table
创建了目录树文件后，就可以使用目录树录入数据了。
假定有一个表，希望使用下图所示的目录树输入数据：



设计步骤

1、首先按照上一节介绍的那样，创建一个目录树文件。

2、然后选择“大类”列，打开其列属性设置窗口，选择目录树文件和数据接收列:



提示：目录树文件最好保存在项目的 Attachments 文件夹中，以便于系统调用，即使放在其它目录，在列属性设置中选择目录树文件后，也会自动将所选择的目录树文件复制到 Attachments 文件夹中，调用的也是此文件夹中的目录树文件。

3.6.2.4.2.3 使用内容分割符号

本节内容可以参考 CaseStudy 目录下的文件：文件型目录树.Table

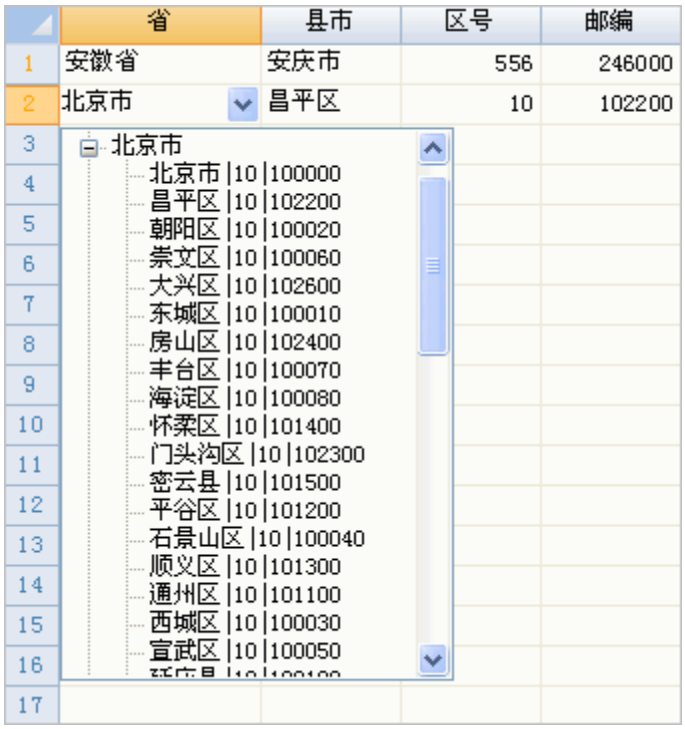
有的时候，目录树一个节点的内容，可能包括多项内容。
例如对于下面的目录树，二级节点就包括县市、区号、邮编三项内容，三项内容用符号“|”隔开了：



在使用目录数列表的时候，为了将二级节点的三项内容对应地填入县市、区号、邮编三列中，可以按照下图所示设置列属性：



经过上述设置，当我们选择某个节点的时候，二级节点的内容能够对应地输入到县市、区号、邮编三列中：



3.6.2.4.3 数据表型目录树

3.6.2.4.3.1 数据表目录树

本节内容可以参考 CaseStudy 目录下的文件：数据表目录树.Table
除了根据文件，还可以根据数据表生成目录树。

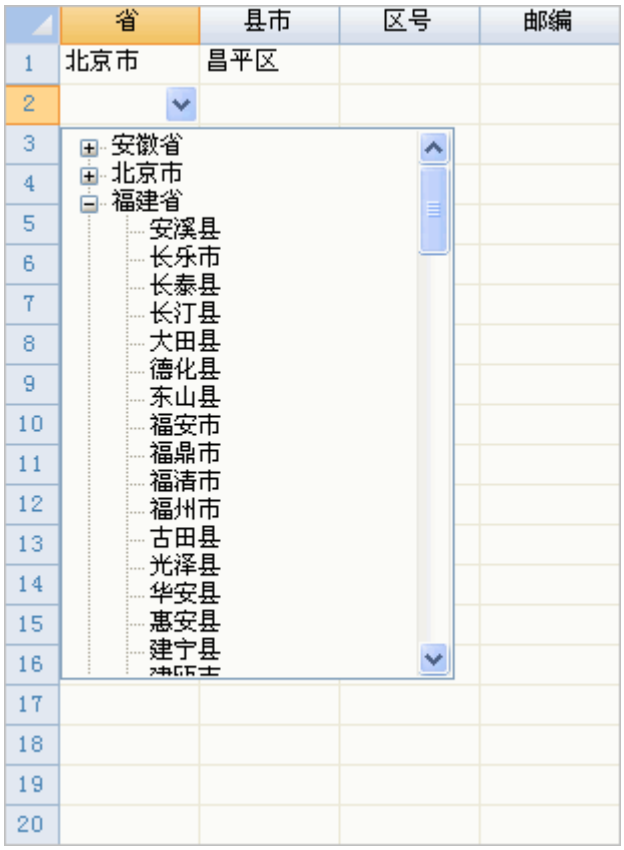
例如有一个行政区域表，结构如下：

	省	县市	区号	邮编
1	北京市	北京市	10	100000
2	北京市	东城区	10	100010
3	北京市	西城区	10	100030
4	北京市	崇文区	10	100060
5	北京市	宣武区	10	100050
6	北京市	朝阳区	10	100020

我们希望在输入数据的时候，可以根据该数据表自动生成目录树，用于输入省和县市列的内容。
为此可以在数据输入表的“省”列，按下图所示设置列属性：



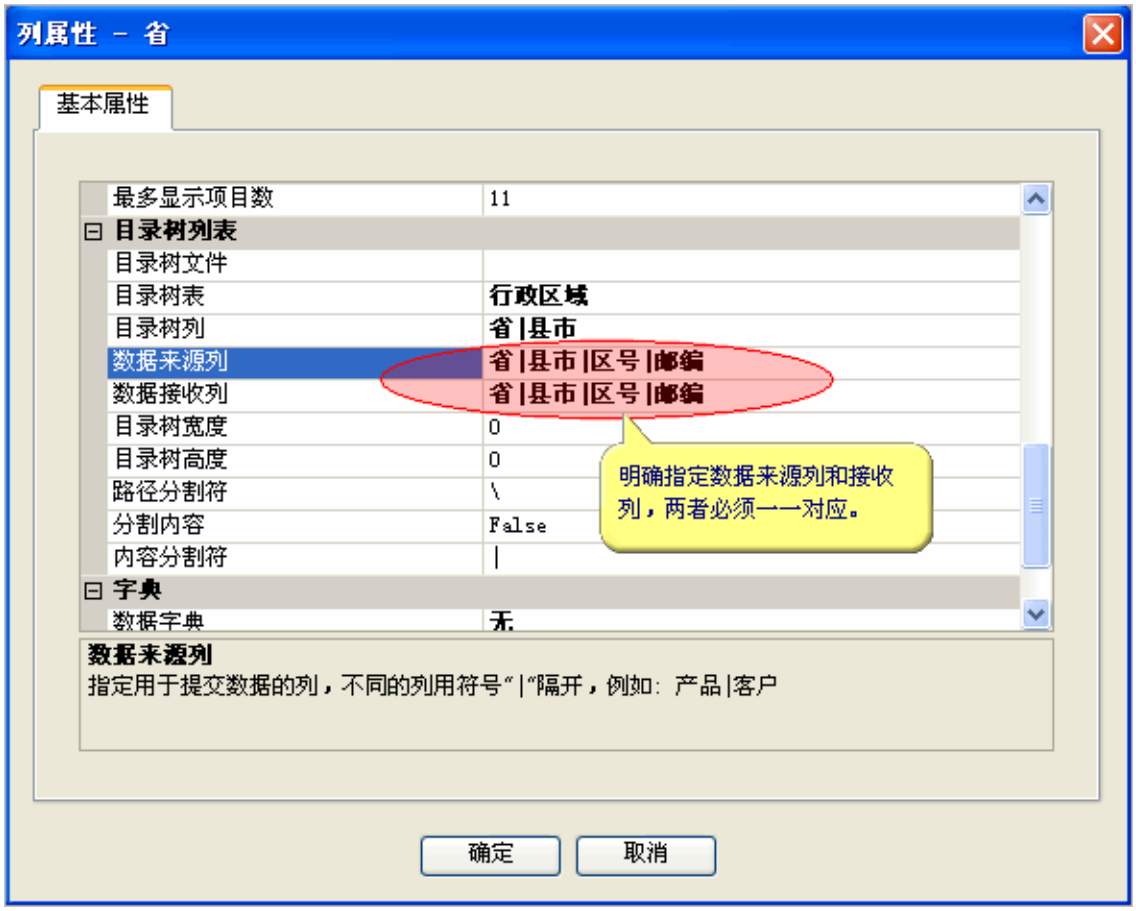
即可得到下图所示的下拉目录树：



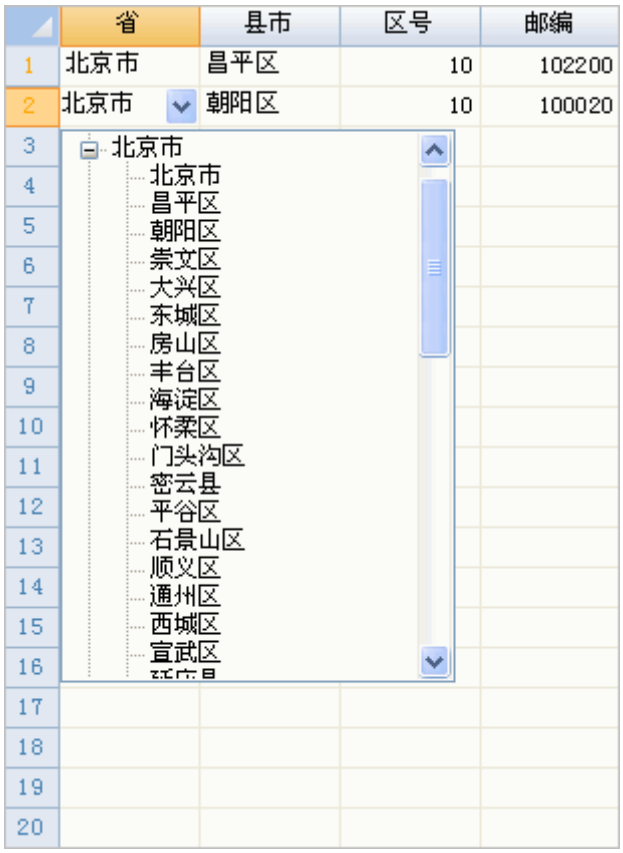
明确指定数据来源列

上面的设置只能输入省、县市两列的内容，而区号和邮编由于没有参与目录树的生成，所以无法输入到数据表中。

不过我们可以通过明确指定数据来源列，来输入区号和邮编两列的数据：



通过上面的设置后，可以正常输入区号和邮编，尽管这两列并没有用于生成目录树：



3.6.2.4.3.2 使用内容分割符

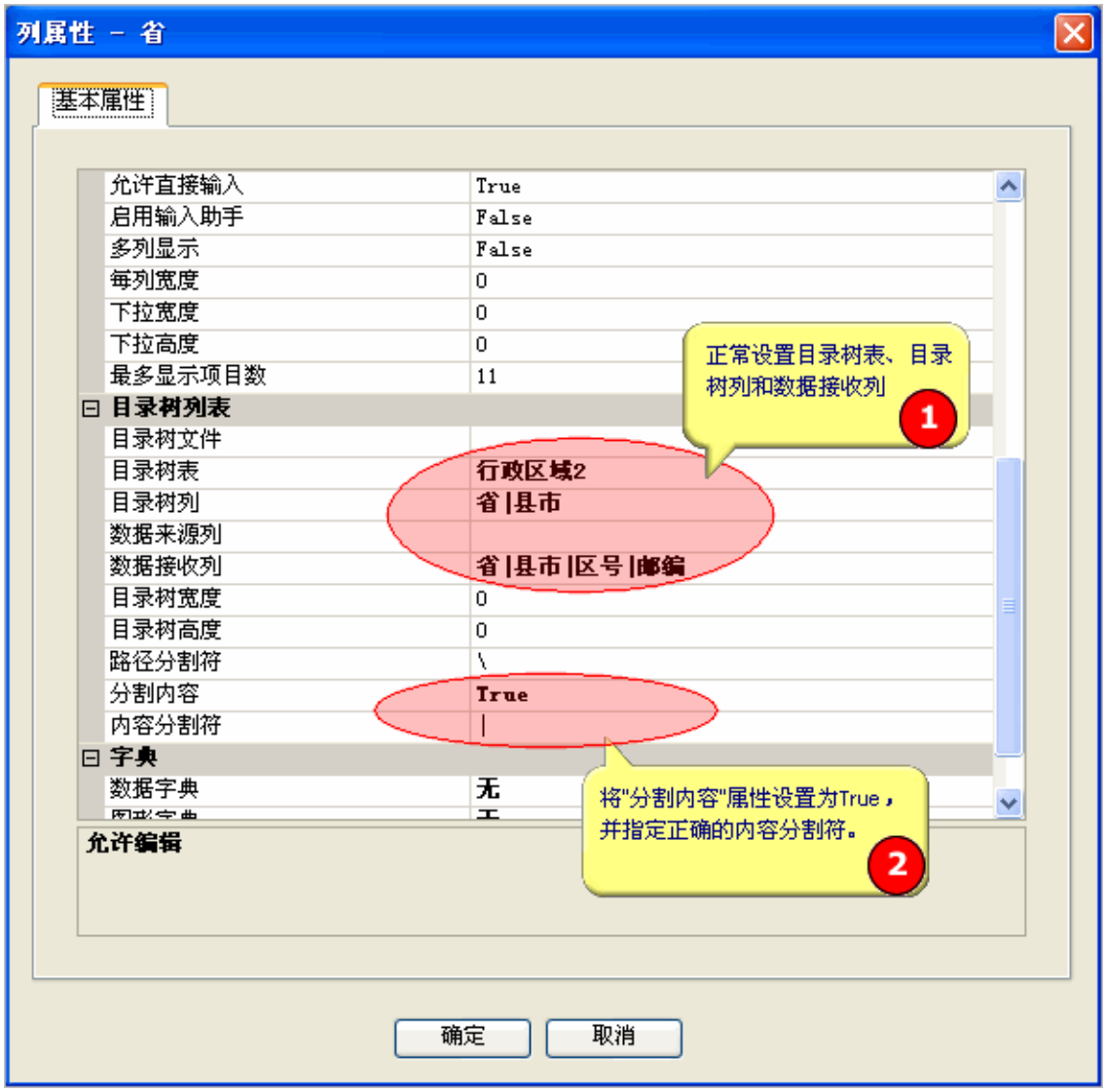
本节内容可以参考 CaseStudy 目录下的文件：数据表目录树.Table

有的时候，用于生成目录树的表，其中一列的内容包括多个栏目的内容。

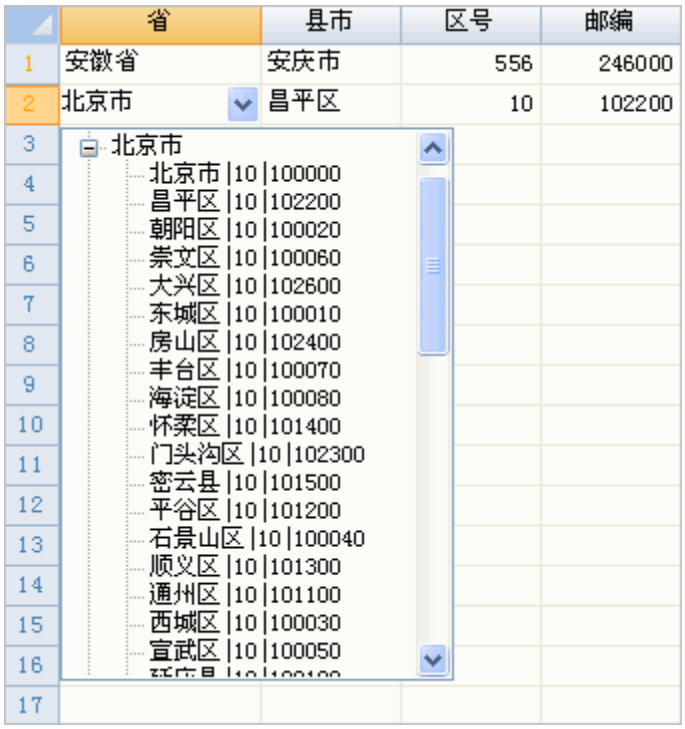
例如下面的县市列，就包括县市、区号、邮编三个栏目的内容：

	省	县市
1	北京市	北京市|10|100000
2	北京市	东城区|10|100010
3	北京市	西城区|10|100030
4	北京市	崇文区|10|100060
5	北京市	宣武区|10|100050
6	北京市	朝阳区|10|100020
7	北京市	丰台区|10|100070
8	北京市	石景山区|10|100040
9	北京市	海淀区|10|100080
10	北京市	门头沟区|10|102300

为了在目标表中，能够将上图中县市列的内容拆分成 3 个部分，对应地输入到县市、区号、邮编三列中，必须按下图所示设置列属性：



经过上述设置，当我们选择某个节点的时候，二级节点的内容能够对应地输入到县市、区号、邮编三列中：



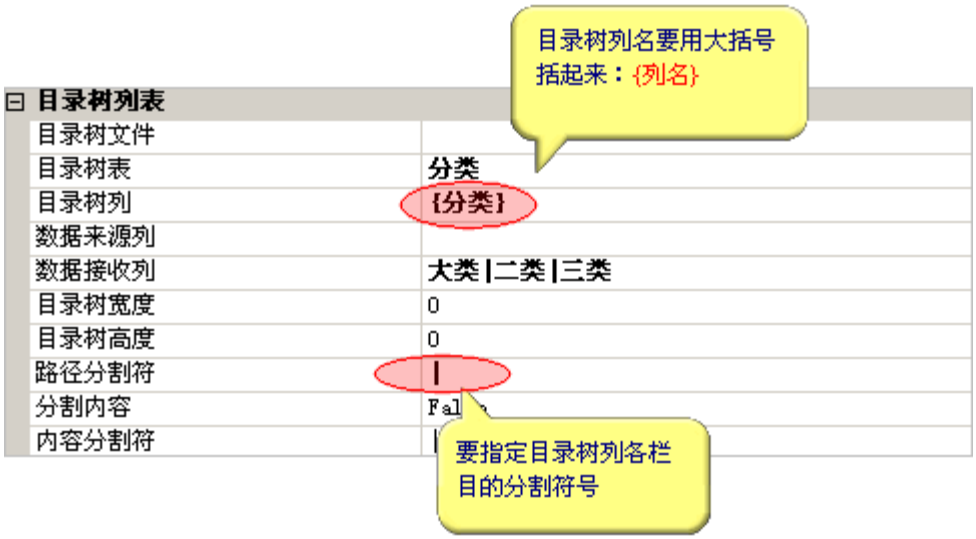
3.6.2.4.3.3 根据单列内容生成目录树

本节内容可以参考 CaseStudy 目录下的文件：数据表目录树.Table

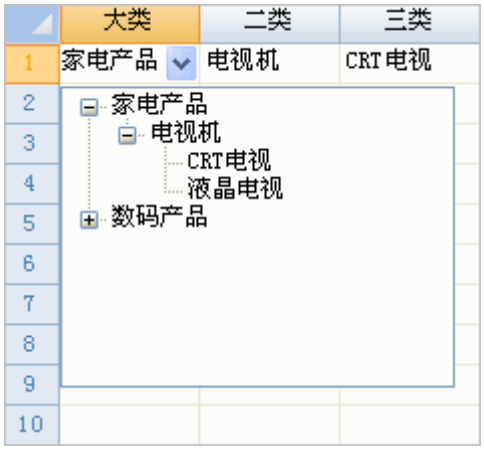
有的时候，目录树表的所有分类内容都在同一列中，如下图：

	分类
1	数码产品|电脑|笔记本
2	数码产品|电脑|台式机
3	数码产品|电脑|服务器
4	数码产品|相机|DC
5	数码产品|相机|DV
6	家电产品|电视机|液晶电视
7	家电产品|电视机|CRT电视
8	数码产品|电脑配件|机箱
9	数码产品|电脑配件|主板

如果需要根据这一列的内容，来生成一个多层的目录树，可以按下图所示设置:



这是录入效果：



3.6.2.4.4 忽略某级节点

前面提到，设置数据接收列的时候，不同的列之间用符号“|”隔开，例如：大类|二类|三类。
有的时候，不是所有层级的节点都是需要输入到数据表中的，如果要忽略某层级的节点，可以在设置数据接收列的时候，对应不输入该层级的接收列。
例如:
列名称 1||列名称 2
将忽略二级节点
再例如：
||列名称 1
则只有第三级节点才会输入到数据表中。

3.6.2.5 数据字典

3.6.2.5.1 什么是数据字典

所谓数据字典，就是在单元格中不显示其真正的值，而代之以一个更具描述性的值。
例如订单表有一个产品编号列，虽然通过列表项目能够有效解决产品编号的输入问题，但是在订单表中查阅数据的时候，我们看到的是产品编号，很难将其和实际的产品名称联系起来。
你也许会说，订单表中直接增加一个产品名称列不就行了吗？当然也可以，不过这毕竟增加了一列。
利用数据字典功能，我们可以让订单表中的产品编号列显示产品名称，尽管其真正的值是产品编号。

3.6.2.5.2 自定义数据字典

1、选择订单表的产品编号列，打开其列属性设置窗口，将数据字典属性设为“标准”，然后单击“数据字典”页：



2、按下图所示设置数据字典：



经过上述设置，现在产品编号列显示的将是产品名称，同时我们也看到，数据字典也能提供列表项目的功能：

	产品编号	单价	数量
1	牛奶	15.2	20
2	苹果汁	15.2	50
3	牛奶	17	65
4	番茄酱	17	12
5	盐	24	15
6	麻油	15.2	35
7	酱油	14.4	45
8	海鲜	8	30
9	麻油	17	20
10	苹果汁	14.4	18
11	番茄酱	15.2	40

为了证明产品编号列虽然显示的是产品名称，但其实际的值是产品编号，我们可以增加一个表达式列，将其表达式设为：

[产品编号]

你可以看到表达式列的内容确实是产品编号，而不是你在产品编号列所看到的产品名称，这证明数据字典确实在发挥作用。

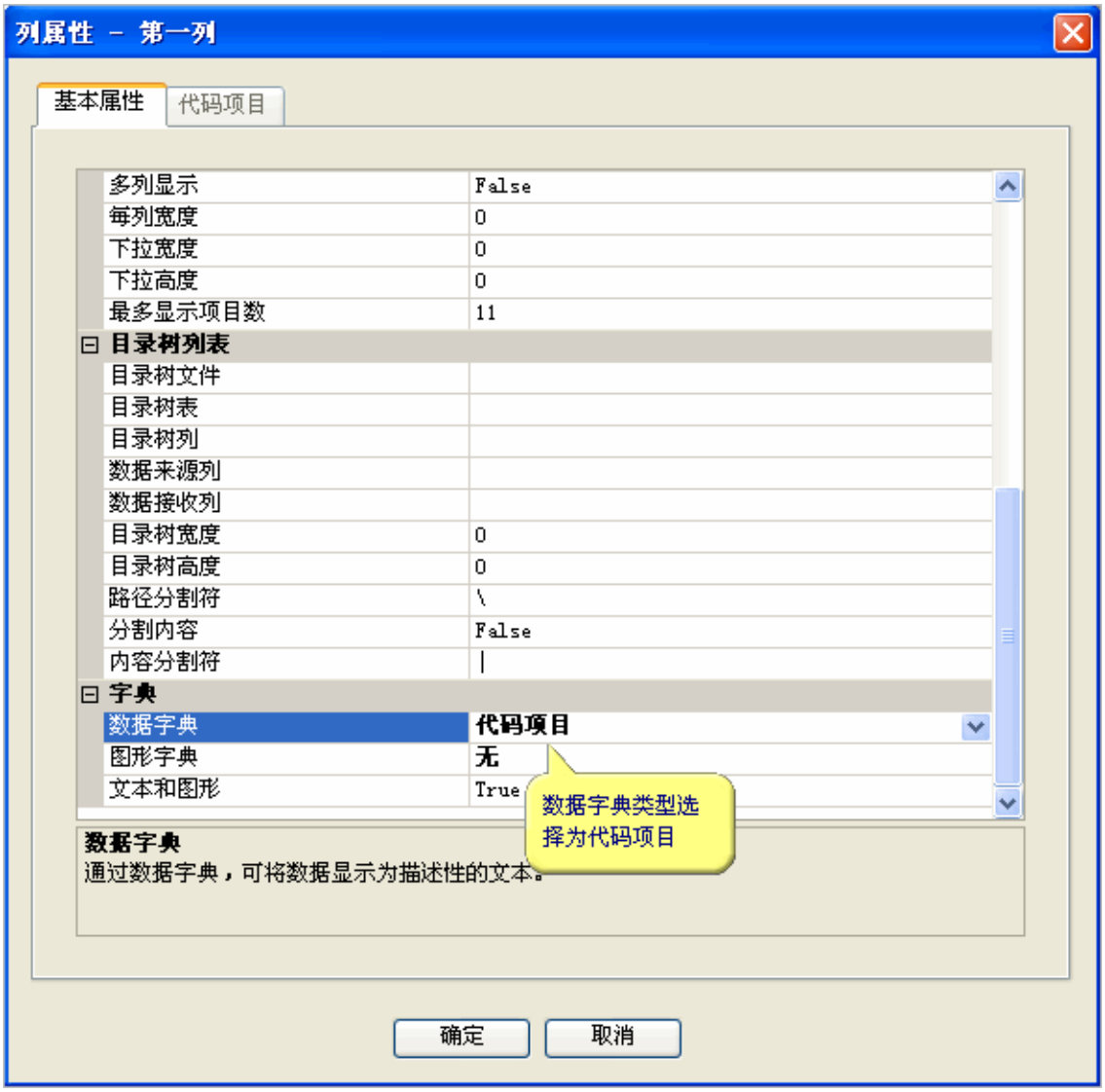
可以说，数据字典是下拉列表的扩展，它不仅具备下拉列表的功能，而且具备类似字典的“翻译”功能。

3.6.2.5.3 使用代码项目

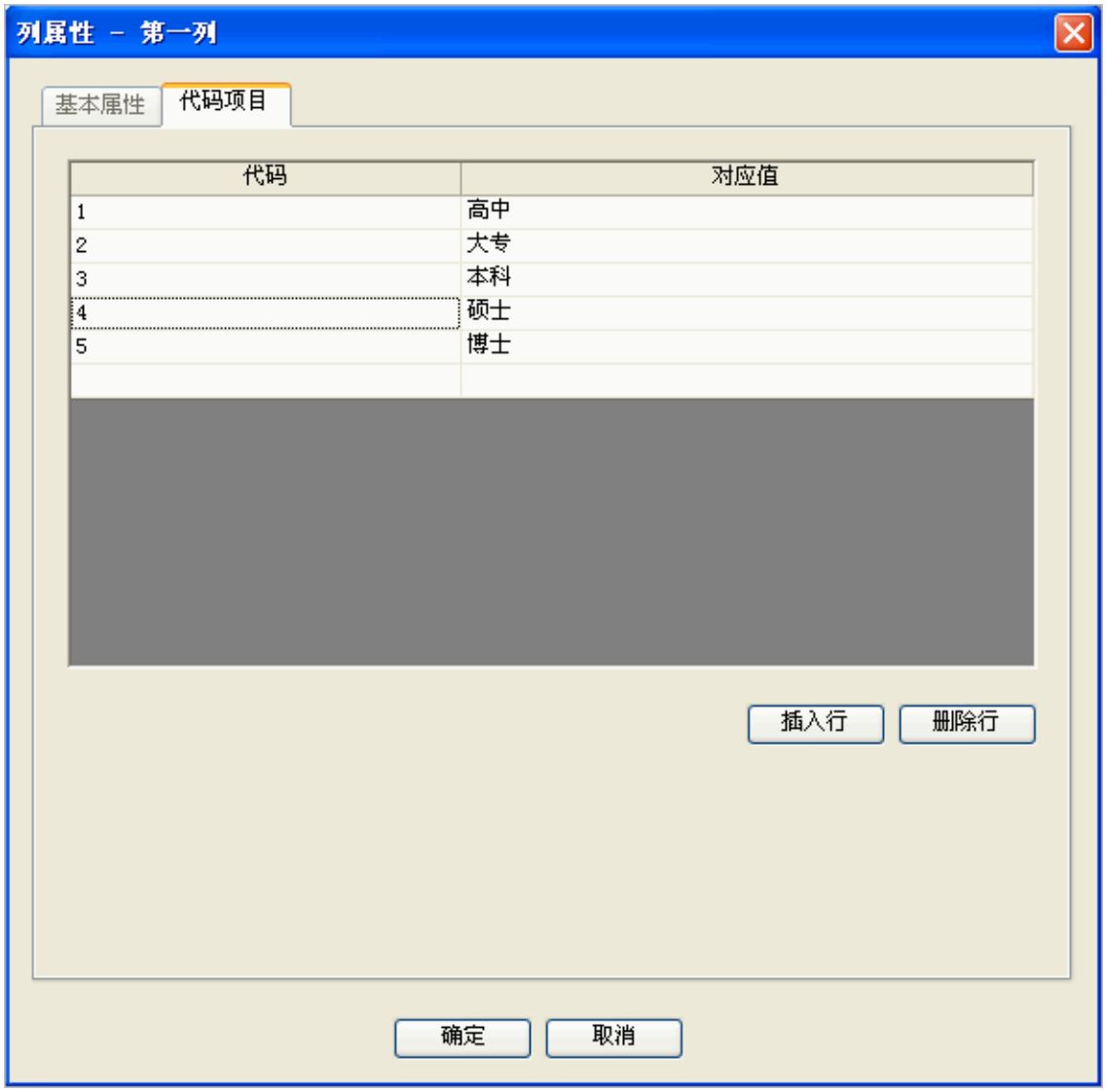
严格意义来说，代码项目并不属于数据字典。

所谓代码项目就是用一些简单的代码来表示某些项目，而用户在录入的时候，只需录入代码， 系统会自动将该代码转换为对应的项目。

1、选择要设置代码项目的列，打开其列属性设置窗口，将数据字典属性设为“代码项目”，然后单击“代码项目”页：

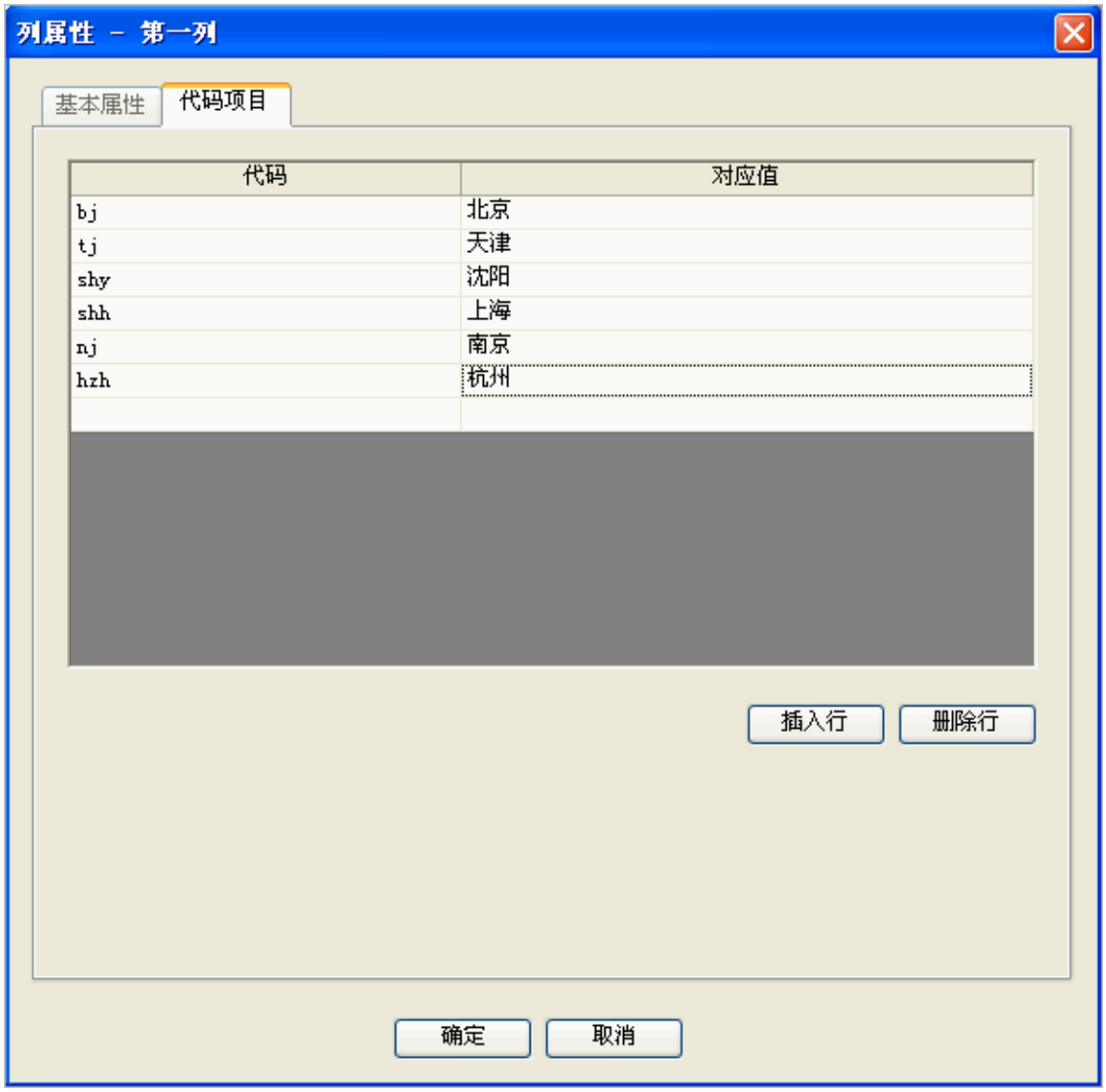


2、按下图所示设置代码项目：



经过上述设置后，我们在此列输入“1”，会被自动替换为“高中”，输入“2”，会被替换为“大专”，其余类推。

数字过于抽象，不便于记忆，实际应用的时候，代码最好使用拼音缩写，例如下面的代码使用拼音缩写来代表城市：



这样我们输入“bj”，即可自动替换为“北京”，输入“shh”，即可自动替换为“上海”，这比使用数字作为代码更加方便。

3.6.2.5.4 用数据表做字典

前面提到通过自定义数据字典，在产品编号列不显示产品编号，而是用对应的产品名称代替。

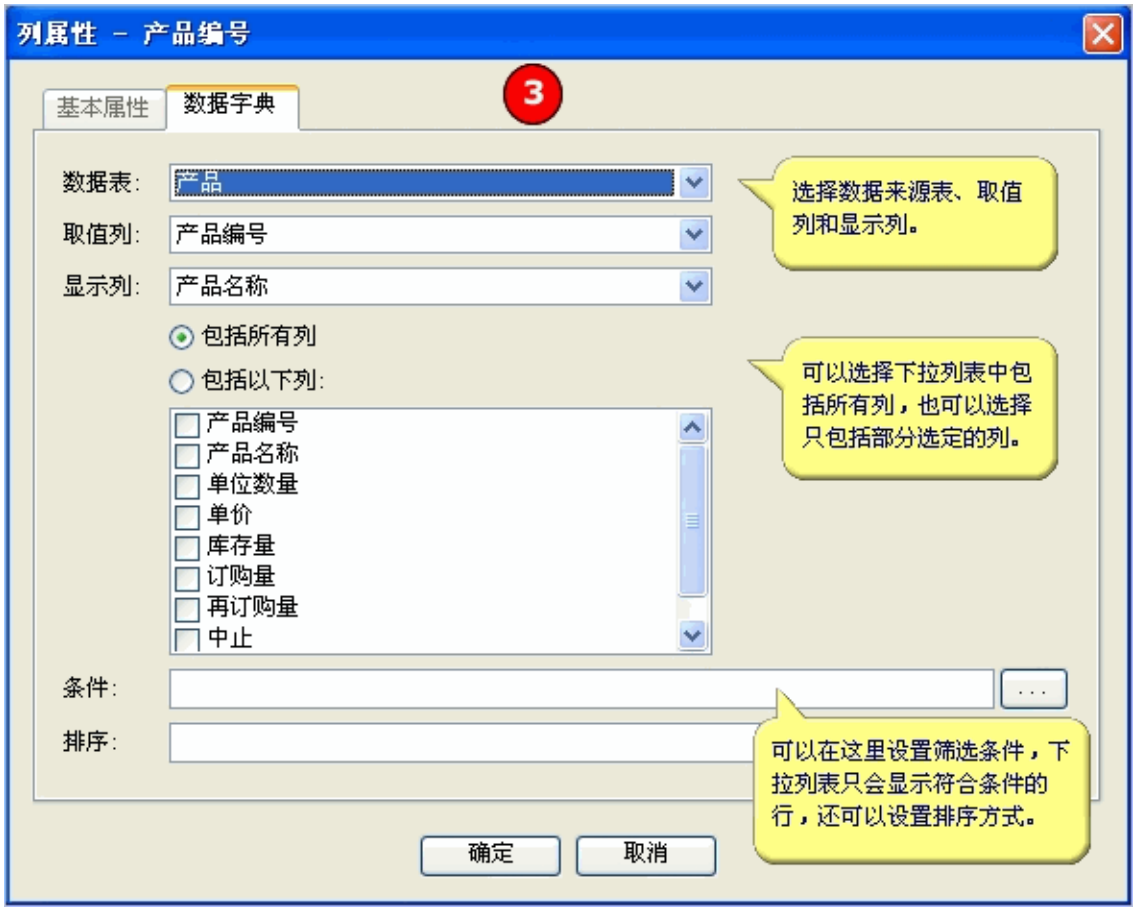
但是这个解决办法还是不够好，因为产品编号和产品名称已经在产品表中输入，能不能直接调用产品表中的呢，当然可以。

现在我们来看看如何利用产品表作为数据字典，来实现同样的功能。

1、选择订单表的产品编号列，打开其列属性设置窗口，将数据字典属性设为“数据表”，然后单击“数据字典”页：



2、按下图所示设置数据字典：



经过上述设置，订单表中的产品编号列，显示的将不再是产品编号，而是该产品编号对应的产品名称；而输入数据的时候，下拉列表是一个表格：



3.6.2.5.5 条件与排序

前面已经看到，使用数据表作为字典的时候，可以设置筛选条件和排序字段。

1、条件

例如你希望数据字典只使用库存量大于 0 的产品，那么可以将筛选条件设为：

[库存量] > 0

但是千万要注意，如果订单表中已经输入过库存量等于 0 的产品，那么产品编号列将显示为空白，原因很简单，因为程序在所设置的数据字典中，找不到该产品编号对应的产品名称；所以为数据字典设置条件，要慎重。

2、排序

例如你希望下拉列表能够按照单价顺序排列，那么可以将排序字段设为：

单价

默认的排序方式为升序，也就是单价低的排在前面，如果你希望降序排序，单价高的排在前面，可以在排序字段后面加上关键词"DESC"，例如：

单价 DESC

如果要同时根据多列排序，不同排序字段之间用逗号分开即可。

3.6.2.5.6 自定义多列数据字典

用数据表作为数据字典，下拉列表可以有多列。

但是有的时候，数据字典的内容并非来自于现成的数据表，只能使用自定义的数据字典。

自定义的数据字典能不能够像数据表字典一样，可以有多列呢？当然可以。

你只需在选择数据字典类型的时候，选择为“多列”即可：



3.6.2.6 图形字典

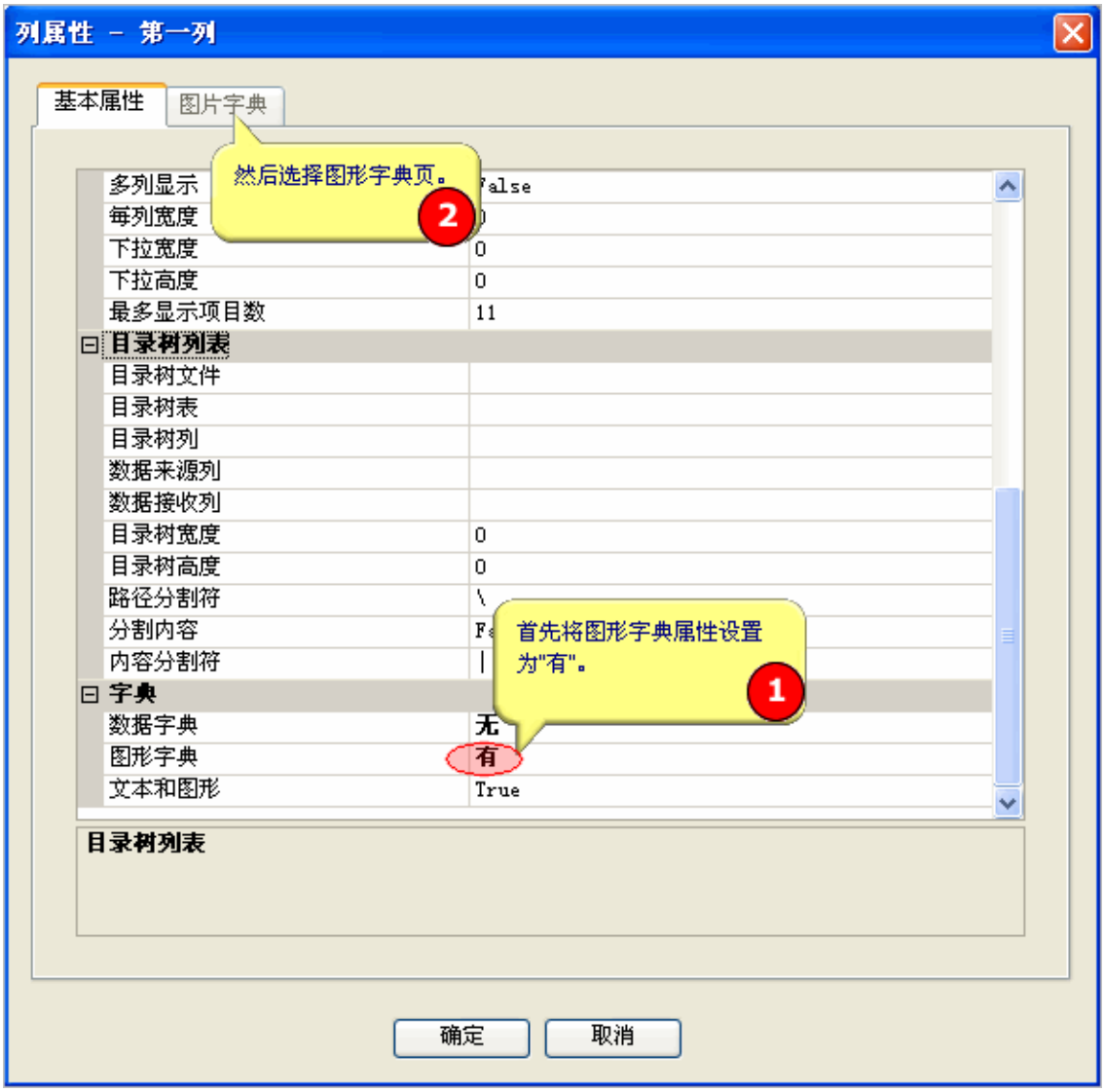
3.6.2.6.1 什么是图形字典

前面已经看到，数据字典能用较为容易理解的字符表示抽象的数据。
而图形字典的作用类似，不过不是用字符，而是用更为直观的图形来表示抽象的数据。
例如下图的国家列，不仅显示了国名，而且显示了每个国家的国旗，这样看起来更为直观，这就是用图形字典实现的：

	国家	地址
1	德国	
2	法国	
3	澳洲	
4	巴西	
5	丹麦	
6	加拿大	

3.6.2.6.2 设置图形字典

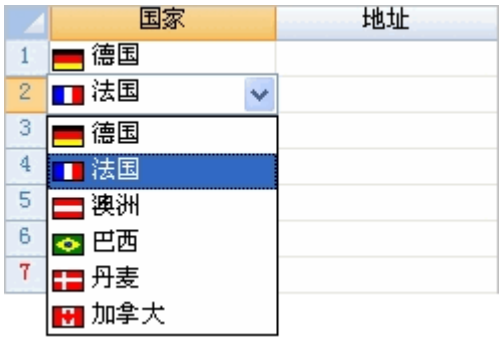
1、选择国家列，打开其列属性设置窗口，将图形字典属性设为“有”，然后单击“图形字典”页：



2、按下图所示，设置图形字典：



经过上述设置，在国家列中，会同时显示国家名称和所对应的国旗，而且输入的时候，同样有一个下拉列表供选择：



3.6.2.6.3 只显示图形

在默认情况下，设置了图形字典的列，会同时显示图形和文本。
如果你希望只显示图形，可以在设置图形字典的时候，将“文本与图形”属性设为 False(否)即可。

3.6.2.6.4 图形文件在哪里？

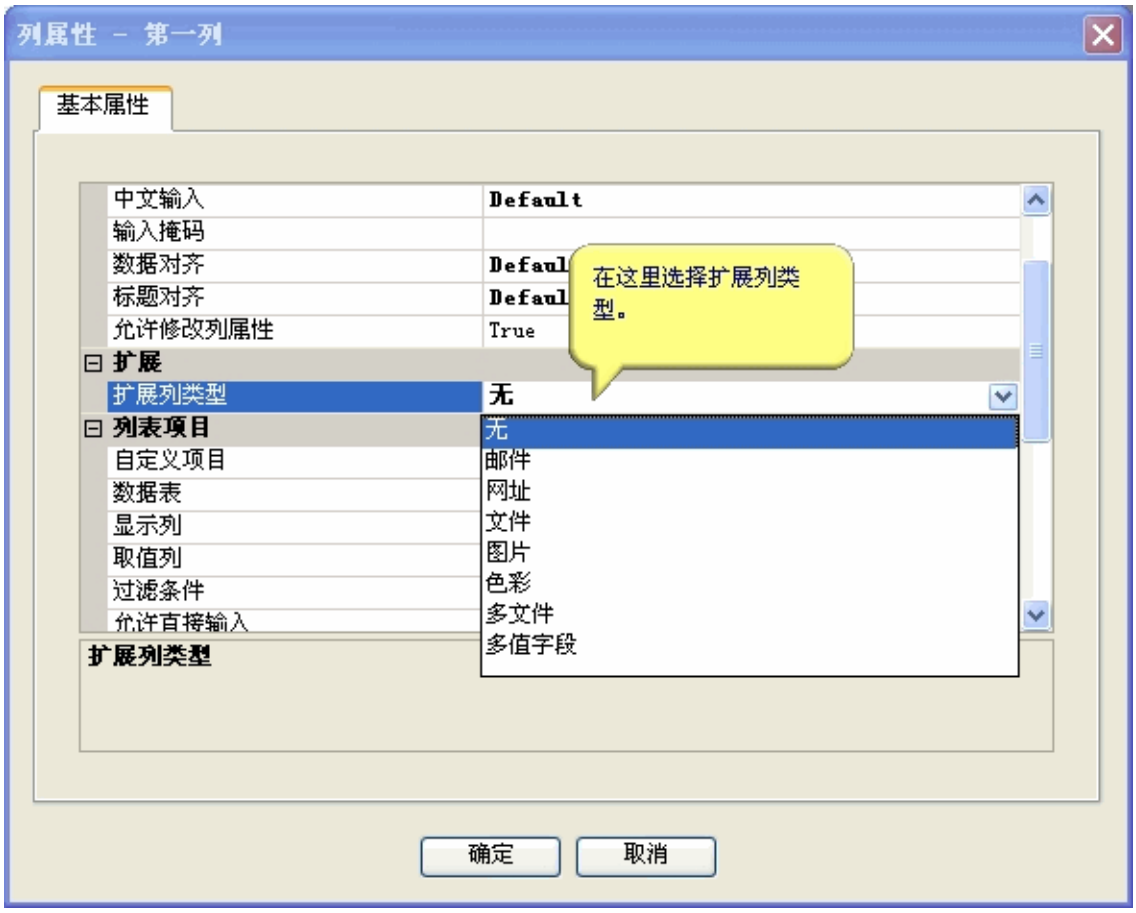
图形字典使用了很多图形文件，不管这些文件原来在哪里，一旦为图形字典所采用，将被自动复制到项目文件所在目录的子目录 Images 下，图形字典使用的是这些新复制的文件，和原来的文件

不再有任何联系。
这样处理会带来很大的便利，你可以很方便地将管理项目移到任何电脑的任何目录，要做的只是复制整个项目文件目录到目标电脑而已。

3.6.3 扩展列类型

3.6.3.1 关于扩展列类型

Foxtable 提供了很多列类型，但是还是不够用；例如有的列用于输入邮件、网址，有的列用于管理图片和文件，还有的列用于管理颜色，这些列需要一些独有的特性才能符合用户的实际需求，例如某一列输入了邮件地址，用户可能会希望单击某一处能够直接向该邮件地址发送邮件。
扩展列类型就是为了满足这些特殊的应用而提供，接下来我们会逐个介绍每一种扩展列类型。
列属性中，有一个“扩展列类型”属性，用于选择扩展列类型，默认值是“无”：



3.6.3.2 邮件型

字符型列可以扩展为邮件型，邮件型列可以正常输入数据，不同的是，单元格右边有一个按钮；如果单元格中已经输入邮件地址，单击该按钮，会调用邮件客户端软件(如 OutLook、FoxMail)向该邮件地址发送邮件。



有人问为什么邮件类型不做成 Excel 那样的，其实 Foxtable 的处理方式更好，发送邮件的时候单击旁边的按钮，单击单元格只是选定该单元格而已，不至于误操作。

3.6.3.3 网址型

网址型和邮件型完全一样，只是内容为网址。

3.6.3.4 QQ 型

字符列可以扩展为 QQ 型列，用于存储 QQ 号码，如果某个单元格已经输入 QQ 号码，那么单击单元格右侧的按钮，会自动启动 QQ 对话窗口，与这个 QQ 号码对话。

3.6.3.5 旺旺型

字符列可以扩展为旺旺型列，用于存储旺旺账号，如果某个单元格已经输入旺旺账号，那么单击单元格右侧的按钮，会自动启动旺旺对话窗口，与这个旺旺账号对话。

3.6.3.6 色彩型

只有整数型列才可以扩展为色彩型，也就是说，要将某列扩展为色彩列，必须先将其数据类型设为整数型。

色彩型单元格显示的是颜色，你不能直接输入数据，只能单击右边的单元格按钮来选择颜色：



获得色彩单元格的值：

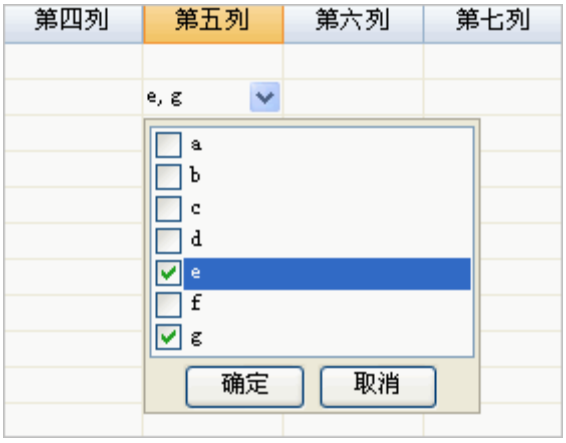
由于色彩型列只显示颜色，而不显示值，如果要查看单元格具体的值，可以选择该单元格，单击工具栏中的复制按钮，然后粘贴到任何地方，即可看到单元格具体的值。你也可以增加一个表达式列，将该列的表达式设为：

[颜色]

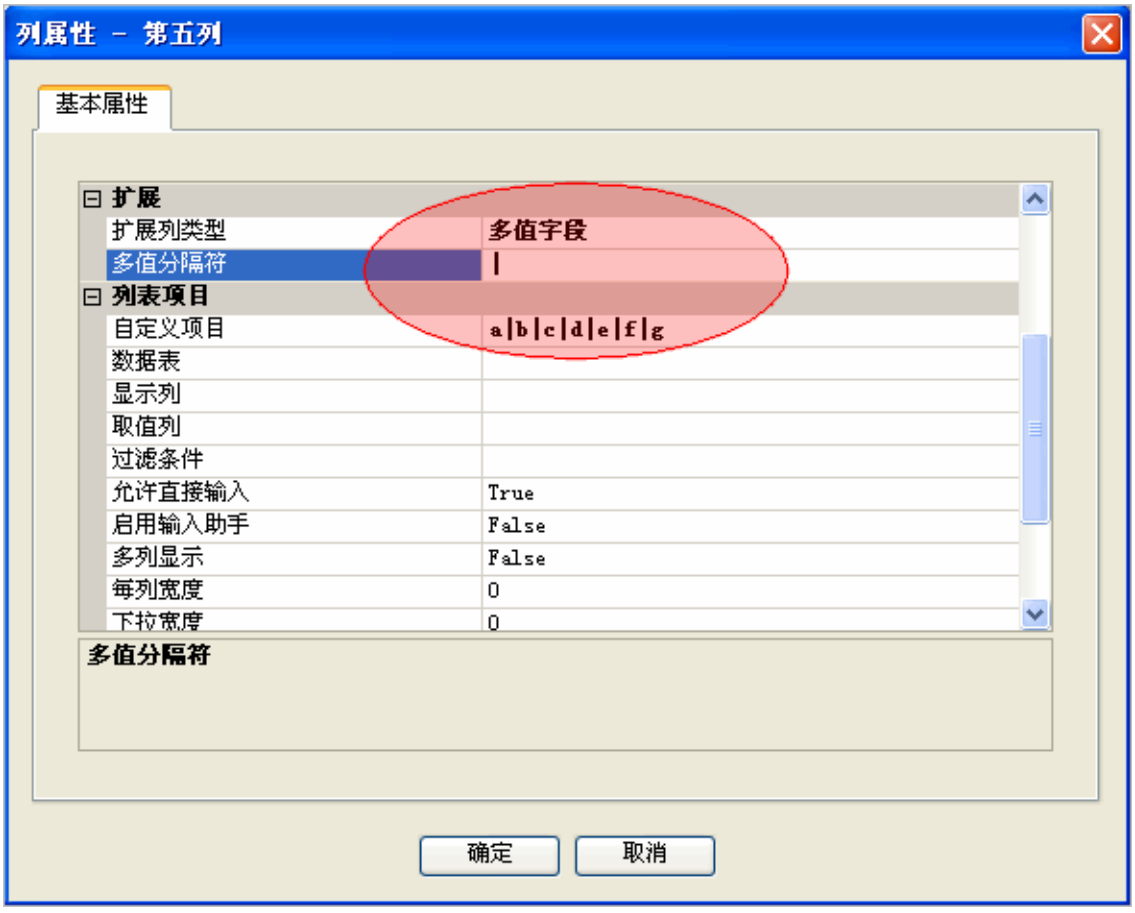
这样查看起来更方便。

3.6.3.7 多值字段

扩展列类型设为多值字段，列表项目即可多选，如图所示：



默认的值分隔符为逗号，可以在列属性设置中将其改为其它符号：



3.6.3.8 时段型

双精度小数型的列，可以扩展为时段型。

时段型列用于输入和计算时段，时段列虽然以“时:分:秒 毫秒”格式显示数据，但是其值本质上是

一个双精度小数，以秒为单位。
例如“12:30:45”，表示 12 个小时 30 分钟 45 秒，其值为：12 * 3600 + 30 * 60 + 45，也就是 45045。
下面的表，"时段"列是一个双精度小数列，扩展列类型为时段，“值”列是一个双精度小数型的表达式列，表达式为“[时段]”:

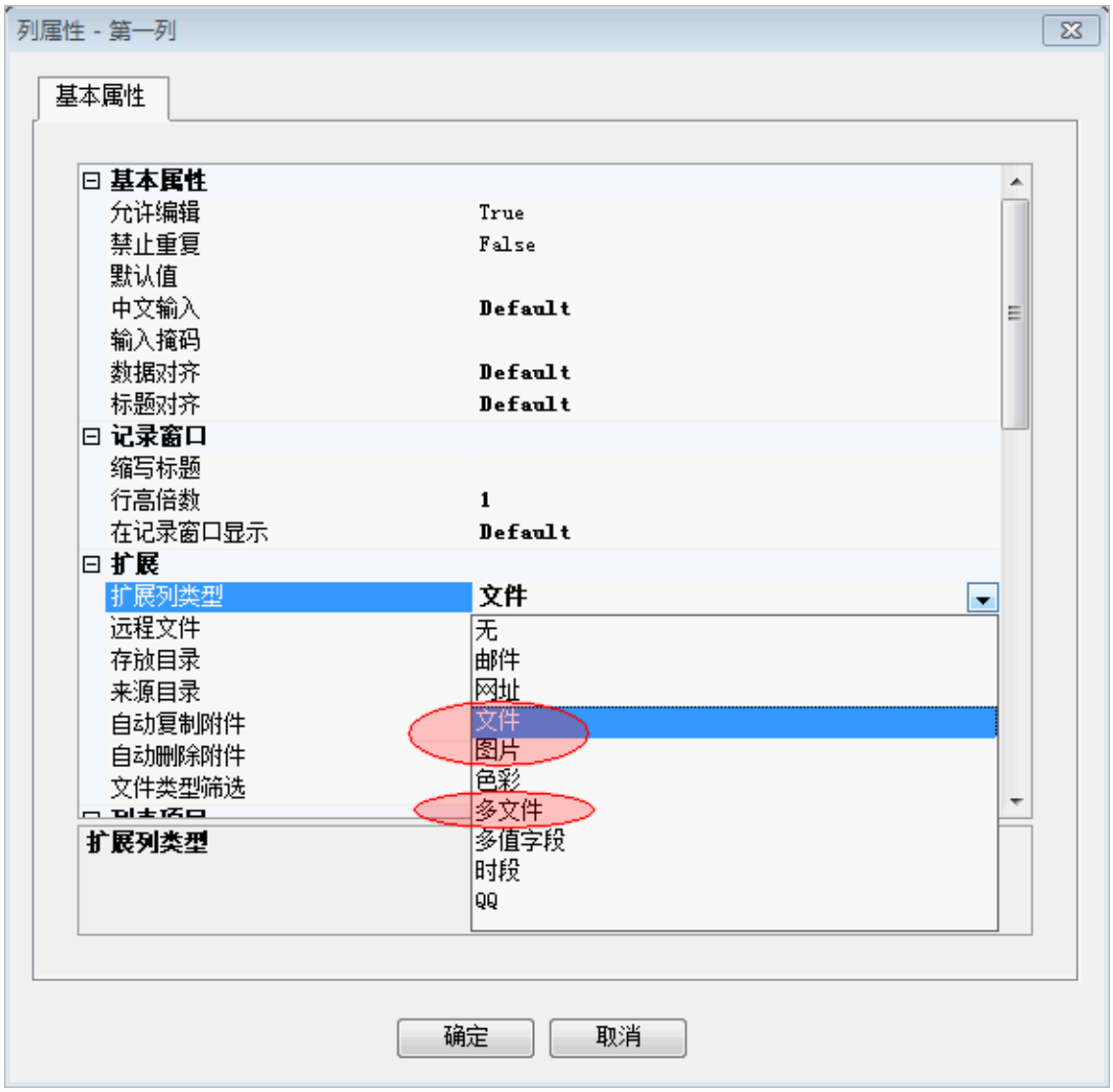
	时段	值	说明
1	12	43200	12小时
2	12:30	45000	12小时30分钟
3	12:30:45	45045	12小时30分钟45秒
4	12:30:45 500	45045.5	12小时30分钟45秒零500毫秒
5	00:30	1800	30分钟
6	00:30:45	1845	30分钟45秒
7	00:00:45	45	45秒
8	00:00:00 800	0.8	800毫秒
9	198	712800	198小时
10	198:45:00 800	715500.8	198小时45分零800毫秒

3.6.3.9 文件与图片的管理

3.6.3.9.1 关于文件与图片管理

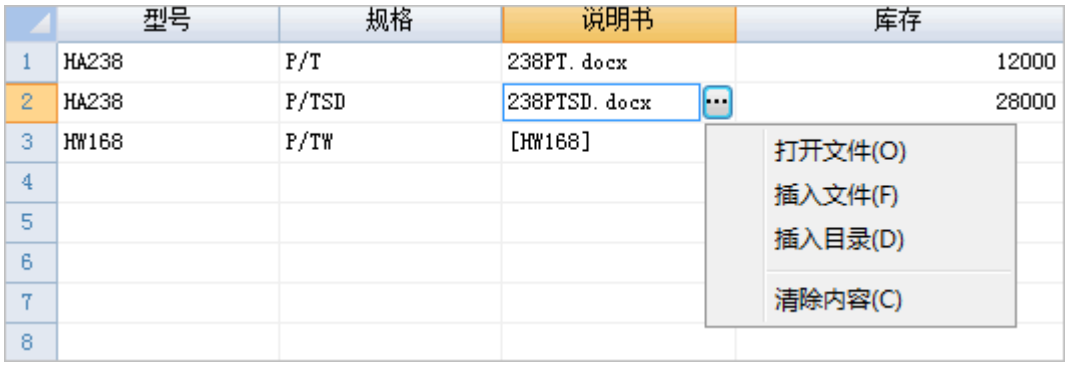
一个数据管理系统，通常不单单是管理数据，图片和文件的管理也是非常重要的。

通过扩展列类型，Foxtable 可以很方便地管理图片和文件，而且和数据管理一样，图片和文件的管理也可以基于局域网或互联网来完成。
Foxtable 提供了三种扩展列类型，分别是文件、图片、多文件，用于管理文件与图片：



3.6.3.9.2 文件型

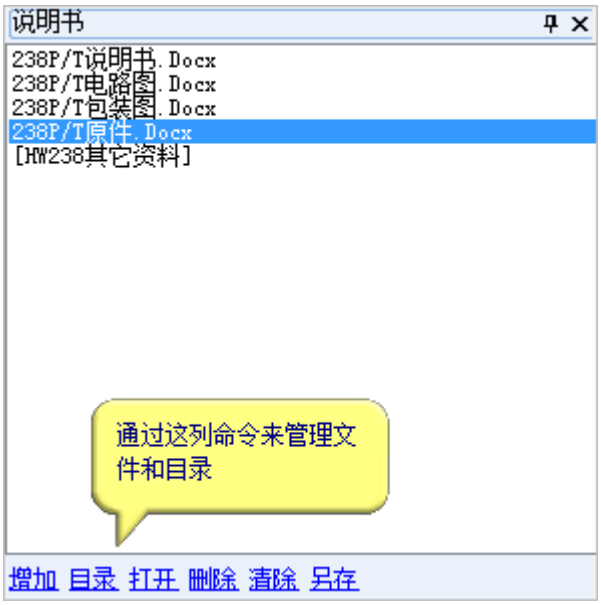
文件型列用于管理单个文件或单个目录，只有字符型列才能扩展为文件型。
文件型列不可以直接输入内容，单击单元格右边的按钮，会出现一个菜单，用于打开文件、插入文件、插入目录，以及清除单元格内容：



从上图可以看出，文件列不仅可以插入文件，而且可以插入目录，插入的目录会自动用方括号括起来，例如：[HW168]。

3.6.3.9.3 多文件型

文件型列只能管理一个文件或一个目录，如果一行数据有可能涉及到多个文件或多个目录，那么请将列扩展为多文件型。
只有字符型或者备注型列才能扩展为多文件型，如果使用字符列，请注意合理设置字符列的长度。
对于多文件型列，有一个专门的列窗口用于管理文件：



上图包括四个文件和一个目录，插入的目录会自动用方括号括起来，例如：[HW238 其它资料]。
文件管理器窗口各命令说明：

- 增加
  向文件管理器增加一个文件。

- 目录
  向文件管理器增加一个目录。

- 打开
  打开选定的文件或者目录，当然也可以双击文件或者目录来打开之。

- 删除
  删除选定的文件或者目录。

- 清除
  清除所有文件或者目录。

- 另存
  另存选定的文件。

3.6.3.9.4 图片型

图片型列用于管理图片，可以同时管理多个图片，非常方便。
通常使用备注列或者字符列来扩展为图片型，如果使用字符列，请注意合理设置字符列的长度。
下面是图片型列的管理窗口：



你可以在图片管理窗口，增加、删除、打开图片，还可以切换到全屏模式查看。
图片管理器窗口各命令说明：

- 增加
  在图片管理器窗口插入一个图片。

- 打开
  调用默认的程序打开选定的图片。

- 删除
  删除选定的图片。

- 全屏
  以全屏方式查看图片。

- 缩放
  单击这个命令，将以原尺寸显示图片，再次单击，将以缩放形式显示图片。

- 另存
  另存选定的图片。

3.6.3.9.5 文件在哪里？

默认情况下，每当我们在文件型、多文件型、以及图片型列中插入一个文件时，程序会自动将该文件复制到项目文件所在目录的子目录 Attachments 下，之后我们使用的都是这些新复制的文件，和原来的文件不再有任何联系。

这样的处理方式，为我们带来很大的便利，因为插入的文件成为了项目的一个附件，我们可以随时将项目复制到任何电脑、任何目录运行，一点也不会影响运行结果，要做的只是将整个项目目录复制到目标位置而已。

当我们从这些列中删除文件时，会出现一个窗口，提示是否同时删除本地附件，也就是 Attachments 目录下的文件。

有两个列属性用于改变这种默认的处理方式：

自动复制附件

将此属性设为 False(否)，不会将插入的文件复制到 Attachments 子目录下。

自动删除附件

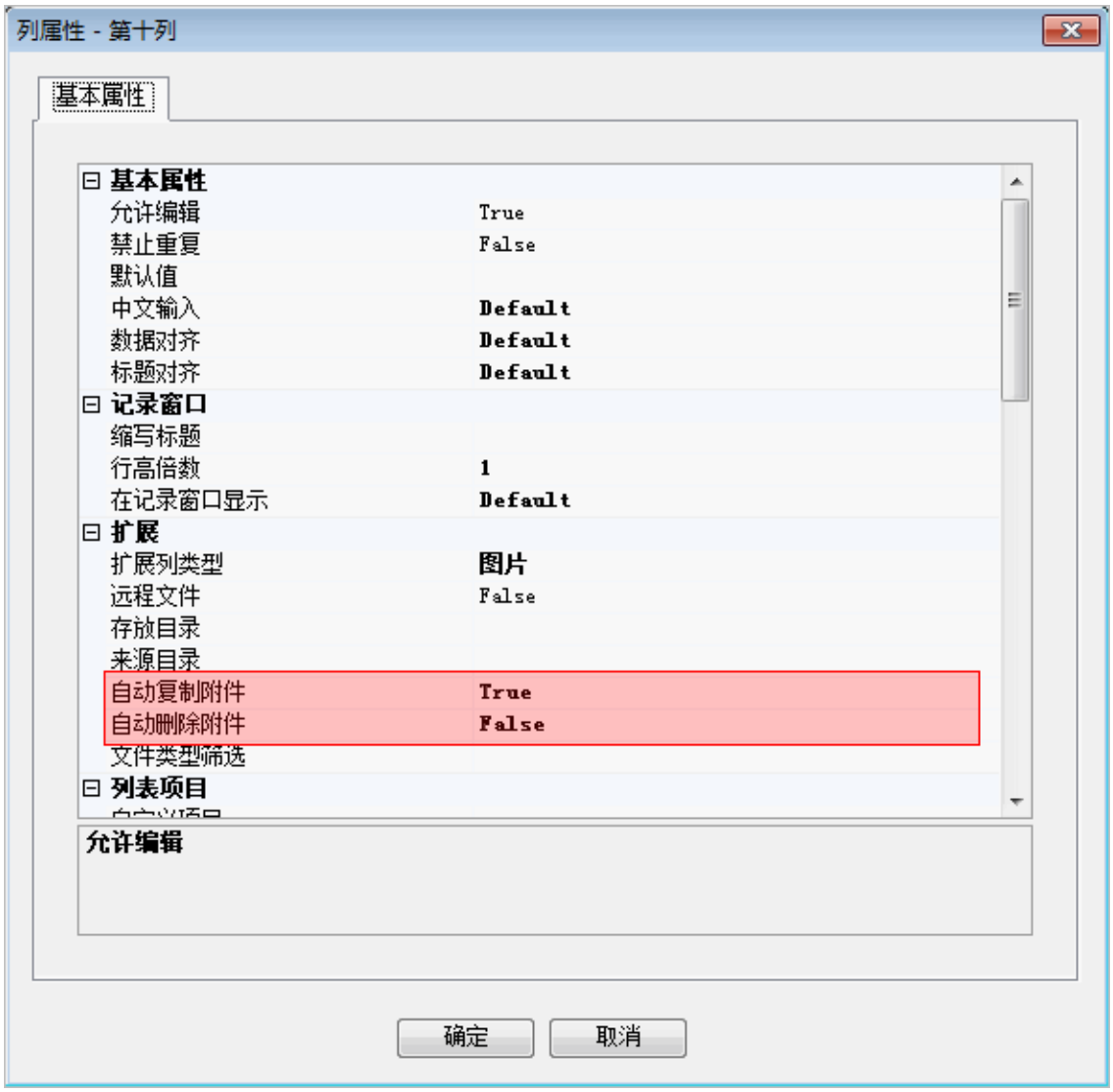
此属性有三个可选值：

Default：出现确认窗口，由用户选择是否删除本地附件。

True： 删除本地附件。

False： 保留本地附件

提示：只有扩展类型为文件、多文件、图片时，列属性设置窗口才会出现这两个属性:



3.6.3.9.6 文件类型筛选

为更方便地管理文件，对于扩展类型为图片、文件或者多文件的列，在列属性窗口有一个独有的属性“文件类型筛选”，通过设置该属性，在插入文件的时候，可以只列出指定扩展名的文件。

文件筛选类型属性分为两个部分，前面是文件类型说明，后面是筛选模式（*.文件扩展名），中间用符号“|”隔开。

例如只插入文本文件，可设为：

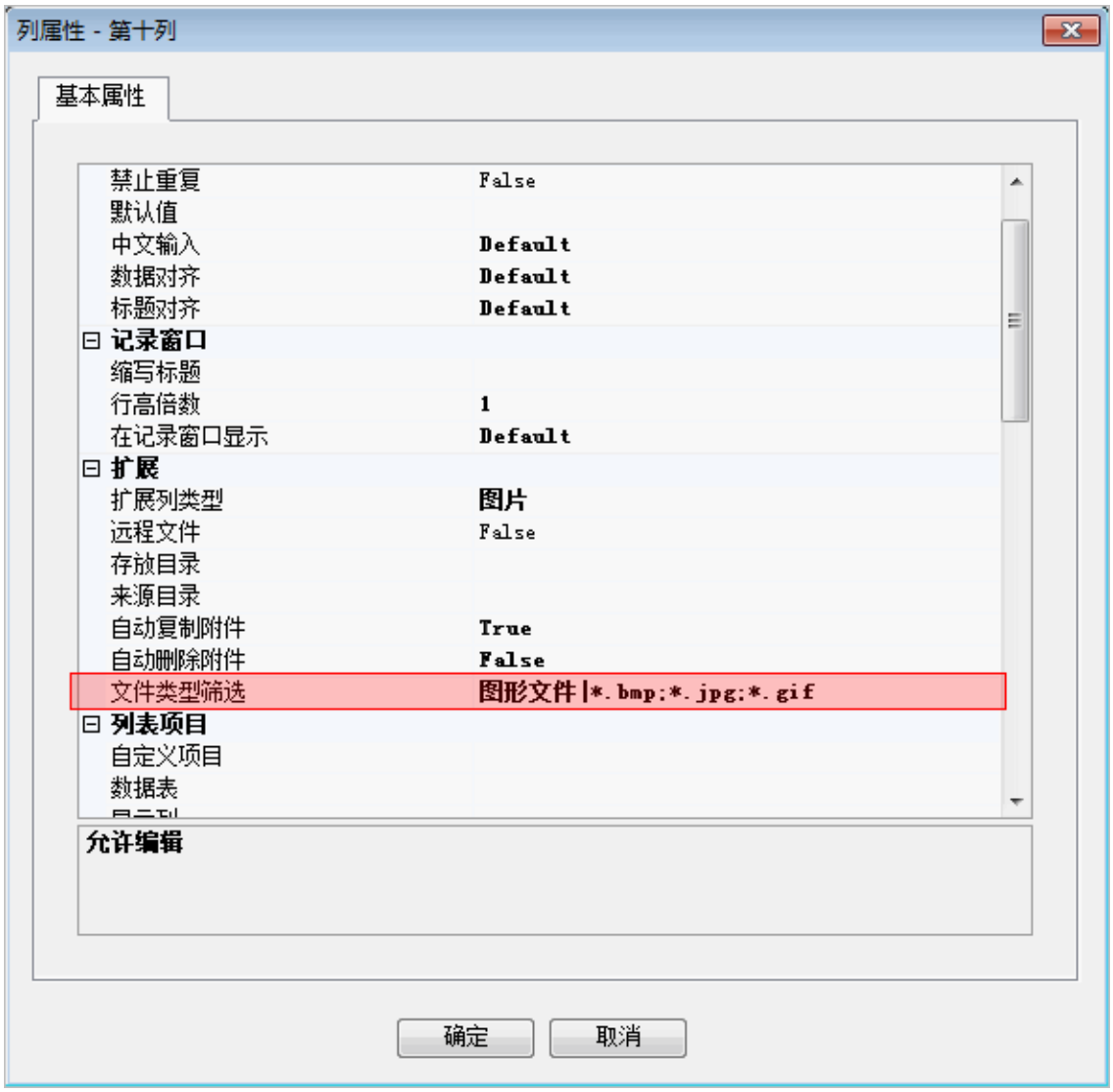
文本文件|*.txt

如果存在多个扩展名，不同的扩展名之间用分号(;)隔开，例如只插入常用图片文件，可设为：

图形文件|*.bmp;*.jpg;*.gif

可以同时设置多个过滤器，用符号“|”隔开即可，例如：

Excel 文件|*.xls|Word 文件|*.doc|Access 文件|*.mdb

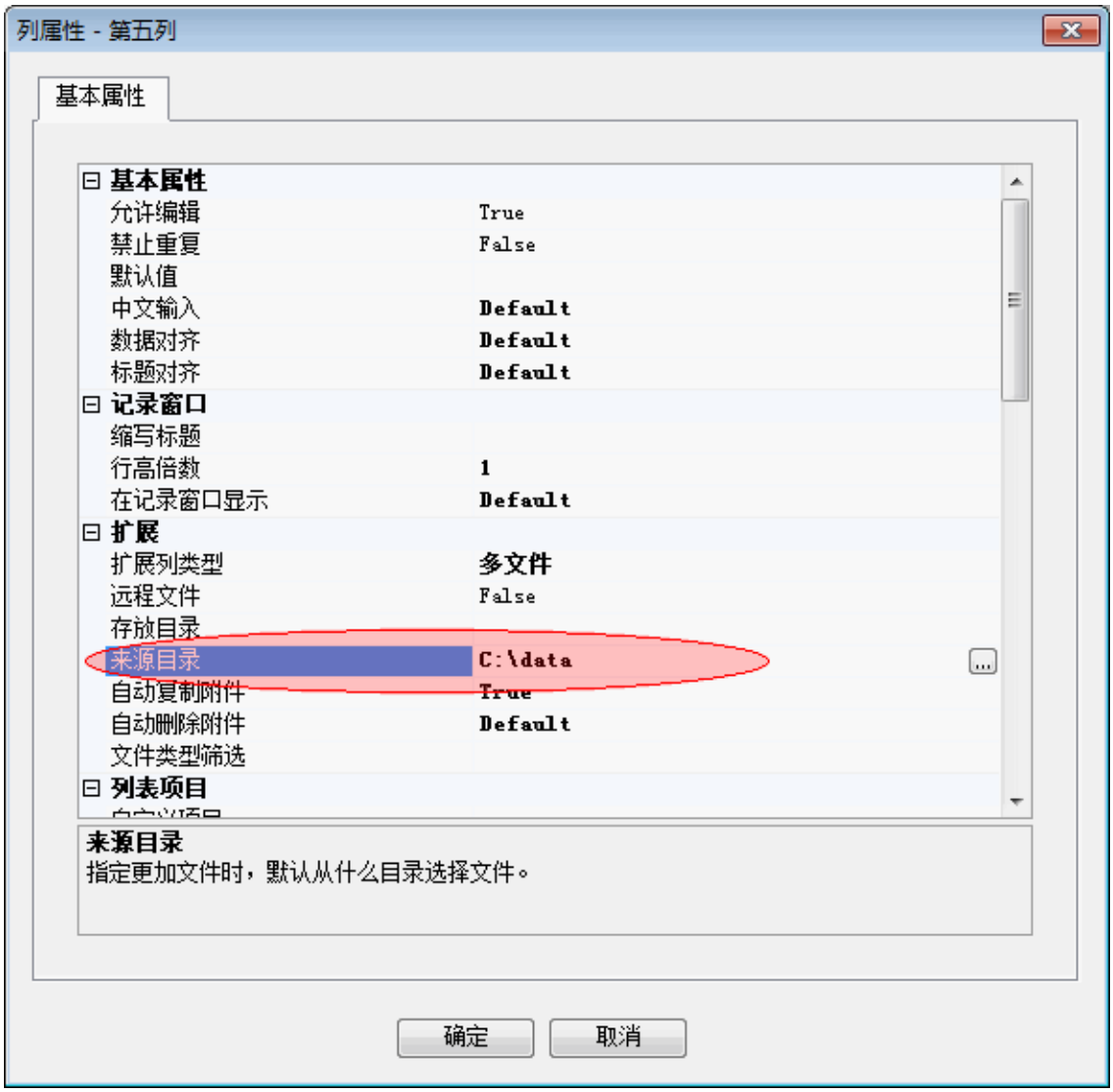


3.6.3.9.7 隐藏原来的列

由于备注型、图片型、多文件型列，都有专用的列窗口，所以我们建议你设置好这些列的属性后，在表中将这些列隐藏起来，因为此时显示这些列已经没有意义，而且占据有限的屏幕空间。
如果希望重新设置这些列的属性，取消隐藏列即可。

3.6.3.9.8 指定文件来源位置

如果每次都需要从固定的目录插入文件到某列，那么可以在列属性指定来源目录，这样每次插入文件，都会自动打开这个目录，无需再去手工选择此目录。



3.6.3.9.9 局域网中的文件管理

前面提到在默认情况下，每当我们在文件型、多文件型、以及图片型列中插入一个文件时，程序会自动将该文件复制到项目文件所在目录的子目录 Attachments 下。

有的时候，我们需要改变默认的存放位置，为此 Foxtable 提供了一个列属性“存放目录”，可以用于指定文件的存放目录。

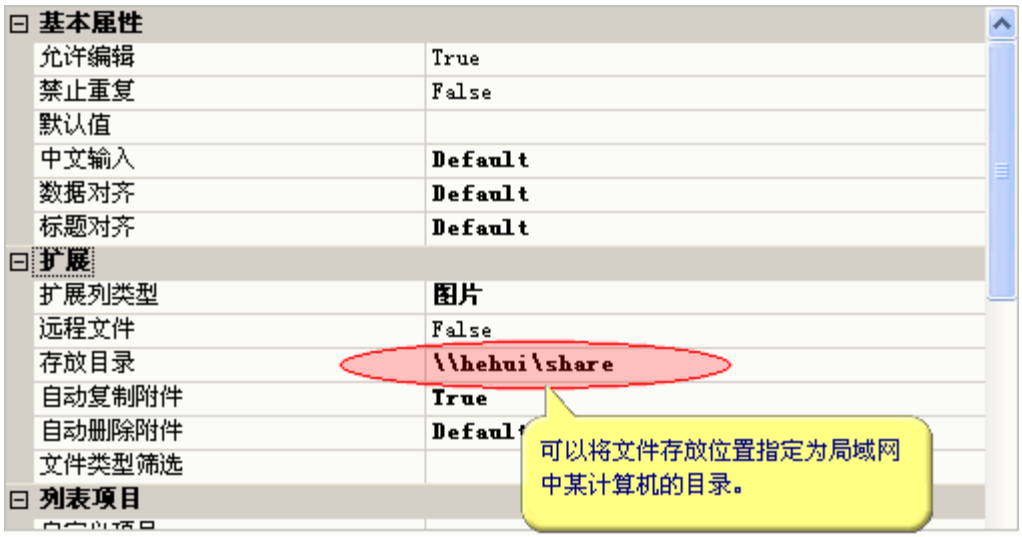
存放目录可以是绝对路径，例如可以将某列的存放目录指定为"c:\data\photo",以后在该列添加文件时，被添加的文件将直接复制到该目录中

存放目录也可以是相对路径，例如可以将某列的存放目录指定为"photo"，那么程序会在子目录 Attachments 下创建一个下级子目录“photo”，用于存放该列新添加的文件。

这样我们可以为不同的目录指定不同的存放位置。

不过 在局域网中，需要多用户协同工作的时候，上面的处理方式会带来很大的麻烦，因为每个用户添加的文件都处在各自的计算机中，无法实现文件的协同管理和使用。

要解决这个问题很简单，你可以将存放目录指定为一个网络路径，例如下图中，将文件存放位置指定为“\\hehui\share”:



这样不同用户添加的文件，均处在同一个位置，可以协同管理和使用。
重要提示：

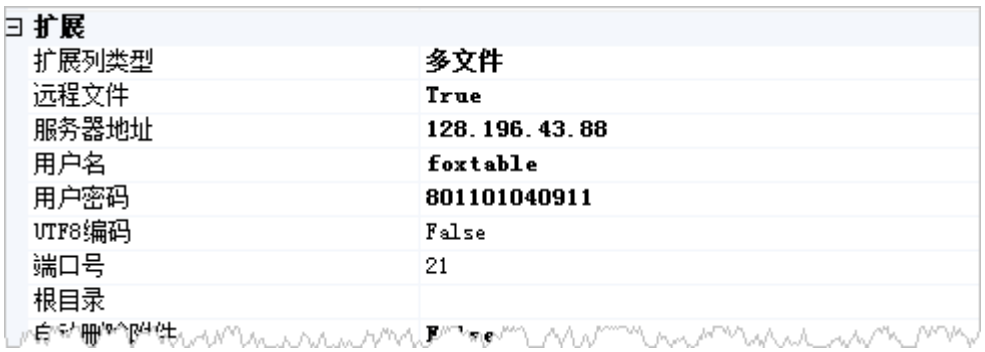
如果你希望对文件和图片管理功能进行权限设置，那么请不要使用这种共享文件夹的方式管理文件和图片，应该采用 FTP 实现，下一节我们会开始介绍如何用 FTP 实现网络环境下的文件和图片管理功能。

3.6.3.9.10 互联网中的文件管理

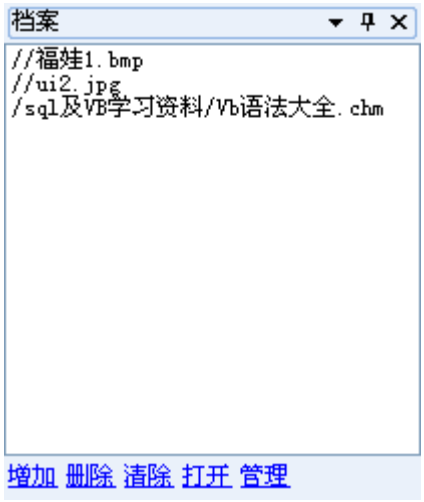
普通用户可以忽略本节内容。

本节内容适用于互联网，也适用于局域网。

Foxtable 可以利用 FTP 服务器，实现远程的文件管理和共享。
如果列的扩展类型设为图片、文件或多文件，且希望使用远程的 FTP 来管理文件，只需将“远程文件”属性设为 True，然后输入 FTP 服务器地址、帐号、密码即可：



一旦设置完成，就可以像使用本地文件一样，使用 FTP 上的文件：



有所不同的是，单击上图的“增加”命令，不再是从本地选择文件，而是会打开一个 FTP 窗口，用于从 FTP 选择文件添加到文件管理器：



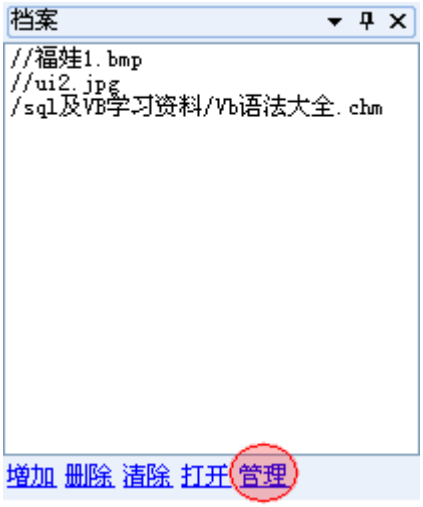
提示：

1、添加某一个文件，直接双击之即可；如果要添加多个文件，先按 Ctrl 或 Shift 键选择多个文件，然后单击确定按钮。

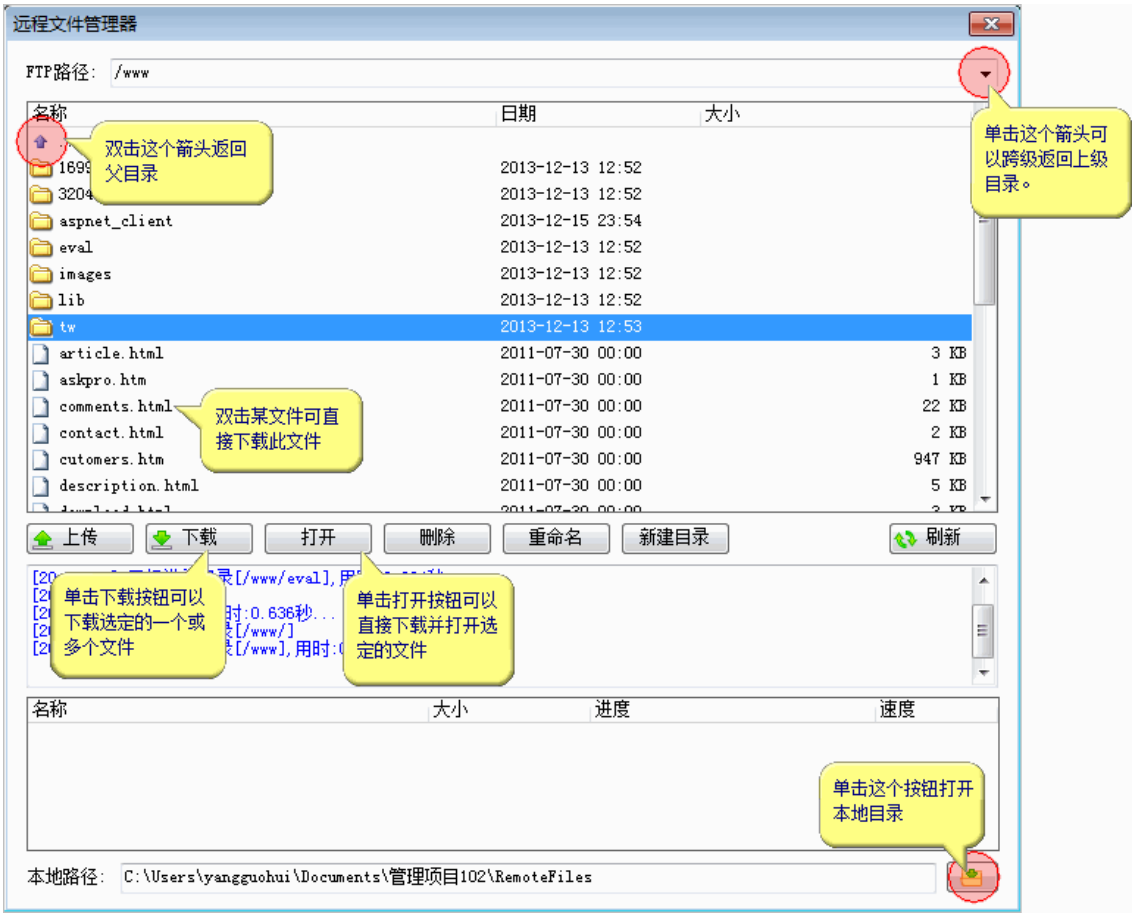
2、如果使用远程文件管理，请保证你的网络是畅通的，否则很容易导致 Foxtable 长时间无响应。

3.6.3.9.11 FTP 客户端使用说明

在使用 FTP 管理文件或图片时，文件或图片管理窗口会有一个管理命令：



单击这个命令，会打开内置的 FTP 客户端(远程文件管理器)，下图是此客户端的使用图示：

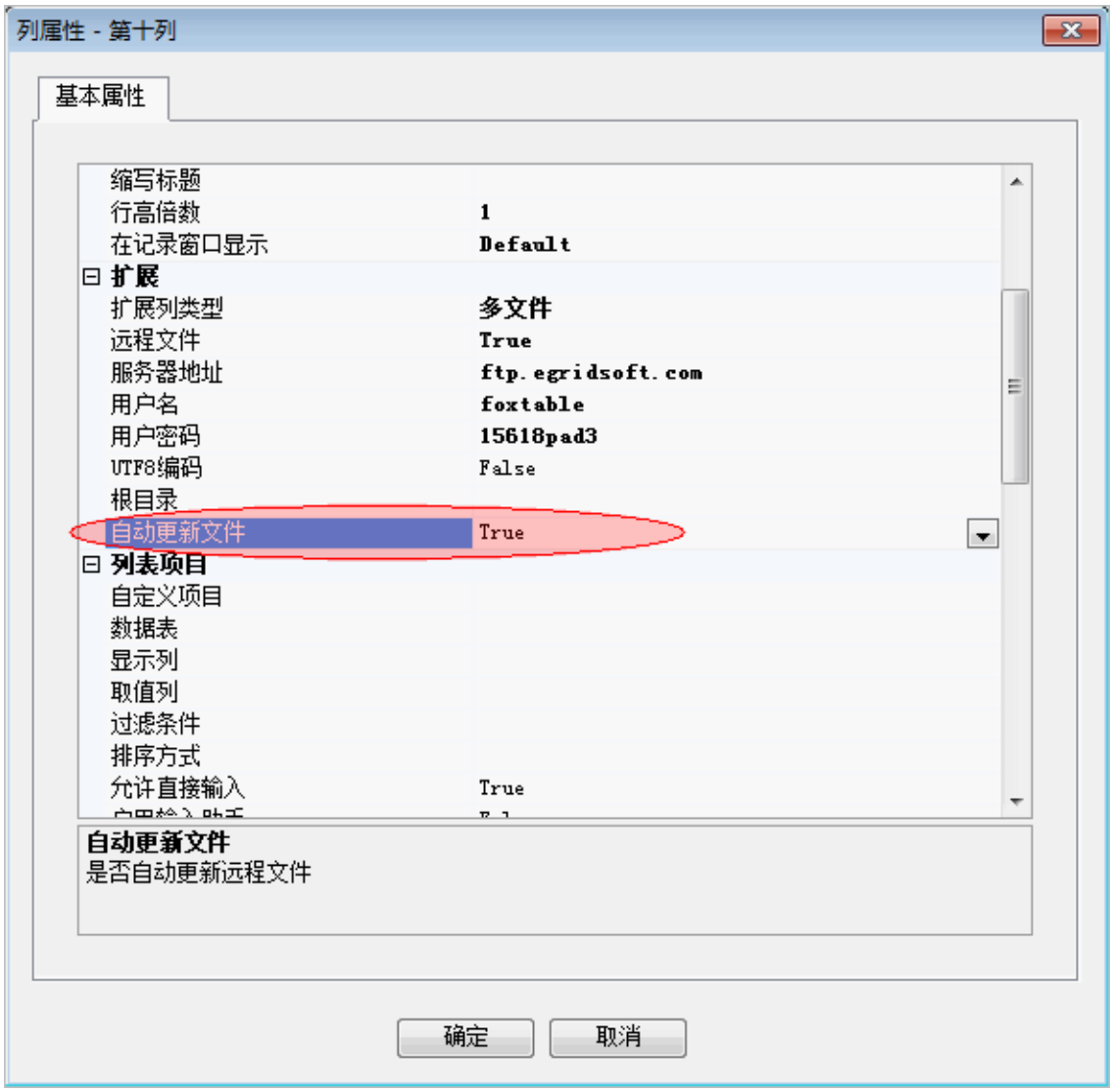


此客户端下载的文件，全部位于项目所在目录的 RemoteFiles 子目录中，你也许会奇怪，为什么不允许用户自己选择本地目录？

这是因为这个客户端是配合 Foxtable 的文件和图片管理器使用的，这两个管理器都使用 RemoteFiles 子目录在本地缓存文件。
如果你要下载某文件到其他目录，只能在下载完成后，打开 RemoteFiles 目录，将文件复制到其它目录；或者在上图中，单击打开按钮来下载文件，下载完成后会自动打开文件，然后另存为即可。

3.6.3.9.12 远程文件和本地文件

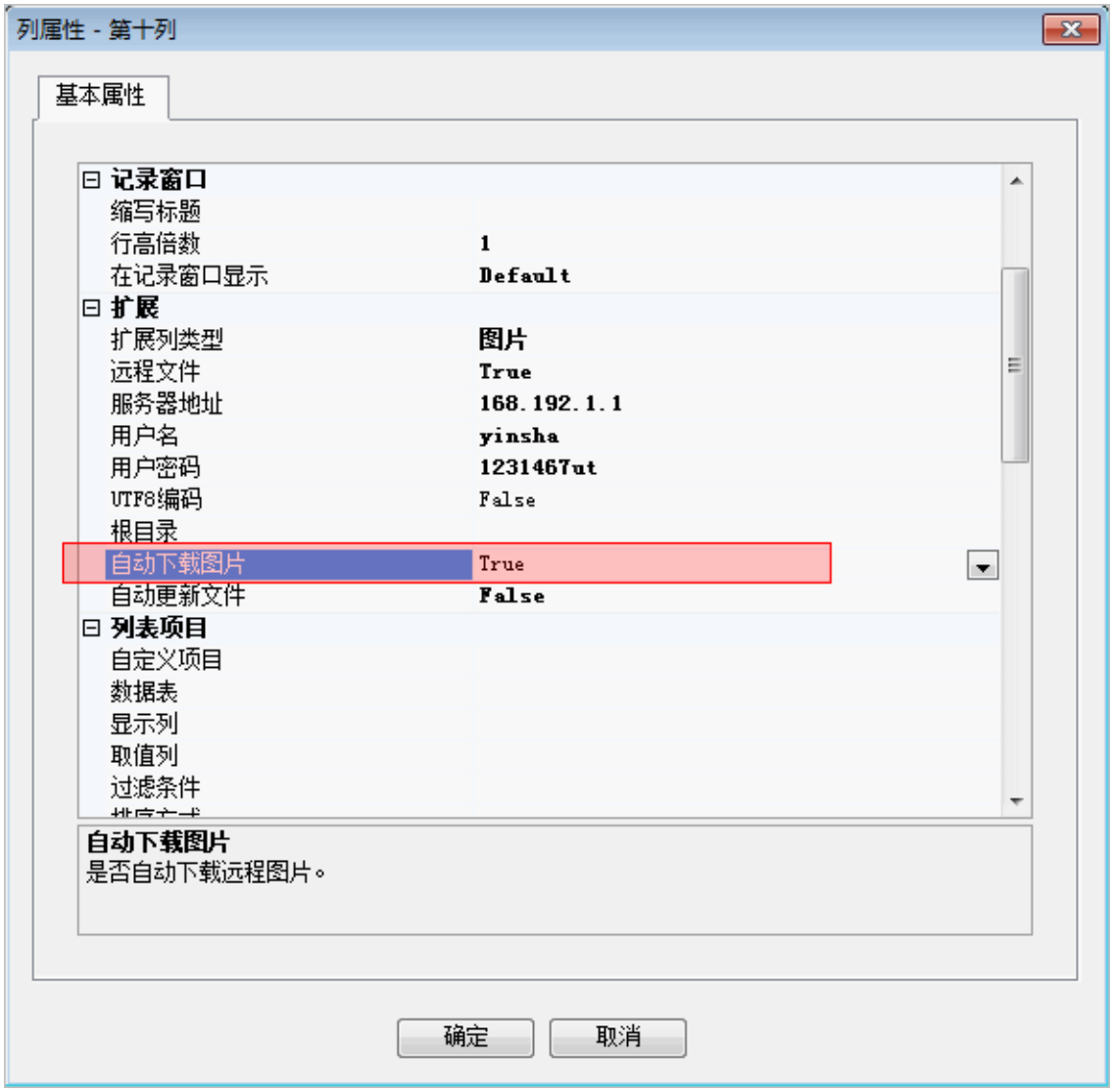
Foxtable 的远程文件管理是“智能”的，从 FTP 上下载的文件会自动保存在项目文件所在目录的"RemoteFiles"子目录中。当我们打开 FTP 上的某个文件时，系统首先会在本地(RemoteFiles 目录)中查找此文件，如果本地存在此文件，会将此文件和 FTP 上的文件进行比较，如果 FTP 上的文件比较新，将重新从 FTP 下载并打开此文件，否则直接打开本地文件。
这种特性不仅节省网络带宽，而且将大大提高远程文件管理的性能。
当使用远程文件管理的时候，列属性中将有一个名为“自动更新文件”的属性，如果将此属性设置为 False，将关闭上面所述的文件自动比较和更新的功能，始终优先使用本地文件：



3.6.3.9.13 自动下载远程图片

当使用 FTP 管理远程图片的时候，Foxtable 默认会自动从 FTP 下载并显示图片，如果你的网速不错， 会感觉和管理本地图片差不多。

如果你的网速太慢，或者不太稳定，建议在列属性设置中，将“自动下载图片”属性设置为 False：



如果禁用了自动下载图片的功能，那么只有手工双击文件名才会从 FTP 下载并显示图片：

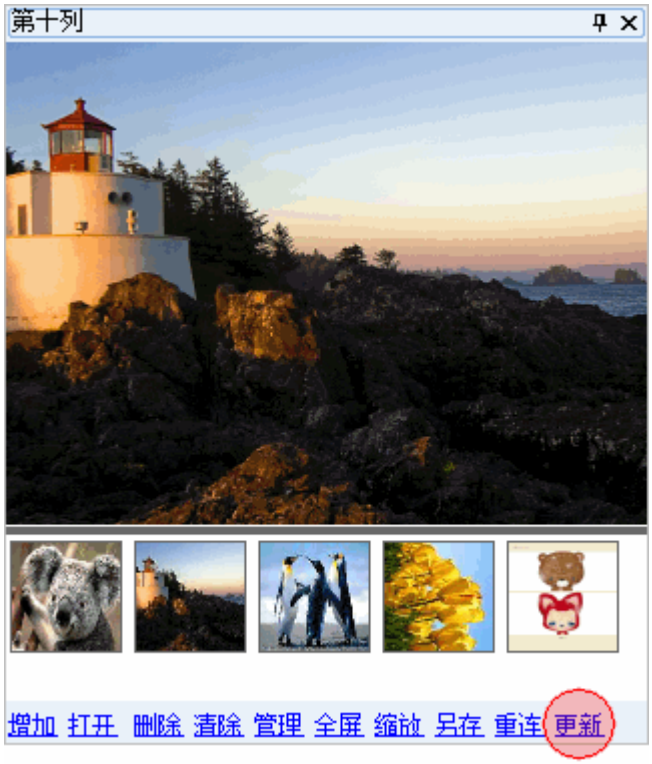


对于已经下载过的图片，以后会自动显示，无需再次双击下载。

手动更新图片

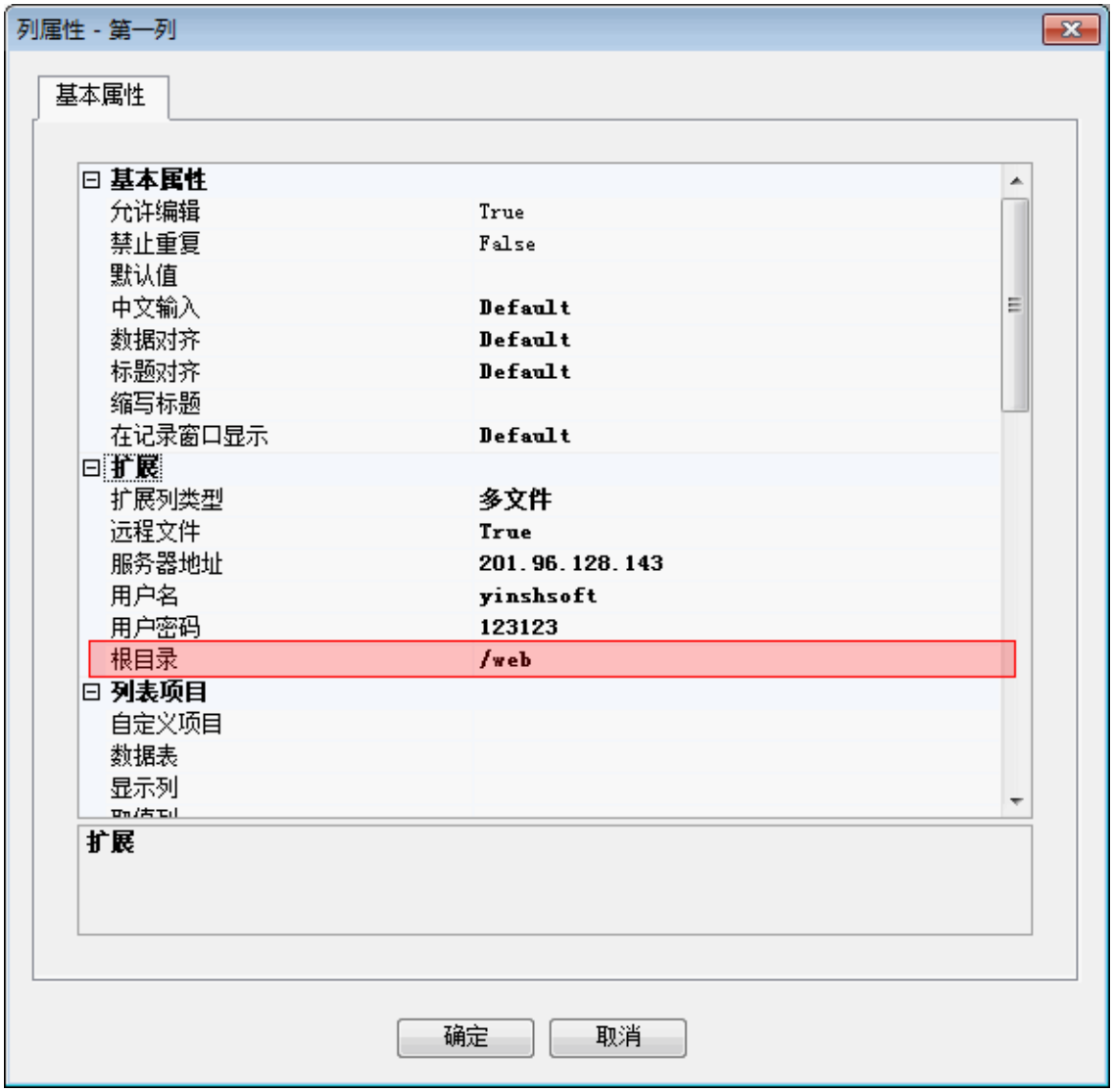
1、 如果按住 Ctrl 键的同时双击某图片，会从 FTP 重新下载此图片，相当于强制更新此图片。

2、如果启用了自动下载图片的功能，还可以单击下图所示的“更新”命令，重新下载所有图片：



3.6.3.9.14 远程文件管理的根目录

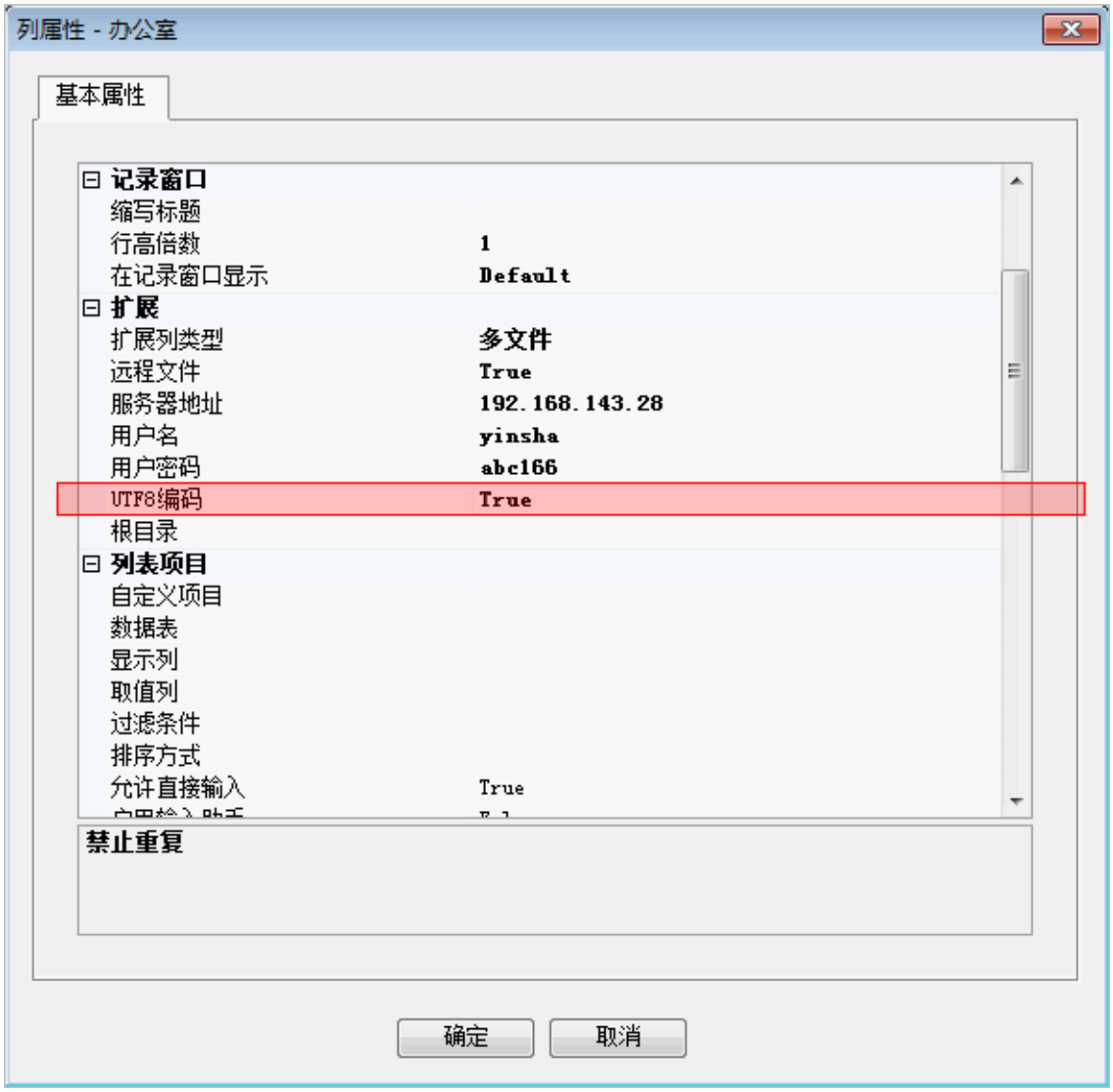
远程文件管理，离不开 FTP，一个 FTP 可能有多种类型的文件，分别放置在不同的目录中。
设置远程文件管理的属性时，可以将 FTP 上的某个目录设置为根目录，这样用户将只能管理此目录(含子目录)下的文件:



3.6.3.9.15 关于 FTP 的编码

如果需要远程文件管理，官方推荐直接采用 IIS 搭建 FTP 服务器即可，简单快捷。
不过一些用户更喜欢用专业的 FTP 软件，例如 Serv-U。
Serv-U 默认采用 UTF8 编码，在 Foxtable 中会出现中文乱码，你可以在 Serv-U 中关闭 UTF8 编码，避免乱码的出现。

如果不想关闭 UTF8 编码，可以在 Foxtable 的列属性设置中，将 UTF8 编码属性设置为 True，同样可以解决乱码问题：

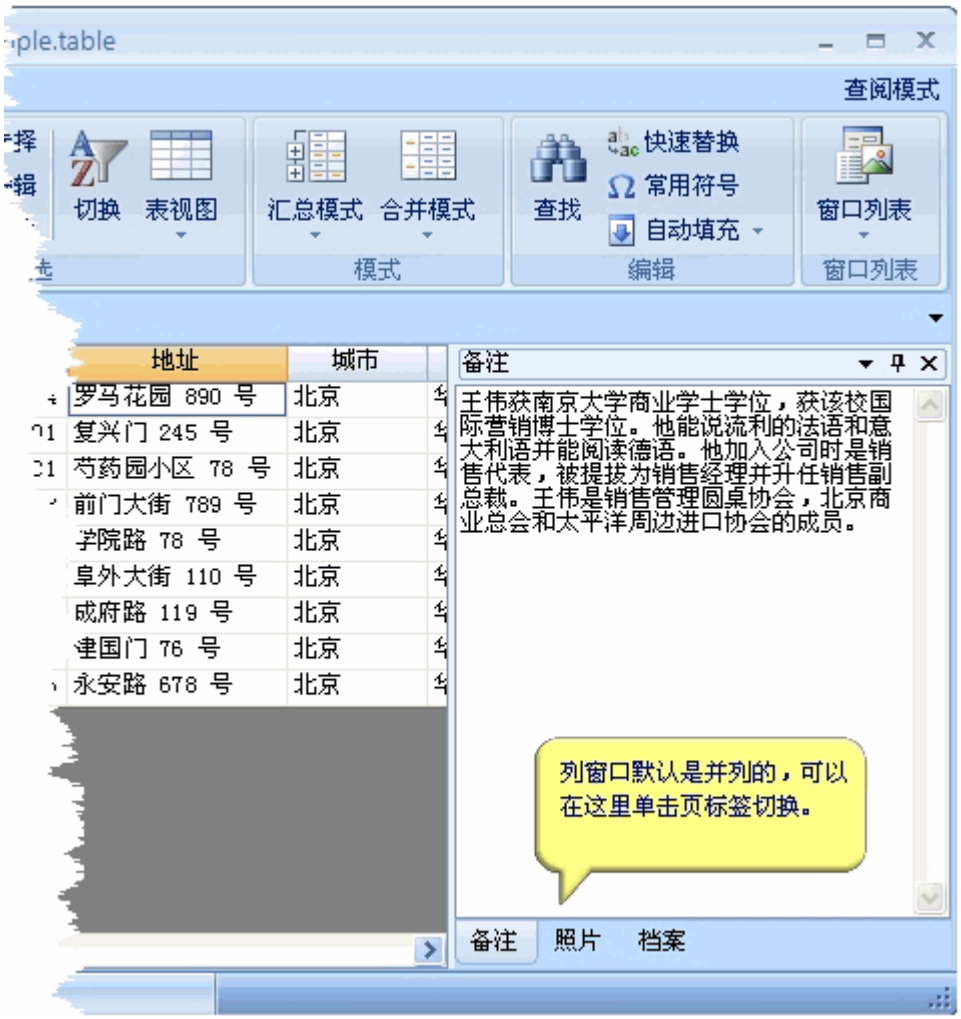


3.6.4 使用列窗口

3.6.4.1 关于列窗口

对于备注列、以及扩展类型为图片或者多文件的列，程序会在主表的右边，为这些列单独开出一个列窗口，用于备注编辑、图片管理、文件管理。

例如下图是一个员工资料管理系统，包括三个特殊类型的列，分别是备注列(备注型)，照片列(扩展类型为图片)、档案列(扩展类型为多文件)，程序在窗口右边为这三列分别开出了一个窗口：



3.6.4.2 基本操作

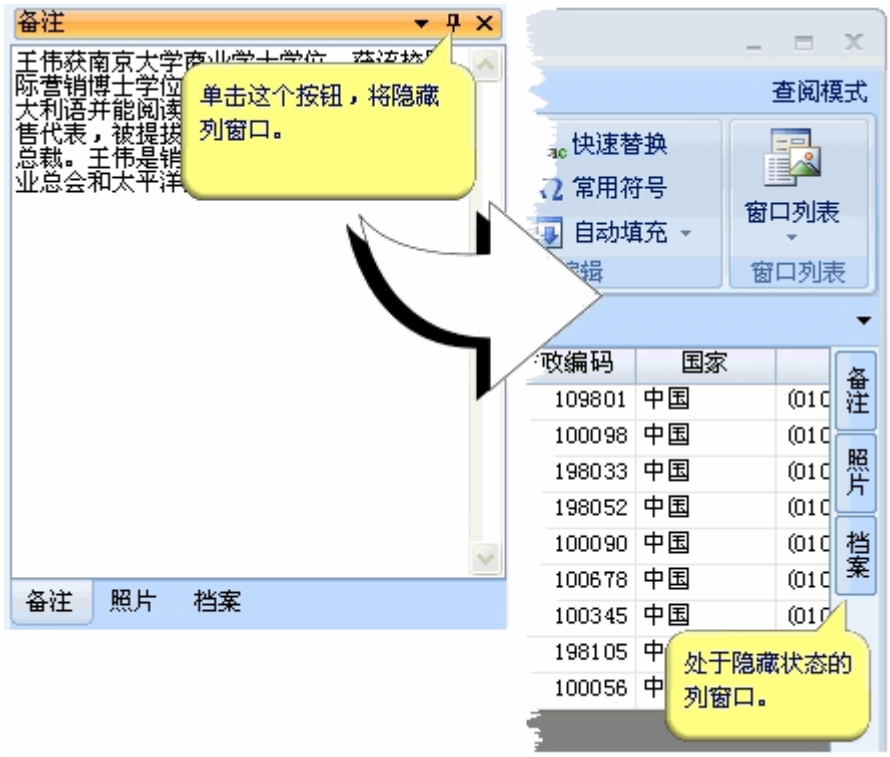
3.6.4.2.1 切换列窗口

有两种方法可以在不同的列窗口之间切换，如下图所示:



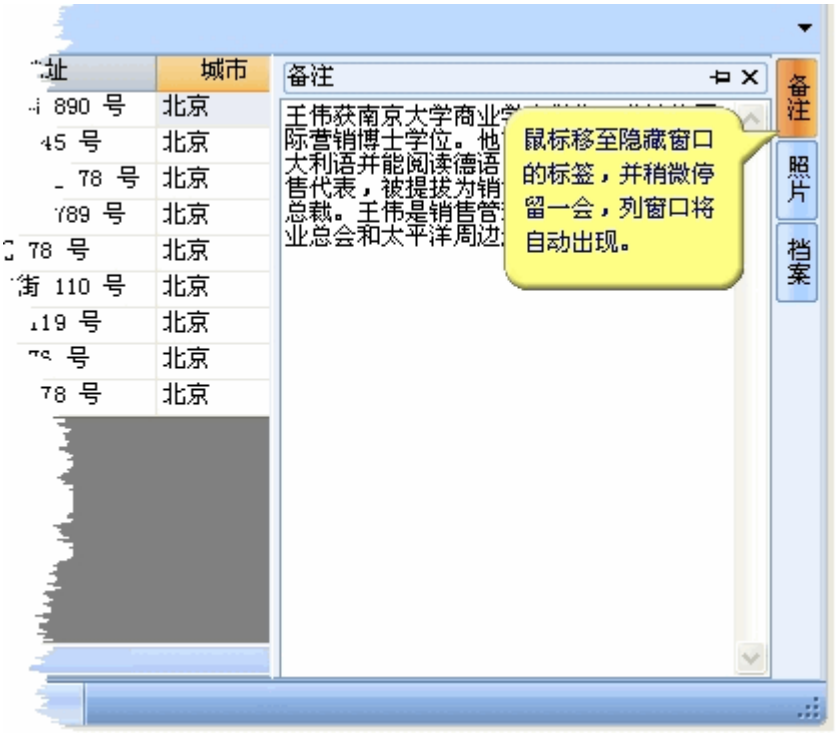
3.6.4.2.2 隐藏列窗口

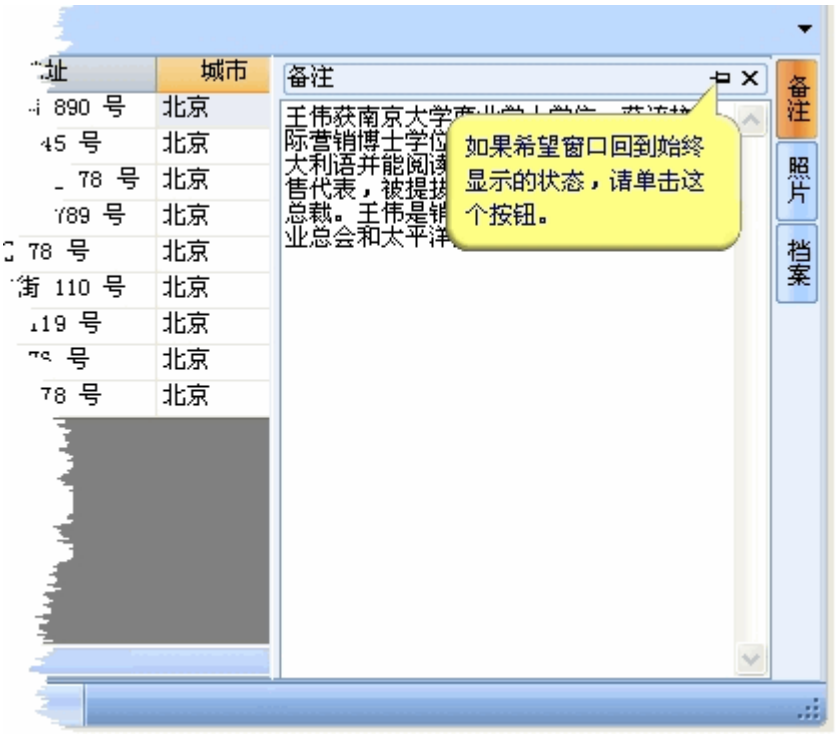
隐藏列窗口的操作如下图所示：



3.6.4.2.3 显示列窗口

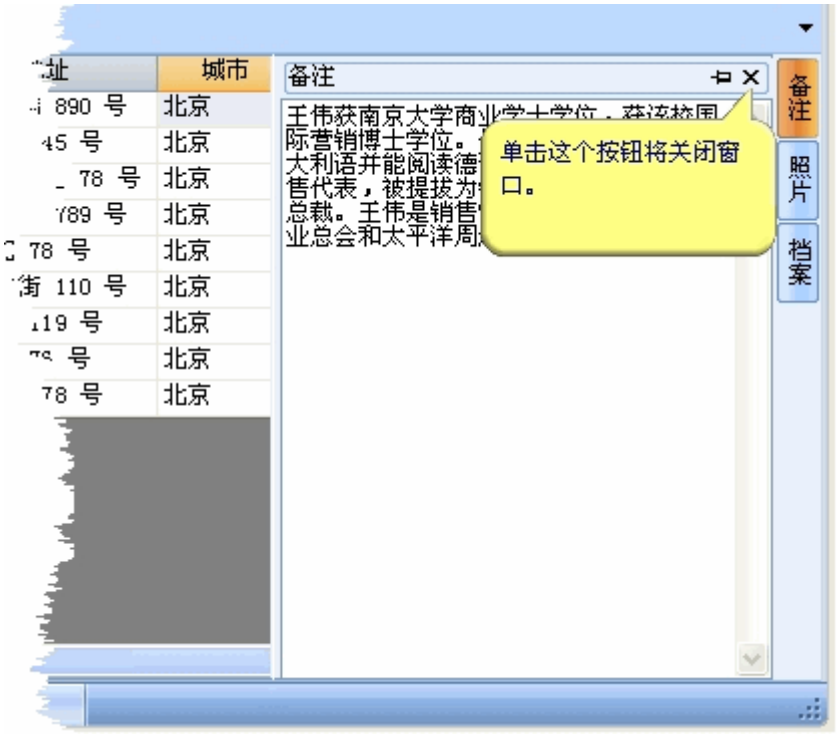
显示列窗口的操作如下图所示：





3.6.4.2.4 关闭列窗口

关闭列窗口的方法如下图所示：



3.6.4.2.5 打开列窗口

如果列窗口已经关闭，那么单击“日常工作”功能区的“窗口”按钮，在下拉菜单中指向“列窗口”，子菜单将列出所有具备列窗口的列，单击即可打开对应的列窗口：

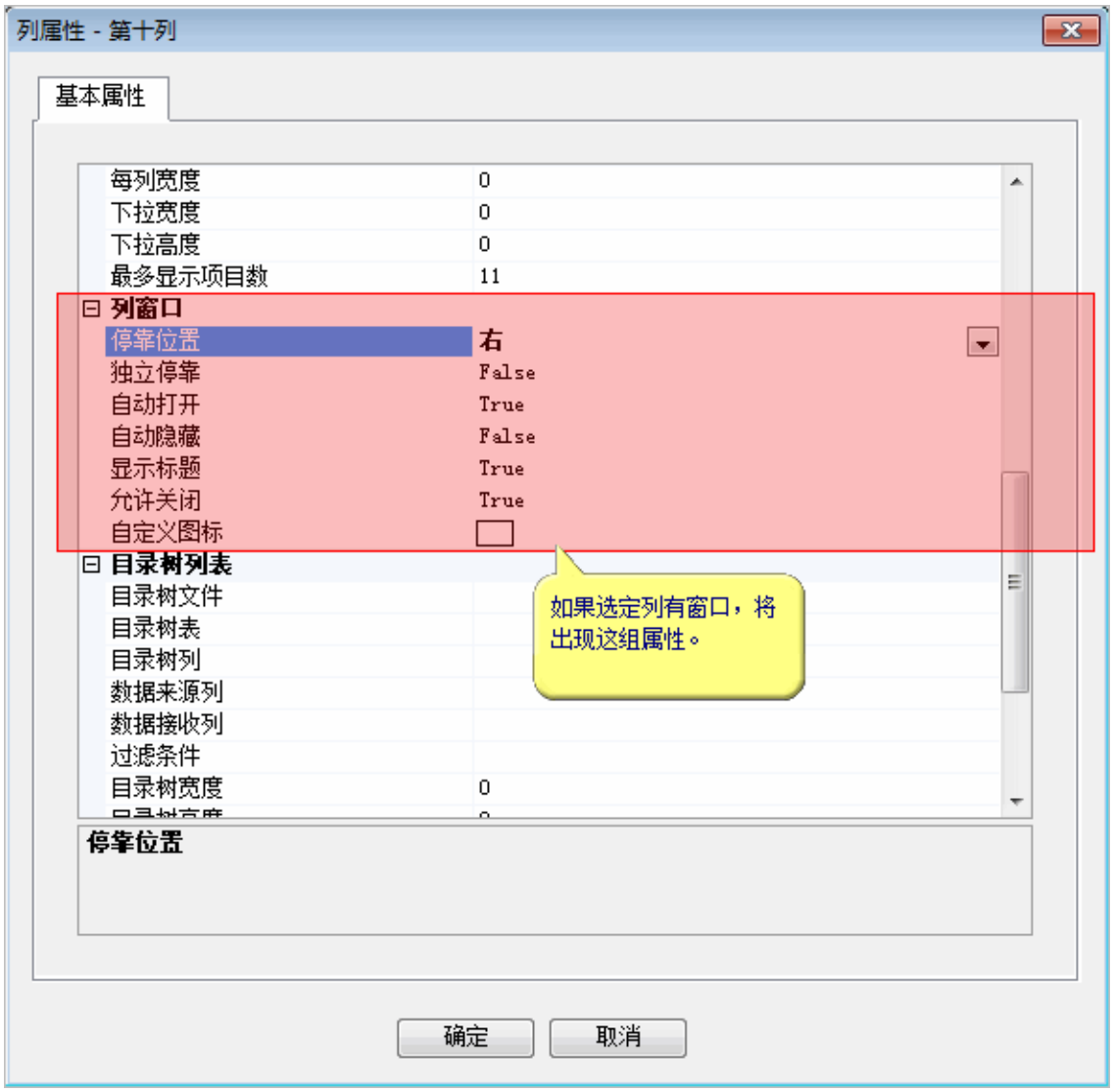


3.6.4.3 属性设置

3.6.4.3.1 相关属性

对于有窗口的列，在列属性设置窗口，将出现一组“列窗口”属性，这些属性用于设置列窗口的相

关特性。



下面我们将逐个介绍这些列属性。

3.6.4.3.2 停靠位置

列窗口默认的停靠位置是靠右，通过设置“停靠位置”属性，你可以将列窗口停靠在其它位置。
停靠位置可选值有：
左
右
上
底
需要注意的是，设置“停靠位置”属性后，只有重新打开项目或者窗口，新的设置才会生效。

3.6.4.3.3 独立停靠

在默认的情况下，所有窗口都是停靠在一起，每次只能显示其中一个。
如果将“独立停靠”属性设为 True(是)，那么该列的窗口将独立停靠，例如下图中，备注列是独立停靠的，而其它的列是正常停靠：

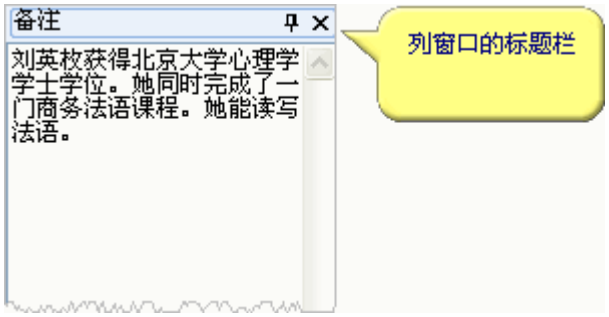


3.6.4.3.4 自动打开

打开项目文件后，列窗口默认是自动打开的，如果你不希望这样，可以在列属性设置窗口中，将“自动打开”属性设为 False(否)。

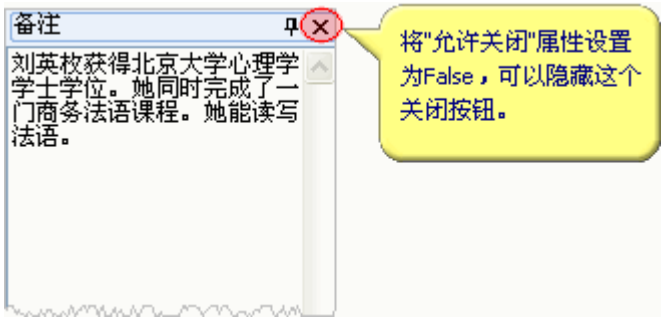
3.6.4.3.5 显示标题

如果希望列窗口不显示标题栏，可以将“显示标题”属性设置为 False。



3.6.4.3.6 允许关闭

如果希望列窗口的标题栏不显示关闭按钮，可以将“允许关闭”属性设置为 False。



3.6.4.3.7 自动隐藏

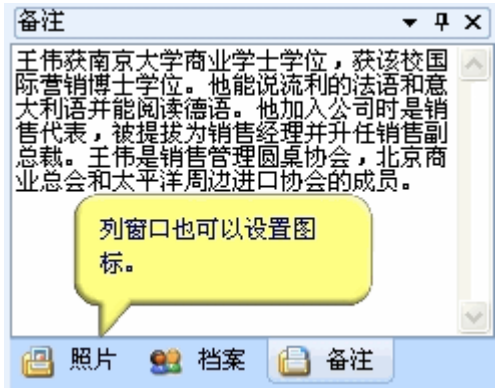
如果将“自动隐藏”设置为 True，那么列窗口打开后，将自动处于隐藏状态：



请参考：隐藏列窗口。

3.6.4.3.8 自定义图标

如果项目属性“显示图标”设为 True(是)，那么列窗口页签处会显示图标：



利用列属性“自定义图标”，可以为每一个列窗口指定个性化的图标。
只有重新打开项目或窗口，自定义图标才会生效。

3.6.4.3.9 相同停靠位置的列窗口

需要注意的是，同一个停靠位置可以停靠多个列窗口，而这些列窗口是共享标题栏的，所以对于相同停靠位置的列窗口，以下三个属性，以最后一个打开的列窗口的设置为准：
显示标题
允许关闭
自动隐藏
例如有备注和照片两个列窗口，均停靠在右边，一个列属性“显示标题”为 False，另一个为 True，那么最终列窗口有无标题是不能确定的，这要看最后打开的那个列窗口，其属性“显示标题”是否为 True。
所以对于同一停靠位置的列窗口，最好停靠设置也相同，以免用户出现困惑。

3.7 关联表

3.7.1 本章示例文件

本章课程使用的示例文件为“关联演示.Table”，位于安装目录的子目录“CaseStudy”下。
该文件是一个简化的订单管理系统，包括产品、客户、订单三个表，表的结构如下：

	产品编号	产品名称	单价
1	1	HCD238(3)P/TSD	98
2	2	HCD238(32)P/TSI	65
3	3	HW238(2)P/TD	200
4	4	HA238(8)P/T	25

产品表结构

	客户编号	客户名称	联系人	电话
1	1	联想电脑公司	周瑜	010-88667722
2	2	TCL商贸公司	关羽	020-22669900
3	3	神州数码公司	刘备	021-33667733
4	4	盛大网络公司	张飞	023-55443123

客户表结构

	日期	产品编号	客户编号	数量	单价	金额
1	2008-06-06	2	1	6	65	390
2	2008-06-06	1	2	3	98	294
3	2008-06-07	3	3	7	200	1400
4	2008-06-08	4	4	2	25	50

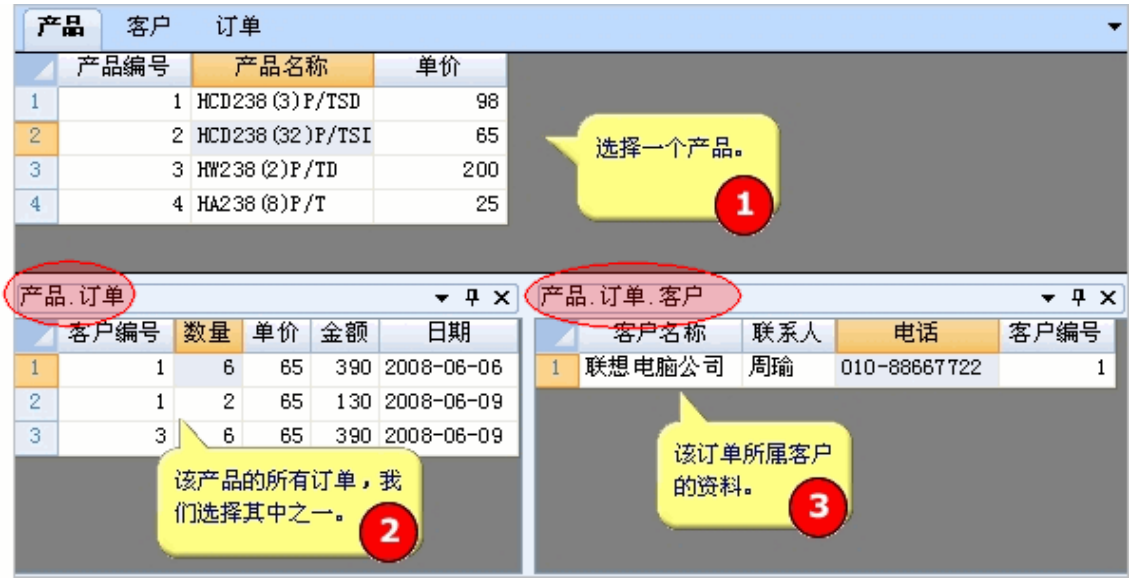
订单表结构

表结构虽然简单，但是却相当经典。

请在学习本章内容之前，先打开此文件。

3.7.2 关于表间关联

Foxtable 可以在表与表之间建立关联，而且具有一些其他数据管理软件所不具备的特性。
例如在订单管理系统中，通常包括客户、产品、订单三个表，显然订单表中的每一个订单，都分别在客户表和产品表中有对应的行，也就是说，每一个订单都会对应有一个客户和一个产品；同样，客户表或者产品表中每一行，会在订单表中对应有若干行(也就是订单)，而且订单表中的单价列是需要从产品表中提取的；如果我们能够在客户表与订单表、产品表与订单表之间建立关联，那么这些表就可以相互引用数据进行计算， 或快速查阅关联数据。
例如：
1、订单表可以引用产品表中的单价数据来计算金额，产品表也可以用表达式统计出本产品的总销售数量和金额。
2、在订单表中选择一个订单，即可用关联表的形式列出该订单所属的产品和客户。
3、在产品表中选择一个产品，即可用关联表的形式列出该产品所有的订单，而在该关联表中选择一个订单，会用另一个关联表列出该订单所属客户的资料。
4、在客户表中选择一个客户，即可用关联表的形式列出该客户所有的订单，而在该关联表中选择一个订单，会用另一个关联表列出该订单所订购产品的资料。
通过关联表，可以让所有相关数据唾手可得：



接下来，我们看看如何来建立关联、使用关联表；有关关联表之间的数据引用和统计，在“表达式”这一章讲述。

3.7.3 设置表间关联

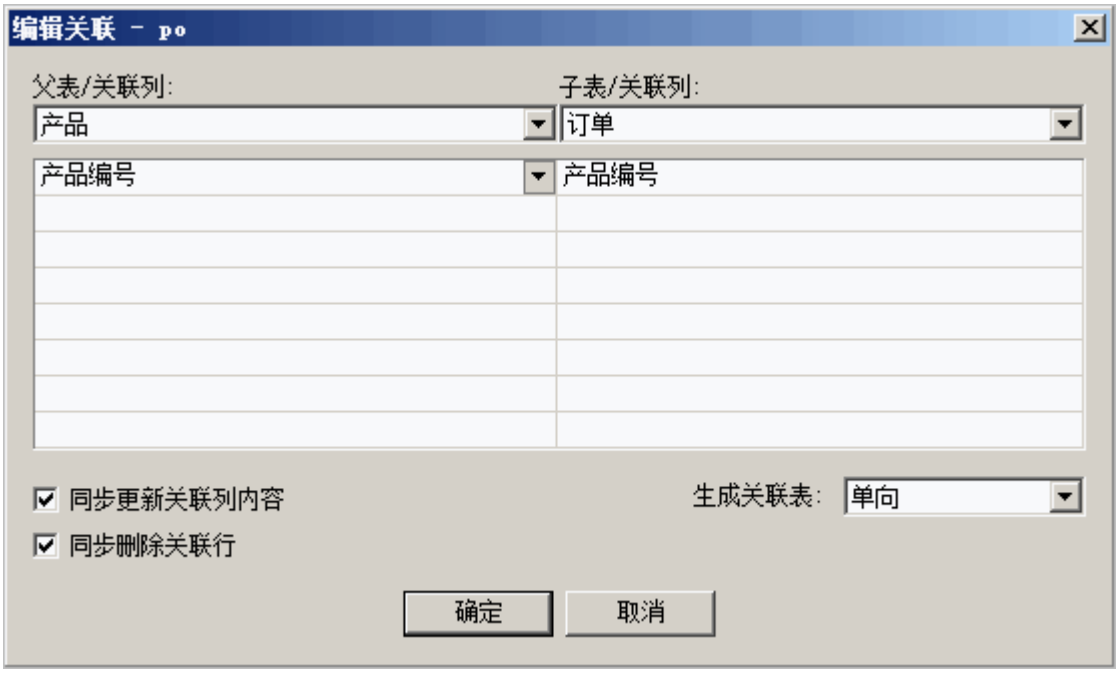
表与表之间是通过列关联起来的，例如产品表和订单表通过共有的产品编号列关联起来，客户表和订单表通过共有的客户编号列关联起来，这种将表关联起来的列，我们称之为关联列。
关联列必须具备相同的数据类型，通常也会同名，但不是必须同名。
下面我们分别在产品表和订单表、客户表和订单表之间建立两个关联。
1、在菜单的“数据表”功能区，单击“表”功能组中的“关联”按钮：



2、出现表间关联窗口，单击“增加”按钮：



3、出现关联设置窗口，按下图所示，在产品和订单表之间，通过产品编号列建立关联：

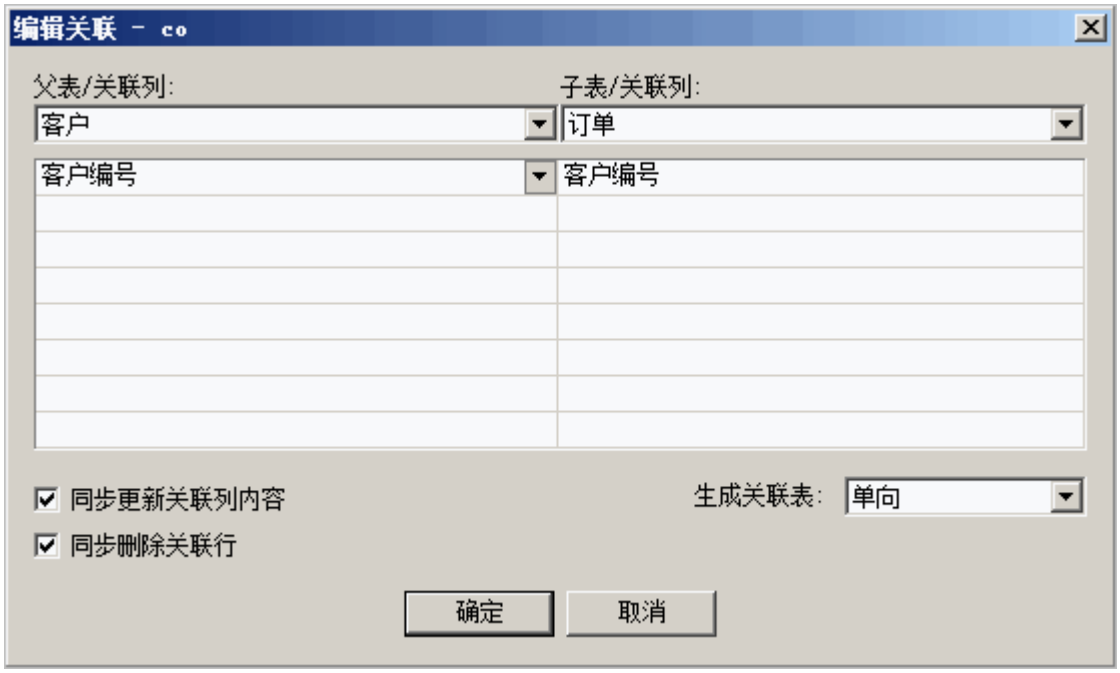


设置好之后，单击确定按钮，系统要求输入关联名称，你可以输入“产品_订单”，表示这个产品表和订单表之间的关联，我个人更喜欢简洁的名称，所以我将这个关联命名为“po”，“p”表示产品(product)，“o”表示订单(order)；你可以使用自己喜欢的名称，一个简洁易解的关联名称，会使您的计算公式更为简洁，计算公式我们会在接下来讲述。

和表名、列名一样，关联名称不能以数字开头，更不能全是数字，也不能包括空格和各种符号(下

划线除外)。

4、按同样的方法，增加客户表和订单表之间的关联：



这个关联我们命名为“co”，“c”表示客户(customer)，“o”表示订单(order)，也就是客户和订单之间的关联，你可以取一个你认为更合理的名字。

5、设置好关联之后，单击“确定”按钮退出。

提示：

1、表达式列不能作为关联列使用。

2、父表关联列和子表关联列的数据类型必须相同。

3、如果你的项目中有较多的表和关联，那么千万不要用 po、co 这样过于简洁的关联名称，而应该使用可读性更强的名称，例如“产品_订单”、“客户_订单”

3.7.4 父表和子表

如果您细心的话，可能已经留意到，在关联编辑窗口，我们将其中一个表称为父表，另一个表称为子表，那么父表和子表是如何界定的呢？还是举个例子来说明吧，例如对于产品表和订单表，订单表中的每一条记录只能对应产品表中的一个产品（记录），而每一个产品在订单表中可能对应有多个记录（订单），就像一个父亲可以有多个儿子，而一个儿子只能有一个父亲一样，这种一对多的关系，我们也称之为父子关系 ；很明显，在产品表和订单表之间建立关联的时候，产品表是父表，订单表是子表，他们之间建立的关联也称之为父子关联。



实际上，Foxtable 中这种父子关系的约束不是很严格，也就是说，如果你喜欢，子表中的一条记录，也可以在父表中对应有多条记录， 例如一个作者可以写多本书，而一本书可以有多个作者，不过实际应用的时候，这种情况并不多见。

3.7.5 自动合计子表

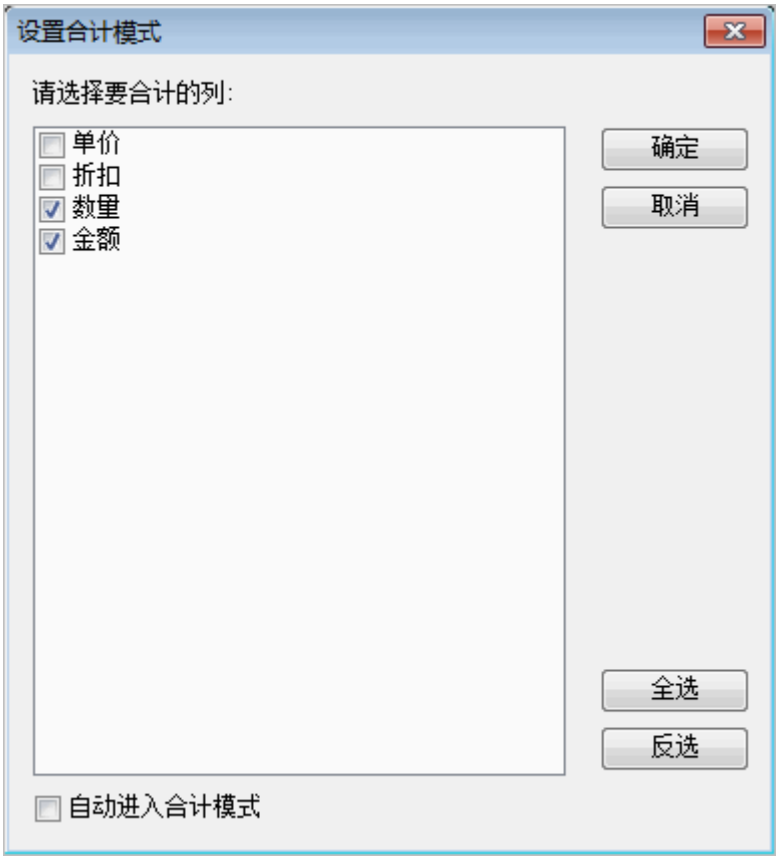
如果想在关联表中，显示一个合计行，且在增加、删除、修改数据后，合计结果能自动刷新：



要实现这个功能很简单，首先选择要显示合计行的关联子表，然后在菜单的"日常工作"区，单击"数据统计"按钮，指向"合计模式"，会出现合计模式的相关菜单命令：



执行设置“设置合计模式”命令，出现下图所示的设计窗口，勾选要合计的列，单击“确定”按钮即可：



提示： 如果希望打开项目后，能够自动进入合计模式，可以勾选选项“自动进入合计模式”。

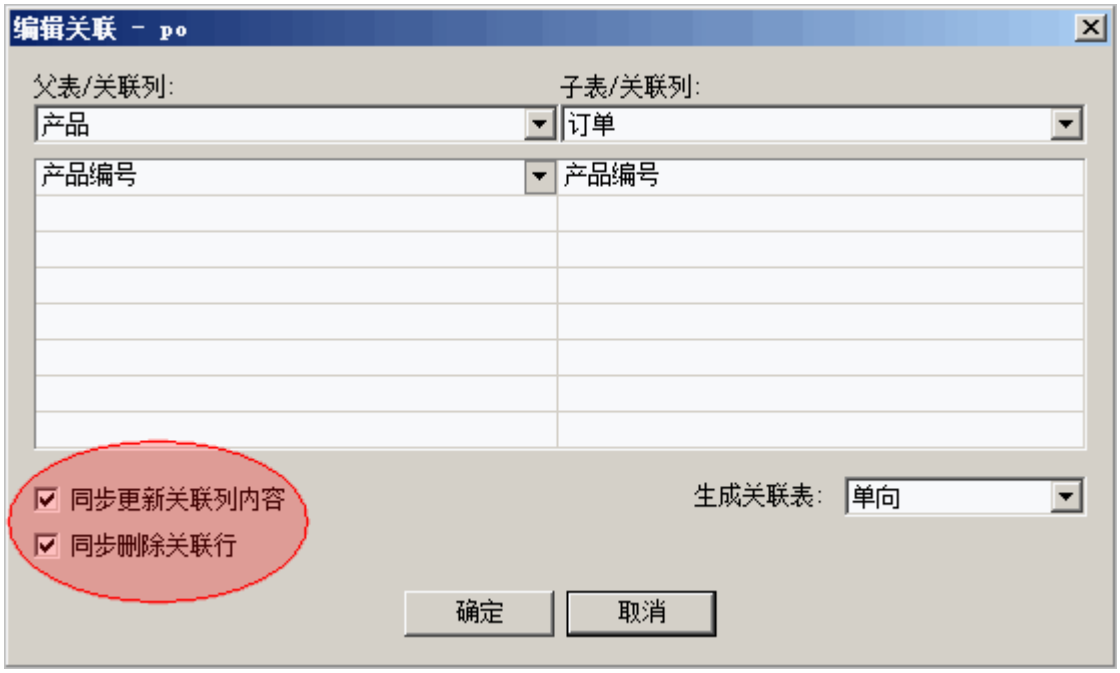
3.7.6 关联选项

在关联设置窗口，可以看到两个选项，分别是：

- 同步更新关联列内容
  如果选中此选项，在父表修改某行关联列内容后，子表中对应行的关联列内容会同步更新，使得两个表之间的关联行始终保持联系。
  例如产品表和订单表通过产品编号建立了关联，如果修改产品表中某产品的产品编号，那么

订单表中对应行的产品编号也会自动修改为新的编号。

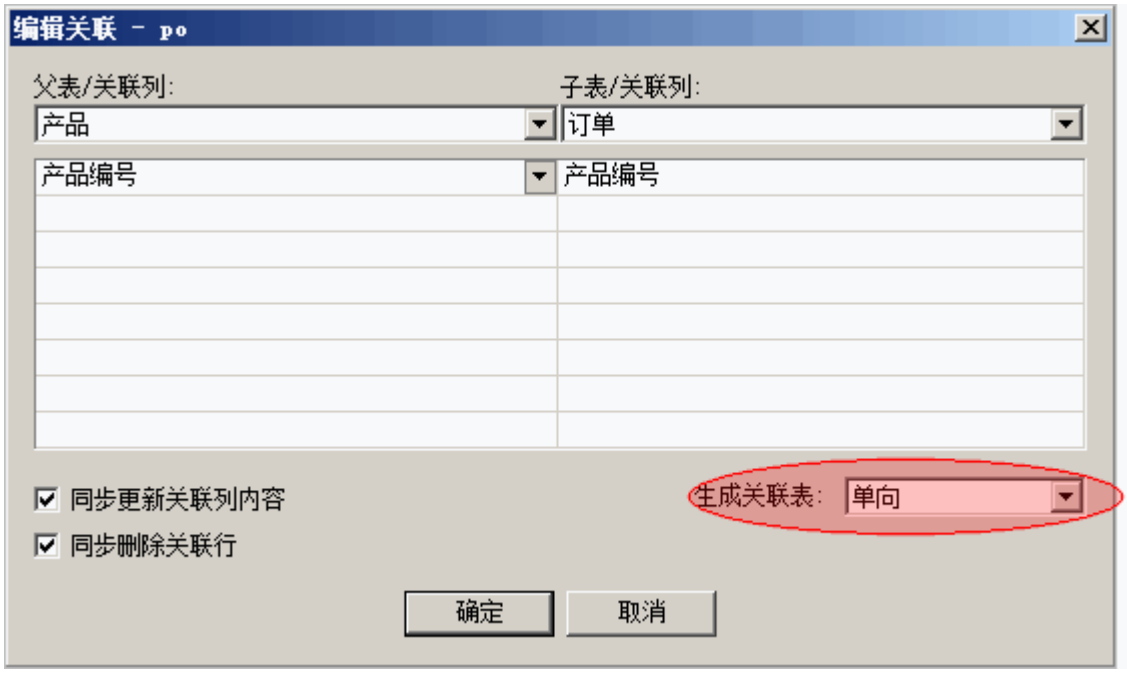
- 同步删除关联行
  如果选中此选项，在父表中删除某行，在子表中对应的关联行会同步删除。
  例如产品表和订单表通过产品编号建立了关联，如果在产品表中删除某产品，那么订单表中此产品的全部订单也会自动被删除。



提示：
如果父表中某一行在子表中存在关联行，此时你可以修改其关联列内容，但是无法删除关联列内容。

3.7.7 关联表生成模式

在设置关联的时候，有一个选项用于选择如何生成关联表：



关联表有三种生成模式，分别为单向、双向、无。

以产品和订单之间的关联为例，我们来分别介绍一下这三种生成模式。

双向

在双向生成模式下：

1、产品表的底端会出现一个关联表，名称为"产品.订单"，选择某产品时，此关联表会显示该产品的所有订单:



2、订单表的底端同样会出现一个关联表，名称为"订单.产品"，选择某订单时，此关联表会显示该订单订购产品的资料。



单向

单向生成是默认的模式，在单向生成模式下：

1、产品表的底端会出现一个关联表，名称为"产品.订单"，选择某产品时，此关联表会显示该产品的所有订单:



2、但是订单表的底端并不会出现关联表，这样在订单表选择某订单时，我们无法通过关联表查看该订单订购产品的资料。

无

如果生成模式选择“无”，那么不管是产品表和还是订单表，底端都不会出现关联表，这样既不能在选择某产品时，查看订购此产品的订单，也不能在选择某订单时，查看其订购产品的资料。

你也许会感到奇怪，既然如此，那么还建立关联干什么？

其实建立关联，有时不仅仅是为了方便查看关联数据，更重要的是关联表之间相互可以进行数据的引用和计算，随后的“表达式”一章以及后续的“开发指南”对此会有详细介绍。

下面来总结一下：

1、选择双向生成模式，会分别在父表和子表生成关联表，可以双向查看关联数据。

2、选择单向生成模式，只会在父表生成关联表，用于查看子表关联数据，这是默认的生成模式。

3、如果生成模式选择"无"，则不会生成任何关联表。

3.7.8 双向生成的优势

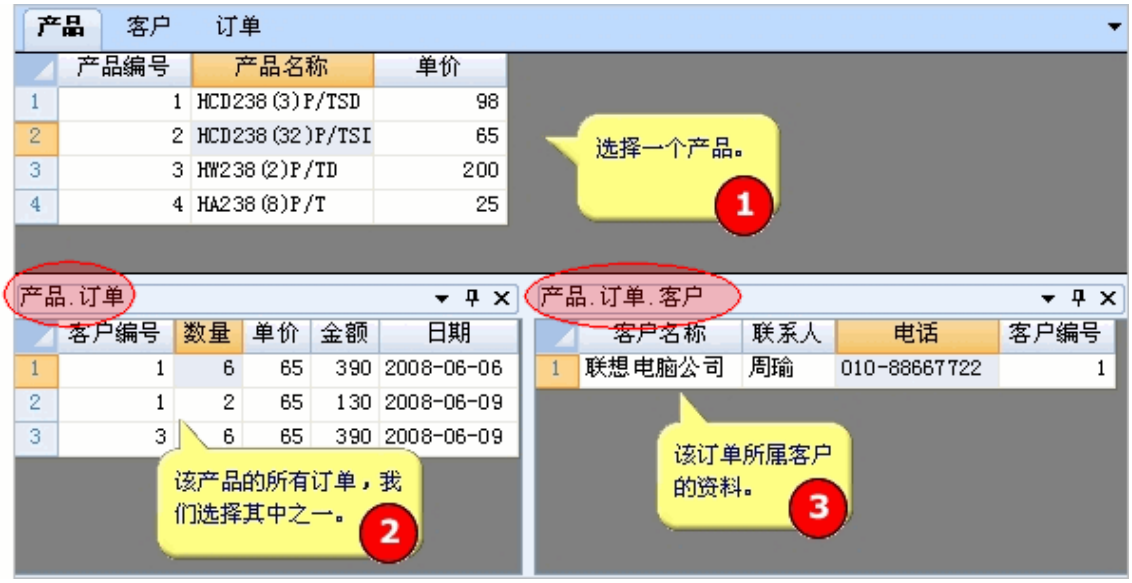
双向生成模式可以给我们的工作带来很多便利，使得无论我们选择哪一个表，所有相关的数据都可以信手拈来。

假定我们分别在产品表和订单表之间、客户表和订单表之间建立了关联：

1、在产品表

在产品表会生成两个关联表，分别为"产品.订单"和"产品.订单.客户"。

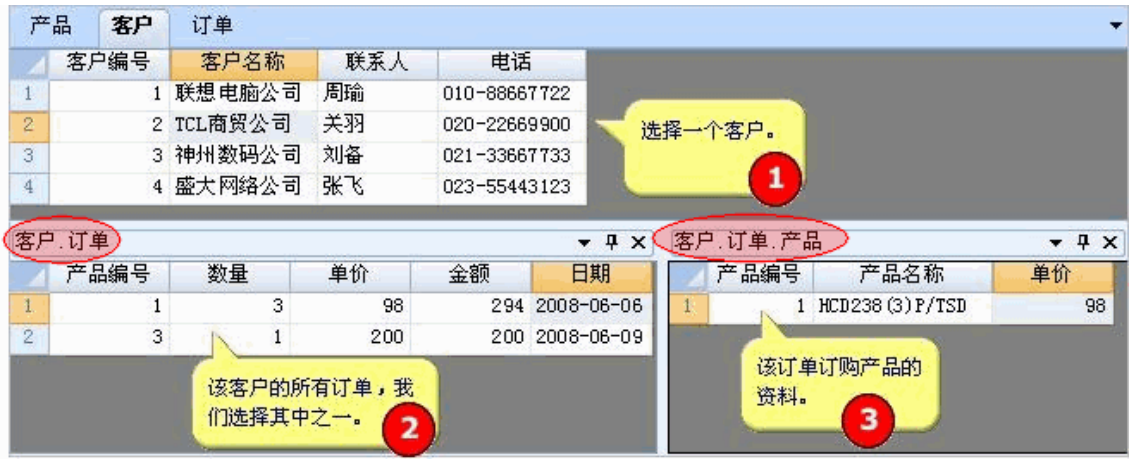
在产品表选中某产品，关联表"产品.订单"会列出该产品的所有订单，在关联表"产品.订单"选择某订单，关联表"产品.订单.客户"会列出此订单所属的客户：



2、在客户表

在客户表会生成两个关联表，分别为"客户.订单"和"客户.订单.产品"。

在客户表选中某客户，关联表"客户.订单"会列出该客户的所有订单，在关联表"客户.订单"选择某订单，关联表"客户.订单.产品"会列出此订单订购的产品：



3、在订单表

在订单表会生成两个关联表，分别为"订单.客户"和"订单.产品"。

在订单表选中某订单，关联表"订单.客户"会列出此订单所属的客户，而关联表"订单.产品"则会列出此订单订购的产品：



如果是单向生成，那么就只有两个关联表，一个显示在产品表底端，名为"产品.订单"，用于显示选定产品的全部订单，一个显示在客户表的底端，名为"客户.订单"，用于显示选定客户的全部订单，无法做到双向生成那样环环相扣，相关数据信手拈来。

3.7.9 双向生成的劣势

正如我们上一节介绍的那样，假定我们分别在产品表和订单表之间、客户表和订单表之间建立两个关联，那么：
1、在双向生成模式下，系统会产生 6 个关联表，加上原来的 3 个数据表，合计就有 9 个表
2、在单向生成模式下，系统会产生 2 个关联表，加上原来的三个数据表，合计 5 个表。
可以看出，双向生成产生的关联表数量，远远超过了单向生成。

在双向生成模式下， 随着关联的增加，表的数量会按平方数增长，例如有 5 个数据表，建立了 4 个关联，而这 4 个关联都选择了双向生成，那么就将产生 20 个关联表，加上原有的 5 个数据表，合计就有 25 个表 。
打开项目的时候，要分别生成这 25 个表，运行的过程中还要维护这 25 个表，肯定会对性能会有一定程度的影响。
当然实际使用的时候，我们不可能将 4 个关联都开启双向生成，这里只是举个例子来说明滥用双向生成可能带来的影响而已。

所以如果要追求更好的性能，更快的启动速度，那么除非有必要，不要轻易开启双向关联。
如果你建立关联只是为了引用和统计数据，那么连单向生成关联表都可以禁止，在设置关联的时候，直接将生成模式选择为“无”。

请各位不要因此而“歧视”双向生成，适当使用双向生成，对性能并不一定会有明显影响，何况如今的电脑普遍性能过剩，而便利也是我们的主要追求之一。
这里只是想告诉大家，不要滥用双向生成，以免对性能造成不必要的影响而已，实际应用的时候，每个关联都应该根据具体情况来选择生成模式，无需过于敏感。

3.7.10 多级关联

如果表 A 和表 B 建立了关联，表 B 和表 C 建立了关联，也就是表 A 是表 B 是父表，表 B 又是表 C 的父表，三者的关系类似：父(表 A) → 子(表 B) → 孙(表 C)

在双向生成模式下：

表 A 生成两个关联表，分别是：表 A.表 B，表 A.表 B.表 C
表 B 生成两个关联表，分别是：表 B.表 A，表 B.表 C
表 C 生成两个关联表，分别是：表 C.表 B，表 C.表 B.表 A
合计有 6 个关联表。
在单向生成模式下：
表 A 生成两个关联表，分别是：表 A.表 B，表 A.表 B.表 C
表 B 生成一个关联表：表 B.表 C
表 C 不生成任何关联表。
合计有 3 个关联表。
可以看出，单向生成并非每个关联只生成一个关联表，而是按照“父 → 子 → 孙”这样的路径逐级生成，而双向生成除了这条路径外，还多了个反向的“孙 → 子 → 父”的路径，所以生成的关联表数量至少要翻倍。
在双向生成的时候，按照“父 → 子 → 孙”路径生成的关联表有：
表 A.表 B
表 B.表 C
表 A.表 B.表 C
按照“孙 → 子 → 父”路径生成的关联表有：
表 B.表 A
表 C.表 B
表 C.表 B.表 A

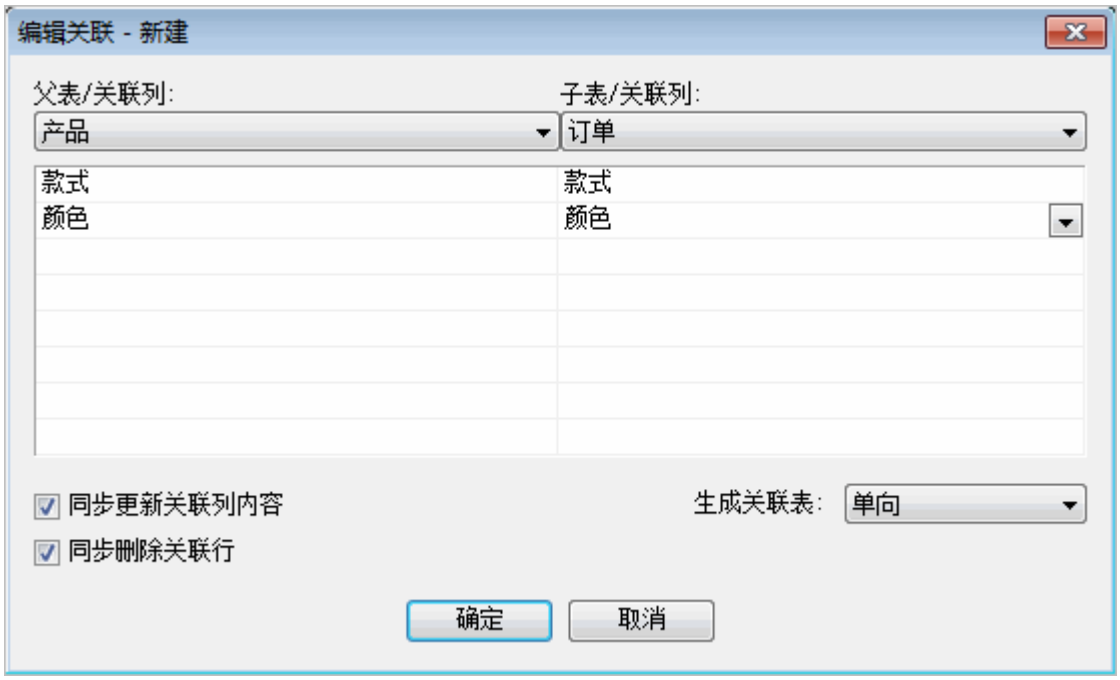
3.7.11 同表关联

关联通常是将两个不同的表通过关联列联系起来。
实际上建立关联的时候，父表和子表也可以是同一个表。
例如有一个员工表，其中有一列名为“上级”，用于输入此员工的直接上司的姓名。
如果我们新建一个关联，父表为员工表，关联列为姓名，子表同样是员工表，关联列为上级，这样我们选择某个员工的时候，会自动列出此员工的所有直接下属。
提示：
1、双向生成对同表关联无效，即使你选择双向生成，系统还是会按照单向生成模式生成关联表。
2、建立同表关联的时候，父表和子表的关联列必须不同。

3.7.12 多个关联列

两个表之间可以通过多个列建立关联，例如下面打产品表和订单表，通过款式和颜色两列关联起

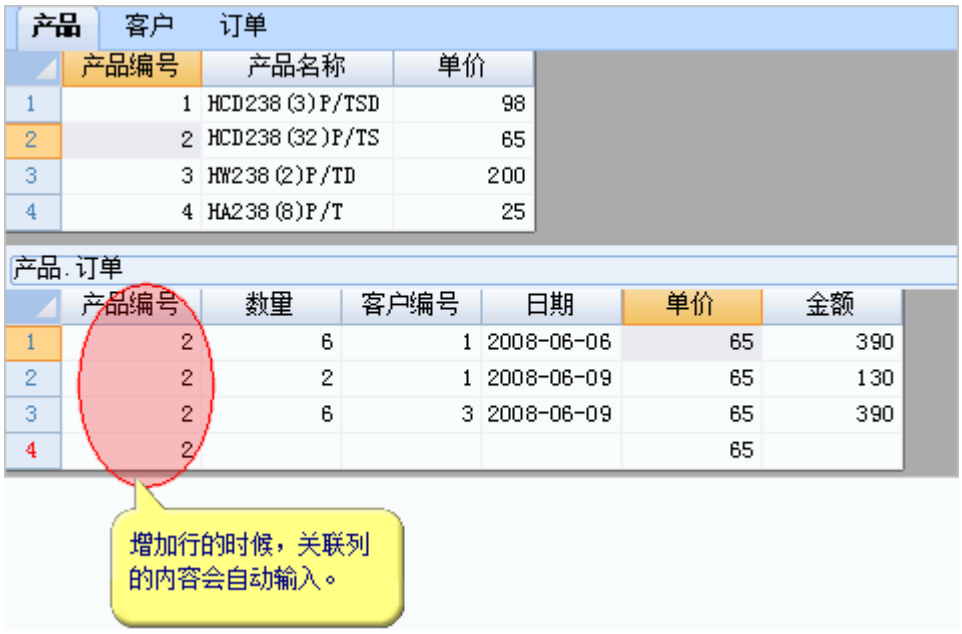
来：



虽然没有限制，但是一般建议关联列的个数不要超过两个，例如产品表和订单表，如果要通过款式、尺码、颜色三列建立关联，不如新增一个产品编号列，然后通过这个产品编号列建立关联。

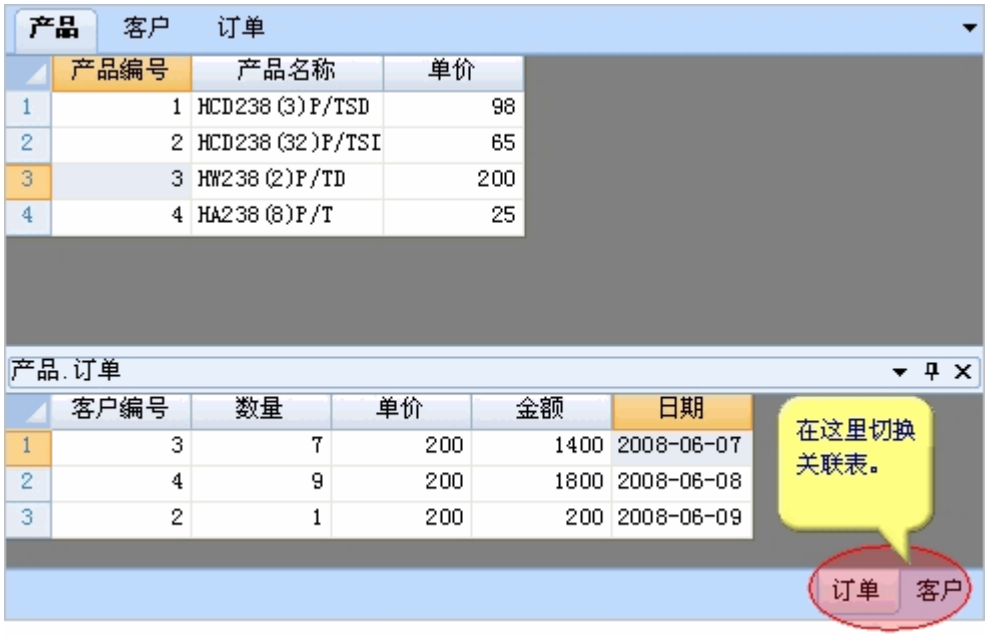
3.7.13 在关联表增加行

首先选择关联表，同时按“Ctrl + A”即可增加行，当然也可以通过主菜单或者快捷菜单增加行。
在关联表增加行的时候，关联列的内容会自动输入。
例如下图中，产品表和订单表通过产品编号列建立关联，当我们在产品表中选择编号为“2”的产品，然后在关联表“产品.订单”中增加行，新增行的产品编号列会自动输入“2”：



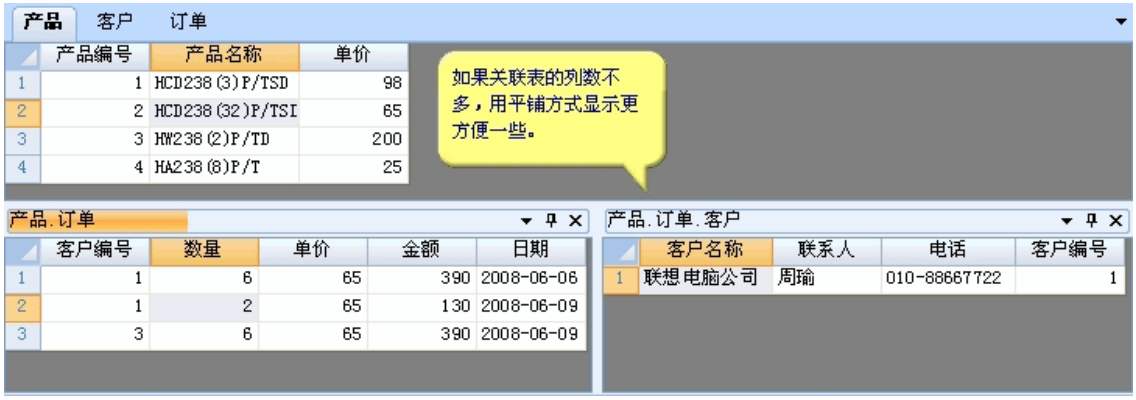
3.7.14 关联表默认位置

在默认情况下，关联表并不是我们之前看到的那样平铺显示，而是和主表一样并列显示，每次只能显示一个关联表，只不过切换关联表的页签位于右下角：



如果关联表的列数较多，用默认的这种方式比较好，因为可以同时查看关联表中更多的列数。

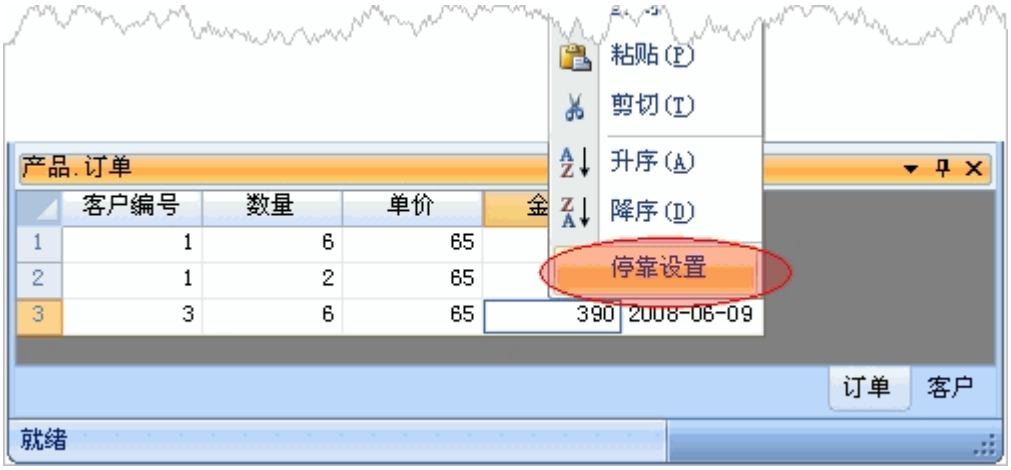
如果关联表的列数较少，那么最好通过设置，让关联表以平铺的方式显示，这样可以同时查看两个甚至更多关联表的内容：



接下来我们学习如何让关联表以平铺的方式显示。

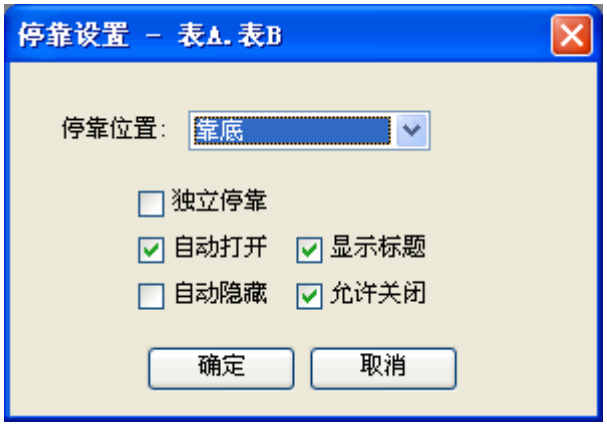
3.7.15 设置关联表窗口

1、右击关联表，在弹出的快捷菜单，单击命令“停靠设置”：



提示：关联表至少要有一行数据，快捷菜单才会出现“停靠设置”命令。

2、出现关联表停靠设置窗口：



该窗口的设置说明如下：

- 停靠位置
  指定关联表窗口的停靠位置，有三个位置可选，分别是：左、右、底，默认为底。

- 独立停靠
  选中该选项，此关联表将不再和其他关联表并列在一起，而是独立停靠，始终显示；我们之前看到的那种平铺效果，就是通过这个选项实现的。

- 自动打开
  打开项目的时候，关联表窗口默认是自动打开的，如果不希望此关联表窗口自动打开，请不要勾选此选项即可。

- 显示标题
  如果希望关联表窗口不显示标题栏，可以将“显示标题”属性设置为 False。

- 允许关闭
  如果希望关联表窗口的标题栏不显示关闭按钮，可以将“允许关闭”属性设置为 False。

- 自动隐藏
  如果将“自动隐藏”设置为 True，那么关联表窗口打开后，将自动处于隐藏状态。

需要注意的是，同一个停靠位置可以停靠多个关联表，而这些关联表是共享标题栏的，所以对于相同停靠位置的关联表，上面最后四个设置，以最后一个打开的关联表的设置为准。
同一停靠位置的关联表，最好停靠设置也相同，以免用户出现困惑。

3.7.16 打开关联表窗口

如果关联表窗口已经关闭，那么单击“日常工作”功能区的“窗口”按钮，在下拉菜单中指向“关联表”，子菜单将列出所有关联表名称，单击即可打开对应的窗口：



关联表窗口的隐藏、显示、关闭、切换等操作，和之前介绍的列窗口完全一致，这里就不再复述一次了。

3.7.17 关联表的列位置

关联表的列位置和列宽，是独立的，并不会从主表继承。
我们可以单独调整关联表的列宽和列位置，然后单击一次菜单中的保存按钮，重新打开项目，关联表的列位置和列宽会保持不变。

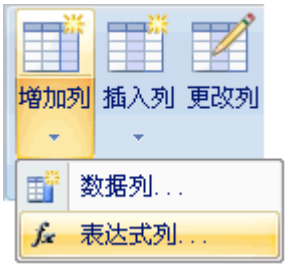
3.8 表达式

3.8.1 本章示例文件

本章课程使用的示例文件为“表达式演示.Table”，位于安装目录的子目录“CaseStudy”下。
请在学习本章内容之前，先打开此文件。

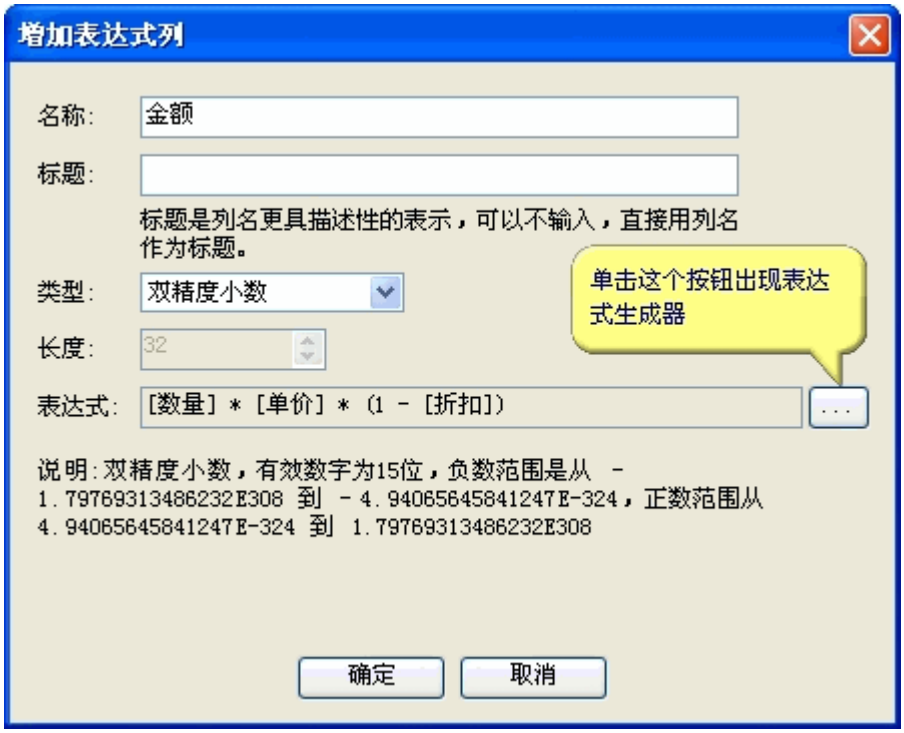
3.8.2 增加表达式列

在“数据表”功能区的“列”功能组，单击下面的按钮，可以增加表达式列：



注意增加列是一个组合按钮，你需要单击按钮的下半部分，才会出现下拉菜单。

这是增加表达式列的窗口：



这是表达式生成器：



3.8.3 修改表达式

在“数据表”功能区的“列相关”功能组，单击下面的按钮，可以修改选定列的表达式：



如果选定列不是表达式列，会提示你是否新建一个表达式列。

3.8.4 运算符与函数

本节详细介绍表达式所支持的运算符和函数。

CaseStudy 目录下有一个示例文件“表达式.table”，可以帮助大家快速掌握本节介绍的运算符和函数。

运算符

在创建比较表达式时，允许使用下列运算符：

<

>

<=

>=

<>

=

In

Like

在表达式中还支持下列算术运算符：

+（加）

-（减）

*（乘）

/（除）

%（模数）

对于数值，允许使用小数和科学记数法。
列名称推荐用方括号括起来，字符串的值应括在单引号内。
日期值应放在符号 (#) 内，顺序为“月/日/年”例如：

[姓名] = '贺辉'

[单价] <= 50.00

[出生日期] < #9/17/1970#
#9/17/1970#表示 1970 年 9 月 17 日。
若要连接字符串，请使用 + 字符，例如：
'HA' + 'PT'
可以用 And 或者 Or 串联条件表达式。
用 And 串联的条件表达式必须同时满足，例如：
[省籍] = '湖南' And [学历] = '博士'
表示省籍为湖南，且学历为博士。
用 Or 串联的条件表达式只需满足一个即可，例如：
[省籍] = '湖南' Or [省籍] = '广东
表示省籍为湖南或者广东。
计算表达式值的时候，And 优先于 Or，可以使用括号组合表达式来强制改变优先级，例如：

([省籍] = '湖南' Or [省籍] = '广东') And [学历] = '博士'

表示省籍是湖南或者广东的博士。
Like 运算符用于比较字符串，在 Like 比较中，* 和 % 作为通配符，两者可以互换，表示任意个数的字符，通配符只能在开头或者结尾出现。例如：

[姓名] Like '%赫%'
[姓名] Like '赫%'
[姓名] Like '%赫'
分别表示姓名中包括“赫”，姓名开头是“赫”(姓赫)，姓名结尾是“赫”
在字符串的中间不允许使用通配符。例如，不允许 '赫*丰'，也不允许 '赫%丰'。
由于* 和 % 默认作为通配符，如果要将其作为比较内容的一部分，必须用方括号括起来，例如：
[型号] Like 'A[*]%'
表示型号以“A*”开头。
在 Like 运算符之前，可以加上 Not 关键词，表示不类似的意思，例如：
[姓名] Not Like '李%'
表示姓名不是以“李”开头，也就是不是姓李的。
In 运算符用来判断某一个值是否在指定的一系列值中，例如：
[国籍] In ('中国','美国','英国','法国','俄罗斯')
表示判断国籍是否是上述五国之一，显然这比用 OR 来连接多个表达式要简洁很多。
记得数值型的列，值不要用单引号括起来，例如：
[订单号] In (1, 3, 4, 6, 7,12)
而日期型的列，值需要用符号#括起来，例如：
[日期] In (#7/2/2012#, #7/12/2012#, #8/30/2012#)
同样如果在 In 运算符之前，加上 Not 关键词，表示不在指定的一系列值之中，例如：
[国籍] Not In ('中国','美国','英国','法国','俄罗斯')
表示国籍非上述指定的五国。

函数

表达式支持以下函数：

IsNull

说明	判定一个表达式的值是否为空，如果不为空，则返回表达式的值，否则返回一个替代值。
语法	IsNull(expression, replacementvalue)
参数	expression — 要判定的表达式。 replacementvalue — 指定一个替代值，如果表达式的值为空，则返回此值。

例如：IsNull([折扣], 0)
如果折扣为空，则返回 0，否则返回折扣列内容。
例如：IsNull([第一列], [第二列])
如果第一列内容为空，则返回第二列，否则返回 第一列，也就是说第一列有内容时，使用第一列的内容，否则使用第二列的内容。

IIF

说明	计算一个逻辑表达式，如果表达式的结果为 True，则返回一个指定值，否则返回另一个指定值。

语法	IIF(expr, truepart, falsepart)
参数	expr — 要计算的逻辑表达式。 truepart — 逻辑表达式的计算结果为 True 时，返回此值。 falsepart — 逻辑表达式的计算结果为 False 时，返回此值。

例如：IIF([金额] > 1000, '贵', '便宜')

Trim

说明	移除字符串的前后空格
语法	Trim(expression)
参数	expression — 要剪裁的表达式。

例如：Trim([型号])

Substring

说明	从字符串中的指定位置开始，返回指定长度的子字符串。
语法	Substring(expression, start, Length)
参数	expression — 原字符串。 start — 起始位置，从 1 开始编号。 Length — 返回长度。

例如：Substring([电话号码], 3, 9)

Len

说明	返回字符串的长度。
语法	Len(expression)
参数	expression — 原字符串。

例如：Len('abc')

3.8.5 表达式不计算？

下图是一个学生成绩表：

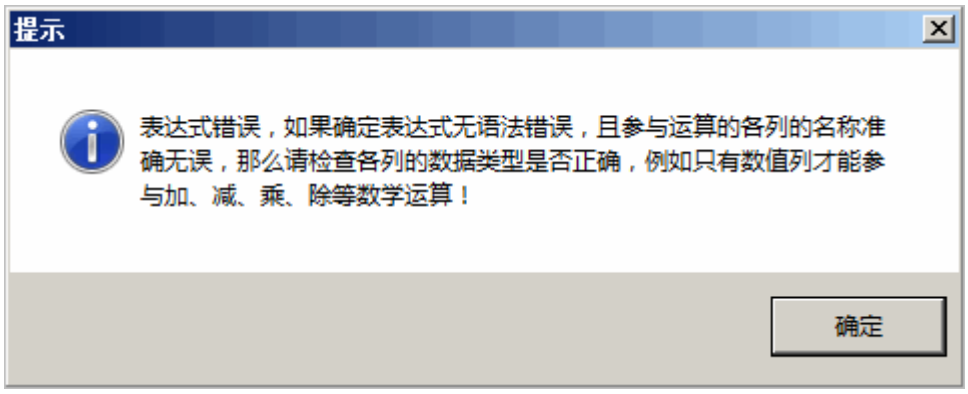
	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136		71	75	
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529

其中总分列的计算表达式为：

[语文] + [数学] + [英语] + [物理] + [化学]
现在的问题是，必须输入每一科的成绩，总分列的内容才会计算得出，例如第一个同学“陈成进”，因为没有参加英语考试，英语成绩为空，导致他的总分为空，当然如果英语成绩输入 0，总分能正常计算得出，但是零分和没有参加考试是有区别的。显然如果有办法让没有输入成绩的科目，作为零参与计算，那么问题就很好解决。
为此我们将总分列的计算表达式修改为：
IsNull([语文],0) + IsNull([数学],0) + IsNull([英语],0) + IsNull([物理],0) + ISNull([化学],0)
IsNull 函数我们前面已经介绍过，在这里用于判断某一科目的成绩是否为空，如果为空，则返回 0，否则返回该科目的成绩。

3.8.6 一定要注意列类型

一些初次接触 Foxtable 的用户，特别那些从 Excel 转过来的用户，经常会问一个类似这样的问题：我将 C 列的表达式设置为"A + B"，在 A 列输入 1，B 列输入 2，为什么 C 列结果是 12，而不是 3 呢？
这是因为 A、B 两列至少有一列是字符型的原因，只需将 A、B 两列的数据类型改为任何一种数值型，即可得到正确的计算结果。
如果表达式不是"A + B"，而是"A * B"，只要 A、B 两列中有一列是字符型，那么在设置表达式的时候，就会出现下图所示的错误提示：



正确设置列的数据类型，是一个管理项目正常运转的基础，初学者一定要注意。
例如一个订单表，客户列和产品列是字符型，订购日期列是日期时间型，数量列是整数型，折扣列和金额列则是双精度小数型。
如果要检查或更改某一列的数据类型，请参考：更改列
如果要检查某个表所有列的数据类型，请参考：查看表结构
要详细了解各种数据类型，请参考：数据类型详解

3.8.7 关于方括号

前面的例子中，表达式中的列名都用方括号括起来了，例如：
[语文] + [数学] + [英语] + [物理] + [化学]
不过方括号并非是必须的，我们也可以这样写：
语文 + 数学 + 英语 + 物理 + 化学
但是有一种情况例外，如果使用外部表，可能会出现纯数字或者包括特殊符号的列名，此时列名

必须用方括号括起来：
[a%2] + [a%1] + [123]
前两列因为包括特殊符号“%”，所以必须用方括号，而第三列的列名“123”是纯数字的，用方括号告诉系统：这是一个列名，不是数字。
关于外部表，我们会在专门的章节讲述。
官方的建议是：尽管方括号并非必须的，但是表达式中的列名应该尽量用方括号括起来。

3.8.8 关于空值

表达式用 Null 表示空值，例如将折扣列的表达式设为：
IIF([数量] > 1000, 0.1, Null)
意思是：如果数量大于 1000，折扣设为 0.1，否则无折扣。
可以用 Is Null 语句判断某列内容是否为空。
例如在订单表中，金额列的公式可以设为：
IIf([折扣] Is Null, [数量] * [单价], [数量] * [单价] * (1 - [折扣]))
意思是，如果折扣列没有输入内容，那么金额等于：
[数量] * [单价]
如果已经输入折扣，那么金额等于：
[数量] * [单价] * (1 - [折扣])
实际上，上面的公式用 IsNull 函数来编写更为简单：
[数量] * [单价] * (1 - IsNull([折扣], 0))
但是有的时候，我们是没有办法用 IsNull 函数的，只能用 Is Null 语句，例如使用表达式筛选时，需要从订单表中筛选出折扣为空的行，那么筛选表达式为：
[折扣] Is Null
关于筛选，会有专门的章节讲述。
IsNull 函数可以返回列的值，所以多用于计算，而 Is Null 语句返回一个逻辑值，所以多用于条件筛选，关于 Is Null 语句的实际应用，我们将在开发指南中接触到。
如果要判断内容是否不为空，可以加上 Not 关键词，例如：
[折扣] Is Not Null

3.8.9 聚合函数

聚合函数用于数据统计，表达式支持下列聚合函数：
Sum：求和
Avg：平均
Min：最小值
Max：最大值
Count：计数
StDev：标准偏差
Var：方差
例如要对当前表的金额列汇总，公式为：
Sum([金额])
实际上，聚合函数更多的时候是用于统计子表的数据，接下来我们会介绍。

3.8.10 统计子表数据

假定我们在产品表和订单表之间通过产品编号建立关联，而且我们希望在产品表中增加两列，分别统计该产品总的订购数量和金额。

首先我们假定选择产品表，按照前面的讲述，增加一个整数型的表达式列，列名称设为“订购数量”，表达式设为：

Sum(Child.数量)；

然后增加一个双精度小数型的表达式列，列名称设为“订购金额”，计算公式设为：

Sum(Child.金额)；

下图是设置好的产品表：

	产品编号	产品名称	单价	订购数量	订购金额
1	1	HCD238(3)P/TSD	98	3	294
2	2	HCD238(32)P/TSI	65	14	910
3	3	HW238(2)P/TD	200	17	3400
4	4	HA238(8)P/T	25	2	50

所有的过程都很简单，就连表达式也是自我描述性的，对于订购金额列的表达式：Sum(Child.金额)，字面意思就是对子表中的金额列求和。

表达式中的 Sum 就是前面介绍的聚合函数之一。

如此简单的表达式，前提是产品表只和订单表之间建立了关联，所以 Child 很显然就是订单表；如果产品表不仅和订单表之间建立了父子关联，还和其他表建立了父子关联，那么我们就需要修改一下公式：

Sum(Child(po).金额)

还记得"po"是什么吗，是我们在产品表和订单表之间建立的关联的名称，我们用它来明确在有多个子表的情况下，到底统计的是哪一个子表的数据。

即使在只有一个子表的情况下，我们也应该在表达式中明确地指定关联，除非你有足够的把握，以后不会再增加关联了。

3.8.11 引用父表数据

很明显，订单表中的单价数据应该来自产品表中的单价列，所以该列的表达式为：

Parent(po).单价

还记得"po"是什么吗，是我们在产品表和订单表之间建立的关联的名称，我们用它来明确在有多个父表的情况下，到底引用的是哪一个父表的数据。

如果订单表只有一个父表，那么表达式可简写为：

Parent.单价

因为示例文件中的订单表，有两个父表，分别是产品表和客户表，显然这里的 po 是不能省略

的。

重要提示：

如果子表要引用父表的数据，那么对于父表的关联列，应该在列属性设置中，将其属性“禁止重复”设为 True，也就是禁止在父表关联列输入重复内容。因为此时父表和子表之间就必须是严格的一对多关系，理由很明显，如果不是一对多的关系，而是多对多的关系，子表在引用数据的时候，就不知道该引用父表中哪一条记录的数据了，从而出现错误提示。

如果已经遇到这种问题，可以利用重复值筛选找出父表重复行，然后删除之。

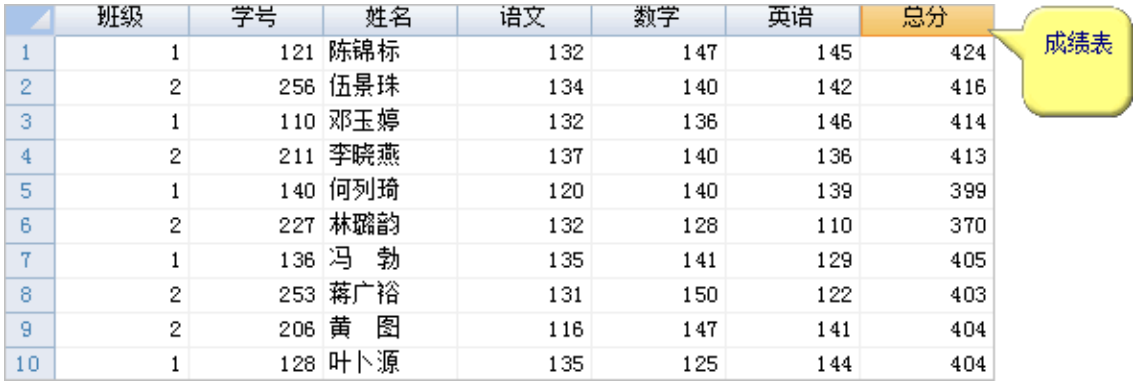
3.8.12 表达式的综合示例一

本节的内容可以参考 CaseStudy 目录下的文件：表达式统计成绩.Table
这个文件有两个表，分别为：



	班级	数学			语文			英语			总分		
		最低分	最高分	平均分	最低分	最高分	平均分	最低分	最高分	平均分	最低分	最高分	平均分
1	1	38	147	105.26	71	135	117.45	44	135	115.6	185	424	338.31
2	2	48	150	111.43	63	137	118.66	33	137	107.52	174	416	337.6

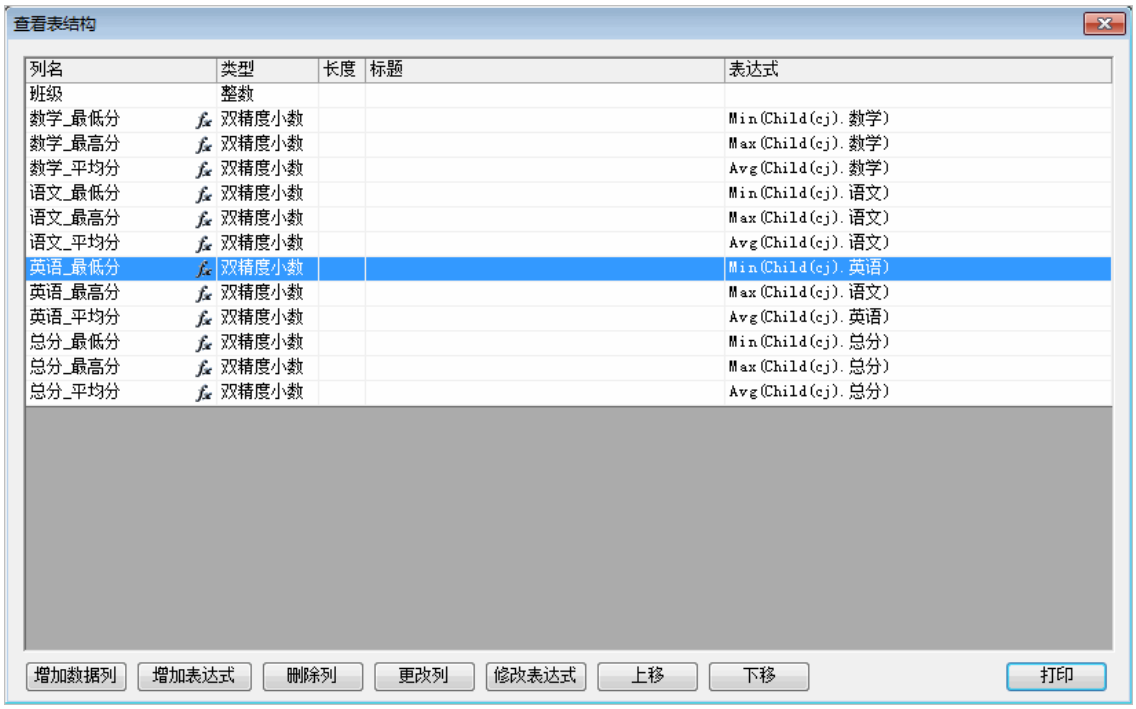
统计表



	班级	学号	姓名	语文	数学	英语	总分
1	1	121	陈锦标	132	147	145	424
2	2	256	伍景珠	134	140	142	416
3	1	110	邓玉婷	132	136	146	414
4	2	211	李晓燕	137	140	136	413
5	1	140	何列琦	120	140	139	399
6	2	227	林璐韵	132	128	110	370
7	1	136	冯　勃	135	141	129	405
8	2	253	蒋广裕	131	150	122	403
9	2	206	黄　图	116	147	141	404
10	1	128	叶卜源	135	125	144	404

成绩表

希望统计表能自动计算得出每一科的最低分、最高分和平均分。
首先两个表通过班级列建立关联，统计表为父表，成绩表为子表，假定关联的名称为"cj"。
成绩表的总分列为表达式列，其表达式为：[语文] + [数学] + [英语]
统计表除班级列外，所有列都是表达式列，各列的表达式参考下图：



3.8.13 表达式的综合示例二

本节内容可以参考 CaseStudy 目录下的文件：关联与表达式的综合示例.Table
这个文件包括三个表：

	产品	初始库存	入库		出库		库存		
			数量	金额	数量	金额	数量	单价	金额
1	PD0	1000	400	8240	420	14400	980	35	34300
2	PD02	1500	1300	21100	1700	45000	1100	25	27500
3	PD03	800	2250	22410	750	11130	2300	15	34500
4	PD04	4000	800	5600	2112	20359.2	2688	10	26880
5	PD05	200	700	14400	500	14500	400	30	12000

库存表

	日期	产品	供应商	数量	单价	金额
1	2011-08-01	PD01	GY01	100	20	2000
2	2011-08-02	PD01	GY01	180	20	3600
3	2011-08-11	PD01	GY03	120	22	2640
4	2011-08-04	PD02	GY04	500	15	7500
5	2011-08-16	PD02	GY04	800	17	13600
6	2011-08-02	PD03	GY02	1200	10	12000
7	2011-08-30	PD03	GY02	150	10	1500
8	2011-09-07	PD03	GY03	900	9.9	8910
9	2011-09-19	PD04	GY02	800	7	5600
10	2011-09-01	PD05	GY05	300	20	6000
11	2011-09-01	PD05	GY05	400	21	8400

入库表

	日期	产品	客户	数量	单价	金额
1	2011-08-01	PD01	CS01	120	35	4200
2	2011-08-03	PD01	CS03	300	34	10200
3	2011-08-01	PD02	CS02	700	25	17500
4	2011-08-24	PD02	CS04	1000	27.5	27500
5	2011-08-17	PD03	CS05	450	15	6750
6	2011-08-13	PD03	CS01	300	14.6	4380
7	2011-09-01	PD04	CS05	1000	10	10000
8	2011-09-13	PD04	CS02	812	9.1	7389.2
9	2011-09-28	PD04	CS04	300	9.9	2970
10	2011-09-16	PD05	CS01	500	29	14500

出库表

我们希望发生出库和入库之后，库存表能够即时统计出各产品的库存数量和金额。

设计步骤

1、首先建立两个关联:

关联名称	父表	关联列	子表	关联列
rk	库存	产品	入库	产品
ck	库存	产品	出库	产品

2、库存表的以下各列为表达式列，其表达式为：

列	表达式
入库_数量	Sum(Child(rk).数量)
入库_金额	Sum(Child(rk).金额)
出库_数量	Sum(Child(ck).数量)
出库_金额	Sum(Child(ck).金额)
库存_数量	IsNull([初始库存],0) + IsNull([入库_数量],0) - IsNull([出库_数量],0)
库存_金额	[库存_数量] * [库存_单价]

3、入库表的金额列为表达式列，其表达式为： [数量] * [单价]

4、出库表的金额列为表达式列，其表达式为： [数量] * [单价]

至此这个简单的进销存系统已经设计完毕，选择某个产品，会列出这个产品所有的出库和入库记录，在出库表和入库表新增行或修改数据，会及时算出所有产品的库存数据：



3.8.14 时段列与表达式

我们知道，双精度小数列可以扩展为时段型。

时段型列用于输入和计算时段，时段列虽然以“时:分:秒 毫秒”格式显示数据，但是其值本质上是一个双精度小数，以秒为单位。
例如“12:30:45”，表示 12 个小时 30 分钟 45 秒，其值为：12 * 3600 + 30 * 60 + 45，也就是 45045。
所以在表达式中使用时段列的时候，必须记得时段列的值是秒，要进行相应的换算。
例如下面是一个打字速度统计表：

	姓名	字数	用时	字数/分
1	岳飞	3200	01:01:30	52
2	刘备	4500	02:02:50	37
3	关羽	600	00:05:10	116
4	孔明	200	00:02:10	92

"字数/分"是一个表达式列，其表达式为： [字数] / [用时] * 60
再例如，下面是一个加班工资计算表：

	姓名	加班时薪	加班小时数	应发加班工资
1	岳飞	50	48:30	2425
2	刘备	60	78	4680
3	关羽	75	50:45	3806.25
4	孔明	45	100:30	4522.5

"应发加班工资"是一个表达式列，其表达式为： [加班时薪] * [加班小时数] / 3600

3.8.15 有条件统计子表数据

首先必须说明的是，通过聚合函数统计子表数据，只能是无条件的。
但是实际应用的时候，有条件的统计是非常普遍的，怎样才能实现条件统计呢？
解决的思路是在子表增加表达式列，计算出需要统计的数据，然后在父表统计子表的这些表达式列。

下面用一个简单的例子来说明。

假定有产品和订单两个表：

	产品编号	产品名称	单价	总销量	总销售额	已收款金额	未收款金额
1	1	HCD238(3)P/TSD	98	12	1176	882	294
2	2	HCD238(32)P/TS	78	14	1092	468	624
3	3	HW238(2)P/TD	200	17	3400	1800	1600
4	4	HA238(8)P/T	25	2	50	50	0

产品表

	产品编号	产品名称	日期	数量	单价	折扣	金额	已付款
1	2	HCD238(32)P/	2008-06-06	6	78	0	468	
2	1	HCD238(3)P/T	2008-06-06	3	98	0	294	
3	3	HW238(2)P/TD	2008-06-07	7	200	0	1400	☐
4	4	HA238(8)P/T	2008-06-08	2	25	0	50	☑
5	3	HW238(2)P/TD	2008-06-08	9	200	0	1800	☑
6	3	HW238(2)P/TD	2008-06-09	1	200	0	200	☐
7	2	HCD238(32)P/	2008-06-09	2	78	0	156	☐
8	2	HCD238(32)P/	2008-06-09	6	78	0	468	☐
9	1	HCD238(3)P/T	2008-06-07	9	98	0	882	☑

订单表

这两个表通过产品编号建立了关联，假定关联的名称是"po"，现在要求产品表能够自动统计出总销量、总销售额、已收款金额、未收款金额。

显然，按照常规的方法，是没有办法统计已收款金额和未收款金额的，但是我们可以绕个弯，在订单表增加一个表达式列，名称为“已付款金额”，将其表达式设置为：

IIF([已付款] = True,[金额],Null)

	产品编号	产品名称	数量	日期	折扣	单价	金额	已付款	已付款金额
1	2	HCD238(32)P/TSD	6	2008-06-06	0	78	468	☑	468
2	1	HCD238(3)P/TSD	3	2008-06-06	0	98	294	☐	
3	3	HW238(2)P/TD	7	2008-06-07	0				
4	4	HA238(8)P/T	2	2008-06-08	0				50
5	3	HW238(2)P/TD	9	2008-06-08	0				1800
6	3	HW238(2)P/TD	1	2008-06-09	0				
7	2	HCD238(32)P/TSD	2	2008-06-09	0	78	156	☐	
8	2	HCD238(32)P/TSD	6	2008-06-09	0	78	468	☐	
9	1	HCD238(3)P/TSD	9	2008-06-07	0	98	882	☑	882

这个表达式列，只是出于统计的需要而存在，实际使用的时候，可以隐藏此列。

现在可以设置产品表中各列的表达式了：

列名	表达式
总销量	Sum(Child(po).数量)
总销售额	Sum(Child(po).金额)
已收款金额	Sum(Child(po).已付款金额)
未收款金额	[总销售额] - [已收款金额]

提示：实际使用的时候，可以将关联表的“已付款金额”列隐藏起来，因为将这一列呈现给最终用户并没有意义。

3.8.16 不重复编号的生成

也许你已经留意到，在表达式生成器中，有一个在表中并未出现的列名：_Identify

_Identify 是实实在在存在的列，只是你看不到而已；新增记录的时候，该列的值会自动增量，步长为 1，但是该值是临时的，保存的时候，才会生成真正的值。

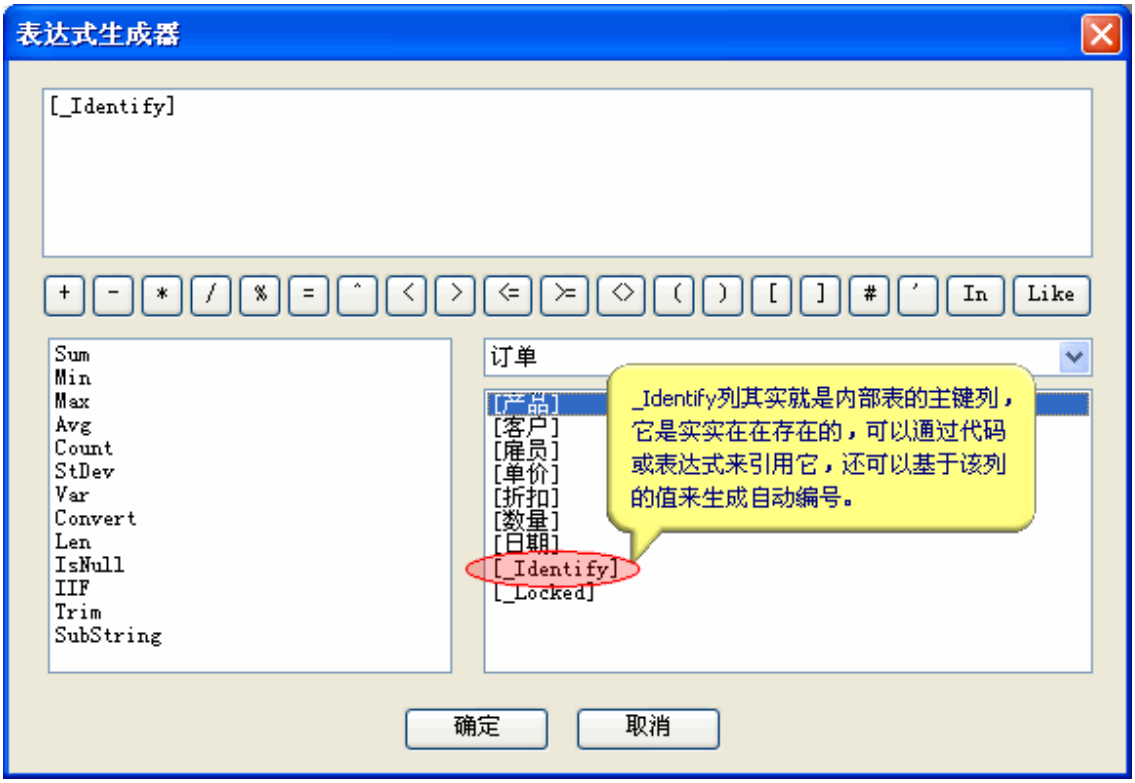
即使多人同时增加和保存数据的时候，该列的值也是不重复的，这个列类似于 Access/Sql Server 的自动增量主键，不同的是，多人同时保存的时候，在 Foxtable 中，每个人得到的编号是连续的，而 Access/Sql Server 是分散的。

有了_Identify，你可以删除原来的产品编号、客户编号、订单编号等列，新增一个同名的整数型表达式列，表达式设为：

[_Identify]

这样不仅编号可以生成，而且不管有多少人同时向表中增加行，生成的编号都是连续的，而且是不重复的。

副作用是，你永远无法修改编号列的内容，不过从数据安全角度来看，也许这样更好！



提示：

1、只有通过 Foxtable 创建的数据表才有_Identify 列。
2、直接用_Identify 作为编号的前提，是你对编号没有特殊的格式要求，否则保留原来的编号列，手工输入编号较好。
3、_Identify 列的值一旦生成，就不能修改，所以如果需要手工调整编号，就不能使用_Identify 作为编号使用了。

3.8.17 自动编号与关联

前面已经提到，由于自动编号列_Identify 的存在，我们可以删除原来的产品编号、客户编号、订单编号等列，新增一个同名的表达式列，表达式设为：

[_Identify]

现在有一个问题，在我们的示例文件中，产品表通过产品编号和订单表建立关联、客户表通过客户编号列和订单表建立关联，但是现在的产品编号、客户编号是表达式列，而表达式列是不能作为关联列使用的，那怎么办呢？
很简单，直接用_Identify 列代替产品编号、客户编号列，作为产品表和客户表的关联列使用。
提示：只有父表才可以用_Identify 和子表建立关联，理由想想就明白了。



需要注意的是，新增行的_Identify 列的内容是临时性的，保存的时候，_Identify 列会重新生成一个值，这个值才是永久性的，所以使用_Identify 作为关联列的时候，务必选中选项“同步更新关联列内容”。
提示：_Identify 是整数型的，所以和其关联的列，必须也是整数型的数据列。

3.8.18 锁定标志列

在表达式生成器中，你还可以看到一个名称为“_Locked”的列，这是一个逻辑列，表示行的锁定状态。
行被锁定的时候，该列的值为 True(是)，否则该列的值为 False(否)。
你可以直接在表达式或者代码中引用该列的值。

3.8.19 数据转换

将表达式转换为指定数据类型。

语法	Convert(expression, type)
参数	expression — 要转换的表达式。 type — 值将转换成的数据类型，包括有： System.String：字符型

	System.DateTime：日期时间型 System.Boolean：逻辑型 System.Byte：微整数 System.Int16：短整数 System.Int32：整数 System.Int64：长整数 System.Single：单精度小数 System.Double：双精度小数 System.Decimal：高精度小数。

所有转换都是有效的，只有下列情况例外：Boolean 只能与 Byte、SInt16、Int32、Int64、String 和它本身相互转换；DateTime 只能与 String 和它本身相互转换。

例如：

Convert([金额], 'System.Int32')

表示将金额数据转换为整数。

3.8.20 如何实现四舍五入

表达式并没有提供传统的四舍五入函数 Round，我们只能用 Convert 函数来间接实现。

例如金额列原来的计算公式为:

[数量] * [单价] * (1 - [折扣])

为了保留两位小数，可以将公式改为：

Convert([数量] * [单价] * (1 - [折扣]) * 100, 'System.Int64') / 100

原理很简单：将计算结果乘以 100，然后取整，最后除以 100。

如果你要保留三位小数，将表达式中的 100 改为 1000 即可，其余类推。

3.8.21 如何计算年月日

表达式并没有提供根据日期计算出年月的函数，但是就像四舍五入一样，我们可以用间接的方式计算得出。

下面的表达式看上去有点复杂，不需要刻意去理解他，实际使用的时候直接套用即可。

由于 Windows 7/Vista/XP 默认的日期格式不同，所以表达式也有所不同。

计算年的表达式

SubString(Convert([日期],'System.String'),1,4)

计算月的表达式

Windows XP:

SubString(Convert([日 期],'System.String'),6,IIF(SubString(Convert([日 期],'System.String'),7,1) = '-',1,2))

Windows 7/Vista:

SubString(Convert([日 期],'System.String'),6,IIF(SubString(Convert([日 期],'System.String'),7,1) = '/',1,2))

通用(适合 Windows XP/Vista/7):

SubString(Convert([日期],'System.String'),6,IIF(SubString(Convert([日期],'System.String'),7,1) In ('-','/'),1,2))

计算日的表达式

Windows XP:

SubString(Convert([日期],'System.String'),IIF(SubString(Convert([日期],'System.String'),7,1) = '-',8,9),2)

Windows 7/Vista:

SubString(Convert([日期],'System.String'),IIF(SubString(Convert([日期],'System.String'),7,1) = '/',8,9),2)

通用(适合 Windows XP/Vista/7):

SubString(Convert([日期],'System.String'),IIF(SubString(Convert([日期],'System.String'),7,1) In ('-','/'),8,9),2)

CaseStudy 目录下有一个示例文件“表达式求年月日.table”，其中年、月、日列的值就是用上述表达式计算得出，大家可以打开这个文件参考一下。

提示：上述表达式只对简体中文的操作系统有效，且不能修改控制面板中区域和语言选项的日期时间格式。

3.8.22 计算功能太弱?

因为表达式的函数有限，对于一些较为复杂的计算要求，有时会显得无能为力。

其实 Foxtable 有两种方式来实现计算和统计，一种是利用本章所讲述的表达式，另一种是利用代码；表达式使用简单，计算速度快，即使 10 万行的计算也可以瞬间完成，而且不占据存储空间，只是在运行的时候生成计算结果，但是只能使用函数，且支持的函数有限；而 用代码计算具备无比的灵活性，让您能随心所欲地计算，而且设计得当的话，速度同样非常快，不过使用上要比表达式复杂一些。

两种方式各有优点，很难说谁更有优势。

关于如何使用代码进行计算，会在开发篇详细讲述。

3.8.23 数据无法保存?

在 Foxtable 的测试阶段，经常有人提问：为什么某些列的数据无法保存。

绝大多数时候，都是同一个原因造成的：这些列是表达式列，而表达式列的内容是不会保存的。

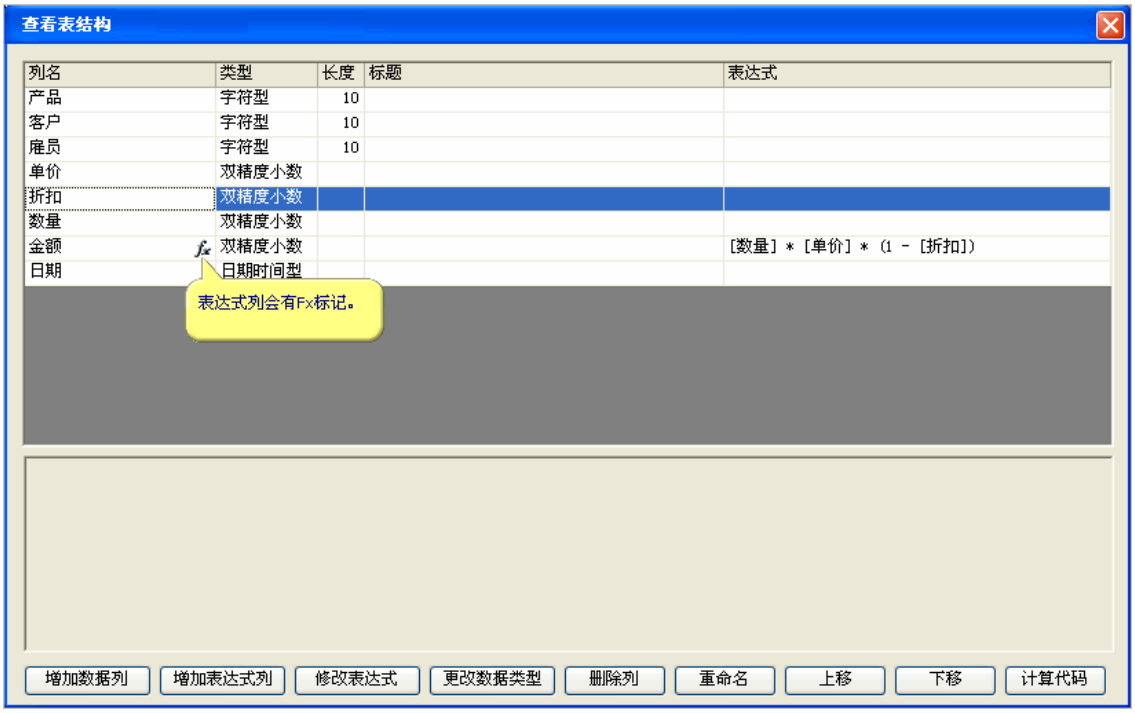
可是不少用户会反驳：我这个列不是表达式列，这一列没有设置表达式，列中的数据是我手工输入或用代码计算得出的啊。

其实判断某列是否是表达式列，是不能用是否设置了表达式或是否能输入内容来判断的，判断的原则很简单，只需选择该列，然后单击下面的按钮：

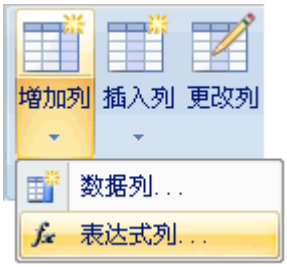


如果该列能够设置表达式，说明该列就是表达式列，其内容肯定是不能保存的。

此外在设计表窗口，也能清楚地看出哪些列是表达式列：



本该是数据列，却阴差阳错地成了表达式列，导致这种情况发生的原因是：在增加数据列的时候，本应该单击“数据列”命令，却误单击了“表达式列”按钮：



另一种不能保存的情况我们只遇到过一次，原因更简单：用户将项目文件的属性设置成只读了。

3.8.24 美化自动编号

前面已经提到，由于自动编号列_Identify 的存在，我们可以删除原来的产品编号、客户编号、订单编号等列，新增一个同名的整数型表达式列，表达式设为：

[_Identify]

不少人会希望，编号能够按照下面的格式显示：

0001

0002

0003

....

要实现这样的格式很简单，只需在列属性设置中，将该列的“固定整数位数”设为 4 即可，你需要多少位都可以。

有的时候我们对于编号的格式会有更多的要求，例如要求编号为：

MP0001

MP0002

MP0003

...

MP0123

MP0124

...

MP1000

MP1001

编码的规则是：前面是两个字母，后接 4 个数字，不够的时候自动补充 0。

为实现这个目的，可以将编号列的表达式改为：

```
'MP' + SubString('0000', 1, 4 - Len(Convert([_Identify],'System.String'))) + Convert([_Identify], 'System.String')
```

实际上，上面的表达式是通用的，如果要求后接 6 个数字，不够的时候自动补充 0，表达式为：

```
'MP' + SubString('000000', 1, 6 - Len(Convert([_Identify],'System.String'))) + Convert([_Identify], 'System.String')
```

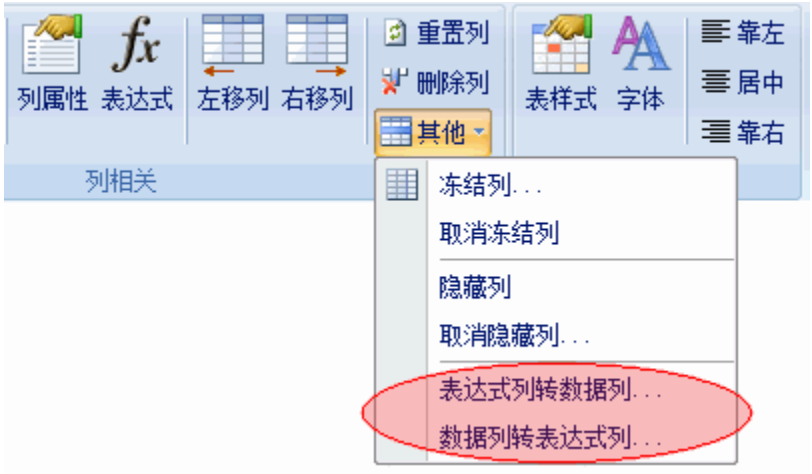
> 基于_Identify 生成的编号，大概也只能做成这样了，如果你对编号有特殊的格式要求，那么还是保留原来的编号列，手工输入编号较好。

3.8.25 表达式列和数据列的相互转换

什么时候使用表达式列，什么时候使用数据列，并没有一个明确的规则。

所以在开发项目的过程中，我们难免会有反复，需要将数据列转换为表达式列，反之亦然。

Foxtable 的菜单提供了两个命令，可以很方便地进行这种转换。



提示：

1、转换结束后会自动重新打开项目。

2、数据列转表达式列成功后，请设置其表达式。

3.9 行相关

3.9.1 增加行

在“日常工作”功能区的“数据”功能组，有一个“增加行”按钮：



这是一个组合按钮，单击按钮的上半部分，可以增加一行，单击按钮的下半部分，会出现一个菜单，用以选择要增加一行，还是要增加多行。
有两个表属性和增加行有关：

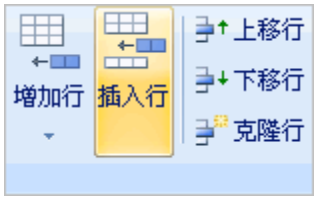
- 允许增加行
  此属性默认为 True，如果设为 False，将禁止增加行。

- 自动增加行
  此属性默认为 False，如果设为 True，在最后一行的最后一列按回车键时，将自动增加一行。

提示：增加行的快捷键是“Ctrl + A”，同时按这两个按键，即可增加一行。

3.9.2 插入行

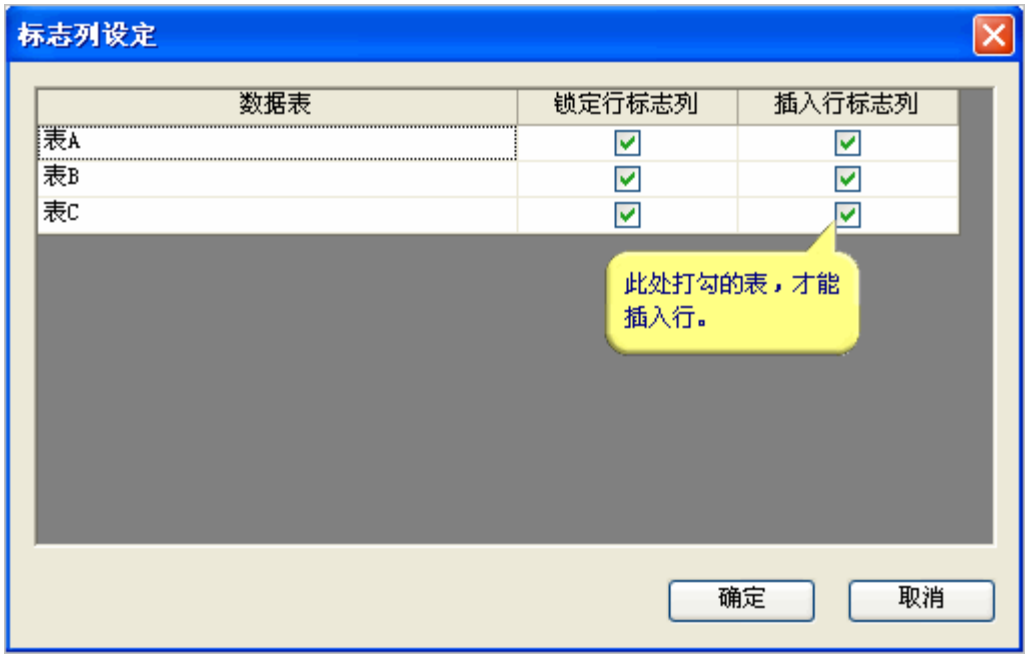
在“日常工作”功能区的“数据”功能组，有一个“插入行”按钮：



单击此按钮，可以在当前位置插入一行。
在默认情况下，只有内部表才能插入行，而外部表是没有插入行功能的。
不过不管是外部表还是内部表，都可以开启和关闭插入行功能；在菜单的“数据表”功能区的“表相关”功能组，单击“设置标志列”按钮：



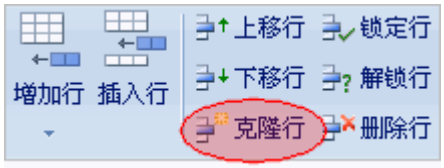
会出现下图所示窗口，可以开启和关闭插入行功能：



如果某个表不需要使用插入行的功能，那么最好关闭其插入行功能，以节省资源。

3.9.3 克隆行

在“日常工作”功能区的“数据”功能组，有一个“克隆行”按钮：



单击此按钮，会新增一行，新增行各列内容和当前行完全一样。

3.9.4 删除行

在“日常工作”功能区的“数据”功能组，单击下面的“删除行”按钮，可以删除选定行：

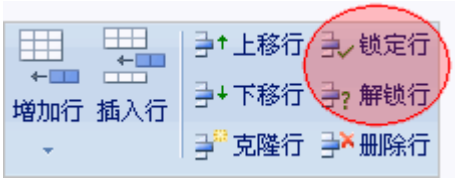


对于内部数据表，如果刚刚删除了大量的行，最好执行一次压缩项目命令。

如果在表属性设置中，将“允许删除行”属性设为 False，将禁止用户删除行。

3.9.5 锁定行

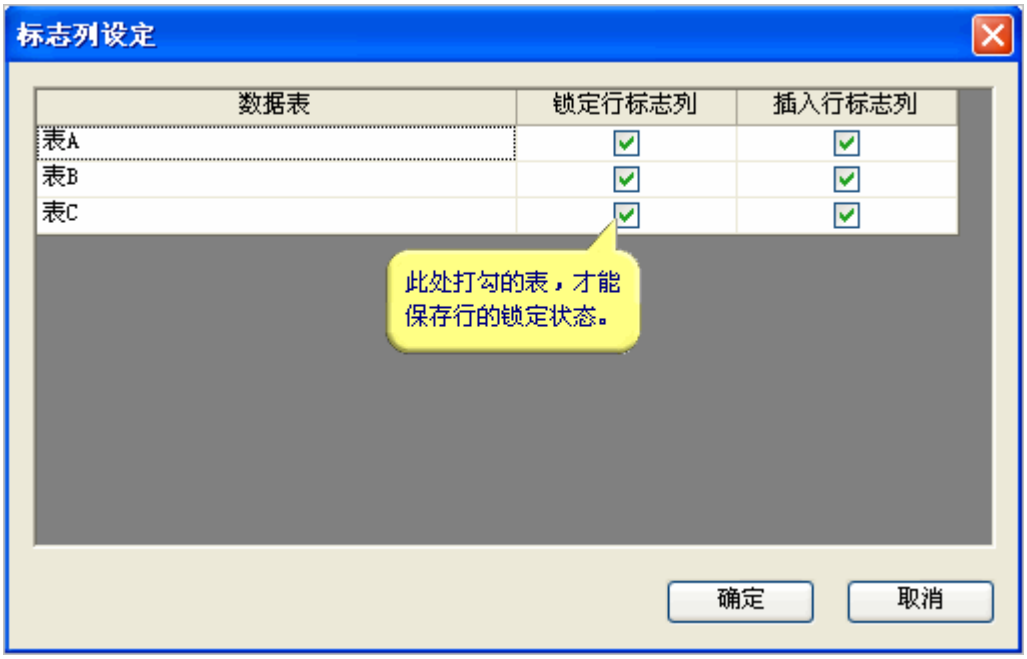
所谓锁定行，就是将行设为只读状态，既不能被编辑，也不能被删除。
锁定行和取消锁定行的按钮，位于“日常工作”功能区的“数据”功能组：



表属性中有“允许锁定行”和“允许取消锁定行”两个属性，其作用是不言而喻的。
在默认情况下，只有内部表的锁定状态才能保存，而外部表在重新打开项目后，行会重新回到非锁定状态。
不过不管是外部表还是内部表，都可以开启和关闭锁定状态保存功能；在菜单的“数据表”功能区的“表相关”功能组，单击“设置标志列”按钮：



会出现下图所示窗口，可以开启和关闭锁定状态保存功能：



如果某个表不需要保存行的锁定状态，那么最好关闭此功能，以节省资源。

3.9.6 排序行

和排序相关的按钮，位于“日常工作”功能区的“排序与筛选”功能组：



单列排序

选择要排序的列，然后单击“升序”或“降序”按钮。

或者按住 Ctrl 键，然后单击某列的列标题，即可根据该列排序。

多列排序

如果是连续的多列，那么选定这些列，然后单击“升序”或“降序”按钮。

如果是非连续的多列，那么根据排序顺序的优先顺序，从低到高逐列排序即可。

例如对于订单表，希望根据产品排序，在产品相同的情况下，根据客户排序，而且产品列和客户列中间还有其他的列，那么排序操作的流程为：

1、先选择客户列，单击“升序”或“降序”按钮。

2、然后选择产品列，单击“升序”或“降序”按钮。

取消排序

有两个按钮可以用于取消排序，一是单击上图中的“取消”按钮，二是单击上图中的“切换”按钮。

如果单击“切换”按钮取消排序，那么再次单击该按钮，会回到排序状态。

排序标记

排序列标题的右侧，会出现一个箭头，下箭头表示升序，上箭头表示降序：

	产品 ↓	客户 ↑	雇员	单价	折扣
1	PDO1	CSO5	EPO5	18	0
2	PDO1	CSO5		4.4	0
3	PDO1	CSO5		18	0
4	PDO1	CSO5		18	0
5	PDO1	CSO5	EPO3	18	0.2
6	PDO1	CSO5	EPO2	18	0
7	PDO1	CSO5	EPO3	18	0

不同方向的箭头，表示不同的排序方式。

如果不希望显示排序标记，可以在表样式设置中，将“显示排序标记”属性设为 False(否)。

3.9.7 调整行位置

在“日常工作”功能区的“数据”功能组，有两个按钮用于调整行的位置：



提示：

1、只有开启插入行功能后，才可以调整行的位置。

2、不能在排序状态下调整行位置。

3.9.8 筛选行

3.9.8.1 本章示例文件

本章课程使用的示例文件为“筛选演示.Table”，位于安装目录的子目录“CaseStudy”下。
该文件就一个表：

	产品	客户	雇员	单价	折扣	数量	金额	日期	已付款
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02	☑
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03	☑
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03	☐
4	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04	☑
5	PD03	CS03	EP04	10	0	490	4900	1999-01-04	☑
6	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07	☐
7	PD05	CS03	EP04	17	0	160	2720	1999-01-07	☐
8	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08	☑
9	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08	☑

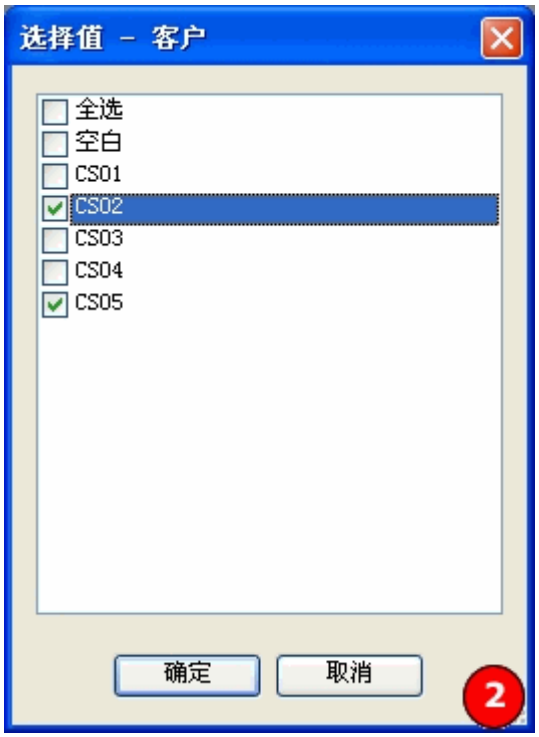
请在学习本章内容之前，先打开此文件。

3.9.8.2 按值筛选

1、如果你希望筛选出某几个客户的订货记录，可先选择客户列，单击“日常工作”功能区“排序与筛选”功能组中的“选择”按钮：



2、出现“选择值”窗口，选择你要筛选的客户，例如下图选择了两个客户，分别是“CS02”和“CS05”：



3、单击“确定”按钮，即可筛选出 CS02 和 CS05 客户的所有订单：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08
2	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
3	PD04	CS05	EP04	22	0.05	500	10450	1999-01-11
4	PD02	CS05	EP04	19	0.1	200	3420	1999-01-13
5	PD02	CS05	EP04	15.2	0	400	6080	1999-01-14
6	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
7	PD05	CS05	EP02	21.35	0	60	1281	1999-01-22
8	PD03	CS02	EP05	8	0.25	700	4200	1999-01-26
9	PD03	CS02	EP03	10	0.1	200	1800	1999-01-28
10	PD01	CS05	EP05	18	0	200	3600	1999-02-01
11	PD05	CS02	EP03	17	0	150	2550	1999-02-01

4、如果你希望得到 CS02 和 CS05 客户订购产品 PD01 的订单，可以在完成上述筛选后，选择产品列，再次单击菜单中的“选择”按钮，在“选择值”窗口中，选择“PD01”即可。

提示：如果要筛选空白内容，例如筛选客户列没有输入内容的行，在“选择值”窗口选择“空白”值即可。

3.9.8.3 智能筛选

3.9.8.3.1 关于智能筛选

智能筛选的关键是下图中的“筛选”按钮：



这是一个非常“聪明”的按钮，单击该按钮会出现一个下拉菜单，该菜单的命令不是固定的，会随着选定列的数据类型、选定单元格的内容不同，而出现不同的筛选命令。

接下来我们会讲述选择不同数据类型的列，该按钮会出现哪些不同的命令，是如何实现“智能”筛选的。

3.9.8.3.2 字符列筛选

假定我们在客户列中，选择一个单元格，单元格的内容为“CS03”，单击“筛选”按钮，出现的下拉菜单为：



1、如果你要筛选出“CS03”客户的所有订单，那么单击“等于 CS03”即可。

2、如果你要筛选出非“CS03”客户的所有订单，那么单击“不等于 CS03”即可。

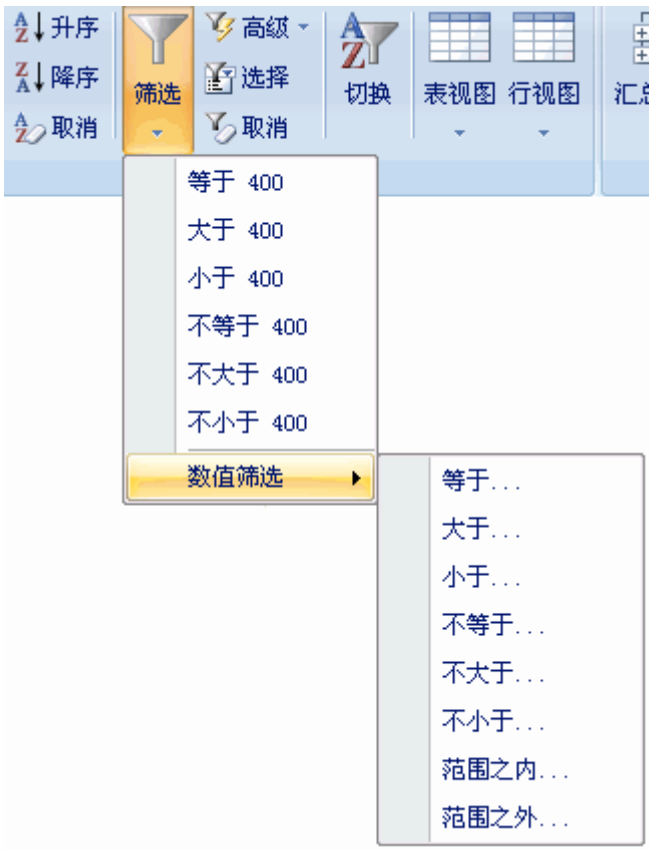
3、指向“文本筛选”，会列出很多命令，例如等于、包含、类似等等，几乎囊括了任何可能的文本筛选命令。

4、例如你要筛选的是客户名称中包括“联想”二字的记录，可以单击“包含”命令，出现下图所示的窗口，输入“联想”，单击“确定”即可：



3.9.8.3.3 数值列筛选

假定我们在数量列中，选择一个单元格，单元格的内容为 400，单击“筛选”按钮，出现的下拉菜单为：



1、首先是一些和选定单元格的值进行比较的命令，例如你要筛选出订货数量超过 400 的订单，单击命令“大于 400”即可。

2、指向数值筛选，会列出很多命令，例如等于、小于、大于等等，几乎囊括了所有可能的数值筛选命令。

3、 例如你要筛选出订货数量在 600 到 800 之间的订单，可以单击命令“范围之内”，出现下图所示的窗口，分别输入起始值和终止值，单击“确定”即可：



3.9.8.3.4 日期列筛选

假定我们在日期列中，选择一个单元格，单元格的内容为“1999-01-04”。

1、单击“筛选”按钮，首先出现的是一些和选定单元格的日期进行比较的命令，假如你要筛选出1999 年 1 月 4 日的所有订单，单击命令“等于 1999-01-04”即可。



2、指向“指定时段”命令，会出现下图所示的菜单，利用这些命令，你可以筛选出指定月份或季度的数据，例如你要筛选出 3 月份的所有订单，只需单击命令“三月”即可：



3、指向“相同时段”命令，会出现下图所示的菜单，利用这些命令，你可以筛选出和选定行处于同一时段范围内的所有行,例如你要筛选出和 1999-01-04 这一天属于同一周的订单,单击命令“同周”即可。



4、指向“日期筛选”，会列出很多命令，例如等于、早于、晚于等等，囊括了所有可能的日期筛选命令：



例如你要筛选 1999-02-01 到 1999-02-10 这一日期范围内的订单，只需单击“范围之内”命令，输入起始日期和终止日期，单击“确定”即可：



3.9.8.3.5 逻辑列筛选

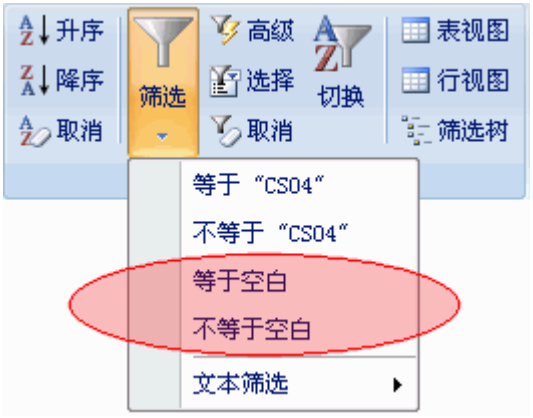
"已付款"是一个逻辑列，在该列中选择一个单元格，单击“筛选”按钮，出现的下拉菜单为：



1、单击命令“选中”，将筛选出所有已经付款的订单。
2、单击命令“未选中”，将筛选出所有未付款的订单。

3.9.8.3.6 空值的筛选

选择任何一个单元格(逻辑列除外)，然后单击“筛选”按钮，出现的下拉菜单会有用于筛选空值的命令：



1、单击命令“等于空白”，将筛选出选定列内容为空的记录。
2、单击命令“不等于空白”，将筛选出选定列内容不为空的记录。

3.9.8.4 筛选树

3.9.8.4.1 筛选树

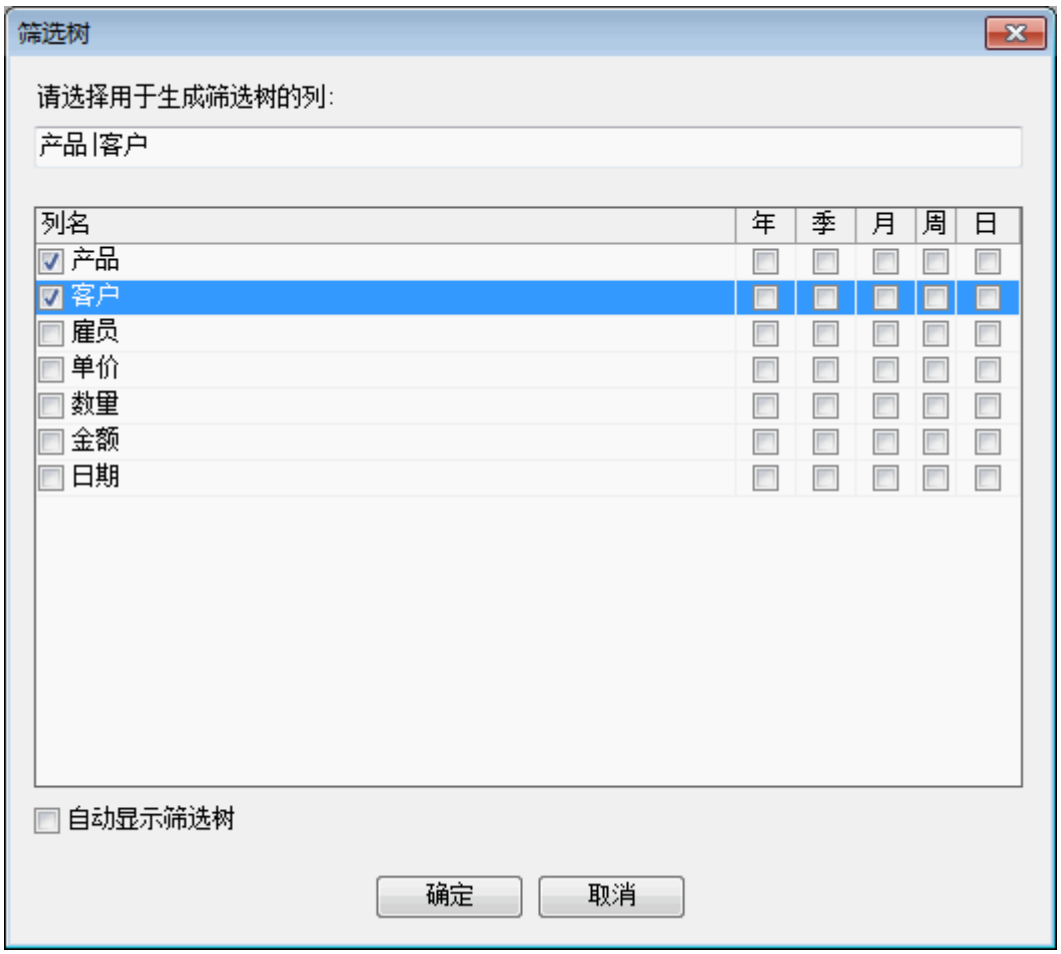
筛选树能够以目录的形式快速筛选出所需要的数据，是所有筛选方式中最为方便的。
在菜单的“日常工作”区的“排序与筛选”功能组中，单击下面的按钮，可以打开筛选树：



提示：
1、如果筛选树已经关闭，希望按照上次设置重新打开筛选树，那么执行“显示筛选树”命令。
2、如果希望重新设置并打开筛选树，那么执行“设置筛选树”命令。

示例

例如对于订单表，例如按照下图所示设置筛选树：



即可显示下图所示的筛选树，我们只需单击对应的产品或客户，即可筛选出相关数据。

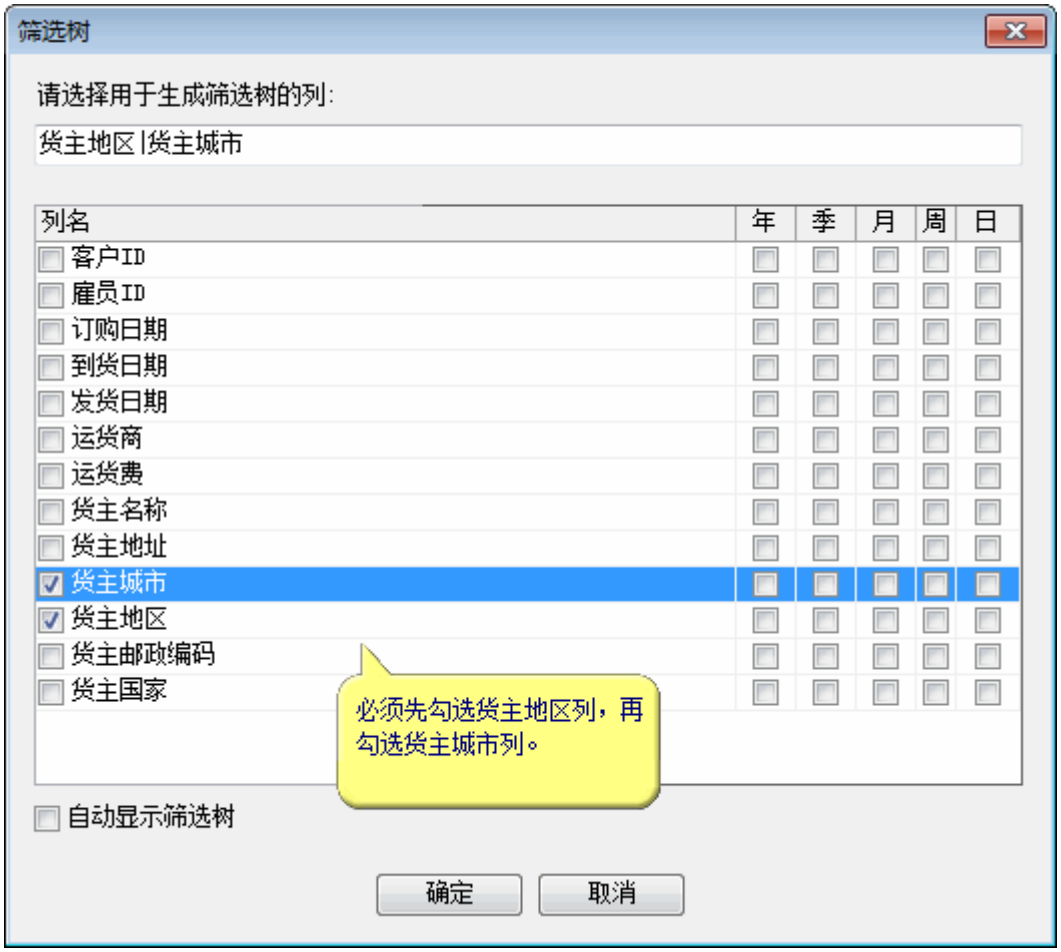


	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
2	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
3	PD01	CS02	EP05	14.4	0.15	150	1836	1999-02-22
4	PD01	CS02	EP03	14.4	0	200	2880	1999-02-26
5	PD01	CS02	EP02	18	0	100	1800	1999-03-04
6	PD01	CS02	EP01	14.4	0	700	10080	1999-03-17
7	PD01	CS02	EP03	18	0	400	7200	1999-03-19
8	PD01	CS02	EP02	14.4	0	60	864	1999-04-03
9	PD01	CS02	EP03	18	0.25	600	8100	1999-04-15
10	PD01	CS02	EP01	14.4	0.05	160	2188.8	1999-04-17
11	PD01	CS02	EP01	18	0	100	1800	1999-05-07
12	PD01	CS02	EP03	14.4	0.2	450	5184	1999-05-28
13	PD01	CS02	EP01	18	0	300	5400	1999-06-30

例如在上图中，要得到 CS01 客户订购产品 PD01 的订单，只需单击节点"PD01\CS01"即可。

注意勾选顺序

在设置目录树的时候，必须注意勾选列的顺序，首先要勾选大的分组列，然后再勾选小的分组列。例如在下面的窗口中，尽管货主城市列位于货主地区列之前，但地区是大的分组列，例如东北地区会包括很多个城市，所以必须首先勾选货主地区列，然后再勾选货主城市列：



这样才能正确地生成筛选树：



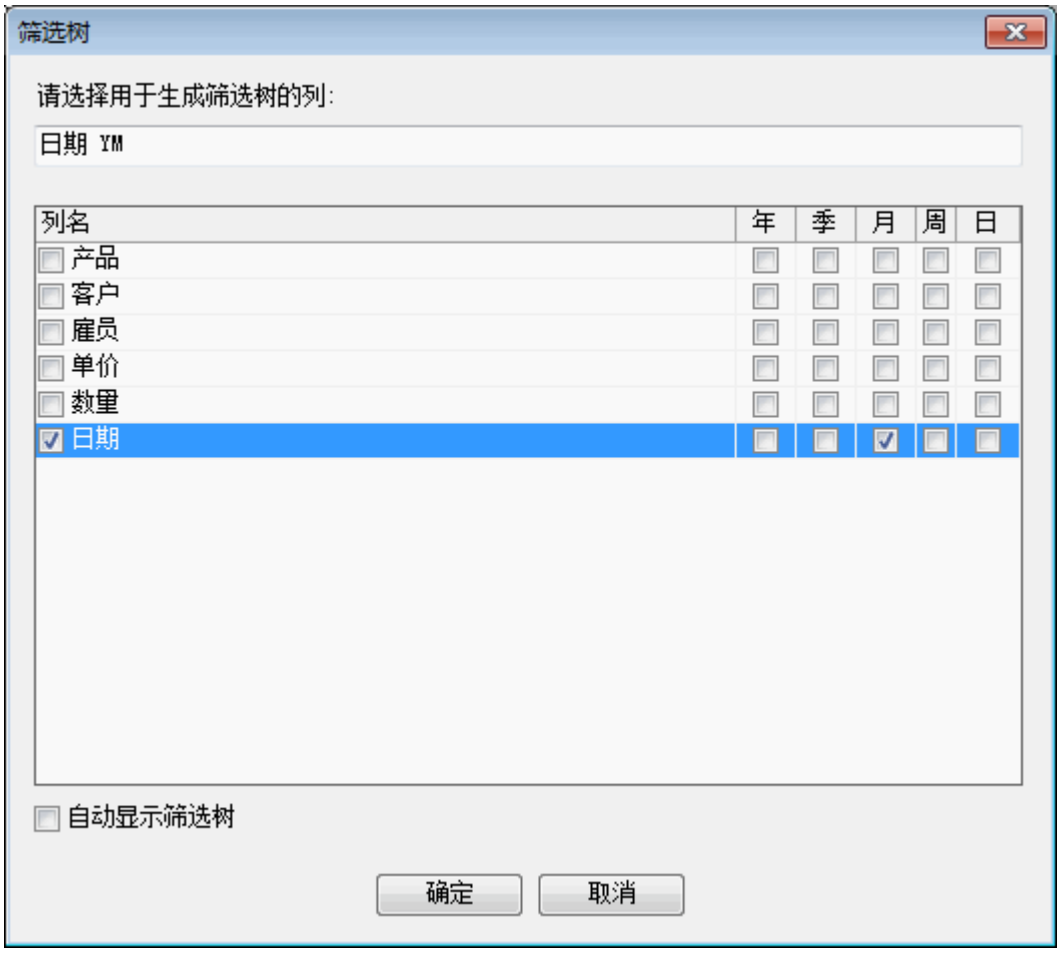
保存设置

筛选树的设置和宽度是可以随数据保存的，所以设置好筛选树及其宽度后，单击保存按钮，下次打开项目后，只需执行菜单命令“显示筛选树”，即可按照之前的设置和宽度打开筛选树，无需重复设置。

3.9.8.4.2 按时段筛选

在设置筛选树的时候，对于日期型列，提供了年、季、月、周、日五种分组方式。

例如对于订单表，按下图所示设置筛选树：



即可按月分组生成筛选树：



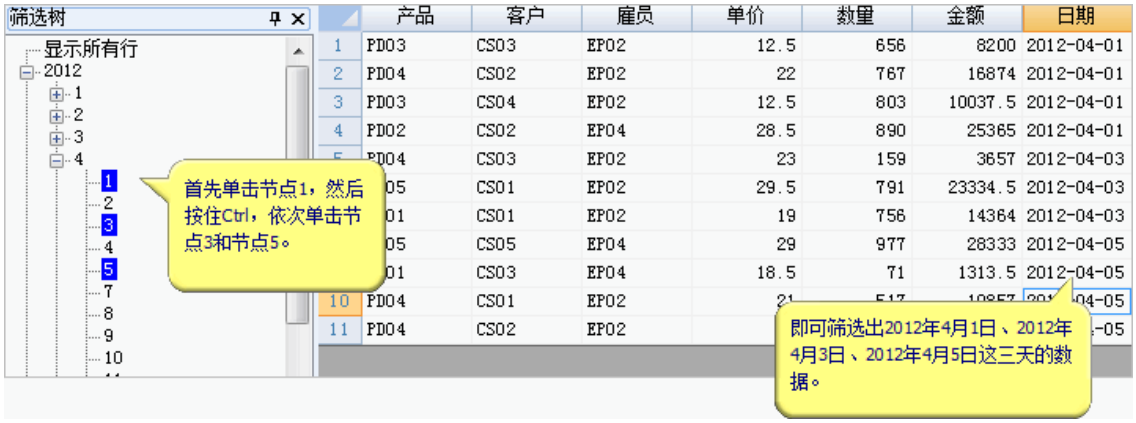
例如在上图中，要筛选 2010 年 1 月份的订单，只需单击节点“2010\1”即可。

3.9.8.4.3 筛选树可以多选

在默认情况下，单击筛选树某个节点，可以筛选出此节点对应的数据。

我们还可以同时筛选出多个节点下的数据，只需按住 Ctrl 键，依次单击各节点即可。

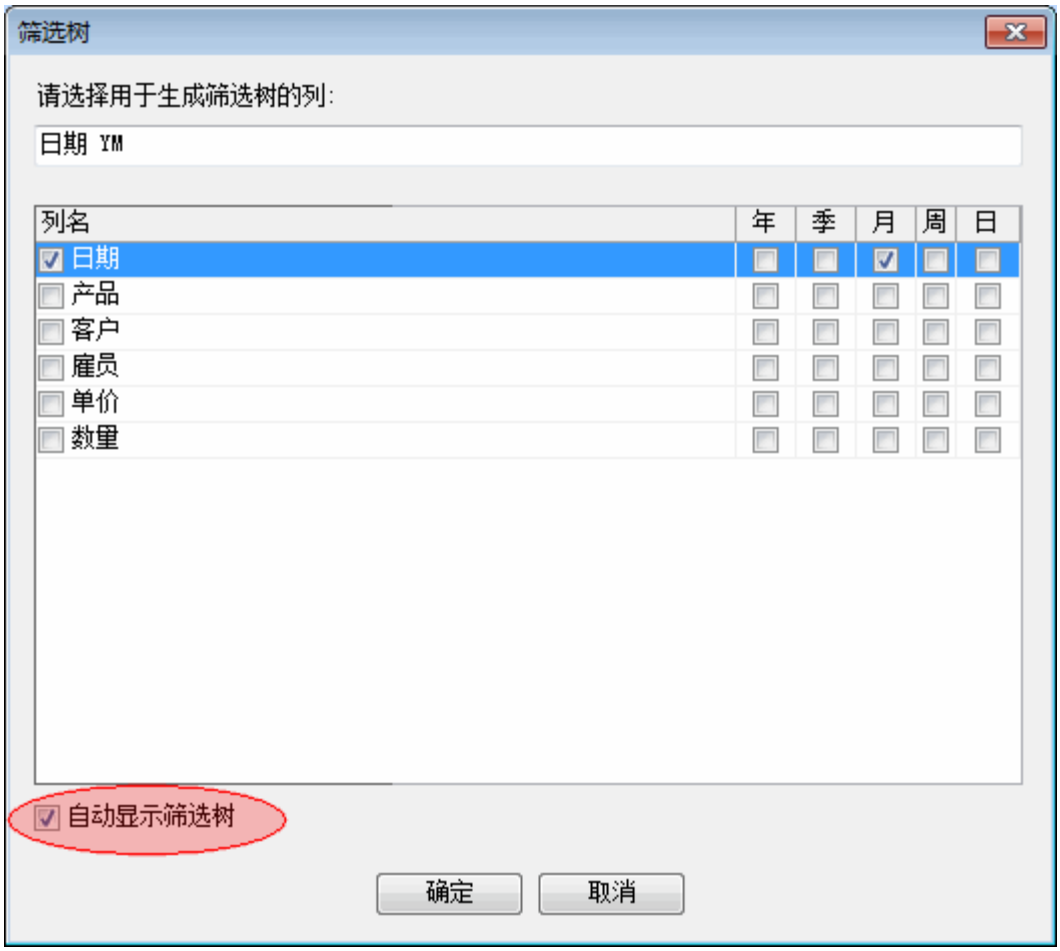
例如下图所示的筛选树，如果要筛选出 2012 年 4 月 1 日、2012 年 4 月 3 日、2012 年 4 月 5 日这三天的数据，可以先展开节点"2012\2"，显示这个月的所有日期，然后直接单击节点 1，接着按住 Ctrl 键，依次单击节点 3 和节点 5 即可：



	产品	客户	雇员	单价	数量	金额	日期
1	PD03	CS03	EP02	12.5	656	8200	2012-04-01
2	PD04	CS02	EP02	22	767	16874	2012-04-01
3	PD03	CS04	EP02	12.5	803	10037.5	2012-04-01
4	PD02	CS02	EP04	28.5	890	25365	2012-04-01
	PD04	CS03	EP02	23	159	3657	2012-04-03
	05	CS01	EP02	29.5	791	23334.5	2012-04-03
	01	CS01	EP02	19	756	14364	2012-04-03
	05	CS05	EP04	29	977	28333	2012-04-05
	01	CS03	EP04	18.5	71	1313.5	2012-04-05
10	PD04	CS01	EP02	21			04-05
11	PD04	CS02	EP02				-05

3.9.8.4.4 自动显示筛选树

如果希望打开项目后，某个表能自动显示筛选树，可以在设置筛选树的时候，选择选项“自动显示筛选树”：



3.9.8.5 重复值筛选

在任何时候，单击“排序与筛选”功能组中的“高级”按钮，在出现的下拉菜单中，指向“重复筛选”，会出现下图所示的菜单：



该菜单包括三个项目

显示重复值

如果当前列中某一个单元格的内容是唯一的，那么该单元格所在行将被隐藏。

也就是说单元格的内容至少在当前列中出现两次，该单元格所在的行才会显示。

排除重复值

如果当前列中有多个单元格具备某一相同的值，将只显示其中一个单元格所在的行。

显示唯一值

如果当前列某一单元格的值是唯一的，不存在重复，那么该单元格所在的行将显示，否则被隐藏。

显示冗余值

如果当前列中有 n(n>1)个单元格具备某一相同的值，将只显示其中 n-1 行。

例如表的初始状态：

	第一列	第二列
1	1	A
2	2	B
3	1	C
4	2	D
5	1	E
6	2	F
7	3	G
8	4	H

显示第一列的重复值（只显示存在重复的值，"3"和"4"不重复，所以不显示）：

	第一列 ↓	第二列
1	1	A
2	1	C
3	1	E
4	2	B
5	2	D
6	2	F

排除第一列的重复值（每个值不管是否有重复，都显示一个）：

	第一列 ↓	第二列
1	1	A
2	2	B
3	3	G
4	4	H

显示第一列的唯一值（"3"和"4"不重复，所以显示，"1"和"2"有重复，所以不显示）：

	第一列	第二列
1	3	J
2	4	K

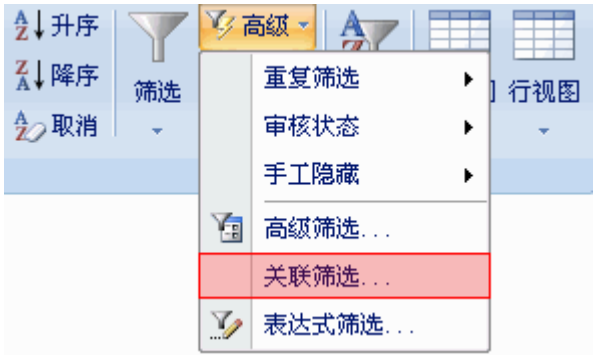
显示第一列的冗余值 （"1"和"2"都有 3 个重复值，显示其中 2 个冗余的，我们可以用这个方法找出冗余数据后删除之，剩下的就是唯一的、非冗余的）：

	第一列 ↓	第二列
1	1	C
2	1	E
3	2	D
4	2	F

3.9.8.6 关联筛选

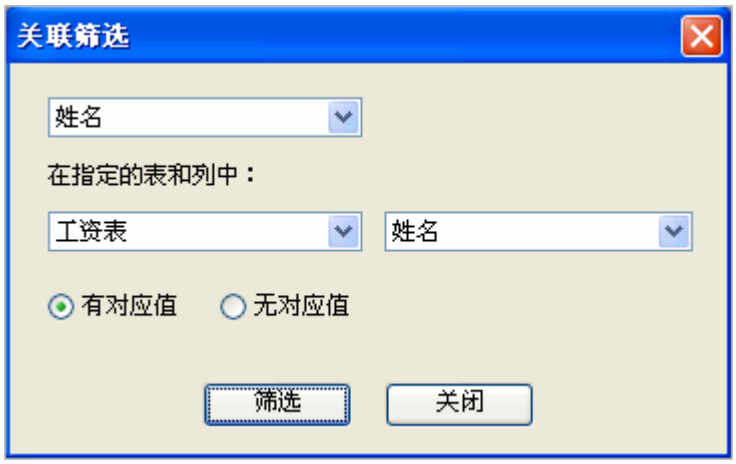
在表与表之间建立关联后，一个表中的每条记录在关联表中也应该有对应的一条记录，但是由于数据输入错误或其他原因，某些记录可能在关联表中没有对应记录。为了快速找出这些“失联”记录，可以 使用关联筛选。

在任何时候，单击“排序与筛选”功能组中的“高级”按钮，在出现的下拉菜单中，单击“关联筛选”：



将出现关联筛选窗口。

例如在人事管理中，在基本信息表中按下图所示进行关联筛选，即可快速找出在基本信息表中有记录而在工资表中没有记录的员工：



提示：不仅是关联表，任何存在一定关系的表，都可以进行这种关联筛选。

3.9.8.7 锁定状态筛选

单击“排序与筛选”功能组中的“高级”按钮，在出现的下拉菜单中，指向“锁定状态”，出现的菜单如下：

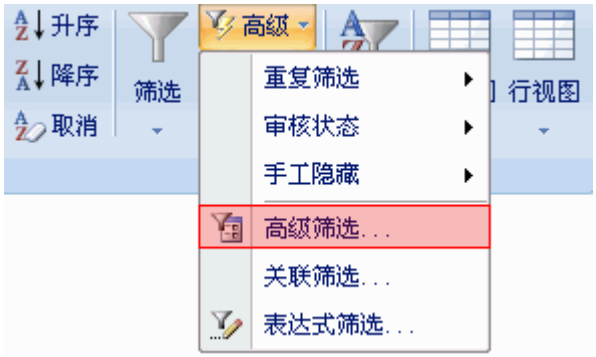


1、单击命令“已锁定行”，将筛选出所有已经锁定的行。

2、单击命令“未锁定行”，将筛选出所有未锁定的行。

3.9.8.8 高级筛选

1、单击“排序与筛选”功能组中的“高级”按钮，在出现的下拉菜单中，单击“高级筛选”：



2、将出现高级筛选窗口，高级筛选可以同时对多列设置条件进行筛选，例如下图是筛选产品为“PD01”且客户为“CS01”的订单：



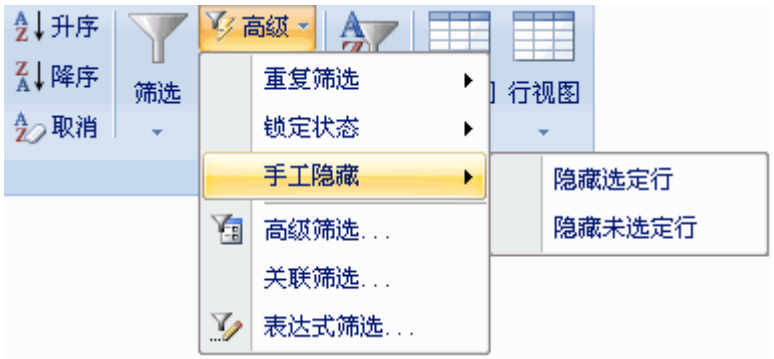
3、条件之间的连接有两种方式，分别为 And、Or，如果连接方式为 And，记录必须同时满足所连接的条件，才算符合条件；如果连接方式为 Or，记录只需满足所连接条件中的任何一个，就算满足筛选条件。

4、通过下图所示的按钮，可以保存高级筛选设置，并在需要的时候调用之前保存的设置：



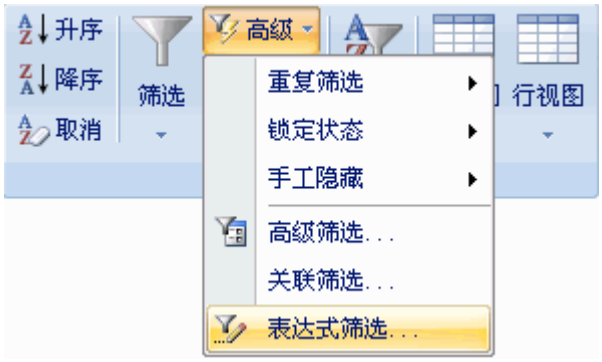
3.9.8.9 手工隐藏行

通过下图所示的菜单命令，可以手工隐藏行：

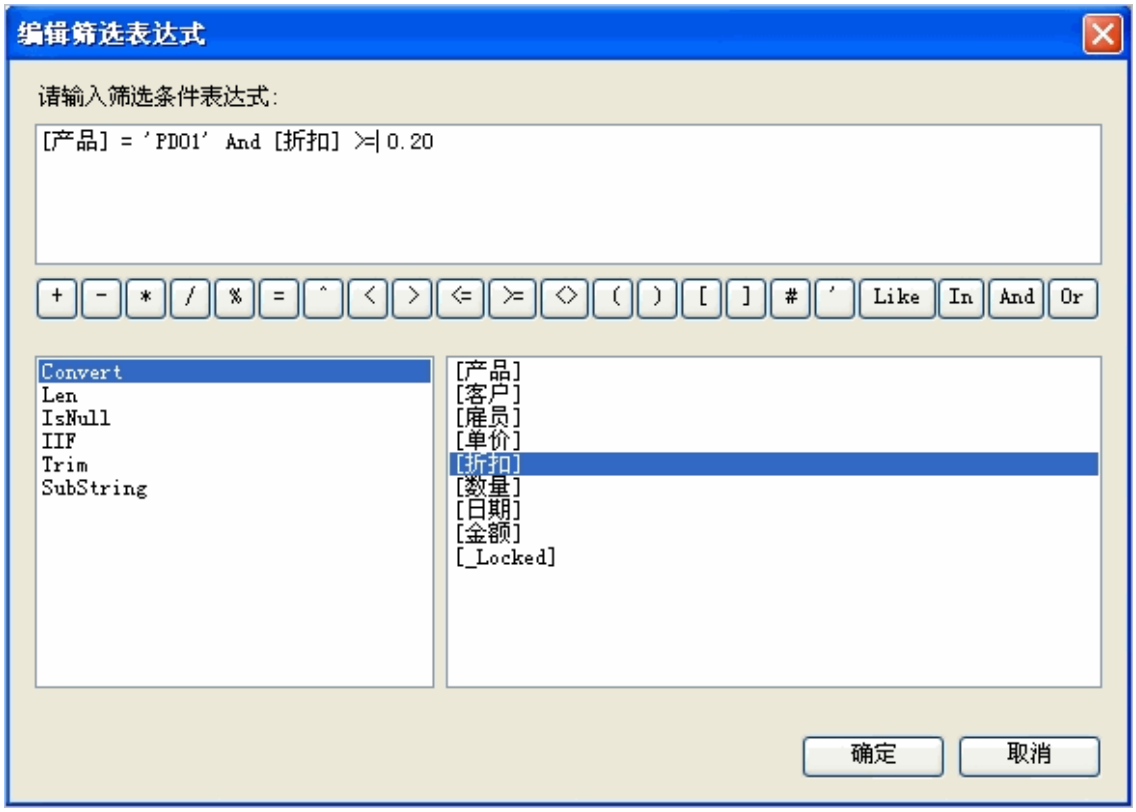


3.9.8.10 表达式筛选

1、单击“日常工作”功能区“排序与筛选”功能组中，单击下面的按钮：



2、出现表达式编辑窗口，你可以在这里输入筛选条件表达式，例如下图筛选产品为 PD01，且折扣不小于 0.2 的订单：



3.9.8.11 条件表达式

大多数时候，我们并不需要直接输入条件表达式去筛选，但是下面的内容还是应该认真看看，特别是以后学习开发篇的时候，这些内容是很有用的：

示例一

筛选客户为 CS03 的行：

客户 = 'CS03'

注意字符要用单引号括起来。

示例二

筛选客户 CS03 订购产品 PD05 的行：

客户 = 'CS03' And 产品 = 'PD05'

示例三

筛选客户为 CS01 或者客户为 CS03 的行：

客户 = 'CS01' Or 客户 = 'CS03'

示例四

客户 CS01 或者 CS03 订购产品 PD01 的行：

(客户 = 'CS01' Or 客户 = 'CS03') And 产品 = 'PD01'

示例五

筛选产品为 PD01 且折扣大于等于 0.05 的行：

产品 = 'PD01' And 折扣 >= 0.05

示例六

筛选数量大于 100 且折扣等于 0 的行

数量 > 100 And 折扣 = 0

注意数字不能用单引号，不少人会在这个地方犯错误。

示例七

筛选雇员列没有输入内容的行：

雇员 Is Null

示例八

筛选雇员不为空的行

雇员 Is Not Null

示例九

筛选 1999 年 2 月 1 日的行：

[日期] = #2/1/1999#

注意日期用符号#括起来

示例十

筛选 1999 年 3 月份的订单

[日期] >= #3/1/1999# And [日期] <= #3/31/1999#

示例十一

筛选产品为 PD01、PD02、PD03 的行：

产品 In ('PD01','PD02','PD03')

或者

产品 = 'PD01' or 产品 = 'PD02' or 产品 = 'PD03'

显然这个时候，用 In 运算符要简洁很多。

记得数值型的列，值不要用单引号括起来，例如：

[订单号] In (1, 3, 4, 6, 7,12)

而日期型的列，值需要用符号#括起来，例如：

[日期] In (#7/2/2012#, #7/12/2012#, #8/30/2012#)

示例十二

筛选产品不是 PD01、PD02、PD03 的行：

产品 Not In ('PD01','PD02','PD03')

示例十三

我们知道表达式中的字符串用单引号括起来，如果字符串本身包括单引号，那么就用两个单引号

表示一个，例如：
型号 = 'HA"28'
表示型号等于：HA'28，中间连续的两个单引号表示一个单引号。

示例十四

所有的表都有一个名为_Locked 的列，用于保存行的锁定状态。
如果要筛选已经锁定的行，表达式为：
[_Locked] = True
如果要筛选未锁定的行，表达式却不能简单地设为：
[_Locked] = False
这是因为_Locked 列中可能有空值，所以正确的表达式为：
[_Locked] = False Or [_Locked] Is Null
虽然我们用的是_Locked 这样一个特殊的列作为例子，但是上面的知识适合任何逻辑列。

示例十五

*或%可以作为通配符，表示任意个数的字符，两者是等价的，可以相互替换。
例如：

姓名 Like '%赫%'
姓名 Like '赫%'
姓名 Like '%赫'
分别表示姓名中包括“赫”，姓名开头是“赫”(姓赫)，姓名结尾是“赫”
可以在 Like 前面加上 Not 关键词，表示不类似的意思，例如：
姓名 Not Like '李%'
表示不是姓李的。
由于* 和 % 默认作为通配符，如果要将其作为比较内容的一部分，必须用方括号括起来，例如：
型号 Like 'A[*]%'
表示型号以“A*”开头。
通配符只能出现在字符串的首尾，不允许在字符串的中间出现，例如不允许 '中*部'，也不允许 '中%部'。

3.9.8.12 取消筛选

有两个按钮可以用于取消筛选，一是单击下图中的“取消”按钮，二是单击下图中的“切换”按钮。
如果单击“切换”按钮取消筛选，那么再次单击该按钮，会回到筛选状态。



3.9.8.13 快捷菜单中的筛选命令

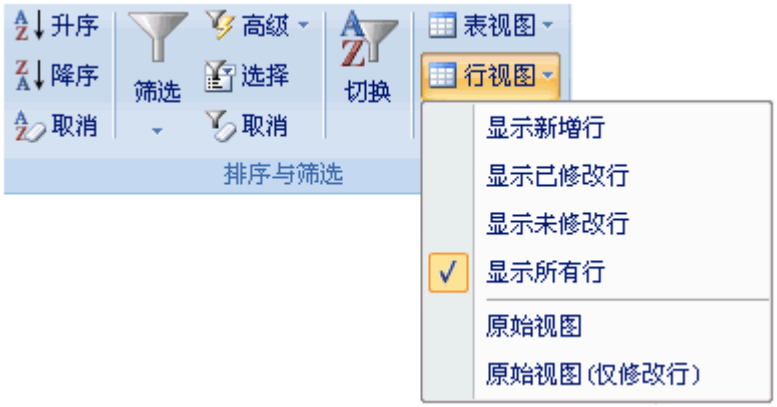
右击任何一个单元格，弹出的快捷菜单中会包括一个筛选菜单，该菜单同样是“智能”的，会随着选定列的数据类型、选定单元格的内容不同，而出现不同的筛选命令。



我们之前介绍的筛选功能，大部分可以通过这个菜单执行。
快捷菜单中集成功能强大的筛选命令，可以让我们进一步提高筛选效率。

3.9.9 行视图

行视图是 Foxtable 一个独特的功能，能够快速列出不同状态的行。
这样你随时可以找出刚刚修改了哪些行，哪些行又是新增加的。
行视图浏览按钮位于“日常工作”功能区“排序与筛选”功能组：



该按钮是一个下拉菜单，包括以下命令：

- **显示新增行**
  显示自上次保存以来新增加的行。

- **显示已修改行**
  显示自上次保存以来修改过的行。

- **显示未修改行**
  显示自上次保存以来没有修改过的行。

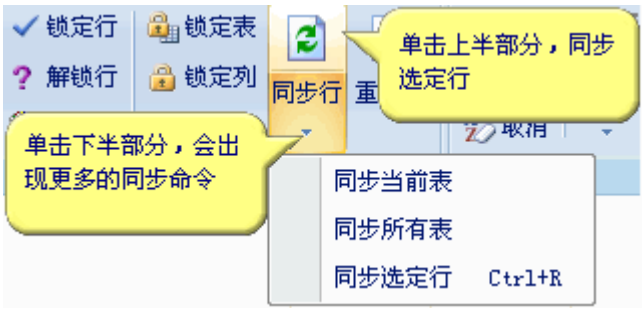
- **显示所有行**
  回到正常状态，也就是各种状态的行全部显示，这也是默认的状态。

- **原始视图**
  显示除新增行的所有行，不过内容是修改前的，而不是现在的。

  - **原始视图(仅修改行)**
    显示已经修改过的行，不过内容是修改前的，而不是现在的。

3.9.10 同步数据

在网络环境下，有可能你正在修改的数据，已经被其他人修改，为了得到最新版本的数据，可以使用同步功能。
同步按钮位于“日常工作”功能区“数据”功能组，这是一个组合按钮。



需要注意的是：
1、如果对方虽然修改了数据，但是并未保存，此时同步是没有作用的。
2、同步不会覆盖你已经做出的修改，例如你在第一列输入 A，对方在第二列输入 B 并保存，那么同步之后，第一列的内容为 A，第二列的内容为 B。

3.9.11 行高设置

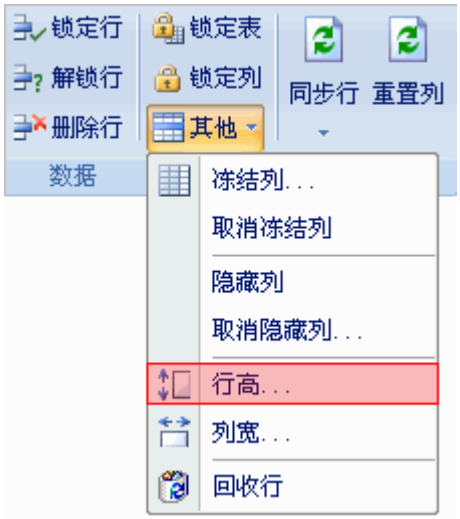
将鼠标移到行号列的行分界线处，鼠标形状变为上下箭头样式，拖动鼠标至所需行高即可：

	日期	产品编号	客户编号	数量	单价	金额
1	2008-06-06	2	1	6	65	390
2	2008-06-06	1	2	3	98	294
3	2008-06-07	3	3	7	200	1400
4	2008-06-08	4	4	2	25	50
5	2008-06-08	3	4	9	200	1800
6	2008-06-09	3	2	1	200	200

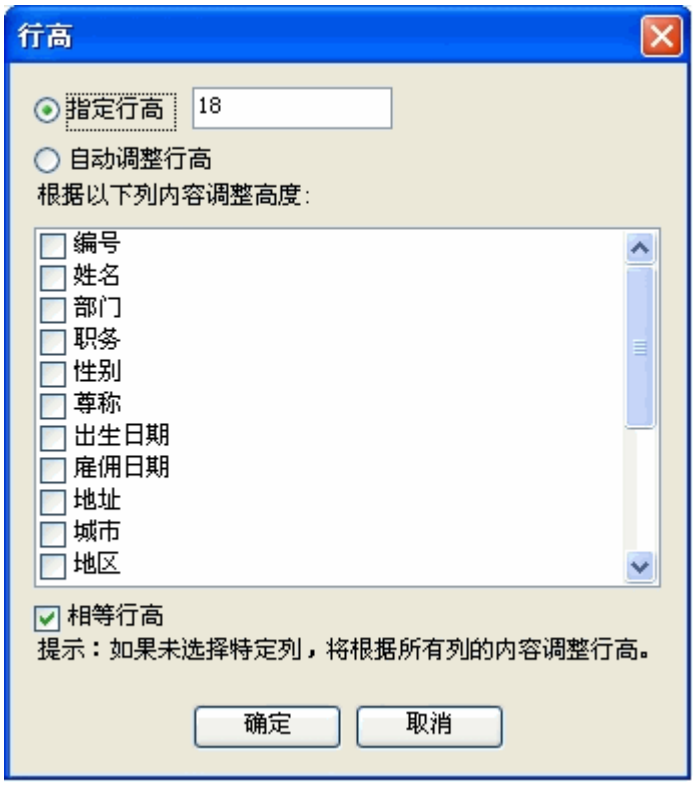
数据行和标题行的高度是分别调整的。

在表属性设置中，如果将属性“允许调整行高”设为 False(否)，用户将不能按上图所示那样通过鼠标拖动来调整行高。

在“日常工作”功能区的“数据”功能组，单击“其他”按钮，出现的下拉菜单中，有精确设置行高的命令：



单击“行高...”命令，会出现行高设置窗口：



精确设置行高

选择选项“指定行高”，然后在右边的框中输入行高（单位为像素），单击“确定”。

自动行高

选择选项“自动调整行高”，然后选择根据哪些列的内容调整行高，单击“确定”。

调整之后，所有行的高度等于最高的那一行，如果希望各行分别调整行高，不要选择选项“相等行高”即可。

提示：单元格内容不会自动换行的，除非输入的内容包含换行符，否则自动行高也就是恢复为默认的单行高度。

3.9.12 行号的特殊意义

在默认的情况下，表的左边会显示行号，行号的颜色是有特殊意义的，不同的颜色表示不同的行状态。

例如在下图中，灰色的行号(2、3 行)表示行被锁定，无法编辑；橙色的行号（4、5 行），表示行的内容被修改了；红色的行号（8、9 行），表示这是新增加的行：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03
4				11	0.15	80	748	1999-01-04
5	PD			12	0	490	5880	1999-01-04
6	PD			17.6	0.12	240	3717.12	1999-01-07
7	PD05	CS03	EP04	17	0	160	2720	1999-01-07
8								
9								

不同状态的行，行号的颜色是不同的。

通过表样式设置，可以改变默认的行号颜色。

3.9.13 快捷菜单中的行操作命令

右击表中任何一个单元格，在弹出的快捷菜单中有一个“行相关”菜单，此菜单会列出大部分和行相关的命令：



3.10 编辑数据

3.10.1 在单元格中输入数据

输入数据

选择要输入数据的单元格，直接输入数据，按 Enter 或 Tab 键或移到其他单元格，即可完成输入。

修改数据

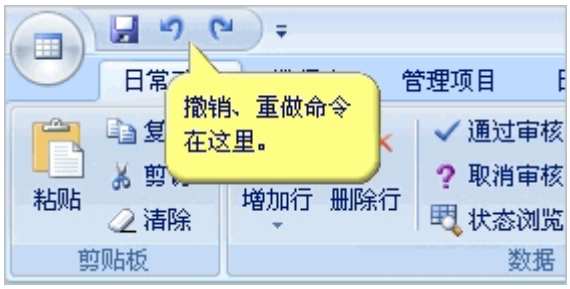
选择要修改的单元格，按空格键或者 F2 键进入修改状态；双击单元格也可以进入编辑状态，并会将光标置于双击位置。

快速复制

按"Ctrl + D"可以将上一行相同位置单元格的内容复制到当前单元格。

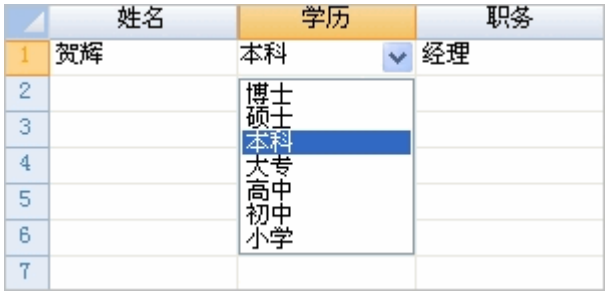
撤销错误操作

如果在按 Enter 或 Tab 键之前，要撤销正在输入的内容，请按“Esc”键；
在"快速访问栏"，单击“撤销”，可以撤销已经完成的输入操作。如果要撤销多步输入操作，反复单击“撤销”即可。



3.10.2 列表项目

通过设置列表项目，可以提高某些数据的录入速度，并能减少录入错误。例如在人事管理系统中，可以将学历列的列表项目设为“博士|硕士|本科|大专|高中|初中|小学”，这样一旦选择学历列，将出现一个下拉列表框，可以从中选择学历：



请参考：

设置自定义列表项目

禁止直接输入

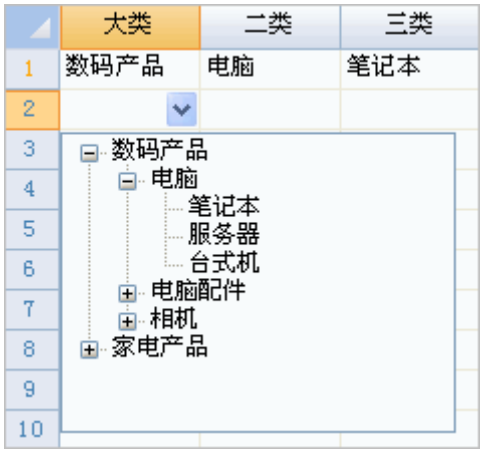
用键盘选择列表项目

启用输入助手

列表项目来自于列

3.10.3 目录树列表

列表项目还有另一种形式：目录树列表，可以更为高效地输入数据：



如果使用文件型目录树，请参考：

[创建目录树文件]

[使用目录树文件]

[使用内容分割符号]

如果要基于数据表生成目录树，请参考：

[数据表目录树]

[使用内容分割符]

[根据单列内容生成目录树]

3.10.4 输入日期

对于日期列，我们推荐将[列属性]“使用内置输入器”设为 True(是)。

在使用内置输入器的情况下：

1、单击单元格右侧的按钮，会出现一个日历，单击日历中的一个日期，即可在单元格中输入所单击日期。



2、选择日期列中一个空白单元格，按空格键进入编辑状态，再次按空格键，即可输入当天日期。

3、如果直接输入日期，必须按格式“yyyy-MM-dd”输入，即四位年、两位月、两位日， 如果月和日只有一位，输入的时候必须在前面补 0，例如“2011-01-01”。实际输入的时候，中间的横线会自动填入的，用户只需输入年月日即可。

提示：即使没有使用内置输入器，上述 2、3 点也同样有效。

3.10.5 输入数值

数值的输入并无技巧可言，但是需要注意：

1、对于输入的数值，默认可保留 4 位小数，你可以在列属性中调整这个值，属性名为“最大小数位数”。

2、如果某列的值，在输入之前，经常要进行一些临时性的运算，可以在列属性设置中，将该列的“使用内置输入器”属性设为 True(是)，这样单击单元格右侧的按钮，会出现一个计算器，用于临时计算：

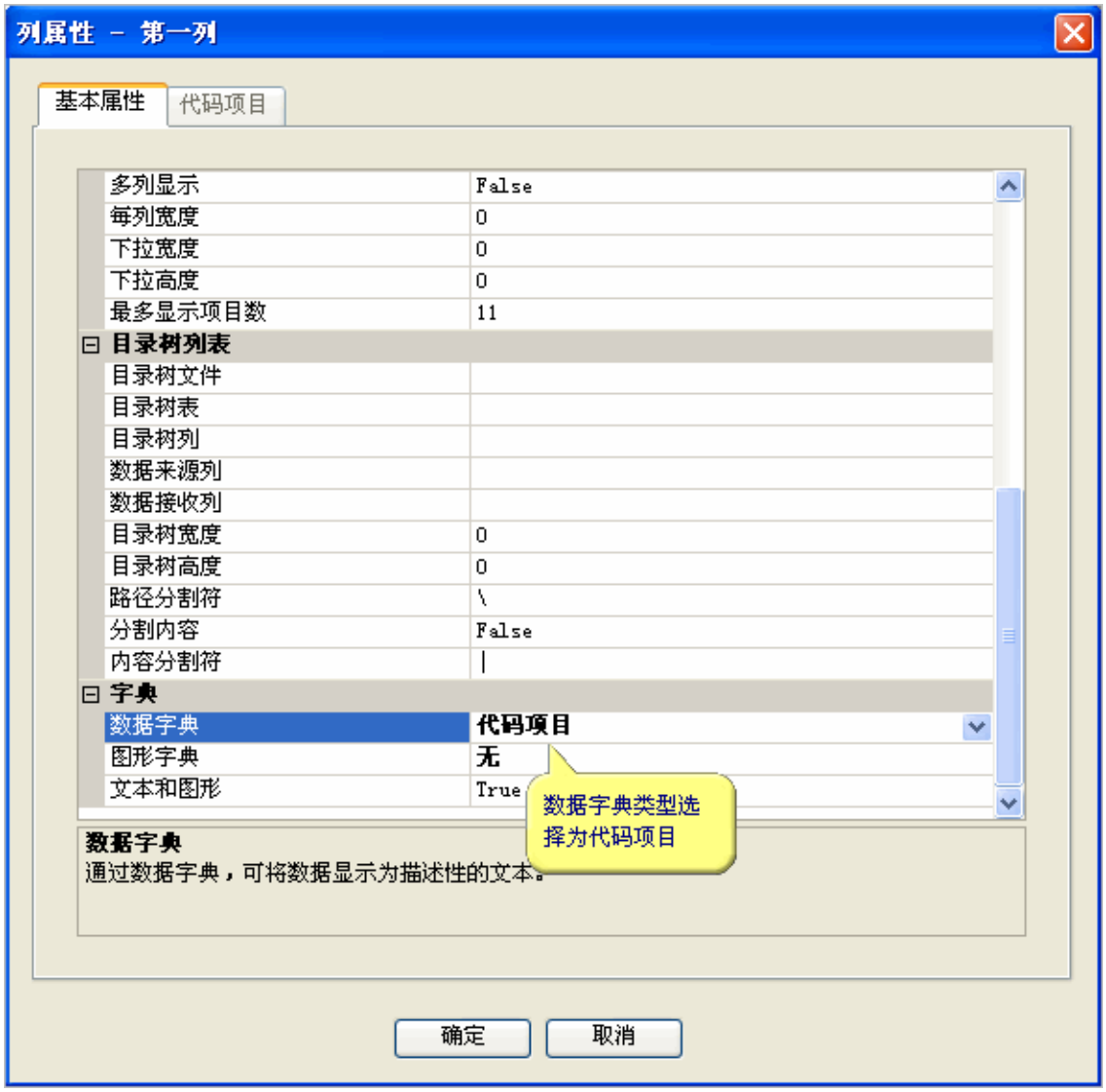


3.10.6 代码项目

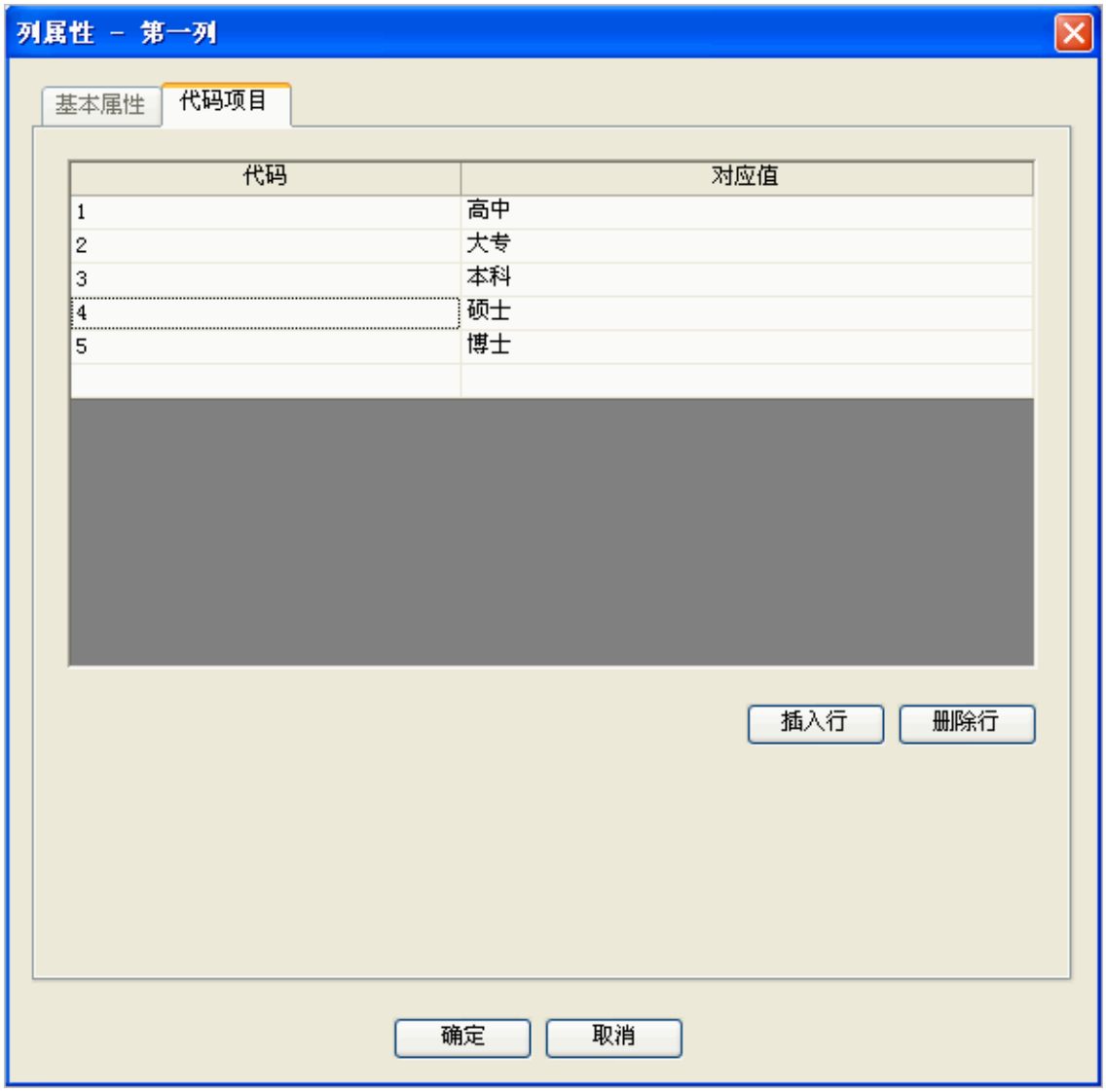
严格意义来说，代码项目并不属于数据字典。

所谓代码项目就是用一些简单的代码来表示某些项目，而用户在录入的时候，只需录入代码， 系统会自动将该代码转换为对应的项目。

1、选择要设置代码项目的列，打开其列属性设置窗口，将数据字典属性设为“代码项目”，然后单击“代码项目”页：

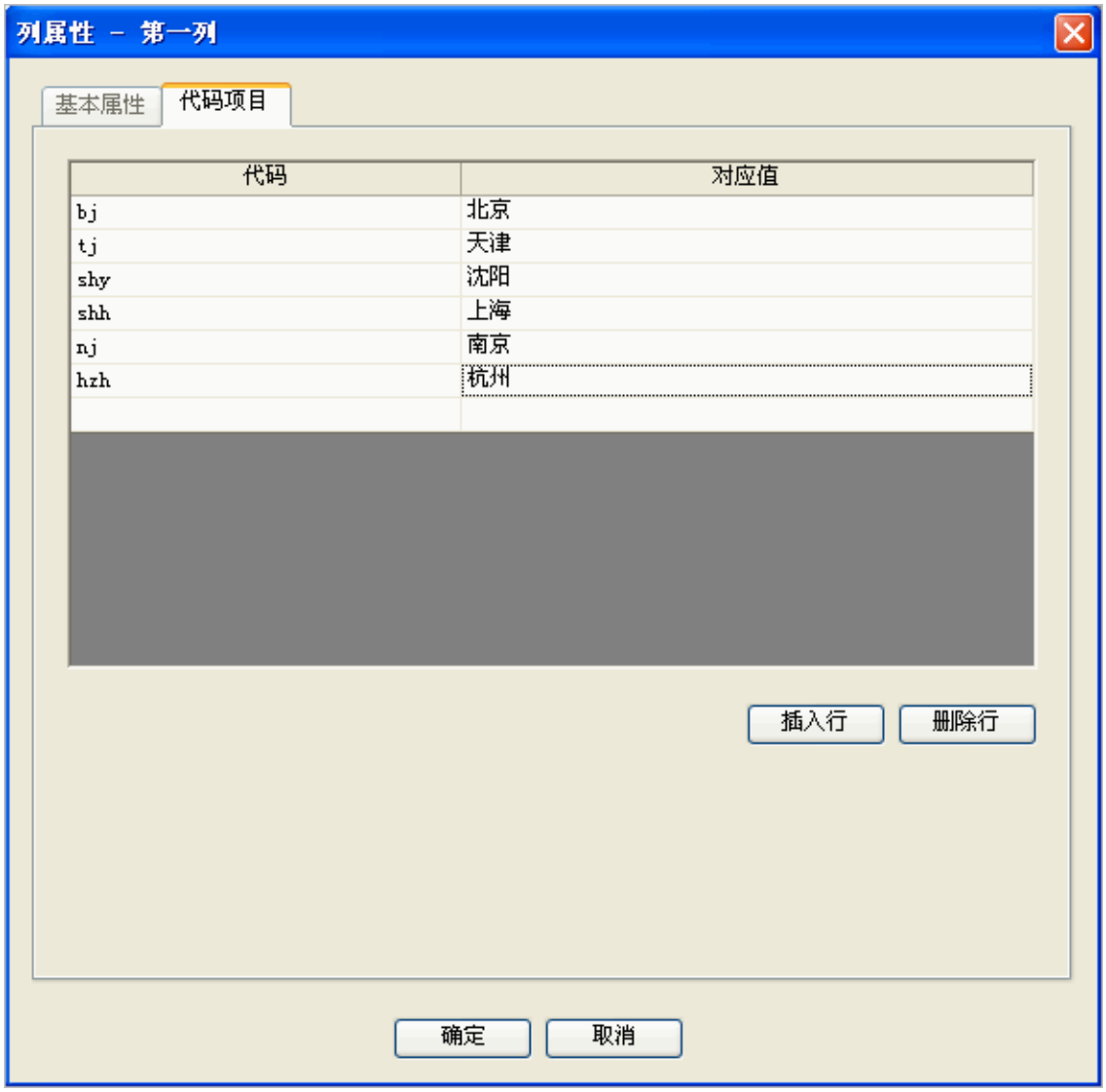


2、按下图所示设置代码项目：



经过上述设置后，我们在此列输入“1”，会被自动替换为“高中”，输入“2”，会被替换为“大专”，其余类推。

数字过于抽象，不便于记忆，实际应用的时候，代码最好使用拼音缩写，例如下面的代码使用拼音缩写来代表城市：



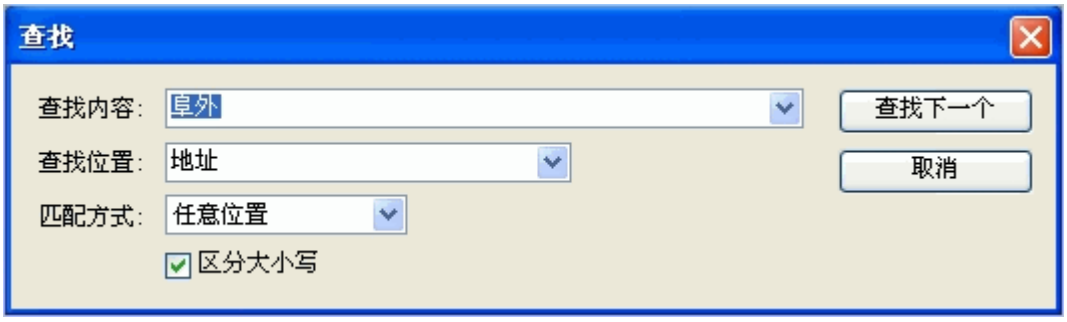
这样我们输入“bj”，即可自动替换为“北京”，输入“shh”，即可自动替换为“上海”，这比使用数字作为代码更加方便。

3.10.7 查找

1、在“日常工作”区的“编辑”功能组，单击“查找”按钮：



2、出现查找窗口，输入要查找的内容，选择要查找的列以及匹配方式；连续单击“查找下一个”按钮，会依次找出符合条件的行。



3、有四种匹配方式可选，分别是：

- 完全匹配
  单元格内容必须和查找内容完全相同，才算符合条件。

- 开始位置
  单元格内容以查找内容开头即可，例如你查找“abc”，那么“abcde”也是符合条件的。

- 结束位置
  单元格内容以查找内容结尾即可，例如你查找“abc”，那么“xyzabc”也是符合条件的。

- 任意位置
  只要单元格内容包含查找内容即可，例如查找“abc”，那么“abcde”、“xyzabc”、“xabcy”等都是符合条件的。

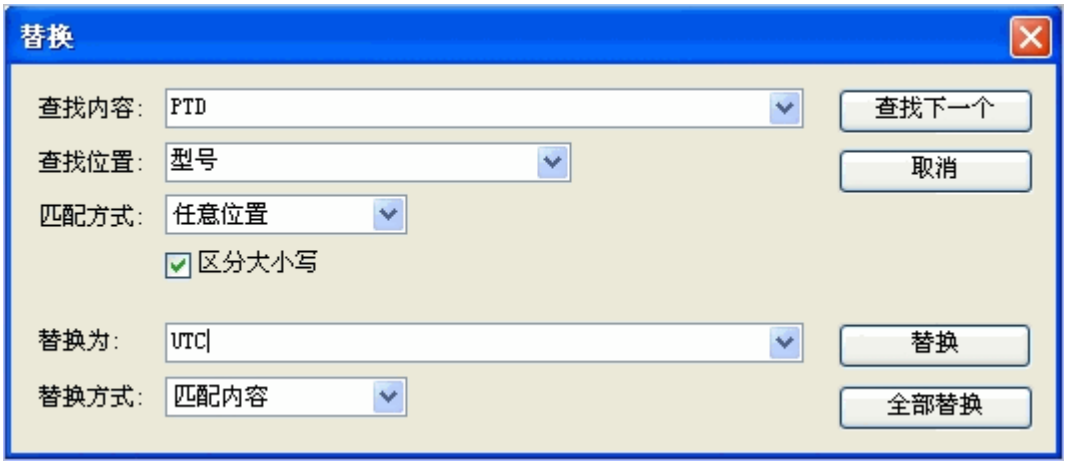
4、如果查找的时候，不想区分大小写，请不要选择选项“区分大小写”。

3.10.8 替换

1、在“日常工作”区的“编辑”功能组，单击“替换”按钮：



2、在替换窗口输入查找内容，选择要查找的列以及匹配方式，输入要替换的内容：



3、连续单击“查找下一个”按钮，会依次找出符合条件的行

4、单击“替换”按钮，会用新内容替换旧内容，并定位到下一个符合条件的行。

5、单击“全部替换”，将找出所有符合条件的行，并用新内容替换旧内容，结束之后，会提示总共有多少处被替换。

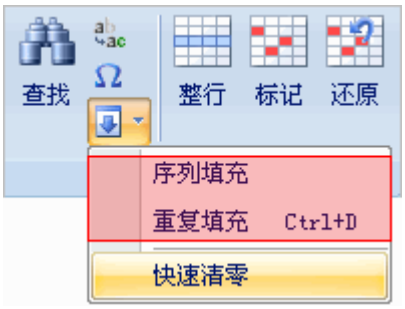
6、替换方式有两种，分别是：

- 全部内容
  原内容将完全被替换，例如匹配方式为任意位置，查找内容为“PTD”，替换内容为“UTC”，找到“HAPTD”，“HAPTD”会被替换为“UTC”

- 匹配内容
  原内容中只有和查找内容匹配的部分才会被替换，例如匹配方式为任意位置，查找内容为“PTD”，替换内容为“UTC”，找到“HAPTD”，“HAPTD”会被替换为“HAUTC”

3.10.9 数据填充

数据填充命令位于“日常工作”功能区的“编辑”功能组：



填充都是垂直方向的，也就是只能在某一列中向下填充；

有两种填充方式，分别是序列填充和重复填充。

序列填充

1、在填充区域的前两个单元格中，输入序列中的头两个值，例如 001、002。

2、然后选定整个要填充的区域。

3、单击“序列填充”命令。



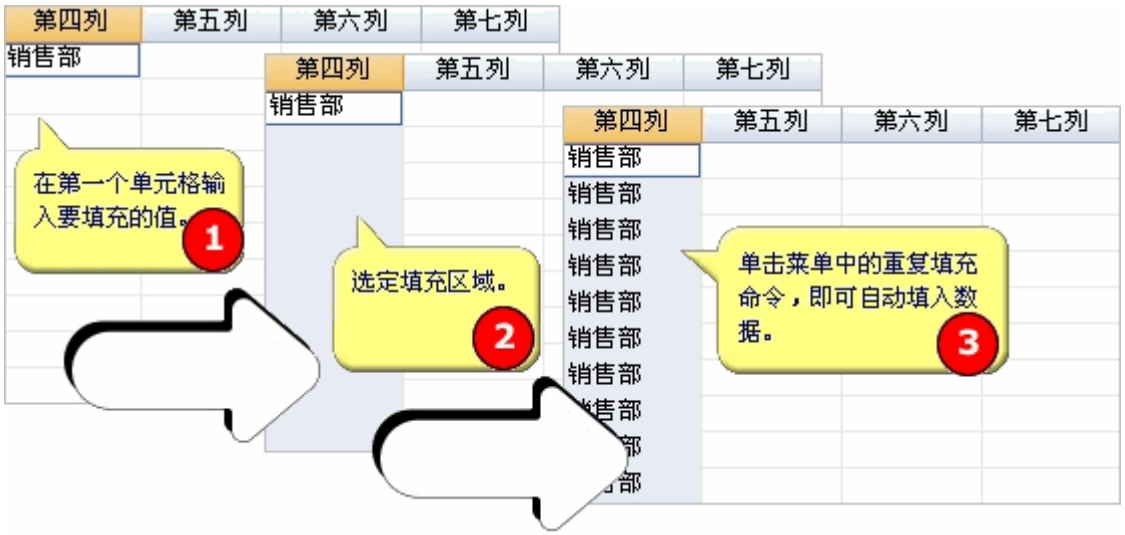
4、填充内容可以包括前缀，例如 A01、A02、A03...。

重复填充

1、在填充区域的第一个单元格输入要填入的值。

2、然后选定整个要填充的区域。

3、单击“重复填充”命令。



3.10.10 快速清零

快速清零命令位于“日常工作”功能区的“编辑”功能组：



单击快速清零，选定区域数值列内容为 0 的单元格，其内容将全部被清除。
提示：清零仅对数值列有效

3.10.11 插入符号

插入符号命令位于“日常工作”功能区的“编辑”功能组：



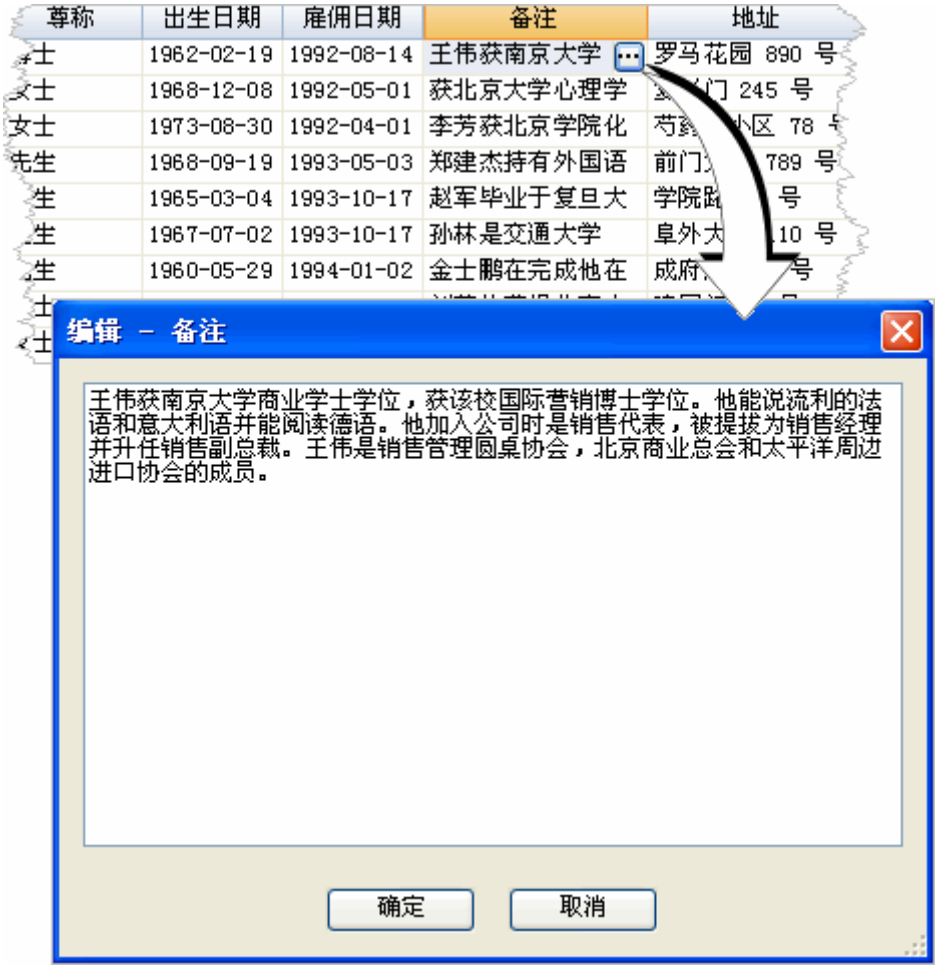
单击该按钮，出现插入符号的窗口，双击一个符号，即可将该符号插入到选定单元格中：



小技巧：这些符号是通过安装目录下的文件 SYMBOL.TXT 定义的，修改这个文件，你可以加入自己的常用符号，删除不常用的符号。

3.10.12 长文本的编辑

对于长文本的编辑，我们通常用备注型列解决。
如果你不想用备注列，希望将该列作为一个普通的字符列使用，但是希望也能用窗口形式输入内容。
解决的办法很简单，只需将该列的列表项目设置为“...”，这样选择该列某一单元格的时候，单元格右边会出现一个按钮，单击该按钮，就会出现一个输入窗口：



如果希望既能在单元格输入，又能在窗口输入，可以将列表项目设置为“|...”。

3.10.13 在单元格换行输入

直接在单元格中输入长文本内容时，可以同时按 Ctrl+Enter 键，换行输入内容。

3.10.14 记录窗口

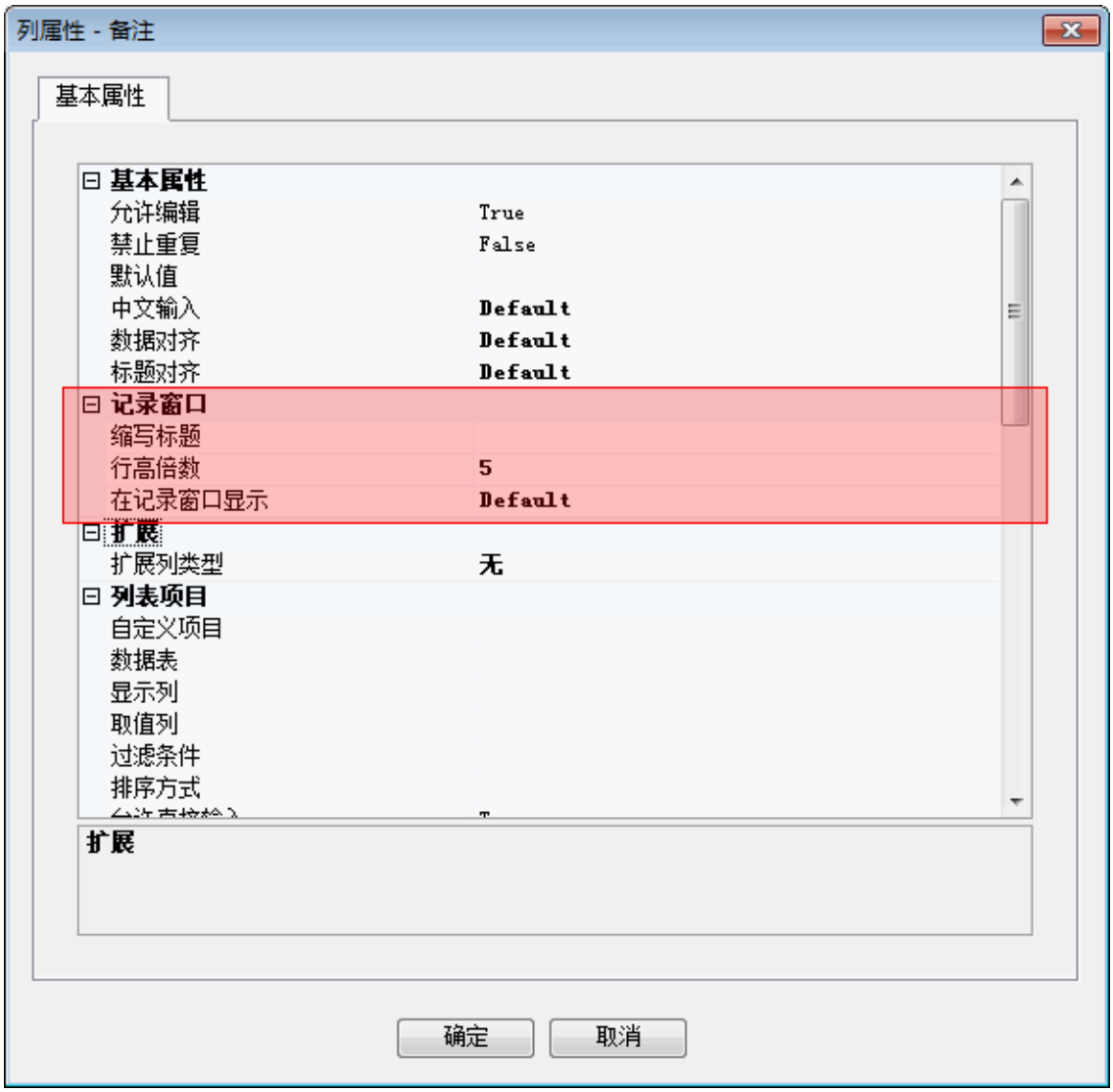
如果一个表有很多列，我们将不得不左右滚动才能查看全部数据，以至经常顾此失彼，而 Foxtable 的记录窗口可以轻松解决这个问题。记录窗口以垂直方式显示数据，每次只显示一条记录，如下图所示:



通过配置栏的“记录窗口”按钮，可以打开和关闭记录窗口：



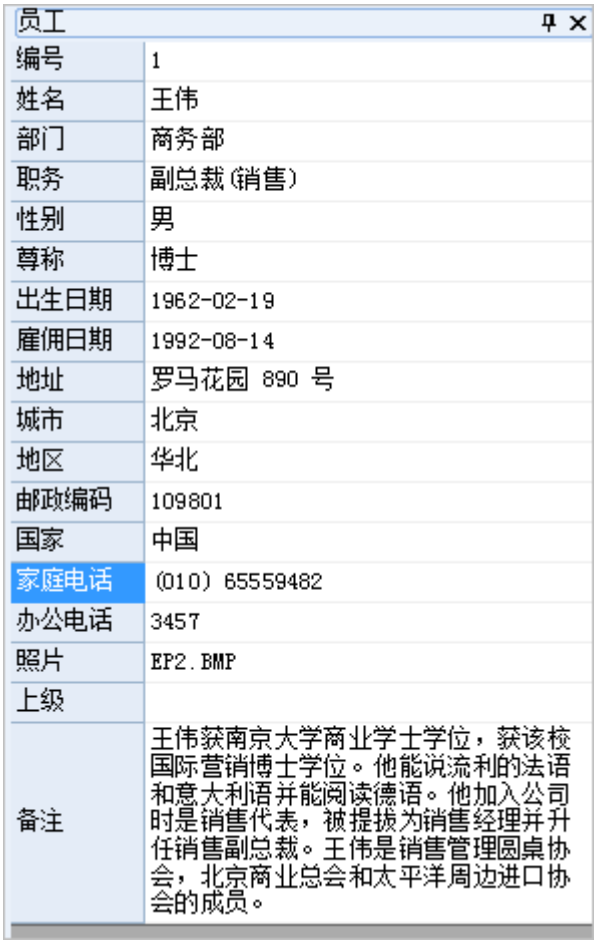
在列属性设置中，有三个属性是针对记录窗口的：



- **缩写标题**
  有的时候列标题过长，不适合记录窗口，可以通过此属性设置一个较短的标题，用于在记录窗口显示。

- **在记录窗口显示**
  在默认情况下，隐藏列不会显示在记录窗口，可以通过此属性明确设置某列是否出现在记录窗口。

- **行高倍数**
  设置此列在记录窗口显示时，行高是默认行高的多少倍。
  例如下面的记录窗口，备注的行高倍数是 5：



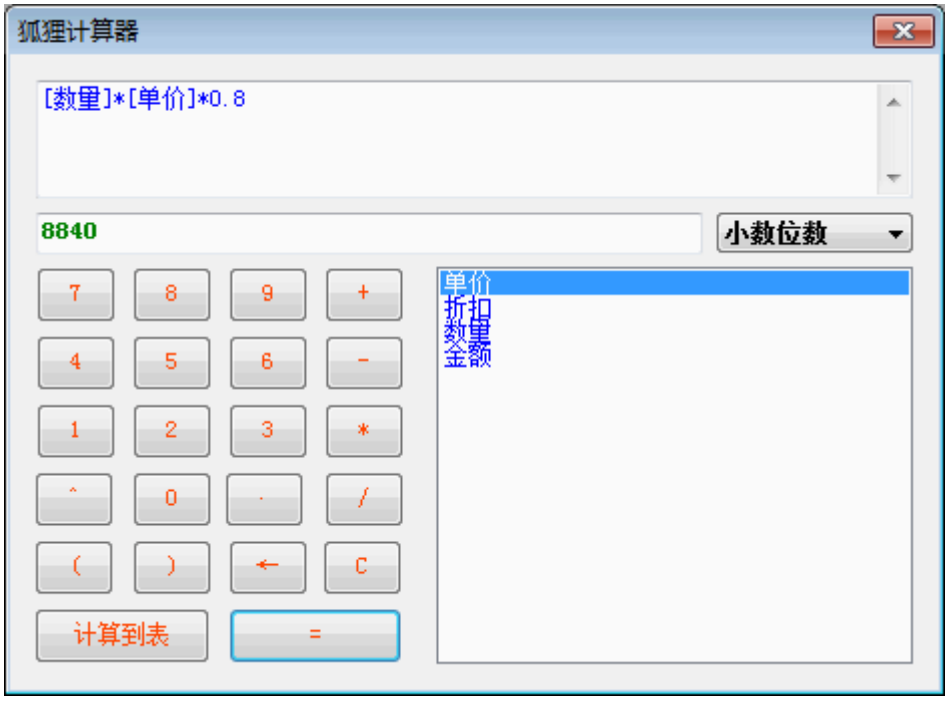
员工	
编号	1
姓名	王伟
部门	商务部
职务	副总裁(销售)
性别	男
尊称	博士
出生日期	1962-02-19
雇佣日期	1992-08-14
地址	罗马花园 890 号
城市	北京
地区	华北
邮政编码	109801
国家	中国
家庭电话	(010) 65559482
办公电话	3457
照片	EP2.BMP
上级	
备注	王伟获南京大学商业学士学位，获该校国际营销博士学位。他能说流利的法语和意大利语并能阅读德语。他加入公司时是销售代表，被提拔为销售经理并升任销售副总裁。王伟是销售管理圆桌协会，北京商业总会和太平洋周边进口协会的成员。

3.10.15 计算器

在配置栏单击下面的按钮，可以打开 Foxtable 内置的计算器：



和一般的计算器不同，内置的计算器可以使用当前表的数值列参与计算，而且可以将计算结果写入当前表的活动单元格：

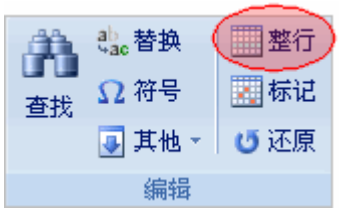


3.10.16 高级功能

3.10.16.1 整行突出

在编辑和查看数据的时候，如果表很宽，左右滚动表格时，很容易将相邻行的数据误作本行的数据，影响了数据处理的准确性。

在菜单的日常工作区的编辑功能组中，有一个名为“整行”的状态按钮:



一旦按下按钮，表中当前行将整行突出显示，这样可以有效避免上述情况的发生：



通过表样式的配色方案设置，可以改变整行突出显示时当前行的背景颜色和字体颜色 ：



3.10.16.2 修改标记

在菜单的日常工作区的编辑功能组中，有一个名为“标记”的状态按钮:



如果按下此按钮，可以在表中标出修改过的单元格。

例如订单表原来为：

	产品	客户	雇员	单价	折扣	数量	日期
1	PD05	CS03	EP04	17	0.1	650	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	1999-01-03
3	PD05	CS04	EP02	17	0	320	1999-01-03
4	PD01	CS03	EP04	18	0.15	80	1999-01-04
5	PD03	CS03	EP04	10	0	490	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	1999-01-07
7	PD05	CS03	EP04	17	0	160	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1999-01-08

如果修改了订单数据后，按下标记按钮，将标出所有已经修改过的单元格：

	产品	客户	雇员	单价	折扣	数量	日期
1	PD05	CS03	EP04	17	0.1	650	1999-01-02
2	PD02	CS01	EP01	21	0.1	400	1999-01-03
3	PD05	CS04	EP02	17	0	320	1999-01-03
4	PD01	CS03	EP04	18	0.2	80	1999-01-04
5	PD03	CS05	EP04	10	0	490	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	1999-01-07
7	PD05	CS03	EP04	17	0	160	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1999-01-08

默认用红底白字标记修改过的单元格，通过表样式的配色方案设置，可以修改其背景颜色和字体颜色，对应的区域为“已修改单元格”。

3.10.16.3 还原数据

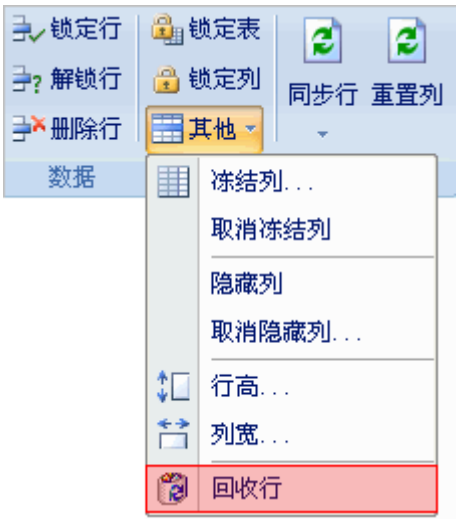
上一节讲到了如何标出修改过的单元格。
如果需要撤销对某指定单元格的修改，可以先选定此单元格，然后单击下图所示的“还原按钮”：



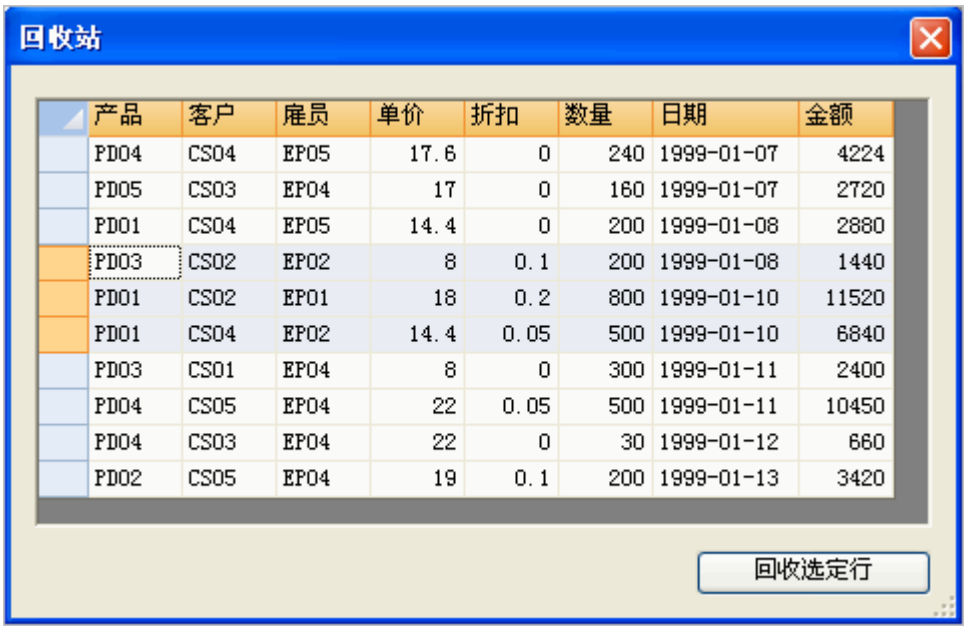
显然此还原按钮，比标准的撤销按钮要方便很多，因为撤销按钮只能逐步撤销上一步的操作，而无法单独跨步直接撤销对某个单元格的修改。
如果选定某个区域后，再单击还原按钮，将撤销该区域的所有修改。

3.10.16.4 回收已经删除的行

在“日常工作”功能区的“数据”功能组，单击“其他”按钮，出现的下拉菜单中，单击“回收行”：

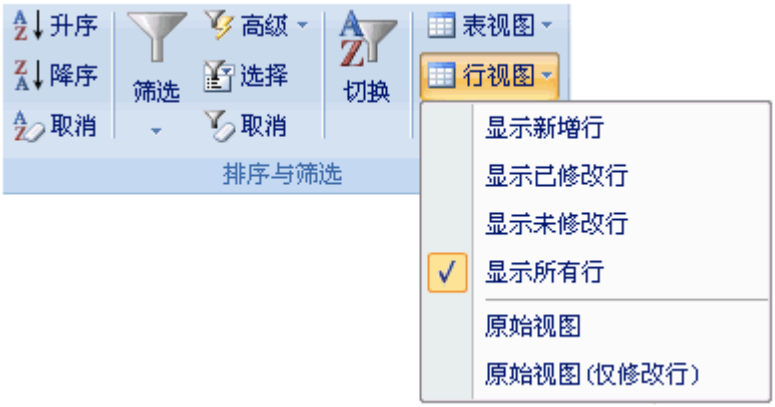


将出现一个窗口，此窗口会列出所有已经被删除的行，选定某些行，然后单击按钮“回收选定行”，这些行将重新回到数据表中：



3.10.16.5 行视图

行视图是 Foxtable 一个独特的功能，能够快速列出不同状态的行。
这样你随时可以找出刚刚修改了哪些行，哪些行又是新增加的。
行视图浏览按钮位于“日常工作”功能区“排序与筛选”功能组：



该按钮是一个下拉菜单，包括以下命令：

- **显示新增行**
  显示自上次保存以来新增加的行。

- **显示已修改行**
  显示自上次保存以来修改过的行。

- **显示未修改行**
  显示自上次保存以来没有修改过的行。

- **显示所有行**
  回到正常状态，也就是各种状态的行全部显示，这也是默认的状态。

- **原始视图**
  显示除新增行的所有行，不过内容是修改前的，而不是现在的。

  - **原始视图(仅修改行)**
    显示已经修改过的行，不过内容是修改前的，而不是现在的。

3.10.17 快速移动

按“Ctrl + ↑”键移到同一列的第一个单元格。
按“Ctrl + ↓”键移到同一列的最后一个单元格。

按“Home”键或按“Ctrl + ←”键移到同一行第一个单元格。
按“End”键或按“Ctrl + →”键移到同一行最后一个单元格。
按上下左右箭头键可以分别向上、向下、向左、向右移动单元格。

3.10.18 中文输入

同一个表中的不同列，有的可能是中文的，有的可能是字母或者数字的；这样在输入数据的时候，需要不停地切换输入法的中英文状态。
在列属性设置中，通过设置“中文输入”属性，你可以控制选择某一列时，自动打开或关闭中文输入法，还是保持当前状态不变。
该属性有三个可选值：
Default：保持当前输入法状态不变
True： 自动打开中文输入法
False： 自动关闭中文输入法
例如对于姓名列，我们应该将此属性设为 True，因为中国人的姓名自然是中文的；而对于产品型号列，应该将此属性设为 False，因为型号一般由字母和数字组成。
有了此属性，我们就可以避免手工切换输入法的中英文状态，而由系统根据输入列的设置自动选择。
此属性相当于模拟 Shift 按键。
提示：在数值列输入数字的时候，有的时候第一个输入的数字会无效，只是使得单元格进入编辑状态，如果发生这种情况，请将该列的中文输入属性设为 False。

3.10.19 回车键的移动方向

在输入数据的过程中，按回车键会向右移动到下一单元格，这是默认的情况。
如果你希望按回车键后，能够向下移动，在表属性设置中，将属性“按回车键向下移动”设为 True(是)即可。

3.10.20 Tab 键的移动方向

在输入数据的过程中，按 Tab 键会向右移动到下一单元格，这是默认的情况。
如果你希望按 Tab 键后，能够向下移动，在表属性设置中，将属性“按 Tab 键向下移动”设为 True(是)即可。

3.10.21 编辑模式

通过单击窗口右上角配置拦中的“编辑模式”按钮，可以进入或退出编辑模式。



并非只有在编辑模式下才能修改数据，非编辑模式下同样可以修改数据。
差别在于，在编辑模式下，编辑完某个单元格，按回车键、Tab 键或箭头键移到其他单元格后，新的单元格会自动进入编辑状态，相当于自动按了一次空格键或 F2 键；直到用鼠标选择其它单元格，或按 ESC 键，新选择的单元格才不会自动进入编辑状态。

如果你使用的是 Windows 10，个别输入法会有兼容性问题，在输入中文时，第一个字母偶尔会被"吞没"，需要先按一下空格键，让单元格先进入编辑状态，然后再输入中文 ，在连续输入数据时非常不便；此时可以开启编辑模式，将不会有这个问题。

3.10.22 选定数据或单元格

若要选择	请执行
单元格中的文本	选中并双击该单元格，再选取其中的文本。
单个单元格	单击相应的单元格，或按箭头键移动到相应的单元格。
某个单元格区域	单击区域的第一个单元格，再拖动鼠标到最后一个单元格。
较大的单元格区域	单击区域中的第一个单元格，再按住 Shift 单击区域中的最后一个单元格。可以先滚动到最后一个单元格所在的位置。
工作表中所有单元格	单击左上角第一个固定单元格。
整个行或列	单击行号或列标题。
相邻的行或列	在行号或列标题中拖动鼠标。或者先选中第一行或第一列，再

	按住 Shift 选中最后一行或最后一列。
增加或减少活动区域中的单元格	按住 Shift 单击需要包含在新选定区域中的最后一个单元格。
取消单元格选定区域	单击表中的任意单元格。



3.11 统计数据

3.11.1 本章示例文件

本章课程使用的示例文件为“统计演示.Table”，位于安装目录的子目录“CaseStudy”下。
本章主要应用的是该文件的第一个表：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03
4	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
5	PD03	CS03	EP04	10	0	490	4900	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07
7	PD05	CS03	EP04	17	0	160	2720	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08
10	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
11	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10

请在学习本章内容之前，先打开此文件。

3.11.2 汇总模式

3.11.2.1 关于汇总模式

在汇总模式下，程序会自动插入一些分组行，用于显示分组统计结果，例如下图是根据产品和客户分组，统计数量和金额的累计值：

204	[illegible]	CS04	EP03	17		20	340	1999-03-21
205	PD05	CS04	EP02	17	0.15	240	3468	1999-04-01
206	PD05	CS04	EP02	17	0	100	1700	1999-04-10
207	PD05	CS04	EP03	21.35	0	150	3202.5	1999-04-28
208	PD05	CS04	EP01	17	0	250	4250	1999-05-21
209	PD05	CS04	EP05	21.35	0	210	4483.5	1999-05-21
210	PD05	CS04	EP03	17	0	140	2380	1999-06-05
211	PD05	CS04	EP03	21.35	0	300	6405	1999-06-15
⊟	CS04 小计					3350	66256	
212	PD05	CS05	EP02	21.35	0	60	1281	1999-01-22
213	PD05	CS05	EP02	17	0	700	11900	1999-03-30
214	PD05	CS05	EP04	21.35	0.25	180	2882.25	1999-04-03
215	PD05	CS05	EP04	21.35	0.15	240	4355.4	1999-05-12
216	PD05	CS05	EP02	21.35	0.2	300	5124	1999-05-25
⊟	CS05 小计					1480	25542.65	
⊟	PD05 小计					10400	193023.15	
	总计					52370	825651.57	

处于汇总模式下的表，是只读的，必须先退出汇总模式，才可以增加、编辑数据。
汇总模式的设置非常简单，你只需单击几次鼠标，即可得到各种各样的统计结果。

3.11.2.2 设置汇总模式

在菜单的“日常工作”区，单击“数据统计”按钮，指向“汇总模式”，会出现汇总模式的相关菜单命令：



提示：

1、执行“设置汇总模式”，会打开汇总模式设置窗口，并加载最近一次统计的设置。

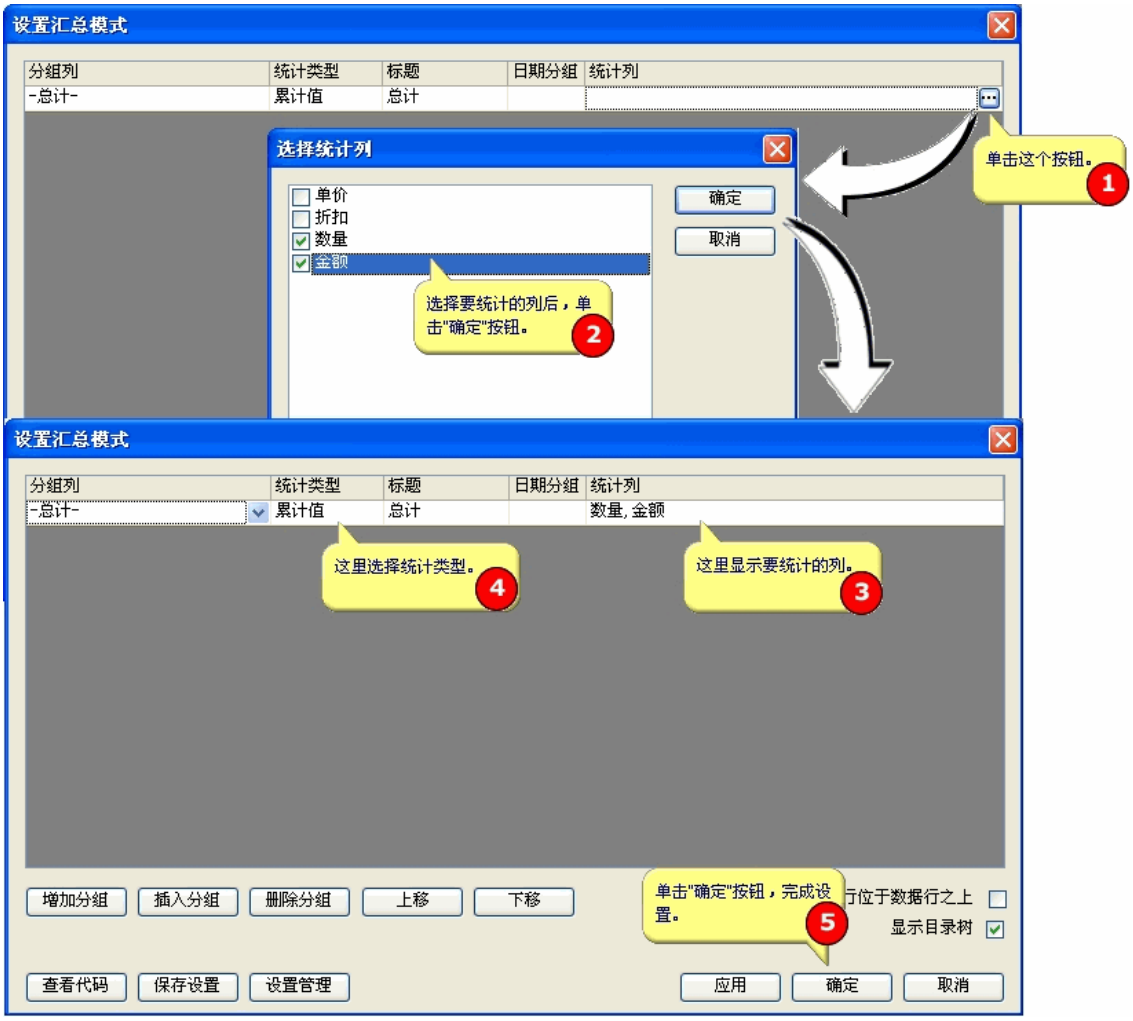
2、执行“新的汇总模式”，会打开汇总模式设置窗口，不加载最近一次统计的设置。

3、执行“切换汇总模式”，可以反复在汇总模式和普通模式下切换，例如为了输入新的数据，可以先单击该按钮退出汇总模式，新数据输入完成后，可以再次单击该按钮重新进入汇总模式，得到最新的统计结果。

4、执行“重复上次统计”，可以根据最近一次的设置，重新进入汇总模式。

汇总模式设置

下面是汇总模式设置窗口，默认已经有一个总计分组，按照下图所示进行设置：



经过上述简单的设置，即可获得数量和金额的总计，滚动到表格的最后，你可以看到总计行：

209	PD01	CS05	EP02	18	0	500	9000	1999-06-25
210	PD05	CS02	EP03	17	0.2	120	1632	1999-06-25
211	PD05	CS03	EP04	17	0	300	5100	1999-06-27
212	PD05	CS01	EP02	17	0	20	340	1999-06-28
213	PD02	CS02	EP02	15.2	0	400	6080	1999-06-29
214	PD02	CS03	EP02	19	0	30	570	1999-06-29
215	PD01	CS02	EP01	18	0	300	5400	1999-06-30
216	PD01	CS05	EP03	18	0	60	1080	1999-06-30
	总计					52370	825651.57	

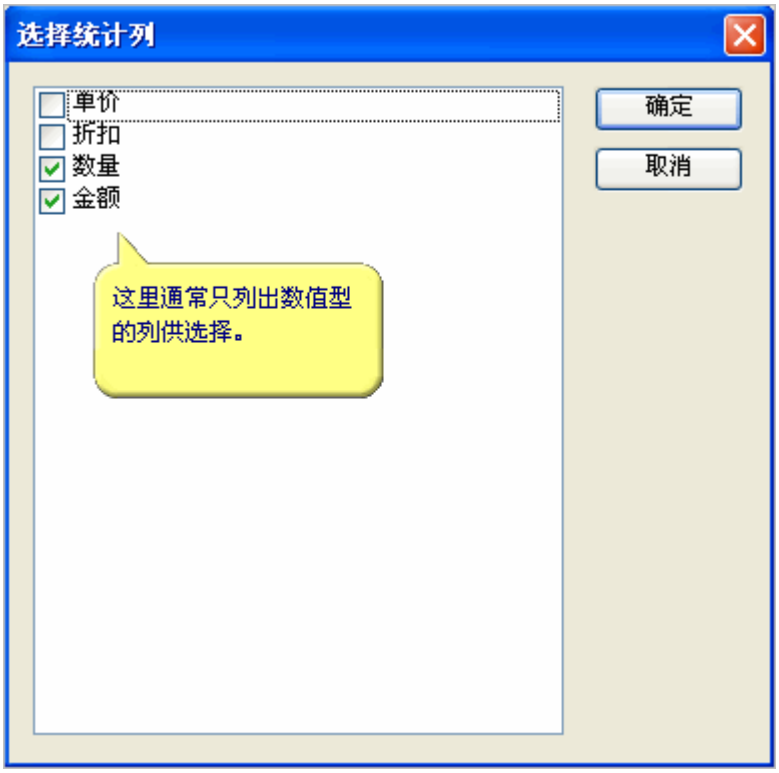
完成上述统计，我们只是单击了几下鼠标而已，可见设置汇总模式是非常简单的。

无法选择统计列？

在公测阶段，有的用户曾问我：为什么没有办法选择统计列？

通常这是因为列的数据类型设置不对，累计、平均、百分比等等类型的统计，显然是针对数值的，所以统计列选择窗口只会列出数值型列。

如果统计列选择窗口没有列出你希望统计的列，通常都是这个原因造成的。



如果要检查或更改某一列的数据类型，请参考：更改列
如果要检查某个表所有列的数据类型，请参考：查看表结构

3.11.2.3 增加分组列

上一节只是加入了总计行，如果你还希望根据产品分组，累计每一个产品的销售数量和金额，可以再次单击“设置汇总模式按钮”，之后按下图所示操作：



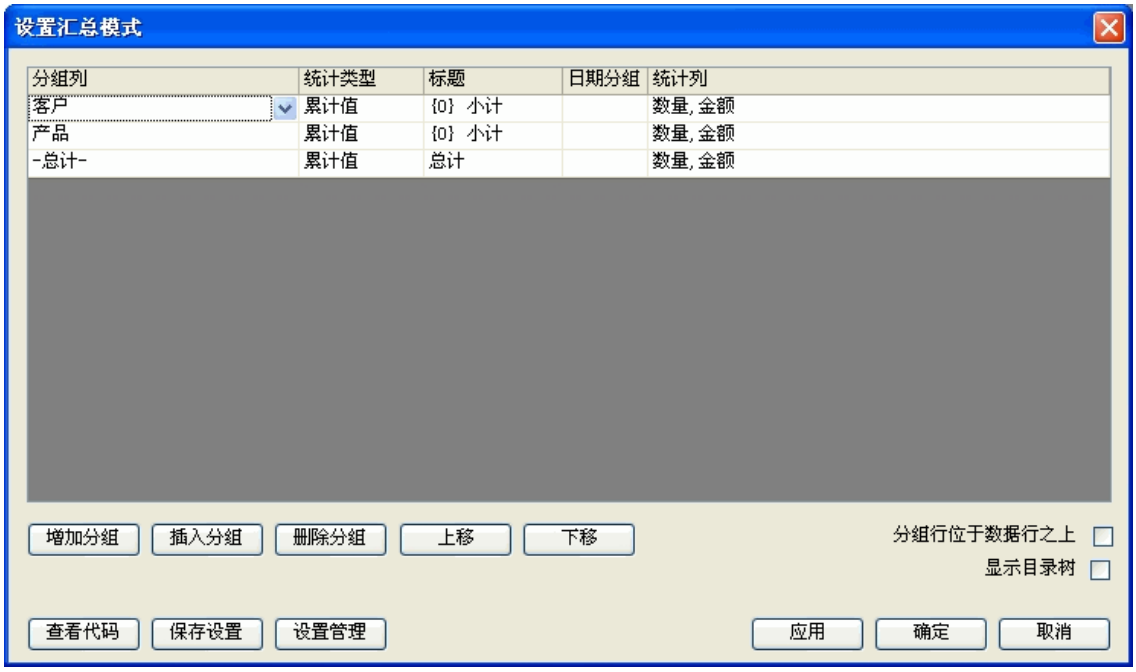
这是统计结果：

211	[illegible]	CS0	[illegible]	21.35	[illegible]	[illegible]	686	[illegible]
212	PD05	CS04	EP03	21.35	0	300	6405	1999-06-15
213	PD05	CS01	EP01	21.35	0	40	854	1999-06-22
214	PD05	CS02	EP03	17	0.2	120	1632	1999-06-25
215	PD05	CS03	EP04	17	0	300	5100	1999-06-27
216	PD05	CS01	EP02	17	0	20	340	1999-06-28
	PD05 小计					10400	193023.15	
	总计					52370	825651.57	

在生成汇总模式的过程中，会自动根据分组列进行排序，然后在每个分组之后插入一个分组行，用于显示分组统计结果。

3.11.2.4 更多的分组

你可以插入任意多个分组，但是需要注意分组的位置，小的分组放在前面。
例如下面的统计，首先是根据产品分组，对于产品相同的订单，再根据客户进行分组：



这是统计结果：

		CS	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
210	PD05	CS04	EP03	17	0	140	2380	1999-06-05
211	PD05	CS04	EP03	21.35	0	300	6405	1999-06-15
	CS04 小计					3350	66256	
212	PD05	CS05	EP02	21.35	0	60	1281	1999-01-22
213	PD05	CS05	EP02	17	0	700	11900	1999-03-30
214	PD05	CS05	EP04	21.35	0.25	180	2882.25	1999-04-03
215	PD05	CS05	EP04	21.35	0.15	240	4355.4	1999-05-12
216	PD05	CS05	EP02	21.35	0.2	300	5124	1999-05-25
	CS05 小计					1480	25542.65	
	PD05 小计					10400	193023.15	
	总计					52370	825651.57	

小技巧：
建议先设置好总计分组的统计类型和统计列，然后再增加分组，这样新增分组会自动从总计分组

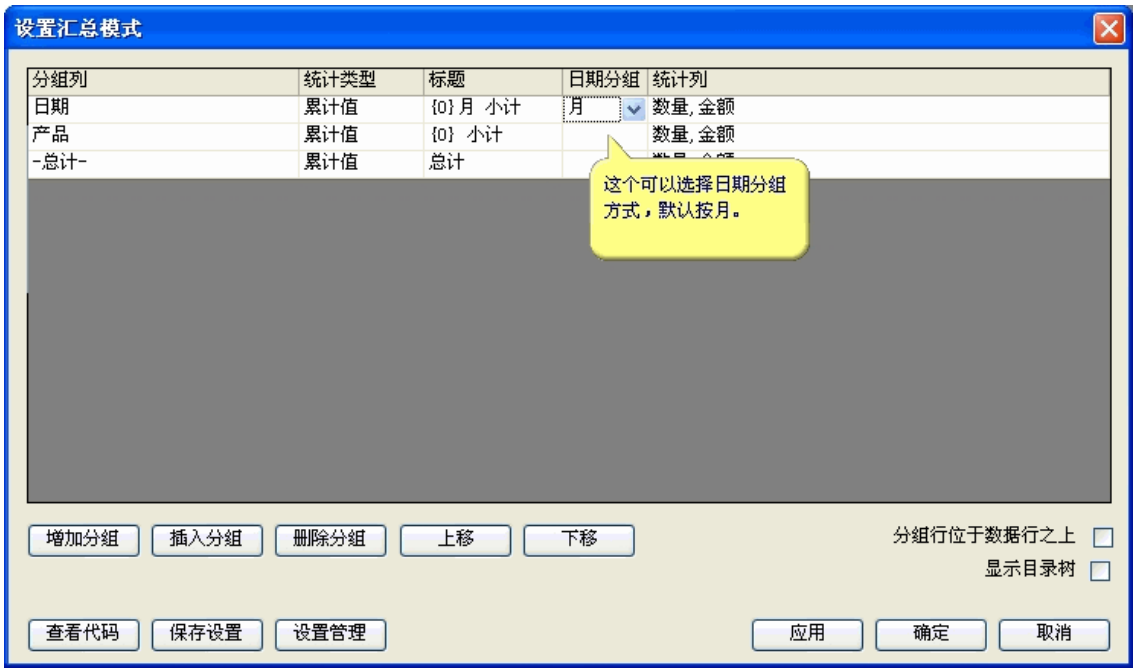
继承这些设置，无需反复设置。

3.11.2.5 对日期进行分组

在日常的数据管理中，我们经常要对不同时段之间的数据进行比较分析，例如不同月份、不同季度之间的数据比较。

对于这种统计，我们同样单击几次鼠标即可完成。

例如统计每个产品各月的销售数量和金额，可以增加一个日期分组，分组类型设为“月”，如下图所示：

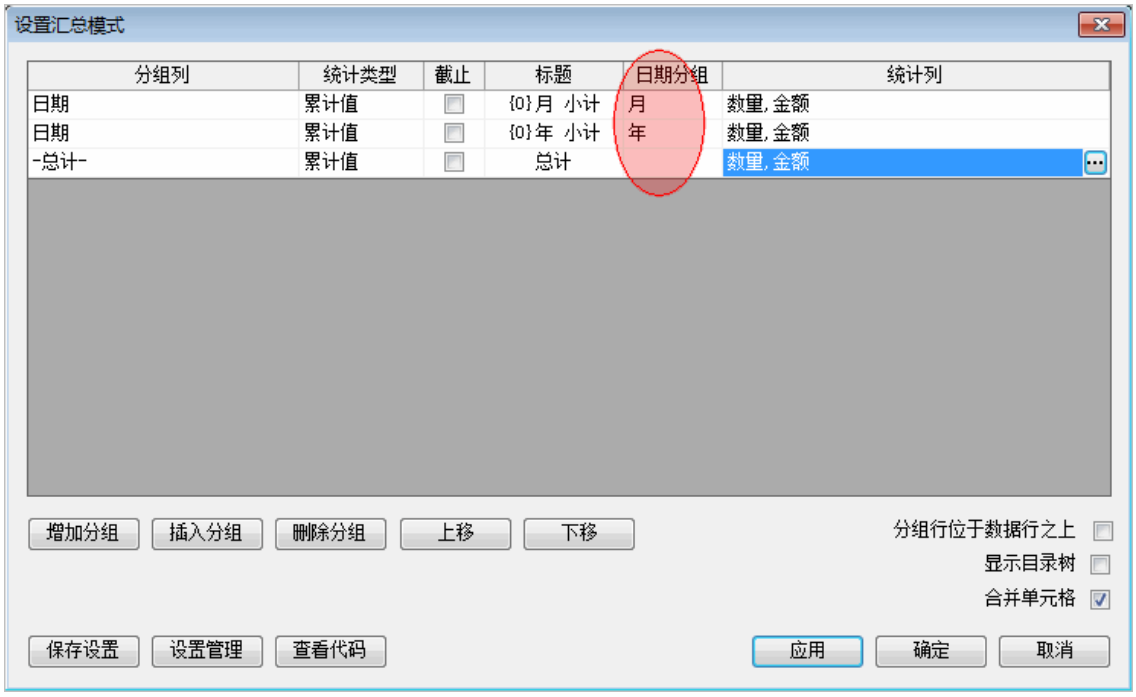


这是统计结果：

207	PD05	CS04	EP01	17	0	250	4250	1999-05-21
208	PD05	CS04	EP05	21.35	0	210	4483.5	1999-05-21
209	PD05	CS05	EP02	21.35	0.2	300	5124	1999-05-25
	5月 小计					2400	44030.4	
210	PD05	CS04	EP03	17	0	140	2380	1999-06-05
211	PD05	CS01	EP03	21.35	0	360	7686	1999-06-13
212	PD05	CS04	EP03	21.35	0	300	6405	1999-06-15
213	PD05	CS01	EP01	21.35	0	40	854	1999-06-22
214	PD05	CS02	EP03	17	0.2	120	1632	1999-06-25
215	PD05	CS03	EP04	17	0	300	5100	1999-06-27
216	PD05	CS01	EP02	17	0	20	340	1999-06-28
	6月 小计					1280	24397	
	PD05 小计					10400	193023.15	
	总计					52370	825651.57	

除按月分组外，Foxtable 还提供了年、季度、周等日期分组方式。

例如我们希望按年、月统计所有产品的销售数量和金额，可以按下图所示设置汇总模式，注意汇总模式是小的分组在前面，所以设置汇总模式的时候，先是月，然后是年，最后是总计：

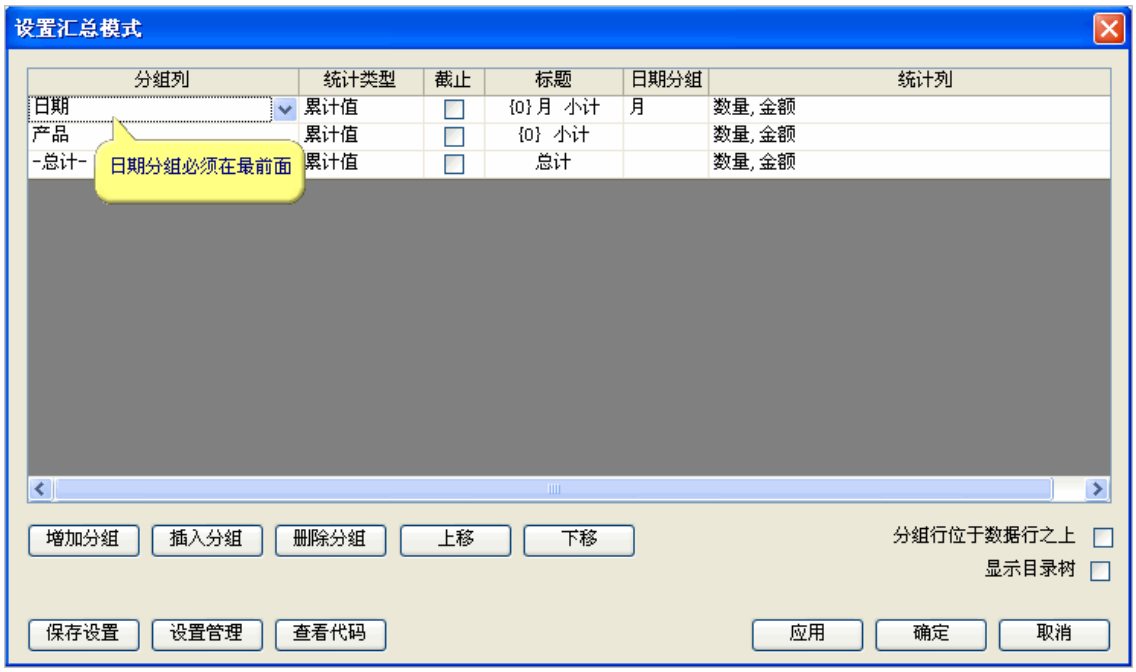


得到的汇总模式如下：

		CS03	EP04	18.5	297	5494.5	2000-11-25
228	PD01	CS03	EP01	19	686	13034	2000-11-26
229	PD01	CS03	EP01	17.5	311	5442.5	2000-11-27
230	PD01	CS01	EP02	18.5	976	18056	2000-11-28
	11月 小计				5738	105767.5	
231	PD01	CS01	EP05	17.5	11	192.5	2000-12-08
232	PD01	CS02	EP02	17.5	241	4217.5	2000-12-14
233	PD01	CS02	EP05	18.5	701	12968.5	2000-12-16
234	PD01	CS03	EP02	18	905	16290	2000-12-20
235	PD01	CS03	EP05	18	260	4680	2000-12-21
236	PD01	CS02	EP05	18	500	9000	2000-12-21
237	PD01	CS01	EP05	19	828	15732	2000-12-22
238	PD01	CS02	EP02	18	243	4374	2000-12-24
239	PD01	CS01	EP03	18.5	589	10896.5	2000-12-24
240	PD01	CS01	EP01	19	186	3534	2000-12-26
241	PD01	CS03	EP03	19	518	9842	2000-12-28
	12月 小计				4982	91727	
	2000年 小计				67491	1234519.5	
	总计				125624	2300988	

3.11.2.6 再谈日期分组

如果有多个分组列，那么日期型分组列必须位于最前面，否则汇总模式将不正常。



如果有特别的要求，确实需要日期分组位于其它分组之后，可以考虑增加一个表达式列，用表达式取得日期的年、月、日，然后将该表达式列作为分组列使用。

参考：

[用表达式计算年月日]

3.11.2.7 设置分组标题

可以给每一个分组指定标题，在设置标题的时候，可以用字符“{0}”来表示分组的名称，例如分组标题设为“{0} 小计”，在生成汇总模式的时候，如果某个分组的名称为“PD05”，那么分组标题将显示为“PD05 小计”。



209	PD05	CS05	EP02	21.35	0.2	300	5124	1999-05-25
210	PD05	CS04	EP03	17	0	140	2380	1999-06-05
211	PD05	CS01	EP03	21.35	0	360	7686	1999-06-13
212	PD05	CS04	EP03	21.35	0	300	6405	1999-06-15
213	PD05	CS01	EP01	21.35	0	40	854	1999-06-22
214	PD05	CS0			2	120	1632	1999-06-25
215	PD05	CS0			0	300	5100	1999-06-27
216	PD05	CS0			0	20	340	1999-06-28
	PD05 小计					10400	193023.15	
	总计					52370	825651.57	

生成汇总模式的时候，"{0}"被实际的分组名称代替，例如这里用产品名称"PD05"代替"{0}"，所以"{0} 小计"成了"PD05 小计"。

3.11.2.8 同一分组进行多种统计

之前每一种分组都只进行了一种统计，就是累计值。

其实 Foxtable 提供了很多统计方式，包括：累计值、平均值、最大值、最小值、记录数、百分比、标准偏差、总体标准偏差、采样方差、总体方差。

更重要的是，即使是同一个分组，也可以进行多种不同类型的统计。

示例一

打开本章示例文件“统计演示.Table”的第二个表“学生成绩”，我们希望同时统计出最大值、最小值、平均值，可以按照下图所示设置汇总模式：



得到的统计结果是：

37	[illegible]	程康文	116	20		78		
38	4班	邓玉婷	118	89	90	77	78	452
39	4班	林景阳	119	133	95	57	61	465
40	4班	苏　捷	124	118	124	83	88	537
41	4班	谢杰立	129	118	139	88	80	554
42	4班	谢晓霞	134	113	132	85	87	551
43	4班	杨景亮	132	136	146	93	94	601
44	4班	张李华	133	125	140	71	75	544
	最小			38	74	36	18	300
	最大			136	146	93	100	601
	平均			104	118	74	77	493

示例二

要求和示例一相同，但是分班统计，汇总模式的设置如下图所示，注意这次我们使用了自定义的分组标题，例如“{0} 最低分”：



这是统计结果：

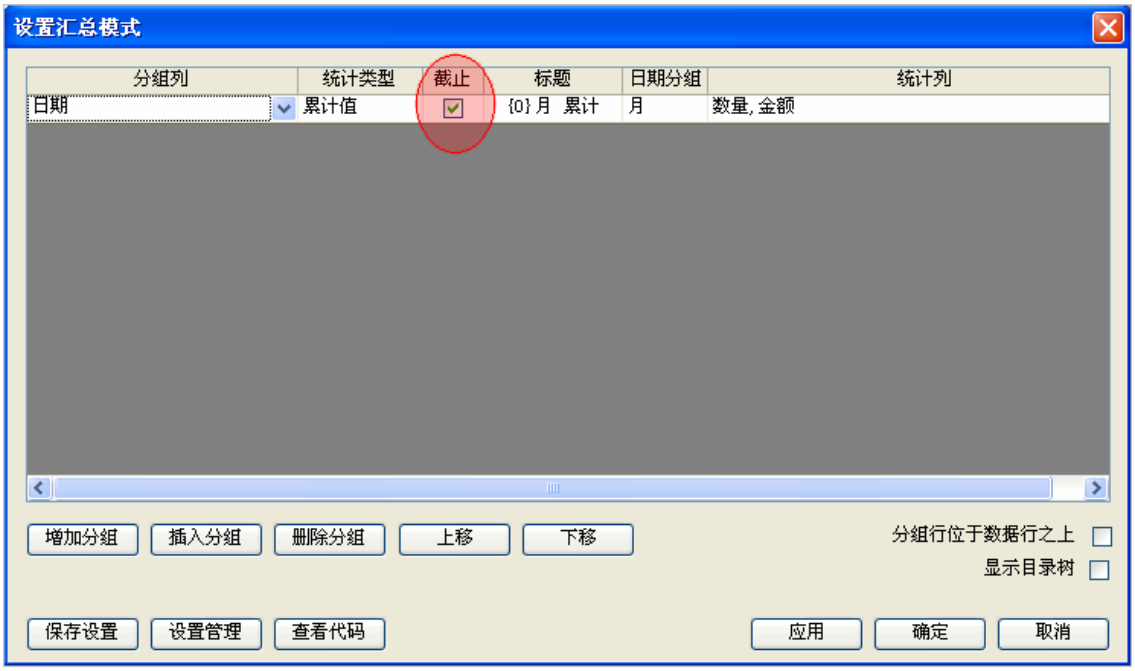
	4班	状态	113	5			61	
37	4班	程康文	116	120	132	78	83	529
38	4班	邓玉婷	118	89	90	77	78	452
39	4班	林景阳	119	133	95	57	61	465
40	4班	苏　捷	124	118	124	83	88	537
41	4班	谢杰立	129	118	139	88	80	554
42	4班	谢晓霞	134	113	132	85	87	551
43	4班	杨景亮	132	136	146	93	94	601
44	4班	张李华	133	125	140	71	75	544
	4班 最低分			55	90	36	46	361
	4班 最高分			136	146	93	98	601
	4班 平均分			108	119	73	79	501
	最低分			38	74	36	18	300
	最高分			136	146	93	100	601
	平均分			104	118	74	77	493

3.11.2.9 截止统计

汇总模式默认是各个分组独自统计，可是有的时候，我们需要的并非是每个分组的统计结果，而是截止到每个分组时的统计结果。

例如在对订单表进行统计的时候，我们希望得到截止到每个月的时候，总的订购数量。

要得到这样的统计结果非常简单，只需在设置统计列的时候，选择选项“截止”即可：

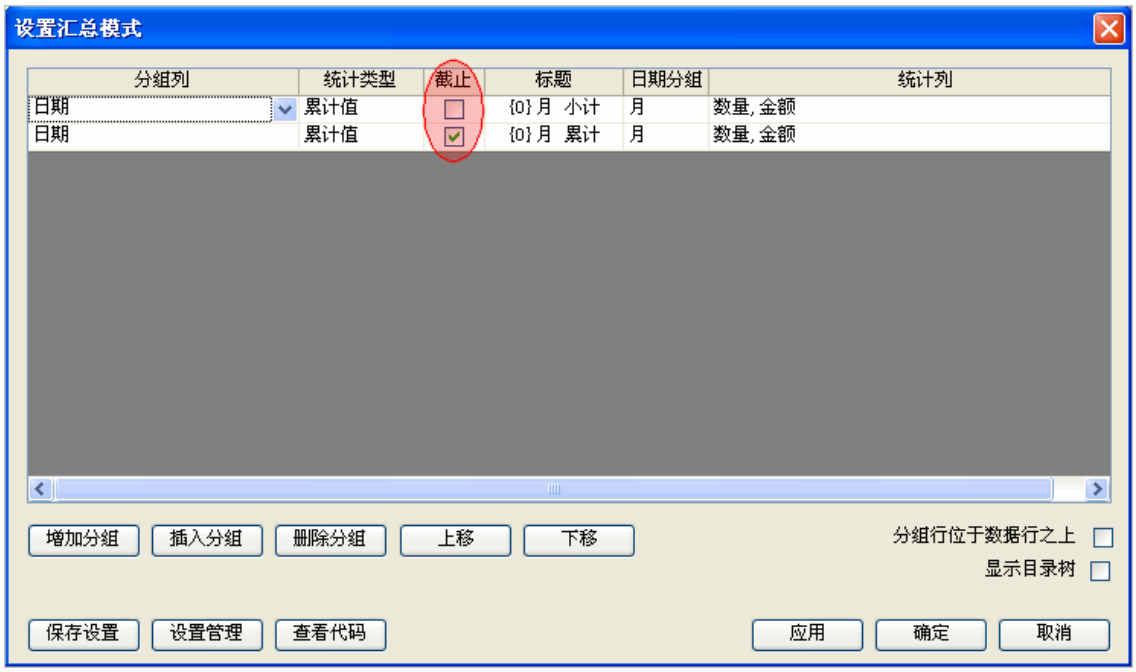


这是得到的统计结果：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07
2	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
3	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
4	PD03	CS02	EP05	8	0.25	700	4200	1999-01-26
	1月 累计					1240	12744	
5	PD01	CS05	EP05	18	0	200	3600	1999-02-01
6	PD02	CS04	EP05	19	0	100	1900	1999-02-21
7	PD01	CS02	EP05	14.4	0.15	150	1836	1999-02-22
	2月 累计					1690	20080	
8	PD05	CS03	EP05	21.35	0.1	200	3843	1999-03-11
9	PD01	CS03	EP05	18	0	100	1800	1999-03-24
10	PD02	CS02	EP05	15.2	0	600	9120	1999-03-24
11	PD05	CS02	EP05	21.35	0.25	300	4803.75	1999-03-29
	3月 累计					2890	39646.75	
12	PD02	CS04	EP05	15.2	0.2	5[illegible]	6080	1999-04-07
13	PD01	CS04	EP05	18	0.25			
14	PD04	CS05	EP05	22	0.25			

这并非3月份的订购数量，而是截至到3月份的累计订购数量。

如果你希望同时得到每月和截止到该月的累计，可以按下图所示设置：



这是统计结果：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07
2	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
3	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
4	PD03	CS02	EP05	8	0.25	700	4200	1999-01-26
	1月 小计					1240	12744	
	1月 累计					1240	12744	
5	PD01	CS05	EP05	18	0	200	3600	1999-02-01
6	PD02	CS04	EP05	19	0	100	1900	1999-02-21
7	PD01	CS02	EP05	14.4	0.15	150	1836	1999-02-22
	2月 小计					450	7336	
	2月 累计					1690	20080	
8	PD05	CS03	EP05	21.35	0.1	200	3843	1999-03-11
9	PD01	CS03	EP05	18	0	100	1800	1999-03-24
10	PD02	CS02	EP05	15.2	0	600	9120	1999-03-24
11	PD05	CS02	EP05	21.35	0.25	300	4803.75	1999-03-29
	3月 小计					1200	19566.75	
	3月 累计					2890	39646.75	
12	PD02	CS04	EP05	15.2	0.2	500	6080	1999-04-07
13	PD01	CS04	EP05	18	0.25	40	540	1999-04-10

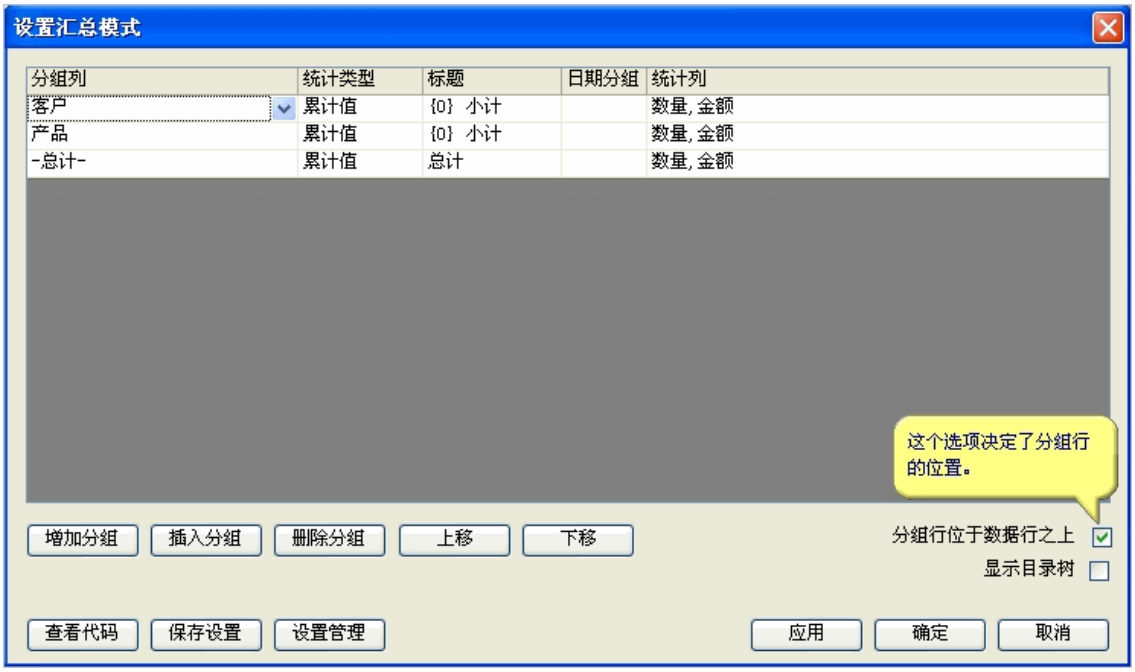
提示：如果选择选项“分组行位于数据行之上”，则 截止统计无效。

3.11.2.10 分组行的位置

分组行默认在数据行之下，如果你喜欢，也可以先显示分组行，再显示数据行，如下图所示：

	产品	客户	雇员	单价	折扣	数量	金额	日期
	总计					52370	825651.57	
	PD01 小计					11290	171754.92	
	CS01 小计					260	3777.12	
1	PD01	CS01	EP04	18	0	20	360	1999-02-07
2	PD01	CS01	EP02	18	0	80	1440	1999-02-07
3	PD01	CS01	EP02	14.4	0.15	150	1836	1999-04-12
4	PD01	CS01	EP02	14.4	0.02	10	141.12	1999-06-23
	CS02 小计					4120	60292.8	
5	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
6	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
7	PD01	CS02	EP05	14.4	0.15	150	1836	1999-02-22
8	PD01	CS02	EP03	14.4	0	200	2880	1999-02-26
9	PD01	CS02	EP02	18	0	100	1800	1999-03-04

要使分组行位于数据行之上，只需在设置汇总模式的时候，选中选项“分组行位于数据行之上”即可：



3.11.2.11 显示目录树

当表处于汇总模式的时候，可以在表的左边显示一个目录树，这样可以用类似于 Windows 资源管理器的方式来显示和管理表中数据：

1 2 *	产品	客户	雇员	单价	折扣	数量	金额	日期
	总计					52370	825651.57	
⊟	PD01 小计					11290	171754.92	
⊟	CS01 小计					260	3777.12	
1	PD01	CS01	EP04	18	0	20	360	1999-02-07
2	PD01	CS01	EP02	18	0	80	1440	1999-02-07
3	PD01	CS01	EP02	14.4	0.15	150	1836	1999-04-12
4	PD01	CS01	EP02	14.4	0.02	10	141.12	1999-06-23
⊞	CS02 小计					4120	60292.8	
⊞	CS03 小计					1980	28638	
⊞	CS04 小计					3390	52155	
⊞	CS05 小计					1540	26892	
⊞	PD02 小计					18200	296522.5	
⊞	PD03 小计					7000	61490	
⊞	PD04 小计					5480	102861	
⊞	PD05 小计					10400	193023.15	

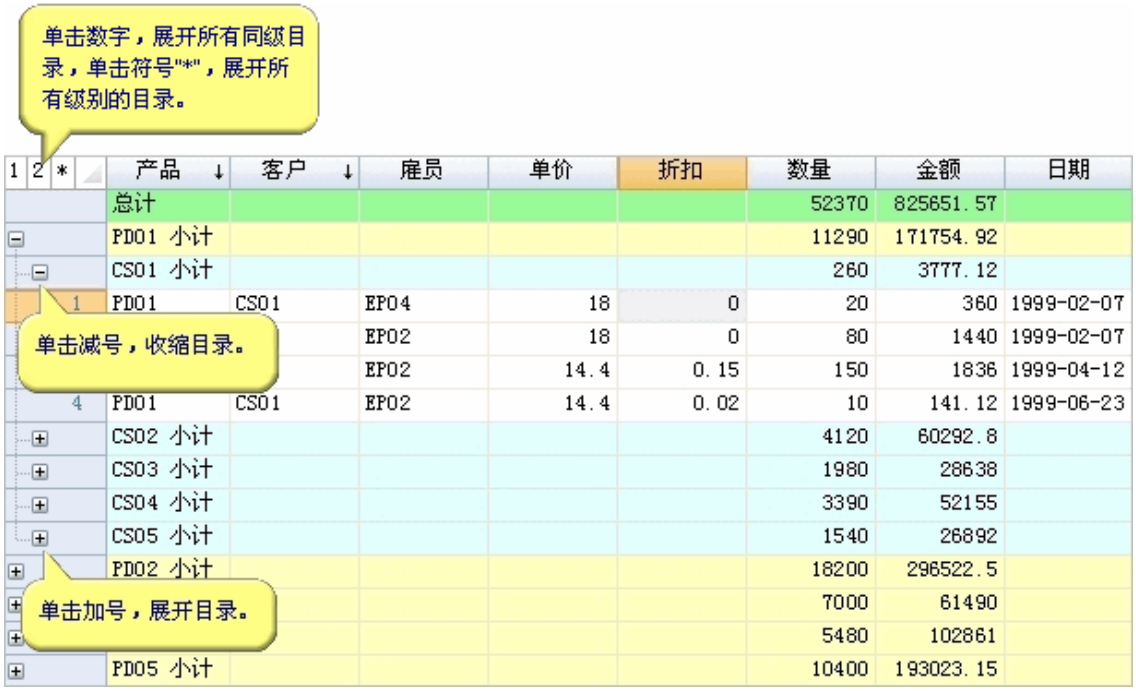
要得到上图所示的效果，操作非常简单，只需在设置汇总模式的时候，选择“分组行位于数据行之上”、“显示目录树”这两个选项即可：



提示：并非一定要选择“分组行位于数据行之上”这个选项的，只是选择了这个选项，目录树的样式会更符合我们的习惯。

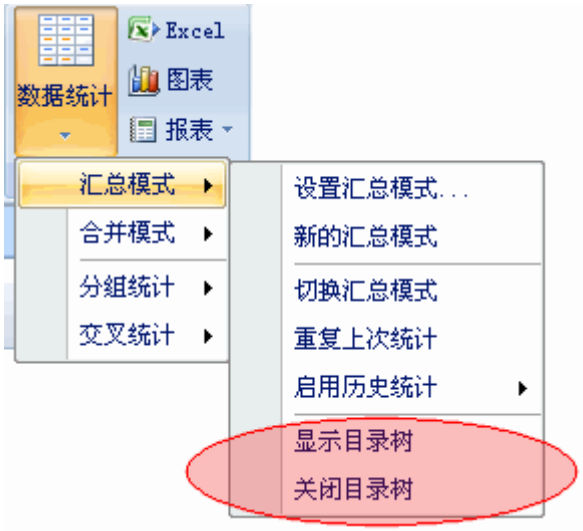
3.11.2.12 操作目录树

目录树显示在表左边的行号列，单击加、减号，可以展开和收缩目录；单击行号列顶端的数字，可以展开所有同级目录；单击符号“*”，可以展开所有目录：



1 2 *	产品	客户	雇员	单价	折扣	数量	金额	日期
	总计					52370	825651.57	
	PD01 小计					11290	171754.92	
	CS01 小计					260	3777.12	
1	PD01	CS01	EP04	18	0	20	360	1999-02-07
			EP02	18	0	80	1440	1999-02-07
			EP02	14.4	0.15	150	1836	1999-04-12
4	PD01	CS01	EP02	14.4	0.02	10	141.12	1999-06-23
	CS02 小计					4120	60292.8	
	CS03 小计					1980	28638	
	CS04 小计					3390	52155	
	CS05 小计					1540	26892	
	PD02 小计					18200	296522.5	
						7000	61490	
						5480	102861	
	PD05 小计					10400	193023.15	

此外，菜单中还有两个命令，用于关闭和显示目录树：



3.11.2.13 改变分组行的样式

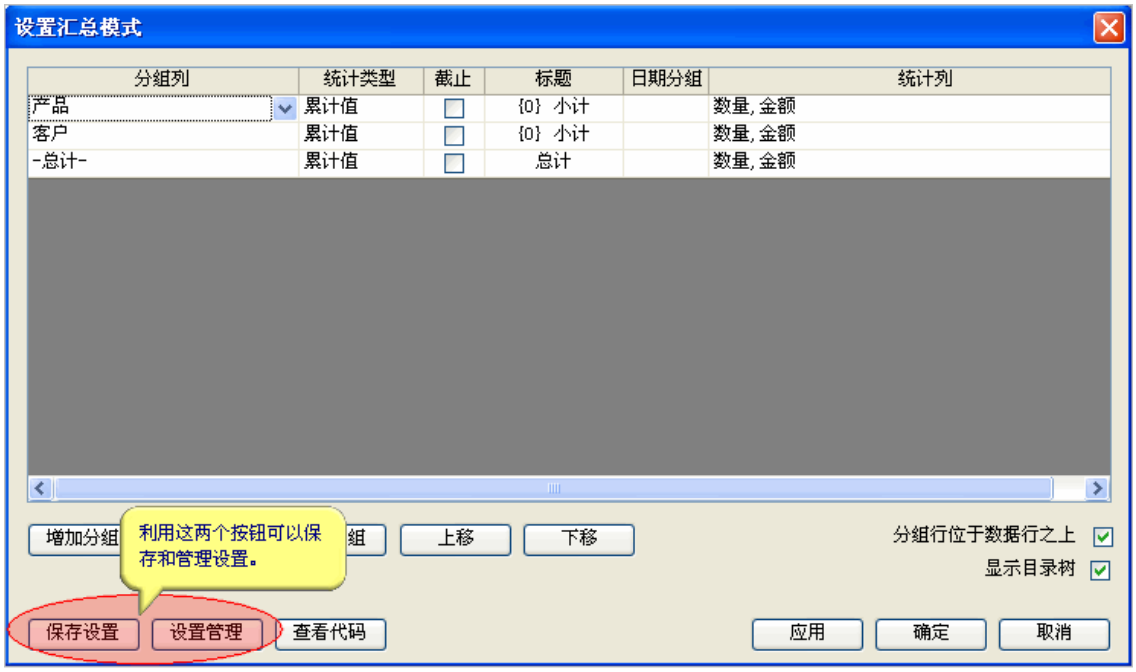
如果觉得默认的分组行背景颜色和字体颜色不符合你的要求，可以通过表样式的配色方案设置 ：



注意表样式是针对表的，不是针对整个系统的，所以你的设置只会对当前表有效。

3.11.2.14 设置管理

你可以将汇总模式的设置保存起来，便于以后快速调用，这样一些复杂的统计只需设置并保存一次，以后就可以直接调用，无需反复设置。

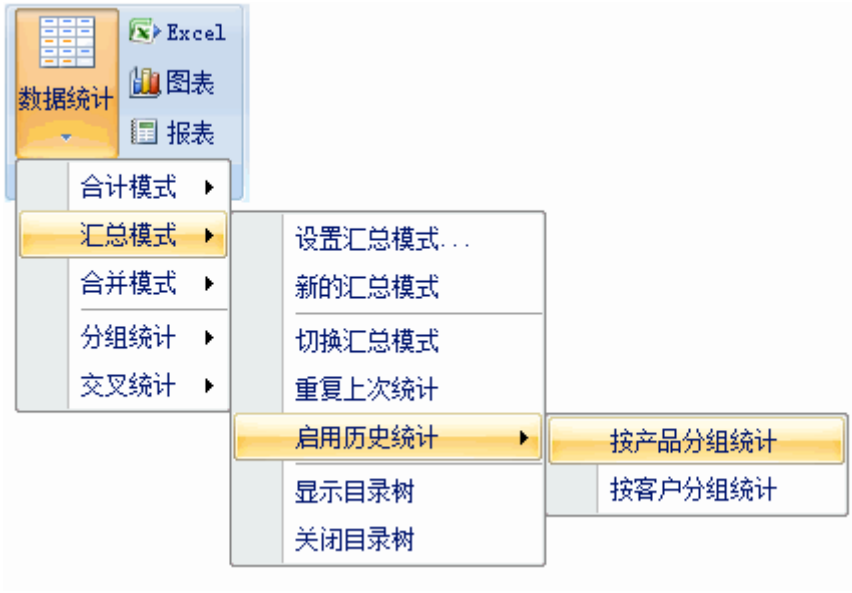


- 保存设置
  在汇总模式设置窗口，先设置好汇总模式，然后单击“保存设置”按钮，输入设置名称，单击“确定”按钮即可。

- 修改设置
  在汇总模式设置窗口，先设置好汇总模式，然后单击“保存设置”按钮，选择之前保存的设置名称，单击“确定”按钮即可覆盖原来的设置。

- 设置管理
  在汇总模式设置窗口，单击“设置管理”按钮，可以删除、重命名已经保存的设置。

- 调用设置
  在菜单的“日常工作”区，单击“数据统计”按钮，选择“汇总模式”，指向“启用历史统计”，会列出所有保存好的设置名称，单击其中一个，即可自动根据该设置进入汇总模式。



- 重复上次统计
  在上图中，单击“重复上次统计”命令，即可根据最近一次生效的汇总模式设置，重新生成汇总模式。

3.11.3 合计模式

和汇总模式相比，本节介绍的合计模式设置简单，功能单一，只有一个总的合计行。
但合计模式有个极大的优点，在此模式下，你可以正常编辑数据、增加行、删除行，还可以进行筛选和排序等操作，合计结果会即时刷新，而之前介绍的汇总模式，数据是只读的。

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
2	PD01	CS03	EP04	14.4	0.25	200	2160	1999-01-14
3	PD01	CS03	EP01	14.4	0	240	3456	1999-01-21
4	PD01	CS03	EP04				720	1999-02-07
5	PD01	CS03	EP01				5760	1999-03-03
6	PD01	CS03	EP01				1728	1999-03-14
7	PD01	CS03	EP01				2025	1999-03-16
8	PD01	CS03	EP05				1800	1999-03-24
9	PD01	CS03	EP01				1620	1999-05-27
10	PD01	CS03	EP01	18	0.25	350	4725	1999-06-02
	合计					1780	25218	

在合计模式下，可以直接编辑数据、增加、删除行，也可以筛选、排序数据，合计结果会自动更新。

合计模式和汇总模式各有优势，一个可以编辑数据，一个功能强大，我们可以根据需要选用。
在菜单的"日常工作"区，单击"数据统计"按钮，指向"合计模式"，会出现合计模式的相关菜单命令：



执行设置“设置合计模式”命令，出现下图所示的设计窗口，勾选要合计的列，单击“确定”按钮即可：



提示： 如果希望打开项目后，能够自动进入合计模式，可以勾选选项“自动进入合计模式”。

3.11.4 合并模式

合并模式是一种清晰诱人的数据显示方式，在普通的电子表格软件中，也许要经过无数次合并单元格操作才能得到类似的效果，而在 Foxtable 中，这一切都是自动的。下图就是一个处于合并模式下的表：

	国家	金属	类型	数量
1	美国	白银	出口	70
2			进口	321
3		黄金	出口	78
4			进口	789
5	日本	白银	出口	123
6			进口	167
7		黄金	出口	567
8			进口	400
9	中国	白银	出口	70
10			进口	120
11		黄金	出口	70
12			进口	130

在菜单的"日常工作"区，单击"数据统计"按钮，指向"合并模式"，会出现合并模式的相关菜单命令：



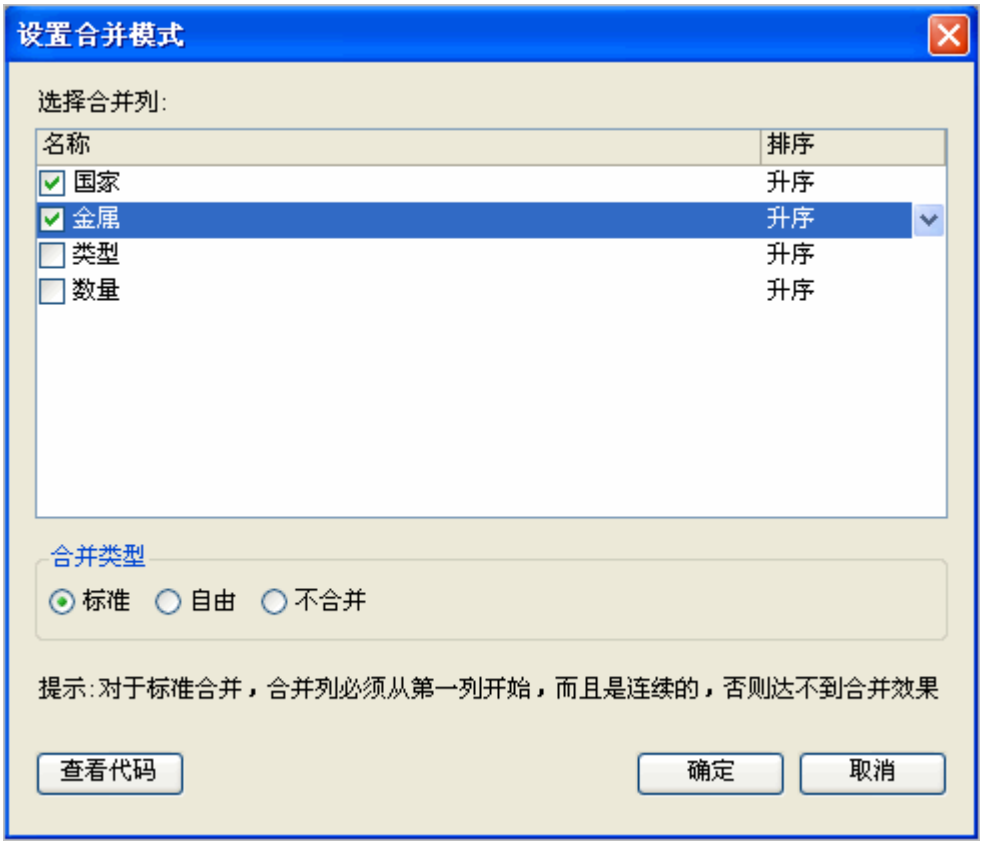
提示：

1、如果要设置新的合并模式，请执行“设置合并模式”命令。

2、如果执行“切换合并模式”命令，可以反复在合并模式和普通模式之间切换。

设置合并模式

执行“设置合并模式”命令后，会出现下图所示的合并模式设置窗口，选择要合并的列，单击“确定”按钮即可：



在生成合并模式的过程中，程序会自动根据合并列排序，并将合并列内容相同且相邻的单元格合并为一个单元格显示。

合并类型

你也许已经留意到，在合并模式设置窗口有两种合并类型供选择：

- 标准
  相邻单元格是否合并，除了内容相同外，左边的单元格还必须已经合并。

- 自由
  只需内容相同，相邻单元格即可合并。

例如下面两个图，同样是选择合并国家和金属两列，但是因为合并类型的不同，得到了不同的结果：

	国家	类型	金属	数量
1	美国	出口	白银	70
2		进口	白银	321
3		出口	黄金	78
4		进口	黄金	789
5	日本	出口	白银	123
6		进口	白银	167
7		出口	黄金	567
8		进口	黄金	400
9	中国	出口	白银	70
10		进口	白银	120
11		出口	黄金	70
12		进口	黄金	130

标准合并

	国家	类型	金属	数量
1	美国	出口	白银	70
2		进口		321
3		出口	黄金	78
4		进口		789
5	日本	出口	白银	123
6		进口		167
7		出口	黄金	567
8		进口		400
9	中国	出口	白银	70
10		进口		120
11		出口	黄金	70
12		进口		130

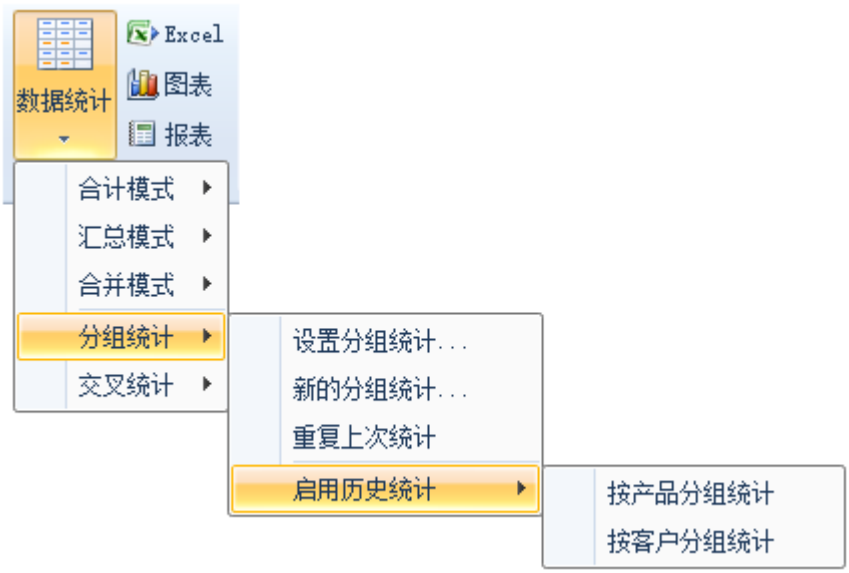
自由合并

尽管自由合并能让你合并中间的列，但我还是觉得将要合并的列全部移到左边较好，没有别的原因，只是这样看上去比较美观。

3.11.5 分组统计

3.11.5.1 设置分组统计

汇总模式直接在原表中显示统计结果，而分组统计会生成一个新表，在这个新表中显示统计结果。在菜单的“日常工作”区，单击“数据统计”按钮，指向“分组统计”，会出现分组统计的相关菜单命令：



提示：

1、执行“设置分组统计”，会打开分组统计设置窗口，并加载最近一次统计的设置。

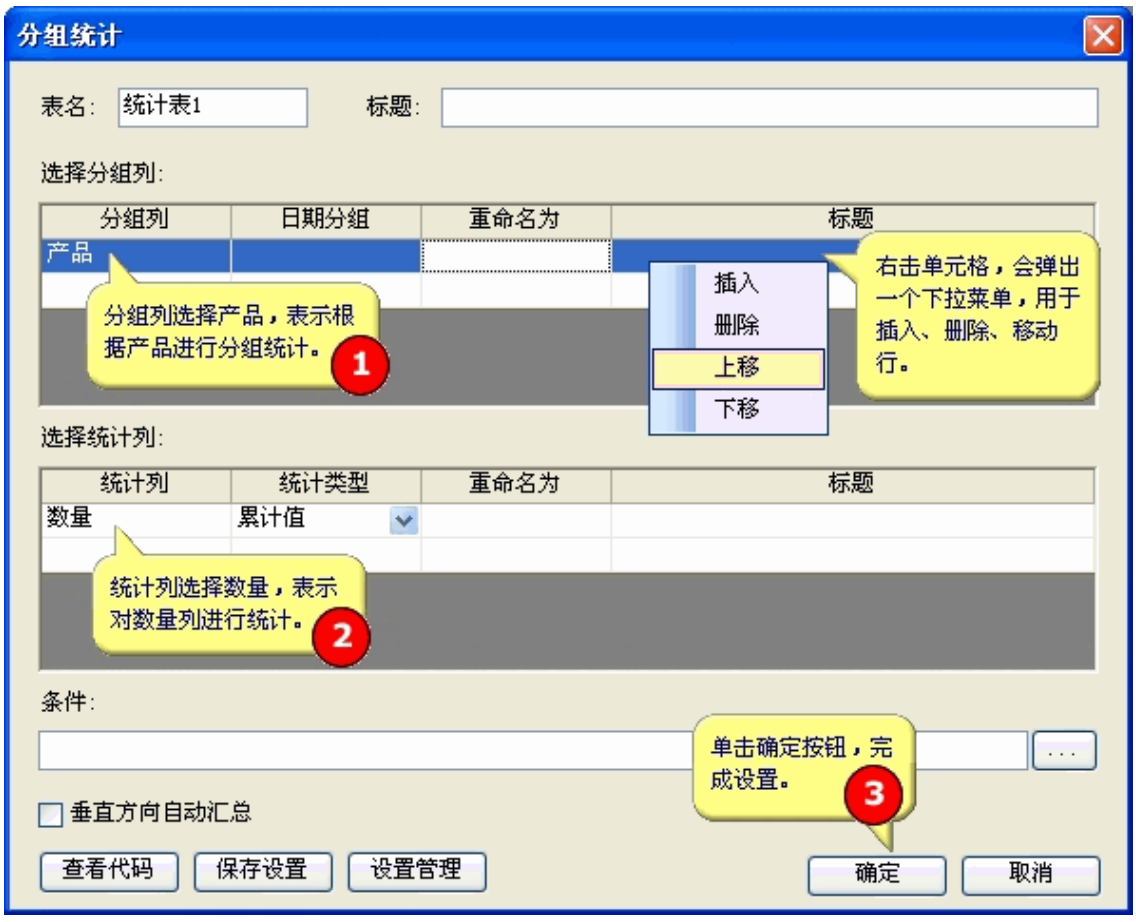
2、执行“新的分组统计”，会打开分组统计设置窗口，不加载最近一次统计的设置。

3、执行“重复上次统计”，可以根据最近一次的设置，重新生成分组统计表。

设置分组统计

首先我们看看如果进行简单的分组统计，任务是统计不同产品的销售数量：

1、在分组统计设置窗口，按下图所示设置：



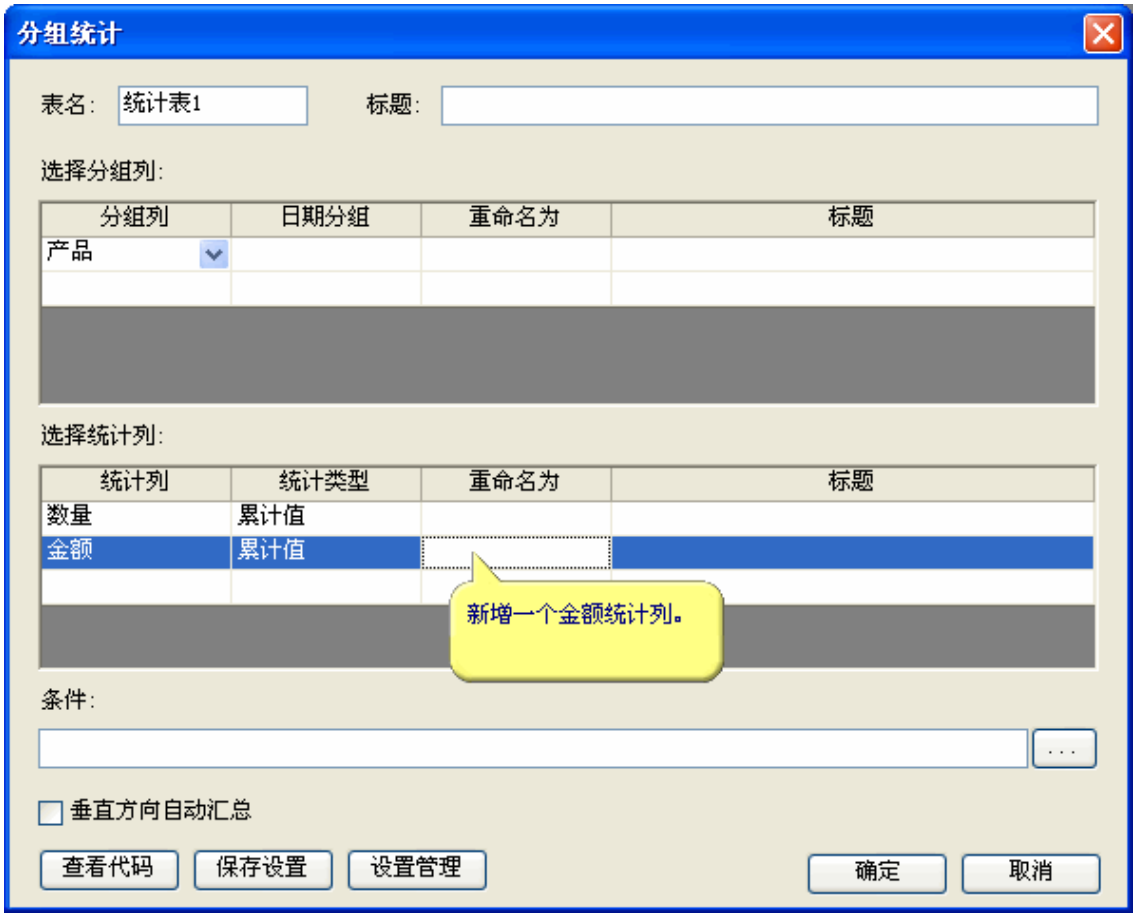
2、单击“确定”按钮后，生成一个新表，表中数据就是各产品的销售量：

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

可见，分组统计非常简单，我们从头到尾只是单击了几次鼠标而已。
分组统计生成的统计表是临时的，重新打开项目后，这些表将不复存在。

3.11.5.2 对多列进行统计

现在我们不仅希望得到不同产品的销售数量，也希望同时统计出销售金额。
再次单击分组按钮，在上次设置的基础上，增加一个统计列“金额”，统计类型还是“累计值”：



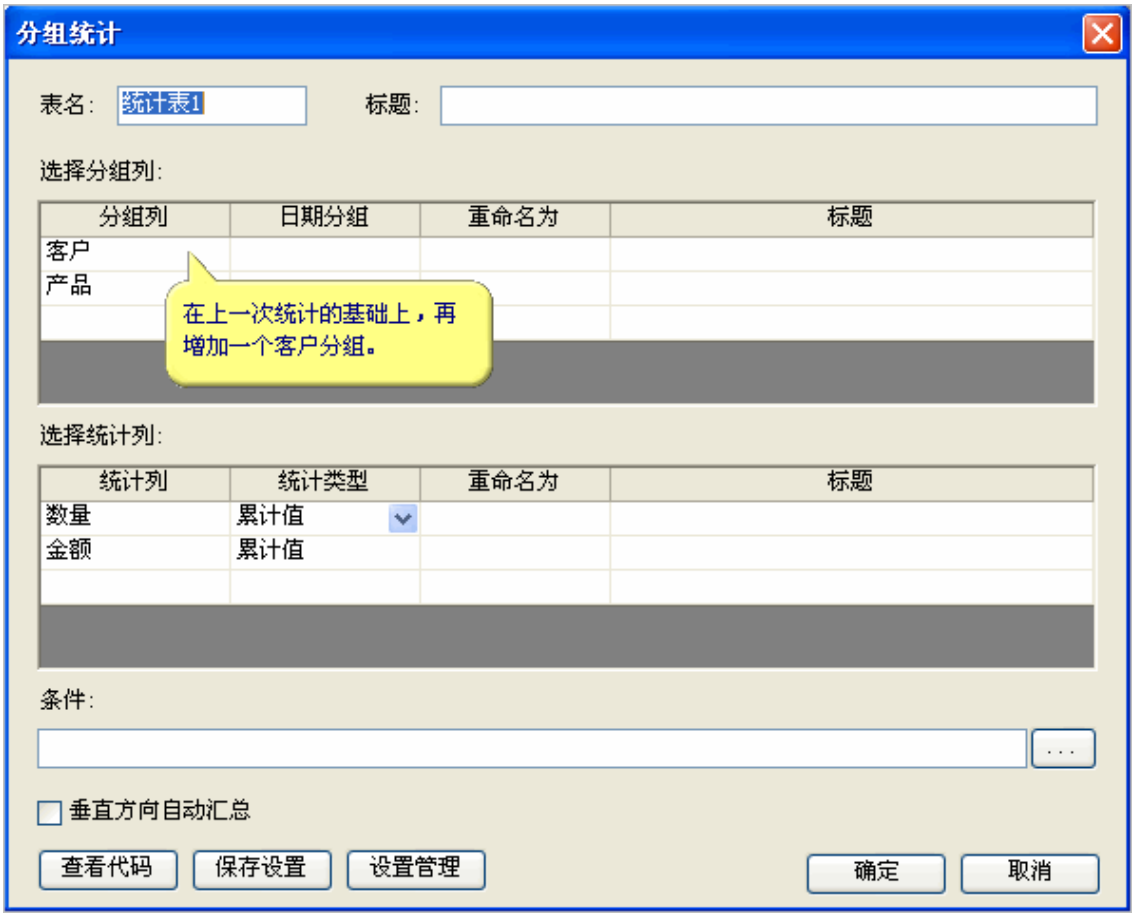
这是统计结果：

	产品	数量	金额
1	PD01	11290	171754.92
2	PD02	18200	296522.5
3	PD03	7000	61490
4	PD04	5480	102861
5	PD05	10400	193023.15

3.11.5.3 多个分组列

可以同时根据多列进行分组统计。

例如希望统计出每个客户订购不同产品的数量和金额，我们只需在上一节的基础上，再增加一个客户分组列即可：



这是统计结果：

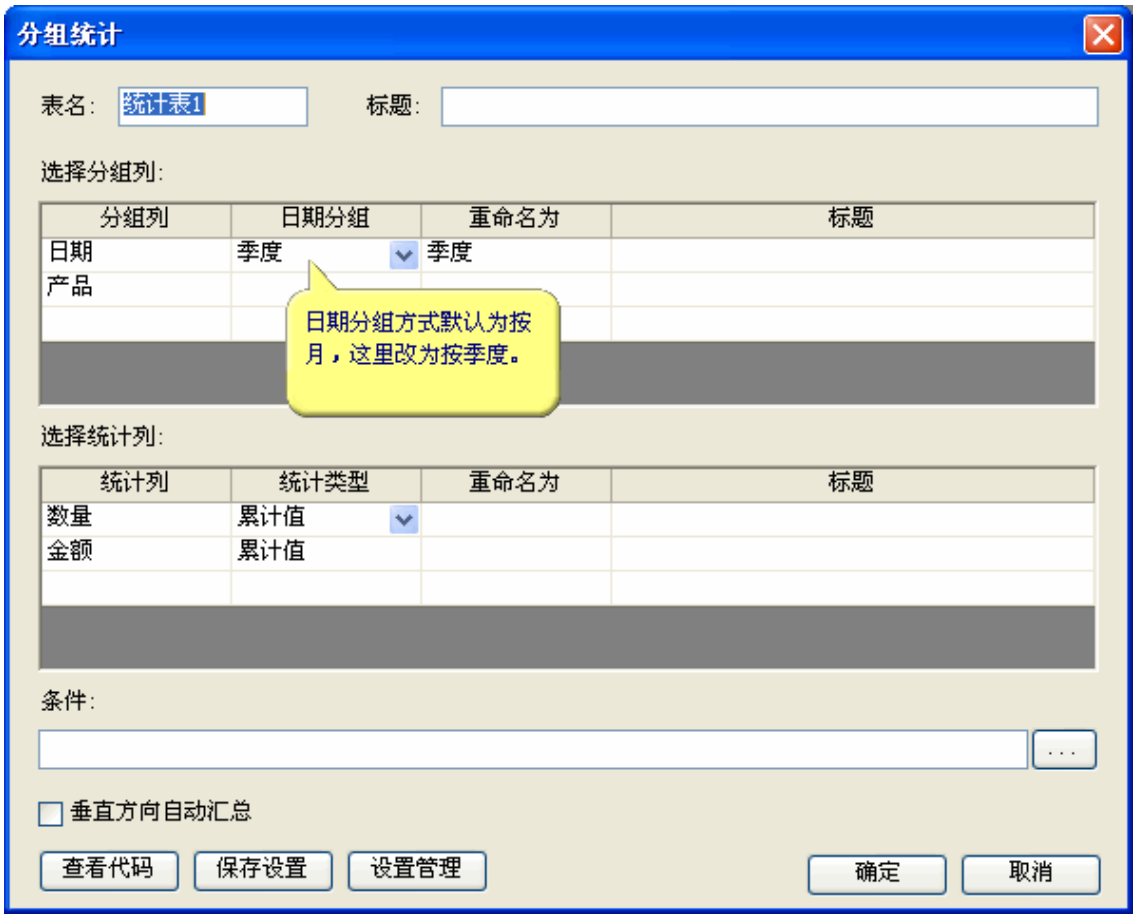
	客户	产品	数量	金额
1	CS01	PD01	260	3777.12
2	CS01	PD02	2250	34814.6
3	CS01	PD03	2180	17720
4	CS01	PD04	230	3951.2
5	CS01	PD05	600	12338.7
6	CS02	PD01	4120	60292.8
7	CS02	PD02	4300	64634.2
8	CS02	PD03	2250	18940
9	CS02	PD04	1830	32414.8
10	CS02	PD05	2550	45418.05
11	CS03	PD01	1980	28638
12	CS03	PD02	6300	106352.5
13	CS03	PD03	970	9680
14	CS03	PD04	1200	23320
15	CS03	PD05	2420	43467.75
16	CS04	PD01	3390	52155
	CS04	PD02		

3.11.5.4 根据日期进行分组

在日常的数据管理中，我们经常要对不同时段之间的数据进行比较分析，例如不同月份、不同季度之间的数据比较。

对于这种统计，我们同样单击几次鼠标即可完成。

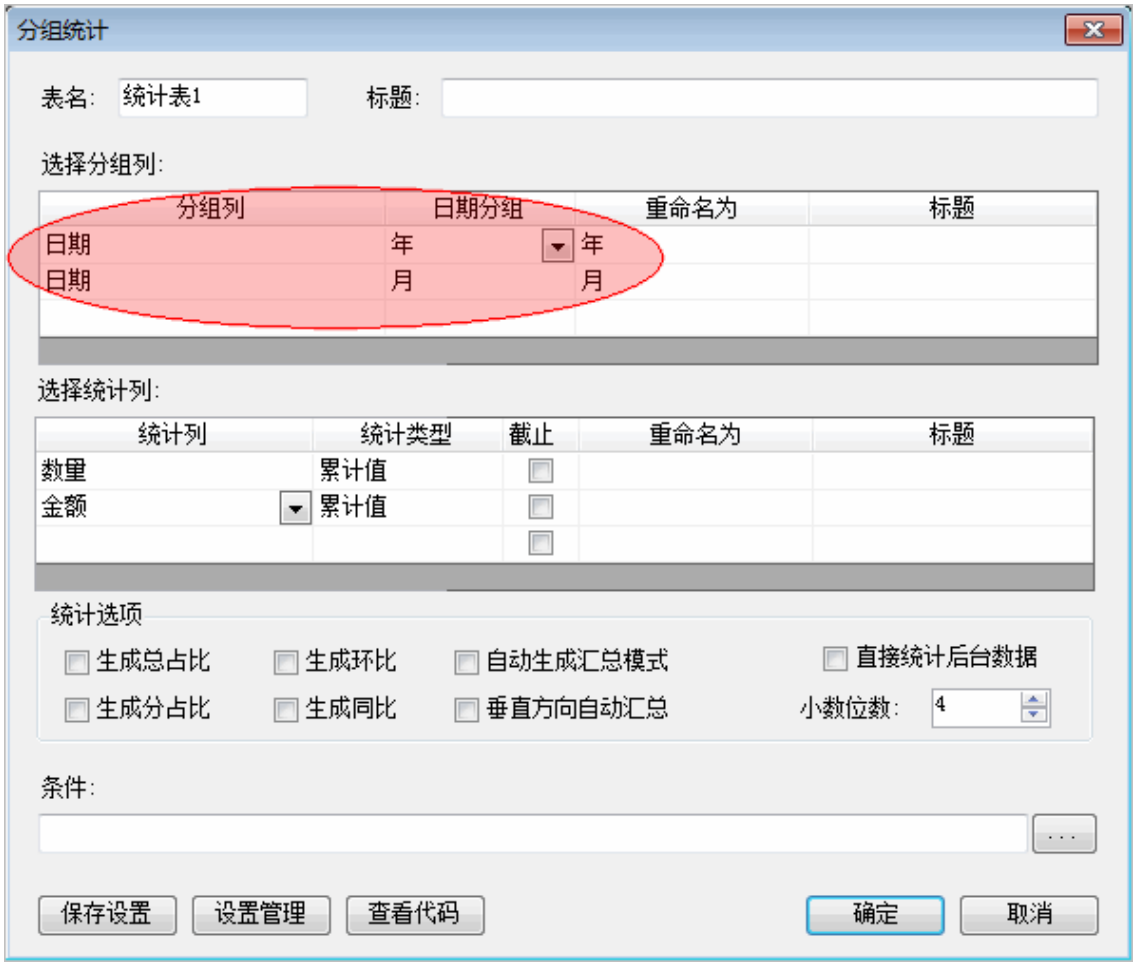
例如要按季度统计不同产品的销售数量和金额，可按下图所示进行分组统计：



下面是统计结果：

	季度	产品	数量	金额
1	1	PD01	6150	92196
2	1	PD02	6510	106746.7
3	1	PD03	4320	36960
4	1	PD04	2490	49214
5	1	PD05	5500	102134.25
6	2	PD01	5140	79558.92
7	2	PD02	11690	189775.8
8	2	PD03	2680	24530
9	2	PD04	2990	53647
10	2	PD05	4900	90888.9

再例如，我们希望按年、月统计不同产品的销售数量和金额：



得到的统计结果为：

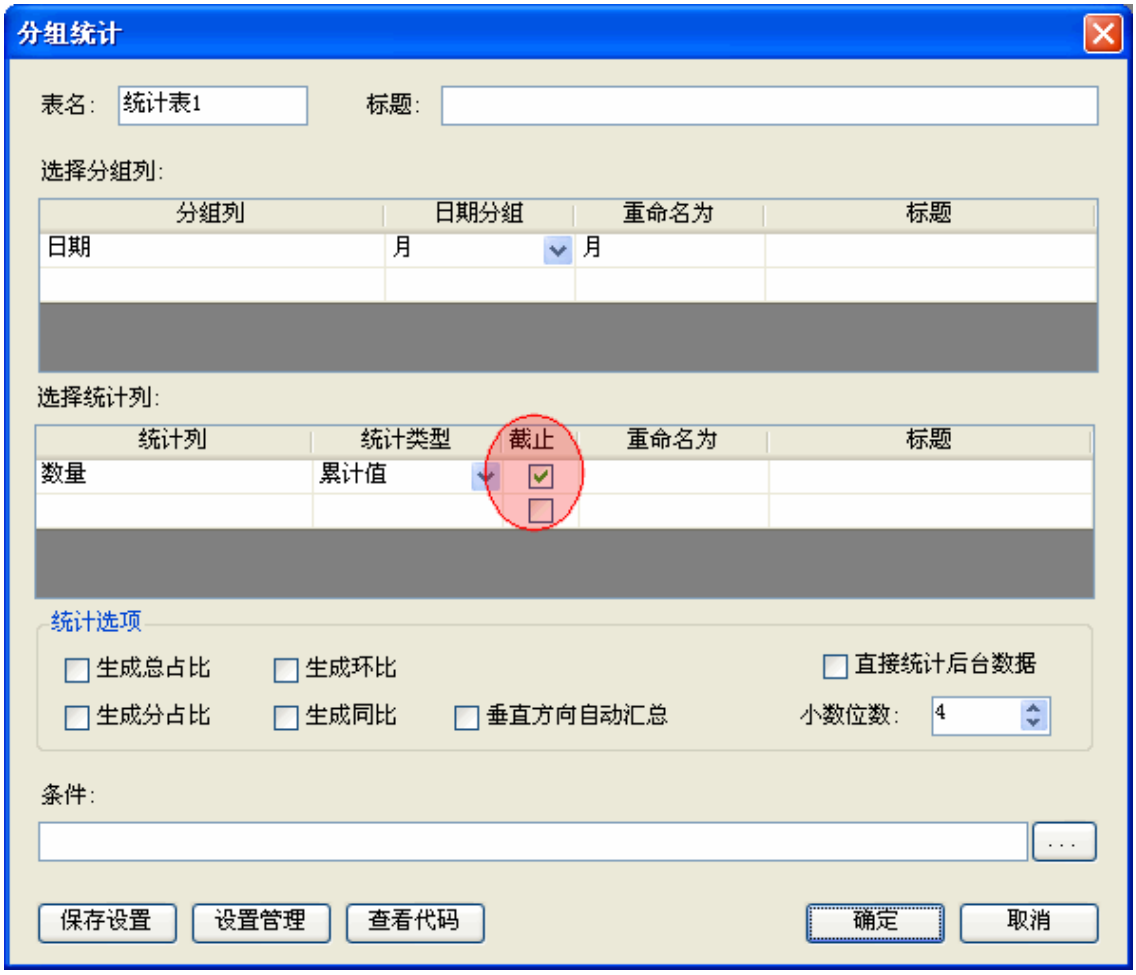
	年	月	数量	金额
1	1999	1	7560	104923.5
2	1999	2	8990	150142.4
3	1999	3	8420	132185.05
4	1999	4	10910	168037.2
5	1999	5	10530	170795
6	1999	6	5960	99568.42

3.11.5.5 截止统计

分组统计默认是各个分组独自统计，可是有的时候，我们需要的并非是每个分组的统计结果，而是截止到每个分组时的统计结果。

例如在对订单表进行统计的时候，我们希望得到截止到每个月的时候，总的订购数量。

要得到这样的统计结果非常简单，只需在设置统计列的时候，选择选项“截止”即可：

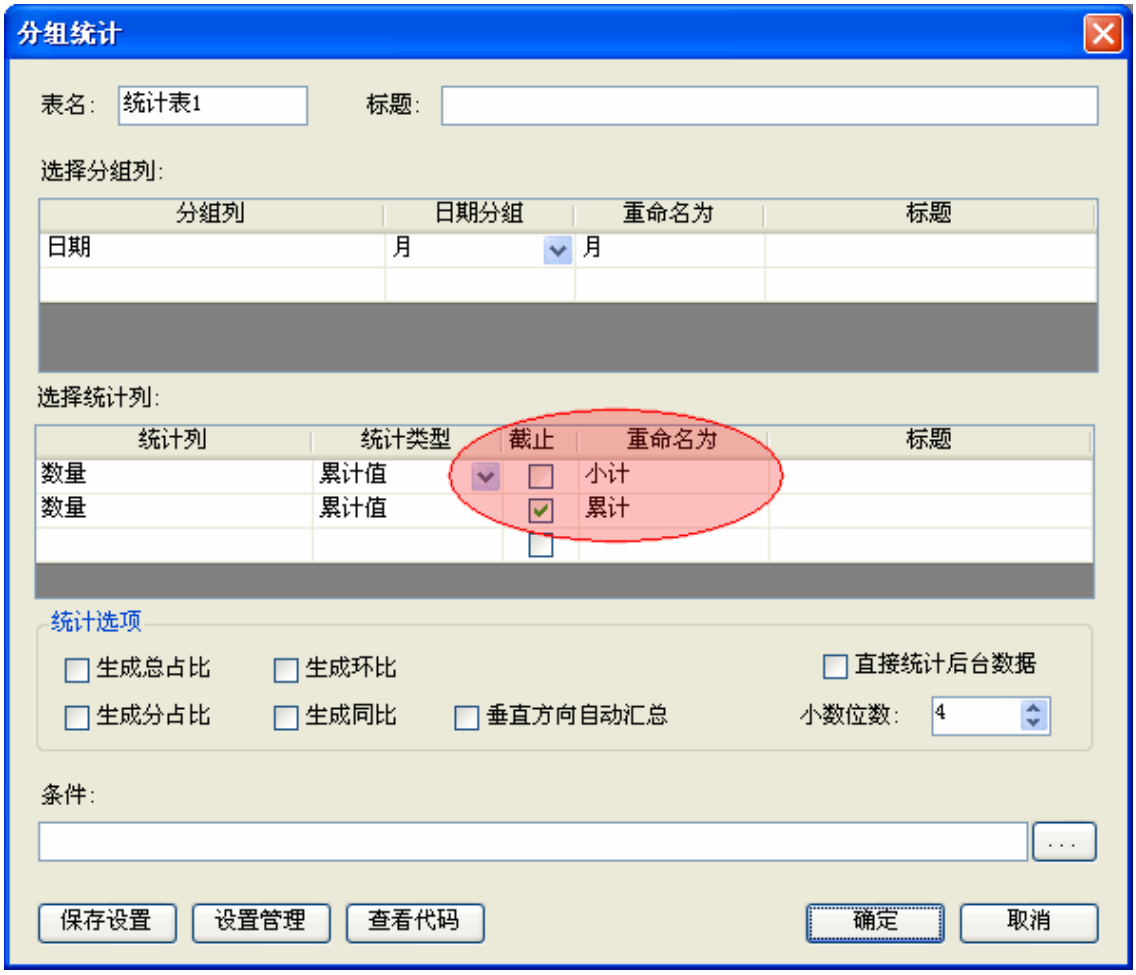


这是得到的统计结果：

	月	数量
1	1	1240
2	2	1690
3	3	2890
4	4	4280
5	5	5110
6	6	6010

注意这并非3月份的销售数量，而是截止到3月份的累计销售数量。

如果你希望同时得到每月销量和截止到该月的累计销量，可以按下图所示设置：

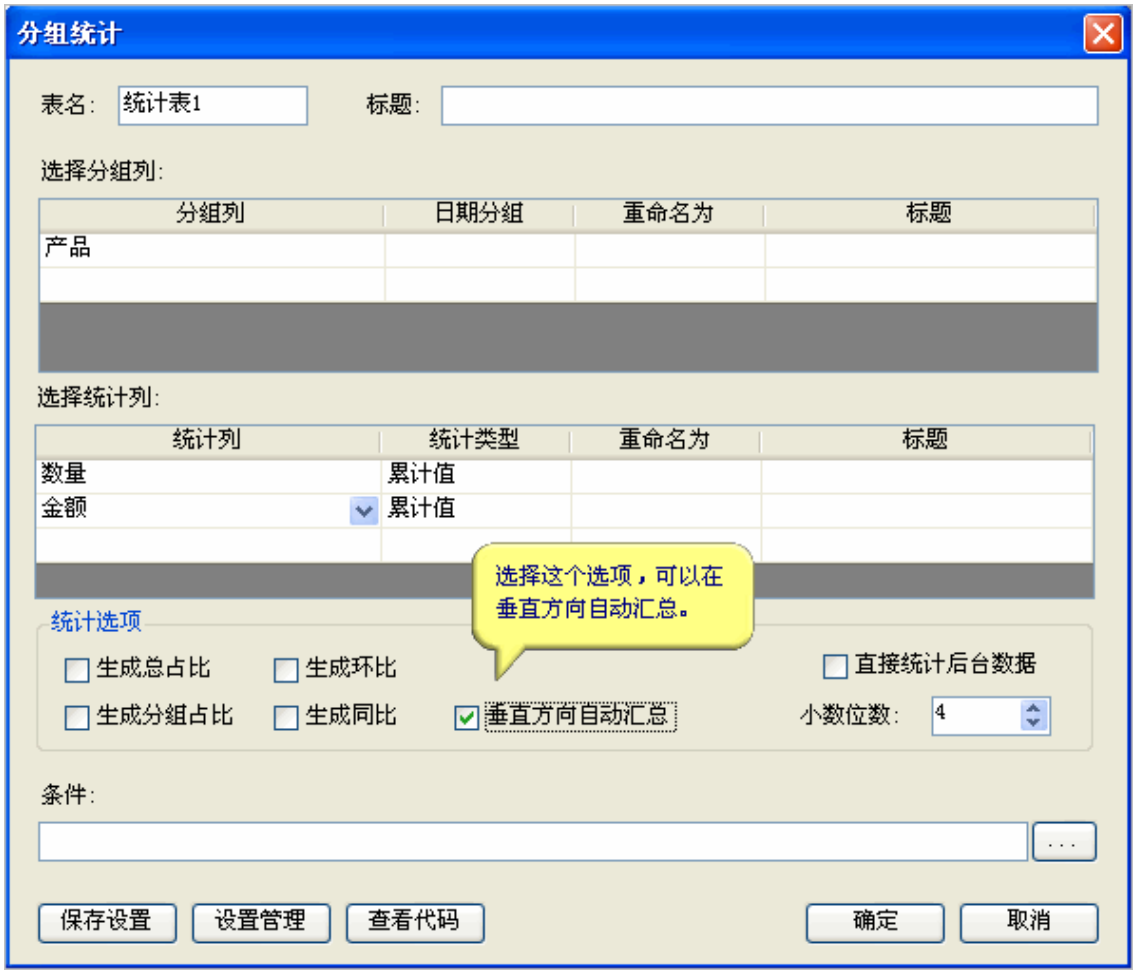


这是统计结果：

	月	小计	累计
1	1	1240	1240
2	2	450	1690
3	3	1200	2890
4	4	1390	4280
5	5	830	5110
6	6	900	6010

3.11.5.6 垂直汇总

在设置分组统计的时候，如果选择选项“垂直方向自动汇总”，会在统计表的最后面增加一行，用于显示每一列的合计值：



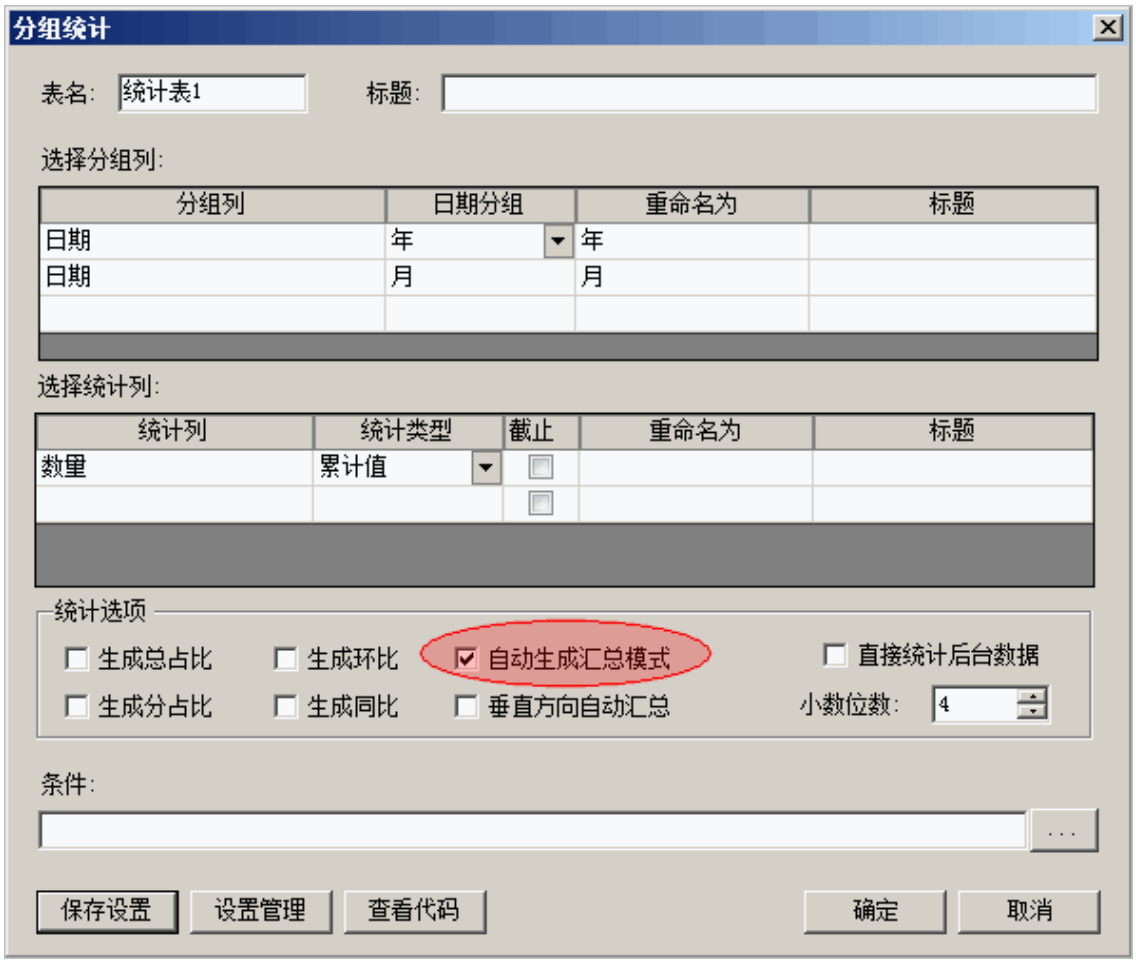
这是统计结果，可以看到后面多了一个合计行：

	产品	数量	金额
1	PD01	11290	171754.92
2	PD02	18200	296522.5
3	PD03	7000	61490
4	PD04	5480	102861
5	PD05	10400	193023.15
6	合计	52370	825651.57

3.11.5.7 汇总模式和分组统计

在设置分组统计的时候，如果选择选项“自动生成汇总模式”，会自动对生成的统计表进行二次统计，生成汇总模式。

例如在按年月分组统计销售数量的时候，选择选项“自动生成汇总模式”:



会得到下图所示的统计结果，不仅统计出了每月数量，还利用汇总模式，统计出了每年数量和总的数量：

	年 ↓	月	数量
1	1999	1	41426
2	1999	2	44734
3	1999	3	42236
4	1999	4	43971
5	1999	5	39369
6	1999	6	39201
7	1999	7	39295
8	1999	8	51104
9	1999	9	51236
10	1999	10	33347
11	1999	11	41818
12	1999	12	37859
	小计 1999		505596
13	2000	1	44005
14	2000	2	29402
15	2000	3	39801
16	2000	4	41544
17	2000	5	42099
18	2000	6	34300
19	2000	7	43653
20	2000	8	46453
21	2000	9	37710
22	2000	10	47441
23	2000	11	38726
24	2000	12	39758
	小计 2000		484892
	总计		990488

3.11.5.8 同一统计列进行多种统计

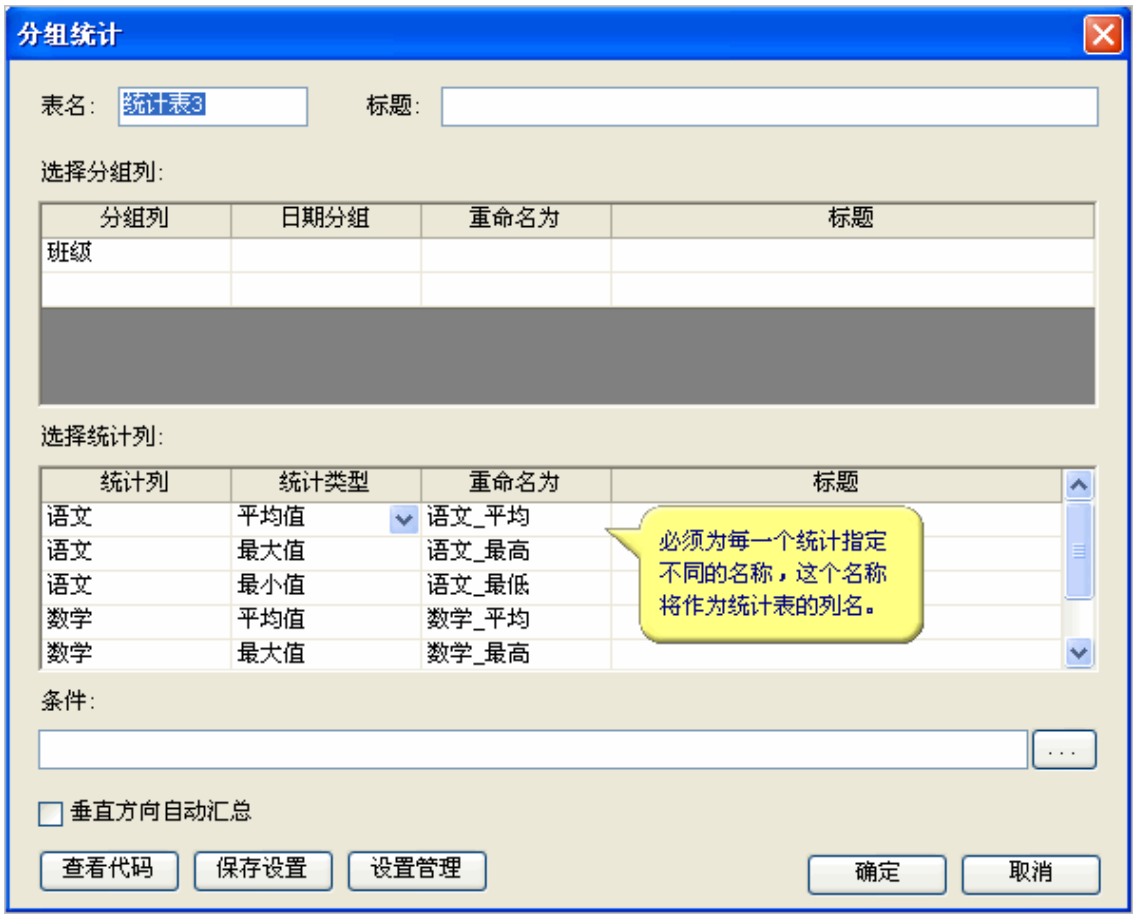
之前每一种分组都只进行了一种统计，就是累计值。

其实 Foxtable 提供了很多统计方式，包括：累计值、平均值、最大值、最小值、记录数、百分比、标准偏差、总体标准偏差、采样方差、总体方差。

更重要的是，即使是同一个统计列，也可以进行多种不同类型的统计。

需要注意的是，如果同一个统计列要进行多种统计，必须在设置的时候，为每一种统计指定新的列名。

打开学生成绩表，我们按班级来统计每一科目的平均分、最高分、最低分，按下图所示设置分组统计即可：



这是统计结果，我们巧妙地利用列名实现了多层表头：

	班级	语文			数学		
		平均	最高	最低	平均	最高	最低
1	1班	122	133	110	105	136	38
2	2班	122	134	110	108	136	55
3	3班	110	120	100	95	120	70
4	4班	121	134	110	108	136	55

你应该已经留意到，设置分组统计的时候，分组列和统计列不仅可以重新命名，还可以设置标题，所以你也可用列标题来实现多层表头，只要你喜欢。

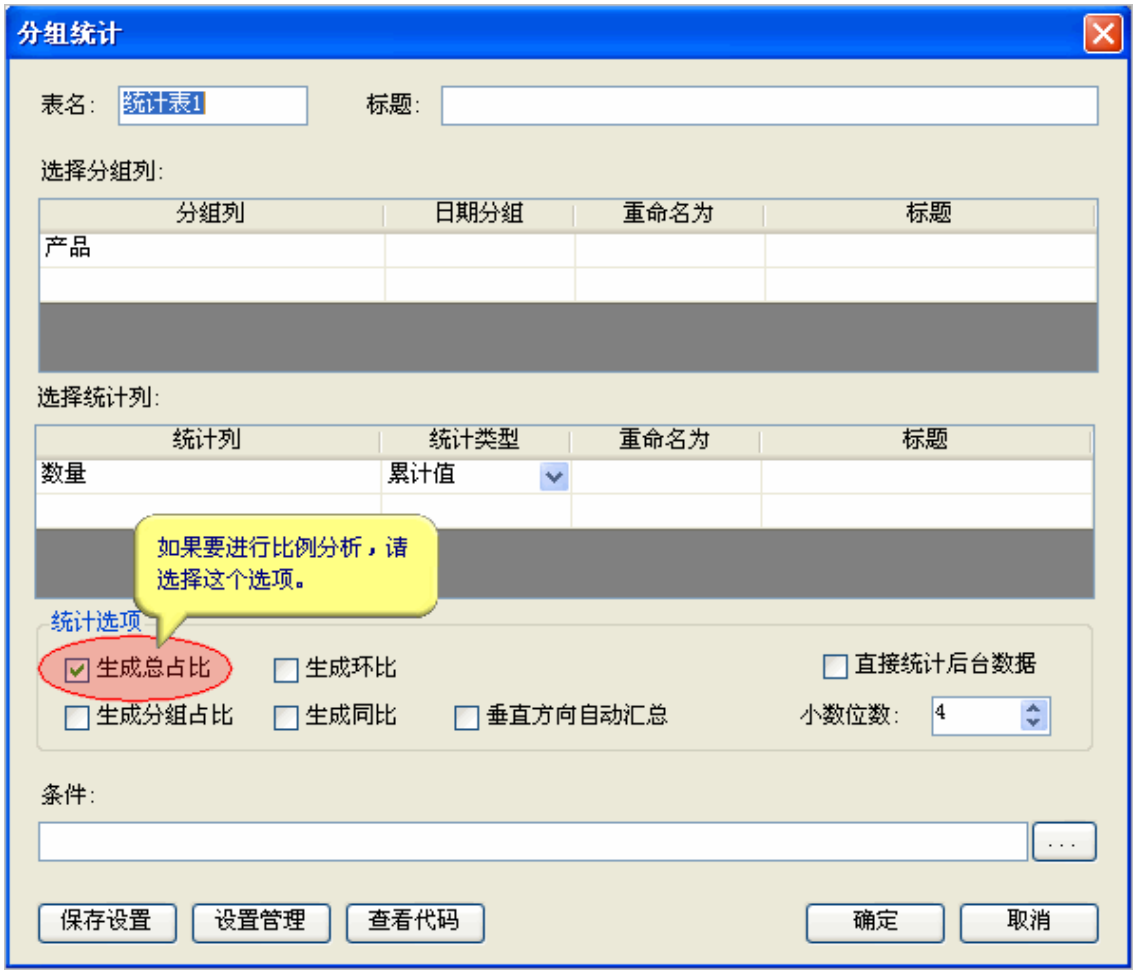
3.11.5.9 分组统计之数据分析

分组统计提供了很多辅助数据分析功能。

在学习本节内容之前，请打开 CaseStudy 目录下的文件：数据分析.Table

占比统计

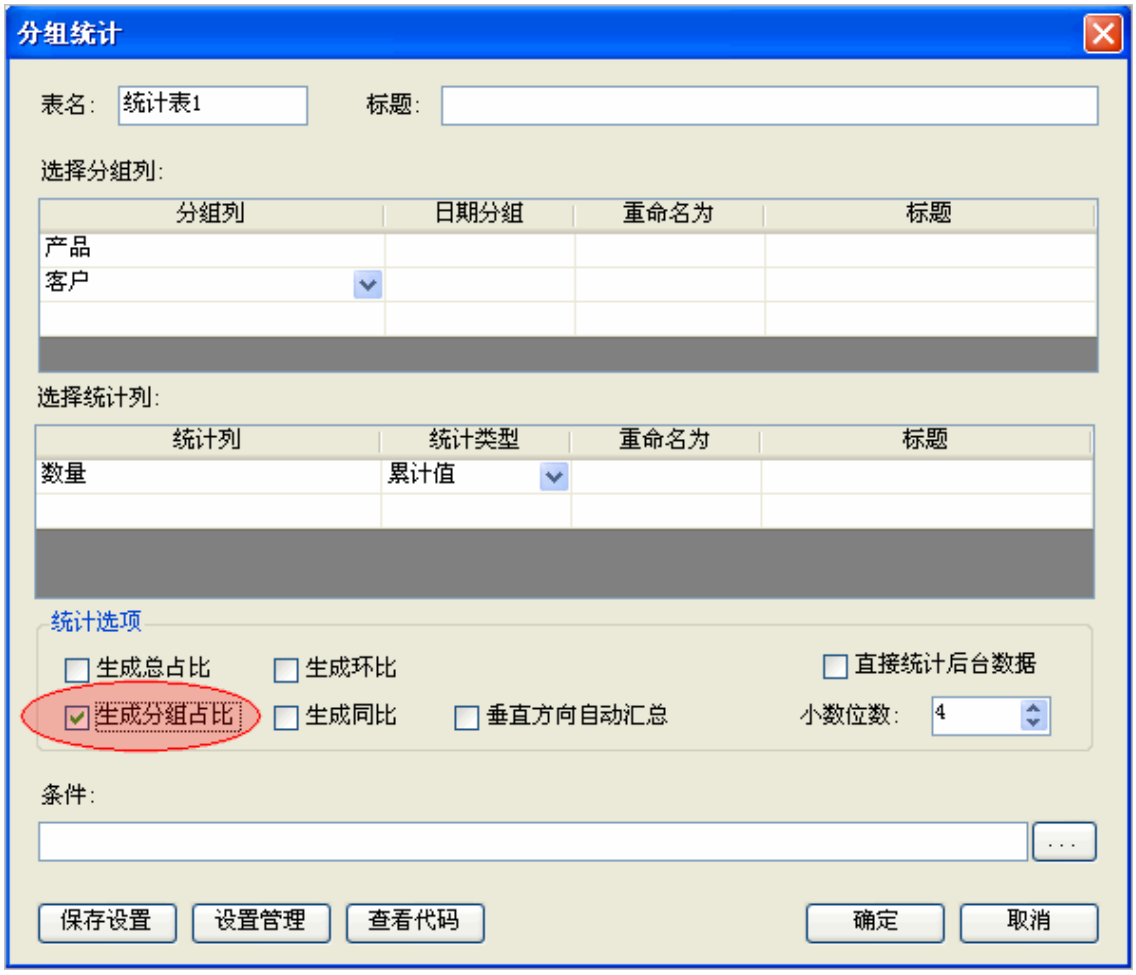
如果在设置窗口选择选项“生成总占比”：



生成的统计表会自动加上一列，用于显示每个统计值在总量中占的百分比，例如下图不仅统计出了每个产品的销售数量，还统计出了各产品在总销售数量中占的比例：

	产品	数量	占比
1	PD01	11290	21.58%
2	PD02	18200	34.79%
3	PD03	7000	13.38%
4	PD04	5480	10.47%
5	PD05	10351	19.78%

不仅可以生成总的占比，在有多个分组的情况下，选择选项“生成分组占比”，可以生成每个统计值在上级分组中所占的比重：



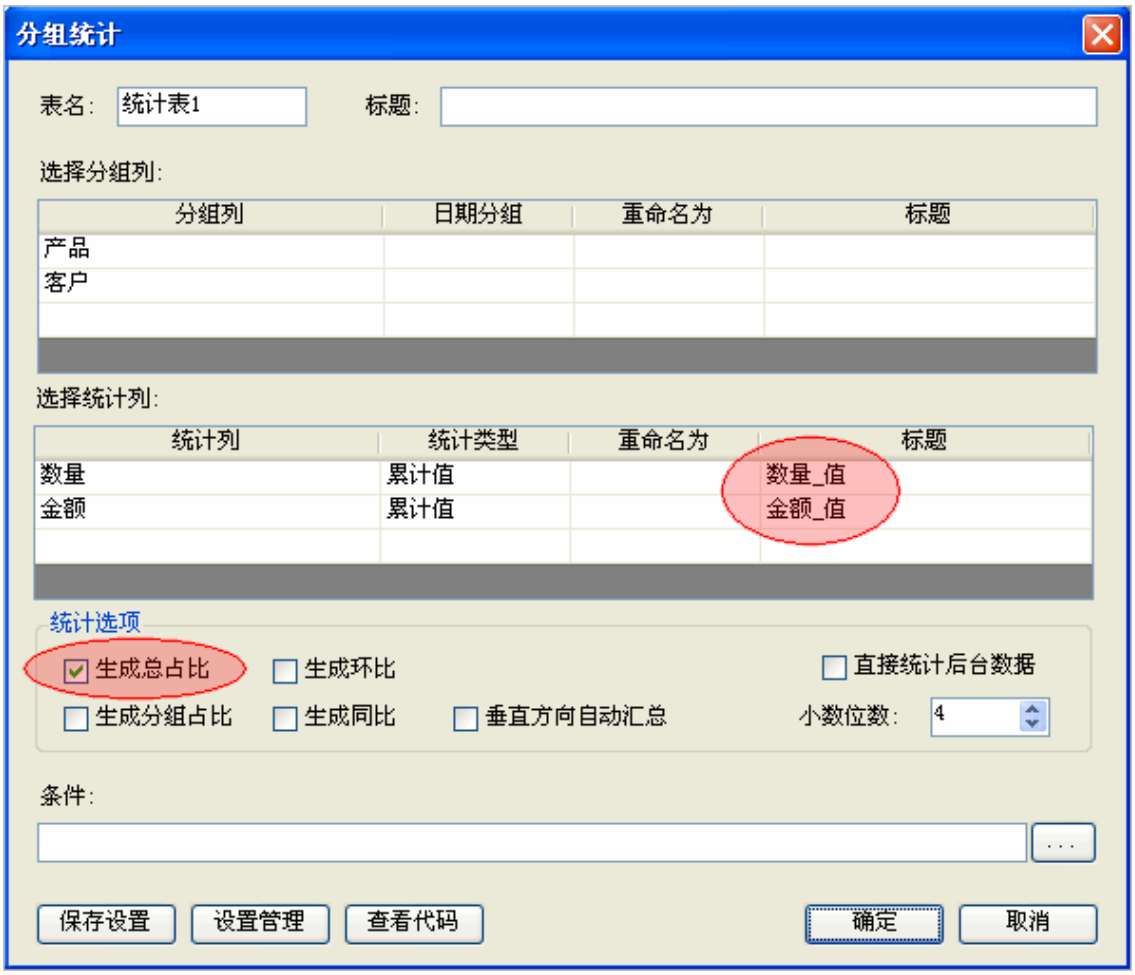
例如下图不仅统计出了每个产品不同客户的订购数量，还统计出了每个客户订购某个产品的数量，在该产品总订购数量中占的比重：

	产品	客户	数量	占比
1	PD01	CS01	260	2.30%
2	PD01	CS02	4120	36.49%
3	PD01	CS03	1980	17.54%
4	PD01	CS04	3390	30.03%
5	PD01	CS05	1540	13.64%
6	PD02	CS01	2250	12.36%
7	PD02	CS02	4300	23.63%
8	PD02	CS03	6300	34.62%
9	PD02	CS04	3110	17.09%
10	PD02	CS05	2240	12.31%
11	PD03	CS01	2180	31.14%
12	PD03	CS02	2250	32.14%
13	PD03	CS03	970	13.86%

当然你可以选择同时生成分组占比和总占比：

	产品	客户	数量	占比	
				产品	总
1	PD01	CS01	260	2.30%	0.50%
2	PD01	CS02	4120	36.49%	7.87%
3	PD01	CS03	1980	17.54%	3.78%
4	PD01	CS04	3390	30.03%	6.48%
5	PD01	CS05	1540	13.64%	2.94%
6	PD02	CS01	2250	12.36%	4.30%
7	PD02	CS02	4300	23.63%	8.22%
8	PD02	CS03	6300	34.62%	12.04%
9	PD02	CS04	3110	17.09%	5.94%
10	PD02	CS05	2240	12.31%	4.28%

如果同时统计多列的占比,通过适当地设计统计列标题，可以让生成的统计表更为美观，例如：

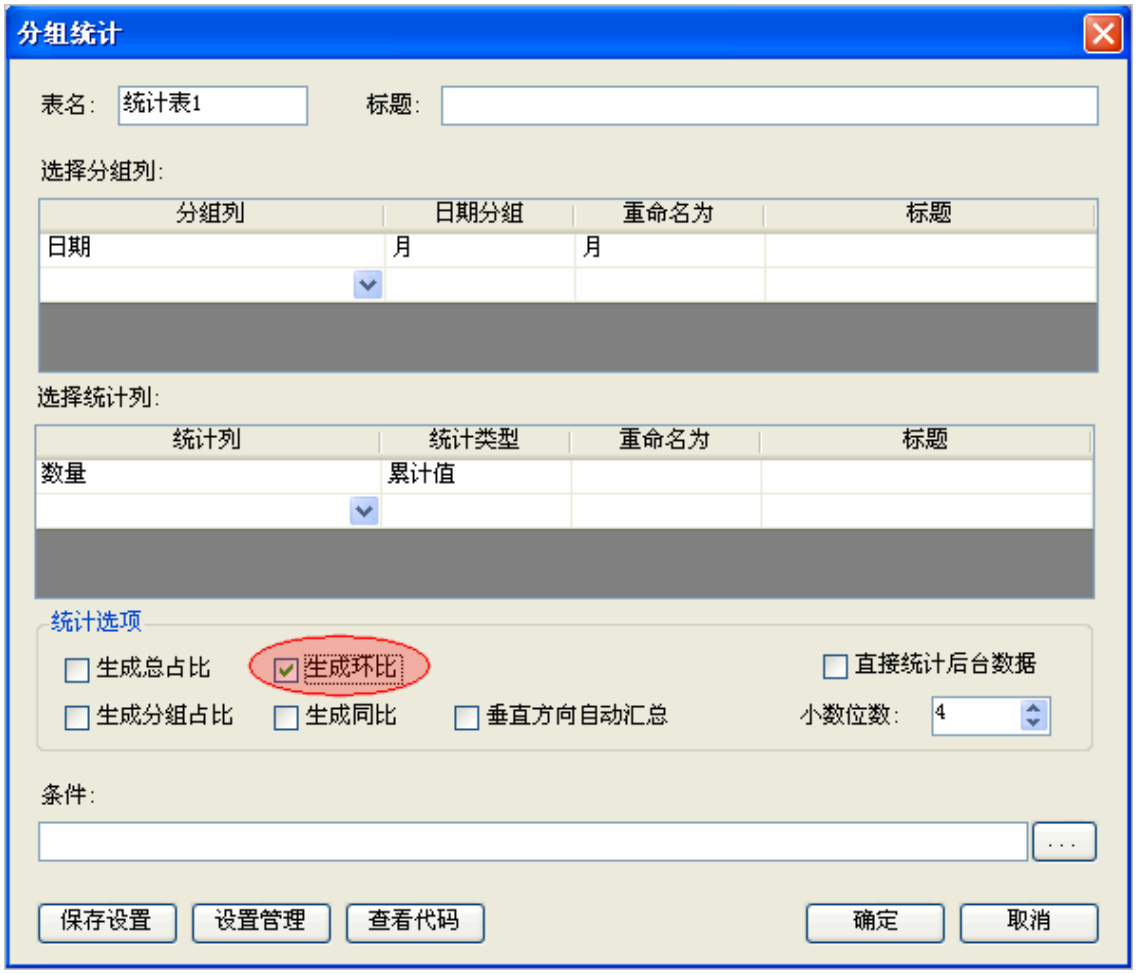


这是生成的统计表：

	产品	数量		金额	
		值	占比	值	占比
1	PD01	11290	21.58%	171754.92	20.82%
2	PD02	18200	34.79%	296522.5	35.95%
3	PD03	7000	13.38%	61490	7.45%
4	PD04	5480	10.47%	102861	12.47%
5	PD05	10351	19.78%	192190.15	23.30%

增长率统计

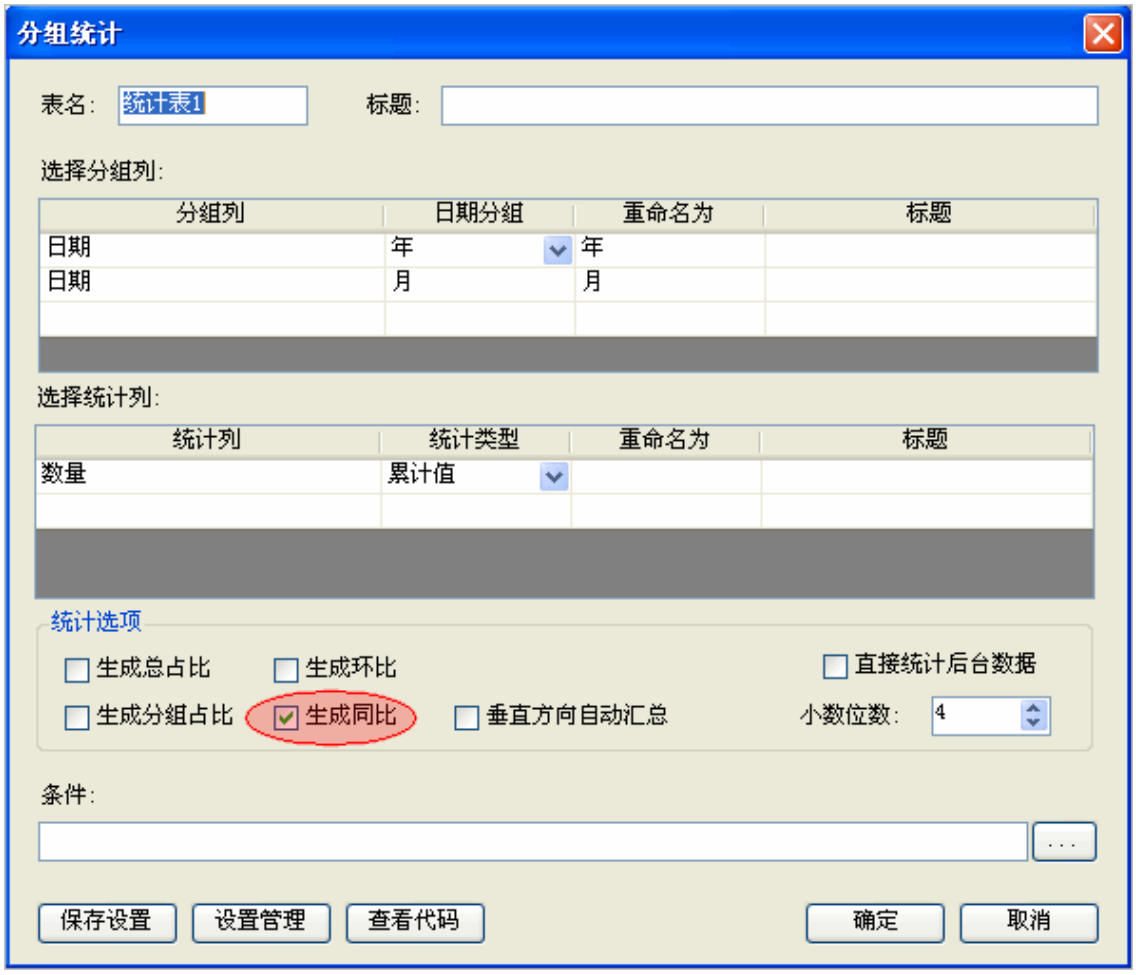
在根据日期进行分组统计的时候，选择选项“生成环比”：



则会在统计表自动增加一列，用于显示相对于上个时段的环比增长率，例如下表不仅统计出了每月的销售数量，还统计出了每月相对上月的增长率：

	月	数量	环比
1	1	7511	
2	2	8990	19.69%
3	3	8420	-6.34%
4	4	10910	29.57%
5	5	10530	-3.48%
6	6	5960	-43.40%

如果同时有多个日期分组，例如同时根据日期的年和月分组统计，那么可以选择选项“生成同比”：



则会在统计表自动增加一列，用于显示相对于去年同期的同比增长率，例如下表不仅统计出了每月的销售数量，还统计出了相对去年同月的增长率：

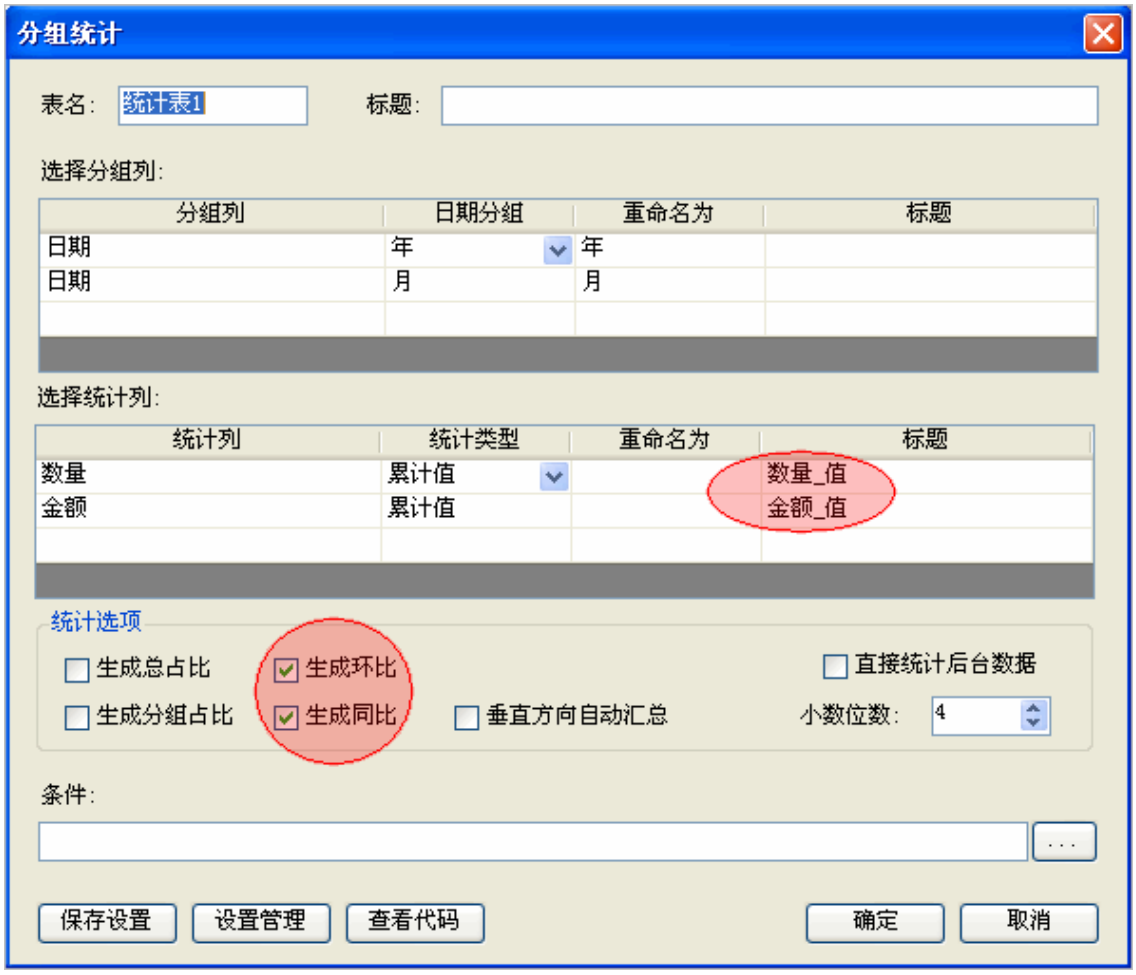
	年	月	数量	同比
1	1999	1	7552	
2	1999	2	9136	
3	1999	3	9134	
4	1999	4	11078	
5	1999	5	11331	
6	1999	6	5253	
7	1999	7	7310	
8	1999	8	9194	
9	1999	9	7847	
10	1999	10	11260	
11	1999	11	10525	
12	1999	12	6019	
13	2000	1	7305	-3.27%
14	2000	2	9577	4.83%
15	2000	3	9173	0.43%
16	2000	4	11048	-0.27%
17	2000	5	9660	-14.75%
18	2000	6	6688	27.32%
19	2000	7	7017	-4.01%
20	2000	8	8511	-7.43%
21	2000	9	9138	16.45%
22	2000	10	10609	-5.78%
23	2000	11	11125	5.70%
24	2000	12	5527	-8.17%

显然，必须有跨年的数据，同比统计才有意义，因为同比是相对上年同时段而言的。

当然可以同时生成环比和同比：

	年	月	数量	环比	同比
1	1999	1	7552		
2	1999	2	9136	20.97%	
3	1999	3	9134	-0.02%	
4	1999	4	11078	21.28%	
5	1999	5	11331	2.28%	
6	1999	6	5253	-53.64%	
7	1999	7	7310	39.16%	
8	1999	8	9194	25.77%	
9	1999	9	7847	-14.65%	
10	1999	10	11260	43.49%	
11	1999	11	10525	-6.53%	
12	1999	12	6019	-42.81%	
13	2000	1	7305	21.37%	-3.27%
14	2000	2	9577	31.10%	4.83%
15	2000	3	9173	-4.22%	0.43%
16	2000	4	11048	20.44%	-0.27%
17	2000	5	9660	-12.56%	-14.75%
18	2000	6	6688	-30.77%	27.32%
19	2000	7	7017	4.92%	-4.01%
20	2000	8	8511	21.29%	-7.43%
21	2000	9	9138	7.37%	16.45%
22	2000	10	10609	16.10%	-5.78%
23	2000	11	11125	4.86%	5.70%
24	2000	12	5527	-50.32%	-8.17%

在对多列进行统计的时候，通过适当地设置标题，可以让生成的统计表更为美观，例如:

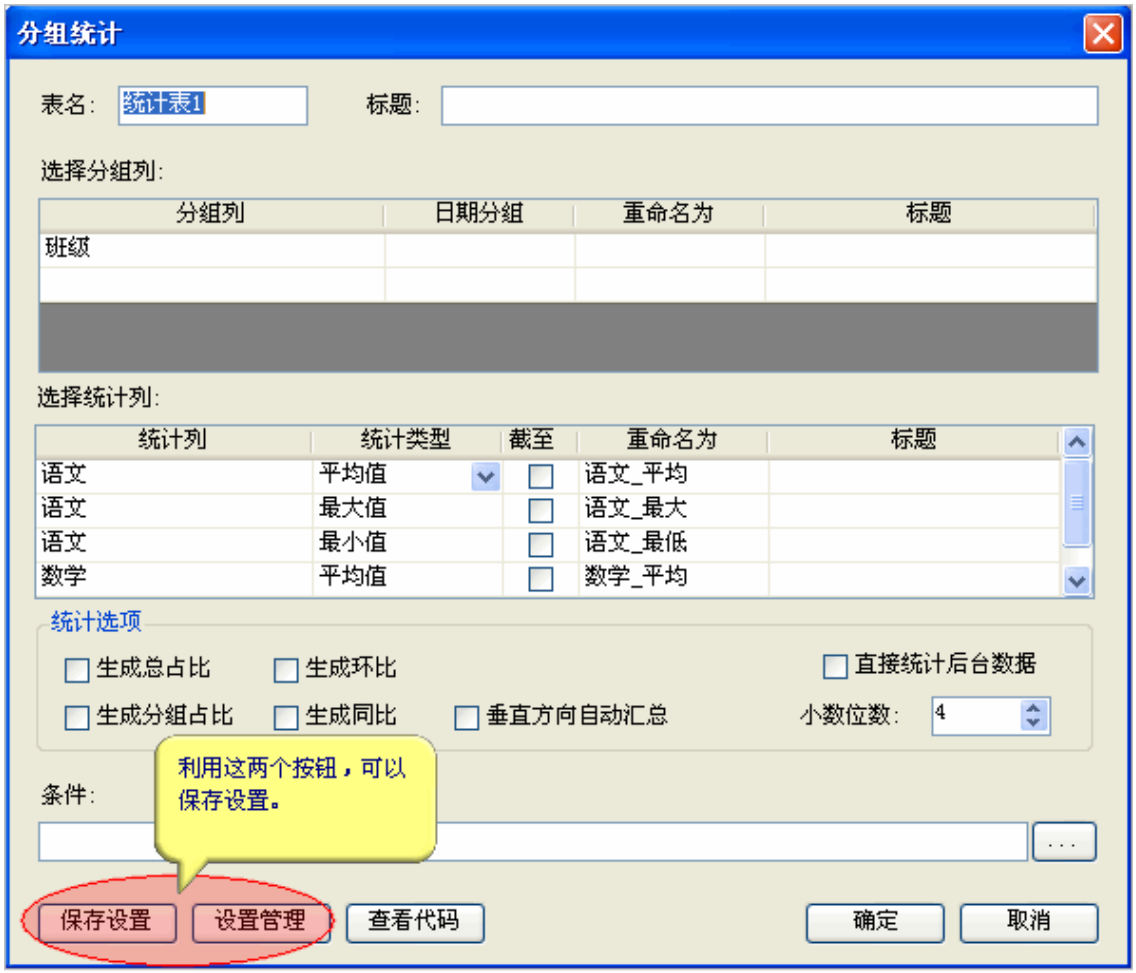


这是生成的统计表：

	年	月	数量			金额		
			值	同比	环比	值	同比	环比
1	1999	1	7552			105629.17		
2	1999	2	9136		20.97%	151020.067		42.97%
3	1999	3	9134		-0.02%	143293.87		-5.12%
4	1999	4	11078		21.28%	171847.637		19.93%
5	1999	5	11331		2.28%	183151.207		6.58%
6	1999	6	5253		-53.64%	86066.03		-53.01%
7	1999	7	7310		39.16%	100822.49		17.15%
8	1999	8	9194		25.77%	155207.07		53.94%
9	1999	9	7847		-14.65%	124687.985		-19.66%
10	1999	10	11260		43.49%	175934.742		41.10%
11	1999	11	10525		-6.53%	171353.722		-2.60%
12	1999	12	6019		-42.81%	102384.326		-40.25%
13	2000	1	7305	-3.27%	21.37%	99036.85	-6.24%	-3.27%
14	2000	2	9577	4.83%	31.10%	159381.51	5.54%	60.93%
15	2000	3	9173	0.43%	-4.22%	143529.2	0.16%	-9.95%
16	2000	4	11048	-0.27%	20.44%	168608.34	-1.88%	17.47%
17	2000	5	9660	-14.75%	-12.56%	155910.525	-14.87%	-7.53%
18	2000	6	6688	27.32%	-30.77%	113760.398	32.18%	-27.03%
19	2000	7	7017	-4.01%	4.92%	94971.32	-5.80%	-16.52%
20	2000	8	8511	-7.43%	21.29%	143204.222	-7.73%	50.79%

3.11.5.10 设置管理

你可以将分组统计的设置保存起来，便于以后快速调用，这样一些复杂的统计只需设置并保存一次，以后就可以直接调用，无需反复设置。

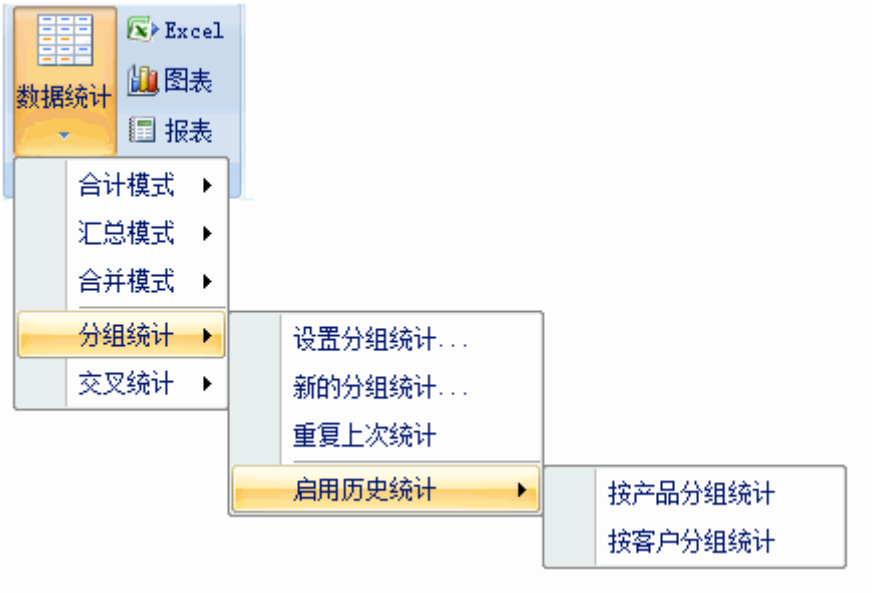


- 保存设置
  在分组统计设置窗口，先设置好分组统计，然后单击“保存设置”按钮，输入设置名称，单击“确定”按钮即可。

- 修改设置
  在分组统计设置窗口，先设置好分组统计，然后单击“保存设置”按钮，选择之前保存的设置名称，单击“确定”按钮即可覆盖原来的设置。

- 设置管理
  在分组统计设置窗口，单击“设置管理”按钮，可以删除、重命名已经保存的设置。

- 调用设置
  在菜单的“日常工作”区，单击“数据统计”按钮，选择“分组统计”，然后指向“启用历史统计”，会列出所有保存好的设置名称，单击其中一个，即可自动根据该设置生成分组统计表。

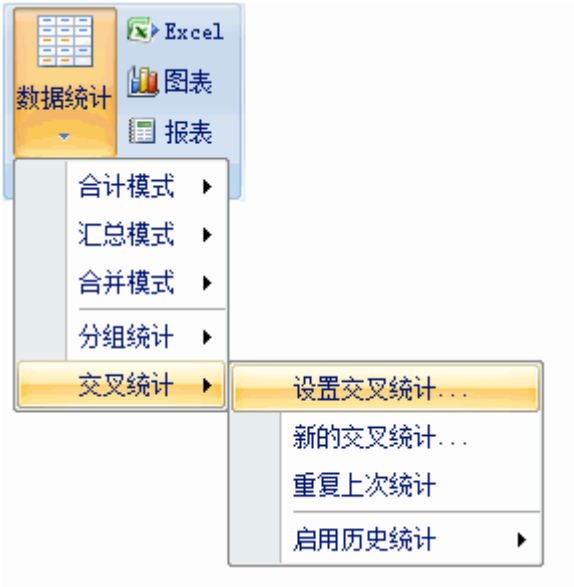


- 重复上次统计
  在上图中，单击“重复上次统计”命令，即可根据最近一次的分组统计设置，重新生成统计表。

3.11.6 交叉统计

3.11.6.1 设置交叉统计

在菜单的“日常工作”区，单击“数据统计”按钮，指向“交叉统计”，会出现交叉统计的相关菜单命令：



提示：
1、执行“设置交叉统计”，会打开交叉统计设置窗口，并加载最近一次统计的设置。
2、执行“新的交叉统计”，会打开交叉统计设置窗口，不加载最近一次统计的设置。
3、执行“重复上次统计”，可以根据最近一次的设置，重新生成交叉统计表。

设置交叉统计

首先我们看看如何进行一个简单的交叉统计，任务是用交叉表的形式统计每个客户订购不同产品的数量。

1、在交叉统计设置窗口，按下图所示设置：



2、设置完成后，单击“确定”按钮，得到下图所示的统计结果，可以看到第一列的内容为产品名称，而其余各列的列标题为客户名称：

	产品	CS01	CS02	CS03	CS04	CS05
1	PD01	260	4120	1980	3390	1540
2	PD02	2250	4300	6300	3110	2240
3	PD03	2180	2250	970	700	900
4	PD04	230	1830	1200	1410	810
5	PD05	600	2550	2420	3350	1480

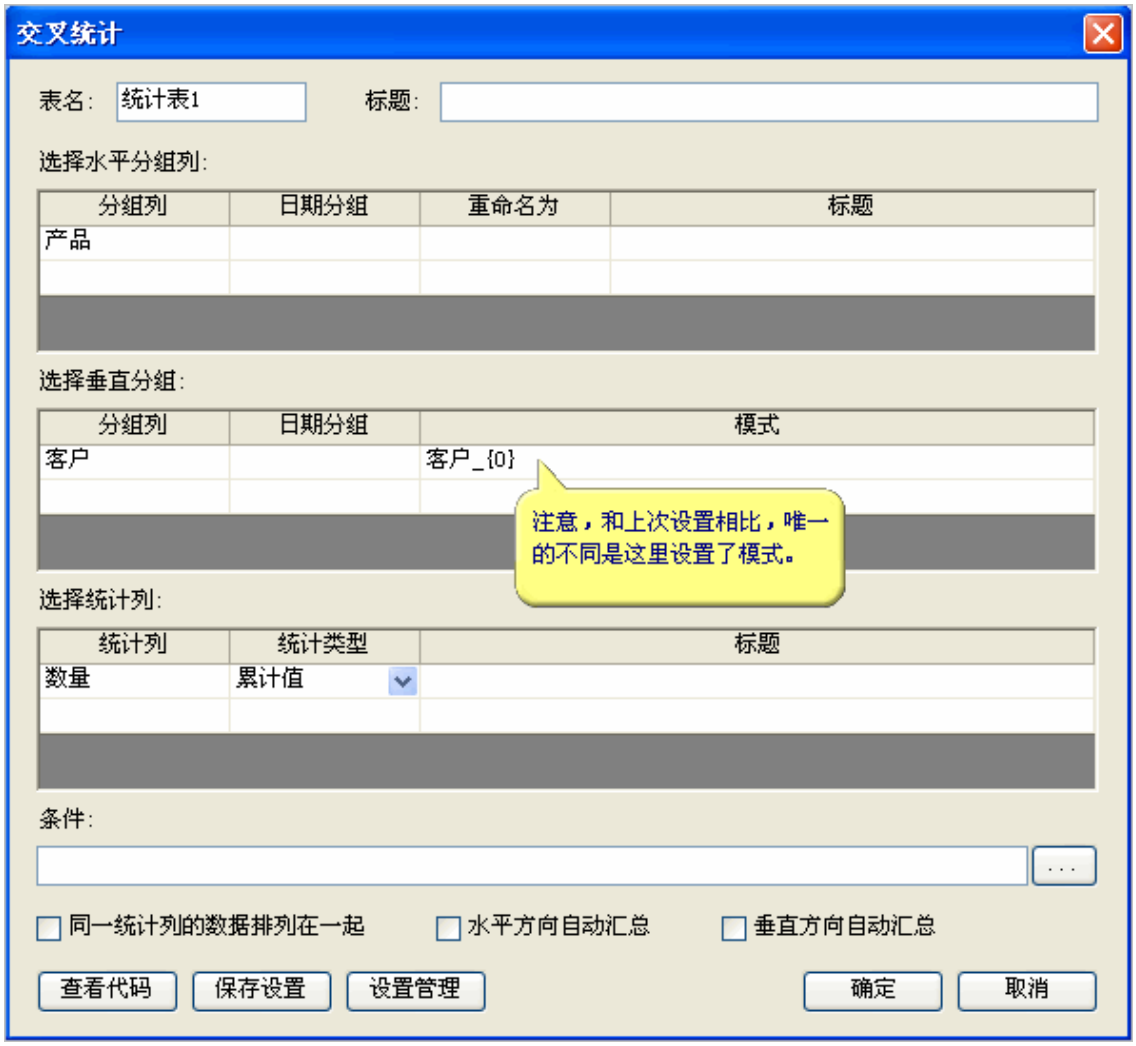
交叉统计生成的统计表是临时的，重新打开项目后，这些表将不复存在。

3.11.6.2 设置垂直分组列的标题

这次的任务同样是用交叉的表的形式统计每个客户订购不同产品的数量，不过我们希望做得美观一些。

在交叉统计设置窗口，按下图所示设置，其实和上一次的设置基本一样，只是我们设置了垂直分

组列的标题模式：

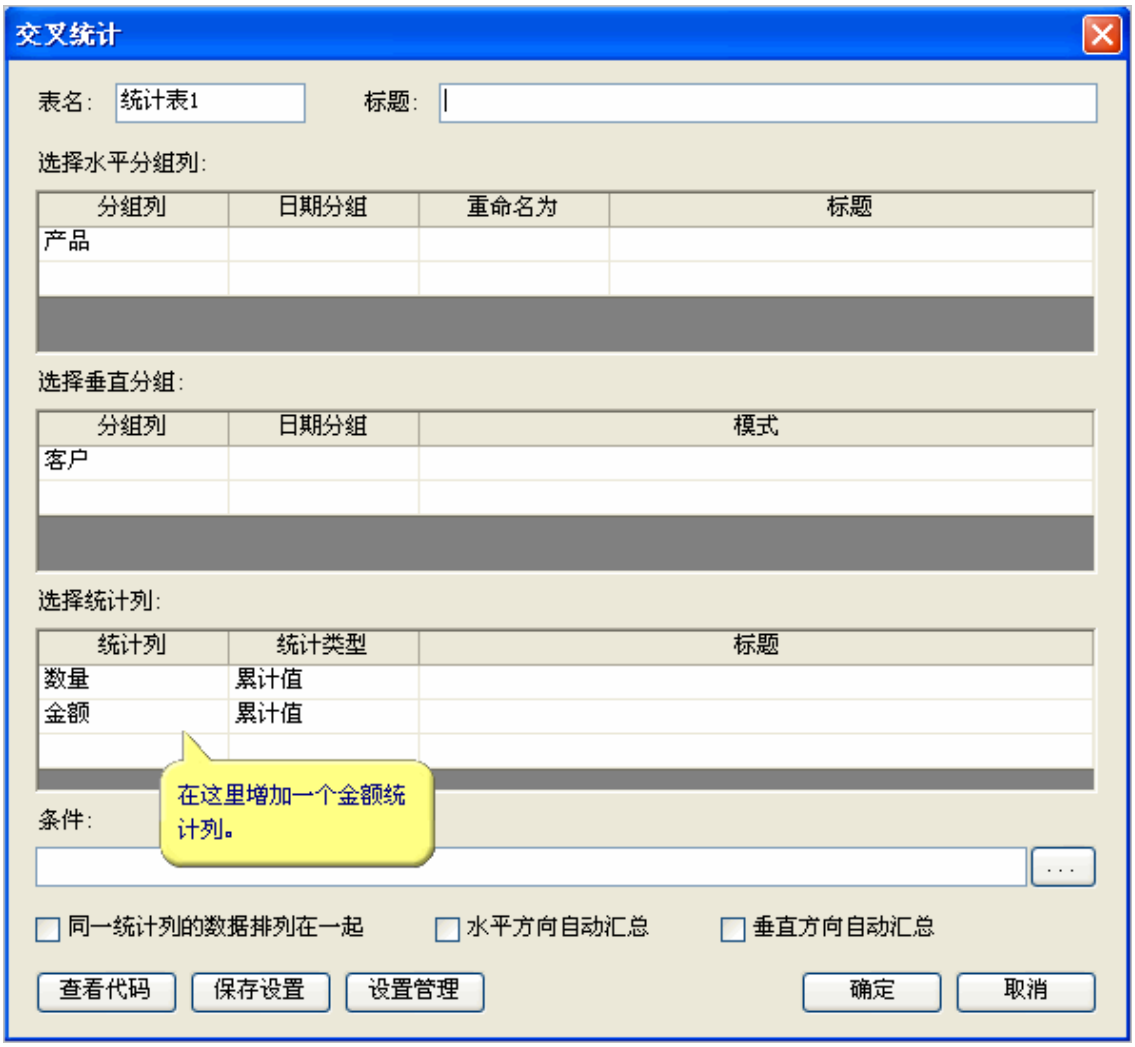


在生成统计表的时候，标题模式中的{0}会被实际的客户名称代替，我们巧妙地利用标题模式生成了一个有多层表头的统计表：



3.11.6.3 统计多列数据

这次的任务是统计出每个客户订购不同产品的数量和金额。
设置很简单，只是在统计列设置中加多一个金额列而已：



这是统计结果：

	产品	CS01		CS02		CS03		CS04		CS05	
		数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	PD01	260	3777.12	4120	60292.8	1980	28638	3390	52155	1540	26892
2	PD02	2250	34814.6	4300	64634.2	6300	106352.5	3110	52706	2240	38015.2
3	PD03	2180	17720	2250	18940	970	9680	700	6550	900	8600
4	PD04	230	3951.2	1830	32414.8	1200	23320	1410	27280	810	15895
5	PD05	600	12338.7	2550	45418.05	2420	43467.75	3350	66256	1480	25542.65

3.11.6.4 统计数据的排列方式

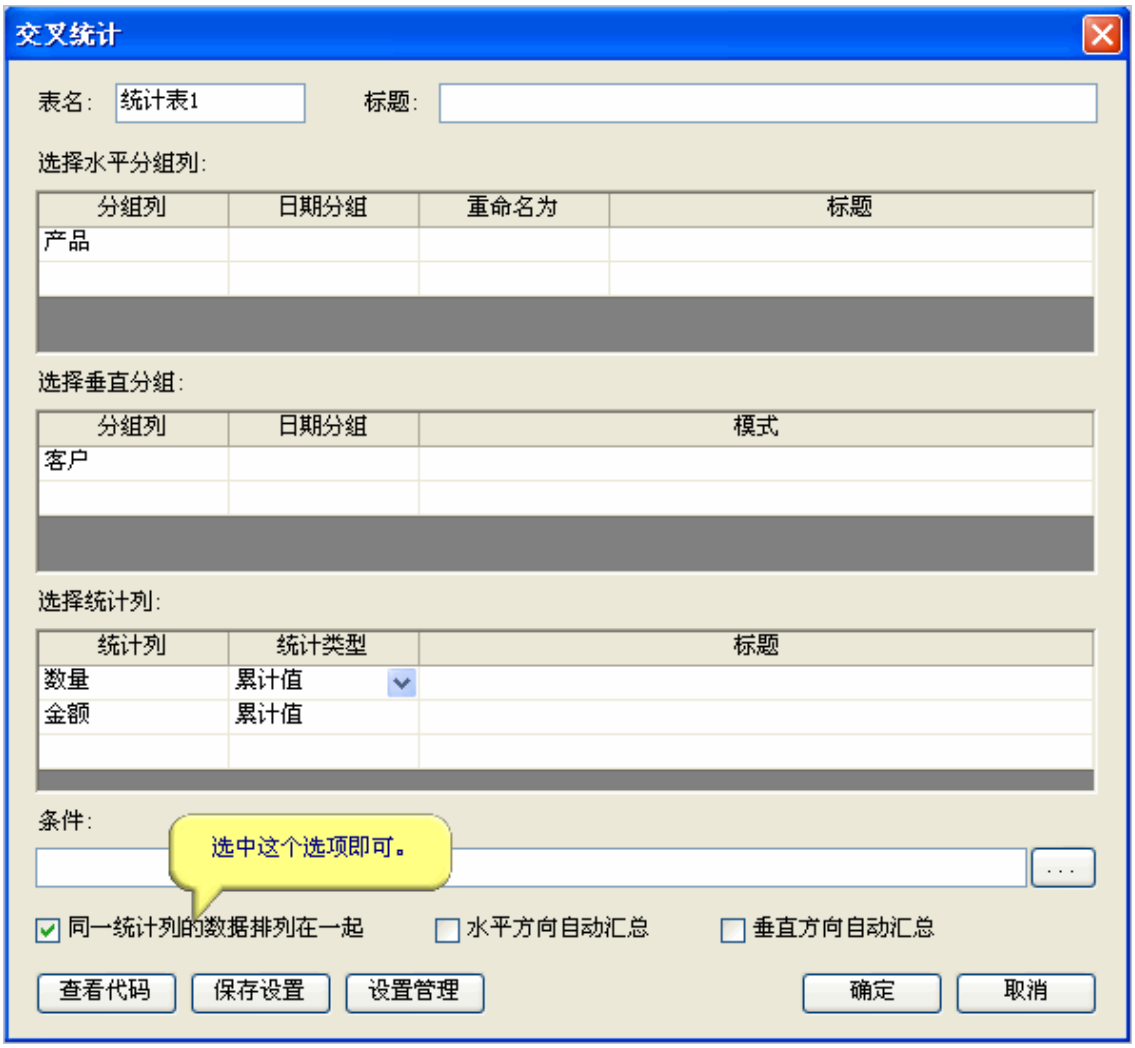
在上一节生成的统计表中，同一客户的销售数量和销售金额排列在一起：

	产品	CS01		CS02		CS03		CS04		CS05	
		数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	PD01	260	3777.12	4120	60292.8	1980	28638	3390	52155	1540	26892
2	PD02	2250	34814.6	4300	64634.2	6300	106352.5	3110	52706	2240	38015.2
3	PD03	2180	17720	2250	18940	970	9680	700	6550	900	8600
4	PD04	230	3951.2	1830	32414.8	1200	23320	1410	27280	810	15895
5	PD05	600	12338.7	2550	45418.05	2420	43467.75	3350	66256	1480	25542.65

有的时候，你也许会希望不同客户的数量和金额分别排列在一起，以便于比较：

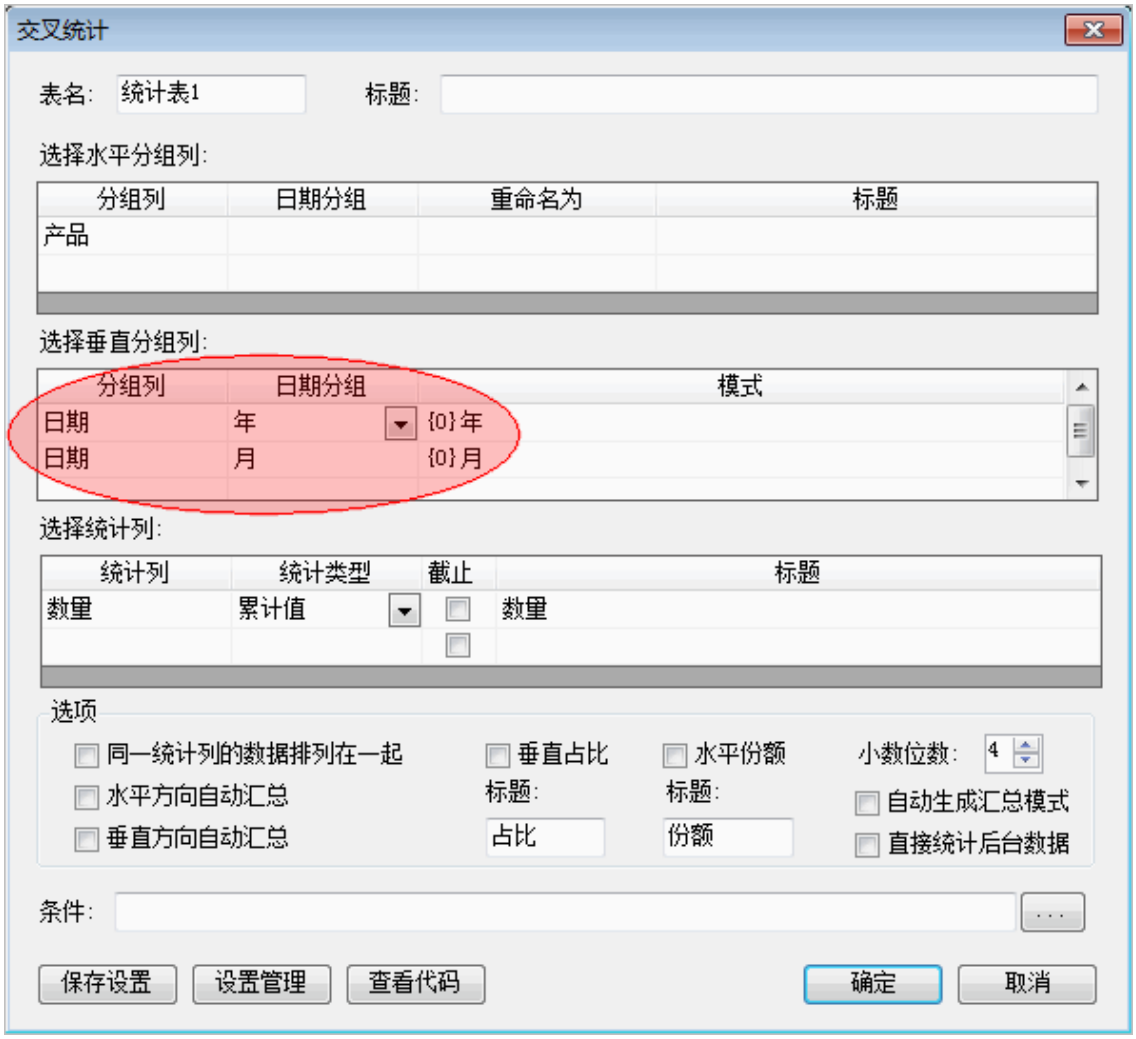
	产品	数量					金额				
		CS01	CS02	CS03	CS04	CS05	CS01	CS02	CS03	CS04	CS05
1	PD01	260	4120	1980	3390	1540	3777.12	60292.8	28638	52155	26892
2	PD02	2250	4300	6300	3110	2240	34814.6	64634.2	106352.5	52706	38015.2
3	PD03	2180	2250	970	700	900	17720	18940	9680	6550	8600
4	PD04	230	1830	1200	1410	810	3951.2	32414.8	23320	27280	15895
5	PD05	600	2550	2420	3350	1480	12338.7	45418.05	43467.75	66256	25542.65

要实现上图所示的排列效果，只需在设置交叉统计的时候，选中选项“同一统计列的数据排列在一起”：



3.11.6.5 根据日期分组统计

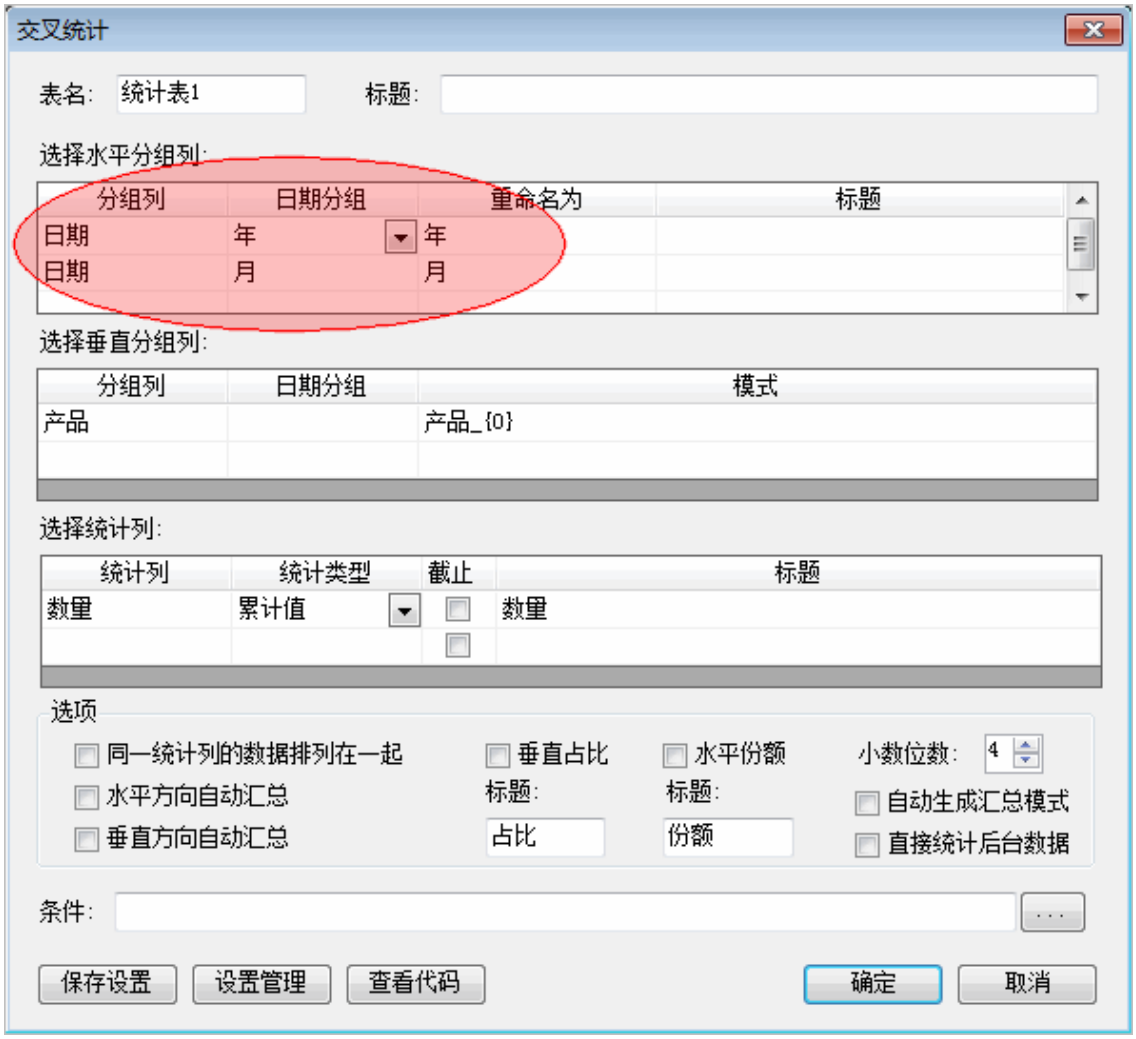
统计不同产品每月销售数量，交叉统计的设置如下：



这是统计结果:

	产品	1999年					
		1月	2月	3月	4月	5月	6月
1	PD01	2120	1600	2430	2000	1870	1270
2	PD02	1300	3170	2040	5350	4140	2200
3	PD03	1930	1290	1100	1400	980	300
4	PD04	970	510	1010	940	1140	910
5	PD05	1240	2420	1840	1220	2400	1280

也可以将年月作为水平分组，例如：

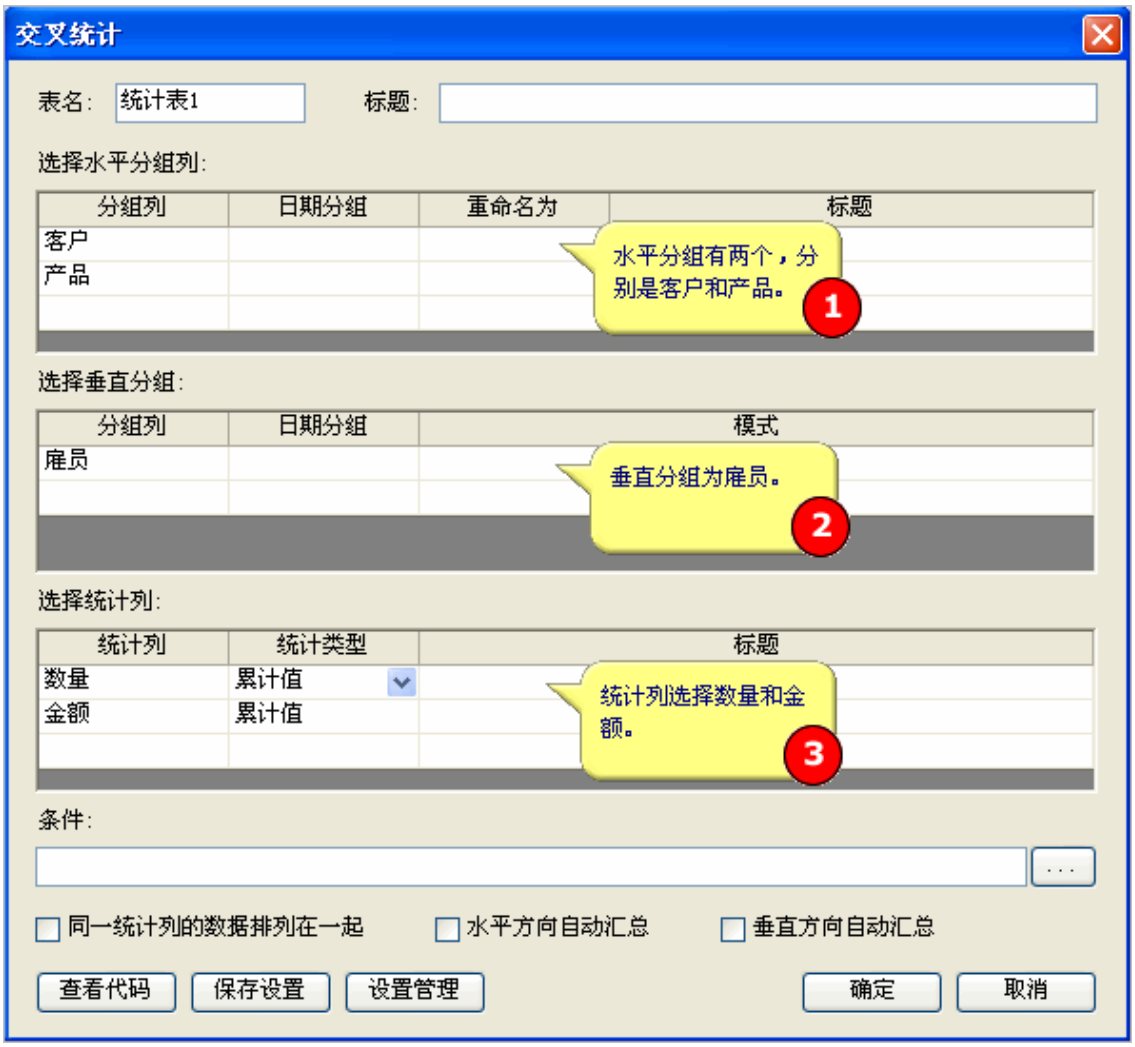

交叉统计
表名:
统计表1
标题:
选择水平分组列:
分组列
日期分组
重命名为
标题
日期
年
年
日期
月
月
选择垂直分组列:
分组列
日期分组
模式
产品
产品_{0}
选择统计列:
统计列
统计类型
截止
标题
数量
累计值
数量
选项
同一统计列的数据排列在一起
水平方向自动汇总
垂直方向自动汇总
垂直占比
标题:
占比
水平份额
标题:
份额
小数位数:
4
自动生成汇总模式
直接统计后台数据
条件:
保存设置
设置管理
查看代码
确定
取消

得到的统计结果：

	年	月	产品				
			PD01	PD02	PD03	PD04	PD05
1	1999	1	5569	12720	8638	7325	7174
2	1999	2	9685	10004	6029	8633	10383
3	1999	3	9533	10753	8109	5759	8082
4	1999	4	9026	5337	7644	13885	8079
5	1999	5	4751	7840	7768	8903	10107
6	1999	6	7555	6951	6389	9841	8465
7	1999	7	5204	6807	6088	11864	9332
8	1999	8	8103	9755	8579	13565	11102
9	1999	9	11014	8657	10107	12863	8595
10	1999	10	6001	6504	7644	7302	5896
11	1999	11	7346	10433	8843	6165	9031
12	1999	12	6124	5120	4929	7714	13972
13	2000	1	8548	6409	11585	9142	8321
14	2000	2	5449	7677	3445	6328	6503
15	2000	3	11845	4223	9943	7001	6789
16	2000	4	8228	11208	7213	6194	8701
17	2000	5	10069	9154	8709	6439	7728
18	2000	6	8519	4516	9555	5244	6466
19	2000	7	6861	12682	12709	3821	7580
20	2000	8	6830	8774	9137	7926	13786
21	2000	9	9805	8897	5660	6915	6433
22	2000	10	13623	7286	8444	9471	8617
23	2000	11	6448	10598	7529	6231	7920
24	2000	12	7807	8205	9322	6658	7766

3.11.6.6 多个水平分组

现在希望得到每个业务员和哪些客户有联系，都销售了哪些产品，销售数量和金额是多少？听起来很复杂，做起来其实很容易，按下图所示设置交叉统计即可：

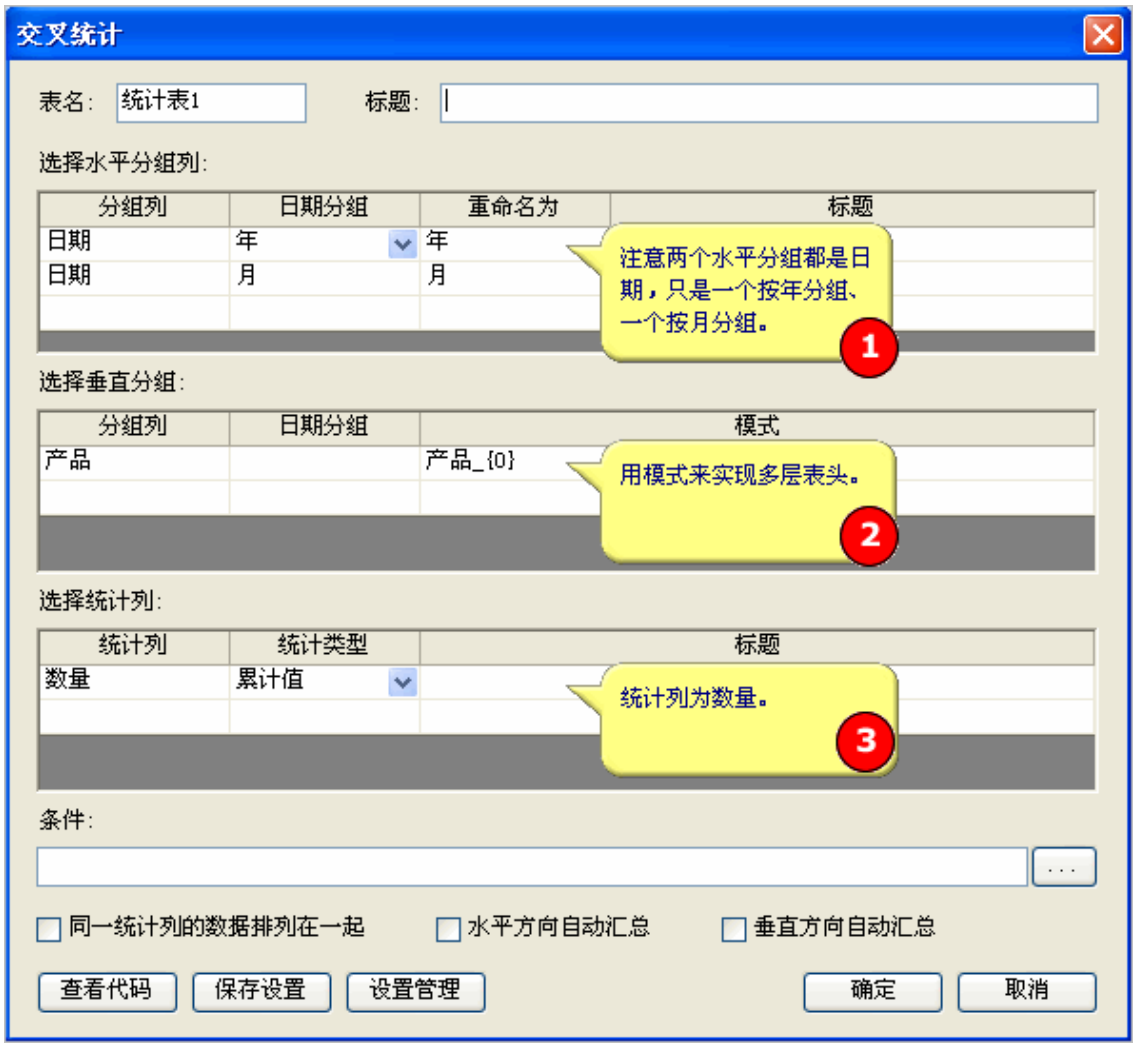

交叉统计
表名:
统计表1
标题:
选择水平分组列:
分组列
日期分组
重命名为
标题
客户
产品
水平分组有两个，分别是客户和产品。
1
选择垂直分组:
分组列
日期分组
模式
雇员
垂直分组为雇员。
2
选择统计列:
统计列
统计类型
标题
数量
累计值
金额
累计值
统计列选择数量和金额。
3
条件:
同一统计列的数据排列在一起
水平方向自动汇总
垂直方向自动汇总
查看代码
保存设置
设置管理
确定
取消

这是统计结果：

	客户	产品	EP01		EP02		EP03		EP04		EP05	
			数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	CS01	PD01			240	3417.12			20	360		
2	CS01	PD02	700	11760	500	7531.6	400	5168	600	9595	50	760
3	CS01	PD03	240	2400	440	3320	1200	9600	300	2400		
4	CS01	PD04	50	1100					180	2851.2		
5	CS01	PD05	40	854	200	3798.7	360	7686				
6	CS02	PD01	2060	30988.8	160	2664	1650	23364			250	3276
7	CS02	PD02	240	3648	700	11020	500	6853.3	1700	24130	1160	18982.9
8	CS02	PD03			200	1440	650	6300	700	7000	700	4200
9	CS02	PD04			800	13200	710	13970	320	5244.8		
10	CS02	PD05	200	4270	360	5304	1690	31040.3			300	4803.75
11	CS03	PD01	1360	19314			200	3420	320	4104	100	1800
12	CS03	PD02	1910	31350	1810	30590	600	10117.5	1480	26695	500	7600
13	CS03	PD03			300	3000	20	180	490	4900	160	1600
14	CS03	PD04	200	3520	170	3520	550	10120	280	6160		
15	CS03	PD05	100	2135			1010	19724.75	1110	17765	200	3843
16	CS04	PD01	60	1080	1940	30780	500	7380	650	9495	240	3420
17	CS04	PD02	60	1140	420	6840	830	14136	600	11210	1200	19380
18	CS04	PD03	200	1700	100	850	300	3000	100	1000		
19	CS04	PD04			500	9900	120	2376	550	10780	240	4224
20	CS04	PD05	250	4250	820	14024	1360	28340	710	15158.5	210	4483.5
21	CS05	PD01			500	9000	590	9792			450	8100
22	CS05	PD02			650	10925	700	13300	890	13790.2		
23	CS05	PD03			300	3000	600	5600				
24	CS05	PD04							560	11770	250	4125
25	CS05	PD05			1060	18305			420	7237.65		

3.11.6.7 日期列作为水平分组

按年和月统计每个产品的销售数量。
交叉统计的设置如下：



这是统计结果：

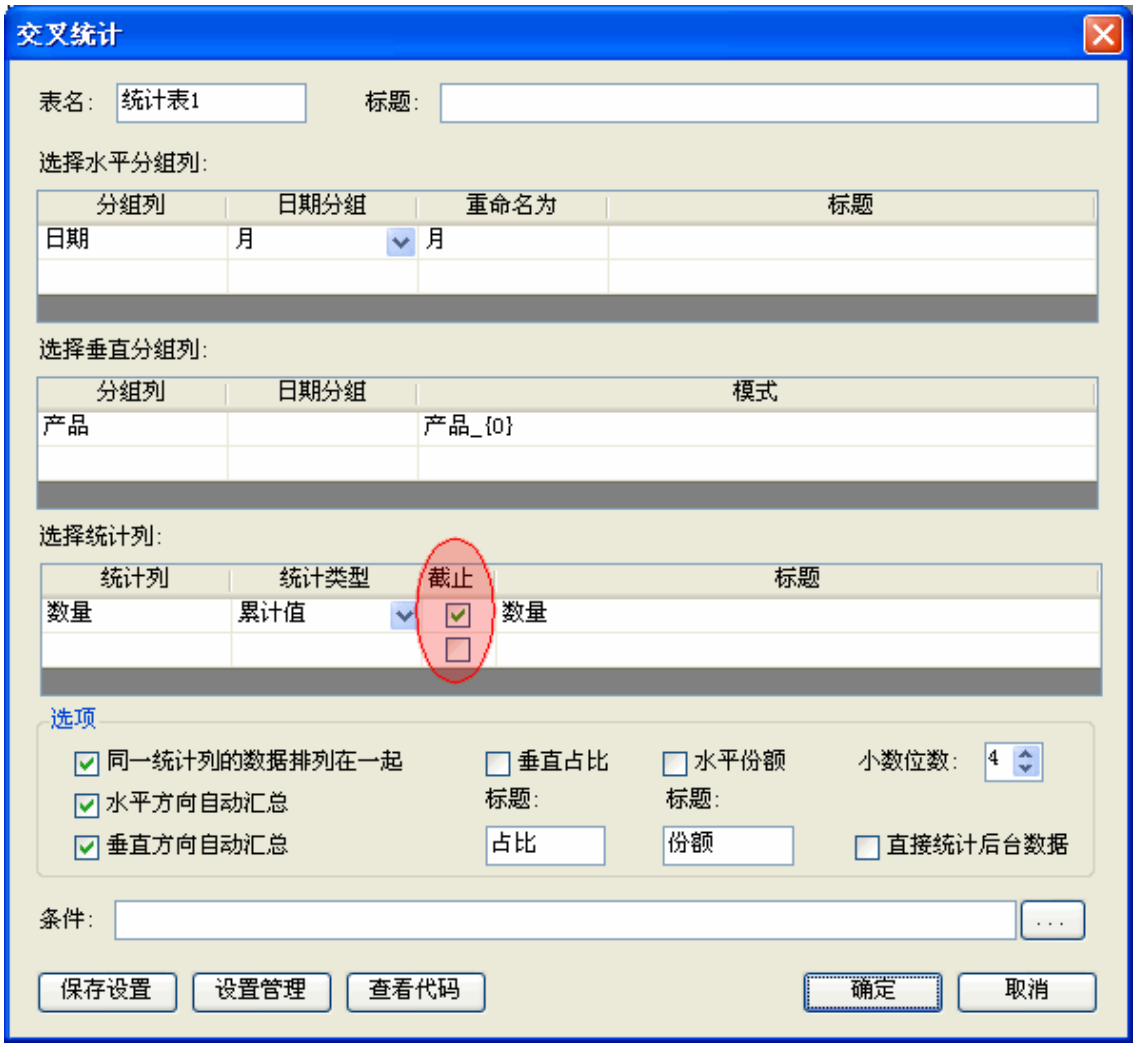
	年	月	产品				
			PD01	PD02	PD03	PD04	PD05
1	1999	1	2120	1300	1930	970	1240
2	1999	2	1600	3170	1290	510	2420
3	1999	3	2430	2040	1100	1010	1840
4	1999	4	2000	5350	1400	940	1220
5	1999	5	1870	4140	980	1140	2400
6	1999	6	1270	2200	300	910	1280

3.11.6.8 截止统计

交叉统计默认是各个分组独自统计，可是有的时候，我们需要的并非是每个分组的统计结果，而是截止到每个分组时的统计结果。

例如在对订单表进行统计的时候，我们希望得到截止到每个月的时候，各产品的累计订购数量。

要得到这样的统计结果非常简单，只需在设置统计列的时候，选择选项“截止”即可：



这是统计结果：

	月	产品				
		PD01	PD02	PD03	PD04	PD05
1	1	14117	19129	20223	16467	15495
2	2	29251	36810	29697	31428	32381
3	3	50629	51786	47749	44188	47252
4	4	67883	68331	62606	64267	64032
5	5	82703	85325	79083	79609	81867
6	6	98777	96792	95027	94694	96798
7	7	110842	116281	113824	110379	113710
8	8	125775	134810	131540	131870	138598
9	9	146594	152364	147307	151648	153626
10	10	166218	166154	163395	168421	168139
11	11	180012	187185	179767	180817	185090
12	12	193943	200510	194018	195189	206828

注意这里的结果并非该月的产品销售数量，而是截至到该月为止的累计产品销售数量。

如果你希望同时得到每月销量和截止到该月的累计销量，可以按下图所示设置：



这是得到的统计结果，小计列表示产品在该月的销量，截止列表示截止到该月为止此产品的累计销量：

	月	PD01		PD02		PD03		PD04		PD05	
		小计	截止	小计	截止	小计	截止	小计	截止	小计	截止
1	1	14117	14117	19129	19129	20223	20223	16467	16467	15495	15495
2	2	15134	29251	17681	36810	9474	29697	14961	31428	16886	32381
3	3	21378	50629	14976	51786	18052	47749	12760	44188	14871	47252
4	4	17254	67883	16545	68331	14857	62606	20079	64267	16780	64032
5	5	14820	82703	16994	85325	16477	79083	15342	79609	17835	81867
6	6	16074	98777	11467	96792	15944	95027	15085	94694	14931	96798
7	7	12065	110842	19489	116281	18797	113824	15685	110379	16912	113710
8	8	14933	125775	18529	134810	17716	131540	21491	131870	24888	138598
9	9	20819	146594	17554	152364	15767	147307	19778	151648	15028	153626
10	10	19624	166218	13790	166154	16088	163395	16773	168421	14513	168139
11	11	13794	180012	21031	187185	16372	179767	12396	180817	16951	185090
12	12	13931	193943	13325	200510	14251	194018	14372	195189	21738	206828

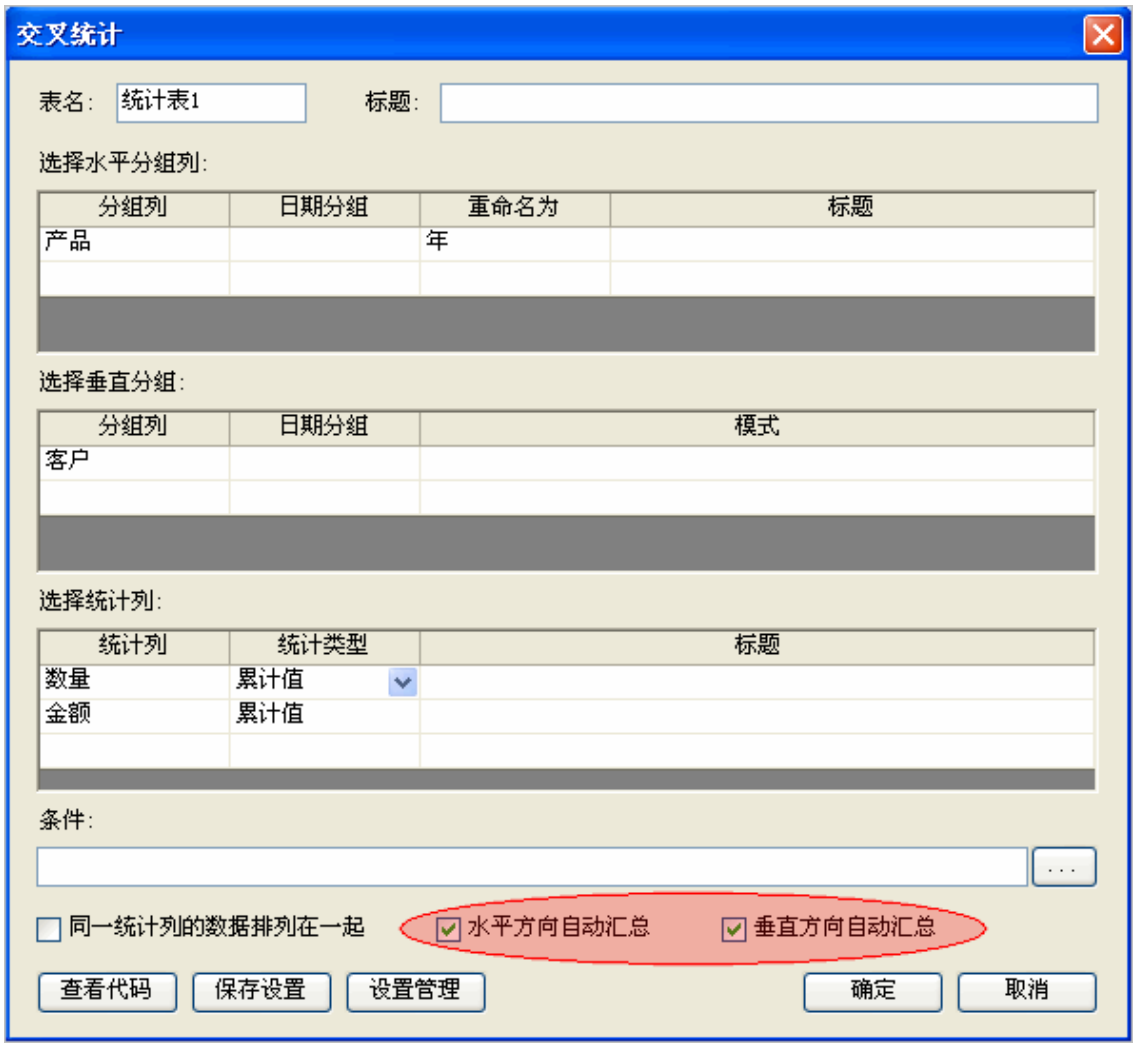
3.11.6.9 自动汇总

在交叉统计设置窗口，有两个选项：

- 水平方向自动汇总
  选中该选项，统计表将为每一个统计列增加一列，用于显示每一行的累计值。

- 垂直方向自动汇总
  选中该选项，将在统计表后面增加一行，显示所有数值列的累计值。

例如下图的交叉统计，用于统计不同客户订购每种产品的数量和金额，而且选中了上述两个选项：



得到的统计结果为：

	年	CS01		CS02		CS03		CS04		CS05		合计	
		数量	金额	数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	PD01	260	3777.12	4120	60292.8	1980	28638	3390	52155	1540	26892	11290	71754.92
2	PD02	2250	34814.6	4300	64634.2	6300	106352.5	3110	52706	2240	38015.2	18200	296522.5
3	PD03	2180	17720	2250	18940	970	9680	700	6550	900	8600	7000	61490
4	PD04	230	3951.2	1830	32414.8	1200	23320	1410	27280	810	15895	5480	102861
5	PD05	600	12338.7	2550	45418.05	2420	43467.75	3350	66256	1480	25542.65	10400	93023.15
6	合计	5520	72601.62	15050	21699.85	12870	11458.25	11960	204947	6970	14944.85	52370	25851.57

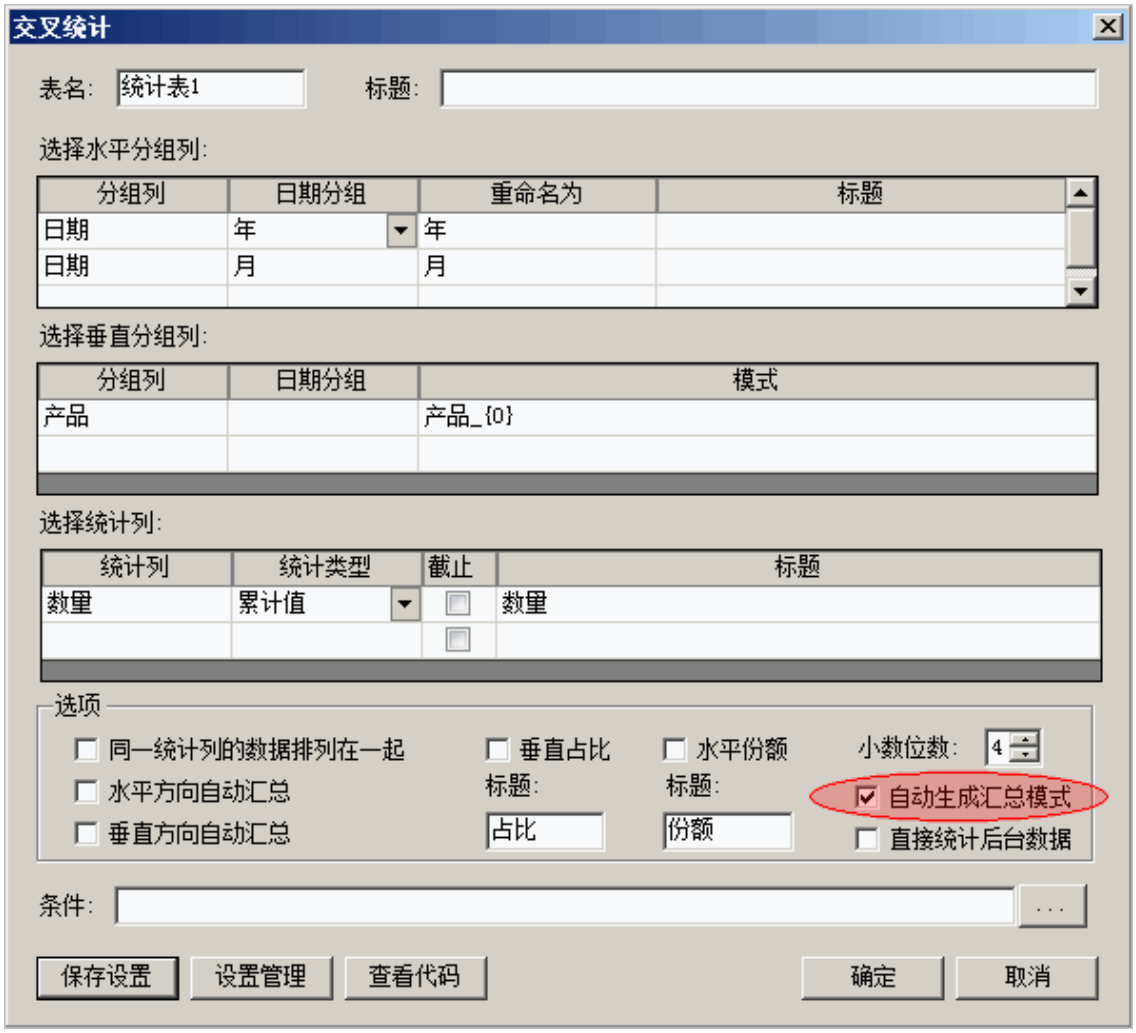
自动加的垂直汇总行。

自动增加了两列，分别用于水平汇总数量和金额。

3.11.6.10 汇总模式和交叉统计

在设置交叉统计的时候，如果选择选项“自动生成汇总模式”，会自动对生成的统计表进行二次统计，生成汇总模式。

例如在按年月分组统计各产品销售数量的时候，选择选项“自动生成汇总模式”:



会得到下图所示的统计结果，不仅统计出了每月各产品数量，还利用汇总模式，统计出了每年数量和总的数量：

	年	月	产品				
			PD01	PD02	PD03	PD04	PD05
1	1999	1	5569	12720	8638	7325	7174
2	1999	2	9685	10004	6029	8633	10383
3	1999	3	9533	10753	8109	5759	8082
4	1999	4	9026	5337	7644	13885	8079
5	1999	5	4751	7840	7768	8903	10107
6	1999	6	7555	6951	6389	9841	8465
7	1999	7	5204	6807	6088	11864	9332
8	1999	8	8103	9755	8579	13565	11102
9	1999	9	11014	8657	10107	12863	8595
10	1999	10	6001	6504	7644	7302	5896
11	1999	11	7346	10433	8843	6165	9031
12	1999	12	6124	5120	4929	7714	13972
	小计 1999		89911	100881	90767	113819	110218
13	2000	1	8548	6409	11585	9142	8321
14	2000	2	5449	7677	3445	6328	6503
15	2000	3	11845	4223	9943	7001	6789
16	2000	4	8228	11208	7213	6194	8701
17	2000	5	10069	9154	8709	6439	7728
18	2000	6	8519	4516	9555	5244	6466
19	2000	7	6861	12682	12709	3821	7580
20	2000	8	6830	8774	9137	7926	13786
21	2000	9	9805	8897	5660	6915	6433
22	2000	10	13623	7286	8444	9471	8617
23	2000	11	6448	10598	7529	6231	7920
24	2000	12	7807	8205	9322	6658	7766
	小计 2000		104032	99629	103251	81370	96610
	总计		193943	200510	194018	195189	206828

3.11.6.11 分组列的选择

对于交叉统计，合理选择分组列是很重要的。

假定你销售 10 个产品，大概有 1000 家左右的固定客户，如果选择客户列作为垂直分组列，那么生成的统计表将有 1000 列，显然这不是一个好的选择。

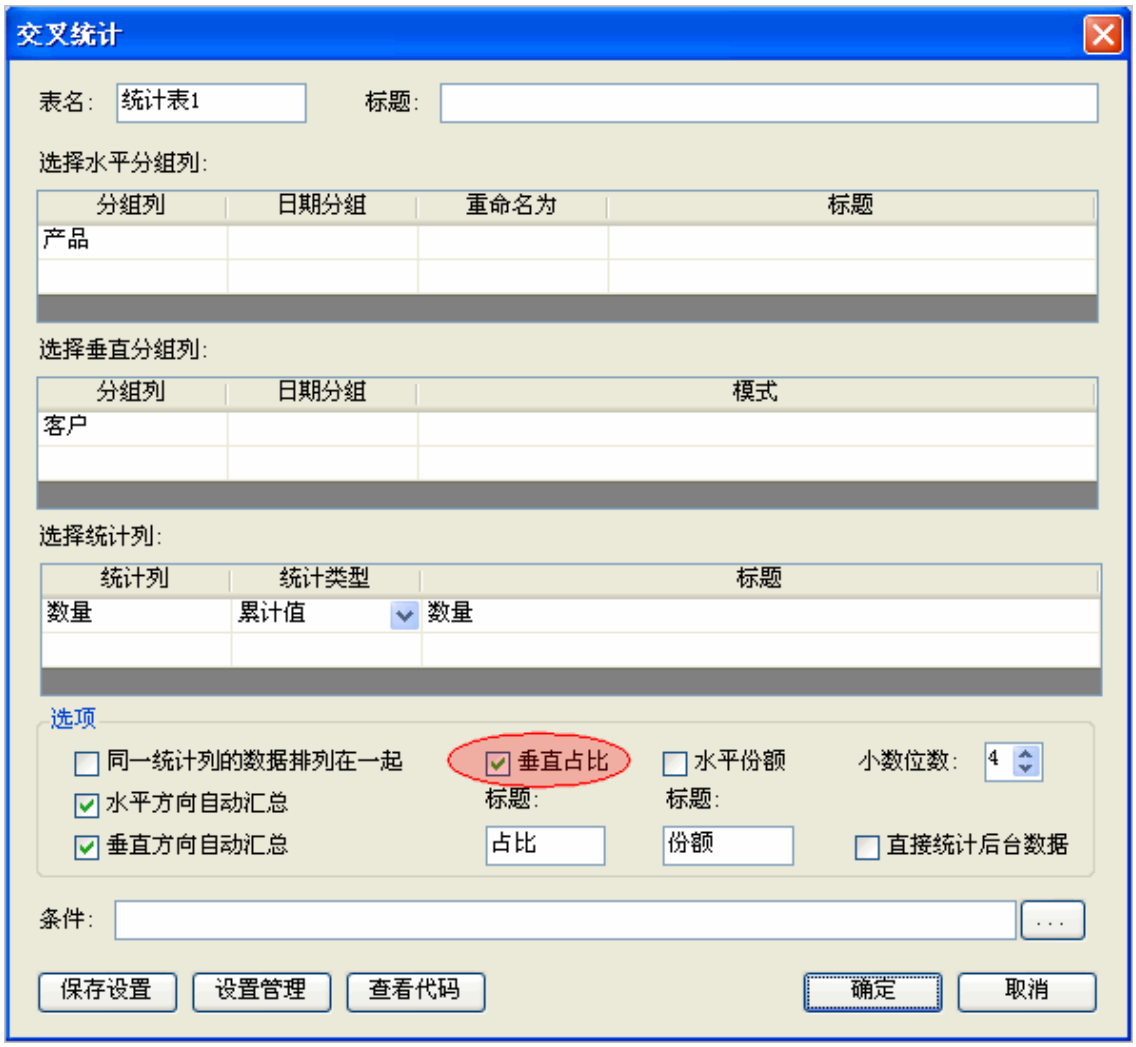
如果选择产品列作为垂直分组，客户列作为水平分组，生成的统计表将是 1000 行×10 列，显然这样的统计表查阅起来会更方便一些。

3.11.6.12 交叉统计之数据分析

和分组统计一样，交叉统计也提供了强大的数据分析功能。

在学习本节内容之前，请打开 CaseStudy 目录下的文件：数据分析.Table

在进行交叉统计的时候，如果选择选项“垂直占比”：



生成的统计表会自动加上一些列，用于显示每个统计结果在垂直方向的累计值中所占的比重:

	产品	CS01	CS01占比	CS02	CS02占比	CS03	CS03占比	CS04
1	PD01	260	4.71%	4120	27.38%	1980	15.44%	33
2	PD02	2250	40.76%	4300	28.57%	6300	49.14%	3
3	PD03	2180	39.49%	2250	14.95%	970	7.57%	7
4	PD04	230	4.17%	1830	12.16%	1200	9.36%	14
5	PD05	600	10.87%	2550	16.94%	2371	18.49%	3
6	合计	5520	100.00%	15050	100.00%	12821	100.00%	119

通过适当地设置标题，可以让生成的统计表更为美观，例如按下图所示设置：



生成的统计表为：

	产品	CS01		CS02		CS03		CS04		CS05		合计
		数量	比例	数量	比例	数量	比例	数量	比例	数量	比例	
1	PD01	260	4.71%	4120	27.38%	1980	15.44%	3390	28.34%	1540	22.09%	11290
2	PD02	2250	40.76%	4300	28.57%	6300	49.14%	3110	26.00%	2240	32.14%	18200
3	PD03	2180	39.49%	2250	14.95%	970	7.57%	700	5.85%	900	12.91%	7000
4	PD04	230	4.17%	1830	12.16%	1200	9.36%	1410	11.79%	810	11.62%	5480
5	PD05	600	10.87%	2550	16.94%	2371	18.49%	3350	28.01%	1480	21.23%	10351
6	合计	5520	100.00	15050	100.00	12821	100.00	11960	100.00	6970	100.00	52321

不仅可以生成垂直方向的占比，也可以生成水平方向的占比(为了区分，水平方向用份额表示)。例如按下图所示设置：



生成的统计表为：

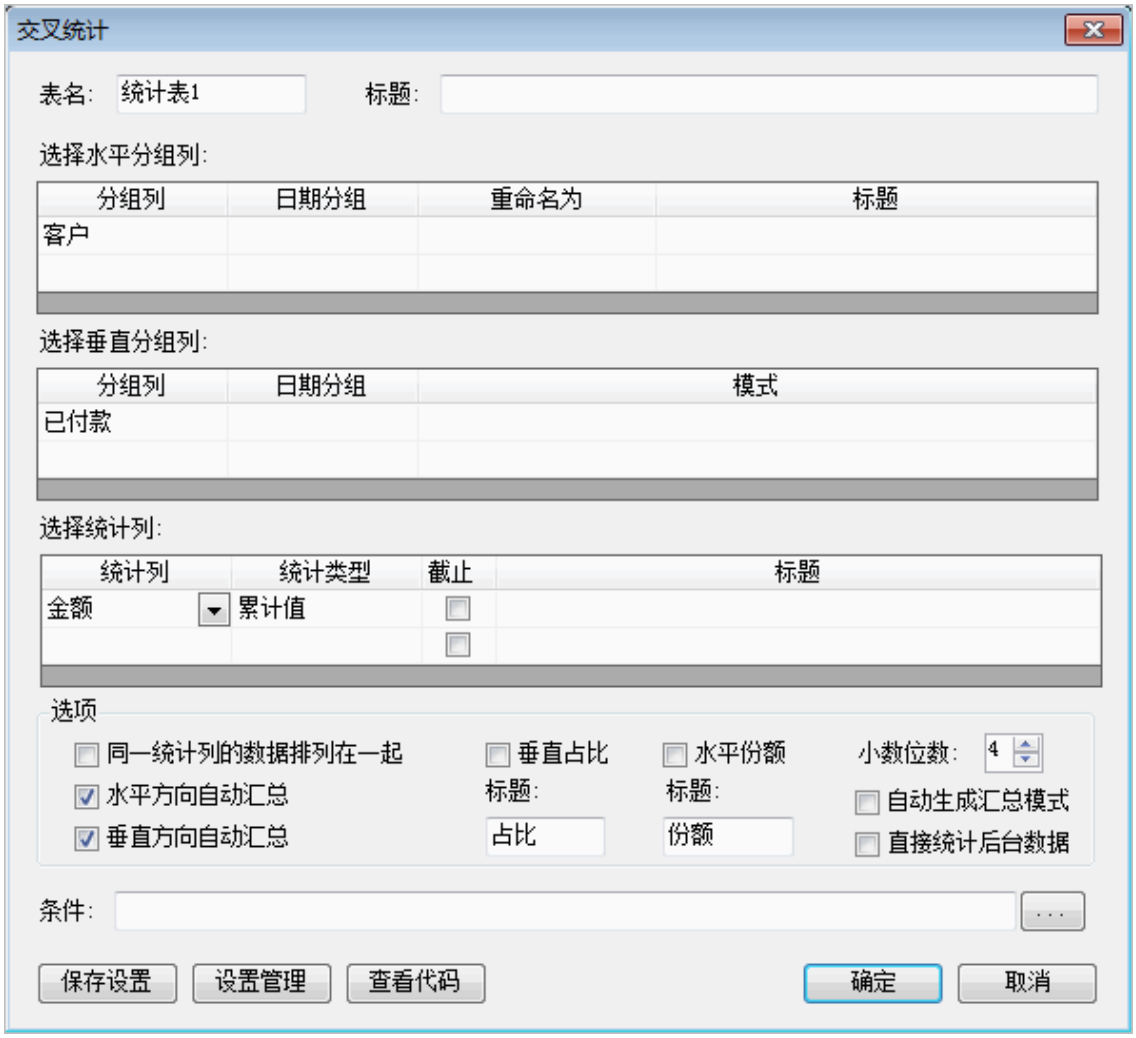
	产品	1999年		2000年		合计
		数量	份额	数量	份额	
1	PD01	21791	48.83%	22837	51.17%	44628
2	PD02	37148	50.09%	37012	49.91%	74160
3	PD03	14448	50.42%	14207	49.58%	28655
4	PD04	12210	52.68%	10967	47.32%	23177
5	PD05	20042	49.57%	20392	50.43%	40434
6	合计	105639		105415		211054

如果你愿意，可以同时生成垂直占比和水平份额：

	产品	1999年			2000年			合计
		数量	份额	占比	数量	份额	占比	
1	PD01	21791	48.83%	20.63%	22837	51.17%	21.66%	44628
2	PD02	37148	50.09%	35.17%	37012	49.91%	35.11%	74160
3	PD03	14448	50.42%	13.68%	14207	49.58%	13.48%	28655
4	PD04	12210	52.68%	11.56%	10967	47.32%	10.40%	23177
5	PD05	20042	49.57%	18.97%	20392	50.43%	19.34%	40434
6	合计	105639		100.00%	105415		100.00%	211054

3.11.6.13 逻辑列与交叉统计

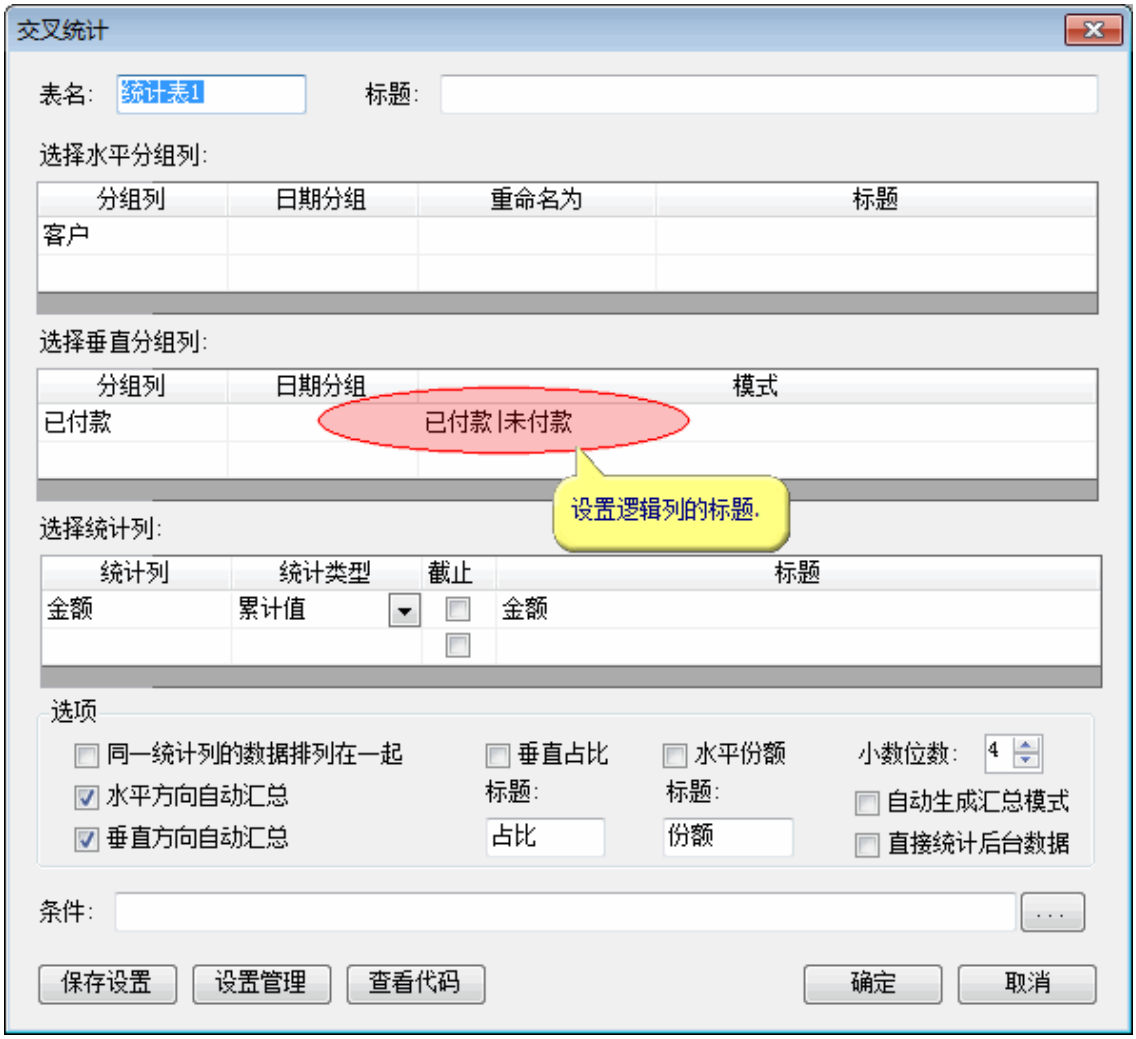
假定订单表有个逻辑列，名为“已付款”，用于标记某个订单是否已经付款。
现在要统计每个客户已经付款的金额、未付款的金额以及总的金额，通常会这样设置交叉统计：



得到的结果是：

	客户	金额		合计
		1	2	
1	CS01	16520	56081.62	72601.62
2	CS02	23307.75	198392.1	221699.85
3	CS03	29845.75	180119.5	209965.25
4	CS04	39966	164981	204947
5	CS05	11620	103324.85	114944.85
6	合计	121259.5	702899.07	824158.57

虽然得到了期望的统计结果，但是出现“1”和“2”这样的列标题，是不够“友善”的。
我们可以修改逻辑列的标题 ，标题用符号“|”格式，第一部分表示值为 True 时的标题，第二部分表示值为 False 时的标题:



现在得到的统计结果是：

	客户	未付款	已付款	合计
1	CS01	16520	56081.62	72601.62
2	CS02	23307.75	198392.1	221699.85
3	CS03	29845.75	180119.5	209965.25
4	CS04	39966	164981	204947
5	CS05	11620	103324.85	114944.85
6	合计	121259.5	702899.07	824158.57

3.11.6.14 设置管理

你可以将交叉统计的设置保存起来，便于以后快速调用，这样一些复杂的统计只需设置并保存一次，以后就可以直接调用，无需反复设置。

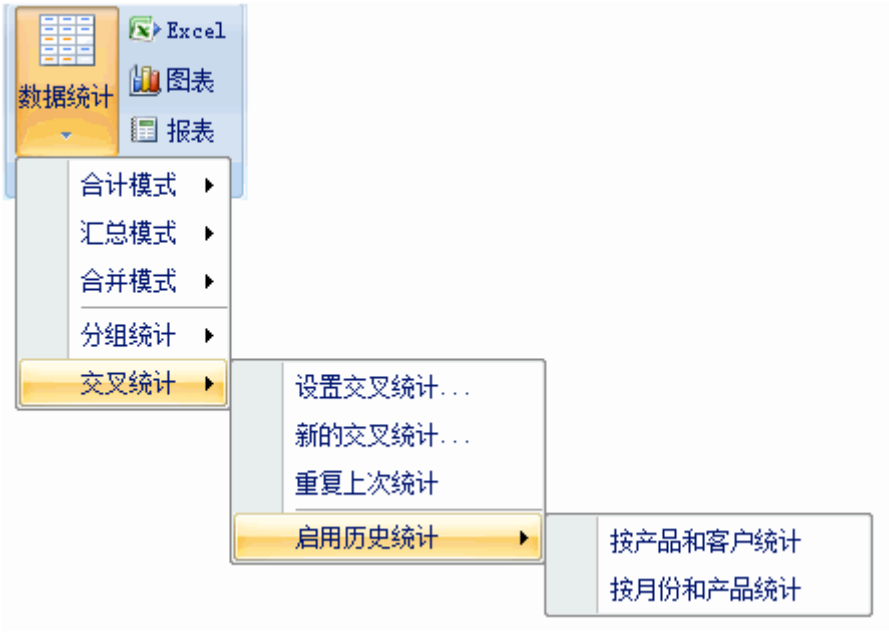


- 保存设置
  在交叉统计设置窗口，先设置好交叉统计，然后单击“保存设置”按钮，输入设置名称，单击“确定”按钮即可。

- 修改设置
  在交叉统计设置窗口，先设置好交叉统计，然后单击“保存设置”按钮，选择之前保存的设置名称，单击“确定”按钮即可覆盖原来的设置。

- 设置管理
  在交叉统计设置窗口，单击“设置管理”按钮，可以删除、重命名已经保存的设置。

- 调用设置
  在菜单的“日常工作”区，单击“数据统计”按钮，选择“交叉统计”，然后指向“启用历史统计”，会列出所有保存好的设置名称，单击其中一个，即可自动根据该设置生成交叉统计表。



- 重复上次统计
  在上图中，单击“重复上次统计”命令，即可根据最近一次的 交叉统计设置，重新生成统计表。

3.11.7 快速统计

如果你的屏幕分辨率达到 1280 或更高，那么日常工作区会出现一个“快速统计”的功能组：



- 切换汇总模式
  根据最近一次设置，重新进入汇总模式。

- 切换合并模式
  根据最近一次设置，重新进入合并模式。

- 重新分组统计
  根据最近一次设置，重新生成分组统计表。

- 重新交叉统计
  根据最近一次设置，重新生成交叉统计表。

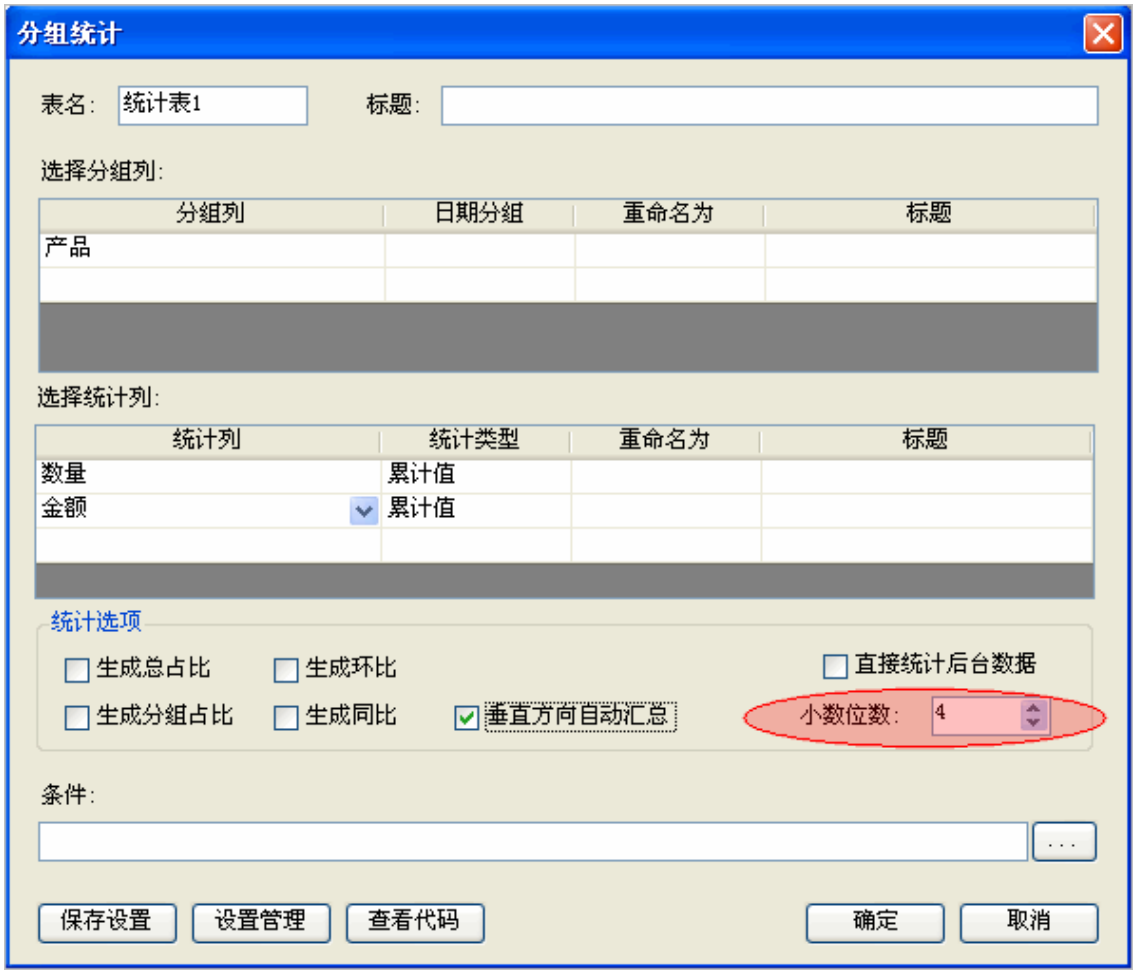
利用上述四个按钮，可以更加便捷地完成日常的统计工作，例如我们筛选或者输入新的数据后，只需单击“重新分组统计”按钮，即可根据上一次的设置，重新生成分组统计表，如果之前没有进行分组统计，则会出现分组统计设置窗口。

如果你的屏幕分辨率低于 1280，则不会出现“快速统计”功能组，不过你仍然可以使用快速统计功能，不过要多按两次鼠标；例如要根据上次设置，重新生成分组统计表，只能执行下图所示的菜单命令：



3.11.8 统计精度

分组统计和交叉统计默认的统计精度为 4 位小数，用户可以自己调整统计精度：



3.11.9 统计条件

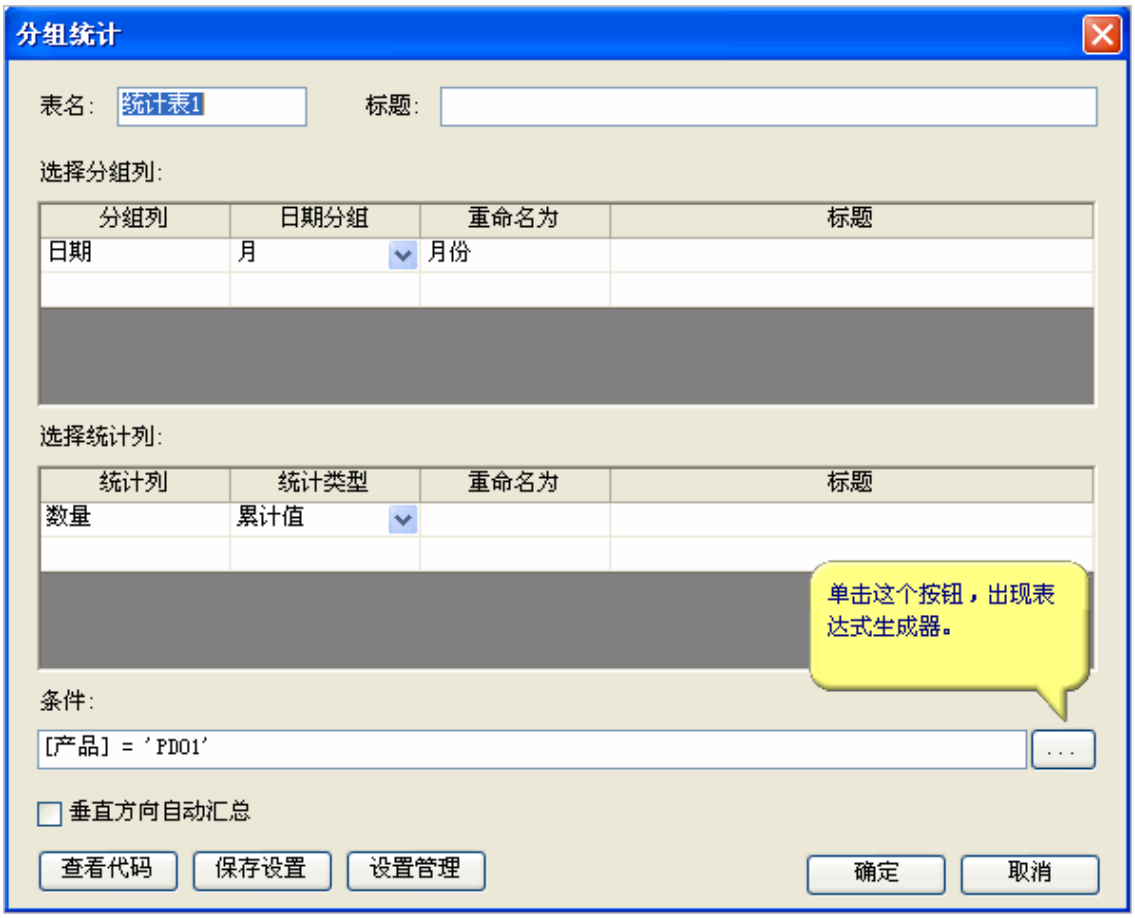
在分组统计和交叉统计设置窗口，都可以设置统计条件。

统计条件是一个条件表达式，只有符合条件的行才会参与统计。

例如下图设置了条件：

[产品] = 'PD01'

也就是说统计产品 PD01 的各月销售额：



通常直接在表中筛选出符合条件的行，然后再统计，会更方便一些。

如果你希望预先定义好一些统计设置，并保存下来供以后快速调用执行，此时统计条件就可能派上用场。

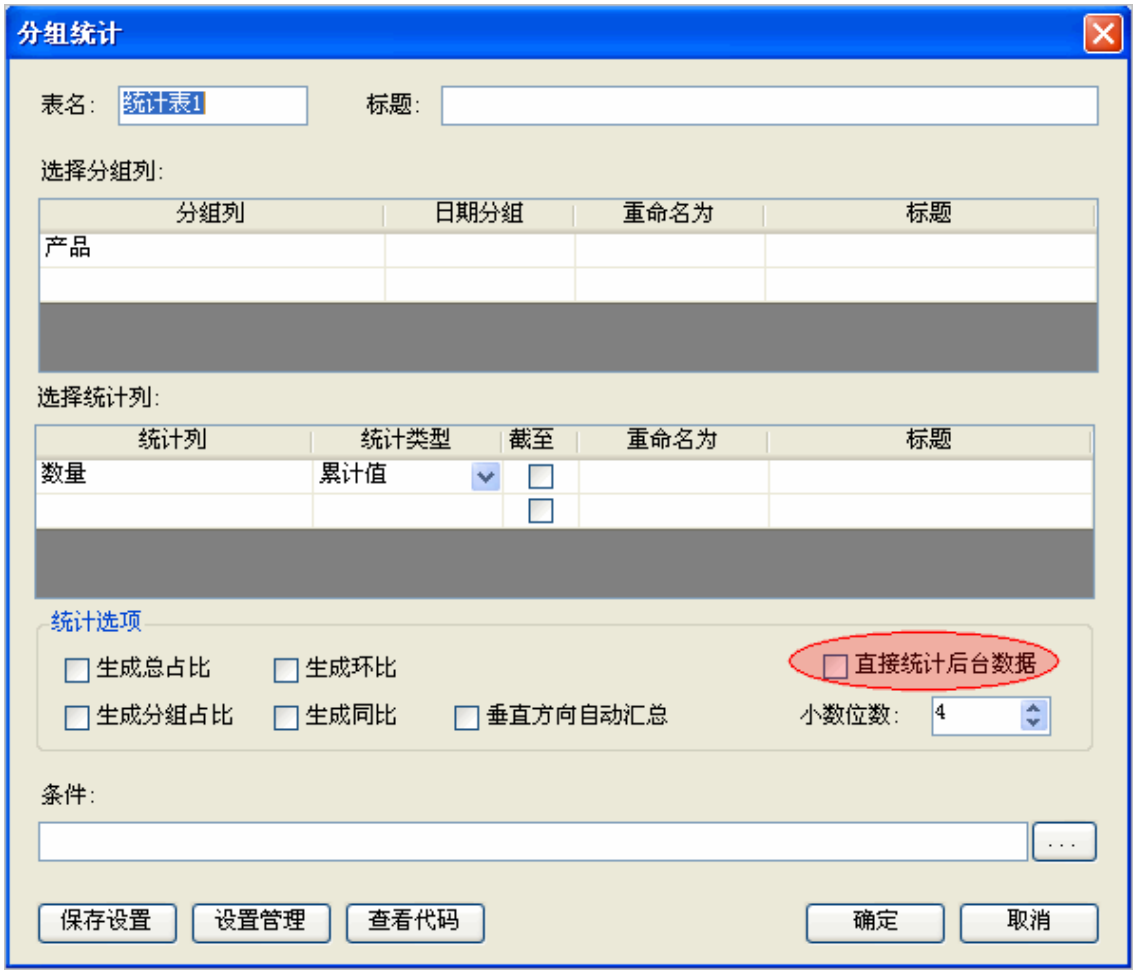
请参考表达式的 运算符和函数 以及 条件表达式

3.11.10 统计后台数据

普通用户可以暂时忽略本节内容。

显然，面对海量数据，我们是不可能将所有数据加载到 Foxtable 中的，毕竟电脑的容量和处理能力都是有限的，所以 Foxtable 提供了动态加载数据的功能，可以根据需要加载符合条件的部分数据到 Foxtable 中进行处理，这样就使海量数据的管理成为可能。

虽然利用动态加载，可以只加载部分符合条件的数据到 Foxtable 中，但是数据统计却往往是针对所有数据的。不管是分组统计，还是交叉统计，默认都是统计已经加载的数据，不过这两个统计都提供了一个“直接统计后台数据”的选项，有了这个选项，我们就可以直接统计后台的海量数据，而无须将其加载到 Foxtable 中。



关于动态加载数据，开发篇会有专门章节进行详细介绍。

3.11.11 自动计算

在默认情况下，选定连续的多个数值单元格，状态栏右侧会出现选定单元格的数值累计值：



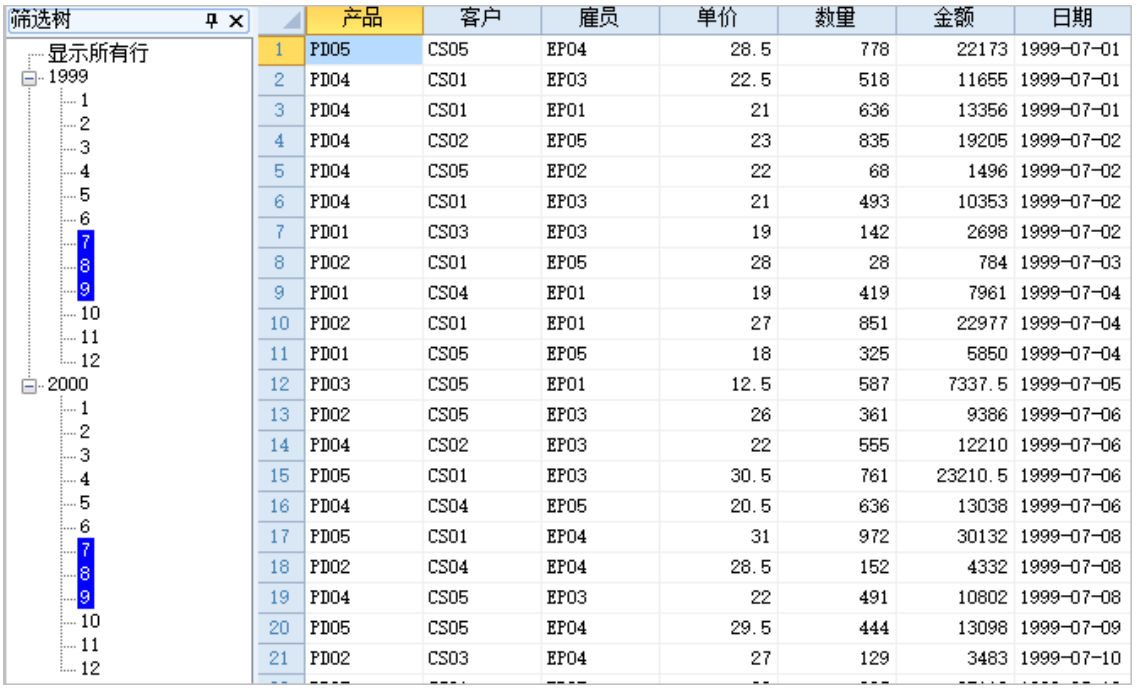
在“杂项”功能区，有一个“自动计算”功能组，用于启动、停止自动计算，以及选择要计算的内容：



提示：
自动计算是针对数值列的，如果是字符列，即使选定区域输入的是数字，右下角也不会出现计算结果。

3.11.12 筛选树与数据统计

利用筛选树，我们可以很快捷地对特定数据进行统计。
例如需要统计比较相邻两年的 7、8、9 月份的销售数据，只需在筛选树中选择这两年的 7、8、9 月份节点：



	产品	客户	雇员	单价	数量	金额	日期
1	PD05	CS05	EP04	28.5	778	22173	1999-07-01
2	PD04	CS01	EP03	22.5	518	11655	1999-07-01
3	PD04	CS01	EP01	21	636	13356	1999-07-01
4	PD04	CS02	EP05	23	835	19205	1999-07-02
5	PD04	CS05	EP02	22	68	1496	1999-07-02
6	PD04	CS01	EP03	21	493	10353	1999-07-02
7	PD01	CS03	EP03	19	142	2698	1999-07-02
8	PD02	CS01	EP05	28	28	784	1999-07-03
9	PD01	CS04	EP01	19	419	7961	1999-07-04
10	PD02	CS01	EP01	27	851	22977	1999-07-04
11	PD01	CS05	EP05	18	325	5850	1999-07-04
12	PD03	CS05	EP01	12.5	587	7337.5	1999-07-05
13	PD02	CS05	EP03	26	361	9386	1999-07-06
14	PD04	CS02	EP03	22	555	12210	1999-07-06
15	PD05	CS01	EP03	30.5	761	23210.5	1999-07-06
16	PD04	CS04	EP05	20.5	636	13038	1999-07-06
17	PD05	CS01	EP04	31	972	30132	1999-07-08
18	PD02	CS04	EP04	28.5	152	4332	1999-07-08
19	PD04	CS05	EP03	22	491	10802	1999-07-08
20	PD05	CS05	EP04	29.5	444	13098	1999-07-09
21	PD02	CS03	EP04	27	129	3483	1999-07-10

然后根据产品（水平分组）按年月（垂直分组）对数量进行交叉统计：



得到的统计结果为：

	产品	1999年			2000年		
		7月	8月	9月	7月	8月	9月
1	PD01	5204	8103	11014	6861	6830	9805
2	PD02	6807	9755	8657	12682	8774	8897
3	PD03	6088	8579	10107	12709	9137	5660
4	PD04	11864	13565	12863	3821	7926	6915
5	PD05	9332	11102	8595	7580	13786	6433

3.11.13 图示统计数据

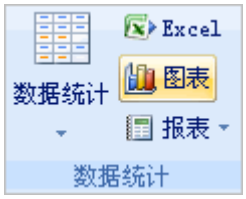
3.11.13.1 图示统计数据

图表可以用最直观的方式展示统计结果。

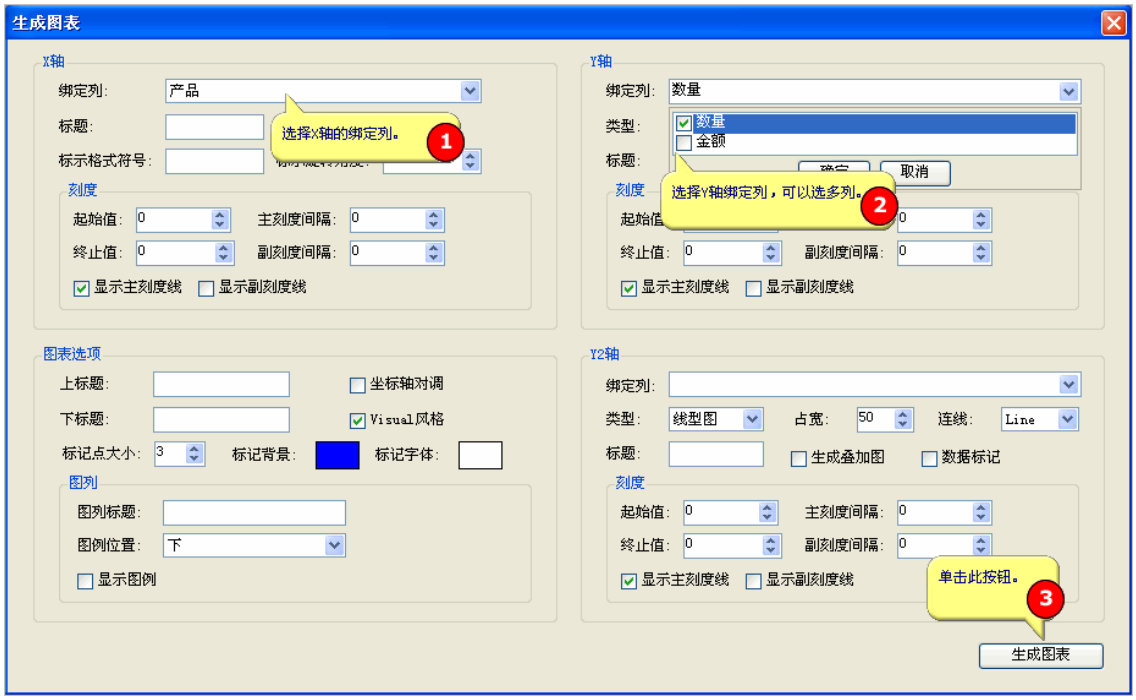
在 Foxtable 中生成图表非常简单，例如对于下面的表：

	产品	数量	金额
1	PD01	11290	171754.92
2	PD02	18200	296522.5
3	PD03	7000	61490
4	PD04	5480	102861
5	PD05	10351	192190.15

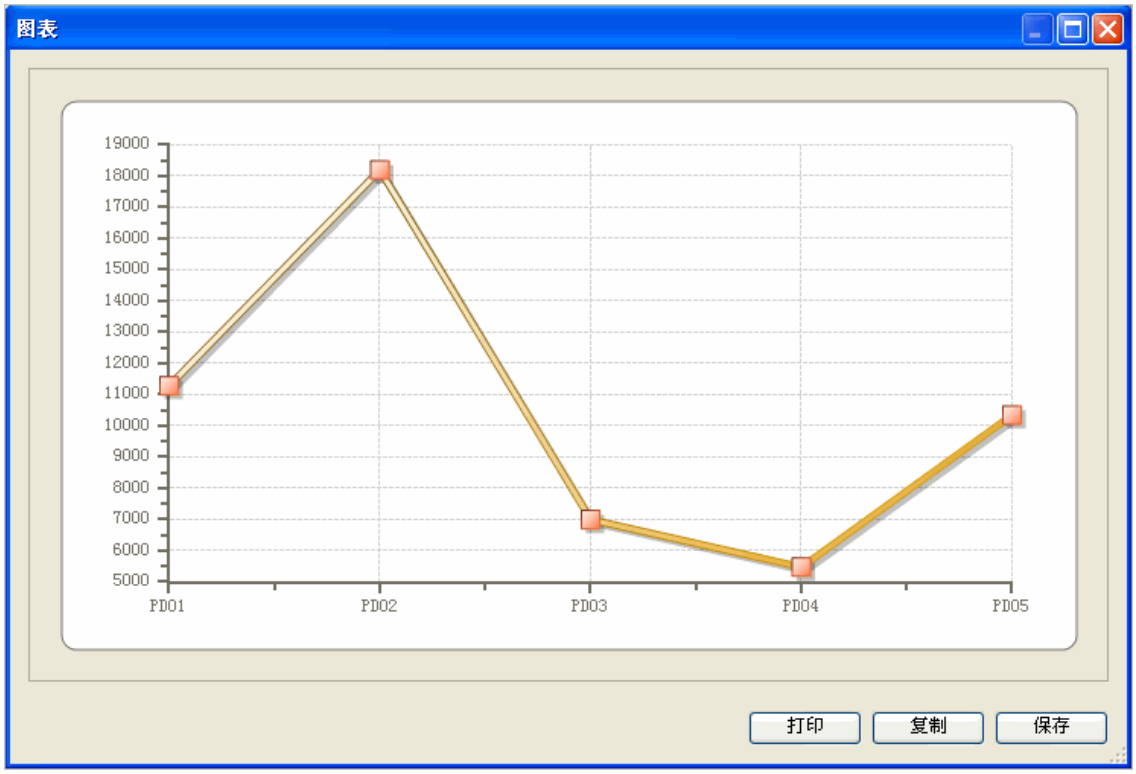
只需单击菜单“日常工作”区的“图表”按钮：



然后按下图所示选择 X 轴和 Y 轴的绑定列，然后单击“生成图表”按钮：

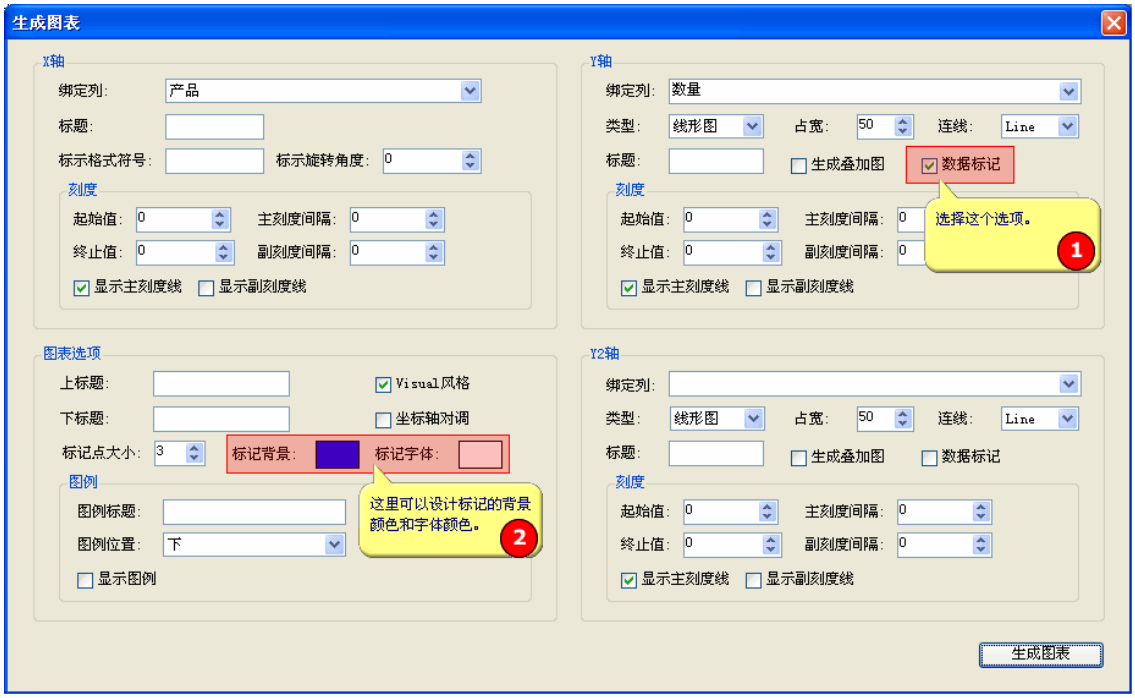


即可生成以下图表：

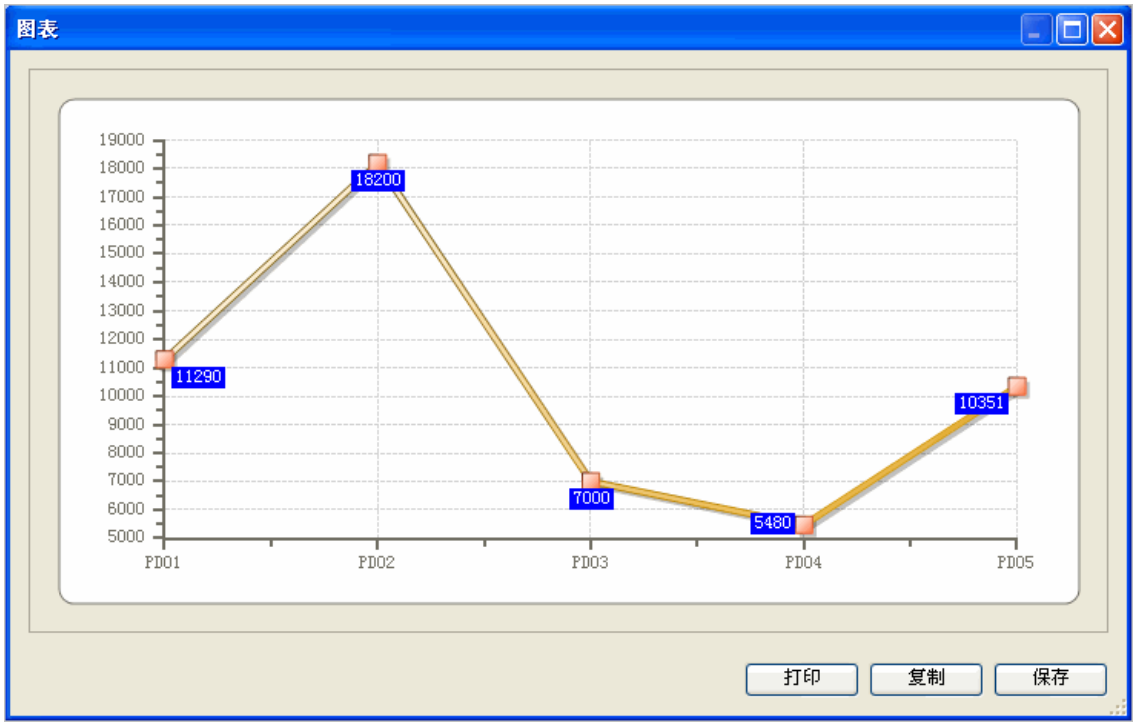


3.11.13.2 数据标记

如果要在图表中标记出具体的数值，可以按下图所示设置：



这是生成的图表：



默认的数据标记是蓝底白字，你可以根据需要修改。

3.11.13.3 连线类型

线形图有三种连线方式，分别是：

Line： 直线

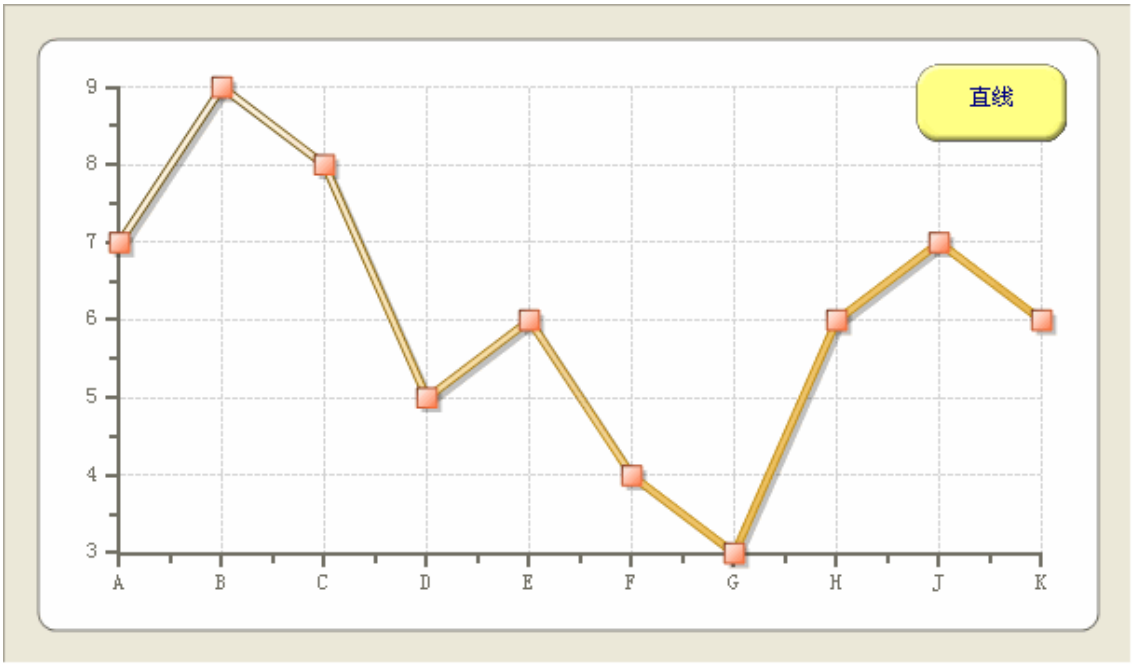
Spline： 曲线

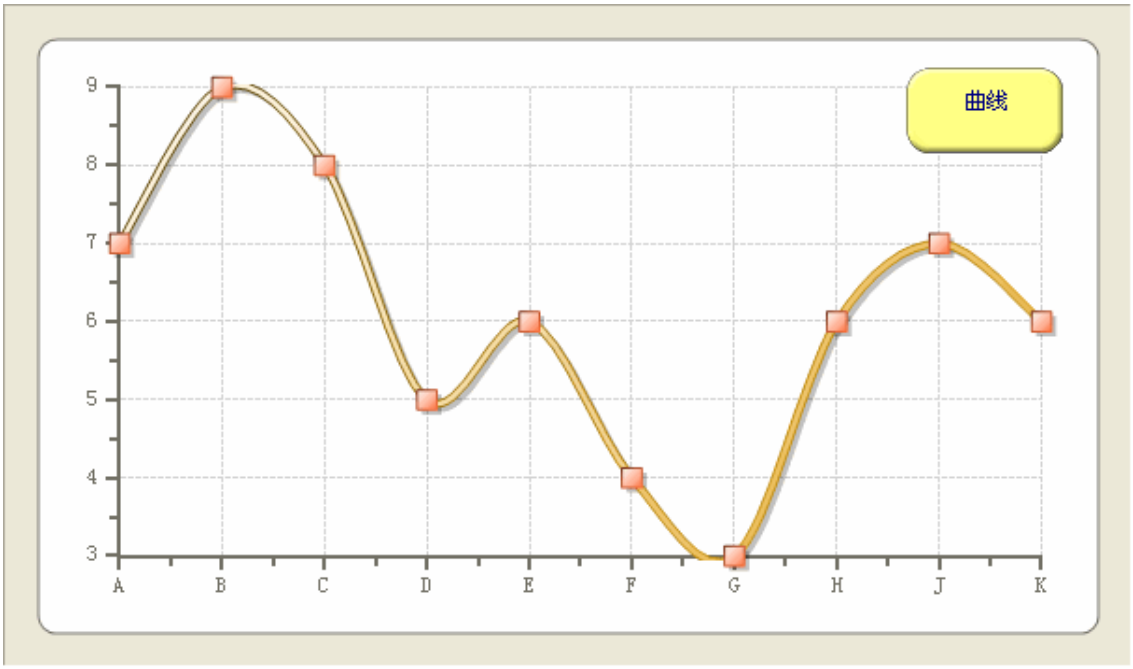
Beziers：贝塞尔曲线

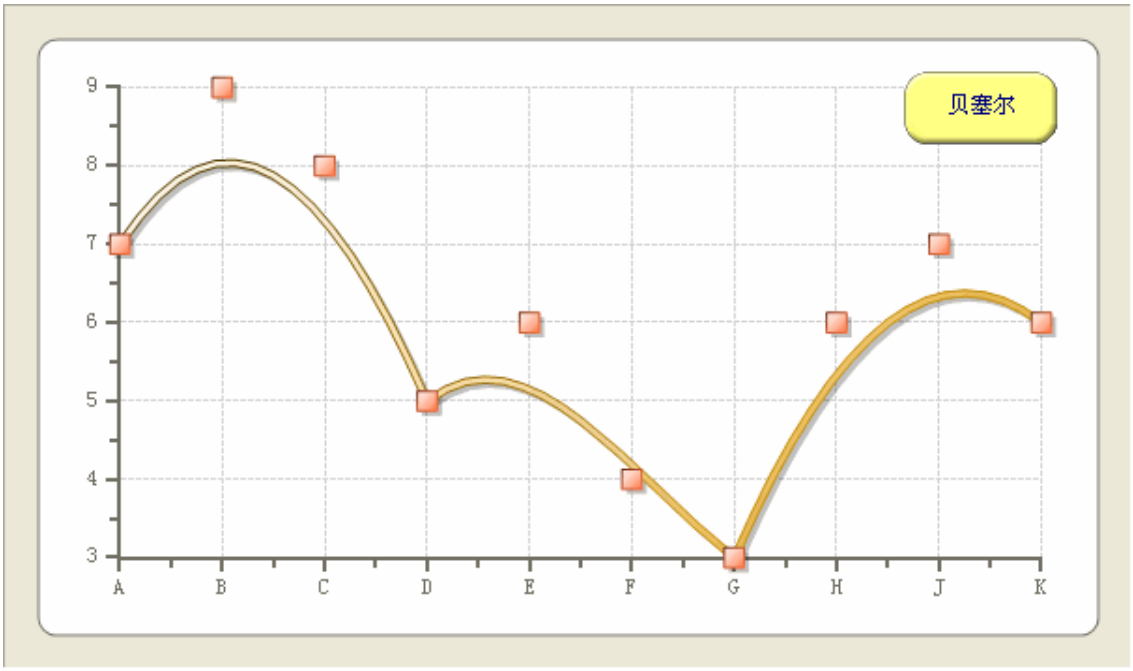
可以通过下面的选项设置连线方式：



三种连线的效果：


直线
9
8
7
6
5
4
3
A
B
C
D
E
F
G
H
J
K


曲线
9
8
7
6
5
4
3
A
B
C
D
E
F
G
H
J
K



3.11.13.4 标示格式符号

X 轴的标示格式符号用于设置 X 轴的标示格式，类似于 Format 函数。

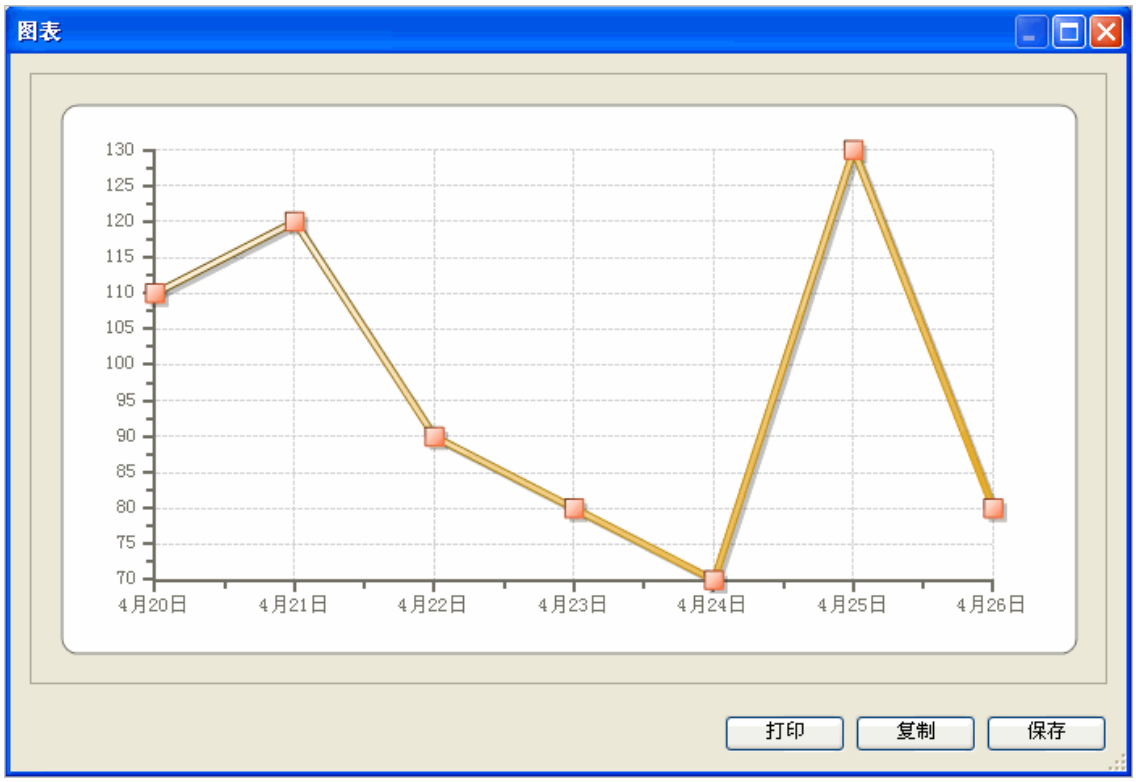
例如对于下图的表：

日期	数量
2010-04-20	110
2010-04-21	120
2010-04-22	90
2010-04-23	80
2010-04-24	70
2010-04-25	130
2010-04-26	80

可以按下图所示设置格式符号：



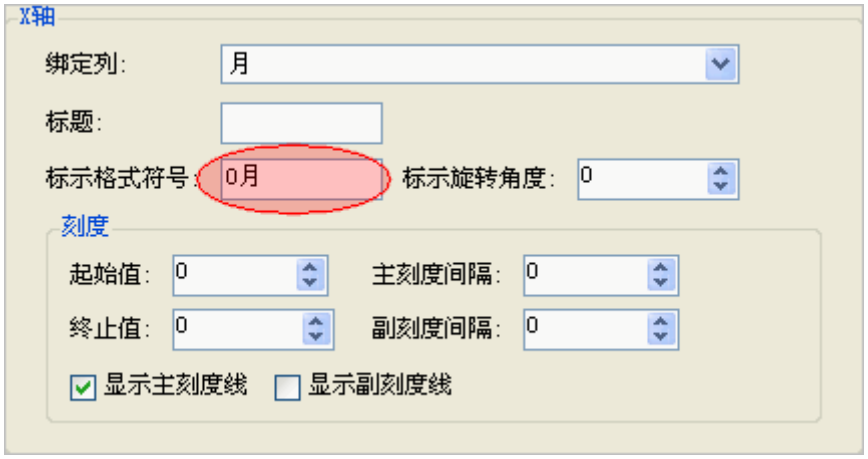
得到的图表为：



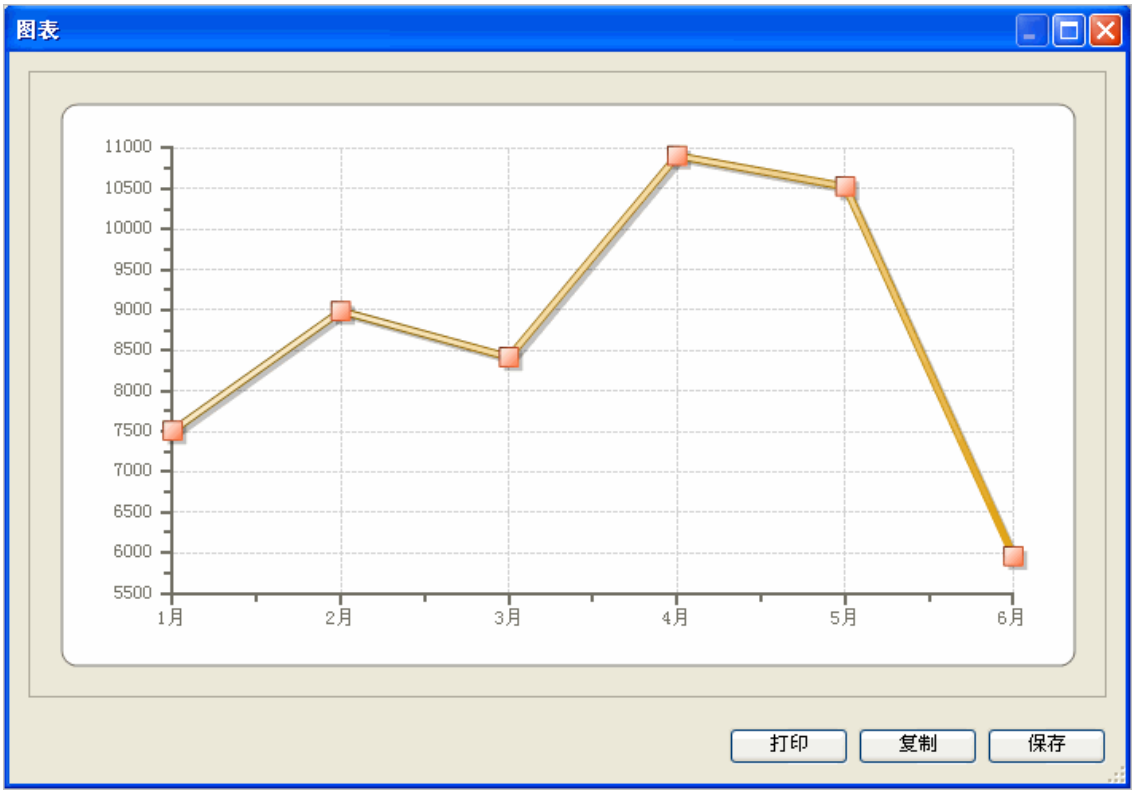
可以用 0 表示分组值，再例如对于下面的表：

	月	数量
1	1	7511
2	2	8990
3	3	8420
4	4	10910
5	5	10530
6	6	5960

可以按下图所示设置格式符号：



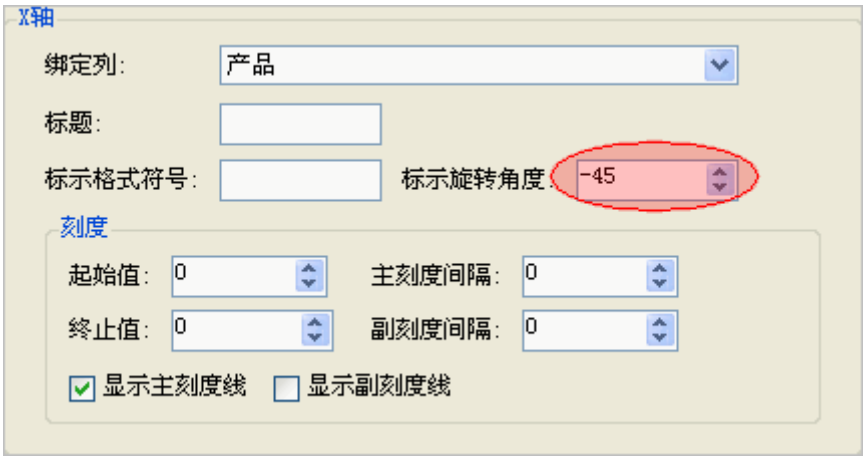
得到的图表为：



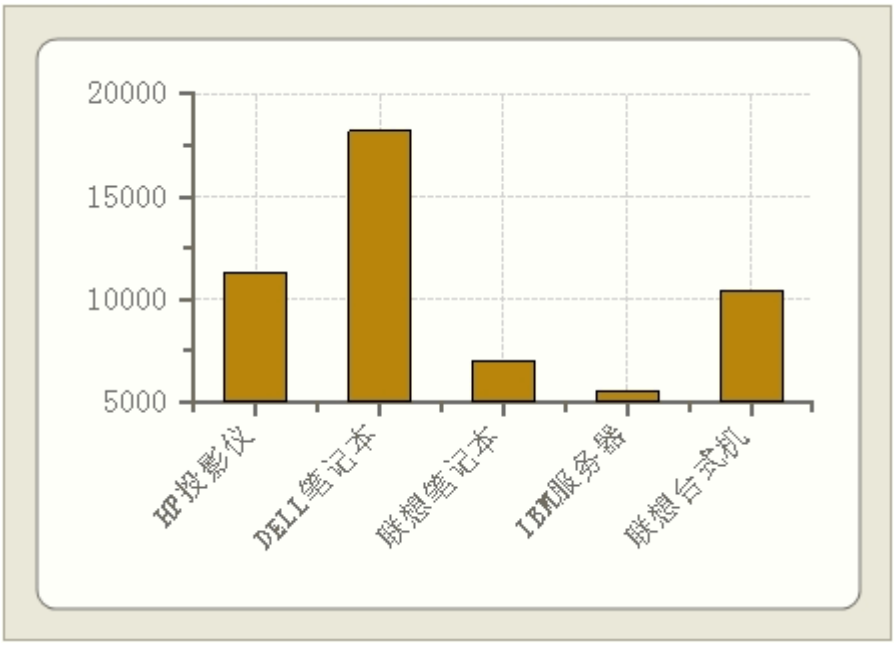
3.11.13.5 标示旋转角度

如果 X 轴的标示较长，可以设置 X 轴标示的旋转角度，以便完整地显示标示。

例如将旋转角度设置为逆时针旋转 45 度(负数表示逆时针)：



可以生成下图所示的图表：

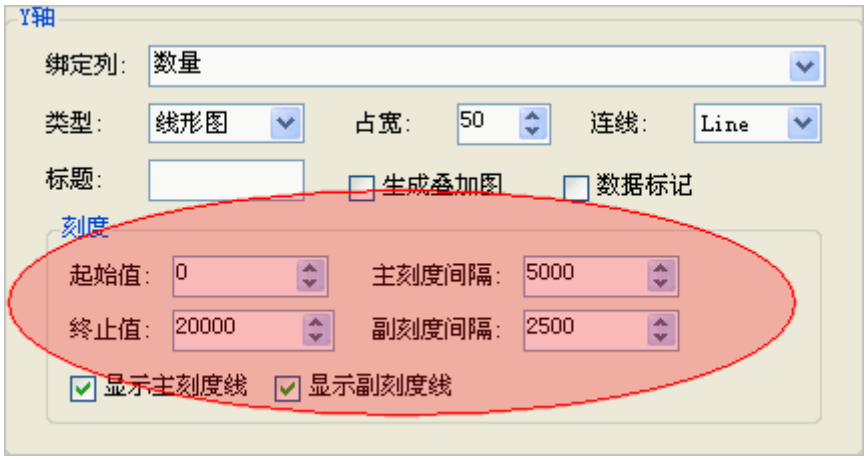


3.11.13.6 刻度设置

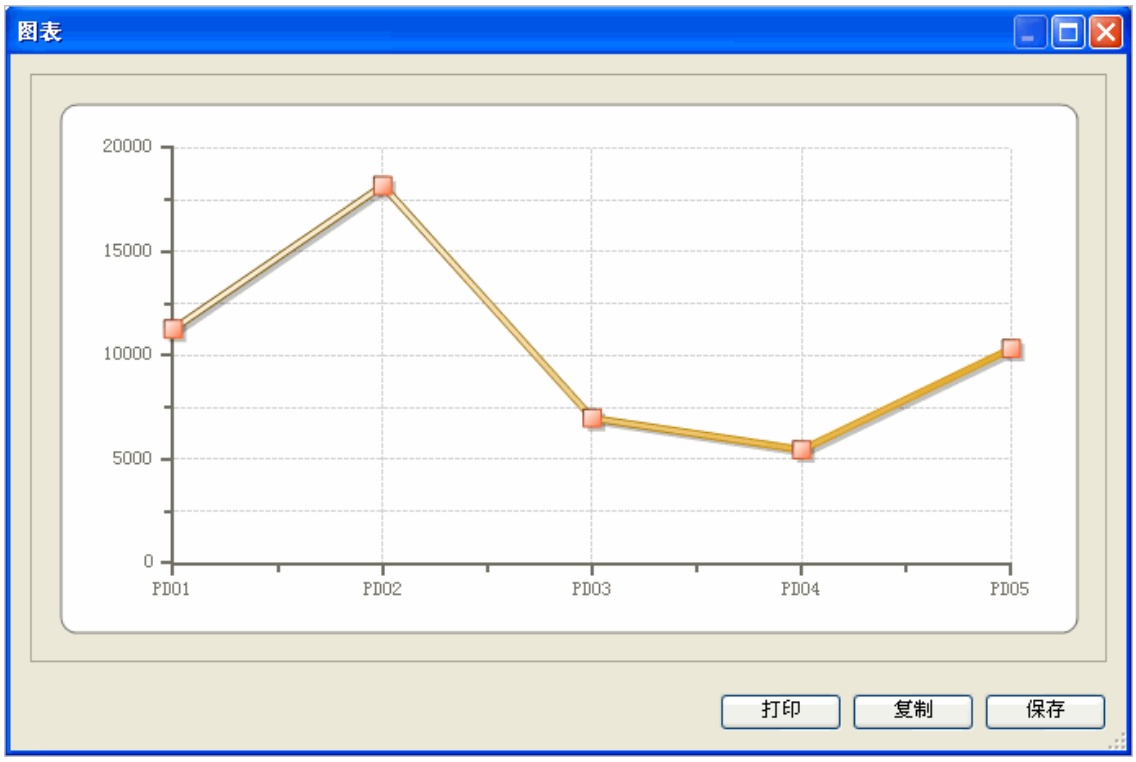
在默认情况下，系统会自动确定每个坐标轴的起始值和终止值、主刻度和副刻度的间隔，并显示主刻度线。

可以根据需要改变这种默认的设置。

例如按照下图所示设置 Y 轴的刻度：

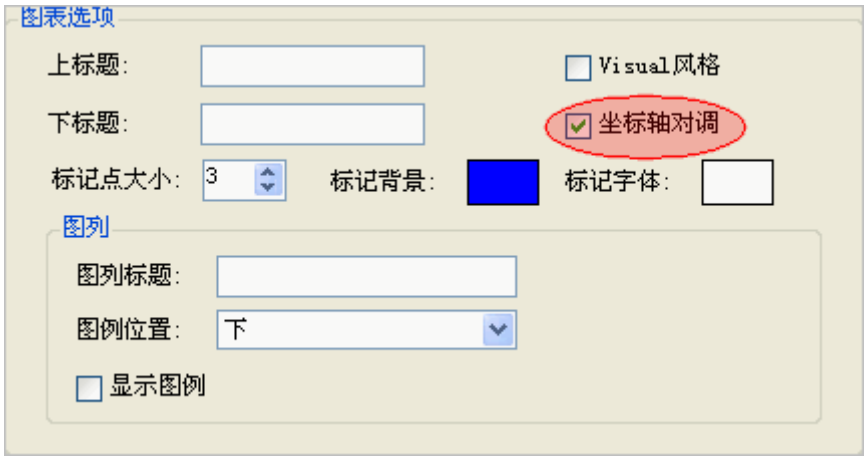


得到的图表为：

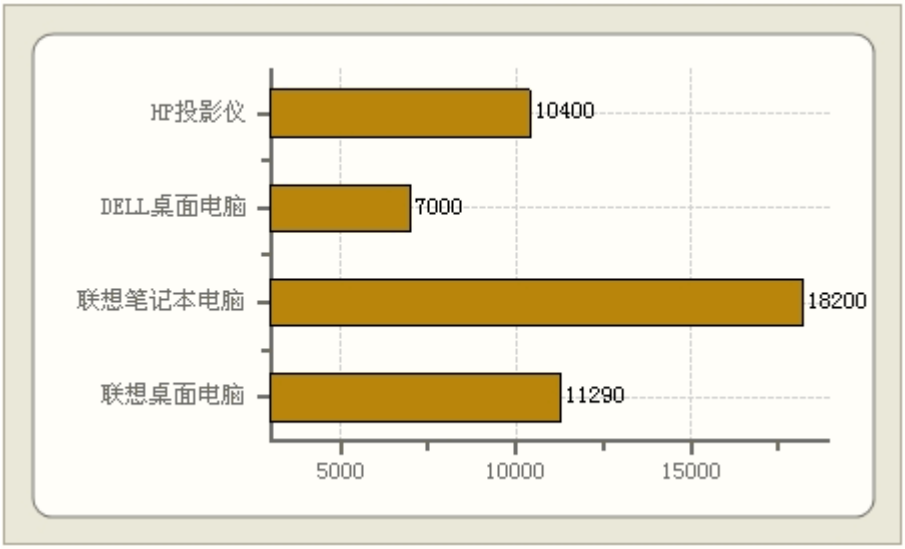


3.11.13.7 对调坐标轴

有的时候 X 轴的标示内容很长，可以选择选项“对调坐标轴”，可方便完整地显示标示：



得到的图表为：



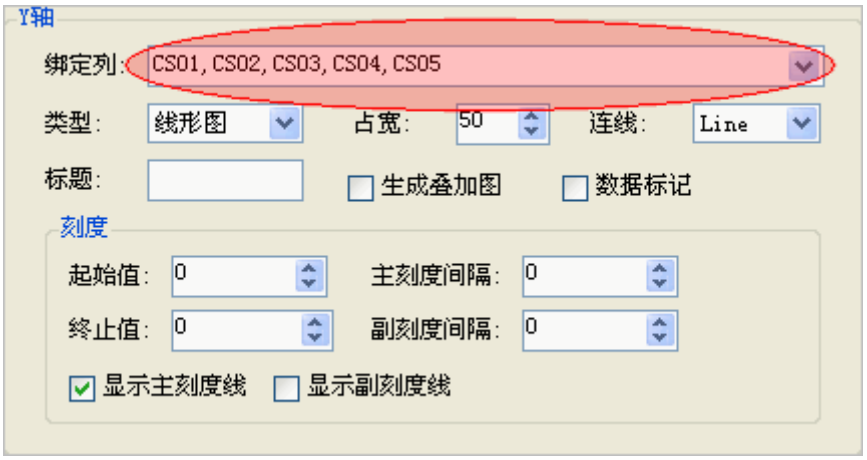
3.11.13.8 图示多列数据

Foxtable 可以图示多列数据。

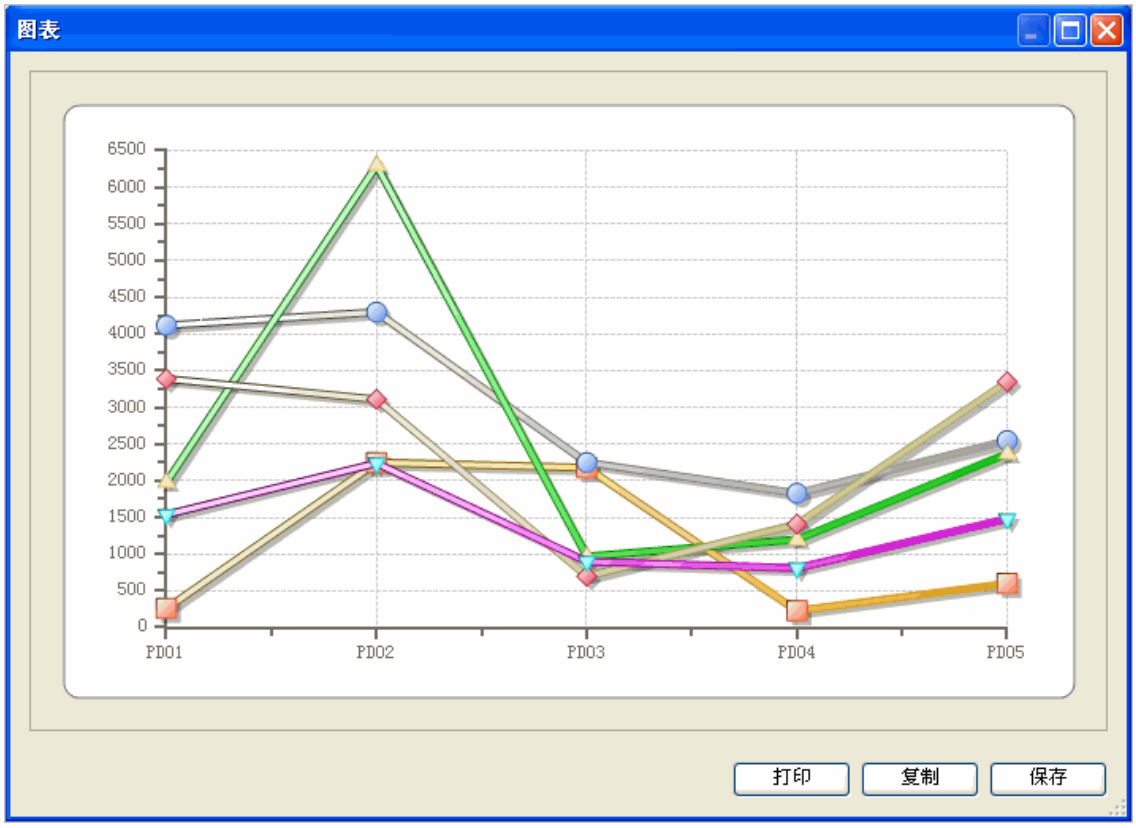
例如对于下面的表：

	产品	CS01	CS02	CS03	CS04	CS05	合计
1	PD01	260	4120	1980	3390	1540	11290
2	PD02	2250	4300	6300	3110	2240	18200
3	PD03	2180	2250	970	700	900	7000
4	PD04	230	1830	1200	1410	810	5480
5	PD05	600	2550	2371	3350	1480	10351

Y 轴可以选择多列：



生成的图表为:



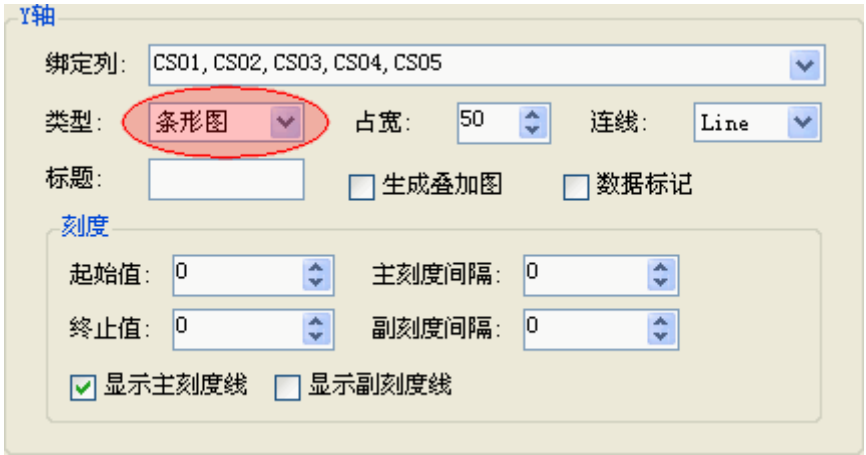
3.11.13.9 条形图

之前使用的都是线形图，下面介绍条形图。

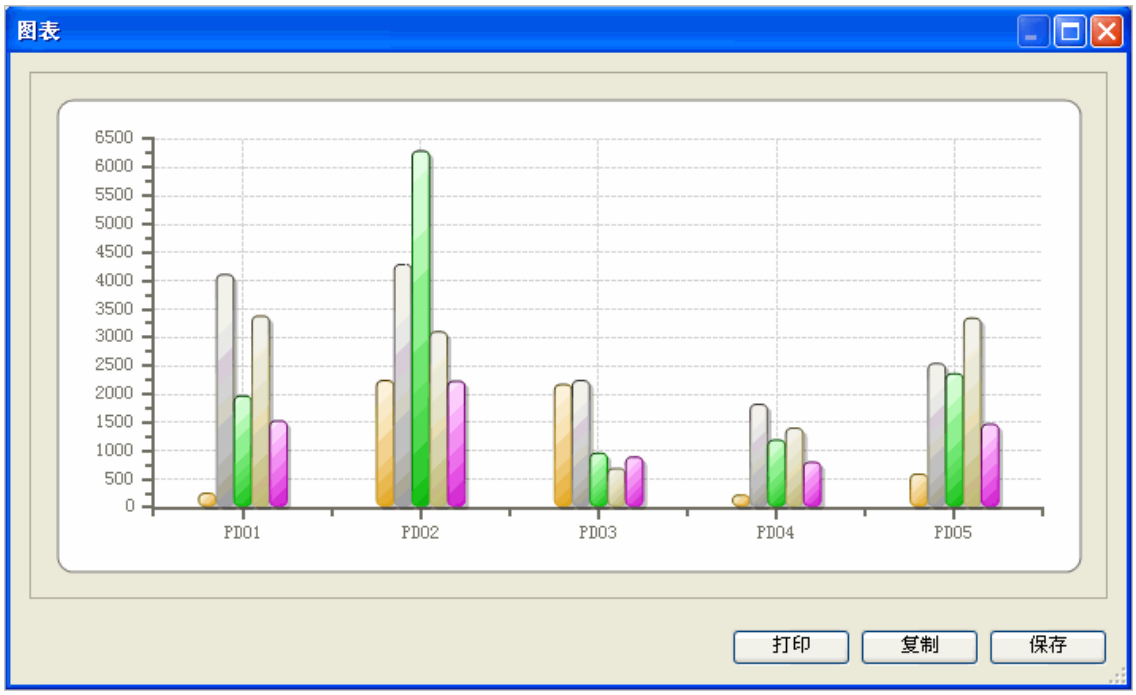
例如对于下面的表：

	产品	CS01	CS02	CS03	CS04	CS05	合计
1	PD01	260	4120	1980	3390	1540	11290
2	PD02	2250	4300	6300	3110	2240	18200
3	PD03	2180	2250	970	700	900	7000
4	PD04	230	1830	1200	1410	810	5480
5	PD05	600	2550	2371	3350	1480	10351

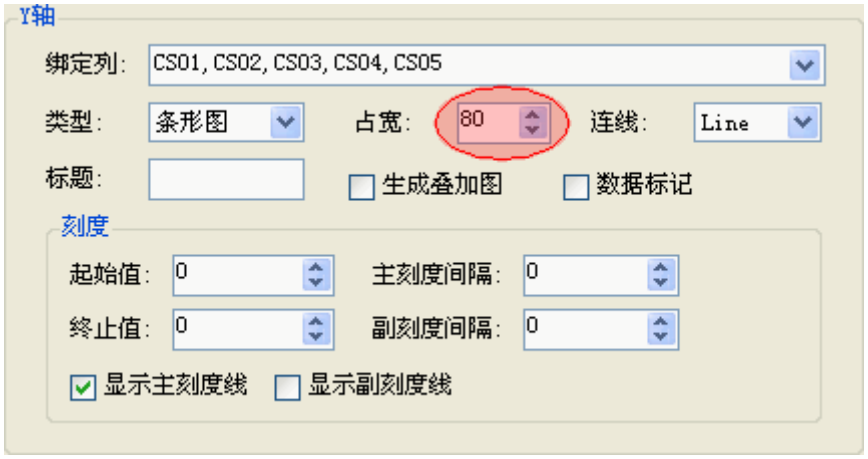
按下图所示设置：



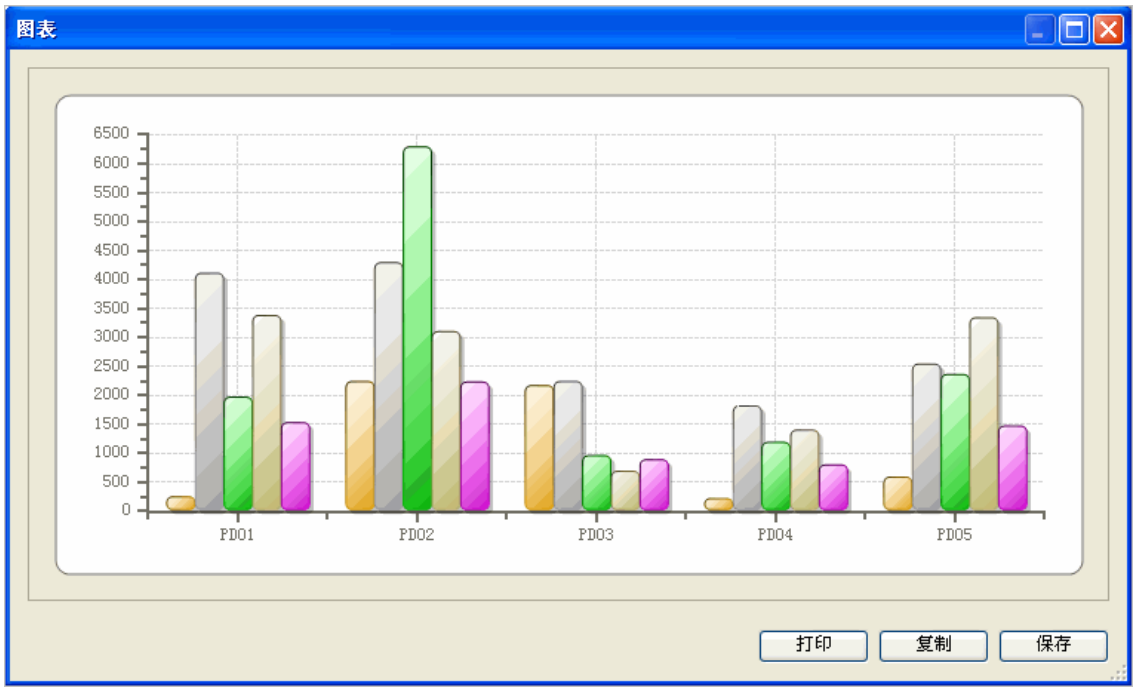
生成的图表为：



条形图默认的占宽为 50%，可以通过下面的设置，调整占宽，例如下图将占宽调整为 80%：

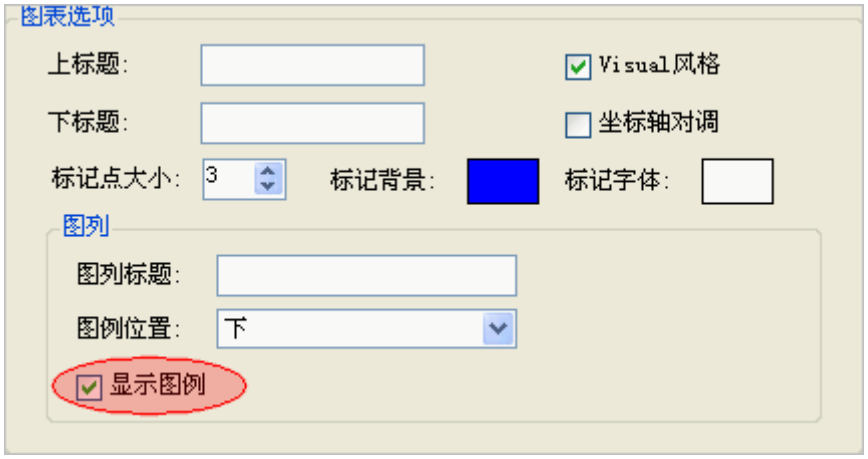


生成的图表为：

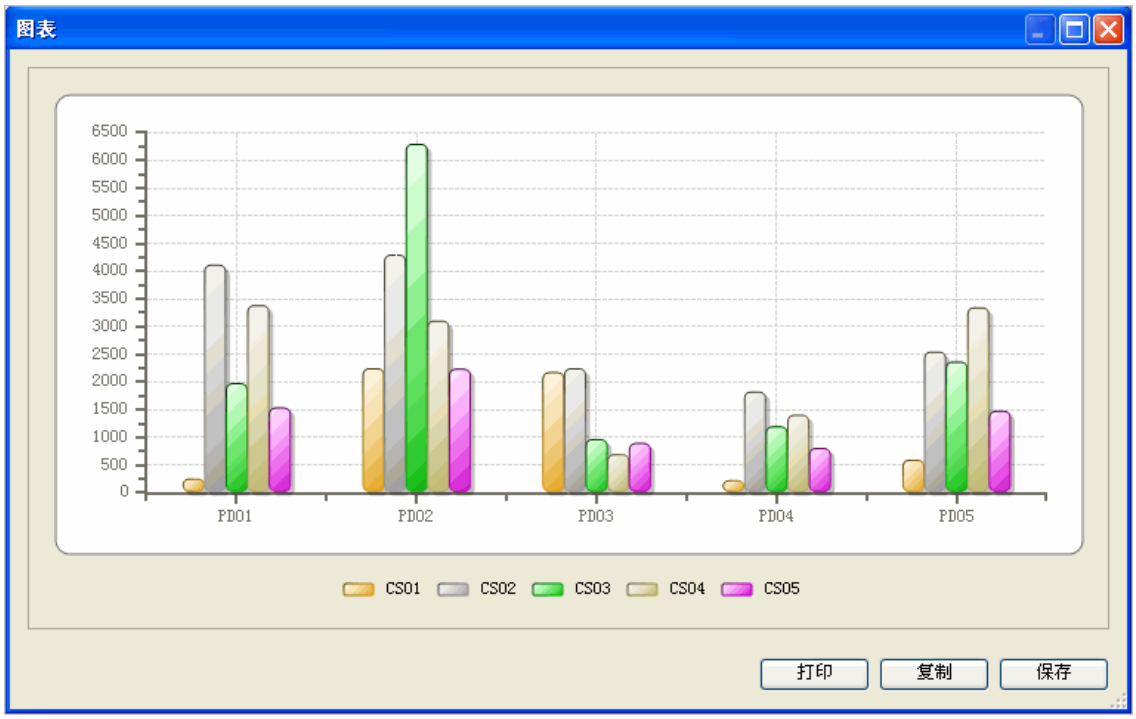


3.11.13.10 显示图例

图示多列数据的时候，应该选择选项“显示图例”：



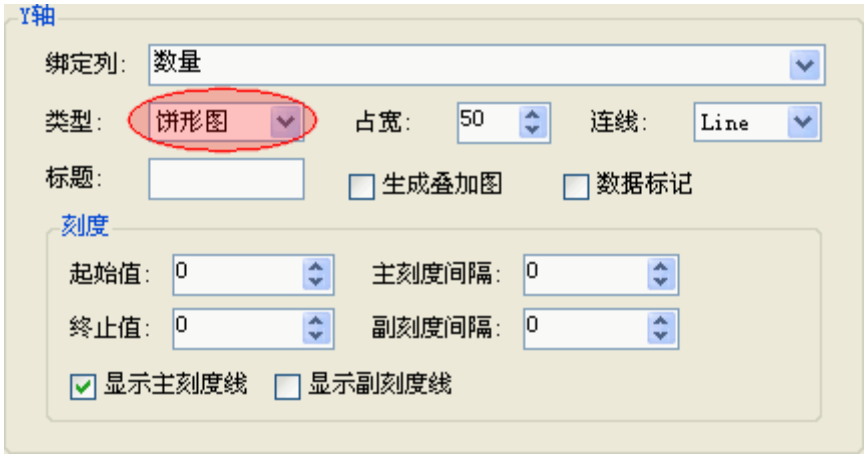
这是生成的图表，下端列出了图例：



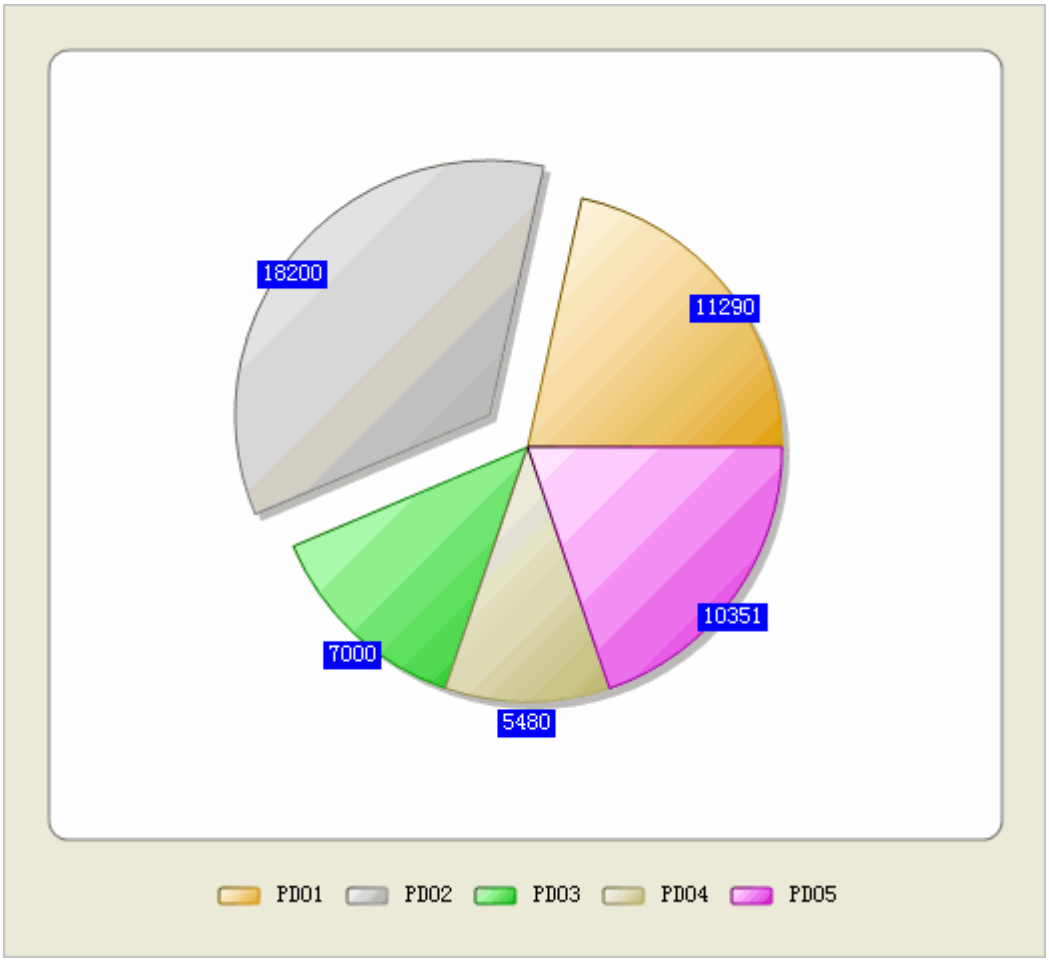
3.11.13.11 饼形图

除了线形图和条形图，还有饼形图。

例如按下图设置：



得到的图表为：



在饼型图中，单击某个扇形，即可让其离开圆心，再次单击可回到原位置。

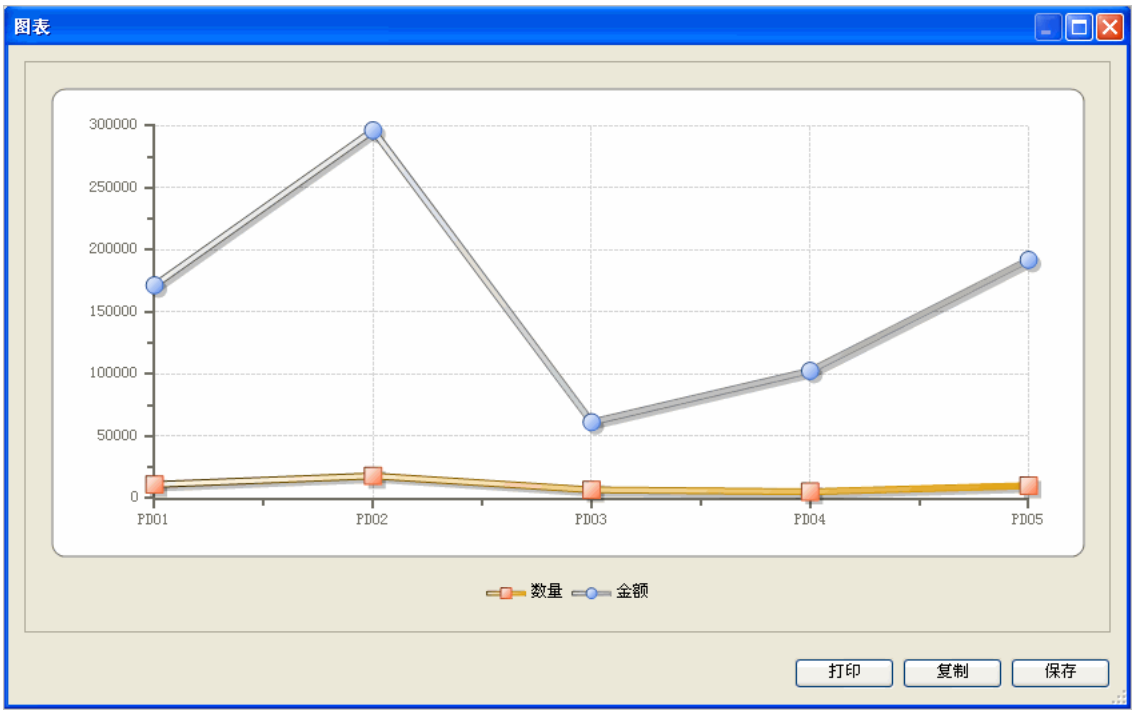
3.11.13.12 使用双图表

3.11.13.13 启用双图表

假定要图示下表的数据，同时图示金额和数量列：

	产品	数量	金额
1	PD01	11290	171754.92
2	PD02	18200	296522.5
3	PD03	7000	61490
4	PD04	5480	102861
5	PD05	10351	192190.15

得到的图表为：

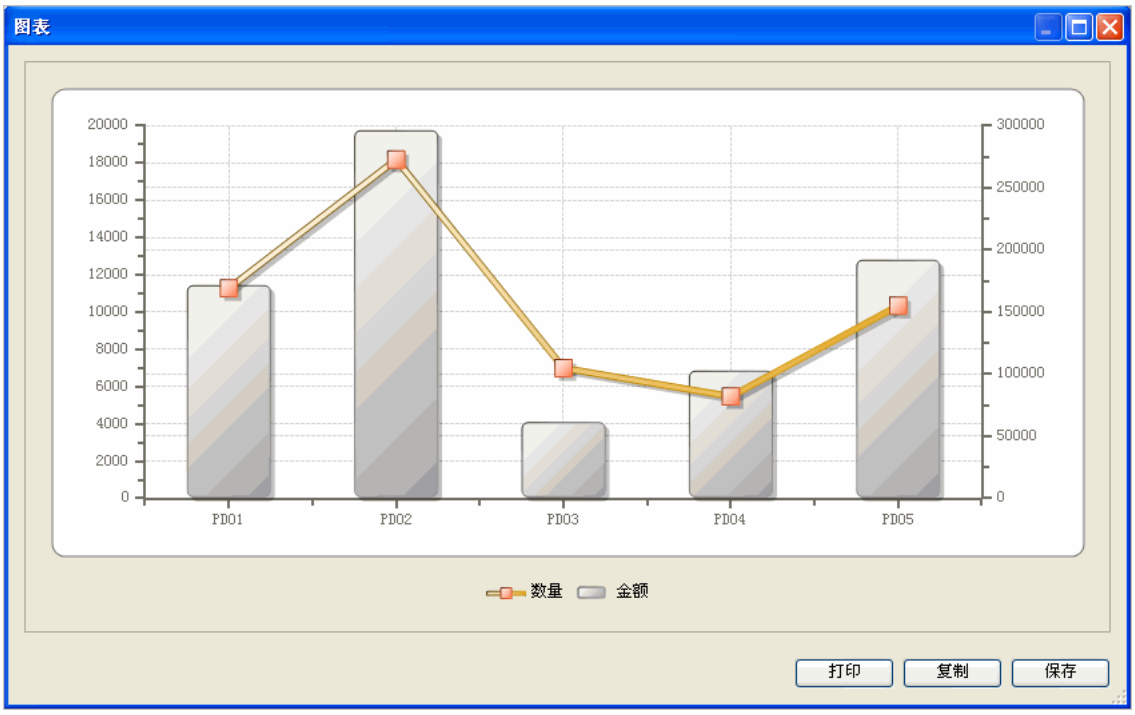


由于数量和金额的值相差比很大，所以上图中数量的图示效果不明显。

为解决这个问题，可以启用双图表，Y 轴图示数量列，Y2 轴图示金额列，而且一个使用线形图，一个使用条形图：



得到的图表为：



3.12 打印

3.12.1 打印表

在“打印输出”功能区，单击“打印设置”按钮，除了通常的页面设置外，还可以设置以下内容：

标题

通常将主标题设为“[TableName]”，表示表名。

副标题显示在标题下方，分左、中、右三部分，最常用的是中副标题。

后间距指的是标题和随后内容之间的距离。

标题默认只在首页打印，如果希望每页都打印，请不要选择选项“仅在首页打印标题”。



页眉页脚

页眉和页脚分为左、中、右三部分。

页眉的后间距表示页眉和正文之间的距离。

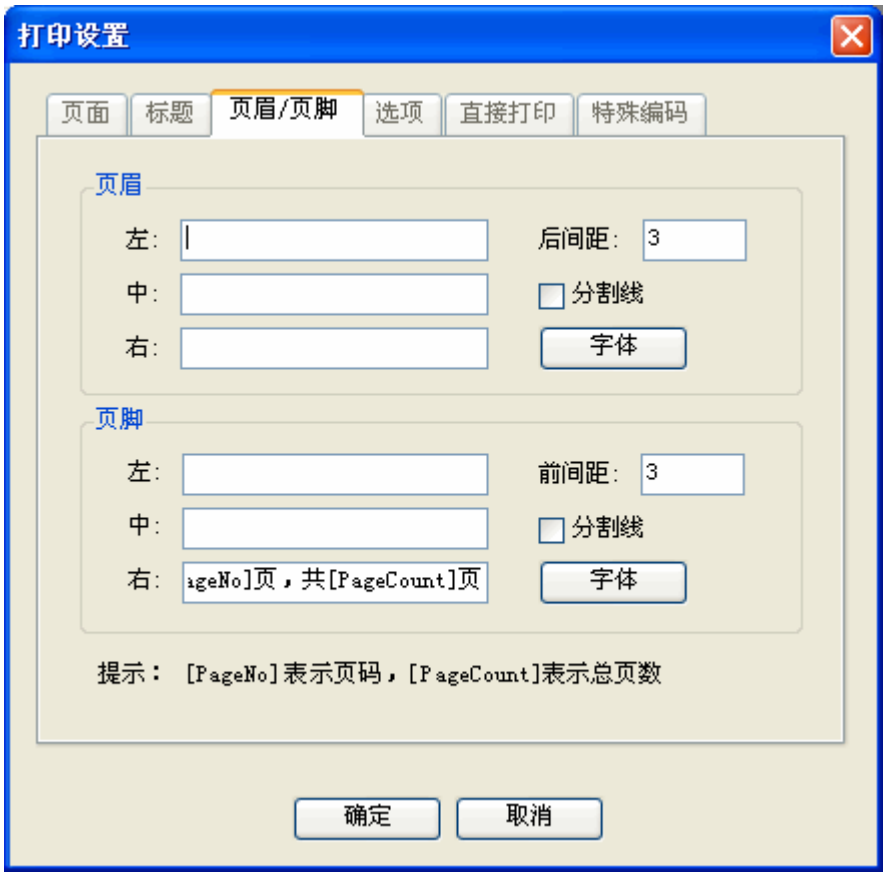
页脚的前间距表示正文和页脚之间的距离。

“分割线”选项用于决定是否在正文和页眉页脚之间，显示一条细横线。

页眉和页脚可以分别设置字体。

默认的页脚是：第[PageNo]页，共[PageCount]页 。

[PageNo]表示页码，[PageCount]表示总页数。



在标题、页眉、页脚中可以使用的特殊编码有：

[PageNo]：页码

[PageCount]：总页数

[Today]：日期

[Time]：时间

[ShortTime]：时间，不含秒

[Year]：年

[Month]：月

[Day]：日

[Hour]：时

[Minute]：分

[Second]：秒

[WeekDay]：星期几

[TableName]：表名

打印选项

- 适应页宽
  自动缩小或加宽表格，使得表格的宽度恰好等于页宽。

- 自动扩展
  当表格宽度小于页宽时，是否自动加宽最后一列，使得表格的宽度恰好等于页宽。

- 调整字体
  自动调整表宽时，字体大小是否也随之调整。

- 自动换行
  备注列的内容是否自动换行打印。

- 自动行高
  备注列的内容超出列宽时，是否自动调整行高，以容纳所有备注内容。

- 重复标题行
  是否在每一页都打印标题行。

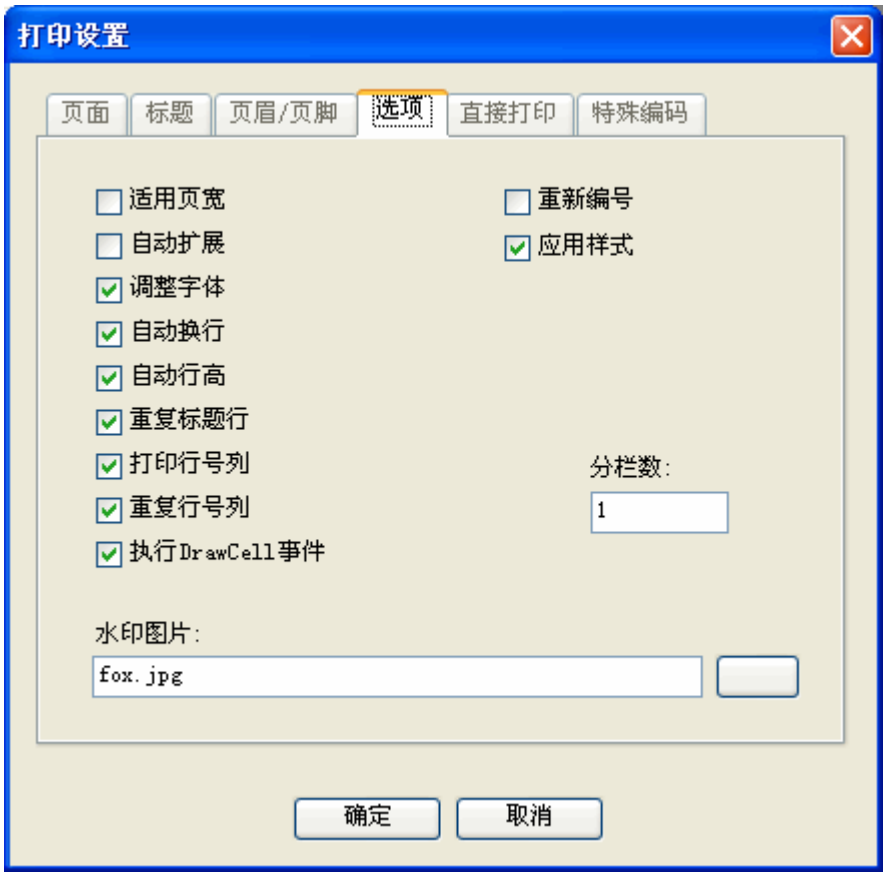
- 打印行号列
  是否打印行号列。

- 重复行号列
  当表格宽度超出页宽，必须水平换页时，是否在水平分页上打印行号列。

- 重新编号
  打印选定行时，是否重新编排行号。例如你选定 7、8、9 三行打印，打印出来的行号也是 7、8、9，如果选择此选项，那么打印出来的行号是 1、2、3。

- 应用样式
- 执行 DrawCell 事件
  这两个属性需要使用开发篇的知识，这里就不讲述了。

除上述选项外，你还可以设置水印图：



工具栏中提供了一些常用的打印设置选项，便于用户快速设置：



直接打印

直接打印是另一种打印方式，完全真实于你所看到的表。

直接打印没有标题，但是可以设置页眉、页脚。

直接打印的选项有：

- 适应页宽

  自动缩小或加宽表格，使得表格的宽度恰好等于页宽。

- 自动扩展

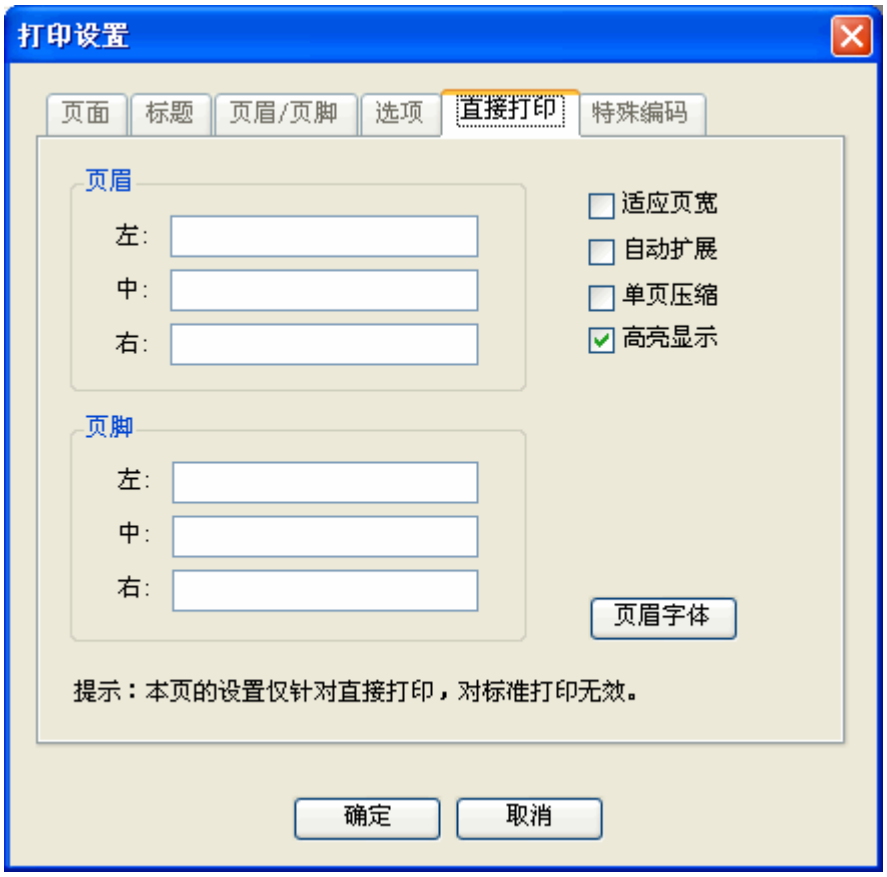
  当表格宽度小于页宽时，是否自动加宽最后一列，使得表格的宽度恰好等于页宽。

- 单页压缩

  压缩表格的宽度和高度，使得恰好能在一页打印完成。

- 高亮显示

  打印的时候，是否突出打印选定的单元格。



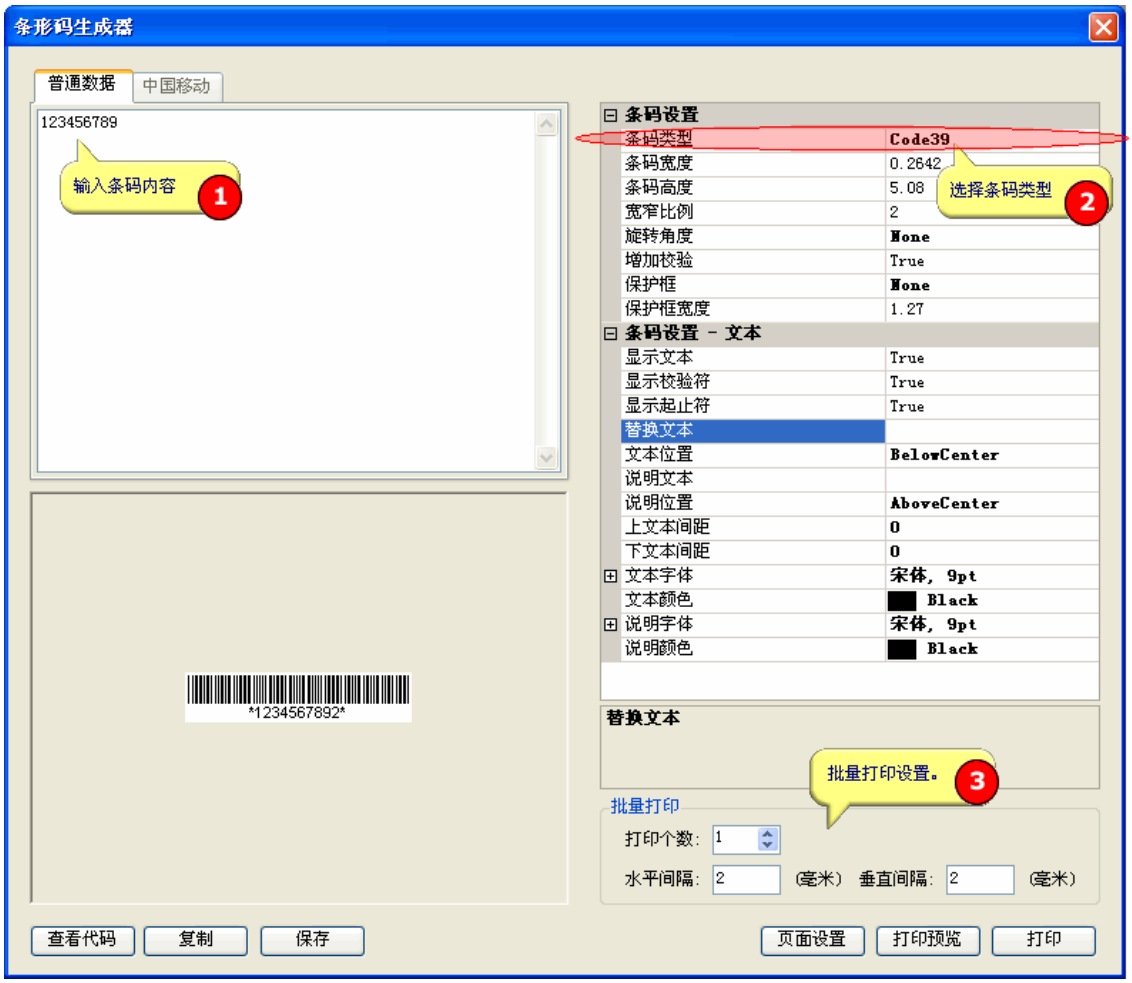
3.12.2 批量打印条形码

商品社会是离不开条形码的。

如果需要临时打印一批条形码，可以使用 Foxtable 提供的批量打印条形码功能，此命令位于菜单的“杂项”功能区，



这是打印窗口：



3.12.3 中国移动的二维码应用

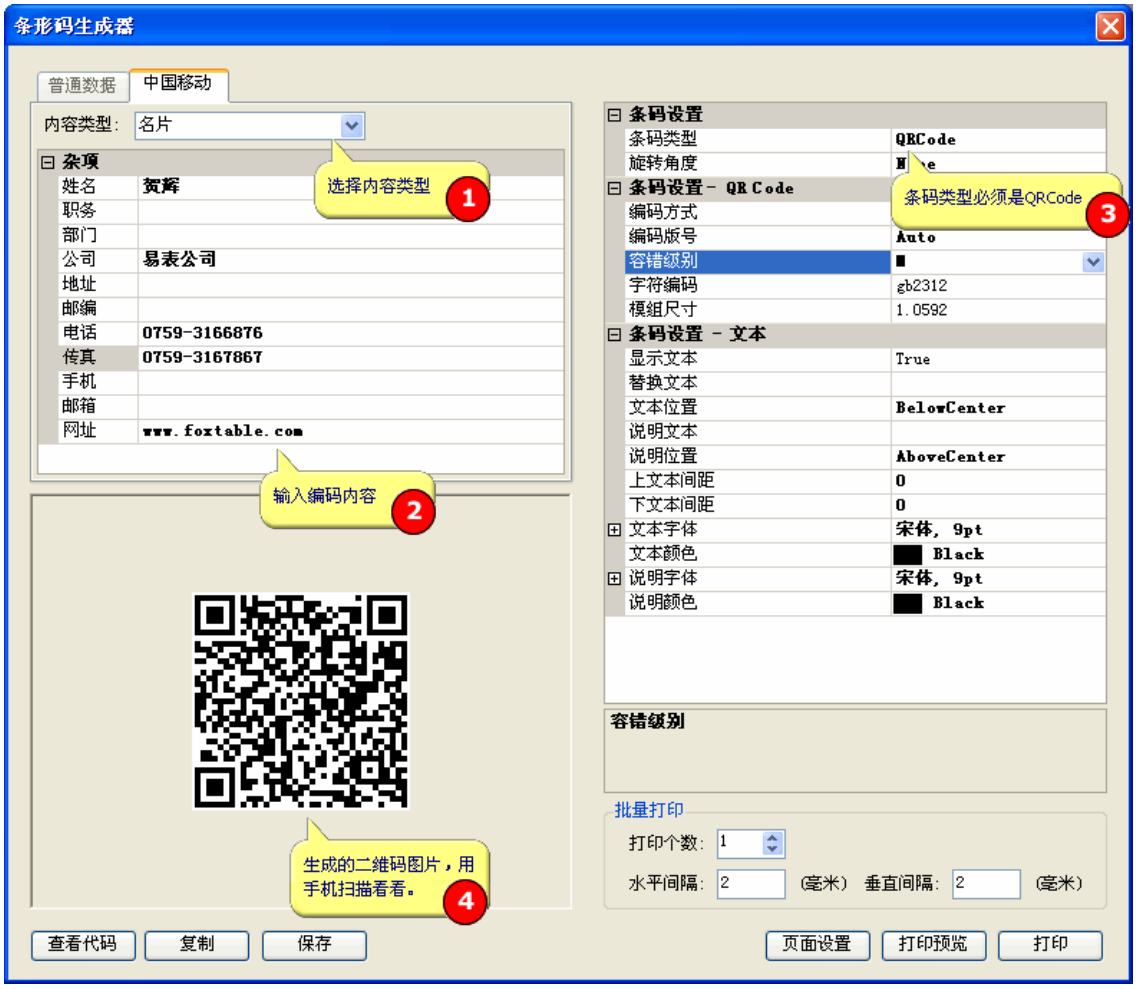
Foxtable 可以生成中国移动格式(采用 QR Code 编码)的二维码图片，供手机直接扫描使用。

例如你可以在你的名片上加上二维码图片，这样用户直接扫描一下，即可快速准确地将你的名片信息读取到手机的通讯录中，你还可以在宣传资料中加入二维码，用户只需扫描一下，即可将预先设置的短信内容发送到指定的号码，同样也可以将网址编码成一个二维码，让用户扫描一下即可访问你的网址，无需手工输入网址。

目前大部分手机(特别是智能手机)都具备 QRCode 扫描功能，如果你的手机还没有，可以到以下网址下载：

http://www.quickmark.com.tw/

下图是利用 Foxtable 生成中国移动格式二维码图片的过程：



3.13 Excel 报表

3.13.1 关于 Excel 报表

Excel 报表能够以现成的 Excel 文件作为模板，通过设置一些特殊的设置，自动从 Foxtable 的数据表中提取数据填入到 Excel 表格中，这样我们就可以利用 Excel 强大的制表和格式设置功能， 轻松设计出复杂的报表：



Excel 报表使用起来很轻松，效率远远超过传统的可视化报表设计器，而且无需任何专业知识，非常适合普通用户掌握应用。

Foxtable 还提供了 Word 报表功能，同样可以直接利用 Word 可视化设置报表模板，自动根据模板的设置从 Foxtable 提取数据填入到 Word 中，批量生成报表或文章。

Excel 报表和 Word 报表各有优势，两者结合使用，基本上能满足绝大多数场合的需要，大家可以根据需要选用。

为了照顾更专业用户的需求，Foxtable 还提供了专业报表功能，专业报表的功能更为强大，使用也相对复杂一些，所以专业报表的内容将完全在开发篇讲述。

对于一般用户来说，掌握 Excel 报表和 Word 报表足以应付日常工作的需要，无需再去研究专业报表。

我们接下来首先介绍 Excel 报表，然后是 Word 报表，至于专业报表，则在开发指南讲述。

3.13.2 本章示例文件

本章课程使用的示例文件为“Excel 报表.Table”，位于安装目录的子目录“CaseStudy”下。

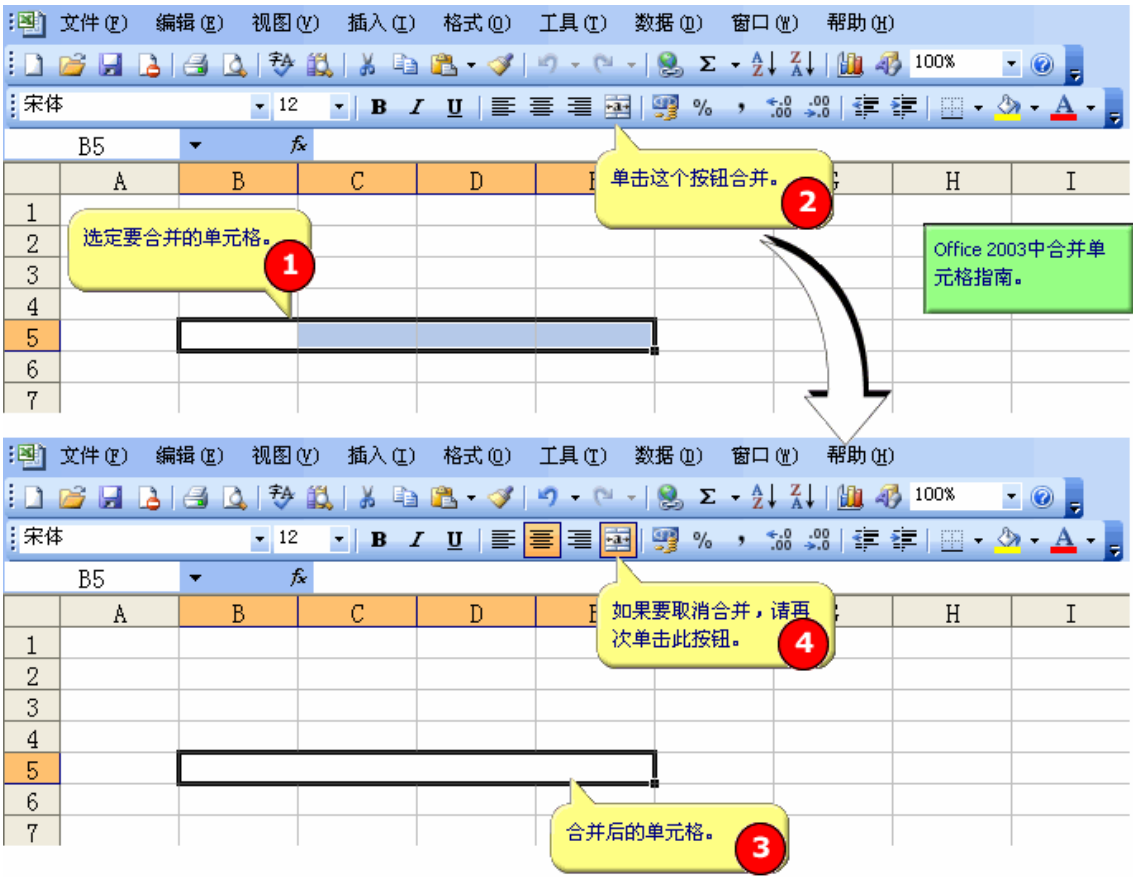
请在学习本章接下来的内容之前，先打开此文件。

需要注意的是，你的电脑必须安装有 Excel，才可以正常使用 Excel 报表功能。

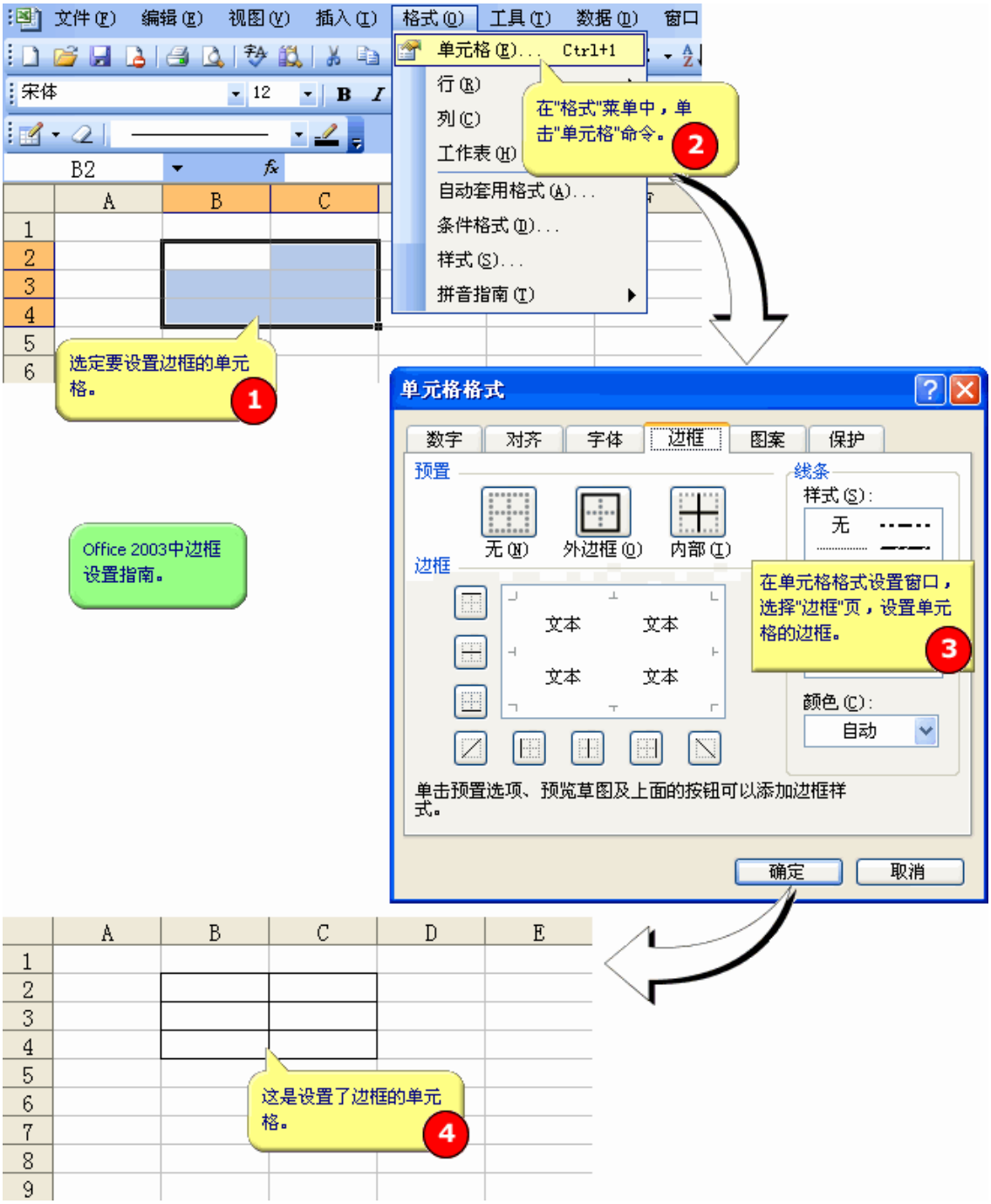
3.13.3 Excel 制表快速入门

本节介绍如何利用 Excel 来制表，只是针对从来没有用过 Excel 制表的用户。
如果你熟悉 Excel，请跳过此节。

- 合并单元格



- 设置边框



Excel 2007 的操作类似，只是菜单位置有所变化。

3.13.4 细节区和数据引用

Excel 报表模板的第一列用于定义细节区，细节区的定义格式为：

<表名>

在细节区，可以引用表中的数据，引用数据的格式为：

[列名]

例如：

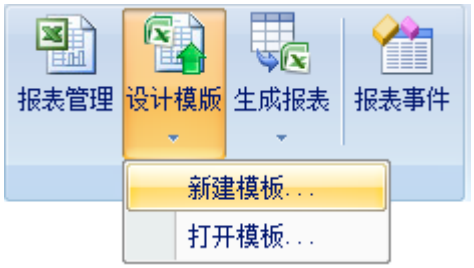


对于每一个数据行，都会重复生成一个细节区，例如上图的模板，可以生成下图所示的报表：

	A	B	C	D	E	F	G	H
1			订单					
2								
3	产品	客户	雇员	单价	折扣	数量	金额	日期
4	PD05	CS03	EP04	17	0.1	650	9945	36162
5	PD02	CS01	EP01	20	0.1	400	7200	36163
6	PD05	CS04	EP02	17	0	320	5440	36163
7	PD01	CS03	EP04	18	0.15	80	1224	36164
8	PD03	CS03	EP04	10	0	490	4900	36164
9	PD04	CS04	EP05	17.6	0	240	4224	36167
10	PD05	CS03	EP04	17	0	160	2720	36167
11	PD01	CS04	EP05	14.4	0	200	2880	36168
12	PD03	CS02	EP02	8	0.1	200	1440	36168
13	PD01	CS02	EP01	18	0.2	800	11520	36170
14	PD01	CS04	EP02	14.4	0.05	500	6840	36170
15	PD03	CS01	EP04	8	0	300	2400	36171
16	PD04	CS05	EP04	22	0.05	500	10450	36171
17	PD04	CS03	EP04	22	0	30	660	36172
18	PD02	CS05	EP04	19	0.1	200	3420	36173
19	PD01	CS03	EP04	14.4	0.25	200	2160	36174
20	PD02	CS01	EP04	15.2	0	100	1520	36174
21	PD02	CS05	EP04	15	0	60	900	36174

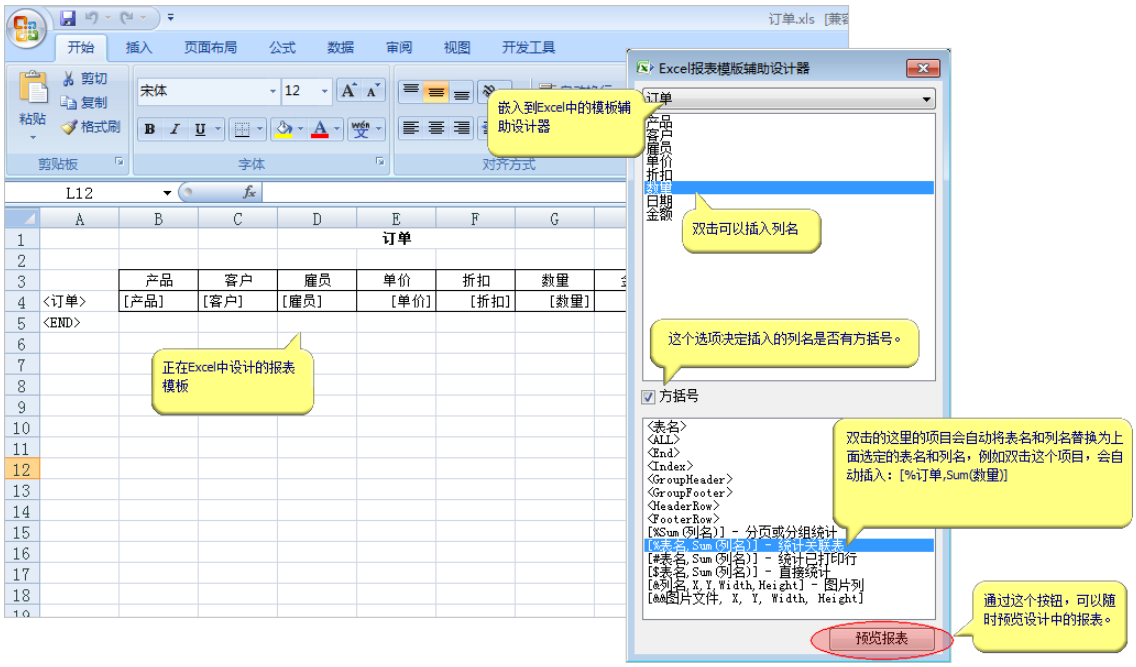
3.13.5 设计报表模板

在菜单的“打印输出”功能区的“Excel 报表”功能组，单击“设计模板”，可以新建报表模板，也可以打开现有模板修改：



所有的模版默认保存在项目所在目录的 Attachments 子目录中，为方便管理，最好不要改变这个存储目录。

模版设计器会启动 Excel，并在 Excel 中附加一个下图所示的辅助设计器，通过这个设计器，可以快速向 Excel 的活动单元格插入列名和各种标记，还可以随时预览设计中的报表：



有了这个辅助设计器，Excel 报表模板的设计将变得非常轻松，效率会提高 10 倍以上，准确度也会大大提高。

当然你也可以直接启动 Excel 来设计模版，将设计好的模板保存在 Attachments 目录就行，只是没有了辅助设计器，设计效率会大打折扣。

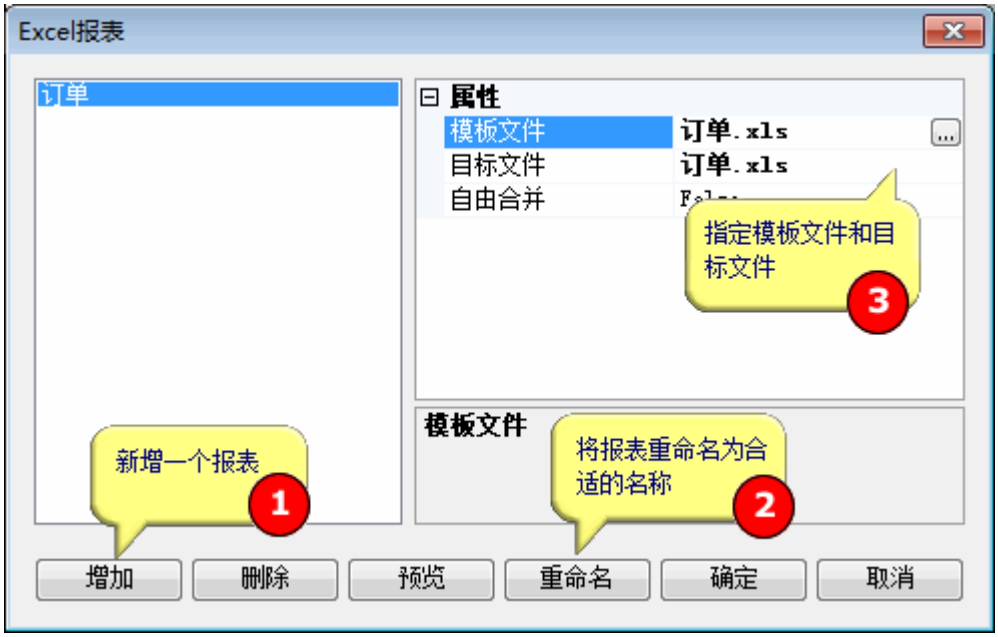
3.13.6 添加报表模板

设计好报表模版之后，我们需要将报表模版添加到菜单中。

在菜单的“打印输出”功能区的“Excel 报表”功能组，单击“报表管理”按钮：



会出现下图所示的窗口，通过这个窗口，可以添加已经设计好的报表模版：

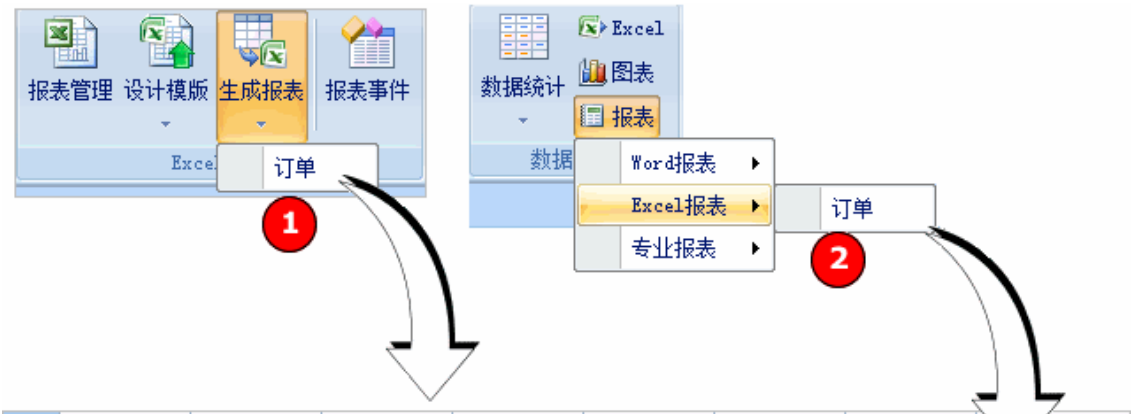


目标文件指的是根据模板文件生成的报表文件的文件名。

3.13.7 生成 Excel 报表

设计好 Excel 报表模版，并将模版添加到菜单后，就可以通过菜单来生成 Excel 报表了。
菜单中有两处地方可以生成 Excel 报表：
1、在菜单的“打印输出”功能区的“Excel 报表”功能组，单击“生成报表”按钮 ，会列出已经添加的模版，单击其中一个，即可生成报表。

2、在菜单的“日常工作”功能区的“数据统计”功能组，单击“报表”按钮，指向“Excel 报表”,也会列出已经添加的模版，单击其中一个，即可生成报表。



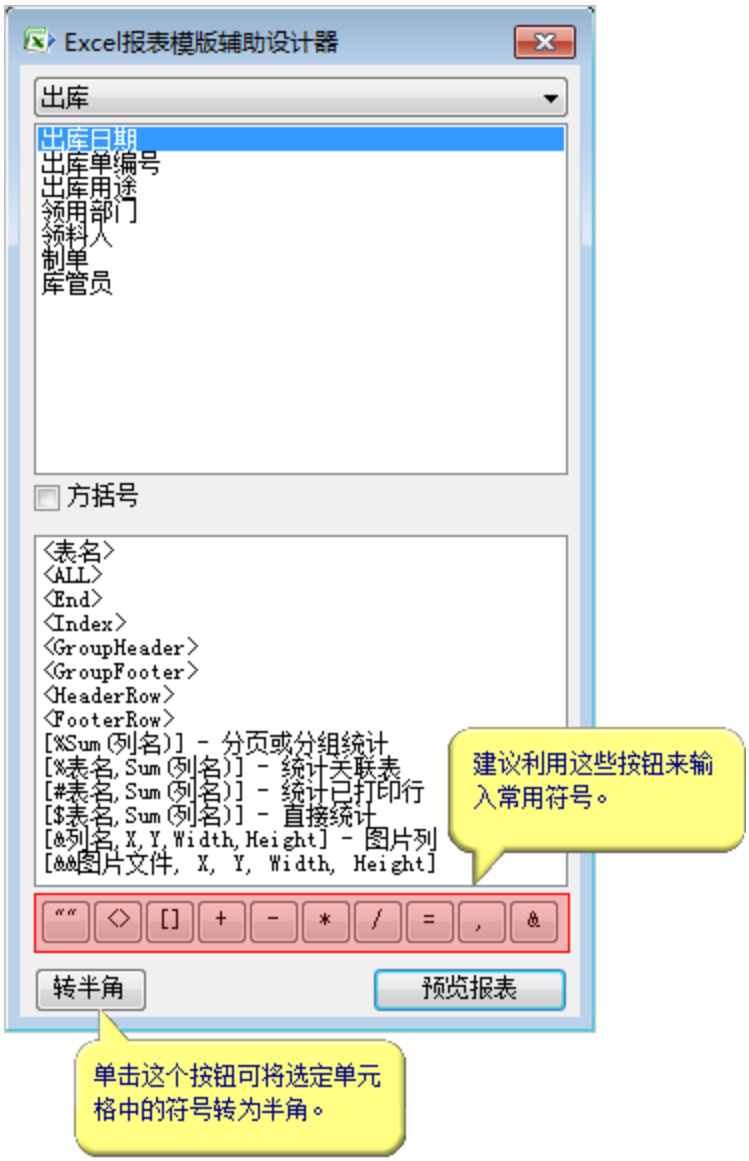
	A	B	C	D	E	F	G	H
1				订单				
2								
3	产品	客户	雇员	单价	折扣	数量	金额	日期
4	PD05	CS03	EP04	17	0.1	650	9945	1999/1/2
5	PD02	CS01	EP01	20	0.1	400	7200	1999/1/3
6	PD05	CS04	EP02	17	0	320	5440	1999/1/3
7	PD01	CS03	EP04	18	0.15	80	1224	1999/1/4
8	PD03	CS03	EP04	10	0	490	4900	1999/1/4
9	PD04	CS04	EP05	17.6	0	240	4224	2012/8/11
10	PD05	CS03	EP04	17	0	160	2720	2012/8/11
11	PD01	CS04	EP05	14.4	0	200	2880	1999/1/8
12	PD03	CS02	EP02	8	0.1	200	1440	1999/1/8
13	PD01	CS02	EP01	18	0.2	800	11520	1999/1/10
14	PD01	CS04	EP02	14.4	0.05	500	6840	2012/8/11
15	PD03	CS01	EP04	8	0	300	2400	2012/8/11
16	PD04	CS05	EP04	22	0.05	500	10450	1999/1/11
17	PD04	CS03	EP04	22	0	30	660	1999/1/12
18	PD02	CS05	EP04	19	0.1	200	3420	1999/1/13

系统会自动打开生成的报表文件，这些文件默认存放在目录目录的 Reports 子目录下。
小提示： 模版和表是有对应关系的，例如你在表 A 添加的报表模版，必须先选择表 A，才能生成报表。

3.13.8 符号要半角的

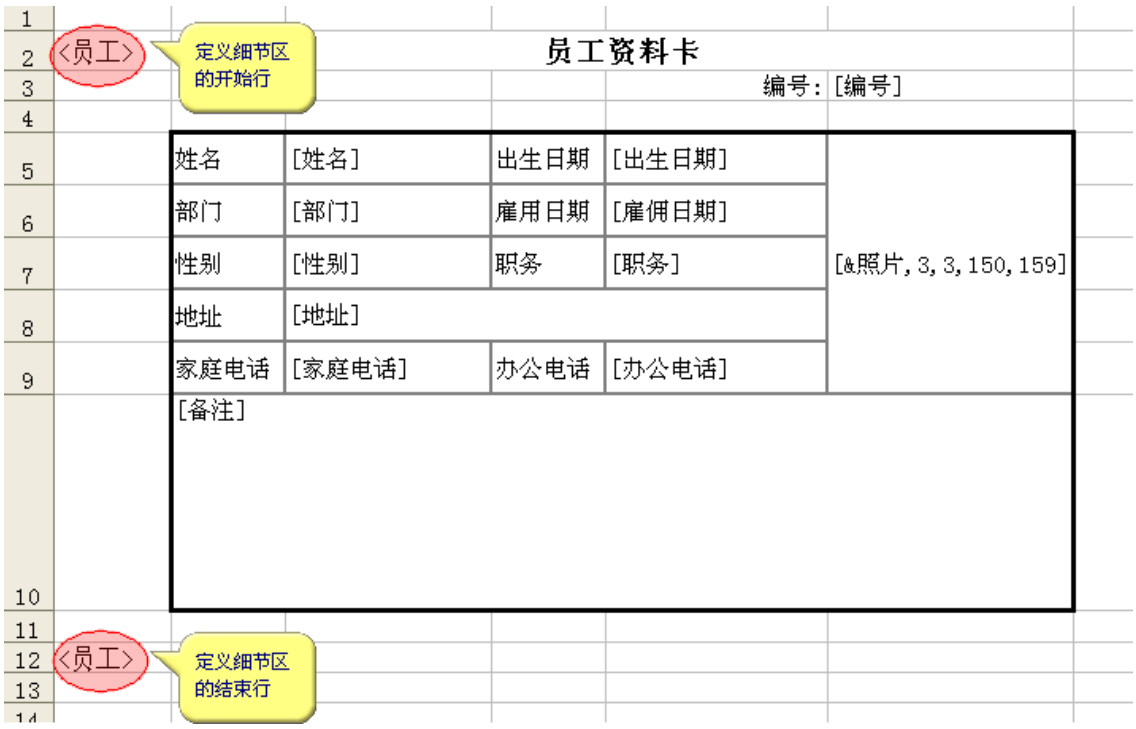
在设计 Excel 模板的时候，我们需要插入很多数据引用标记和表达式。

需要注意的是，标记中使用的符号必须是半角的，不能是全角的。
例如下面的表达式不会正确计算：
〈CLNum([金额])〉
正确的写法是：
<CLNum([金额])>
二者的差别在于前者首尾的尖括号时全角的，后者则是半角的。
使用汉字的我们，很难察觉这种错误，也很难避免这种错误，设计模板的一定要小心。
为了减少这种错误，Excel 模板辅助设计器后来加上了两个按钮，分别用于快速输入常用的半角符号 ，以及将选定单元格中的符号转换为半角：

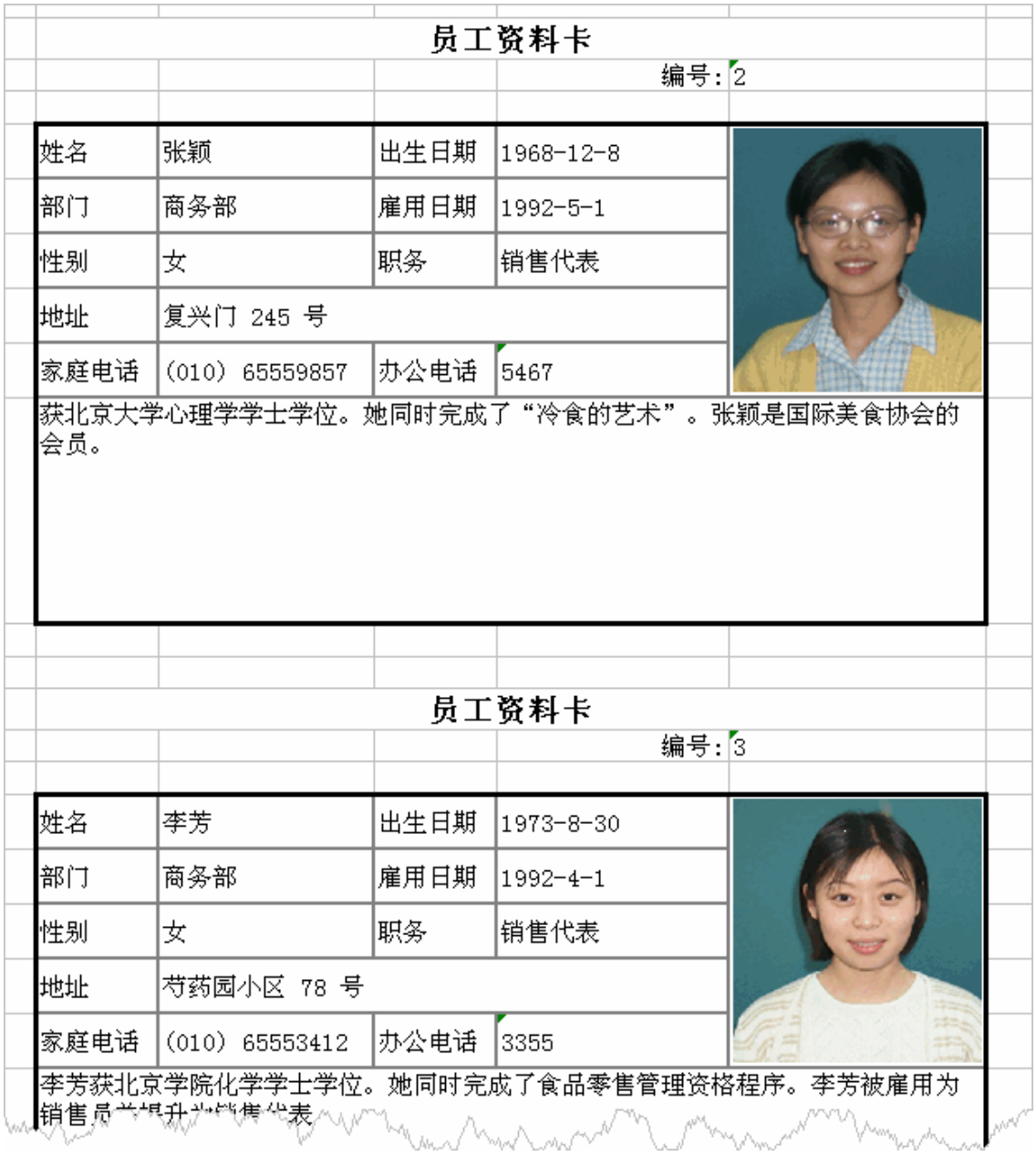


3.13.9 多行细节区

Excel 报表的细节区可以包括多行，而且可以任意复杂。
例如我们要 Excel 报表打印员工资料卡，可以按照下图所示设计模板：


员工资料卡
编号：[编号]
<员工>
定义细节区的开始行
姓名
[姓名]
出生日期
[出生日期]
部门
[部门]
雇用日期
[雇佣日期]
性别
[性别]
职务
[职务]
[&照片,3,3,150,159]
地址
[地址]
家庭电话
[家庭电话]
办公电话
[办公电话]
[备注]
<员工>
定义细节区的结束行

打印效果



员工资料卡

编号：2

姓名	张颖	出生日期	1968-12-8
部门	商务部	雇用日期	1992-5-1
性别	女	职务	销售代表
地址	复兴门 245 号		
家庭电话	(010) 65559857	办公电话	5467

获北京大学心理学学士学位。她同时完成了“冷食的艺术”。张颖是国际美食协会的会员。

员工资料卡

编号：3

姓名	李芳	出生日期	1973-8-30
部门	商务部	雇用日期	1992-4-1
性别	女	职务	销售代表
地址	芍药园小区 78 号		
家庭电话	(010) 65553412	办公电话	3355

李芳获北京学院化学学士学位。她同时完成了食品零售管理资格程序。李芳被雇用为销售员并提升为销售代表

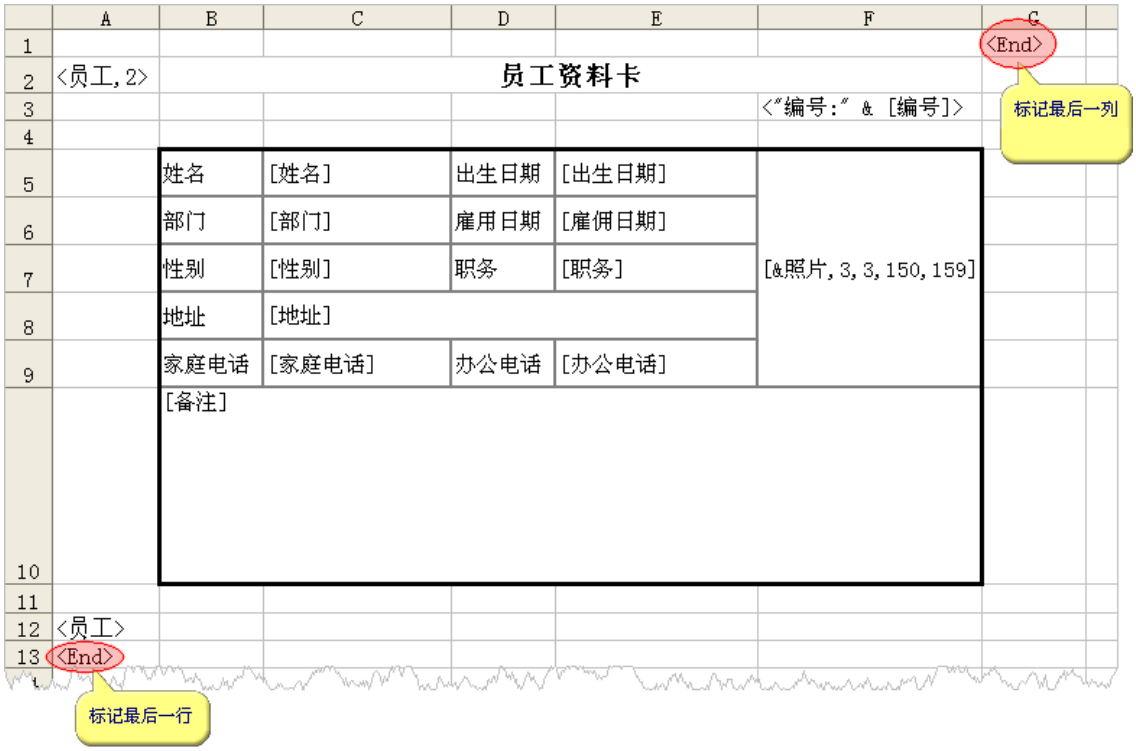
3.13.10 指定有效区域

在 Excel 报表的测试阶段，我们总是遇到一些用户反馈：在生成 Excel 报表的过程中死机。

其实并不是死机，而是他们在设计 Excel 报表模板的过程中，无意中设计了一个超大的表格，有的甚至达到 Excel 2007 的上限：16384 列。

为了避免这种意外的发生，你可以用<End>标记指定报表的有效区域。

在第一行的最后一列，以及第一列的最后一行，分别输入<End>标记即可：

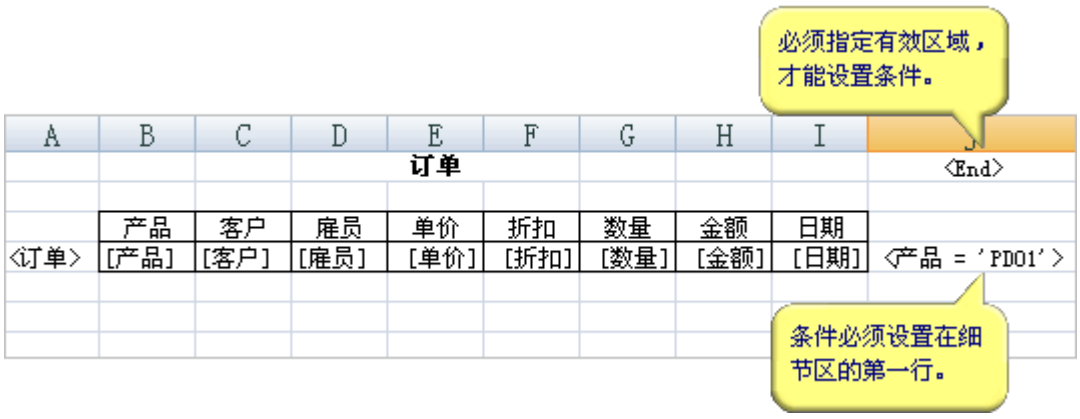


生成报表的时候，<End>标记之外的行列，包括<End>标记所在的行列，将会被忽略。

3.13.11 设置打印范围

需要注意的是，Excel 报表默认只会打印选定行(记录)，如果你只选定 1 行，就会打印 1 行，如果你选定了 10 行，就会打印 10 行。

可以设置一个条件表达式，例如只打印产品为 PD01 的行：



请参考表达式的 运算符和函数 以及 条件表达式

特别强调一下：必须设置有效区域，打印条件才会生效。

如果要打印所有行，可以按下图所示设置：

A	B	C	D	E	F	G	H	I	J
				订单					<End>
	产品	客户	雇员	单价	折扣	数量	金额	日期	
<订单>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>

ALL表示打印所有行。

注意这个 ALL 不是一个条件表达式，只是告诉系统：没有条件，全部打印。

3.13.12 关于日期列

也许你已经留意到，前面打印出来的报表，日期列的内容全部变成了数字：

订单

产品	客户	雇员	单价	折扣	数量	金额	日期
PD05	CS04	EP02	17	0	320	5440	36163
PD01	CS03	EP04	18	0.15	80	1224	36164
PD03	CS03	EP04	10	0	490	4900	36164
PD04	CS04	EP05	17.6	0	240	4224	36167
PD05	CS03	EP04	17	0	160	2720	36167
PD01	CS04	EP05	14.4	0	200	2880	36168
PD03	CS02	EP02	8	0.1	200	14	3
PD01	CS02	EP01	18	0.2	800	115	0

日期变成了数字

为避免这种情况的发生，在设计报表模板的时候，右击日期单元格，从快捷菜单中执行“单元格格式”命令，然后选择合适的日期格式即可：



3.13.13 换页控制

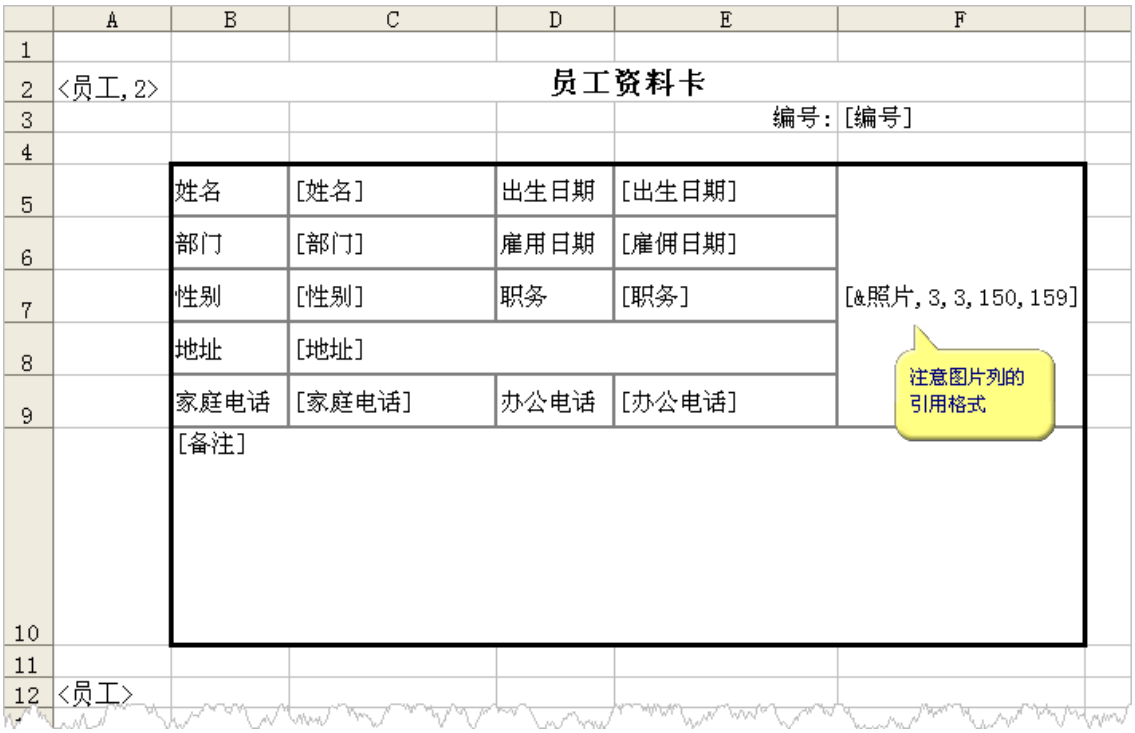
前一节已经讲述如何利用多行细节区来打印员工资料卡。
由于一个页面可以打印多个资料卡，这样同一个资料卡可能跨越前后两个页面，显然这是我们不希望看到的。
我们可以在定义细节区开始行的时候，增加一个参数，用于指定单个页面可打印细节区的个数，格式为：
<表名,n>
一旦某一页面的细节区数量达到参数 n 的设定值，将自动插入一个换页符。
例如：



在上图中，我们指定每页打印资料卡 2 个，这样就不会出现资料卡因为换页而一分为二的情况。

3.13.14 引用图片列

也许你已经留意到，前面的员工资料卡，正确地打印出了每个员工的照片。
引用图片列的格式为：
[&列名,X,Y,Width,Height]
X: 左边距
Y： 上边距
Width： 图片宽度
Height：图片高度
上述参数的单位为像素。
例如：



还有另外两种简化引用格式：

[&列名]

或

[&列名,X,Y]

我们知道，图片列的一个单元格可能包括多个图片，上述的引用格式打印的是第一个图片。

我们可以指定打印第几个图片，方法是在列名后面加上“&X”，X 是一个整数，表示要打印的图片的位置，从 0 开始编号。

例如：

[&照片&1,3,3,150,159]

表示打印第二个图片。

再例如打印第三个图片：

[&照片&2,3,3,150,159]

3.13.15 按比例缩放图片

我们已经知道，引用图片列的格式为：

[&列名,X,Y,Width,Height]

X: 左边距

Y： 上边距

Width： 图片宽度

Height：图片高度

有的时候，我们可能不需要同时指定高度或宽度，而是指定其中一个值，另一个值按比例生成。

例如：

[&照片,3,3,120,-1]

表示宽度固定为 120，高度按比例生成，如果图片原本的宽度和高度分别为 180 和 150，那么打印出来的图片，宽度是 120，高度是 100。

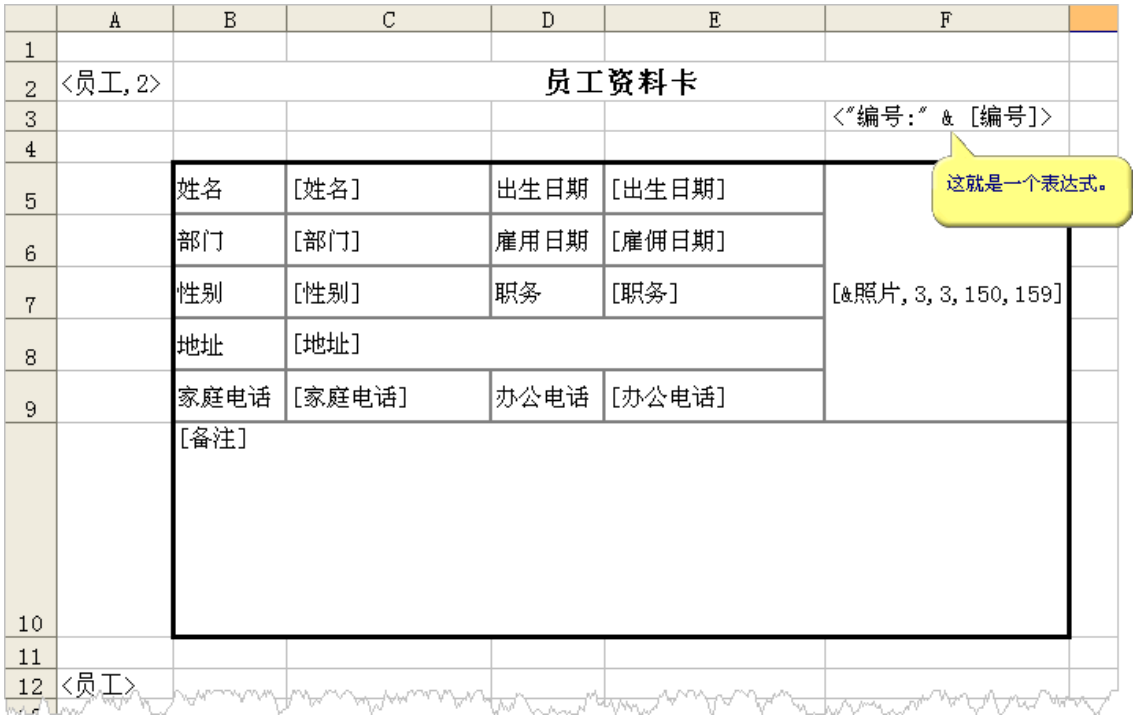
再例如：
[&照片,3,3,-1,100]
表示高度固定为 100，宽度按比例生成，如果图片原本的宽度和高度分别为 300 和 200，那么打印出来的图片，宽度是 150，高度是 100。

3.13.16 使用表达式

可以在 Excel 报表中直接进行表达式计算，表达式必须用符号"<"和">"括起来，例如：
<[数量] * [单价]>

<[语文] + [英语] + [数学]>

<"编号:" & [编号]>
员工资料卡中的编号，就是利用表达式来打印的：



表达式中的列名必须用方括号括起来，例如：<"编号：" & [编号]>，如果没有方括号：<"编号：" & 编号>，表达式将不能计算。

表达式采用 VBScript 语法，而且 Foxtable 对 VBScript 进行了扩展，提供了很多非常实用的函数。例如你要将出生日期以中文大写的形式打印出来，可以这样设置模板：<CUDATE([出生日期])>
再例如，假定已付款是个逻辑列，希望打印的时候，已付款的用符号"√"，可以这样设置模板：IIF([已付款], "√", "")
下面介绍一些常用的函数，更多的函数和更具体的用法，请参考：VBSCript 参考

函数名	说明
Format	格式化日期或数字。

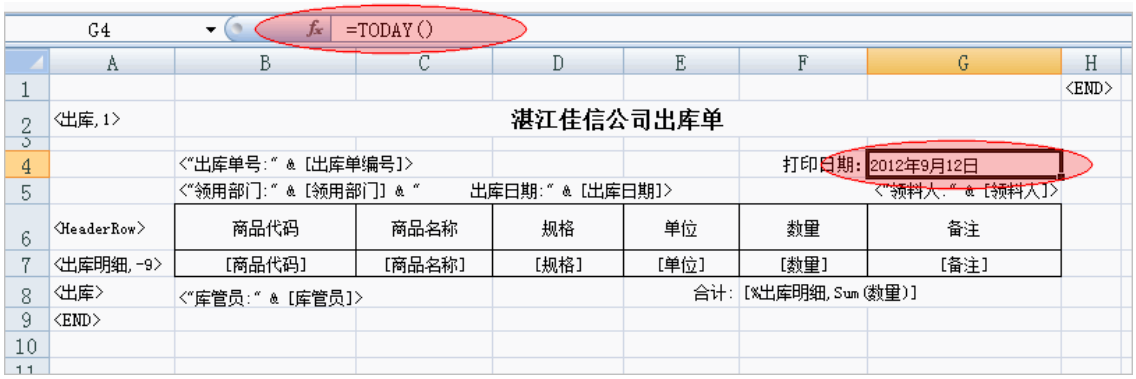
	格式化数字的时候，用 0 表示占位符，如果指定位置有数字，则显示数字，如果没有数字，则用 0 补充。 例如需要金额保留两位小数显示： <Format([金额],"#0.00")> 如果金额的值是 123.567，则显示为 123.57，如果金额的值是 123.5，则显示为 123.50，而不是 123.5。 如果希望以百分比的形式显示，可以加上符号%，例如希望以百分比形式显示合格率，保留一位小数： <Format([合格率],"#0.0%")> 格式化日期的时候，yyyy 表示四位年，yy 表示两位年，MM 表两 2 位月,M 表示一位月，dd 表示两位日,d 表示一位日，例如： <Format([出库日期],"yyyy 年 MM 月 dd 日")> 显示的日期格式为：2013 年 09 月 08 日 <Format([出库日期],"yyyy 年 M 月 d 日")> 显示的日期格式为：2013 年 9 月 8 日 <Format([出库日期],"yy 年 M 月 d 日")> 显示的日期格式为：13 年 9 月 8 日 注意 M 是大写。
IIF	语法为： IIF(expression, truevalue, falsevalue) 如果逻辑表达式 expression 的计算结果为 True，将返回 truevalue，否则返回 falsevalue。 例如： IIF([已付款], " √ ", "")
Round	对数值进行四舍五入，例如：<Round([金额],2)>
Date	返回当前日期，例如：<Format(Date,"yyyy 年 MM 月 dd 日")> 日期可以直接和数字进行加减，例如： <Date - 2> 表示前天的日期 <Date + 1> 表示明天的日期 <Format(Date + 1,"yyyy 年 MM 月 dd 日")> 表示以四位年两位月两位日的格式显示明天的日期。
Now	放回当前时间，例如：<Now>
Year	返回日期的年份，例如: <Year([出生日期])>
Month	返回日期的月份，例如: <Month([出生日期])>
Day	返回日期的日部分，例如: <Day([出生日期])>
Weekday	用数字形式返回星期几，例如: <WeekDay([出生日期])>
WeekDayName	用名称形式返回星期几，例如: <WeekDayName([出生日期])>
CUDate	将日期 Date 转换为大写中文格式。

	例如： <CUDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：贰零壹叁年拾贰月玖日
CLDate	将日期 Date 转换为小写中文格式。 例如： <CLDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：二〇一三年十二月九日
CUMoney	将数值转换为中文货币格式。 例如： <CUMoney([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍圆整
CUNumber	将数值转换为中文大写格式。 例如： <CUNumber([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍
EUNumber	将整数转换为英文大写。 例如： <EUNumber([数量])> 假定数量的值是 12345，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five
EUMoney	将数值转换为英文货币格式。 例如： <EUMoney([金额])> 假定金额的值是 12345.45，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five Dollars And Forty Five Cents

3.13.17 设置单元格公式

Excel 本身有非常丰富的函数，我们在在设计报表模版的时候，使用 Excel 的函数和语法来设置单元格公式，基于模版生成的报表，会保留之前的单元格公式。

例如在设计出库单模版的时候，打印日期需要调用当天的日期，可以直接将这个单元格的公式设置为： =Today()



	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>				打印日期:	2012年9月12日	
5		<"领用部门:" & [领用部门] & "		出库日期:" & [出库日期]>			<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

Excel 报表功能是非常灵活的，你既可以使用表达式，也可以使用 Excel 原有的单元格公式，充分利用各自的优势。

3.13.18 打印关联表

Excel 报表的细节区可以嵌套，这样可以非常方便地打印关联表数据。

打开示例文件“Excel 报表.Table”，可以看到两个表，分别是出库和出库明细。

	出库日期	出库单编号	出库用途	领用部门	领料人	制单	库管员
1	2008-07-26	CK-20030726001	销售	销售部	林海	管理员	陶正东
2	2008-07-26	CK-20030726002	生产领用	商函广告公司	朱志欣	管理员	李政旭

出库表

出库明细表

	出库单编号	商品类别	商品代码	商品名称	规格	单位	数量	单价	金额	出库用途	备注
1	CK-20030726001	微机耗材	MC00101	打印纸	80列	箱	50	50.00	2,500.00	销售	
2	CK-20030726001	微机耗材	MC00102	打印纸	40列	箱	20	29.00	580.00	销售	
3	CK-20030726002	办公用品	BG00201	记事本	16开精装	本	8	12.00	96.00	生产领用	
4	CK-20030726002	业务用品	YW00101	特快详情单		张	1000	0.83	830.00	生产领用	

出库表和出库明细表通过出库单号建立了关联，其中出库表为父表，出库明细表为子表。

现在我们要针对出库表设计一个 Excel 报表，用于打印出库单。

启动 Excel，如下图所示设计好出库单模板，请留意是如何引用关联表数据的：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & "		出库日期:" & [出库日期]>			<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

在项目文件目录的“CaseStudy\Attachments”子目录下，有一个已经按上图所示设计好的 Excel 模板文件，名为“出库单.XLS”，可以直接 参考使用。

需要注意：

1、通常不会将两个出库单打印在同一张纸上，所以出库单细节区的首行定义为：<出库,1>

2、出库单的细节区有多行，所以不能省略第 8 行(结束行)的定义：<出库>

3、第 7 行的数据来自于出库明细表，所以在第 7 行加入一个嵌套的细节区定义：<出库明细>

根据上面设计好的模板，生成的出库单：

湛江佳信公司出库单

出库单号:CK-20030726001

领用部门:销售部　　出库日期:2008/7/26　　领料人:林海

商品代码	商品名称	规格	单位	数量	备注
HC00101	打印纸	80列	箱	50	
HC00102	打印纸	40列	箱	20	

库管员:陶正东　　合计：70

3.13.19 统计关联表

上一节以出库单为例，我们学会了如何打印关联表。

这次我们还是以出库单为例，看看如何在报表中累计出库单数据。

统计某列数据的格式为：

[%表名,统计表达式,统计条件]

统计表达式使用聚合函数，例如：Sum(数量)

统计条件是可选的，语法采用我们之前介绍的表达式，请参考表达式的运算符和函数 和条件表达式。

为了给出库单底部加上数量合计，我们按下图修改出库单模板文件：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & "　　出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

在这里对数量进行合计

这是打印结果：

	B	C	D	E	F	G
		湛江佳信公司出库单				
	出库单号:CK-20030726001					
	领用部门:销售部	出库日期:2008-7-26			领料人:林海	
	商品代码	商品名称	规格	单位	数量	备注
	HC00101	打印纸	80列	箱	50	
	HC00102	打印纸	40列	箱	20	
	库管员:陶正东			合计:	70	

3.13.20 打印多个关联表

Excel 报表中可以同时嵌入多个关联表细节区。
假定表 A 同时和表 B 和表 C 建立了关联，在设计报表的时候，可以在表 A 的细节区中，同时嵌入表 B 和表 C 的细节区。

	A
1	<表A>
2	
3	<表B>
4	
5	<表C>
6	
7	<表A>

关联表的细节区，也可以包括多行，例如下图中，作为关联表的表 C，其细节区就有四行:

	A
1	<表A>
2	
3	<表B>
4	
5	<表C>
6	
7	
8	<表C>
9	<表A>

3.13.21 自动添加空行

有的时候，我们不仅希望设置每页最多可打印的细节区数，而是希望在细节区数达不到设定值的时候，能够自动补充空行。

例如下面的出库单，希望每张出库单至少 9 行出库明细，在不够的情况下，自动补充空行：

湛江佳信公司出库单

出库单号:CK-20030726001

领用部门:销售部　　出库日期:2008-7-26　　领料人:林海

商品代码	商品名称	规格	单位	数量	备注
HC00101	打印纸	80列	箱	50	
HC00102	打印纸	40列	箱	20	

库管员:陶正东　　合计: 70

实现的方法很简单，只需在细节区后面增加一个参数，我们只需用负数来设置最多可打印的细节区数即可：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & "　　出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

必须用负数来设置每页最多可打印的细节区数，才能自动补空行。

3.13.22 设置打印顺序

在定义细节区时，可以加入排序参数，格式为：

<表名,排序列>

默认排序方式是升序，如果希望降序排序，格式为：

<表名,排序列 Desc>

对于主表，必须指定打印范围，排序参数才会发生作用，例如下图按日期顺序打印订购产品 PD01 的行：

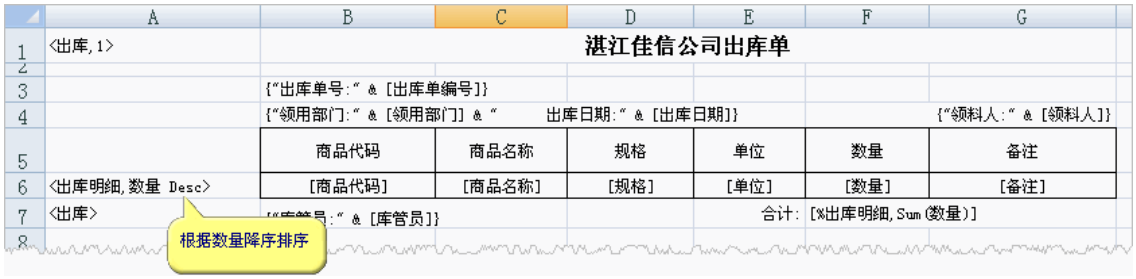
A	B	C	D	E	F	G	H	I	J
				订单					<End>
	产品	客户	雇员	单价	折扣	数量	金额	日期	
<订单,日期>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<产品 = 'PD01'>

按日期顺序打印

必须设置打印范围，排序参数才会发生作用

如果要打印所有行，将条件单元格的内容设为"<ALL>"即可。

对于关联表，不需要设置打印范围，排序参数也能发生作用：



还可以同时加入排序和细节区数量设置，例如：

<出库明细,数量,-9>

3.13.23 根据多列排序

打印顺序可以指定多个排序列，不同的排序列之间用符号"|"隔开。

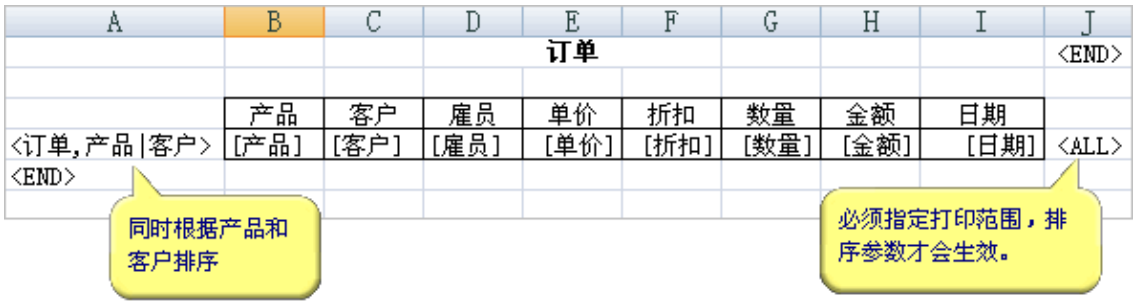
例如：

产品|客户

表示根据产品和客户列排序

产品|日期 Desc

表示根据产品和日期排序，其中日期排序方式为降序。



3.13.24 指定页首行

如果指定了每页最大可打印的细节区数，我们就可以将某一行或某几行设为页首行，这样每次换页的时候，将自动插入这几行作为页首使用。

指定页首行的格式为：

<HeaderRow>

需要注意的是，必须明确指定每页可打印的细节区数量，页首行的设置才会生效。

例如：



	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>				合计:	[%出库明细.Sum(数量)]	
9	<END>							
10								

如果页首行有多行，请分别在首行和尾行加上标记：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4	<HeaderRow>	<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>				合计:	[%出库明细.Sum(数量)]	
9	<END>							
10								

3.13.25 页尾行与分页汇总

如果指定了每页最大可打印的细节区数，我们就可以将某一行或某几行设为页尾行，这样每次换页的时候，将自动插入这几行作为页尾使用。

指定页尾行的格式为:

<FooterRow>

如果页尾行有多行，请分别在首行和尾行加上上述标记。

可以在页尾行对本页数据进行统计，格式为：

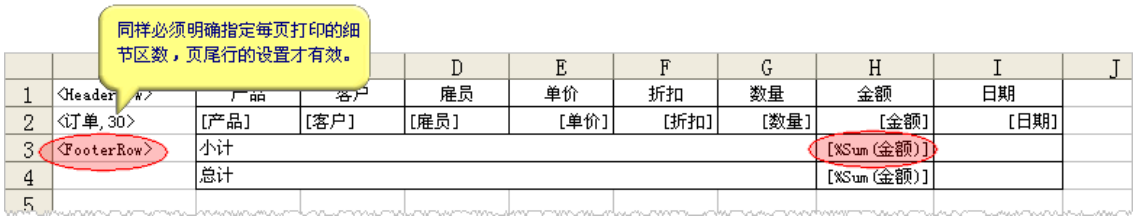
[%统计表达式]

统计表达式使用聚合函数，例如：Sum(数量)。

必须明确指定每页可打印的细节区数量，页尾行的设置才会生效。

示例

下面的模板用于打印订单表，在每页的结尾加上金额小计，最后加上合计:



				D	E	F	G	H	I	J
1	<Header...>	产品	客户	雇员	单价	折扣	数量	金额	日期	
2	<订单,30>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	
3	<FooterRow>	小计						[%Sum(金额)]		
4		总计						[%Sum(金额)]		
5										

打印结果：

产品	客户	雇员	单价	折扣	数量	金额	日期
PD05	CS03	EP04	17	0	300	5100	1999-6-27
PD05	CS01	EP02	17	0	20	340	1999-6-28
PD02	CS02	EP02	15.2	0	400	6080	1999-6-29
PD02	CS03	EP02	19	0	30	570	1999-6-29
PD01	CS02	EP01	18	0	300	5400	1999-6-30
PD01	CS05	EP03	18	0	60	1080	1999-6-30
小计						18570	
总计						825651.57	

3.13.26 汇总已经打印的行

如果需要统计已经打印行，可以采用以下格式：
[#表名,统计表达式]
统计表达式使用聚合函数，例如：Sum(数量)
假定我们不仅要汇总出每一页的小计，而且需要得截止到当前页为止的累计，可以按下图所示设计报表模板：

	A	B	C	D	E	F	G	H	I
1	<HeaderRow>	产品	客户	雇员	单价	折扣	数量	金额	日期
2	<订单,30>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]
3									
4	<FooterRow>	小计:	[%Sum(金额)]		当前累计:	[#订单,Sum(金额)]			
5									

打印效果：

PD03	CS04	EP05	19	0	100	1900	1999-2-21
PD02	CS05	EP03	19	0	100	1900	1999-2-21
PD03	CS05	EP02	10	0	300	3000	1999-2-21
PD05	CS02	EP02	17	0.05	160	2584	1999-2-21
PD01	CS02	EP05	14.4	0.15	150	1836	1999-2-22
PD01	CS04	EP02	14.4	0	400	5760	1999-2-23
PD01	CS05	EP03	14.4	0	180	2592	1999-2-23
PD02	CS03	EP02	19	0.2	150	2280	1999-2-23
PD05	CS02	EP03	21.35	0.05	240	4867.8	1999-2-24
PD05	CS02	EP02	17	0.2	200	2720	1999-2-25
小计:		107775.9	当前累计:	233794.4			

3.13.27 分组统计报表

Excel 报表也提供了分组统计功能。
定义分组头的格式为：
<GroupHeader,表名,列名>

或

<GroupHeader,表名,列名,1>

不同的是，后者在换页之后，会重复打印一次分组头。

定义分组尾的格式为：

<GroupFooter,表名,列名>

或者

<GroupFooter,表名,列名,1>

不同的是，后者打印完分组尾之后，会自动换页。

在分组头和分组尾可以引用数据，引用数据的格式为：

[列名]

在分组尾可以统计数据，统计数据格式为：

[%统计表达式]

统计表达式使用聚合函数，例如：Sum(数量)。

需要特别提示的是： 分组报表的细节区必须指定排序参数，且排序列必须为分组列。

示例

希望根据产品分组打印订单，并统计每个产品的销售数量和金额，并自动换页，模板可按下图所示设计：



这是打印效果：

PD03	CS01	EP01	10	0	240	2400	1999-02-[illegible]
PD03	CS05	EP03	10	0	200	2000	1999-02-10
PD03	CS05	EP02	10	0	300	3000	1999-02-21
PD03	CS03	EP02	10	0	300	3000	1999-02-26
小计PD03					1290	12900	
产品	客户	雇员	单价	折扣	数量	金额	日期
PD04	CS04	EP02	22	0.1	500	9900	1999-02-14
PD04	CS03	EP02	22	0	10	220	1999-02-26
小计PD04					510	10120	
产品	客户	雇员	单价	折扣	数量	金额	日期
PD05	CS02	EP03	17	0	150	2550	1999-02-01
PD05	CS04	EP04	21.35	0	700	14945	1999-02-01
PD05	CS04	EP04	21.35	0	10	213.5	1999-02-04
PD05	CS04	EP03	21.35	0	700	14945	1999-02-07
PD05	CS03	EP03	17	0	100	1700	1999-02-20
PD05	CS02	EP02	17	0.05	160	2584	1999-02-21
PD05	CS02	EP03	21.35	0.05	240	4867.8	1999-02-24
PD05	CS02	EP02	17	0.2	200	2720	1999-02-25
PD05	CS04	EP02	21.35	0	160	3416	1999-02-28
小计PD05					2420	47941.3	

可以同时使用分组头和分组尾，而且不管是分组头，还是分组尾，都可以是多行的，例如下图的分组头包括两行：

A	B	C	D	E	F	G	H	I
	订单							<END>
<GroupHeader,订单,产品>						<"产品："&[产品]>		
<GroupHeader,订单,产品>	客户	雇员	单价	折扣	数量	金额	日期	
<订单,产品>	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
<GroupFooter,订单,产品,1>	<"小计"&[产品]>				[%Sum(数量)]	[%Sum(金额)]		
<END>								

这是打印结果：



订单

产品：P001

客户	雇员	单价	折扣	数量	金额	日期
CS03	EP04	18	0.15	80	1224	1999-1-4
CS04	EP05	14.4	0	200	2880	1999-1-8
CS02	EP01	18	0.2	800	11520	1999-1-10
CS04	EP02	14.4	0.05	500	6840	1999-1-10
CS03	EP04	14.4	0.25	200	2160	1999-1-14
CS02	EP05	14.4	0	100	1440	1999-1-20
CS03	EP01	14.4	0	240	3456	1999-1-21
CS05	EP05	18	0	200	3600	1999-2-1
CS01	EP04	18	0	20	360	1999-2-7
CS01	EP02	18	0	80	1440	1999-2-7

很多时候，我们希望换页之后，能够重复打印一次分组头，此时可以将 GroupHeader 的第四个参数设为 1，并指定每页的细节区数。

例如：

A	B	C	D	E	F	G	H	I
	订单							<END>
<GroupHeader,订单,产品,1>						<"产品："&[产品]>		
<GroupHeader,订单,产品>	客户	雇员	单价	折扣	数量	金额	日期	
<订单,产品,30>	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
<GroupFooter,订单,产品,1>	<"小计"&[产品]>				[%Sum(数量)]	[%Sum(金额)]		
<END>								

不管是分组头还是分组尾，都可以多行，分别在第一行和最后一行加上对应的格式定义即可，例如下图中，分组头有三行，分组尾有四行：

	A
1	
2	
3	<GroupHeader,订单,产品>
4	
5	<GroupHeader,订单,产品>
6	<订单,产品>
7	<GroupFooter,订单,产品>
8	
9	
10	<GroupFooter,订单,产品,1>
11	<END>

3.13.28 多个分组列

上一节介绍的分组统计报表，都只有一个分组列。

其实 Excel 报表一样可以有多个分组列，例如下面的报表模板，有两个分组尾，分别是客户和产品，也就是根据产品和客户进行分组统计，并生成报表：

A	B	C	D	E	F	G	H	I	J
				订单					<END>
<HeaderRow>	产品	客户	雇员	单价	折扣	数量	金额	日期	
<订单,产品|客户>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
<GroupFooter,订单,客户>		<"小计" & [客户]>				[%Sum(数量)]	[%Sum(金额)]		
<GroupFooter,订单,产品,1>	<"小计" & [产品]>					[%Sum(数量)]	[%Sum(金额)]		
<END>									

需要特别注意的是，细节区的排序参数，必须加上所有的分组列，而且大分组列应该位于小分组列之前，例如上面的报表，产品列是一级分组，在产品相同的情况下，再根据客户分组，所以排序参数必须是“产品|客户”，而不能是“客户|产品”。

3.13.29 引用父表数据

引用父表数据的格式为：

[父表路径,列名]

例如有一个订单管理系统，其产品表通过产品编号和订单明细表建立关联，客户表通过客户编号和订单表建立关联，而订单表通过订单编号和订单明细表建立关联。

其中订单明细表的结构如下：

	订单编号	产品编号	单价	数量	折扣
1	10248	17	14	12	0
2	10248	42	9.8	10	0
3	10248	72	34.8	5	0
4	10249	14	18.6	9	0
5	10249	51	42.4	40	0
6	10250	41	7.7	10	0
7	10250	51	42.4	35	0.15
8	10250	65	16.8	15	0.15
9	10251	22	16.8	6	0.05
10	10251	57	15.6	15	0.05

现在希望通过 Excel 报表来打印订单明细表，为了便于阅读，希望能够包括产品名称和公司名称两列，为此我们可如下设置 Excel 报表模板：

	A	B	C	D	E	F	G
1		订单ID	产品	客户	单价	数量	折扣
2	<订单明细>	[订单ID]	[产品.产品名称]	[订单.客户.客户名称]	[单价]	[数量]	[折扣]
3							
4							

因为订单明细表和产品表之间有直接的关联，所以从产品表引用产品名称的方法很简单：

[产品,产品名称]

订单明细表没有直接和客户表建立关联，而是通订单表间接地关联到客户表，所以从客户表引用客户名称，就稍微复杂一点，需要给出完整的路径：

[订单.客户,客户名称]

这是打印结果：

	A	B	C	D	E	F
1	订单ID	产品	客户	单价	数量	折扣
2	10248	猪肉	山泰企业	14	12	0
3	10248	糙米	山泰企业	9.8	10	0
4	10248	酸奶酪	山泰企业	34.8	5	0
5	10249	沙茶	东帝望	18.6	9	0
6	10249	猪肉干	东帝望	42.4	40	0
7	10250	虾子	实翼	7.7	10	0
8	10250	猪肉干	实翼	42.4	35	0.15
9	10250	海苔酱	实翼	16.8	15	0.15
10	10251	糯米	千固	16.8	6	0.05
11	10251	小米	千固	15.6	15	0.05
12	10251	海苔酱	千固	16.8	20	0
13	[illegible]	桂花糕	[illegible]	[illegible]	10	[illegible]

如果被引用的是一个图片列，引用格式为：

[父表路径,&列名]

[父表路径,&列名,x,y]

[父表路径,&列名,x,y,width,height]
x:　左边距
y：　上边距
width：　图片宽度
height：图片高度

3.13.30 直接引用数据

前面我们已经提供了引用父表和子表数据的方法，但是有的时候，我们希望从一个毫无关系的表中引用数据。
这样的引用同样简单，格式为:
[@表名,列名]
或
[@表名,列名,行位置]
[@表名,列名,+行位置]
利用直接引用，我们可以在细节区之外引用表中数据。
行位置是可选的，如果省略，则引用的是表中选定行的数据。
如果在行位置之前加上符号“+”，则表示相对位置，也就是距离选定行的位置。
例如：
[@产品,名称]
[@产品,名称,+1]
[@产品,名称,+2]
[@产品,名称,9]

假定产品表选定行的位置为 3，那么上述标示分别表示：
第三行的名称列内容
第四行的名称列内容
第五行的名称列内容
第十行的名称列内容(行号从 0 开始编号，9 表示第十行)

直接引用对于关联表也有效，例如引用关联表“出库.出库明细”中第二行商品代码列的内容：
[@出库.出库明细,商品代码,+1]
如果直接引用的是图片列，可以按照以下格式：
[@表名,&列名,行位置]
[@表名,&列名,行位置,x,y]
[@表名,&列名,行位置,x,y,width,height]
或者
[@表名,&列名,+行位置]
[@表名,&列名,+行位置,x,y]
[@表名,&列名,+行位置,x,y,width,height]
x:　左边距
y:　上边距
width:　图片宽度
height:图片高度
需要注意的是，对于图片列，即使是当前行，也必须明确指定行位置，例如：
[@员工,&照片,+0]

3.13.31 直接统计

如果要直接统计某个表的数据，格式为：
[$表名,统计表达式]
或
[$表名,统计表达式,统计条件]
统计表达式使用聚合函数，例如：Sum(数量)
请参考表达式的 运算符和函数 以及 条件表达式
示例
例如要用 Excel 报表统计不同产品的销售数量和金额，可以按照下图所示设计模板：

A	B	C	D
	产品	数量	金额
	PD01	[$订单,Sum(数量),产品 = 'PD01']	[$订单,Sum(金额),产品 = 'PD01']
	PD02	[$订单,Sum(数量),产品 = 'PD02']	[$订单,Sum(金额),产品 = 'PD02']
	PD03	[$订单,Sum(数量),产品 = 'PD03']	[$订单,Sum(金额),产品 = 'PD03']
	PD04	[$订单,Sum(数量),产品 = 'PD04']	[$订单,Sum(金额),产品 = 'PD04']
	PD05	[$订单,Sum(数量),产品 = 'PD05']	[$订单,Sum(金额),产品 = 'PD05']

这是生成的报表：

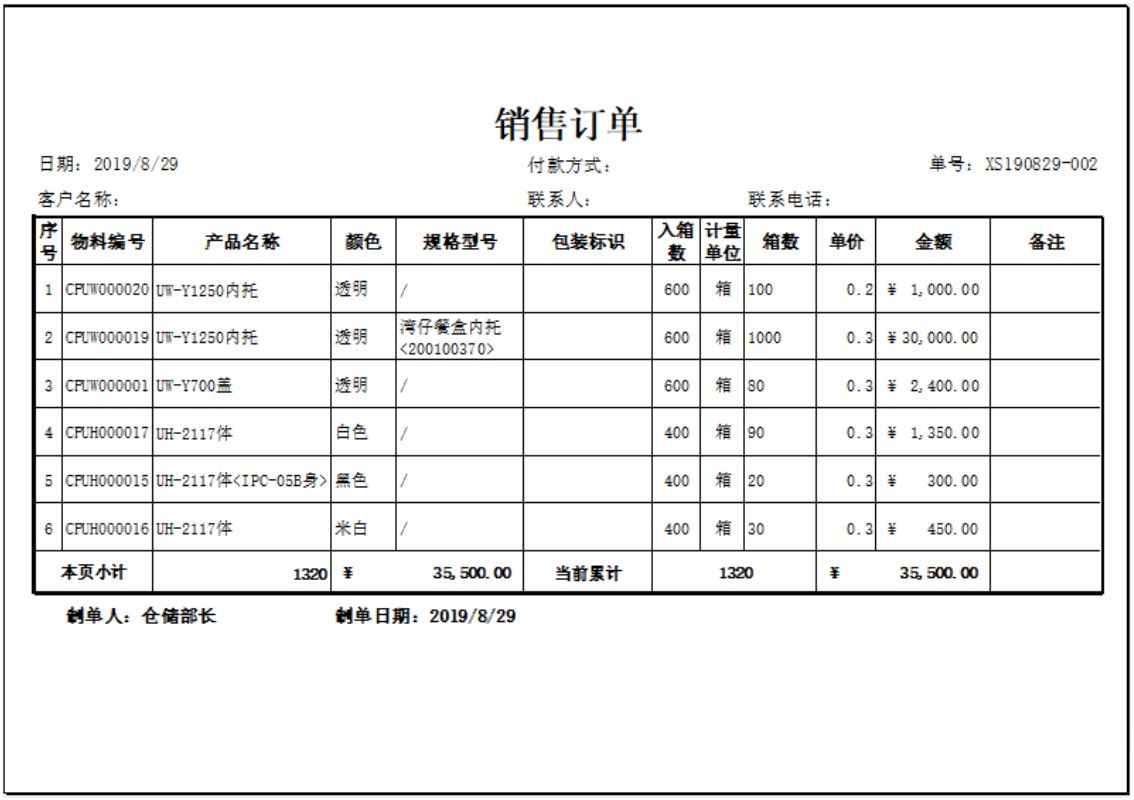
A	B	C
产品	数量	金额
PD01	11290	171754.92
PD02	18200	296522.5
PD03	7000	61490
PD04	5480	102861
PD05	10400	193023.15

3.13.32 引用顺序号

顺序号的初始值为 1，每打印一个细节区，顺序号就会增加 1。
可以在 Excel 报表中引用顺序号，引用格式为：
<Index>

3.13.33 又一个关联表打印示例

假定有两个表，分别是订单表和订单明细表，二者之间已经建立关联，希望通过 Excel 报表实现下图所示的订单打印功能：



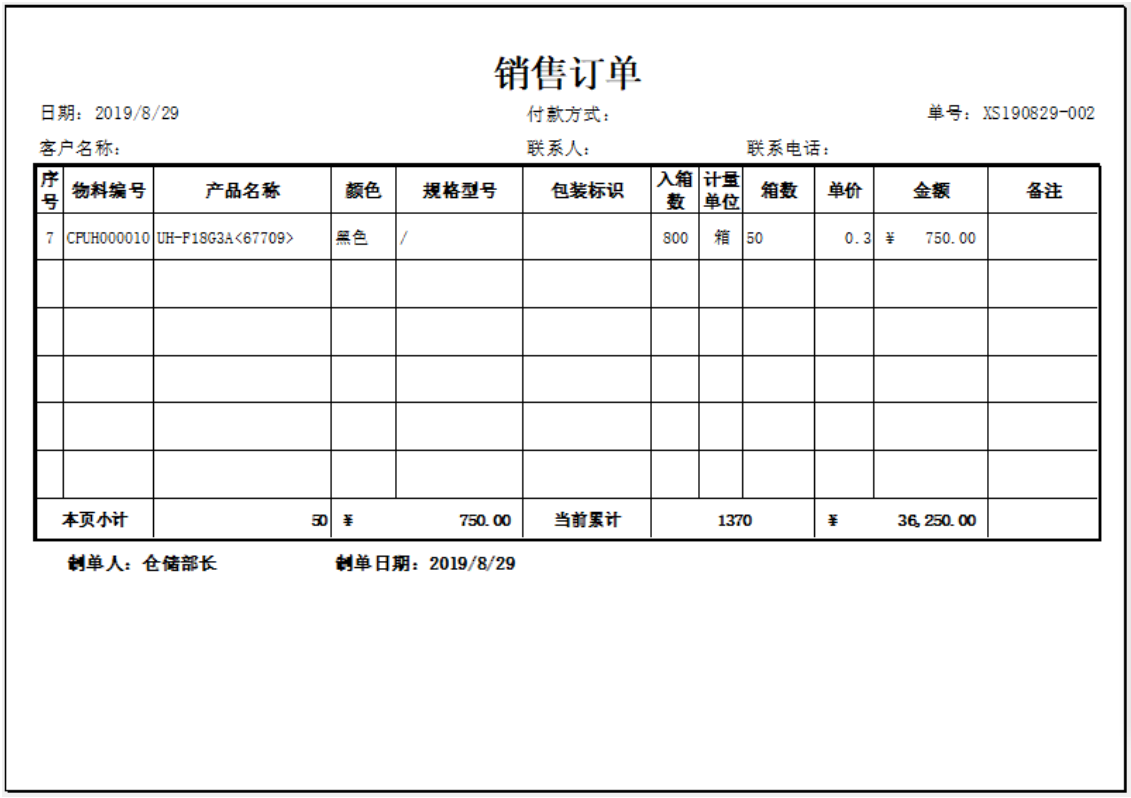
销售订单

日期：2019/8/29　　付款方式：　　单号：XS190829-002

客户名称：　　联系人：　　联系电话：

序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单位	箱数	单价	金额	备注
1	CPUW000020	UW-Y1250内托	透明	/		600	箱	100	0.2	¥ 1,000.00	
2	CPUW000019	UW-Y1250内托	透明	湾仔餐盒内托<200100370>		600	箱	1000	0.3	¥ 30,000.00	
3	CPUW000001	UW-Y700盖	透明	/		600	箱	80	0.3	¥ 2,400.00	
4	CPUH000017	UH-2117体	白色	/		400	箱	90	0.3	¥ 1,350.00	
5	CPUH000015	UH-2117体<IPC-05B身>	黑色	/		400	箱	20	0.3	¥ 300.00	
6	CPUH000016	UH-2117体	米白	/		400	箱	30	0.3	¥ 450.00	
本页小计		1320	¥	35,500.00	当前累计	1320			¥	35,500.00	

制单人：仓储部长　　制单日期：2019/8/29



销售订单

日期：2019/8/29　　付款方式：　　单号：XS190829-002

客户名称：　　联系人：　　联系电话：

序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单位	箱数	单价	金额	备注
7	CPUH000010	UH-F18G3A<67709>	黑色	/		800	箱	50	0.3	¥ 750.00	
本页小计		50	¥	750.00	当前累计	1370			¥	36,250.00	

制单人：仓储部长　　制单日期：2019/8/29

这个报表比之前的关联表打印稍微复杂一点，页头和也尾都有多行，有分页汇总和总计，但设计的原则是一样的：

首尾是父表细节区定义，中间是子表细节区定义，HeaderRow 和 FooterRow 加在需要的地方。

把握了这个原则，报表模板设计起来一样的轻松：



	A	C	D	E	F	G	H	I	J	K	L	M	N	P
1														<End>
2	<订单,1>													
3	<HeaderRow>		销售订单											
4			<"日期："&[日期]>				<"付款方式："&[付款方式]>					<"单号："&[单号]>		
5			<"客户名称："&[客户名称]>				<"联系人："&[联系人]>			<"联系电话："&[联系电话]>				
6	<HeaderRow>		序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单	箱数	单价	金额	备注
7	<订单明细,-6>	Inde	[物料编号]	[产品名称]	[颜色]	[规格型号]	[包装标识]	入箱数	量单	[箱数]	[单价]	[金额]	[备注]	
8	<FooterRow>		本页小计	[%Sum(箱数)]		[%Sum(金额)]	当前累计	订单明细,Sum(箱数		订单明细,Sum(箱数		订单明细,Sum(金额)		
9	<FooterRow>		<"制单人："&[制单人]>			<"制单日期："&[制单日期]>								
10	<订单>													
11	<End>													

隐藏无需打印的标记行

实际预览或打印报表的时候，你会发现上述模板有问题：第一页的顶部页边距明显大过第二页。本节头两个图片也说明了这个问题。

导致这个问题的出现，是因为 Excel 报表需要一些没有实质内容的标记行，例如上述模板中的第 1、2、10、11 四行，而这些行也会参与打印。正式因为模板中第 1 行和第 2 行，导致了第一页的顶部页边距比其他页面大，所以解决这个问题很简单，模板设计完成之后，隐藏无需参与打印的标记行：

	A	C	D	E	F	G	H	I	J	K	L	M	N	P
3	<HeaderRow>		销售订单											
4			<"日期："&[日期]>				<"付款方式："&[付款方式]>					<"单号："&[单号]>		
5			<"客户名称："&[客户名称]>				<"联系人："&[联系人]>			<"联系电话："&[联系电话]>				
6	<HeaderRow>		序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	重单	箱数	单价	金额	备注
7	<订单明细,-6>	Inde	[物料编号]		[产品名称]	[颜色]	[规格型号]	[包装标识]	入箱数	量单	[箱数]	[单价]	[金额]	[备注]
8	<FooterRow>		本页小计		[%Sum(箱数)]		[%Sum(金额)]	当前累计	订单明细.Sum(箱数		订单明细.Sum(金额)			
9	<FooterRow>		<"制单人："&[制单人]>			<"制单日期："&[制单日期]>								
12														

重要提示：

不要在设计过程隐藏标记行，应该等报表设计完全达到要求之后再隐藏。

3.13.34 打印工资条

有了 Excel 报表，打印工资条是非常轻松的了。

我们只需设计一个这样简单的模板：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	<工资,7>	单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
2		[单位]	[姓名]	[职务工资]	[级别工资]	[基础工资]	[工龄工资]	[奖金津贴]	[教龄津贴]	[教护工资]	[特殊津贴]	[物价补贴]	[技术津贴]	[山区津贴]
3		适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
4		[适当补贴]	[保健津贴]	[从优待警]	[警衔津贴]	[补差工资]	[补发工资]	[年终奖金]	[应发工资]	[扣所得税]	[养老保险]	[扣发合计]	[实发工资]	
5	<工资>													

然后将这个模板加入到工资表的 Excel 报表管理器中即可，就是这么简单，谁都可以完成。

上图设置每页打印 7 个工资条，具体请根据需要设定。

3.13.35 标签打印

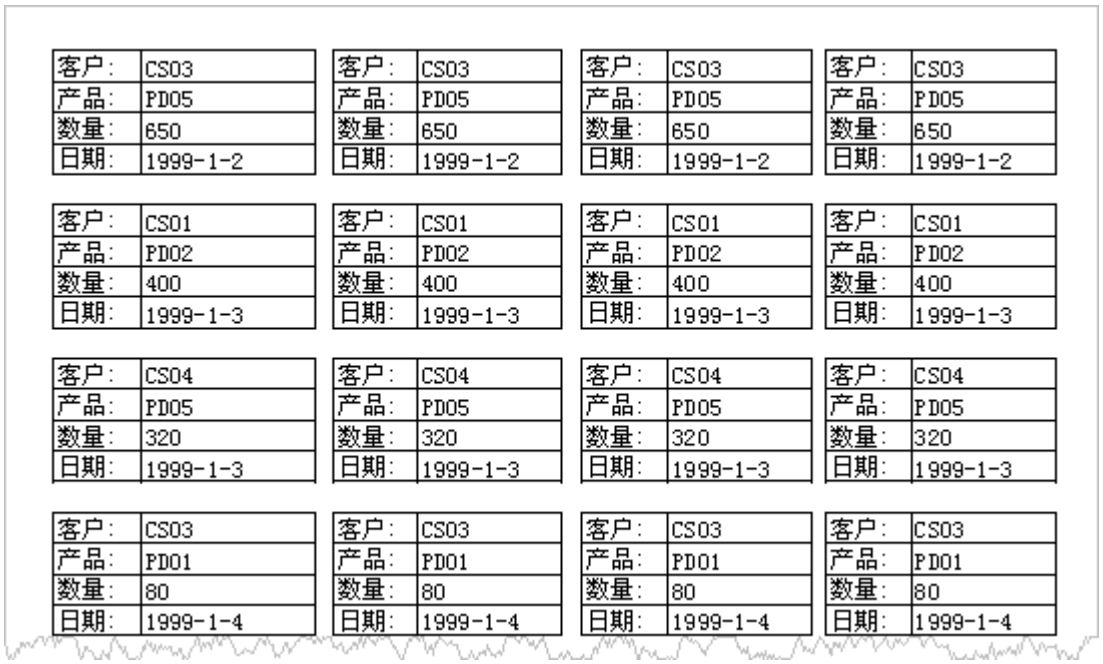
如果同一细节区包括多个标签，而且这些标签基于同一个数据行，那么模板设计很简单。

例如每一个订单打印四个标签，可以按下图所示设计模板：

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2	<订单,11>	客户:	[客户]		客户:	[客户]		客户:	[客户]		客户:	[客户]
3		产品:	[产品]		产品:	[产品]		产品:	[产品]		产品:	[产品]
4		数量:	[数量]		数量:	[数量]		数量:	[数量]		数量:	[数量]
5		日期:	[日期]		日期:	[日期]		日期:	[日期]		日期:	[日期]
6	<订单>											
7												

注意为了避免换页导致标签被分割，应该设置每页可打印的细节区数，上面设置的是 11。

标签打印结果：



客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

可是很多时候，同一个细节区的多个标签，可能来自不同的数据行；例如一个细节区包括四个标签，每个标签来自于不同的数据行。

我们知道，每一个细节区针对一个数据行，之前的引用方式只能引用该数据行的数据，如果要引用其它行的数据，必须使用相对定位的方式：

[列名+X]

X 是一个整数，表示细节区数据行之后的第几行，例如：

[客户+2]

由于同一个细节区不再是对应一个数据行，而是对应多个数据行，所以在定义细节区的时候，要增加一个参数，用于指定每个细节区对应的数据行数，这个参数以符号@开头：

<订单,@4>

综上所述，为了一个细节区打印四个标签，每个标签来自不同数据行，可以按下图所示设计模板：

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													<END>
2	<订单,@4,11>	客户:	[客户]		客户:	[客户+1]		客户:	[客户+2]		客户:	[客户+3]	<ALL>
3		产品:	[产品]		产品:	[产品+1]		产品:	[产品+2]		产品:	[产品+3]	
4		数量:	[数量]		数量:	[数量+1]		数量:	[数量+2]		数量:	[数量+3]	
5		日期:	[日期]		日期:	[日期+1]		日期:	[日期+2]		日期:	[日期+3]	
6	<订单>												
7	<END>												
8													

打印结果：



客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD03
数量:	490
日期:	1999-1-4

客户:	CS04
产品:	PD04
数量:	240
日期:	1999-1-7

客户:	CS03
产品:	PD05
数量:	160
日期:	1999-1-7

客户:	CS04
产品:	PD01
数量:	200
日期:	1999-1-8

客户:	CS02
产品:	PD03
数量:	200
日期:	1999-1-8

客户:	CS02
产品:	PD01
数量:	800
日期:	1999-1-10

客户:	CS04
产品:	PD01
数量:	500
日期:	1999-1-10

客户:	CS01
产品:	PD03
数量:	300
日期:	1999-1-11

客户:	CS05
产品:	PD04
数量:	500
日期:	1999-1-11

客户:	CS03
产品:	PD04
数量:	30
日期:	1999-1-12

客户:	CS05
产品:	PD02
数量:	200
日期:	1999-1-13

客户:	CS03
产品:	PD01
数量:	200
日期:	1999-1-14

同样，如果希望一个细节区打印四个标签，前两个标签来自一个数据行，后两个标签来自一个数据行，可以这样设计模板：

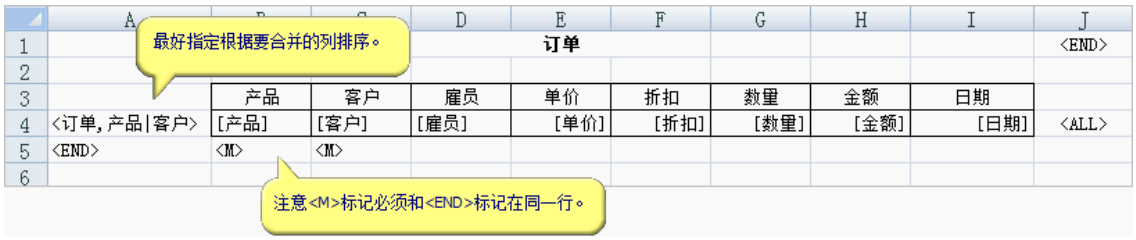
	A	B	C	D	E	F	G	H	I	J	K	L	M
1													<END>
2	<订单,@2,11>	客户:	[客户]		客户:	[客户]		客户:	[客户+1]		客户:	[客户+1]	<ALL>
3		产品:	[产品]		产品:	[产品]		产品:	[产品+1]		产品:	[产品+1]	
4		数量:	[数量]		数量:	[数量]		数量:	[数量+1]		数量:	[数量+1]	
5		日期:	[日期]		日期:	[日期]		日期:	[日期+1]		日期:	[日期+1]	
6	<订单>												
7	<END>												
8													

3.13.36 自动合并单元格

Excel 报表提供了自动合并单元格的功能，可以将同一列内容相同的单元格自动合并为一个。

上一节提到用<End>标记指定报表的有效区域，如果在有效区标记行的某一列加上标记<M>，会自动合并该列的相同内容。

例如：



	A	B	C	D	E	F	G	H	I	J
1					订单					<END>
2										
3		产品	客户	雇员	单价	折扣	数量	金额	日期	
4	<订单,产品|客户>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
5	<END>	<M>	<M>							
6										

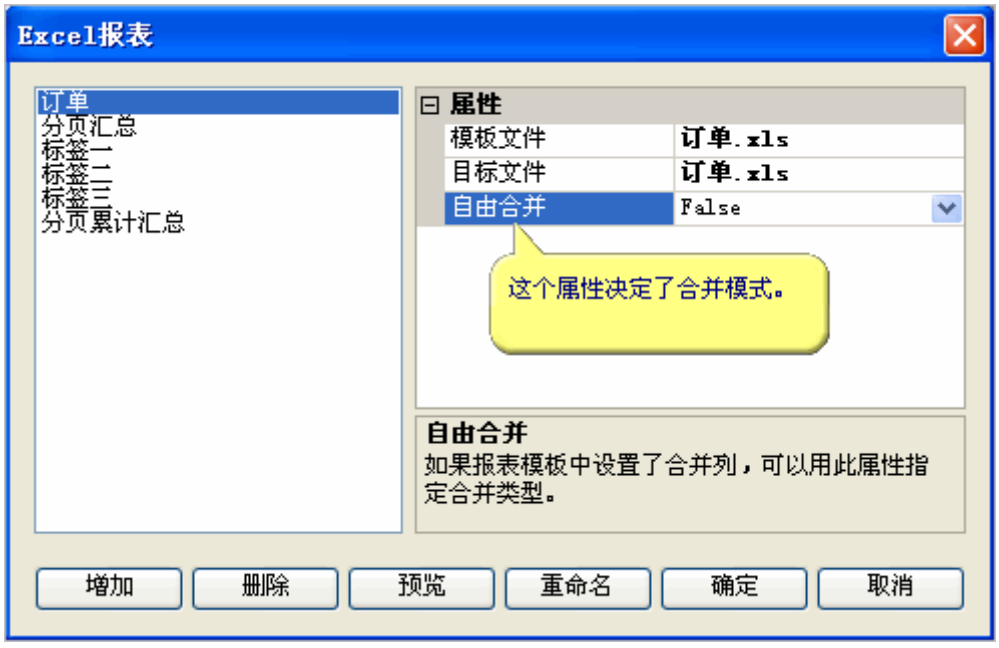
提示：因为只有内容相同且相邻的单元格才会合并，所以设置模板的时候，通常会将排序列设置为要合并的列。

上图的报表模板指定要合并产品列和客户列，这是输出结果：

A	B	C	D	E	F	G	H
			订单				
产品	客户	雇员	单价	折扣	数量	金额	日期
PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
		EP05	14.4	0	100	1440	1999-01-20
	CS03	EP04	18	0.15	80	1224	1999-01-04
		EP04	14.4	0.25	200	2160	1999-01-14
		EP01	14.4	0	240	3456	1999-01-21
	CS04	EP05	14.4	0	200	2880	1999-01-08
		EP02	14.4	0.05	500	6840	1999-01-10
PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
		EP04	15.2	0	100	1520	1999-01-14
	CS03	EP01	15.2	0.1	200	2736	1999-01-17
	CS05	EP04	19	0.1	200	3420	1999-01-13
		EP04	15.2	0	400	6080	1999-01-14
PD03	CS01	EP04	8	0	300	2400	1999-01-11
	CS02	EP02	8	0.1	200	1440	1999-01-08
		EP05	8	0.25	700	4200	1999-01-26
		EP03	10	0.1	200	1800	1999-01-28
	CS03	EP04	10	0	490	4900	1999-01-04
	CS04	EP04	10	0	40	400	1999-01-17
	CS03	EP04	[illegible]	0	[illegible]	660	1999-01-12

合并类型有两种，分别是自由合并和标准合并，标准合并必须是前一列和本列内容都相同才合并，自由合并只需本列内容相同即可合并。

Excel 报表默认是标准合并，你可以通过下面的属性来选择合并类型：



3.13.37 引用图片文件

引用图片文件的完整格式为：

[&&文件名, X, Y, Width, Height]

X:　　左边距

Y： 上边距
Width： 图片宽度
Height：图片高度
还有另外两种简化引用格式：
[&&文件名]
或
[&&文件名, X, Y]
示例
如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可：
[&&myfox.gif, 2, 2]
否则需要包括路径，例如：
[&&c:\Images\myfox.gif, 2, 2]
对于固定的图片文件，不如在设计模板的时候直接插入，无需采用引用的方式。
如果图片是动态生成的，则只能采用引用的方式，通常是事先用代码生成图片文件，并保存在项目的 Images 子目录下，然后再生成 Excel 报表。

3.13.38 再谈表达式计算

前面的章节已经提到，可以在 Excel 报表中直接进行表达式计算，表达式用符号"<"和">"括起来。例如：
<"编号：" & [编号]>

<CUMoney([金额])>
不过下面两个表达式却不能正常计算：
<[%Sum(金额)] & "元">
<CUMoney([%Sum(金额)])>
[%Sum(金额)]这种用于数据统计的格式，在表达式中出现的时候，必须用符号*括起来，所以上面两个表达式应该修改为：
<*[%Sum(金额)]* & "元">
<CUMoney(*[%Sum(金额)]*)>
实际上除了**[列名]**这种标准数据引用格式外，所有其他格式，在表达式中出现的时候，都必须用符号*括起来。
下面我们来总结一下这些非标准的格式。
正如前面所言，数据统计格式在表达式中出现的时候，必须用符号*括起来，例如下面的表达式是错误的：
<Round([%Sum(金额)] /10000,2) & "万元">
正确的表达式为：
<Round(*[%Sum(金额)]* /10000,2) & "万元">
我们在学习标签打印的时候，已经知道下面的格式：
[列名+X]
表示细节区数据行之后的第 X 行，X 是个整数，例如：
[数量+2]
表示细节区数据行之后第 2 行的数量列。

这种跨行引用数据的格式参与表达式计算的时候，也必须用符号*括起来，例如：
<IIF(*[数量+1]* > 100, *[数量+1]*, 0)>
此外，直接引用格式在表达式中出现的时候，也必须用符号*括起来，例如：
<"单号:" & *[@任务,单号,+0]*>
同样，表达式中引用父表数据的格式，也必须用符号*括起来，例如要在订单表的细节区的表达式中，引用父表产品表中的产品名称：
<"品名" & *[产品,产品名称]*>

3.13.39 金额的按位打印

如果要用 Excel 打印或套打一些财务报表和单据，会涉及到金额的中文大写和分位打印：



金额的大小写比较简单，设计模板的时候，只需此单元格的内容设置为：<CUMoney([金额])>
金额分位打印稍微复杂的，需要用 GetDigit 函数逐位设置，下表列出了设计模板的时候，每位金额对应的公式：

位	公式
分	<GetDigit([金额],-2,"￥")>
角	<GetDigit([金额],-1,"￥")>
元	<GetDigit([金额],0,"￥")>
十	<GetDigit([金额],1,"￥")>
百	<GetDigit([金额],2,"￥")>
千	<GetDigit([金额],3,"￥")>
万	<GetDigit([金额],4,"￥")>
十万	<GetDigit([金额],5,"￥")>
百万	<GetDigit([金额],6,"￥")>
千万	<GetDigit([金额],7,"￥")>
亿	<GetDigit([金额],8,"￥")>

正如上一节所介绍，如果需要对合计的金额大写或分位打印，记得加上*，例如：
<CUMoney(*[%出库明细,Sum(数量)]*)>
<GetDigit(*[%出库明细,Sum(数量)]*,3,"￥")>

3.13.40 再谈有效区域的设置

在编写 Excel 报表示例文件的时候，我按照下图所示设置了列的有效区域：

A	B	C	D	E	F	G	H
<出库,1>	湛江佳信公司出库单						<END>
	<"出库单号:" & [出库单编号]>						
	<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
<END>							

表面上看没有任何问题，但是结果却让我大吃一惊，生成报表的时候提示错误“未找到列[END]”。原来出库表的细节区是从第一行开始的，而细节区的此位置是用于定义打印条件的，而不是用于定义有效区域的，但是有效区域的定义必须出现在表格的第一行，如何来解决这种冲突呢？

我们可以另外插入一行，用于定义有效区域，如下图所示：

A	B	C	D	E	F	G	H
							<END>
<出库,1>	湛江佳信公司出库单						
	<"出库单号:" & [出库单编号]>						
	<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
<END>							

这样处理虽然解决了冲突的问题，但是却浪费了一行的打印空间，如果你在意这个问题，可以将第一行隐藏起来，这样既定义了有效区域，同时又不会对打印效果带来任何影响。

3.13.41 使用条形码字体

通过条形码字体，可以在 Excel 报表中批量生成条形码。

77 种条形码字体的下载地址：

http://www.foxtable.com/download/codebarfont.rar

解压后，双击某个字体文件，可以看到字体名称和对应的条形码。

使用方法：

如果是 Windows 7/vista，右击某条形码字体文件，从弹出的快捷菜单中执行“安装”命令。

如果是 Windows xp 系统，则进入控制面板，切换到经典视图，再进入字体文件夹，将需要使用的条形码字体文件复制粘贴到此文件夹中。

安装条形码字体后,在 Excel 报表批量生成条形码就非常简单了，可以按常规引用某字段，然后将其字体设置为条形码字体，并设置合适的字体大小即可。

提示：

为了提高条形码扫描器的识别率，有些条形码需要加上首尾识别符号，例如 Code39 条形码，就建议在首尾各附加一个*号，为此在 Excel 模板可以这样引用条码列：

<"*" & [条码列] & "*">

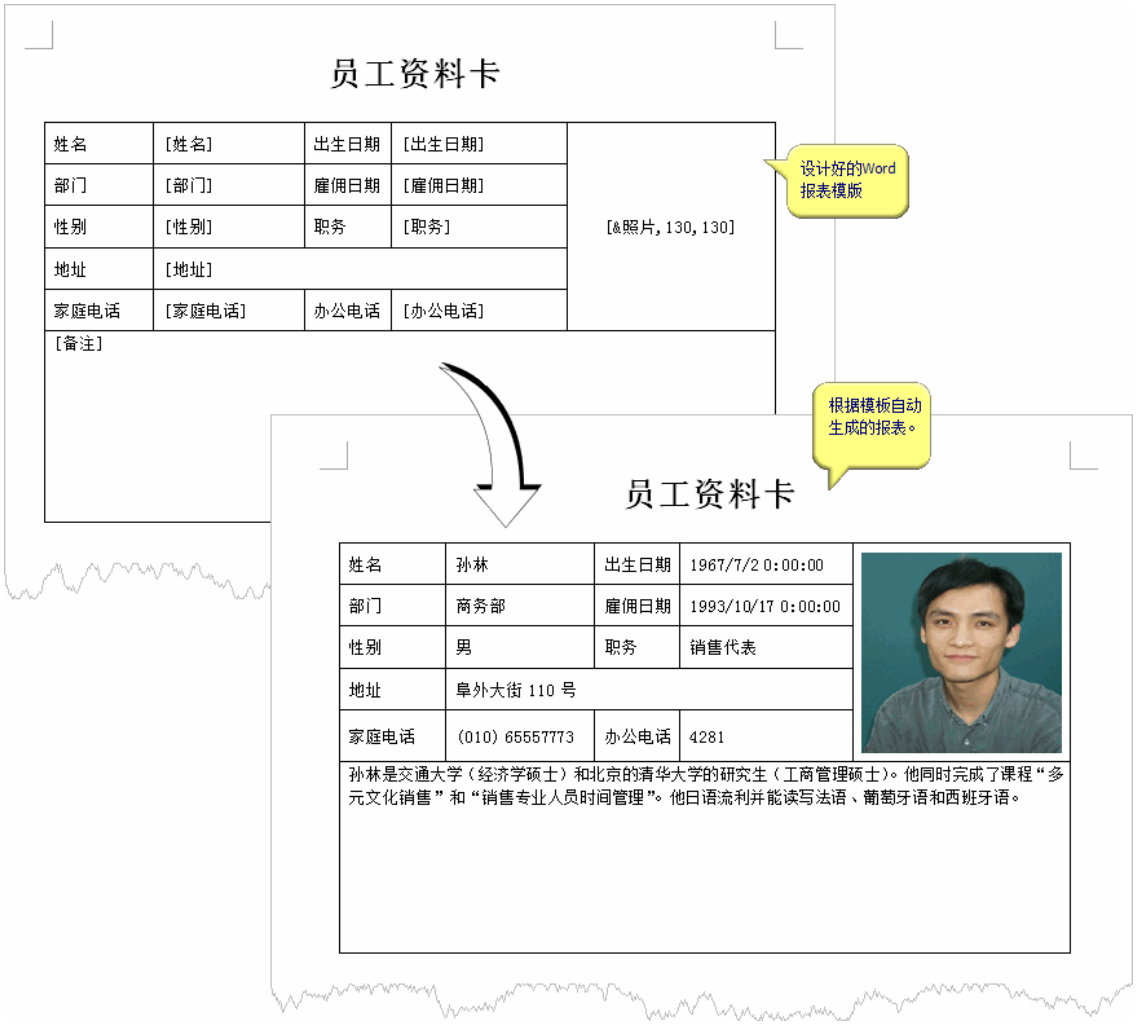
3.13.42 分割打印

前面提到，我们可以通过设置细节区标示的参数，来指定每页可打印的细节区数，例如：
<学生,-30>
表示每页最多打印 30 个学生，在不够 30 个的情况下，自动补充空行。
有的时候，我们会有一些特殊的要求，例如我们希望分组打印学生花名册，每组 9 个学生，每页最多打印 3 组，可以这样设置
<学生,(-9|3)>

3.14 Word 报表

3.14.1 关于 Word 报表

实际上，在 Foxtable 的规划之中，最初是没有 Word 报表的，报表主要通过 Excel 报表来实现。
Word 作为字处理软件的王者，在复杂版面的设计方面具有无可比拟的优势和灵活性，所以 Foxtable 从 2014 版开始，提供了 Word 报表功能。
Foxtable 和 Excel 的长处是数据管理，而不是字处理，字处理的王者是微软的 Word。
有了 Word 报表，Foxtable 可以和 Word 完美地结合起来，你可以用 Word 可视化地设计一个报表模版，Foxtable 能根据这个模版的设定，自动填入数据，批量生成报表：



3.14.2 数据引用说明

标记	说明
[列名]	引用主表指定列数据，例如: [客户] 表示引用客户列的数据
[&列名,宽度,高度]	引用主表某列，以图片的形式显示，例如： [&照片,130,160] 表示引用照片列的内容，并以图片的形式显示，宽度是 130，高度是 160，单位为磅 如果照片列有多个图片，可以用下面的格式引用指定位置的图片： [&照片&x,100,120] x 数个数字，表示图片位置编号，0 表示第一个图片，1 表示第二个图片。
[@父表,列名]	引用父表指定列的数据，例如订单表和产品表建立了关联，产品

	表是父表，那么在订单表中设计模版的时候，下面的标记： [@产品,规格] 表示引用父表（产品表）的规格列的数据
[@子表,列名,行号]	引用子表指定行的指定列的数据，因为子表通常会存在多个对应行，所以要有行号参数表示引用子表第几行的数据。 例如假定出库和出库明细建立了关联，出库表是父表，在出库表设计模版的时候，下面的标记： [@出库明细,数量,2] 表示引用子表（出库明细表）第 2 行的数量列的内容。
[@&父表,列名,宽度,高度]	引用父表某列，以图片的形式显示，例如假定订单表和产品表建立了关联，产品表是父表，那么在订单表中设计模版的时候，下面的标记： [@&产品,照片,100,120] 表示引用父表（产品表）的照片列，并以图片的形式显示,宽度是 100，高度是 120，单位为磅。 如果照片列有多个图片，可以用下面的格式引用指定位置的图片： [@&产品,照片&x,100,120] x 数个数字，表示图片位置编号，0 表示第一个图片，1 表示第二个图片。
[@&子表,列名,行号,宽度,高度]	引用子表某列，以图片的形式显示，因为子表通常会存在多个对应行，所以要有行号参数表示引用子表第几行的数据。 例如假定员工表和家庭成员表建立了关联，员工表是父表，那么在员工表中设计模版的时候，下面的标记： [@&家庭成员,照片,1,60,80] 表示引用子表（家庭成员表）的第一行的照片列，并以图片的形式打印,宽度是 60，高度是 80，单位为磅 如果照片列有多个图片，可以用下面的格式引用指定位置的图片： [@&家庭成员,照片&x,1,60,80] x 数个数字，表示图片位置编号，0 表示第一个图片，1 表示第二个图片。
[%子表,Sum(列名)]	统计子表指定列的数据，如出库和出库明细建立了关联，出库表是父表，在出库表设计模版的时候，下面的标记： [%出库明细,Sum(数量)] 表示累计出库明细的数量列，Sum 是聚合函数，表示求和，可以使用其它聚合函数进行其他统计，参考：聚合函数
[&&图片文件,宽度,高度]	打印指定的图片文件，文件事先复制到 Images 或 Attachments 目录下，则只需指定文件名，例如： [&&Denmark.png,48,48] 否则需要包括路径，例如： [&&c:\data\Denmark.png,48,48]

<表达式>	对表达式进行计算，采用 VBScript 语法和函数，VBSCript 提供了很多函数，具体请参考：VBScript 参考 例如你要将出生日期以中文大写的形式打印出来，可以这样设置模板： <CUDATE([出生日期])>
<Index>	<Index>表示顺序号，主表每打印一行，顺序号就会增加 1

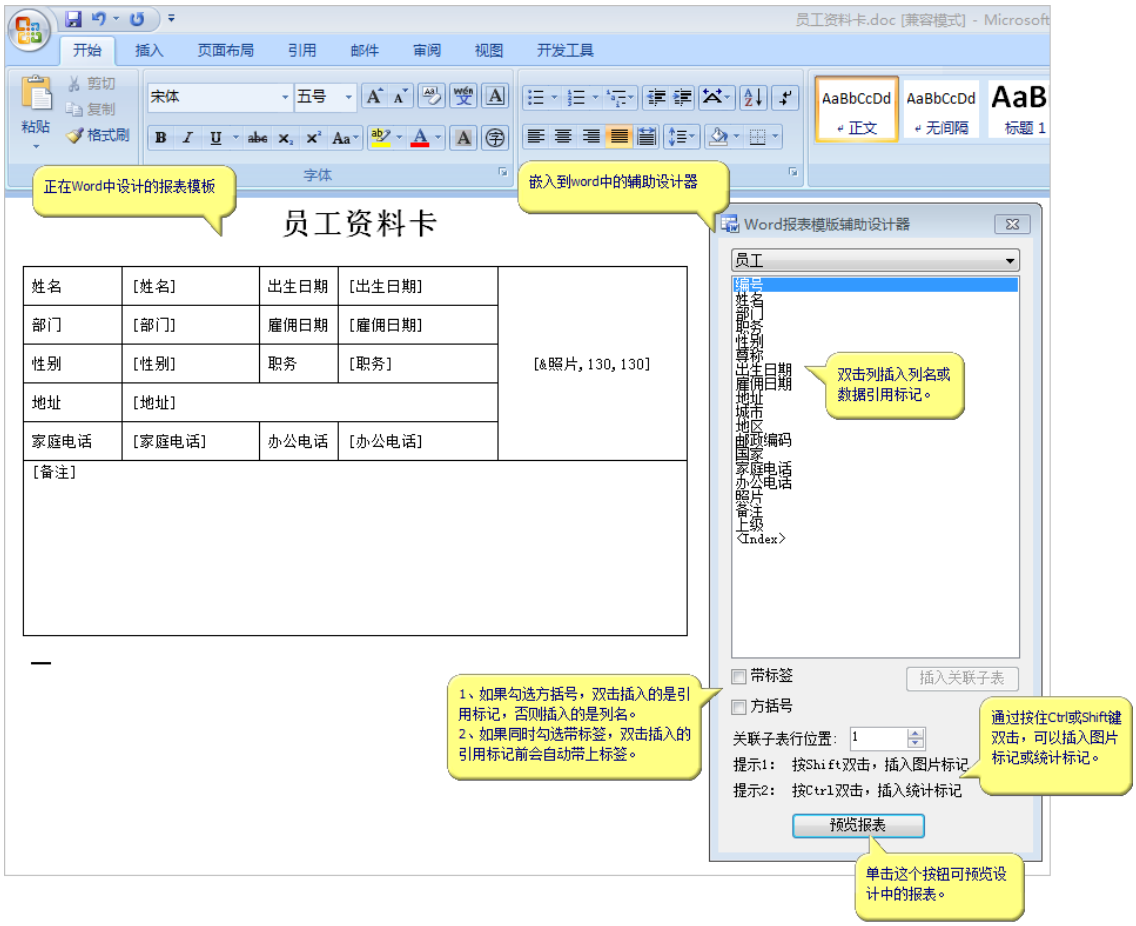
3.14.3 设计报表模版

在菜单的“打印输出”功能区的“Word 报表”功能组，单击“设计模版”，可以新建模版，或打开现有模版修改：



所有的模版默认保存在项目所在目录的 Attachments 子目录中，为方便管理，最好不要改变这个存储目录。

Foxtable 直接利用 Word 来设计模版，而且在 Word 中嵌入了一个辅助设计器，一切都是可视化，你无须手工输入，只需双击，即可插入列名和 各种数据引用标记，而且还可以一边设计，一边预览效果：



Word 先天具备无可比拟的设计优势，再加上 Foxtable 给 Word 嵌入的模版辅助设计器，可以说双剑合璧，如虎添翼，将大大提高我们的报表设计效率和能力 。

当然你也可以直接启动 Word 来设计报表模版，将设计好的模板文件保存在 Attachments 目录就行，只是没有了辅助设计器，设计效率会大打折扣。

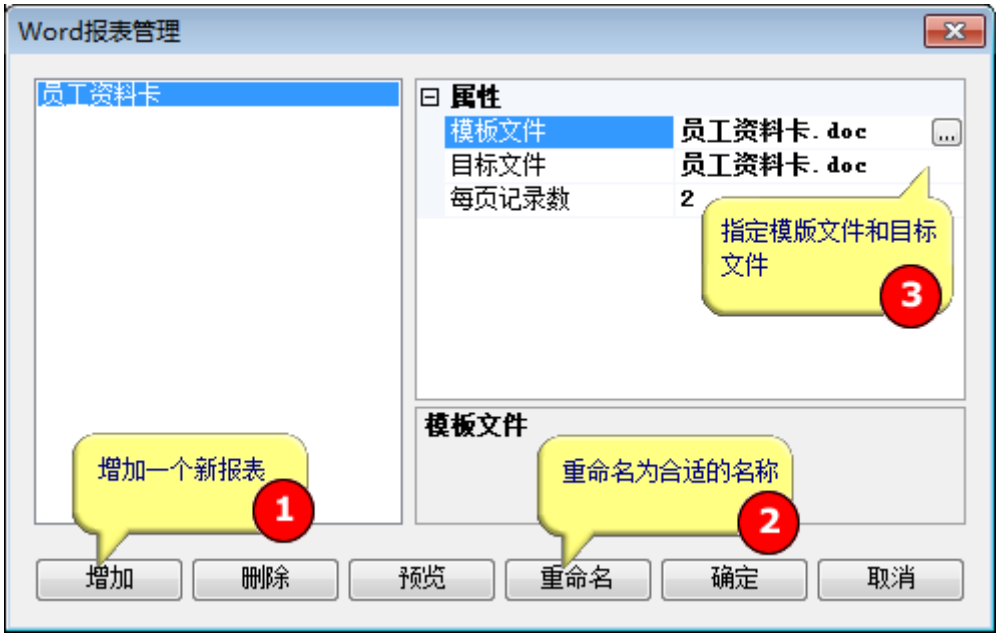
3.14.4 添加报表模版

设计好报表模版之后，我们需要将报表模版添加到菜单中。

在菜单的“打印输出”功能区的“Word 报表”功能组，单击“报表管理”按钮：



会出现下图所示的窗口，通过这个窗口，可以添加已经设计好的报表模版：



目标文件指的是根据模板文件生成的报表文件的文件名。

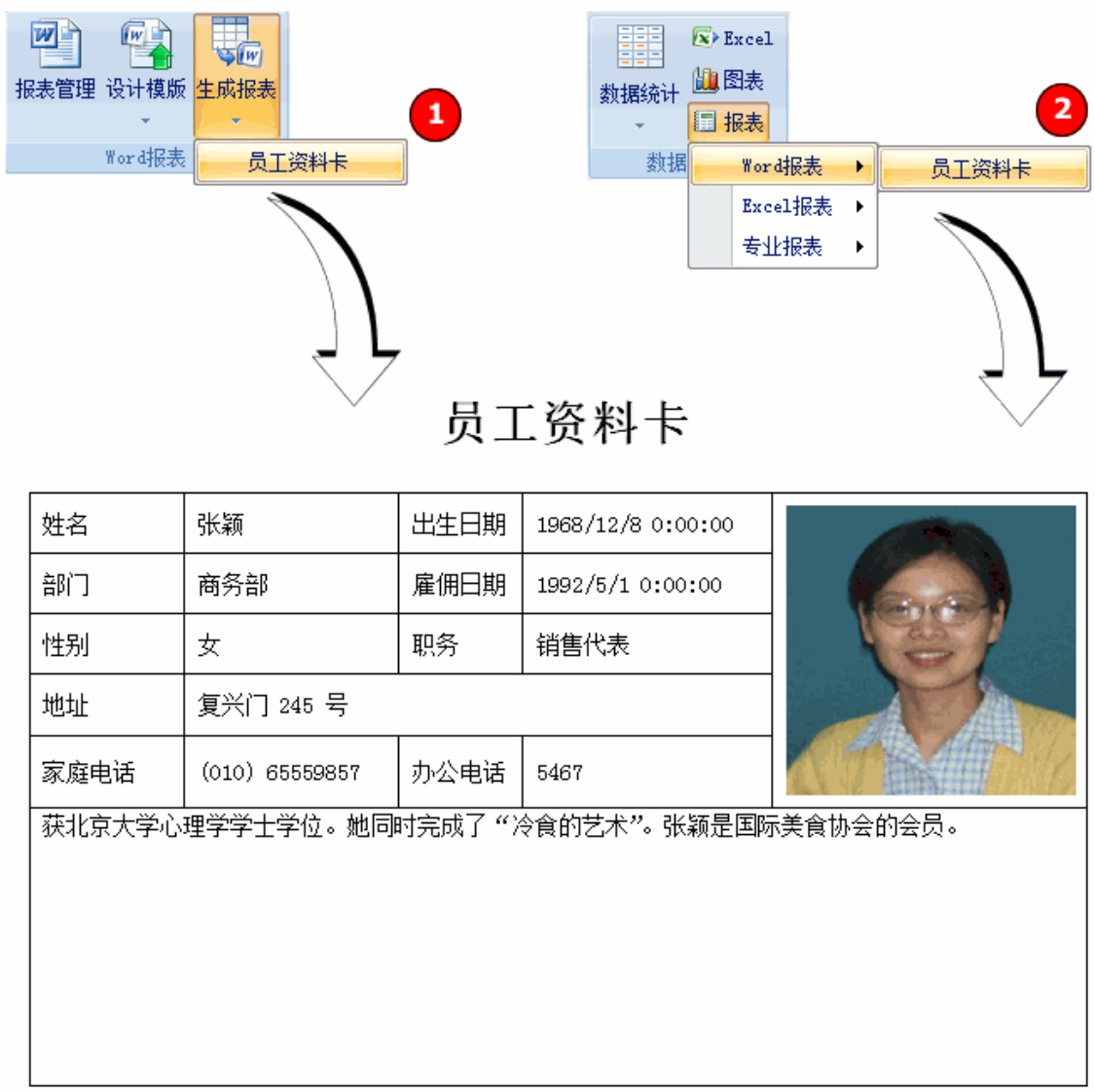
3.14.5 生成 Word 报表

设计好 Word 报表模版，并将模版添加到菜单后，就可以通过菜单来生成 Word 报表了。

菜单中有两处地方可以生成 word 报表：

1、在菜单的“打印输出”功能区的“Word 报表”功能组，单击“生成报表”按钮 ，会列出已经添加的模版，单击其中一个，即可生成报表。

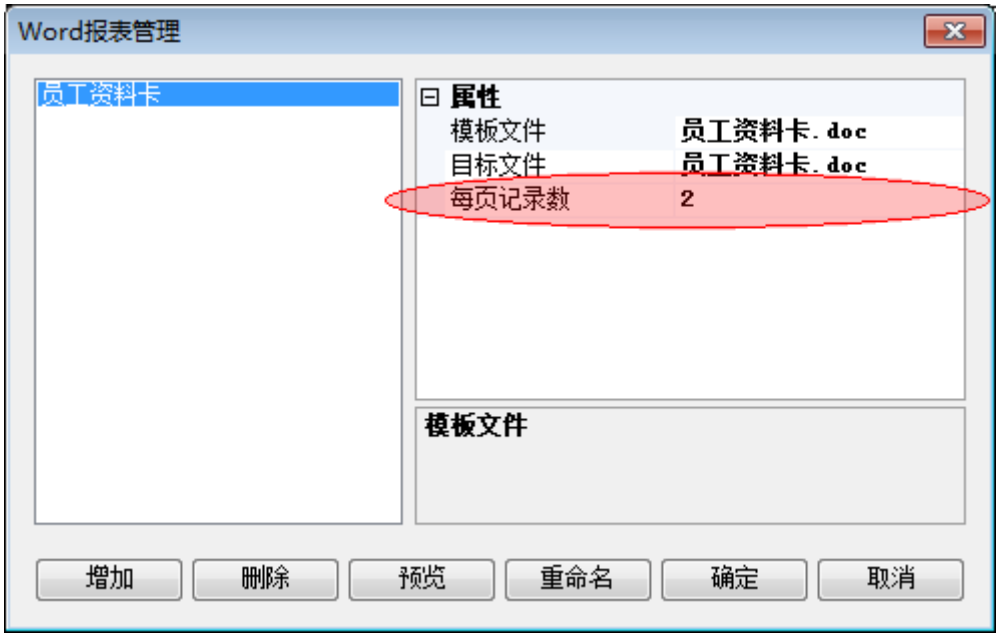
2、在菜单的“日常工作”功能区的“数据统计”功能组，单击“报表”按钮，指向“Word 报表”,也会列出已经添加的模版，单击其中一个，即可生成报表。



系统会自动打开生成的报表文件，这些文件默认存放在目录目录的 Reports 子目录下。

3.14.6 自动换页控制

在添加模板的时候，我们可以指定每页打印的记录数，例如下图意味着每页打印两个员工的资料卡：



这样生成报表的时候，每打印两个员工的资料卡，就会另起一页。
如果我们不指定每页打印的记录数（参数设为 0），由系统自动换页，那么某员工的资料卡可能跨越两页打印，显然这不是我们所希望的。

3.14.7 Word 文章套打

用 Word 报表，可以轻松实现文章套打，例如我们可以用 Word 设计一个下图所示的模版：

尊敬的[客户]：

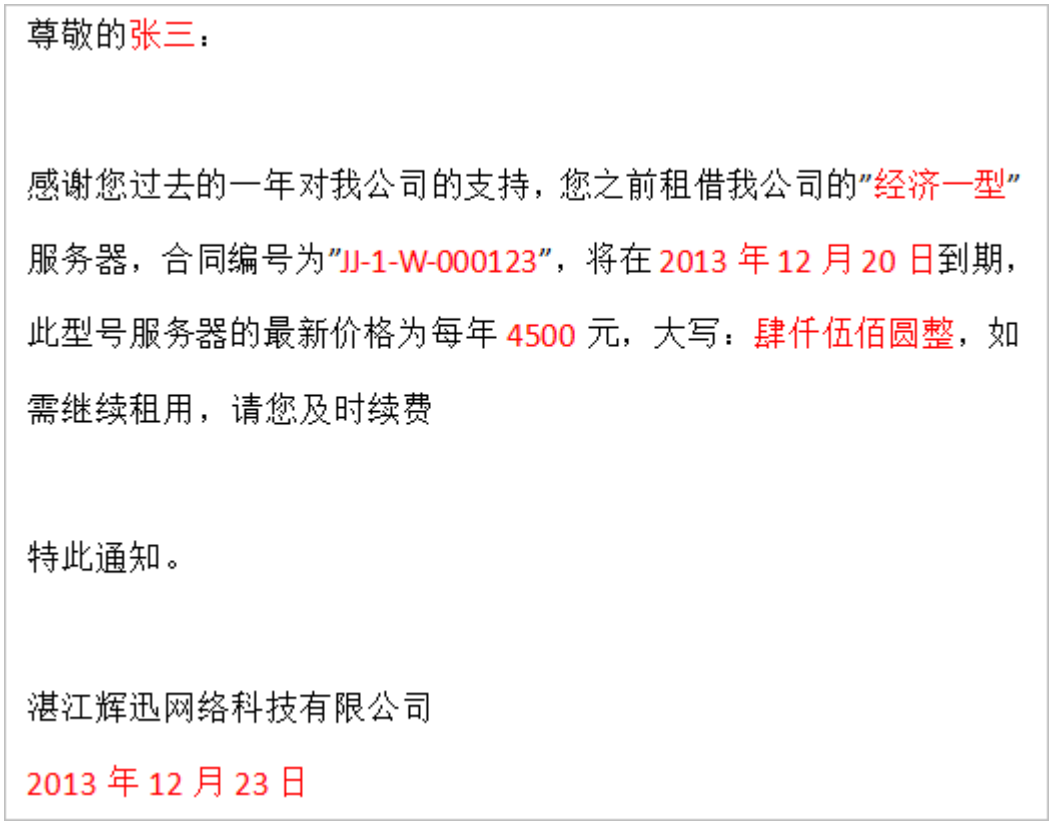
感谢您过去的一年对我公司的支持，您之前租借我公司的"[型号]"服务器，合同编号为"[合同编号]"，将在<Format([到期日], "yyyy 年 MM 月 dd 日")>到期，此型号服务器的最新价格为每年[价格]元，大写：<CUMoney([价格])>，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

<Format(Date, "yyyy 年 MM 月 dd 日")>

上图的模版中，我们用红色标出了数据引用标记，有直接引用，例如：[合同编号]，也有表达式，例如：<CUMoney([价格])>。
当我们根据这个模板生成报表时，Foxtable 会自动填入数据并计算表达式，得到正式的文章：


尊敬的张三：

感谢您过去的一年对我公司的支持，您之前租借我公司的"经济一型"服务器，合同编号为"JJ-1-W-000123"，将在 2013 年 12 月 20 日到期，此型号服务器的最新价格为每年 4500 元，大写：肆仟伍佰圆整，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

2013 年 12 月 23 日

3.14.8 打印关联子表

父表一行，在子表通常会存在多个对应行，就象一个父亲会有多个儿子一样，所以我们引用关联子表的时候，要有行号参数表示引用子表第几行的数据。

引用子表的标记格式为：

[@子表,列名,行号]
例如假定出库和出库明细建立了关联，出库表是父表，在出库表设计模版的时候，下面的标记：

[@出库明细,数量,5]

表示引用子表（出库明细表）第 5 行的数量列的内容，如果子表不够 5 行，则生成报表时此标记不会显示任何内容。
如果要统计子表，标记格式为：
[%子表,Sum(列名)]
例如:
[%出库明细,Sum(数量)]
表示累计出库明细子表的数量列。

如果 我们在出库单表设计报表时，按下图所示设计模板：



湛江辉迅公司出库单

出库单号：[出库单编号]					
领用部门：[领用部门]		出库日期：[出库日期]		领料人：[领料人]	
商品代码	商品名称	规格	单位	数量	备注
[@出库明细,商品代码,1]	[@出库明细,商品名称,1]	[@出库明细,规格,1]	[@出库明细,单位,1]	[@出库明细,数量,1]	[@出库明细,备注,1]
[@出库明细,商品代码,2]	[@出库明细,商品名称,2]	[@出库明细,规格,2]	[@出库明细,单位,2]	[@出库明细,数量,2]	[@出库明细,备注,2]
[@出库明细,商品代码,3]	[@出库明细,商品名称,3]	[@出库明细,规格,3]	[@出库明细,单位,3]	[@出库明细,数量,3]	[@出库明细,备注,3]
[@出库明细,商品代码,4]	[@出库明细,商品名称,4]	[@出库明细,规格,4]	[@出库明细,单位,4]	[@出库明细,数量,4]	[@出库明细,备注,4]
[@出库明细,商品代码,5]	[@出库明细,商品名称,5]	[@出库明细,规格,5]	[@出库明细,单位,5]	[@出库明细,数量,5]	[@出库明细,备注,5]
库管员：[库管员]			合计：	[%出库明细,Sum(数量)]	

可以生成下图所示的出库单：



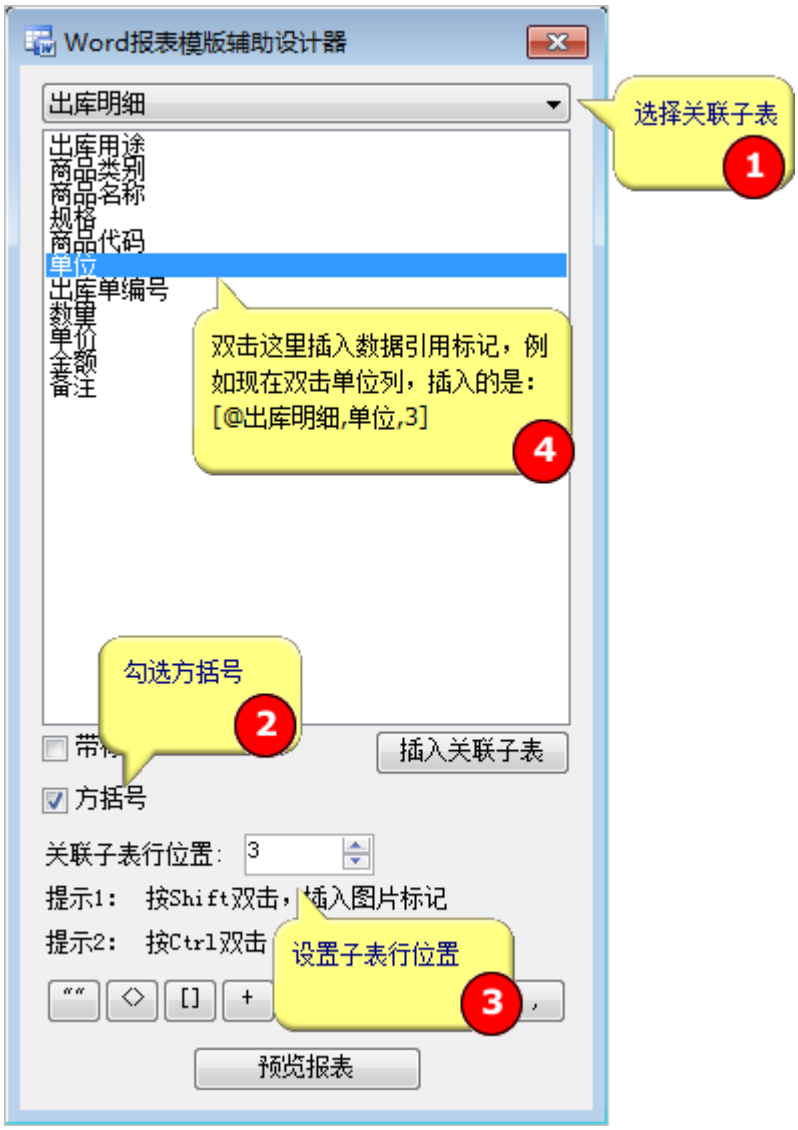
湛江辉迅公司出库单

出库单号：CK-20030726002
领用部门：商函广告公司　　出库日期：2008/7/26 0:00:00　　领料人：朱志欣

商品代码	商品名称	规格	单位	数量	备注
BG00201	记事本	16 开精装	本	8	
YW00101	特快详情单		张	1000	

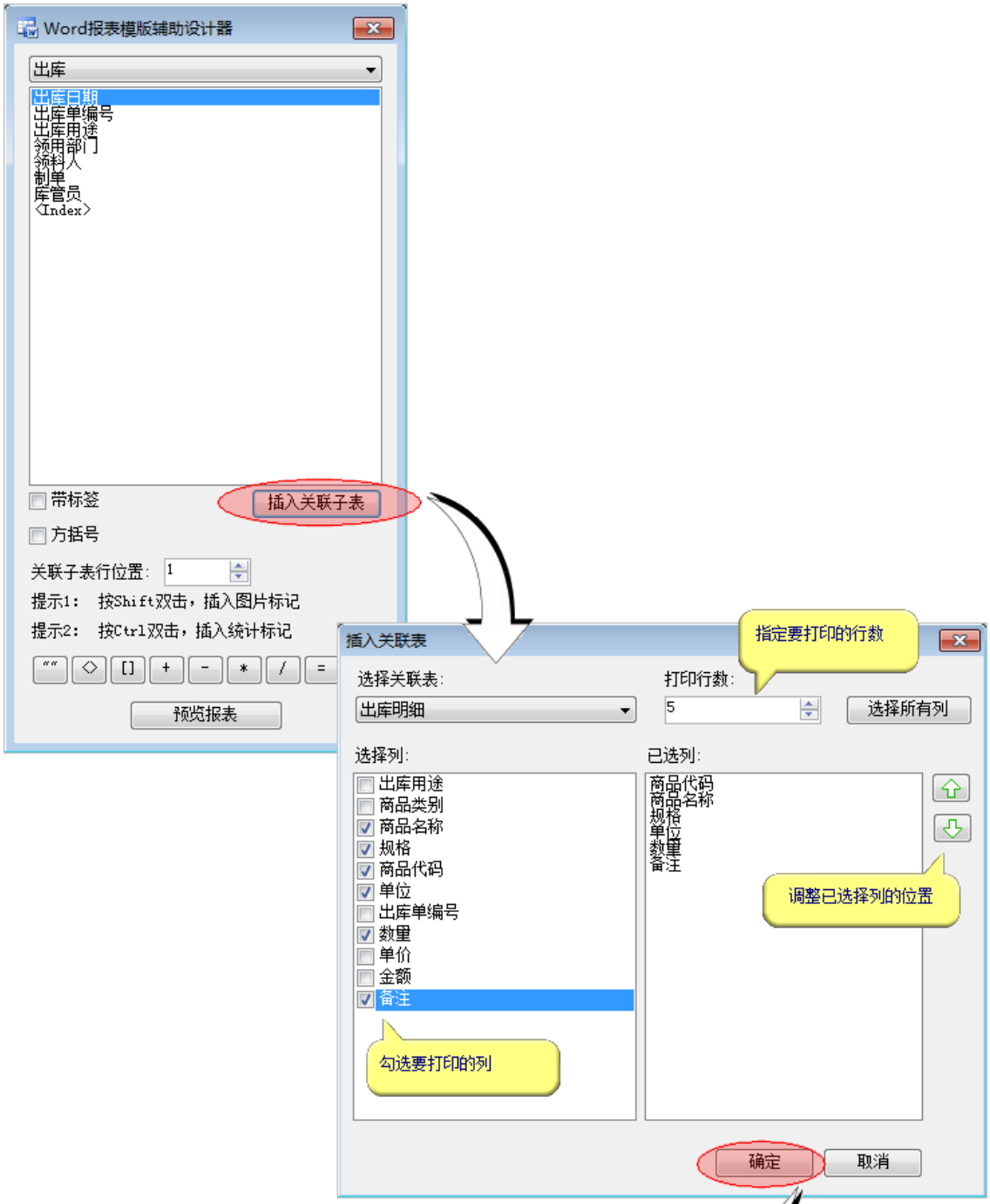
库管员：李政旭　　合计：　1008

在设计模板的时候，可以按照下面的提示快速插入关联子表的数据引用标记：



3.14.9 插入关联子表

在打印关联子表的时候，我们并不需要逐个单元格去插入引用标记，模板设计辅助器提供一个插入按钮，可以一次整个插入关联子表，轻松无比：



商品代码	商品名称	规格	单位	数量	备注
[@出库明细,商品代码,1]	[@出库明细,商品名称,1]	[@出库明细,规格,1]	[@出库明细,单位,1]	[@出库明细,数量,1]	[@出库明细,备注,1]
[@出库明细,商品代码,2]	[@出库明细,商品名称,2]	[@出库明细,规格,2]	[@出库明细,单位,2]	[@出库明细,数量,2]	[@出库明细,备注,2]
[@出库明细,商品代码,3]	[@出库明细,商品名称,3]	[@出库明细,规格,3]	[@出库明细,单位,3]	[@出库明细,数量,3]	[@出库明细,备注,3]
[@出库明细,商品代码,4]	[@出库明细,商品名称,4]	[@出库明细,规格,4]	[@出库明细,单位,4]	[@出库明细,数量,4]	[@出库明细,备注,4]
[@出库明细,商品代码,5]	[@出库明细,商品名称,5]	[@出库明细,规格,5]	[@出库明细,单位,5]	[@出库明细,数量,5]	[@出库明细,备注,5]

在Word中插入的关联子表

3.14.10 常用函数介绍

我们已经知道在设计 Word 模板的时候，可以插入表达式，格式为：

<表达式>
需要注意的是，表达式中的列名，必须用方括号括起来，例如：
<Round([数量] * [单价],2)>
表示数量乘以单价，并保留两位小数，如数量和单价不用方括号括起来，此表达式将不能正确计算。
表达式采用 VBScript 语法，而且 Foxtable 对 VBScript 进行了扩展，提供了很多非常实用的函数。
例如你要将出生日期以中文大写的形式打印出来，可以这样设置模板：<CUDATE([出生日期])>
再例如，假定已付款是个逻辑列，希望打印的时候，已付款的用符号"√"，可以这样设置模板：IIF([已付款], "√", "")
我们下面介绍一些常用的函数，更多的函数和更具体的用法，请参考：VBSCript 参考

函数名	说明
Format	格式化日期或数字。 格式化数字的时候，用 0 表示占位符，如果指定位置有数字，则显示数字，如果没有数字，则用 0 补充。 例如需要金额保留两位小数显示： <Format([金额],"#0.00")> 如果金额的值是 123.567，则显示为 123.57，如果金额的值是 123.5，则显示为 123.50，而不是 123.5。 如果希望以百分比的形式显示，可以加上符号%，例如希望以百分比形式显示合格率，保留一位小数： <Format([合格率],"#0.0%")> 格式化日期的时候，yyyy 表示四位年，yy 表示两位年，MM 表两 2 位月,M 表示一位月，dd 表示两位日,d 表示一位日，例如： <Format([出库日期],"yyyy 年 MM 月 dd 日")> 显示的日期格式为：2013 年 09 月 08 日 <Format([出库日期],"yyyy 年 M 月 d 日")> 显示的日期格式为：2013 年 9 月 8 日 <Format([出库日期],"yy 年 M 月 d 日")> 显示的日期格式为：13 年 9 月 8 日 注意 M 是大写。
IIF	语法为： IIF(expression, truevalue, falsevalue) 如果逻辑表达式 expression 的计算结果为 True，将返回 truevalue，否则返回 falsevalue。 例如： IIF([已付款], " √ ", "")

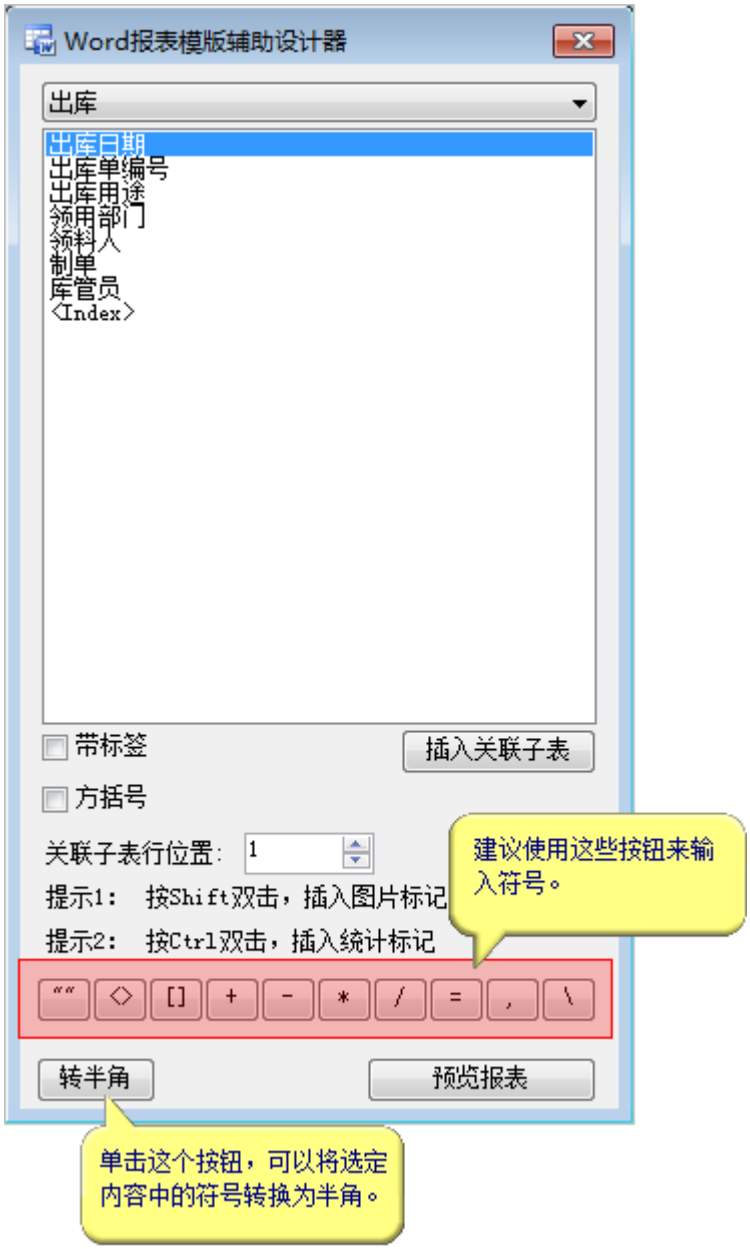
Round	对数值进行四舍五入，例如：<Round([金额],2)>
Date	返回今天的日期，例如： <Format(Date,"yyyy 年 MM 月 dd 日")> 表示以四位年两位月两位日的格式显示今天的日期。 日期可以直接和数字进行加减，例如： <Date - 2> 表示前天的日期 <Date + 1> 表示明天的日期 <Format(Date + 1,"yyyy 年 MM 月 dd 日")> 表示以四位年两位月两位日的格式显示明天的日期。
Now	放回当前时间，例如：<Now>
Year	返回日期的年份，例如: <Year(出生日期)>
Month	返回日期的月份，例如: <Month(出生日期)>
Day	返回日期的日部分，例如: <Day(出生日期)>
Weekday	用数字形式返回星期几，例如: <WeekDay(出生日期)>
WeekDayName	用名称形式返回星期几，例如: <WeekDayName(出生日期)>
CUDate	将日期 Date 转换为大写中文格式。 例如： <CUDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：贰零壹叁年拾贰月玖日
CLDate	将日期 Date 转换为小写中文格式。 例如： <CLDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：二〇一三年十二月九日
CUMoney	将数值转换为中文货币格式。 例如： <CUMoney([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍圆整
CUNumber	将数值转换为中文大写格式。 例如： <CUNumber([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍
EUNumber	将整数转换为英文大写。 例如： <EUNumber([数量])> 假定数量的值是 12345，上面的表达式显示的是：Twelve Thousand Three

	Hundred Forty Five
EUMoney	将数值转换为英文货币格式。 例如： <EUMoney([金额])> 假定金额的值是 12345.45，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five Dollars And Forty Five Cents

3.14.11 符号要半角的

在设计 Word 模板的时候，我们需要插入很多数据引用标记和表达式。

需要注意的是，标记中使用的符号必须是半角的，不能是全角的。
例如下面的表达式不会正确计算：
〈CLNum([金额])〉
正确的写法是：
<CLNum([金额])>
二者的差别在于前者首尾的尖括号时全角的，后者则是半角的。
使用汉字的我们，很难察觉这种错误，也很难避免这种错误，设计模板的一定要小心。
为了减少这种错误，Word 模板辅助设计器后来加上了两个按钮，分别用于快速输入常用的半角符号 ，以及将选定内容中的符号转换为半角：



3.14.12 报表与表的关系

显然报表的数据来自于表，所以每个报表是属于特定表的。
例如我们之前设计的"员工资料卡.doc"这个模版，生成报表的时候，会从员工表提取数据，所以这个报表是属于员工表的。

但是在设计 Word 报表模版的时候，并没有办法指定这个报表属于那个表；我们只能在设计“员工资料卡.doc”这个模板的时候，首先选择员工表，这样模板辅助设计器会自动列出员工表的所有列；同样将这个设计好的模板添加到菜单中，或者基于这个模板生成报表的时候，也必须确保当前表为员工表。
Foxtable 自动认为所有和 Word 报表相关的操作，都是基于当前表的。

3.14.13 表达式中的运算符冲突

在 Word 报表模版中，"<"和">"有特殊意义，分别用于标记表达式 的开始和结束，例如：
<CUDATE([出生日期])>
但是在表达式中，"<"和">"本身也是一个运算符，分别表示大于和小于，当然还有表示不等于的"<>"。
为了避免冲突，Foxtable 要求：
1、作为运算符的"<"、">"和"<>"，前后必须有空格。
2、用于表达式开始标记的"<"，后面不能有空格；作为表达式结束标记的">"，前面不能有空格。
下面列出一些表达式，有的正确，有的错误，请大家体会：

表达式	说明
<IIF([成绩] > 60, "及格","不及格")>	正确的表达式
<IIF([出生日期] <> #0:00#, CUDATE([出生日期]), "")>	正确的表达式
<IIF([成绩] >60, "及格","不及格")>	错误的表达式，运算符>的后面没有空格。
<IIF([成绩]> 60, "及格","不及格")>	错误的表达式，运算符>的前面没有空格。
<IIF([等级]<>"A", "录取","淘汰")>	错误表达式，运算符<>的前后都没有空格。
< IIF([成绩]> 60, "及格","不及格")>	错误表达式，开始标记<的后面有空格。
<IIF([成绩]> 60, "及格","不及格") >	错误表达式，结束标记>的前面有空格

需要特别注意的是，一个表达式编写错误，可能会导致后续其它表达式也无法正确计算。
例如假定模版中有连续的两个表达式：

是否及格：<IIF([成绩] > 60, "及格","不及格") >
是否录取：<IIF([等级] <> "A", "录取","淘汰")>

看起来只是第一个表达式有错，实际上两个表达式都无法计算，因为系统在分析模版的时候，由于第一个表达式的结尾标记">"的前面有空格，所以会继续往后查找，一直找到第一个符合要求的结尾标记(也就是第二个表达式的结尾标记)，这样生成的时候，Foxtable 会将两个表达式合并为一个，当然这个合并后的表达式完全不符合语法要求，自然也就无法计算 ，最终的结果就是两个表达式都没有结算，尽管其中一个表达式并没有错误。

所以在分析表达式错误的时候，应该从第一个没有正确计算的表达式开始分析，逐个往后处理。

3.14.14 Word 报表还是 Excel 报表?

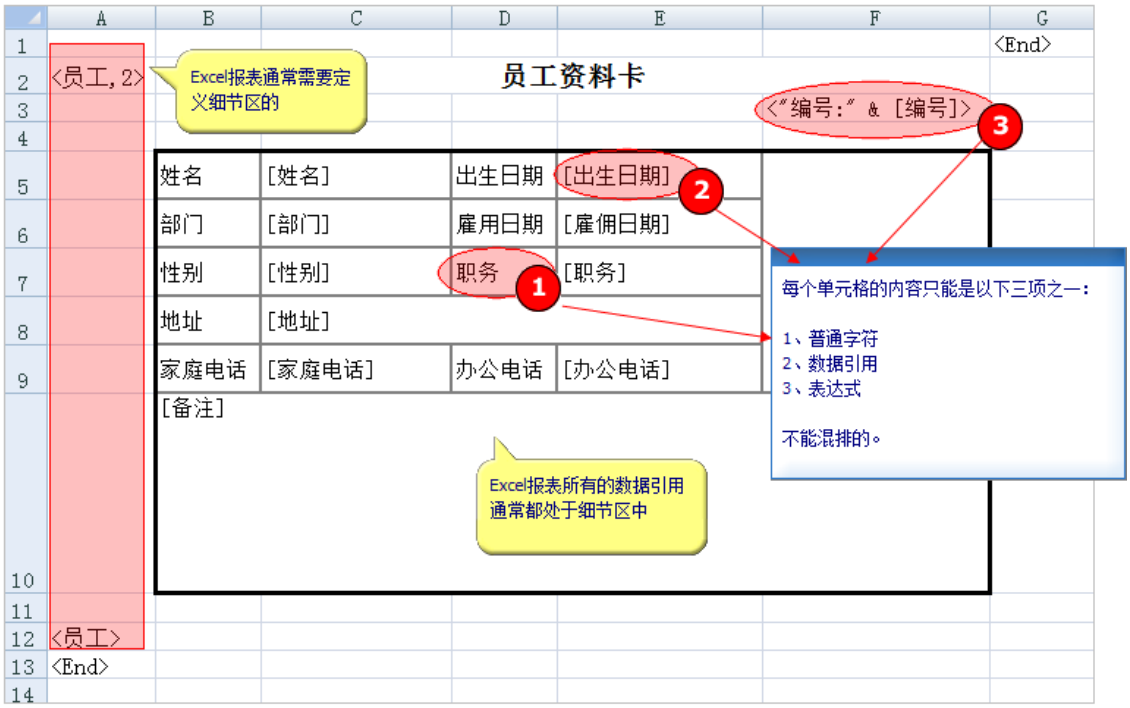
根据 Word 报表和 Excel 报表在帮助中所占的篇幅可知，Excel 报表是 Foxtable 的重点，确实 Excel 报表能给我们的， 要比 Word 报表多一些 。

但是 Word 报表也有自己的优势：

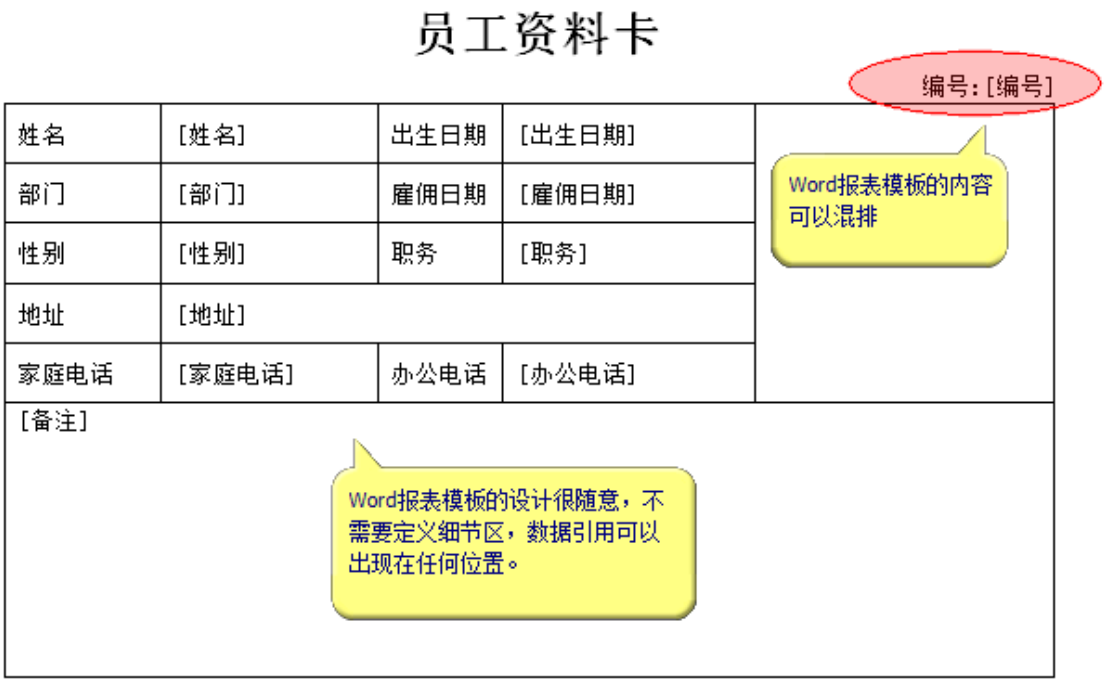
1、Word 在排版方面更具优势，更便于设计版面复杂的报表。

2、Word 报表在数据引用方面更随意。

Excel 报表在数据引用方面是比较严谨的，通常所有的数据引用标记都必须位于细节区中，而且每个单元格内容，只能是普通字符、数据引用或表达式三者中的一个，不能混排：



而 Word 报表在设计模板的时候，则“随意”很多，不需要定义细节区，单元格可以将多种内容混排在一起：



因为这种随意的特性，所以用 Word 模板可以很方便地实现文章套打，例如设计一个下图的模板：

尊敬的[客户]：

感谢您过去的一年对我公司的支持，您之前租借我公司的"[型号]"服务器，合同编号为"[合同编号]"，将在<Format([到期日], "yyyy 年 MM 月 dd 日")>到期，此型号服务器的最新价格为每年[价格]元，大写：<CUMoney([价格])>，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

<Format(Date, "yyyy 年 MM 月 dd 日")>

可以自动生成如下的文章：

尊敬的张三：

感谢您过去的一年对我公司的支持，您之前租借我公司的"经济一型"服务器，合同编号为"JJ-1-W-000123"，将在 2013 年 12 月 20 日到期，此型号服务器的最新价格为每年 4500 元，大写：肆仟伍佰圆整，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

2013 年 12 月 23 日

总结：

word 报表和 Excel 报表各有优势，大家可以根据需要选择；一般来说，根据单条记录生成一个格

式较为复杂但相对固定的表格（包括文章），或者虽然要打印关联子表，但是子表行数有限，可以在报表中 预留固定行数显示子表，此时 Word 报表较有优势，其他场合 Excel 报表优势明显。

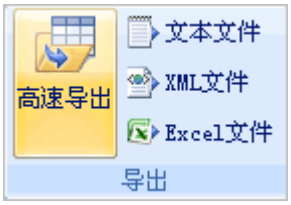
3.15 导入导出

3.15.1 导出

3.15.1.1 高速导出

高速导出能够将表格导出为文本、Access、Excel、dBase 格式

1、在“杂项”功能区的“导出”功能组，单击按钮“高速导出”。

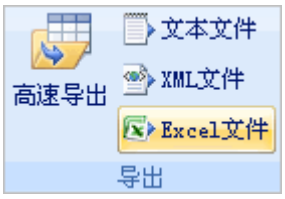


2、出现一个保存文件对话框，选择要导出的文件类型，并选择或输入接收导出数据的文件。

3、在导出设置窗口，选择要导出的列，单击“确定”按钮。

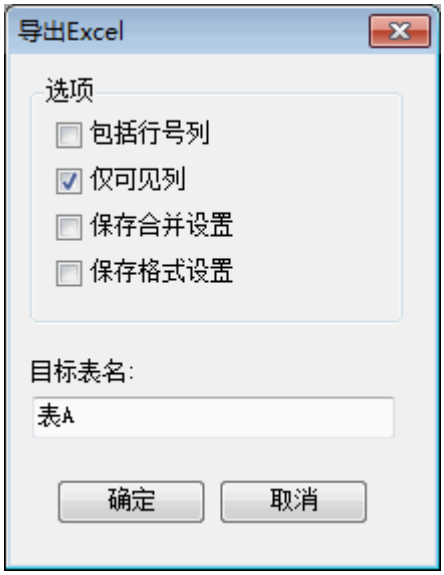
3.15.1.2 导出 Excel

1、在“杂项”功能区的“导出”功能组，单击按钮“Excel 文件”：



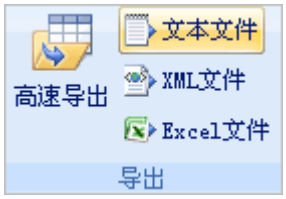
2、出现一个保存文件的对话框，输入或选择要接受导出数据的 Excel 文件。

3、其实高速导出也能导出为 Excel 文件，但是专用的 Excel 导出窗口提供了更专业的选项：



3.15.1.3 导出为文本文件

1、在“管理项目”功能区的“导出”功能组，单击按钮“文本文件”：

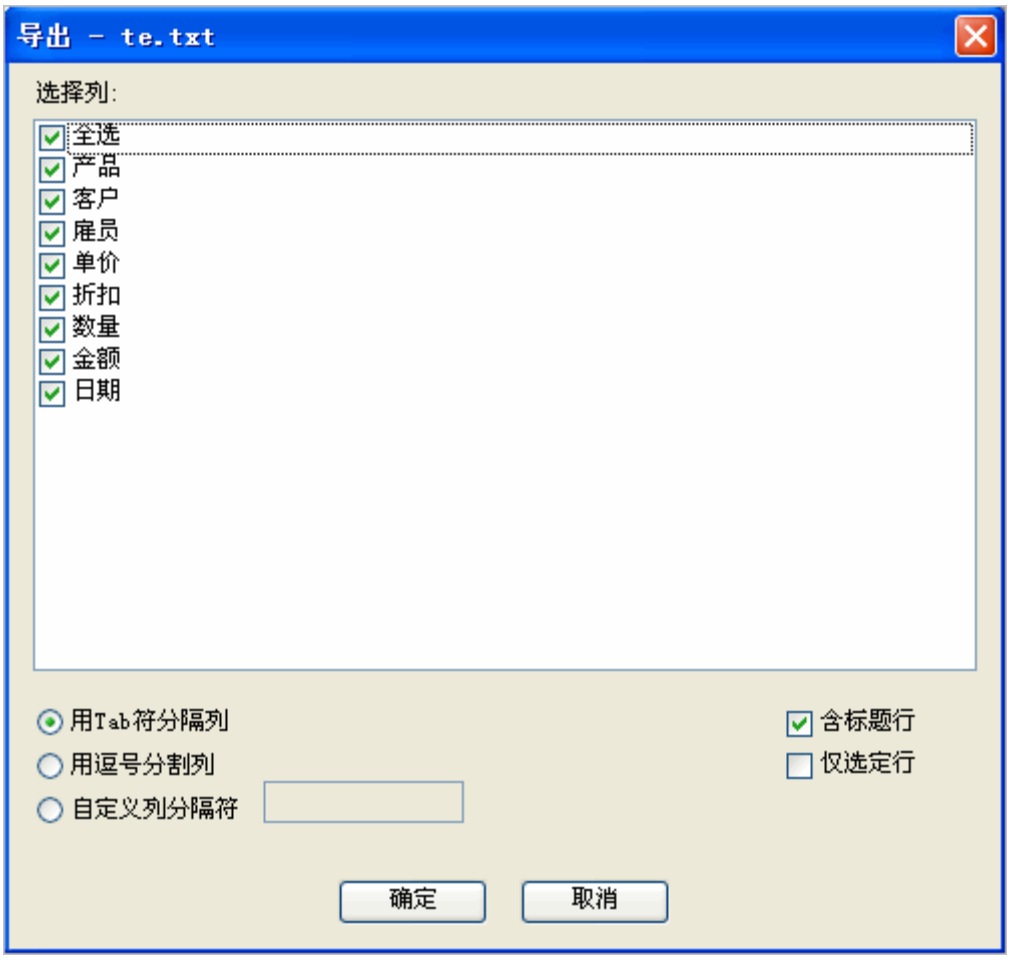


2、出现一个保存文件的对话框，选择或输入要导出的文本文件名。

3、在导出设置窗口，选择列分隔符，或者输入自定义列分隔符。

4、如果只希望导出选定的行，选择选项“仅选定行”即可。

5、导出数据默认包括列标题，如果只希望导出数据，请不要选择选项“含标题行”。



6、单击“确定”按钮。

3.15.2 导入

3.15.2.1 高速导入

高速导入能够导入 Access、dBase、Excel 文件。

1、在“杂项”功能区的“导入”功能组，单击按钮“高速导入”。



2、出现一个打开文件对话框，选择要导入的文件类型和文件。

4、在导入设置窗口，选择要导入的表和列，单击“确定”按钮。



5、输入表名和标题，单击“确定”按钮。

3.15.2.2 导入文本文件

1、在“杂项”功能区的“导入”功能组，单击按钮“文本文件”。



2、出现一个打开文件对话框，选择要导入的文本文件

3、在导入设置窗口，选择或输入自定义列分隔符：



4、删除不需要导入的列。

5、如果文本文件的首行是列标题，请选择选项“首行为标题”。

6、单击“确定”按钮。

3.15.2.3 导入其它类型

普通用户请跳过此节

1、在“杂项”功能区的“导入”功能组，单击按钮“其它类型”。

2、出现“数据链接属性”设置窗口，链接到要导入数据所在的数据源。

3、在导入设置窗口，选择要导入的表。

4、删除不需要导入的列。

5、单击“确定”按钮。

利用这个功能，可以导入 SQL Server、Oracle 中的数据。

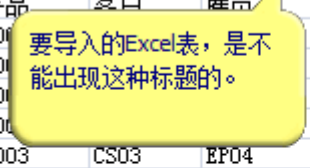
3.15.2.4 排除不支持的数据类型

Foxtable 不支持一些特殊的列，例如 Access 的超链接列和附件列，在导入的时候必须排除之，否则将导入失败，提示错误通常为“异常来自 HRESULT:0x800A0CB3”

3.15.2.5 Excel 导入的一个常见问题

经常会有客户问我们，为什么他的 Excel 表无法正常导入到 Foxtable。
大多数时候，对于因为其 Excel 表不够规范，出现下图所示的表标题所致：

	A	B	C	D	E	F	G
1			订单表				
2	产品	客户	雇员	单价	折扣	数量	日期
3	PD			17	0.1	650	1999/1/2
4	PD			20	0.1	400	1999/1/3
5	PD			17	0	320	1999/1/3
6	PD			18	0.15	80	1999/1/4
7	PD03	CS03	EP04	10	0	490	1999/1/4
8	PD04	CS04	EP05	17.6	0	240	1999/1/7
9	PD05	CS03	EP04	17	0	111	1999/1/7
10	PD01	CS04	EP05	14.4	0	200	1999/1/8
11	PD03	CS02	EP02	8	0.1	200	1999/1/8
12	PD01	CS02	EP01	18	0.2	800	1999/1/10
13	PD01	CS04	EP02	14.4	0.05	500	1999/1/10
14	PD03	CS01	EP04	8	0	300	1999/1/11



Foxtable 作为一个数据库软件，只能导入标准的表格：

	A	B	C	D	E	F	G
1	产品	客户	雇员	单价	折扣	数量	日期
2	PD05	CS03	EP04	17	0.1	650	1999/1/2
3	PD02	CS01	EP01	20	0.1	400	1999/1/3
4	PD05	CS04	EP02	17	0	320	1999/1/3
5	PD01	CS03	EP04	18	0.15	80	1999/1/4
6	PD03	CS03	EP04	10	0	490	1999/1/4
7	PD04	CS04	EP05	17.6	0	240	1999/1/7
8	PD05	CS03	EP04	17	0	111	1999/1/7
9	PD01	CS04	EP05	14.4	0	200	1999/1/8
10	PD03	CS02	EP02	8	0.1	200	1999/1/8
11	PD01	CS02	EP01	18	0.2	800	1999/1/10
12	PD01	CS04	EP02	14.4	0.05	500	1999/1/10

3.15.3 合并

3.15.3.1 高速合并

高速合并能够合并 Access、dBase、Excel 文件到当前表中。

基本操作

1、在“杂项”功能区的“合并”功能组，单击按钮“高速合并”。



2、出现一个打开文件对话框，选择要合并的文件类型和文件。
3、在合并设置窗口，选择要合并的数据表。



4、单击“确定”按钮。

合并方式

- 更新同主键记录
  合并表中的每一行数据，在合并到当前表中之前，都会在当前表中查找相同主键的行；
  如果没有找到相同主键的行，则直接将该行复制到当前表中。
  如果找到相同主键的行，则用该行内容更新所找的行。

- 跳过同主键记录
  合并表中的每一行数据，在合并到当前表中之前，都会在当前表中查找相同主键的行；
  如果找到相同主键的行，则跳过该行。
  如果没有找到相同主键的行，则将该行复制到当前表中。

- 合并过程不比较主键
  合并表中的行，将无条件地复制到当前表中。

提示：合并过程会比较列名，合并表中的某一列必须在当前表中有对应的同名列，才会合并，否则会忽略此列。

3.15.3.2 合并文本文件

1、在“杂项”功能区的“合并”功能组，单击按钮“文本文件”。
2、出现一个打开文件对话框，选择要合并的文本文件。

3、在合并设置窗口，选择或输入自定义列分隔符。
4、选择合并模式。
5、单击“确定”按钮。

3.15.3.3 合并其他类型

普通用户请跳过此节。
1、在“杂项”功能区的“合并”功能组，单击按钮“其它类型”。
2、出现“数据链接属性”设置窗口，链接到要合并数据所在的数据源。
3、在合并设置窗口，选择要合并的表。
4、选择合并模式。
5、单击“确定”按钮。
利用这个功能，可以合并 SQL Server、Oracle 中的数据。

3.16 用户管理

3.16.1 用户管理

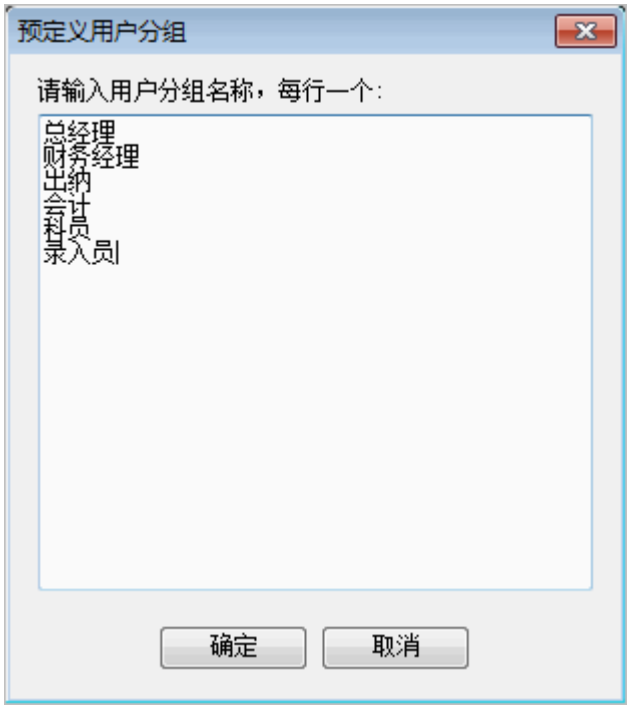
在默认情况下，我们不需要输入密码，即可直接打开某个项目。
如果需要使用密码保护项目，只需在用户管理中为此项目增加一个用户，或修改一下现有用户的属性(如用户名、密码等)即可。
以下是用户管理的简单说明。
1、在菜单的“管理项目”功能区，单击按钮“用户管理”：



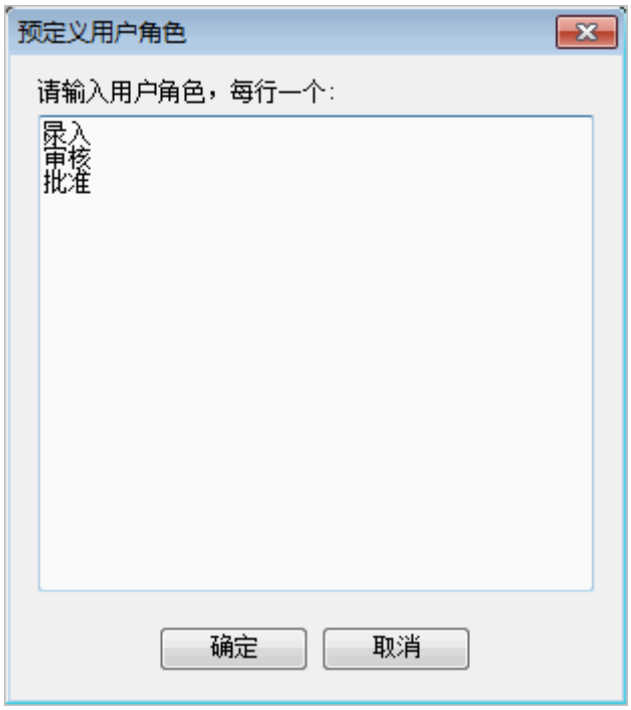
2、出现用户管理窗口，默认已经有两个用户，分别是开发者和管理员：



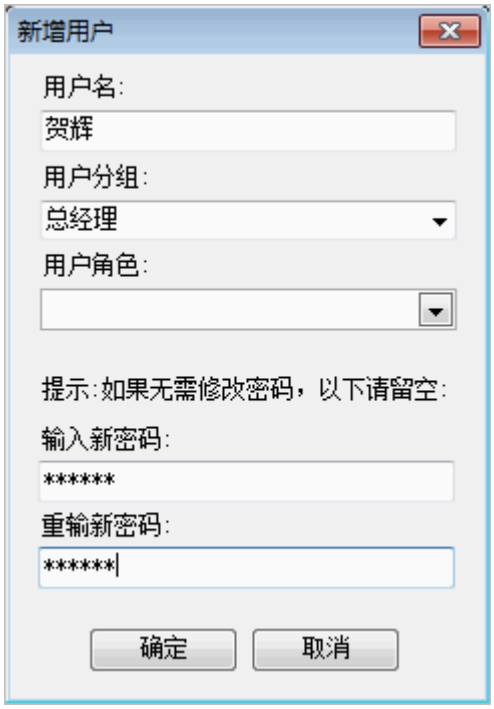
3、单击“组定义”按钮，可以预先定义好一些用户分组：



4、单击“角色定义”，可以预先定义好一些用户角色：



5、单击“增加”按钮，出现增加用户窗口，一个用户包括用户名、用户分组，用户角色、登录密码四个信息：



5、当然，你还可以删除、修改现有用户。

提示：

用户分组和用户角色不是必须的，一个用户只能属于一个分组，但可以是多个角色。

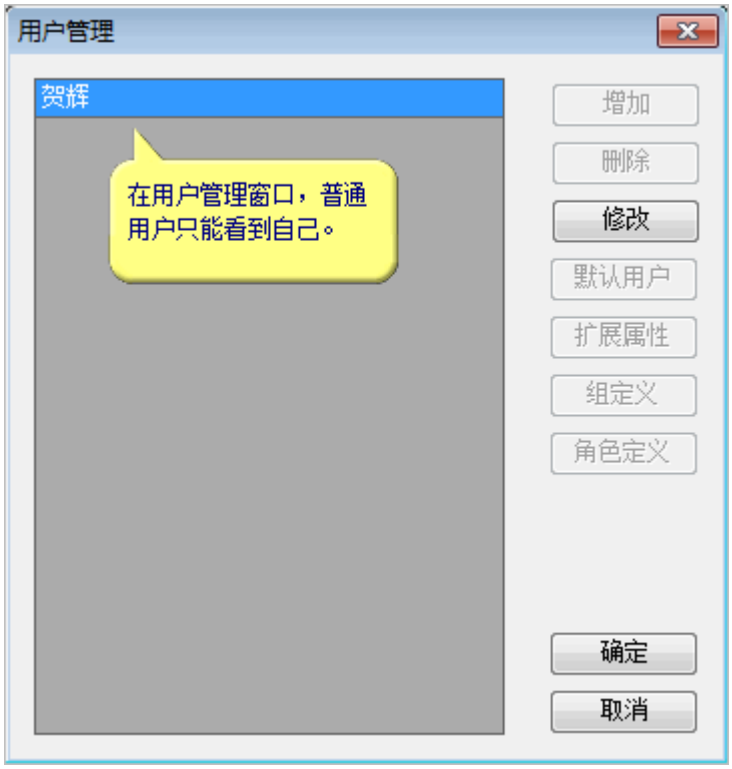
3.16.2 用户级别

我们知道系统默认已经有两个用户，分别是开发者和管理员，这两个用户是不能被删除的，但可以被重命名。

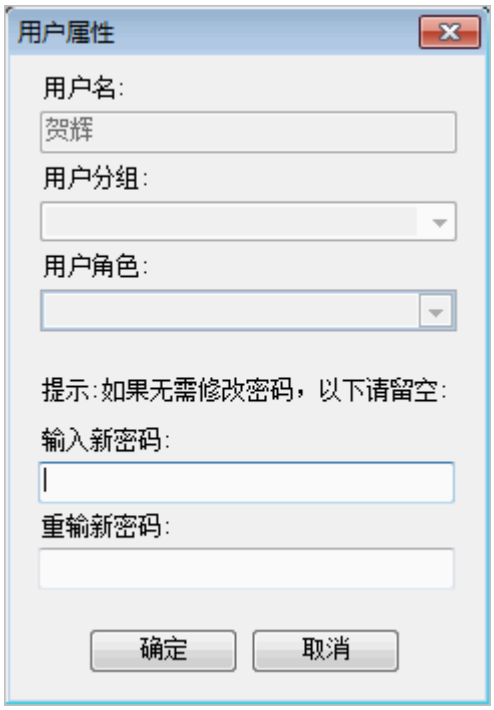
开发者是就项目的开发人员，拥有所有的权限，是最高级别的用户。
管理员的级别仅次于开发者，在项目启用后，可以代替开发者行使用户管理的权限。
以管理员身份登录项目，进入用户管理窗口后，会列出除开发者之外的所有用户，显然管理员是不能管理开发者的：



如果以普通用户登录，进入用户管理窗口，将只能看到自己，而且增加、删除、组定义、角色定义等按钮是无效的：

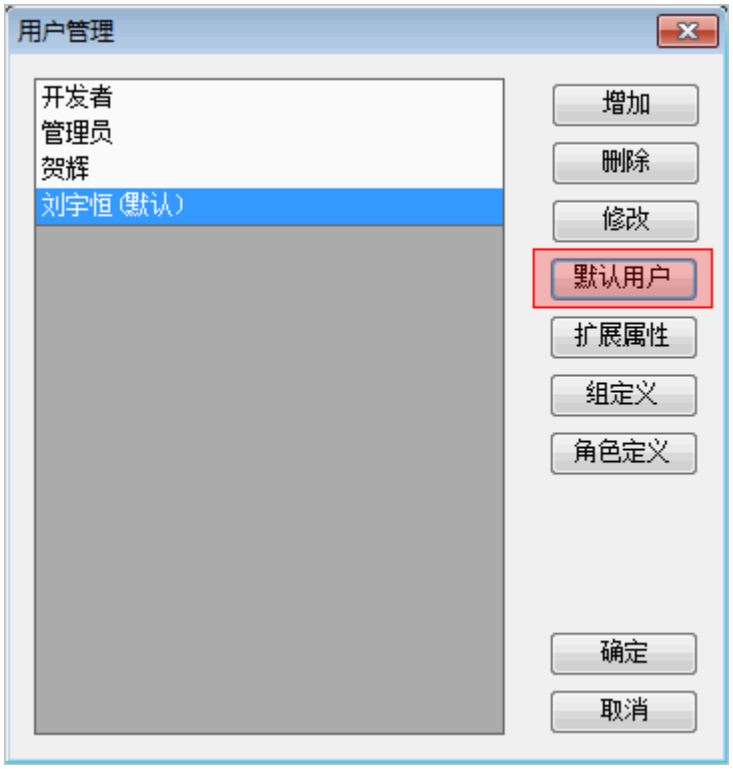


普通用户单击修改按钮后，只能修改密码，不能修改自己的用户名、分组和角色：



3.16.3 默认用户

单击“默认用户”按钮，可以将选定用户设为默认用户：



一旦设置了默认用户，打开项目的时候，将不再出现用户登录窗口，而是直接以默认用户身份登录。如果在打开项目的时候，始终按住 Ctrl 键，将正常出现登录窗口。

提示：再次单击“默认用户”按钮，可以取消默认用户设置。

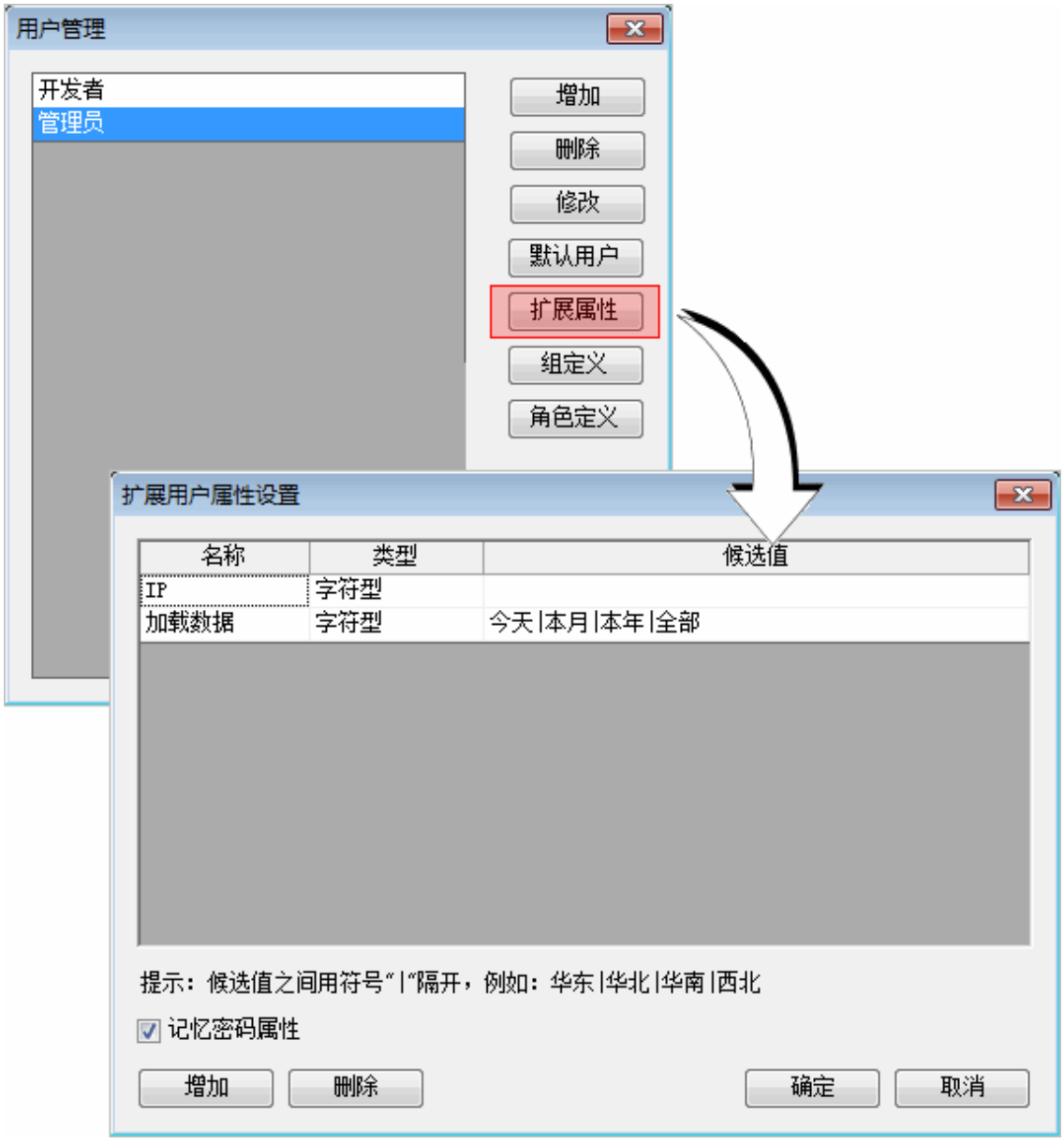
3.16.4 切换用户

运行过程中，如果想切换用户，在菜单的“管理项目”功能区，单击下面的按钮即可：

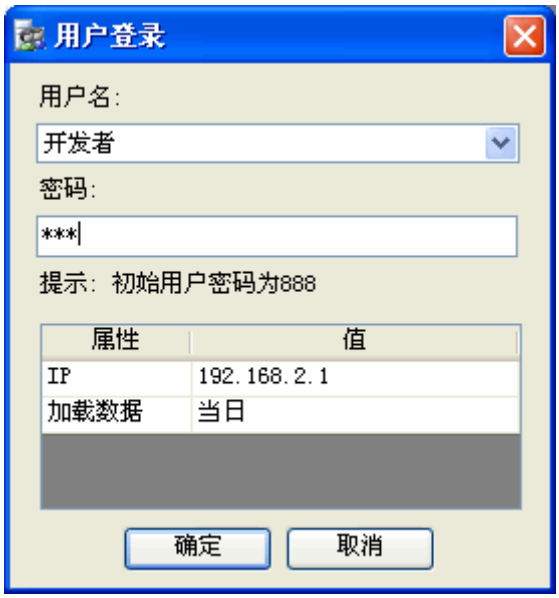


3.16.5 扩展用户属性

在用户管理窗口，单击“扩展属性”按钮，即可为用户增加额外的属性。



如果设置了扩展用户属性，在用户登录窗口，就会出现一个表格，用于输入扩展属性：



显然，扩展用户属性是用于开发的，例如让程序根据用户登录时输入的扩展属性，从不同的数据源加载不同条件的数据，我们会在开发篇讲述扩展用户属性的应用。

3.16.6 关于权限管理

在项目属性设置中，有一个“关闭开发功能”的属性，将其设为 True(是)，将禁止除开发者之外的用户使用开发功能。

除此之外，项目属性、表属性、列属性，以及用户管理中，都没有设置用户权限的选项，那么 Foxtable 如何来进行权限管理呢？

原理 很简单，因为用户进行任何的操作都会触发事件，例如用户在编辑单元格之前，会触发一个 PrepareEdit 事件，你可以在这个事件中进行复杂的逻辑判断，如果不符合特定的条件，则取消这次编辑操作。这种在运行过程中发生的权限判断，比任何预先的、固定的权限设置都要灵活强大。

在开发篇我们会接触到很多这样的例子，目前你只需大概了解权限管理的原理即可。

3.17 网络与数据源

3.17.1 关于数据源

在默认情况下，我们新建的数据表位于项目文件中，项目文件就是内部数据源，项目文件中的数据表，我们称之为内部数据表 ， 如果是单机使用，内部数据源是一个很好的选择。

如果要在网络环境下协同工作，应该考虑使用外部数据源，外部数据源可以是 Access、SQL Server 或 Oracle，来自于外部数据源的数据表，我们称之为外部数据表。

如果是局域网，可以用 Access 或者 SQl Server 作为数据源，如果是互联网，应该使用 SQL Server 作为数据源(注：高级开发版可以在互联网环境下使用 Access 数据源)。

对于 Foxtable 来说，不管你的环境单机、局域网还是互联网，使用和设计上并没有什么差别，只是数据库在不同的位置而已。

快速入门

以 Access 数据源为例，基于 Foxtable 搭建一个局域网管理系统的步骤为：

1、新建一个 Access 文件，参考：新建 Access 文件。

2、将此 Access 文件或原有的 Access 文件复制到服务器上的某个共享文件夹中，注意此共享文件夹必须开启读写权限。

3、新建一个数据源，连接到服务器上的 Access 文件，参考：连接 Access2003 文件

4、现在你可以直接在 Foxtable 中为此 Access 文件新建数据表，参考：新建外部数据表

5、也可以将此 Access 文件中原有的数据表添加到 Foxtable 中，参考：添加外部数据表

6、接下来就可以进行下一步的开发工作，使用和开发和内部表一样，不会有任何差别。

上面只是一个大概，接下来我们将详细讲述如何连接到外部数据源，并将外部数据源中的表加载到 Foxtable 中进行处理。

3.17.2 关于主键

在讲述如何连接到外部数据源，如何加载外部数据表之前，我们有必要了解一下主键的概念。

对于数据管理来说，主键是一个非常重要的概念。

我们每个人都有一个身份证号码，每个身份证号码都是唯一的，在正常情况下，不会再有其他人的身份证号码与你相同，这个号码就是我们的“身份标识”，是社会生活能有序进行的基础。

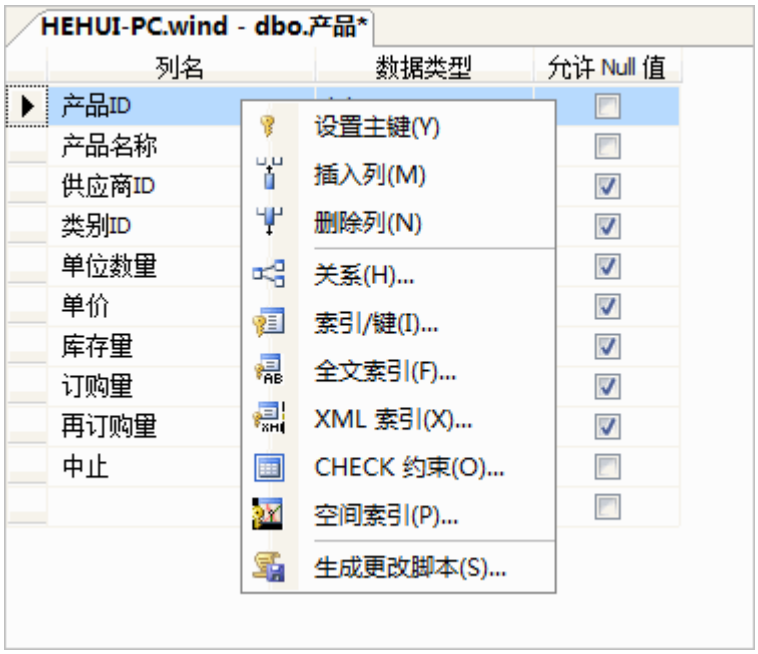
同样数据表中的每一行，都有一个主键值，每个主键值都是唯一的，不同的行，其主键值绝对不会相同，所以行的主键值类似人的身份证号码，是行的“身份标识”，是数据管理能有序进行的基础。

在设计表的时候，应该指定一列用于存储行的主键值，这一列就是主键列。

在 Foxtable 新建的数据表，不管是内部数据表还是外部数据表，都有一个名为“_Identify”的列，这就是主键列，每增加一行，该列的值就会自动在上一行的基础上加 1，所以对于每一行来说，_Identify 列的值都不会相同，是有效的“身份标识”。

如果不是在 Foxtable 中，而是在外部数据源中设计表，例如 Access 或者 SQl Server 中建表，那么必须手工设置主键列。

在 Access 或者 SQL Server 的表设计器中， 右击要设置为主键的列，从弹出的快捷菜单中执行"设置主键"命令，即可将此列设置为主键：

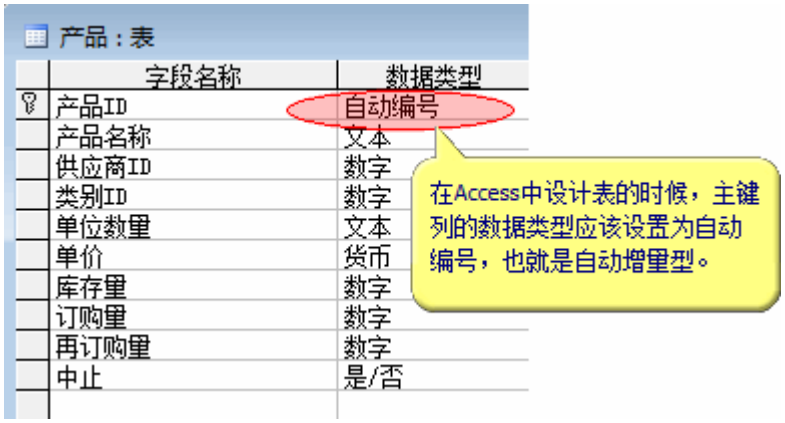


关于外部表的主键，我们建议：尽量使用自动增量列（也称为自动编号），由系统自动生成编号。如果主键为自动增量列，那么外部表和内部表在实际使用的时候，表现将完全一样，用户将感受不到任何的差别。

如果主键列不是自动增量列，则每次只能增加一行，且必须输入新增行的主键列内容才能增加下一行。

此外，如果你希望外部表的主键列能够象内部表一样自动隐藏，可以将主键列的名称设置为“_Identify”。

在 Access 中设置自动编号：



在 SQL Server 中设置自动编号：

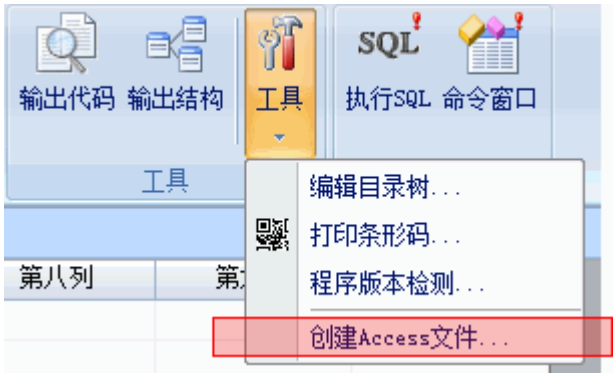


3.17.3 连接外部数据源

3.17.3.1 创建 Access 文件

使用 Access 作为数据源，并不需要在服务器或本机安装 Access，你可以直接将 Access 文件复制到服务器，然后在 Foxtable 中新建一个数据源，连接到此 Access 文件即可。

Foxtable 本身也提供了一个创建 Access 文件的命令，位于菜单的“杂项”功能区：

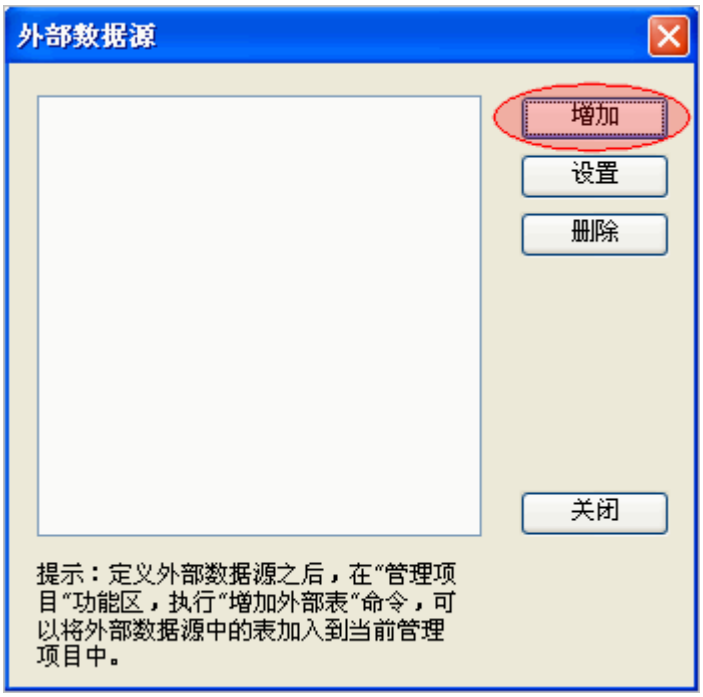


3.17.3.2 连接到 Access 2000/2003

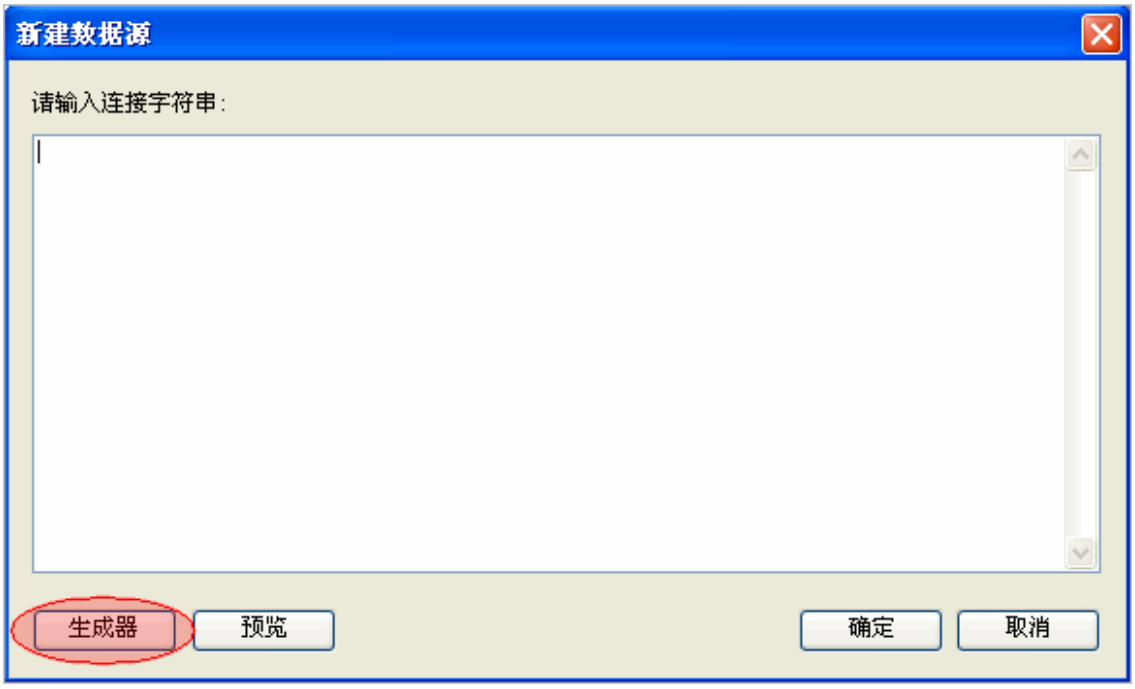
1、在数据表功能区单击下面的按钮：



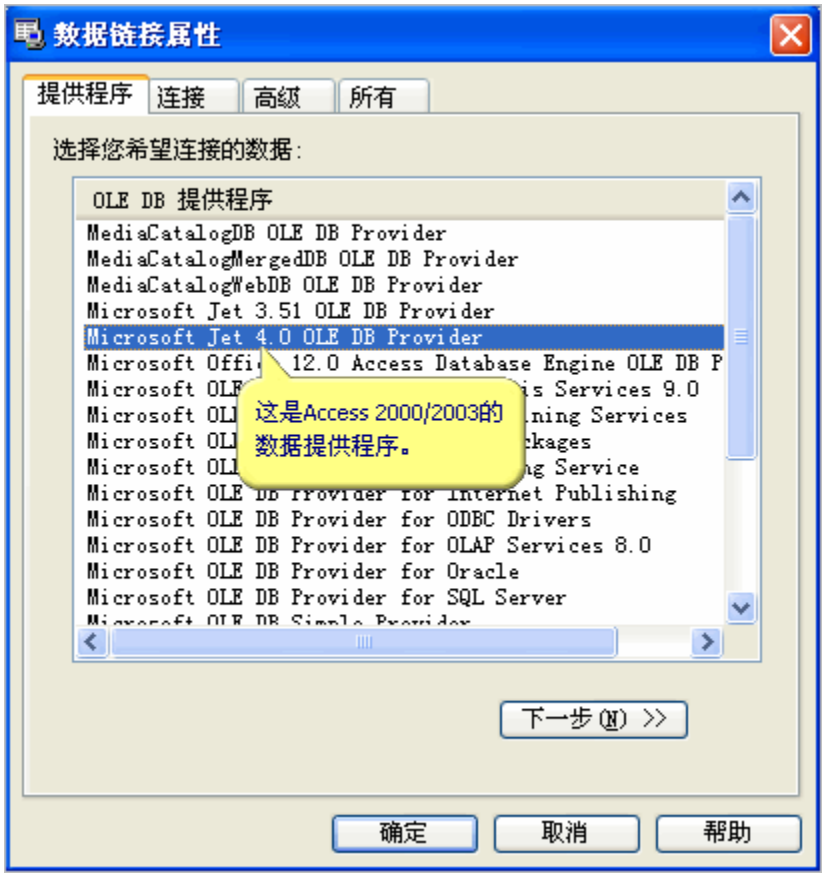
2、在外部数据源管理窗口，单击“增加”按钮：



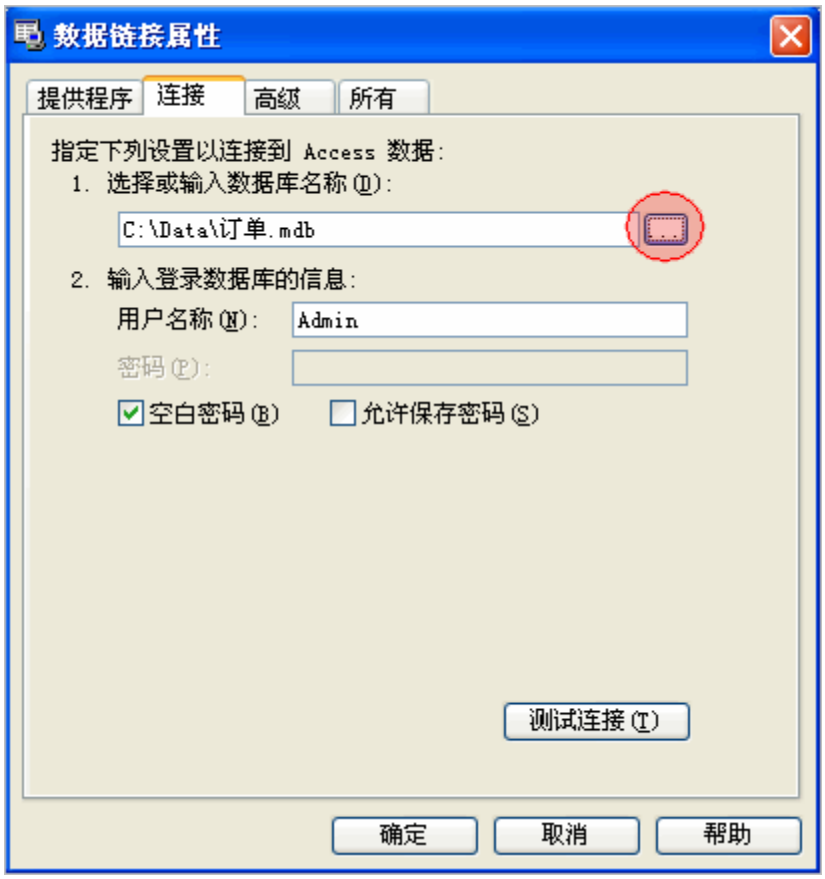
3、在新建数据源窗口单击按钮“生成器”：



4、在数据链接属性设置窗口，选择下图所示的数据提供程序：



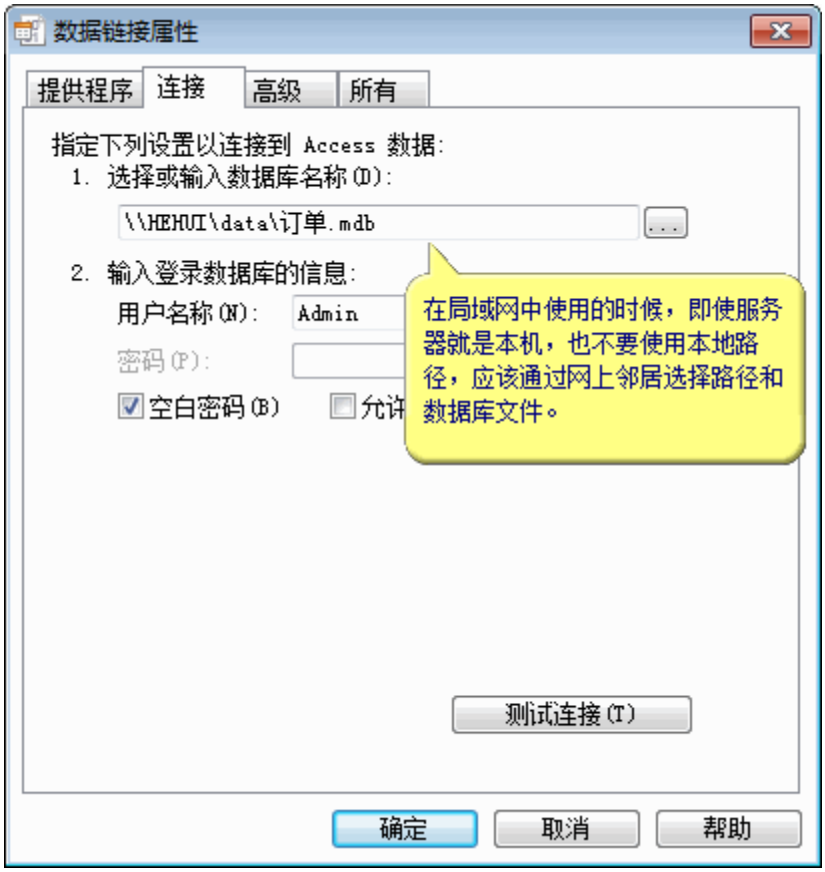
5、然后单击下面的按钮，选择要连接的 Access 文件:



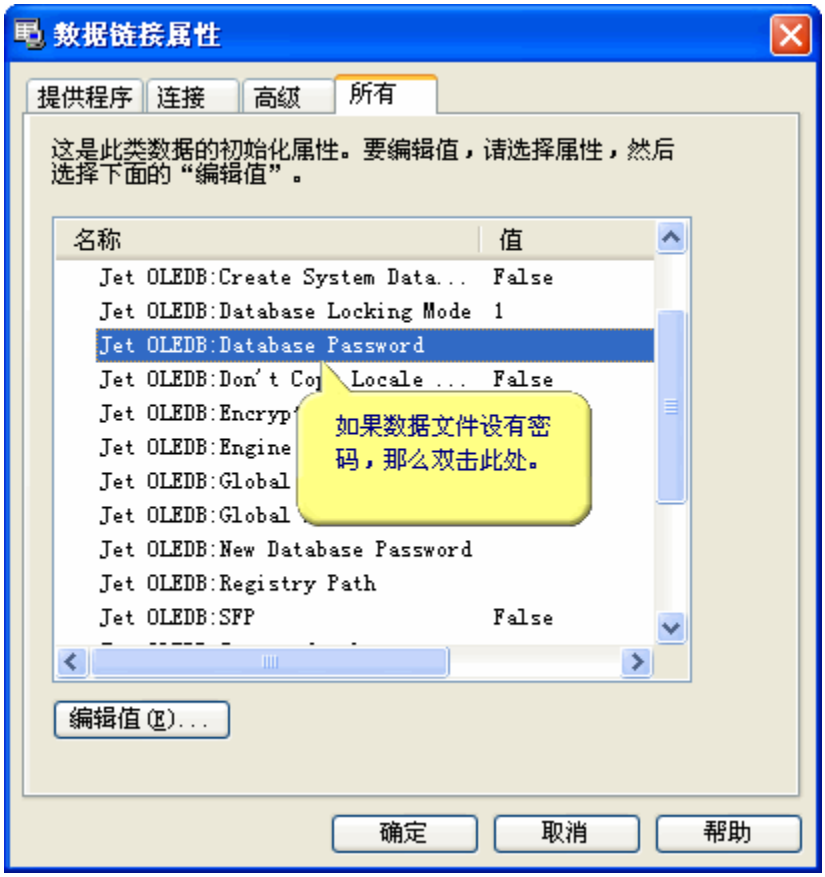
在局域网中使用的时候，即使服务器就是本机，也就是数据库文件在本机，也不要使用本地路径，

因为如果使用"c:\data"这样的路径，项目复制到其他电脑使用的时候，将无法连接数据库文件，因为其他电脑的"c:\data"，肯定不是服务器上的"c:\data"。

所以，对于一个准备在局域网使用的项目，不管服务器是否是本机，都应该通过网上邻居来选择路径和数据库文件：

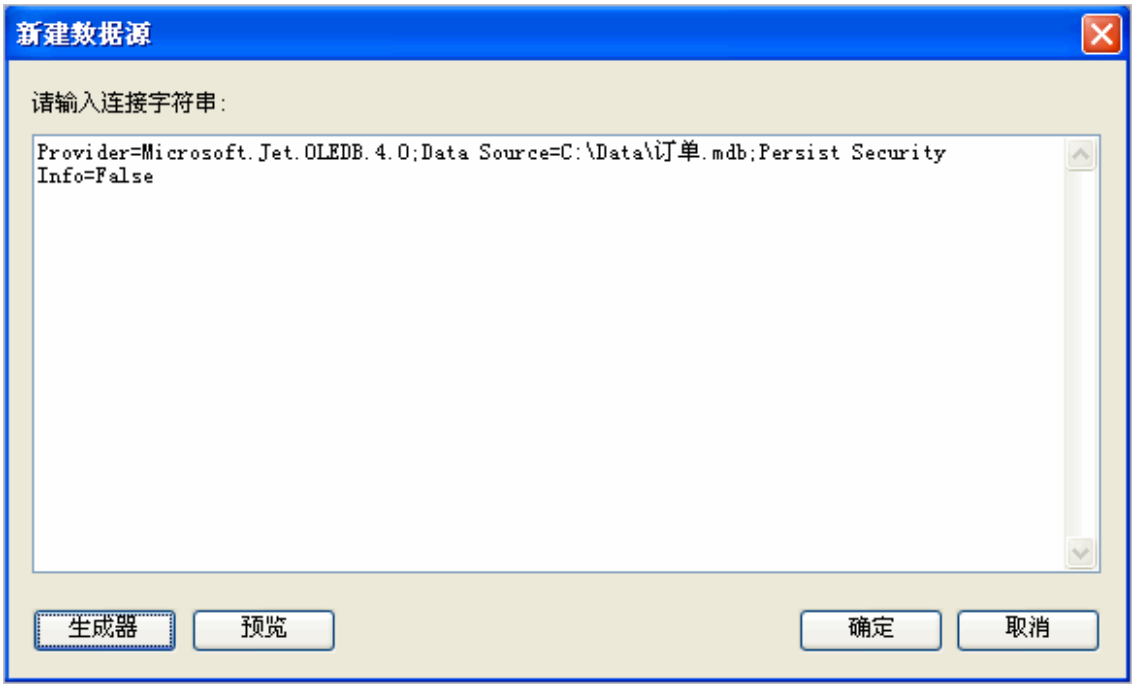


6、如果数据文件设置有密码，那么按下图所示操作：





7、设置完成后，单击“确定”按钮，即可看到生成的链接字符串：



8、单击“确定”按钮，输入数据源名称，完成设置。

3.17.3.3 连接到 Access 2007/2010

首先，如果你没有安装 Access 2007/2010,但需要通过 Foxtable 访问 Access 2007\2010 的文件，可以下载安装下面的数据访问组件:

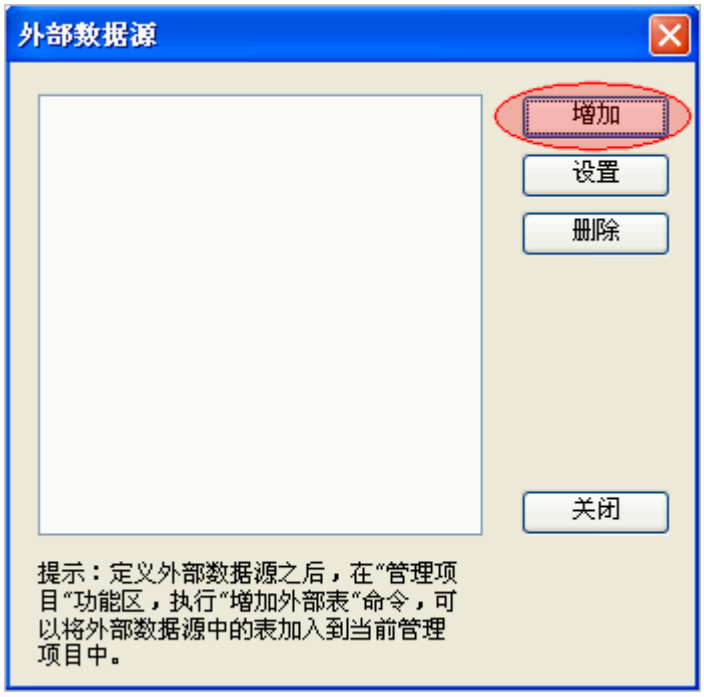
http://www.foxtable.com/download/AccessDatabaseEngine.exe

连接到 Access 2007/2010 的步骤：

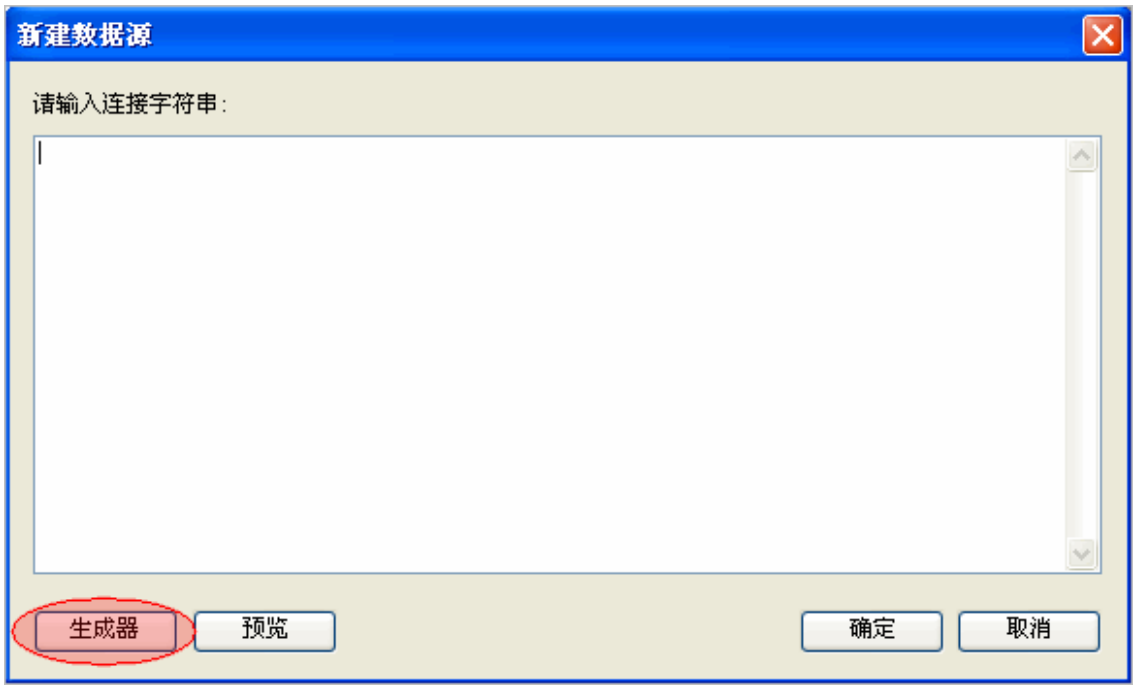
1、在数据表功能区单击下面的按钮：



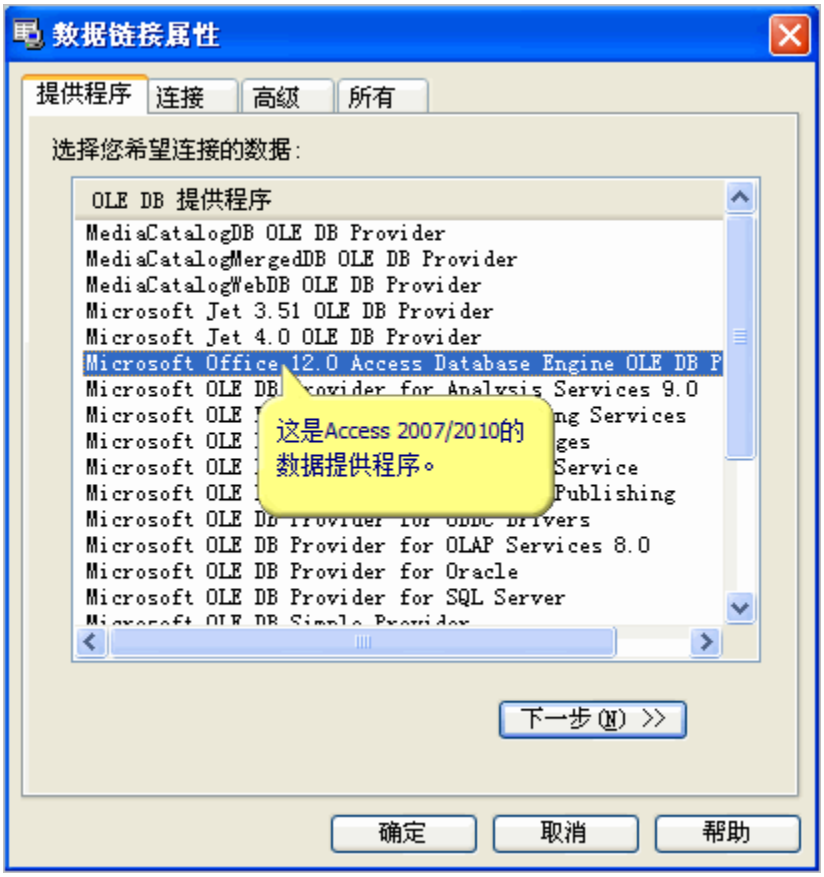
2、在外部数据源管理窗口，单击“增加”按钮：



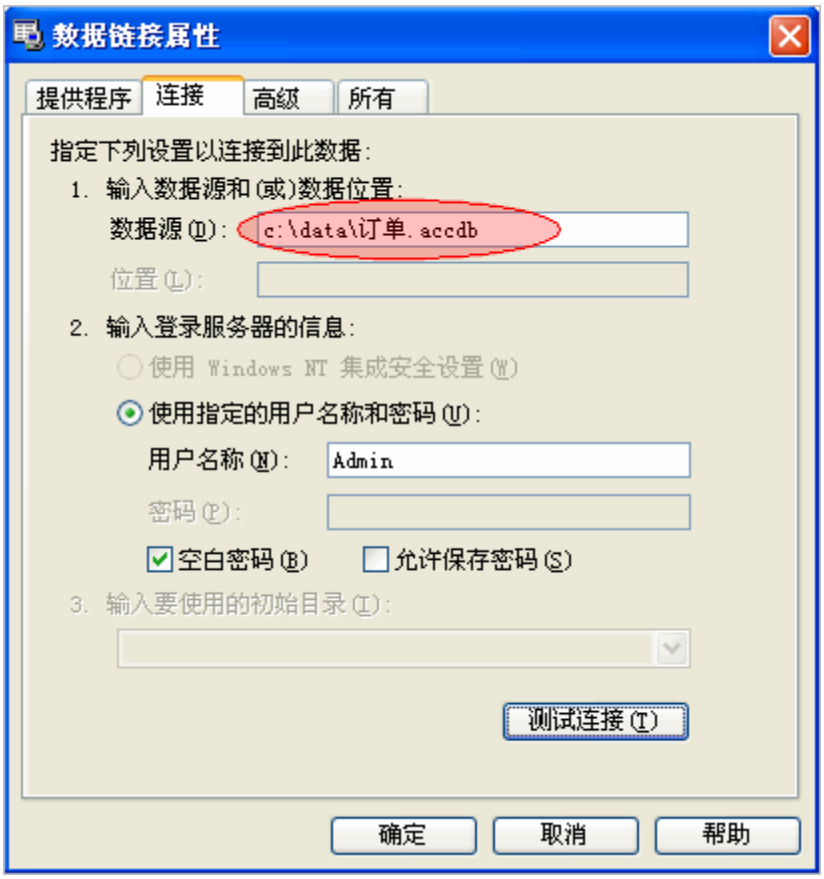
3、在新建数据源窗口单击按钮“生成器”：



4、在数据链接属性设置窗口，选择下图所示的数据提供程序：



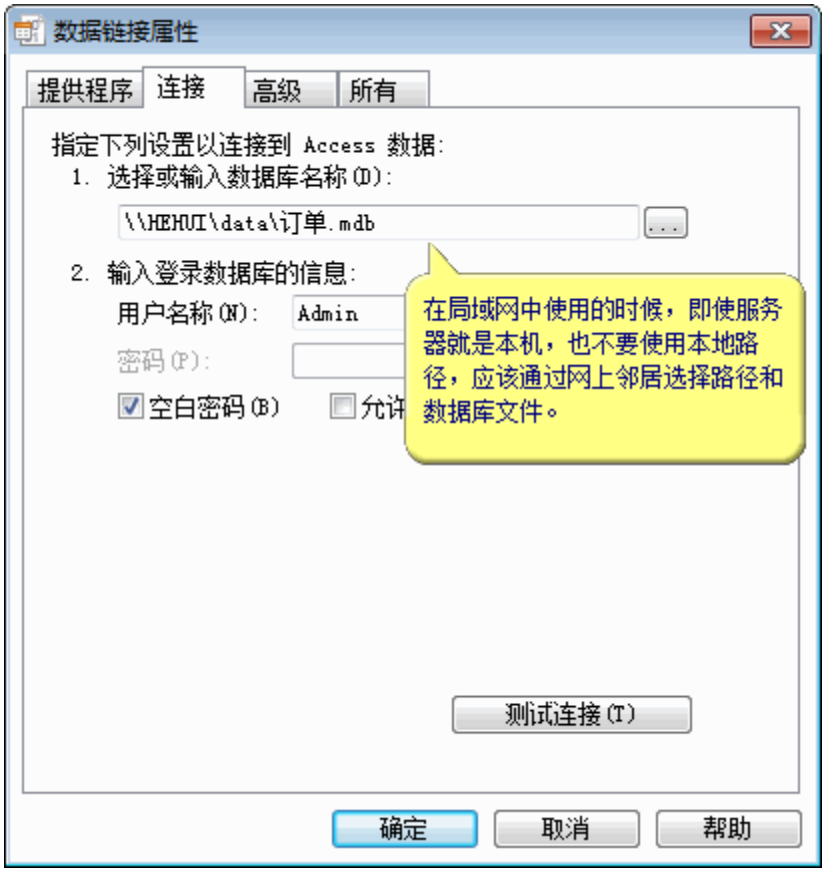
5、然后输入要连接的 Access 文件:



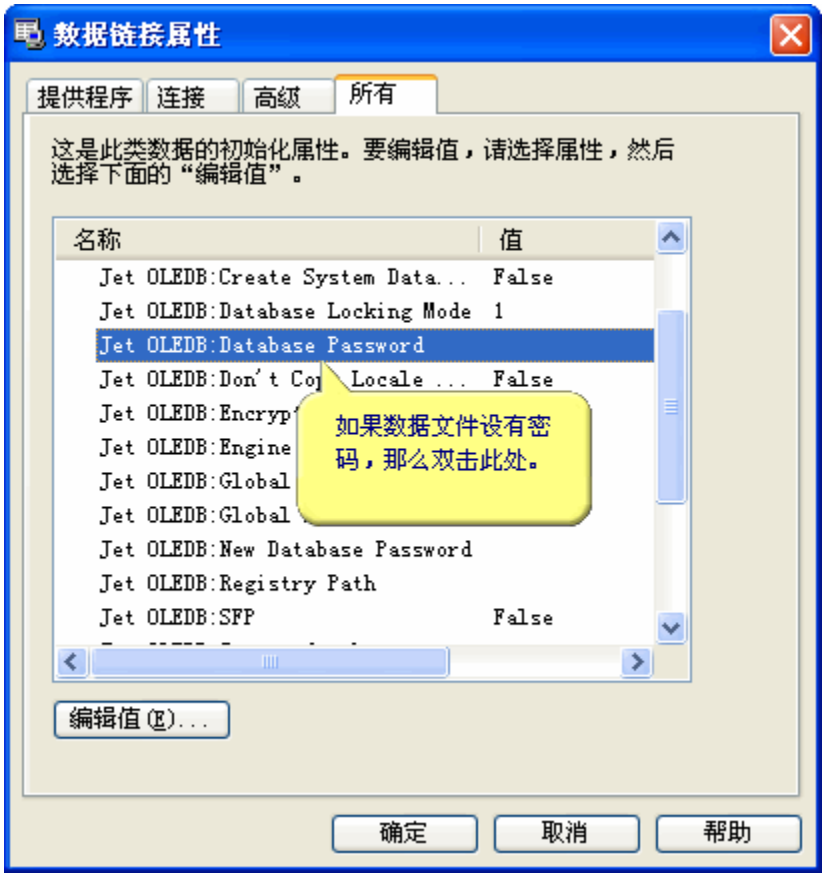
在局域网中使用的时候，即使服务器就是本机，也就是数据库文件在本机，也不要使用本地路径，

因为如果使用"c:\data"这样的路径，项目复制到其他电脑使用的时候，将无法连接数据库文件，因为其他电脑的"c:\data"，肯定不是服务器上的"c:\data"。

所以，对于一个准备在局域网使用的项目，不管服务器是否是本机，都应该通过网上邻居来选择路径和数据库文件：

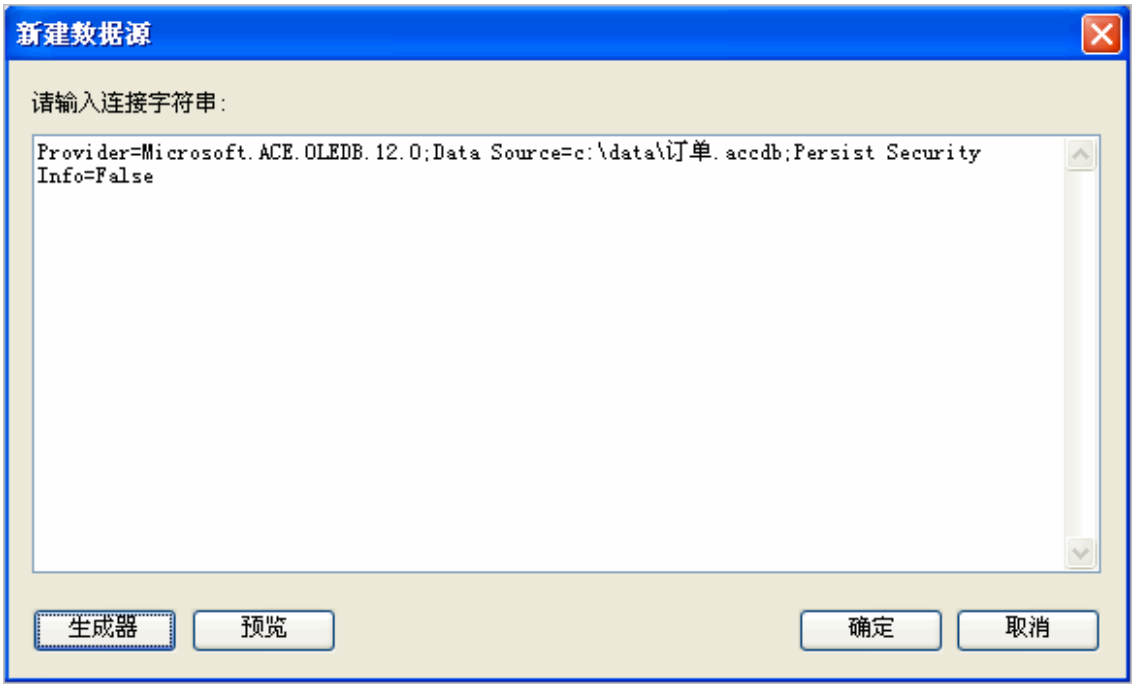


6、如果数据文件设置有密码，那么按下图所示操作：





7、设置完成后，单击“确定”按钮，即可看到生成的链接字符串：



8、单击“确定”按钮，输入数据源名称，完成设置。

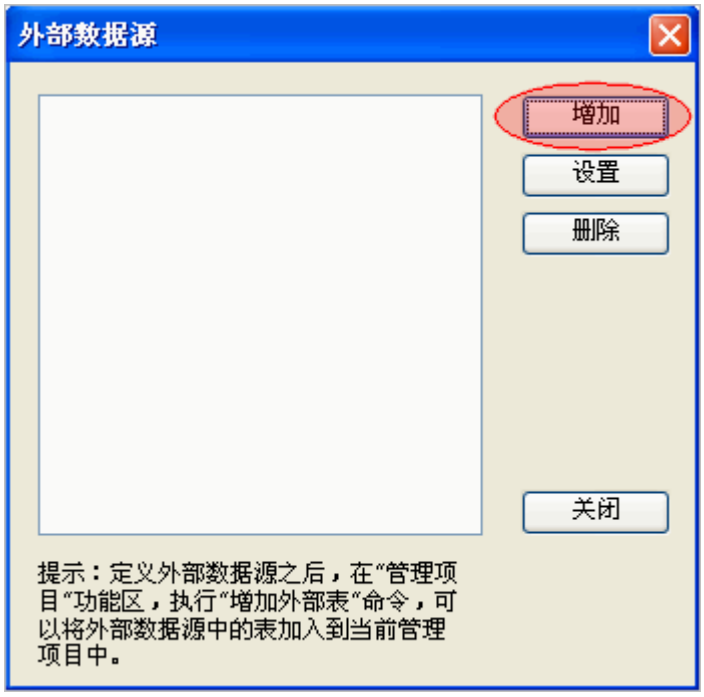
3.17.3.4 连接到 SQL Server 2000/2005

Foxtable 直接支持 SQL Server 2000/2005 或更早版本。

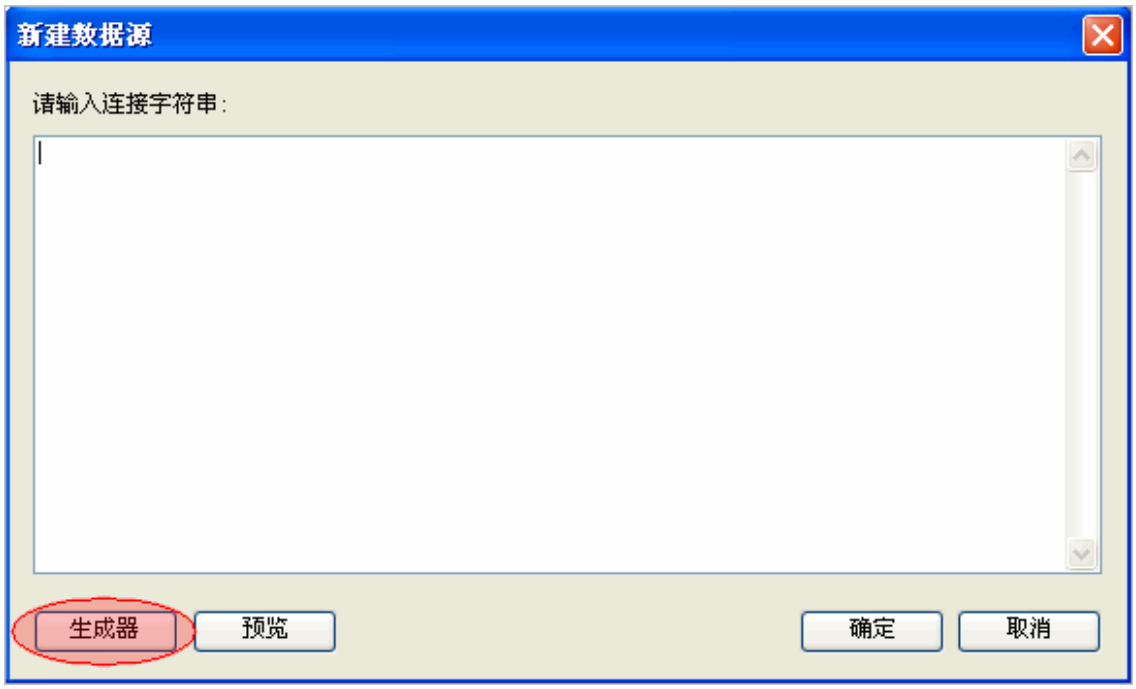
1、在数据表功能区单击下面的按钮：



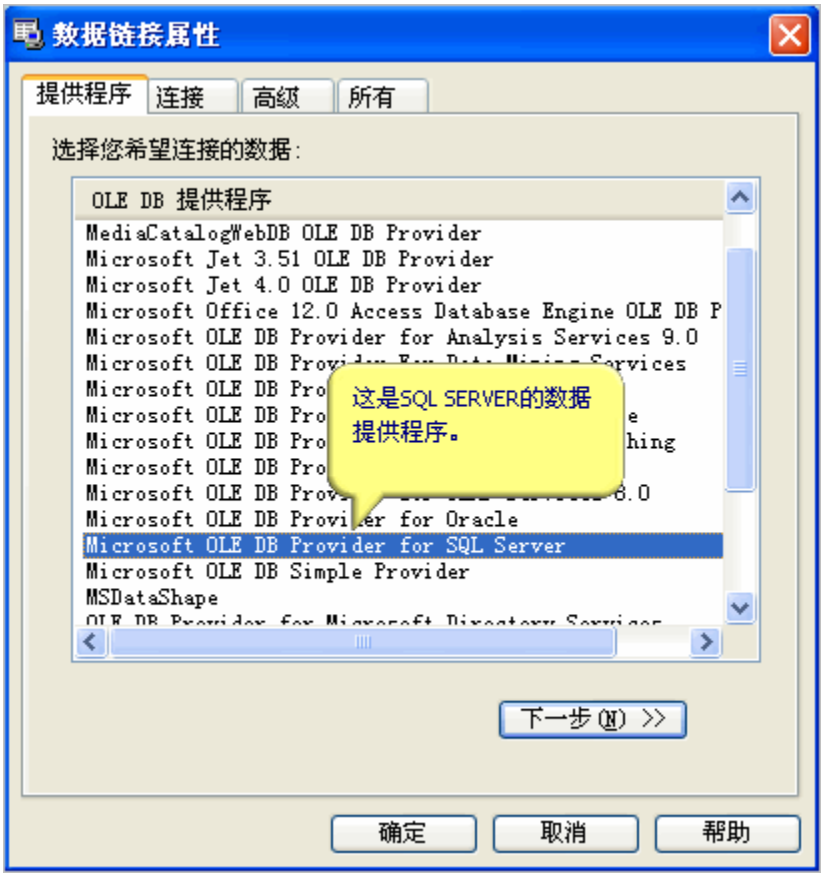
2、在外部数据源管理窗口，单击“增加”按钮：



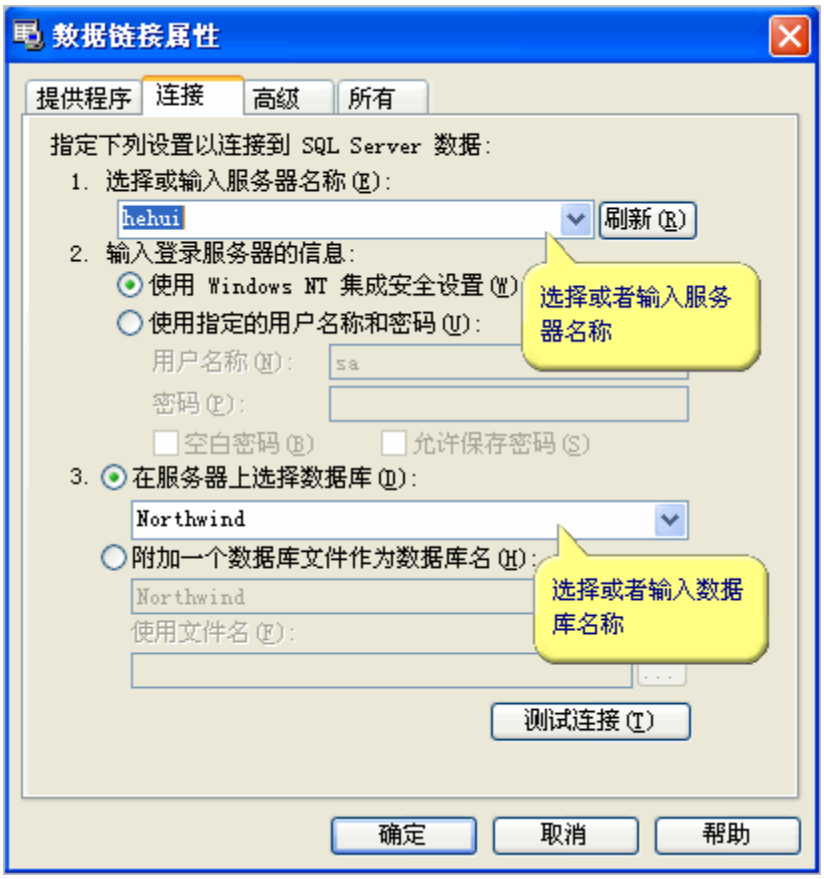
3、在新建数据源窗口单击按钮“生成器”：



4、在数据链接属性设置窗口，选择下图所示的数据提供程序：

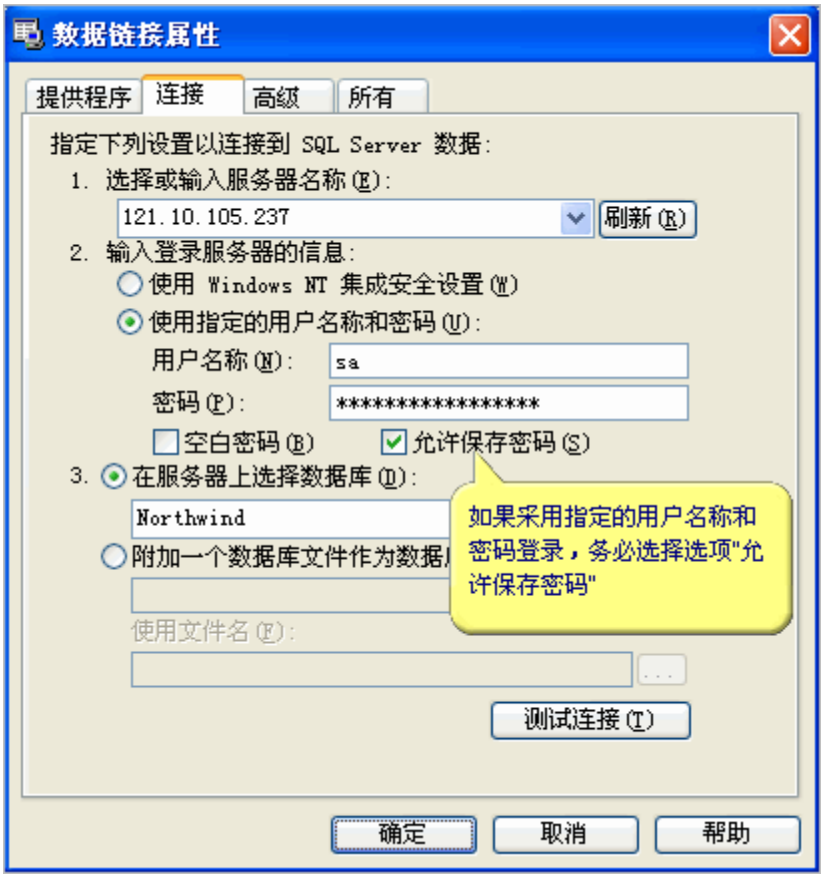


5、然后选择要连接的服务器数据库，SQL Server 有两种登录方式，下图采用的是“集成安全设置”：



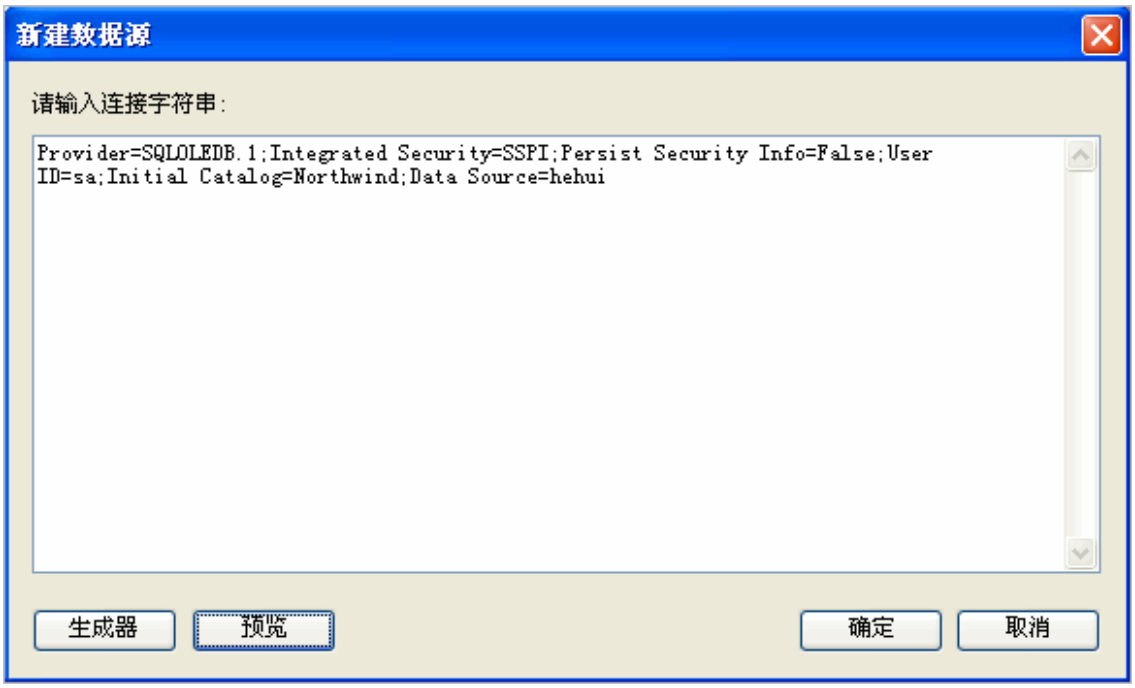
6、可以直接输入服务器的 IP 地址，如果采用指定的用户名称和密码登录，务必选择选项“允许保

存密码”：



提示：经常有用户问为什么测试连接可以，设置好之后却连接不上，原因在于没有选择这个“允许保存密码”的选项。

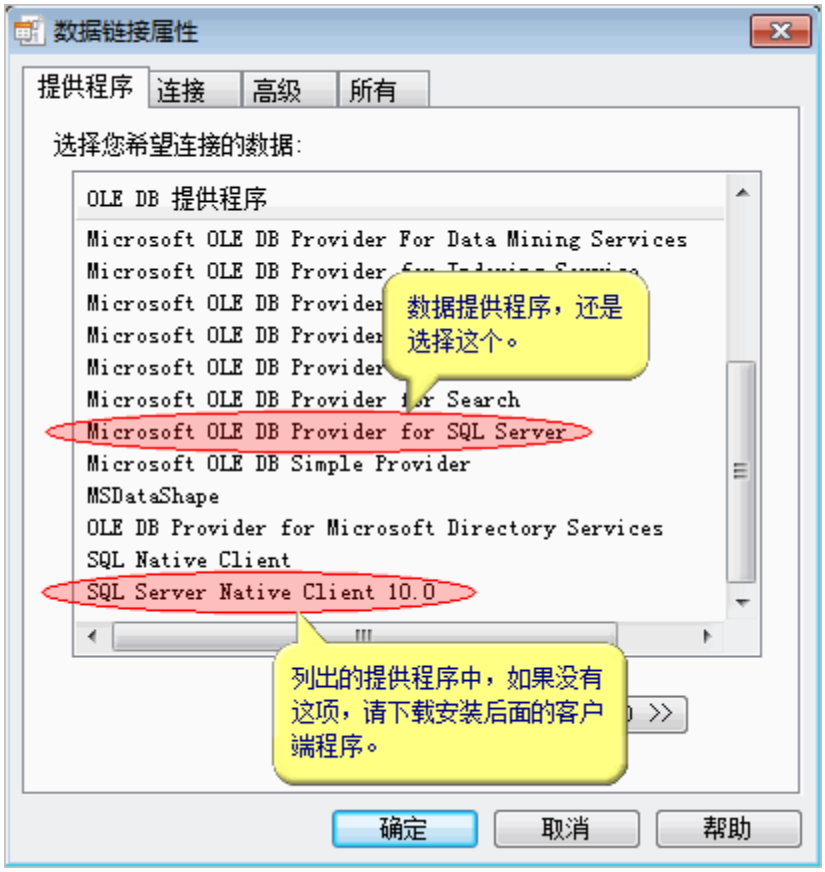
7、设置完成后，单击“确定”按钮，即可看到生成的链接字符串：



8、单击“确定”按钮，输入数据源名称，完成设置。

3.17.3.5 连接到 SQL Server 2008 或更新版本

如果要连接到 SQL Server 2008 或更新版本，请检查数据提供程序中是否有下图所示的"SQL Server Native Client 10.0"：



如果没有，请下载安装下面的客户端程序：

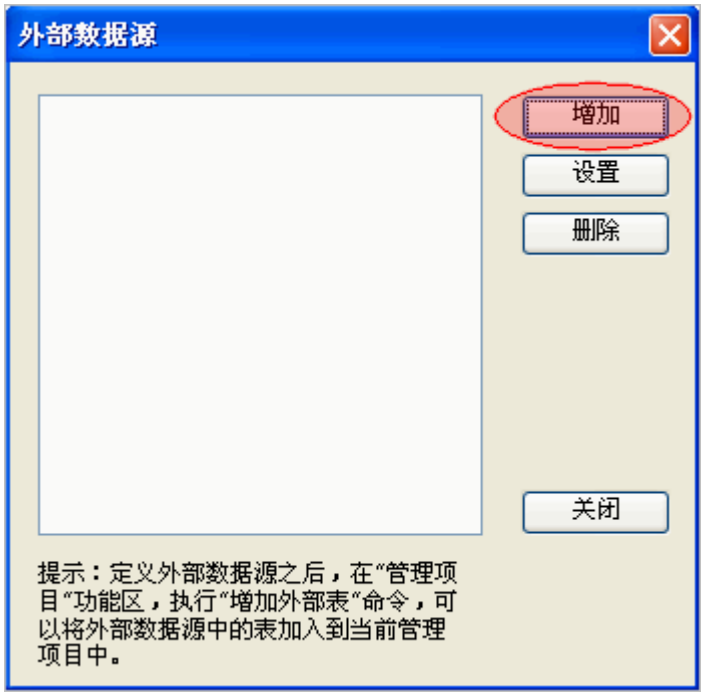
http://www.foxtable.com/download/sqlncli.msi

连接到 SQL Server 2008 或更新版本的步骤，和连接 2000/2005 的步骤是完全一致的：

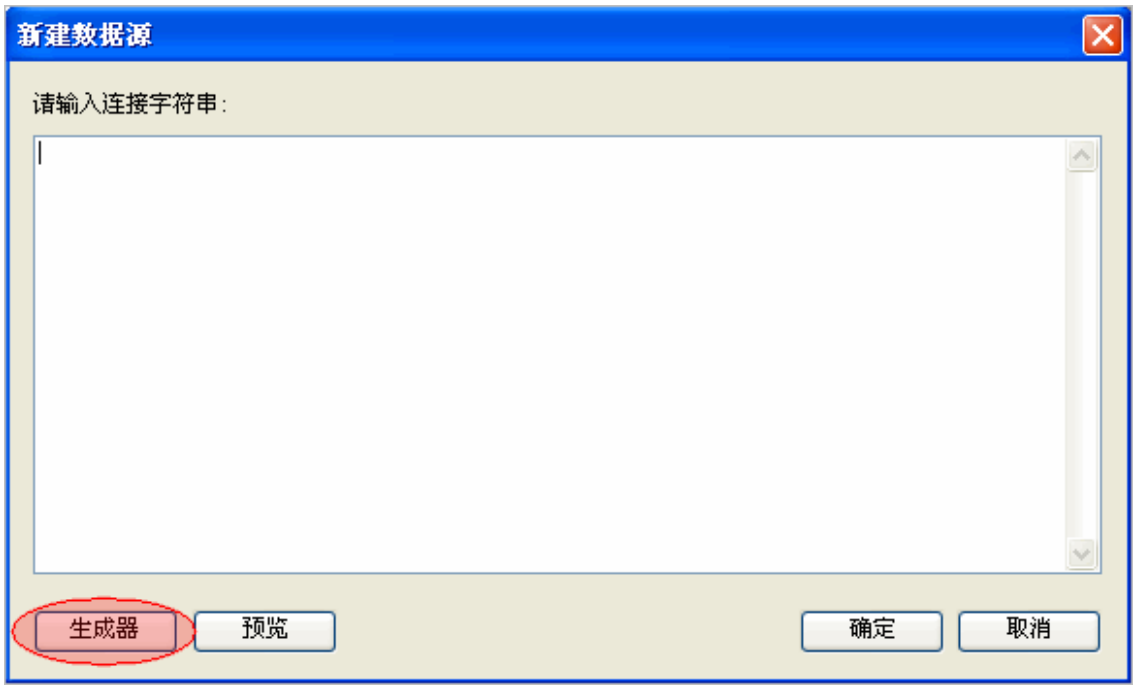
1、在数据表功能区单击下面的按钮：



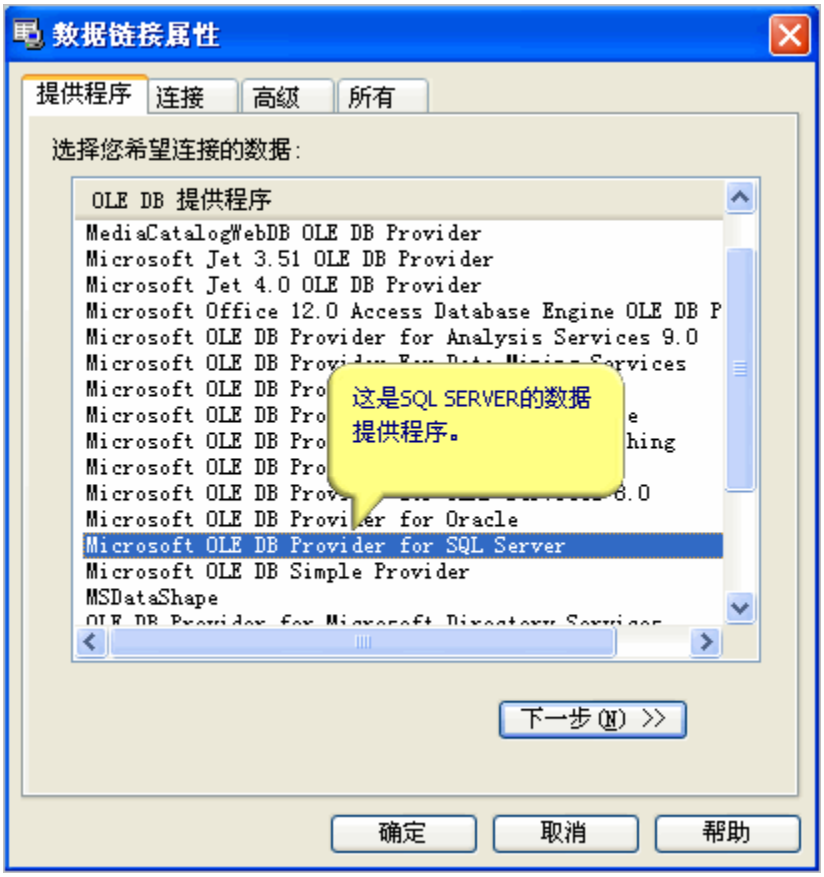
2、在外部数据源管理窗口，单击“增加”按钮：



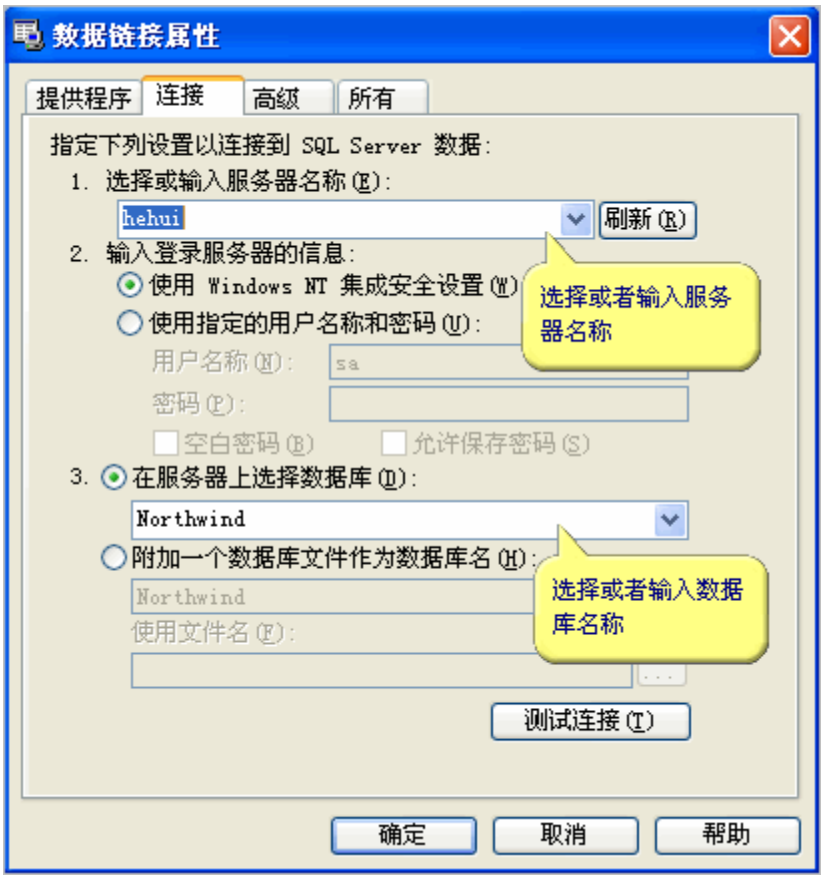
3、在新建数据源窗口单击按钮“生成器”：



4、在数据链接属性设置窗口，选择下图所示的数据提供程序：

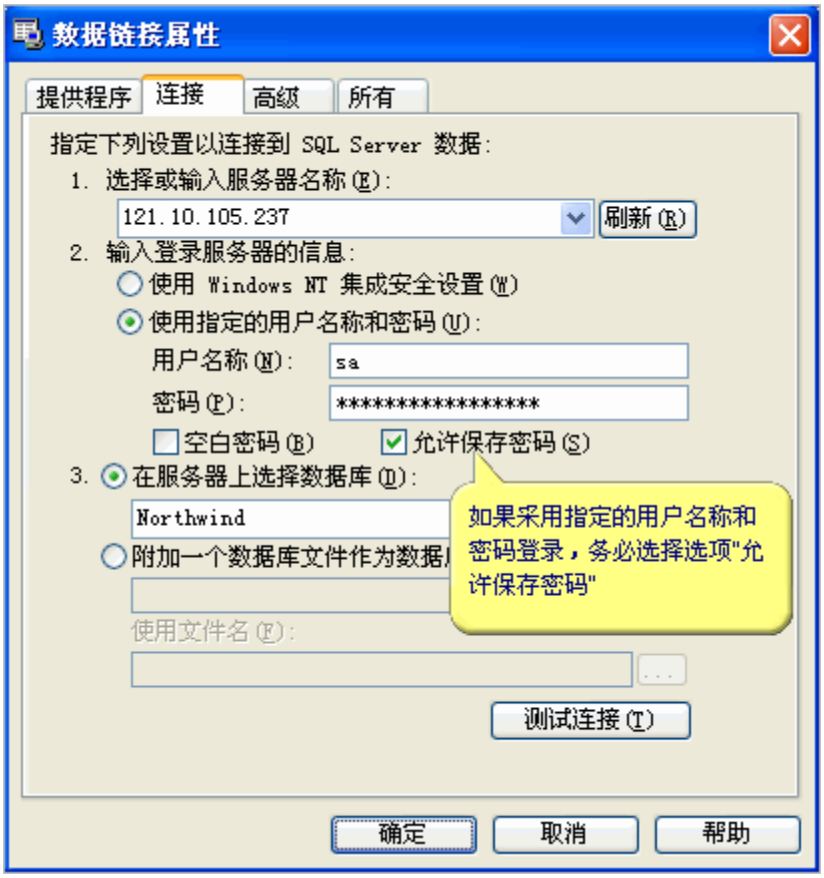


5、然后选择要连接的服务器数据库，SQL Server 有两种登录方式，下图采用的是“集成安全设置”：



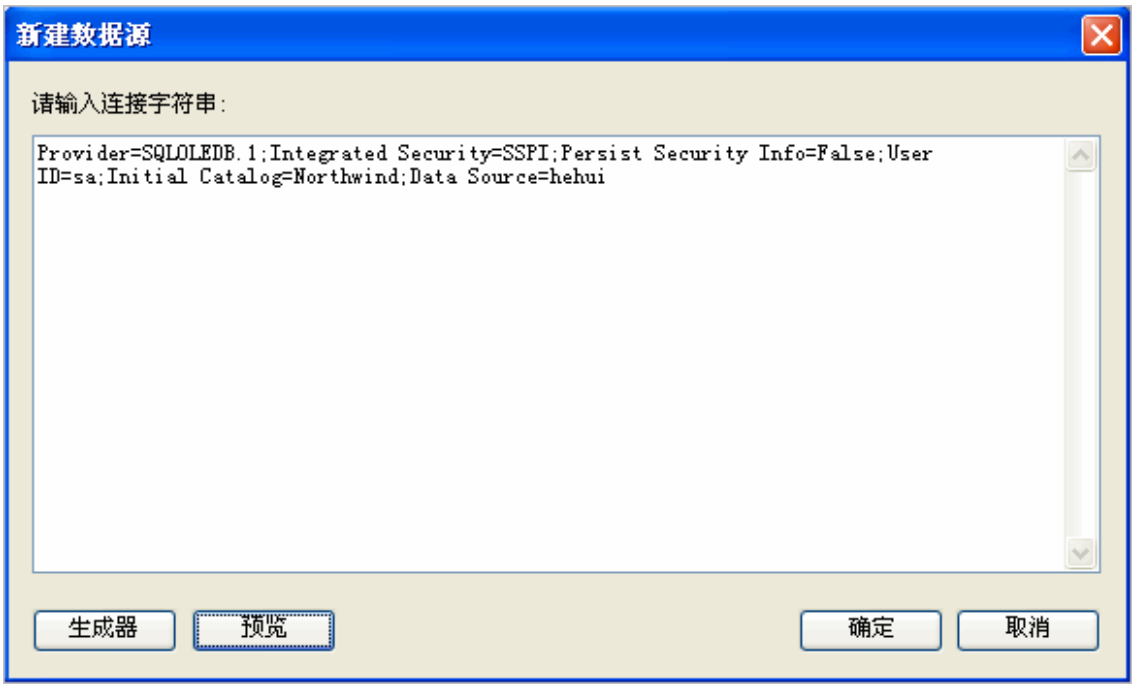
6、可以直接输入服务器的 IP 地址，如果采用指定的用户名称和密码登录，务必选择选项“允许保

存密码”：



提示：经常有用户问为什么测试连接可以，设置好之后却连接不上，原因在于没有选择这个“允许保存密码”的选项。

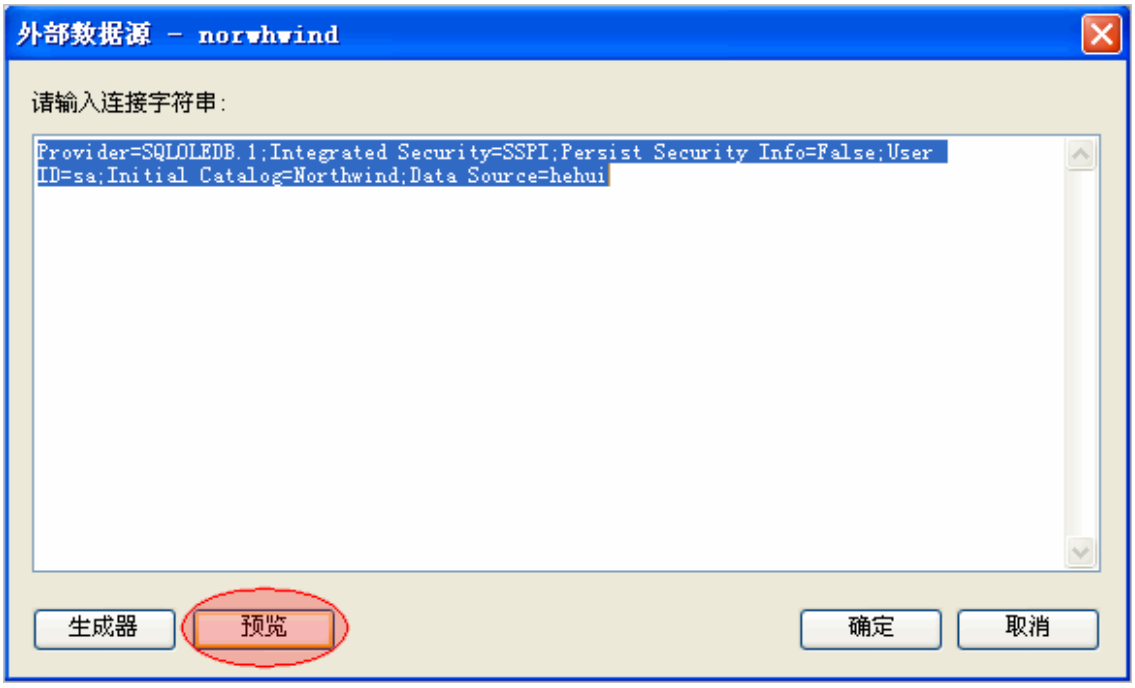
7、设置完成后，单击“确定”按钮，即可看到生成的链接字符串：



8、单击“确定”按钮，输入数据源名称，完成设置。

3.17.3.6 预览外部数据源

在外部数据源设置窗口，单击预览按钮：

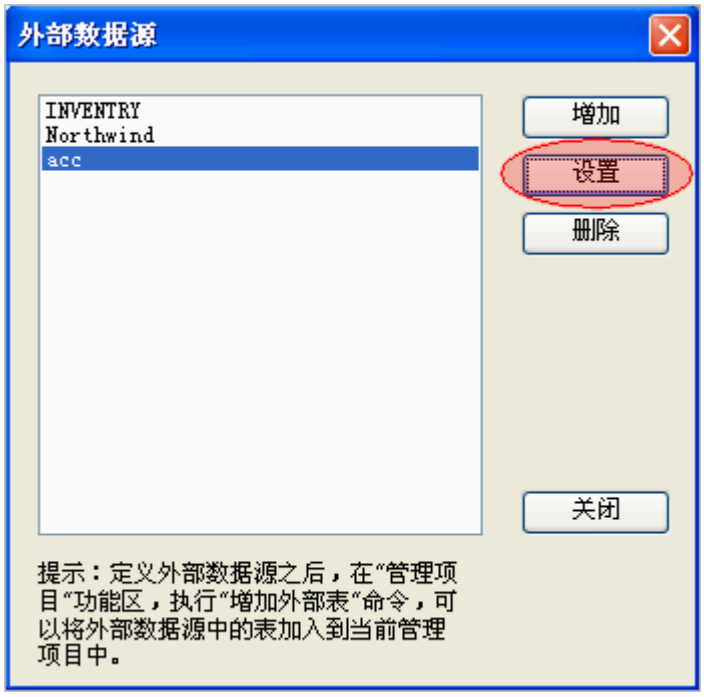


可以预览该数据源中的所有表：



3.17.3.7 修改外部数据源设置

在外部数据源设置窗口，单击“设置”按钮，可以修改外部数据源设置。



如果已经加载外部数据源中的表，必须保证修改后的数据源包括所有已经加载的外部表，且表结构完全相同。

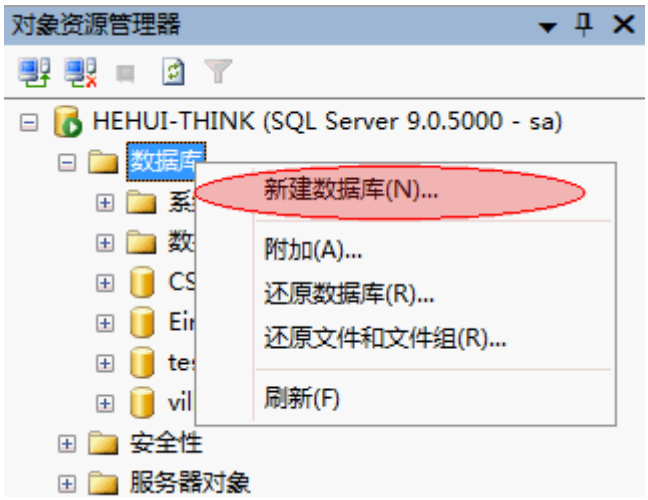
3.17.3.8 在 SQL Server 中创建数据库和用户

创建数据库

1、启动 SQL Server Management Studio，以最高管理权限 sa 帐户登陆：



2、右击数据库，单击"新建数据库"：

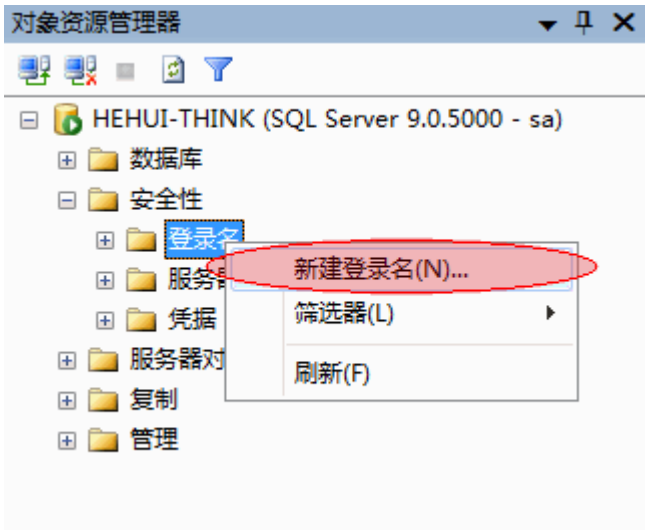


3、输入数据库名，设置数据库存放文件夹，单击确定按钮后，就可以创建一个新的数据库

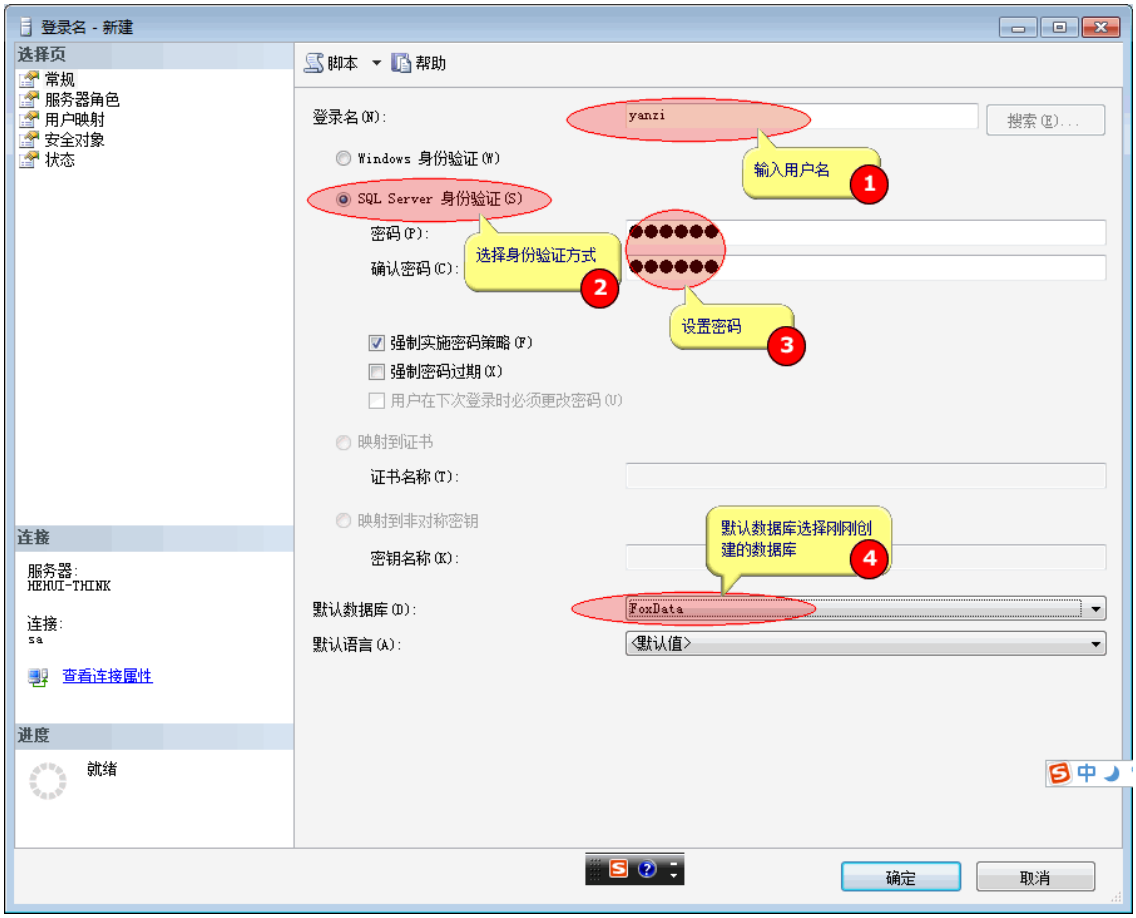


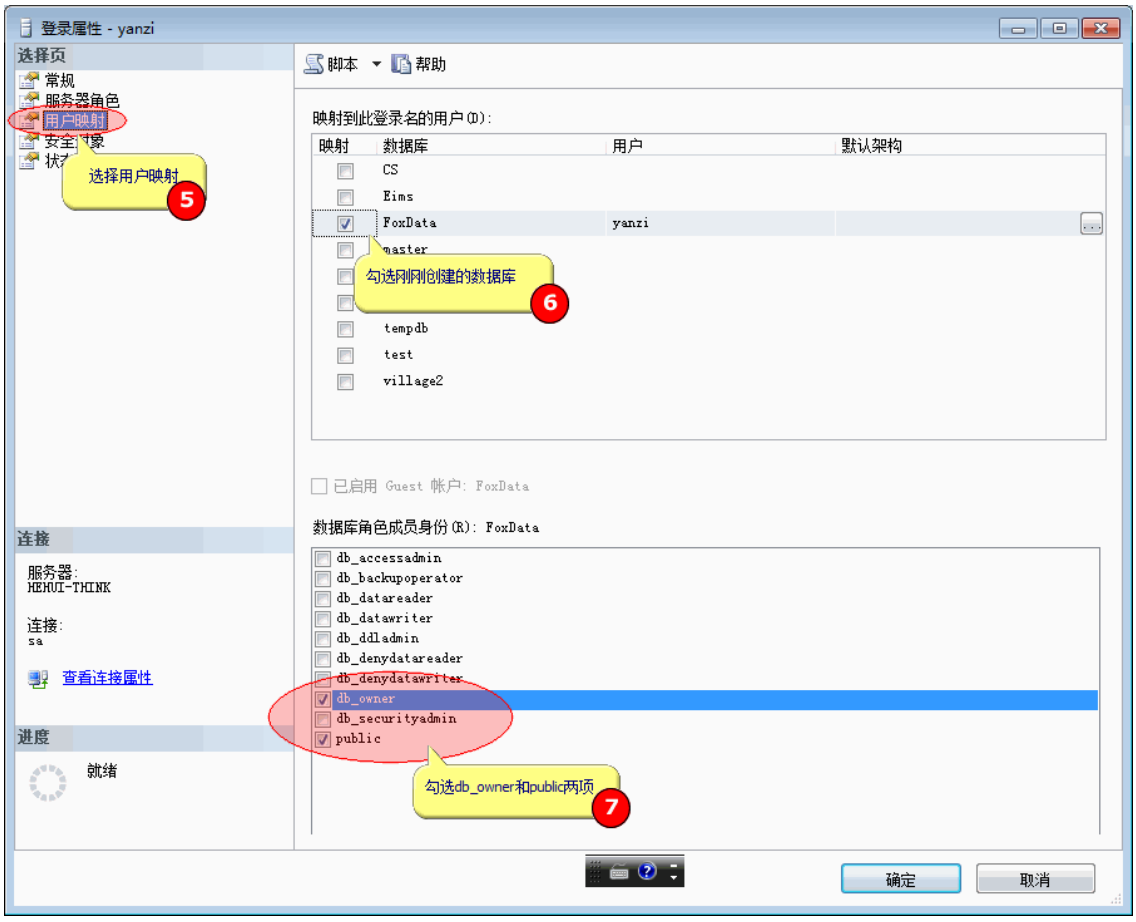
创建用户

1、在“安全性”下右击“登录名”，单击“新建登录名”：



2、按下图所示设置登录名属性：


登录名 - 新建
选择页
常规
服务器角色
用户映射
安全对象
状态
脚本
帮助
登录名(N):
yanzi
搜索(E)...
输入用户名
1
Windows 身份验证(W)
SQL Server 身份验证(S)
选择身份验证方式
2
密码(P):
确认密码(C):
设置密码
3
强制实施密码策略(F)
强制密码过期(X)
用户在下次登录时必须更改密码(U)
映射到证书
证书名称(T):
映射到非对称密钥
密钥名称(K):
默认数据库选择刚刚创建的数据库
4
默认数据库(D):
FoxData
默认语言(A):
<默认值>
连接
服务器:
MEHUI-THINK
连接:
sa
查看连接属性
进度
就绪
确定
取消


登录属性 - yanzi
选择页
常规
服务器角色
用户映射
安全对象
状态
选择用户映射
5
脚本
帮助
映射到此登录名的用户(M):
映射
数据库
用户
默认架构
CS
Eims
FoxData
yanzi
master
勾选刚刚创建的数据库
6
tempdb
test
village2
已启用 Guest 帐户: FoxData
数据库角色成员身份(R): FoxData
db_accessadmin
db_backupoperator
db_datareader
db_datawriter
db_ddladmin
db_denydatareader
db_denydatawriter
db_owner
db_securityadmin
public
勾选db_owner和public两项
7
连接
服务器:
MEHUI-THINK
连接:
sa
查看连接属性
进度
就绪
确定
取消

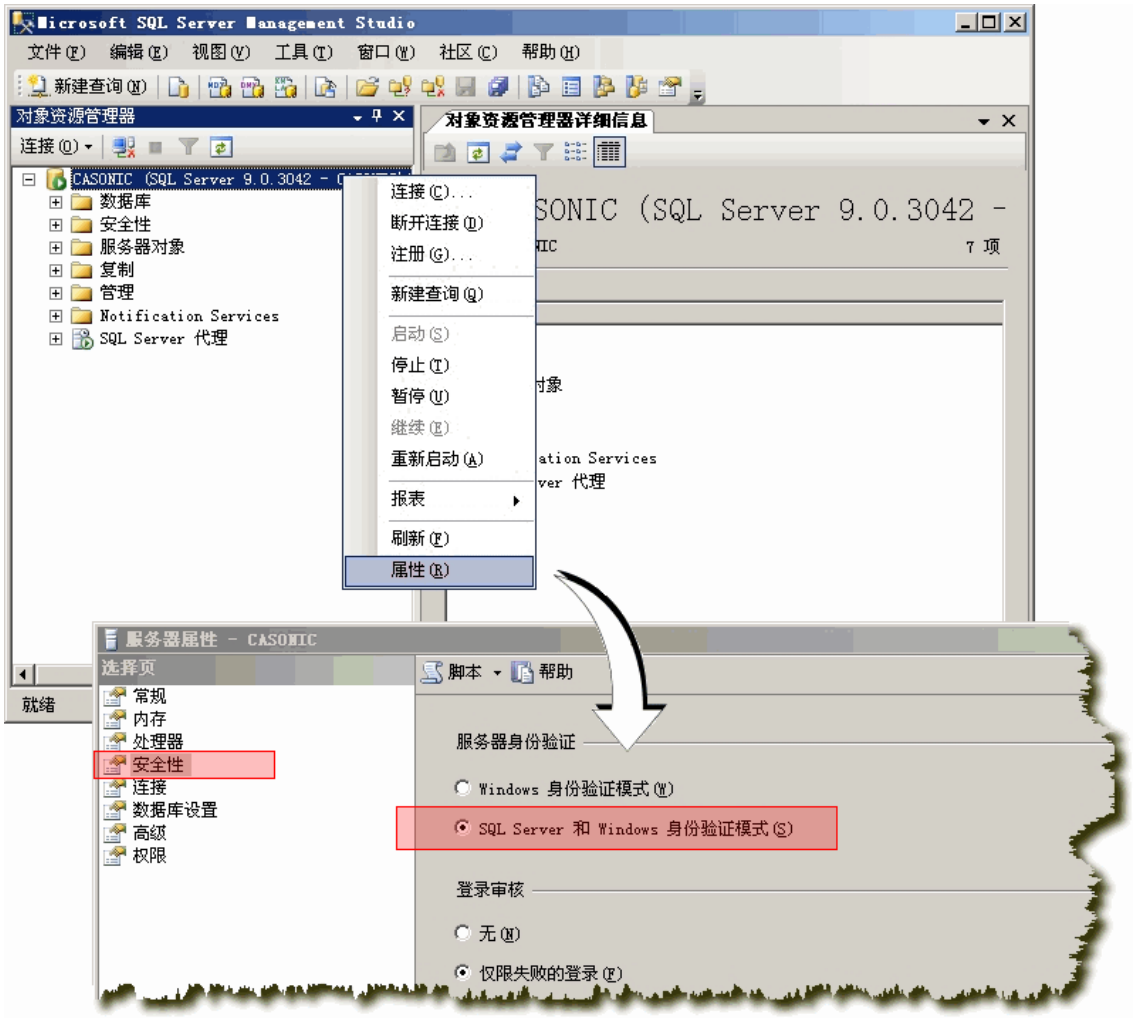
3.17.3.9 设置 SQL Server 以允许远程连接

3.17.3.9.1 设置 SQL Server 2005 以允许远程连接

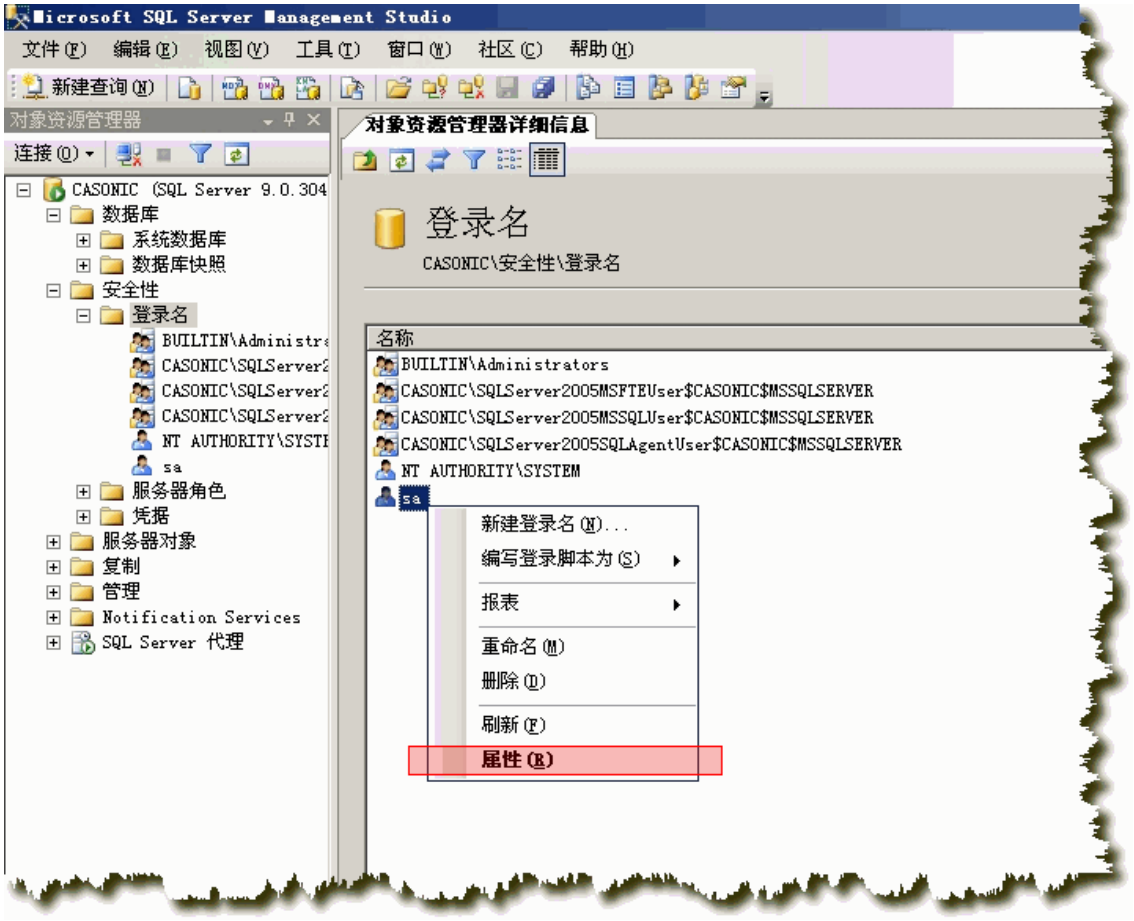
如果无法通过 IP 地址远程连接你的 SQL Server 2005 服务器，可以参考下面的内容进行设置。
在进行下述设置之前，应该确保你的网络已经安装设置完毕，服务器已经正常连接到网络中。

1、设置身份验证模式

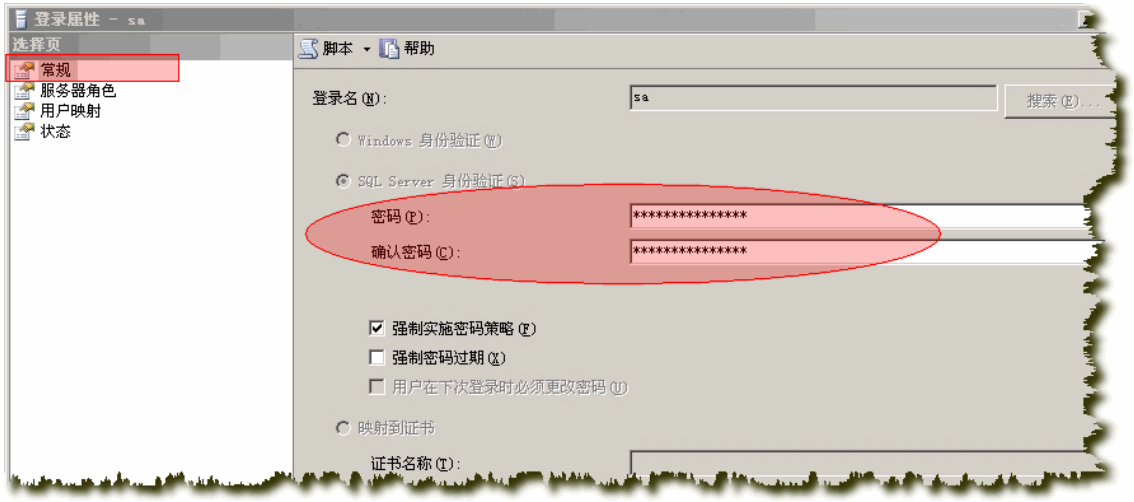
单击 Windows 的"开始"，依次选择"所有程序"、"Microsoft SQL Server 2005"，然后单击“SQL Server Management Studio”，在对象资源管理器中右击你的服务器，从快捷菜单中执行“属性”命令，在服务器属性窗口选择“安全性”，将服务器身份验证选择为“SQL Server 和 Windows 身份验证模式”：



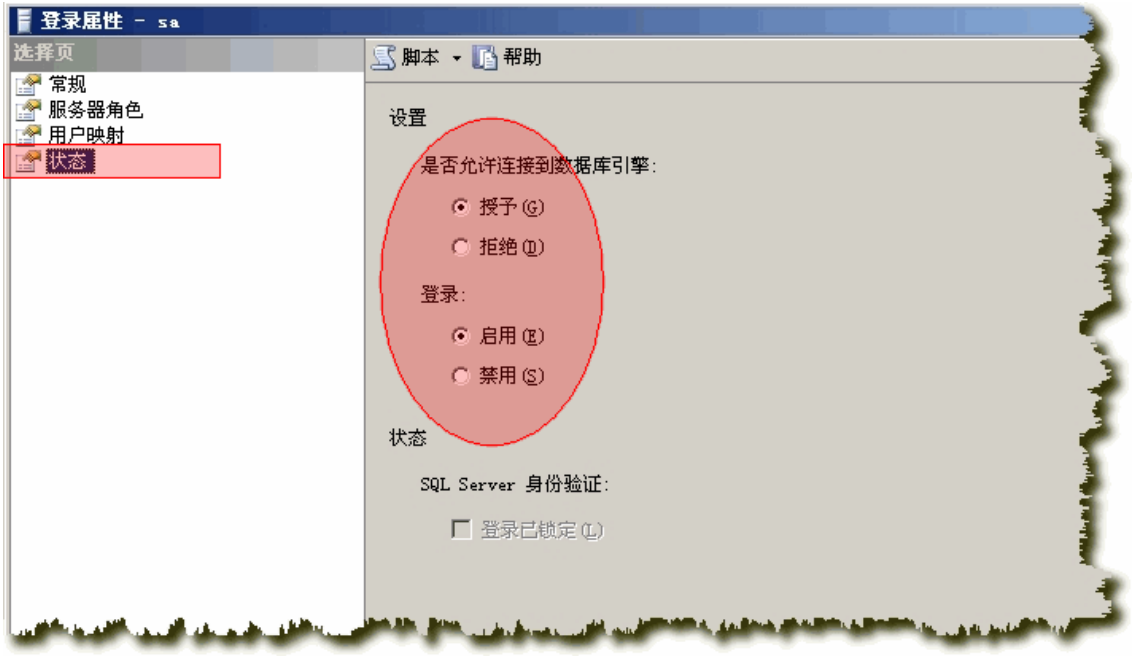
然后回到对象资源管理器，在左边展开节点“安全性”，选择子节点“登录名”，窗口右边会列出所有登录名，右击登录名"sa"，从快捷菜单中执行“属性”：



首先在登录属性设置窗口的左边选择“常规”，在右边选择“SQL Server 身份验证”，并输入密码：



然后在窗口左边选择“状态”，右边选择选项“授予”和“启用”：



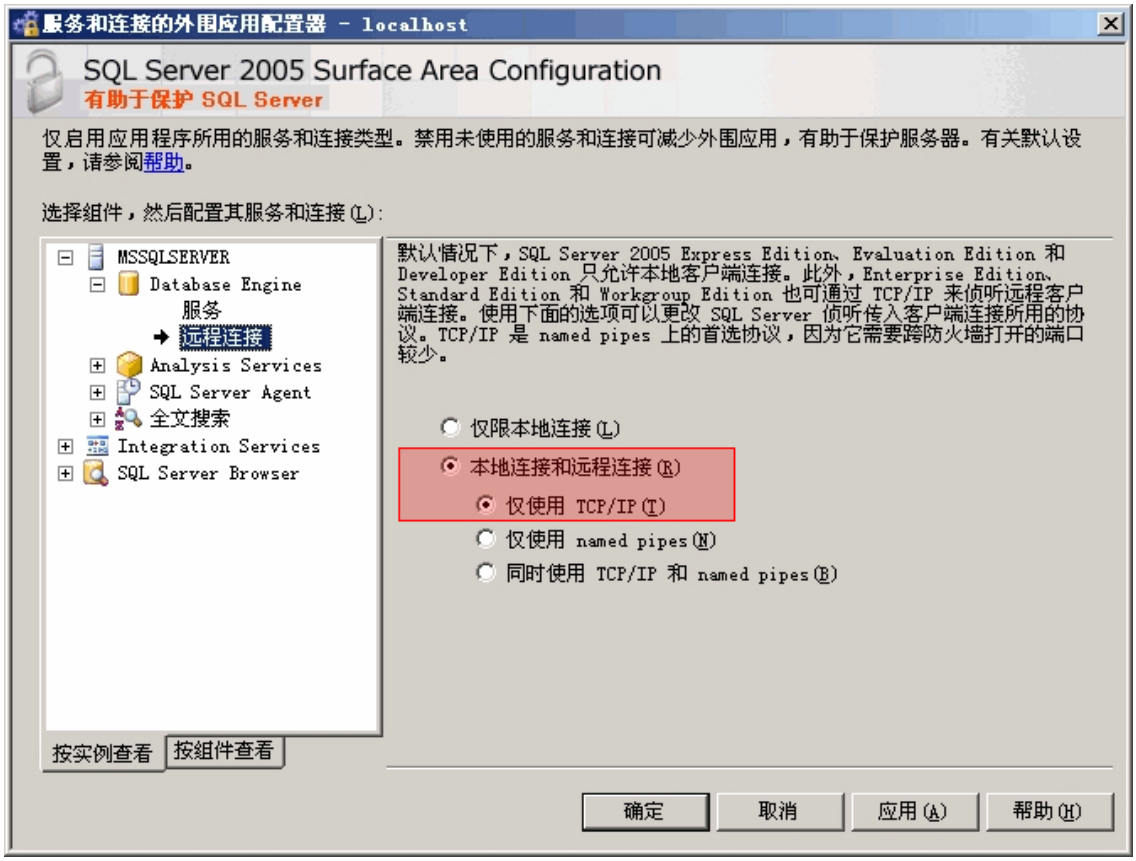
2、开启远程连接功能

单击 Windows 的"开始"，依次选择"所有程序"、"Microsoft SQL Server 2005"、"配置工具"，然后单击"SQL Server 外围应用配置器"。

启动“SQL Server 2005 外围应用配置器”，然后单击“服务和连接的外围应用配置器”：



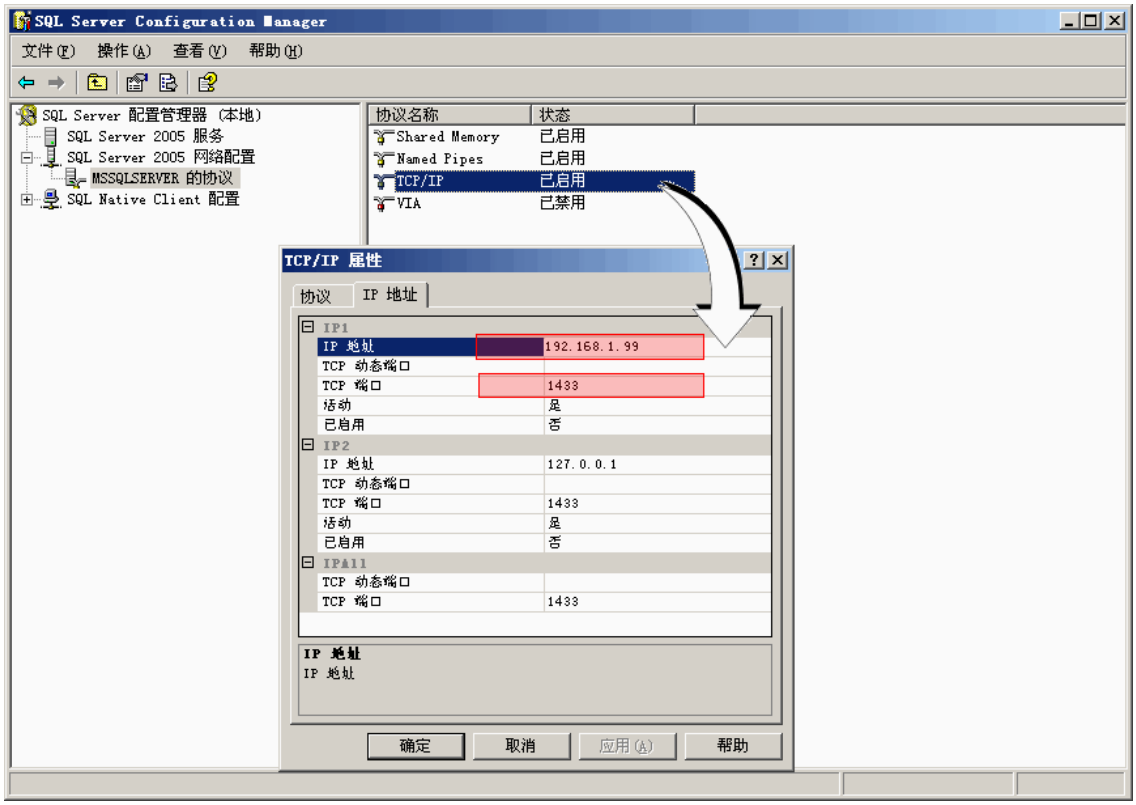
在“服务和连接的外围应用配置器”窗口，展开左边的目录树，选择“Database Engine”节点下的“远程连接”，然后在右边选择选项“本地连接和远程连接”和“仅使用 TCP/IP”，最后单击“确定”：



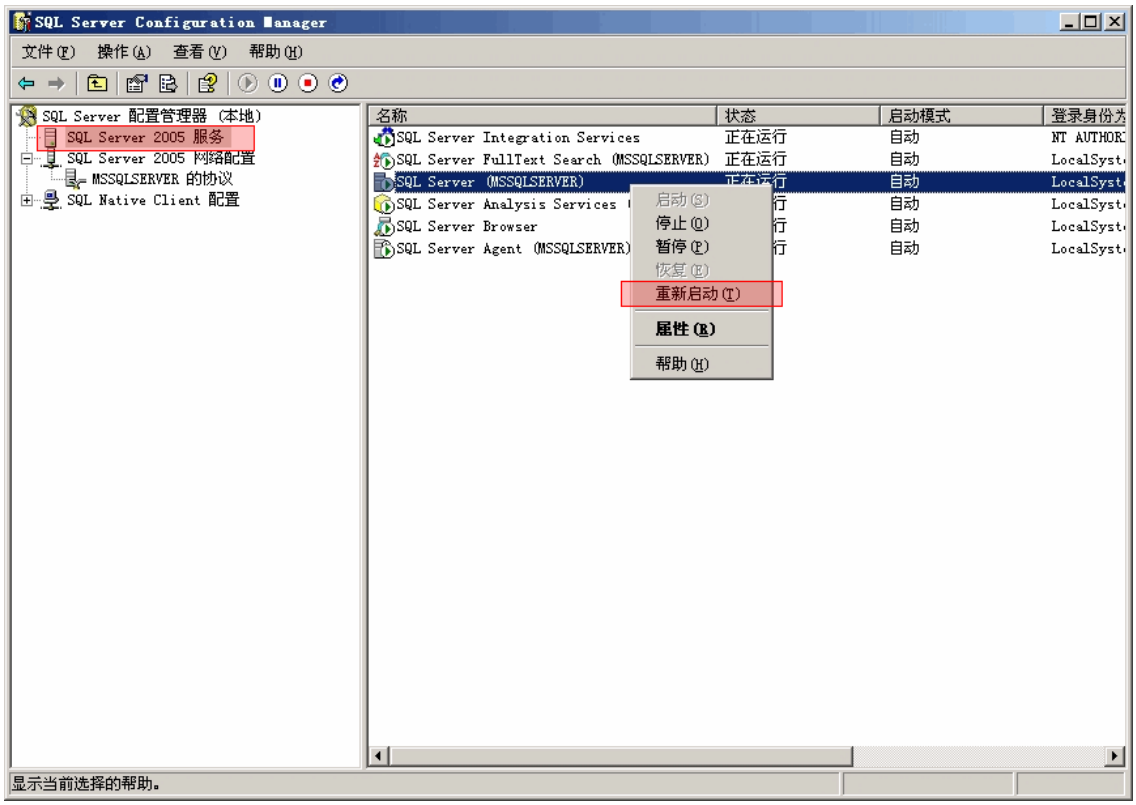
3、设置 IP 和端口

单击 Windows 的"开始"，依次选择"所有程序"、"Microsoft SQL Server 2005"、"配置工具"，然后单击"SQL Server 配置管理器"。

展开窗口左边的目录树，选择"SQL Server 2005 网络配置"节点下的"MSSQLSERVER 的协议"，然后双击右边的"TCP/IP"，打开"TCP/IP"属性窗口，选择“IP 地址页”，设置 IP 地址和端口，端口一般为 1433：



然后在窗口左边选择节点“SQL Server 2005 服务”，在窗口右边右击“SQL Server (MSSQLSERVER)”，从快捷菜单中执行“重新启动”：



4、设置防火墙例外

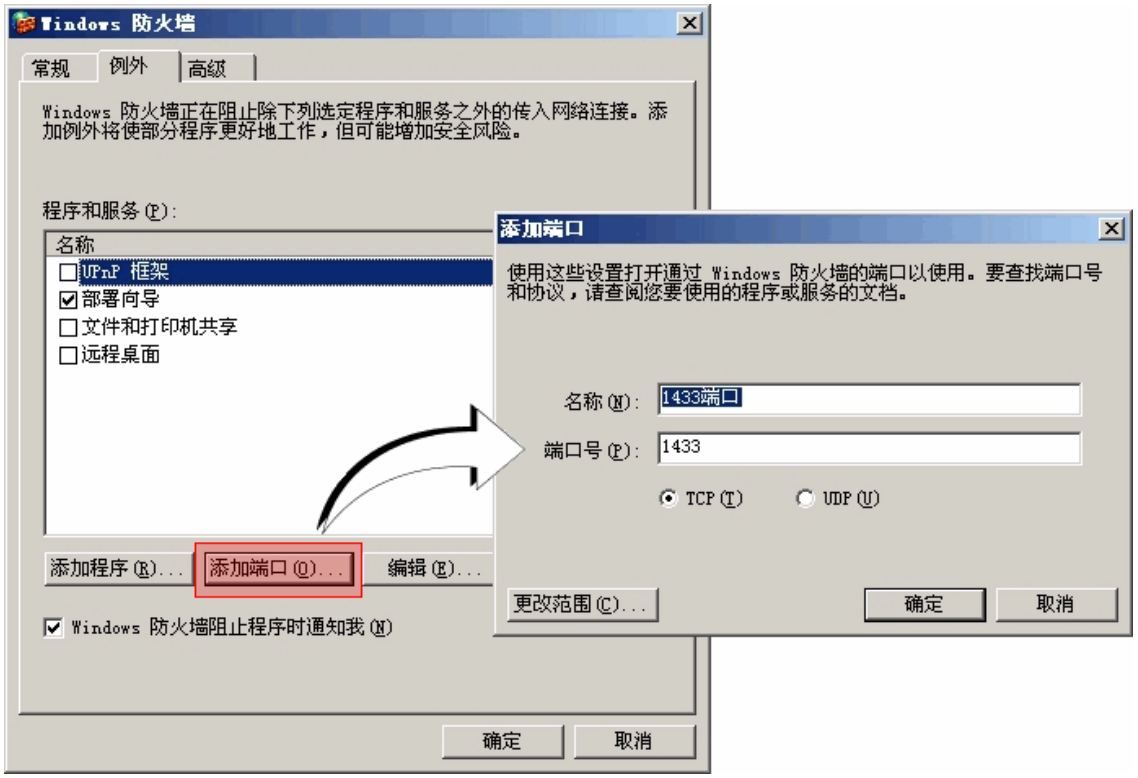
如果服务器已经启用了 Windows 的防火墙，那么必须添加例外，才能允许其它电脑连接此服务器

的 SQL Server 服务。

以下内容以 Windows XP 和 Windows Server 2003 为例，如果你使用的是 Windows 7/Vista 或者 Windows Server 2008，请参考：[设置 Windows Vista/7/2008 的防火墙]

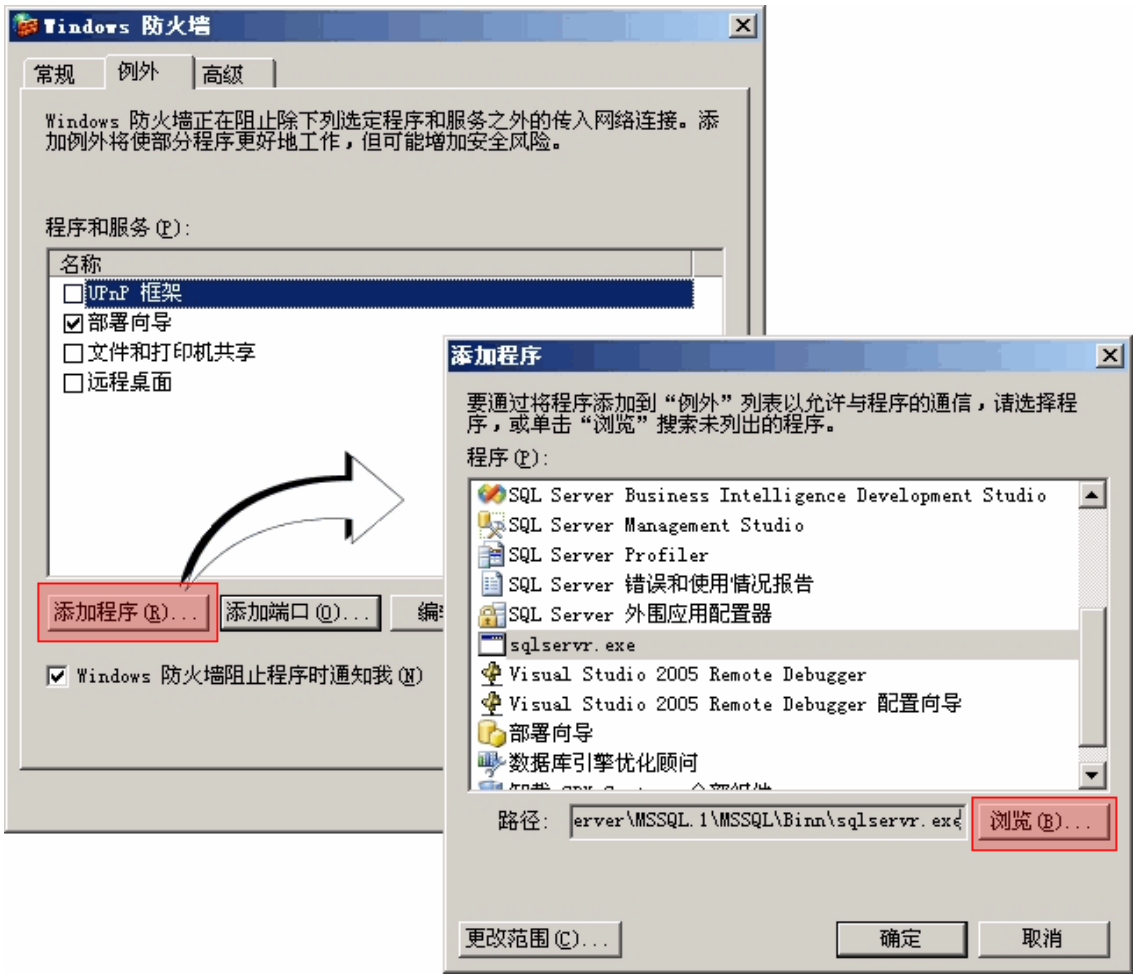
添加端口例外

在 Windows 防火墙中，选择"例外"选项卡，接着单击"添加端口"，输入端口号 1433，然后单击“确定”按钮：



添加程序例外

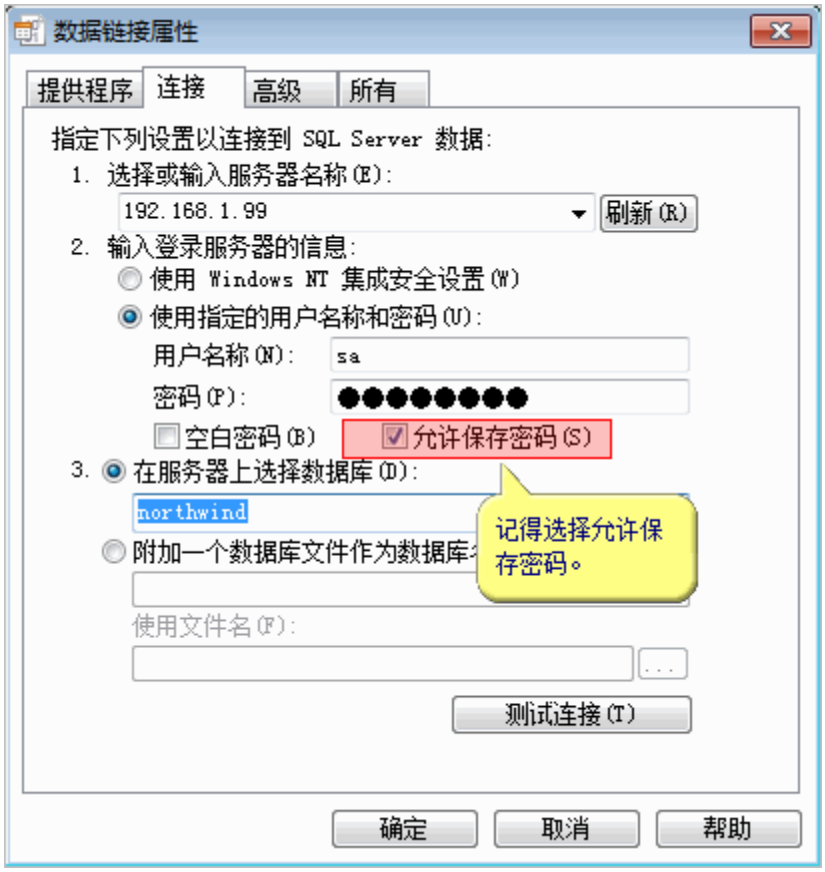
在 Windows 防火墙中，选择"例外"选项卡，接着单击"添加程序"，在“添加程序”窗口中，单击“浏览”，选择 “C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\sqlservr.exe” 可执行文件，然后单击“确定”按钮：



提示：上述可执行文件路径可能会根据 SQL Server 2005 的安装位置而不同。

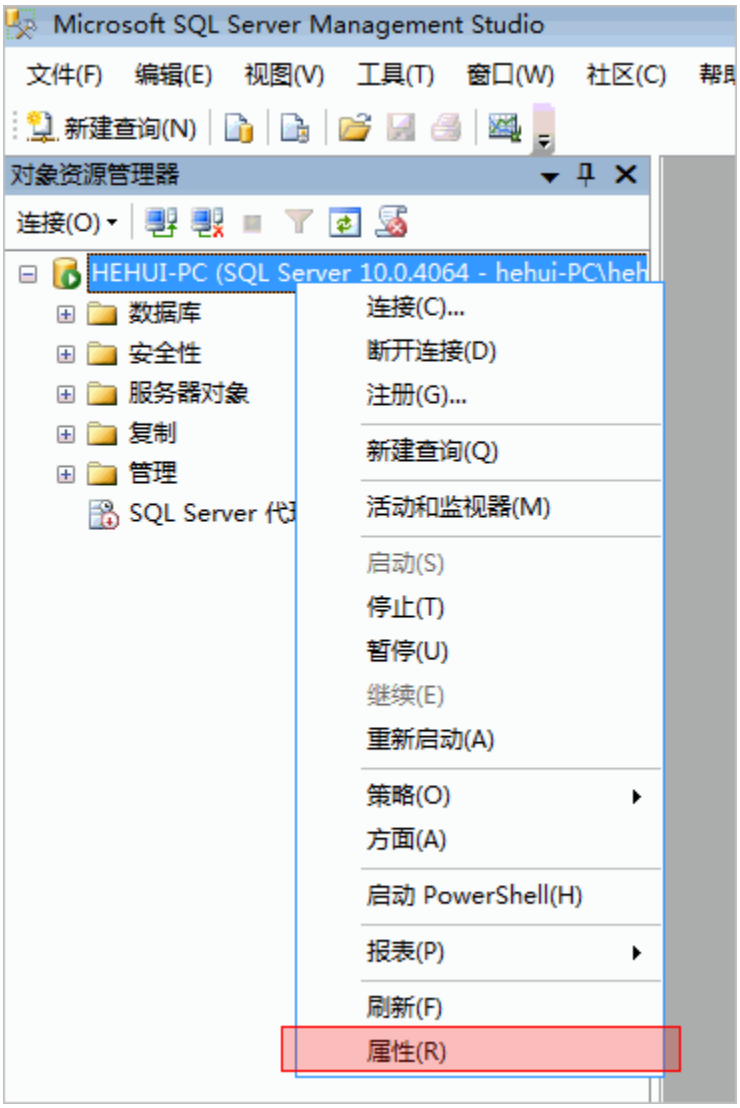
经过上述设置，其它电脑即可通过 IP 地址连接此服务器，连接方法请参考：连接 SQL Server 2000/2005

需要特别提示的是，通过 IP 地址连接远程 SQL Server 服务器的时候，应该选择选项“允许保存密码”：



3.17.3.9.2 设置 SQL Server 2008 以允许远程连接

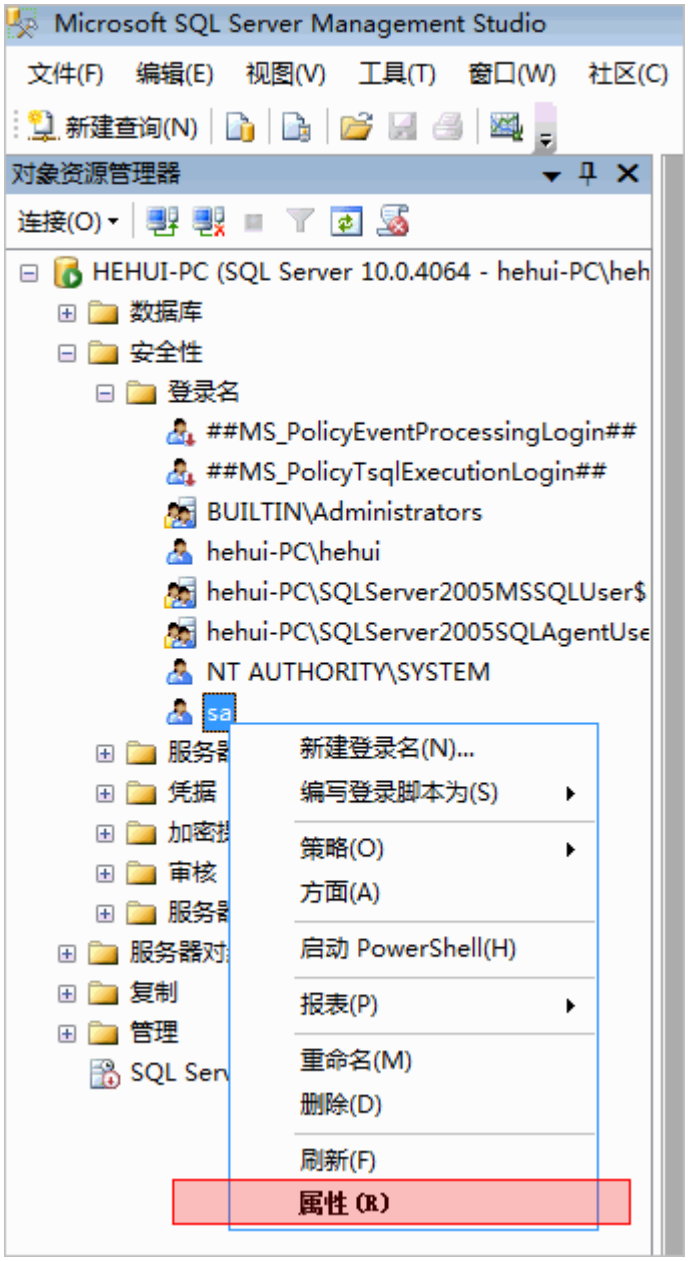
如果无法通过 IP 地址远程连接你的 SQL Server 2008 服务器，可以参考下面的内容进行设置。
在进行下述设置之前，应该确保你的网络已经安装设置完毕，服务器已经正常连接到网络中。
1、单击 Windows 的"开始"，依次选择"所有程序"、"Microsoft SQL Server 2008"，然后单击“SQL Server Management Studio”，在对象资源管理器中，右击你的服务器，从快捷菜单中执行“属性”：



2、在服务器属性窗口左侧选择“安全性”，右侧选择“SQL Server 和 Windows 身份验证模式”：



3、回到对象资源管理器，在节点“安全性 → 登录名”下右击登录名“sa”，从快捷菜单中执行“属性”：



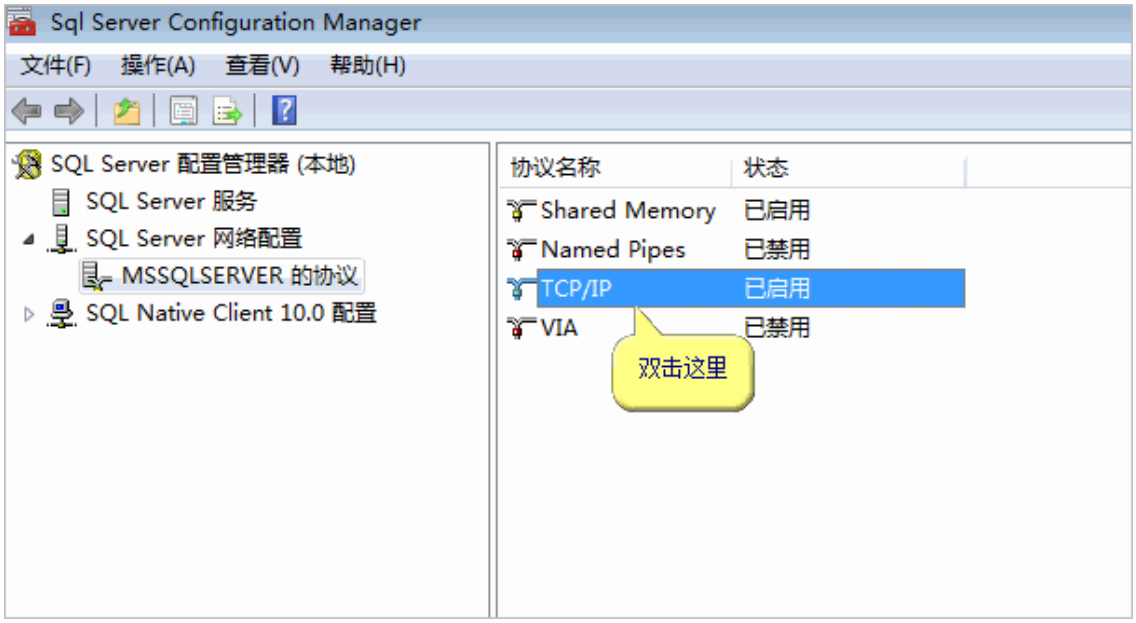
4、首先在登录属性窗口，左侧选择"常规"，右侧输入密码:



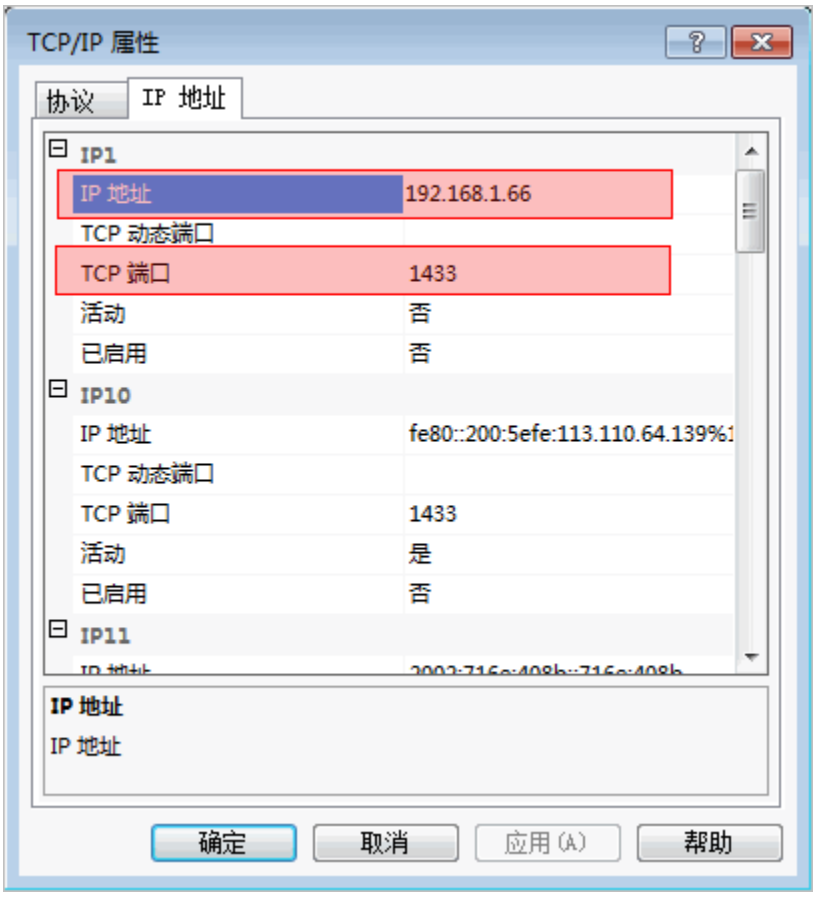
5、然后在登录属性窗口左侧选择"状态"，右侧选择"授予"和"启用"：



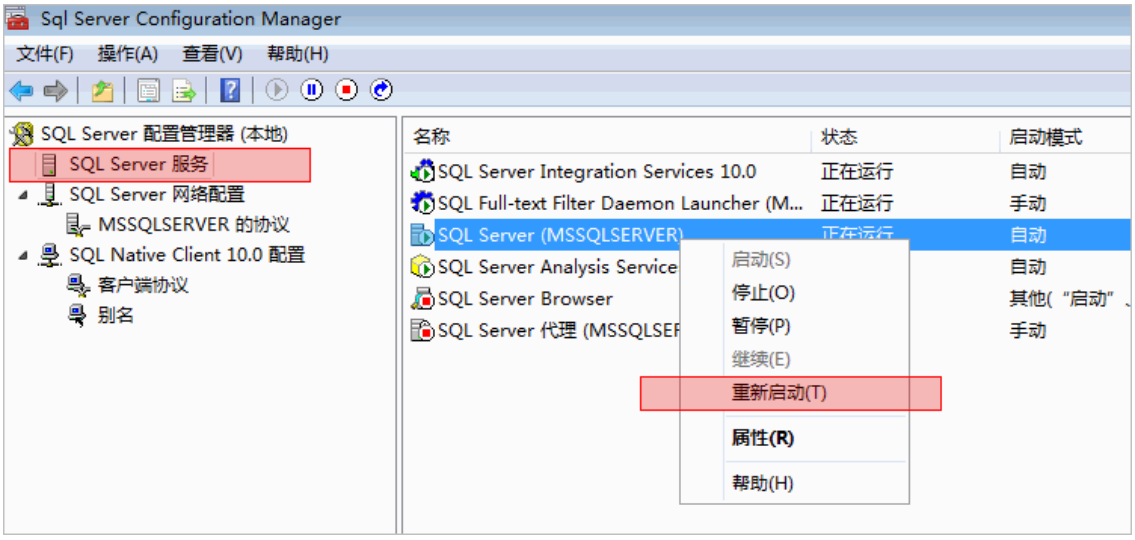
6、单击 Windows 的"开始"，依次选择"所有程序"、"Microsoft SQL Server 2008"、"配置工具"，然后单击"SQL Server 配置管理器"，在窗口左侧选择节点"MSSQLSERVER 的协议"，在右侧双击"TCP/IP"：



7：在 TCP/IP 属性窗口，选择“IP 地址”选项卡，设置好 IP 地址和端口号，端口号一般是 1433:



8，然后选择窗口左侧的节点“SQL Server 服务”，在右侧右击“SQL Server(MSSQLSERVER)”，从快捷菜单中执行“重新启动”：



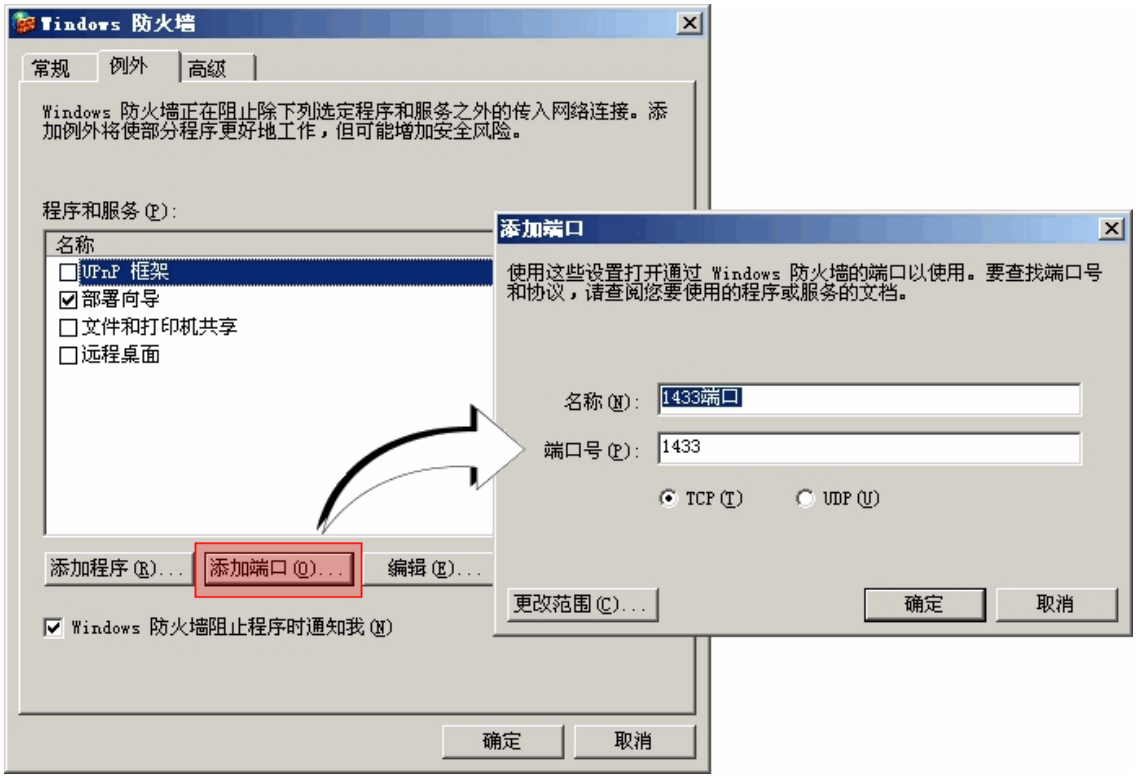
9、如果服务器已经启用了 Windows 的防火墙，那么必须添加例外，才能允许其它电脑连接此服务器的 SQL Server 服务。

以下内容以 Windows XP 和 Windows Server 2003 为例，如果你使用的是 Windows 7/Vista 或者 Windows Server 2008，请参考：[设置 Windows Vista/7/2008 的防火墙]

添加端口例外

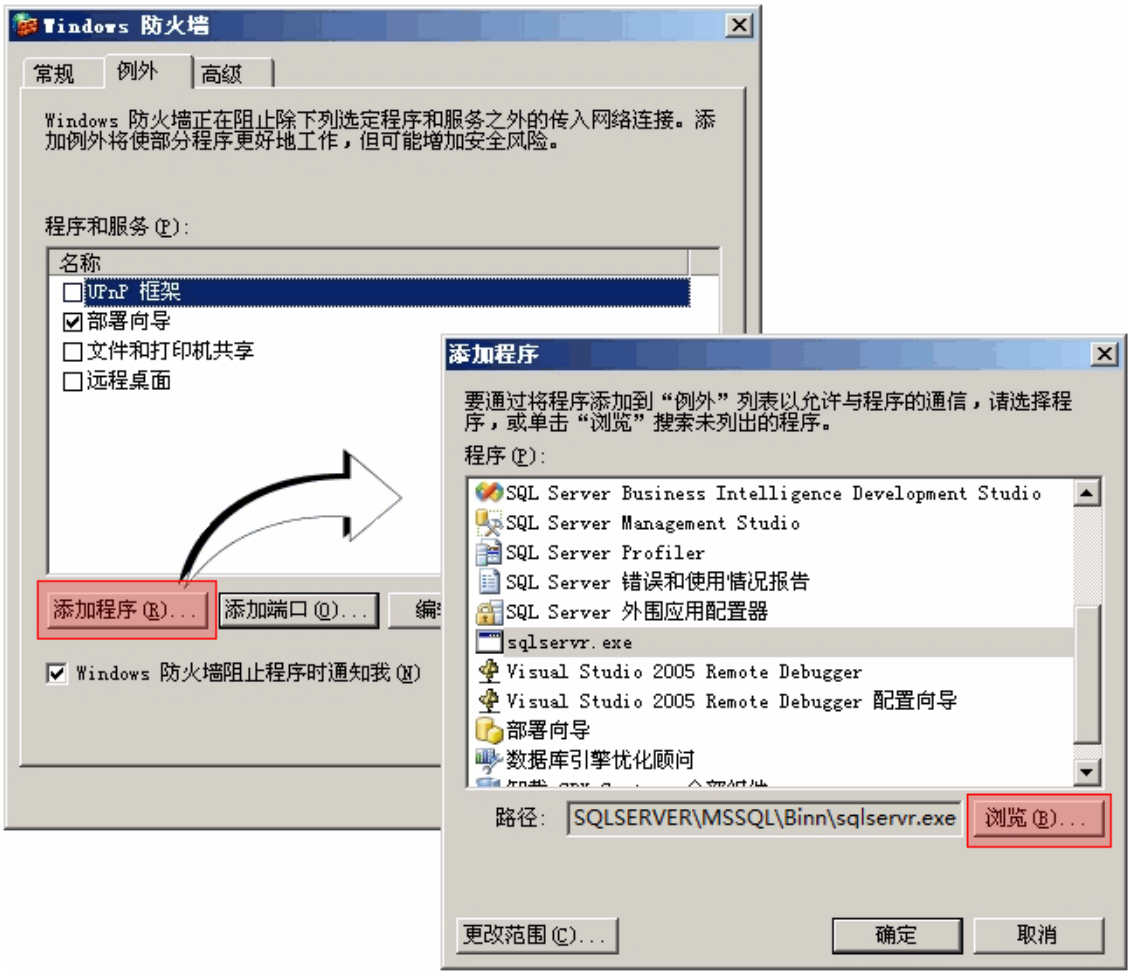
在 Windows 防火墙中，选择"例外"选项卡，单击"添加端口"，输入端口号 1433，然后单击“确定”

按钮：



添加程序例外

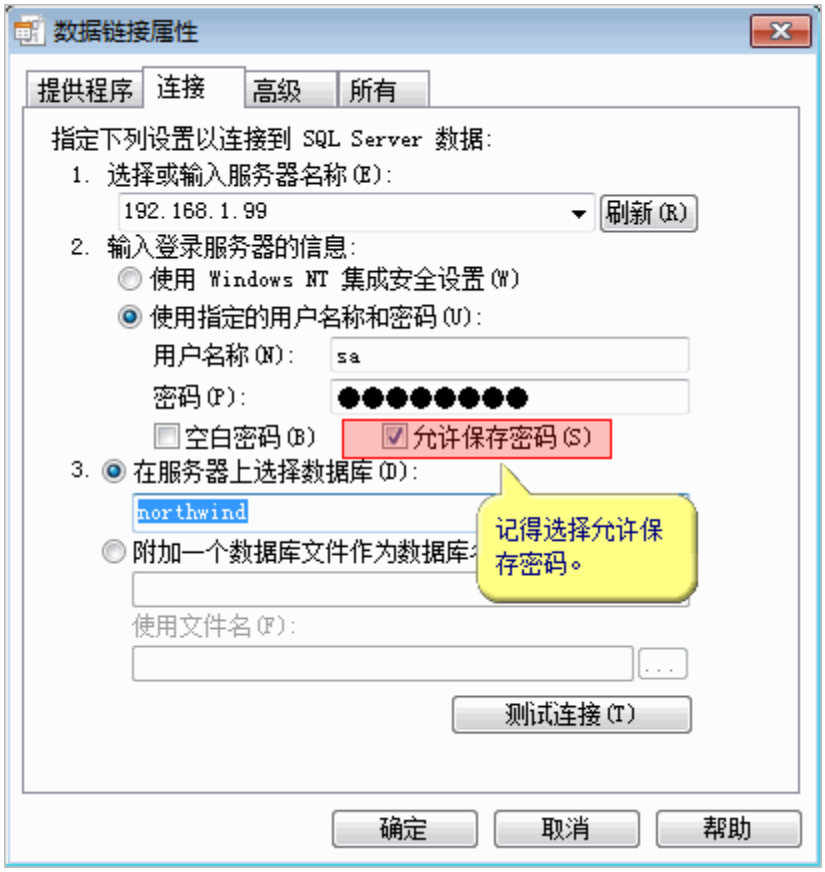
在 Windows 防火墙中，选择"例外"选项卡，单击"添加程序"，在“添加程序”窗口中，单击“浏览”，选择 “C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Binn\sqlservr.exe”可执行文件，然后单击“确定”按钮：



提示：上述可执行文件路径可能会根据 SQL Server 2008 的安装位置而不同。

经过上述设置，其它电脑即可通过 IP 地址连接此服务器，连接方法请参考：连接 SQL Server 2008

需要特别提示的是，通过 IP 地址连接远程 SQL Server 服务器的时候，应该选择选项“允许保存密码”：



3.17.3.9.3 设置 Windows Vista/7/2008 的防火墙

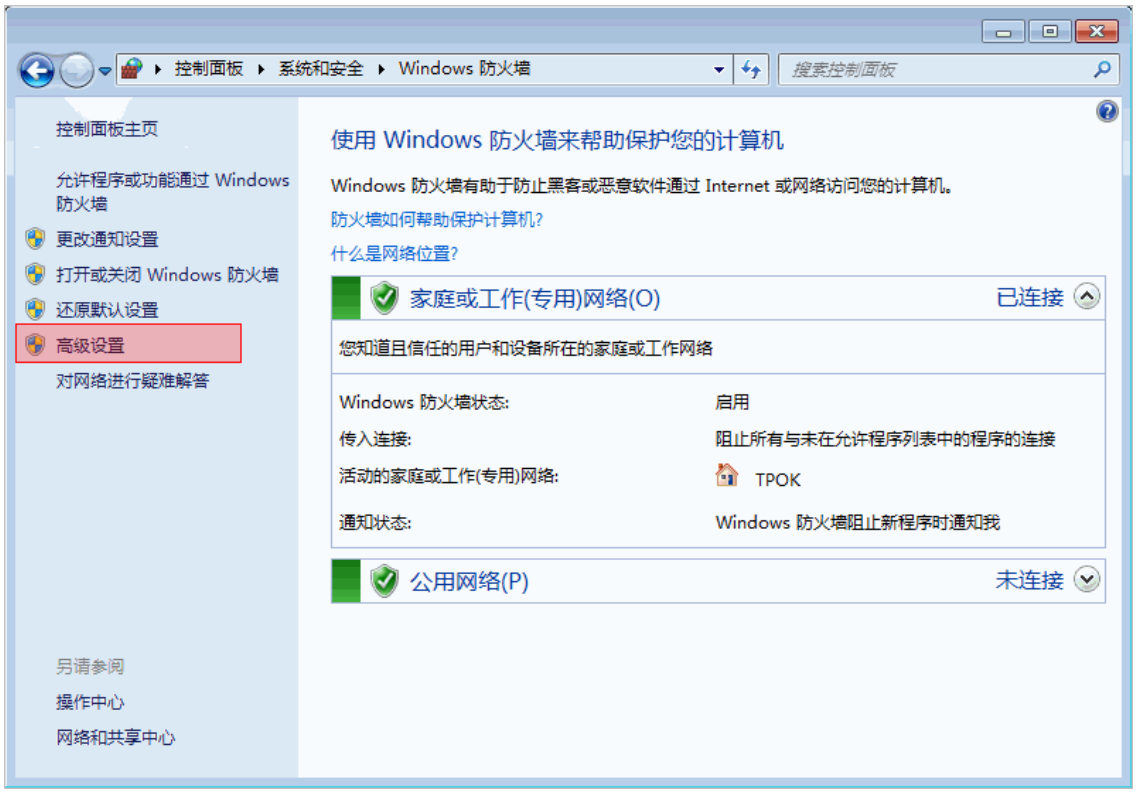
前面已经提到，在服务器启用 Windows 防火墙的情况下，需要添加例外，才能允许通过 IP 连接此服务器的 SQL Server。

添加例外的方法是基于 Windows XP 和 Windows Server 2003 的防火墙介绍的。

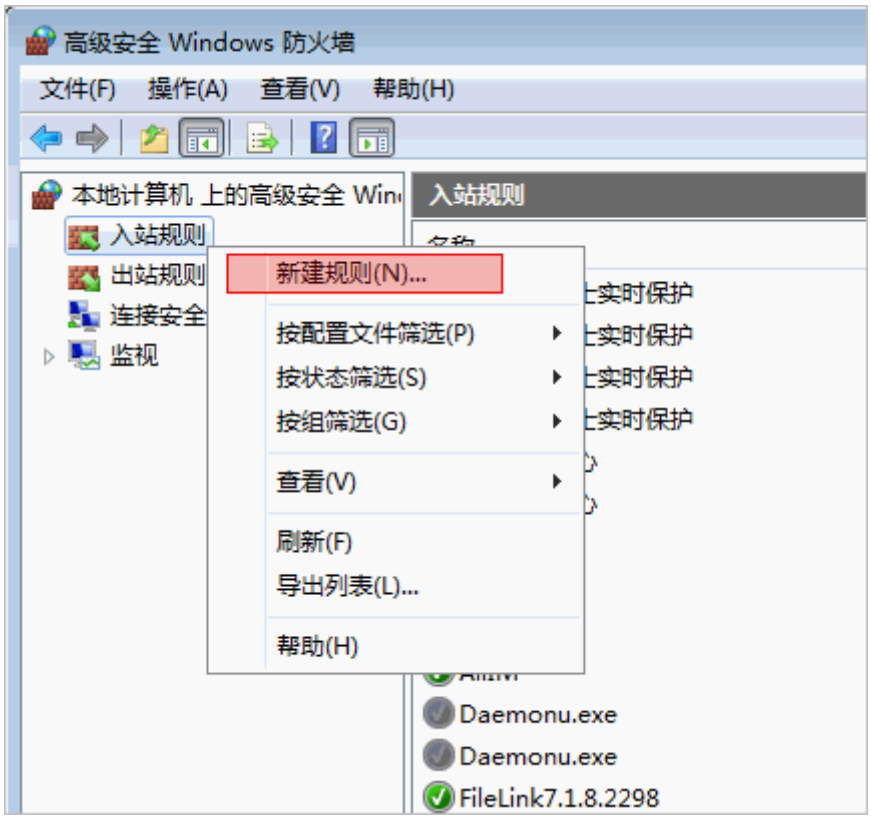
如果你使用的是 Windows Vista/7 或 Windows Server 2008，其防火墙的添加例外的方式有所不同。

添加端口例外

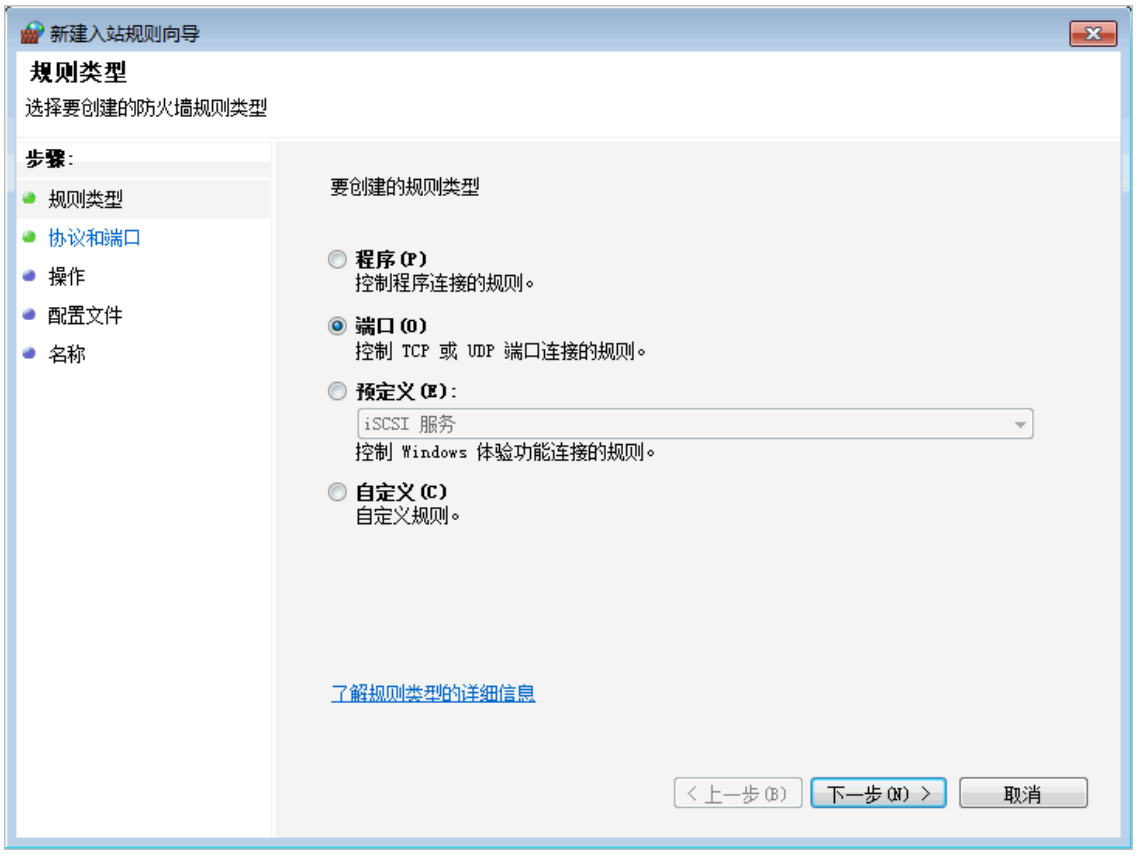
1、打开 Windows 防火墙，单击高级设置：



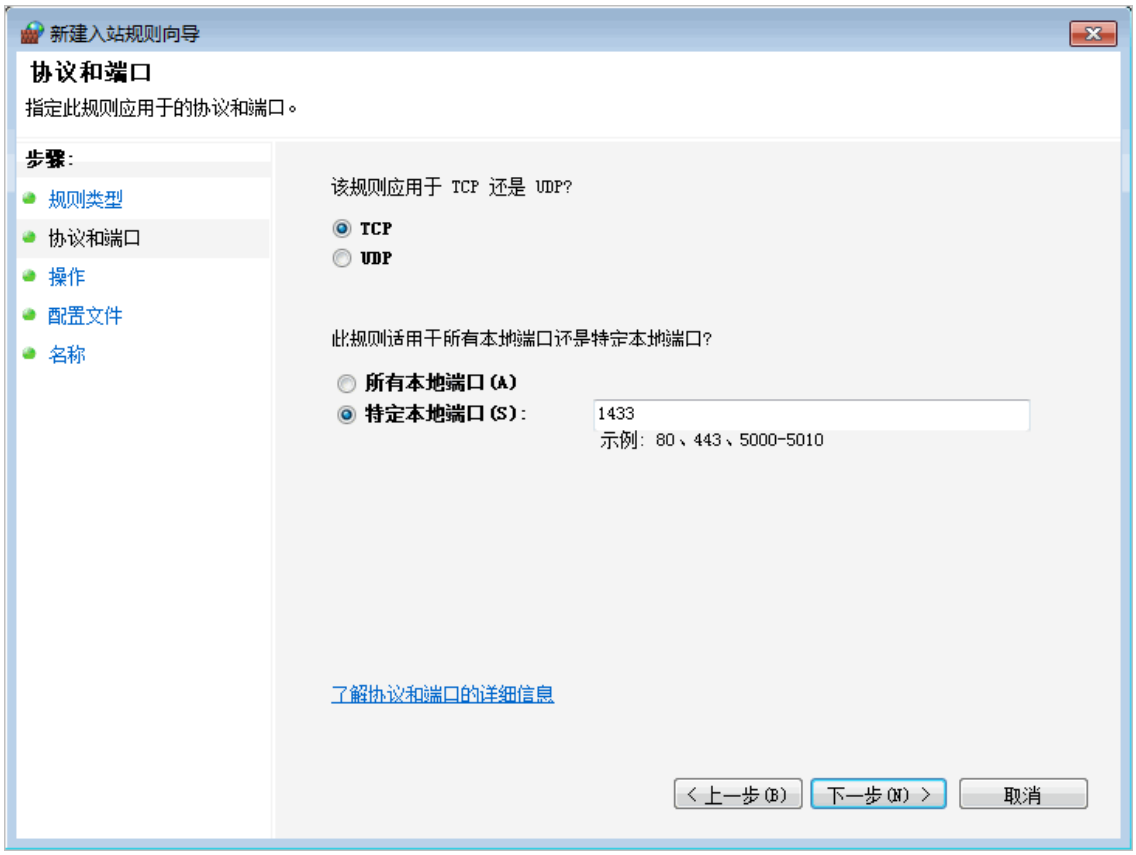
2、在高级设置窗口，先单击“入站规则”，再右击“入站规则”，从快捷菜单执行“新建规则”：



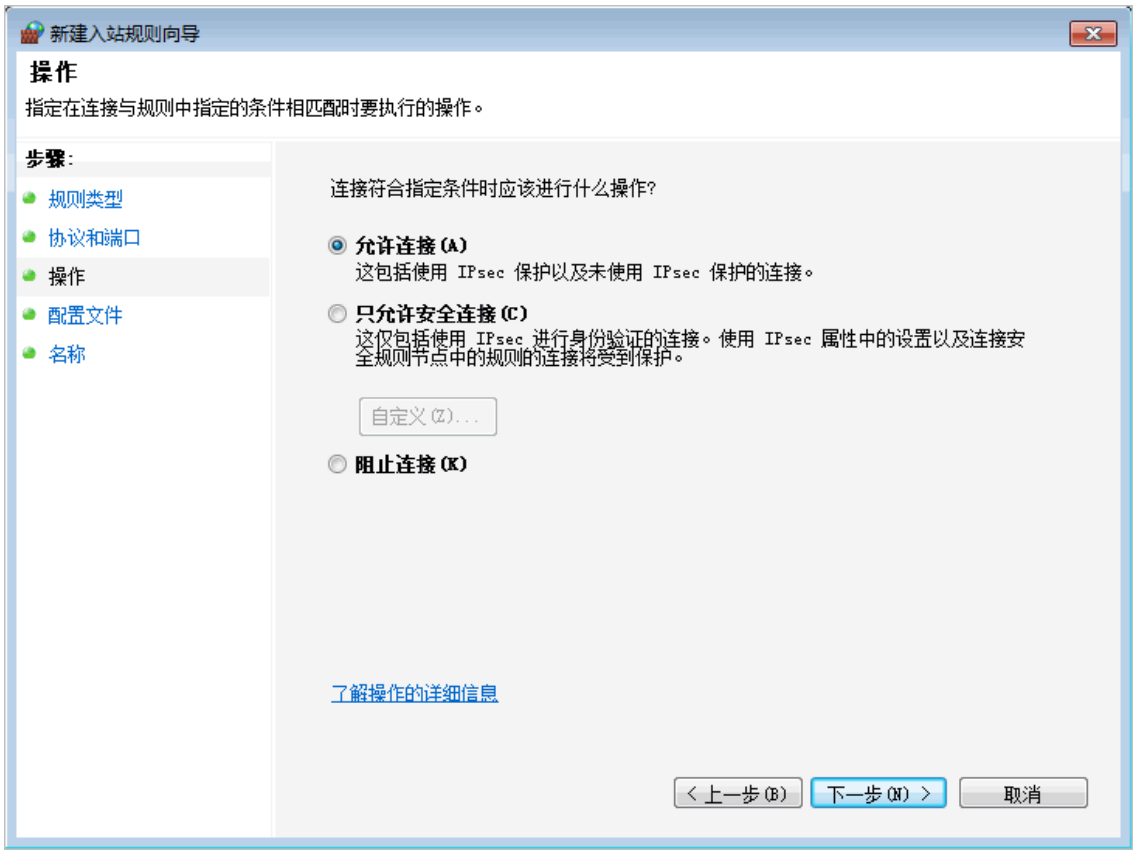
3、在新建规则窗口，规则类型选择“端口”，单击下一步：



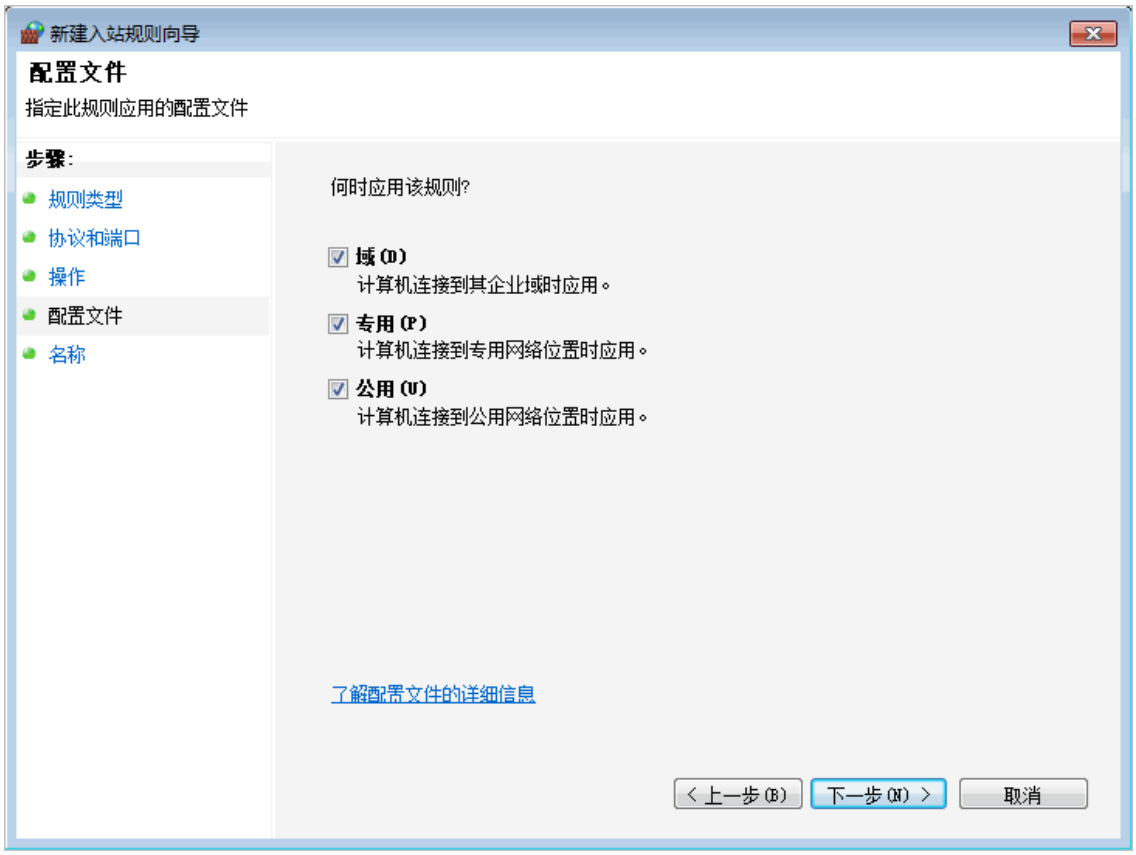
4、规则应用选择“TCP”，输入端口号 1433，单击下一步：



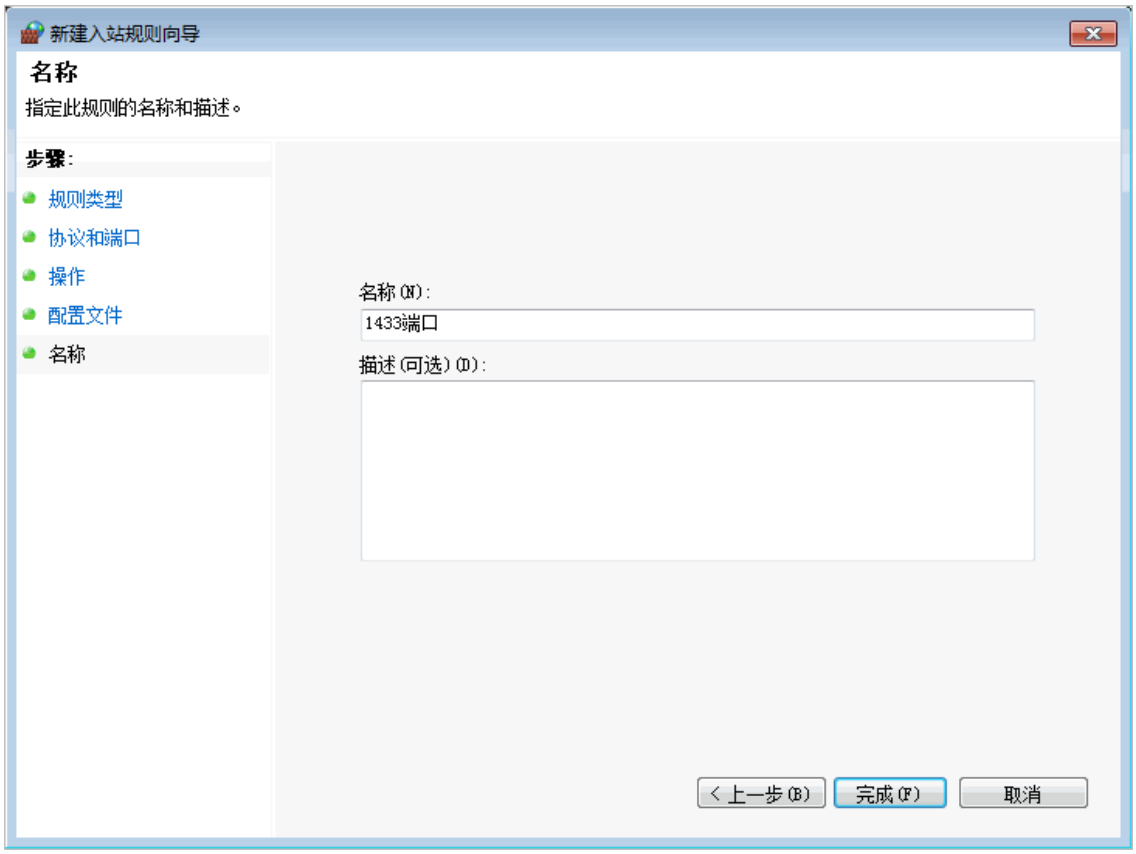
5、选择“允许连接”，单击下一步：



6、根据需要选择规则，单击下一步：

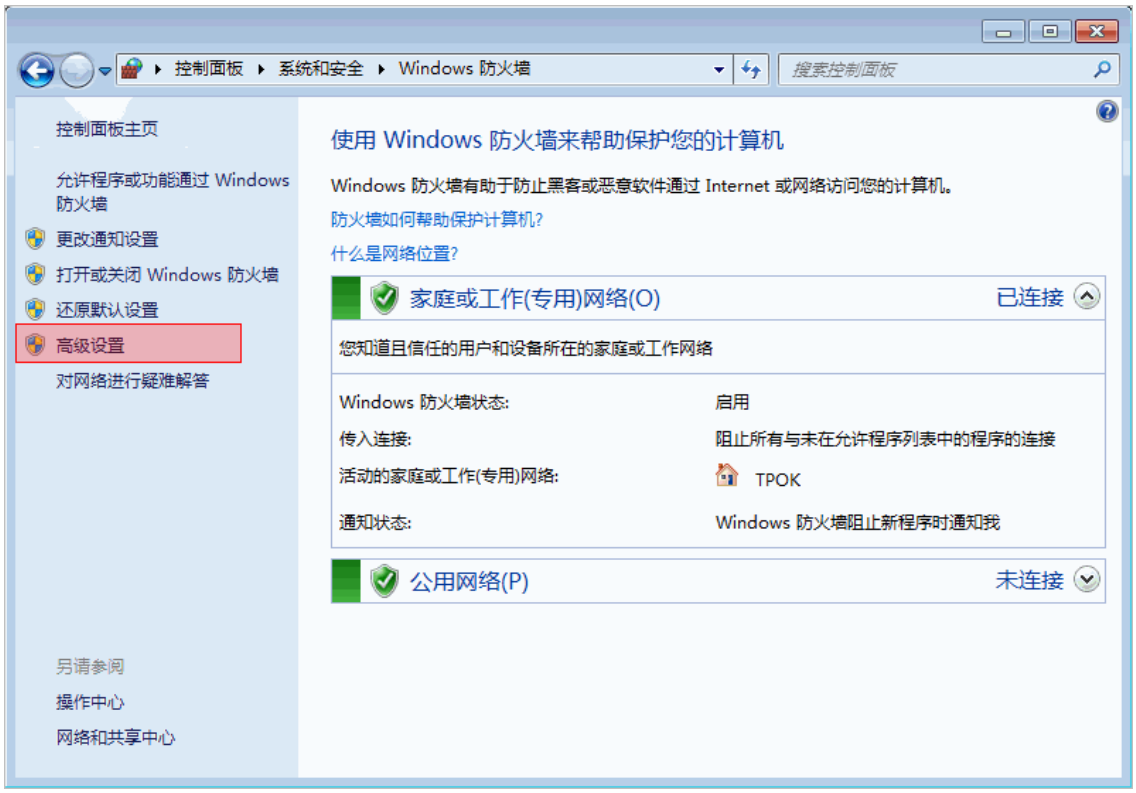


7、输入名称，单击“完成”按钮：

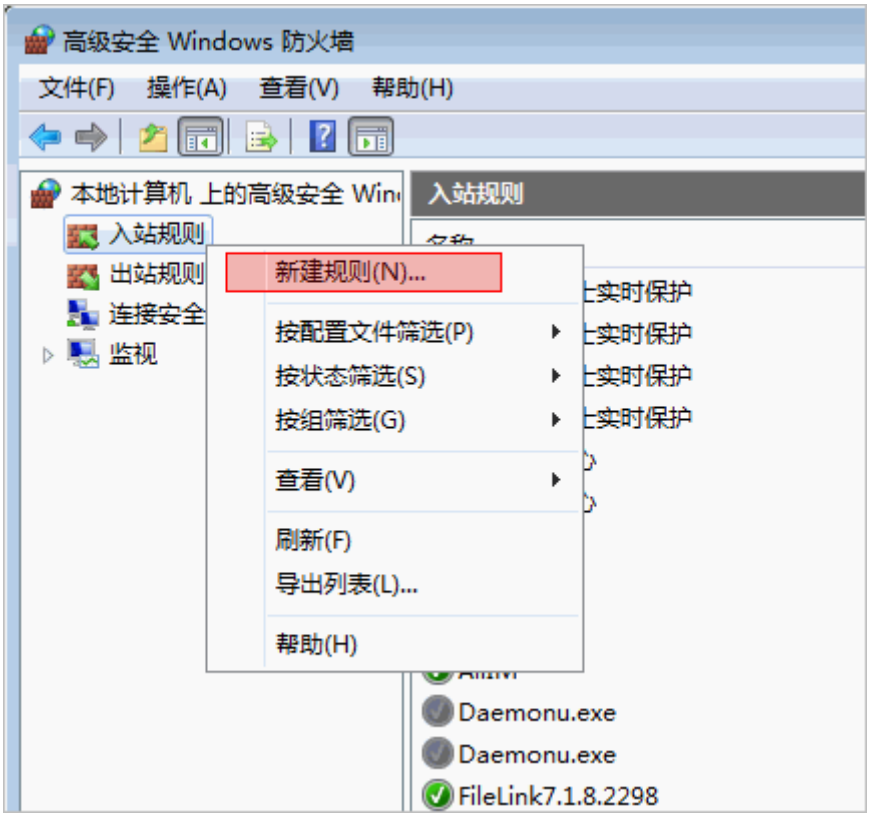


添加程序例外

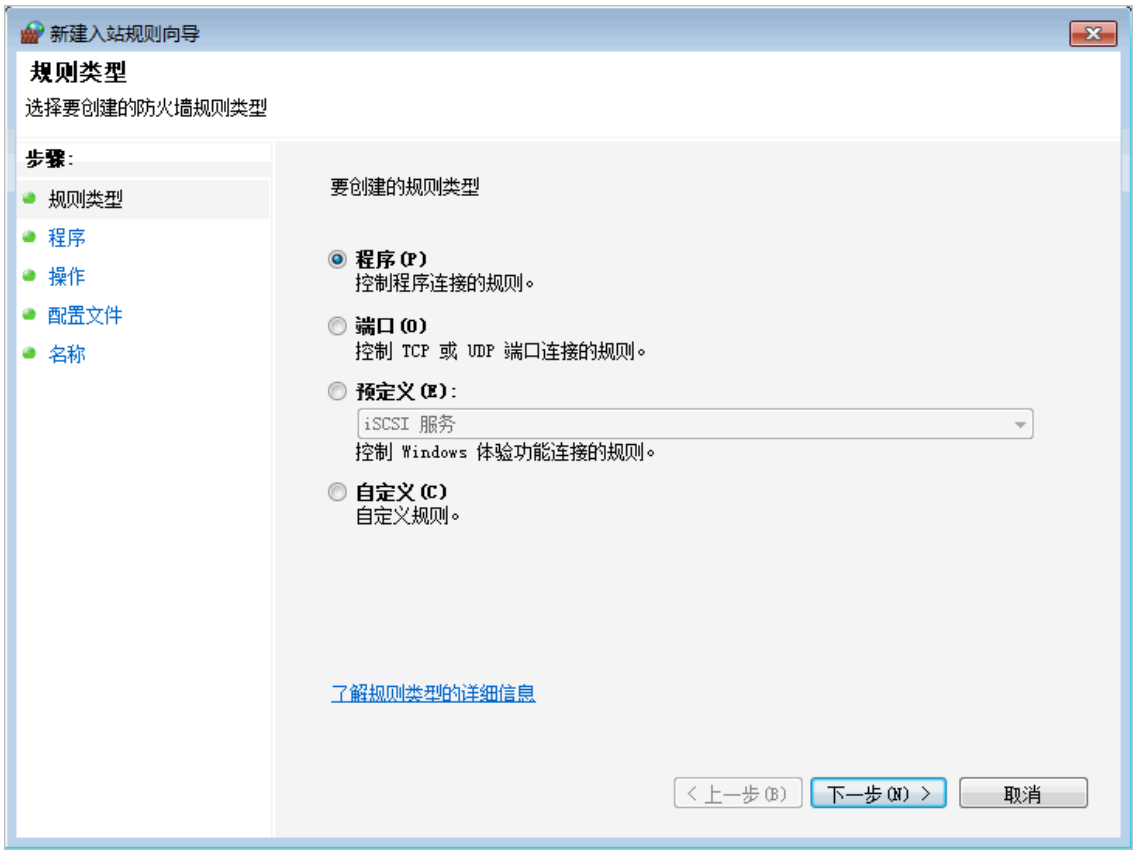
1、打开 Windows 防火墙，单击高级设置：



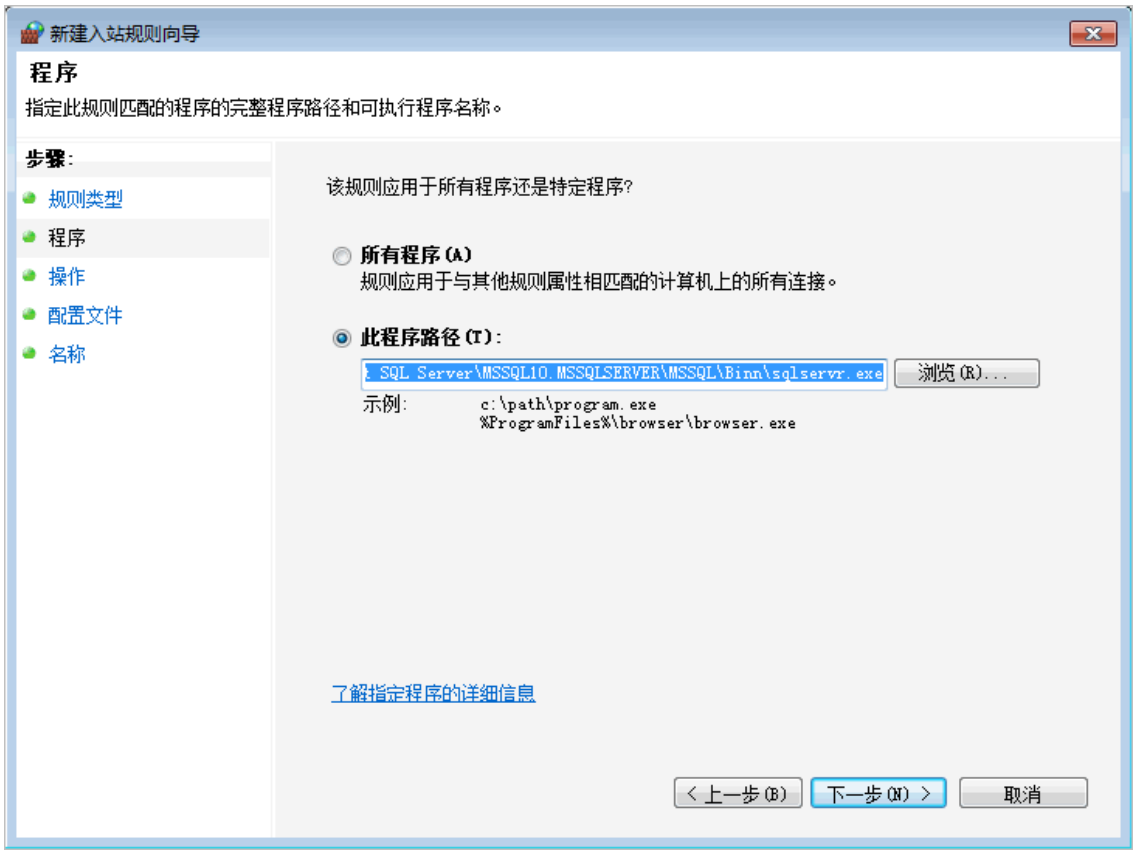
2、在高级设置窗口，先单击“入站规则”，再右击“入站规则”，从快捷菜单执行“新建规则”：



2、规则类型选择“程序”，单击下一步：

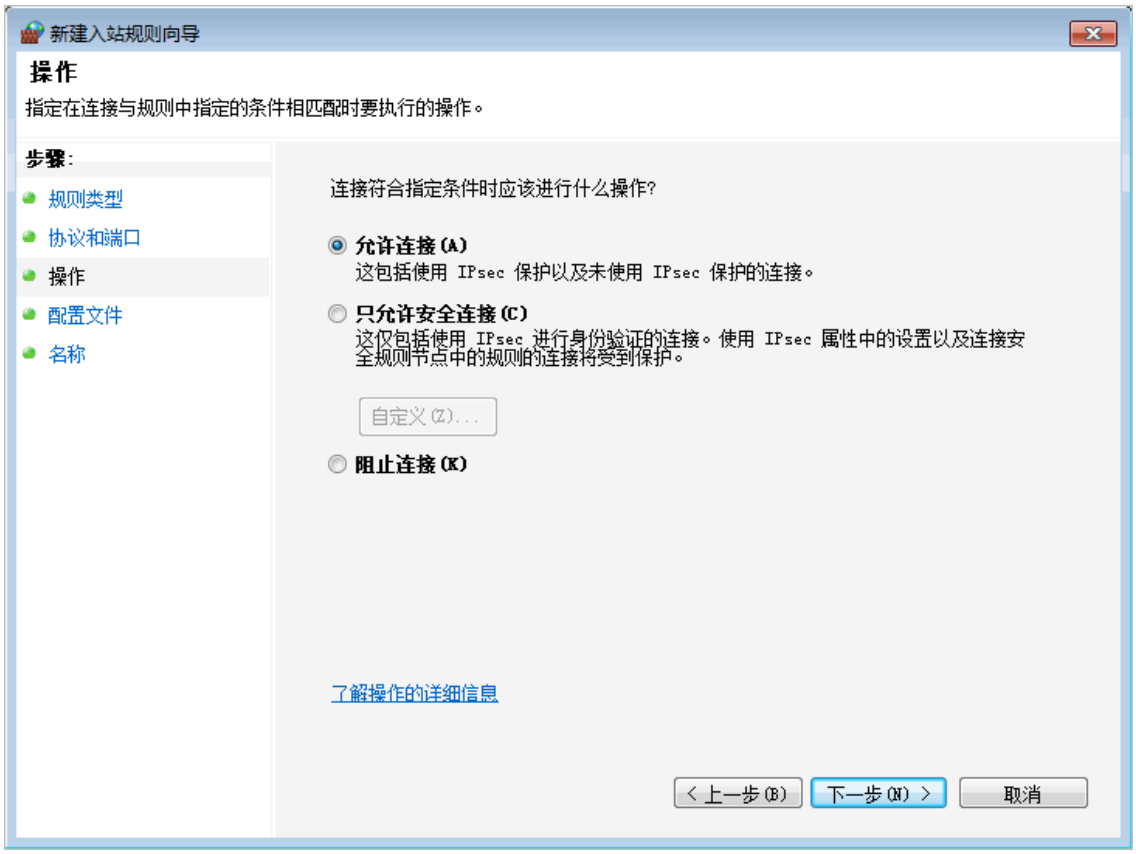


3：程序路径选择“C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\sqlservr.exe”，单击下一步：

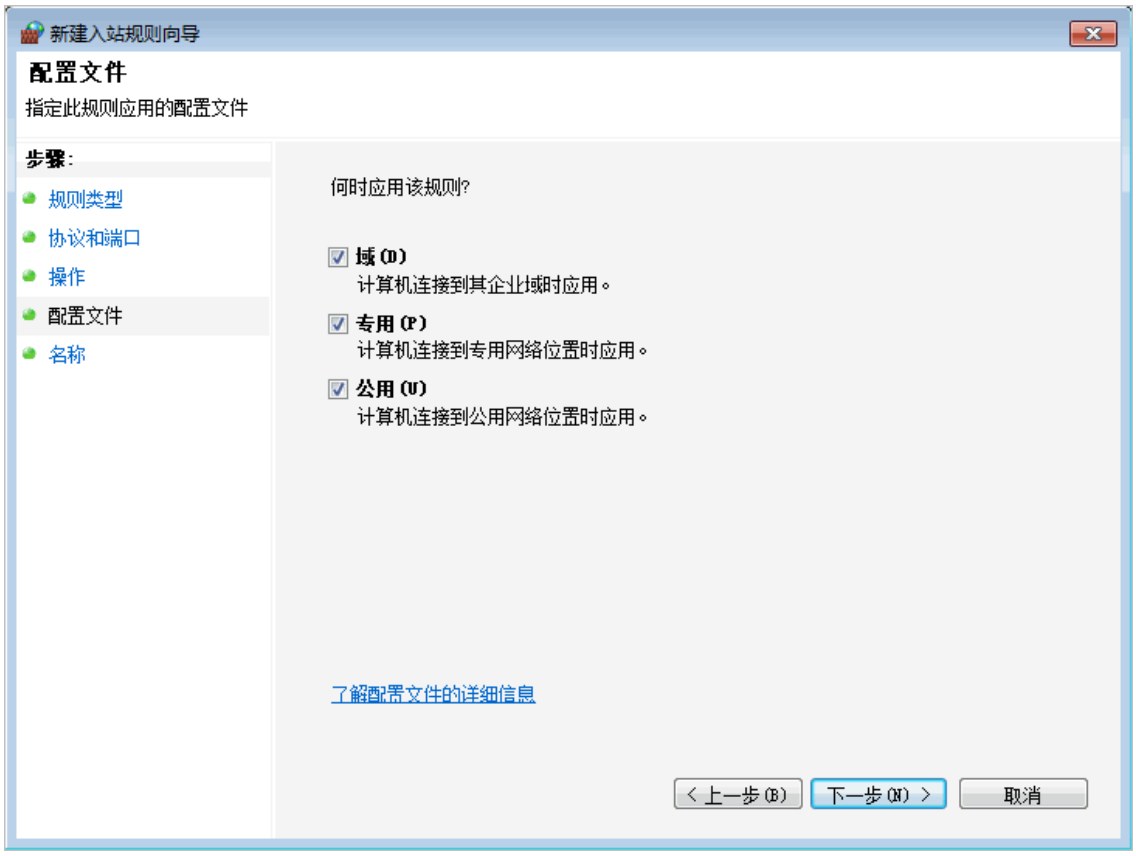


提示：随 SQL Server 的版本和安装路径的不同，此路径会有所不同：

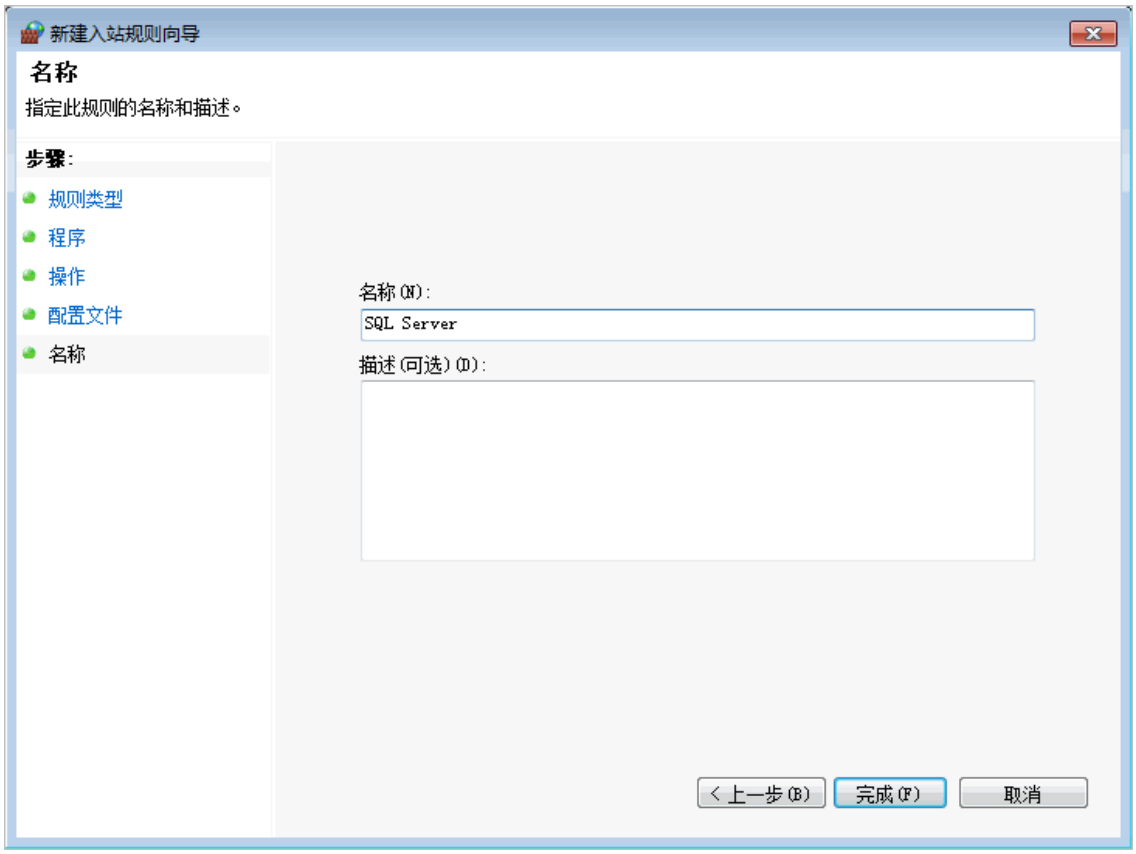
4、选择“允许连接”，单击下一步：



6、根据需要选择规则，单击下一步：



7、输入名称，单击“完成”按钮：



3.17.4 加载外部数据表

3.17.4.1 添加外部数据表

你可以将外部数据源中的表添加到 Foxtable 中，操作步骤为：

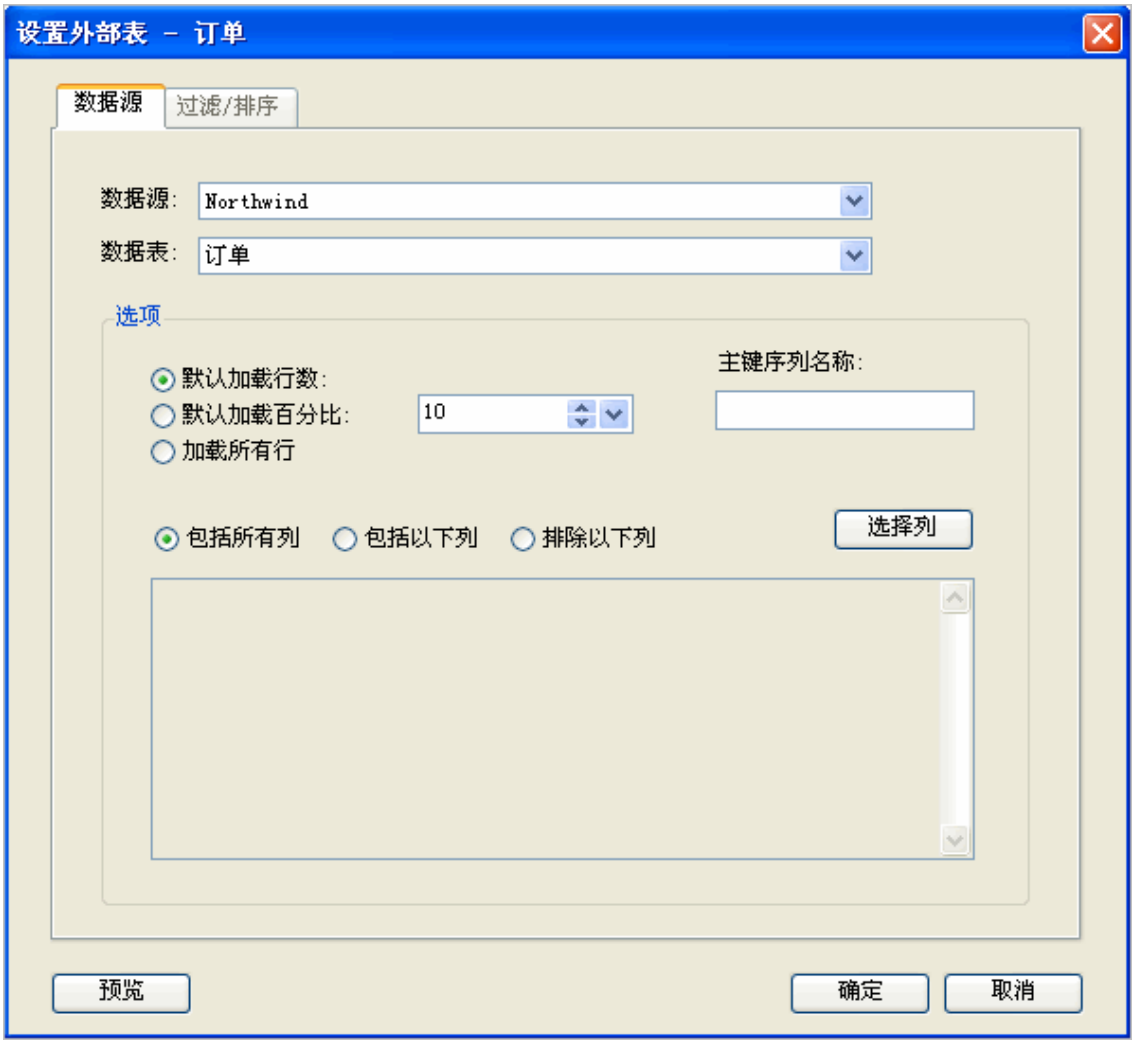
1、在数据表功能区，单击下面的按钮：



2、在外部表管理窗口，单击增加按钮：



3、然后选择数据源和数据表：



4、最后单击确定按钮，输入外部数据表在 Foxtable 中的名称，通常采用和原数据表相同的名称即可。

提示：

只有具备主键的外部数据表，才能加载到 Foxtable 中。

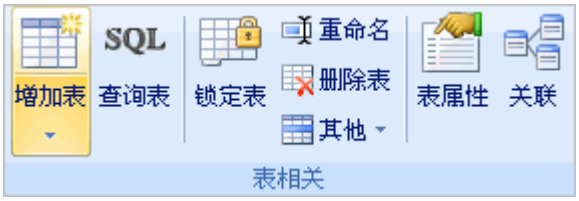
主键尽量使用自动增量列（也称为自动编号），这样外部表在实际使用的时候，表现将和内部表完全一致，用户不会感受到任何差别。

如果主键列不是自动增量列，则每次只能增加一行，且必须输入新增行的主键列内容才能增加下一行。

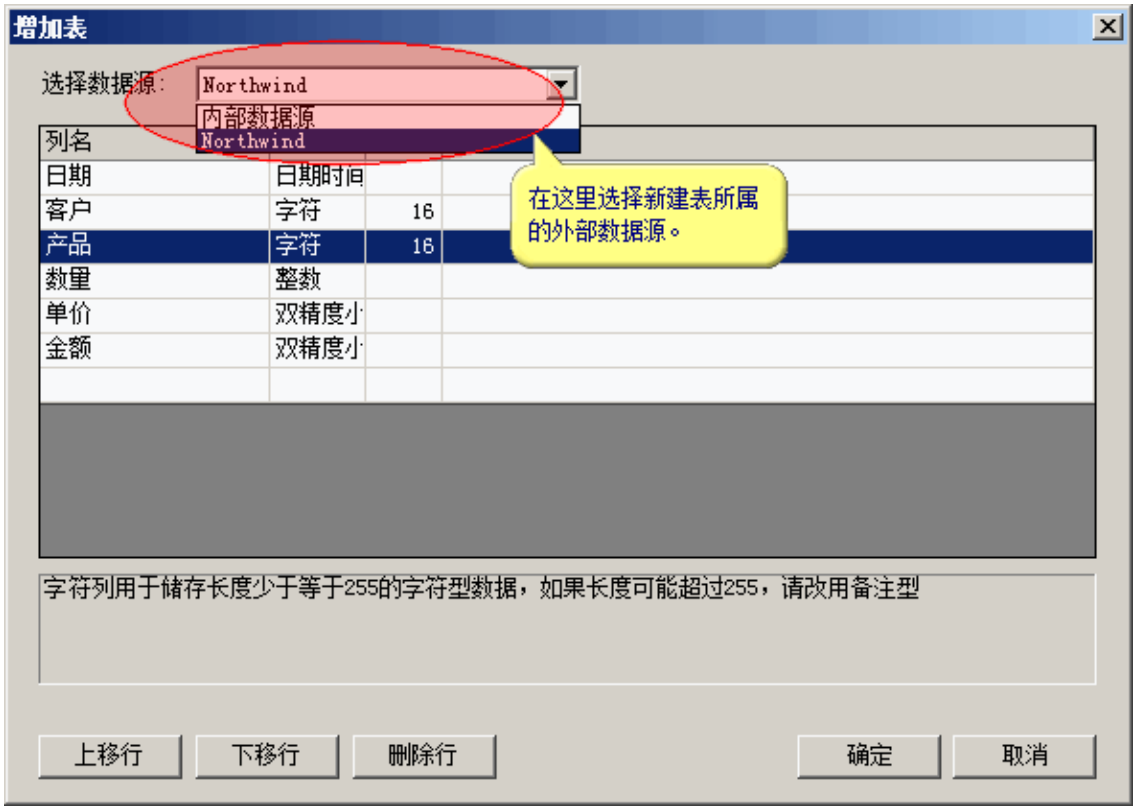
此外，如果你希望外部表的主键列能够象内部表一样自动隐藏，可以将主键列的名称设置为“_Identify”。

3.17.4.2 新建外部数据表

1、在菜单的“数据表”功能区，单击“增加表”按钮：



2、在增加表窗口选择外部数据源：



提示：

通过 Foxtable 新建的外部表，会自动具备名为"_Identify"的自动增量型主键。

3.17.4.3 修改外部表的设置

1、在数据表功能区，单击下面的按钮：



2、在外部表管理窗口，选择某个外部表，单击“编辑”按钮，可以修改外部表的设置：



提示：

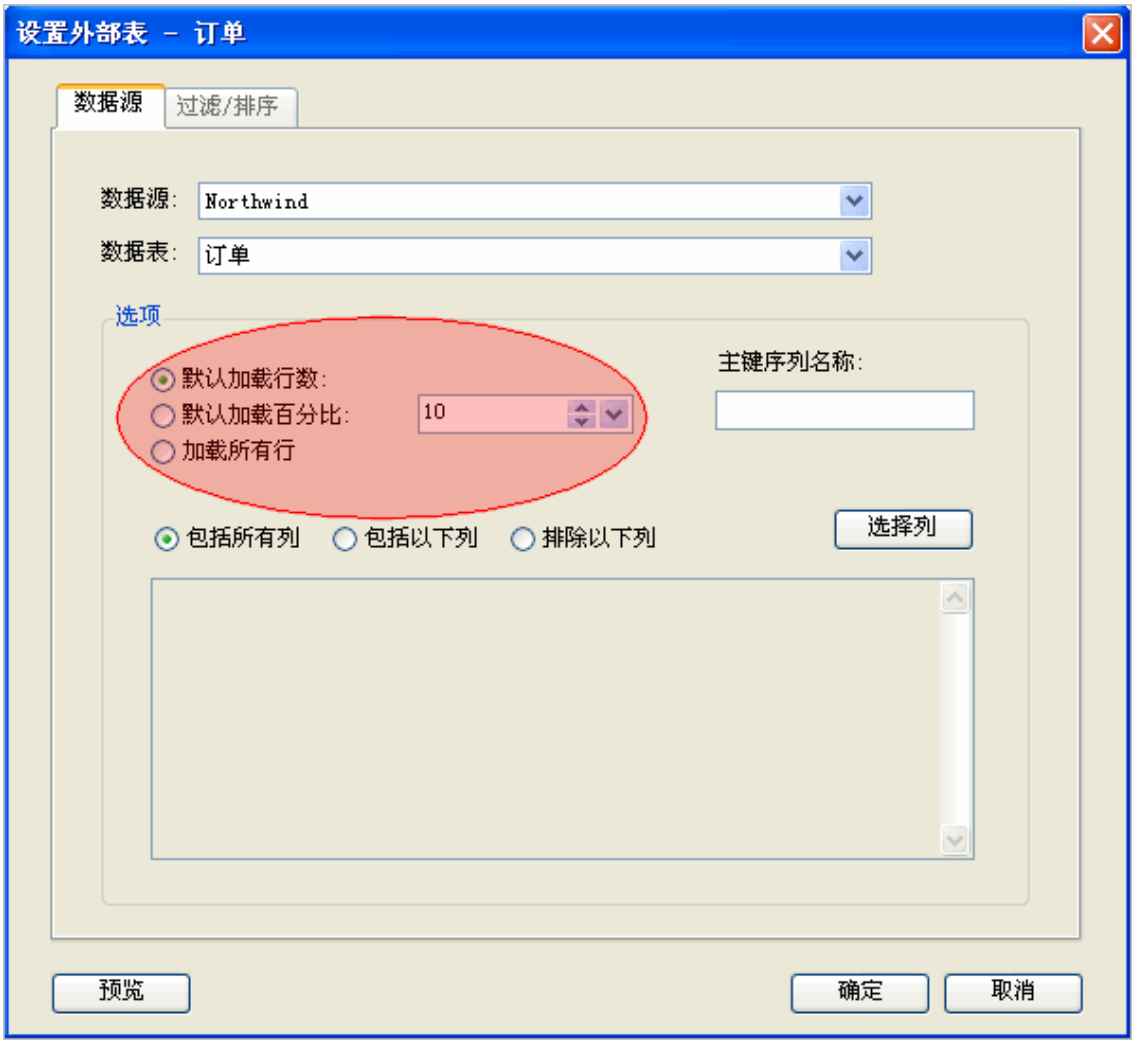
1、如果外部表已经和其它表建立关联，那么在修改设置的时候，一定要包括关联列。

2、通过 Foxtable 直接新建的外部表，也可以在这里进行设置。

3、如果要将外部表重新指向到后台的另一个表，可能还不如先删除现有的外部表，再增加一个外部表合适，因为这样可以彻底清除原外部表的相关设置。

3.17.4.4 控制默认的加载行数

在默认的情况下，外部数据表只加载 10 行数据，你可以修改这个数值，也可以选择全部加载：



在使用 Access 或者 SQL Server 作为数据源的时候，还可以选择按百分比加载。

3.17.4.5 设置默认的加载条件

在设置外部表的时候，可以在"过滤/排序"页，设置一个默认的加载条件，例如希望默认加载 1999 年的订单，可以将加载条件设置为：

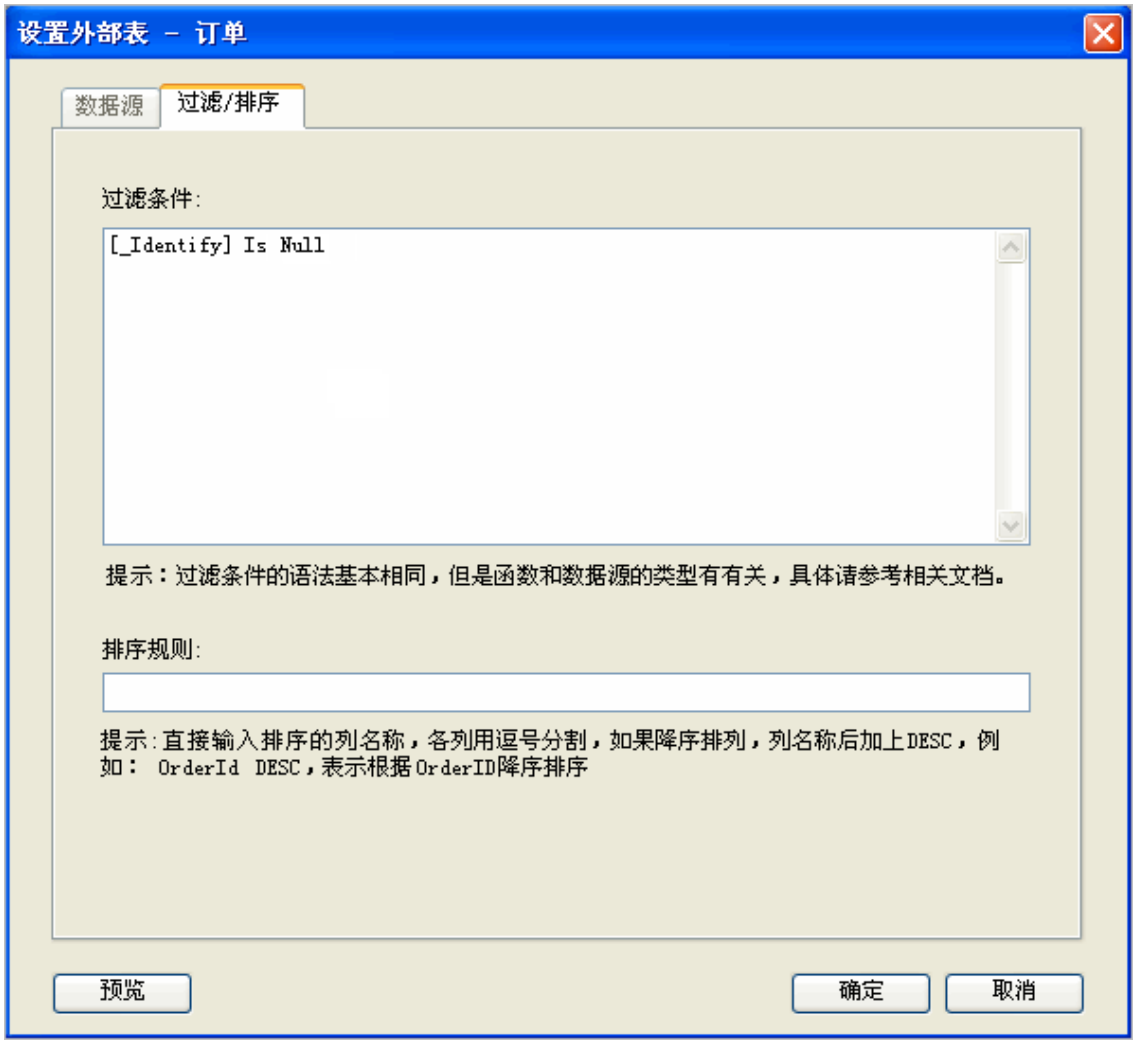
Year(订购日期) = 1999

加载条件是一个条件表达式，需要注意的是，这个表达式和我们前面介绍的表达式不同，其语法和支持的函数，与使用的数据源有极大的关系，我们会在开发篇进行介绍。

很多时候，我们希望默认不加载任何数据，而是在启动项目后，根据用户输入的条件来动态加载数据，此时可以将加载条件设置为：

[主键列名称] Is NULL

因为主键列是行的身份标识，是不可能为空的，所以加载主键列为空的行，等于就是不加载任何行：



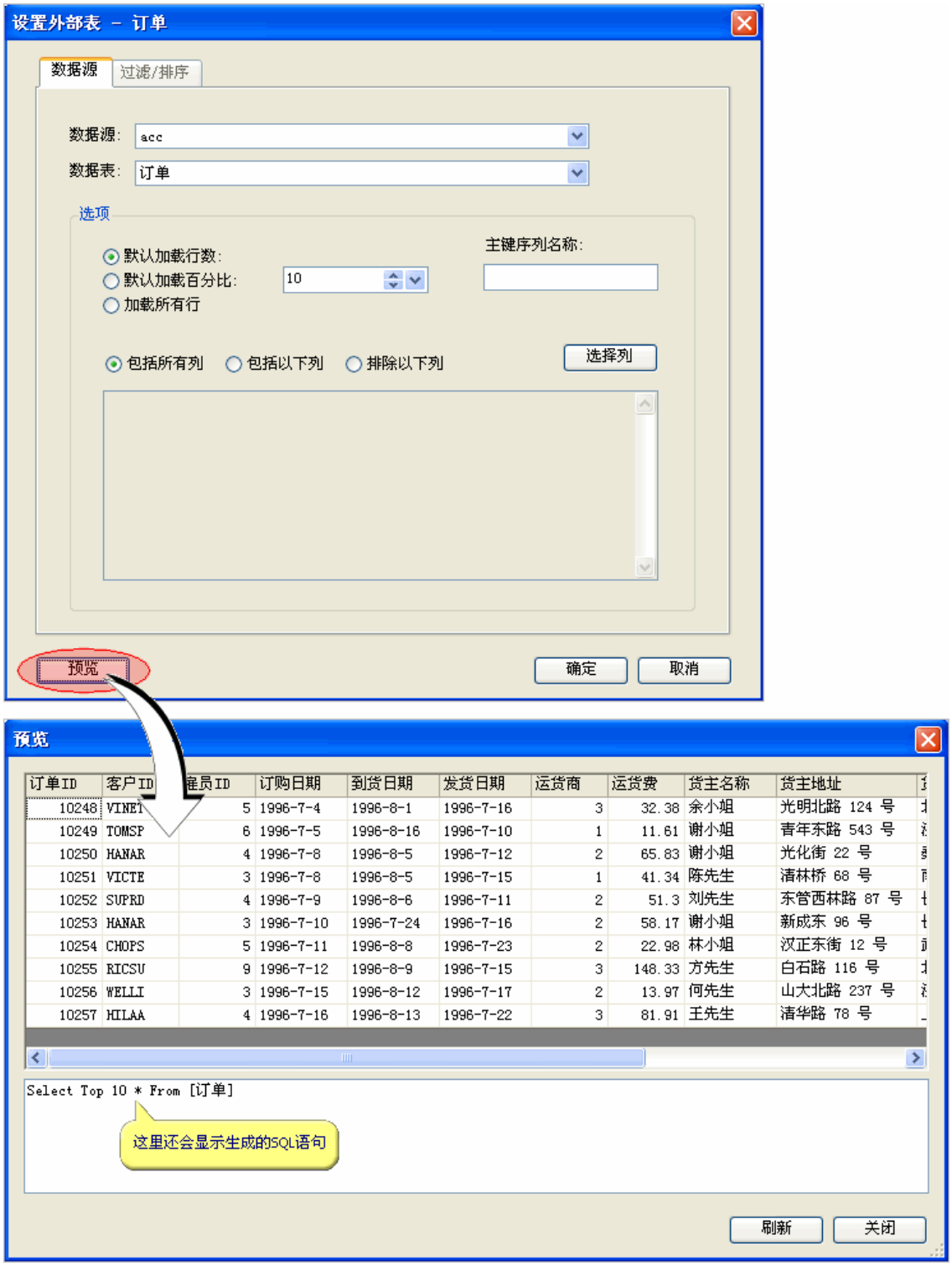
3.17.4.6 加载部分列

外部表默认是加载所有的列，用户可以选择只加载部分列。
需要特别注意的时，如果选择加载部分列，一定要将主键列包括在内。
可以按下图所示操作，选择要加载的列：



3.17.4.7 预览外部表

在设置外部表的时候，可以单击预览按钮查看生成的外部表以及 SQL 语句：



3.17.4.8 重命名外部表

和内部表一样，外部表也可以重命名或者设置新的标题。
你可以直接在外部表管理窗口，重命名外部表。
也可以单击下面的按钮，指定新的表名和标题：



这个表名是外部表加载到 Foxtable 后的名称，不是数据源中的表名，例如你将一个名为“Orders”的表，从 SQL Server 加载到 Foxtable 中，不管你将其命名为“订单”还是“定单”，它在 SQL Server 中的名称还是“Orders”。

如果有公式或代码使用了这个表的表名，那么重命名表可能会导致这些公式和代码无法运行。

所以关于表名，我们应该把握两个原则：

1、事先就要慎重考虑，为每个表设置一个合适的表名，项目一旦启用，不要轻易去修改表名。

2、如果一定要重命名某个表，而且有大量的公式和代码牵涉到这个表，那么应该考虑用修改表标题来代替，例如你需要将“客户”表重命名为“客户资料”，那么可以将该表的标题改为“客户资料”，而表名保持不变，这样就不会影响项目的正常运行了。

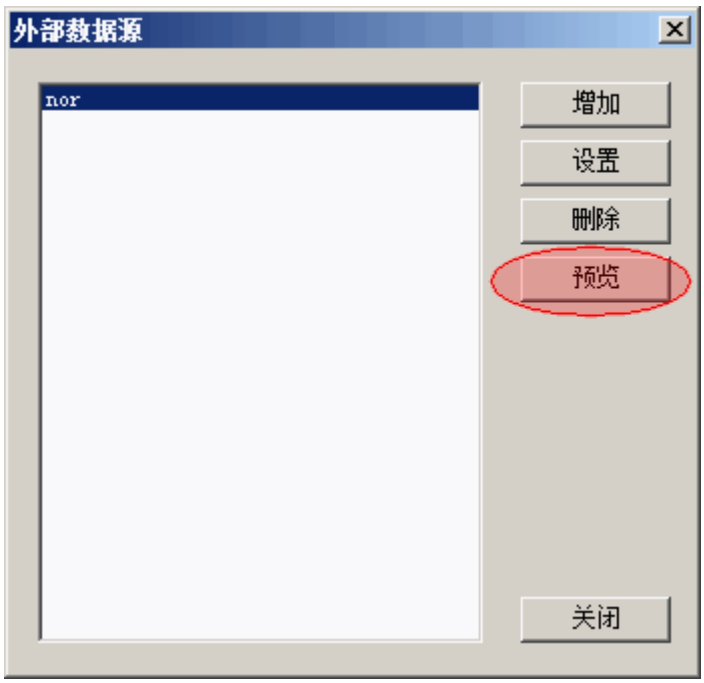
重命名后台表

如果希望直接修改数据源中的表名，操作步骤如下：

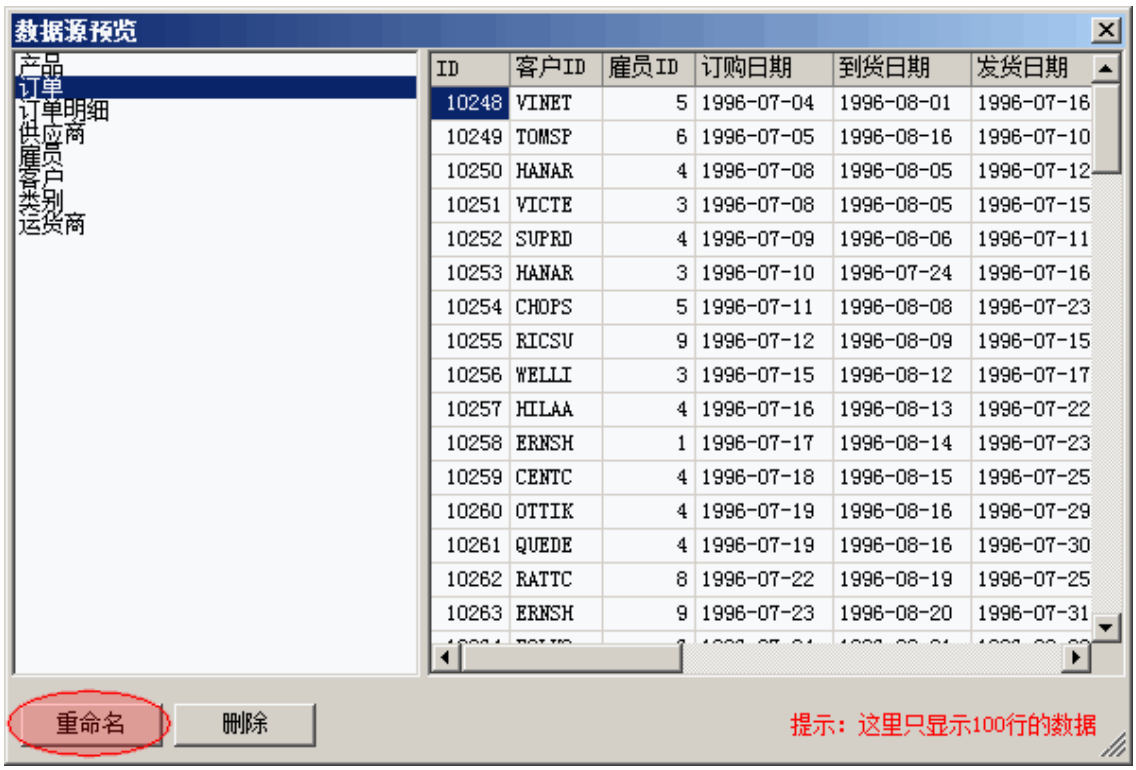
1、在数据表功能区单击下面的按钮：



2、在外部数据源管理窗口，选择此数据表所在的数据源，单击“预览”按钮。



3、选择要改名的数据表，单击“重命名”按钮：



3.17.4.9 删除外部表

在菜单的“数据表”功能区，单击“外部数据表”按钮，打开“外部表管理”窗口，我们可以在这个窗口删除外部表:



此外，不管是内部表还是外部表，都可以直接在菜单的“数据表”功能区，单击下面的菜单按钮来删除：



需要注意的是，这些都不是真正意义的删除，只是下次打开项目的时候，不会再从外部数据源中加载这个表，此表依然存在于数据源中，需要的时候你可以重新将其添加到 Foxtable 中。

删除表的同时，附属于此表的窗口也将被删除，且不可撤销，所以在删除前要慎重考虑。

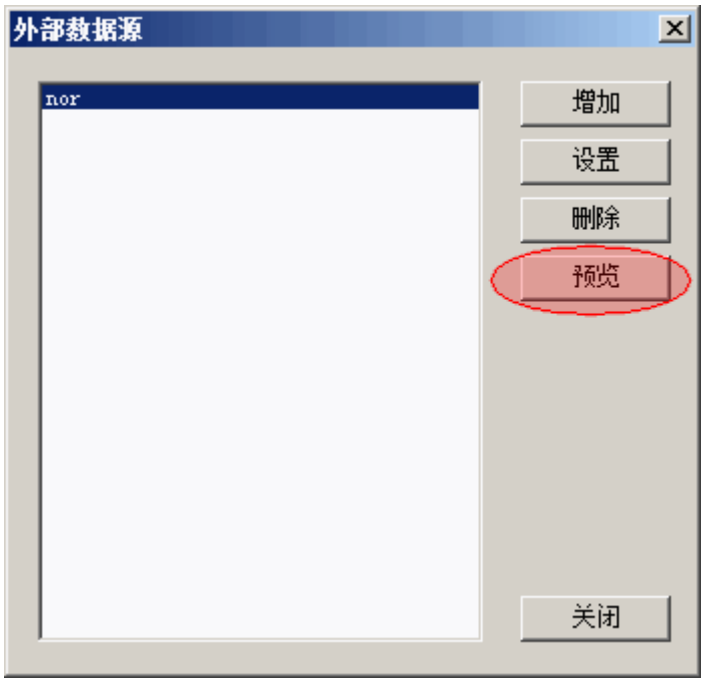
从后台删除表

如果我们要从数据源物理删除某个数据表，操作步骤如下：

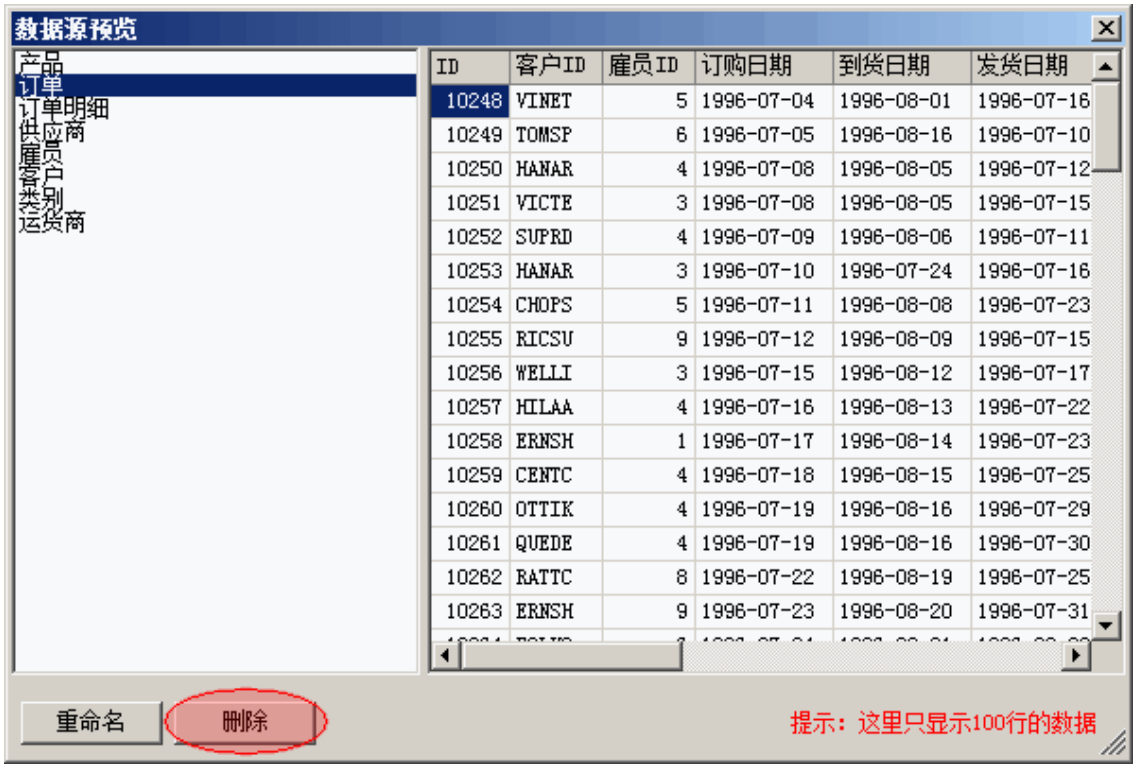
1、在数据表功能区单击下面的按钮：



2、在外部数据源管理窗口，选择此数据表所在的数据源，单击“预览”按钮。



3、选择删除的数据表，单击“删除”按钮：



3.17.4.10 修改外部表的结构

你可以直接通过菜单修改外部表的结构，包括增加、删除、重命名列，或者更改列类型，这一切和内部表没有任何差别。

3.17.4.11 关闭多层表头

如果列名包括下划线(_)，列标题默认会分多层显示，例如有结构如下图所示的外部表：

列名	类型	长度
EMPLOYEE_ID	短整数	
FIRST_NAME	字符型	20
LAST_NAME	字符型	25
EMAIL	字符型	25
PHONE_NUMBER	字符型	20
HIRE_DATE	日期时间型	
JOB_ID	字符型	10
SALARY	双精度小数	
COMMISSION_PCT	双精度小数	
MANAGER_ID	短整数	
DEPARTMENT_ID	短整数	

加载到 Foxtable 中后会显示为：

EMPLOYEE ID	FIRST NAME	LAST NAME	EMAIL	PHONE NUMBER	HIRE DATE	JOB ID	SALARY	COMMISSION PCT	MANAGER ID	DEPARTMENT ID
100	Steven	King	SKING	515.123.45	1987-06-17	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.45	1989-09-21	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.45	1993-01-13	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.45	1990-01-03	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.45	1991-05-21	IT_PROG	6000		103	60
105	David	Austin	DAUSTIN	590.423.45	1997-06-25	IT_PROG	4800		103	60
106	Valli	Pataballa	VPATABAL	590.423.45	1998-02-05	IT_PROG	4800		103	60
107	Diana	Lorentz	DLORENTZ	590.423.55	1999-02-07	IT_PROG	4200		103	60

显然这样的显示并不符合设计初衷，当然我们可以修改列名来避免这个问题，但如果表是事先已经设计好，而且需要和其他软件共用，可能不允许重新修改列名，我们可以用两个方法解决这个问题：

1、在表属性设置中，将“启用多层表头”属性设置为 False，这样表头将显示为单层：

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	1987-06-17	AD_PRES	24000			90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	1989-09-21	AD_VP	17000		100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	1993-01-13	AD_VP	17000		100	90
103	Alexander	Hunold	AHUNOLD	590.423.4567	1990-01-03	IT_PROG	9000		102	60
104	Bruce	Ernst	BERNST	590.423.4568	1991-05-21	IT_PROG	6000		103	60
105	David	Austin	DAUSTIN	590.423.4569	1997-06-25	IT_PROG	4800		103	60

2、或者给列设置标题，例如将列 Last_Name 的标题设置为 LastName。

3.17.4.12 恢复致命的错误设置

如果外部表的数据量较大，高达百万行，甚至千万行，而你不幸在增加外部表的时候，选择了全部加载，那么迎接你的将是漫长的等待，甚至是系统崩溃。

如果遇到这种情况，可以中断 Foxtable 进程，重新启动 Foxtable 后，按住 Ctrl 键，然后打开你的项目文件，直到项目文件被完全打开，再松开 Ctrl 键；如果打开的过程中出现了登录窗口，先松开 Ctrl 键，选择以开发者身份登录，输入密码，然后重新按住 Ctrl 键，单击登录窗口的确定按钮，直到项目文件被完全打开，再松开 Ctrl 键。

以这种方式打开项目文件，外部表不会加载任何数据，正常进入项目后，打开外部表管理窗口，

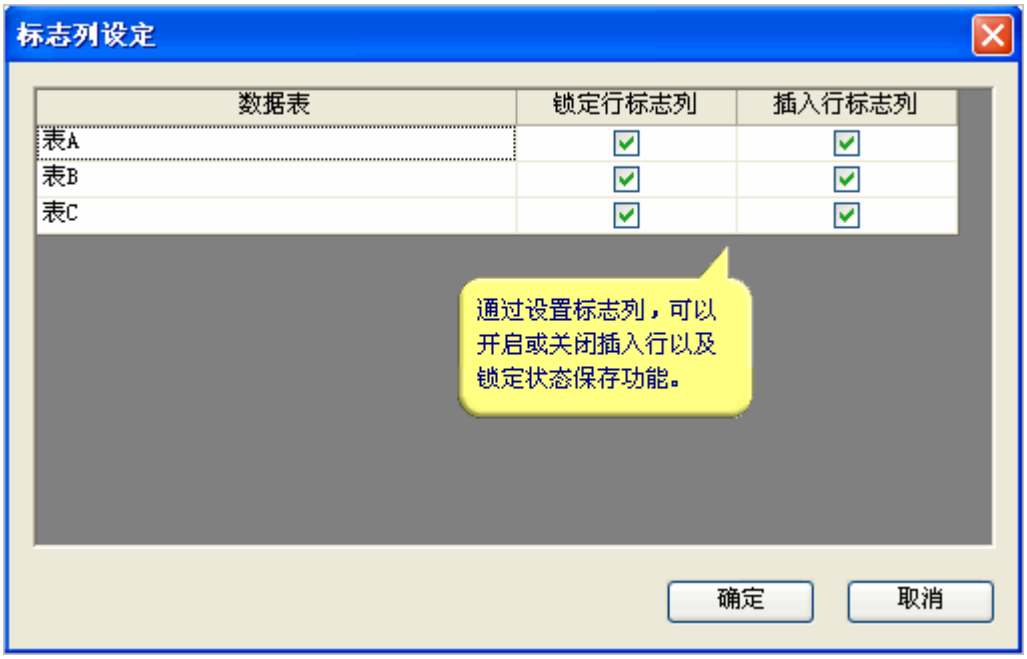
重新设置要加载的行数或者加载条件即可。

3.17.4.13 设置标志列

在默认情况下，外部表只能增加行，而没有插入行的功能，此外也不能保存行的锁定状态。
不过通过设置标志列，可以开启或者关闭上述功能。
1、在菜单的“数据表”功能区，单击“设置标志列”按钮：



2、根据需要选择要开启的功能：



3、Foxtable 分别用一列来保存行的锁定状态及其位置，所以除非确有必要，否则请不要开启这些功能，以节省资源。

3.17.4.14 关于二进制字段

很多数据库软件都有二进制类型的列（字段），例如 Access 的 Ole 类型，SQl Server 的 binary 和 image 类型，Oracle 的 blob 类型。
出于效率和易用双方面的考虑，Foxtable 不建议使用二进制字段，而是建议采用附件的形式管理图片以及各种文档。
Foxtable 内置的文件和图片管理，不仅既支持本地储存，也支持远程 FTP，而且性能和功能都强于二进制列。
所以大家在增加外部表的时候，务必要排除二进制类型的列。
有关文件和图片的管理，请参考帮助“使用指南 → 列相关 → 扩展列类型 → 文件与图片的管理”。

3.17.5 使用 Oracle 作为数据源

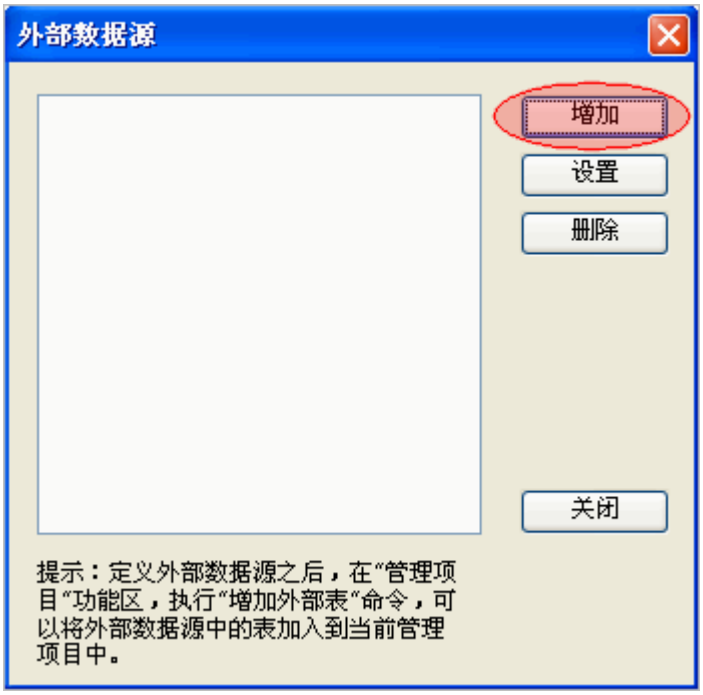
3.17.5.1 连接到 Oracle

虽然我们推荐的数据源是 Access 和 SQL Server，但考虑到 Oracle 在整个行业中的地位，也为了便于用户整合现有的管理系统，Foxtable 最终还是提供了对 Oracle 的支持。

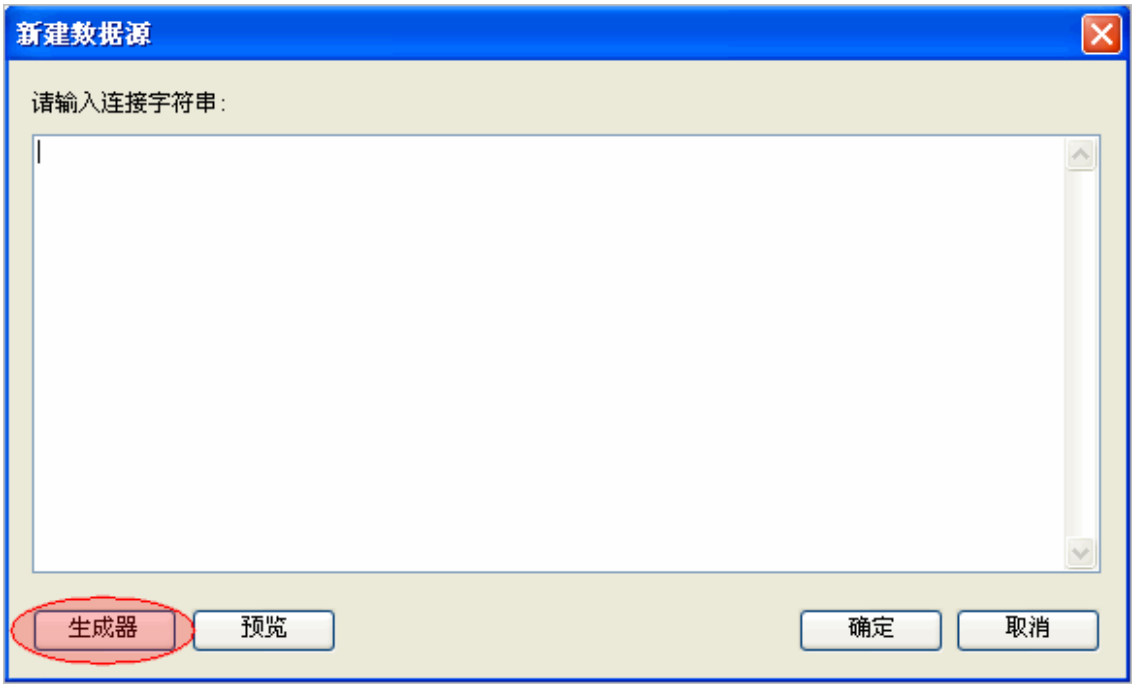
1、在数据表功能区单击下面的按钮：



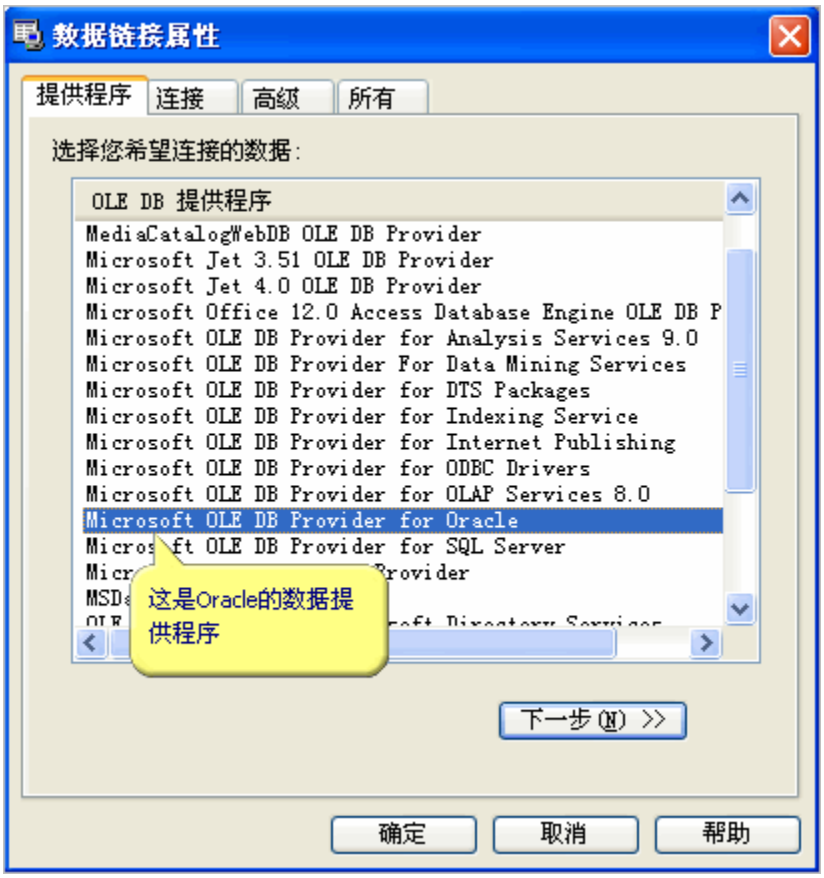
2、在外部数据源管理窗口，单击“增加”按钮：



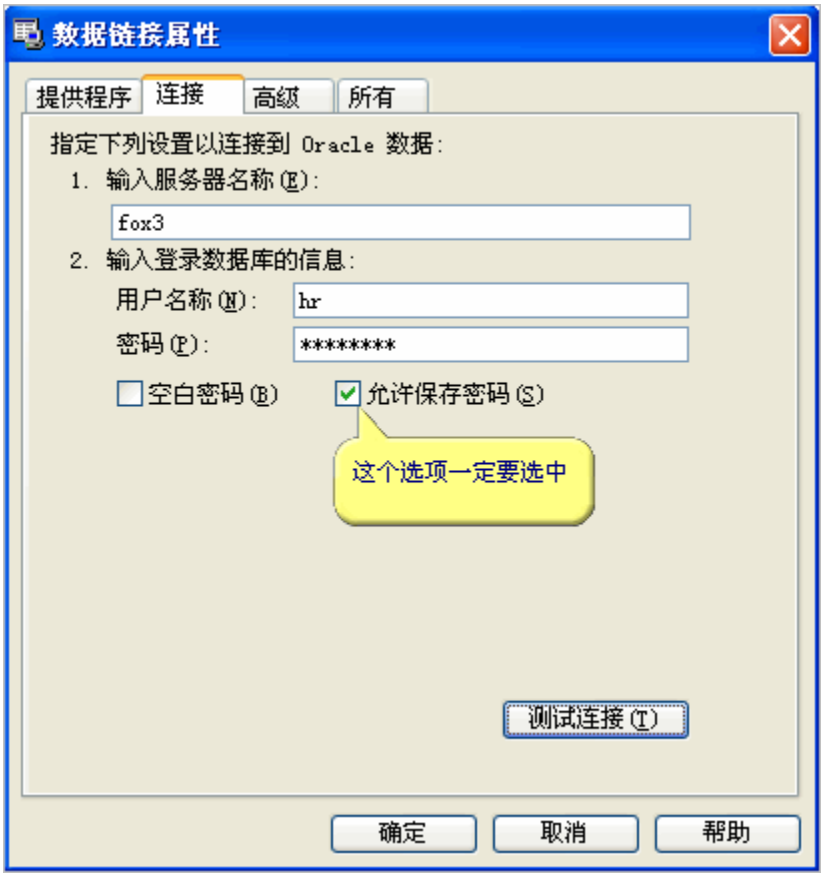
3、在新建数据源窗口单击按钮“生成器”：



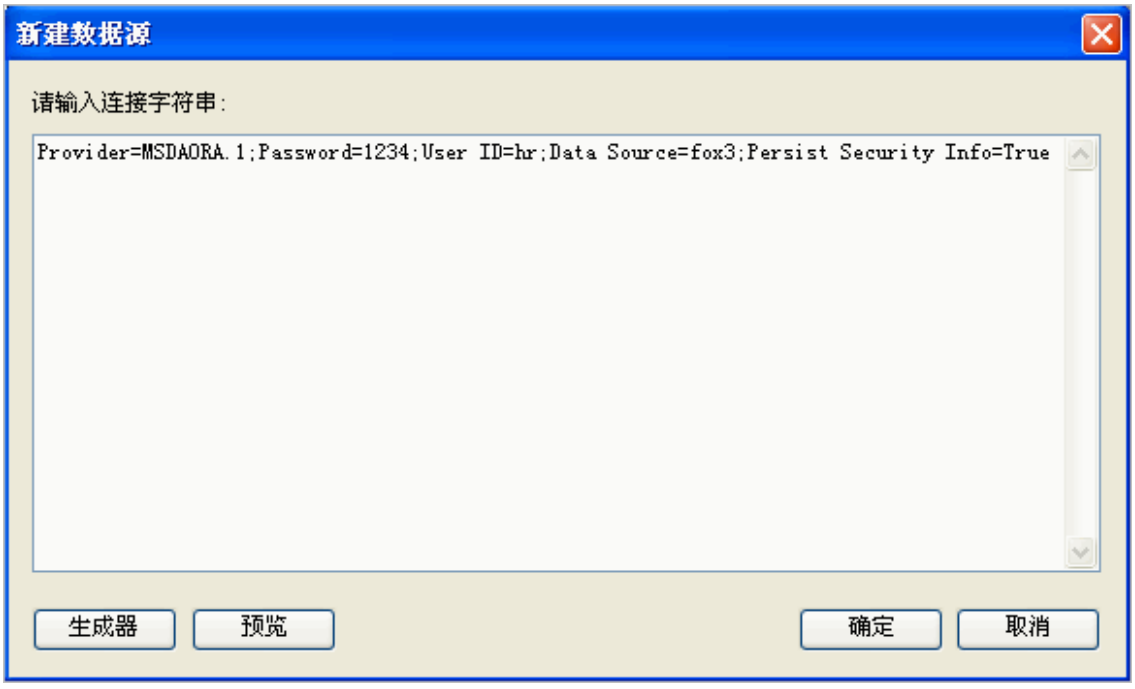
4、在数据链接属性设置窗口，选择下图所示的数据提供程序：



5、输入服务器名称、用户名称及其密码，并选中选项“允许保存”密码：



6、设置完成后，单击“确定”按钮，即可看到生成的链接字符串：



7、单击“确定”按钮，输入数据源名称，完成设置。

3.17.5.2 主键名称与大小写

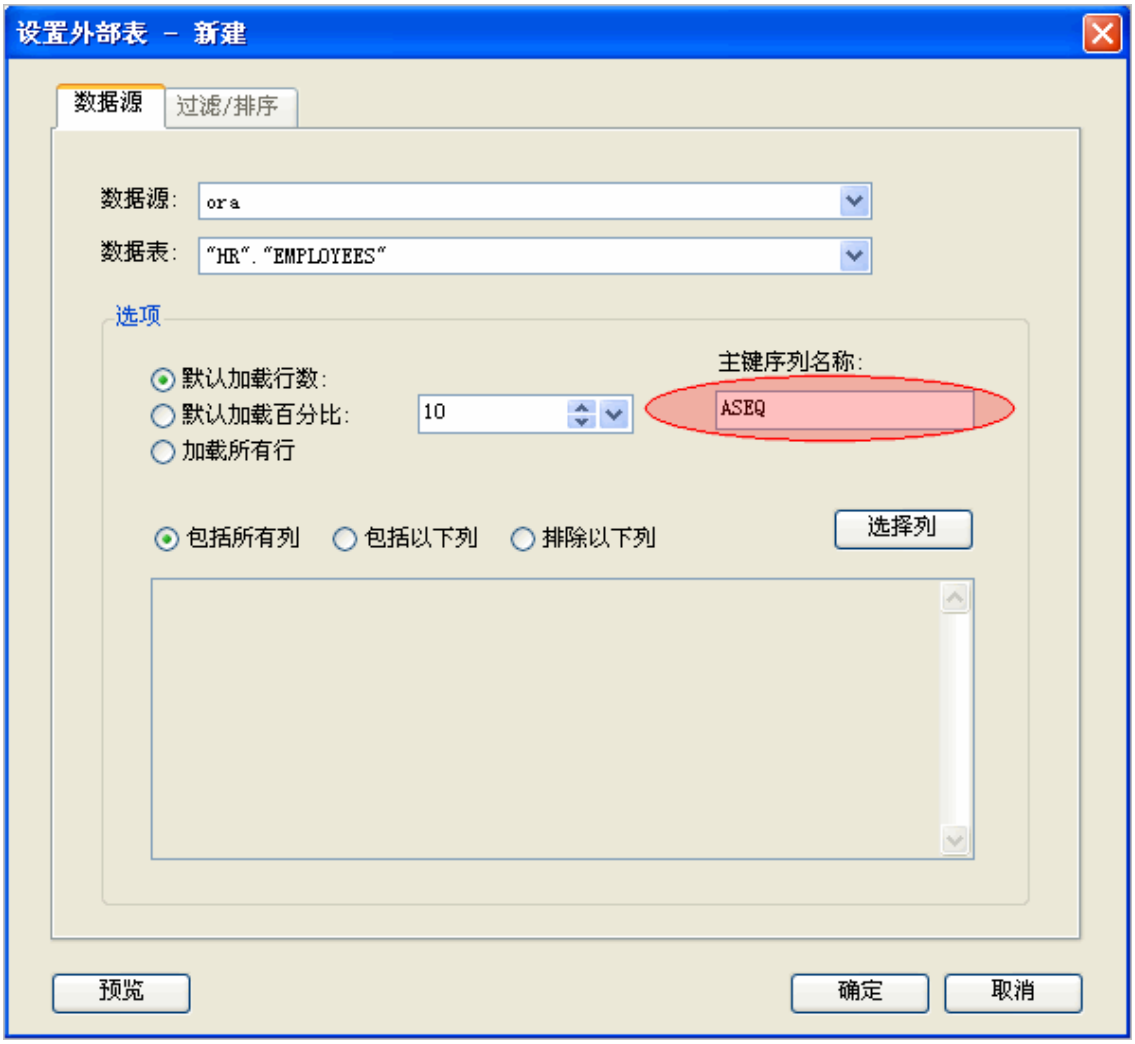
我们已经知道，如果希望外部表的主键列能够象内部表一样自动隐藏，可以将主键列的名称设置

为“_Identify”，注意是区分大小写的。
在 Oracle 中设计表的时候，列名默认会自动转为大写，对于区分大小写的列名，需要用双引号括起来：



3.17.5.3 使用序列

Oracle 没有自动增量列，必须为每个表创建一个序列，在增加外部表的时候，为该表指定一个序列名称。
这样保存新增行的时候，Foxtable 会自动根据该序列生成主键值，使用起来和自动增量列完全一样：



如果数据表已经有数据，那么创建序列的时候，其初始值应该大于主键列目前的最大值，道理是显而易见的。

3.17.6 无法连接数据源时

在打开项目的时候，如果无法连接到某个数据源，会出现一个数据源设置窗口，允许临时修改数据源的设置。

必须注意的是：

1、这种设置默认是临时的，启动项目后，需要重新打开外部数据源管理窗口，重新设置数据源。

2、如果希望保存这种设置，可以在设置窗口选中选项“保存设置”，这样下次打开项目之后，会按照新的设置连接数据源，无需重复设置。

3、只有开发者和管理员，才有这种临时设置的权限，其他用户打开项目的时候，如果某个数据源无法连接，将直接退出。

4、如果不希望管理员有这种临时修改数据源的权限，可以在项目属性设置中，将“允许设置数据源”属性设置为 False(否)。

3.17.7 隐藏共享文件夹

我们知道，在局域网环境下使用 Access 数据源，必须共享服务器中存放 Access 数据文件的文件夹。
一些用户可能会担心共享文件夹，数据文件存在被恶意删除的可能性。
为了降低共享文件夹带来的风险，我们可以隐藏共享文件夹。
隐藏共享文件夹的方法：
重命名共享文件夹，名称以符号“$”结尾，这样别人在“网络邻居”里就不会看见此共享文件夹，要访问该文件夹时，只能直接输入地址，例如：\\HEHUI\Data$

3.17.8 数据源的转换

在网络环境下，使用外部数据源，能带来更好的性能和安全性，而且在互联网环境下，是必须使用外部数据源的。
如果你已经基于内部表设计好了管理项目，那么如何将其转换为外部数据源呢？
Foxtable 提供了重定向的功能，你只需单击几次鼠标，即可将基于内部数据表设计的项目转换为基于 Access 或者 Sql Server 的数据源，也可以将基于 Access 或者 Sql Server 数据源设计的项目，转换为基于内部数据表。
重定向之前必须备份你的项目，理由：
1、一旦重定向中途失败，可能导致你的项目再也无法回到正常状态。
2、内部数据表重定向到外部数据源之后，会被自动删除。
3、Foxtable 在重定向之前，也会自动备份项目，但是这不能代替你自己的备份，多一个备份，多一个保险

转换为 Access 2000/2003 数据源

1、首先打开项目，然后在菜单的“杂项”功能区，执行下图所示的“ToAccess”命令：



执行此命令后，会生成一个 Access 文件，此文件包括项目文件中所有的内部数据表及其数据。
2、将刚刚生成的 Access 文件复制到服务器上，新建一个外部数据源，链接到服务器上的这个 Access 文件，请参考：链接到 Access 2000/2003
3、假定新建数据源的名称为“FishMIS”，在杂项功能区，执行"Redirect"命令，将内部表全部重定向到这个新建的数据源：



现在你在局域网中的任何一台电脑打开此目，数据表都来自于服务器上的 Access 文件，由于只需传输数据，这比直接共享 Table 文件要高效很多。

转换为 Access 2007/2010 数据源

ToAccess 生成的文件是 Access 2000/2003 格式，如果要转换成 Access 2007/2010 格式的数据源，操作步骤有所不同：

1、首先执行打开项目，然后在菜单的“杂项”功能区，执行下图所示的“ToAccess”命令：



执行此命令后，会生成一个 Access 文件，此文件包括项目文件中所有的内部数据表及其数据。

2、利用 Access 2007 或者 Access 2010 打开刚刚生成的 Access 文件，另存为 Access 2007/2010 格式，然后将这个文件复制到服务器上。

3、新建一个外部数据源，链接到服务器上的这个 Access 文件，请参考：链接接到 Access 2007/2010

4、假定新建数据源的名称为“FishMIS”，在杂项功能区，执行"Redirect"命令，将内部表全部重定向到这个新建的数据源：



转换为 SQL Server 数据源

如果要在互联网中应用基于 Foxtable 开发的管理项目，或数据量非常庞大，或有更高的要求，建议采用 SQL Server 作为数据源

1、首先执行打开项目，然后在菜单的“杂项”功能区，执行下图所示的“ToAccess”命令：



执行此命令，会生成一个 Access 文件，此文件包括项目文件中所有的内部数据表及其数据。

2、在服务器的 SQL Server 上新建一个空数据库。

3、在 Foxtable 中新建一个外部数据源，链接到服务器上刚刚新建的数据库，请参考：链接到 SQL Server

4、假定新建数据源的名称为“FishMIS”，在杂项功能区，执行"Redirect"命令，将内部表全部重定向到这个新建的数据源：



5、重新打开项目后，你会发现所有的表都是无数据的，你还需要通过合并表功能，从第一步生成的 Access 文件中将相关表的数据合并进来。

3.17.9 数据类型对照

你可以直接在 Foxtable 中新建外部表，修改外部表的结构，这一切和内部表没有任何差别。
当然你也可以直接在外部数据源，例如 Access 或者 Sql Server 中设计好表，然后添加到 Foxtable 中，下表是常用数据源的数据类型对照，在设计表的时候可以参考，其中 Visual Basic 不是数据源，而是一个开发语言，我们会在开发篇介绍：

内部表	Access	SQL Server	Visual Basic
字符	文本	nvarchar	String
备注	备注	ntext	String
日期	日期/时间	datetime	Date
逻辑	是/否	bit	Boolean
整数	长整型	int	Integer
无此类型	无此类型	bigint	Long

短整数	整型	smallint	Short
微整数	字节	tinyint	Byte
双精度小数	双精度型	float	Double
单精度小数	单精度型	real	Single
高精度小数	小数	numeric	Decimal

注意：

如果使用 SQL Server 2008 作为数据源，对于日期时间型列，请使用 datetime 或者 samlldatetime 数据类型，不要使用 date、datetime2、datetimeoffset、time 数据类型， 因为这四种类型 Foxtable 无法正常识别，会当做字符型列处理。

3.17.10 外部数据源与用户管理

在默认情况下，Foxtable 的用户名、用户密码、用户分组、用户角色等数据保存在项目文件中。在多用户环境下，增加、删除、更改用户后，需要更新各终端的项目文件，这对于大型的应用系统，特别是远程应用的时候，是一件很麻烦的事情。

不过从 Foxtable 2013 开始，用户可以很方便地使用外部数据表存储用户信息，要做的只是单击几次鼠标。

步骤

1、首先要在外部数据库中新建一个数据表，用于保存用户信息，这个表必须包括三列，分别是：

列名	数据类型	说明
Name	字符型	必须将此列设置为主键！
Type	整数型	
Config	备注型	

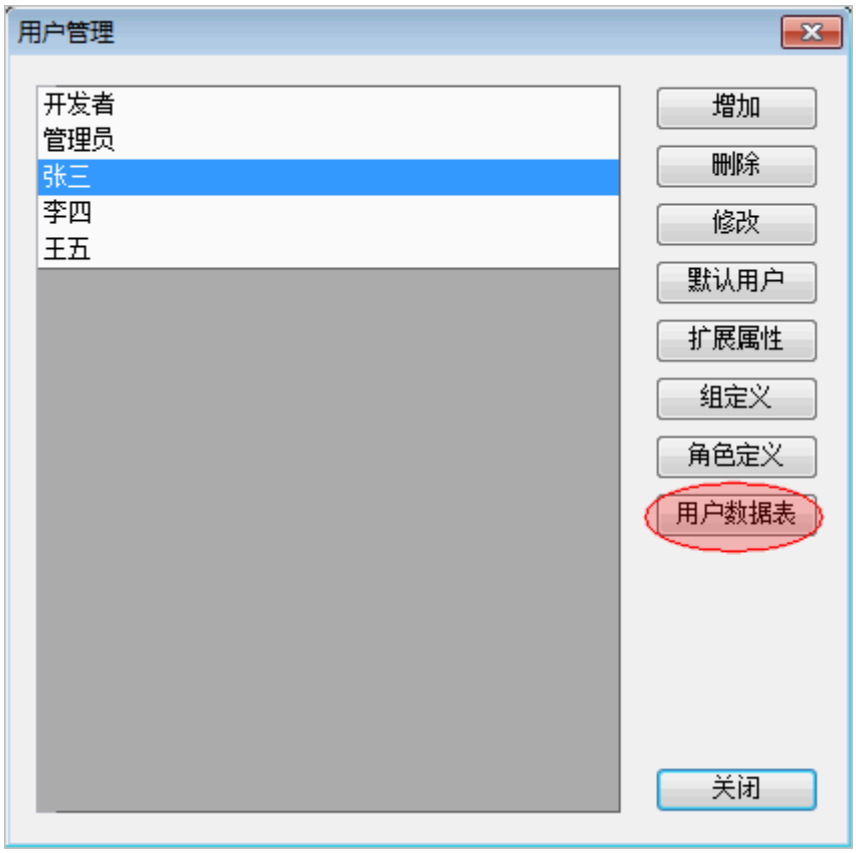
假定创建的这个表的名称为“Users”。

如果你直接在 SQL Server 中创建此表，注意要按下图所示设置列类型：

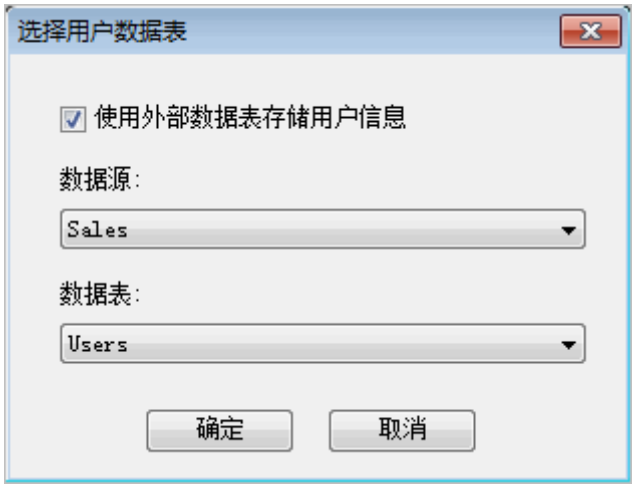


2、创建一个数据源，连接到这个数据库。

3、在用户管理中单击“用户数据表”按钮：



4、然后用于存储用户信息的外部数据源和外部数据表：



需要注意的是：

1、开发者和管理员的设置信息，始终保存在项目文件中，只有普通用户的设置信息，才会保存在外部表中，这样即使外部数据源无法连接，我们依然可以登录系统，重新调整设置。

2、项目文件中的用户设置信息，和外部数据表中的用户设置信息，是各自独立存在的，可以随时切换。

3、由于使用外部表存储用户信息时，是先连接数据源，再打开登录窗口，所以不能在BeforeConnectOuterDataSource 事件中使用扩展用户属性，否则会出错。

3.17.11 关于三层架构

Foxtable 默认是两层架构，客户端和数据库直连。
Foxtable 高级开发版提供了三层架构的功能，客户端可以不直连数据库，而是通过一个中间层和数据库交互。
Foxtable 三层架构建立非常简单，而且其使用和开发，和常规的两层架构没有任何区别，我们可以在一分钟内将一个两层结构的管理系统转换为三层架构，反之亦然。
所以对于有三层架构需求，但目前没有 Foxtable 高级开发版的用户，可以继续按照前述方式使用两层架构设计系统，之后条件许可时，可随时转换为三层架构，实际上就是已经发布好的系统，也可以随意在两层和三层架构之间切换。

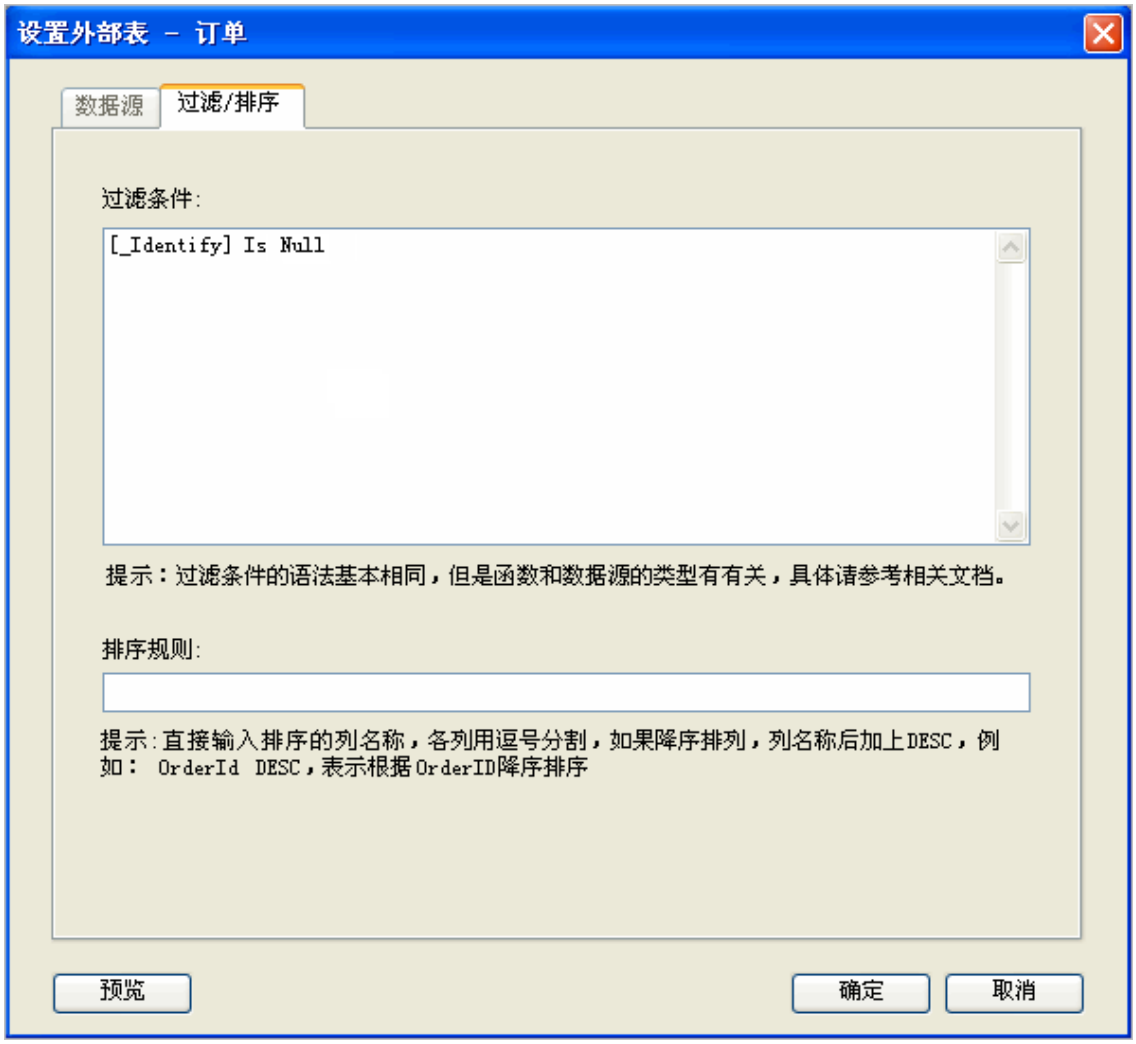
3.18 动态加载数据

3.18.1 控制初始加载量

如果数据量非常大，例如高达几十万行甚至数百万行，那么将其全部加载到 Foxtable 是不现实的。
Foxtable 提供了动态数据加载功能，可以根据需要加载部分数据，这样即使面对几千万行这样的海量数据，也能应付自如。
动态加载的第一步就是控制初始加载量，通常就是项目启动时默认不加载任何数据，而是在运行过程中动态加载。

外部表

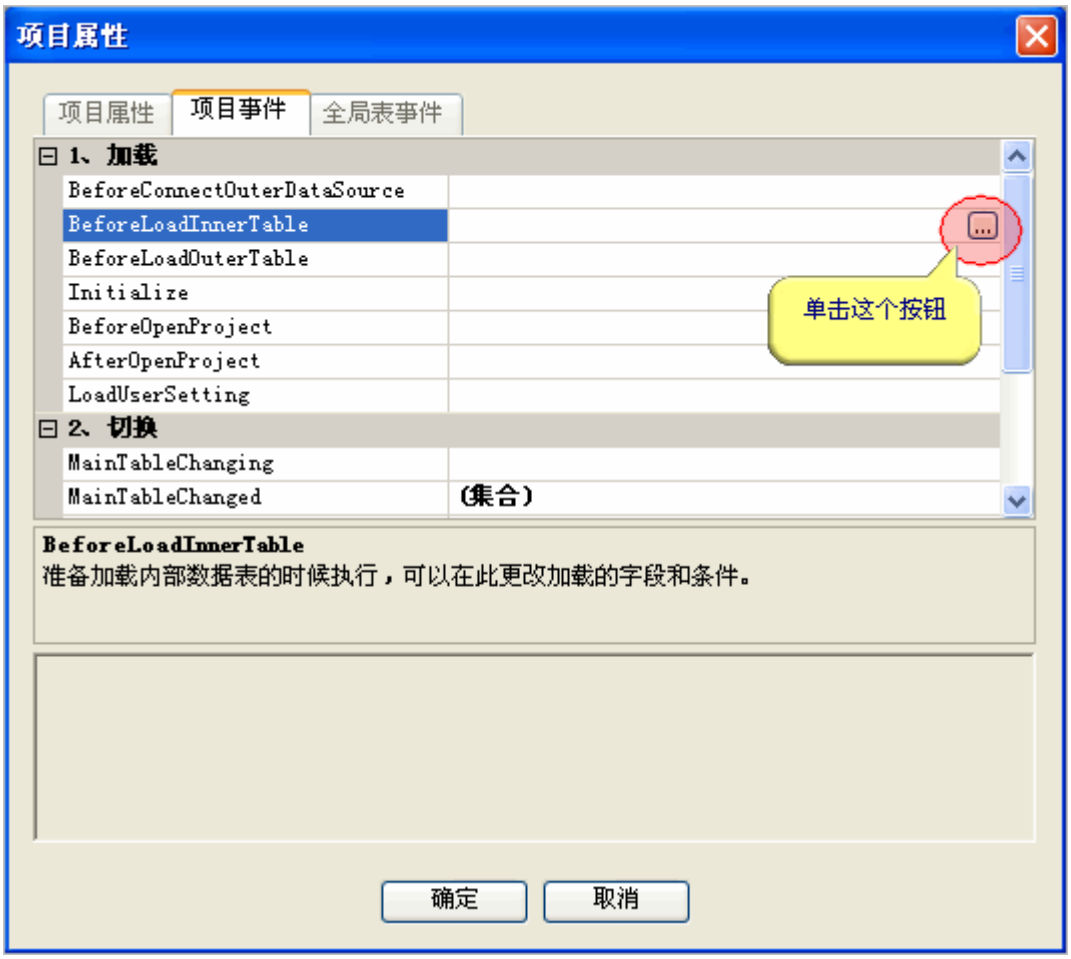
对于外部表，如果希望默认不加载任何数据，可以在设置外部表的时候，将加载条件设置为：
[主键列名称] Is NULL
例如假定主键列名称为_Identify，可以按下图所示设置：



内部表

对于内部表，如果希望初始不加载任何数据，处理起来会相对麻烦一些，需要使用代码，这是应用篇中唯一需要使用代码的地方。

假定希望订单,产品，客户三个表初始不加载任何数据，首先打开项目属性设置窗口，选择项目事件页，单击 BeforeLoadInnerTable 事件旁的按钮"...":



最后将代码设置为：

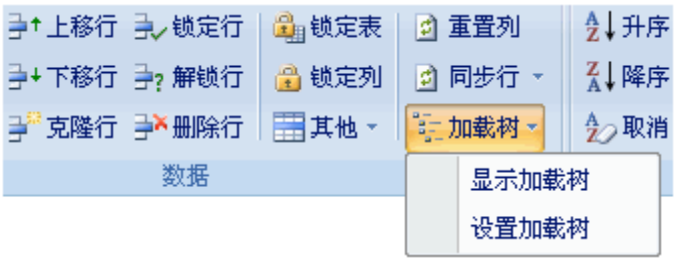
```
Select Case e.DataTableName
    Case "订单","产品","客户"
        e.Filter = "[_Identify] Is Null"
End Select
```

如果你还没有接触过开发篇，那么无需去理解上面的代码，这段代码是通用的，你要做的只是 将其复制到自己的项目中，然后添加和修改表名。

3.18.2 加载树

上一节介绍使得表初始不加载任何数据的方法，这一节介绍运行过程中动态加载数据的方法。

在菜单的“日常工作”区中，单击下面的按钮，可以打开加载树：



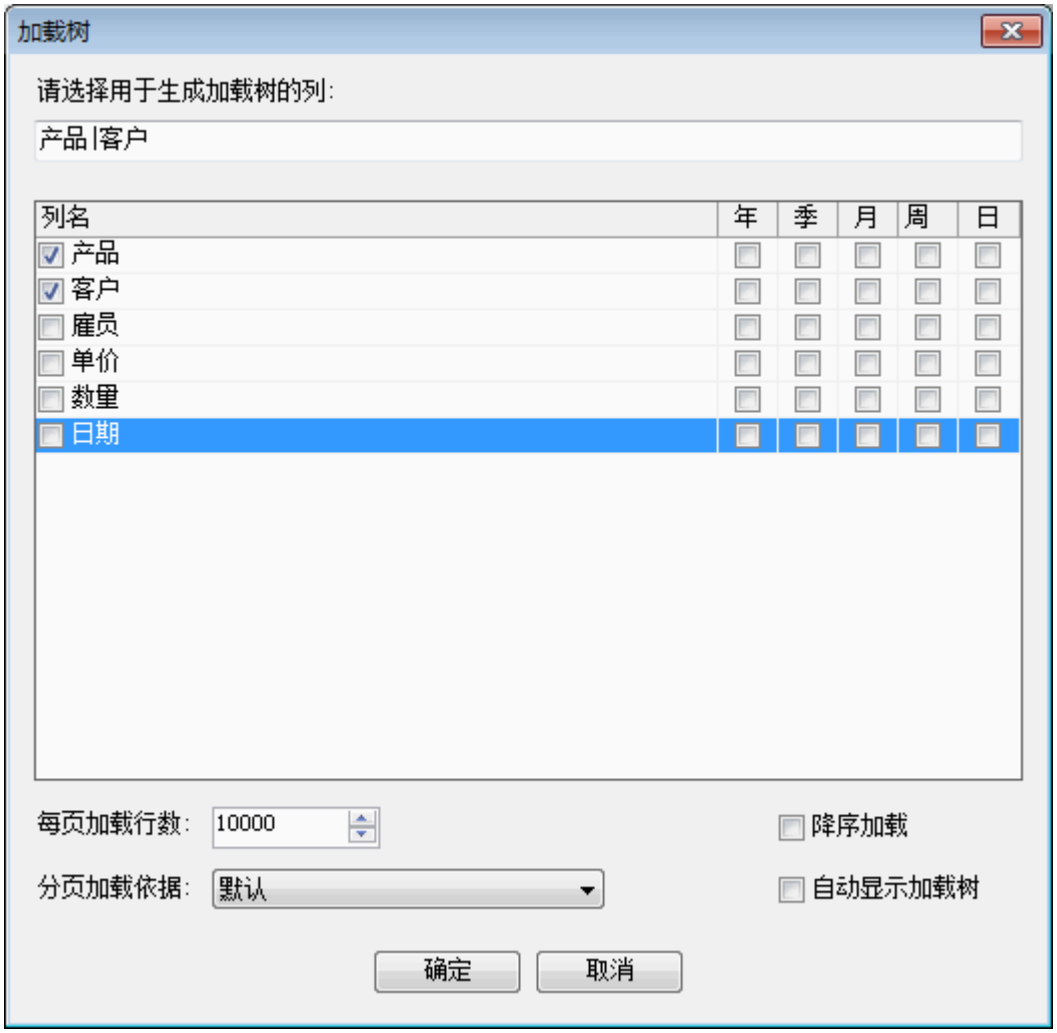
提示：

1、如果加载树已经关闭，希望按照上次设置重新打开加载树，那么执行“显示加载树”命令。

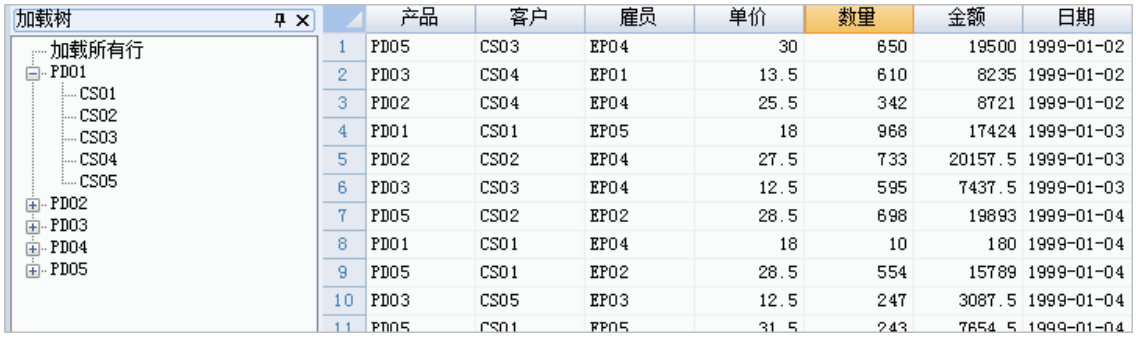
2、如果希望重新设置并打开加载树，那么执行“设置加载树”命令。

示例

例如对于订单表，按照下图所示设置加载树：



即可显示下图所示的加载树：

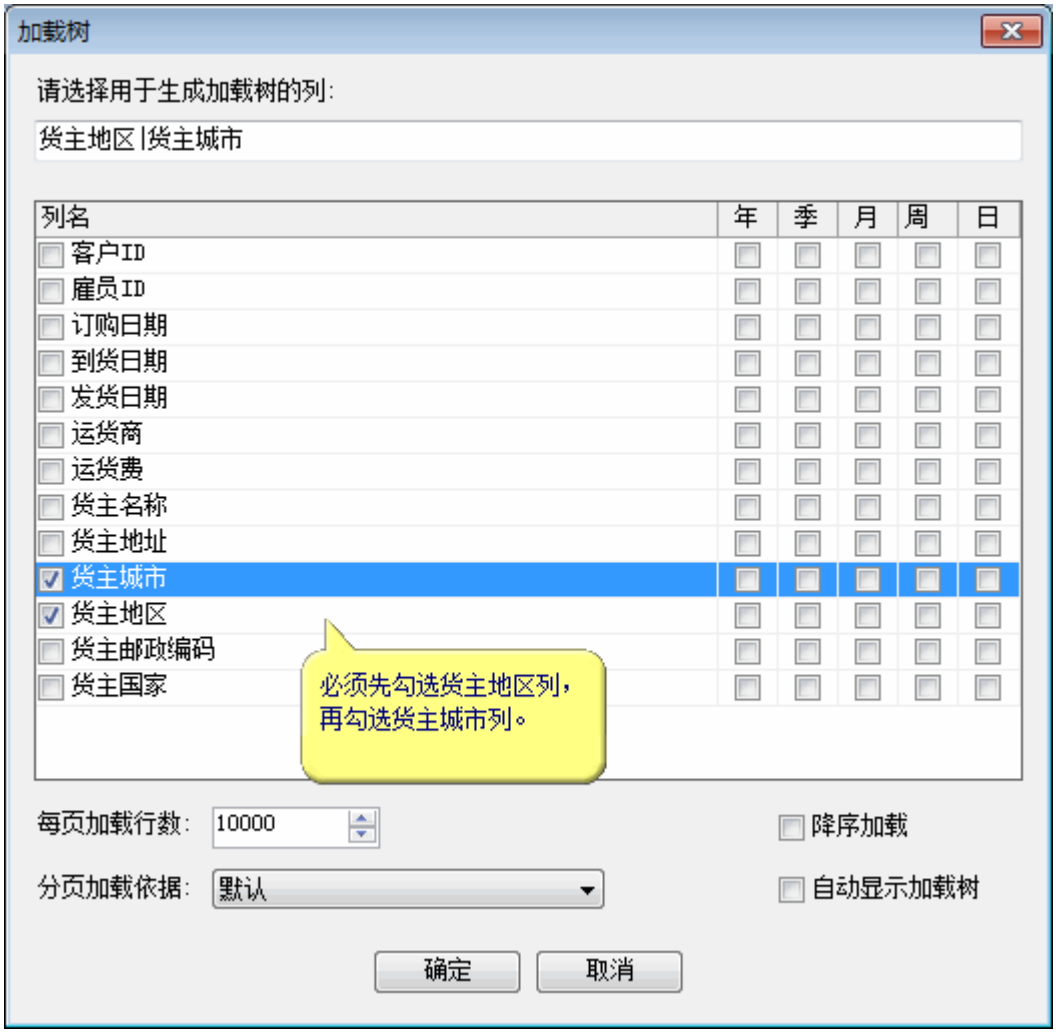


例如在上图中，要加载 CS01 客户订购产品 PD01 的订单，只需单击节点"PD01\CS01"即可。

注意勾选顺序

在设置加载树窗口的时候，必须注意勾选列的顺序，首先要勾选大的分组列，然后再勾选小的分组列。

例如在下面的窗口中，尽管货主城市列位于货主地区列之前，但地区是大的分组列，例如东北地区会包括很多个城市，所以必须首先勾选货主地区列，然后再勾选货主城市列：



这样才能得到我们需要的加载树：

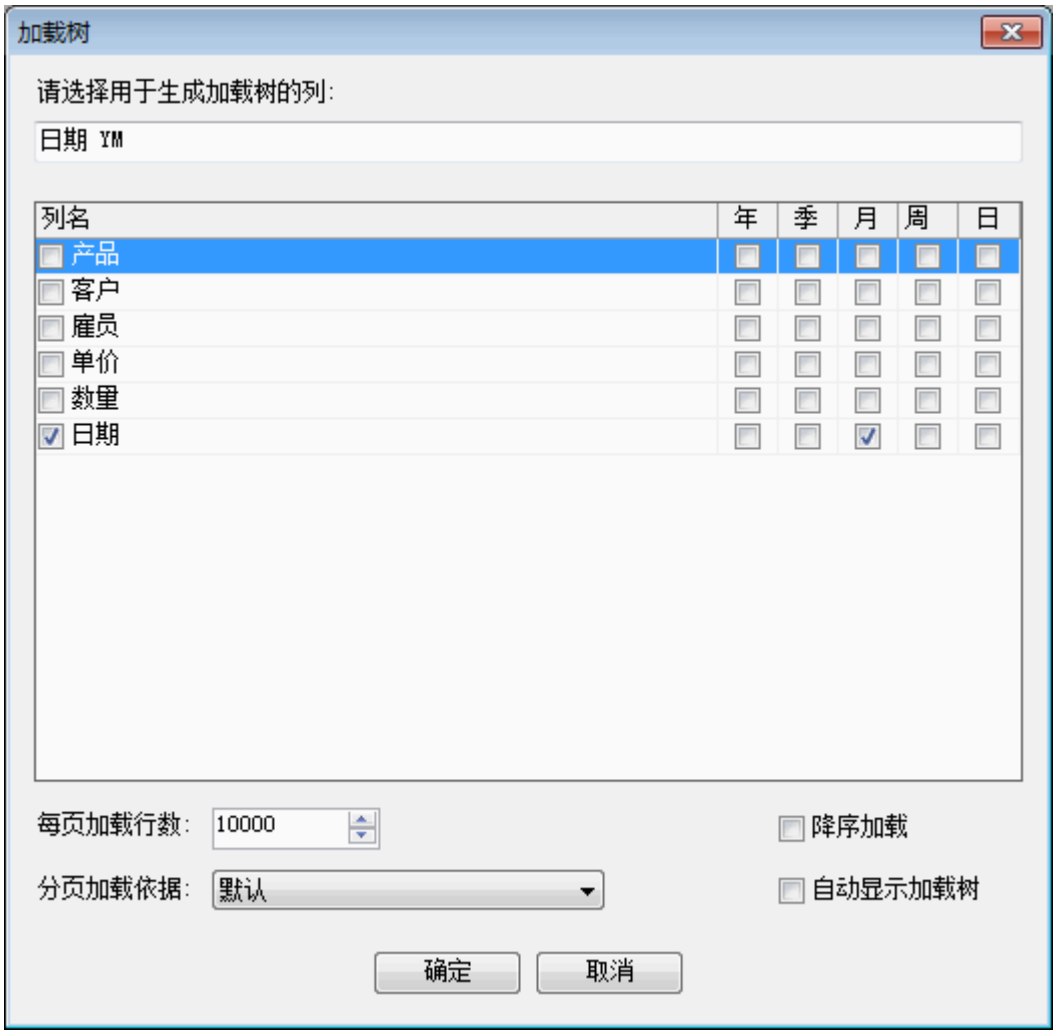


保存设置

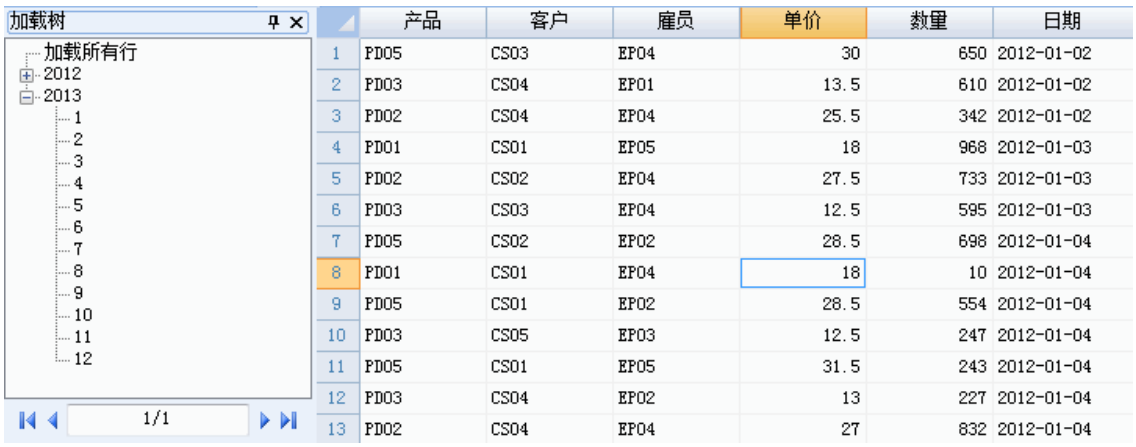
加载树的设置和宽度是可以随数据保存的，所以设置好加载树及其宽度后，单击保存按钮，下次打开项目后，只需执行菜单命令“显示加载树”，既可按照之前的设置和宽度打开加载树，无需重复设置。

3.18.3 分时段加载

在设置加载树的时候，对于日期型列，提供了年、季、月、周、日共五种分组方式。
例如对于订单表，按下图所示设置加载树：



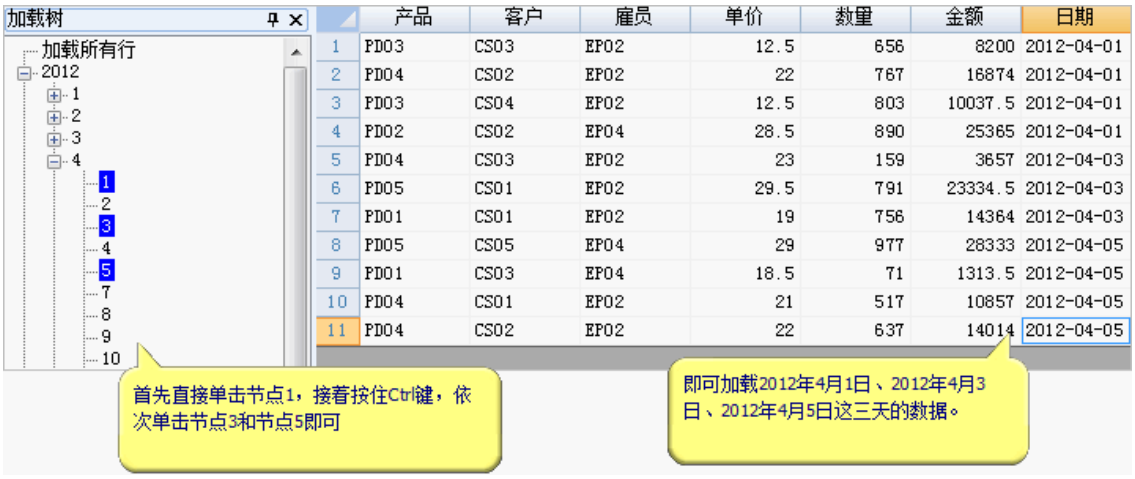
即可得到下图所示的加载树：



例如在上图中，要加载 2010 年 1 月份的订单，只需单击节点“2010\1”即可。

3.18.4 加载树可以多选

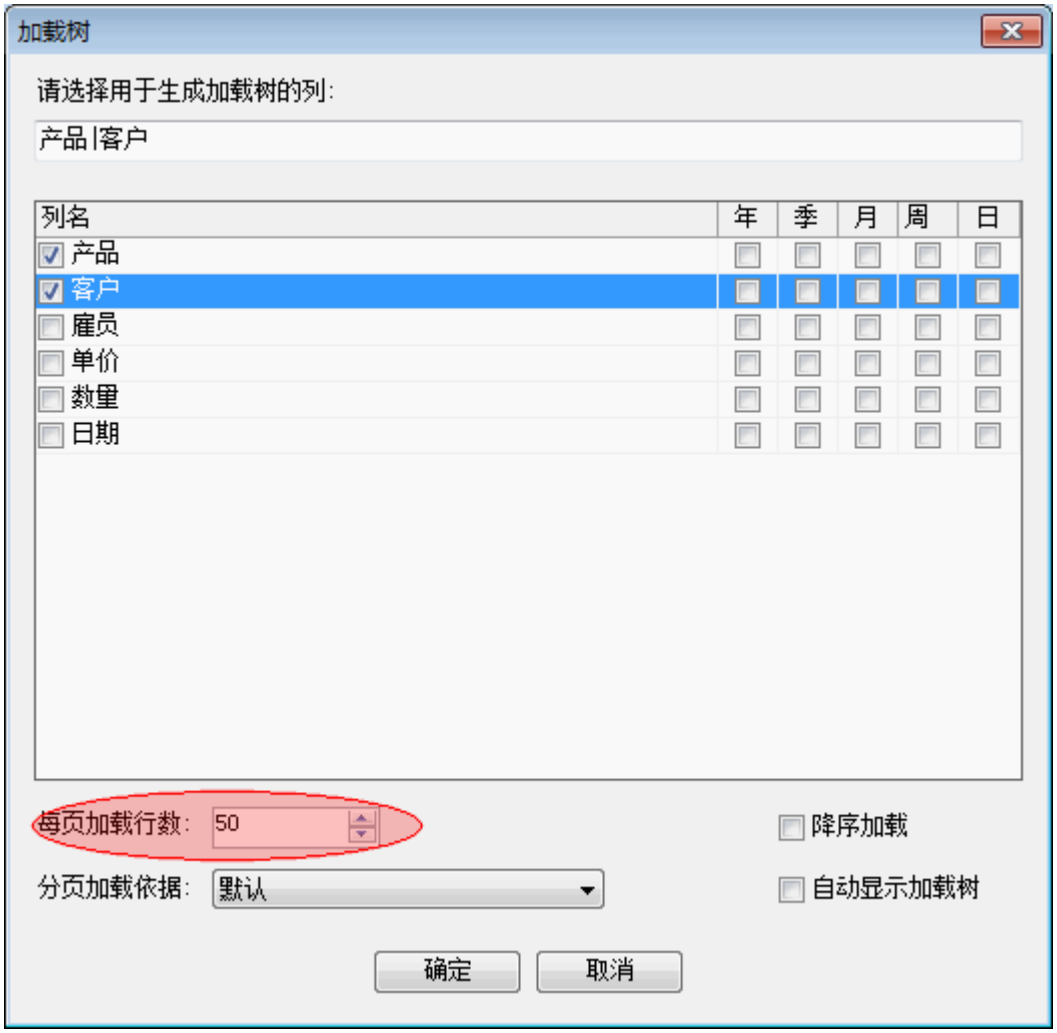
在默认情况下，单击加载树某个节点，可以加载此节点对应的数据。
我们还可以同时加载多个节点下的数据，只需按住 Ctrl 键，依次单击各节点即可。
例如下图所示的加载树，如果要加载 2012 年 4 月 1 日、2012 年 4 月 3 日、2012 年 4 月 5 日这三天的数据，可以先展开节点"2012\2"，显示这个月的所有日期，然后直接单击节点 1，接着按住 Ctrl 键，依次单击节点 3 和节点 5 即可：



	产品	客户	雇员	单价	数量	金额	日期
1	PD03	CS03	EP02	12.5	656	8200	2012-04-01
2	PD04	CS02	EP02	22	767	16874	2012-04-01
3	PD03	CS04	EP02	12.5	803	10037.5	2012-04-01
4	PD02	CS02	EP04	28.5	890	25365	2012-04-01
5	PD04	CS03	EP02	23	159	3657	2012-04-03
6	PD05	CS01	EP02	29.5	791	23334.5	2012-04-03
7	PD01	CS01	EP02	19	756	14364	2012-04-03
8	PD05	CS05	EP04	29	977	28333	2012-04-05
9	PD01	CS03	EP04	18.5	71	1313.5	2012-04-05
10	PD04	CS01	EP02	21	517	10857	2012-04-05
11	PD04	CS02	EP02	22	637	14014	2012-04-05

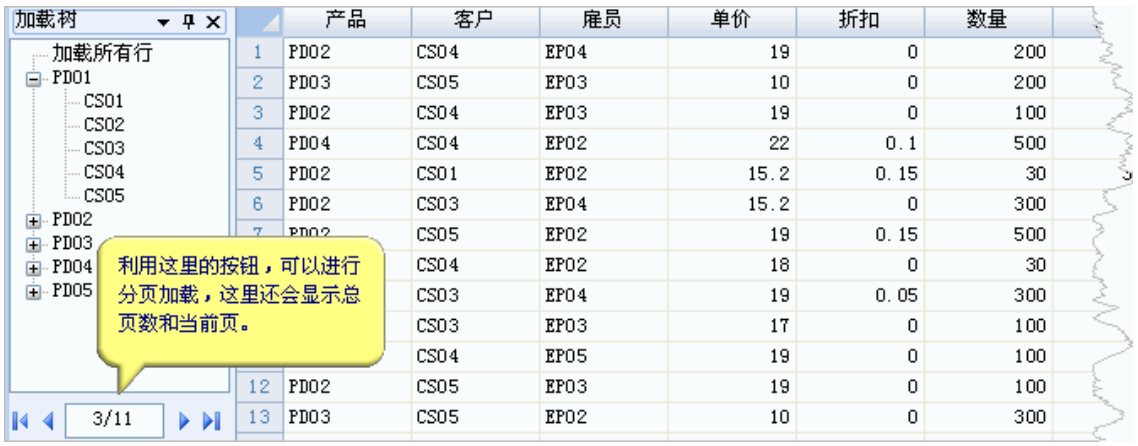
3.18.5 分页加载

在设置加载树的时候，你也许已经留意到，可以设置每页加载行数：



假如用户要加载的数据有几十万行甚至上百万行，此时用户面对的将是漫长的等待甚至崩溃，有了每页加载行数设置后，可以避免这个问题。

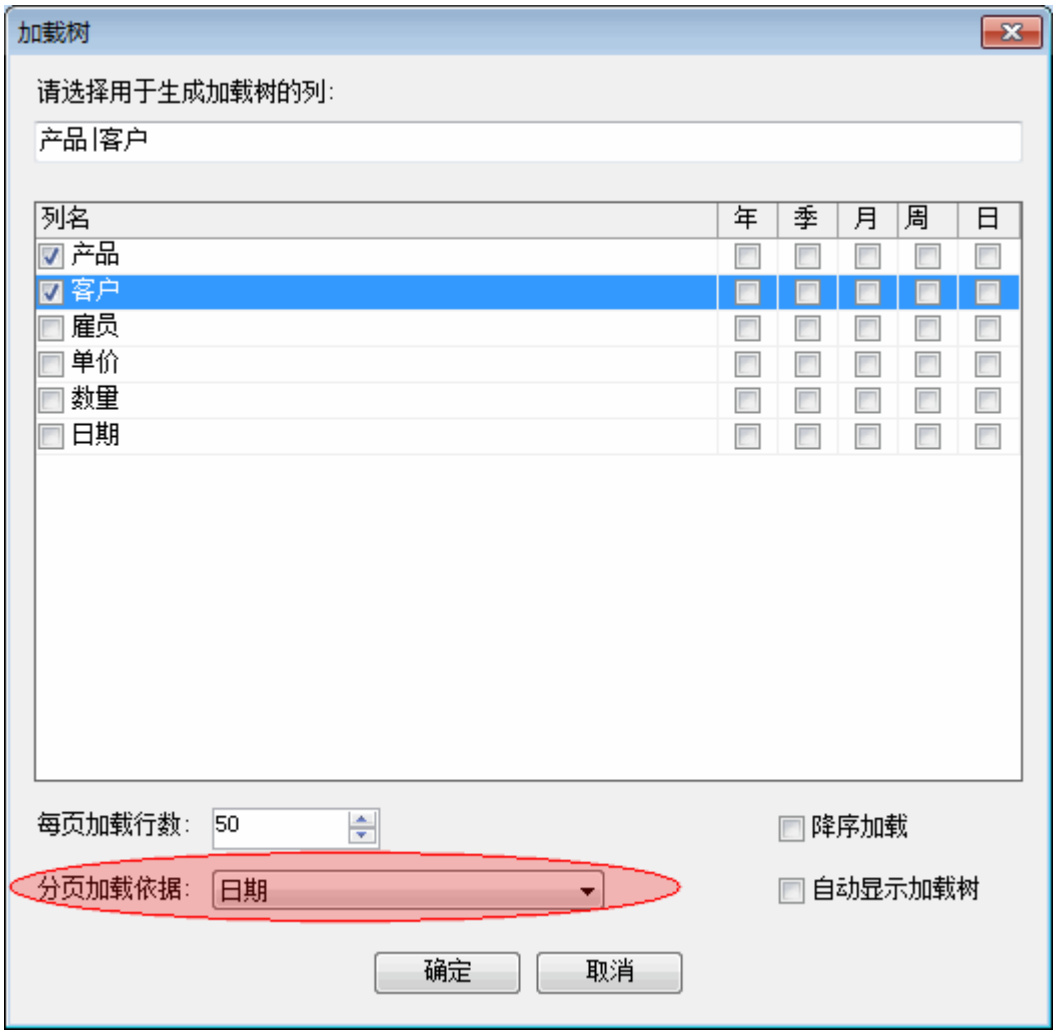
假定用户设置的每页加载行数为 50，那么不管要加载的行数有多少，每次最多只会加载 50 行，但是用户可以通过加载树下面的按钮，选择加载上一页、下一页、第一页、最末页：



提示： 双击当前节点，可以重新加载当前页的数据，相当于刷新。

3.18.6 选择分页列

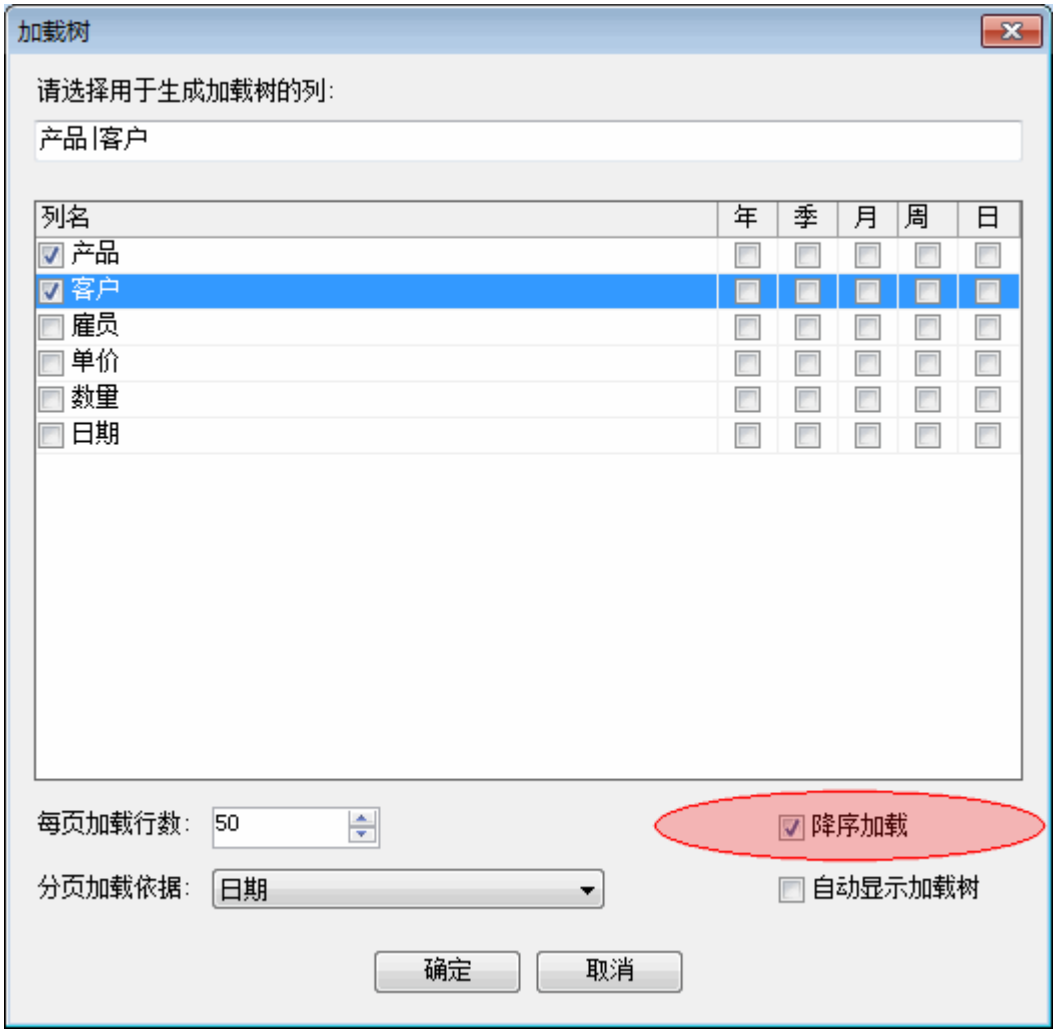
在默认情况下，Foxtable 会选择主键列作为分页依据列。
不过在设置加载树的时候，你可以改变分页依据列，例如下图选择日期列作为分页依据列：



我们的建议是：尽量采用主键列或者其他值不会重复的列（如身份证号码），作为分页依据列，因为如果分页依据列的值有重复，将会影响分页的准确性，例如上图根据日期分页，每页显示 50 行，如果某一天有 100 个订单，那么这 100 个订单会显示在同一个页面中，而不会按我们期望的那样分成两页显示。

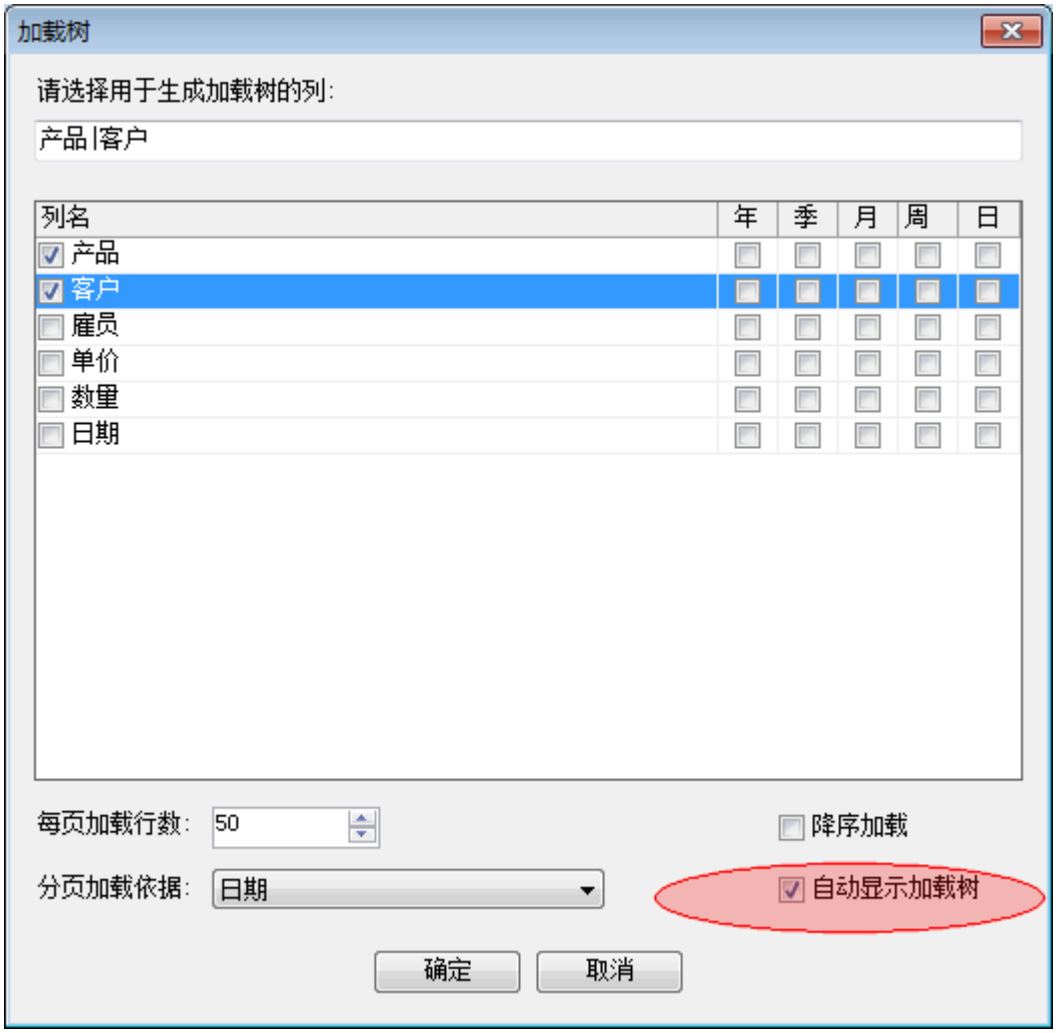
3.18.7 改变加载顺序

进行分页加载的时候，默认情况下，是根据分页列升序加载。
例如，假定订单表根据日期列分页加载，那么第一页加载的是最早的订单，最末页加载的是最近的订单，加载顺序是由旧到新。
如果希望改变默认的加载方式，首先加载最新的订单，可以在设置加载树的时候，选择选项“降序加载”：



3.18.8 自动显示加载树

如果希望打开项目后，某个表能自动显示加载树，可以在设置加载树的时候，选择选项“自动显示加载树”：

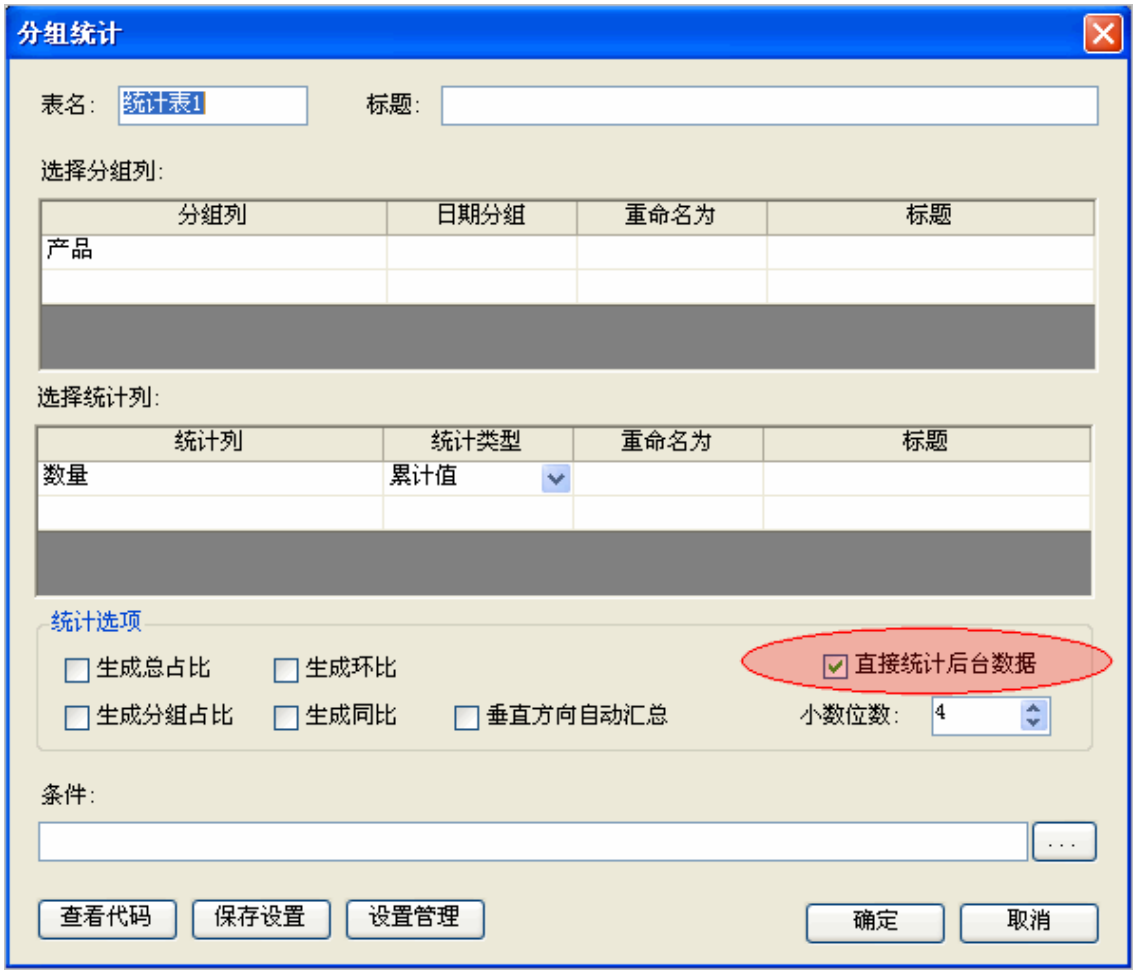


3.18.9 后台统计

如果数据量非常大，例如上百万甚至千万级别的记录，那么全部加载到内存是不可能的，所以 Foxtable 提供了动态加载数据的功能，使得在程序的运行过程中，也能随时根据需要加载符合条件的数据，这让 Foxtable 管理超大型数据库成为可能。

但是这有一个问题，动态加载部分数据是没问题的，但是很多时候，我们的统计是针对所有数据的，而 Foxtable 的分组统计和交叉统计默认都是针对已经加载的数据，如果我要统计所有数据，难道需要将一百万甚至一千万记录全部加载到 Foxtable？显然这是不可能的。

为此 Foxtable 的交叉统计和分组统计，都提供了“直接统计后台数据”的选项：



如果选择这个选项，统计的将不是已经加载的数据，而是后台所有的数据。

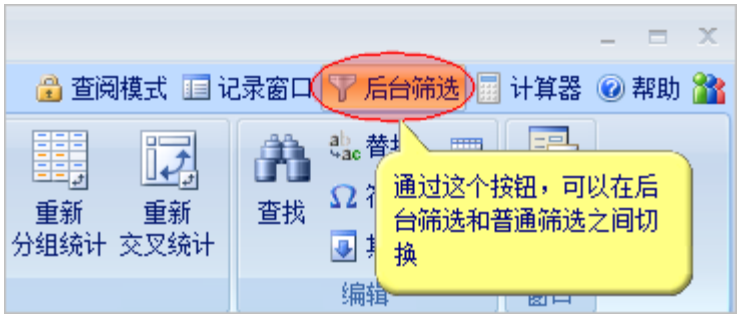
提示：

如果打开了加载树并开启了分页加载，那么后台统计并不会统计所有数据，而是统计选定节点下的所有数据，不管这些数据是否已经加载；如果要统计所有数据，请先选择节点"加载所有行"。

3.18.10 后台筛选

如果设置了分页加载数据，当我们通过菜单进行筛选的时候，默认只是对当前页的数据进行筛选，不会筛选出所有符合条件的行，显然这不够方便。

从 2014 版的 Foxtbale 开始，在分页加载的时候，我们依然可以对所有数据进行筛选，要做的只是按下配置栏中的“后台筛选”按钮，然后正常筛选数据就行：



提示：只有在分页加载的时候，才能按下这个按钮。

有了后台筛选，我们可以在只加载少量数据的时候，直接对所有数据进行筛选，感觉就像所有数据都已经加载了一样：



提示：

1、后台筛选默认是关闭的，需要通过按下配置栏中的“后台筛选”按钮来启动，如果希望某表默认开启后台筛选，可以在表属性设置中，将“分页后台筛选”属性设置为 True。

2、菜单中的所有筛选，都支持后台筛选，只有筛选树例外，因为如果用筛选树进行后台筛选，不如直接用加载树算了。

3、关联表不能进行后台筛选，只能进行普通筛选。

3.18.11 分页加载与数据统计

在显示加载树并开启分页加载功能后，通过菜单进行分组统计或交叉统计时，需要区分三种情况：

1、统计选定节点下的所有数据

如果满足以下三个条件：

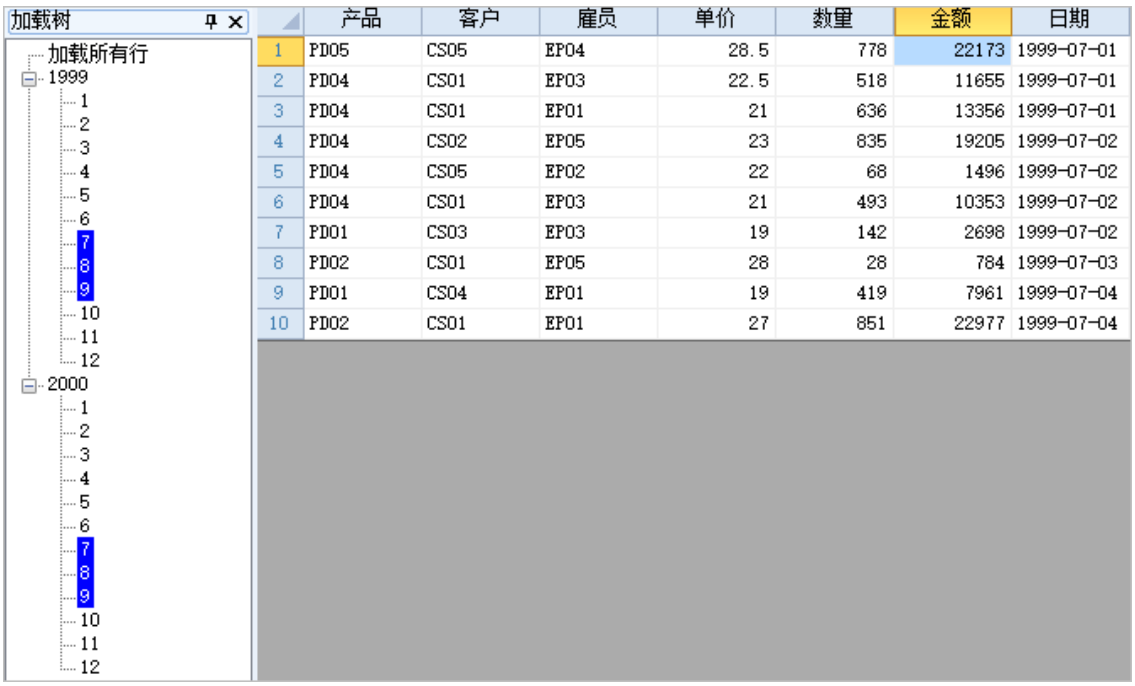
1、显示加载树

2、启用了分页加载功能

3、开启了后台筛选功能 ；或者虽然没有开启后台筛选，但统计的时候，勾选了"直接统计后台数据"。

那么通过菜单进行交叉统计或分组统计，统计的是所有选定节点（单个或多个）下的全部数据，不管这些数据是否已经加载。

例如我们需要统计比较相邻两年的 7、8、9 月份的销售数据，只需在加载树中选择这两年的 7、8、9 月份节点：



然后根据产品（水平分组）按年月（垂直分组）对数量进行交叉统计：



得到的统计结果为：

	产品	1999年			2000年		
		7月	8月	9月	7月	8月	9月
1	PD01	5204	8103	11014	6861	6830	9805
2	PD02	6807	9755	8657	12682	8774	8897
3	PD03	6088	8579	10107	12709	9137	5660
4	PD04	11864	13565	12863	3821	7926	6915
5	PD05	9332	11102	8595	7580	13786	6433

2、仅统计已经加载的数据

如果没有开启后台筛选功能，且没有勾选"直接统计后台数据"，那么通过菜单进行的分组统计或交叉统计，将仅统计已经加载的数据，也就是当前页面的数据。

3、统计后台所有数据

显然，如果要统计所有数据，必须先选择节点"加载所有行"，然后开启后台筛选功能，或者统计的时候勾选"直接统计后台数据"。

3.19 使用查询表

3.19.1 增加查询表

查询表的数据可以来自多个数据表，而且可以不使用主键。

查询表默认是锁定的，如果要修改查询结果，需要先解锁表，需要注意的是，对于查询表的任何修改都是不能保存的。

显然当某个表载入 Foxtable 的目的是查询，而不是编辑，那么使用查询表是个更好的选择。

1、在“数据表”功能区，单击下面的菜单按钮：



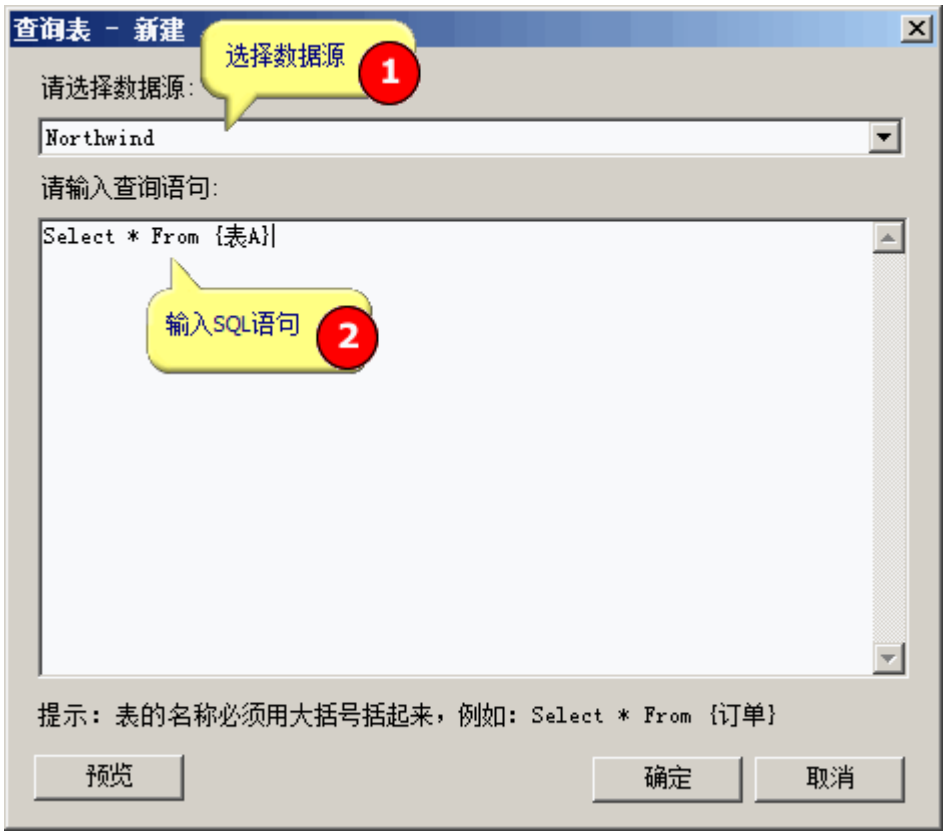
2、在查询表管理窗口，单击增加按钮，会出现下图所示的菜单：



你可以选择用 Foxtable 的查询表生成器来生成查询表，也可以选择直接输入 Select 语句来生成查询表。

3.19.2 Select 语句

如果在增加查询表的时候，选择“Select 语句”，会出现下图所示的窗口：



选择数据源，输入 Select 语句，单击确定按钮，输入查询表名称即可。

提示：例如你有 100 万甚至更多的行，将其全部载入 Foxtable 的唯一后果就是计算机崩溃，所以

数据量大的时候，记得设置 Select 语句的条件。
这种方法适合熟练掌握 SQL 语言的用户，对于大多数用户来说，使用查询表生成器更加合适一些。
接下来我们来介绍查询表生成器的使用方法。

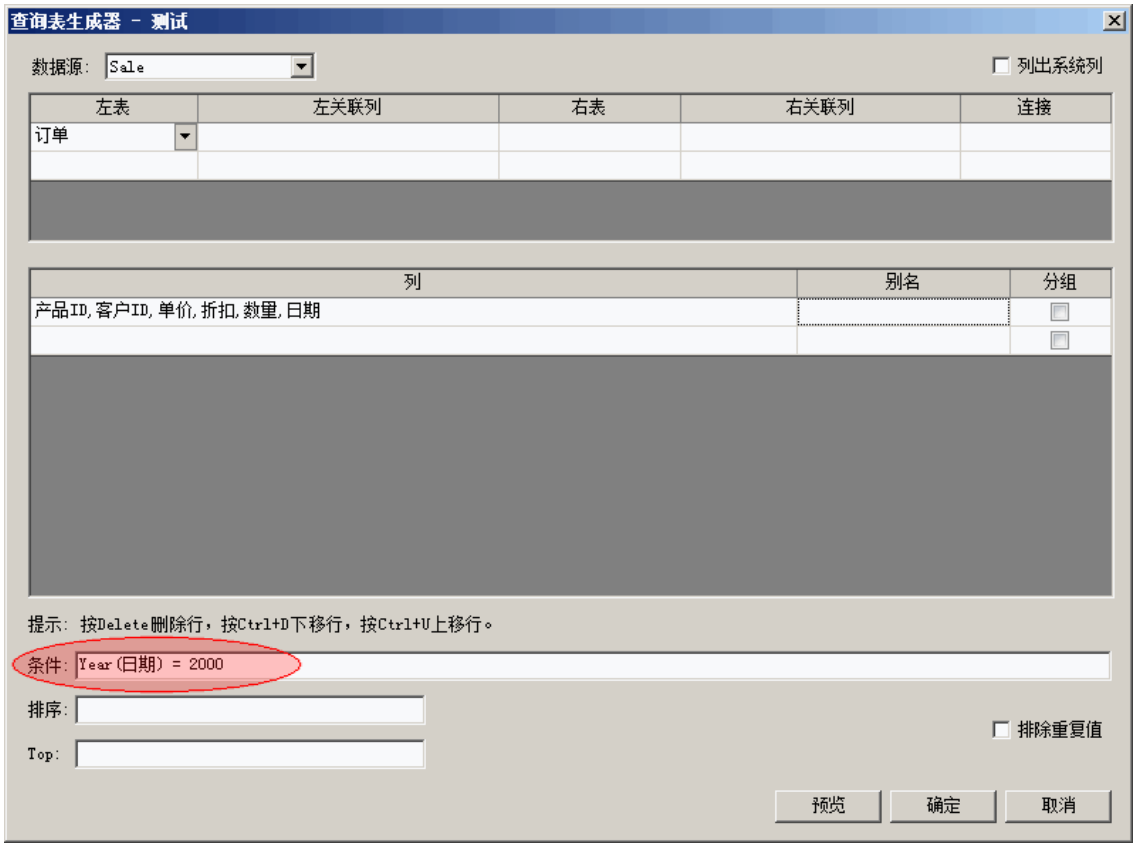
3.19.3 单表查询

如果在增加查询表的时候，选择的是查询表生成器，会打开查询表生成器窗口。
如果查询表的数据只来自一个表，可以按下图所示在查询表生成器进行设置：



提示：

例如你有 100 万甚至更多的行，将其全部载入 Foxtable 的唯一后果就是计算机崩溃。
所以不管是单表查询，还是多表查询，如果数据量很大，都应该设置查询条件：

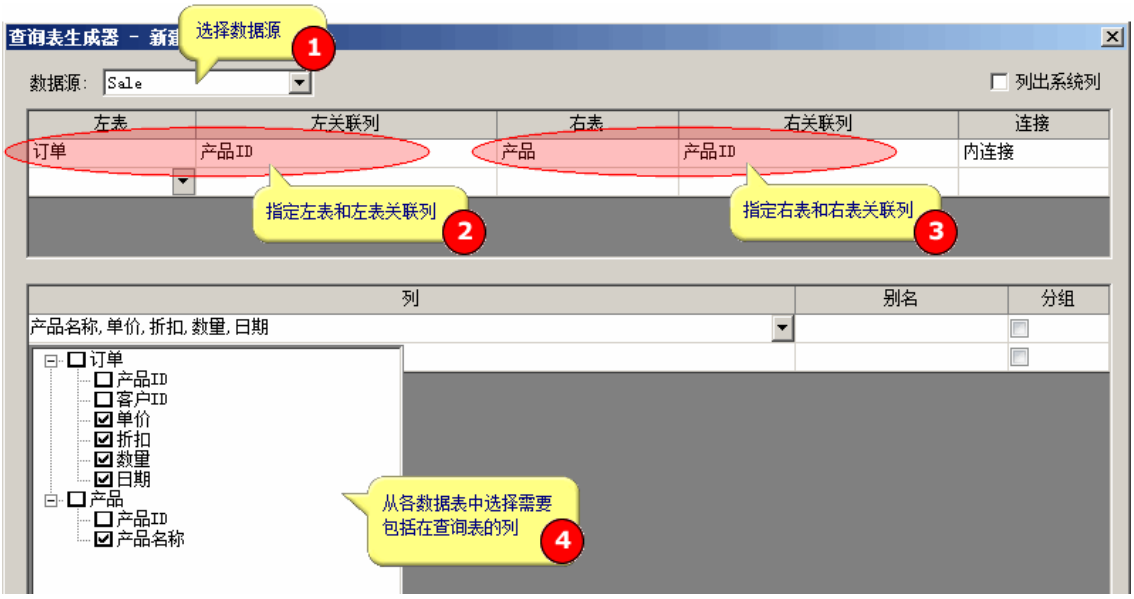


接下来我们看看，如果查询表的数据来自多个表，应该如何设置。

3.19.4 多表查询

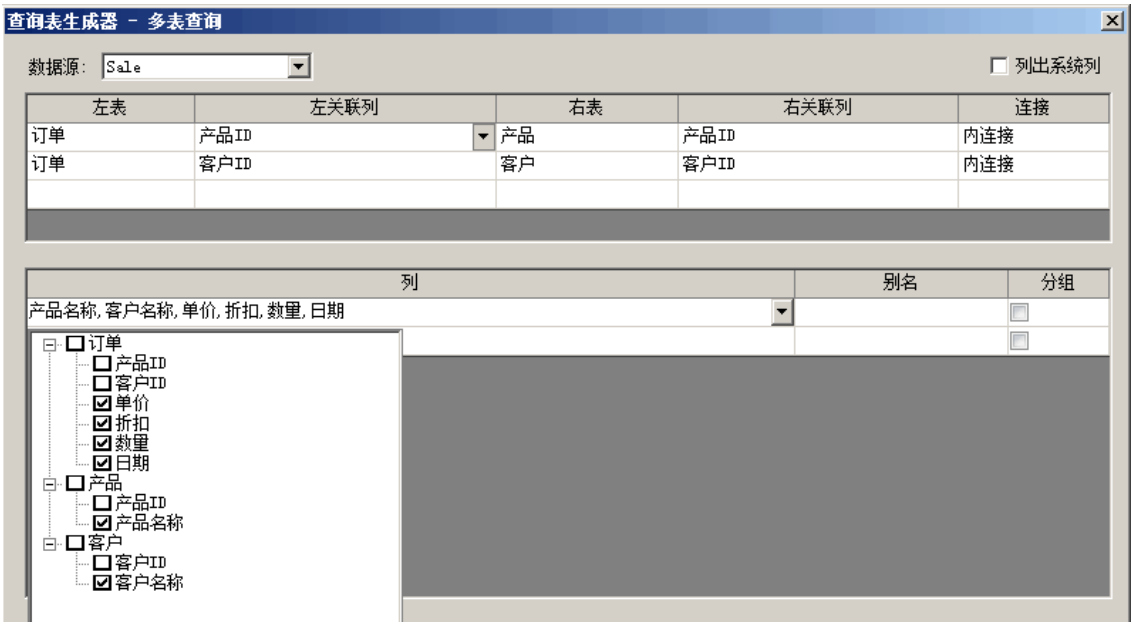
在学习本节内容前，可以打开 CaseStudy 目录下的文件：多表查询.Table

查询表的数据可以来自于多个表。
例如下图新建了一个查询表，该查询表的产品名称列来自产品表，单价、折扣、数量、日期等列来自订单表：



查询表的数据可以来自更多的表。

例如下图新建一个查询表，查询表的产品名称列来自产品表，客户名称列来自客户表，单价、折扣、数量、日期等列来自订单表：



提示：

参与查询的表通常存在一对多的关系，例如产品表和订单表，客户表和订单表，都是一对多的关系，因为一个产品或客户会有多个订单。

这种一对多的关系，通常称之为父子关系。

在查询表生成器添加数据表的时候，应该将多方(就是子表)作为左表，一方(就是父表)作为右表。

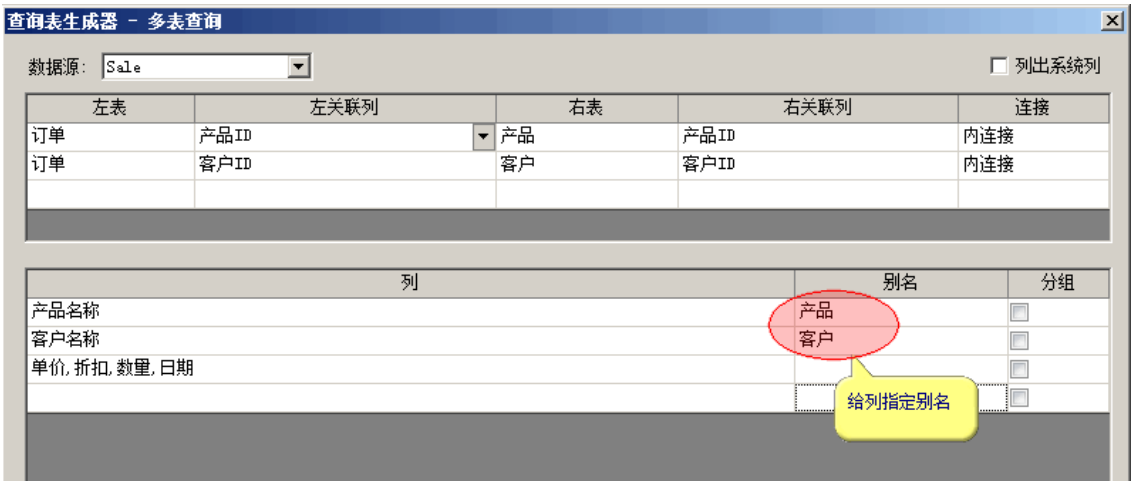
所以在上图中，订单表始终是左表，而产品表和客户表都是右表。

3.19.5 指定别名

在学习本节内容前，可以打开 CaseStudy 目录下的文件：多表查询.Table

在选择列的时候，可以给列指定别名。

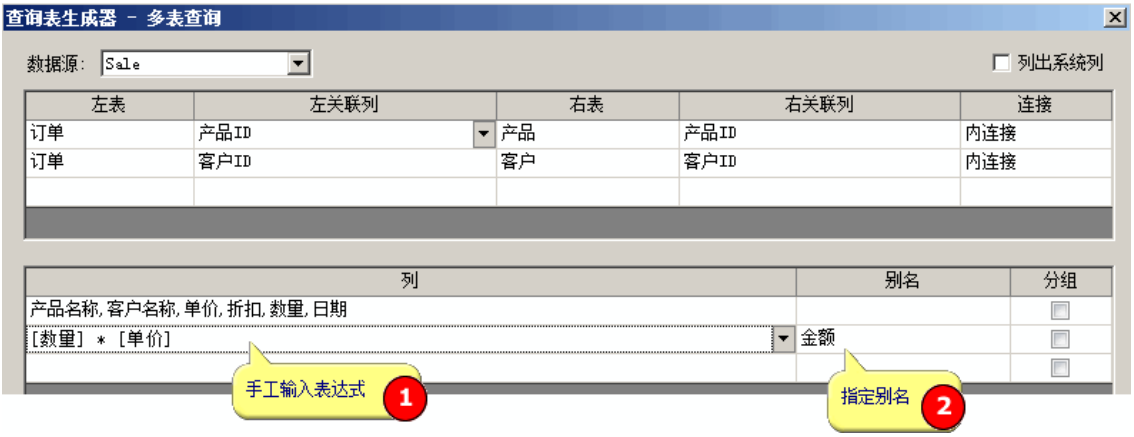
指定别名后，列在查询表中的名称不再是原来的名称，而是指定的别名。
例如下图产品名称列的别名为产品，客户名称列的别名为客户：



提示：如果要为某列指定别名，那么在选择列的时候，此列必须是单独的一行。

3.19.6 表达式列

在学习本节内容前，可以打开 CaseStudy 目录下的文件：多表查询.Table
设置查询表的时候，可以使用表达式列。
例如下图表示给查询表增加一个金额列，其计算公式为：[数量] * [单价]



同名列的处理

假定产品表和订单表都有单价列，那么表达式必须指定单价列的来源：
[数量] * {订单}.[单价]
{订单}.[单价] 表示这是订单表的单价，注意表名要用大括号括起来。

3.19.7 分组统计

在学习本节内容前，可以打开 CaseStudy 目录下的文件：多表查询.Table
查询表还可以进行分组统计。
Foxtable 本身已经提供了强大的数据统计和分析功能，一般并不需要通过查询表来进行分组统计，

当然你可以根据具体情况选择。

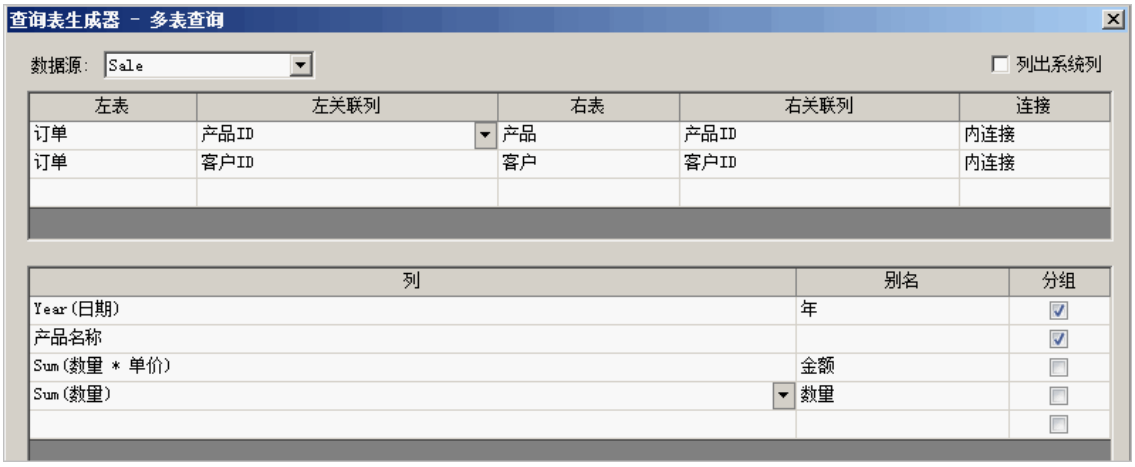
例如下图统计各客户订购不同产品的数量和金额：



得到的查询表为：

	产品名称	客户名称	数量	金额
1	浓缩咖啡	福星制衣厂股份	3776	83072
2	浓缩咖啡	浩天旅行社	4920	102220.8
3	浓缩咖啡	红阳事业	1241	23804
4	浓缩咖啡	立日股份有限公	5592	114136
5	浓缩咖啡	威航货运有限公	7345	158584.8
6	三合一麦片	福星制衣厂股份	3554	34040
7	三合一麦片	浩天旅行社	4301	43010

分组列也就可以是表达式，例如下图按年统计各产品的销量：



得到的查询表为：

	年	产品名称	金额	数量
1	1999	浓缩咖啡	250932	11907
2	1999	三合一麦片	132986	14448
3	1999	温馨奶酪	659253	37148
4	1999	盐水鸭	394915	20042
5	1999	运动饮料	364172.4	21791
6	2000	浓缩咖啡	230885.6	10967
7	2000	三合一麦片	129486	14207
8	2000	温馨奶酪	653192.2	37017

聚合函数

在上面的设置中，我们使用了聚合函数 Sum，用于对指定的列或表达式求和，例如：

Sum(数量)

表示对数量列求和。

聚合函数有：

SUM

返回指定表达式的总计值。

AVG

返回指定表达式的平均值。

MAX

返回指定表达式的最大值。

MIN

返回指定表达式的最小值。

COUNT

返回符合指定条件的记录数

Var

返回指定表达式的总体方差

VarP

返回指定表达式的总体方差

StDev

返回指定表达式的总体标准偏差

StDevP

返回指定表达式的总体样本标准偏差

提示：不同的数据源会有自己特有的一些聚合函数。

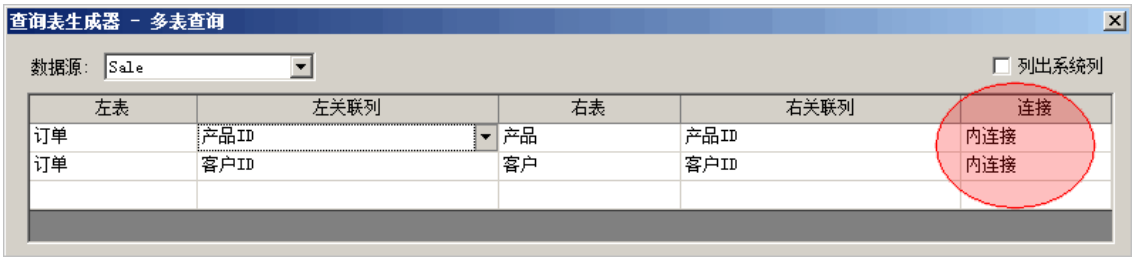
3.19.8 刷新结果

如果要刷新查询表，右击查询表任意一个单元格，在弹出的快捷菜单中，单击“同步表”即可：



3.19.9 连接类型

你也许已经留意到，在添加表的时候，可以指定连接类型：



有三种连接类型：

- 内连接
  左表和右表都具备的数据才会出现在查询表中。
  在上图中，如果某个订单在产品表中没有对应的产品，那么这个订单不会出现在查询表中，同样如果某个产品在订单表中没有对应的订单，那么这个产品也不会出现在查询表中。

- 左连接
  左表的所有数据出现在查询表中。
  在上图中，由于订单表是左表，所以所有订单都会出现在查询表中，不管这个订单在产品表中是否有对应的行。

- 右连接
  右表的所有数据出现在查询表中。
  在上图中，由于产品表是右表，所以所有产品都会出现在查询表中，不管这个产品是否有订单。

提示：

1. 内部数据源和 Access 数据源对于连接类型的支持不如 SQl Server 全面，例如上图中如果全部选择为右连接，预览的时候会提示“不支持的连接表达式”。

2. 参与查询的表通常存在一对多的关系，例如产品表和订单表，客户表和订单表，都是一对多的关系，因为一个产品或客户会有多个订单。
   这种一对多的关系，通常称之为父子关系。
   在查询表生成器添加数据表的时候，应该将多方(就是子表)作为左表，一方(就是父表)作为右表。
   所以在上图中，订单表始终是左表，而产品表和客户表都是右表。

3.19.10 选择系统列

在默认情况下，用户通过目录树选择列的时候，并不能选择“_Identify”和“_Locked”这样的系统列。
如果希望查询表中包括这样的列，可以在查询表设置窗口，选择选项“列出系统列”：



3.19.11 选择所有列

如果查询表要包括某个数据表的所有列，那么无需逐列勾选，直接勾选此表即可。
例如下图新建的查询表，将包括订单表的所有列和产品表的产品名称列：



3.19.12 限制加载行数

设置加载行数

在设置查询表的时候，可以限制加载行数。

例如要加载 50 行数据，可以将行数设置为：50

行数可以是百分比，例如要加载 30%的行数，可以将行数设置为：30 Percent

设置加载行数的时候，通常也会设置排序参数，否则没有太多实际意义。

排序方式由列名组成，默认按升序排序，如果要按降序排序，请在列名后加上“DESC”。

例如要按日期排序，旧的日期排在前面，排序设置为：日期

例如要按数量排序，数量小的排在前面，排序设置为：数量

例如要按日期排序，新的日期排在前面，排序设置为：日期 DESC

例如要按数量排序，数量大的排在前面，排序设置为：数量 DESC

示例

例如要查询最近 50 个订单：



如果要查询最早 50 个订单，只需将排序改为“日期”即可，也就是去掉“DESC”。

同样：

如果要查询订购数量最多的前 50 个订单，只需将排序参数改为：数量 DESC

如果要查询订购数量最少的前 50 个订单，只需将排序参数改为：数量

提示：

排序参数只在查询的时候发生作用，并不会对生成的查询表的数据进行排序。

不加载数据

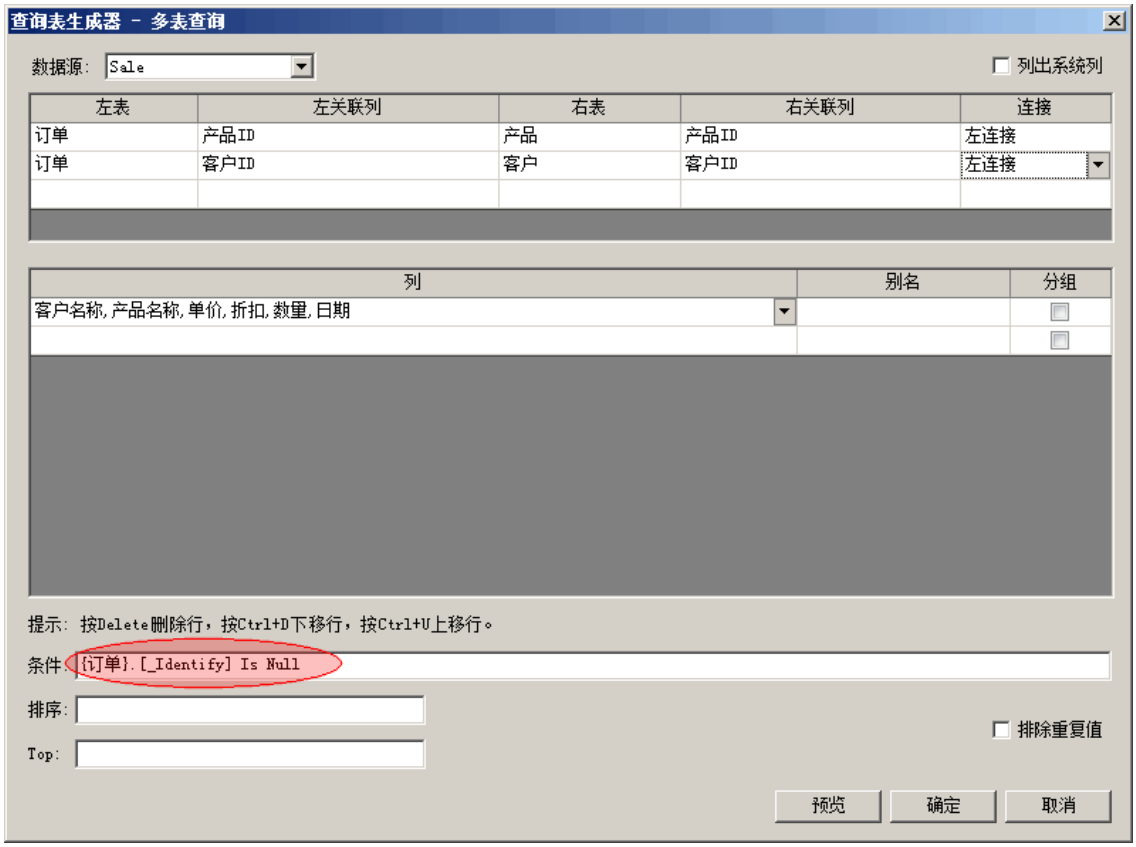
另外如果希望查询表初始不加载任何数据，可以将加载条件设置为：

[_Identify] Is Null

如果查询表的内容来自于多个表，必须将加载条件设置为：

{表名}.[_Identify] Is Null

例如：



提示：

如果使用的是外部数据源，请将[_Identify]改为实际的主键列名。

3.19.13 查询表与加载树

查询表也可以使用加载树，而且也可以分页加载，不过要注意：

1、用于生成加载树的列不可以是表达式列。

2、如果查询表的数据来自于多个表，那么在设置查询表的时候，建议请将子表作为左表，父表作为右表。

3、选择查询表所包括的列时，最好包含子表的主键列，因为默认使用此列作为分页列。

4、如果查询表没有包括子表的主键列，或者子表的主键列名称不是"_Identify"，那么在设置加载树时，必须明确指定分页列。

如果不想使用查询表生成器，而是直接编写 Select 语句来生成查询表，也请注意 2、3 点。

既然使用加载树，通常就希望查询表默认不加载全部数据，如果你用 Select 语句直接定义查询表，可以用 Top 参数指定默认加载的行数，例如：

Select Top 100 * From {表名} Order By 日期 Desc

表示按日期顺序默认加载最近的 100 行。

如果使用的是可视化的查询表设计器，可以直接指定加载行数，参考：限制加载行数

使用后台数据库视图

如果查询表的 Select 语句比较复杂，有时即使满足上面的 4 个条件，可能也无法正常使用加载树。我们可以将复杂的 Select 语句定义为后台数据库的一个视图(Access 称之为查询)，这样我们就可以将这个 Select 语句当做一个普通的表使用，以常规的方式来显示加载树。

以 Access 数据源为例，假定有一段如下所示用于销售分析的 Select 语句：

```
SELECT 订单.[订单 ID], 订单.订单日期, 员工扩展信息.员工姓名 AS 员工, 客户扩展信息.公司 AS 客户名称, 产品.产品名称, 扩展订单明细.总价 AS 销售, 客户扩展信息.[省/市/自治区], 客户扩展信息.[国家/地区], 员工扩展信息.ID AS [员工 ID], 产品.ID AS [产品 ID], 客户扩展信息.ID AS [客户 ID], Year([订单日期]) AS 年份, Month([订单日期]) AS 月份, DatePart("q",[订单日期]) AS 季度, Format([订单日期],"mmmm") AS 月份名, 产品.类别, 产品.产品名称 AS 产品, 客户扩展信息.公司 AS 客户, (Month([订单日期]) Mod 3) AS 季度月
FROM ((((订单 INNER JOIN 客户扩展信息 ON 订单.[客户 ID] = 客户扩展信息.ID) LEFT JOIN 员工扩展信息 ON 订单.[员工 ID] = 员工扩展信息.ID) INNER JOIN 扩展订单明细 ON 订单.[订单 ID] = 扩展订单明细.[订单 ID]) INNER JOIN 产品 ON 扩展订单明细.[产品 ID] = 产品.ID) INNER JOIN 发票 ON 订单.[订单 ID] = 发票.[订单 ID];
```

显然这段语句过于复杂了，如果在 Foxtable 中用这段语句定义一个查询表，将没有办法正常使用加载树功能。

我们可以这样解决：

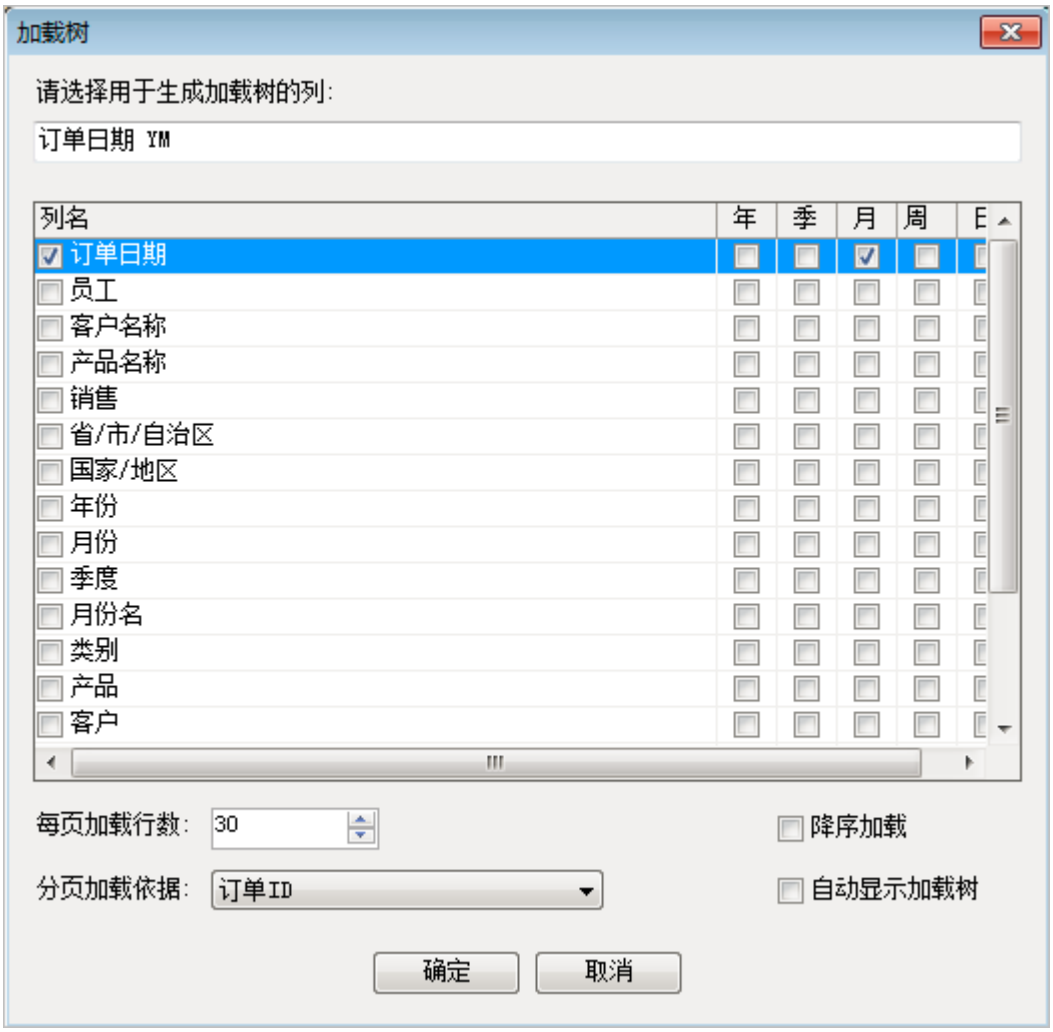
1、用 Access 打开这个数据库文件，然后定义一个查询，查询的名称为"销售分析"，查询的 SQL 语句就设置为上面这段 Select 语句。

2、在 Foxtable 增加一个查询表，用 Select 语句定义：

```
Select Top 30 * From {销售分析}
```

提示：定义的查询表默认加载 30 行数据。

3、现在选择这个查询表，就可以按下图所示正常设置加载树：



提示：由于本例的查询表没有_Identify 列，所以设置加载树的时候，要选择分页依据列，否则无法正常分页加载，这里选择的是“订单 ID”作为分页依据。

总结：

通过在后台数据库定义查询(SQL Server称之为视图)，将复杂的Select查询语句当做一个普通的表来使用，让Foxtable只处理自己所擅长的部分，可以大大提高动态加载的兼容性。

3.19.14 查询表的替代者

一、用表达式列显示多个表的数据

Foxtable 有表达式列，通过表达式，可以实现列与列之间的计算，还可以引用父表数据，或统计子表数据。

假定有产品、客户、订单三个表，结构如下：

	产品编号	产品名称	单价
1	1	HCD238(3)P/TSD	98
2	2	HCD238(32)P/TSI	65
3	3	HW238(2)P/TD	200
4	4	HA238(8)P/T	25

产品表结构

	客户编号	客户名称	联系人	电话
1	1	联想电脑公司	周瑜	010-88667722
2	2	TCL商贸公司	关羽	020-22669900
3	3	神州数码公司	刘备	021-33667733
4	4	盛大网络公司	张飞	023-55443123

客户表结构

	日期	产品编号	客户编号	数量	单价
1	2008-06-06	2	1	6	65
2	2008-06-06	1	2	3	98
3	2008-06-07	3	3	7	200
4	2008-06-08	4	4	2	25
5	2008-06-08	3	4	9	200

订单表结构

其中产品表和订单表之间通过产品编号建立关联，关联名为"po"，客户表和订单表之间通过客户编号建立关联，关联名为"co"。

我们可以在订单表增加三个表达式列，分别为：

名称	类型	表达式
金额	双进度小数	[数量] * [单价]
产品名称	字符	Parent(po).产品名称
客户名称	字符	Parent(co).客户名称

现在订单表的结构如下图所示，不是查询表，却胜似查询表：

	日期	产品编号	产品名称	客户编号	客户名称	数量	单价	金额
1	2008-06-06	2	HCD238(32)P/TSD	1	联想电脑公司	6	65	390
2	2008-06-06	1	HCD238(3)P/TSD	2	TCL商贸公司	3	98	294
3	2008-06-07	3	HW238(2)P/TD	3	神州数码公司	7	200	1400
4	2008-06-08	4	HA238(8)P/T	4	盛大网络公司	2	25	50
5	2008-06-08	3	HW238(2)P/TD	4	盛大网络公司	9	200	1800

二、统计查询表的实现

不管是汇总模式、还是分组统计或交叉统计，都提供了设置管理和调用功能，类似于 Access 的统计查询 ，但是功能却强大得多。

以交叉统计为例，你可以在交叉统计设置窗口，单击下图所示的两个按钮，将设置好的交叉统计保存取来：



- 保存设置
  在交叉统计设置窗口，先设置好交叉统计，然后单击“保存设置”按钮，输入设置名称，单击“确定”按钮即可。

- 修改设置
  在交叉统计设置窗口，先设置好交叉统计，然后单击“保存设置”按钮，选择之前保存的设置名称，单击“确定”按钮即可覆盖原来的设置。

- 设置管理
  在交叉统计设置窗口，单击“设置管理”按钮，可以删除、重命名已经保存的设置。

- 调用设置
  在菜单的“日常工作”区，单击“数据统计”按钮，选择“交叉统计”，然后指向“启用历史统计”，会列出所有保存好的设置名称，单击其中一个，即可自动根据该设置生成交叉统计表。



- 重复上次统计
  在上图中，单击“重复上次统计”命令，即可根据最近一次的 交叉统计设置，重新生成统计表。

三、众多专业统计工具和查询表生成器

在学习本帮助的开发指南部分时，你将发现 Foxtable 的查询和统计是分开的，Foxtable 有四大统计工具，分别是 GroupTableBuilder、CrossTableBuilder、SQLGroupTableBuilder、SQLCrossTableBuilder，其数据统计和分析能力，远远强于 Access。

此外 Foxtable 还提供了查询表生成器：SQLJoinTableBuilder。

这些工具完全采用傻瓜式的编程，不需要任何 SQL 语言知识，任何人都可以快速掌握，能随心所欲地进行数据统计或生成各种查询表。

这些工具都可以在运行过程中动态生成和绑定，在你设计各种专业管理软件时，这将是一个莫大的优势。

3.19.15 用数据表还是查询表?

外部表是以数据表的形式添加到 Foxtable，还是以查询表的形式添加，是需要慎重考虑的。

一般来说，如果某个外部表的数据禁止修改，加载到 Foxtable 只是为了查询和统计分析，那么应该以查询表的形式加入；如果某个外部表加载到 Foxtable 后，允许用户编辑，并且编辑结果需要回存到数据源，那么就应该以数据表的形式加入。

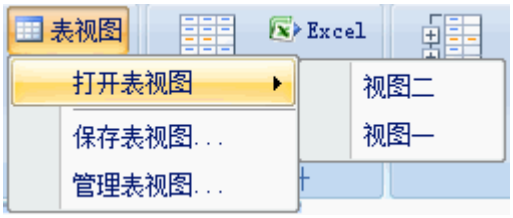
3.20 表视图

表视图能够将一些设置保存下来，供以后快速调用恢复。

能够保存到表视图中的信息有：

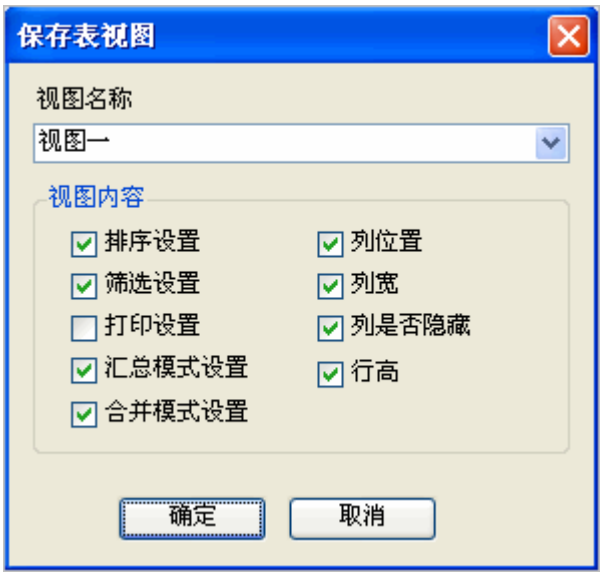
- 排序
- 筛选
- 列位置
- 列宽
- 行高
- 列是否隐藏
- 打印设置
- 汇总模式设置
- 合并模式设置

表视图按钮位于“日常工作”功能区的“排序与筛选”功能组：



新建表视图

设置好表，单击“保存表视图”命令，输入视图名称，选择要保存的项目，单击“确定”按钮。



修改表视图

设置好表，单击“保存表视图”命令，选择要覆盖的表视图名称，单击“确定”按钮。

管理表视图

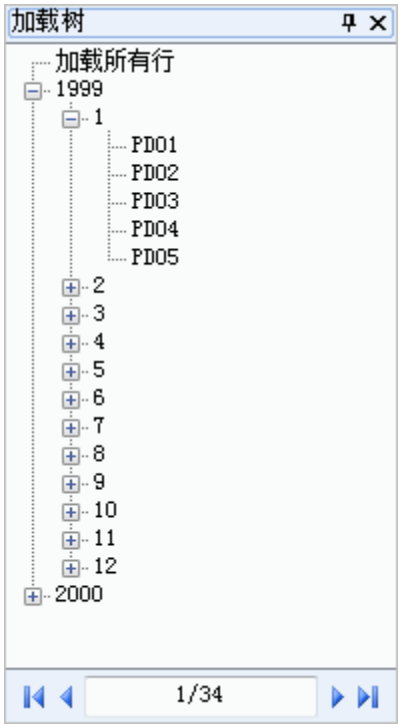
单击“管理表视图”命令，可以删除、重命名视图。

打开表视图

鼠标指向“打开表视图”命令，会列出所有已经定义好的视图名称，单击其中之一，即可打开对应的表视图。

3.21 网络环境下的数据同步

经常有用户会提问： 多用户在网络环境下同时操作，怎么实现数据同步？
其实这对于 Foxtable 来说，是非常简单的，只需使用前面介绍的加载树，这样每次单击某个节点，或者单击上一页、下一页按钮，得到的都是最新的数据。



此外，双击节点，可以刷新当前页的数据。
除了加载树，你也可以执行菜单中的同步数据命令，来加载最新的数据 ，不过显然使用加载树更方便一些。
当然，在《开发指南》部分，我们还会介绍更多的同步数据的方法。

3.22 我还需要继续吗?

到目前为止，Foxtable 的应用篇我们已经讲述完毕，即将进入开发篇。
在进入开发篇之前，你有必要问自己：我还需要继续吗？
Foxtable 是作为一个应用软件而推出的，首先瞄准的目标用户群就是那些不会编程的用户，所以 Foxtable 融合了电子表格和数据库软件的优势，以电子表格的形式将数据库软件特有的功能提供给用户，使得用户不需要任何的编程知识，也可高效地完成输入、查询，统计、报表等日常数据管理工作，很多时候，用户要做的只是单击几次鼠标而已。
但是为了让一些熟悉业务，却不会编程的人，也能设计出专业的、个性化的数据管理软件，所以 Foxtable 同样提供了优秀的编程功能，正因为面对的是不会编程的人，所以相对其他开发工具来说，Foxtable 的开发篇是很容易上手的，而且开发效率很高，用 Foxtable 一天完成的开发任务， 其它开发工具也许需要十天才能完成。
但是再容易的开发工具，也需要一定的时间来学习和实践，才能真正掌握的，如果你很忙，而且

应用篇的功能能基本满足你的要求，那么我建议你到此为止。
如果你最近有些时间，而且有兴趣学习编程，那么请继续。
如果你不打算继续了，我还是建议你看看后面的发布项目这一章，因为这一章和编程无关，能够让你设计好的项目脱离 Foxtable，直接复制到其他电脑运行。

4. 开发指南

4.1 写给初学者

如果你是初次接触编程，那么我们给你几个非常重要的建议：
一、凡事就怕不敢入门
虽然 Foxtable 看上去比易表难了一些，但是只要你静下心来看帮助，你会发现其实并没有难到哪去，深入之后，你也许会觉得更容易。
凡事就怕不敢入门，一旦入门就会觉得：不过如此嘛！
二、不要尝试去记住一切
学习编程的重点，不是记住什么，而是要了解每一段示例代码的原理和逻辑。
要知道，优秀的程序员之所以优秀，绝不是因为他们记住了什么，他们和你我一样，需要不停地借助帮助文件。
三、不要急于求成
磨刀不误砍柴工，心急也吃不得热豆腐。
帮助文件非常详细，提供了大量的示例，内容也是循序渐进的，只要你多点耐心，就一定能掌握。
有的用户只是粗略地看了一次帮助，连基本的概念都没有掌握，就开始开发系统，结果往往是事倍功半。
四、实践出真知
要想全面、深入地掌握 Foxtable，在学习初期，应该将帮助中的示例全部实际操作一次，以便加深印象。
学习的后期，可以自己尝试写一写代码进行测试，特别是有疑惑的地方。
五、三人行必有我师
如果你在学习和应用的过程中遇到难题，自己无法解决，可以访问 Foxtable 社区。
在寻求帮助的同时，也别忘记帮助他人，让我们一起进入 Foxtable 编程的奇妙天地，一起来提高吧。
Foxtable 社区地址：
http://www.foxtable.com/

4.2 最重要的两章

Foxtable 的开发篇非常庞大，其中最重要的有两章，分别是《编程基础》和《Foxtable 编程》。
这两章既是基础，又是精华，可以说掌握了这两章的内容，剩余的内容都是“小菜”，因为开发篇除这两章之外的内容，基本上都是参考性质，在学习的过程中，只需做到有所了解即可，在实际

开发的时候再来查阅 。
所以我们在学习开发篇的时候，应该在这两章花上最多的精力，特别是《Foxtable 编程》这一章，要反复看，细致看，多实践，只有基础打好了，才能融会贯通，才能在开发过程中做到游刃有余。

4.3 编程基础

4.3.1 灾难恢复

再好的程序员，也不能保证自己的代码和设置 100%正确，肯定要反复调试，才能最终达到目标。一些错误可能直到运行的时候才被发现，普通的错误没关系，重新修改代码和设置即可；不过有的错误可能非常严重，以致每次打开项目文件时，Foxtable 就会崩溃退出，或者无法正常进入操作界面；怎样才能回到正常的操作界面，给开发者提供更改错误的机会呢？
很简单，首先按住 Ctrl 键，然后打开出现错误的项目文件，直到项目文件被完全打开，再松开 Ctrl 键即可。
如果打开的过程中出现了登录窗口，先松开 Ctrl 键，选择以开发者身份登录，输入密码，然后重新按住 Ctrl 键，单击登录窗口的确定按钮，直到项目文件被完全打开，再松开 Ctrl 键。
以这种方式打开的项目文件：
1、不会执行用户代码。
2、外部表不会加载数据，只加载结构。
3、系统菜单将代替用户菜单。
4、表达式列不会生效。
在修改错误代码和设置后，重新打开项目文件。
提示：
如果你确认你的代码没有问题，但是还是出现莫名其妙的错误，可以尝试先删除项目目录下的子目录 Bin，然后重新打开项目。

4.3.2 编程语言

FoxTable 使用的编程语言是 Visual Basic，即使您完全没有接触过编程，也不用担心，因为 Foxtable 使用的只是 Visual Basic 中很少的一部分，大多数人都可以在短时间内掌握。
Visual Basic 近似于自然语言，不区分大小写，非常适合普通用户学习使用，这就是我们选择 Visual Basic 作为用户二次开发语言的原因。

4.3.3 运行示例代码

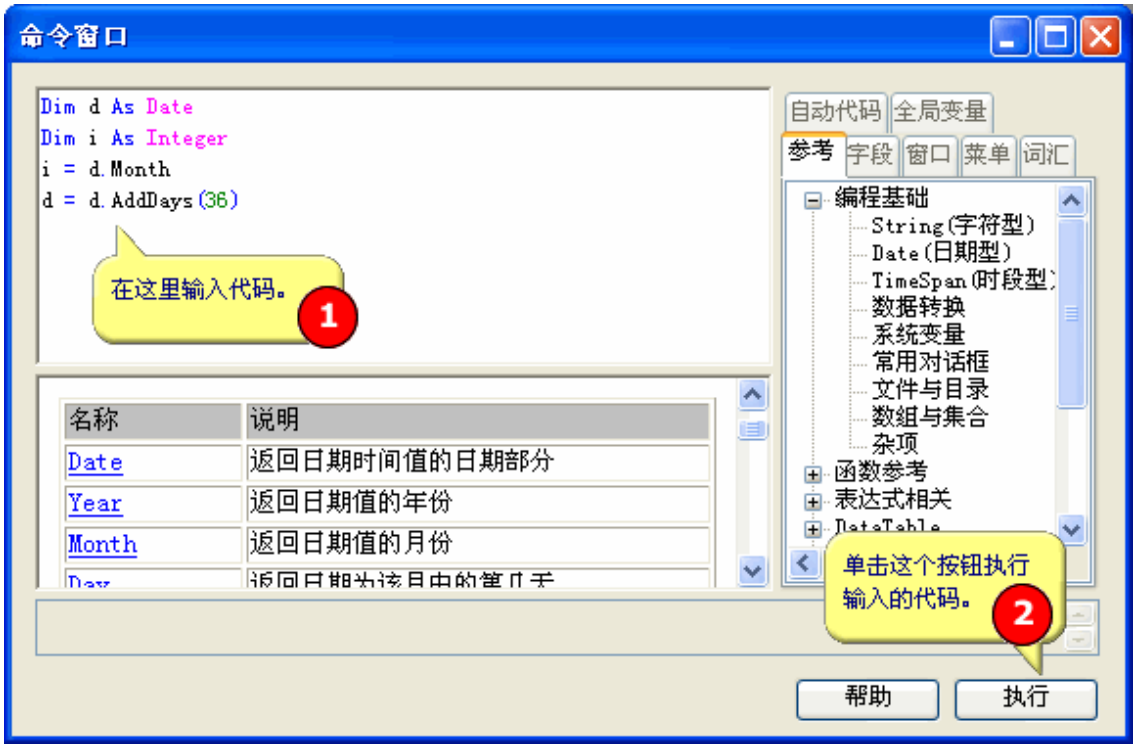
为了便于我们接下来的学习，我们有必要先讲一下命令窗口。
我们在学习的过程中，会接触到大量的示例代码，有时我们有必要通过执行示例代码，来加深印象、解除疑虑。
命令窗口可以直接输入并执行代码 。
需要注意的是，实际开发的时候，并不是在命令窗口编写代码，命令窗口只是为了方便大家在学习过程中，测试运行示例代码，或开发过程中临时性地执行和测试某段代码。

执行代码

在“杂项”功能区，单击下面的按钮，可以打开命令窗口：



命令窗口右下角有一个“执行”按钮，用于执行你输入的代码：



4.3.4 基本概念

4.3.4.1 面向对象编程

Visual Basic 编程是面向对象的，而面向对象的编程，就离不开属性、方法和事件。

程序开发的任务，就是根据用户的动作与要求来设置属性、执行方法和响应事件。

那么什么是属性、方法和事件呢？

我们暂不讲编程，先来一个现实生活的例子，例如对于一辆汽车来说：

1、品牌、型号、颜色、座椅个数、发动机排量等等，就是汽车的属性。

2、启动、熄火、转弯、刹车就是汽车的方法，汽车通过执行这些方法来完成日常工作。

3、为什么踩油门会加速，而踩刹车会减速？ 这是因为驾驶员的每一个操作，都会被当作一个“事件”，并通过传感装置通知到汽车的控制系统。

再来看 Foxtable 中的表，表名、行数、列数就是它的属性；增加行、删除行就是它的方法；而执行方法，又会引发对应的事件，通知系统某个表已经增加了一行、已经删除了一行。

任何你在 Foxtable 中所看到的，包括表，行、列，窗口、菜单等等，都可以看作是一个对象，有自己的属性、方法和事件。

甚至简单到一个数据，也可以看作是一个对象，也会有自己的属性、方法；例如一个字符串，它也有自己的属性，例如长度，还有自己的方法，例如替换、插入、大小写转换等等。

4.3.4.2 数据类型

数据类型	说明
String	任意个数的字符，常称为字符串。
Date	日期时间
Boolean	逻辑值，逻辑值就两个，分别是 True(是)和 False(否)。
Byte	微整数，介于 0 到 255 之间的微整数
Short	短整数，介于 -32768 到 32767 之间的短整型
Integer	整数，介于 -2147483648 到 2147483647 之间的整数。
Long	长整数，值的范围为 -9223372036854775808 到 9223372036854775807。
Single	单精度小数，单精度小数，有效数字为 7 位。
Double	双精度小数，有效数字为 15 位。
Decimal	高精度小数，有高达 28 位的有效数字，适用于要求使用大量有效的整数及小数位数并且没有舍入错误的财务计算。
Object	通用型，可存放任何类型的数据。

数值类型较多，一般情况下，小数用 Double 型，整数用 Integer 型即可，如果整数范围较大，可以用 Long 型。

我们最常用的数据类型是：String、Date、Boolean、Integer、Long、Double。

什么是有效数字？

简单地说，就是从左边第一个不是 0 的数字算起，直到最后一个数字，就是一个数值的有效数字，例如 1.324 的有效数字是 4 位(1、3、2、4)，1.3240 的有效数字是 5 位(1、3、2、4、0)，而 0.024 的有效数字是两位(2、4)。

4.3.4.3 声明变量

变量用来在执行代码的过程中临时存放数据，要使用一个变量，必须先声明它。

声明一个变量，就是要指定变量的名称和它的数据类型。

声明变量的语法是：

Dim 变量名 As 数据类型

例如：

```
Dim Name As String
```

表示声明了一个新的 String(字符串)型变量，变量名为“Name”。

再例如：

```
Dim riqi As Date
```

表示声明了一个新的 Date(日期)型变量，变量名为“riqi”。

可以在定义变量的同时指定其初始值，例如：

```
Dim Name As String = "foxtable"
```

表示声明了一个新的 String(字符串)型变量，变量名为“Name”，初始值为“foxtable”。

可以在同一行定义多个变量，前提是这些变量的类型相同，变量名之间用逗号分割，例如：

```
Dim Val1,Val2 As Integer
Dim Str1,Str2 As String
```

上面的代码，定义了两个整数型变量 Val1、Val2，以及两个字符型变量 Str1、Str2。

提示：上面出现的彩色字符，是 Foxtable 的代码编辑器自动加上去的，只是为了便于阅读，并无特殊的意义。

关于变量名

变量名称必须以字符或者下划线(_)开始，而且至少应该包括一个字符或者数字。

变量名不能包含除下划线(_)之外的符号和空格。

变量名也不能是关键词，所谓关键词，就是对 Visual Basic 有特殊意义的单词，例如上面的 Dim、As。

关键词有几百个，你很难全部记住的，但你只要记住一点，如果输入代码，单击“确定”按钮后，提示“关键词作为标识符无效!”，那么就表明你可能用关键词作为变量名了。

4.3.4.4 圆点符号

任何对象的方法和属性，都必须通过圆点符号才能调用和访问，例如：

```
Dim d As Date
Dim i As Integer
i = d.Month
d = d.AddDays(36)
```

日期型数据有一个属性 Month，用于返回该日期的月份，我们通过圆点符号来访问该属性：

```
i = d.Month
```

现在变量 i 的值等于日期 d 的月份。

日期型数据有一个 AddDays 方法，用于给日期加上指定的天数，并返回新的日期，我们同样通过原点符号调用这个方法：

```
d = d.AddDays(36)
```

现在变量 d 等于原日期加上 36 天后的新日期。

上面的属性和方法我们会在后续章节介绍，你现在只需了解圆点符号的用法即可。

4.3.4.5 共享成员

例如我们定义了一个日期变量 d，那么通过该变量的 Month 属性可得到其月份值：

```
Dim d As Date = #12/31/2008#
Dim i As integer = d.Month
```

上面的代码通过 Month 属性获得日期型变量 d 的月份值，并保存在整数型变量 i 中。

Month 属性和具体的日期变量相关，不同的日期变量，Month 属性返回不同的值；事实上，绝大多

数属性，都和 Month 属性一样，是和具体变量相关的。
但是有一些属性是不同的，它不是通过变量，而是通过类型名称访问，例如要得到今天的日期值：

```
Dim d As Date
d = Date.Today
```

上面的代码将今天的日期值保存在变量 d 中。
请注意用于返回当天日期值的 Today 属性，它不属于任何一个日期型变量，而是属于日期类型 Date，我们直接通过 Date 类型来访问这个属性。
这很好理解，Today 这个属性返回的是今天的日期值，和具体的日期变量无任何关系。
我们将这种直接通过类型名称访问的属性，称为共享属性。
同样直接通过类型名称执行的方法，称为共享方法。
在这里，共享的意思就是：大家共用的，不属于某个具体变量的。

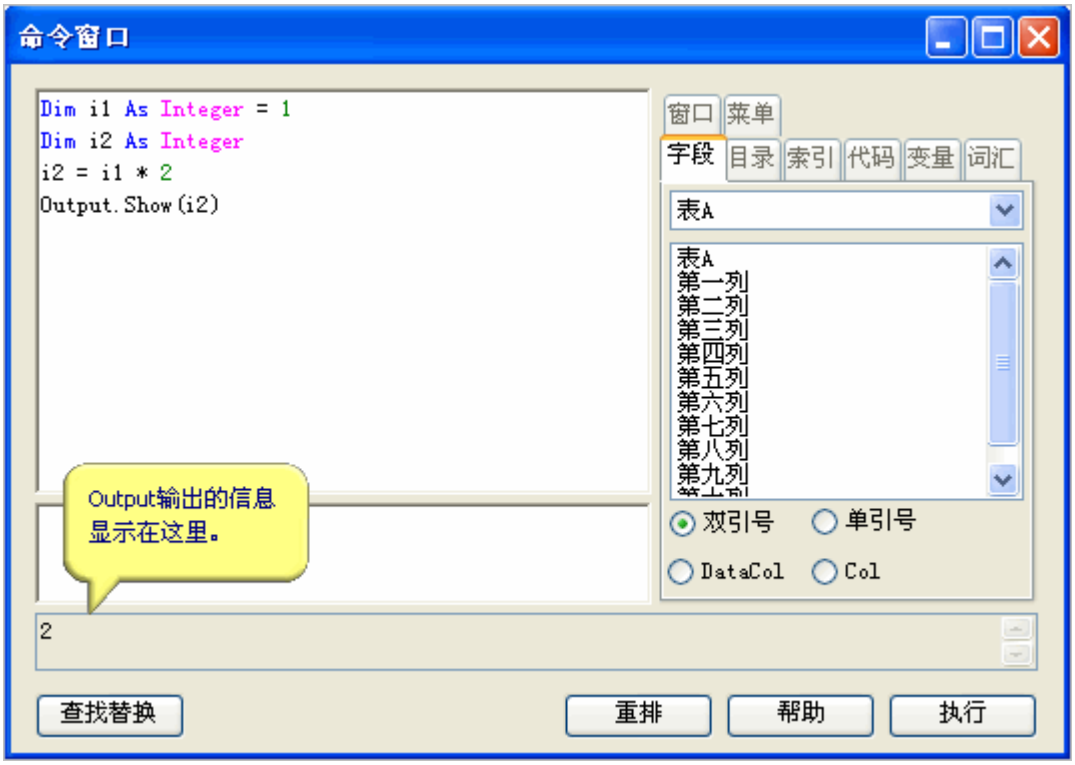
4.3.4.6 命令窗口

4.3.4.6.1 查看代码执行结果

在继续讲述新的内容之前，我们有必要再回头了解一下命令窗口的更多特性。
Foxtable 提供了一个 Output 类型，该类型有一个共享方法 Show，用于在命令窗口显示信息。
例如将下面的代码复制到命令窗口执行：

```
Dim i1 As Integer = 1
Dim i2 As Integer
i2 = i1 * 2
Output.Show(i2)
```

Output.Show 显示的信息，位于命令窗口的信息显示区，具体位置请看下面的图片：



重要特性

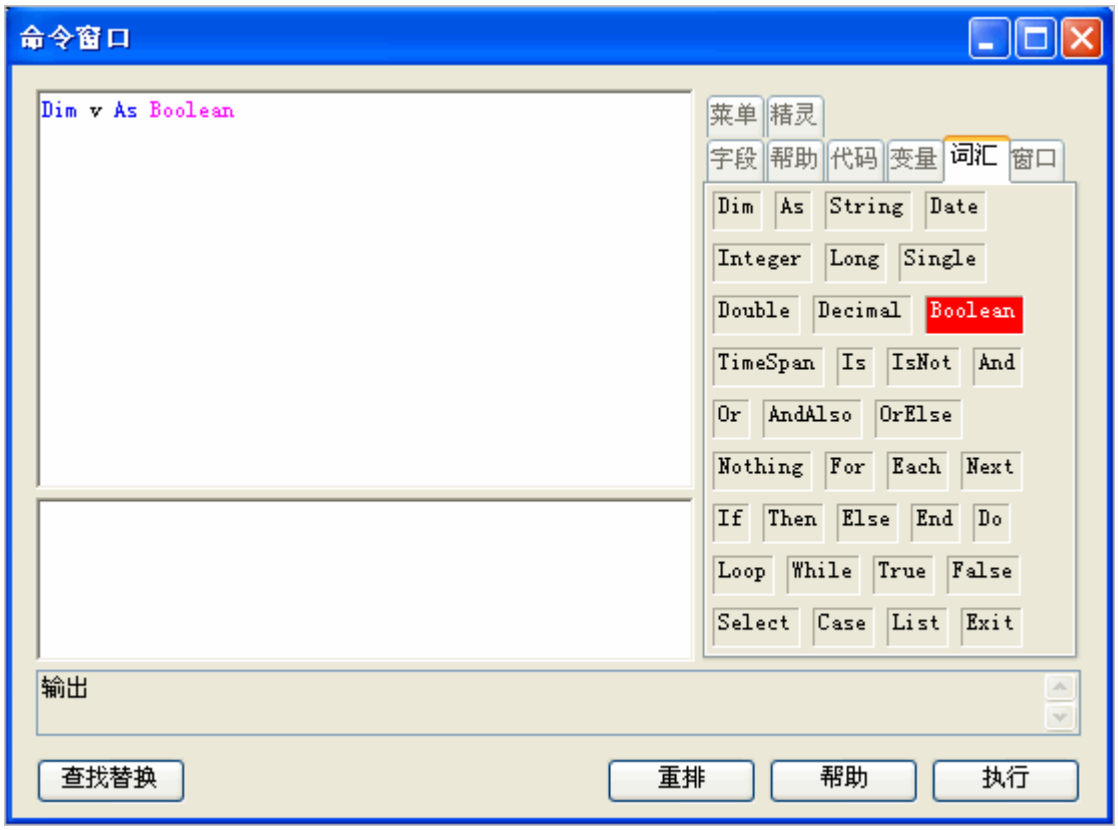
命令窗口为非模式窗口，可以在不关闭命令窗口的情况下，继续进行其它操作，例如编辑数据、执行菜单命令等等。

4.3.4.6.2 命令窗口的辅助功能

词汇

对于初学编程的人来说，记住 Integer、Double、String 这些单词不是一件很容易的事，英文基础差的用户更甚。

不过没关系，命令窗口的词汇页，列出了所有的常见词汇，双击某个词汇，即可自动输入该词汇：

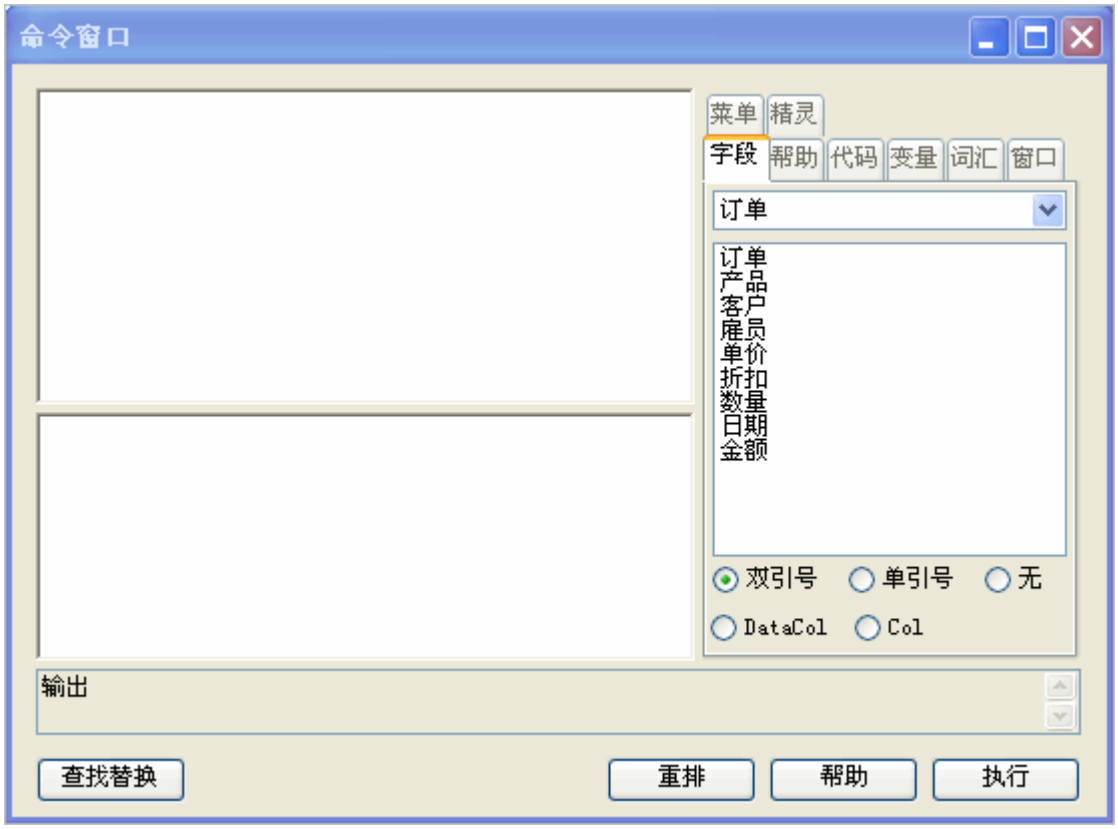


字段

在编写代码的时候，要记住所有的表名和列名，几乎是一个不可能的任务。

其实完全不需要为此担心，命令窗口的字段页列出了所有表的字段(列)，双击即可自动插入字段名。

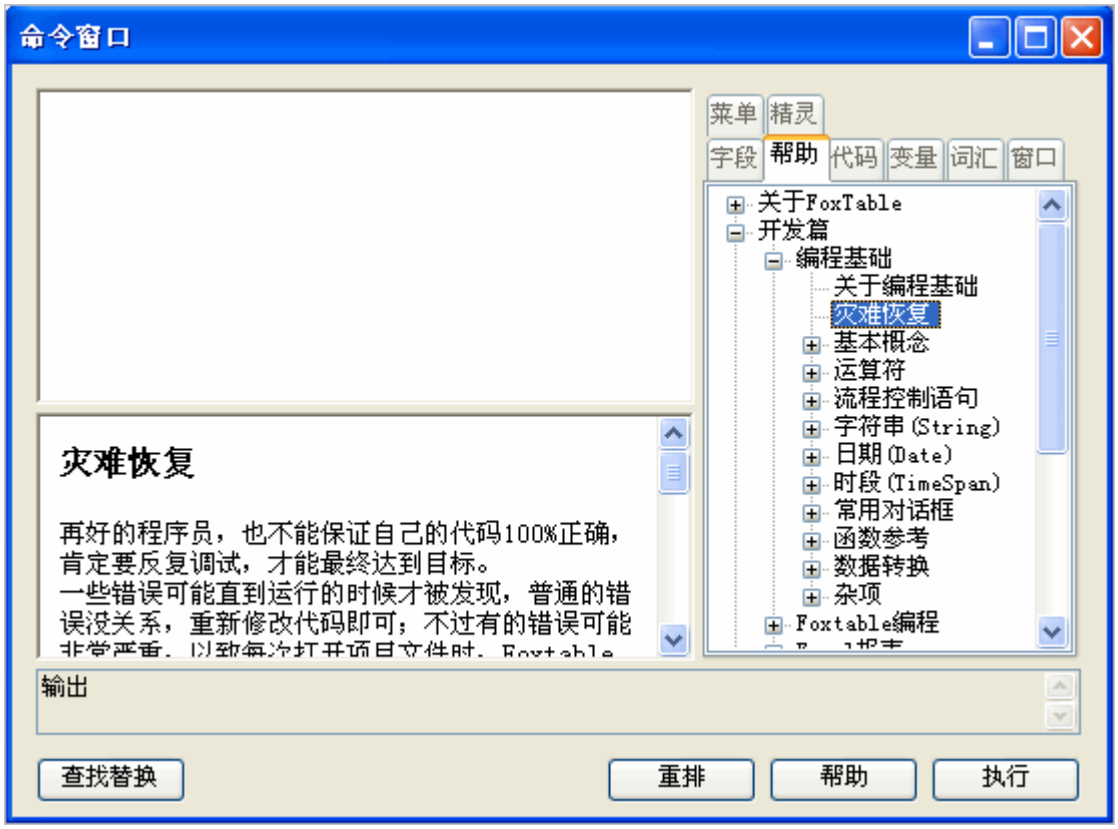
你还可以选择插入的字段名是否用双引号或单引号括起来，还是不需要任何引号。



帮助

命令窗口的帮助页内置了整套帮助文档，可以让大家更便捷地获得帮助。

不过我个人不太喜欢这个内置的帮助，我觉得独立的帮助文档使用起来更方便。

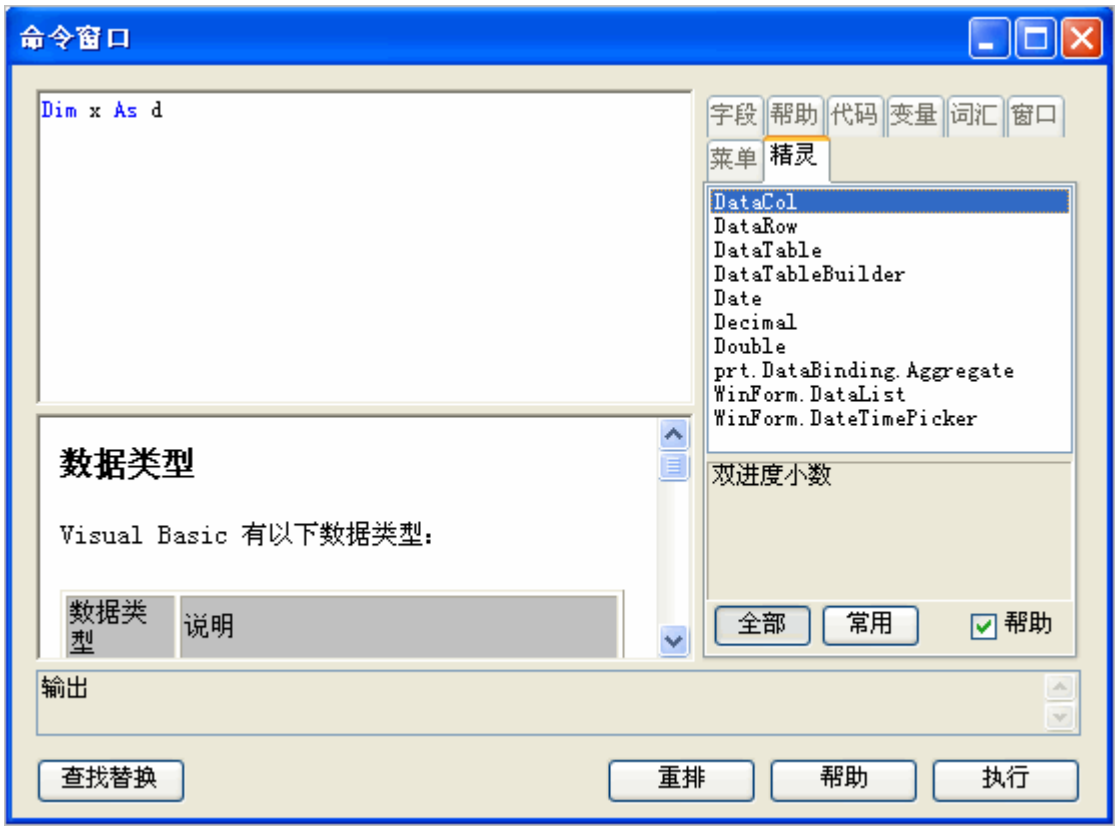


4.3.4.6.3 命令窗口的自动输入功能

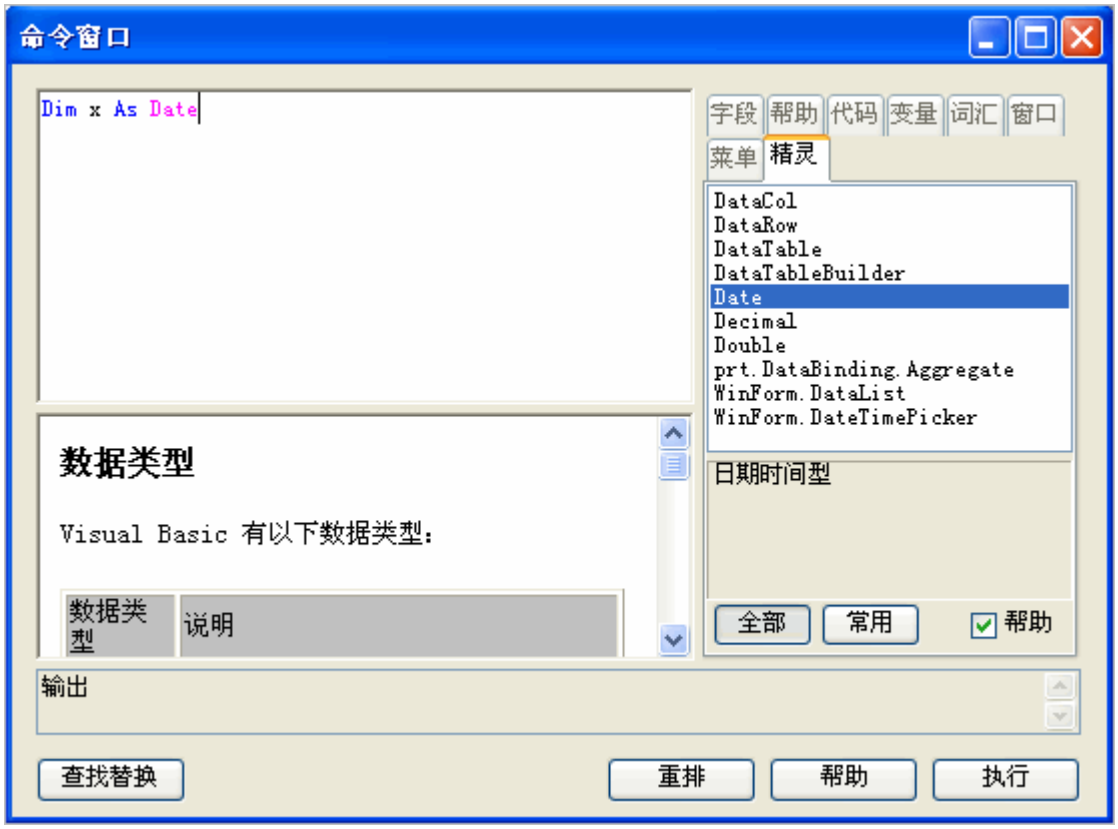
在命令窗口输入：

```
Dim x As d
```

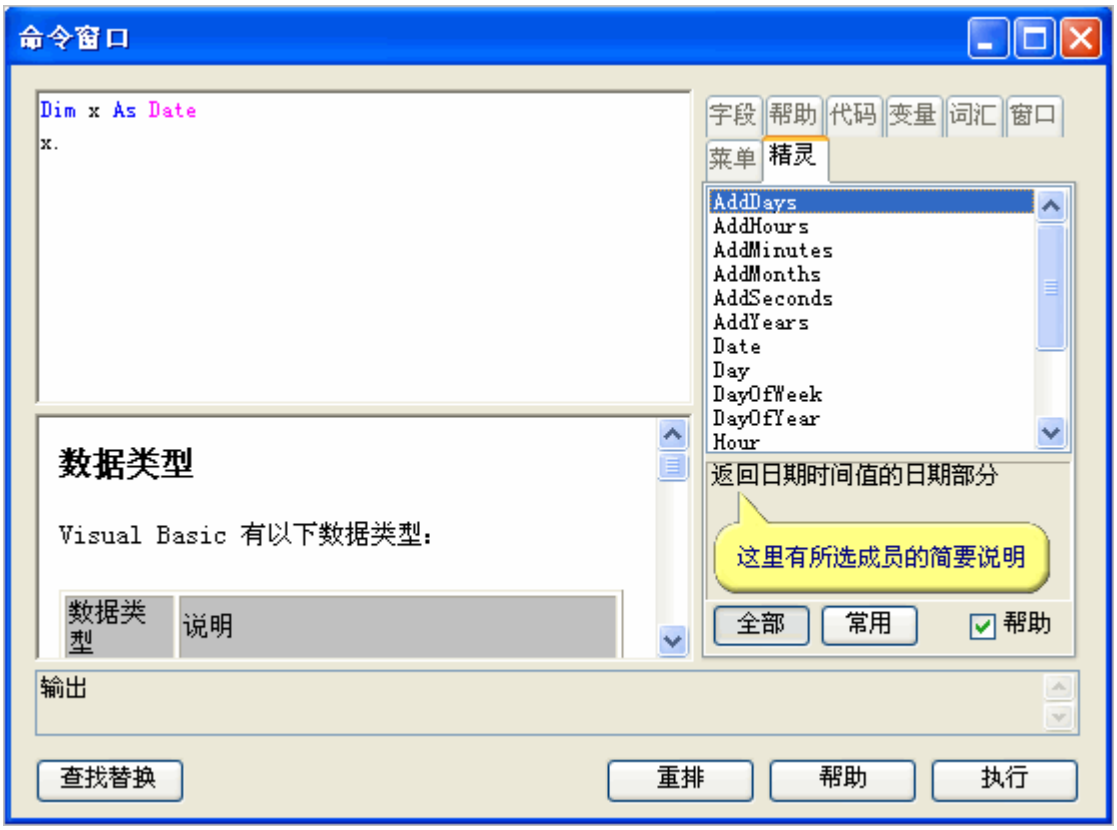
然后按 Tab 键，将列出所有以 d 开头的类型供选择：



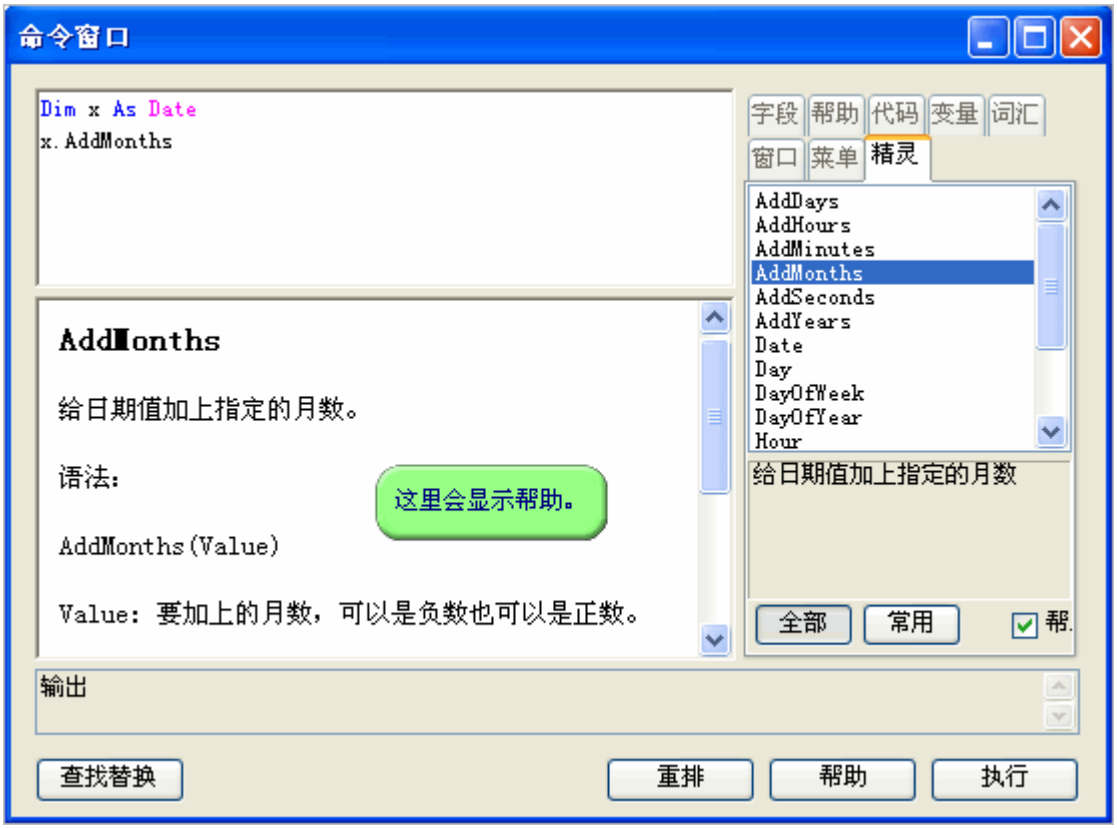
按上下箭头键移动光标，可以选择类型，将光标移到 Date，按回车即可输入类型 Date：



回车后，按下图所示输入"x."，然后按 Tab 键，系统会自动判断出变量 x 为日期型，然后列出日期类型所有的方法和属性供选择：



选择 AddMonths，然后按回车，会自动输入 AddMonths，同时显示 AddMonths 的帮助：



不是任何时候都需要选择的，例如输入：

```
Dim x As String
x.e
```

按 Tab 键，会自动输入 EndsWith:

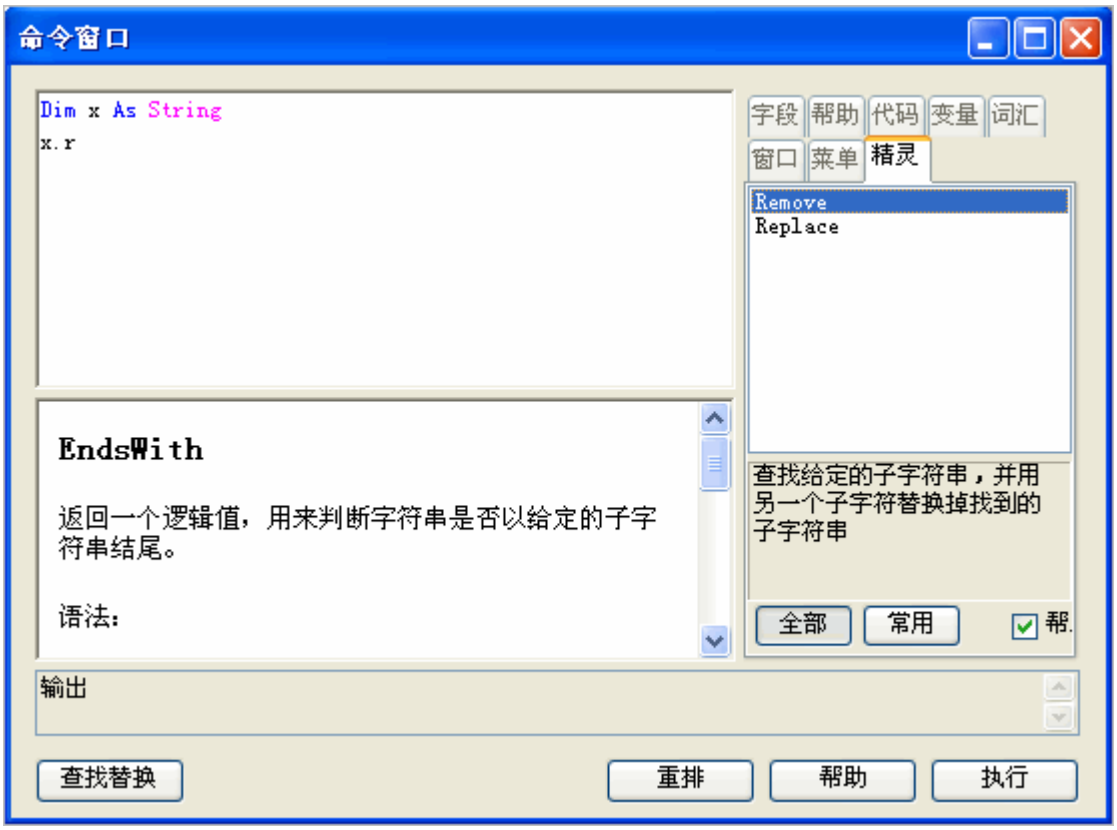
```
Dim x As String
x.EndsWith
```

这是因为 x 是一个字符型变量，而字符型变量以 E 开头的属性或者方法，就只有 EndsWith。

再例如：

```
Dim x As String
x.r
```

按 Tab 键，将列出字符型以 r 开头的属性和方法：



显然手工输入能记住的开头字符，可以更加快速的定位所需要的成员。

目前介绍的是命令窗口(代码编辑器)的基本功能，其它功能，我们会随着学习的深入逐个介绍。

可以看到，在 Foxtable 中编写代码，并不需要死记的，不仅可以自动输入类型、属性和方法，甚至连帮助都是自动显示的。

4.3.4.6.4 自动全角转换和着色

在默认的情况下，代码编辑器会自动将全角字符转换为半角，如果你不喜欢这种强制转换，可以在第一行代码输入：'"。

例如：

```
'"
Dim s1 As String
s1 = "我喜欢 Foxtable，因为..."
Output.Show(s1)
```

此外编辑器会用不同的颜色标出各种关键词，以及表名和列名，如果你的电脑性能不太好，代码又比较长，这种代码自动着色的功能会耗费较长的时间，此时你只需在第一行输入'"..."，即可关闭

自动着色。
例如：

```
'...
Dim s1 As String
s1 = "我喜欢 Foxtable，因为..."
Output.Show(s1)
```

如果你希望同时关闭全角转换和着色，只需在第一行输入"'"..."。
例如：

```
"'...
Dim s1 As String
s1 = "我喜欢 Foxtable,因为..."
Output.Show(s1)
```

4.3.4.6.5 代码注释

给代码的关键位置加上注释，是一个很好的习惯。
注释以符号 '开始，通常是单独一行，或者位于某一行代码之后。
如果注释和代码在同一行，注释和代码之间应该至少有一个空格。
注释不会对代码的运行有任何影响。
例如:

```
'下面的代码用于测试 Month 属性，请复制到命令窗口执行
Dim d1 As Date   '定义变量 d1
d1 = #12/31/2008#   '给变量赋值
Output.Show(d1.Month)   '在命令窗口显示月份
```

上面的代码只是为了说明如何加注释，实际使用的时候，只需给关键的位置加上注释即可，太多的注释，反而会影响代码的可读性。
在代码编辑窗口，注释的颜色为灰色。

4.3.4.7 字符常量

```
Dim s As String
s = "NBA"
```

等效于：

```
Dim s As String = "NBA"
```

表示定义一个字符变量 s，并将 s 的值设为“NBA”。
可以在定义变量的时候，设定该变量为指定个数的某字符组成，语法为：

```
Dim  变量名  As New String(字符,  重复次数)
```

例如：

```
Dim s As New String("*", 5)
```

变量 s 的值将等于"*****"，也就是由 5 个符号*组成。
也可以在定义变量后，用 New String 语句给变量赋值，例如：

```
Dim s As String
```

```
s = New String("*", 5)
```
实际上上面的语句完全等效于：
```
Dim s As String
s = "*****"
```
或：
```
Dim s As String = "*****"
```
不过假定你要将变量设置为 100 个空格或 100 个符号*，此时用 New String 赋值显然要简洁得多。
如果字符串中本身包括双引号，那么用每两个双引号表示一个双引号。
将下面的代码复制到命令窗口执行：
```
Dim s1 As String = "利马是有名的""无雨之都"""
Output.Show(s1)
```
你会发现 s1 的值为：利马是有名的"无雨之都"

4.3.4.8 日期常量

日期型常量必须包括在符号#之间，格式为：#月/日/年#
例如：
```
Dim d1 As Date = #12/31/2008#
```
等效于：
```
Dim d1 As Date
d1 = #12/31/2008#
```
表示定义一个日期变量 d1，初始值为 2008 年 12 月 31 日。
下面是另一种定义日期变量的语法：
```
Dim  变量名  As New Date(年,月,日)
```
例如：
```
Dim StartDate As New Date(2008,12,31)
```
同样表示定义一个日期变量 StartDate，初始值为 2008 年 12 月 31 日。
New Date 表示创建一个新的日期，既然是创建一个日期，当然得指定年、月、日。
New Date 是可以单独使用的，例如：
```
Dim StartDate As Date
StartDate = New Date(2008,12,31) '变量值设为 2008 年 12 月 31 日
Output.Show(StartDate)
```
这一行代码：
```
StartDate = New Date(2008,12,31)
```
等效于：
```
StartDate = #12/31/2008#
```

4.3.4.9 逻辑常量

逻辑常量值就两个，分别为 True(是)和 False(否)。
逻辑变量通常用于保存某一个条件表达式的值。
例如：
```
Dim v1 As Boolean = (1 = 2)
```

```
Dim v2 As Boolean = (1 < 2)
Dim v3 As Boolean = True
Dim v4 As Boolean
```

上述代码执行后，变量 v1 的值等于 False，变量 v2 的值等于 True,因为 1=2 是不成立的，而 1<2 是成立的。

逻辑变量的默认初始值为 False，所以变量 v4 的值等于 False，而变量 v3 的值显然等于 True。

4.3.4.10 枚举

枚举就是一组有名称的整数，例如星期一、星期二、星期三....，就可以看作是枚举，因为他们实际上代表着一个整数，这个整数表示这一天是该星期的第几天。

程序中的枚举，通常用于选项设置，选项通常是一些整数，0、1、2 表示不同的设置，为了提高程序的可读性，我们用一些更具描述性的字符型名称来代替这些整数，这些代表着不同整数的名称，就是枚举。

例如在列属性设置中，有一个名为“中文输入”的属性，它就是一个枚举类型，包括三个值，分别是：

Default：保持当前输入法状态不变

True： 自动打开中文输入法

False： 自动关闭中文输入法

显然，对于用户来说，用 Default、True、False 表示不同的设置，比用 0、1、2 表示不同的设置，要容易理解得多。

关于枚举，你现在只需要了解这么多，今后我们会接触到更具体的枚举类型，那时你会觉得枚举其实是很容易理解的。

4.3.4.11 数组

数组是同一数据类型的一组变量的集合。

定义数组语法为：

```
Dim 数组名(索引上限) As 数据类型
```

例如：

```
Dim Names(3) As String
```

表示定义了一个字符型(String)数组，数组名为“Names”，数组的索引从 0 开始，所以该数组包括 4 个字符型变量（我们称之为数组元素），分别是：

Names(0), Names(1), Names(2), Names(3)

数组有一个 Length 属性，用于返回数组的长度。

下面是一段完整的代码，用于测试数组的基本特性，请复制到命令窗口执行：

```
Dim Names(3) As String
Names (0) = "杨国辉"
Names (1) = "柯迎"
Names (2) = "陈绍月"
Names (3) = "朱志欣"
Output.Show(Names(1)) '在命令窗口显示第二个数组元素的值。
Output.Show(Names.Length) '在命令窗口显示数组的长度。
```

可以在定义数组的时候，直接给数组元素赋值，例如：

```
Dim Values() As Integer = {1,3,5}
```

上面的代码定义了一个整数型数组，数组包括三个元素，分别是 1、3、5。
再例如：

```
Dim s1() As String = {"中国","美国","日本","俄罗斯"}
```

上面的代码定义了一个字符型数组，数组包括四个元素。
如果采用直接赋值的方式定义数组，不需要指定数组的索引上限，此时索引上限由值的个数确定。
可以在定义数组之后，再给数组批量赋值：

```
Dim s1() As String
s1 = New String() {"中国","美国","日本","俄罗斯"}
```

Foxtable 提供了一个类 Array，这个类有大量的共享方法用于对数组进行操作，其中比较常用的有：

名称	说明
IndexOf	获取某个值在数组中第一次出现的位置，如果未出现，返回-1。
LastIndexOf	获取某个值在数组中最后一次出现的位置，如果未出现，返回-1。
Sort	对数组排序。
Reverse	反转数组顺序。

请在命令窗口执行以下代码，用以测试上述方法：

```
Dim vals() As Integer = {1,3,5,2,4,5,6}
Output.Show(Array.Indexof(Vals, 5))
Output.Show(Array.LastIndexof(Vals, 5))
Array.Sort(Vals)
Output.Show(Array.Indexof(vals, 5))
Array.Reverse(vals)
Output.Show(Array.Indexof(vals, 5))
```

得到的输出结果依次为 2、5、4、1，注意索引从 0 开始，所以 1 其实表示位置 2。

Sort 方法可以对两个数组进行联动排序，下面用一个例子来说明，假定一个数组存储了学生姓名，一个数组存储了学生分数，元素位置一一对应，希望按分数高低排序学生姓名，参考代码：

```
Dim nms() As String = {"A","B","C"} '学生姓名
Dim vals() As Integer = {8, 6, 7} '对应分数
Array.Sort(vals, nms) '第一个数组为排序数组,第二个数组跟随第一个数组排序
Output.Show(nms(0) & "=" & vals(0))
Output.Show(nms(1) & "=" & vals(1))
Output.Show(nms(2) & "=" & vals(2))
```

在命令窗口执行上述代码，输出结果依次为：B=6、C=7、A=8，显示两个数组都进行了正确的排序。

4.3.4.12 集合

集合类似于数组，但是比数组更为灵活，定义集合的语法是：

Dim 集合名称 As New List(Of 数据类型）

例如：

```
Dim Names As New List(Of String)
```

表示定义了一个字符型(String)集合，集合的名称为 Names。

集合的成员包括：

名称	类型	说明
Add	方法	向集合中增加一个值
Insert	方法	向集合的指定位置插入一个值
Remove	方法	删除指定的值
RemoveAt	方法	删除指定位置的值
Contains	方法	判断集合中是否包括某个值
Indexof	方法	返回指定值在集合中的位置，如果集合中不存在该值，则返回-1
LastIndexOf	方法	返回指定值在集合中最后一次出现的位置，如果集合中不存在该值，则返回-1
AddRange	方法	批量增加多个值
ToArray	方法	将集合转换为数组
Clear	方法	清除集合中的所有成员
Sort	方法	对集合中的元素进行排序
Reverse	方法	反转集合中元素的顺序
Count	属性	返回集合中值的个数

一旦定义了集合，我们就可以用 Add 方法向其中添加值，例如：

```
Names.Add("电视机")
Names.Add("电冰箱")
Names.Add("洗衣机")
```

可以用 Remove 方法删除指定的值，例如：

```
Names.Remove("电视机")
```

可以用 RemoveAt 删除指定位置的值，例如：

```
Names.RemoveAt(0)
```

表示删除第一个值。

还可以用 Insert 方法将值插入到指定位置，例如：

```
Names.Insert(0,"电视机")
```

可以用 Contains 方法判断集合中是否包括某一个对象，例如：

```
If Names.Contains("电视机")= False Then
   Names.Add("电视机")
End If
```

和数组一样，我们通过索引来引用集合中指定位置的值，例如：

```
Names(0)
Names(1)
```

当然也可以改变指定位置的值，例如：

```
Names(0) = "微波炉"
```

可以用集合的 Count 属性返回集合中值的个数：

```
Names.Count
```

下面是一段完整的代码，用于测试集合的基本特性，请复制到命令窗口执行：

```
Dim Names As New List(Of String)
Names.Add( "电视机")
Names.Add("电冰箱")
Names.Add("洗衣机")
Names.Remove("电视机") '删除电视机
If Names.Contains("电视机")= False Then   '判断集合是否包括"电视机"
   Names.Add("电视机")
End If
Names.Insert(0,"传真机") '将"传真机"作为第一个元素插入
Names(2) = "微波炉" '将第三个元素的值改为"微波炉"
Output.Show(Names.Count) '显示集合的长度
OutPut.Show(Names(0)) '显示第一个值
Output.Show(Names.Indexof("电视机")) '显示电视机在集合中的位置
Names.Sort() '对集合中的元素排序
Output.Show(Names.Indexof("电视机")) '显示电视机在集合中的位置
Names.Reverse() '反转集合中元素的顺序
Output.Show(Names.Indexof("电视机")) '显示电视机在集合中的位置
```

用 AddRange 方法，可以将一个数组或另一个集合中的所有元素加入到集合中，例如：

```
Dim v1() As String = {"中国","美国","日本","俄罗斯"}
Dim s1 As New List(of String)
Dim s2 As New List(of String)
s1.AddRange(v1) '将数组中的元素全部加入到集合中
s2.AddRange(s1) '将集合 s1 中的元素全部加入到集合 s2 中
```

用 ToArray 方法，可以将集合转换为一个数组，例如:

```
Dim Lst As New List(Of String)
Dim Arys() As String
Lst.Add( "电视机")
```

```
Lst.Add("电冰箱")
Lst.Add("洗衣机")
Lst.Remove("电视机")
Arys = Lst.ToArray() '将集合转换为数组
For Each nm As String In lst
    Output.Show(nm)
Next
```

4.3.4.13 字典

Dictionary 表示键与值的集合，俗称字典。

定义一个字典的语法是：

Dim 变量名 As New Dictionary(Of 键类型，值类型)

例如：

Dim zd As New Dictionary(Of Integer, String)

表示定义了一个字典 zd，键的类型是整数型(Integer)，值的类型是字符型(String)。

键和值可以是任何类型，例如：

Dim zd As New Dictionary(Of String, Date)

字典的成员包括：

名称	类型	说明
Add	方法	将指定的键和值添加到字典中
Remove	方法	移除指定的键和值
Clear	方法	清除所有的键和值
ContainsKey	方法	判断是否包含指定的键
ContainsValue	方法	判断是否包括指定的值
Count	属性	返回字典中键/值对的数目
Keys	属性	返回键的集合
Values	属性	返回值的集合

示例

```
Dim zd As New Dictionary(Of Integer, String)
zd.Add(1,"苹果")
zd.Add(2,"香蕉")
zd.Add(3,"波罗")
zd.Add(4,"桔子")
zd.Add(5,"桃子")
zd.Remove(1) '移除键 1 及其对应的值
zd.Remove(5) '移除键 5 及其对应的值
Output.Show(zd(2)) '显示键 2 对应的值
Output.Show(zd(3)) '显示键 3 对应的值
If zd.ContainsKey(1) = False Then '是否存在键 1
    zd.Add(1,"苹果")
```

```
End If
If zd.ContainsValue("桃子") = False Then '是否存在值"桃子"
    zd.Add(5,"桃子")
End If
For Each k As Integer In zd.Keys '显示所有键及其对应的值
    Output.Show(K & ":" & zd(k))
Next
For Each v As String In zd.Values '显示所有值
    Output.Show(v)
Next
zd.Clear '清除所有的键和值
```

4.3.5 运算符

4.3.5.1 算术运算符

运算符	说明
+	加
-	减
*	乘
/	除，例如：12/8，等于 1.5
\	整除，只返回结果的整数部分，例如：12\8，等于 1
^	指数运算，例如：2^3，表示 2 的 3 次方，等于 8
Mod	取模，例如：10 Mod 3，表示 10 除以 3 的余数，等于 1

请将下面的代码复制到命令窗口执行：

```
Output.Show(12/8) '测试除法
Output.Show(12\8) '测试整除
Output.Show(12 Mod 8) '获得余数
Output.Show(2^4) '指数运算
```

4.3.5.2 关系运算符

运算符	说明
=	等于

<>	不等于
>	大于
<	小于
>=	大于等于
<=	小于等于

4.3.5.3 逻辑运算符

逻辑运算符用于连接两个条件表达式，根据左右表达式的计算结果，返回一个新的逻辑值。

运算符	说明
And	逻辑与，如果左右表达式的值都为 True，则返回 True，否则返回 False
AndAlso	逻辑与，如果左右表达式的值都为 True，则返回 True，否则返回 False
Or	逻辑或，如果左右表达式的值至少有一个 True，则返回 True，否则返回 False
OrElse	逻辑或，如果左右表达式的值至少有一个 True，则返回 True，否则返回 False
Not	逻辑非，如果后边的表达式为 True，则返回 False，否则返回 True，也就是对后边表达式的值取反。

补充说明：

And 和 Or，都是先计算左右两个表达式的值，然后再返回结果。

AndAlso 先计算左边表达式的值，如果计算结果为 False，则直接返回 False，不再计算右边表达式的值。

OrElse 先计算左边表达式的值，如果计算结果为 True，则直接返回 True，不再计算右边表达式的值。

所以通常用 AndAlso 和 OrElse 代替 And 和 Or，因为效率高点。

示例

```
Dim v1 As Boolean = True 'True
Dim v2 as Boolean 'False
Dim v3 As Boolean = (1 < 2 AndAlso 2 > 3) 'False
Dim v4 As Boolean = (1 < 2 OrElse 2 > 3) 'True
Dim v5 As Boolean = (v1 AndAlso v4) 'True
Dim v6 As Boolean = (v2 OrElse v3) 'False
```

4.3.5.4 字符连接符

运算符&来连接字符串，例如：

```
Dim s As String
s = "abc" & "123" 's 的值将是"abc123"
```

&可以用来连接任何数据，例如日期型、数值型，在连接之前会自动将非字符型的数据转换为字符型。

例如在命令窗口输入并执行下面的代码，即可显示当天的日期：

```
Output.Show( "今天是：" & Date.Today)
```

需要注意的是，&运算符前后必须有空格，将下面的代码复制到命令窗口执行：

```
Dim s1 As String = "abc"
Dim s2 As String = "def"
Output.Show(s1&s2) '因为没有空格,执行时会出现错误.
```

上述代码在执行时，会出现错误提示，要正确执行，必须改为：

```
Dim s1 As String = "abc"
Dim s2 As String = "def"
Output.Show(s1 & s2) '&运算符前后有了空格,可以正常执行了
```

4.3.5.5 Like 运算符

Like 使用通配符来比较字符串。
可用的通配符有：

字符	说明
?	任何单个字符
*	零或多个字符
#	任何单个数字（0 到 9）
[字符列表]	字符列表中的任何单个字符，例如[abx]表示 a、b、x 三个字符中的任何一个。
[!字符列表]	不在字符列表中的任何单个字符，例如[!abx]表示除 a、b、x 之外的任何一个字符。

方括号中的字符列表，可以使用连字符（-）将范围的上下限分开，例如[a-e]表示字母 a 到 e 中的任意一个，[a-z]就表示任意一个字母，而[0-9]表示任意一数字。

将下面的代码复制到命令窗口执行，请留意注释和执行结果：

```
'任意 3 个字符，后接 3 个数字
Output.Show( "ABC123" Like "???###") '匹配
```

```
Output.Show( "A1212A" Like "???###") '不匹配
'任意一个字符，随后是 ABC 三字符中的任意一个
Output.Show( "RB" Like "?[ABC]") '匹配
Output.Show( "RD" Like "?[ABC]") '不匹配
'任意一个字符，随后是除 ABC 三字符之外的任意一个字符
Output.Show( "RB" Like "?[!ABC]") '不匹配
Output.Show( "RD" Like "?[!ABC]") '匹配
'任意一个字母，随后是两个数字
Output.Show( "a12" Like "[a-z]##") '匹配
Output.Show( "112" Like "[a-z]##") '不匹配
```

4.3.6 流程控制语句

4.3.6.1 If 语句

If 语句有三种形式

单分支形式：

```
If 条件表达式 Then
    代码
End If
```

如果条件表达式的计算结果为 True，也就是条件成立的话，则执行代码。

例如在命令窗口输入下面的代码：

```
Dim Mark As Integer = 61
If Mark >= 60 Then
    Output.Show( "及格")
End If
```

执行之后，上述的代码会得到结果"及格"，如果将变量 Mark 的值改为 59，则不会返回任何内容。

双分支形式:

```
If 条件 Then
    代码 1
Else
    代码 2
End If
```

条件成立，则执行代码 1，不成立，则执行代码 2。

例如，在命令窗口输入下面的代码：

```
Dim Mark As Integer = 61
If Mark >= 60 Then
    Output.Show( "及格")
Else
    Output.Show( "不及格")
End If
```

执行之后，上述的代码会得到结果"及格"，如果将变量 Mark 的值改为 59，则会得到结果"不及格"。

多分支形式：

```
If 条件 1 Then
    代码 1
ElseIf 条件 2 Then
    代码 2
ElseIf 条件 3 Then
    代码 3
    .....
Else
    代码 X
End If
```

条件 1 成立，执行代码 1，条件 2 成立，执行代码 2，其余类推，所有条件均不成立，则执行代码 X。

例如在命令窗口输入：

```
Dim Mark As Integer = 90
If Mark >= 90 Then
    OutPut.Show("优秀")
ElseIf Mark >= 80 Then
    OutPut.Show("良好")
ElseIf Mark >= 60 Then
    OutPut.Show("及格")
Else
    OutPut.Show("不及格")
End If
```

在命令窗口改变变量 Mark 的值，执行看看会得到什么结果。

条件拦截：

在多分支形式下，即使多个条件成立，if 语句只会执行第一个满足条件的分支。

例如在命令窗口执行下面的代码：

```
Dim Mark As Integer = 90
If Mark >= 60 Then
    OutPut.Show("及格")
ElseIf Mark >= 80 Then
    OutPut.Show("良好")
ElseIf Mark >= 90 Then
    OutPut.Show("优秀")
Else
    OutPut.Show("不及格")
End If
```

虽然变量 Mark 等于 90，但是输出的结果是“及格”，而不是“优秀”，因为 Mark 等于 90 的时候，第一个条件(Mark >=60)成立，所以输出“及格”，其余分支不再执行，直接跳转到 End If。

4.3.6.2 Select Case 语句

Select Case 语句也是条件判断语句，通用性不如 If 语句，只能根据一个值的清单来测试一个变量，

但是 Select Case 语句比 If 语句要简洁一些。

Select Case 语句的语法为：

```
Select Case  变量
    Case  值 1
       代码 1
    Case  值 2
       代码 2
    Case  值 3
       代码 3
    Case Else
       代码 X
End Select
```

如果变量的值等于值 1，则执行代码 1，如果等于值 2，则执行代码 2，依次类推，如果不等于已经列出的任何值，则执行代码 X

例如，在命令窗口输入：

```
Dim Value As Integer = 5
Dim Result As String
Select Case Value
    Case 1
       Result = "低于 20 分"
    Case 2
       Result = "低于 60 分"
    Case 3
       Result = "低于 80 分"
    Case 4
       Result = "低于 90 分"
    Case 5
       Result = "超过 90 分"
    Case Else
       Result = "无效值"
End Select
Output.Show(Result)
```

将变量 Value 的值分别设为 1、2、3、4、5、6，然后执行代码，看看会得到什么结果。

一个 Case 语句可以同时设置多个值，各个值之间用逗号分开，例如：

```
Dim Value As Integer = 5
Select Case Value
    Case 1,3,5,7,9
       OutPut.Show("奇数")
    Case 2,4,6,8,10
       OutPut.Show("偶数")
End Select
```

Case 语句还可以是一个范围，起始值和终止值用关键词 To 隔开，例如：

```
Dim Mark As Integer = 65
Dim Result As String
```

```
Select Case Mark
    Case 0 To 40
        Result = "E"
    Case 41 To 60
        Result = "D"
    Case 61 To 70
        Result = "C"
    Case 71 To 90
        Result = "B"
    Case 91 To 100
        Result = "A"
End Select
OutPut.Show(Result)
```

Select Case 语句可以对各种数据类型的变量进行比较，包括字符型、日期型、逻辑型、各种数值型，例如下面使用字符型变量：

```
Dim Value As String = "A"
Dim Result As String
Select Case Value
    Case "E"
        Result = "低于 20 分"
    Case "D"
        Result = "低于 60 分"
    Case "C"
        Result = "低于 80 分"
    Case "B"
        Result = "低于 90 分"
    Case "A"
        Result = "超过 90 分"
    Case Else
        Result = "无效值"
End Select
Output.Show(Result)
```

4.3.6.3 For...Next 语句

For...Next 语句是循环语句的一种，可以将一段代码重复执行设定的次数，语法为：

```
For 计数器变量 = 初始值 To 终止值 Step 步长值
    代码
Next
```

每执行一次代码，计数器变量就加上步长值，然后再次执行代码，直到计数器变量超过终止值。

实际使用的时候，如果步长为 1，可以省略 Step 语句。

例如，求 1 + 2 + 3 +... + 100 的和：

```
Dim Sum As Integer
Dim i As Integer
```

```
For i = 1 to 100
    Sum = Sum + i
Next
Output.Show(Sum)
```

如果要累加 1 到 100 之间的偶数，那么代码应该是：

```
Dim Sum As Integer
Dim i As Integer
For i = 2 To 100 Step 2
    Sum = Sum + i
Next
Output.Show(Sum)
```

在循环过程中，如果提前退出循环，可以用 Exit For 语句，例如：

```
Dim Sum As Integer
Dim i As Integer
For i = 1 To 100
    If i = 51 Then
        Exit For
    End If
    Sum = Sum + i
Next
OutPut.Show(Sum)
```

上面的代码实际上只是求得 1 到 50 的累计值，因为在计数器变量 i 等于 51 的时候,，程序退出了循环。

如果符合特定条件的时候，不执行循环代码，而是直接进行下一次循环，可以用 Continue For 语句，例如对于下面的语句：

```
Dim Sum As Integer
Dim i As Integer
For i = 1 To 100
    If i Mod 2 = 0 Then
        Continue For
    End If
    Sum = Sum + i
Next
Output.Show(Sum)
```

上面的代码累加了 1 到 100 之间的奇数，因为碰到偶数，就直接进行下一次循环，而没有累加到变量 Sum 中。

你可以直接在 For 语句中声明计数器变量，语法为：

```
For 计数器变量 As 数据类型 = 初始值 To 终止值 Step 步长值
    代码
Next
```

此时计数器变量只能在循环代码中使用，例如：

```
Dim Sum As Integer
For i As Integer = 1 To 100
    Sum = Sum + i
```

```
Next
Output.Show(Sum)
```

For...Next 语句经常用于遍历数组或集合，例如下面的代码，定义一个数组，然后给每个数组元素设置值：

```
Dim Values(100) As Integer
For Index As Integer = 0 To Values.Length - 1
    Values(Index) = Index * 2
Next
```

再例如下面的代码，首先定义一个字符型集合，接着向集合加入一些值，最后利用 For...Next 语句列出集合中所有的值：

```
Dim Values As New List(Of String)
Values.Add("北京市")
Values.Add("上海市")
Values.Add("天津市")
For i As Integer = 0 To Values.Count -1
    Output.Show(Values(i))
Next
```

我们可以用 Step 语句将步长设为负值，这样可以倒序来执行，例如：

```
For i As Integer = 100 To 1 Step -1
    Output.Show(i)
Next
```

在命令窗口执行下面的代码，会得到：

```
100
99
98
97
...
3
2
1
```

4.3.6.4 For Each 语句

For Each 语句用来遍历一个集合或数组的全部成员 ，语法为：

```
For Each 循环变量 In 集合
    代码
Next
```

针对集合中每一个成员执行代码；语句中的循环变量，其数据类型必须和集合成员的类型相同。

如下面的代码，首先定义一个字符型集合，接着向集合加入一些值，最后利用 For Each 语句列出集合中所有的值：

```
Dim Values As New List(Of String)
Dim Value As String
Values.Add("北京市")
Values.Add("上海市")
```

```
Values.Add("天津市")
For Each Value In Values
    Output.Show(Value)
Next
```

你可以直接在 For 语句中声明循环变量，语法为：

```
For Each 循环变量 As 变量类型 In 集合
    代码
Next
```

此时循环变量只能在循环代码中使用，例如：

```
Dim Values As New List(Of String)
Values.Add("北京市")
Values.Add("上海市")
Values.Add("天津市")
For Each Value As String In Values
    Output.Show(Value)
Next
```

For Each 语句，同样支持 Exit For 和 Continue For。

例如下面的代码，检查集合中的每一个值，如果是北京市，则继续下一次循环，如果是重庆市，则提示“找到了”，并退出循环；如果是其它值，则显示该值：

```
Dim Values As New List(Of String)
Values.Add("北京市")
Values.Add("上海市")
Values.Add("天津市")
For Each Value As String In Values
    If Value = "北京市" Then
        Continue for
    ElseIf Value = "重庆市" Then
        Output.Show("找到了！ ")
        Exit For
    End If
    OutPut.Show(Value)
Next
```

实际上，For Each 语句不仅可以用来遍历集合，还可以遍历数组，例如：

```
Dim Names As String() = {"日期","客户","雇员"}
For Each Name As String In Names
    Output.Show(Name)
Next
```

上面的代码等效于：

```
Dim Names As String() = {"日期","客户","雇员"}
For i As Integer = 0 To Names.Length -1
    Output.Show(Names(i))
Next
```

用何种方式，看各人喜好，我个人更喜欢前者。

4.3.6.5 Do...Loop 语句

Do...Loop 语句比 For...Next 语句更加灵活，因为它既可以在循环的开始，也可以在循环的结尾测试条件。

Do...Loop 语句有两种形式，分别为：

```
Do While 条件
    代码
Loop
```

和：

```
Do
    代码
Loop While 条件
```

两者的差别是第二种形式会至少执行一次代码，因为它是在执行完代码后再比较条件，而第一种形式是先比较条件，条件符合再执行代码。

例如，求 1 + 2 + 3 +... + 100 的和：

```
Dim Sum As Integer
Dim i As Integer = 1
Do While i <= 100
    Sum = Sum + i
    i = i + 1
Loop
Output.Show(Sum)
```

注意，For...Next 中，计数器变量是根据步长设置自动增加的，而在 Do...Loop 中，需要自行编写代码来改变计数器变量的值(粗体代码部分)，否则程序将处于死循环之中。

如果要累加 1 到 100 之间的偶数，那么代码应该是：

```
Dim Sum As Integer
Dim i As Integer = 2
Do While i <= 100
    Sum = Sum + i
    i = i + 2
Loop
Output.Show(Sum)
```

在循环过程中，如果提前退出循环，可以用 Exit Do 语句，例如：

```
Dim Sum As Integer
Dim i As Integer = 1
Do While i <= 100
    Sum = Sum + i
    i = i + 1
    If i > 50 Then
        Exit Do
    End If
Loop
Output.Show(Sum)
```

上面的代码实际上只是求得 1 到 50 的累计值，因为在 i 大于 50 的时候退出了循环。
如果在满足指定条件的时候，不执行循环代码，而是直接进行下一次循环，可以用 Continue Do 语句，例如对于下面的 代码：

```
Dim Sum As Integer
Dim i As Integer = 0
Do While i < 100
    i = i + 1
    If i Mod 2 = 0 Then
        Continue Do
    End If
    Sum = Sum + i
Loop
Output.Show(Sum)
```

上面的代码累加了 1 到 100 之间的奇数，因为碰到偶数，就直接进行下一次循环，而没有累加到变量 Sum 中。
累加 1 到 100 之间的奇数，更合理的方法是：

```
Dim Sum As Integer
Dim i As Integer = 1
Do While i < 100
    Sum = Sum + i
    i= i + 2
Loop
Output.Show(Sum)
```

循环变量 i 的初始值为 1，每次循环之后，i 的值加 2。

4.3.6.6 With...End With 语句

该语句可以用来简化代码，With 之后紧跟一个对象，在 With 和 End With 之间，可以用圆点符号“.”代替“变量.”。
例如：

```
Dim StartDate As Date = Date.Today
With StartDate
    Output.Show("年:" & .Year)
    Output.Show("月:" & .Month)
    Output.Show("日:" & .Day)
End With
```

上面的代码等效于：

```
Dim StartDate As Date = Date.Today
Output.Show("年:" & StartDate.Year)
Output.Show("月:" & StartDate.Month)
Output.Show("日:" & StartDate.Day)
```

Year、Month、Day 都是日期型数据的属性，后面的章节我们会介绍。

4.3.6.7 Return 语句

Return 语句用于终止代码的执行，如果需要，还可以返回值。
例如：

```
Dim Sum As Integer
Dim i As Integer
For i = 1 To 100
    If i = 51 Then '如果 i 等于 51
        Return Sum '终止执行,并返回值
    End If
    Sum = Sum + i
Next
```

提示，在命令窗口中使用 Return 时，必须返回一个值，而用于事件代码时，没有这个限制。

4.3.6.8 流程语句的编排

相信任何人看到下面的代码，都会觉得不舒服：

```
Dim Sum As Integer
Dim i As Integer
For i = 1 To 100
If i Mod 2 = 0 Then
Continue For
End If
Sum = Sum + i
Next
Output.Show(Sum)
```

我们应该将每个流程语句中的代码缩进一定数量的空格，这样看起来就清爽多了：

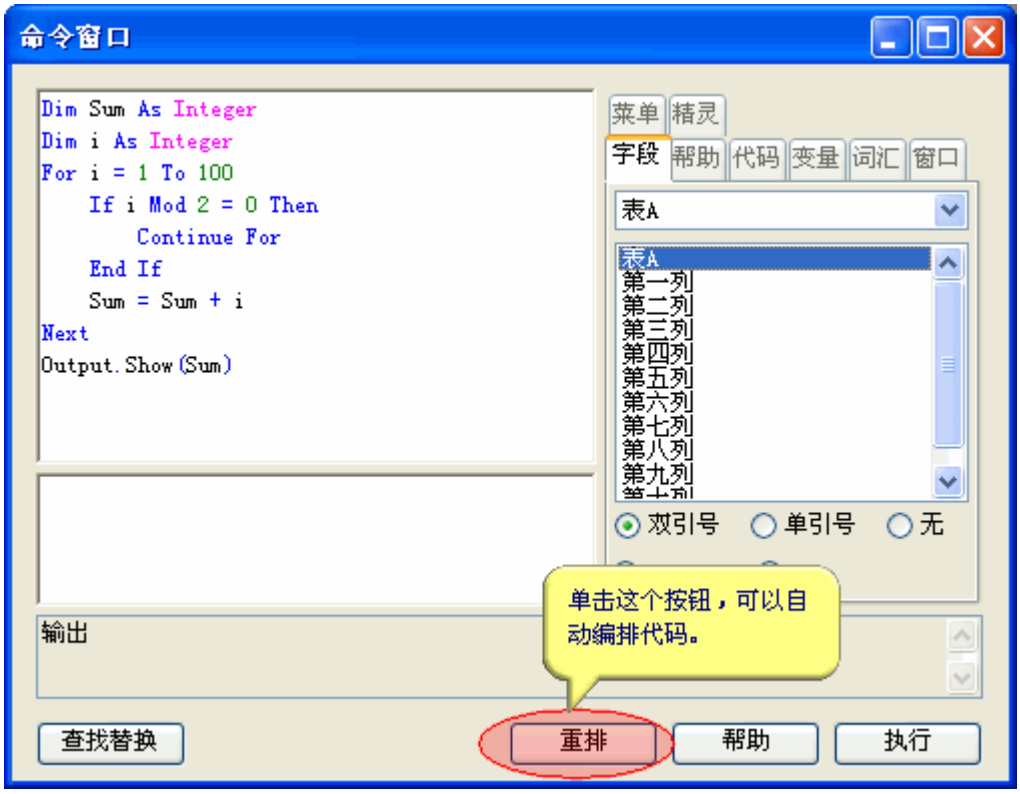
```
Dim Sum As Integer
Dim i As Integer
For i = 1 To 100
    If i Mod 2 = 0 Then
        Continue For
    End If
    Sum = Sum + i
Next
Output.Show(Sum)
```

手动编排

编写代码的时候，在行的开始处，每按一次 Tab 键，就会自动插入四个空格，有了这个特性，就可以轻松地缩进流程语句中的代码了。

自动编排

对于已经输入的代码，可以单击下面按钮，即可自动编排代码，无需手工逐行调整：



4.3.7 字符串(String)

4.3.7.1 字符串概述

字符串有一个 Length 属性，用于返回字符串的长度，例如：

```
Output.Show("中国 ABC".Length)
```

在命令窗口执行上述代码，得到的结果是 5。

如果要判断某一个字符串是否包括另一个字符串，可以用 Contains 方法，例如：

```
Dim Str1 As String = "中华人民共和国"
Output.Show(Str1.Contains("人民"))
```

在命令窗口执行上述代码，得到的结果是 True。

如果需要得到一个字符串在另一个字符串中的位置，可以用 IndexOf 方法，例如：

```
Dim Str1 As String = "中华人民共和国"
Output.Show(Str1.IndexOf("人民"))
```

在命令窗口执行上述代码，得到的结果是 2，注意位置是从 0 开始计算的，第一个字符的位置是 0。

如果要判断字符串是否以另一个字符串开始，可以用 StartsWith 方法，如果要判断字符串是否以另一个字符串结尾，可以用 EndsWith 方法，例如：

```
Output.Show("中国上海".StartsWith("中国"))
Output.Show("中国上海".EndsWith("上海"))
```

用 Trim 方法可以删除字符串前后的空格，例如：

```
Dim Str1 As String = "  人民  ".Trim()
Output.Show(Str1)
```

执行后可看到变量 Str1 的值为“人民”，前后的空格被删除了。

用 Replace 方法可以将字符串中的指定内容替换为另一个字符串，例如：

```
Dim s As String = "abc123abc"
s = s.Replace("123","789") '将 123 替换为 789
OutPut.Show(s)
```

在命令窗口执行上述代码，得到的结果是：abc789abc

ToUpper 将字符串转换为大写形式，ToLower 将字符串转换为小写形式，例如：

```
Dim Str1 As String = "Foxtable".ToUpper()
Dim Str2 As String = "Foxtable".ToLower()
Output.Show(Str1 & "|" & Str2)
```

在命令窗口执行上述代码，得到的结果是：FOXTABLE|foxtable

可以使用 SubString 方法获得从指定位置开始的、指定长度的字符串。

SubString 第一个参数指定开始位置，第二个参数可选，用于指定长度，如果省略这个参数，则返回自指定 位置开始的全部字符，例如：

```
Dim s As String = "I Like Foxtable"
Output.Show(s.SubString(0,6)) '获得左边 6 个字符: I Like
Output.Show(s.SubString(s.Length - 8)) '获得右边 8 个字符:Foxtable
Output.Show(s.SubString(2,4)) '获得从第 3 个字符开始的 4 个字符: Like
```

如果只是希望获得某一位置的单个字符，可以用 Chars 属性，该属性带一个参数，返回该参数指定位置的字符，例如：

```
Output.Show("abcdef".Chars(2))
```

在命令窗口执行上面的代码，结果为“c”，注意编号是从 0 开始计算的。

可以使用 Remove 方法删除从指定位置开始的、指定长度的字符串，例如：

```
Output.Show("123456789".Remove(2,4)) '删除从第 3 个字符开始的连续 4 个字符
```

在命令窗口执行上述代码，得到的结果是：12789

如果要在字符串的指定位置插入另一个字符串，可以用 Insert 方法，例如：

```
Output.Show("123789".Insert(3,"456")) '在位置 4 插入 456
```

在命令窗口执行上述代码，得到的结果是：123456789

利用 Split 方法，可以用指定的分割符号，将一个字符串拆分成一个字符数组，例如：

```
Dim Tel As String = "0759-3166876"
Dim Parts() As String = Tel.Split("-")
Output.Show("区号:" & Parts(0))
Output.Show("电话:" & Parts(1))
```

在命令窗口执行上述代码，得到的结果是：

区号:0759

电话:3166876

4.3.7.2 全部成员

名称	说明
Contains	判断字符串是否包含给定的子字符串。
StartsWith	判断字符串是否以给定的子字符串开头
EndsWith	判断字符串是否以给定的子字符串结尾

Trim	删除字符串前后的空格或指定的字符
TrimEnd	删除字符串后面的空格或指定的字符
TrimStart	删除字符串前面的空格或指定的字符
IndexOf	查找给定的子字符串第一次出现的位置
LastIndexOf	查找给定的子字符串最后一次出现的位置
ToUpper	将字符串转换为大写
ToLower	将字符串转换为小写
SubString	返回从指定位置开始指定长度的子字符串
Replace	查找给定的子字符串，并用另一个子字符替换掉找到的子字符串
Remove	从指定位置开始，删除指定个数的字符
PadLeft	在字符串左侧填充空格或指定的字符，使得字符串的长度达到指定的长度
PadRight	在字符串右侧填充空格或指定的字符，使得字符串的长度达到指定的长度
Split	用指定的分隔符将字符串分隔成一个字符串数组
Insert	在指定位置插入一个子字符串

属性

名称	说明
Chars	返回指定位置的字符
Length	返回字符串的长度

共享成员

名称	说明
Compare	用于比较字符串，可以区分大小写。
Join	指定的分割符号，将一个字符串数组的全部元素合并为一个字符串。

4.3.7.3 两个重要函数

如果我们要取字符串左边指定个数的字符，例如取左边 3 个字符，代码通常为：

```
Dim v As String  = "Foxtable"
Output.show(v.SubString(0,3))
```

如果要取右边3个字符，则更麻烦：

```
Dim v As String  = "Foxtable"
Dim i As Integer = v.Length
If i >= 3 Then
    Output.show(v.SubString(i - 3,3))
```

```
Else
    Output.show(v)
End If
```

而在开发过程中，取左侧或右侧固定个数的字符是很常用的需求，所以我们介绍两个VB6时代的函数，分别为：

- **Left**
  取左侧固定个数的字符

- **Right**
  取右侧固定个数的字符

例如：

```
Dim s1 As String  = "Foxtable"
Dim s2 As String = Left(s1,3)  '取左侧3个字符
Dim s3 As String = Right(s1,3) '取右侧三个字符
```

显然这比使用 SubString 方法方便很多。
VB6 时代的函数都可以在 Foxtable 使用，但多数函数已经被相关属性和方法完美替代，没有必要使用了，Left 和 Right 两个函数是少有的还有使用价值的函数，所以专门介绍一下。

4.3.8 日期(Date)

4.3.8.1 日期与时间

日期和时间的数据类型都是 Date，也就是说 Date 类型的变量，既可以存储日期，也可以存储时间，也可以同时存储日期和时间。
只是出于简单考虑，我们将类型 Date，简称为日期型。
时间包括时、分、秒三个部分，时分秒之间用符号冒号隔开，例如：
#9:30:21#
也可以只包括时分，例如：
#9:30#
日期和时间可以连在一起，中间用一个空格隔开，例如：
#12/31/2007 9:30:21#
表示 2007 年 12 月 31 日 9 时 30 分 21 秒。
再例如下面的代码，定义了三个日期型变量：

```
Dim Day As Date = #12/31/2008# '只包括日期
Dim Time As Date = #9:54:21# '只包括时间
dim DayTime As date = #12/31/2007 10:30:19# '同时包括日期和时间
```

4.3.8.2 日期概述

Date 类型有一个共享属性 Today，用于返回系统日期，如果要返回系统时间，用共享属性 Now，例如：

```
Output.Show("今天是:" & Date.Today())
Output.Show("现在是:" & Date.Now())
```

在命令窗口执行上述代码，即可显示今天的日期和时间。

如果要判断某一年是否是闰年，可以用共享方法 IsLeapYear，例如：

```
Output.Show(Date.IsLeapYear(1999)) '判断 1999 年是否是闰年
```

利用共享方法 DaysInMonth 可以获得指定月份的天数，例如：

```
Output.Show("2008 年 2 月有" & Date.DaysInMonth(2008,2) & "天")
```

获得某个月的第一天和最后一天的日期，是在开发过程中经常碰到的一个任务，利用 DaysInMonth 可以很轻松地完成这个任务，以 2008 年 2 月份为例：

```
Dim FirstDay As Date = New Date(2008,2,1) '第一天
Dim LastDay As Date = New Date(2008,2,Date.DaysInMonth(2008,2)) '最后一天
```

利用 Year、Month、Day 属性，可以获得给定日期的年、月、日值，例如：

```
Dim d1 As Date = Date.Today
Dim year As Integer = d1.Year
Dim month As Integer = d1.Month
Dim day as Integer = d1.Day
```

利用 DayOfWeek 属性，可以知道给定的日期是星期几，例如：

```
Dim d As Date = #10/31/2008#
Output.Show(d.DayOfWeek) '显示 5
```

如果是星期天，返回值是 0，因为星期天被认为是每周的第一天。

利用方法 AddYears、AddMonths、AddDays 方法，可以分别给日期加上一定的年数、月数、天数，并返回相加后的日期，例如：

```
Output.Show("50 天后是" & Date.Today.AddDays(50))
```

4.3.8.3 全部成员

Year	返回日期值的年份
Month	返回日期值的月份
Day	返回日期为该月中的第几天
DayOfWeek	返回日期为该星期的第几天，也就是星期几
DayOfYear	返回日期是该年中的第几天
Hour	返回日期时间值的小时部分
Minute	返回日期时间值的分钟部分
Second	返回日期时间值的秒部分

方法

名称	说明

AddYears	给日期值加上指定的年数
AddMonths	给日期值加上指定的月数
AddDays	给日期值加上指定的天数
AddHours	给时间值加上指定的小时数
AddMinutes	给时间值加上指定的分钟数
AddSeconds	给时间值加上指定的秒数

共享成员

名称	说明
Now	返回计算机的当前时间(含日期)。
Today	返回计算机的当前日期，不含时间。
IsLeapYear	判断指定的年份是否是闰年。
DaysInMonth	返回指定月份的天数

4.3.8.4 DateYMD

DateYMD 函数并不是 Date 的成员，是一个独立的函数，是 Foxtable 考虑到实际需要，额外提供的一个函数。

DateYMD 函数用于计算两个日期之间相差的年月日

语法为：

DateYMD(dt1, dt2, y, m, d)

dt1: 第一个日期

dt2: 第二个日期

y: 整数型变量，用于保存两个日期之间相差的年数

m: 整数型变量，用于保存两个日期之间相差的月数

d: 整数型变量，用于保存两个日期之间相差的日数

示例

```
Dim y,m,d As Integer
Dim dt1 As Date = #2/28/2012#
Dim dt2 As Date = #2/22/2016#
DateYMD(dt1,dt2,y,m,d)
Output.Show(y & "年" & m & "月" & d & "日")
```

得到的结果是：3年11月25日

4.3.9 时段(TimeSpan)

4.3.9.1 时段概述

时段的类型为 TimeSpan，所谓时段，就是一段时间范围。
定义一个新的 TimeSpan 的语法为：
New TimeSpan(days, hours, minutes, seconds, milliseconds)
New TimeSpan(days, hours, minutes, seconds)
New TimeSpan(hours, minutes, seconds)
days: 天数
hours: 小时数
minutes: 分数
seconds: 秒数
milliseconds: 毫秒数
例如：

```
Dim tp1 As New TimeSpan(20,13,45) '20 小时 13 分 45 秒
Dim tp2 As New TimeSpan(1,20,13,45) '1 天 20 小时 13 分 45 秒
Dim tp3 As New TimeSpan(1,20,13,45,200) '1 天 20 小时 13 分 45 秒 200 毫秒
Dim tp4 As New TimeSpan(0,0,0,200) '200 秒
Dim tp5 As New TimeSpan(0,0,0,0,200) '200 毫秒
```

时段数据有以下常用属性：

属性	说明
Days	获得时段数据的天数
Hours	获得时段数据的小时数
Minutes	获得时段数据的分钟数
Seconds	获得时段数据的秒数

两个日期时间数据相减，得到的就是一个时段型数据。
例如：

```
Dim d1 as Date = #3/17/2002 12:30:29#
Dim d2 as Date = #1/1/2008 10:11:49#
Dim t As TimeSpan = d2 - d1
Output.Show(t.Days & "天" & t.Hours & "小时" & t.Minutes & "分" & t.Seconds & "秒")
```

在命令窗口执行上述代码，得到的输出结果是：2115 天 21 小时 41 分 20 秒
利用时段数据的下述属性，可以将时段转换为一个数值：

属性	说明
TotalDays	将时段值转换

	为以天为单位的数值
TotalHours	将时段值转换为以小时为单位的数值
TotalMinutes	将时段值转换为以分为单位的数值
TotalSeconds	将时段值转换为以秒为单位的数值

例如：

```
Dim d1 as Date = #3/17/2002 12:30#
Dim d2 as Date = #1/1/2008 18:30#
Dim t As TimeSpan = d2 - d1
Output.Show(t.TotalDays & "天")
```

在命令窗口执行上述代码，得到的输出结果是：2116.25天

利用时段数据的Add和Subtract方法，可以进行时段的加减，并得到一个新的时段值。

例如：

```
Dim t0 As New TimeSpan(10,0,5)
Dim t1 As New TimeSpan(40,6,30,15)
Dim t2 As New TimeSpan(40,20,45)
Dim t3 As TimeSpan
t3 = t1.Subtract(t2) 't3 等于 t1 减去 t2
t3 = t3.Add(t0) 't3 加上 t0
Output.Show(t3.Days & "天" & t3.Hours & "小时" & t3.Minutes & "分" & t3.Seconds & "秒")
```

在命令窗口执行上述代码，得到的输出结果是：39 天 0 小时 9 分 35 秒

有时为了测试代码的效率，我们需要记录代码执行所耗费的秒数，可以参考下面的代码:

```
Dim t1 As Date = Date.Now
Dim Sum As Long
For i As Integer = 1 To 10000000
    Sum = Sum + i
Next
Output.Show("计算结束，耗时: " & (Date.Now - t1).TotalSeconds & "秒")
```

4.3.9.2 全部成员

名称	说明
Days	返回时段数据的天数
Hours	返回时段数据的小时数
Minutes	返回时段数据的分钟数
Seconds	返回时段数据的秒数
Milliseconds	返回时段数据的毫秒数
TotalDays	将时段值转换为以日为单位的数值，含小数，例如某个时段数据的值是 1 天 6 小时，那么其 TotalDays 属性值为 1.25
TotalHours	将时段值转换为以小时为单位的数值，含小数，例如某个时段数据的值是 1 天 6 小时 30 分，那么其 TotalHours 属性为 30.5
TotalMinutes	将时段值转换为以分钟为单位的数值，例如某个时段数据的值是 1 天 6 小时 30 分 15 秒，那么其 TotalMinutes 属性为 1830.25
TotalSeconds	将时段值转换为以秒为单位的数值，含小数，例如某个时段数据的值是 1 天 6 小时 30 分 15 秒，那么其 TotalSeconds 属性为 109815
TotalMilliseconds	将时段值转换为以毫秒为单位的数值，例如某个时段数据的值是 1 天 6 小时 30 分 15 秒 200 毫秒，那么其 TotalMilliseconds 属性为 109815200

方法

名称	说明
Add	将时段数据加上另一个时段数据，返回相加后的时段数据值
Subtract	将时段数据减去另一个时段数据，返回相减后的时段数据值

4.3.10 常用对话框

MessageBox
信息显示对话框

OpenFileDialog
打开文件对话框

SaveFileDialog
保存文件对话框

FolderBrowserDialog
目录选择对话框

4.3.10.1 MessageBox

MessageBox 是一个预定义对话框，用于向用户显示信息，并根据用户单击按钮的动作返回不同的值。

MessageBox 有一个共享方法 Show，用于显示对话框，语法是：

MessageBox.Show(Message, Caption, Buttons, Icon)

参数	说明
Message	要显示的信息
Caption	对话框标题
Buttons	可选参数，指定要显示的按钮，MessageBoxButtons 型枚举，包括以下可选值： AbortRetryIgnore：包含“中止”、“重试”和“忽略”按钮。 OK：包含“确定”按钮 OKCancel：包含“确定”和“取消”按钮 RetryCancel：包含“重试”和“取消”按钮 YesNo：包含“是”和“否”按钮 YesNoCancel：包含“是”、“否”和“取消”按钮 如果省略，则显示“确定”按钮。
Icon	可选参数，指定要显示的图标，MessageBoxIcon 型枚举，包括以下可选值： Error：显示错误图标，由一个红色背景的圆圈及其中的白色 X 组成的。 Information：显示信息图标，由一个圆圈及其中的小写字母 i 组成的。 None：不显示图标。 Question：显示提问图标，是由一个圆圈和其中的

	Warning：	一个问号组成的。 显示警告图标，一个黄色背景的三角形及其中的一个感叹号组成的。

Show 方法会返回一个值，该值是一个 DialogResult 型枚举，对应于用户在关闭窗口之前所单击的按钮，包括以下可选值：

Abort： 单击了“中止”按钮
Cancel： 单击了“取消”按钮
Ignore： 单击了“忽略”按钮
No： 单击了“否”按钮
OK： 单击了“确定”按钮
Retry： 单击了“重试”按钮
Yes： 单击了“是”按钮

下面提供了一些 MessageBox 的示例，请分别复制到命令窗口执行测试。

示例一

最简单的显示：

```
MessageBox.Show("Visual Basic 一点不难！ ")
```

显示的对话框：



示例二

加上标题：

```
MessageBox.Show("Visual Basic 一点不难！ ", "提示")
```

显示的对话框：



示例三

指定要显示的按钮：

```
MessageBox.Show("Visual Basic 一点不难，是不是？ ", "提示", MessageBoxButtons.YesNo)
```

显示的对话框：



示例四

加上图标：

MessageBox.Show("Visual Basic 一点不难，是不是？", "提示", MessageBoxButtons.YesNo, MessageBoxIcon.Question)
显示的对话框：



示例五
判断用户单击了哪个按钮：

```
Dim Result As DialogResult
Result = MessageBox.Show("Visual Basic 一点不难，是不是？", "提示", MessageBoxButtons.YesNo, MessageBoxIcon.Question)
If Result = DialogResult.Yes Then
    MessageBox.Show("不错，说明你已经认真学习了。","提示")
Else
    MessageBox.Show("好好学习吧，你会认同这个看法的。","提示")
End if
```

4.3.10.2 OpenFileDialog

打开文件对话框。
属性

- **FileName**

  返回选择的文件名，含路径。

- **MultiSelect**

  逻辑型，是否允许选择多个文件。

- **FileNames**

  字符型数组，返回选定的多个文件，含路径。

- **Filter**

  设置文件名筛选器。筛选器用符号“|”分为两个部分，前面是筛选器说明，后面是筛选器模式。
  例如要选择文本文件，可设为：
  文本文件|*.txt
  一个筛选器可以包括多个筛选模式，不同的模式之间用分号隔开，例如：

图形文件|*.bmp;*.jpg;*.gif
可以同时设置多个过滤器，用符号“|”隔开即可，例如：
Excel 文件|*.xls|Word 文件|*.doc|Access 文件|*.mdb

- **Title**

  对话框标题

- **InitialDirectory**

  打开对话框时，最初显示的目录。

方法

- **ShowDialog**

  显示打开文件对话框。Show 的返回值是 DialogResult 型枚举，如果用户单击的是确定按钮，则返回 DialogResult.OK，如果单击的是“取消”按钮，则返回 DialogResult.Cancel。

示例一
默认每次只能选择一个文件，通过 FileName 属性返回选择的文件名：

```
Dim dlg As New OpenFileDialog '定义一个新的 OpenFileDialog
dlg.Filter= "Excel 文件|*.xls|Word 文件|*.doc|Access 文件|*.mdb" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then '如果用户单击了确定按钮
    MessageBox.Show("你选择的是： " & dlg.FileName,"提示") '提示用户选择的文件
End If
```

示例二
如果将 MultiSelect 属性设置为 True，可以选择多个文件，通过 FileNames 属性返回选择的多个文件，例如：

```
Dim dlg As New OpenFileDialog
dlg.MultiSelect = True
If dlg.ShowDialog = DialogResult.OK Then
    OutPut.Show("你选择了" & dlg.FileNames.Length & "个文件，分别是:")
    For Each fl As String In dlg.FileNames
        Output.Show(fl)
    Next
End If
```

4.3.10.3 SaveFileDialog

保存文件对话框。
属性

- **Title**

  对话框标题

- **InitialDirectory**

  打开对话框时，最初显示的目录。

- **FileName**

  在对话框中选定的文件名。

- **Filter**

  设置文件名筛选器。筛选器用符号“|”分为两个部分，前面是筛选器说明，后面是筛选器模式。
  例如要选择文本文件，可设为：
  文本文件|*.txt
  一个筛选器可以包括多个筛选模式，不同的模式之间用分号隔开，例如：
  图形文件|*.bmp;*.jpg;*.gif
  可以同时设置多个过滤器，用符号“|”隔开即可，例如：
  Excel 文件|*.xls|Word 文件|*.doc|Access 文件|*.mdb

- **OverwritePrompt**

  逻辑型，对于已经存在的文件名，是否出现覆盖警告。

方法

- **ShowDialog**

  显示保存文件对话框。Show 的返回值是 DialogResult 型枚举，如果用户单击的是确定按钮，则返回 DialogResult.OK，如果单击的是“取消”按钮，则返回 DialogResult.Cancel。

示例

请将下面的代码复制到命令窗口执行：

```
Dim dlg As New SaveFileDialog '定义一个新的 SaveFileDialog
dlg.Filter= "文本文件|*.txt" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then '如果用户单击了确定按钮
    MessageBox.Show("你要保存为：" & dlg.FileName,"提示") '提示用户选择的文件
End If
```

4.3.10.4 FolderBrowserDialog

选择目录对话框。

属性

- **Description**
  对话框的说明。

- **SelectedPath**

  用户选定的目录。

- **ShowNewFolderButton**
  逻辑型，是否显示“新建文件夹”按钮。

方法

- **ShowDialog**
  显示打开文件对话框。Show 的返回值是 DialogResult 型枚举，如果用户单击的是确定按钮，则返回 DialogResult.OK，如果单击的是“取消”按钮，则返回 DialogResult.Cancel。

示例
请将下面的代码复制到命令窗口执行：

```
Dim dlg As New FolderBrowserDialog
If dlg.ShowDialog = DialogResult.Ok Then
    MessageBox.Show("你选择的目录是：" & dlg.SelectedPath,"提示")
End If
```

4.3.10.5 命令窗口与枚举值

前面介绍的对话框，都大量使用了枚举，要记住每个枚举的全部成员几乎是不可能的，不过不用担心，因为命令窗口可以自动输入枚举值。
我们还是用例子来说明：
首先在命令窗口输入：

```
Dim o As op
```

按 Tab，上面的代码将自动变为：

```
Dim o As OpenFileDialog
```

并同时显示 OpenFileDialog 的帮助，然后回车，输入：

```
If o.s
```

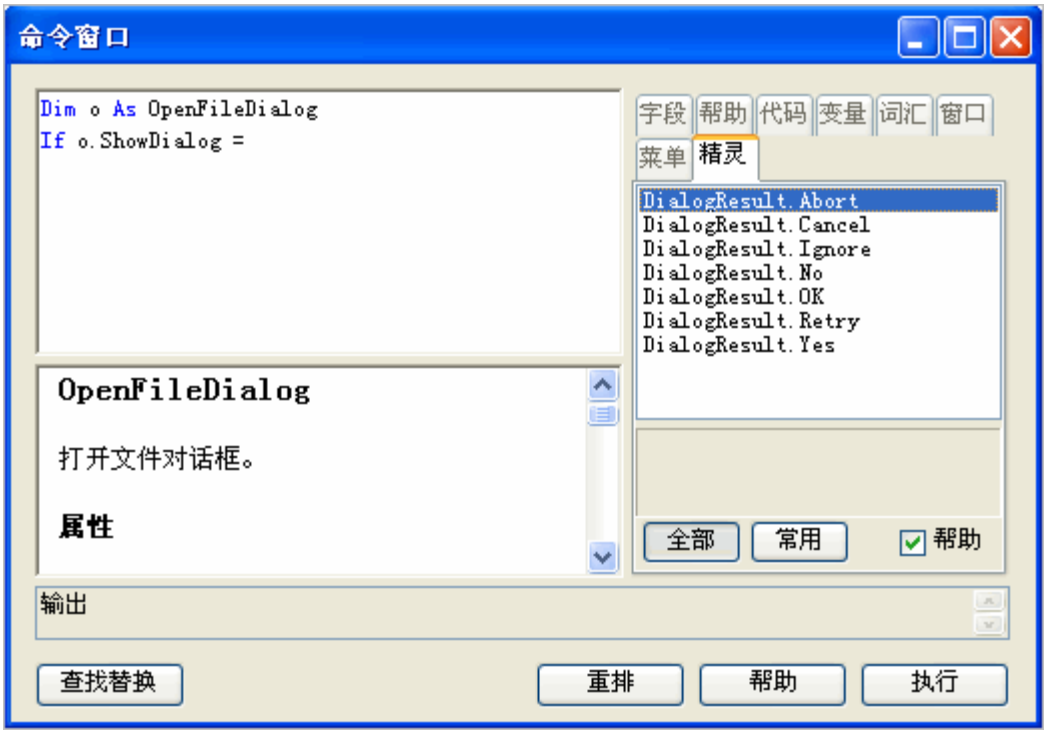
按 Tab，将自动变为：

If o.ShowDialog

接着输入等于号：

If o.ShowDialog =

按 Tab,将自动列出 DialogResult 的全部成员供选择：



按上下箭头键，选择要输入的枚举值，按回车键输入选择的枚举值。

如果输入的是:
if o.ShowDialog = o

然后按 Tab,不仅会自动列出 DialogResult 的全部成员供选择，还会自动定位到 DialogResult.OK，只需再按一次 Tab 或回车键，即可自动变为：
if o.ShowDialog = DialogResult.OK

我们再看看最常用的 MessageBox 是如何高效输入的。
在命令窗口输入：
Me

按 Tab， 即可自动替换为：

MessageBox

然后输入圆点符号，接着按 Tab，即可自动输入 Show，这是因为 MessageBox 只有一个成员 Show：

MessageBox.Show

按常规输入提示内容：

MessageBox.Show("大家好，我是 FoxTable！","提示",

然后按 Tab，即可自动列出枚举 MessageboxButtons 的全部成员供选择，假定选择的是 MessageboxButtons.YesNo，然后输入逗号，现在是：

MessageBox.Show("大家好，我是 FoxTable！","提示",MessageBoxButtons.YesNo,)

接着按 Tab, 将自动列出枚举 MessageBoxIcon 的全部成员供选择，假定选择的是 MessageBoxIcon.Question，然后输入括号和等于号，现在是：

MessageBox.Show("大家好，我是 FoxTable！","提示",MessageBoxButtons.YesNo, MessageBoxIcon.Question) =

按 Tab，会自动列出 DialogResult 的全部成员供选择。
整个过程我们不需要精确记忆任何东西，只需要记得大概的语法即可。
本节用对话框作为例子，实际上大部分枚举，都是可以这样输入的。

4.3.11 函数参考

4.3.11.1 数据转换函数

在程序开发的过程中，我们经常需要将某一类型的数据转换为其它数据类型后使用。
为此 Visual Basic 提供了以下转换函数：

- CStr
  转换为字符串(String)。

- CDate
  转换为日期(Date)。

- CDbl
  转换为双精度小数(Double)。

- CInt
  转换为整数(Integer)。

- CLng
  转换为长整数(Long)。

- CSng
  转换为单精度小数(Single)。

- CDec
  转换为高精度小数(Decimal)。

- Cbyte
  转换为微整数(Byte)。

- CShort
  换为短整数(Short)。

示例

```
Dim s As String = "123.12"
Output.Show(Cdate("1999-12-31"))
Output.Show(CInt(s))
Output.Show(CDbl(s) + 100)
```

请参考：
带判断的转换
最直接的转换

4.3.11.2 字符转换函数

Foxtable 提供了很多字符转换函数，例如 CUMoney 可以将数字转换为中文货币格式：
例如：

```
Dim v As String
v = CUMoney(12345.12)
Output.Show(v)
```

输出为：壹万贰千叁佰肆拾伍圆壹角贰分
下表列出了所有的字符转换函数。

函数名	说明
ReadBirthDay	从身份证号码中读取出生日期
ReadSex	从身份证号码中读取性别
ValidPIN	用于校验身份证号码是否正确
CLDate	将日期转换为小写中文格式
CUDate	将日期转换为大写中文格式
CCDate	将日期转换为大写中文格式，符合支票格式
CUMoney	将数值转换为中文货币格式

CUNumber	将数值转换为中文大写格式
CUNum	将阿拉伯数字转换为中文大写格式
CLNum	将阿拉伯数字转换为中文小写格式
CCNumber	将阿拉伯数字转换为中文格式，符合支票格式，只能处理整数
EUMoney	将数值转换为英文货币格式
EUNumber	将整数转换为英文大写
GetPinYin	将指定汉字转换为拼音格式，此函数已经停用，请参考：汉字与拼音
GetPY	将指定汉字转转为拼音格式，此函数已经停用，请参考：汉字与拼音
GetDigit	从一个数值中返回指定位的数字，可用于支票套打
StrToWide	将字符串转换为全角
StrToNarrow	将字符串转换为半角
StrToSimplified	将字符串转换为中文简体
StrToTraditional	将字符串转换为中文繁体
StrReverse	将字符串的字符排列顺序颠倒
StrToTitleCase	将字符串转换为词首字母大写

4.3.11.3 汉字与拼音

Foxtable 提供了一个名为“PY”的类，用于获取汉字的拼音。

PY 类提供的函数有：

GetPY 函数

用于获取汉字的拼音，语法：

GetPY(Input)

GetPY(Input, TitleCase)

GetPY(Input, TitleCase, Mode)

Input:	字符型，汉字字符串。
TitleCase	逻辑型，是否返回首字母大写的拼音
Mode	整数型，用于设置返回拼音的格式，以汉字"我"为例，不同 Mode 值返回的拼音分别如下: 0：wo， 无声调 1：wo3，带数字声调 2：wǒ， 带声调 3：w， 仅返回拼音的首字母

示例

在命令窗口执行下面的代码：

```
Output.Show(PY.GetPY("我是狐表用户"))
```

```
Output.Show(PY.GetPY("我是狐表用户", True))
Output.Show(PY.GetPY("我是狐表用户", True, 1))
Output.Show(PY.GetPY("我是狐表用户", True, 2))
Output.Show(PY.GetPY("我是狐表用户", False, 2))
Output.Show(PY.GetPY("我是狐表用户", True, 3))
Output.Show(PY.GetPY("我是狐表用户", False, 3))
```

得到的结果为：

wo shi hu biao yong hu

Wo Shi Hu Biao Yong Hu

Wo3 Shi4 Hu2 Biao3 Yong4 Hu4

Wǒ Shì Hú Biǎo Yòng Hù

wǒ shì hú biǎo yòng hù

WSHBYH

wshbyh

GetNamePY 函数

用于获取汉字姓名的拼音，语法：

GetNamePY(Input)

GetNamePY(Input, TitleCase)

GetNamePY(Input, TitleCase, Mode)

Input:	字符型，姓名字符串。
TitleCase	逻辑型，是否返回首字母大写的拼音
Mode	整数型，用于设置返回拼音的格式，以汉字"我"为例，不同 Mode 值返回的拼音分别如下: 0：wo， 无声调 1：wo3，带数字声调 2：wǒ， 带声调 3：w， 仅返回拼音的首字母

GetNamePY和GetPY函数的用法完全一样，不同的是，GetNamePY会将第一个字符作为姓氏，以尽可能返回正确的拼音。

示例

在命令窗口执行下面的代码：

```
Output.Show(PY.GetPY("覃波"))
Output.Show(PY.GetPY("仇和"))
Output.Show(PY.GetNamePY("覃波"))
Output.Show(PY.GetNamePY("仇和"))
```

得到的结果为：

tan bo

chou he

qin bo

qiu he

可以看到GetNamePY函数得到了准确的姓名拼音，而GetPY得到的结果是错误的，因为其将"覃"和"仇"作为普通字符处理了，而不是作为姓氏。

IsHanzi 函数

用于判断给定的字符是否是汉字，例如：

```
Output.Show(PY.IsHanzi("格"))  '结果为True
Output.Show(PY.IsHanzi("A"))  '结果为False
```

拼音数据库

由于汉字有大量的多音字，所以不管我们如何设计，获取的拼音结果都很难满足所有用户的要求。为解决此难题，Foxtable 将拼音数据库公开给了用户，这样用户能根据自己的需要修改任意汉字的拼音和声调。

在 Foxtable 的安装目录下有个名为“pydb.mdb”的文件，这是一个 access 格式的数据库，里头包含了 2 万多个汉字的拼音，以及 500 多个姓氏的拼音，大家可以根据需要随意修改。

重要提示：修改之后拼音数据库之后，一定要记得备份，避免卸载或重装 Foxtable 的时候被无意删除或覆盖。

4.3.11.4 数学函数

数学函数、三角函数都是用 Math 类的共享方法实现的，这些函数的调用格式为：

Math.函数名

例如要计算某个数值的绝对值，代码是：

```
Dim dbl As Double
dbl = Math.Abs(-1.23)
```

下面分类列出了 Math 类中所有的函数，对于大多数用户来说，掌握数学函数即可，三角函数则可

以忽略。

数学函数

函数名	说明
Abs	返回指定数字的绝对值
Round	将值舍入到最接近的整数或指定的小数位数
Ceiling	返回大于或等于指定数字的最小整数
Floor	返回小于或等于指定数字的最大整数
Max	返回两个指定数字中较大的一个
Min	返回两个指定数字中较小的一个
Sqrt	返回指定数字的平方根
Exp	返回 e 的指定次幂
Log	返回指定数字的对数。
Log10	返回指定数字以 10 为底的对数

三角函数

函数名	说明
Acos	返回余弦值为指定数字的角度
Asin	返回正弦值为指定数字的角度
Atan	返回正切值为指定数字的角度
Atan2	返回正切值为两个指定数字的商的角度
Cos	返回指定角度的余弦值
Cosh	返回指定角度的双曲余弦值
Sin	返回指定角度的正弦值
Sinh	返回指定角度的双曲正弦值
Tan	返回指定角度的正切值
Tanh	返回指定角度的双曲正切值

共享属性

名称	说明
E	表示自然对数的底，它由常数 **e** 指定
PI	表示圆的周长与其直径的比值，它通过常数 π 指定

4.3.11.5 财务函数

函数名	说明

DDB	返回用双倍余额递减法或其他指定某些方法，计算的特定周期内资产的折旧
FV	返回基于等额分期付款和固定利率的未来年金值
IPmt	返回在定期偿还固定款项和利率不变的条件下，年金在给定期次内所支付的利息
IRR	返回一系列定期现金流（支出和收入）的内部收益率
MIRR	返回一系列的周期性现金流（支出或收入）的修正内部利率
NPer	返回根据定期定额支付和固定利率而定的年金的期数
NPV	基于一系列定期现金流（支出和收入）和贴现率，返回投资的净现值
Pmt	指定根据定期定额付款和固定利率而定的年金的付款金额
PPmt	返回根据定期定额支付和固定利率而定的年金在指定期间内的本金偿付额
PV	返回根据要在将来支付的定期、定额付款和固定利率指定年金的现值
Rate	返回每一期的年金利率
SLN	指定在一段时间内资产的直线折旧
SYD	返回在指定期间内资产按年限总和折旧法计算的折旧
InTax	计算个人所得税

4.3.11.6 加密与解密

函数名	说明
EncryptText	将字符串加密成另一个字符串
DecryptText	将加密后的字符串解密
MD5Encrypt	用 MD5 算法加密字符串，加密后的字符串不能被解密
CRCCheckFile	用于校验文件，判断文件是否已经被他人篡改。
CRCCheckString	用于校验字符串，判断自己的重要资料（例如代码）是否已经被他人篡改。
EncryptFile	用于加密文件。
DecryptFile	用于解密文件。

4.3.11.7 格式化数据

Format 用于格式化数据。
在打印或者显示的时候，用户可能有一些特殊的格式要求，例如固定小数位数，用中文的年月日代替日期中的分割横线等等，Format 函数正是针对这种需要而提供的。
语法：
Format(Value, Expression)
Value: 数据

Expression: 格式表达式
例如：

```
Output.Show(Format(1,"000"))
Output.Show(Format(1,"""A""000"))
Output.Show(Format(1.2,"0.00"))
Output.Show(Format(#12/31/2008#,"yyyy 年 MM 月 dd 日"))
```

上述代码的输出为：

```
001
A001
1.20
2008 年 12 月 31 日
```

请参考:
格式化数字
格式化日期和时间

4.3.11.8 IIF 函数

IIF 函数根据一个表达式的计算结果，返回两个值中的一个。
语法：

```
IIF(expression, TrueValue, FalseValue)
```

expression: 表达式
TrueValue： 表达式成立的时候，返回此值
FalseValue：表达式不成立的时候，返回此值
示例

```
Dim Val As Integer = 1000
Dim m As String = IIF(Val > 500, "大", "小")
```

上面的语句完全等效于：

```
Dim Val As Integer = 1000
Dim m As String
If Val > 500 Then
    m = "大"
Else
    m = "小"
End If
```

可以看出，对于一些简单的分支判断，使用 IIF 函数能够让代码显得更为简洁。

4.3.11.9 进制转换函数

Foxtable 提供了以下函数，用于数字的进制转换。

- DecToHex
  将 10 进制数转换为 16 进制

- DecToOct
  将 10 进制数转换为 8 进制

- DecToBin
  将 10 进制数转换为 2 进制

- HexToDec
  将 16 进制数转换为 10 进制

- OctToDec
  将 8 进制数转换为 10 进制

- BinToDec
  将 2 进制数转换为 10 进制

例如

```
Output.Show(DecToHex(1000)) '3E8
Output.Show(DecToBin(1000)) '1111101000
Output.Show(HexToDec("7B")) '123
```

4.3.12 数据转换

4.3.12.1 数据转换函数

在程序开发的过程中，我们经常需要将某一类型的数据转换为其它数据类型后使用。
为此 Visual Basic 提供了以下转换函数：

- CStr
  转换为字符串(String)。

- CDate
  转换为日期(Date)。

- CDbl
  转换为双精度小数(Double)。

- CInt
  转换为整数(Integer)。

- CLng
  转换为长整数(Long)。

- CSng
  转换为单精度小数(Single)。

- CDec
  转换为高精度小数(Decimal)。

- Cbyte
  转换为微整数(Byte)。

- CShort
  换为短整数(Short)。

示例

```
Dim s As String = "123.12"
Output.Show(Cdate("1999-12-31"))
Output.Show(CInt(s))
Output.Show(CDbl(s) + 100)
```

请参考：
带判断的转换
最直接的转换

4.3.12.2 带判断的转换

显然任何类型的数据，都可以转换为字符型。
但是对于其它类型的数据，却有转换失败的可能，例如：

```
Dim s As String = "abc"
Dim v As Long = 123
Return CLng(s) + v
```

如果在命令窗口执行上面的代码，会出现错误提示：字符串"abc"无法转换为长整数。
为了解决这个问题，我们可以采用另一种转换方法。

除了 String(字符)类型外，所有的基本数据类型，都有一个 TryParse 方法，用于将其它类型的数据转换为本类型的数据。

语法

Type.TryParse(Value,Variant)

说明

Type： 目标数据类型，例如 Date、Integer、Long、Double 等等。
Value： 要进行转换的数据。
Variant: 用于存放转换结果的变量，变量的类型必须和 Type 指定的类型一致。

如果转换成功，则将转换结果存储在变量 Variant 中，并返回 True，否则返回 False。

例如：

```
Dim s As string = "123.1"
Dim d As Double
Double.TryParse(s, d) '将变量 s 的内容转换为数值，并存放在变量 d 中
Output.Show(d + 100) '输出结果是 223.1
```

再例如：

```
Dim d As Date '变量 d 用于存储转换结果
If Date.TryParse("1999/12/31", d) Then '如果转换成功
    Output.Show(d) '输出转换结果
Else
    Output.Show("无效日期格式") '给出错误提示
End If
```

4.3.12.3 最直接的转换

前面已经讲到，使用转换函数转换数据类型，在转换失败的情况下，会出现错误提示。
使用 TryParse 方法，可以避免错误提示，但是代码略显繁琐。
最常见的转换应该是将字符转换为数字，对于这种转换，有更直接的方法。
Visual Basic 提供了一个 Val 函数，用于将字符转换为数值，即使转换失败，也不会报错，而是返回 0。
例如：

```
Dim s1 As String = "abc"
Dim s2 As String = "123.12"
Output.Show(Val(s1) + 100) '等于 100
Output.Show(Val(s2) + 100) '等于 223.12
```

虽然变量 s1 是无法转换为数值的，但是在命令窗口执行上面的代码，却不会有任何错误提示，因为 Val 函数将 s1 被转换为 0 了。
此外 Visual Basic 还提供了一个 IsNumeric 函数，用于判断某个给定的字符串是否能够转换为数值，毕竟有时 会希望在转换前能够判断一下的。
例如：
假定某个数组中，一些元素是数字，一些元素是字符，我们需要统计有多少个元素是数字，以及这些数字元素的和：

```
Dim s(2) As String
Dim v As Double
Dim n As Integer
```

```
s(0)= "36.2"
s(1) = "fox"
s(2) = "16.4"
For i As Integer = 0 To 2
    If IsNumeric(s(i)) Then
        v = v + Val(s(i))
        n = n + 1
    End If
Next
Output.Show("数值个数为:" & n) '2
Output.Show("数值的和为:" & v) '52.6
```

4.3.13 杂项

4.3.13.1 系统变量

系统变量是无需定义的，它们是由系统提供、能够直接使用的变量。
最常用的系统变量为：

- ProjectFile
  字符型，返回已经打开的项目文件名

- ProjectPath
  字符型，返回项目文件所在目录

- ApplicationPath
  返回 Foxtable 主程序所在目录，通常就是 Foxtable 的安装目录

- ApplicationTitle
  返回或设置 Foxtable 主窗口的标题。

- MouseButton
  该变量是 MouseButtons 型枚举，用于返回当前鼠标按键的状态

- ModifierKey
  Keys 型枚举，用于判断 Control、Shift、Alt 三个按键的状态

- VisualStyle
  用于返回或设置界面风格

- DefaultPrinter
  返回默认打印机的名称。

- InstalledPrinters
  返回一个字符串集合，包括所有已经安装的打印机的名称。

- LastInputTickCount
  返回一个整数，单位为毫秒，表示用户已经有多长时间没有操作计算机。

示例一

打开一个项目文件，然后在命令窗口执行下面的代码，即可显示相关的路径信息：

```
Output.Show("项目目录:" & ProjectPath)
Output.Show("项目文件:" & ProjectFile)
Output.Show("Foxtable 目录:" & ApplicationPath)
```

示例二

在命令窗口执行下面的代码，可以列出所有已经安装的打印机名称：

```
For Each p As String in InstalledPrinters
    Output.Show(p)
Next
```

4.3.13.2 随机数与字符

Rand 类用于生成随机数和随机字符串，它包括三个共享方法。

Next

用于生成随机整数。

语法

```
Next()
Next(MaxValue)
Next(MinValue, MaxValue)
```

MinValue: 允许生成的最小随机数

MaxValue: 允许生成的最大随机数

例如：

```
Dim Val1 As Integer = Rand.Next() '生成一个随机整数
Dim Val2 As Integer = Rand.Next(100) '生成 0 到 100 之间的随机整数
Dim Val3 As Integer = Rand.Next(200,300) '生成 200 到 300 之间的随机整数
```

NextDouble

产生一个 0 到 1 之间的随机小数。

例如你可以在命令窗口执行下面的代码，得到 10 个随机小数：

```
For i As integer = 0 To 9
    Output.Show(Rand.NextDouble)
Next
```

NextString

用于生成指定长度的随机字符串(最长为 32 个字符)。

语法

```
NextString(Length)
```

Length: 要生成的随机字符串长度，最大为 32
例如：

```
Dim Val As String
Val = Rand.NextString(12) '生成长度为 12 的随机字符串
Output.show(Val)
```

4.3.13.3 播放声音

Audio 类用于播放声音文件，目前只能播放 wav 格式的声音。
Audio 包括以下共享方法:

Play

播放声音文件。
语法：

```
Play(File)
Play(File, Mode)
```

参数	说明
File	要播放的声音文件。
Mode	播放模式，AudioPlayMode 型枚举，包括以下可选值： Background: 在后台播放声音,代码继续执行。 BackgroundLoop: 在后台循环播放声音，直到调用 Stop 方法，代码继续执行。 WaitToComplete： 播放声音并等待播放完毕，然后再继续执行代码。

例如：

```
Audio.Play("C:\music\test.wav")
Audio.play("C:\music\test.wav", AudioPlayMode.WaitToComplete)
```

Stop

停止播放声音，例如：

```
Audio.Stop()
```

4.3.13.4 文件与目录

方法	说明
CopyDirectory	将目录中的内容复制到另一个目录

CreateDirectory	创建目录
DeleteDirectory	删除目录
DirectoryExists	判断指定的目录是否存在
RenameDirectory	重命名目录
MoveDirectory	将目录中的内容移到另一个目录，并删除原目录
FileExists	判断指定的文件是否存在
CopyFile	将文件复制到新位置
DeleteFile	删除文件
MoveFile	将文件移到一个新的位置
RenameFile	重命名文件
GetFiles	返回一个字符集合，该集合包括指定目录下的所有文件名
GetDirectories	返回一个字符集合，该集合包括指定目录下的所有子目录名。
GetParentPath	返回父目录
GetName	返回不含路径的文件名。
ReadAllText	读取文本文件的内容
WriteAllText	向文本文件中写入内容

4.3.13.5 获取文件信息

Foxtable提供了一个FileInfo类，用于获取文件信息。

FileInfo的属性有：
CreationTime： 返回或设置文件的创建时间
LastAccessTime： 返回或设置文件最后一次访问的时间
LastWriteTime： 返回或设置文件最后一次修改的时间
Length： 返回文件大小
Name： 返回文件名称
Extension： 返回文件扩展名
Path： 返回文件路径
ReadOnly： 返回或设置文件是否只读
Hidden： 返回或设置文件是否隐藏
示例一

如果要获取某个文件的完整信息，可以参考下面的代码：

```
Dim ifo As new FileInfo("c:\data\123.xls")
Output.Show("文件创建时间:" & ifo.CreationTime)
```

```
Output.Show("上次修改时间:" & ifo.LastWriteTime)
Output.Show("上次访问时间:" & ifo.LastAccessTime)
Output.Show("是否只读:" & ifo.ReadOnly)
Output.Show("是否隐藏:" & ifo.Hidden)
output.Show("文件路径:" & ifo.Path)
Output.Show("文件大小:" & ifo.Length)
Output.Show("文件名称:" & ifo.Name)
Output.Show("扩展名:" & ifo.Extension)
```

示例二

隐藏某个文件，并将其设置为只读：

```
Dim ifo As New FileInfo("c:\data\123.xls")
Ifo.ReadOnly = True
Ifo.Hidden = True
```

4.3.13.6 运行外部程序

Process 用于执行外部程序。

属性

- File
  指定要执行的文件。

- WaitForClose
  逻辑型，设为 True，直到关闭外部程序，才会执行接下来的代码。

- Arguments
  命令行参数。

- Verb
  字符型，要执行的动作。

- Verbs
  返回一个字符数组，包括即将执行的文件支持哪些动作。

方法

- Start
  开始执行。

- Close
  关闭。

- WaitForInputIdle
  等待程序启动完成并进入空闲状态。

- WaitForExit
  等待程序运行结束并退出。

示例一

打开文件"C:\test.xls"，代码非常简单：

```
Dim Proc As New Process '定义一个新的 Process
Proc.File = "C:\test.xls" '指定要打开的文件
Proc.Start()
```

示例二

利用属性 Verb 可以设置打开的动作，不同的外部程序，支持的动作会不同，对于 Excel 文件，支持三个 Verb 动作：

Open：打开，这也是默认的动作。

OpenAsReadOnly：以只读方式打开

Print：打印，完成打印后自动关闭

例如：

```
Dim Proc As New Process
Proc.File = "C:\test.xls"
Proc.Verb = "Print" '指定动作
Proc.Start()
```

上面的代码将打开文件 c:\test.xls，然后执行打印命令，打印结束后自动关闭 Excel。

如果要获得某一文件类型支持的全部 Verb 动作，可以参考下面的代码：

```
Dim Proc As New Process
Proc.File = "C:\test.txt"
For Each Verb As String in Proc.Verbs
    Output.show(Verb)
Next
```

在命令窗口执行上面的代码，可以列出文本文件支持的 verb 动作。

示例三

下面的代码，列出了扩展名为 Doc 的文件所支持的动作。

```
Dim Proc As New Process
Proc.File = "C:\test.doc"
For each Verb As String In Proc.Verbs
    Output.Show(verb)
Next
```

示例四

下面的代码，调用记事本打开文件“c:\table.txt”。

```
Dim Proc As New Process
Proc.File = "Notepad.exe" '指定要执行的文件
Proc.Arguments = "c:\table.txt" '指定命令行参数
```

```
Proc.Start()
```

示例五

通过 WaitForInputIdle，可以等待程序启动结束并进入空闲状态，然后向程序发送按键信息。

例如：

```
Dim Proc As New Process
Proc.File = "C:\table.txt"
Proc.Start
Proc.WaitForInputIdle() '等程序自动结束并进入空闲状态
Sendkeys.Send("I Like Foxtable") '发送按键信息
```

参考: SendKeys

示例六

Process 不仅可以打开和执行文件，还可以用来打开网页：

```
Dim Proc As New Process '定义一个新的 Process
Proc.File = "http://bbs.foxtable.com" '指定要打开的网页地址
Proc.Start()
```

甚至可以用来发送邮件：

```
Dim Proc As New Process '定义一个新的 Process
Proc.File = "mailto:zjtdr@21cn.net" '邮件地址前要加上"mailto:"
Proc.Start()
```

4.3.13.7 字符编码

```
Asc("a") '等于 97
Asc("你") '等于-15133
```

通过函数 Chr 可以获得指定编码的字符。

例如：

```
Chr(97) '等于"a"
Chr(-15133) '等于"你"
```

通过 Chr 函数，可以让我们在字符串中加入一些不可见的控制字符，例如回车符、换行符、Tab 符等等。

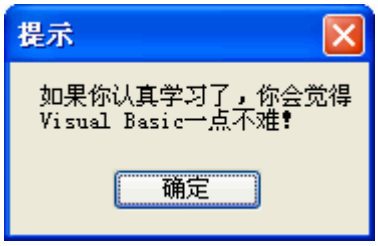
下表是常用的、不可见的字符编码表：

字符	编码
回车符	13
换行符	10
Tab 符	9

例如在 MessageBox 显示信息的时候，希望分行显示：

```
MessageBox.Show("如果你认真学习了，你会觉得" & Chr(13) & Chr(10) & "Visual Basic 一点不难！", "提示")
```

上面的代码通过 Chr 函数，在显示信息的中间插入回车符和换行符，从而实现了信息换行显示：



由于换行符使用的频率很高，所以 Visual Basic 提供了一个常量 vbcrlf，其值等于：Chr(13) & Chr(10)
所以上面的代码也可以改写为：

```
MessageBox.Show("如果你认真学习了,你会觉得" & vbcrlf & "Visual Basic 一点不难!", "提示")
```

4.3.13.8 单字符判断

类型 Char 型提供了很多共享方法，用于判断某个字符类别，例如判断某个字符是否是数字：

```
If Char.IsDigit("1") Then
    Messagebox.Show("这是一个数字！ ")
End If
```

部分判断函数：

函数	说明
IsControl	判断指定的字符是否属于控制字符类别。
IsDigit	判断指定的字符是否属于十进制数字类别。
IsLetter	判断指定的字符是否属于字母类别。
IsLetterOrDigit	判断指定的字符是属于字母类别还是属于十进制数字类别。
IsLower	判断指定的字符是否属于小写字母类别。
IsNumber	判断指定的字符是否属于数字类别。
IsPunctuation	判断指定的字符是否属于标点符号类别。
IsSeparator	判断指定的字符是否属于分隔符类别。
IsSymbol	判断指定的字符是否属于符号字符类别。
IsUpper	判断指定的字符是否属于大写字母类别。
IsWhiteSpace	判断指定的字符是否属于空白类别。

4.3.13.9 关于字体

普通用户可以跳过本节内容。
字体的类型为 Font，定义一个字体的语法为：

```
Dim 变量名 As New Font(Name, Size, Style)
```

Name: 字体名称。
Size: 字体大小，单位为磅。
Style: 可选参数，字体样式，FontStyle 型枚举。

下面是 FontStyle 型枚举的可选值：
Bold：　加粗。
Italic：　倾斜。
Regular：　普通。
Strikeout：　中间有直线通过。
Underline：　带下划线。
例如定义一个黑体、大小为 12 磅、带下划线的字体：

```
Dim fnt As New Font("黑体",12,FontStyle.Underline)
```

样式可以用 Or 运算符叠加，例如定义一个宋体、大小为 12 磅、加粗且带下划线的字体：

```
Dim fnt As New Font("宋体",12,FontStyle.Bold or FontStyle.Underline)
```

字体包括下面三个属性：
Name: 返回字体名称。
Size: 返回字体大小，单位为磅。
Style: 返回字体样式，FontStyle 型枚举。
例如下面的代码可以显示当前表的字体属性：

```
With CurrentTable.Font
    Output.Show(.Name)
    Output.Show(.Size)
    Output.Show(.Style)
End With
```

CurrentTable 表示当前表，表有一个属性 Font，用于返回或设置表的字体，这个以后我们会讲述的。
字体的属性都是只读的，不能被修改，例如要将当前表的字体大小改为 14 磅，正确的代码为：

```
Dim fnt As New Font(CurrentTable.Font.Name, 14, CurrentTable.Font.Style)
CurrentTable.Font = fnt
```

如果直接修改 Font 的 Size 属性，编译过程会出现错误提示：

```
CurrentTable.Font.Size = 14 '这行代码是不能通过编译的。
```

4.3.13.10 关于颜色

普通用户可以忽略本节内容。
颜色的类型是 Color，定义一个颜色变量的语法为：

```
Dim 变量名 As Color
```

Color 本身是一个枚举，包括很多预先定义好的颜色，例如 Color.Red 表示红色，color.Pink 表示粉红色。
要查询更多定义好的颜色，请参考附录中英文颜色对照表。
例如下面的代码定义了一个颜色变量，并将其设为红色：

```
Dim clr As Color = Color.Red
```

下面的代码，将当前表的字体颜色设为粉红色：

```
CurrentTable.ForeColor = Color.Pink
```

颜色由四个分量组成，分别是 alpha、红、绿、蓝，所以颜色有四个属性，用于返回这个分量的值：
A：返回颜色的 alpha 分量
B: 返回颜色的蓝色分量
G: 返回颜色的绿色分量
R：返回颜色的红色分量

每个分量的范围都是 0 到 255 之间。
例如下面的代码，列出了粉红色的各组成分量值：

```
Dim clr As Color = Color.Pink
Output.Show("A:" & clr.A)
Output.Show("B:" & clr.B)
Output.Show("G:" & clr.G)
Output.Show("R:" & clr.R)
```

通过共享方法 FromARGB，可以根据指定的分量，合成一个颜色，语法为：

```
FromARGB(A,R,G,B)
```

A：alpha 分量
R：红色分量
G: 绿色分量
B: 蓝色分量
例如：

```
Dim clr As Color = Color.FromARGB(255,255,192,203)
```

FromARGB 还可以根据一个整数来生成颜色：

```
Dim clr As Color = Color.FromARGB(-16181)
```

通过共享方法 ToARGB，可以将颜色转换为一个整数，例如：

```
Dim Val AS Integer = Color.Pink.ToARGB
Output.Show(Val)
```

4.3.13.11 1 不等于 1?

在命令窗口输入下面的代码：

```
Dim d As Double
d = 0.7 + 0.2 + 0.1
Output.Show(d)
Output.Show(d = 1)
```

执行之后，你会惊讶地发现，d 和 1 并不相等。
这是怎么回事，0.7 + 0.2 + 0.1 明明就是 1，而且第三条语句也证明了 d 的结果就是 1 啊。
这是因为 d 是 Double 类型，是浮点数，浮点数的 1，可能并不会精确地等于 1，而只是近似等于 1。
毋需过分担心这种误差，这种误差只有 10 的负 16 次方左右，对日常计算的结果不会有任何影响，但是会影响数值比较的结果
如果需要精确比较两个数值，该如何处理呢，可以将上述代码改为：

```
Dim d As Double
d = 0.7 + 0.2 + 0.1
Output.Show(d)
Output.Show(Math.Round(d, 6) = 1)
```

上面的代码，用 Round 函数将 d 的小数位数四舍五入到 6 位，然后再和 1 比较。
或者用 Decimal 类型代替 Double 类型：

```
Dim d As Decimal
d = 0.7 + 0.2 + 0.1
Output.Show(d)
Output.Show(d = 1)
```

Decimal 是高精度小数，不会有 Double 的那种误差。

4.3.13.12 空值(Nothing)

变量可能不包括任何内容，也就是空值。
在 Visual Basic 中，空值用 Nothing 表示。
有两个特殊的运算符，用于判断变量的值是不是 Nothing，分别是 Is 和 IsNot，前者的意思是“是”，后者的意思是“不是”。
语法为：
变量名 Is Nothing
如果变量值为 Nothing，则返回 True，否则返回 False。
变量名 IsNot Nothing
如果变量值不为 Nothing，则返回 True，否则返回 False。
例如：

```
Dim str1 As String
Dim Str2 AS String = "Foxtable"
If str1 Is Nothing Then
    Output.Show("Str1 是空值！")
End If
If str2 IsNot Nothing Then
    Output.Show("Str2 不是空值！")
End If
```

需要注意的是，数值、日期、逻辑三种类型的变量，是永远不会为空的，其它任何类型的变量，如果没有明确赋值，其初始值都是 Nothing。
数值型的初始值为 0，逻辑型变量的初始值为 False，日期型的初始值为#1/1/1 0:00:00#，如果你将 Nothing 赋给这些变量，也只是使得这些变量回到初始值而已。
我们可以将下面的代码复制到命令窗口执行，用以验证这个特性：

```
Dim V1 As Date = Nothing
Dim V2 As Double = Nothing
Dim V3 As Boolean = Nothing
Output.Show("V1 = " & V1)
Output.Show("V2 = " & V2)
Output.Show("V3 = " & V3)
```

4.4 Foxtable 编程

4.4.1 基本类型

4.4.1.1 基本概念(必读)

下面的概念非常重要，务必要掌握。

DataTable 和 Table

假定在一个简化的订单管理系统中，包括产品、客户、订单三个表：

	产品编号	产品名称	单价
1	1	HCD238(3)P/TSD	98
2	2	HCD238(32)P/TSI	65
3	3	HW238(2)P/TD	200
4	4	HA238(8)P/T	25

产品表结构

	客户编号	客户名称	联系人	电话
1	1	联想电脑公司	周瑜	010-88667722
2	2	TCL商贸公司	关羽	020-22669900
3	3	神州数码公司	刘备	021-33667733
4	4	盛大网络公司	张飞	023-55443123

客户表结构

	日期	产品编号	客户编号	数量	单价	金额
1	2008-06-06	2	1	6	65	390
2	2008-06-06	1	2	3	98	294
3	2008-06-07	3	3	7	200	1400
4	2008-06-08	4	4	2	25	50

订单表结构

Foxtable 中有一个集合 DataTables，通过该集合可以获得指定名称的表，例如：

```
DataTables("产品")  '产品表
DataTables("客户")  '客户表
DataTables("订单")  '订单表
```

实际上你是看不到 DataTable 的，你看到的是 Table，也就是说，上面的三个图，实际上就是三个 Table。

那么什么是 Table 呢？你可以这样理解，DataTable 就像一个仓库，所有从数据文件中加载的数据，都存放在这里，而且这个仓库是不直接面对用户的；而 Table 就像一个展厅， 是负责和用户打交道的，这个展厅(Table)从仓库(DataTable)中取得数据，并呈现给用户。

同样，Foxtable 中也有一个集合 Tables，通过该集合可以获得指定名称的 Table，在没有建立关联的情况下，我们这个简化的订单管理系统包括三个 Table，分别是：

```
Tables("产品")
Tables("客户")
Tables("订单")
```

在初始的情况下，Table 会从 DataTable 中提取所有数据呈现给用户，如果你设置了筛选条件，Table 就会从 DataTable 提取符合筛选条件的数据，然后呈现给用户。

例如订单表有 1000 行数据，那么 DataTables("订单")从头到尾，始终都会有 1000 行数据，不受筛

选的影响。而 Tables("订单")却不一样，如果没有进行筛选，那么 Tables("订单")也会包括所有数据，也就是 1000 行数据全部可见。如果现在从中筛选出产品为 PD01 的订单，且 PD01 的订单是 200 个，那么经过筛选后，Tables("订单")会包括 200 行数据，也就是你看到的那 200 个产品为 PD01 的订单。

所以关于 DataTable 和 Table 的关系，较为完整的描述是：

- 从数据文件加载到 Foxtable 中的数据，是存放在 DataTable 中的，它就像一个仓库，不直接面对用户；而 Table 就像一个展厅，是负责和用户打交道的，它根据指令从仓库(DataTable)中提取符合条件的数据，然后呈现给用户；通过菜单进行的日常数据管理工作，例如增加行、删除行、排序、筛选、汇总等等，都是在 Table 中进行的；除了编程，用户没有办法直接对 DataTable 进行操作，因为它是不可见的。

- 在任何时候，DataTable 都包括所有已经加载的行。

- Table 中的行，既可以排序，也可以筛选，所有 Table 中行的数量和顺序都是可变化的，我们平时所看到的、所操作的，都是 Table 中的行。

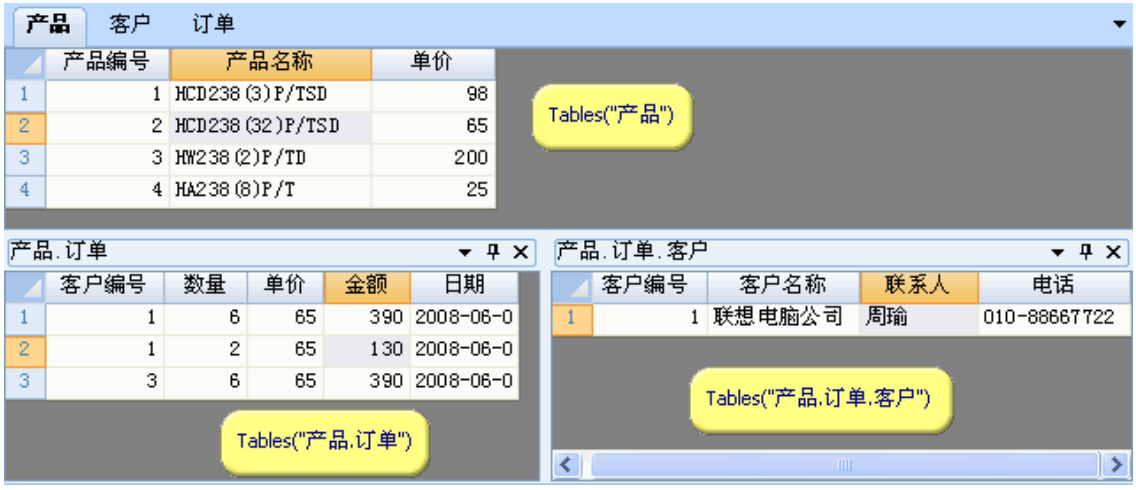
- 我们在 Table 中进行的任何操作，最终都会反映到 DataTable，例如我们通过菜单在 Table 中删除行，DataTable 也会删除一行。

关联表和 Table

假定在上面这个简化的订单管理系统中，我们建立如下两个关联：

父表	子表	关联列
产品	订单	产品编号
客户	订单	客户编号

因为关联的建立，DataTable 和 Table 不再是一一对应的关系，例如我选择产品表的时候，就会出现三个 Table，分别是：



三个 Table 的作用为：

Tables("产品")：主表，用于显示所有产品

Tables("产品.订单")：如果在 Tables("产品")中选定一个产品，这里会显示该产品的全部订单。

Tables("产品.订单.客户")：如果在 Tables("产品.订单")中选定一个订单，这里会显示该订单所属的

客户。
同样在选择客户表的时候，也会出现三个 Table:
Tables("客户")
Tables("客户.订单")
Tables("客户.订单.产品")
而选择订单表，出现的三个 Table 为：
Tables("订单")
Tables("订单.产品")
Tables("订单.客户")
由此我们可以看出，同一个 DataTable 可以有多个 Table，例如下面三个 Table:
Tables("产品")
Tables("订单.产品")
Tables("客户.订单.产品")
它们的数据全部来自于 DataTables("产品")。
由此可见，DataTable 的数量是固定的，而 Table 的数量会随着关联的增加而增加，同一个 DataTable 可以有多个 Table，每个 Table 都各取所需地从 DataTable 中提取数据呈现给用户。

4.4.1.2 DataTable

4.4.1.2.1 概述

通过 DataTables 集合，可以获得指定名称的 DataTable(表)。
例如：

```
Dim dt As DataTable
dt = DataTables("订单")
```

DataCol 表示 DataTable 中的列。
通过 DataCols 集合，可以获得指定名称的 DataCol(列)。
例如：

```
Dim dt As DataTable = DataTables("订单")
Dim dc As DataCol = dt.DataCols("日期")
```

DataRow 表示 DataTable 中的行。
通过 DataRows 集合，可以获得某一位置的 DataRow(行)。
例如:

```
Dim dr As DataRow
dr = DataTables("订单").DataRows(0)
```

行是从 0 开始编号的，所以 0 表示第一行，1 表示第二行。
通过列名称，我们可以得到或设置某一行指定列的内容。
例如：

```
Dim r As DataRow
Dim v As Double
r = DataTables("订单").DataRows(0)
r("数量") = 100
v = r("折扣")
```

由此可知，DataTable、DataRow、DataCol 构成了表、行、列这样一个完整的结构体系，接下来我们具体介绍这三个对象。

4.4.1.2.2 DataTable

4.4.1.2.2.1 常用属性

通过 DataTables 集合，可以获得指定名称的 DataTable(表)。

例如：

```
Dim dt As DataTable
dt = DataTables("订单")
```

如果要禁止编辑某个 DataTable，只需将其 AllowEdit 属性设为 False，例如：

```
DataTables("订单").AllowEdit = False
```

在命令窗口执行上述代码，你会发现订单表的左上角出现一个锁形标记，表示此表已经被锁定，不能在其中输入数据。

如果你要取消锁定表，只需将其 AllowEdit 属性重新设为 True：

```
DataTables("订单").AllowEdit = True
```

下表列出了 DataTable 的常用设置属性，它们的用法和 AllowEdit 属性完全一样：

属性名	说明
AllowEdit	是否允许修改表中数据
AllowAddNew	是否允许增加行
AllowClipBoard	是否允许复制粘贴数据
AllowInitialize	是否允许初始化此表
AutoAddNew	是否允许自动增加行，也就是在最后一行的最后一个单元格按回车键时，是否自动增加一行
AllowDelete	是否允许删除行
AllowLockRow	是否允许锁定行
AllowUnlockRow	是否允许取消锁定行

我们在编辑数据的时候，如果按回车键或 Tab 键，光标会向右移到下一单元格，通过下面的属性，你可以改变这种默认的光标移动方式：

属性名	说明
EnterKeyActionDown	按回车键是否向下移到另一单元格
TabKeyActionDown	按 Tab 键是否向下移到另一单元格

例如你希望按回车键向下移动光标，而不是向右移动，只需将 EnterKeyActionDown 属性设为 True 即可：

```
DataTables("订单").EnterKeyActionDown = True
```

最后三个常用的属性为：

属性名	说明
Name	返回 DataTable 的名称

Type	返回一个整数，表示 DataTable 的类型： 1 内部数据表 2 内部查询表 3 外部数据表 4 外部查询表 5 临时表
HasChanges	逻辑型，判断 DataTable 的数据是否已经被修改

例如执行下面的代码，将列出所有已经修改过的 DataTable：

```
Output.Show("已经修改的数据表:")
For Each dt As DataTable In DataTables
    If dt.Type = 1 Orelse dt.Type = 3 Then
        If dt.HasChanges Then
            Output.Show(dt.Name)
        End If
    End If
Next
```

4.4.1.2.2.2 常用方法

4.4.1.2.2.2.1 AddNew

在 DataTable 中增加一行或多行，并返回所增加的第一行。

AddNew()

AddNew(Count)

Count: 可选参数，要增加的行数，如果省略，则只增加一行。

例如：

```
Dim dr As DataRow
dr = DataTables("订单").AddNew()
dr("日期") = Date.Today '将新增行的日期设为当天日期。
```

其实我们很少直接向 DataTable 增加行，更多的时候，我们是向 Table 中增加行，原因以后会讲述。

4.4.1.2.2.2.2 Save

保存数据。

语法：

Save()

Save(Setting)

Setting: 可选参数，是否保存设置。

例如单单保存数据：

```
DataTables("订单").Save()
```

同时保存数据和设置：

```
DataTables("订单").Save(True)
```

保存设置比较耗时，会影响保存速度。

菜单中的保存命令是同时保存数据和设置的。
如果你学会了设计菜单或窗口，你可以自己设计一个保存按钮，将其代码设为:

```
For Each dt As DataTable In DataTables
    dt.Save()
Next
```

这样单击这个按钮就能保存所有表，但是不会保存设置；对于一个成熟的、已经交付使用的项目，有时是没有必要保存设置的。

4.4.1.2.2.2.3 GetValues

从指定列中，获取不重复的值，以集合的形式返回。
语法：
GetValues(ColumnName,Filter,Sort)
ColumnName：列名称，从此列中提取不重复的值。
Filter:　可选参数，指定一个条件表达式，只返回符合此条件的值；请参考表达式的运算符和函数 和 条件表达式
Sort:　可选参数，指定排序列，如果省略，则根据取值列排序，通常无需设置。

示例一

列出产品表中所有的产品名称：

```
Dim Products As List(Of String)
Products = DataTables("产品").GetValues("产品名称")
For Each Product As String In Products
    Output.Show(Product)
Next
```

示例二

返回的值默认按照取值列排序，我们可以另外指定排序列；例如按产品编号顺序，列出产品表中所有的产品名称：

```
Dim Products As List(Of String)
Products = DataTables("产品").GetValues("产品名称","","产品编号")
For Each Product As String In Products
    Output.Show(Product)
Next
```

示例三

给排序列加上 DESC 关键词，返回值可以降序排序，例如按最近一次订货的日期顺序，列出订单表中的客户名单：

```
Dim Customers As List(Of String)
Customers = DataTables("订单").GetValues("客户","","日期 Desc")
For Each Customer As String In Customers
    Output.Show(Customer)
Next
```

示例四

可以设置取值条件，列出产品表中单价大于 100 的产品：

```
Dim Products As List(Of String)
Products = DataTables("产品").GetValues("产品名称","单价 > 100")
For Each Product As String In Products
    Output.Show(Product)
Next
```

示例五

可以同时设置取值条件和排序列，例如按日期顺序，列出 2012 年 6 月 1 日后订购过 PD01 产品的客户名单：

```
Dim Customers As List(Of String)
Customers = DataTables("订单").GetValues("客户","[产品]= 'PD01' And [日期] > #6/1/2012#","日期")
For Each Customer As String In Customers
    Output.Show(Customer)
Next
```

示例六

下面这个例子，一般用户客户忽略。

可以同时从多列提取不重复的值，此时返回的不是字符的集合，而是字符数组的集合。

例如从客户列和产品列提取不重复的值：

```
'定义数组集合的时候，要在类型后加上括号，表示这是一个数组集合。
Dim Arys As List(Of String())
Arys = DataTables("订单").GetValues("产品|客户") '列名用符号|分割
'注意循环变量是字符型数组，所以类型是 String()，而不是 String
For Each Ary As String() In Arys
    Output.Show(Ary(0) & "|" & Ary(1))
Next
```

上面的代码是从客户和产品两列提取不重复的值，返回的不是一个字符集合，而是一个字符数组集合，每个数组包括两个元素，第一个元素是客户值，第二个元素是产品值。

4.4.1.2.2.2.4 GetComboListString

从指定的列中提取不重复的值，用符号"|"将这些值连接成一个字符串，并返回这个字符串。

此方法通常用于动态设置列表项目。

语法：

GetComboListString(ColumnName，Filter, Sort)

ColumnName：列名称，从此列中提取不重复的值。

Filter:　可选参数，指定一个条件表达式，只返回符合此条件的值。

Sort:　可选参数，指定排序列，如果省略，则根据取值列排序，通常无需设置。

请参考表达式的运算符和函数 和 条件表达式

示例一

列出订单表所有的客户：

```
Dim s As String  = DataTables("订单").GetComboListString("客户")
output.show(s)
```

示例二

返回的值默认按取值列排序，可以另外指定排序列，例如按产品编号顺序，列出产品表中所有的产品名称：

```
Dim s As String = DataTables("产品").GetComboListString("产品名称","","产品编号")
output.show(s)
```

示例三

可以给排序列加上关键词，例如按最近一次订货的日期顺序，列出订单表中的客户名单：

```
Dim s As String  = DataTables("订单").GetComboListString("客户","","日期 Desc")
Output.Show(s)
```

实例四

可以设置取值条件，例如从客户表中提取华北地区的客户名单：

```
Dim s As String
s = DataTables("客户").GetComboListString("客户名称","[地区] = '华北'")
Output.Show(s)
```

示例五

可以同时设置取值条件和排序列，例如按日期顺序，列出 2012 年 6 月 1 日后订购过 PD01 产品的客户名单：

```
Dim s As String
s = DataTables("订单").GetComboListString("客户","[产品]= 'PD01' And [日期] > #6/1/2012#","日期")
Output.Show(s)
```

4.4.1.2.2.2.5 Compute

根据条件统计表中数据。

语法：

Compute(Expression, Filter)

Expression: 要计算的表达式，使用聚合函数进行统计。

Filter: 可选参数，用于设置计算条件，请参考表达式的运算符和函数 和 条件表达式。

在执行以下示例之前，请打开 CaseStudy 目录下的示例文件"统计演示.Table"。

示例一

计算总的销售数量和金额：

```
Dim Total As Integer
Dim Amount As Double
With DataTables("订单")
    Total = .Compute("Sum(数量)")
```

```
    Amount = .Compute("Sum(金额)")
End With
Output.Show("数量:" & Total)
Output.Show("金额:" & Amount)
```

示例二

计算产品 PD01 的销售数量：

```
Dim Total As Long
Total = DataTables("订单").Compute("Sum(数量)", "产品  = 'PD01'")
Output.Show(Total)
```

示例三

统计雇员EP01成交的订购数量超过500的订单数：

```
Dim cnt As Integer
cnt = DataTables("订单").Compute("Count([客户])", "雇员  = 'EP01' And  数量  > 500")
Output.Show("订单数:" & cnt)
```

示例四

计算每个客户的订购数量：

```
Dim dt As DataTable = DataTables("订单")
Dim Total As Integer
Dim Customers As List(Of String)
Customers = dt.GetValues("客户")
For Each Customer As String In Customers
    Total = dt.Compute("Sum(数量)", "[客户] = '" & Customer & "'")
    Output.Show(Customer & ":" & Total)
Next
```

上述代码中，Filter 参数分成了三部分，各部分用运算符&连接起来：

"[客户] = '" & Customer & "'"

如果客户名称为 CS01，那么三部分组合后，Filter 参数就等于：

"[客户] = 'CS01'"

通过代码动态合成条件表达式，是一种基本的技能，大家务必要掌握。

参考：GetValues

4.4.1.2.2.2.6 Find

在 DataTable 查找符合条件的行，如果找到的话，返回找到的行，否则返回 Nothing。

如果有多个符合条件的行，默认返回第一个，也可以指定返回第几个符合条件的行。

语法：

```
Find(Filter,Sort,Index)
```

Filter: 条件表达式，请参考表达式的运算符和函数 和 条件表达式。

Sort: 可选参数，指定排序方式。

Index: 可选参数，指定返回第几个符合条件的行，0 表示第一行。

示例一：

```
Dim dr As DataRow
dr = DataTables("产品").Find("产品编号 = '03'") '找出编号为 03 的产品
With DataTables("订单")
    dr = .Find("产品 = 'PD01'","日期") '找出第一次订购 PD01 产品的记录
    dr = .Find("产品 = 'PD01'","日期",1) '找出第二次订购 PD01 产品的记录
End With
```

示例二：

有的时候，我们需要查找倒数第几行数据，例如最近一次订购某产品的记录。

可以参考下面的代码：

```
Dim dr As DataRow
With DataTables("订单")
    dr = .Find("产品 = 'PD01'","日期 Desc") '找出最后一次订购 PD01 产品的记录
    dr = .Find("产品 = 'PD01'","日期 Desc",1) '找出倒数第二次订购 PD01 产品的记录
End With
```

可以看到代码和原来几乎一样，唯一的变化是排序参数，被改为：日期 Desc

加上 DESC 使得日期按照降序排序，最后的日期排在最前面，我们所找出的第一条记录，就是最后一次订购产品 PD01 的记录。

示例三

通常应该在代码中判断是否找到了符合条件的行，然后再运行后续的代码。

例如要找出最近一次订购产品数量超过 1000 的订单，并显示订单的日期和客户：

```
Dim dr As DataRow
dr = DataTables("订单").Find("数量 > 1000","日期 Desc")
If dr IsNot Nothing Then '如果找到的话
    Output.Show("日期:" & dr("日期"))
    Output.Show("客户:" & dr("客户"))
End If
```

如果我们不加上判断，直接：

```
Dim dr As DataRow
dr = DataTables("订单").Find("数量 > 1000","日期 Desc")
Output.Show("日期:" & dr("日期"))
Output.Show("客户:" & dr("客户"))
```

一旦订单表并不存在订购数量超过 1000 的订单，那么 Find 方法返回 Nothing，导致后续代码运行出错。

Find 函数只能找出一条符合条件的行，如果要同时找出所有符合条件的行，可以使用 Select 方法。

4.4.1.2.2.2.7 Select

以集合的形式，返回所有符合指定条件的行。

语法：

```
Select(Filter)
Select(Filter,Sort)
```

Filter: 条件表达式，请参考表达式的运算符和函数 和 条件表达式。

Sort: 可选参数，指定排序方式

我们经常需要对符合某一条件的记录，统一进行处理，此时 Select 方法就派上用场了。

示例一

例如，对于 1999 年 1 月 4 日订购 PD01 的订单，希望将其折扣统一设置为 0.12，代码为：

```
Dim drs As List(Of DataRow)
drs = DataTables("订单").Select("[产品] = 'PD01' And [日期]= #1/4/1999#")
For Each dr As Datarow In drs
    dr("折扣") = 0.12
Next
```

示例二

再例如，希望列出 1999 年 1 月 4 日订购 PD01 的客户，按订购的数量排序：

```
Dim drs As List(Of DataRow)
drs = DataTables("订单").Select("[产品] = 'PD01' And [日期]= #1/4/1999#","数量  DESC")
For Each dr As Datarow In drs
    Output.show(dr("客户"))
Next
```

上面的代码将 Sort 参数设置为“数量 DESC”，这样返回的行不仅按数量排序，而且数量多的行排在前面。

示例三

Filter 参数不能省略， 如果希望返回所有行，将 Filter 参数设置为""即可。

例如希望按总分高低，依次显示所有学生的姓名：

```
For Each dr As DataRow In DataTables("成绩表").Select("","总分  DESC")
   Output.Show(dr("姓名"))
Next
```

4.4.1.2.2.2.8 DeleteFor

删除符合条件的行。

语法：

DeleteFor(Filter)

Filter: 一个表达式，用于指定删除条件，请参考表达式的运算符和函数 和 条件表达式

例如：

删除订单表中 2007 年 2 月 1 日以前的行，代码为：

```
DataTables("订单").DeleteFor("[日期] < #2/1/2007#")
```

4.4.1.2.2.2.9 RemoveFor

移除符合条件的行。

所谓移除行，只是让用户再也看不到这些被移除的行，就像这些行从来没有被加载过一样；移除行从来都不会真正从文件中删除行，重新打开文件后，被移除的行将再次出现。

语法：

RemoveFor(Filter)

Filter: 指定移除条件。请参考表达式的运算符和函数 和 条件表达式。
例如：

```
DataTables("订单").RemoveFor("[产品] = 'PD01'")
```

4.4.1.2.2.2.10 ReplaceFor

找出符合条件的行，并将指定列的内容替换为指定值。
语法：

```
ReplaceFor(DataColName,Value,Filter)
```

DataColName：替换的列
Value：　替换值
Filter：　替换条件，请参考表达式的运算符和函数 和 条件表达式。
例如将订单表中，订购数量大于 600 的订单的折扣设为 0.15，只需简单的一行代码即可：

```
DataTables("订单").ReplaceFor("折扣", 0.15, "[数量] > 600")
```

如果不用 ReplaceFor，最精简的代码也是：

```
For Each dr As DataRow in DataTables("订单").Select("[数量] > 600")
    dr("折扣") = 0.15
Next
```

4.4.1.2.2.2.11 RejectChanges

撤销自打开文件或最近一次保存以来，对该表做出的修改。
示例
撤销对表 A 的修改：

```
DataTables("表 A").RejectChanges()
```

4.4.1.2.2.2.12 StopRedraw

我们对表做的任何变动，例如编辑数据、增加行、删除行、调整行高列宽，都会导致 Table 重新绘制，以便显示变动后的结果。
如果要对 DataTable 连续地进行大量的操作，为了避免相关 Table(表)不停地闪烁，可以先执行 StopRedraw 方法禁止绘制表，操作完成后再执行 ResumeRedraw 方法重新绘制表。
StopRedraw 方法会禁止绘制所有基于该 DataTable 的 Table，直到执行 ResumeRedraw 方法恢复绘制。
例如我们要在订单表中增加 500 行，代码如下：

```
With DataTables("订单")
    .StopRedraw
    For i As Integer = 1 To 500
        .AddNew()
    Next
    .ResumeRedraw
End With
```

注意最后一定要记得执行 ResumeRedraw 方法，否则表格不会再刷新。
如果你将代码修改为：

```
With DataTables("订单")
```

```
    For i As Integer = 1 To 500
        .AddNew()
    Next
End With
```

你可以看到代码执行过程中会不停地闪烁，而且执行速度也比之前慢很多。

StopRedraw 和 ResumeRedraw 必须配对执行，如果执行了两次 StopRedraw，那么对应的就必须执行两次 ResumeRedraw，才会恢复绘制 Table。

上面的代码只是用于演示，实际上增加 500 行最简单的代码是：

```
DataTables("订单").AddNew(500)
```

4.4.1.2.2.3 全部成员

Name	字符型，返回 DataTable 的名称
Caption	字符型，返回 DataTable 的标题
HasChanges	逻辑型，判断 DataTable 的数据是否已经被修改
AllowEdit	逻辑型，是否允许用户修改表中数据
AllowAddNew	逻辑型，是否允许用户增加行
AllowClipBoard	逻辑型，是否允许复制粘贴数据
AllowCopyHeader	逻辑型，复制数据的时候是否包括列名
AllowInitialize	逻辑型，是否允许初始化此表
AutoAddNew	逻辑型，是否自动增加行
AllowDelete	逻辑型，是否允许用户删除行
AllowLockRow	逻辑型，是否允许用户锁定行
AllowUnlockRow	逻辑型，是否允许用户取消锁定行
AllowDragColumn	逻辑型，是否允许通过拖动列标题来调整列位置
AllowFreezeColumn	逻辑型，是否允许通过鼠标拖动来调整冻结区
AllowResizeColumn	逻辑型，是否允许通过鼠标拖动来调整列宽
AllowResizeRow	逻辑型，是否允许通过鼠标拖动来调整行高
AllowResizeSingleRow	逻辑型，是否允许单独调整某一行的高度
EnterKeyActionDown	逻辑型，按回车键是否向下移到另一单元格
TabKeyActionDown	逻辑型，按 Tab 键是否向下移到另一单元格
MultiRowHeader	逻辑型，是否启用多层表头。
PreserveEditMode	逻辑型，是否进入编辑模式。
ConnectionName	字符型，返回数据源名称。
FillLoadTimeOut	整数型，设置加载数据时的超时时限，默认 30 秒，一般不需要设置此属性。
SourceType	整数型，返回数据源的类型，1 表示 Access，2 表示 SQL Server，3 表

	示 Oracle。

方法

方法	说明
AddNew	在 DataTable 中增加一行，并返回所增加的行
Compute	根据指定的条件计算指定的内容
Find	在 DataTable 查找符合条件的行，如果找到的话，返回找到的行，否则返回 Nothing
Select	以集合的形式，返回所有符合指定条件的行
Save	保存数据
ReplaceFor	找出符合条件的行，并将指定列的内容替换为指定值
DeleteFor	删除符合条件的行
RemoveFor	移除符合条件的行
RejectChanges	撤销自打开文件或最近一次保存以来，对该表做出的修改。
AcceptChanges	接受所有修改结果，使得这些修改不被保存。
GetValues	从指定列中，获取不重复的值，以集合的形式返回。
GetUniqueValues	从指定列中，获取不重复的值，以集合的形式返回(此方法已经被 GetValues 取代，请不要再使用)
GetComboListString	从指定的列中提取不重复的值，用符号"|"将这些值连接成一个字符串，并返回这个字符串
ResumeRedraw	恢复绘制表格
StopRedraw	暂停绘制表格
BuildHeader	重新生成表头
AddUserStyle	增加自定义样式

下表的属性和方法可以暂时忽略，这些是和动态加载相关的属性和方法，将在《动态加载》这一章专门讲述：

名称	类型	说明
LoadFilter	属性	字符型，用于设置重新加载数据的条件表达式，语法和所使用的数据源有关。
LoadOrder	属性	字符型，指定加载数据的顺序。
LoadTop	属性	字符型，指定要加载的行数，如果是分页加载，则用于指定每页的行数。
LoadReverse	属性	逻辑型，是否反向加载，即是否先加载新数据，此属性只有在分页加载的时候才有效。
LoadOver	属性	字符型，用于指定分页加载依据列。
LoadPage	属性	整数型，指定要加载的页号
TotalPages	属性	整数型，返回分页加载时，总的可加载总页数

Load	方法	加载数据
SQLLoad	方法	根据指定的 SQL 语句加载数据
AppendLoad	方法	追载符合条件的新数据
LoadChildren	方法	加载指定子表的数据，且只加载那些父表已经存在对应行的子表数据。
SQLCompute	方法	计算后台所有数据
SQLReplaceFor	方法	批量更新后台数
SQLDeleteFor	方法	批量删除后台数据
SQLSelect	方法	从后台查询数据
SQLUpdate	方法	保存通过 SQLSelect 查询得到的数据
SQLFind	方法	从后台查找数据
SQLGetValues	方法	从后台的指定列中提取不重复值，以集合的形式返回
SQLGetComboListString	方法	从后台的指定的列中提取不重复的值，用符号"|"将这些值连接成一个字符串，并返回这个字符串

4.4.1.2.2.4 关于 DataTables

```
DataTables("产品").Save()
DataTables("订单").AllowEdit = False
```

既然是集合，我们就可以通过 For Each 语句遍历集合中所有元素。

例如下面的代码，列出所有 DataTable 的名称：

```
For Each dt As DataTable In DataTables
    Output.Show(dt.Name)
Next
```

DataTables 的属性：

- **Count**
  用于返回 DataTable 的数量。

- **AllowEdit**
  是否允许编辑数据。
  默认为 True，如果将此属性设为 False，将锁定所有 DataTable，相当于单击菜单中的“查阅模式”按钮。

- **PreserveEditMode**
  逻辑型，是否进入编辑模式。

例如：

```
Output.Show("总表数:" & DataTables.Count)
```

```
DataTables.AllowEdit = False '锁定所有表,进入查阅模式
DataTables.AllowEdit = True
```

只有 DataTables 和 DataTable 的 AllowEdit 属性都为 True 的情况下，我们才能够在 DataTable 中输入数据。

DataTables 的方法：

Save

保存所有表，等同于单击菜单中的“保存”按钮。

- **Contains**
  判断是否存在指定名称的 DataTable，如果存在，则返回 True，否则返回 False

- **RejectChanges**
  撤销自打开文件或最近一次保存以来的全部修改。

- **Load**
  加载表，这个方法的使用会在动态加载这一章进行介绍

- **Unload**
  卸载表，这个方法的使用会在动态加载这一章进行介绍

例如：

```
DataTables.Save() '保存所有表
If DataTables.Contains("统计表 1") '如果存在统计表 1
    MessageBox.Show("此表已经存在")
End If
```

4.4.1.2.3 DataCol

4.4.1.2.3.1 概述

DataCol 表示 DataTable 中的列。

通过 DataCols 集合，可以获得指定名称的 DataCol(列)。

例如：

```
Dim dt As DataTable = DataTables("订单")
Dim dc As DataCol = dt.DataCols("日期")
```

DataCol 常用的属性有：

- **Name**
  返回列的名称。

- **AllowEdit**
  是否允许编辑该列。

- **Expression**
  返回或设置列的计算表达式，

如果希望禁止编辑某一 DataCol，也就是锁定该列，只需将其 AllowEdit 属性设为 False。
例如：
DataTables("订单").DataCols("折扣").AllowEdit = False
如果要是取消锁定该列，只需将其 AllowEdit 属性重新设为 True：
DataTables("订单").DataCols("折扣").AllowEdit = True
Expression 属性只对表达式列有效，有了这个属性，我们可以动态地修改表达式列的计算公式。
例如：
DataTables("订单").DataCols("金额").Expression = "数量 * 单价"
DataTables("订单").DataCols("金额").Expression = "数量 * 单价 * (1 - 折扣)"
有关表达式的常识和语法，请参考应用篇中和表达式相关的章节，有非常详细的介绍。

4.4.1.2.3.2 全部成员

名称	说明
Name	返回列的名称
Caption	返回或设置列标题
AllowEdit	是否允许编辑该列
Expression	返回或设置列的计算表达式
DefaultValue	返回或设置列的默认值
Unique	是否禁止该列输入重复值
MaxLength	如果是字符列，返回该列允许输入的最大长度，否则返回-1
IsNumeric	判断该列是否是数值型
IsString	判断该列是否是字符串型
IsDate	判断该列是否是日期型
IsBoolean	判断该列是否是逻辑型
ExtendType	返回或设置列的扩展类型
DefaultFolder	如果列的扩展类型为图片、文件、或者多文件，可以用此属性指定文件的存储位置。
SourceFolder	如果列的扩展类型为图片、文件、或者多文件，可以用此属性指定文件的来源位置。
Decimals	设置数值列允许输入的最大小数位数。
FileFilter	对于扩展类型为图片、文件或者多文件的列，可以用此属性设置文件类型
FTPClient	如果某列被扩展为多文件型，并用于远程文件管理，那么可以用此属性返回一个 FTPClient
Remote	是否采用远程 FTP 管理文件
ImeMode	设置此列的中文输入法状态

DataTable	返回该列所属的 DataTable

方法

名称	说明
SetDateTimeFormat	设置列的日期时间格式。
SetFormat	设置列的显示格式
RaiseDataColChanged	手工针对该列触发 DataColChanged 事件

4.4.1.2.3.3 关于 DataCols

DataTables("订单").DataCols("数量")

返回订单表的数量列。

既然是集合，我们就可以通过 For Each 语句遍历集合中所有元素。

例如执行下面的代码，可以列出订单表中所有列的名称：

```
For Each dc As DataCol In DataTables("订单").DataCols
    Output.Show(dc.Name)
Next
```

通过 DataCols 的 Count 属性，可以返回总的列数。

下面代码列出所有表的列数和列名称：

```
For Each dt As DataTable In DataTables
    Output.Show(dt.Name & "包括以下列:" )
    For Each dc As DataCol In dt.DataCols
        Output.Show(" " & dc.Name)
    Next
    Output.Show("总列数:" & dt.DataCols.Count)
Next
```

DataCols 包括三个方法：

方法	说明
Contains	判断是否包括指定名称的列
Add	增加一个临时列
Delete	删除通过 Add 方法增加的临时列。

利用 Add 方法，可以增加临时列，并可给临时列设置表达式。

通过代码增加的临时列，在重新打开项目文件后，将不复存在。

Add 方法的语法有六个：

```
Add(ColumnName, GetType(Type))
Add(ColumnName, GetType(Type), Expression)
Add(ColumnName, GetType(Type), Expression, Caption)
Add(ColumnName, GetType(Type), MaxLength)
Add(ColumnName, GetType(Type), MaxLength, Expression)
Add(ColumnName, GetType(Type), MaxLength, Expression, Caption)
```

后三个语法是针对字符列的。

参数说明：

- Name
  新增列的名称

- Type
  新增列的数据类型，需要配合 GetType 关键字来获得数据类型，例如 GetType(String)表示字符型，GetType(Boolean)表示逻辑型。

- Expression
  可选参数，指定新增列的计算表达式。
  有关表达式的常识和语法，请参考应用篇中和表达式相关的章节，有非常详细的介绍。

- MaxLength
  可选参数，用于指定字符列的长度，超过 255 将作为备注型。

- Caption
  可选参数，用于设置列标题。

例如：

```
DataTables("订单").DataCols.Add("金额", GetType(Double), "[数量] * [单价] * (1 - [折扣])")
```

有了 Add 方法，表达式列可以不通过菜单，而是在运行的时候通过代码动态添加，这样可提供更好的灵活性。
通过 Add 方法增加的临时列，可以用 Delete 方法删除。
例如：

```
If DataTables("订单").DataCols.Contains("金额") Then
    DataTables("订单").DataCols.Delete("金额")
End If
```

4.4.1.2.4 DataRow

4.4.1.2.4.1 概述

DataRow 表示 DataTable 中的行。
通过 DataRows 集合，可以获得某一位置的 DataRow(行)。
例如:

```
Dim dr As DataRow
dr = DataTables("订单").DataRows(0)
```

变量 dr 引用了订单表的第一行。
注意这个第一行可能不是你看到的第一行，你看到的是 Table 中的第一行。
DataRow 通过列名称来返回或设置各列的值,例如:

```
Dim dr As DataRow
dr = DataTables("订单").DataRows(1) '引用第二行
```

```
dr("数量") = 100 '设置该行数量列的值
dr("单价") = 20 '设置该行单价列的值
Output.Show(dr("金额")) '显示金额列的内容
```

DataRow 的常用属性有：

名称	说明
IsNull	判断某一列的内容是否为空，通常用于判断某一列是否已经输入内容。
Locked	返回或设置行的锁定状态

例如：

```
With DataTables("订单").DataRows(0) '引用第一行
    If .IsNull("结账日期") = False Then '如果已经输入结账日期
        .Locked = True '那么锁定此行
    End If
End With
```

DataRow 的常用方法有:

名称	说明
Save	保存行
Delete	删除行
Remove	移除行
Load	重新加载行，如果其他人已经修改并保存此行，执行此方法，即可将该行最新的内容同步显示在表中。
Reject	撤销自打开文件或最近一次保存以来，对该行做出的修改
GetChildRows	返回一个 DataRow 集合，集合中包括指定子表中和该行对应的全部关联行
GetParentRow	返回该行在指定父表中对应的关联行

例如：

```
With DataTables("订单")
    .DataRows(0).Save() '保存第一行
    .DataRows(1).Delete() '删除第二行
    .DataRows(2).Load() '重新加载第三行
    .DataRows(3).Reject() '撤销对第四行的修改
End With
```

GetChildRows 返回一个集合，这个集合包括指定子表中全部对应的关联行。

假定产品表和订单表之间通过产品编号建立了关联，下面的代码获得第一个产品的全部订单，并将这些订单的折扣设为 0.1：

```
Dim drs As List(Of Datarow)
drs = DataTables("产品").DataRows(0).GetChildRows("订单")
For Each dr As DataRow In drs
    dr("折扣") = 0.1
Next
```

如果要返回父表中对应的关联行，需要使用 GetParentRow 方法。

例如要找出某一订单在父表(产品表)对应的关联行，可参考下面的代码：

```
Dim dr As DataRow = DataTables("订单").DataRows(0)
```

```
Dim pr As DataRow = dr.GetParentrow("产品")
Output.Show(pr("产品名称"))
```

Remove 方法用于移除行，例如移除订单表的第一行：

```
DataTables("订单").DataRows(0).Remove()
```

移除行和删除行是不同的，移除只是使得表中暂时不再包括该行，就像该行从来没有被加载过一样；不管你是否保存文件，下次打开后，被移除的行还会出现。

4.4.1.2.4.2 全部成员

名称	说明
IsNull	判断某一列的内容是否为空
Locked	返回或设置行的锁定状态
RowState	返回行的状态
OriginalValue	获得指定列修改前的值
OriginalIsNull	判断某一列的原始值是否为空
Lines	以字符集合的形式返回或设置单元格内容
DataTable	返回该行所属的 DataTable

方法

名称	说明
Delete	删除行
Remove	移除行
Save	保存行
Load	重新加载行
Reject	撤销自打开文件或最近一次保存以来，对该行做出的修改
Accept	接受此行的所有修改，使得保存的时候忽略此行。
GetChildRows	返回一个 DataRow 集合，集合中包括指定子表中和该行对应的全部关联行
GetParentRow	返回该行在指定父表中对应的关联行
GetParentRows	如果关联是多对多关联，可以用此函数获得一个集合，该集合包括该行在指定父表中所有对应的关联行
SetError	给该行的指定列设置错误信息
GetError	返回该行指定列的错误信息
ClearErrors	清除该行所有列的错误信息

以下方法用于处理后台数据和二进制列，可以暂时忽略，将在《二进制列》这一节进行专门讲述：

名称	说明
SQlGetValue	从后台提取某列的数据
SQlSetValue	向后台某列写入数据
SQLInsertFile	向后台的二进制列插入文件

SQLLoadFile	从后台的二进制列提取文件
SQlLoadImage	从后台的二进制列提取图片

4.4.1.2.4.3 关于 DataRows

DataRows 表示数据行的集合。

例如:

```
Dim dr As DataRow
dr = DataTables("订单").DataRows(0)
```

变量 dr 引用了订单表的第一行。

注意这个第一行可能不是你看到的第一行，你看到的是 Table 中的第一行。

DataTable 包括所有行，而且行始终是按照加入的顺序排序。

既然是集合，当然可以用 For Each 语句遍历集合中的所有元素。

例如：

```
Dim Sum As Integer
For Each dr As DataRow In DataTables("订单").DataRows
    Sum = Sum + dr("数量")
Next
Output.Show("总销售数量为:" & Sum)
```

当然，上面的代码只是为了说明如何遍历 DataRows 这个集合，更简单的代码应该是：

```
Dim Sum As Integer = DataTables("订单").Compute("Sum(数量)")
Output.Show("总销售数量为:" & Sum)
```

利用 DataRows 的 Count 属性，可以返回总的行数， 例如：

```
MessageBox.Show("目前行数:" & DataTables("订单").DataRows.Count)
```

DataRows 返回的集合包括所有行，而且是无顺序的，如果希望得到符合条件的部分行，或者要按特定顺序返回行，可以参考 DataTable 的 Select 方法。

DataRows 的常用方法：

名称	说明
AddNew	增加行，等同于 DataTable 的 AddNew 方法。
Delete	删除指定位置的行
Clear	清除所有行

例如：

```
With DataTables("订单")
    .DataRows.AddNew() '增加一行
    .DataRows.AddNew(3) '增加 三行
    .DataRows.Delete(0) '删除第一行
    .DataRows.Clear() '清除所有行
End With
```

Clear 方法会直接从后台清除所有行，并即刻生效，且不能撤销，所以一定要慎重使用。

4.4.1.3 Table

4.4.1.3.1 为什么会有 Table?

DataTable 包括所有已经从数据文件加载的数据，就像一个仓库，而 Table 就像一个展厅，从仓库中提取符合条件的数据，按指定的顺序展示给客户。
所以 Table 有 Filter(筛选)和 Sort(排序)属性，用于决定展厅会展示哪些数据，按什么样的顺序展示。
DataTable 则不会有这些属性，它始终包括所有数据。
展厅是对外开放的，是看得见的，所以我们平时看到的和操作的，都是 Table 中的数据。
仓库是不对外开放的，是不可见的，我们只有通过代码才可能操作 DataTable。

为什么要区分 DataTable 和 Table 呢?
我们的代码有时是针对所有数据的，有时则是针对看得见的数据，所以区分一下当然比较好。
但是这不是理由，最根本的理由是一个 DataTable 可能有多个 Table，例如在产品表和订单表建立关联的时候，那么订单表就会有两个 Table，分别是:

```
Tables("订单")
Tables("产品.订单")
```

而 DataTable 始终只有一个：

```
DataTables("订单")
```

也就是说，现在有了两个展厅，Tables("订单")展示所有订单(如果你没有筛选的话)，Tables("产品.订单")展示产品表中选定产品的订单；而仓库还是只有一个: DataTables("订单")。
这样一来，我们就可以针对不同的"展厅"进行设置或操作，例如希望只能在"产品.订单"这个关联表中编辑数据，而不能在订单主表中编辑数据，只需：

```
Tables("订单").AllowEdit = False
Tables("产品.订单").AllowEdit = True
```

如果没有 Table 的概念存在，上述要求是不可能实现的。
这只是一个例子， 以后我们会看到，这种双层结构，将给编程带来极大的灵活性。
那么实际编程的时候，什么时候用 DataTable，什么时候用 Table 呢？其实很简单：如果你要对看得见的行进行操作，你就用 Table，如果你要对所有行进行操作，你就用 DataTable。

4.4.1.3.2 概述

通过 Tables 集合，可以获得指定名称的 Table(表)，例如：

```
Dim t1 As Table
Dim t2 As Table
t1 = Tables("订单")
t2 = Tables("产品.订单")
```

Col 表示 Table 中的列。
通过 Cols 集合，可以获得指定名称的列(Col)，例如：

```
Dim c1 As Col = Tables("订单").Cols("客户")
```

Row 表示 Table 中的行。

通过 Table 的 Rows 集合可以获得指定位置的行，例如：

```
Dim r1 As Row = Tables("订单").Rows(1)
```

行是从 0 开始编号的，所以 0 表示第一行，1 表示第二行。

在默认情况下，Rows 在计算行号的时候，是排除分组行的，也就是说 Rows(n)返回的是第 n+1 个数据行。

下图可以说明这种特性：

	产品	客户	雇员	单价	折扣	数量
1	PDO1	CSO2	EPO1	18	0	800
2	PDO1	CSO2	EPO5	14.4	0.05	100
3	PDO1	CSO3	EPO4	18	0.1	80
4	PDO1	CSO3	EPO4		0	200
5	PDO1	CSO3	EPO1		0	240
6	PDO1	CSO4	EPO5		0	200
7	PDO1	CSO4	EPO2		.05	500

Rows(2)返回此行，也就是第三个数据行。

	产品	客户	雇员	单价	折扣	数量
1	PDO1	CSO2	EPO1	18	0	800
2	PDO1	CSO2	EPO5	14.4	0.05	100
	CSO2 小计					900
3	PDO1	CSO3	EPO4	18	0.1	80
4	PDO1	CSO3	EPO4			200
5	PDO1	CSO3	EPO1			240
	CSO3 小计					520
6	PDO1	CSO4	EPO5			200
7	PDO1	CSO4	EPO2			500

在汇总模式下，Rows(2)返回的还是此行，即第三个数据行，不受前面的分组行的影响。

通过列名称，我们可以得到或设置某一行指定列的内容。

例如：

```
Dim r As Row
Dim v As Double
r = Tables("订单").Rows(0)
r("数量") = 200
v = r("折扣")
```

由此可知，Table、Row、Col 构成了表、行、列这样一个完整的结构体系，接下来我们具体介绍这三个对象。

4.4.1.3.3 Table

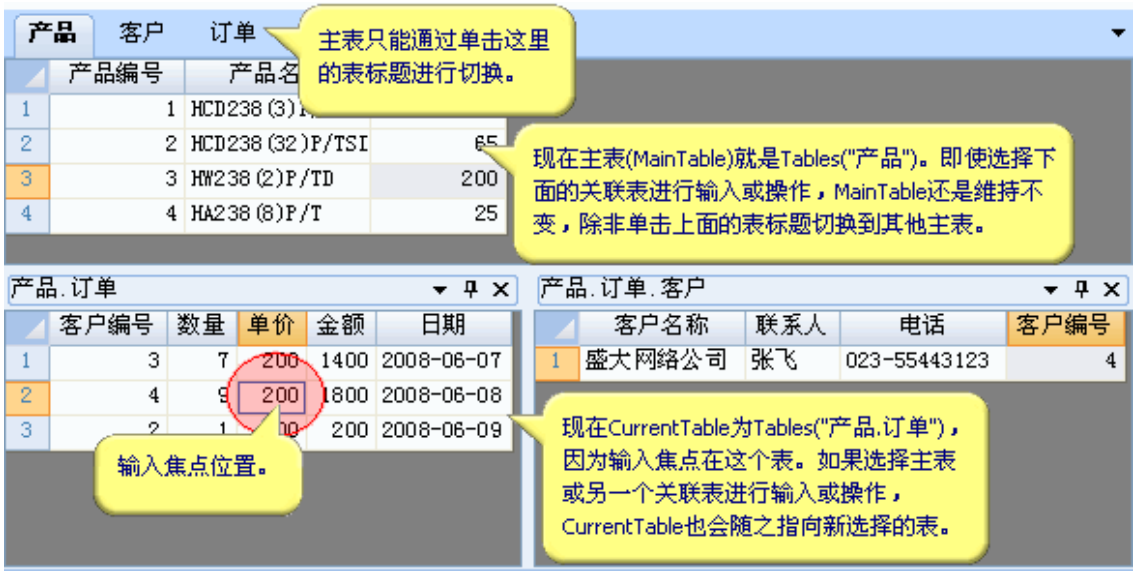
4.4.1.3.3.1 主表和活动表(必读)

有两个系统变量，分别用于返回主表和活动表

- **MainTable**
  返回当前主表，也就是通过表标题选择的表。

- **CurrentTable**
  返回当前表，也称为活动表，指的是当前输入焦点所在的表；可能是主表，也可能是主表的某个关联表。

下图说明了主表(MainTable)和当前表(CurrentTable)的关系：



通过设置 MainTable 变量，可以打开指定的主表，例如：

```
Maintable = Tables("订单")
```

表示选择订单表作为主表。

4.4.1.3.3.2 常用属性

4.4.1.3.3.2.1 设置 Table

可以利用下表中的属性设置 Table：

属性名	说明
AllowEdit	是否允许用户修改表中数据
AllowAddNew	是否允许用户增加行
AutoAddNew	是否自动增加行
AllowDelete	是否允许用户删除行
AllowLockRow	是否允许用户锁定行
AllowUnlockRow	是否允许用户取消锁定行
EnterKeyActionDown	按回车键是否向下移到另一单元格
TabKeyActionDown	按 Tab 键是否向下移到另一单元格

你也许已经留意到，上述属性 DataTable 也是具备的。

例如 AllowEdit 属性，在 Table 和 DataTable 都有，那么两者如何配合使用呢？

原则是：Table 的同名属性优先。

如果 Table 没有设置过该属性的值，那么从 DataTable 继承，如果 Table 已经设置了该属性的值，那么以 Table 的设置为准。

例如在一个简单的订单管理系统中，产品表和订单表通过产品编号建立关联，客户表和订单表通过客户编号建立关联。
现在希望产品表内容只能在 Tables("产品")修改，也就是只能在主表修改，不能在关联表修改，可以这样设置代码：

```
DataTables("产品").AllowEdit = False
Tables("产品").AllowEdit = True
```

由于 Tables("产品")的 AllowEdit 属性被设置为 True，所以它是可以编辑的。
其它基于 DataTables("产品")的 Table，例如 Tables("订单.产品")、Tables("客户.订单.产品")，因为他们没有设置 AllowEdit 属性，所以其 AllowEdit 属性的值从 DataTables("产品")继承，而 DataTables("产品")的 AllowEdit 属性被设为 False，所以这些 Table 是不能被编辑的，也就是被锁定了。
显然，只有不同的 Table 需要不同的设置时，才需要使用 Table 的设置属性，否则直接用 DataTable 的设置属性即可。
实际上 DataTable 本身是不能编辑的，因为用户根本就看不到 DataTable，用户能编辑的只能是 Table，所以 AllowEdit 属性本应该只出现在 Table 中， 而 DataTable 之所以也提供这个属性，只是为了统一设置的方便。
例如，假定不管是作为主表还是关联表，都要禁止编辑订单表，只需：

```
DataTables("订单").AllowEdit = False
```

如果 DataTable 没有 AllowEdit 属性，那么就必须逐个 Table 设置，代码显得繁琐很多：

```
Tables("订单").AllowEdit = False
Tables("产品.订单").AllowEdit = False
Tables("客户.订单").AllowEdit = False
```

我们这里只是以 AllowEdit 属性为例说明，所有 DataTable 和 Table 同时具备的属性，都有和 AllowEdit 属性一样的作用原理。

4.4.1.3.3.2.2 筛选与排序

下表中的 Table 属性，用于筛选与排序：

属性名	说明
Filter	返回或设置筛选条件表达式，请参考表达式的运算符和函数 和 条件表达式。
Sort	返回或设置排序方式

例如在订单表中需要筛选出客户为"CS01"，订购日期为 2007 年的记录：

```
Tables("订单").Filter = "[客户] = 'CS01' And [日期] >= #1/1/2007# And [日期] <= #12/31/2007#"
```

再例如，要筛选出折扣为空的记录：

```
Tables("订单").Filter = "[折扣] Is Null"
```

如果要撤销筛选：

```
Tables("订单").Filter = ""
```

排序方式由列名组成，如果有多个排序列，用逗号隔开即可。
在默认情况下列按升序排序，如果要按降序排序，请在列名后加上“DESC”，两者之间要有一个空格。

例如根据日期排序：

```
Tables("订单").Sort = "日期"
```

同样根据日期排序，不过是降序：

```
Tables("订单").Sort = "日期  DESC"
```

根据客户和日期排序，其中日期为降序，也就是相同客户的订单，最新的订单排在前面：

```
Tables("订单").Sort = "客户,日期  DESC"
```

如果要取消排序：

```
Tables("订单").Sort = ""
```

4.4.1.3.3.2.3 当前行相关

所谓当前行，就是用户选定的行，也就是光标所在的行。
当前行通常也称为选定行。
下表的属性，和当前行有关：

属性名	说明
Current	返回当前行
Position	返回或设置当前数据行位置

Position 属性是一个整数，表示当前行的位置。
位置的计算是从 0 开始的，所以当我们选择第一行的时候，Position 属性的值是 0，而不是 1；同样当 Position 属性是 1 的时候，表示选择的是第二行，而不是第一行。
例如：

```
CurrentTable.Position = 2 '选择第三行
Output.Show(CurrentTable.Position) '显示当前行的位置.
```

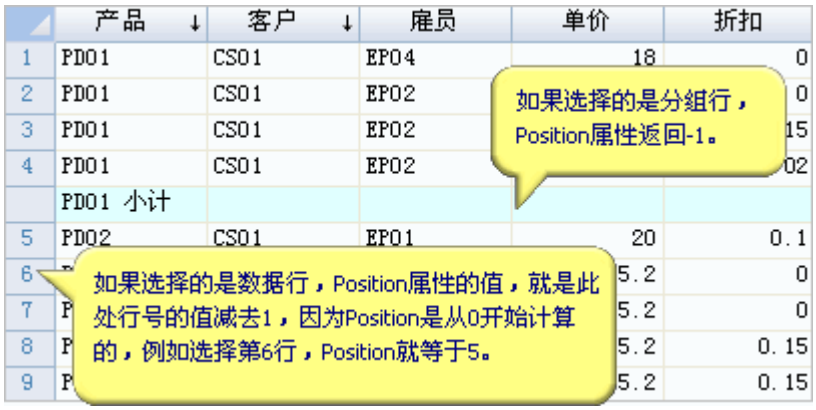
再提示一次，CurrentTable 表示当前选定的表，可能是主表，也可能是某个关联表，而 MainTable 表示主表。
可以用 Current 属性获得当前行(Row)，然后对其进行操作。
例如将订单表当前行的数量列内容设为 100：

```
Tables("订单").Current("数量") = 100
```

再例如锁定订单表的当前行：

```
Tables("订单").Current.Locked = True
```

Position 属性在计算的时候，排除汇总模式下的分组行，如果你选择了分组行，此时 Position 返回-1，而 Current 属性则返回 Nothing：



4.4.1.3.3.2.4 处理选定的多行

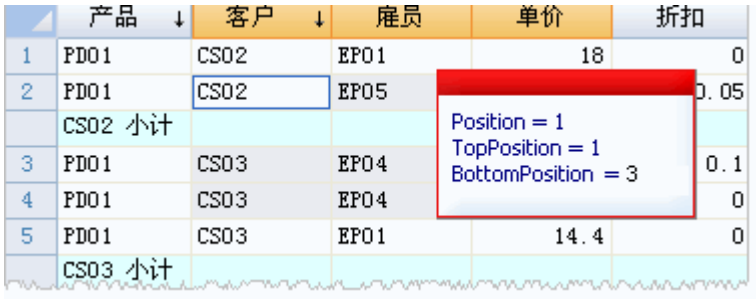
如果我们选定了一行数据，那么通过 Position 属性可以得到选定行的位置，通过 Current 属性，可以直接获得选定的行，即当前行。

如果我们选定了多行，又该如何获得所有选定行的位置，以及选定的是哪些行呢？

Foxtable 提供了两个属性，用于批量处理选定的多行

属性名	说明
TopPosition	返回选定区域第一个数据行的位置，如果只选定一行，返回值和 Position 属性相同。
BottomPosition	返回选定区域最后一个数据行的位置，如果只选定一行，返回值和 Position 属性相同。

和 Position 属性一样，这两个属性在计算的时候，均不包括分组行，下图清晰地说明了这种特性 ，记住行号是从 0 开始编号的：



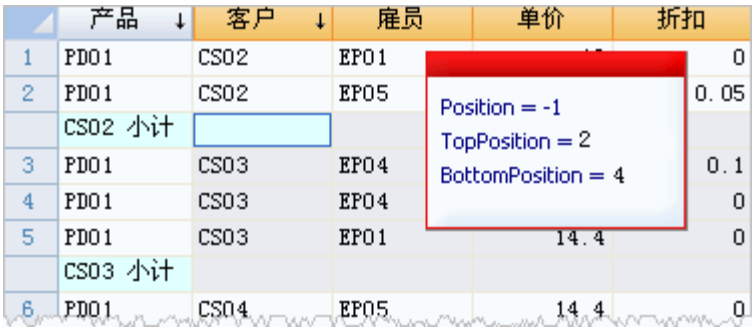
但是也稍有不同，如果选定区域的第一行是分组行，TopPosition 会向下寻找，直到找到一个不是分组行的数据行，然后返回此行的位置；

同样，如果选定区域的最后一行是分组行，BottomPosition 会向上寻找，直到找到一个不是分组行的数据行，然后返回此行的位置。

除非只选定了一行，而且此行恰好是分组行，TopPosition 和 BottomPosition 属性才会返回-1。

而 Position 属性不具备上述特性，只要选定的是分组行，就直接返回-1。

下图可以很好地说明这种特性：



这种忽略分组行的处理方式，能大大减少编码的复杂程度，因为我们的代码通常只是处理数据行的，对于分组行是要忽略的。

示例

锁定用户选定的行：

```
With Tables("订单")
    For i as Integer = .TopPosition To .BottomPosition
        .Rows(i).Locked = True
    Next
End With
```

上述代码，不管是在汇总模式下，还是普通模式下，都能正常地锁定选定的一行或多行，因为代码能够自动排除分组行。

但是如果当前选定的是一行，而且这一行恰好是分组行，那么 Position、TopPosition、BottomPosition 三个属性都返回-1，导致上述代码运行出错。

所以比较完美的代码是：

```
With Tables("订单")
    If .TopPosition > -1 Then '如果选定区域包括数据行
        For i as Integer = .TopPosition To .BottomPosition
            .Rows(i).Locked = true
        Next
    End If
End With
```

注意我们根据 TopPosition 是否大于-1，来判断选定区域是否包括分组行，其理由前面已经讲述，就不再重复。

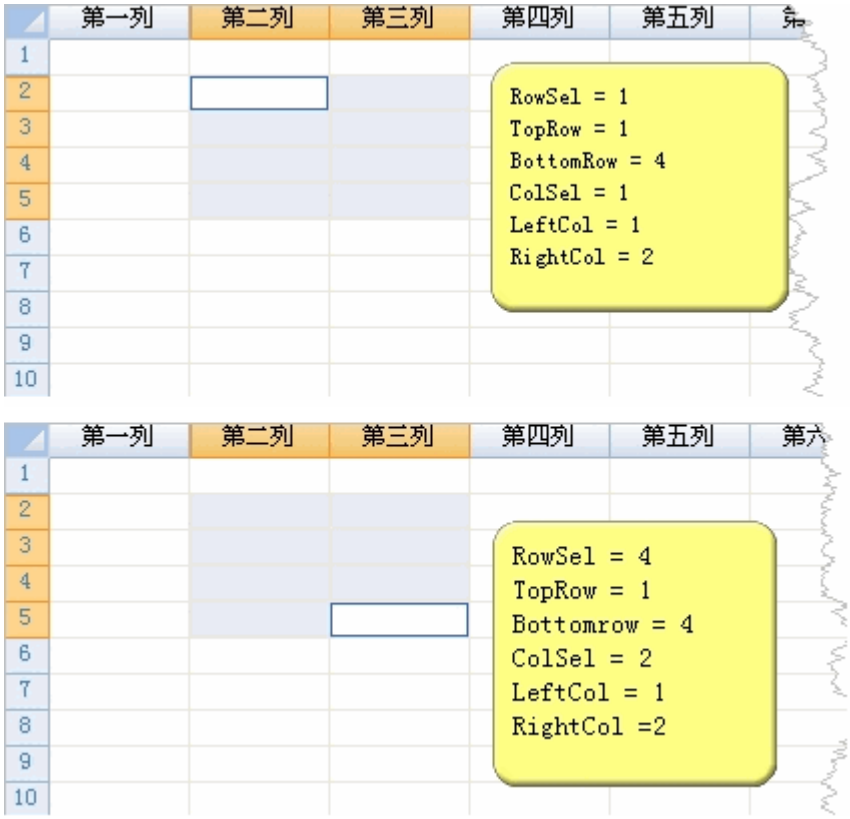
4.4.1.3.3.2.5 坐标与选定区域

下面这些属性是相关的，所以放在一起讲述。

- RowSel
  返回选定行位置。
- ColSel
  返回选定列的位置
- TopRow
  返回选定区域的最上面一行的位置。
- BottomRow
  返回选定区域的最下面一行的位置。

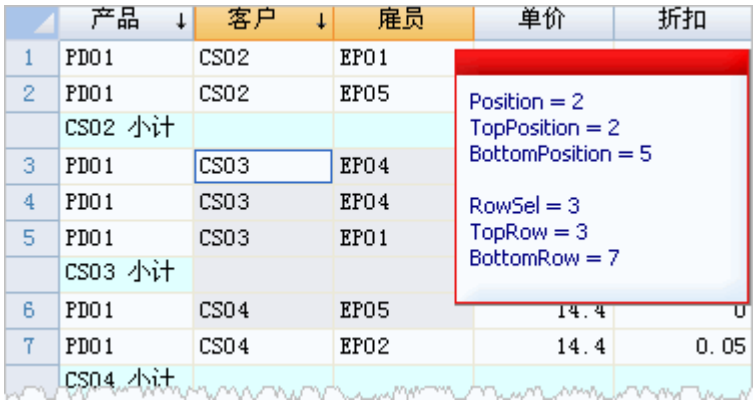
- LeftCol
  返回选定区域的最左边一列的位置
- RightCol
  返回选定区域的最右边一列的位置

下面用两幅图片来说明这些坐标属性的关系：



上述属性在计算位置的时候，是从 0 开始的；例如选定第一列，ColSel 属性的值等于 0，而不是等于 1；同样 ColSel 属性等于 1 的时候，表示选择的是第二列，而不是第一列。

RowSel、TopRow、BottomRow 三个属性，似乎和之前介绍的 Position、TopPosition、BottomPosition 属性有些重复，其实不然，如果 Table 处于汇总模式下，Position、TopPosition、BottomPosition 在计算位置的时候，是排除分组行的，而本节介绍的 RowSel、TopRow、BottomRow 等属性在计算的时候，是包括分组行的。
下图可以清晰地说明这种差别：



Position 属性是可设置的，通过设置 Position 属性，我们可以选定指定位置的行。
而本节介绍的属性是只读的，我们只能通过执行 Select 方法来选择某一区域。

Select 方法的语法是：

Select(Row, Col)
Select(TopRow, LeftCol, BottomRow, RightCol)
参数
Row： 行位置
Col: 列位置
TopRow: 选定区域最上边一行的位置
LeftCol: 选定区域最左边一列的位置
BottomRow: 选定区域最下边一行的位置
RightCol: 选定区域最右边一列的位置
上述位置在计算的时候，均包括分组行。
例如选定当前表的第 1 行第 1 列:

```
CurrentTable.Select(0,0)
```

例如选定当前表第二行第三列至第八行第 7 列之间的区域：

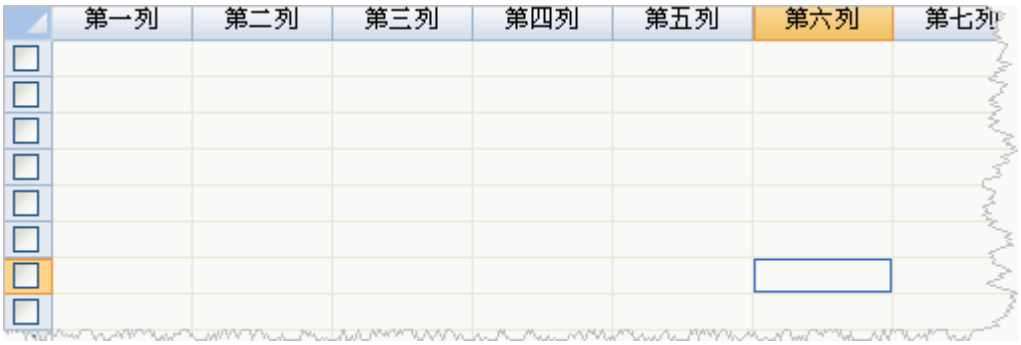
```
CurrentTable.Select(1,2,7,6)
```

再例如需要获得选定的列，可以参考下面的代码：

```
Dim t As Table = Tables("订单")
Dim c As Col = t.Cols(t.ColSel)
MessageBox.Show("选定列的名称是:" & c.Name)
```

4.4.1.3.3.2.6 处理复选框

Table 有一个名为 ShowCheckBox 的逻辑属性，此属性默认为 False，设为 True，将在行号列显示复选框：



例如，希望在员工表的行号列显示复选框：

```
Tables("员工").ShowCheckBox = True
```

通过 Table 的 GetCheckedRows 方法，可以获得一个 Row 类型的集合，该集合包括所有复选框已经选中的行。

例如删除员工表中所有复选框已经选中的行：

```
For Each r As Row In Tables("员工").GetCheckedRows
    r.Delete
Next
```

执行 Table 的 ClearCheckedRows 方法，可以清除所有复选框的选中标记。

例如：

```
Tables("员工").ClearCheckedRows
```

Row 有一个名为 Checked 的逻辑属性，用于返回或设置某一行的复选框选中状态。

例如选中订单表所有折扣超过 0.1 的行：

```
For Each r As Row In Tables("订单").Rows
    If r("折扣") > 0.1 Then
        r.Checked = True
    End If
Next
```

4.4.1.3.3.2.7 设置合并模式

下面是 Table 类型和合并模式相关的属性：

属性名	说明
MergeMode	获得或者设置 Table 的合并方式
MergeCols	一个字符集合，包括需要合并列的名称
MergeSort	返回或设置启用合并模式后的排序方式
AllowMerge	逻辑型，用于切换合并模式

MergeMode 属性是 MergeModeEnum 型枚举，包括以下可选值：

值	说明
None	不合并
Standard	标准合并模式，左边的单元格必须处于合并状态下，内容相同的单元格才会合并。

Free	自由合并，不管左边单元格是否处于合并模式，只要相邻单元格内容相同，均会合并

示例

下面的代码合并订单表中的客户、雇员两列，因为是标准合并模式，请在执行之前，将客户、雇员两列移到最左边。

完整的合并模式设置代码：

```
With Tables("订单")
    .MergeMode = MergeModeEnum.Standard '标准合并模式
    .MergeCols.Clear() '清除原来的合并列
    .MergeCols.Add("客户") '加入要合并的列
    .MergeCols.Add("雇员")
    .MergeSort = "客户,雇员" '设置合并模式下的排序方式
    .AllowMerge = True '启用合并模式
End With
```

如果要关闭合并模式：

```
Tables("订单").AllowMerge = False
```

如果要恢复上次合并：

```
Tables("订单").AllowMerge = True
```

自动生成代码

要学习合并模式设置的方法和属性，最好的办法是在合并模式窗口设置好合并模式，然后单击按钮"查看代码"，即可根据现有设置生成等效的代码。



两种合并模式

下图说明了两种合并模式的不同：



显然，对于标准合并方式，我们应该将需要合并的列移到最左边。

4.4.1.3.3.2.8 设置汇总模式

下面是 Table 类型和汇总模式相关的属性和方法。

属性

属性名	说明
GroupAboveData	逻辑型，该属性决定汇总模式下，分组行是否位于数据行之前，默认为 False，设为 True，分组行将位于数据行之前
TreeVisible	逻辑型，该属性决定在汇总模式下是否显示目录树，默认为 False，设为 True，

	将显示目录树
IsSubtotal	逻辑型，判断 Table 是否处于汇总模式
SubtotalGroups	集合，包括汇总模式下的各个分组设置

方法

方法名	说明
Subtotal	根据现有的分组设置生成汇总模式
ClearSubtotal	退出汇总模式

最重要的属性是 SubtotalGroups，这是一个 SubtotalGroup 型集合，包括汇总模式下的各个分组设置。

SubtotalGroup 指的是汇总模式下的分组，包括以下属性：

属性	说明
GroupOn	指定分组列，如果是总计分组，用星符号(*)表示
TotalOn	指定统计列，如果要对多列统计，列与列之间用逗号分隔。
Aggregate	AggregateEnum 型枚举，指定统计类型，包括以下可选值： Average：平均值 Count：记录数 Max：最大值 Min：最小值 Percent：百分比 Std：标准偏差 StdPop：总体标准偏差 Sum：累积值 Var：方差 VarPop：总体方差
DateGroup	DateGroupEnum 型枚举，指定日期列的分组方式，包括以下可选值： None：无 Year: 按年 Quarter：按季度 Month：按月 Week：按周 Day：按天 Hour：按小时 Minute：按分 Second：按秒
Caption	字符型，指定标题，标题中使用"{0}"表示分组名称
Upto	逻辑型，是否是截止统计。

例如：

```
Dim g As New SubtotalGroup '定义一个新的分组
g.Aggregate = AggregateEnum.Sum '统计类型为求和
```

```
g.GroupOn = "客户" '分组列为客户列
g.TotalOn = "数量,金额" '对数量和金额进行统计
g.Caption = "{0} 小计" '设置标题
With Tables("订单")
    .SubtotalGroups.Clear() '清除原有的分组
    .SubtotalGroups.Add(g) '加入刚刚定义的分组
    .Subtotal() '生成汇总模式
End With
```

在进行新的分组统计之前，应该先清除原来的分组。

一个汇总模式可能有多个分组，添加分组的先后顺序会影响统计结果，原则是：首先添加小的分组，然后添加大的分组，最后总计分组。

此外，根据日期型列进行分组的时候，日期分组必须位于所有分组之前。

示例一

下面的代码对订单表设置汇总模式，根据客户和产品分组，累积数量和金额列。

```
Dim t As Table = Tables("订单")
Dim g As SubtotalGroup
t.SubtotalGroups.Clear()
'定义客户分组
g = New SubtotalGroup '定义一个新的分组
g.Aggregate = AggregateEnum.Sum '统计类型为求和
g.GroupOn = "客户" '分组列为客户
g.TotalOn = "数量,金额" '统计数量和金额列
g.Caption = "{0} 小计" '设置标题
t.SubtotalGroups.Add(g) '加到分组集合中
'定义产品分组
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "产品"
g.TotalOn = "数量,金额"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)
'定义总计分组
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "*" '注意总计分组用符号*表示。
g.TotalOn = "数量,金额"
g.Caption = "总计"
t.SubtotalGroups.Add(g)
t.Subtotal() '生成汇总模式
```

代码虽然比较长，但是每一段几乎都一样，不同的只是列名。

示例二

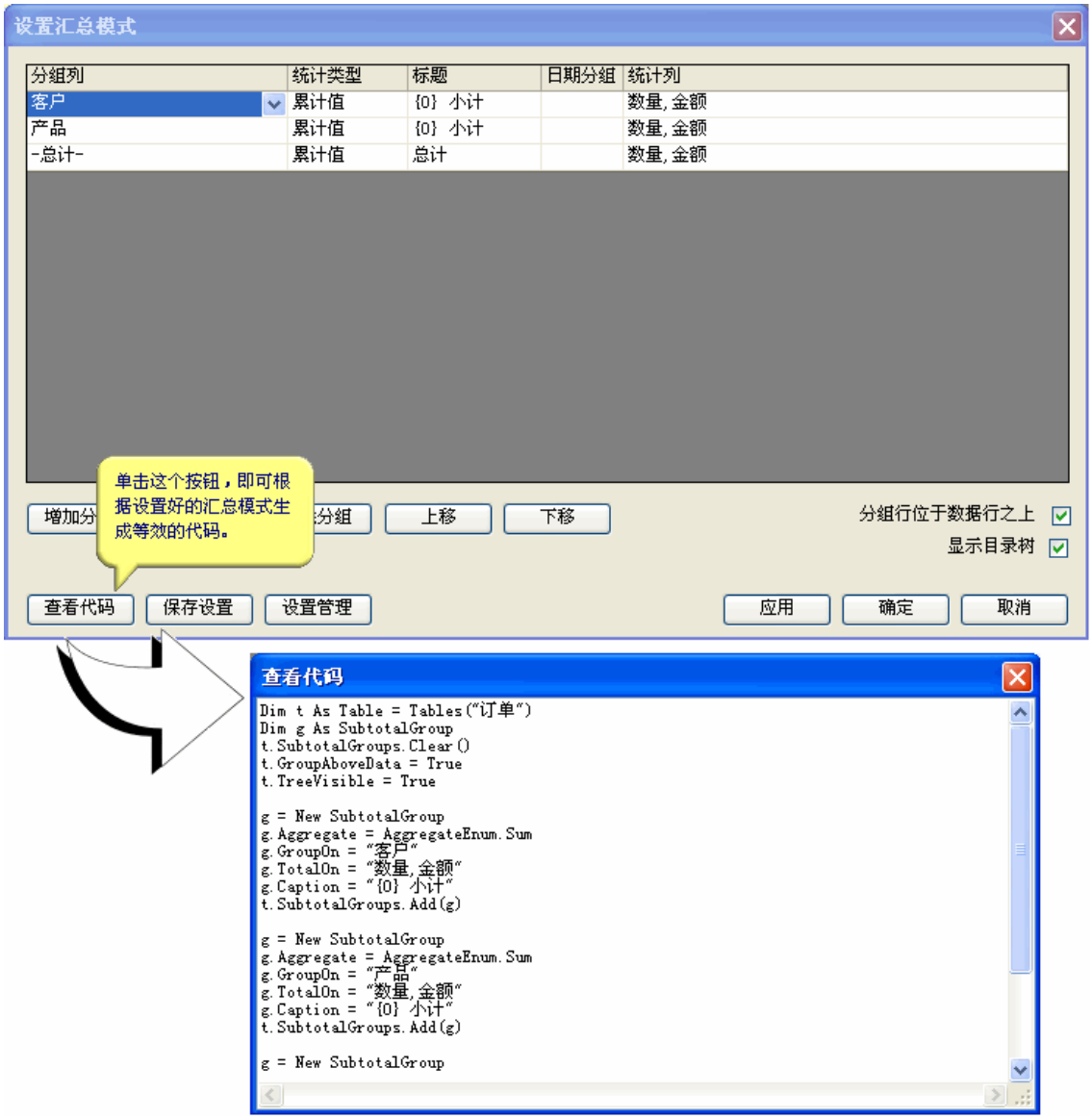
按月统计不同产品的销售数量：

```
Dim t As Table = Tables("订单")
Dim g As SubtotalGroup
t.SubtotalGroups.Clear()
```

```
t.GroupAboveData = False
t.TreeVisible = False
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "日期"
g.TotalOn = "数量"
g.Caption = "{0}月 小计"
g.DateGroup = DateGroupEnum.Month
t.SubtotalGroups.Add(g)
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "产品"
g.TotalOn = "数量"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)
t.Subtotal()
```

自动生成代码

要学习汇总模式设置的方法和属性，最好的办法是在汇总模式设置窗口中，先设置好汇总模式，然后单击按钮"查看代码"，即可根据现有设置生成等效的代码。



接下来的内容，一般用户可以忽略。

隐藏汇总模式下的非相关列

如果一个表有很多列，可能希望执行汇总模式后，能够隐藏非相关列，只显示分组列和统计列。要实现这个目的，可以参考下面的代码：

```
Dim lst As New List(of String)
Dim tbl As Table = Tables("订单")
For each g As SubtotalGroup In tbl.SubtotalGroups
    For Each s As string In g.Totalon.Split(",")
        If lst.Contains(s) = False Then
            lst.Add(s)
        End If
    Next
    If lst.Contains(g.Groupon) = False Then
        lst.Add(g.GroupOn)
```

```
    End If
Next
For Each c As Col In tbl.Cols
    c.Visible = lst.Contains(c.Name)
Next
```

上面的代码很有意思的，这是帮助中少有的真正利用集合来编程的例子。

如果希望重新显示所有列，可以利用表视图，例如建立了一个包括所有列的表视图，名称为“默认视图”，可以直接执行下面的代码来显示所有列：

```
Tables("订单").OpenView("默认视图")
```

如果不想建立表视图，也可以使用下面的代码来显示所有列：

```
For Each c As Col In Tables("订单").Cols
    c.Visible = True
Next
```

自定义排序

在默认情况下，Subtotal 方法会根据分组列进行排序，然后再生成汇总模式。

Subtotal 有一个可选参数，如果将其设置为 True，那么在生成汇总模式时，可以自定义排序参数，而不是根据分组列排序。

例如根据日期和客户排序，分组列为客户，统计列为数量与金额，生成汇总模式：

```
Dim t As Table = Tables("订单")
Dim g As Subtotalgroup
t.SubtotalGroups.Clear()
g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "客户"
g.TotalOn = "数量,金额"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)
g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "*"
g.TotalOn = "数量,金额"
g.Caption = "总计"
t.SubtotalGroups.Add(g)
t.Sort = "日期,客户" '指定排序字段
t.Subtotal(True) '生成汇总模式
```

4.4.1.3.3.2.9 其它常用属性

下表的几个属性，对于 Table 来说，也是较为常用的：

属性名	说明
Name	返回 Table 的名称
DataTable	返回 Table 所属的 DataTable

HeaderRows	如果 Table 存在多层表头，可以用此属性返回表头的层数
Visible	逻辑型，返回或者设置 Table 是否可见

例如：

```
Tables("订单").Visible = False
Tables("产品.订单").Visible = False
```

上面的代码隐藏了“订单”表和“产品.订单”表。

有的时候，我们需要获得 Table 所属的 DataTable，以便访问 DataTable 的属性或方法，例如：

```
CurrentTable.DataTable.ReplaceFor("折扣", 0.1, "产品  = 'PD01'")
```

4.4.1.3.3.3 常用方法

4.4.1.3.3.3.1 AddNew

在 Table 中增加一行或多行，并返回所增加的第一行。

语法：

AddNew()

AddNew(Count)

Count: 可选参数，要增加的行数，如果省略，则只增加一行。

例如在当前表增加一行：

```
Dim r As Row = CurrentTable.AddNew()
```

在产品表增加一行：

```
Tables("产品").AddNew()
```

再例如在关联表“产品.订单”中增加 10 行：

```
Tables("产品.订单").AddNew(10)
```

和 DataTable 的 AddNew 方法相比，用 Table 的 AddNew 方法增加行有以下优点：

1、光标会自动移到新增加的行，这样就可以通过 Table 的 Current 属性引用新增行。

2、如果在关联子表中增加行，其关联列内容会自动填入。

所以大多数时候，我们用 Table 的 AddNew 方法增加行，因为这样更方便。

4.4.1.3.3.3.2 InsertNew

在当前位置插入一行，并返回所插入的行。

示例

```
Dim r As Row = Tables("表 A").InsertNew()
r("日期") = Date.Today()
```

提示：内部表默认可以插入行，而外部表默认是不可以的；有关如何开启和关闭插入行功能，请参考：设置标志列

4.4.1.3.3.3.3 Select

选定指定位置的单元格，或者某一区域。

语法

Select(Row, Col)
Select(TopRow, LeftCol, BottomRow, RightCol)
参数
Row： 行位置
Col: 列位置
TopRow: 选定区域最上边一行的位置
LeftCol: 选定区域最左边一列的位置
BottomRow: 选定区域最下边一行的位置
RightCol: 选定区域最右边一列的位置
如果 Table 处于汇总模式之下，上述参数在计算位置的时候，均包括分组行。
例如选定当前表的第 1 行第 1 列:

```
CurrentTable.Select(0,0)
```

例如选定当前表第二行第三列至第八行第 7 列之间的区域：

```
CurrentTable.Select(1,2,7,6)
```

再例如，选择表员工表当前行的姓名列：

```
With Tables("员工")
    .Select(.RowSel, .Cols("姓名").Index)
End With
```

4.4.1.3.3.3.4 Find

在指定列中查找指定的字符内容，如果找到，返回行的位置，否则返回-1。
如果 Table 处在汇总模式下，计算行位置的时候，分组行不包括在内。
语法：
Find(StrFind, RowStart, Col, caseSensitive, fullMatch, Wrap)
Find(StrFind, RowStart, ColName, caseSensitive, fullMatch, Wrap)
下表是参数说明：

参数	说明
StrFind	字符型，要查找的内容
RowStart	整数型，从此行开始查找
Col	整数型，指定要查找的列的位置，也可以是字符型，指定要查找的列的名称。
ColName	字符型，要查找的列的名称。
caseSensitive	逻辑型，是否区分大小写
fullMatch	逻辑型，是否完全匹配，如果设为 False，那么查找“北京”,“北京市”也符合要求。
Wrap	逻辑型，是否循环查找。如果设为 True，从 RowStart 指定的行开始查找，一直找到最后一行， 如果还是没有找到符合条件的行，那么就从第一行开始重新查找。

示例一

在第一列查找“abc”，不区分大小写，不用完全匹配：

```
With CurrentTable
    Dim r As Integer
    r = .Find("abc", 0, 0, False, False, True)
    If r > - 1 Then '如果找到符合条件的行
        .Position = r '则选择该行
    End If
End With
```

示例二

上面的代码，只能定位到第一个符合条件的行。

如果我们需要循环查找，可以参考下面的代码：

```
With CurrentTable
    Dim r As Integer
    r = .Find("abc", .RowSel + 1, 0, False, False, True)
    If r > - 1 Then '如果找到符合条件的行
        .Position = r   '则选择该行
    End If
End With
```

请在当前表的第一列输入一些内容(别忘记要输入一些“abc”)，然后将上述代码复制到命令窗口，反复单击“执行”按钮，进行测试。

上面的代码，每次都从下一行开始查找，如果没有找到，则重新从第一行开始查找，所以能够实现循环查询。

示例三

也可以用列名称来指定要查找的列，例如在订单表的产品列查找产品为"PDO1"的行：

```
With Tables("订单")
    Dim r As Integer
    r = .Find("PD01", .RowSel + 1, "产品", False, False, True)
    If r > - 1 Then '如果找到符合条件的行
        .Position = r '则选择该行
    End If
End With
```

4.4.1.3.3.3.5 FindRow

根据指定的条件表达式查找行，返回整数型值，表示行的位置，如果没有找到符合条件的行，则返回 -1。

本方法也可以用于查找某一个 DataRow 在表(Table)中的位置。

如果 Table 处在汇总模式下，计算行位置的时候，分组行不包括在内。

语法一

FindRow(Expression, RowStart, Wrap)

参数

- Expression
  查找条件表达式，请参考表达式的运算符和函数 和 条件表达式。。

- RowStart
  整数型，从此行位置开始查找；可选参数，如果省略，则从第一行开始查找

- Wrap
  逻辑型，是否循环查找；可选参数，默认为 False，如果设为 True，从 RowStart 指定的行开始查找，一直找到最后一行，如果还是没有找到符合条件的行，那么就从第一行开始重新查找。

例如我们需要在当前表中的公司名称列中，查找单位包括“湛江”二字的行：

```
With CurrentTable
    Dim r As Integer
    r = .FindRow("[公司名称] Like '%湛江%'") '从  第一行开始查找
    If r >= 0 Then '如果找到的话
        .Position = r '定位到找到的行。
    End If
End With
```

上面的代码是从第一行开始查找，如果要从当前行的下一行开始查找，代码如下。

```
With CurrentTable
    Dim r As Integer
    r = .FindRow("[公司名称] Like '%湛江%'", .Position + 1, True ) '从当前行的下一行开始查找
    If r >= 0 Then '如果找到的话
        .Position = r '定位到找到的行。
    End If
End With
```

如果我们做一个按钮来执行上面的代码，那么反复单击按钮，会依次找出符合条件的行。

语法二

FindRow(DataRow)

DataRow：指定一个数据行

返回指定的数据行在 Table 中的位置，如果没找到（例如 Table 处于筛选状态，而此数据行恰好不符合筛选条件），则返回-1。

例如在订单表中，我们希望将光标定位到最后一次订购 PD01 产品的订单：

```
Dim dr As DataRow
dr = DataTables("订单").Find("[产品] = 'PD01'", "日期  Desc")
If dr IsNot Nothing Then
    Dim wz As Integer = Tables("订单").FindRow(dr)
    If wz >= 0 Then
        Tables("订单").Position = wz
    End If
End If
```

上面的代码，首先利用 DataTable 的 Find 方法找出符合条件的 DataRow，然后利用 Table 的 FindRow 方法找出这一 DataRow 在 Table 中的位置 。

说得通俗点，就是先在仓库(DataTable)中查找，看看仓库是否有符合条件的 DataRow，如果有，就到展厅(Table)中找出这个 DataRow 的位置。

4.4.1.3.3.3.6 OpenView

打开指定名称的表视图。
语法：
OpenView(Name)
Name：表视图名称
例如要打开订单表中名为"视图一"的视图：

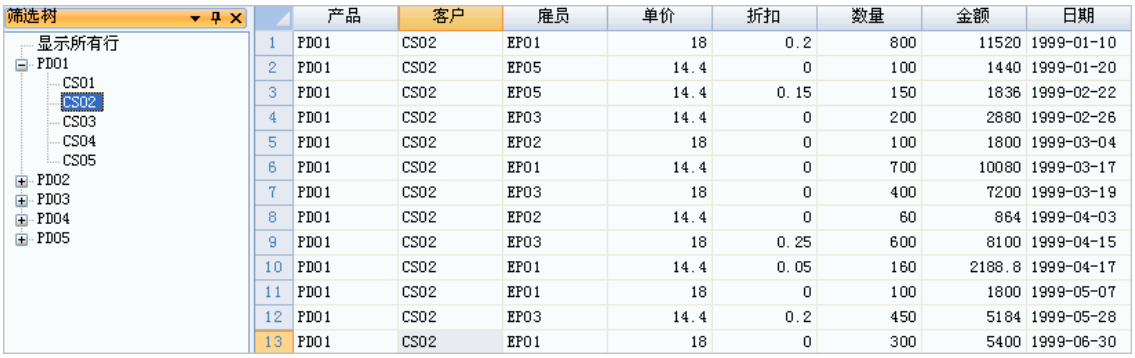
```
Tables("订单").OpenView("视图一")
```

4.4.1.3.3.3.7 OpenFilterTree

打开筛选树。
语法：
OpenFilterTree(ColNames, Size)
ColNames：指定分组列，不同的列之间用符号"|"隔开，例如“产品|客户”
Size：　可选参数，指定筛选树的宽度，如果左停靠区已经有窗口停靠，则此参数无效。
示例 1

```
Tables("订单").OpenFilterTree("产品|客户")
```

得到的筛选树为：



	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
2	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
3	PD01	CS02	EP05	14.4	0.15	150	1836	1999-02-22
4	PD01	CS02	EP03	14.4	0	200	2880	1999-02-26
5	PD01	CS02	EP02	18	0	100	1800	1999-03-04
6	PD01	CS02	EP01	14.4	0	700	10080	1999-03-17
7	PD01	CS02	EP03	18	0	400	7200	1999-03-19
8	PD01	CS02	EP02	14.4	0	60	864	1999-04-03
9	PD01	CS02	EP03	18	0.25	600	8100	1999-04-15
10	PD01	CS02	EP01	14.4	0.05	160	2188.8	1999-04-17
11	PD01	CS02	EP01	18	0	100	1800	1999-05-07
12	PD01	CS02	EP03	14.4	0.2	450	5184	1999-05-28
13	PD01	CS02	EP01	18	0	300	5400	1999-06-30

示例 2
对于日期列，可以设置分组格式，分组格式有五种，分别是：
Y：　按年分组
YQ：　按季分组
YM：　按月分组
YW：　按周分组
YMD：按日分组
例如：

```
Tables("订单").OpenFilterTree("日期  YM|产品")
```

得到的筛选树为：



	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	2010-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	2010-01-03
3	PD05	CS04	EP02	17	0	320	5440	2010-01-03
4	PD01	CS03	EP04	18	0.15	80	1224	2010-01-04
5	PD03	CS03	EP04	10	0	490	4900	2010-01-04
6	PD04	CS04	EP05	17.6	0	240	4224	2010-01-07
7	PD05	CS03	EP04	17	0	111	1887	2010-01-07
8	PD01	CS04	EP05	14.4	0	200	2880	2010-01-08
9	PD03	CS02	EP02	8	0.1	200	1440	2010-01-08
10	PD01	CS02	EP01	18	0.2	800	11520	2010-01-10
11	PD01	CS04	EP02	14.4	0.05	500	6840	2010-01-10
12	PD03	CS01	EP04	8	0	300	2400	2010-01-11

4.4.1.3.3.3.8 SaveExcel

将 Table 保存为 Excel 文件中的一个工作表。

语法

SaveExcel(FileName, SheetName, Flags)

FileName： 目标 Excel 文件名，含路径。

SheetName: 工作表名。

Flags： 可选参数，SaveExcelFlags 类型，用于设置保存选项。

SaveExcelFlags 类型的属性包括：

CellStyle： 是否保存单元格样式，默认为 False

RowNumber： 是否包括行号列，默认为 False

MergedRanges：是否保存合并模式，默认为 True

VisibleOnly： 是否只保存可见列，默认为 True

示例一

用默认设置保存订单表：

```
CurrentTable.SaveExcel("C:\data\test.xls","订单")
```

示例二

保存订单表，包括行号列和单元格样式：

```
Dim flg As New SaveExcelFlags
flg.RowNumber = True
flg.CellStyle = True
Tables("订单").SaveExcel("C:\data\test.xls","订单",flg)
```

示例三：

如果希望打开一个保存文件对话框，保存到用户指定的文件中，可以参考下面的代码：

```
Dim dlg As New SaveFileDialog '定义一个新的SaveFileDialog
dlg.Filter= "Excel文件|*.xls" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then '如果用户单击了确定按钮
    Tables("订单").SaveExcel(dlg.FileName, "订单")  '保存文件
End If
```

4.4.1.3.3.3.9 SetColVisibleWidth

批量设置要显示的列及其列宽。

语法

SetColVisibleWidth(列名 1|宽度 1|列名 2|宽度 2|列名 3|宽度 3...)

示例

```
Tables("订单").SetColVisibleWidth("日期|90|产品|120|客户|120|数量|60")
```

执行上述代码后，订单表将按顺序显示日期、产品、客户、数量四列，宽度分别为 90、120、120、60，而其余各列将被隐藏。

参考： GetColVisibleWidth

4.4.1.3.3.3.10 Print

打印表。

语法：

Print(Preview, ShowDialog)

Preview: 是否打印预览，设为 True 预览，设为 False 直接打印。

ShowDialog：是否显示打印对话框。

例如：

```
CurrentTable.Print(True,False) '打印预览当前表
```

参考： 打印表属性设置

4.4.1.3.3.3.11 StopRedraw

执行 StopRedraw 方法会停止绘制 Table，直到执行 ResumeRedraw 方法才会恢复绘制。

如果要连续地进行大量操作，可以先执行 StopRedraw 方法， 停止绘制 Table，操作完成后再执行 ResumeRedraw 方法，重新绘制 Table，这样可以避免屏幕闪烁，提高运行效率。

示例

例如，要在订单表中增加 1000 行：

```
With Tables("订单")
    .StopRedraw
    For i As Integer = 1 To 1000
        .AddNew()
    Next
    .ResumeRedraw
End With
```

如果在上述代码的开始部分，没有执行 StopRedraw 方法，那么执行过程会有明显的闪烁，且执行的时间也会长很多。

上面的代码只是用于演示，如果要在主表增加 1000 行，简洁的代码是：

```
Tables("订单").AddNew(1000)
```

StopRedraw 和 ResumeRedraw 必须配对执行，如果执行了两次 StopRedraw，那么对应的就必须执行两次 ResumeRedraw，才会恢复绘制 Table。

还记得 DataTable 的 StopRedraw 和 ResumeRedraw 方法吗，它们实现了 同样的功能，如果要停止绘制单个 Table，应该执行此 Table 的 StopRedraw 和 ResumeRedraw 方法，如果要停止绘制基于某个 DataTable 的全部 Table，那么就应该执行此 DataTable 的 StopRedraw 和 ResumeRedraw 方法。

4.4.1.3.3.3.12 Compute

根据指定的表达式和条件进行计算，语法：

Compute(Expression, Filter)

语法：

Compute(Expression, Filter)

Expression: 要计算的表达式，使用聚合函数进行统计。

Filter： 可选参数，用于设置计算条件，请参考表达式的运算符和函数 和 条件表达式。

例如，我们要统计订单表中产品 PD01 的销售金额，代码如下：

```
Dim Sum As Double
Sum = Tables("订单").Compute("Sum(数量)", "产品 = 'PD01'")
Output.Show(Sum)
```

还记得吗？ DataTable 也有 Compute 方法，两者功能一样，但是 DataTable 的 Compute 方法统计所有行，而 Table 的 Compute 方法只会统计 Table 中可见的行。

所以如果没有进行筛选，那么两者的统计结果是一样的，如果进行了筛选，就需要根据具体情况选用了。

例如你在订单表中筛选出了客户 CS01 的所有行，那么：

```
Dim Sum1 As Double = Tables("订单").Compute("Sum(数量)")
Dim Sum2 As Double = DataTables("订单").Compute("Sum(数量)")
```

Sum1 将等于客户 CS01 的订购数量，而 Sum2 将等于所有客户的订购数量。

同样：

```
Dim Sum1 As Double = Tables("订单").Compute("Sum(数量)", "产品 = 'PD01'")
Dim Sum2 As Double = DataTables("订单").Compute("Sum(数量)", "产品 = 'PD01'")
```

Sum1 将等于客户 CS01 订购产品 PD01 的数量，Sum2 将等于所有客户订购产品 PD01 的数量。

4.4.1.3.3.3.13 Aggregate

统计指定区域的数据。

语法：

Aggregate(AggregateType, R1, C1, R2, C2)

参数说明

属性	说明
AggregateType	AggregateEnum 枚举类型，指定某个分组的统计类型，包括以下可选值： Average：平均值 Count：记录数

	Max：最大值 Min：最小值 Percent：百分比 Std：标准偏差 StdPop：总体标准偏差 Sum：累积值 Var：方差 VarPop：总体方差
R1	整数型，统计区域开始行位置
C1	整数型，统计区域开始列位置
R2	整数型，统计区域结束行位置
C2	整数型，统计区域结束列位置

如果 Table 处于汇总模式之下， R1、C1、R2、C2 四个参数在计算位置的时候，均包括分组行，但是在统计数据的时候，分组行的值是不会计算在内的，所以分组行对结果没有影响。

示例一

计算订单表前 10 行的数量之和：

```
Dim p As Integer = Tables("订单").Cols("数量").Index '获得数量列的位置
Dim v As Double
v = Tables("订单").Aggregate(AggregateEnum.Sum, 0, p, 9, p)
Output.Show(v)
```

示例二

统计当前表选定区域的累积值:

```
Dim Sum As Double
With CurrentTable
    Sum = .Aggregate(AggregateEnum.Sum, .TopRow, .LeftCol, .BottomRow, .RightCol)
End With
OutPut.Show(Sum)
```

4.4.1.3.3.3.14 OpenRecordGrid

用于打开记录窗口。

语法：

```
OpenRecordGrid(Size)
```

Size：可选参数，用于指定记录窗口的宽度。

示例：

```
Tables("员工").OpenRecordGrid()
```

4.4.1.3.3.4 全部成员

4.4.1.3.3.4.1 属性

4.4.1.3.3.4.1.1 基本属性

属性名	说明
Name	返回 Table 的名称
Current	返回当前行
Position	返回或设置当前行位置
TopPosition	返回选定区域第一个数据行的位置
BottomPosition	返回选定区域最后一个数据行的位置
Visible	逻辑型，返回或者设置 Table 是否可见
DefaultRowHeight	整数型，返回或设置默认的行高，单位为像素
DefaultColWidth	整数型，返回或设置默认的列宽，单位为像素
HeaderRows	如果 Table 存在多层表头，可以用此属性返回表头的层数
DataTable	返回 Table 所属的 DataTable
ShowErrors	逻辑型，是否在单元格中显示错误图标。

4.4.1.3.3.4.1.2 排序筛选

属性名	说明
Filter	返回或设置过滤条件表达式
Sort	返回或设置排序方式
StateFilter	状态筛选
ApplyFilter	用于切换筛选
ApplySort	用于切换排序
AllowBackEndFilter	逻辑型，当分页加载某表时，是否允许通过菜单筛选后台数据。

4.4.1.3.3.4.1.3 坐标相关

属性名	说明
Position	返回或设置当前数据行位置
TopPosition	返回选定区域的第一个数据行的位置
BottomPosition	返回选定区域的最后一个数据行的位置
RowSel	返回选定行位置，计算的时候包括分组行

ColSel	返回选定列的位置
TopRow	返回选定区域的最上面一行的位置，计算的时候包括分组行
BottomRow	返回选定区域的最下面一行的位置，计算的时候包括分组行
LeftCol	返回选定区域的最左边一列的位置
RightCol	返回选定区域的最右边一列的位置

4.4.1.3.3.4.1.4 判断相关

属性名	说明
IsRelation	判断指定的 Table 是主表还是关联表
IsParent	如果指定的 Table 是关联表，而且显示的是父表的数据，则返回 True，否则返回 False
IsChild	如果指定的 Table 是关联表，而且显示的是子表的数据，则返回 True，否则返回 False
Focused	判断输入焦点是否在指定的 Table 上
CanUndo	判断指定的 Table 是否有可以撤销的编辑操作
CanRedo	判断指定的 Table 是否有可以重做的编辑操作

4.4.1.3.3.4.1.5 设置相关

属性名	说明
ListMode	逻辑型，是否突出显示选定行
RowHeaderVisible	逻辑型，是否显示行号列
ShowCheckBox	逻辑型，是否在行号列显示复选框
ExtendLastCol	逻辑型，是否自动调整最后一列的宽度，以适用表宽。
FocusRect	逻辑型，是否突出显示焦点单元格
AllowClickSort	逻辑型,是否允许单击列标题排序
AllowEdit	逻辑型，是否允许用户修改表中数据
AllowAddNew	逻辑型，是否允许用户增加行
AutoAddNew	逻辑型，是否自动增加行
AllowDelete	逻辑型，是否允许用户删除行
AllowLockRow	逻辑型，是否允许用户锁定行
AllowUnlockRow	逻辑型，是否允许用户取消锁定行
AllowDragColumn	逻辑型，是否允许通过拖动列标题来调整列位置

AllowFreezeColumn	逻辑型，是否允许通过鼠标拖动来调整冻结区
AllowResizeColumn	逻辑型，是否允许通过鼠标拖动来调整列宽
AllowResizeRow	逻辑型，是否允许通过鼠标拖动来调整行高
AllowResizeSingleRow	逻辑型，是否允许单独调整某一行的高度
EnterKeyActionDown	逻辑型，按回车键是否向下移到另一单元格
TabKeyActionDown	逻辑型，按 Tab 键是否向下移到另一单元格
PreserveEditMode	逻辑型，是否进入编辑模式。

4.4.1.3.3.4.1.6 统计相关

属性名	说明
GroupAboveData	逻辑型，该属性决定汇总模式下，分组行是否位于数据行之前，默认为 False，设为 True，分组行将位于数据行之前
TreeVisible	逻辑型，该属性决定在汇总模式下是否显示目录树，默认为 False，设为 True，将显示目录树
SubtotalGroups	集合，包括汇总模式下的各个分组设置
IsSubtotal	逻辑型，判断 Table 是否处于汇总模式
GrandTotal	逻辑型，是否进入合计模式。
MergeMode	获得或者设置 Table 的合并方式
MergeCols	一个字符集合，包括需要合并列的名称
MergeSort	返回或设置启用合并模式后的排序方式
AllowMerge	逻辑型，用于切换合并模式

4.4.1.3.3.4.1.7 可见区域

Table 有以下属性，用 来设置或获得可见区域的位置。

属性名	说明
TopVisibleRow	整数型，返回或设置第一个可见行的位置
BottomVisibleRow	整数型，返回最后一个可见行的位置。
LeftVisibleCol	整数型，返回或设置第一个可见列位置。
RightVisibleCol	整数型，返回最后一个可见列位置。

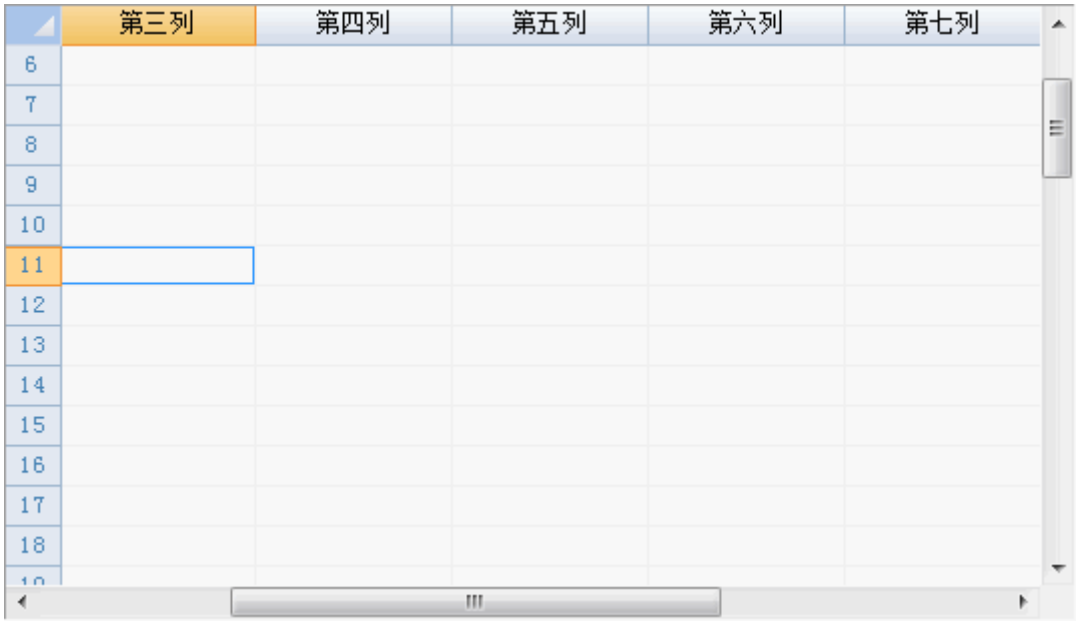
以下图为例，各属性的值为：

TopVisibleRow：5

BottomVisibleRow：18

LeftVisibleCol：2

RightVisibleCol：6



提示：

1、行和列都是从 0 开始编号的，例如上图中，第一个可见行是第 6 行，所以 TopVisibleRow 等于 5，第一个可见列是第 3 列，所以 LeftVisibleCol 等于 2。

2、上图的第 19 行虽然显示不完整，但依然是最后一个可见行，所以 BottomVisibleRow 等于 18，而不是 17，行位置从 0 开始编号，18 表示第 19 行。

通过设置 TopVisibleRow 和 LeftVisibleCol 属性，可以使得表滚动到指定的行列。

例如希望滚动员工表，使得第三行成为第一个可见行，姓名列成为第一个可见列：

```
With  Tables("员工")
        .LeftVisibleCol  =  .Cols("姓名").Index
        .TopVisibleRow  =  2
End  With
```

4.4.1.3.3.4.1.8 杂项属性

属性名	说明
Value	获得或设置指定坐标位置的单元格的值
Text	返回指定坐标位置的单元格的显示内容
Font	返回或设置 Table 的字体
ForeColor	返回或设置 Table 的字体颜色
UseVisualStyle	返回或设置配色方案是否受界面风格的影响
ScrollPosition	返回或设置表的滚动位置，Point 类型。

4.4.1.3.3.4.2 方法

4.4.1.3.3.4.2.1 基本方法

方法名	说明
AddNew	在 Table 中增加行
Find	在指定列中查找指定的字符内容，并返回行的位置
FindRow	根据指定的条件表达式查找行，并返回行的位置，也可查找某一个 DataRow 在表(Table)中的位置
OpenView	打开指定名称的表视图
OpenRecordGrid	打开记录窗口
CloseRecordGrid	关闭记录窗口
GetCheckedRows	获得一个 Row 类型的集合，该集合包括所有复选框已经选中的行。
ClearCheckedRows	清除所有行的复选框的选中标记。
StopRedraw	停止绘制 Table
ResumeRedraw	恢复绘制 Table
SaveExcel	将 Table 保存为 Excel 文件中的一个工作表
SavePDF	将 Table 保存为一个 PDF 文件
ShowToolTip	在指定的单元格位置，动态显示一个提示内容。
HideToolTip	隐藏动态提示
Refresh	重新绘制所有单元格

4.4.1.3.3.4.2.2 行列设置

方法名	说明
SetColVisibleWidth	批量设置要显示的列及其列宽
GetColVisibleWidth	返回一个字符串，包括显示的列及列宽信息。
SetHeaderRowHeight	设置标题行高度。
SetHeaderCellForeColor	设置指定列的标题的字体颜色。
SetHeaderCellBackColor	设置指定列的标题的背景颜色。
SetHeaderCellFont	设置指定列的标题的字体。
AutoSizeHeaderRow	自动设置标题行的高度。
AutoSizeRow	自动调整指定行的高度。
AutoSizeRows	自动设置所有行高度。

AutoSizeCol	自动设置指定列的宽度。
AutoSizeCols	自动设置所有列的宽度。

4.4.1.3.3.4.2.3 统计相关

方法名	说明
Subtotal	根据现有的分组设置生成汇总模式
ClearSubtotal	退出汇总模式
GroupSubtotal	打开分组统计设置窗口
CrossSubtotal	打开交叉统计设置窗口
SetSubtotalMode	打开汇总模式设置窗口
SetMergeMode	打开合并模式设置窗口
Aggregate	统计指定区域的数据

4.4.1.3.3.4.2.4 编辑相关

方法名	说明
StartEditing	进入单元格编辑模式
FinishEditing	结束单元格编辑模式
ClearEditInfos	清除所有的编辑信息，使得撤销和重做按钮无效
EditCopy	复制选定区域的内容到剪贴板
EditCut	剪切选定区域的内容到剪贴板
EditPaste	从剪贴板中读取内容粘贴到表中
EditDelete	删除选定区域的内容
Focus	将输入焦点移到 Table 中
Select	选定指定位置的单元格，或者某一区域
Undo	撤销上一次的编辑，等效于菜单中的撤销按钮
Redo	重做刚撤销的编辑，等效于菜单中的重做按钮

4.4.1.3.3.4.2.5 打印相关

方法名	说明
Print	打印表
DirectPrint	直接打印表

PrintSetting	打开打印设置对话框

4.4.1.3.3.4.2.6 筛选相关

名称	说明
AdvancedFilter	高级筛选
RelationFilter	关联筛选
RepeatFilter	重复值筛选
OpenFilterTree	打开筛选树
CloseFilterTree	关闭筛选树
SetFilterTree	打开筛选树设置窗口
HideSelectedRows	隐藏选定行
HideUnSelectedRows	隐藏未选定行

4.4.1.3.3.4.2.7 加载树

名称	说明
OpenLoadTree	打开加载树
CloseLoadTree	关闭加载树
SetLoadTree	设置加载树

4.4.1.3.3.5 关于 Tables

Tables("产品")

Tables("产品.订单")

既然是集合，我们就可以通过 For Each 语句遍历集合中所有元素。

例如下面的代码，列出所有 Table 的名称：

```
For Each tb As Table In Tables
    Output.Show(tb.Name)
Next
```

Tables 有一个属性 Count，用于返回总的 Table 个数。

Tables 有一个方法 Contains，用于判断是否包括指定名称的 Table。

```
If Tables.Contains("产品.订单") Then
    Output.Show("产品表和订单表之间建立了关联!")
End If
Output.Show("DataTable 数量:" & DataTables.Count)
Output.Show("Table 数量:" & Tables.Count)
```

4.4.1.3.4 Col

4.4.1.3.4.1 常用属性

Col 表示 Table 中的列。

通过 Cols 集合，可以获得指定名称的列(Col)，例如：

```
Tables("订单").Cols("客户")
```

表示订单表的客户列。

也可以获得指定位置的列，例如：

```
Tables("订单").Cols(2)
```

下表是 Col 的常用属性：

属性名	说明
Name	返回列的名称
Index	返回列的位置
Caption	返回列的标题
ComboList	返回或设置列表项目，不同项目之间用符号“|”隔开
Visible	返回或设置列是否可见，设为 False，列将被隐藏
AllowEdit	是否允许编辑列
DataCol	返回 Col 对应的数据列(DataCol)。

通过 Name 属性和 Index 属性，我们可以获得指定名称列的位置，也可以获得指定位置列的名称：例如：

```
Output.Show("金额列的位置是:" & Tables("订单").Cols("金额").Index)
Output.Show("第三列的名称是:" & Tables("订单").Cols(2).Name)
```

再例如我们要选定整个数量列：

```
With Tables("订单")
    Dim c As Integer = .Cols("数量").Index
    .Select(0, c, .Rows.Count - 1, c)
End With
```

需要注意的是，Index 属性是只读的，你无法通过修改 Index 属性来改变该列位置，Col 是通过 Move 方法来调整自身位置的。

例如：

```
With Tables("订单")
    .Cols("日期").Move(0)
    .Cols("客户").Move(1)
    .Cols("产品").Move(2)
End With
```

上述代码将日期、客户、产品移到前三列的位置。

ComboList 属性用于动态设置列表项目，每个项目之间用符号"|"隔开，例如：

```
Tables("订单").Cols("支付方式").Combolist = "现金|转帐|支票"
```

很多时候我们需要从某一列中提取不重复的内容作为列表项目，例如：

```
Dim Str1 As String
```

```
Str1 = DataTables("客户").GetComboListString("客户名称")
Tables("订单").Cols("客户").Combolist = Str1
```

上述的代码，从客户表提取客户名称，作为订单表客户列的列表项目。

通过设置 Visible 属性，可以动态地隐藏和显示某一列，例如：

```
With Tables("订单")
    .Cols("折扣").Visible = False
    .Cols("单价").Visible = False
    .Cols("金额").Visible = False
    .Cols("数量").Visible = True
End With
```

在命令窗口执行上面的代码，将隐藏折扣、单价、金额三列，同时显示数量列。

AllowEdit 属性决定是否允许编辑该列(Col)。

如果没有明确设置 Col 的 AllowEdit 属性，则继承所属的 DataCol 的 AllowEdit 属性值。

例如某个表和其它表建立了关联，假定要求其中一列只能在主表中编辑，而不能在关联表中编辑，可以参考下面的代码：

```
DataTables("表名").DataCols("列名").AllowEdit = False '禁止编辑此列
Tables("表名").Cols("列名").AllowEdit = True '唯独主表例外
```

有的时候，我们需要获得 Col 所属的 DataCol，以便访问 DataCol 的属性和方法，例如重置当前表选定列:

```
With CurrentTable
    .Cols(.ColSel).DataCol.RaiseDataColChanged()
End With
```

4.4.1.3.4.2 常用方法

Col 的常用方法有：

方法名	说明
OpenWindow	打开列窗口
CloseWindow	关闭列窗口
Move	移动列到指定位置

我们在应用篇的时候已经知道，备注列以及扩展类型为图片、多文件的列，都有自己的列窗口。

利用 OpenWindow 和 CloseWindow 方法，可以打开和关闭列窗口，例如：

```
With Tables("员工")
    .Cols("照片").OpenWindow()
    .Cols("备注").CloseWindow()
End With
```

Move 方法用于调整列的位置，例如：

```
With Tables("订单")
    .Cols("日期").Move(0)
    .Cols("客户").Move(1)
    .Cols("产品").Move(2)
End With
```

上述代码将日期、客户、产品移到前三列的位置。

4.4.1.3.4.3 全部成员

属性名	说明
Name	返回列名称
Index	返回列位置
Caption	返回列标题
ComboList	返回或设置列表项目，不同项目之间用符号“|”隔开
AllowEdit	逻辑型，是否允许编辑列
Visible	逻辑型，列是否可见
Width	返回或设置列宽，单位为像素
PrintWidth	返回或设置列宽，单位为毫米
TextAlign	设置列数据对齐方式
TextAlignFixed	设置列标题对齐方式
CodeDictonary	返回或设置列的代码项目。
DataMap	设置列的数据字典
DropTree	设置下拉目录树，请参考：DropTreeBuilder
ImageMap	设置列的图形字典
ImageAndText	逻辑型，设置图形字典后，是否同时显示文本和图形
UsetBuildInEditor	逻辑型，是否使用内置输入器
AllowDirectType	逻辑型，是否允许用户直接输入不在列表项目中的内容
AllowTypeAssistant	逻辑型，是否启用输入助手
DataCol	返回 Col 对应的数据列(DataCol)
GrandTotal	逻辑型，在合计模式下，是否对此列进行合计，参考：设置合计模式
IsNumeric	判断该列是否是数值型
IsString	判断该列是否是字符串型
IsDate	判断该列是否是日期型
IsBoolean	判断该列是否是逻辑型
DroppedDown	逻辑型，判断下拉列表或者下拉窗口是否已经打开
ShortCaption	字符型，设置在记录窗口显示的缩写标题
ShowInRecordGrid	逻辑型，是否在记录窗口显示此列
RecordRowHeight	整数型，在记录窗口显示的行高倍数，默认为 1

方法

方法名	说明
OpenWindow	打开列窗口
CloseWindow	关闭列窗口

Move	移动列到指定位置
OpenDropDown	打开下拉列表或者下拉窗口，只有正在编辑此列的时候，此方法才有效。
CloseDropDown	关闭下拉列表或者下拉窗口。

4.4.1.3.4.4 关于 Cols

通过 Cols 集合，可以获得指定名称的列，例如：

```
Tables("订单").Cols("客户")
```

表示订单表的客户列。

也可以获得指定位置的列，例如：

```
Tables("订单").Cols(2)
```

表示订单表的第三列，注意列号从 0 开始编号。

既然是集合，我们就可以通过 For Each 语句遍历集合中所有元素。

例如下面的代码，列出当前表所有的列名称：

```
For Each cl As Col In CurrentTable.Cols
    Output.Show(cl.Name)
Next
```

Cols 有两个属性：

属性名	说明
Count	返回总列数
Frozen	设置冻结列数

例如要冻结前两列：

```
CurrentTable.Cols.Frozen = 2
```

要取消冻结列：

```
CurrentTable.Cols.Frozen = 0
```

Cols 的方法有：

方法名	说明
Remove	从 Table 中移除指定的列
Contains	用于判断是否包括指定名称的 Col

Remove 方法从 Table 中移除指定的列，注意是移除，不是删除，被移除的列继续存在于 DataTable 中，该列还可以正常参与各种计算，只是用户再也看不到这一列。

移除列类似于隐藏列，不过被隐藏的列，用户可以通过菜单命令撤销隐藏，而被移除的列，除非重新打开项目文件，否则用户再也无法看到该列。

语法：

Remove(ColName1, ColName2, ColName3...)

ColName1：字符型，指定要移除列的列名称，可以指定任意多列。

例如在订单表中移除折扣列：

```
If Tables("订单").Cols.Contains("折扣") Then
    Tables("订单").Cols.Remove("折扣")
End If
```

也可以一次移除多列，例如：

```
Tables("订单").Cols.Remove("折扣","金额")
```

4.4.1.3.5 Row

4.4.1.3.5.1 概述

Row 表示 Table 中的行。

通过 Table 的 Rows 集合可以获得指定位置的行，例如：

```
Dim r As Row = Tables("订单").Rows(1)
```

表示订单表的第二行，行号从 0 开始编号。

在默认情况下，Rows 在计算行号的时候，是排除分组行的，也就是说 Rows(n)返回的是第 n+1 个数据行。

下图可以说明这种特性：

	产品	客户	雇员	单价	折扣	数量
1	PD01	CS02	EP01	18	0	800
2	PD01	CS02	EP05	14.4	0.05	100
3	PD01	CS03	EP04	18	0.1	80
4	PD01	CS03	EP04		0	200
5	PD01	CS03	EP01		0	240
6	PD01	CS04	EP05		0	200
7	PD01	CS04	EP02		.05	500

Rows(2)返回此行，也就是第三个数据行。

	产品	客户	雇员	单价	折扣	数量
1	PD01	CS02	EP01	18	0	800
2	PD01	CS02	EP05	14.4	0.05	100
	CS02 小计					900
3	PD01	CS03	EP04	18	0.1	80
4	PD01	CS03	EP04			200
5	PD01	CS03	EP01			240
	CS03 小计					520
6	PD01	CS04	EP05			200
7	PD01	CS04	EP02			500

在汇总模式下，Rows(2)返回的还是此行，即第三个数据行，不受前面的分组行的影响。

Table 有一个 Current 属性，表示当前选定的行，例如：

```
Dim r As Row = Tables("订单").Current
```

通过列名称，可以获得或设置某一行中指定列的内容，例如：

```
Dim r As Row = Tables("订单").Current
r("折扣") = 0.1
Tables("订单").Rows(0)("客户") = "CS01"
Output.Show(r("折扣"))
Output.Show(Tables("订单").Rows(0)("客户") )
```

Row 的常用属性有：

属性名	说明
IsNull	逻辑型，判断某一列的内容是否为空。

Locked	逻辑型，设置或者返回行的锁定状态
DataRow	获得 Row 所对应的 DataRow

例如：

```
With Tables("订单").Current '引用当前行
    If .IsNull("结账日期") = False Then '如果已经输入结账日期
        .Locked = True '那么锁定此行
    End If
End With
```

有的时候，我们需要获得 Row 所属的 DataRow，以便访问 DataRow 的属性和方法，例如重新加载当前行：

```
Tables("订单").Current.DataRow.Load()
```

Row 的常用方法有：

方法名	说明
Save	保存行
Delete	删除行
Clone	克隆行，所谓克隆，就是新增加一行，且该行和被克隆行的各列内容完全一样

例如：

```
Tables("订单").Current.Save() '保存选定行
Tables("订单").Rows(0).Clone() '克隆第一行
Tables("订单").Rows(1).Delete() '删除第二行
```

4.4.1.3.5.2 全部成员

方法名	说明
Delete	删除行
Move	移动行
Save	保存行
Reject	撤销修改
Remove	移除行
Clone	克隆行
Load	重新加载行

属性

属性名	说明
Index	返回该行(Row)在表(Table)中的位置
IsNull	逻辑型，判断某一列是否有内容
Locked	逻辑型，返回或设置行的锁定状态
Checked	逻辑型，返回或设置行的复选框是否选中
DataRow	获得 Row 所对应的 DataRow

IsGroup	逻辑型，判断该行是否是汇总模式下的分组行
Level	整数型，返回分组行的层级

4.4.1.3.5.3 关于 Rows

通过 Table 的 Rows 集合可以获得指定位置的行，例如：

```
Dim r As Row = Tables("订单").Rows(1)
```

表示订单表的第二行，行号从 0 开始编号。

既然是集合，就可以遍历，例如：

```
Dim Sum As Long
For Each r As Row In Tables("订单").Rows
    Sum = Sum + r("数量")
Next
Output.Show(Sum)
```

利用 Rows 的 Count 属性，可以获得 Table 的总行数。

例如：

```
Dim n As Integer = Tables("订单").Rows.Count
MessageBox.Show("订单数量:" & n)
```

再例如选定整个数量列：

```
Dim p As Integer
With CurrentTable
    p =.Cols("数量").Index
    .Select(0, p, .Rows.Count - 1, p)
End With
```

Rows 的常用方法有：

方法名	说明
AddNew	增加一行或多行，并返回所增加的第一行，同时将输入焦点移到该行上，等同于 Table 的 AddNew 方法
InsertNew	在当前位置插入一行，并返回所插入的行，等同于 Table 的 InsertNew 方法
Delete	删除指定位置的行

例如：

```
Dim r As Row = Tables("订单").Rows.AddNew() '增加一行
r("日期") = Date.Today '新增行的日期设为当天
Tables("订单").Rows.AddNew(8) '一次增加 8 行
Tables("订单").Rows.Delete(0) '删除第一行
Tables("订单").Rows.Delete(Tables("订单").Rows.Count - 1) '删除最后一行
Output.Show("总行数:" & Tables("订单").Rows.Count) '显示总行数
```

4.4.1.4 Relation

4.4.1.4.1 Relation

一般用户可以忽略本节内容，因为很少需要通过代码来管理关联的。
Relation 表示关联。
通过集合 Relations，可以获得指定名称的 Relation。
示例

```
Dim re As Relation
re = Relations("ab")
```

Relation 的属性有：

名称	说明
Name	字符型，返回关联名称
ParentTable	DataTable 类型，返回父表
ChildTable	DataTable 类型，返回子表
CascadeUpdate	逻辑型，是否同步更新关联列内容
CascadeDelete	逻辑型，是否同步删除关联行
ParentColumn	DataCol 类型，返回父表关联列
ChildColumn	DataCol 类型，返回子表关联列
ParentColumns	DataCol 类型数组，返回所有父表关联列，一般通过多列建立关联时才使用。
ChildColumns	DataCol 类型数组，返回所有子表关联列，一般通过多列建立关联时才使用。
RelationPath	RelationPathEnum 型枚举，用于返回关联表生成模式，有三个可选值，分别是:One(单向生成),Both(双向生成),None(不生成)

示例
列出所有的关联及其属性：

```
For Each re As Relation In Relations
    Output.Show("名称:" & Re.Name)
    Output.Show("父表:" & Re.ParentTable.Name)
    Output.Show("父表关联列:" & Re.ParentColumn.Name)
    Output.Show("子表:" & Re.ChildTable.Name)
    Output.Show("子表关联列:" & Re.ChildColumn.Name)
Next
```

4.4.1.4.2 Relations

一般用户可以忽略本节内容，因为很少需要通过代码来管理关联的。

Relations 表示关联集合，可以获得指定名称的关联，例如：

```
Dim re As Relation
re = Relations("ab")
```

Relations 包括以下属性和方法。

Count

返回关联数量。

Add

用于添加关联

语法：

Add(RelationName, ParentCol, ChildCol,Visible, RelationPath)

RelationName:　关联名称

ParentCol:　父表关联列

ChildCol:　子表关联列

Visible:　可选参数,是否显示关联表

RelationPath:　RelationPathEnum 型枚举，用于设置关联表生成模式，有三个可选值，分别是:One(单向生成),Both(双向生成),None(不生成)

示例

在产品表和订单表之间，通过产品编号建立关联：

```
Relations.Add("关联 1",DataTables("产品").DataCols("产品编号"),DataTables("订单").DataCols("产品编号"))
```

ParentCol 和 ChildCol 也可以是数组，这样可以通过多列创建关联，例如产品表和订单表，需要通过品名和型号两列建立关联：

```
Dim DataCols1(1) As DataCol
Dim DataCols2(1) As DataCol
DataCols1(0) = DataTables("产品").DataCols("品名")
DataCols1(1) = DataTables("产品").DataCols("型号")
DataCols2(0) = DataTables("订单").DataCols("品名")
DataCols2(1) = DataTables("订单").DataCols("型号")
Relations.Add("po", DataCols1, DataCols2)
```

Delete

删除指定名称的关联，这个关联必须是通过 Add 方法增加的。

示例

删除名称为“关联 1”的关联：

```
Relations.Delete("关联 1")
```

Contains

判断是否存在指定名称的关联。

示例

判断是否存在名称为“关联 1”的关联：

```
If Relations.Contains("关联 1") Then
    MessageBox.Show("关联 1 已经建立!")
```

```
End If
```

4.4.1.5 Connection

4.4.1.5.1 Connection

一般用户可以忽略本节内容，因为很少需要通过代码来管理数据源的。
Connection 表示数据源。
通过 Connections 集合，可以获得指定名称的数据源。
示例

```
Dim cn As Connection = Connections("test")
```

Connection 的属性有：
Name：　　字符型，数据源的名称
ConnectionString：字符型，返回连接字符串
SourceType：　　整数型，返回数据源的类型，1 表示 Access，2 表示 SQL Server，3 表示 Oracle。
示例
列出所有的数据源及其连接字符串：

```
For Each cn As Connection In Connections
    Output.Show("名称:" & cn.Name)
    Output.Show("链接字符串:" & cn.ConnectionString)
Next
```

Connection 的方法有：
GetTableNames：返回一个字符串集合，包括此数据源所有的表名。
GetViewNames：　返回一个字符串集合，包括此数据源所有的视图名。
示例
列出外部数据源 nwnd 中的所有表名：

```
Dim lst As List(Of String)
lst = Connections("nwnd").GetTableNames
For Each nm As String In lst
    Output.Show(nm)
Next
```

4.4.1.5.2 Connections

一般用户可以忽略本节内容，因为很少需要通过代码来管理数据源的。
Connections 表示数据源集合，可以获得指定名称的数据源。
示例

```
Dim cn As Connection
cn =  Connections("test")
```

Connections 的属性和方法有：
Add
增加数据源。

语法：

Add(Name, ConnectionString)

Name：　　新增数据源的名称。

ConnectionString：新增数据源的链接字符串。

例如：

Connections.Add("test","Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\abc.mdb;Persist Security Info=False")

Delete

删除指定名称的数据源。

语法：

Delete(Name)

Name：　要删除的数据源名称。

TryConnect

测试指定的链接字符串能否连通，如果能连通，则返回 True，否则返回 False。

语法：

TryConnect(ConnectionString,ErrMsg)

ConnectionString：要测试的链接字符串。

ErrMsg：一个可选参数，用于获取导致数据源无法连接的错误原因。

例如：

```
Dim s As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\data\Northwind.mdb;Persist Security Info=False"
If Connections.TryConnect(s) = False Then
    MessageBox.Show("数据源无法连通!")
End If
```

如果要获取数据源无法连接的原因，代码为：

```
Dim ErrMsg As String
Dim Conn As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\data\Northwind.mdb;Persist Security Info=False"
If Connections.TryConnect(Conn,ErrMsg) = False Then
   MessageBox.Show("数据源无法连通,原因:" & ErrMsg)
Else
   MessageBox.Show("数据源可以连通!")
End If
```

Contains

判断是否存在指定名称的数据源，如果存在，就返回 True。

语法：

Contains(Name)

Name：　数据源名称。

Count

返回数据源的数量。

4.4.1.6 User

UserInfo 类型表示用户，包括以下属性：

- Name
  返回用户名

- Group
  返回用户分组

- Roles
  返回用户的角色，一个用户可以有多个角色，用逗号分隔。

- Default
  逻辑型，如果是默认用户，则返回 True，否则返回 False

- Type
  返回用户级别(类型)，UserTypeEnum 型枚举，包括以下可选值:

  Developer: 开发者
  Administrator：管理员
  User：普通用户。

- ExtendedValues
  扩展用户属性集合，可以获得指定名称的扩展用户属性值。

UserInfo 的方法有：

- IsRole
  判断用户是否属于某个角色，语法：

  IsRole(Role)

  Role：角色名称

Foxtable 直接提供了一个 UserInfo 类型的系统变量 User，表示当前登录用户。
在命令窗口执行下面的代码，可以列出当前登录用户的信息：

```
Output.Show("用户名:" & User.Name)
Output.Show("用户分组:" & User.Group)
Output.Show("用户角色:" & User.Roles)
Select Case User.Type
    Case UserTypeEnum.Developer
        Output.Show("用户级别:" & "开发者")
    Case UserTypeEnum.Administrator
```

```
        Output.Show("用户级别:" & "管理员")
    Case UserTypeEnum.User
        Output.Show("用户级别:" & "普通用户")
End Select
```

通过 ExtendedValues 集合，用于获得指定名称的扩展用户属性值，例如：

```
Dim IP As String
IP = User.ExtendedValues("IP")
Output.Show(IP)
```

通过 IsRole 方法，可以判断用户具备某个角色：

```
If User.IsRole("审核") Then
    MessageBox.Show("当前用户具备审核功能!")
End If
```

FoxTable还提供了一个Users集合，用于返回所有的用户，例如：

```
For Each us As UserInfo In Users
    Output.show("分组:" & us.Group)
    Output.Show("名称:" & us.Name)
Next
```

也可以直接获得指定名称的用户，例如：

```
Dim us As UserInfo = Users("周明")
Output.show("分组:" & us.Group)
Output.Show("名称:" & us.Name)
```

4.4.1.7 高效输入

编程是很严谨的，不能有任何的偏差，对于计算机来说，“北京”和“北京市”是两个完全不同的地方。
一个管理系统，通常会有很多表，要完全准确记住这些表名和列名是不可能的。
表、行、列等类型都有大量的属性和方法，代码中的属性和方法名称同样必须准确，否则无法得到正确的结果。
如果没有辅助的输入功能，这样的要求不管是对于初学者，还是对于资深程序员，都是一个噩梦。
但是在 Foxtable 中，完全不用担心这方面的问题，因为 Foxtable 的代码编辑器不仅是智能的，而且是友善的。
下面我们实际测试一下。

自动输入表名和列名

首先在代码编辑器输入：

```
Dim m As da
```

然后按 Tab 键，会自动列出以 da 开头的类型供选择，按上下箭头选择 DataTable，按回车键，即可自动输入：

Dim m As DataTable

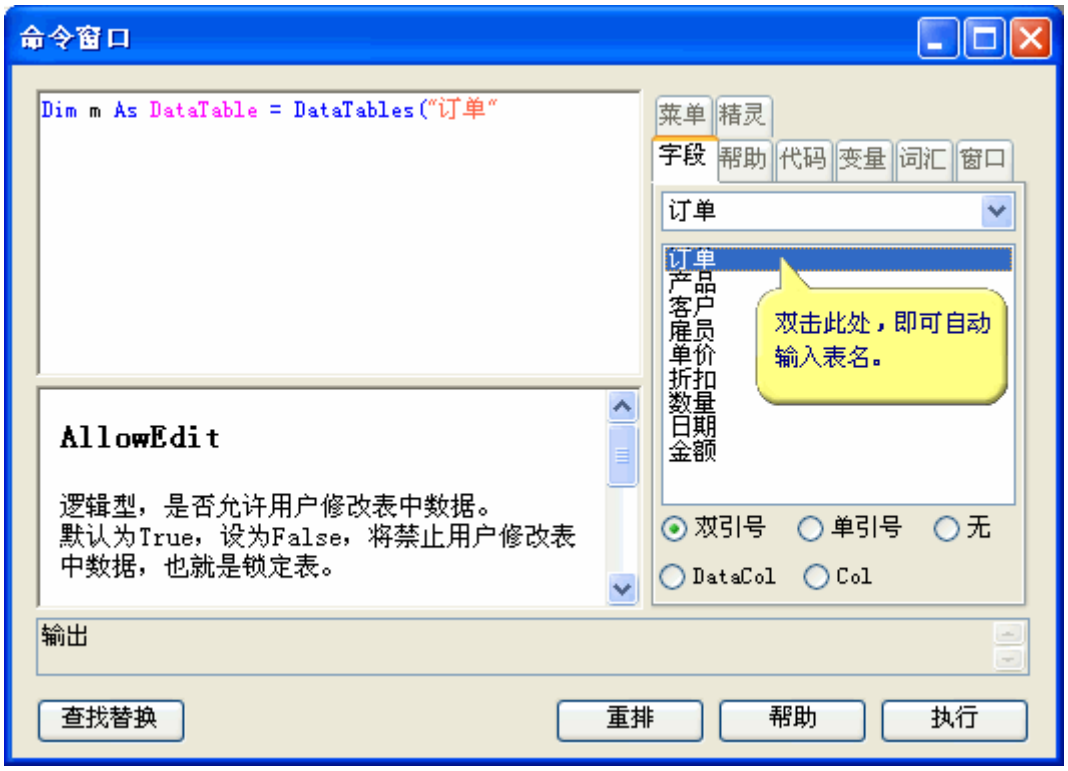
然后继续输入：

Dim m As DataTable = da

接着按 Tab 键，选择 DataTables，然后按回车，即可自动输入：

Dim m As DataTable = DataTables

继续输入一个括号，然后按下图所示双击即可输入表名。



表名可以自动输入，列名同样可以自动输入。

例如， 按前面的操作，输入：

Dim m As DataCol = DataTables("订单").

然后按 Tab 键，选择 DataCols，按回车，即可自动输入：

Dim m As DataCol = DataTables("订单").DataCols

手工输入一个括号，然后双击要输入的列名，例如数量，既可成为：

Dim m As DataCol = DataTables("订单").DataCols("数量")

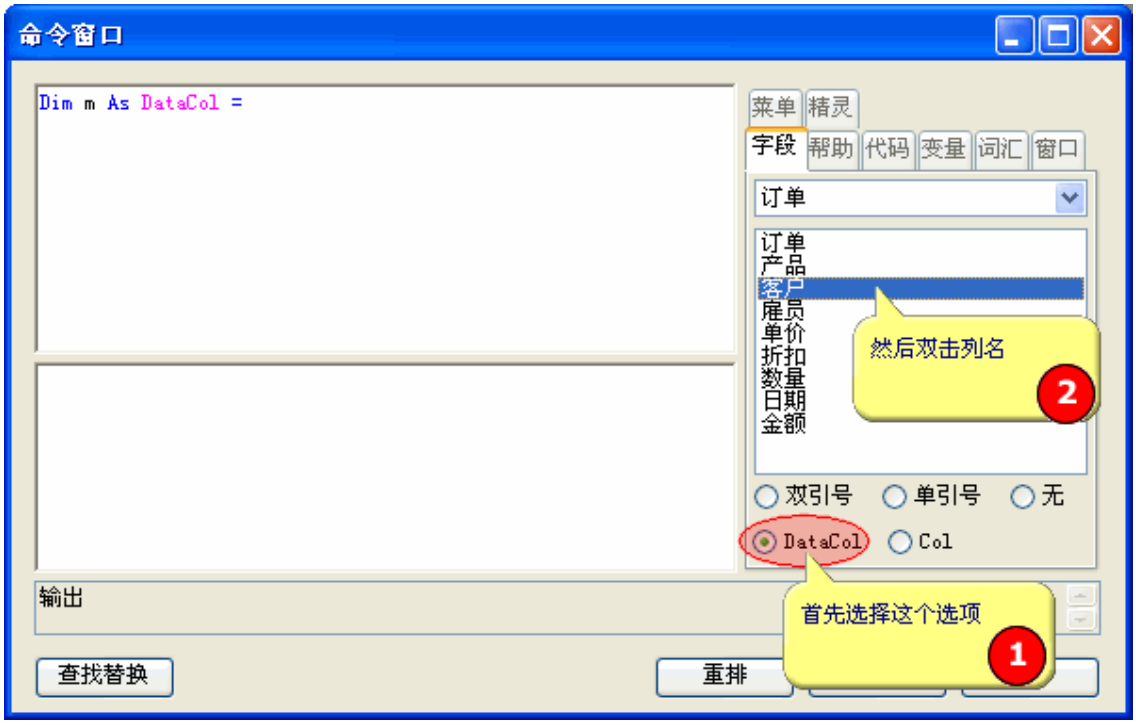
双击输入的表名和列名，默认是双引号括起来的，特殊场合，可以选择用单引号或者不用引号。

更加自动的输入

首先在代码编辑器输入：

Dim m As DataCol =

然后按下图操作：



即可自动输入：

Dim m As DataCol = DataTables("订单").DataCols("客户")

如果你选择的选项是 Col，双击即可自动输入：Tables("订单").Cols("客户")

成员的自动输入

在代码编辑器中输入：

Dim c As DataCol = Tables("订单").Cols("数量")

c.

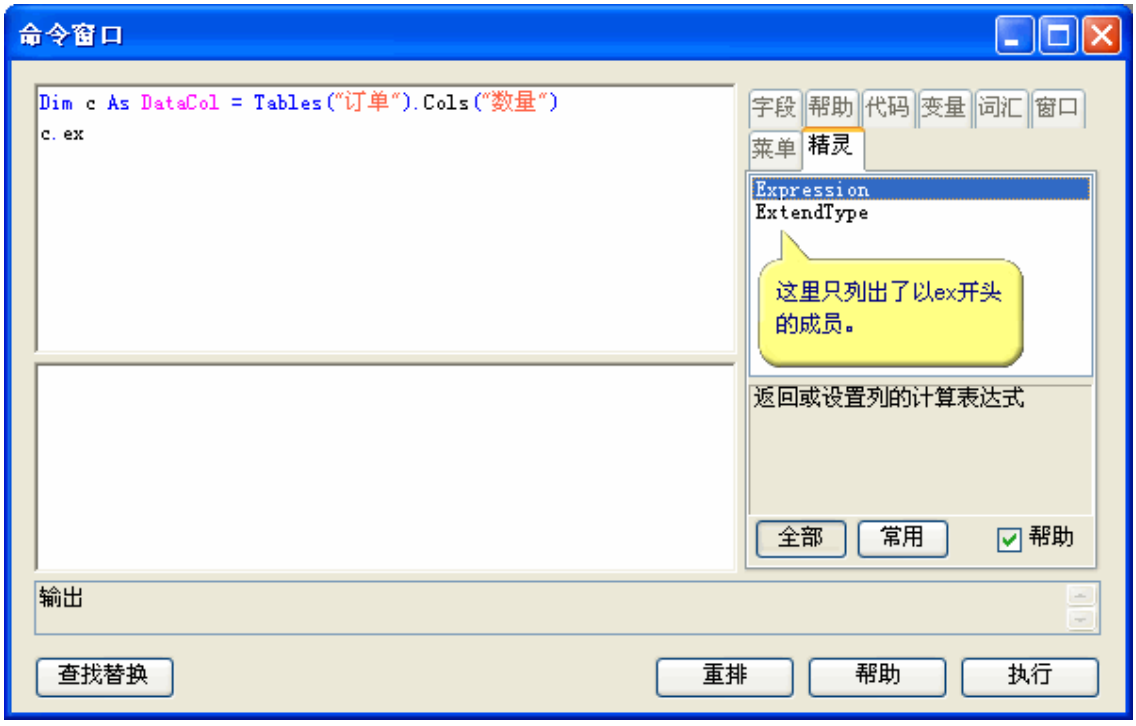
然后按 Tab，会列出 Col 的所有成员（属性和方法）供选择，按上下箭头选择合适的成员，按回车即可自动输入。

如果你输入：

Dim c As DataCol = Tables("订单").Cols("数量")

c.ex

然后按回车，会自动列出以 ex 开头的成员供选择：



选择 ExtendType 后按回车，会自动输入“ExtendType”，并显示此属性的帮助：

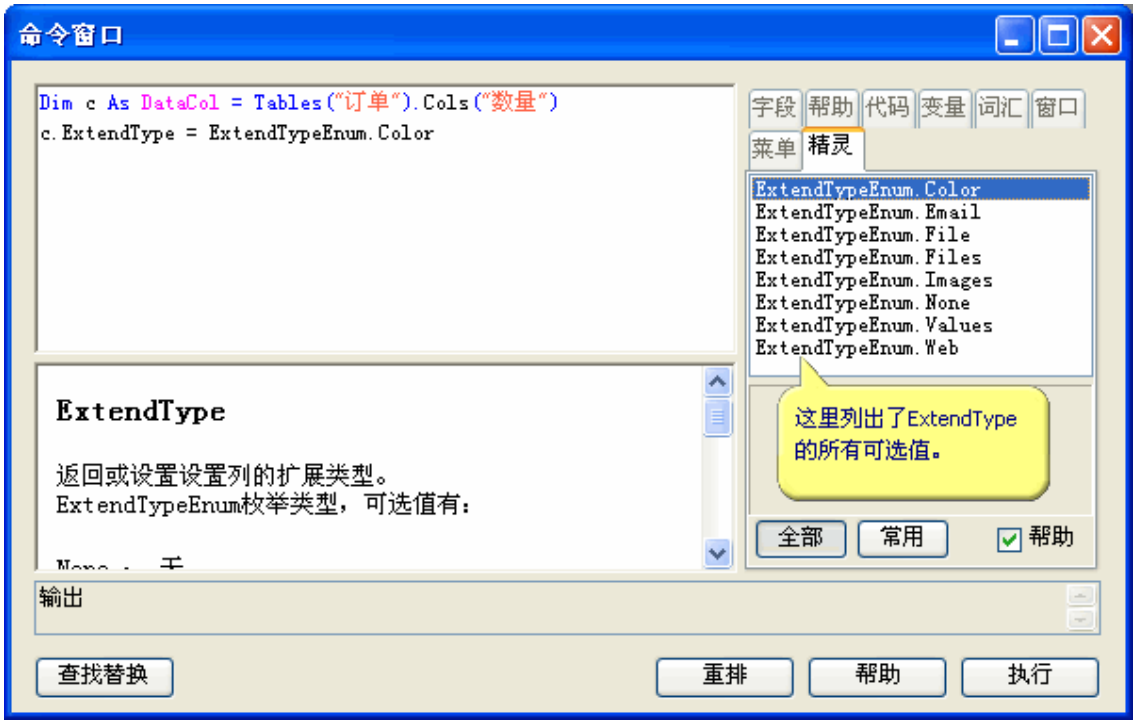


继续输入一个等于号：

Dim c As DataCol = Tables("订单").Cols("数量")

c.ExtendType =

然后按 Tab，会列出 ExtendType 的全部可选值供选择，从中选择一个，按回车即可：



通过上面的介绍，可以看出，在 Foxtable 中输入代码是很轻松的，并不需要你精确地记住什么。

关闭自动帮助

如果需要，可以关闭代码编辑器自动显示帮助的特性，只需将上图中帮助选项的选中标记去掉即可。

4.4.2 事件编程

4.4.2.1 基础知识

4.4.2.1.1 设置事件代码

对于 Foxtable 来说，所谓编程，就是给各种事件编写合适的代码，从这句话你应能体会到事件编程在整个开发篇的 份量。

实践出真知，接下来我们会接触很多和事件编程相关的演示代码，如果条件许可的话，最好都能实际运行测试一下。

Foxtable 有两种类型的事件：

一组事件是针对项目的，例如 AfterOpenProject 事件在打开项目后执行，BeforeCloseProject 事件在关闭项目前执行，这样的事件，我们称之为项目事件。

另一组事件是针对表的，例如 DataColChanged 事件在某列的内容发生变化后执行，AfterEdit 是编辑某单元格的内容后执行，这样的事件，我们称之为表事件。

1、如果要设置表事件，在菜单的“数据表”功能区，单击“表属性”按钮：



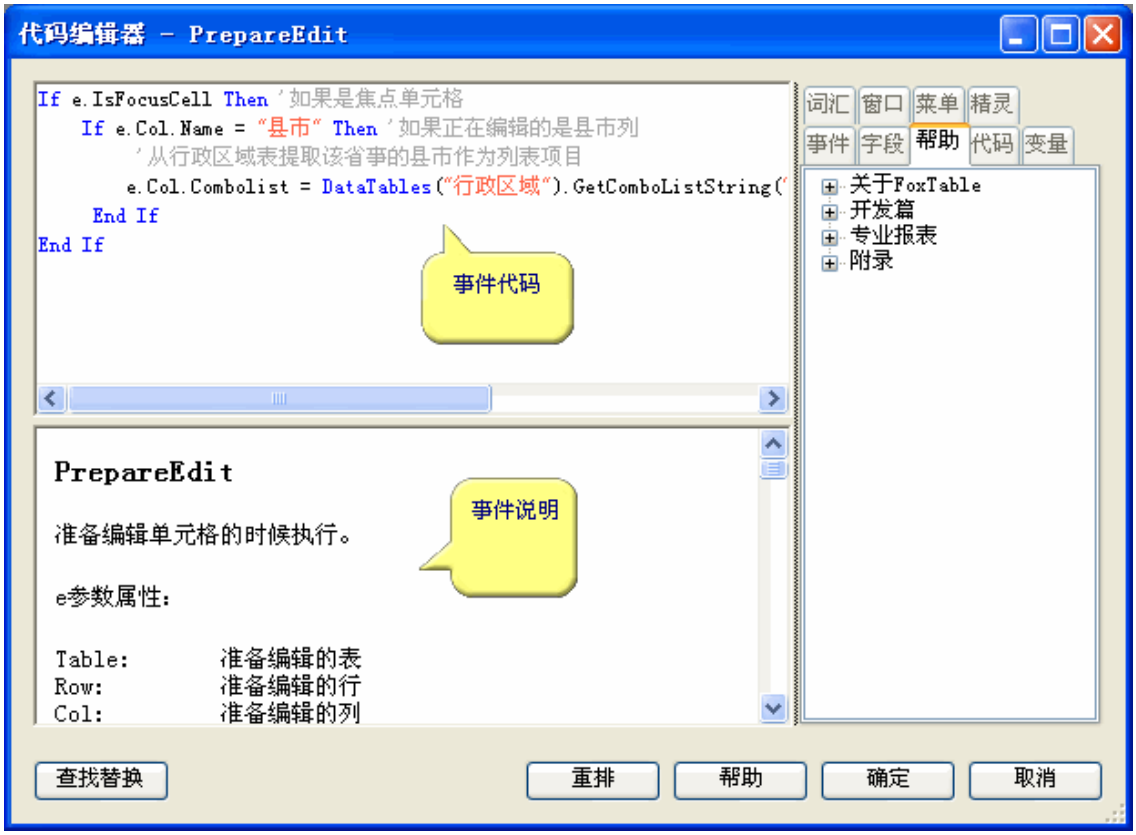
2、如果要设置项目事件，在菜单的“管理项目”功能区，单击“项目属性”按钮：



3、在属性设置窗口选择“事件”页面：



3、下面是代码编辑窗口，每次打开代码编辑器编辑事件代码，都会自动显示该事件的说明：



4.4.2.1.2 事件的 e 参数(必读)

发生任何一个事件，总是会有一些和事件相关的信息，需要通知到系统。
这些信息是通过事件的 e 参数来获得的，几乎所有的事件都有一个 e 参数，不同事件的 e 参数属性是不同的。
例如 DataColChanged 事件，它会在某一列的内容发生变化后触发，其 e 参数就包括三个属性：
e.DataTable：发生变化的表
e.DataCol: 发生变化的列
e.DataRow: 发生变化的行
通过这三个属性，我们能够知道变化发生的具体位置，即在 e.DataTable(表)的 e.DataRow(行)的 e.DataCol(列)发生了变化。
如果没有 e 参数，DataColChanged 事件也就没有了意义。
同样是表事件，e 参数属性也会有所不同的，例如 AfterEdit 事件，这个事件在编辑单元格后触发，它的 e 参数属性有：
e.Table: 编辑的表
e.Row: 编辑的行
e.Col: 编辑的列
这是因为同一个 DataTable 会有多个 Table，我们无法直接编辑 DataTable 的，而只能是编辑其中一个 Table，所以 AfterEdit 的 e 参数属性需要告诉系统，编辑的是哪一个 Table、哪一个 Row、那一个 Col。
我们知道 Table 有一个 DataTable 属性，用于返回该 Table 所属的 DataTable，同样 Row 通过 DataRow 属性，返回该 Row 所属的 DataRow，Col 通过 DataCol 属性，返回该 Col 所属的 DataCol，这样在 AfterEdit 事件中，同样可以快速获得 DataTable 层级的对象：

e.Table.DataTable

e.Row.DataRow

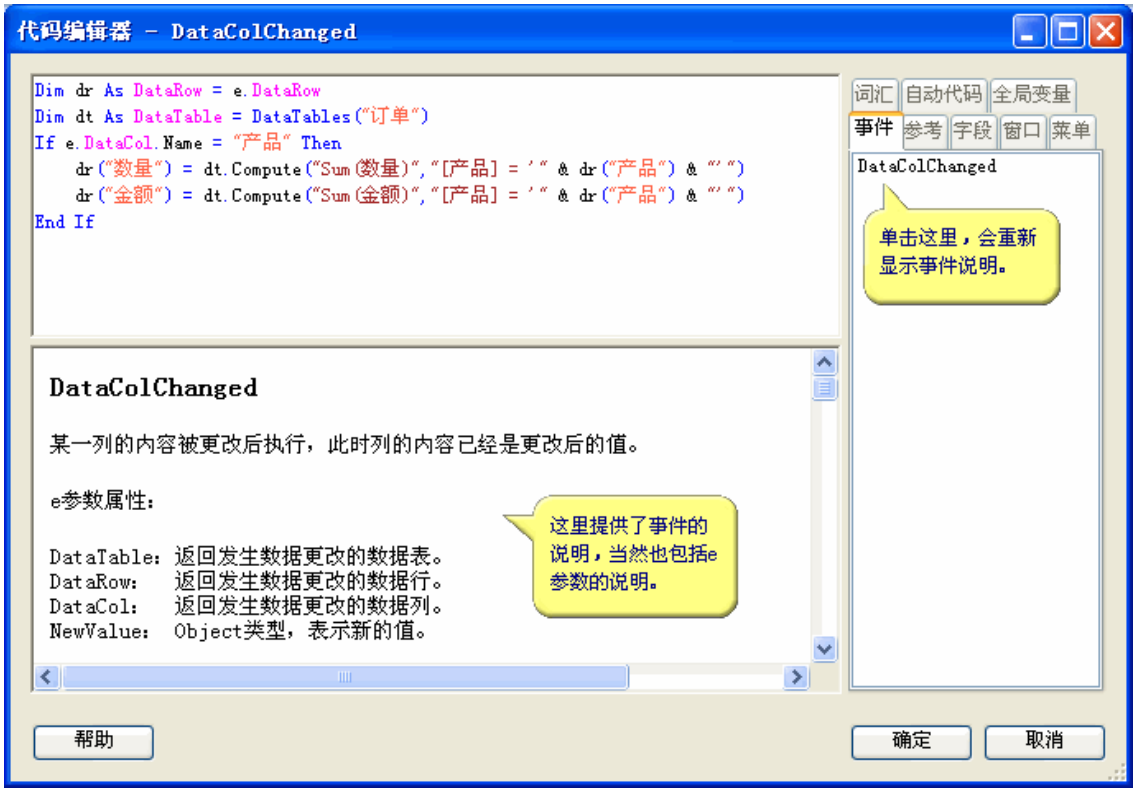
e.Col.DataCol

所以实际上，AfterEdit 事件的 e 参数提供了更多的信息，可以追踪到具体的 Table。

某些事件的 e 参数提供了 DataTable 层级的对象，而某些事件的 e 参数提供的却是 Table 层级的对象，有的用户会对此感到很疑惑，希望上面的说明能够帮助大家“释疑”。

此外也有个别事件没有 e 参数，例如项目事件 AfterOpenProject，此事件在打开项目后执行，显然这样的事件并不需要通过 e 参数来传递什么信息的。

在编辑事件代码的时候，代码编辑器会提供该事件的说明，当然也包括 e 参数的说明。



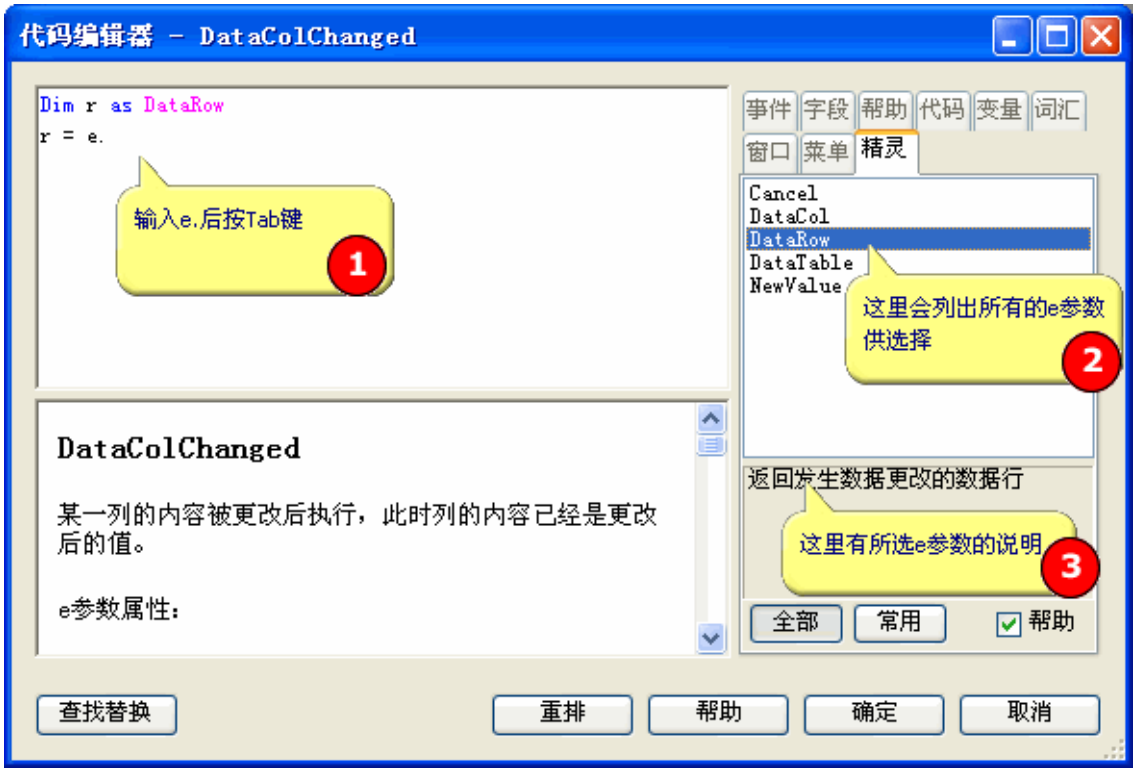
4.4.2.1.3 自动输入 e 参数

不同的事件 e 参数是不同的，除了通过事件说明获得 e 参数信息外，代码编辑器还可以自动输入 e 参数。

在代码编辑器中输入：

e.

接着按 Tab 键，既可自动列出当前事件的 e 参数供选择，按上下箭头键选择 e 参数，然后按回车键，既可自动输入所选的 e 参数。



再配合自动成员列表和输入功能，以及 Foxtable 特有的动态帮助功能，编写代码将无比的轻松：





轻松编写代码的基础是熟练掌握了基本概念，以及各事件的作用原理。

4.4.2.1.4 全局变量

4.4.2.1.4.1 全局变量

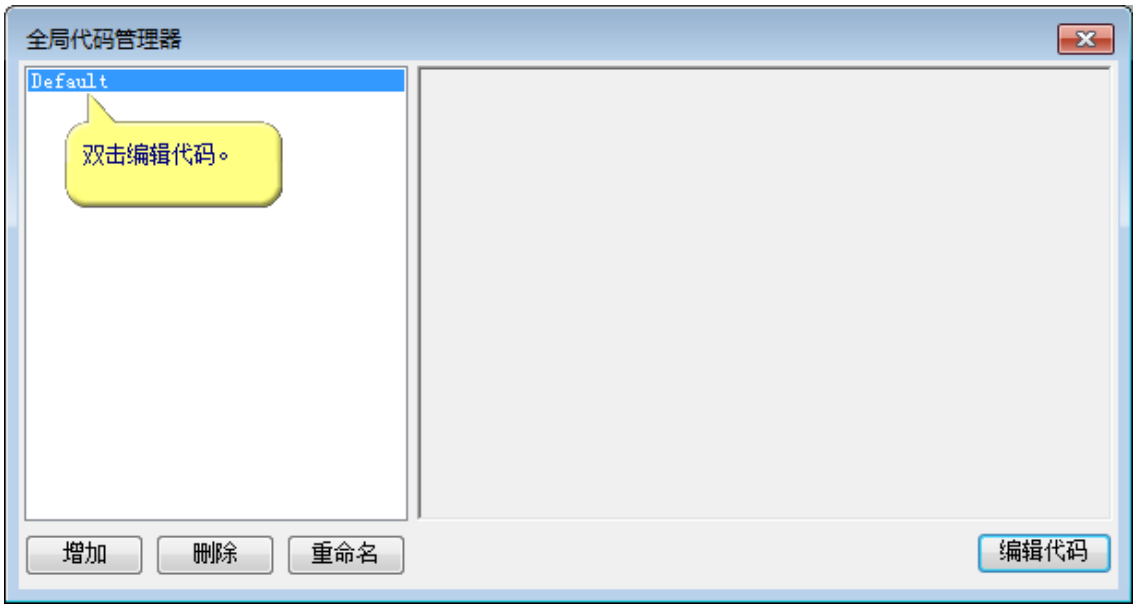
变量有自己的作用范围，在 A 事件中定义的变量，在 B 事件或者命令窗口是不能访问的。
但是在不同的事件中，传递和共享数据，是一个经常要面对的任务，这就需要使用全局变量。
全局变量能够在任何事件代码，包括命令窗口中使用。
Foxtable 提供了两种类型的全局变量，下面一一介绍。

4.4.2.1.4.2 Public 变量

在菜单的“管理项目”功能区，单击下面的按钮:



出现全局代码管理窗口：

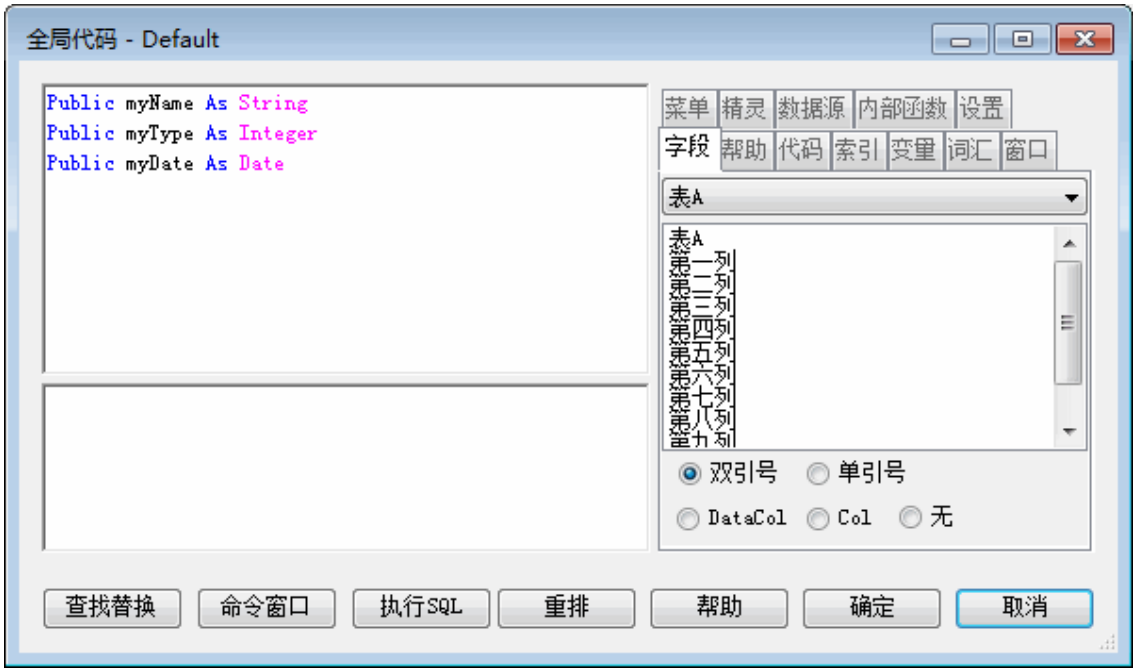


全局代码可以有多个模块，默认已经有一个名为"Default"的模块，双击之即可编写代码。

在全局代码中，定义 Public 变量的语法是：

Public 变量名称 As 变量类型

例如：



在定义 Public 变量的时候，可以同时给变量赋值：

```
Public myName As String = "吴晓晓"
Public myType As Integer = 1
Public myDate as Date = Date.Today()
```

Public 变量一旦定义，就可以在任何地方使用，例如你可以在命令窗口修改或取得 Public 变量的值

```
myName = "吴萧萧"
Return MyName
```

提示：Public 变量只能在全局代码中定义，其它任何地方都不能定义，只能使用。

注意事项：

在实际使用的时候，我发现一个非常奇怪的问题，Public 变量的长度至少应该为 3 个字符，例如我在全局代码中定义三个变量：

```
Public a As Integer
Public ab As Integer
Public abc As Integer
```

其中变量 a 和 ab，在其它地方无法引用，而变量 abc 可以正常引用。

4.4.2.1.4.3 Var 变量

Public 变量的定义和使用都很简单，也能很好地完成在不同事件代码中传递和共享数据的任务。
Foxtable 提供了另一种类型的全局变量：Var 变量。
Var 变量在定义和使用上，都不如 Public 变量简洁，但是 Var 变量有自己的优势：

1、可以在 Excel 报表中引用
2、可以绑定到窗口的控件
3、可以通过事件跟踪其值的变化

除非有上述三点的需要，否则应该使用 Public 变量，而不是 Var 变量。
有关如何在 Excel 报表中引用 Var 变量，以及如何在窗口中将控件绑定到 Var 变量，我们会在相关章节介绍。

管理 Var 变量

在菜单的管理项目功能区，单击下图所示的按钮：



可以管理 Var 变量：



通过代码增加 Var 变量

也可以在运行过程中，通过代码增加 Var 变量。

语法

Vars.Add(VarName,GetType(Type))
Vars.Add(VarName,GetType(Type),Value)

参数

VarName：变量名称
Type：　变量类型，需要配合 GetType 关键字来获得数据类型，例如 GetType(String)表示字符型，GetType(Boolean)表示逻辑型。
Value：　可选参数，用于设置变量的初始值。

例如：

```
Vars.Add("Mark1",GetType(Boolean),False)
```

表示增加一个逻辑型 Var 变量 Mark1，初始值为 False

```
Vars.Add("起始日期",GetType(Date),#12/31/2007#)
```

表示增加一个日期型 Var 变量，名称为“起始日期”，初始值为#12/31/2007#

引用 Var 变量

通过上述方法定义 Var 变量后， 就可以在代码中引用 var 变量了。

引用 Var 变量的语法为：

Vars(VarName)

VarName：变量名称。

例如：

```
If Vars("Mark1") = False Then
    Vars("Mark1") = True
```

```
    Vars("StartDate") = #8/21/2002#
    Output.Show(Vars("StartDate"))
End If
```

并非一定要事先定义

在默认情况下，Var 变量并非一定要事先定义，当我们向一个 Var 变量赋值的时候，系统会检查此 Var 变量是否已经定义，如果没有定义，会根据值的类型，自动增加一个同类型的变量。

例如执行下面的代码：

```
Vars("abc") = 99
```

如果我们事先并没有通过菜单定义 abc 变量，那么系统会自动增加一个名为 abc 的整数型变量，并将其值设置为 99。

变量一旦增加，就不能再改变其类型，例如我们可以改变 abc 变量的值:

```
Vars("abc") = 100
Output.Show(Vars("abc"))
```

但是如果将其他类型，例如一个字符型的数据写入变量 abc，则会出错：

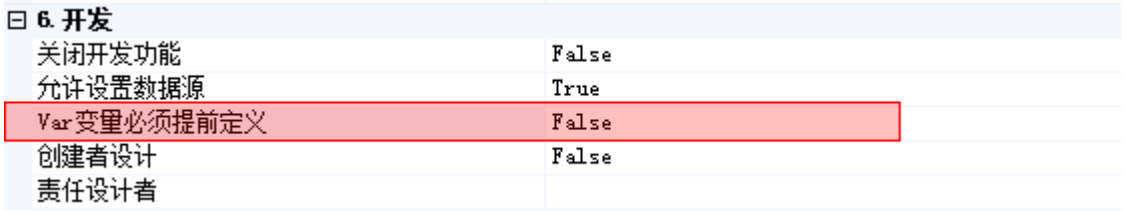
```
Vars("abc") = "Foxtable" '会出错
```

需要注意的是，如果要将某个 Var 绑定到窗口控件，则必须事先通过菜单定义此变量。

强行要求 Var 变量必须事先定义

Var 无需定义，可以直接使用，在编码方面更灵活，但是如果代码中出现了变量名称拼写错误，将没有办法自动给予提示。

如果要求 Var 变量必须事先定义才能使用，可以在项目属性中，将"Var 变量必须提前定义"设置为 True：



4.4.2.1.5 测试事件代码

不同的事件，e 参数不同，所以大多数时候，A 事件的代码，并不能用在 B 事件中。

一些初学者甚至直接将事件代码复制到命令窗口执行，这是更加不允许的，因为命令窗口根本就没有 e 参数。

要测试事件代码，必须人为地触发事件。

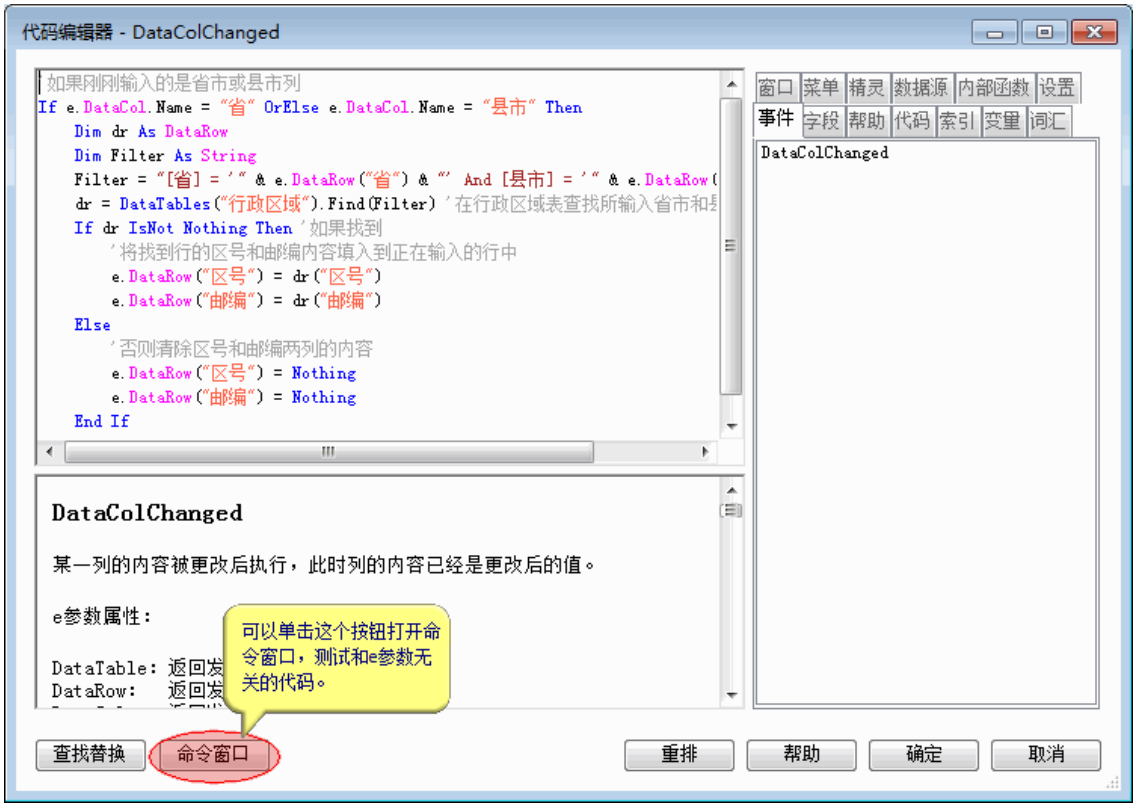
例如事件 DataRowAdded 在增加行后执行，我们将该事件代码设为：

```
e.DataRow("日期") = Date.Today
```

这样可以自动将新增行的日期列内容设为当天的日期。

要测试这段代码，必须增加一行，看看这行的日期值是否会自动设置为当天日期，别无他法。

如果需要在编写事件代码的过程中，测试某段和 e 参数无关的代码，可以单击“命令窗口”按钮，打开命令窗口测试：



4.4.2.2 表事件

4.4.2.2.1 两种类型的表事件

表事件有两种，一种是针对 DataTable 的，一种是针对 Table 的。

我们用一个简单的例子来说明为什么会有两种表事件。

假定产品表和订单表通过产品编号建立了关联，那么对于订单表，会有两个 Table，分别是：

Tables("订单")

Tables("产品.订单")

前者是主表，显示所有订单，后者是关联表，在产品表选择某产品时，此关联表显示该产品的全部订单。

当然，订单表的 DataTable 还是一个：

DataTables("订单")

Foxtable 提供了一个表事件 PrepareEdit，这个事件在用户编辑某一个单元格之前触发；用户可能在主表 Tables("订单")中编辑数据，也可能在关联表 Tables("产品.订单")中编辑数据，显然 PrepareEdit 事件必须能告诉我们用户到底是在哪一个 Table 中编辑数据，以及编辑的是哪一列，哪一行；所以 PrepareEdit 事件是针对 Table 的，该事件的 e 参数提供了以下信息：

Table: 准备编辑的表

Row: 准备编辑的行

Col: 准备编辑的列

Cancel: 逻辑型，默认为 False，设为 True 取消编辑。

实际上，DataTable 是不可见的，我们不可能去编辑它，我们只能编辑看得见的 Table，所以

PrepareEdit 事件只能是针对 Table 的，而不可能针对 DataTable。

表中某一列的内容发生变化后会触发 DataColChanged 事件，这个事件是针对 DataTable 的，因为存放数据的是 DataTable，不管用户在主表 Tables("订单")中编辑数据，还是在关联表 Tables("产品.订单")中编辑数据，或者因为代码导致订单表中的数据发生变化，最终影响的都是 DataTables("订单")中的数据，Table 只是从 DataTable 中提取数据呈现给用户而已；我们不需要知道导致列内容发生变化的原因，只需要知道发生变化的是哪个表的哪一行的哪一列，所以 DataColChanged 事件的 e 参数属性提供了以下信息：

DataTable：发生变化的表

DataCol:　发生变化的列

DataRow:　发生变化的行

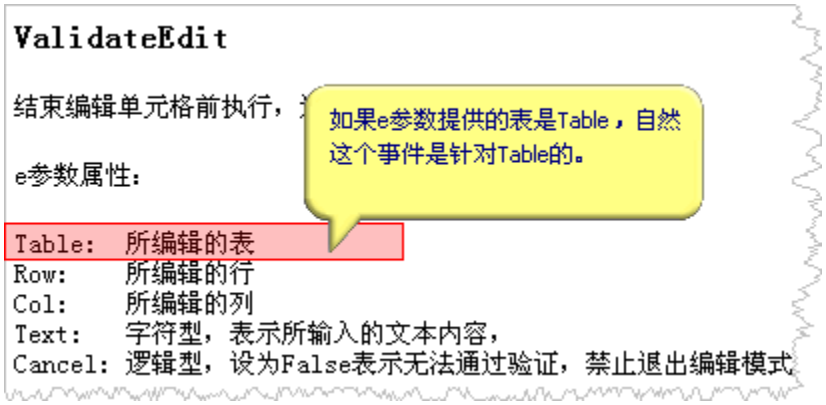
如果你还是不能理解，那么我们看看如果 DataColChanged 事件针对 Table 的时候，会带来怎样的后果；　假定我们在命令窗口执行下面的代码：

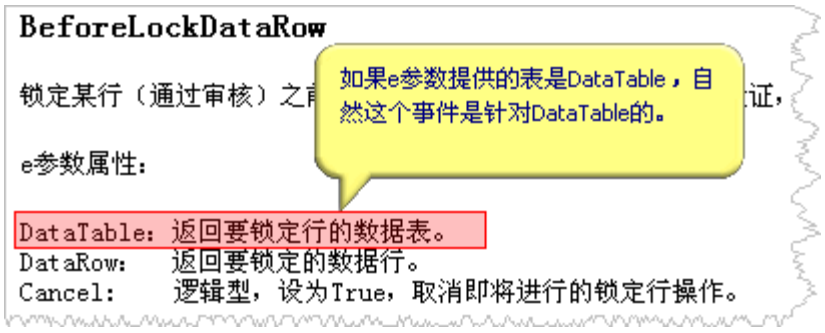
```
Dim r As DataRow
r = DataTables("订单").DataRows(0)
r("数量") = 100
```

上面的代码直接通过 DataRow 设置了数量列的内容，因为一个 DataTable 对应有多个 Table，同样一个 DataRow 可能会出现在多个 Table 中，从而对应有多个 Row，如果 DataColChanged 是针对 Table 的，那么此时应该针对哪一个 Table 和 Row 来触发此事件呢？如果极端一点，我们在 Table 中进行了筛选，而 某一行不符合筛选条件，没有出现在任何 Table 中，那么使用代码修改此行的值后，又该如何触发 DataColChanged 事件呢？

显然 DataColChanged 这样的事件，只有针对 DataTable 才合理，上面的问题才不复存在。

实际学习和应用的时候，判断一个事件是针对 Table 的还是针对 DataTable，是很简单直观的，我们只需看其 e 参数属性即可：





4.4.2.2.2 DataColChanging

某一列的内容发生变化前执行，此时新值还没有写入表中，列的内容还是更改前的值。
利用此事件，我们可以在某列内容发生变化之前进行拦截判断，如果不符合特定的条件，则取消此次变化，或者重新指定一个值。
e 参数属性：

DataTable： 返回发生数据更改的数据表。
DataRow： 返回发生数据更改的数据行。
DataCol： 返回数据更改的数据列。
NewValue： Object 类型，表示新的值
OldValue： Object 类型，表示旧的值
Cancel： 逻辑(Boolean)类型，设为 True 取消更改。

示例一
例如在订单表中，要求折扣列的值不能超过 0.15，可以在 DataColChanging 事件中设置如下代码：

```
If e.DataCol.Name = "折扣" Then '如果是折扣列的内容发生变化。
    If e.NewValue > 0.15 Then '如果新值大于 0.15。
        e.Cancel = True '取消变动。
    End If
End If
```

上面的代码在折扣超出 0.15 后，取消此次变动；如果希望在折扣超出 0.15 后，自动改为 0.15，代码如下：

```
If e.DataCol.Name = "折扣" Then '如果是折扣列的内容发生变化。
    If e.NewValue > 0.15 Then '如果输入的内容大于 0.15。
        e.NewValue = 0.15 '那么改为 0.15。
    End If
End If
```

示例二
DataColChanging 事件触发时，列中的值还没有更改，还是原来的值。这样我们就可以对新值和旧值进行比较。
例如假定表中有一列为“等级”，禁止将等级"A"调整为等级"D"，只需将 DataColChanging 事件代码设为：

```
If e.DataCol.Name = "等级" Then
    If e.OldValue = "A" Then '如果原来的等级是 A
        If e.NewValue = "D" Then '且新的等级为 D
            e.Cancel = True '则取消更改
        End If
    End If
End If
```

在上面的代码中，e.OldVaue 等效于 e.DataRow("等级")，因为 DataColChanging 事件触发的时候，新的值还没有写入到表中。

所以上面的代码也可以改写为：

```
If e.DataCol.Name = "等级" Then
    If e.DataRow("等级") = "A" Then '如果原来的等级是 A
        If e.NewValue = "D" Then '且新的等级为 D
            e.Cancel = True '则取消更改
        End If
    End If
End If
```

示例三

通过更改 NewValue 参数的值，我们可以实现代码输入，例如：

```
If e.DataCol.Name = "产品" Then '如果是产品列的内容发生变化。
    If e.NewValue = "tv" Then '如果新值是"tv"。
        e.NewValue = "电视机" '那么将"tv"替换为"电视机"
    ElseIf e.NewValue = "tel" Then '如果新值是"tel"
        e.NewValue = "电话机" '那么将"tel"替换为"电话机"
    End If
End If
```

在 DataColChanging 事件中设置上述代码后，如果你在产品列输入 tv，将自动替换为电视机，如果输入 tel， 将自动替换为电话机。

示例四

我们知道，列有禁止输入重复内容的属性，不过这个属性有局限，就是只有直接在表格中输入的时候才有效，如果你通过窗口输入或者是通过代码设置，同样可以输入重复内容。

我们可以在 DataColChanging 事件中设置代码，使得不管通过任何方式修改列的内容，都不能输入重复值，例如希望订单号列不能输入重复值：

```
If e.DataCol.Name = "订单号" Then
    Dim dr As DataRow
    dr = e.DataTable.Find("订单号 = '" & e.NewValue & "'")
    If dr IsNot Nothing Then
        MessageBox.Show("此订单号已经存在!")
        e.Cancel = True
    End If
End If
```

示例五

如果要允许修改订单号，但禁止删除订单号，也就是说订单号列内容不允许为空，可以将 DataColChanging 事件代码设置为：

```
If e.DataCol.Name = "订单号" Then
    If e.NewValue Is Nothing Then
        MessageBox.Show("订单号不允许为空!")
        e.Cancel = True
    End If
End If
```

前面提到 DataColChanging 事件触发的时候，新的值还没有写入表中，所以如果要判断新的值是否为空，只能用：e.NewValue Is Nothing。

如果要判断旧值是否为空，则既可用：e.OldValue Is Nothing，也可用 IsNull 方法判断：e.DataRow.IsNull("列名")。

切不可用 e.DataRow("列名") Is Nothing 来判断旧值是否为空，因为对于数值列，如果值为空，e.DataRow("列名")返回的是 0，如果是日期列，则返回#01/01/0001#，显然这都不是空值。

示例六

显然一个表中会存在多种判断，可以将这些代码简单地组合在一起：

```
If e.DataCol.Name = "折扣" Then
    If e.NewValue > 0.15 Then
        e.Cancel = True
    End If
End If
If e.DataCol.Name = "产品" Then
    If e.NewValue = "tv" Then
        e.NewValue = "电视机"
    ElseIf e.NewValue = "tel"
        e.NewValue = "电话机"
    End If
End If
If e.DataCol.Name = "工号" Then
    Dim dr As DataRow
    dr = e.DataTable.Find("工号 = '" & e.NewValue & "'")
    If dr IsNot Nothing Then
        MessageBox.Show("此工号已经存在!")
        e.Cancel = True
    End If
End If
```

也可以用 If... ElseIf...End If 改写，使得代码看上去有条理一点：

```
If e.DataCol.Name = "折扣" Then
    If e.NewValue > 0.15 Then
        e.Cancel = True
    End If
ElseIf e.DataCol.Name = "产品" Then
    If e.NewValue = "tv" Then
        e.NewValue = "电视机"
    ElseIf e.NewValue = "tel"
        e.NewValue = "电话机"
    End If
ElseIf e.DataCol.Name = "工号" Then
    Dim dr As DataRow
    dr = e.DataTable.Find("工号 = '" & e.NewValue & "'")
    If dr IsNot Nothing Then
        MessageBox.Show("此工号已经存在!")
        e.Cancel = True
    End If
End If
```

或者用 Select Case 语句改写：

```
Select Case e.DataCol.Name
    Case "折扣"
        If e.NewValue > 0.15 Then
            e.Cancel = True
        End If
    Case "产品"
        If e.NewValue = "tv" Then
            e.NewValue = "电视机"
        ElseIf e.NewValue = "tel"
            e.NewValue = "电话机"
        End If
    Case "工号"
        Dim dr As DataRow
        dr = e.DataTable.Find("工号 = '" & e.NewValue & "'")
        If dr IsNot Nothing Then
            MessageBox.Show("此工号已经存在!")
            e.Cancel = True
        End If
End Select
```

我个人偏好 Select Case 语句，因为看上去更清晰简洁一些。

4.4.2.2.3 DataColChanged

某一列的内容被更改后执行，此时列的内容已经是更改后的值。

e 参数属性：

DataTable：返回发生数据更改的数据表。

DataRow： 返回发生数据更改的数据行。

DataCol： 返回发生数据更改的数据列。

NewValue： Object 类型，表示新的值。

OldValue： Object 类型，表示旧的值

示例一

假定在客户消费数据表中，有一逻辑型数据列，列名为“已结帐”，我们希望对于已结帐的行，能够自动通过审核(锁定)，以免被误编辑。

为实现这个目标，只需在 DataColChanged 事件中设置如下代码即可：

```
If e.DataCol.Name = "已结帐" Then '如果是已结帐列的内容变动
    If e.NewValue = True Then '而且变动后的值是 True(已勾选)
        e.DataRow.Locked = True '那么锁定此行
    End If
End If
```

示例二

列与列之间的计算，我们通常用表达式列来实现，但不是所有的计算都可以通过表达式来完成的。

例如员工表中有身份证号码、出生日期列，出生日期根据身份证号码计算得出。

为了自动计算出生日期，可以在员工表的 DataColChanged 事件中加入下面的代码：

```
If e.DataCol.Name = "身份证号码" Then '如果更改的是身份证号码列
    If e.DataRow.IsNull("身份证号码") Then '身份证号码是否为空
        e.DataRow("出生日期") = Nothing '如果为空,则清除出生日期
    Else
        '否则从身份证号码列中提取出生日期
        e.DataRow("出生日期") = ReadBirthday(e.DataRow("身份证号码"))
    End If
End If
```

如果表中已经输入数据，那么在加入上述代码后，选择身份证号码列，然后在“数据表”功能区“列相关”功能组，单击下面的“重置”按钮：



即可计算出所有行的出生日期，今后新增数据或者修改身份证号码，出生日期可以自动计算得出，无需人工干预。

上面代码中，e.DataRow("身份证号码")和 e.NewValue 实际上是相等的，因为 DataColChanged 执行的时候，新值已经写入到表中。

所以下面这行代码：

```
e.DataRow("出生日期") = ReadBirthday(e.DataRow("身份证号码"))
```

等效于：

```
e.DataRow("出生日期") = ReadBirthday(e.NewValue)
```

同样，因为新的值已经写入表中，所以如果要判断旧的值是否为空，只能用：e.OldValue Is Nothing。

如果要判断新值是否为空，则既可用：e.NewValue Is Nothing，也可用 IsNull 方法 判断 ，如上面的： e.DataRow.IsNull("身份证号码")。

切不可用 e.DataRow("列名") Is Nothing 来判断新值是否为空，因为对于数值列，如果值为空，e.DataRow("列名")返回的是 0，如果是日期列，则返回#01/01/0001#，显然这都不是空值。

示例三

可以肯定的是，一个表的列与列之间的计算关系往往不只一个。

假定一个表同时包括示例一和示例二的计算关系，你可以将这些代码简单地复制在一起，也可以用 If... ElseIf...End If 改写：

```
If e.DataCol.Name = "已结帐" Then
    If e.NewValue = True Then
        e.DataRow.Locked = True
    End If
ElseIf e.DataCol.Name = "身份证号码" Then
    If e.DataRow.IsNull("身份证号码") Then
        e.DataRow("出生日期") = Nothing
    Else
        e.DataRow("出生日期") = ReadBirthday(e.NewValue)
    End If
```

```
End If
```

或者采用 Select Case 语句进行条件分支判断：

```
Select Case e.DataCol.Name
    Case "已结帐"
        If e.NewValue = True Then
            e.DataRow.Locked = True
        End If
    Case "身份证号码"
        If e.DataRow.IsNull("身份证号码") Then
            e.DataRow("出生日期") = Nothing
        Else
            e.DataRow("出生日期") = ReadBirthday(e.NewValue)
        End If
End Select
```

我个人偏好 Select Case 语句，因为看上去更清晰简洁一些。

一些逻辑较为复杂的关系，可能需要多个 Select Case 语句才能表述：

```
Select Case e.DataCol.Name
    Case "A 列"
        '代码一
    Case "B 列"
        '代码二
    Case "E 列"
        '代码三
End Select
Select Case e.DataCol.Name
    Case "A 列","B 列","C 列"
        '代码四
    Case "E 列", "F 列"
        '代码五
End Select
```

上面这段代码有两个 Select Case 语句，如果修改的是 A 列，则执行代码一和代码四，如果修改的是 E 列，则执行代码三和代码 五，如果修改的是 F 列，则只执行代码五。

4.4.2.2.4 DataRowAdding

增加一行时执行，此时新增行(DataRow)已经创建，但并未真正增加到 DataTable 中，通常在此处对新增行进行默认值的设置。

e 参数属性：

DataTable：返回增加行的数据表。

DataRow： 返回新增加的行。

示例一

例如希望新增行的日期设为当前系统日期，姓名设为当前登录用户名，可以将 DataRowAdding 事件的代码设置为：

```
e.DataRow("日期") = Date.Today()
```

```
e.DataRow("姓名") = User.Name
```

示例二

假定表中有一个编号列，希望新增行的时候，编号列能够自动加 1，可以将 DataRowAdding 事件的代码设置为：

```
e.DataRow("编号") = e.DataTable.Compute("Max(编号)") + 1
```

这个例子其实没有什么意义的，对于外部表使用自动增量主键列，对于内部表使用表达式列引用内部编号列“_Identify”,是更好的选择。

示例三

如果希望新增行能够自动从最后一行复制全部内容，可以将 DataRowAdding 事件的代码设置为：

```
If e.DataTable.DataRows.Count = 0 Then '如果是一个空表
    Return '那么返回
End If
Dim dr As DataRow = e.DataTable.DataRows(e.DataTable.DataRows.Count - 1) '获得最后一行
For Each dc As DataCol In e.DataTable.DataCols
    e.DataRow(dc.Name) = dr(dc.Name) '将最后一行的内容复制到新增行中.
Next
```

示例四

上面的代码是复制所有列，如果只是复制部分列，可以参考下面的代码：

```
If e.DataTable.DataRows.Count = 0 Then '如果是一个空表
    Return '那么返回
End If
Dim ColNames As String() = {"日期","客户","雇员"} '定义一个数组,保存复制列的名称
Dim dr As DataRow = e.DataTable.DataRows(e.DataTable.DataRows.Count - 1) '获得最后一行
For Each ColName As String in ColNames
    e.DataRow(ColName) = dr(ColName) '将最后一行的内容复制到新增行中.
Next
```

4.4.2.2.5 DataRowAdded

增加一行后执行，此时新增行(DataRow)已经真正增加到 DataTable 中。

e 参数属性：

DataTable：返回增加行的数据表.

DataRow： 返回新增加的行。

示例

我们知道每次增加一行，自动增量列的值就会增加 1，不过这个值是临时的，在保存的时候，自动增量列才会生成真正的值。

如果需要在任何时候，自动增量列的值都是真实的，而不是临时的，可以在 DataRowAdded 事件中设置代码：

```
e.DataTable.Save()
```

这样每次增加行后，就会自动保存表，使得自动增量列获得真实的值。

4.4.2.2.6 BeforeDeleteDataRow

删除一行之前执行，可以在此处进行权限或逻辑验证，如果通不过验证，则撤销此次删除行操作。
e 参数属性：
DataTable：返回要删除行的数据表。
DataRow： 返回要删除的数据行。
Cancel: 逻辑型，设为 True，取消删除行操作。
示例
DataTable 和 Table 都有 AllowDelete 属性，用于决定是否允许用户删除行，但是该属性是一刀切，而利用 BeforeDeleteDataRow 事件，你可以在删除某一行之前进行复杂的逻辑判断，来决定是否应该删除这一行。
例如公司规定，对于已经结帐的行，只有“张经理”才能删除，你可以在 BeforeDeleteDataRow 设置如下代码：

```
'如果当前用户不是张经理，而且即将被删除的行已经结帐，那么取消删除。
If User.Name <> "张经理" And e.DataRow("结帐") = True Then
    MessageBox.Show("领导说了：已经结帐的行，只有张经理才能删除，别乱来！", "提示")
    e.Cancel = True
End If
```

4.4.2.2.7 DataRowDeleting

正在删除一行的时候执行，这时行还未被删除。
e 参数属性：
DataTable：返回要删除行的数据表。
DataRow： 返回要删除的数据行。
示例
假定订单表和订单明细表已经建立关联，希望用户在订单明细表删除某订单的最后一个明细时，能自动删除此订单，可以将 DataRowDeleting 事件代码设置为：

```
Dim pr As DataRow = e.DataRow.GetParentRow("订单") '获得此明细对应的订单
If pr IsNot  Nothing Then '如果有对应的订单
    If pr.GetChildRows("订单明细").count  = 1 Then '且这个订单的明细只有一个了
        pr.Delete '删除这个订单
    End If
End If
```

4.4.2.2.8 PrepareEdit

准备编辑单元格的时候执行。
e 参数属性：
Table: 准备编辑的表
Row: 准备编辑的行

Col: 准备编辑的列

IsFocusCell: 逻辑型，是否是焦点单元格

Cancel: 逻辑型，默认为 False，设为 True 取消编辑。

重要提示：千万不要在 PrepareEdit 事件显示 Messagebox 这样的对话框，因为每次进入单元格就会出现提示，这样会导致死循环的出现。

示例一

我们知道，被锁定的行是禁止用户编辑的，但是系统提供的锁定功能，非常的不灵活，因为一旦锁定，任何人都不能编辑。

你可能需要更灵活的锁定功能，例如，可能希望某一状态下的行，对于特定的人员是可以编辑的，而对于其他人是禁止编辑的；也有一种情况，对于某一列，您可能希望得到授权的人才能编辑 。

我们先来看看第一种情况如何实现，假定在一个客户消费数据表，有一逻辑列，名为“已结帐”；如果某一行的已结帐列被勾选，那么我们除了经理之外，员工不可以再修改该行。

实现的代码很简单，在该表的 PrepareEdit 事件中输入：

```
If e.Row("已结帐") = True AndAlso User.Group <> "经理" Then
    e.Cancel = True
End If
```

User.Group 表示登录用户的分组，在用户管理中，可以给用户设置不同的组别。

再看第二种情况，假定消费表中还有一个折扣列，我们希望只有经理级别的人员，才能给客户打折，也就是只有经理级别的操作人员才能在该列中输入或修改数据，实现代码同样很简单，在该表的 PrepareEdit 事件中输入：

```
If e.Col.Name = "折扣" AndAlso User.Group <> "经理" Then
    e.Cancel = True
End If
```

还来一个例子，我们知道易表有一个锁定非空内容的功能，也就是只有空白单元格才能输入数据，一旦输入就不能修改，在 Foxtable 我们也可以很轻松地实现这个功能，例如希望锁定“标注”列的非空内容：

```
If e.Col.Name = "标注" AndAlso e.Row.IsNull("标注") = False Then
    e.Cancel = True
End If
```

从上面的演示可以看出，利用 PrepareEdit 事件，可以带来强大的编辑控制功能，远不是系统的审核功能所能比拟的。

在表中进行剪切、粘贴的时候，会针对每一个可能会受影响的单元格触发 PrepareEdit 事件，从而让编辑控制做到滴水不漏。

示例二

本示例请参考 CaseStudy 目录下的示例文件“自动输入.Table”。

假定项目中有一个名为“行政区域”的基础数据表，这个表已经输入了全国所有县级行政区域的资料，包括省市、市县、区号、邮编四列；现在我们在客户表中输入数据，假定客户表也有这么四列，显然最好的输入方式是，客户表省市列能够根据行政区域表的设置，列出所有的省市供选择，选择省市之后，市县列能列出该省市所有的市县供选择，而选择市县之后，区号和邮编能够自动输入。

区号和邮编的自动输入，我们将在后续章节中介绍，我们现在看看如何利用 PrepareEdit 实现动态的列表项目。

1、先在项目事件 AfterOpenProject 中设置如下代码：

```
Tables("客户").Cols("省市").Combolist = DataTables("行政区域").GetComboListString("省市")
```

这样每次打开项目，就会自动从行政区域表提取不重复的省市名称，作为客户表省市列的列表项目。

2、在客户表的 PrepareEdit 事件中输入如下代码：

```
If e.Col.Name = "县市" Then '如果正在编辑的是县市列
    '从行政区域表提取该省市的县市作为列表项目
     e.Col.Combolist = DataTables("行政区域").GetComboListString("县市", "[省市] = '" & e.Row("省市") & "'")
End If
```

这样在客户表县市列输入内容的时候，会自动根据已经输入的省市，列出该省市的全部县市供选择。

3、到这一步，我们的设计目标已经完成，但是还有一个小问题，我们前面提到在表中进行剪切、粘贴的时候，会针对每一个可能会受影响的单元格触发 PrepareEdit 事件，这样的特性会给本例带来一些副作用，如果你在县市列复制粘贴 1000 行数据，那么就会触发一千次 PrepareEdit 事件，这可能需要一个漫长的执行过程。显然本例的 PrepareEdit 事件，只是针对正在编辑的单元格，也就是焦点所在单元格，对于因为受粘贴而影响的其他单元格，并没有必要执行此事件。为此，我们将 PrepareEdit 事件代码改为：

```
If e.IsFocusCell Then '如果是焦点所在单元格
    If e.Col.Name = "县市" Then '如果正在编辑的是县市列
        '从行政区域表提取该省市的县市作为列表项目
        e.Col.Combolist = DataTables("行政区域").GetComboListString("县市", "[省市] = '" & e.Row("省市") & "'")
    End If
End If
```

4.4.2.2.9 StartEdit

开始编辑的时候执行。

e 参数属性：

Table:　　准备编辑的表

Row:　　准备编辑的行

Col:　　准备编辑的列

IsFocusCell: 逻辑型，是否是焦点单元格

Cancel:　　逻辑型，默认为 False，设为 True 取消编辑。

StartEdit 事件的使用方法和 PrepareEdit 事件完全一样，实际上大部分 PrepareEdit 能完成的工作，都可以用 StartEdit 完成。

但是两者有细微差别；PrepareEdit 事件的代码，一进入单元格就会执行，为可能开始的编辑工作做准备，所以执行效率会差一些，因为并不是每次进入单元格都是需要编辑的；而 StartEdit 只是在真正开始编辑的时候执行，效率会稍微好一点；不过这种效率的差别，我们并不会明显感觉得到。

此外用于设置列表项目的代码，最好不要用在 StartEdit 事件中，否则你选择某一单元格的时候，可能不会出现下拉箭头按钮，因为此时 StartEdit 事件并没有执行，列表项目当然也就没有准备好。

千万不要在 PrepareEdit 事件显示 Messagebox 这样的对话框，因为每次进入单元格就会出现提示，这样会导致死循环的出现；在 StartEdit 事件中显示对话框就 没有这个问题，因为此时只有用户真正做出编辑动作的时候，对话框才会显示。

示例一

假定消费表中还有一个折扣列，我们希望只有经理级别的用户才能编辑该列的数据，如果其他用户试图编辑该列，则给出明确拒绝的提示，为此可在该表的 StartEdit 事件中输入代码：

```
If e.Col.Name = "折扣" AndAlso User.Group <> "经理" Then
    e.Cancel = True '取消编辑
      Messagebox.Show(" 只 有 经 理 才 能 打 折 ！ "," 提 示 ",MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

上面的代码如果设置在 PrepareEdit 事件中，非经理级用户一旦选择折扣列，将不停地出现提示窗口，使得你再也回不到正常的操作界面，你只能借助 Foxtable 提供的灾难恢复了。

而设置在 StartEdit 事件中，就不会有任何问题。

示例二

对于文件列和图片列，如果希望禁止用户编辑单元格内容，可以将 StartEdit 事件代码设置为：

```
If e.Col.name = "列名" Then
    e.Cancel = True
End If
```

你也许会问，设置在PrepareEdit事件中不行吗？ 当然可以，但是二者是有差别的：在PrepareEdit事件中设置，会导致单元格右边的按钮"..."无效；而在StartEdit设置，此按钮依然有效，你可以继续通过这个按钮增加、插入或删除文件，只是不能直接修改单元格内容。

4.4.2.2.10 ValidateEdit

结束编辑单元格前执行，通常用于验证输入的内容。

e 参数属性：

Table: 所编辑的表

Row: 所编辑的行

Col: 所编辑的列

Text: 字符型，表示所输入的文本内容，

Cancel: 逻辑型，设为 True 表示无法通过验证，禁止退出编辑模式。

ValidateEdit 和 DataColChanging 有些类似，但是 ValidateEdit 针对的是输入验证，而 DataColChanging 使用范围更广，因为任何形式导致的数据变动，包括输入以及通过代码设置，都会触发 DataColChanging 事件。 所以实际开发的时候，最好用 DataColChanging 事件，除非你的代码仅仅用于输入验证。

我们先来看看 ValidateEdit 事件是如何进行输入验证的。

假定在订单表中输入数据，要求折扣不能大于 0.15，代码非常简单，在订单表的 ValidateEdit 事件中输入：

```
If e.Col.Name = "折扣" Then '如果正在输入的是折扣列
    If e.Text > "" Then '如果已经输入内容
        Dim v As Double = Val(e.Text) '将输入的内容转换为数值
        If v > 0.15 Then '如果输入的值大于 0.15
```

```
            e.Cancel = True '则禁止退出编辑模式
        End If
    End If
End If
```

上面的代码首先判断在折扣列 输入的内容是否大于 0.15，如果是，则禁止退出编辑模式，除非你修改折扣值直到符合要求，或者按 Esc 键退出编辑。

如果你想幽默一点、友善一点，可以加上提示：

```
If e.Col.Name = "折扣" Then '如果正在输入的是折扣列
    If e.Text > "" Then '如果已经输入内容
        Dim v As Double = Val(e.Text)
        If v > 0.15 Then '将输入的内容转换为数值
            MessageBox.Show(" 兄 弟 , 你 想 老 板 破 产 吗 ?", " 提 示 ", MessageBoxButtons.OK, MessageBoxIcon.Information)
            e.Cancel = True '则取消输入
        End If
    End If
End If
```

再例如只允许输入 10 天内的日期，则可以将 ValidateEdit 事件可以设为：

```
If e.Col.Name = "日期" Then
    If e.Text > "" Then
        Dim v as Date = CDate(e.Text)
        If (Date.Today - v).TotalDays > 10 Then
            e.Cancel = True
        End If
    End If
End If
```

一旦将 e.Cancel 参数设为 True，你将无法退出编辑状态，直到重新输入符合要求的数据，或者按 Esc 键撤销此次编辑。

利用 ValidateEdit 事件，也可以实现代码录入：

```
If e.Col.Name = "产品" Then
    If e.Text = "tv" Then
        e.Text = "电视机"
    ElseIf e.Text = "tel"
        e.Text = "电话机"
    End If
End If
```

设置上述代码后，如果你在产品列输入 tv，将自动替换为电视机，如果输入 tel，自动替换为电话机。

例子很简单，实际应用的时候，希望你能举一反三。

4.4.2.2.11 AfterEdit

结束编辑单元格后执行，通常用于执行一些特定的操作和计算。

e 参数属性：

Table: 被编辑的表
Row: 被编辑的行
Col: 被编辑的列
AfterEdit 和 DataColChanged 有些类似，但是 AfterEdit 针对的是输入，而 DataColChanged 使用范围更广，因为任何形式导致的数据变动，包括输入以及通过代码设置，都会触发 DataColChanged 事件。 所以实际开发的时候，使用 DataColChanged 的场合更多。
此外，在表中进行剪切、粘贴的时候，会针对每一个受影响的单元格触发 AfterEdit 事件，从而让编辑控制做到滴水不漏。
示例
假定产品表中有一个单价列，你认为这一列比较重要，所以希望增加一列，用于记录：谁在什么时候修改了此产品的单价。
你可以在产品的 AfterEdit 事件中设置如下代码：

```
If e.Col.Name = "单价" Then
    e.Row("Log") = User.Name & " " & Date.Now
End If
```

这里假定新增列的名称为“Log”。

4.4.2.2.12 CurrentChanged

选择不同数据行后执行，也就是 Current 属性发生变化后执行。
e 参数属性：
Table: 引发事件的 Table

本事件和 PositionChanged 有些相似，但是有本质区别，例如选定第二行，然后排序数据，由于排序后输入焦点还在第二行，Position 属性没有发生变化，所以 PositionChanged 不会触发，但是排序之后，第二行可能已经不是排序之前的第二行了，从而导致 Current 属性发生变化，因此可能会触发 CurrentChanged 事件。
示例一
假定我正在订单表操作，我们希望在订单表中选定一行时，客户表的光标能够自动定位到该客户，这样我一旦从订单表回到客户表，即可看到刚刚所选订单的客户资料。
为此我们在 CurrentChanged 事件中设置如下代码：

```
If e.Table.Current Is Nothing Then '如果 Current 为 Nothing
    Return '则返回
End If
Dim wz As Integer
Dim dr As DataRow
dr = DataTables("客户").Find("[客户 ID] = '" & e.Table.Current("客户 ID") & "'")
If dr IsNot Nothing Then
    wz = Tables("客户").FindRow(dr)
    If wz >= 0 Then
        Tables("客户").Position = wz
    End If
End If
```

这种在不同表之间联动的技巧应该掌握，也许你觉得通过关联，不就可以在订单表中以关联表的形式显示客户信息吗？
但是：
1、有的时候我们并不希望建立太多的关联。
2、即使建立了关联，我们也可能关闭了 Foxtable 的双向关联特性，这样就无法以关联表的形式查

看父表数据了。

示例二

选择某一行的时候，我们有时会希望根据该行的状态给使用者一些提示。

例如你正在使用一个送水系统，其中有一列为“剩余桶数”，你希望选择某一行时，能够根据剩余桶数给接线员一些提示。

为此我们在 CurrentChanged 事件中设置如下代码：

```
If e.Table.Current Is Nothing Then '如果 Current 为 Nothing
    StatusBar.Message1 = ""
Else
    Dim Val As integer = e.Table.Current("剩余桶数")
    If Val = 0 Then
        StatusBar.Message1 = "已经全部送完， 无水可送！"
    ElseIf Val = 1 Then
        StatusBar.Message1 = "最后一桶，请立即提示客户订水。"
    Elseif Val < 3 Then
        StatusBar.Message1 = "剩余" & Val & "桶水" & "， 请适当提示客户该续订了。"
    Else
        StatusBar.Message1 = "剩余" & Val & "桶水"
    End if
End If
```

注意事项

如果筛选后，表中没有符合条件的行，或者在汇总模式下，选择分组行，此时表的 Current 返回 Nothing，因为此时并不存在当前行，通过 Current 来引用数据会出错。

所以在 CurrentChanged 事件中，一定要判断 Current 是否为 Nothing，再执行相应的代码：

```
If e.Table.Current Is Nothing Then '如果 Current 为 Nothing
    '没有选定行时执行的代码
Else
    '有选定行时执行的代码
End If
```

4.4.2.2.13 CellButtonClick

单击单元格按钮后执行，通常用于实现一些复杂的输入功能。

e 参数属性：

Table: 引发事件的表

Row: 引发事件的行

Col: 引发事件的列

Cancel: 逻辑型，设为 True，取消默认的动作。

在单元格中显示按钮的方法很简单，只需将该列的列表项目设为“...”或“|...”即可，差别是前者不能编辑单元格的内容，后者则可以。

在默认情况下，单击单元格按钮会出现一个编辑窗口，用于向单元格输入内容。

通过将 e 参数 Cancel 设为 True，可以取消这个默认的动作，执行其他操作。

示例

例如希望单击某列的单元格按钮，能够显示一个设计好的窗口，可以将 CellButtonClick 事件代码设为：

```
If e.Col.Name = "第三列" Then
    e.Cancel = True '取消默认动作
    Forms("窗口 1").Open()
```

End If

要执行上面的代码，必须先设计一个名为“窗口 1”的窗口，窗口类型为“模式”，有关窗口设计的知识，会有专门的章节介绍。

4.4.2.2.14 DrawCell

在绘制单元格的时候执行，主要用于标记数据。

e 参数属性：

Table: 准备绘制的表

Row: 准备绘制的行

Col: 准备绘制的列

Style：指定自定义样式的名称，如果用默认的样式绘制单元格，无须设置 Style 属性。

Text： 字符型，获得或者设置要绘制的文本内容

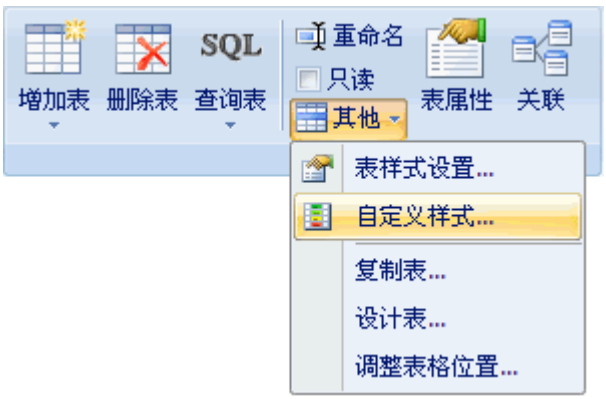
因为 DrawCell 事件执行非常频繁，所以代码必须简洁，不可以有太耗时的复杂代码，也不能有显示对话框的代码，否则会出现死循环，切记切记。

示例一

对于数据管理来说，标出异常或特殊的数据也是一项经常性的工作，Foxtable 并没有单独设置单元格颜色的方法，那么如何来标记呢？

假定我们一个学生成绩表中，需要用红色背景标出 60 分以下的分数，用绿色背景标出 95 分以上的分数。

首先在菜单的数据表功能区，执行自定义样式命令：



然后增加两个样式，分别是优秀和不及格，前者的背景颜色设为绿色，后者的背景颜色设为红色。



最后在 DrawCell 事件中设置如下代码：

```
'如果是数值型列，且不是总分列
If e.Col.IsNumeric AndAlso e.Col.Name <> "总分" Then
    If e.Row.IsNull(e.Col.Name) = False '且该列已经输入内容
        If e.Row(e.Col.Name) < 60 Then '如果该列的值小于 60
            e.Style = "不及格" '那么用"不及格"样式绘制单元格
        ElseIf e.Row(e.Col.Name) > 95 Then '如果单元格的值大于 95
            e.Style = "优秀" '那么用"优秀"样式绘制单元格
        End If
    End If
End If
```

现在程序将用不同的背景颜色，分别标出优秀和不及格的分数：

	姓名	语文	数学	英语	物理	化学	生物	地理
1	贺辉	70	76	72	72	73	88	65
2	周明	84	88	97	88	48	74	87
3	杨刚	95	79	66	60	60	95	73
4	林海	59	69	92	86	82	77	96
5	李鹏	75	63	98	79	64	84	61
6	朱德	79	64	64	68	64	96	82
7	贺龙	67	71	80	77	88	64	86
8	余委	98	81	99	60	88	79	89

提示，自定义样式不仅可以通过菜单添加，也可以通过代码添加，请参考：AddUserStyle

示例二

有的时候，不是标记一个单元格，而是希望整行用不同的颜色标记出来，那么代码更加简单。例如希望总分低于 360 分的行，用红底白字标出，只需首先增加一个名为“不及格”的样式，将其设置为红底白字，然后将 DrawCell 事件代码设置为：

```
If e.Row("总分") < 360 Then
    e.Style = "不及格"
```

```
End If
```

示例三

DrawCell 事件不仅用于标记数据，还可以用他“篡改”数据。

假定有一名为“密码”的列，要求只有经理级别的用户才能查看该列的数据，其他人查看的时候，看到的只是"****"。

实现的代码也非常简单，在该表的 DrawCell 事件中输入：

```
If User.Group = "经理" Then '如果是经理
    Return '那么返回，正常显示数据
End If
If e.Col.Name = "密码" Then '如果正在绘制的是密码列
    e.Text = "****" '那么用*代替原来的内容
End IF
```

DrawCell 事件只是影响显示内容，并不会对真实的值有任何影响。

示例四

汇总模式先的分组行同样会触发 DrawCell 事件，这一点和常规的表事件有所不同。

例如在汇总模式下，对于数量小计超出一定值的汇总结果进行标记：

```
If e.Row.IsGroup AndAlso e.Col.Name = "数量" Then '如果是分组行的数量列
    If e.Row.Level = 0 Then '如果是1级分组
        If e.Row("数量") > 5000 Then
            e.Style = "样式1"
        End If
    ElseIf e.Row.Level = 1 Then '如果是二级分组
        If e.Row("数量") > 1000 Then
            e.Style = "样式2"
        End If
    End If
End If
```

示例五

DrawCell 事件使用自定义样式来标记单元格。

每个 DataTable 都有 Styles 集合，用于获得指定名称的自定义样式，例如：

```
DataTables("成绩表").Styles("不及格")
```

表示成绩表的不及格样式。

样式包括以下属性：

BackColor： 背景颜色

ForeColor： 字体颜色

FontBold： 逻辑型，设为 True，字体加粗

FontItalic：逻辑型，设为 True，字体倾斜

FontStrikeout：逻辑型，设为 True，字体带删除线

FontUnderline：逻辑型，设为 True，字体带下划线

例如对于不及格的成绩，不仅要求用红色标记出来，而且希望能够不停地闪烁，实现的步骤为：

1、增加一个名为“不及格”的自定义样式，样式的背景颜色设为红色。

2、将 DrawCell 事件的代码设为：

```
'如果是数值型列，且不是总分列
If e.Col.IsNumeric AndAlso e.Col.Name <> "总分" Then
    If e.Row.IsNull(e.Col.Name) = False '且该列已经输入内容
        If e.Row(e.Col.Name) < 60 Then '如果该列的值小于 60
            e.Style = "不及格" '那么用"不及格"样式绘制单元格
        End If
```

```
    End If
End If
```

3、在计划管理(位于数据库功能区)中增加一个计划，时间间隔设为 1000，代码设为

```
With DataTables("成绩表").Styles("不及格")
    If .BackColor = Color.Red Then
        .BackColor = Color.White
    Else
        .BackColor = Color.Red
    End If
End With
```

4.4.2.2.15 DataRowLockedChanging

行的锁定状态改变前执行，可以在此处进行权限和逻辑验证，决定是否允许锁定或取消锁定此行。

e 参数属性：

DataTable：返回要锁定行的数据表。

DataRow： 返回要锁定的数据行。

Cancel: 逻辑型，设为 True，取消即将进行的锁定行操作。

你也许会奇怪，怎么知道进行的操作是锁定行还是取消锁定行？ 其实很简单，如果行的当前状态是锁定，那么进行的就是取消锁定行操作，否则就是锁定行。

示例一

假定只允许经理级别的用户改变行的锁定状态，可以设置 DataRowLockedChanging 事件代码为：

```
If User.Group <> "经理" Then
    e.Cancel = True
End If
```

示例二

假定有个表，要求：

1、只有已经审核通过的行，才能锁定。

2、对于已经审核通过并锁定的行，只有经理才能解锁。

要达到上述目的，可以设置 DataRowLockedChanging 事件代码为：

```
If e.DataRow("审核") = False Then  '未审核通过的行
    If e.DataRow.Locked  = False Then '如果此行目前未锁定,也就是准备锁定此行
        Messagebox.show(" 不 能 锁 定 未 审 核 通 过 的 行 !"," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
        e.Cancel = True
    End If
ElseIf e.DataRow("审核") = True Then '已经审核通过的行
   If e.DataRow.Locked  = True Then '如果此行已锁定,也就是准备取消锁定此行
      If User.Group <> "经理" Then
         Messagebox.show(" 只 有 经 理 才 能 解 锁 已 经 审 核 通 过 的 行 !"," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
         e.Cancel = True
      End If
```

```
    End If
End If
```

4.4.2.2.16 DataRowLockedChanged

行的锁定状态改变后执行。
e 参数属性：
DataTable：返回锁定行所在数据表。
DataRow： 返回锁定行。
示例
假定订单和订单明细表已经建立关联，希望锁定某订单时，能同时锁定其对应的全部订单明细，反之亦然。
要实现此目的，只需将订单表的 DataRowLockedChanged 事件代码设置为：

```
For Each dr As DataRow In e.DataRow.GetChildRows("订单明细")
    dr.Locked = e.DataRow.Locked
Next
```

4.4.2.2.17 全部表事件

表事件多达个 60 个，可以对方方面面进行控制，但是最常用的就是我们前面介绍的 13 个，请重点掌握，其余的只需大概知道有这么一个事件即可，需要的时候再来此查阅具体的帮助。
下表列出了所有的表事件：

类别	名称	说明
数据编辑	PrepareEdit	准备编辑单元格的时候执行。 通常在这里进行一些权限和条件的判断，以决定是否允许此次编辑。 也可以在这里进行动态的列表项目设置。
	StartEdit	开始编辑单元格的时候执行。
	ValidateEdit	结束编辑单元格前执行，通常用于验证输入的内容
	AfterEdit	结束编辑单元格后执行，通常用于执行一些特定的操作和计算
	CellButtonClick	单击窗口按钮后执行，通常用于实现一些复杂的输入功能
	ChangeEdit	编辑框中内容发生变化时执行
	KeyPressEdit	在编辑框每输入一个字符时执行
	KeyDownEdit	在编辑框中按下某一按键时执行
	KeyUpEdit	在编辑框中松开某一按键时执行。
数据跟踪	DataColChanging	某一列的内容发生变化前执行，此时列的内容还是更改前的值。 通常利用这个事件进行数据的有效性验证。

	DataColChanged	某一列的内容被更改后执行，此时列的内容已经是更改后的值。 自动输入、计算、统计和操作等等，都是利用这个事件来实现的，这是最常用事件之一。
	BeforeSaveDataRow	在保存某一数据行之前执行，可以在此对数据进行最终的验证。
增加行	BeforeAddDataRow	增加一行之前执行。 通常在这里进行权限和条件判断，以决定是否允许增加行。
	DataRowAdding	增加一行时执行，此时新增行已经创建，但并未真正增加到表中，通常在此处对新增行进行默认值的设置
	DataRowAdded	增加一行后执行，此时新增行已经真正增加到表中。
删除行	BeforeDeleteDataRow	删除一行之前执行。 可以在此处进行权限和条件验证，以决定是否允许删除此行。
	DataRowDeleting	正在删除一行的时候执行，这时行还未被删除
	DataRowDeleted	删除行后执行，此时行已经被删除
锁定行	DataRowLockedChanging	行的锁定状态改变前执行。 可以在此处进行权限和条件验证，以决定是否允许锁定和取消锁定此行。
	DataRowLockedChanged	行的锁定状态改变之后执行。
位置与坐标	PositionChanged	选择不同位置的数据行后执行，也就是 Position 属性发生变化后执行
	CurrentChanged	选择不同数据行后执行，也就是 Current 属性发生变化后执行。
	BeforeSelChange	选择不同的单元格前执行，也就是表的 RowSel、ColSel 属性发生变化前执行
	AfterSelChange	选择不同的单元格后执行，也就是表的 RowSel、ColSel 属性发生变化后执行
	BeforeSelRangeChange	选择不同区域前执行，也就是表的 TopRow、LeftCol、BottomRow、RightCol 等属性发生变化前执行
	AfterSelRangeChange	选择不同区域后执行，也就是表的 TopRow、LeftCol、BottomRow、RightCol 等属性发生变化后执行
调整	BeforeMoveRow	移动行前执行。
	AfterMoveRow	移动行后执行。
	BeforeDragColumn	用鼠标拖动列标题，调整位置之前发生。
	AfterDragColumn	用鼠标拖动列标题，调整位置之后发生
	BeforeFreezeColumn	用鼠标调整冻结区之前发生
	AfterFreezeColumn	用鼠标调整冻结区之后发生

	BeforeResizeColumn	用鼠标拖动列标题边界，调整列宽之前发生
	AfterResizeColumn	用鼠标拖动列标题边界，调整列宽之后发生
	BeforeResizeRow	通过鼠标拖动，调整行高之前发生
	AfterResizeRow	通过鼠标拖动，调整行高之后发生
复选	BeforeCheckRow	修改行复选框状态前执行。
	AfterCheckRow	修改行复选框状态后执行。
加载	BeforeLoad	重新加载数据之前执行
	AfterLoad	重新加载数据后执行
筛选	BeforeFilter	筛选数据前执行
	AfterFilter	筛选数据后执行
排序	BeforeSort	排序数据前执行
	AfterSort	排序数据后执行
附件	BeforeAddFile	在图片列或文件管理器中单击“增加”按钮的时候执行。
	BeforeAttachFile	在向在文件型、多文件型、以及图片型列中添加文件的时候执行
	BeforeOpenFile	在图片列或文件管理器中单击“打开”按钮的时候执行。
	BeforeDeleteFile	在文件或图片管理器，删除文件或图片的时候执行。
绘制	DrawCell	在绘制单元格的时候执行，主要用于标记数据
	BeforeDrawRow	绘制某一行之前执行，通常用于动态设置行高。
鼠标与按键	KeyDown	在表中按下某个按键的时候执行
	KeyUp	在表中松开某个按键的时候执行
	DoubleClick	双击单元格后执行
	DoubleClickRowHeader	双击某行的行号单元格时触发。
	Click	单击鼠标后执行
	MouseEnterCell	鼠标进入某个单元格的时候执行。
	MouseLeaveCell	鼠标离开某个单元格的时候执行。
杂项	AfterScroll	滚动表后执行。
	AfterLoadTableSetting	Table 生成后执行，可以在这里对 Table 或者 Col 进行初始化设置。
	BeforeShowContextMenu	显示快捷菜单之前执行，可以在此禁止显示快捷菜单，或更改要显示的快捷菜单。

4.4.2.2.18 表达式列的特殊性

我们知道，列的值发生变化，会相继触发 DataColChanging 和 DataColChanged 事件。
但是有一个例外： 表达式列的值发生变化时，并不会触发这两个事件。

假定 C 列是一个表达式列，根据 A 列和 B 列的值计算得出，而 D 列的值则根据 C 列的值用代码计算得出。
通常我们会这样编写 DataColChanged 事件的代码：

```
Select Case e.DataCol.Name
    Case "C"
        '计算D列的代码
End Select
```

但是上述代码是无效的，因为作为表达式列，C 列发生变化的时候，并不会触发 DataColChanged 事件。
我们可以换个思路，既然 C 列是有 A 列和 B 列计算得出，那么 C 列发生变化，肯定是因为 A 列或 B 列发生变化引起的。
所以正确的代码是：

```
Select Case e.DataCol.Name
    Case "A" ,"B"
        '计算D列的代码
End Select
```

通过子表触发父表事件

还有一个特殊情况，表达式列并非由本表的列计算得出，而是由子表数据计算得出。
例如假定订单表有个数量列，是表达式列，用于累计订单明细表的数量，其表达式为： Sum(Child.数量)。
再假定订单表的DataColChanged事件设置了如下代码：

```
If e.DataCol.name = "数量" Then
  '相关计算代码
End If
```

当我们在订单明细表输入数量时，订单表的数量会自动更新，但是上面的代码永远不会执行，因为数量是一个表达式列，不会触发事件。
我们需要在订单明细表的DataColChanged事件加上代码：

```
If e.DataCol.name = "数量" Then
    Dim pr As DataRow = e.DataRow.GetParentRow("订单") '找出对应的父行
    If pr IsNot Nothing Then
        DataTables("订单").DataCols("数量").RaiseDataColChanged(pr)  '通知系统此父行的数量列
```

```
发生了改变,触发DataColChanged事件
    End If
End If
```

这样在订单明细表输入数量，就会在订单表针对该订单的数量列触发DataColChanged事件。

有的用户编写代码不严谨，简单地将订单明细的代码设置为：

```
If e.DataCol.name = "数量" Then
    DataTables("订单").DataCols("数量").RaiseDataColChanged
End If
```

上面的代码效率极低，因为修改某个订单明细的数量，就会重算所有订单的数量，而不是只重算相应订单的数量。

4.4.2.2.19 事件中的关联表

我们知道，因为关联表的存在，一个 DataTable 可能对应有多个 Table。
例如订单和订单明细两个表，通过订单编号建立关联，那么订单明细对应有两个 Table，分别是：
Tables("订单明细")
Tables("订单.订单明细")
不管一个表对应有多少个 Table，这些 Table 都是共享事件的，例如我们在订单明细 PrepareEdit 事件中加上代码：
e.Cancel = True
那么 Tables("订单明细")和 Tables("订单.订单明细")都是不可以编辑的了。
如果希望禁止编辑 Tables("订单明细")，但是允许编辑 Tables("订单.订单明细")，代码可以改为：

```
If e.Table.Name = "订单明细" Then
    e.Cancel = True
ElseIf e.Table.Name = "订单.订单明细"
    e.Cancel = False
End If
```

4.4.2.2.20 事件中的父表和子表

父表和子表之间的数据和操作，有时是相互依存的，某一方进行的操作，经常需要在另一方进行条件判断。
在子表的事件中，我们可以通过 GetParentRow 获得父行，在父表的事件中，可以通过 GetChildRows 获得全部子行。

示例一

假定订单和订单明细表已经建立关联，订单表有个客户等级列，订单明细表有个折扣列，要求只有客户等级为 VIP 的时候，才允许折扣超过 10%。

为此可以在订单明细表的 DataColChanging 事件中设置代码：

```
If e.DataCol.Name = "折扣" Then
    If e.NewValue > 0.1 Then
        Dim pr As DataRow = e.DataRow.GetParentRow("订单")
        If pr IsNot Nothing Then
            If pr("客户等级") <> "VIP" Then
                MessageBox.show("非VIP客户的折扣不能超过10%", "提示", MessageBoxButtons.OK,
MessageBoxIcon.Information)
                e.Cancel = True
            End If
        End If
    End If
End If
```

示例二

假定希望锁定某订单时，能同时锁定其对应的全部订单明细，反之亦然。

要实现此目的，只需将订单表的 DataRowLockedChanged 事件代码设置为：

```
For Each dr As DataRow In e.DataRow.GetChildRows("订单明细")
    dr.Locked = e.DataRow.Locked
Next
```

示例三

假定订单和订单明细表都有一个名为“审核”的列，逻辑型，要求某订单的所有订单明细全部审核通过后，才能审核订单。

为此可以将订单表的 PrepareEdit 事件代码设置为：

```
If e.Col.name = "审核" Then
    For Each dr As DataRow In e.Row.DataRow.GetChildRows("订单明细")
        If dr("审核") = False Then
            e.Cancel = True
            Exit For
        End If
    Next
End If
```

示例四

假定订单和订单明细表都有一个名为“审核”的列，逻辑型。

当某个订单的所有订单明细全部通过审核后，此订单也自动审核通过，当某订单的任何一个订单明细取消审核后，此订单也自动取消审核。
为此可以将订单明细表的 DataColChanged 事件代码设置为:

```
If e.DataCol.name = "审核" Then
    Dim pr As DataRow = e.DataRow.GetParentRow("订单")
    If pr IsNot Nothing Then
        Dim crs As List(of DataRow) = pr.GetChildRows("订单明细")
        Dim cnt As Integer
        For Each cr As DataRow In crs
            If cr("审核") = True
                cnt = cnt + 1
            End If
        Next
        pr("审核") = (crs.Count = cnt)
    End If
End If
```

4.4.2.2.21 当前表与表事件

很多用户在写表事件代码的时候，往往忽视事件的 e 参数，直接使用 CurrentTable 和 Current 来表示触发事件的表和行。

例如有人在 DataColChanged 事件写出下面的代码：

```
Dim dr As Row = CurrentTable.Current
Select Case e.DataCol.Name
    Case "日期"
        dr("年") = dr("日期").Year
        dr("月") = dr("日期").Month
End Select
```

这段代码表面上没有什么问题，实际上存在严重的漏洞，因为 CurrentTable.Current 表示当前表的当前行，和触发事件的表和行是两回事。

虽然有些时候，当前表和当前行，确实就是触发事件的表和行，但并非总是如此，例如你在表 B 设计了一个窗口，这个窗口中的部分 TextBox 绑定到表 A 的字段，当你选择表 B 并打开此窗口，在这些 TextBox 输入内容的时候，会触发表 A 的 DataColChanged 事件，显然此时 CurrentTable.Current 表示表 B 的当前行，并非表 A 的当前行。
再例如通过代码直接向某个表写入数据：

```
For Each dr As DataRow In DataTables("表A").DataRows
    dr("日期") = Date.today()
Next
```

显然，执行上述代码会从表A第一行开始逐行写入日期，从而逐行触发DataColChanged事件，如果

你还是用本节开始的内容计算年月，那么除了当前行外，其它行都不会计算。

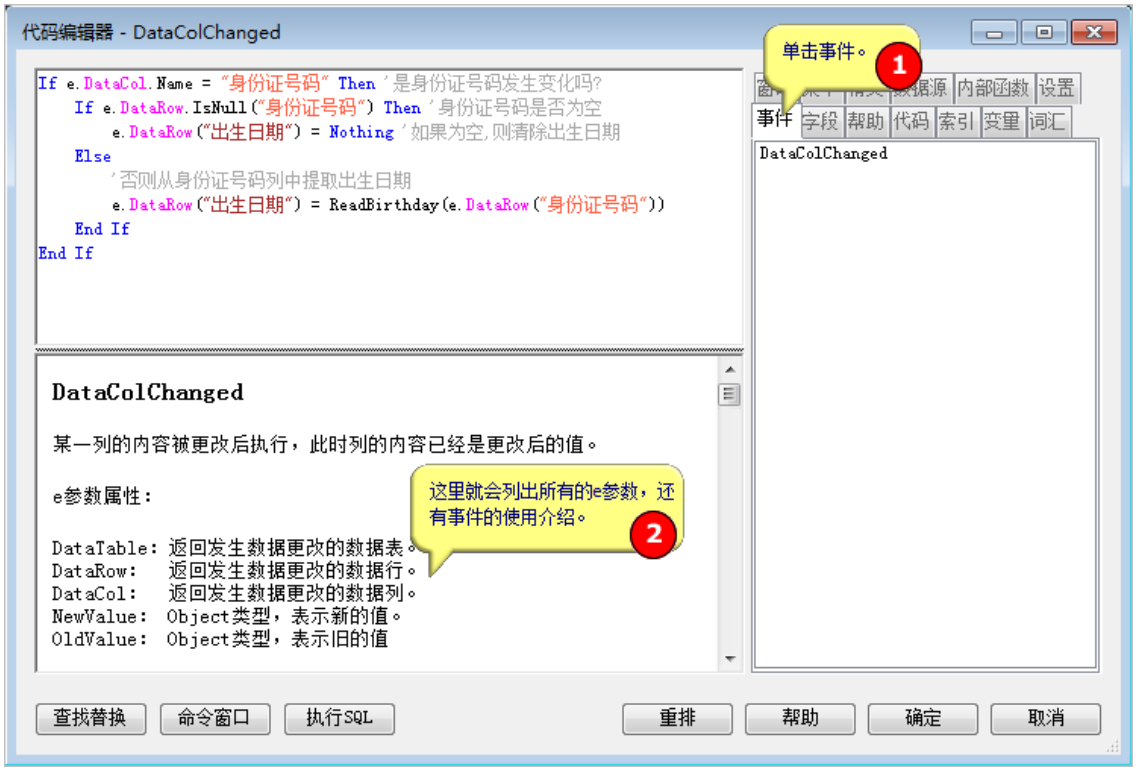
所以任何时候，都不要在表事件中，用CurrentTable和Current来表示触发事件的表和行，一定要用e参数来引用触发事件的 表和行，例如：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "日期"
        dr("年") = dr("日期").Year
        dr("月") = dr("日期").Month
End Select
```

其实很轻松

我们知道，Foxtable有两种类型的表事件，分别是针对DataTable和Table的，它们的e参数是不同的，参考：两种类型的表事件。
就是同一种类型的表事件，e参数也会有一些差异，例如同样是针对Table的表事件DrawCell和AfterEdit，其e参数就有比较大的差异 。

不过我们不要刻意记忆任何e参数，因为在实际编写代码的时候，可以很方便地查看表事件的e参数：



此外在编写代码的时候，输入"e."，然后按Tab，代码精灵不仅会列出该事件的e参数， 还可以自动输入，以PrepareEdit事件为例：



4.4.2.3 项目事件

4.4.2.3.1 MainTableChanging

切换主表前执行，也就是 MainTable 发生变化前执行。

e 参数属性：
OldTableName： 字符型，原来的主表名称
NewTableName： 字符型，新的主表名称
Cancel： 逻辑型，设为 True，将禁止切换到新的主表。
示例
例如禁止直接从表 A 切换到表 C：

```
If e.NewTableName = "表 C" AndAlso e.OldTableName = "表 A" Then
    e.Cancel = True
End If
```

4.4.2.3.2 MainTableChanged

切换主表后执行，也就是 MainTable 发生变化后执行。
此事件无 e 参数，这是因为系统变量 MainTable 可以直接返回所选择的主表。
示例

假定有一个订单管理系统，我们在 MainTableChanged 事件中设置如下代码：

```
If MainTable.Name = "订单" Then '如果选择的主表是订单表
    Tables("订单").Cols("客户").Combolist = DataTables("客户").GetComboListString("公司名称")
End If
```

这样每次选择订单表作为主表，都会从客户表提取公司名称，作为订单表客户列的列表项目。

4.4.2.3.3 CurrentTableChanged

选择不同表后执行，也就是 CurrentTable 发生变化后执行。

此事件无 e 参数，这是因为系统变量 CurrentTable 可以直接返回所选择的表。

如果我们选择不同的主表，会先后触发 MainTableChanged 和 CurrentTableChanged 事件，因为此时 MainTable 和 CurrentTable 都发生了变化。

如果我们选择一个关联表，那么只会触发 CurrentTableChanged 事件，因为此时只有 CurrentTable 发生了变化。

例如将 CurrentTableChanged 事件代码设为：

```
If CurrentTable.DataTable.Name = "订单" Then '如果选择的主表是订单表
    CurrentTable.Cols("客户").Combolist = DataTables("客户").GetComboListString("公司名称")
End If
```

这样我们每次选择一个 Table，如果这个 Table 的数据来自于订单表，都会从客户表提取公司名称，作为该表客户列的列表项目。

4.4.2.3.4 BeforeOpenProject

在打开项目之前执行，可以在此进行一些环境上的判断，决定是否打开项目。

e 参数属性：

File：　　字符型，表示正在打开的项目文件名称，含路径。

Cancel:　　逻辑型，设为 True，拒绝打开项目。

HideSplashForm：　逻辑型，设为 True，关闭 Foxtable 的启动封面。

示例

假定需要在打开项目前，检查项目文件目录中，是否存在名为 ip.txt 的文件，如果不存在，就拒绝打开项目：

```
Dim s As String = FileSys.GetParentPath(e.File) & "\ip.txt"
If FileSys.FileExists(s) = False Then
    e.Cancel = True
    e.HideSplashForm = True
    MessageBox.show("文件丢失,无法打开此项目.")
End if
```

4.4.2.3.5 Initialize

此事件在 数据已经加载，所有的 DataTable 都已经生成，即将生成 Table 和窗口的时候执行，通常

在此对 DataTable 进行初始化设置。
不能在 Initialize 事件中出现针对 Table、Col 和窗口的代码，因为此时上述对象还未生成。
示例
可以在 Initialize 事件中设置代码，修改列标题：

```
With DataTables("订单")
    .DataCols("Name").Caption = "姓名"
    .DataCols("ID").Caption = "编号"
    .DataCols("Date").Caption = "日期"
End With
```

4.4.2.3.6 AfterOpenProject

此事件在项目已经完全打开后执行，此时 DataTable、Table 和窗口都已经生成。
示例一

例如希望产品表内容只能在 Tables("产品")修改，也就是只能在主表修改，而不能在关联表修改，可以在 AfterOpenProject 事件中加入如下代码：

```
DataTables("产品").AllowEdit = False
Tables("产品").AllowEdit = True
```

注意上面的代码不能设置在 Initialize 事件中，因为 Initialize 事件代码执行时，Table 还没有生成的。
示例二
假定打开项目的用户是“经理”级别的时候，则自动打开某个窗口，可以在 AfterOpenProject 事件中加入代码：

```
If User.Group = "经理" Then
    Forms("窗口 1").Open()
End If
```

4.4.2.3.7 LoadUserSetting

打开系统或者切换用户之后执行，可以在此针对用户进行权限方面的设置。
此事件无 e 参数，因为通过系统变量 User 可以直接返回登录用户。
示例
例如只有管理员用户，才能编辑表 A，可以在 LoadUserSetting 事件中加入下面的代码：

```
If User.Type = UserTypeEnum.Administrator Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

提示：
如果项目没有提供切换用户的功能，那么可以直接在 AfterOpenProject 事件中设置上述代码。

4.4.2.3.8 BeforeCloseProject

关闭项目之前执行。
e 参数属性：
Cancel: 逻辑型，设为 True，拒绝关闭项目。
SkipSave: 逻辑型，设为 True，将不保存项目，直接关闭。
可利用本事件检查一些必要的工作是否已经完成，如果没有完成，则拒绝关闭项目。
示例：
假定表 B 有一名为"状态"的列，希望在关闭项目前自动检查，如果发现有状态列为空的记录，则提示用户。
为此，可在 BeforeCloseProject 事件中加入下述代码：

```
'如果表 B 不存在状态列为空的行
If DataTables("表 B").Find("[状态] Is Null") Is Nothing Then
    Return '那么返回
End If
Dim Result As DialogResult
Result = Messagebox.Show("存在状态不确定的记录，确定要关闭项目吗？","提示",
MessageBoxButtons.YesNo, MessageBoxIcon.Question)
If Result = DialogResult.No Then '如果用户选择否
    e.Cancel = True '取消关闭项目
    Tables("表 B").Filter = "[状态] Is Null" '表 B 中筛选出状态为空的记录。
    MainTable = Tables("表 B") '选择表 B 作为主表。
End If
```

4.4.2.3.9 SystemIdle

在系统空闲的时候执行。
此事件执行较为频繁，所以请不要出现"负荷"较重的代码。
示例一
例如为了在状态栏显示用户名、表名、总行数、当前行位置，可以在 SystemIdle 中加入下面的代码：

```
With CurrentTable
    Dim Str1 As String = "用户: " & User.Name
    Str1 = Str1 & "  表名: " & .Name & "  总行数: "
    Str1 = Str1 & .Rows.Count & "  当前行: " & (.Position + 1)
    StatusBar.Message1 = Str1
End With
```

示例二

如果用户30秒内没有进行任何操作，就自动关闭系统，可以在SystemIdle中加入下面的代码：

```
If LastInputTickCount > 30000 Then
    Syscmd.Project.Exit()
```

```
End If
```

提示：LastInputTickCount是一个系统变量，返回距离用户上次操作之后的时间，单位为毫秒。

4.4.2.3.10 AfterSaveProject

保存项目之后执行。

示例

为了能在项目文件目录下生成一个文本文件“save.txt”，记录每次存盘时的用户名和时间，可在AfterSaveProject 事件中加入下面的代码：

```
Dim FileName As String = ProjectPath & "save.txt"
Dim Info As String = User.Name & "|" & Date.Now & vbcrlf
FileSys.WriteAllText(FileName, Info, True)
```

4.4.2.3.11 BeforeSaveProject

保存项目之前执行。

e 参数：

Cancel: 逻辑型，设置为 True 取消保存。

4.4.2.3.12 BeforeLoadInnerTable

普通用户可以忽略本事件。

打开项目，准备加载某内部数据表的时候执行，可以在此更改加载的字段和条件。

e 参数属性：

DataTableName: 正在加载的数据表名称。

Fields: 指定加载的字段(列)，不同字段用逗号分割。

Filter: 设置过滤条件，只加载符合此条件的数据。

User: 登录用户。

Cancel: 逻辑型，设置为 True，将不加载此表。

HideSplashForm： 逻辑型，设为 True，关闭 Foxtable 的启动封面。

注意此事件必须通过 e.User 来引用用户。

如果数据量很大，可以通过用此事件设置条件，只加载部分数据，甚至不加载数据，在运行的过程中根据条件动态加载数据。

有关动态加载数据，会有专门的章节介绍。

示例一

初始不加载任何数据：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

由于_Identify 列是不可能为空的，所以上述代码会使得订单表初始不加载任何数据。

示例二

对于录入员，只加载当天的订单：

```
If e.DataTableName = "订单" AndAlso e.User.Name = "录入员" Then
    e.Filter = "[日期] = #" & Date.Today & "#"
End If
```

示例三

对于录入员，只加载 10 天以内的订单:

```
If e.DataTableName = "订单" AndAlso e.User.Name = "录入员" Then
    Dim StartDate As Date = Date.Today().AddDays(-10)
    e.Filter = "[日期] > #" & StartDate & "#"
End If
```

示例四

对于录入员，只加载本年的订单：

```
If e.DataTableName = "订单" AndAlso e.User.Name = "录入员" Then
    Dim StartDate As New Date(Date.Today.Year, 1, 1)
    Dim EndDate As New Date(Date.Today.Year, 12 ,31)
    e.Filter = "[日期] >= #" & StartDate & "# And [日期] <= #" & EndDate & "#"
End If
```

示例五

只加载业务员为登录用户的订单：

```
If e.DataTableName = "订单"
    e.Filter = "[业务员] = '" & e.User.Name & "'"
End If
```

示例六

只加载折扣在 0.05 和 0.15 之间的订单：

```
If e.DataTableName = "订单" Then
    e.Filter = "[折扣] Between 0.05 And 0.15"
End If
```

示例七

只加载上海和北京地区的订单：

```
If e.DataTableName = "订单" Then
    e.Filter = "[地区] In ('上海','北京')"
End If
```

4.4.2.3.13 BeforeLoadOuterTable

普通用户可以忽略本事件。
准备加载外部数据表的时候执行，可以在此修改 Select 语句，这样不仅可以只加载需要的字段和数据，甚至可以从另一个数据表中加载数据。
如果数据量很大，可以通过用此事件设置条件，只加载部分数据，甚至不加载数据，在运行的过程中根据条件动态加载数据。
有关动态加载数据，会有专门的章节介绍。

e 参数属性：

DataTableName: 正在加载的数据表名称
SelectString: Select 语句
User: 登录用户
Cancel: 逻辑型，设置为 True，将不加载此表。
HideSplashForm： 逻辑型，设为 True，关闭 Foxtable 的启动封面。
注意此事件必须通过 e.User 来引用用户。

本事件的用法和 BeforeLoadInnerTable 事件一样，只不过用 SelectString 代替了 Fields 属性和 Filter 属性，这是为了是考虑到专业级用户的需要，充分发挥外部数据源的特性，并提供更大的灵活性。
使用这个事件需要一定的 Sql 语言知识，后续章节会有专门的 Sql 语言入门介绍，目前只需掌握一下 Select 语句的基本语法。
Select 语句的语法如下：

Select * From [表名] Where 条件表达式

上面的表名是外部数据源的真实表名，不是 Foxtable 中的表名，两者有的时候并不相同。
上面的语句是加载所有字段，如果是加载部分字段，语法是：

Select [字段 1], [字段 2], [字段 3], … From [表名] Where 条件表达式

需要注意的是，如果只是加载部分字段，那么必须包括主键列在内，否则会带来不可预知的错误，这点要切记。

示例一
对于录入员，则只加载今天的订单:

```
If e.DataTableName = "订单" AndAlso e.User.Name = "录入员" Then
    e.SelectString = "Select * From {订单} Where [日期] = #" & Date.Today & "#"
End If
```

示例二
对于录入员，只加载 10 天以内的订单：

```
If e.DataTableName = "订单" AndAlso e.User.Name = "录入员" Then
```

```
    Dim StartDate As Date = Date.Today().AddDays(-10)
    e.SelectString = "Select * From {订单} Where [日期] > #" & StartDate & "#"
End If
```

示例三

只加载折扣在 0.05 和 0.15 之间的订单：

```
If e.DataTableName = "订单" Then
    e.SelectString = "Select * From {订单} Where [折扣] Between 0.05 And 0.15"
End If
```

示例四

如果只是加载部分字段，那么必须包括主键列在内，否则会带来不可预知的错误。

假定客户表的主键是“客户编号”，那么在加载部分字段的时候，客户编号列必须包括在内：

```
If e.DataTableName = "订单" Then
    e.SelectString = "Select [客户编号],[公司名称],[联系人],[地址],[电话],[手机] From {客户}"
End If
```

示例五

初始不加载任何订单：

```
If e.DataTableName = "订单" Then
    e.SelectString = "Select * From {订单} Where [_Identify] Is Null"
End If
```

由于_Identify 列是不可能为空的，所以上述代码会使得订单表初始不加载任何数据。

通过上面例子可以看出：

1、 如果只是掌握我们用得着的，那么 SQL 语言其实很简单，不要把它看得太高深，后续章节会有专门的 SQL 语言入门介绍。

2、 BeforeLoadOuterTable 和 BeforeLoadInnerTable 虽然参数的形式不一样，但是对于大多数人来说，使用起来其实是一回事。

4.4.2.3.14 BeforeConnectOuterDataSource

这个事件是专业用户才用得着的，普通用户不需要研究它。

准备连接外部数据源的时候执行，可以在此修改连接字符串。

e 参数属性：

Name: 外部数据源名称

User: 登录用户

ProjectFile: 项目文件名，含路径。

ProjectPath: 返回项目文件所在目录

ConnectionString: 返回或设置连接字符串。

HideSplashForm: 逻辑型，设为 True，关闭 Foxtable 的启动封面。

有两点要注意：

Foxtable 使用的数据提供程序是 OLE DB .NET。

数据源最好是 SQL SERVER 或者 ACCESS。

e 参数方法：

EditConnectionString：手工连接数据源。
这也是唯一一个带方法的 e 参数。
利用本事件，可以根据登录用户的不同，通过修改 e 参数的 ConnectionString 属性，链接到不同的后台数据库。
注意此事件必须通过 e.User 来引用用户。

示例一

假定有一个成绩管理系统，使用的是一个Access格式的外部数据源，外部数据源的名称为“成绩”，外部数据文件名为"成绩数据.mdb"，和项目文件处于同一目录之下。
为了保证这个项目复制到任何目录都能正常链接该外部数据源，可以在BeforeConnectOuterDataSource事件中加入下面的代码：

```
If e.name = "成绩" Then
    e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & e.ProjectPath & "成绩数据.mdb;Persist Security Info=False"
End if
```

这里假定外部数据源的格式2000/2003，如果是2007，代码为：

```
If e.name = "成绩" Then
    e.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data Source=" & e.ProjectPath & "成绩数据.accdb;Persist Security Info=False"
End if
```

如果有多个数据源，可以：

```
If e.name = "数据源1" Then
    e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & e.ProjectPath & "数据文件1.mdb;Persist Security Info=False"
ElseIf e.name = "数据源2" Then
    e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & e.ProjectPath & "数据文件2.mdb;Persist Security Info=False"
End if
```

示例二

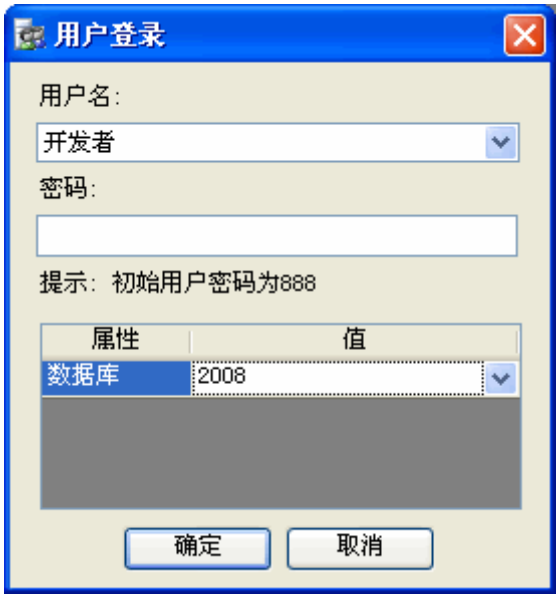
假定你有三个外部数据文件，分别是：2008.MDB、2009.MDB、2010.MDB，希望能够让用户在登录的时候，选择链接这三个数据库之一。
为解决这个问题，可以在用户管理窗口增加一个名为“数据库”的扩展用户属性，将这个扩展属性的候选值设为“|2008|2009|2010”：



然后在BeforeConnectOuterDataSource事件中加入下面的代码：

```
If e.name = "连接名称" Then
    e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & e.ProjectPath & e.User.ExtendedValues("数据库") & ".mdb;Persist Security Info=False"
End if
```

这样用户在登录的时候，不仅可以从下拉列表中选择数据库，还可以直接输入数据库名称。



示例三

如果希望能够在打开项目的时候，自动出现一个对话框，让用户可以自由选择外部数据文件，可以在BeforeConnectOuterDataSource事件中加入下面的代码：

```
If e.Name = "连接名称" Then
    Dim dlg As New OpenFileDialog
    dlg.Filter= "Access 文件|*.mdb"
    If dlg.ShowDialog = DialogResult.Ok Then
        e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & dlg.FileName & ";Persist Security Info=False"
    End If
End if
```

示例四

假定有一个名为“销售数据”远程数据源，默认的 IP 地址为 119.119.120.120，因为服务器的调整，IP 地址可能会发生变化，导致数据源无法连接。

为解决这个问题，可以在用户管理窗口增加一个名为 IP 的扩展用户属性，让用户在登录的时候，可以输入另一个 IP 地址，然后在 BeforeConnectOuterDataSource 事件中加入下面的代码：

```
If e.Name = "销售数据" Then
    Dim IP As String = e.User.ExtendedValues("IP") '获得用户输入的 IP
    If IP > "" '如果用户自己输入了 IP
      '则用户输入的 IP 替换原来的 IP
      e.ConnectionString = e.ConnectionString.Replace("119.119.120.120", IP)
    End If
End If
```

你也可以用另一种方法解决，在项目文件目录下创建一个 ip.txt 的文本文件，在该文件中输入最新的 IP 地址，并在 BeforeConnectOuterDataSource 事件中设置如下代码：

```
Dim ipFile As String = e.ProjectPath & "ip.txt"
If e.Name = "销售数据" Then
    If Filesys.FileExists(ipFile) Then
      Dim ip As String = FileSys.ReadAllText(ipFile)
      e.ConnectionString = e.ConnectionString.Replace("119.119.120.120",ip)
    End If
End If
```

示例五

假定有两个 IP 地址(也可能是两个服务器)，分别是 192.168.1.100 和 119.75.217.56，默认使用第一个 IP 地址，如果第一个 IP 地址无法接通，则自动切换到第二个 IP 地址。

要实现这个功能很简单，首先在设置数据源的时候，使用第一个 IP 地址（192.168.1.100）设置，然后在 BeforeConnectOuterDataSource 事件中设置如下代码：

```
If e.Name = "数据源名称" Then
    If Network.Ping("192.168.1.100",1000) = False  Then '如果第一个IP无法接通
      e.ConnectionString = e.ConnectionString.Replace("192.168.1.100", "119.75.217.56")
    End If
```

```
End If
```

示例六

你也可以在连接某一数据源之前，执行方法 EditConnectionString，让用户手工设置连接字符串。例如你希望在连接“销售数据”的时候，如果登录用户为开发者，则显示数据连接窗口，让用户手动选择外部数据源：

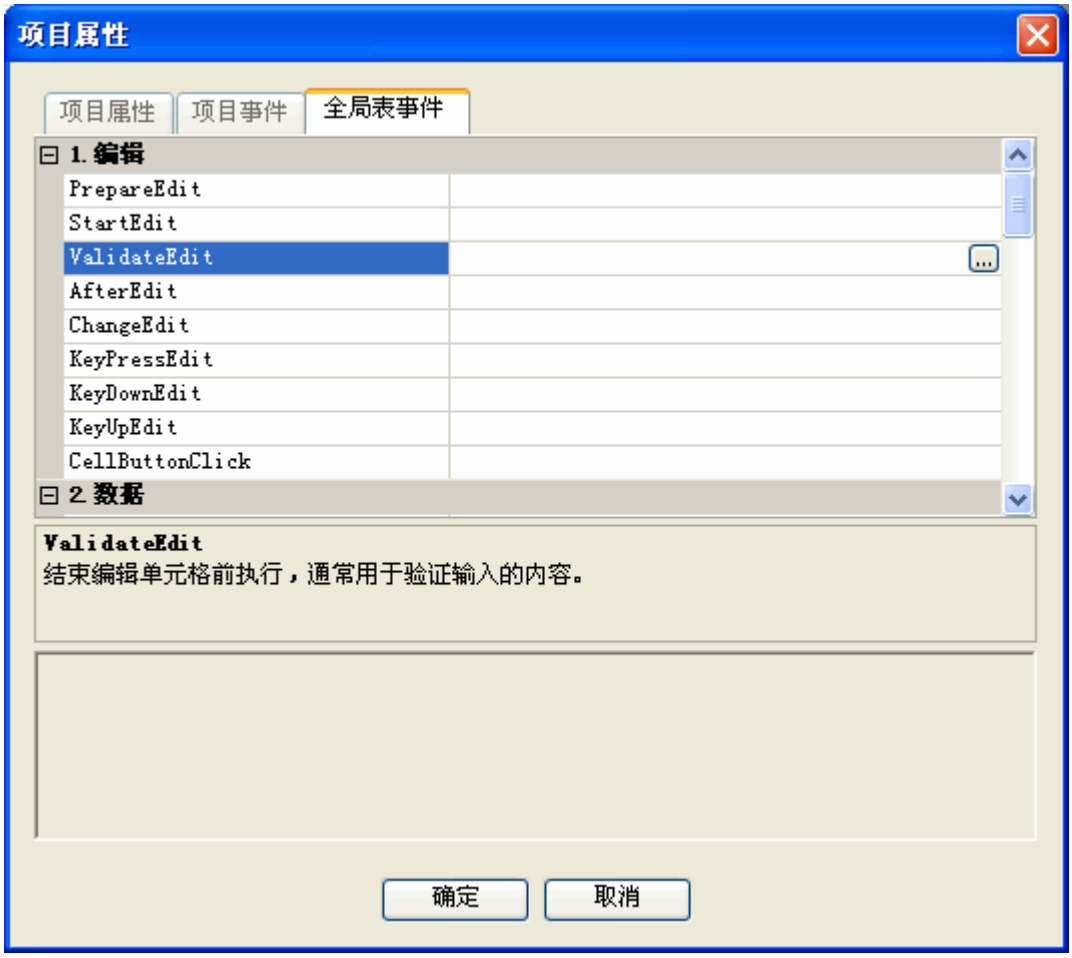
```
If e.User.Type = UserTypeEnum.Developer Then
    e.EditConnectionString()
End If
```

4.4.2.3.15 全部项目事件

事件名称	说明
MainTableChanging	切换主表前执行，也就是 MainTable 发生变化前执行。
MainTableChanged	切换主表后执行，也就是 MainTable 发生变化后执行。
CurrentTableChanged	选择不同表后执行，也就是 CurrentTable 发生变化后执行。
BeforeOpenProject	在打开项目之前执行，可以在此进行一些环境上的判断，决定是否打开项目。
Initialize	在数据已经加载，所有的 DataTable 都已经生成，即将生成 Table 和窗口的时候执行，通常在此对 DataTable 进行初始化设置。
AfterOpenProject	在项目已经完全打开后执行，此时 DataTable、Table 和窗口都已经生成。
LoadUserSetting	打开系统或者切换用户之后执行，可以在此针对用户进行权限方面的设置。
BeforeCloseProject	关闭项目之前执行。
SystemIdle	在系统空闲的时候执行。
AfterSaveProject	保存项目之后执行。
BeforeSaveProject	保存项目之前执行。
BeforeLoadInnerTable	打开项目，准备加载某内部数据表的时候执行，可以在此更改加载的字段和条件。
BeforeLoadOuterTable	准备加载外部数据表的时候执行，可以在此修改 Select 语句。
BeforeConnectOuterDataSource	准备连接外部数据源的时候执行，可以在此修改连接字符串。
GlobalVariableChanged	在某一个 Var 变量的值发生变化后执行
BeforeShowErrorMessage	在 Foxtable 捕获错误并即将显示错误信息前执行。
NetworkAvailabilityChanged	网络可用状态发生变化后执行。
NetworkAddressChanged	网络 IP 地址发生变化后执行。

4.4.2.3.16 全局表事件

在项目属性中，除了可以设置项目事件，还可以设置全局表事件。
全局表事件对所有的表有效，包括通过代码增加的临时表。
全局表事件的 e 参数和对应的表事件完全一样。



为什么要提供全局表事件呢？ 我们用一个最简单的例子说明。
假定我们需要在状态栏，显示当前行位置，以及总的行数，只需在全局表事件 PositionChanged 中加入代码：

```
StatusBar.Message1 = "第" & (e.Table.Position + 1) & "行，共" & e.Table.Rows.Count & "行"
```

由于全局表事件对所有表有效，所以不管我们选择那个表，状态栏都能显示当前行位置以及总的行数。
如果没有全局表事件，我们就需要在每个表的 PositionChanged 加入上述代码，如果有几十个表，就的加入几十次，这将给维护带来极大的不便。

如果你喜欢，你甚至可以不使用表事件，而全部采用全局表事件。
由于全局表事件对所有表有效，而不同的表可能有不同的处理逻辑，需要不同的事件代码，所以通常需要在全局表事件中判断表名：

```
Select Case e.Table.Name
    Case "表 A"
        '针对表 A 的代码
    Case "表 B"
        '针对表 B 的代码
    Case "表 C"
        '针对表 C 的代码
End Select
```

或者：

```
Select Case e.DataTable.Name
    Case "表 A"
        '针对表 A 的代码
    Case "表 B"
        '针对表 B 的代码
    Case "表 C"
        '针对表 C 的代码
End Select
```

重要提示：

为了提高系统效率，在默认情况下，只有 CurrentChanged、PositionChanged、AfterSelChange、AfterSelRangeChange 四个全局表事件对所有表有效， 而其它全局表事件即使设置了代码，也不会发生作用。
当然这并不意味着其它全局表事件永久无效，只是这些表事件需要一个“开关”去开启。
DataTable 有一个 GlobalHandler 属性，该属性包括以下子属性，对应着每一个全局表事件：

AfterCheckRow
AfterDragColumn
AfterEdit
AfterFilter
AfterFreezeColumn
AfterLoad
AfterMoveRow
AfterResizeColumn
AfterResizeRow
AfterSelChange
AfterSelRangeChange
AfterSort
BeforeAddDataRow
BeforeAddFile
BeforeAttachFile
BeforeCheckRow
BeforeDeleteDataRow
BeforeDragColumn
BeforeDrawRow
BeforeFilter
BeforeFreezeColumn
BeforeLoad
BeforeOpenFile
BeforeResizeColumn
BeforeResizeRow
BeforeSaveDataRow
BeforeSelChange
BeforeSelRangeChange
BeforeShowContextMenu
BeforeSort
CellButtonClick

ChangeEdit
Click
CurrentChanged
DataColChanged
DataColChanging
DataRowAdded
DataRowAdding
DataRowDeleted
DataRowDeleting
DataRowLockedChanged
DataRowLockedChanging
DoubleClick
DrawCell
KeyDown
KeyDownEdit
KeyPressEdit
KeyUp
KeyUpEdit
PositionChanged
PrepareEdit
SetupEditor
StartEdit
ValidateEdit
MouseEnterCell
MouseLeaveCell

这些属性都是逻辑型，如果要使某个全局表事件对该 DataTable 有效，只需将对应的属性设为 True 即可。
这些属性中，只有 CurrentChanged、PositionChanged、AfterSelChange、AfterSelRangeChange 的默认值为 True，这也就是为什么默认只有这四个全局表事件有效的原因。

例如要使全局表事件 AfterEdit 事件对表 A 有效，只需：

```
DataTables("表 A").GlobalHandler.AfterEdit = True
```

要使全局表事件 AfterEdit 事件对所有表有效，只需：

```
For Each dt As DataTable In DataTables
    dt.GlobalHandler.AfterEdit = True
Next
```

应用实例

假定项目中有一个名为“录入员”的用户，希望该用户只能修改新增加的行，不能修改已经保存的行，且这个限制对所有表有效。
首先我们应该在全局表事件 PrepareEdit 加入代码：

```
If e.Row.DataRow.RowState <> DataRowState.Added Then
    If User.Name = "录入员"
        e.Cancel = True
    End If
End If
```

然后在项目事件 Initialize 加入代码：

```
For Each dt As DataTable In DataTables
    dt.GlobalHandler.PrepareEdit = True
Next
```

这样我们的任务就已经完成，如果没有全局表事件，这将是一个很繁琐的任务，特别是表很多的时候。

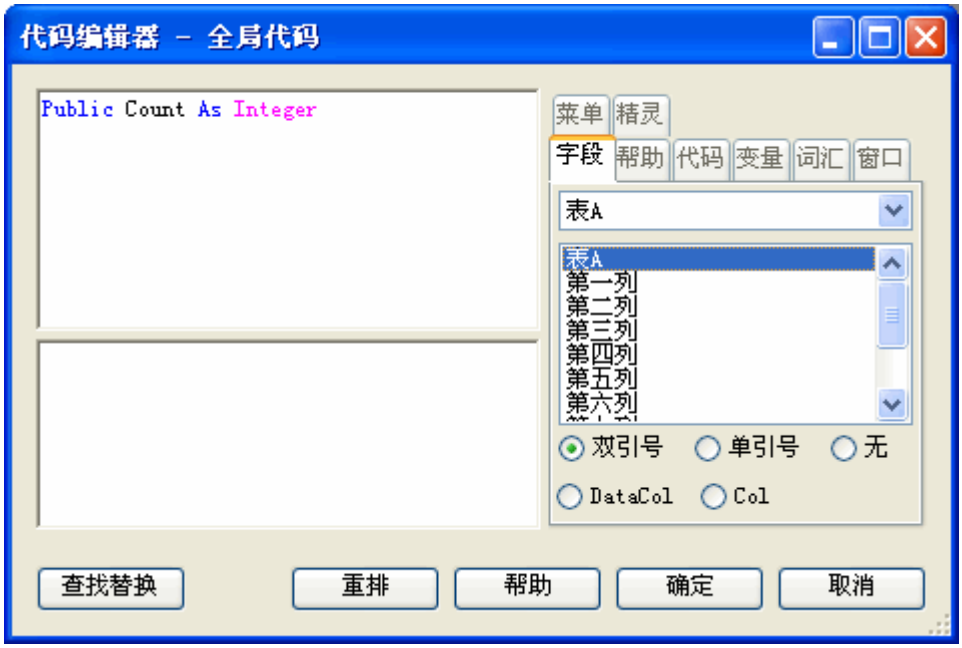
4.4.2.4 事件实例

4.4.2.4.1 使用全局变量

假定我们规定表 A 每天只能增加 500 行，我们分别看看如何用 Public 变量和 Var 变量来完成这个任务。

使用 Public 变量

1、在全局代码定义一个名为 Count 的整数型变量，用于记录已经增加的行数：



2、在表 A 的 DataRowAdded 事件中加入代码：

```
Count = Count + 1
```

这样每增加一行，变量 Count 就会增加 1

3、然后在表 A 的 DataRowDeleted 事件中加入代码：

```
Count = Count - 1
```

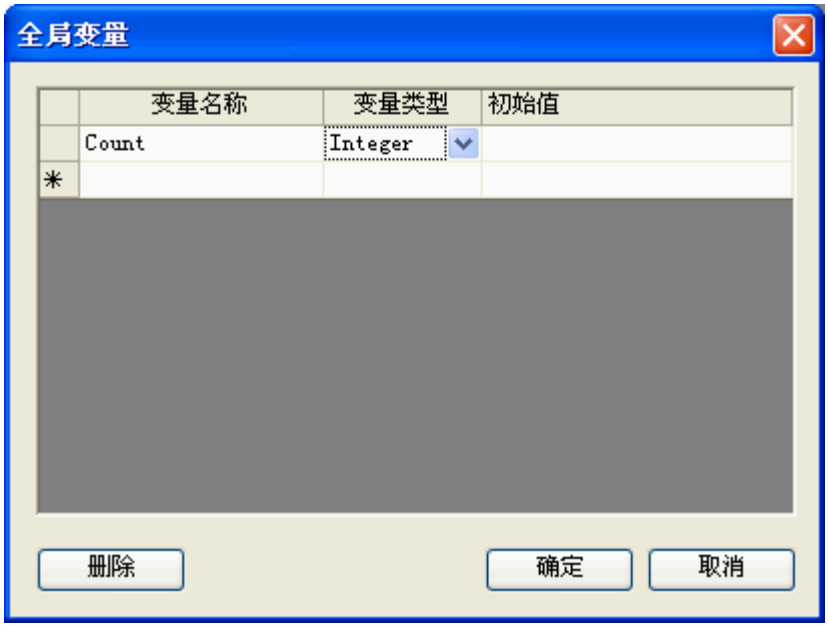
这样每删除一行，变量 Count 就会减 1

4、最后在表 A 的 BeforeAddDataRow 事件中加入代码：

```
If Count >= 500 Then '如果已经增加了 500 行
    Messagebox.Show("每天只能在表 A 中增加 500 行!","提示")
    e.Cancel = True '那么取消增加行
End If
```

使用 Var 变量

1、在全局变量管理窗口，加入一个名为 Count 的整数型 Var 变量，用于记录已经增加的行数：



2、在表 A 的 DataRowAdded 事件中加入代码：

```
Vars("Count") = Vars("Count") + 1
```

这样每增加一行，变量 Count 就会增加 1

3、然后在表 A 的 DataRowDeleted 事件中加入代码：

```
Vars("Count") = Vars("Count") - 1
```

这样每删除一行，变量 Count 就会减 1

4、最后在表 A 的 BeforeAddDataRow 事件中加入代码：

```
If Vars("Count") >= 500 Then '如果已经增加了 500 行
    Messagebox.Show("每天只能在表 A 中增加 500 行!","提示")
    e.Cancel = True '那么取消增加行
End If
```

这个例子并没有多少实际的意义，重新打开项目，我们又可以增加 500 行了，不过这个例子很好地说明了如何使用全局变量在不同的事件中传递和共享数据。

如果要跟踪 Var 变量值的变化，可以使用事件：GlobalVariableChanged

4.4.2.4.2 使用静态变量

我们知道，在事件代码运行过程中，普通变量的值并不会保存，下次运行事件代码时，变量会重新回到初始值。

虽然我们可以用全局变量保存值，以便下次调用，但是全局变量主要用于不同事件之间的数据交换，资源"开销"较大。

如果不需要和其它事件交换数据，仅仅是为了保存数据，以便下次运行事件代码的时候调用，那么请使用静态变量。

和普通变量一样，静态变量只能在定义它的事件代码中访问，但是静态变量的值能够保存到下次运行事件代码。

声明静态变量的语法是：

```
Static 变量名 As 数据类型
```

例如：

```
Static Name As String
```

静态变量通常用于标记。

示例一

为了说明普通变量和静态变量的差别，请在项目事件 MainTableChanged 加入如下代码：

```
Dim Val1 As Integer '定义一个普通变量 Val1
Static Val2 AS Integer '定义一个静态变量 Val2
Val1 = Val1 + 1
Val2 = Val2 + 1
MessageBox.Show("Val1 = " & Val1 & " Val2 = " & Val2)
```

设置完成后，反复切换主表，你会发现 Val1 的值始终是 1，而 Val2 的值在每次切换主表后都会加 1。

这是因为 Val1 是一个普通变量，虽然每次运行时也给 Val1 的值加上了 1，但是增加后的值，并不会保存到下次运行。

而 Val2 是一个静态变量，每次运行给 Val2 的值加 1，这个增加后的值会保存到下次运行，所以每次都是在上一次加 1 的基础上再加 1。

示例二

假定订单表中有一个窗口“订单筛选”，我们希望第一次进入订单表的时候，能够自动打开这个窗口。

问题来了，怎么才能知道是第一次进入订单表呢?

很简单，只需将项目的 MainTableChanged 事件设为：

```
Static i As Integer '定义静态变量 i
If MainTable.Name = "订单" Then '如果进入的是订单表
    If i = 0 Then '假如 i 等于 0
        i = 1 '那么将 i 的值设为 1
        Forms("订单筛选").Show() '打开窗口
    End If
End If
```

代码原理很简单，第一次进入订单表的时候，静态变量 i 的值等于 0，所以打开窗口，而之后进入订单表的时候，由于 i 的值已经等于 1，所以不会打开窗口。

示例三

假定规定某个表每天只能增加 500 行，我们只需该表的 BeforeAddDataRow 事件的代码设为：

```
Static i As integer '定义一个静态变量
i = i + 1 '每次增加行,静态变量的值就加 1
If i > 500 Then '如果增加了 500 行
    Messagebox.Show("每天只能在表 A 中增加 500 行!","提示")
    e.Cancel = True '那么取消增加行
End If
```

4.4.2.4.3 使用事件进行计算

4.4.2.4.3.1 快速入门

假定有 A、B、C 三列，这三列都是数值型，其中 C 列等于 A 列加上 B 列。

对于这种简单的计算，我们完全可以用表达式完成，假定 C 列是一个表达式列，只需将该列的表

达式设为：

[A] + [B]

但是本节我们不用表达式，而是用代码来实现同样的计算。

任何一个数据列的内容发生变化，都会触发表的 DataColChanged 事件。

我们可以在 DataColChanged 中设置代码，实现列与列之间的计算。

需要注意的是：

1、如果某一列需要通过事件代码来计算求值，那么该列必须是一个数据列，不能是一个表达式列。

2、表达式列的内容发生变化，是不会触发 DataColChanged 事件的。

我们用这样一个最简单的列子，是为了让大家快速掌握如何用代码来实现计算。

首先我们得确保 A、B、C 三列都是数据列，而不是表达式列，然后将这个表的 DataColChanged 事件设为：

```
e.DataRow("C") = e.DataRow("A") + e.DataRow("B")
```

上面的代码设置完成后，只要在 A 列或 B 列输入内容，C 列就会自动显示这两列之和。很简单，是不是？

不过我们还得完善一下代码，因为任何一列发生变化，都会触发 DataColChanged 事件，从而重算 C 列的内容。

显然，这样的代码，效率太低，最好加一个判断，只有 A 列或 B 列的内容发生变化，才重算 C 列：

```
If e.DataCol.Name = "A" OrElse e.DataCol.Name = "B" Then
    e.DataRow("C") = e.DataRow("A") + e.DataRow("B")
End If
```

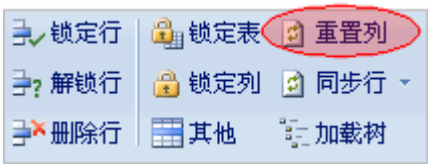
现在的问题是，上面的代码只对新输入的数据有效，这是显然的，因为 DataColChanged 本来就只会在某列内容发生变化后执行。

那么如何要求原有的行，按照新设置的代码，重新计算 C 列的值呢？很简单，只需选定 A 列或 B 列中的任何一列，然后在“数据表”功能区“列相关”功能组，单击下面的“重置”按钮：



即可重算所有行的 C 列值。

菜单的“日常工作”功能区，也有重置列的按钮：



重置命令相当于“欺骗”性地通知系统：某一列(被重置的列)的内容发生了变化；系统收到这个虚假通知后，开始执行 DataColChanged 事件代码，间接实现了重算的目的。

需要注意的是，不能选择 C 列来执行重置命令，必须选择 A 列或 B 列中的一个，这是因为我们在 DataColChanged 中加入了判断条件，只有这两列的内容发生变化，才会重算 C 列的值。

我们也可以通过编码来重置某一列，例如重置 B 列：

```
DataTables("表名").DataCols("B").RaiseDataColChanged()
```

假定这个表包括两个计算关系，C 列等于 A 列加上 B 列，F 列等于 D 列加上 E 列，按照前面的编码方式，我们只需将 DataColChanged 事件改为：

```
If e.DataCol.Name = "A" OrElse e.DataCol.Name = "B" Then
    e.DataRow("C") = e.DataRow("A") + e.DataRow("B")
ElseIf e.DataCol.Name = "D" OrElse e.DataCol.Name = "E" Then
```

```
    e.DataRow("F") = e.DataRow("D") + e.DataRow("E")
End If
```

上面的代码看上去有点繁多，我们可以改用 Select Case 语句：

```
Select Case e.Datacol.Name
    Case "A","B"
        e.DataRow("C") = e.DataRow("A") + e.DataRow("B")
    Case "D","E"
        e.DataRow("F") = e.DataRow("D") + e.DataRow("E")
End Select
```

显然，在存在多个计算关系的时候，或者多列参与计算的时候，用 Select Case 语句更简洁一些。

上面的代码反复使用了 e.DataRow，我们可以用一个变量来代替它，使得代码更加简洁：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "A","B"
        dr("C") = dr("A") + dr("B")
    Case "D","E"
        dr("F") = dr("D") + dr("E")
End Select
```

4.4.2.4.3.2 空值的判断

假定某个表的金额列，由单价、数量、折扣三列计算得出，此外年、月两列，由日期列计算得出。

根据我们之前的经验，可以将该表的 DataColChanged 事件设置为：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "数量","单价","折扣"
        dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
    Case "日期"
        dr("年") = dr("日期").Year
        dr("月") = dr("日期").Month
End Select
```

上面的代码看上去很完美，金额、年、月三列都能自动计算得出，可真的没问题吗？

我们来测试一下，你先在某行输入日期，年、月两列的值确是自动计算得出了，可是如果你删除日期列的内容，按道理年、月列的内容应该为空。

但事实上，年、月都会等于 1， 原因何在呢？

这是因为在日期列为空的时候，通过行返回的日期值并不是空值，而是等于: #01/01/0001#

同样如果没有输入数量和单价，金额也不会为空，而是等于 0，因为数值列为空的时候，其返回的值也不是空值，而是 0。

所以为了“完美”，我们将 DataColChanged 事件代码改为：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "数量","单价","折扣"
        If dr.IsNull("数量") OrElse dr.IsNull("单价") Then '如果数量或单价为空
            dr("金额") = Nothing
        Else
```

```
            dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
        End If
    Case "日期"
        If dr.IsNull("日期") Then '如果日期列为空
            dr("年") = Nothing
            dr("月") = Nothing
        Else
            dr("年") = dr("日期").Year
            dr("月") = dr("日期").Month
        End If
End Select
```

这样输入日期，年、月列的值能自动计算得出，删除日期，年、月列也会自动被清空。

需要注意空值的判断方法，例如有人写出了类似下面这样的代码：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "数量","单价","折扣"
        If dr("数量") Is Nothing OrElse dr("单价") Is Nothing Then
            dr("金额") = Nothing
        Else
            dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
        End If
    Case "日期"
        If dr("日期") Is Nothing Then
            dr("年") = Nothing
            dr("月") = Nothing
        Else
            dr("年") = dr("日期").Year
            dr("月") = dr("日期").Month
        End If
End Select
```

然后很疑惑，为什么上述代码没有正确判断空值？例如在没有输入日期的时候，年、月两列并不为空，而是等于 1。

这是因为 dr("列名") Is Nothing 是不能用来判断某列是否为空的，道理前面已经讲了，对于数值列，如果值为空，dr("列名")返回的是 0，如果是日期列，则返回#01/01/0001#，显然这都不是空值，所以我们只能用 IsNull 方法判断某列是否为空。

实际上任何类型的列，包括字符型列，都不能用 dr("列名") Is Nothing 来判断是否是空值，因为对于字符列，如果值为空，dr("列名")返回的是空字符""，而不是空值 Nothing。

总之，判断某列是否为空，不管列类型是什么，最好使用 IsNull 方法，这是最可靠的方式。

4.4.2.4.3.3 零值的处理

假定某个表的金额列，由单价、数量、折扣三列计算得出，可以将该表的 DataColChanged 事件代码设置为：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
```

```
    Case "数量","单价","折扣"
        dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
End Select
```

显然如果我们只输入数量、单价中的一个，金额列的计算结果等于 0。

即使我们没有输入数量、单价、折扣中的任何一个值，如果重置其中某一列，金额列的内容同样也会等于 0。

如果你希望只有同时输入了单价和数量的情况下，才计算金额，避免金额出现 0 值，可以将 DataColChanged 事件代码改为：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "数量","单价","折扣"
        If dr.IsNull("数量") OrElse dr.IsNull("单价") Then
            dr("金额") = Nothing
        Else
            dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
        End If
End Select
```

这样如果数量或者单价中任何一个为空，金额也会自动清空，而不会出现 0 值。

如果你希望在任何情况下，某一列都不能出现 0 值，那么代码会更简单，同样用金额列为例，只需将 DataColChanging 事件代码设为：

```
If e.DataCol.Name = "金额" Then
    If e.NewValue = 0 Then
        e.NewValue = Nothing
    End if
End If
```

注意上面的代码设置在 DataColChanging 事件中，而不是 DataColChanged 事件中。这样可以在金额列的值发生变化之前进行拦截干预，判断新的值是否为 0，如果为 0，则将新值设为 Nothing，这样金额列的值就永远不可能为 0 了。

如果你希望表中所有的数据列，都不能出现零值，可以将 DataColChanging 事件代码设为：

```
If e.DataCol.IsNumeric Then
    If e.NewValue = 0 Then
        e.NewValue = Nothing
    End if
End If
```

4.4.2.4.3.4 更多的例子

示例一

员工表中有身份证号码、出生日期列，出生日期根据身份证号码计算得出。

为了自动计算出生日期，可以在员工表的 DataColChanged 事件中加入下面的代码：

```
If e.DataCol.Name = "身份证号码" Then '是身份证号码发生变化吗?
    If e.DataRow.IsNull("身份证号码") Then '身份证号码是否为空
        e.DataRow("出生日期") = Nothing '如果为空,则清除出生日期
    Else
```

```
        '否则从身份证号码列中提取出生日期
        e.DataRow("出生日期") = ReadBirthday(e.DataRow("身份证号码"))
    End If
End If
```

如果表中已经输入数据，那么在加入上述代码后，选择身份证号码列，然后在“数据表”功能区“列相关”功能组，单击下面的“重置”按钮：



即可计算出所有行的出生日期，今后新增数据或者修改身份证号码，出生日期可以自动计算得出，无需人工干预。

示例二

再例如下面的表格，完成日期等于起始日期加上执行天数：

	任务	起始日期	执行天数	完成日期
1	A	2008-01-31	2	2008-02-02
2	B	2008-02-20	5	2008-02-25
3	C	2008-02-28	3	2008-03-02
4	D	2008-03-15	8	2008-03-23
5	E	2008-04-02	20	2008-04-22
6	F	2008-04-08	7	2008-04-15

我们可以在该表的 DataColChanged 事件中加入下面的代码：

```
If e.DataCol.Name = "起始日期" OrElse e.DataCol.Name = "执行天数" Then
    If e.DataRow.IsNull("起始日期") OrElse e.DataRow.IsNull("执行天数") Then
        e.DataRow("完成日期") = Nothing
    Else
        e.DataRow("完成日期") = e.DataRow("起始日期").AddDays(e.DataRow("执行天数"))
    End If
End If
```

上面的代码多处使用了 e.DataRow 和 e.DataCol，为了让代码看上去简洁一点，我们可以用变量来代替它们：

```
Dim dc As DataCol = e.DataCol
Dim dr As DataRow = e.DataRow
If dc.Name = "起始日期" OrElse dc.Name = "执行天数" Then
    If dr.IsNull("起始日期") OrElse dr.IsNull("执行天数") Then
        dr("完成日期") = Nothing
    Else
        dr("完成日期") = dr("起始日期").AddDays(dr("执行天数"))
    End If
End If
```

采用何种编码风格，完全看个人的偏好。

示例三

上面的例子是根据开始日期和执行天数计算完成日期，如果反过来，要根据开始日期和完成日期计算执行天数，可以将 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.Name
```

```
    Case "开始日期","完成日期"
        Dim dr As DataRow = e.DataRow
        If dr.IsNull("开始日期") OrElse dr.IsNull("完成日期") Then
            dr("执行天数") = Nothing
        Else
            Dim tp As TimeSpan = dr("完成日期") - dr("开始日期")
            dr("执行天数") = tp.TotalDays
        End If
End Select
```

示例四

下图是一个成绩表，显然总分列是可以用表达式列来计算的，但是这次看看如何使用代码来计算总分。

	班级	学号	姓名	语文	数学	英语	物理	生物	地理	政治	历史	总分
1	13	13	兰亦帆	89	98	111	96	92	93	100	100	779
2	17	58	韩丽红	89	84	114	95	100	95	100	96	773
3	1	42	杨倩	87	88	111	96	96	98	99	94	769
4	13	36	杨林	87	91	116	99	96	82	100	96	767
5	13	53	董雨鑫	90	87	112	91	94	93	100	99	766
6	14	44	郑蔚然	86	92	111	90	94	98	96	97	764
7	17	11	邓少卿	89	91	117	98	96	93	98	80	762
8	13	46	郑哲文	90	89	113	98	92	96	100	84	762
9	3	19	付玉	82	85	116	98	94	94	98	87	754
10	7	26	宋琳	90	85	117	94	90	92	100	85	753

在该表的 DataColChanged 事件中加入下面的代码即可：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "语文","数学","英语","物理","生物","地理","政治","历史"
        dr("总分") = dr("语文")+dr("数学")+dr("英语")+dr("物理")+dr("生物")+dr("地理")+dr("政治")+dr("历史")
End Select
```

上面的例子告诉我们，如果有某一列由很多列计算得来，那么用 Select Case 语句来判断列名，会显得简洁很多。

4.4.2.4.3.5 一定要判断列名

不少人，特别是初学编程的人，没有在 DataColChanged 事件中判断列名的习惯，例如为了在订单表计算金额，往往只是在 DataColChanged 简单地写入代码：

```
Dim dr As DataRow = e.DataRow
dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
```

这样可能会带来严重的后果，首先会导致重复的计算，因为数量、单价或折扣发生变化后，触发 DataColChanged 事件，计算得出新的金额，而金额列内容的变化，会再次触发 DataColChanged 事件，使得金额被重新计算一次；实际上其他不相关列，例如日期、客户、产品等列内容发生变化后，同样会触发 DataColChanged 事件，导致金额列被重新计算。

显然这样的代码，效率实在是太低，而且在机缘巧合的情况下，很可能会出现死循环，导致程序崩溃。

所以我们一定要养成良好的习惯，在编写 DataColChanged 事件代码时，判断发生变化列的名称，只有相关列发生变化时，才进行特定的重算工作。

例如：

```
Dim dr As DataRow = e.DataRow
Select Case e.DataCol.Name
    Case "数量","单价","折扣"
        dr("金额") = dr("数量") * dr("单价") * (1 - dr("折扣"))
End Select
```

这节的内容不仅仅针对 DataColChanged，实际上对于任何表事件都有效，例如 DataColChaning、PrepareEdit、ValidateEdit、AfterEdit 等等。

4.4.2.4.3.6 判断列名的技巧

有的时候某一列可能会参与多列的计算。

例如某个表的 L 列由 A、C、D 三列计算得出，M 列有 A、B、E 三列计算得出，N 列由 B、C、E 三列计算得出，Q 列由 F 列计算得出，有的用户会如下设置代码：

```
Select Case e.DataCol.Name
    Case "A","C","D"
        'L 列的计算代码
    Case "A","B","E"
        'M 列的计算代码
    Case "B","C","E"
        'N 列的计算代码
    Case "F"
        'Q 列计算代码
End Select
```

上面的代码存在严重的问题，因为如果有多个 Case 语句匹配条件，则只会执行第一个匹配条件的 Case 语句，然后直接跳到 End Select 语句。

假定我们修改 B 列，显然第二个 Case 语句和第三个 Case 语句都匹配了条件，但是只有第二个 Case 语句会执行，所以修改 B 列后，只会重算 M 列，而不会像我们预期的那样，重算 M 和 N 两列。

要避免这种情况，可以采用多个 Select Case 语句：

```
Select Case e.DataCol.Name
    Case "A","C","D"
        'L 列的计算代码
End Select
Select Case e.DataCol.Name
    Case "A","B","E"
        'M 列的计算代码
End Select
Select Case e.DataCol.Name
    Case "B","C","E"
        'N 列的计算代码
End Select
If e.DataCol.Name = "F"
```

```
    'Q 列计算代码
End If
```

4.4.2.4.3.7 表达式和代码的混合使用

下图是一个工资表：

	姓名	基本工资	绩效工资	加班费	其他	应付工资	扣养老	扣医疗	扣其它	应税收入	所得税	实发
1	诸葛亮	1500	2000	500	0	4000	100	50	0	3850	212.5	3637.5
2	赵子龙	1000	1300	200	0	2500	50	50	0	2400	55	2345

在上表中，应付工资、应税收入、实发工资三列是通过其它列计算得出的，在设计表的时候，这三列请用表达式列，而所得税列虽然也是由其它列计算得出，但是所得税的计算较为复杂，无法通过表达式计算得出，所以所得税列不能是表达式列，必须是数据列，所得税列将通过代码来计算得出。

首先将应付工资列的表达式设为：

```
IsNull([基本工资],0) + IsNull([绩效工资],0) + IsNull([加班费],0) + IsNull([其他],0)
```

还记得 IsNull 吗？ 用这个函数判断表达式的内容是否为空，如果不为空，则返回表达式的值，否则返回 0。

应税收入列的表达式为：

```
IsNull([应付工资],0) - IsNull([扣养老],0) - IsNull([扣医疗],0) - IsNull([扣其它],0)
```

实发工资列的表达式为：

```
[应税收入] - IsNull([所得税],0)
```

假定起征点为 3500 元，为了计算个人所得税，在 DataColChanged 事件中设置如下代码:

```
'如果发生变化的是所得税列之外的数值型列，才重新计算所得税。
If e.DataCol.Name <> "所得税" AndAlso e.DataCol.IsNumeric Then
    e.DataRow("所得税") ＝ InTax(e.Datarow("应税收入") - 3500)
End If
```

既然所得税列的内容是根据应税收入列的内容计算得出的，为什么代码第一行的判断条件不是：

```
If e.DataCol.Name ＝ "应税收入" Then
```

而是：

```
If e.DataCol.Name <> "所得税" AndAlso e.DataCol.IsNumeric Then
```

这是因为应税收入列是一个表达式列，而表达式列的内容是通过其他列计算得出的，并不会触发 DataColChanged 事件。

4.4.2.4.3.8 详解 RaiseDataColChanged

在进一步学习之前，有必要详细介绍一下 RaiseDataColChanged 方法。

这是 DataCol 的一个方法，用于强行触发该列的 DataColChanged 事件。

语法

RaiseDataColChanged()

或

RaiseDataColChanged(DataRow)

或

RaiseDataColChanged(Filter)

DataRow：可选参数，单单针对此行触发 DataColChanged 事件。

Filter: 可选参数，一个条件表达式，针对符合此条件的行触发 DataColChanged 事件。

如果没有给 RaiseDataColChanged 方法指定参数，则针对所有行触发 DataColChanged 事件。

示例

例如年龄列由出生日期计算得出，我们将 DataColChanged 事件设为：

```
If e.DataCol.Name = "出生日期" Then
    If e.DataRow.IsNull("出生日期") Then '如果没有输入出生日期
        e.DataRow("年龄") = Nothing '则清空年龄
    Else
        '否则计算年龄
        e.DataRow("年龄") = Date.Today.Year - e.DataRow("出生日期").Year
    End If
End If
```

如果到了明年，需要重新计算年龄列，但是除非你重新输入出生日期，否则上面的代码是不会重算年龄的。

不过我们可以在 AfterOpenProject 事件中，设置下面的代码：

```
DataTables("员工").DataCols("出生日期").RaiseDataColChanged()
```

这样每次重新打开项目，就会强行针对出生日期列触发 DataColChanged 事件，效果就像你重新输入了出生日期一样 。

执行某列的 RaiseDataColChanged 方法，等于“欺骗”性地通知系统：这一列(被重置的列)的内容发生了变化；系统收到这个虚假通知后，开始执行 DataColChanged 事件代码，间接实现了重算的目的。

显然，为了重新计算年龄，只能执行出生日期列的 RaiseDataColChanged 方法，因为只有出生日期列的内容发生变化，年龄才会被重算。

如果并不需要每次打开项目都重算一次，那么就没有必要将上面的代码设置在 AfterOpenProject 事件中，可以将代码设置在一个按钮中，需要重算的时候，单击这个按钮即可。

其实我们并不需要编码去完成这样的任务，首先选择出生日期列，然后在“数据表”功能区“列相关”功能组，单击下面的“重置”按钮，即可针对出生日期列触发 DataColChanged 事件，计算出最新的年龄：



但是一些要求自动更新的场合，则只能使用 RaiseDataColChanged 方法了。

4.4.2.4.3.9 时段列与计算

我们知道双精度小数型的列，可以扩展为时段型。
但是这个时段型列的值，并不是真正的时段(TimeSpan)，而是一个以秒为单位的数值，在使用表达式或者代码进行计算的时候，需要特别注意这一点。

示例一

下面的“用时”列是一个时段列，每分钟字数是一个表达式列，根据字数和用时，计算出每分钟字数：

	用时	字数	每分钟字数
1	00:01:30	90	60
2	00:02	100	50
3	00:02:10	195	90

在参与计算的时候，用时列会转换秒数，例如“00:2:10”,表示 2 分 10 秒，转换为秒数就是 130。
所以每分钟字数列的表达式应该设置为：[字数] / [用时] * 60

示例二

时段列可以进行加减，例如下图的超出时间列是个表达式列，其表达式为: [实际用时] - [标准时间]

	任务	标准时间	实际用时	超出时间
1	A	12	12:29	00:29
2	B	12:30	14:50	02:20
3	C	12:30:45	13:21	00:50:15
4	D	12:30:45	12:30:59	00:00:14
5	E	00:30	00:36	00:06
6	F	00:30:45	00:30:45	00
7	G	00:00:45	00:01:45	00:01
8	H	00:00:00 800	00:00:01 200	00:00:00 400
9	J	198	201:45	03:45
10	K	198:45	199:46	01:01

示例三

下图的起始时间和结束时间是日期时间型列，日期时间格式为：DateTime。
本次用时是一个双精度小数列，扩展列类型为：时段型

	起始时间	结束时间	本次用时
1	2012-05-01 08:30	2012-05-03 16:40	56:10
2	2012-05-07 09:00	2012-05-09 06:30	45:30
3	2012-05-08 12:00	2012-05-12 08:00	92

为了根据起始时间和结束时间，计算出本次用时，可以将该表的 DataColChanged 时间代码设置为：

```
Select Case e.DataCol.name
    Case "起始时间","结束时间"
        Dim dr As DataRow = e.DataRow
        If dr.IsNull("起始时间") Or dr.IsNull("结束时间") Then
            dr("本次用时") = Nothing
        Else
            Dim tp As TimeSpan
            tp = dr("结束时间") - dr("起始时间")
```

```
        dr("本次用时") = tp.TotalSeconds() '真正的时段型数据要转换为秒数,才能存入时段列
    End If
End Select
```

示例四

下图的起始时间和计划结束时间是日期时间型列，日期时间格式为：DateTime。
计划用时是一个双精度小数列，扩展列类型为：时段型

	起始时间	计划用时	计划结束时间
1	2012-05-01 08:30	58:30	2012-05-03 19:00
2	2012-05-07 09:00	45:40	2012-05-09 06:40
3	2012-05-08 12:00	92:30	2012-05-12 08:30

为了根据起始时间和计划用时，算出计划结束时间，可以将该表的DataColChanged时间代码设置为：

```
Select Case e.DataCol.name
    Case "起始时间","计划用时"
        Dim dr As DataRow = e.DataRow
        If dr.IsNull("起始时间") Or dr.IsNull("计划用时") Then
            dr("计划结束时间") = Nothing
        Else
            Dim dt As Date = dr("起始时间")
            dr("计划结束时间") = dt.AddSeconds(dr("计划用时")) '注意时段型列是以秒为单位参与计算的
        End If
End Select
```

4.4.2.4.3.10 跨表引用

假定产品表和订单表已经通过品名建立了关联，关联的名称为“产品_订单”。
产品表和订单表都有单价列，其中订单表中的单价数据来自于产品表。
通常我们会在订单表新建一个名为“单价”的表达式列，将其表达式设为：
Parent(产品_订单).单价。
这样在订单表输入品名后，单价列就会自动引用产品表中对应产品的单价。
但是采用表达式列会有以下棘手的问题：
1、如果产品表没有某产品的资料，需要在订单表中手工输入单价的时候，你会发现你是没有办法在订单表的单价列输入数据的。
2、如果这是一个特殊的订单，需要一个例外的价格，同样你会发现，订单表的单价数据是无法修改的。
3、如果从某一天开始，需要按照新的价格销售产品，于是你修改产品表的单价数据，但是修改后你会发现订单表原有订单的单价数据会全部自动更新为新的单价，显然这不是我们所希望的。
上述问题不是表达式列所能解决的，更甚的是，我们可能因为某些特殊原因，并没有在产品表和订单表之间建立关联，此时采用表达式列来引用产品单价根本就无从谈起。
所以我们有必要介绍另一种跨表引用数据的方法，这个方法不使用表达式，而且不管这两个表之

间是否建立了关联，此方法应该同样有效。

以上面介绍的订单表引用产品表的单价为例，首先要确保订单表中的单价列不是表达式列，而是标准的数据列，然后在订单表的 DataColChanged 事件中设置如下代码：

```
1  If e.DataCol.Name = "品名" Then '如果内容发生变动的是品名列
2      If e.NewValue Is Nothing Then '如果新值是空白,也就是品名列的内容为空
3          e.DataRow("单价") = Nothing '那么清空此行单价列的内容
4      Else
5          Dim dr As DataRow
6          '否则在产品表查找同名的产品行,将找到的行赋值给变量 dr
7          dr = DataTables("产品").Find("[品名] = '" & e.NewValue & "'")
8          If dr IsNot Nothing Then '如果找到了同名的产品行,也就是 dr 不是 Nothing
9              e.DataRow("单价") = dr("单价")
10         End If
11     End If
12 End If
```

为便于讲述，我给上面的代码加上了行号，代码的原理为：

第 1 行代码判断发生变动的是否是品名，如果是品名，第 2 行代码判断品名是否为空，如果品名为空，第 3 行代码清除此订单的单价，如果品名不为空，则执行第 7 行代码从产品表找出此产品所在的行， 并赋值给变量 dr，如果在产品表找到此产品（也就是变量 dr 不为 Nothing)，就执行第 9 行代码，将订单的单价设置为此产品的单价。

注意第 7 行代码，其 Find 方法的条件表达式是动态合成的:

```
dr = DataTables("产品").Find("[品名] = '" & e.NewValue & "'")
```

条件表达式分为三部分，用运算符&连接起来；如果输入的品名为"PD01"，那么三部分组合起来的结果是：

```
[品名]  = 'PD01'
```

也就说第 7 行代码等效于：

```
dr = DataTables("产品").Find("[品名]  = 'PD01'")
```

这是我们第二次介绍动态合成表达式，这是基本功，后面还会有专门的一节进行介绍，不掌握是不行的。

上述代码实现了：

1、在订单表中的某行，首次输入品名，或者输入一个不同的品名，那么会自动从产品表找出对应产品的单价，填入单价列中；

2、如果删除订单表某行品名列的内容，单价列的数据会自动清空；

3、订单表的单价列数据可以手工修改；

4、如果修改产品表的单价数据，并不会影响订单表的现有数据，但是新增行会自动获得新的单价；

5、如果某一订单要重新获得新的单价，在品名单元格执行一次剪切粘贴操作即可，原因是什么，请您自己思考；

6、不管是否在产品表和订单表之间建立了关联，都能正常计算，这样可以避免因为关联太多而影响系统速度。

如果产品表和订单表是通过品名、型号、规格这三列联系起来的(不管是否建立了关联)，在订单表输入品名、型号、规格后，再从产品表找出对应产品的单价，填入单价列中，那么订单表的DataColChanged 事件可如下设置代码：

```
Select Case e.DataCol.Name
    Case "品名","型号","规格"
        Dim dr As DataRow = e.DataRow
        Dim pr As DataRow
        If dr.IsNull("品名") OrElse dr.IsNull("型号") OrElse dr.IsNull("规格") Then
            dr("单价") = Nothing
        Else
            Dim filter As String
            filter = "品名 = '" & dr("品名") & "' And 型号 = '" & dr("型号") & "' And 规格 = '" & dr("规格") & "'"
            pr = DataTables("产品").Find(filter)
            If pr IsNot Nothing Then
                dr("单价") = pr("单价")
            End If
        End If
End Select
```

如果订单表有多列数据来自于产品表，一样可以采用类似的设计。
例如，假定产品表和订单表通过产品编号联系起来（不管是否建立了关联），订单表输入产品编号后，品名、型号、规格、单价四列内容从产品表自动继承输入。
为实现此目的，订单表的 DataColChanged 事件代码应设置为：

```
If e.DataCol.Name = "产品编号" Then
    If e.NewValue Is Nothing Then
        e.DataRow("品名") = Nothing
        e.DataRow("型号") = Nothing
        e.DataRow("规格") = Nothing
        e.DataRow("单价") = Nothing
    Else
        Dim dr As DataRow
        dr = DataTables("产品").Find("[产品编号] = '" & e.NewValue & "'")
        If dr IsNot Nothing
            e.DataRow("品名") = dr("品名")
            e.DataRow("型号") = dr("型号")
            e.DataRow("规格") = dr("规格")
            e.DataRow("单价") = dr("单价")
        End If
    End If
End If
```

如果有十几列甚至更多列的数据需要继承，上面代码可能会显得过于繁琐，为此我们可以考虑将

代码改写为：

```
If e.DataCol.Name = "产品编号" Then
    Dim nms() As String = {"品名","型号","规格","单价"}
    If e.NewValue Is Nothing Then
        For Each nm As String In nms
            e.DataRow(nm) = Nothing
        Next
    Else
        Dim dr As DataRow
        dr = DataTables("产品").Find("[产品编号] = '" & e.NewValue & "'")
        If dr IsNot Nothing
            For Each nm As String In nms
                e.DataRow(nm) = dr(nm)
            Next
        End If
    End If
End If
```

这样不管有多少列，我们都只需修改下面这样代码，增加和删除列名即可：

```
Dim nms() As String = {"品名","型号","规格","单价"}
```

这种编码方式在处理大量重复任务的时候很实用，应该仔细体会掌握。

4.4.2.4.3.11 跨表引用的自动更新

上一节提到，如果在订单表的 DataColChanged 事件中加入如下代码：

```
If e.DataCol.Name = "品名" Then '如果内容发生变动的是品名列
    If e.NewValue Is Nothing Then '如果新值是空白,也就是品名列的内容为空
        e.DataRow("单价") = Nothing '那么清空此行单价列的内容
    Else
        Dim dr As DataRow
        '否则在产品表查找同名的产品行,将找到的行赋值给变量 dr
        dr = DataTables("产品").Find("[品名] = '" & e.NewValue & "'")
        If dr IsNot Nothing Then '如果找到了同名的产品行,也就是 dr 不是 Nothing
            e.DataRow("单价") = dr("单价")
        End If
    End If
End If
```

可以实现在订单表中输入品名后，能自动从产品表中找出对应产品的单价，填入到单价列中。

如果修改某一个产品的单价，那么该产品原有订单的单价会保持不变，只有新增的订单才会使用新的单价，这和通过表达式列来引用产品单价是不同的。

可是也有一些场合，可能会要求修改产品单价后，原有订单的产品单价也自动更新。

为此，我们可以在产品表的 DataColChanged 事件中加入如下代码：

```
If e.DataCol.Name = "单价"
    Dim Filter As String = "[品名] = '" & e.DataRow("品名") & "'"
    Dim drs As List(Of DataRow) = DataTables("订单").Select(Filter)
    For Each dr As DataRow In drs
        dr("单价") = e.DataRow("单价")
    Next
End If
```

在产品表修改某个产品的单价后，上面的代码会找出订单表中该产品的所有订单，将这些订单的单价设置为新的单价。

我们还可以将前面的代码写得更为简洁：

```
If e.DataCol.Name = "单价"
    Dim Filter As String = "[品名] = '" & e.DataRow("品名") & "'"
    DataTables("订单").ReplaceFor("单价", e.NewValue, Filter)
End If
```

不过我个人更喜欢第一种写法，虽然代码长了那么一点，但似乎也更容易理解那么一点。

再次提示一下，在上面的 DataColChanged 事件中，e.NewValue 等于 e.DataRow("单价")，因为此时新的单价已经写入表中。

所以下面这行代码：

```
DataTables("订单").ReplaceFor("单价", e.NewValue, Filter)
```

完全等效于：

```
DataTables("订单").ReplaceFor("单价", e.DataRow("单价"), Filter)
```

有条件的跨表更新

上面已经介绍了如何实现跨表引用的自动更新，可是更新有的时候是有条件的，否则还不如直接使用表达式列来引用产品单价。

例如希望在产品表中修改单价后，订单表中已经付款的订单，继续保持原单价不变，而未付款的订单采用新的单价。

为此，我们可以在产品表的 DataColChanged 事件中加入如下代码：

```
If e.DataCol.Name = "单价"
    Dim Filter As String = "[品名] = '" & e.DataRow("品名") & "' And  已付款  = False"
    DataTables("订单").ReplaceFor("单价", e.NewValue, Filter)
End If
```

再例如希望在产品表中修改单价后，对于已经锁定或者已经付款的订单，单价保持不变，其余订单则采用新的单价：

```
If e.DataCol.Name = "单价"
    Dim Filter As String = "[品名] = '" & e.DataRow("品名") & "' And  已付款  = False"
    Dim drs As List(Of DataRow) = DataTables("订单").Select(Filter)
    For Each dr As DataRow In drs
        If dr.Locked = False Then
            DataTables("订单").DataCols("品名").RaiseDataColChanged(dr)
        End If
    Next
End If
```

多列跨表更新

如果订单表有多列数据来自于产品表，一样可以采用类似的设计实现跨表更新。

例如，假定产品表和订单表通过产品编号联系起来(不一定建立了关联)，订单表输入产品编号后，品名、型号、规格、单价四列内容从产品表自动继承输入。

为实现此目的，首先订单表的 DataColChanged 事件代码应设置为：

```
If e.DataCol.Name = "产品编号" Then
    If e.NewValue Is Nothing Then
        e.DataRow("品名") = Nothing
        e.DataRow("型号") = Nothing
        e.DataRow("规格") = Nothing
        e.DataRow("单价") = Nothing
    Else
        Dim dr As DataRow
        dr = DataTables("产品").Find("[产品编号] = '" & e.NewValue & "'")
        If dr IsNot Nothing
            e.DataRow("品名") = dr("品名")
            e.DataRow("型号") = dr("型号")
            e.DataRow("规格") = dr("规格")
            e.DataRow("单价") = dr("单价")
        End If
    End If
End If
```

现在希望在产品表修改上述四列数据后，订单表能够自动更新这些列的数据，为此可以将产品表的 DataColChanged 事件设置为：

```
Select Case e.DataCol.Name
    Case "品名","型号","规格","单价"
        Dim Filter As String = "[产品编号] = '" & e.DataRow("产品编号") & "'"
        Dim drs As List(Of DataRow) = DataTables("订单").Select(Filter)
        For Each dr As DataRow In drs
            dr(e.DataCol.Name) = e.NewValue
        Next
End Select
```

或者：

```
Select Case e.DataCol.Name
    Case "品名","型号","规格","单价"
        Dim Filter As String = "[产品编号] = '" & e.DataRow("产品编号") & "'"
        DataTables("订单").ReplaceFor(e.DataCol.Name, e.NewValue, Filter)
End Select
```

可以看到不管是单列还是多列，自动更新的代码都一样的简洁。

关联表下的编码

不管是产品表和订单表之间是否建立了关联，本节之前所讲述的代码，均是适用的。

如果产品表和订单表之间已经建立了关联，那么代码可以写得稍微简洁一些，因为可以用 GetChildRows 获得某个产品的所有订单，用 GetParentRow 获得某个订单对应的产品。

示例

假定产品表和订单表通过产品编号建立了关联，订单表输入产品编号后，品名、型号、规格、单价四列内容从产品表自动继承输入。
为实现此目的，首先订单表的 DataColChanged 事件代码应设置为：

```
If e.DataCol.Name = "产品编号" Then
    If e.NewValue Is Nothing Then
        e.DataRow("品名") = Nothing
        e.DataRow("型号") = Nothing
        e.DataRow("规格") = Nothing
        e.DataRow("单价") = Nothing
    Else
        Dim dr As DataRow
        dr = e.DataRow.GetParentRow("产品")
        If dr IsNot Nothing
            e.DataRow("品名") = dr("品名")
            e.DataRow("型号") = dr("型号")
            e.DataRow("规格") = dr("规格")
            e.DataRow("单价") = dr("单价")
        End If
    End If
End If
```

现在希望在产品表修改上述四列数据后，订单表能够自动更新这些列的数据，为此可以将产品表的 DataColChanged 事件设置为：

```
Select Case e.DataCol.Name
    Case "品名","型号","规格","单价"
        Dim drs As List(Of DataRow) = e.DataRow.GetChildRows("订单")
        For Each dr As DataRow In drs
            dr(e.DataCol.Name) = e.NewValue
        Next
End Select
```

上面的代码只是作为多列继承的一个例子而已，因为既然已经建立了关联，那么对于订单表中那些始终和产品表对应列内容保持一直的列，如品名、型号、规格等列，显然用表达式列更为合适；通常只有单价这种可能和产品表对应列内容有差异的列，才会考虑使用数据列，通过代码来计算的。

4.4.2.4.3.12 跨表统计

接下来的几节将讲述如何利用 DataColChanged 事件来完成数据统计的任务。
实际上关于数据统计，会有专门的一章讲述，而且会更加容易， 现在接触统计的目的，是为了让大家了解 DataColChanged 事件的更多用法。
假定有一个如下图所示的订单表：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03
4	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
5	PD03	CS03	EP04	10	0	490	4900	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07
7	PD05	CS03	EP04	17	0	160	2720	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08
10	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
11	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10

我们希望新增这么一个表，在该表中输入一个产品名称，能够自动统计出该产品总的订购数量。
首先增加一个表，表的名称为“统计”，该表包括两列，分别是字符型的产品，和整数型的数量：

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10351

然后在该表的 DataColChanged 事件 中设置如下代码：

```
If e.DataCol.Name = "产品" Then '发生变化的是产品列。
    '则累计订单表中同名产品总的销售量，然后赋值给变动行的数量列
    e.DataRow("数量") = DataTables("订单").Compute("Sum(数量)","[产品] = '" & e.NewValue & "'")
End If
```

上面的代码能够自动统计所输入产品的销售数量。
任何时候，如果希望得到最新的统计结果，只需选择产品列(注意不是数量列)，然后在“日常工作”功能区“数据”功能组，单击下面的“重置列”按钮即可：



如果希望一旦选择该表，就能自动刷新统计结果，而不需要人为选择产品列，然后单击“重置”按钮，可以在项目事件 MainTableChanged 中加入下面的代码：

```
If MainTable.Name = "统计" Then
    DataTables("统计").DataCols("产品").RaiseDataColChanged()
End If
```

能不能自动列出所有的产品，无需用户一个一个去输入呢？当然能够， 可是这不是我们现在要学习的内容，后面会有专门讲述数据统计的章节，到时我们再讲。

4.4.2.4.3.13 跨表统计的自动更新

首先我们重复一下上一节的内容。

假定有一个如下图所示的订单表：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03
4	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
5	PD03	CS03	EP04	10	0	490	4900	1999-01-04
6	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07
7	PD05	CS03	EP04	17	0	160	2720	1999-01-07
8	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
9	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08
10	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
11	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10

我们希望新增这么一个表，在该表中输入一个产品名称，能够自动统计出该产品总的销售数量。
首先增加一个表，表的名称为“统计”，该表包括两列，分别是字符型的产品，和整数型的数量：

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10351

然后在该表的 DataColChanged 事件 中设置如下代码：

```
If e.DataCol.Name = "产品" Then
    e.DataRow("数量") = DataTables("订单").Compute("Sum(数量)","[产品] = '" & e.NewValue & "'")
End If
```

请在统计表设置好上述代码后，重置一次该表的产品列。

现在要求在订单表输入或修改某订单的数量后，能够即时更新统计表对应产品的数量统计值。

为此可将订单表的 DataColChanged 事件代码设置为：

```
If e.DataCol.Name = "数量" Then
    Dim pr As DataRow
    pr = DataTables("统计").Find("产品  = '" & e.DataRow("产品") & "'")
    If pr IsNot Nothing Then
        pr("数量")= pr("数量") + e.NewValue - e.OldValue
```

```
    End If
End If
```

代码的原理很简单，修改某订单的数量后，上面的代码会在统计表找出对应的产品，然后将该产品的数量累计值加上此订单的新数量值，并减去此订单的原数量值，得到该产品最新的销售数量。

另一种自动更新方式

假定统计表不单单要产品总销售数量，还要统计总的销售金额：

	产品	数量	金额
1	PD01	11290	171754.92
2	PD02	18200	296522.5
3	PD03	7000	61490
4	PD04	5480	102861
5	PD05	10351	192190.15

首先将统计表的 DataColChanged 事件代码设置为：

```
If e.DataCol.Name = "产品" Then
    Dim Filter As String = "[产品] = '" & e.NewValue & "'"
    e.DataRow("数量") = DataTables("订单").Compute("Sum(数量)", Filter)
    e.DataRow("金额") = DataTables("订单").Compute("Sum(金额)", Filter)
End If
```

同样在统计表设置好上述代码后，请重置一次该表的产品列。

订单表的金额列是一个表达式列，其表达式为：[数量] * [单价] * (1 - [折扣])

我们知道表达式列有一个特性，就是其值发生变化后，并不会触发 DataColChanged 事件，此特性导致我们无法简单地用前面介绍的方法来自动更新统计表中的销售金额。

所以有必要考虑用另一重算方法，将订单表的 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.Name
    Case "数量","单价","折扣"
        Dim pr As DataRow
        pr = DataTables("统计").Find("产品  = '" & e.DataRow("产品") & "'")
        If pr IsNot Nothing Then
            DataTables("统计").DataCols("产品").RaiseDataColChanged(pr)
        End If
End Select
```

代码的原理是，当订单表中数量、单价、折扣三列中任何一列的内容发生变化，就从统计表中找出对应产品所在的行，然后“欺骗性”地通知系统该行的产品列内容发生了变化，于是触发统计表的 DataColChanged 事件，重算 此行的数量和金额。

这种更新方式是很常用的，即使处理的不是表达式列，我们多数时候也会采用这种更新方式。

因为使用 RaiseDataColChanged 更新，代码不会涉及具体的计算，所以代码是通用的，基本上不会因为计算要求的变动，而去修改更新代码。

提醒：

一般来说，统计表是没有必要即时更新的，完全可以像上一节那样简单地在项目事件 MainTableChanged 中加入下面的代码：

```
If MainTable.Name = "统计" Then
    DataTables("统计表").DataCols("产品").RaiseDataColChanged()
End If
```

这样切换到统计表之后，会自动执行上述代码更新统计结果，虽然效率也不如前面的高，但这种差别用户是很难感受得到的，所以这种简单的处理方式足以满足绝大多数场合的需要。

4.4.2.4.4 必不可少的判断

```
1  If e.DataCol.Name = "产品名称" Then '如果内容发生变动的是产品列
2      If e.NewValue Is Nothing Then '如果新值是空白,也就是产品列的内容为空
3          e.DataRow("单价") = Nothing '那么清空此行单价列的内容
4      Else
5          '否则在产品表查找同名的产品行,将找到的行赋值给变量 dr
6          Dim dr As DataRow
7          dr = DataTables("产品").Find("[产品名称] = '" & e.NewValue & "'")
8          e.DataRow("单价") = dr("单价")
9      End If
10 End If
```

为便于讲述，我们给代码加上了编号。

上面的代码看上去是没有问题的，但是疏忽了一个问题，就是订单表所输入的产品，可能在产品表中并不存在对应的行，这样第 7 行代码运行后，变量 dr 等于空值 Nothing，从而导致第 8 行代码出错，出现下图所示的错误提示：



为了避免这个问题，我们必须在代码中判断产品表中是否存在对应的行，也就是变量 dr 是否为 Nothing：

```
1  If e.DataCol.Name = "产品名称" Then '如果内容发生变动的是产品列
2      If e.NewValue Is Nothing Then '如果新值是空白,也就是产品列的内容为空
3          e.DataRow("单价") = Nothing '那么清空此行单价列的内容
4      Else
5          Dim dr As DataRow
6          '否则在产品表查找同名的产品行,将找到的行赋值给变量 dr
7          dr = DataTables("产品").Find("[产品名称] = '" & e.NewValue & "'")
8          If dr IsNot Nothing Then '如果找到了同名的产品行,也就是 dr 不是 Nothing
9              e.DataRow("单价") = dr("单价")
10         End If
11     End If
12 End If
```

上面的第 8 行代码判断 dr 是否为 Nothing，也就是判断是否在产品表中找到了对应的行，如果找到则执行第 9 行代码，将产品表的单价值填入到订单表中。

这里只是以 DataColChanged 事件和 Find 方法作为例子，实际上，在任何进行查找的场合，都要考

虑到找不到的情况。
我们在技术支持过程中，感觉这个 System.NullReferenceException 错误，大概是初学编程的用户提的最多的一个问题，大家一定要注意。

4.4.2.4.5 关于初始化设置

关于初始化设置的代码，我们有如下的建议：
1、设置列标题的代码，必须在项目事件 Initialize 编写。
2、开启全局表事件的代码，最好在项目事件 Initialize 编写。
3、设置 DataTable 的代码，在项目事件 Initialize 或者 AfterOpenProject 编写都可以，不过 AfterOpenProject 更合适一点，因为此时 DataTable 和 Table 都已经生成，可以统一设置。
4、设置用户权限的代码，在项目事件 AfterOpenProject 编写较为合适，如果运行过程中允许切换用户，那么最好在 LoadUserSetting 事件中编写。

示例一

例如下面设置列标题的代码，必须在项目事件 Initialize 事件中编写：

```
With DataTables("表 B")
    .DataCols("Name").Caption = "姓名"
    .DataCols("ID").Caption = "编号"
    .DataCols("Date").Caption = "日期"
End With
```

这是因为真正使用列标题是在生成 Table 的过程中，如果在 AfterOpenProject 设置列标题，此时 Table 已经生成，对于列标题的设置不会再有作用。
但是这并不意味着在运行过程中不能修改列标题，只不过修改后，要执行一次 BuildHeader 方法，而且也不如 Initialize 高效。
请参考：BuildHeader

示例二

例如你在全局表事件 AfterSelChange 中设置了代码，且希望这个事件针对所有表触发，那么应该在项目事件 Initialize 加入代码：

```
For Each dt As DataTable In DataTables
    dt.GlobalHandler.AfterSelChange = True
Next
```

为什么要在 Initialize 事件设置，而不是在 AfterOpenProject 事件设置呢？这是因为后者在项目完全打开后才执行，而在此之前，已经有 Table 会触发 AfterSelChange 事件。

示例三

例如希望产品表内容只能在 Tables("产品")修改，也就是只能在主表修改，而不能在关联表修改，可以在 AfterOpenProject 事件中加入如下代码：

```
DataTables("产品").AllowEdit = False
Tables("产品").AllowEdit = True
```

这样的代码在 AfterOpenProject 设置较为恰当，因为此时 DataTable 和 Table 都已经生成，可以统一对二者进行设置。

示例四

和用户权限设置有关的代码，那么最好 LoadUserSetting 事件中编写，例如：

```
If User.Type = UserTypeEnum.Administrator Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

这是因为 LoadUserSetting 不仅在打开项目后执行，而且每次切换用户也会执行一次，确保权限设置始终有效。

4.4.2.4.6 用代码辅助输入

4.4.2.4.6.1 动态列表项目之一

如果某列的列表项目需要根据另一列输入的内容而发生变化，那么就需要使用动态列表项目。

例如在 A 列内容"值 1"的时候，B 列的列表项目为"项目 1|项目 2"，如果 A 列内容等于"值 2"时，B 列的列表项目等于“项目 3|项目 4”，为此我们可以在 PrepareEdit 事件中加入如下代码：

```
If e.IsFocusCell Then
    If e.Col.Name = "第二列" Then
        Select Case e.Row("第一列")
            Case "值 1"
                e.Col.ComboList = "项目 1|项目 2"
            Case "值 2"
                e.Col.ComboList = "项目 3|项目 4"
        End Select
    End If
End If
```

代码的原理很简单，首先判断编辑的是否是第二列，如果是的，则根据第一列的值来设置不同的列表项目。

注意要用 IsFocusCell 判断一下是否是焦点单元格，否则会影响大面积粘贴数据以及查找替换的效率。

代码虽然简单，但是维护起来却不太方便了，因为如果要增加新的列表项目或者有新的要求，我们就必须重新修改代码，对于已经交付使用的系统，这可能是一个可怕的工作，可怕之处不在于修改代码，而在于逐个终端去更新程序。

所以最好的方式给最终的用户提供自行修改列表项目的功能，实现的步骤是:

1、新建一个表(假定表名为项目)，包括两列，分别是“值”和“列表项目”，用于输入每个值及其对应的列表项目：

	值	列表项目
1	值1	项目1|项目2
2	值2	项目3|项目4
3	值3	项目5|项目6|项目7

2、然后在数据录入表的 PreparEdit 事件中设置如下代码：

```
If e.IsFocusCell Then
```

```
    If e.Col.Name = "第二列" Then
        Dim dr As Datarow
        dr = DataTables("项目").Find("值  = '" & e.Row("第一列") & "'")
        If dr IsNot Nothing '一定要判断是否存在对应的行
            e.Col.ComboList = dr("列表项目")
        End If
    End If
End If
```

代码的原理同样很简单，如果正在输入的是第二列，那么就在项目表查找值等于输入行第一列内容的行，如果找到的话，将列表项目设为该行列表项目列的内容。

4.4.2.4.6.2 动态列表项目之二

我们可以在上一节的基础上扩展一下，假定某些列的列表项目不是动态的，但是随着应用的深入，这些列的列表项目可能会需要调整，虽然通过列属性设置窗口，可以方便地重新设置列表项目。但是对于一个已经交付使用的系统来说，让最终用户可以随意设置列属性是有危险的，有没有安全一些的做法呢？有，通用的做法是：

1、新建一个表(假定表名为项目)，包括三列，分别是表、列、列表项目，用于输入某表某列的列表项目:

	表	列	列表项目
1	表A	第一列	项目1|项目2
2	表A	第二列	项目3|项目4
3	表B	第一列	项目5|项目6|项目7

2、然后在全局表事件 PrepareEdit 中加入如下代码：

```
If e.IsFocusCell Then
    Dim dr As Datarow
    dr = DataTables("项目").Find("[表] = '" & e.Table.Name & "' And [列] = '" & e.Col.Name & "'")
    If dr IsNot Nothing '一定要判断是否存在对应的行
        e.Col.ComboList = dr("列表项目")
    End If
End If
```

在项目事件 Initialize 加入代码，为所有的表开启全局表事件 PrepareEdit：

```
For Each dt As DataTable In DataTables
    dt.GlobalHandler.PrepareEdit = True
Next
```

这样在编辑任何一列之前，都会根据表名和列名，在项目表检查是否存在对应的行，如果找到，将列表项目设为该行列表项目列的内容。

上面的代码简单实用，但是每次编辑单元格之前都去项目表检索一次，运行效率是比较低的。

我们可以删除全局表事件 PrepareEdit 和项目事件 Initialize 中的代码，改为在项目事件 AfterOpenProject 中加入下面的代码：

```
Dim dt as DataTable = DataTables("项目")
Dim dr As DataRow
For Each t As Table In Tables
    For Each c As Col In t.Cols
```

```
        dr = dt.Find("[表] = '" & t.DataTable.Name & "' And [列] = '" & c.Name & "'")
        If dr IsNot Nothing Then
            c.ComboList = dr("列表项目")
        End If
    Next
Next
```

这样在打开项目的时候，就会自动设置列表项目，这个设置是一次性的，所以运行效率要高很多；但是也有一个副作用，就是当你修改项目表的列表项目内容时，必须重新打开项目才会生效。

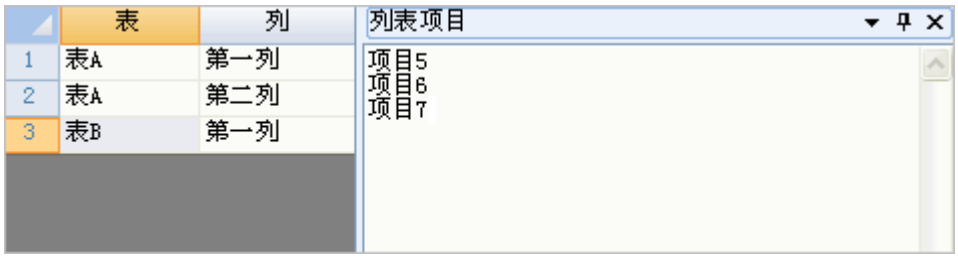
不过列表项目是不需要经常修改的，所以这不是一个大问题，如果你很在意的话，可以删除 AfterOpenProject 事件中的代码，改为在项目事件 CurrentTableChanged 中设置代码：

```
Dim dt as DataTable = DataTables("项目")
Dim dr As DataRow
For Each c As Col In CurrentTable.Cols
    dr = dt.Find("[表] = '" & CurrentTable.DataTable.Name & "' And [列] = '" & c.Name & "'")
    If dr IsNot Nothing Then
        c.ComboList = dr("列表项目")
    End If
Next
```

这样修改项目表中的列表项目后，无需重新打开项目即可生效。

一个小技巧：

我们一直用符号“|”来分隔列表项目，其实是可以用换行符分开的，这样我们可以更方便地录入列表项目，例如下图将列表项目列改为备注型，每行录入一个。



4.4.2.4.6.3 动态列表项目之三

更多的时候，列表项目来自于某个表中已经输入的记录，例如客户表已经输入所有的客户名称及资料，那么订单表的客户名称列的列表项目，应该自动从客户表的客户名称列提取。

如果列表项目来自于某个表的某一列，而这个表会经常增加删除行；那么可以在项目事件 MainTableChanged 中 ，设置代码提取列表项目。

例如：

```
If MainTable.Name = "订单" Then '如果选择的主表是订单表
    Tables("订单").Cols("客户").Combolist = DataTables("客户").GetComboListString("公司名称")
End If
```

上面的代码从客户表提取公司名称，作为订单表客户列的列表项目，由于经常会增加删除客户，所以我们将上面的代码设置在 MainTableChanged 事件中，这样每次切换到订单表，就会重新设置订单表客户列的列表项目，保证其内容是最新的。

如果某列的列表项目不仅来自于另一个表，而且要根据其它列已经输入的内容来决定显示哪些条目，那么这样的代码就只能设置在表事件 PrepareEdit 中。

例如：

```
If e.IsFocusCell Then '如果是焦点所在单元格
    If e.Col.Name = "县市" Then '如果正在编辑的是县市列
        '从行政区域表提取该省市的县市作为列表项目
        e.Col.Combolist = DataTables("行政区域").GetComboListString("县市", "[省市] = '" & e.Row("省市") & "'")
    End If
End If
```

上面的代码，根据省市列输入的内容，从行政区域表提取该省市的所有县市，作为县市列的列表项目，这种根据另一列内容来刷新列表项目的代码，必须设置在 PrepareEdit 事件中。

还有一个最简单的情况，即列表项目来自于某个表的某一列，而这个来源表的内容基本上是固定的，很少机会增加或删除行，那么可以将代码设置在项目事件 AfterOpenProject 中。

例如：

```
Tables("客户").Cols("省市").Combolist = DataTables("行政区域").GetComboListString("省市")
```

上面的代码从行政区域表提取省市列的内容，作为客户表省市列的列表项目，由于行政区域是固定的，基本上不需要增加删除行，所以从这个表中提取数据作为列表项目，代码可设置在 AfterOpenProject 事件中。

4.4.2.4.6.4 自动输入的实现

实现自动输入的基本原理是：

1、有一个基础数据表，输入所有的基础信息，例如订单管理系统中，产品表会输入所有的产品信息。

2、在另一个表输入某关键列的内容，然后自动从基础数据表提取其它列的内容输入到相关列中，例如在订单表输入产品编号，可以从产品表提取该产品的名称、型号、规格，自动输入到订单表的相关列中。

示例一

例如希望在订单表输入产品编号，能够从产品表提取该产品的名称、型号、规格，自动输入到订单表的相关列中。

为实现此目的,可在订单表的 DataColChanged 事件中输入如下代码：

```
If e.DataCol.Name = "产品编号" Then '发生变化的是产品编号吗?
    '在产品表找出该产品
    Dim dr As DataRow
    dr = DataTables("产品").Find("编号 = '" & e.DataRow("产品编号") & "' ")
    If dr IsNot Nothing '如果找到, 则设置各列内容
        e.DataRow("品名")= dr("品名")
        e.DataRow("型号")= dr("型号")
        e.DataRow("规格")= dr("规格")
    End If
End If
```

如果希望在产品表修改了某产品的品名、型号或规格后，能自动更新订单表中的品名、型号、规格，可以在产品表的 DataColChanged 事件中加入代码：

```
Select Case e.DataCol.Name
```

```
    Case "品名","型号","规格"
        Dim Filter As String = "[产品编号] = '" & e.DataRow("编号") & "'"
        Dim drs As List(Of DataRow) = DataTables("订单").Select(Filter)
        For Each dr As DataRow In drs
            dr(e.DataCol.Name) = e.DataRow(e.DataCol.Name)
        Next
End Select
```

或者：

```
Select Case e.DataCol.Name
    Case "品名","型号","规格"
        Dim Filter As String = "[产品编号] = '" & e.DataRow("编号") & "'"
        Dim nm As String = e.DataCol.Name
        DataTables("订单").ReplaceFor(nm, e.DataRow(nm), Filter)
End Select
```

示例二

假定项目中有一个名为“行政区域”的基础数据表，分别是省市、市县、区号、邮编，这个表已经输入了全国所有县级行政区域的资料；现在我们在客户表中输入数据，假定客户表也有这么四列，希望输入省市、县市两列的内容后，区号、邮编能够自动输入。

为实现此目的,可在客户表的 DataColChanged 事件中输入如下代码：

```
'如果刚刚输入的是省市或县市列
If e.DataCol.Name = "省市" Orelse e.DataCol.Name = "县市" Then
    Dim dr As DataRow
    Dim Filter As String
    Filter = "[省市] = '" & e.DataRow("省市") & "' And [县市] = '" & e.DataRow("县市") & "'"
    dr = DataTables("行政区域").Find(Filter ) '在行政区域表查找所输入省市和县市的行
    If dr IsNot Nothing Then '如果找到
        '将找到行的区号、邮编内容填入到正在输入的行中.
        e.DataRow("区号") = dr("区号")
        e.DataRow("邮编") = dr("邮编")
    Else
        '否则清除区号和邮编两列的内容
        e.DataRow("区号") = Nothing
        e.DataRow("邮编") = Nothing
    End If
End If
```

4.4.2.4.6.5 综合示例

本示例请参考 CaseStudy 目录下的示例文件“自动输入.Table”。

假定项目中有一个名为“行政区域”的基础数据表，这个表已经输入了全国所有县级行政区域的资料，包括省市、市县、区号、邮编四列；现在我们在客户表中输入数据，假定客户表也有这么四列，显然最好的输入方式是，客户表省市列能够根据行政区域表的设置，列出所有的省市供选择，选择省市之后，市县列能列出该省市所有的市县供选择，而选择市县之后，区号和邮编能够自动输入。

一、先在项目事件 AfterOpenProject 中设置如下代码：

```
'从行政区域表提取省市列的内容，作为客户表省市列的列表项目
Tables("客户").Cols("省市").Combolist = DataTables("行政区域").GetComboListString("省市")
```

我们不在表事件 PrepareEdit 中设置客户表“省市”列的列表项目，而是在项目事件 AfterOpenProject 中设置，是因为省市的列表项目是固定的，如果在 PrepareEdit 事件中设置，每选择一个单元格都会重新设置一次列表项目，显然是没有必要的。

二、在客户表的 PrepareEdit 事件中输入如下代码：

```
If e.IsFocusCell Then '如果是焦点所在单元格
    If e.Col.Name = "县市" Then '如果正在编辑的是县市列
      '从行政区域表提取该省  市的县市作为列表项目
       e.Col.Combolist = DataTables("行政区域").GetComboListString("县市", "[省市] = '" & e.Row("省市") & "'")
    End If
End If
```

再次强调，动态列表项目是针对正在输入的焦点单元格，所以一定要用 IsFocusCell 判断触发事件的是否是焦点单元格，因为在表中进行剪切、粘贴的时候，会针对每一个可能会受影响的单元格触发 PrepareEdit 事件，如果你在县市列复制粘贴 1000 行数据，那么就会触发一千次 PrepareEdit 事件， 如果不加判断，就等于要执行一千次 GetComboListString，这可能需要一个漫长的执行过程。

三、客户表的 DataColChanged 事件中输入如下代码：

```
'如果刚刚输入的是省市或县市列
If e.DataCol.Name = "省市" Orelse e.DataCol.Name = "县市" Then
    Dim dr As DataRow
    Dim Filter As String
    Filter = "[省市] = '" & e.DataRow("省市") & "' And [县市] = '" & e.DataRow("县市") & "'"
    dr = DataTables("行政区域").Find(Filter) '在行政区域表查找所输入省市和县市的行
    If dr IsNot Nothing Then '如果找到
      '将找到行的区号和邮编内容填入到正在输入的行中
       e.DataRow("区号") = dr("区号")
       e.DataRow("邮编") = dr("邮编")
    Else
      '否则清除区号和邮编两列的内容
       e.DataRow("区号") = Nothing
       e.DataRow("邮编") = Nothing
    End If
End If
```

4.4.2.4.6.6 动态目录树列表

利用 DropTreeBuilder，同样可以动态生成目录树列表。

本节的内容可以参考 CaseStudy 目录下的文件：动态目录树列表.Table

假定有一个“行政区域“表，结构如下：

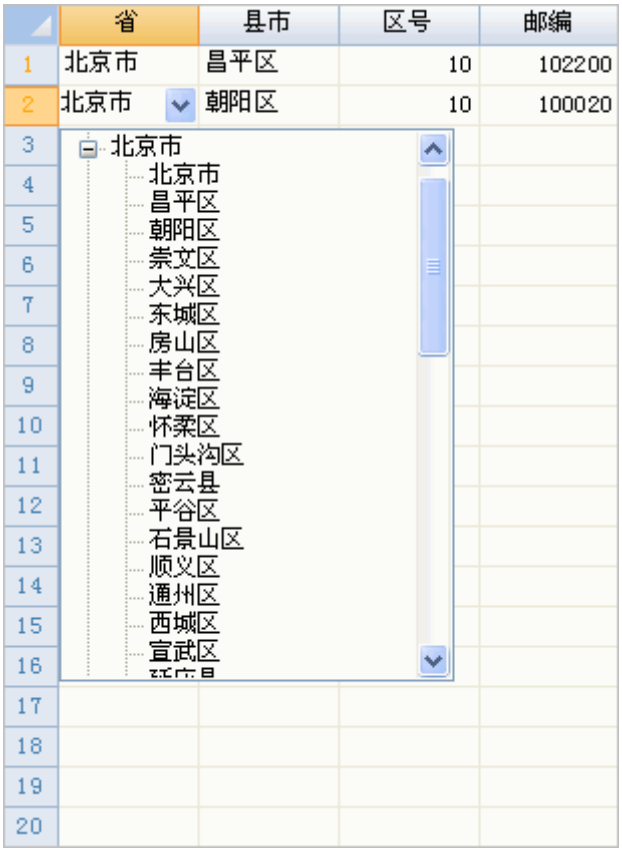
	省	县市	区号	邮编
1	北京市	北京市	10	100000
2	北京市	东城区	10	100010
3	北京市	西城区	10	100030
4	北京市	崇文区	10	100060
5	北京市	宣武区	10	100050
6	北京市	朝阳区	10	100020

我们希望在客户表输入数据的时候，能够根据该”行政区域“表自动生成目录树，用于输入省、县市、区号、邮编等列的内容 。

为此可以在项目的 AfterOpenProject 事件中加入代码：

```
Dim tb As New DropTreeBuilder
tb.SourceTable = DataTables("行政区域") '指定目录树表
tb.TreeCols = "省|县市" '指定用于生成目录树的列
tb.SourceCols = "省|县市|区号|邮编" '指定数据来源列
tb.ReceiveCols = "省|县市|区号|邮编" '指定数据接收列
Tables("客户").Cols("省").DropTree = tb.Build()
```

这样打开项目后，选择客户表的省列，单击下拉按钮，即可出现下图所示的目录树，选择某个县市，可以一次输入省、县市、区号和邮编四列的内容：



上面的代码设置在 AfterOpenProject 事件中，这有一个问题，就是在“行政区域”表中修改数据后，上述代码生成的目录树不会自动更新，除非你重新打开项目文件。

为此可以将代码移至 MainTableChanged 事件中：

```
If MainTable.Name = "客户" Then
    Dim tb As New DropTreeBuilder
```

```
    tb.SourceTable = DataTables("行政区域") '指定目录树表
    tb.TreeCols = "省|县市" '指定用于生成目录树的列
    tb.SourceCols = "省|县市|区号|邮编" '指定数据来源列
    tb.ReceiveCols = "省|县市|区号|邮编" '指定数据接收列
    Tables("客户").Cols("省").DropTree = tb.Build()
End If
```

这样每次选择客户表，都会根据“行政区域”表的最新数据，重新生成目录树列表。

不管是动态列表项目还是动态目录树列表，合理选择代码位置是很重要的，如果项目来源表很少需要更新，例如上面的行政区域表，那么将代码放在项目事件 AfterOpenProject 最为合理，偶尔更新的时候，也只需重新打开项目文件即可；如果项目来源表经常需要在运行过程中更新，那么应该将代码放在项目事件 MainTableChanged 中；如果需要根据其它列已经输入的内容来生成列表项目或目录树，那么就只能将代码放在表事件 PrepareEdit 中了。

4.4.2.4.6.7 动态数据字典

利用类型 TableDataMap 来根据数据表创建多列数据字典，该类型包括以下属性：

- **DataTable**
  多列数据字典来源表的名称。

- **ValueCol**
  取值列的名称。

- **DisplayCol**
  显示列的名称。

- **ListCols**
  指定下拉的表格中，要显示哪些列的内容，不同的列之间用逗号隔开。

- **Sort**
  排序方式。

- **Filter**
  过滤条件表达式，输入的时候，下拉列表只显示符合此条件的行。

我们知道，通过菜单中的列属性设置的数据字典，在原数据表更新后，数据字典并不会自动更新。通过 TableDataMap， 可以很方便地动态更新数据字段。

假定我们希望在显示的时候，订单表的"客户 ID"列用对应的客户名称代替，同时在输入的时候，能够自动列出客户表中的所有客户及相关信息，供使用者选择客户，如下图所示：



订单 ID	客户 ID	员工 ID	订单日期	发货日期	运货商 ID	发货名称	发货地址	发货城市
30	文成	9	2006-01-15	2006-01-22	2	方建文	南开北路 3 号	南京
31	国顶有限公司	3	2006-01-20	2006-01-22	1	李先生	广汇东区甲 2 号	南京
32	4 国顶有限公司 方 先生 销售代表 (0571) 45557788 (0571) 45556750 天府东街 30 号 深圳 890879							
33	11 光明杂志 谢 丽秋 销售代表 (0571) 45551212 (0571) 45552312 黄石路 50 号 深圳 760908							
34	20 正人资源 谢 小姐 销售代理 (0571) 76753425 (0571) 76753426 临江东街 62 号 深圳 906853							
35	21 红阳事业 王 先生 市场助理 (0571) 75559857 (0571) 75559658 外滩西路 238 号 深圳 687759							
36	29 东旗 王 先生 市场经理 (0571) 20334560 (0571) 20334561 尊石路 238 号 深圳 411012							
37	3 坦森行贸易 王 炫皓 物主 (0321) 5553932 (0321) 55365425 黄台北路 780 号 石家庄 985060							
38	康浦	9	2006-03-10	2006-03-11	3	刘先生	新技术开发工业区 32 号	烟台
39	迈多贸易	3	2006-03-22	2006-03-24	3	刘先生	新技术开发工业区 66 号	青岛

例如我们在项目事件 MainTableChanged 事件中设置代码：

```
If MainTable.Name = "订单" Then
    Dim dmp As New TableDataMap
    dmp.DataTable = "客户" '指定数据来源表
    dmp.ValueCol = "ID" '指定取值列
    dmp.DisplayCol = "公司" '指定显示列
    '指定下拉列表时显示哪些列的数据
    dmp.ListCols = "ID,公司,姓氏,名字,职务,业务电话,传真号,地址,城市,邮政编码"
    dmp.Sort = "城市" '指定排序方式
    Tables("订单").Cols("客户  ID").DataMap = dmp.CreateDataMap() '生成并设置DataMap
End If
```

这样在客户表修改数据后，重新切换到订单表，就会自动重新生成数据字典，保证数据字典的内容是最新的。

重要提示

如果要动态设置条件，那么最好不要用数据字典，建议编号和名称分开两列，用帮助介绍的自定义录入界面选择数据录入，我们在学习窗口设计的时候，会接触到这部分内容。
假定你的编号列有1、2、3、4、5等值，如果设置数据字典的条件后，字典只包括1、2、3三个编号，那么编号列的4、5会显示空白，如果数据字典没有符合条件的数据，那么编号列全部都会显示空白。

4.4.2.4.7 流水账的设计

4.4.2.4.7.1 流水账的设计

本示例可以参考 CaseStudy 目录下的文件：流水账.Table
流水账对于行的顺序有严格的要求，而且有时免不了要插入行或者调整行位置。
所以对于要设计流水账的表，应该启用插入行功能。
提示：新建的内部表默认已经有插入行的功能
启用插入行功能后，表会自动增加一名为“_SortKey”的高精度小数列，该列的值代表着行在表中的顺序，这是设计流水账的基础。
假定有个简单的流水帐式的表格，输入收入和支出，能够自动计算出余额:

	日期	收入	支出	余额
1	1999-02-01	1		1
2	1999-02-03	7	5	3
3	1999-02-05	2		5
4	1999-02-07	4	6	3
5	1999-02-09	8		11
6	1999-02-11	4		15
7	1999-02-13	6	4	17
8	1999-02-15	3		20
9	1999-02-17	9	6	23
10	1999-02-19	4		27

实现的代码不复杂，只需将 DataColChanged 事件设为:

```
Select Case e.DataCol.Name
    Case "收入","支出"
        For Each dr As DataRow In e.DataTable.Select("[_SortKey] >= " & e.DataRow("_SortKey"))
            Dim Val1 As Double = e.DataTable.Compute("Sum(收入)","[_SortKey] <= " & dr("_SortKey"))
            Dim Val2 As Double = e.DataTable.Compute("Sum(支出)","[_SortKey] <= " & dr("_SortKey"))
            dr("余额") = Val1 - Val2
        Next
End Select
```

该段代码的原理很简单：如果修改了某行的收入或支出，那么针对顺序在该行之后的每一行，计算出截止到此行的收入和支出，二者相减即可得到此行的余额。

现在的问题是，如果上移行或下移行，余额值肯定要自动更新，表事件 AfterMoveRow 在移动行后执行，为此我们可以在该事件中加入代码：

```
Dim Key As Decimal
Dim Index As Integer
Dim dc As DataCol
Index = Math.Min(e.OldIndex, e.NewIndex)
Key = e.Table.Rows(Index)("_SortKey")
dc = e.Table.DataTable.DataCols("收入")
dc.RaiseDataColChanged("[_SortKey] >= " & Key)
```

上述代码的原理是，取得行原位置和新位置的最小值，然后从此最小位置开始重算余额，例如将 8 行移到第 2 行，则从第 2 行开始重算，将第 3 行移到第 9 行，则从第 3 行开始重算。

此外，如果删除某行，此行之后的所有余额也应该更新，所以还得将 DataRowDeleting 事件代码设置为：

```
e.DataRow("收入") = 0
e.DataRow("支出") = 0
```

上面的代码在行即将被删除时，将该行的收入和支出设置为 0，从而触发 DataColChanged 事件，重算此行之后的所有其他行的余额 ，由于重算时，此行的收入和支出已经被设置为 0，等于其它行在计算余额的时候，不再考虑此行了。

更多的时候，流水账是区分产品的，例如下图的库存表：

	日期	产品	入库	出库	库存
1	1999-02-01	PD01	1		1
2	1999-02-03	PD02	7	5	2
3	1999-02-05	PD03	2	2	0
4	1999-02-07	PD01	4	3	2
5	1999-02-09	PD02	8		10
6	1999-02-11	PD03	4		4
7	1999-02-13	PD01	6	4	4
8	1999-02-15	PD02	3		13
9	1999-02-17	PD03	9	6	7
10	1999-02-19	PD01	4		8
11	1999-02-17	PD02	7	4	16

此类流水账的设计方法要复杂一些，DataColChanged 事件代码为：

```
Select Case e.DataCol.Name
    Case "产品","入库","出库"
        Dim drs As List(of DataRow)
        Dim Filter As String
        Filter = "[_SortKey] >= " & e.DataRow("_SortKey") & " And [产品] = '" & e.DataRow("产品") &
"'"
        drs = e.DataTable.Select(Filter)
        For Each dr As DataRow In drs
            Filter = "[_SortKey] <= " & dr("_SortKey") & " And [产品] = '" & dr("产品") & "'"
            Dim Val1 As Double = e.DataTable.Compute("Sum(入库)",Filter)
            Dim Val2 As Double = e.DataTable.Compute("Sum(出库)",Filter)
            dr("库存") = Val1 - Val2
        Next
        If e.DataCol.Name = "产品" AndAlso e.OldValue IsNot Nothing AndAlso e.OldValue <>
e.NewValue Then
            Filter = "[_SortKey] > " & e.DataRow("_SortKey") & " And [产品] = '" & e.OldValue & "'"
            drs = e.DataTable.Select(Filter)
            For Each dr As DataRow In drs
                Filter = "[_SortKey] <= " & dr("_SortKey") & " And [产品] = '" & dr("产品") & "'"
                Dim Val1 As Double = e.DataTable.Compute("Sum(入库)",Filter)
                Dim Val2 As Double = e.DataTable.Compute("Sum(出库)",Filter)
                dr("库存") = Val1 - Val2
            Next
        End If
End Select
```

```
1
2
2
2
3
```

上述代码的 3 到 12 行很好理解，当我们修改产品、入库、出库三列内容后，即从此行开始重算同产品所有行的余额。

假定我们修改的是产品，例如将产品由 PD01 改为 PD02，3 到 12 行会从此行开始重算所有产品为 PD02 的行的余额；显然此行之后的产品为 PD01 的行的余额，也是应该重算的，于是就有了 13 到 22 行的代码，注意第 14 行在合成条件的时候，用 e.OldValue 取得原产品名称。

AfterMoveRow 事件代码变化不大，条件表达式中加入产品比较即可：

```
Dim Key As Decimal
Dim Index As Integer
Dim Filter As String
Dim r As Row
Index = Math.Min(e.OldIndex, e.NewIndex)
Key = e.Table.Rows(Index)("_SortKey")
r = e.Table.Rows(e.NewIndex)
Filter = "[_SortKey] >= " & Key & " And [产品] = '" & r("产品") & "'"
e.Table.DataTable.DataCols("入库").RaiseDataColChanged(Filter)
```

而 DataRowDeleting 事件代码则完全相同：

```
e.DataRow("入库") = 0
e.DataRow("出库") = 0
```

4.4.2.4.7.2 高效率的流水账设计

本示例可以参考 CaseStudy 目录下的文件：高效率流水账.Table

上一节介绍的流水账设计方法是比较低效的，下面介绍一种更为高效的设计方法，希望大家仔细体会。

示例一

同样假定有个简单的流水帐式的表格，输入收入和支出，能够自动计算出余额:

	日期	收入	支出	余额
1	1999-02-01	1		1
2	1999-02-03	7	5	3
3	1999-02-05	2		5
4	1999-02-07	4	6	3
5	1999-02-09	8		11
6	1999-02-11	4		15
7	1999-02-13	6	4	17
8	1999-02-15	3		20
9	1999-02-17	9	6	23
10	1999-02-19	4		27

我们采用和上一节完全不同的方法，不再通过累计出之前的收入和支出，利用二者相减得出余额，而是通过上一行的余额和本行收入和支出，计算出本行的余额，显然这样的计算方式要高效很多。

为此将 DataColChanged 事件设为:

```
1  Select Case e.DataCol.Name
2     Case "收入","支出"
3        Dim dr As DataRow
4        Dim drs As List(of DataRow)
5        dr = e.DataTable.Find("[_SortKey] < " & e.DataRow("_SortKey"), "[_SortKey] Desc") '找出上
6  一行
7        If dr Is Nothing Then '如果没有找到上一行,说明本行就是第一行
8           e.DataRow("余额") = e.DataRow("收入") - e.DataRow("支出")
9           dr = e.DataRow
10       End If
11       drs = e.DataTable.Select("[_SortKey] >= " & dr("_SortKey"), "[_SortKey]")
12       For i As Integer = 1 To drs.Count - 1 '重算余下行的余额
13          drs(i)("余额") = drs(i-1)("余额") + drs(i)("收入") - drs(i)("支出")
14       Next
   End Select
```

上述代码的原理是：

- 当我们修改某行的收入和支出后，第 5 行代码会找出被修改行的上一行并保存在变量 dr 中，如果找到上一行的话(也就是 dr 不是 Nothing)，则直接执行第 10 行代码，筛选出从上一行开始的所有行，并保存在集合 drs 中；第 11 到 13 行代码用于重算这些行的余额，每一行的余额，都等于上一行的余额加上此行收入，再减去此行支出。注意第 11 行代码的循环变量不是通常那样从 0 开始，而是从 1 开始循环，也就说 11 到 13 行代码在重算余额的时候，并不包括被修改行的上一行，而是从被修改的行开始重算，道理是显然的，因为修改某行的收入和支出，并不会影响上一行的余额，我们之所以要找出被修改行的上一行，只是为了得到上一行的余额，用于计算被修改行的余额。

  注意第 5 行代码和第 10 行代码的排序参数设置，如果不能理解，请参考 Find 和 Select

- 如果第 5 行代码没有找到被修改行的上一行(也就是变量 dr 是 Nothing)，说明被修改的行就是第一行，那么该行的余额就直接等于此行的收入减去支出，于是第 7 行代码计算出被修改行的余额，第 8 行代码将被修改行保存在变量 dr 中；接下来执行第 10 行代码，筛选出从被修改行开始的所有行，并保存在集合 drs 中；第 11 到 13 行代码用于重算这些行的余额，每一行的余额，都等于此行上一行的余额加上此行收入，再减去此行支出。注意第 11 行代码的循环变量不是通常那样从 0 开始，而是从 1 开始循环，也就说 11 到 13 行代码在重算余额的时候，并不包括被修改行，因为被修改行的余额已经在第 7 行代码计算得出。

表事件 AfterMoveRow 的代码保持不变：

```
Dim Key As Decimal
Dim Index As Integer
Dim dc As DataCol
Index = Math.Min(e.OldIndex, e.NewIndex)
Key = e.Table.Rows(Index)("_SortKey")
```

dc = e.Table.DataTable.DataCols("收入")
dc.RaiseDataColChanged("[_SortKey] >= " & Key)
表事件 DataRowDeleting 的代码同样保持不变：
e.DataRow("收入") = 0
e.DataRow("支出") = 0
如果是多人同时编辑数据，还需要在项目事件 AfterOpenproject 中加入代码：

```
Dim dr As DataRow
dr = DataTables("例子一").Find("","[_SortKey]")
If dr IsNot Nothing Then
    '模拟第一行的支出发生变化，刷新已加载行的余额.
    DataTables("例子一").DataCols("支出").RaiseDataColChanged(dr)
End If
```

这样每次打开项目，都会自动重算余额列，确保数据准确，注意我们没有必要重置所有行，只需重置第一行即可

示例二
对于下图所示这种区分产品的流水账：

	日期	产品	入库	出库	库存
1	1999-02-01	PD01	1		1
2	1999-02-03	PD02	7	5	2
3	1999-02-05	PD03	2	2	0
4	1999-02-07	PD01	4	3	2
5	1999-02-09	PD02	8		10
6	1999-02-11	PD03	4		4
7	1999-02-13	PD01	6	4	4
8	1999-02-15	PD02	3		13
9	1999-02-17	PD03	9	6	7
10	1999-02-19	PD01	4		8
11	1999-02-17	PD02	7	4	16

需要将 DataColChanged 事件代码改为：

```
Select Case e.DataCol.Name
    Case "产品","入库","出库"
        Dim dr As DataRow
        Dim mr As DataRow = e.DataRow
        Dim drs As List(of DataRow)
        dr = e.DataTable.Find("[_SortKey] < " & mr("_SortKey") & " And [产品] = '" & mr("产品") & "'",
"[_SortKey] Desc")
        If dr Is Nothing Then
            mr("库存") = mr("入库") - mr("出库")
            dr = mr
        End If
        drs = e.DataTable.Select("[_SortKey] >= " & dr("_SortKey") & " And [产品] = '" & dr("产品") &
"'", "[_SortKey]")
```

```
1       For i As Integer = 1 To drs.Count - 1
2         drs(i)("库存") = drs(i-1)("库存") + drs(i)("入库") - drs(i)("出库")
1       Next
3       If e.DataCol.Name = "产品" AndAlso e.OldValue IsNot Nothing AndAlso e.OldValue <>
1 e.NewValue Then
4         dr = e.DataTable.Find("[_SortKey] < " & mr("_SortKey") & " And [产品] = '" & e.OldValue & "'",
1 "[_SortKey] Desc")
5         If dr Is Nothing Then
1           dr = e.DataTable.Find("[产品] = '" & e.OldValue & "'", "[_SortKey]")
6           If dr IsNot Nothing Then
1             dr("库存") = dr("入库") - dr("出库")
7           End If
1         End If
8         If dr IsNot Nothing Then
1           drs = e.DataTable.Select("[_SortKey] >= " & dr("_SortKey") & " And [产品] = '" & dr("产品")
9 & "'", "[_SortKey]")
2           For i As Integer = 1 To drs.Count - 1
0             drs(i)("库存") = drs(i-1)("库存") + drs(i)("入库") - drs(i)("出库")
2           Next
1         End If
2       End If
2 End Select
2
3
2
4
2
5
2
6
2
7
2
8
2
9
3
0
```

上述代码的第一部分(3 到 14 行)，用于在修改某行的产品、入库、出库后，重算从此行开始的所有同产品的行的余额，代码原理已经在上面不区分产品的流水账中介绍，不同的只是在条件表达式中加入了产品比较，请大家自己分析。

第二部分代码（15 到 29 行）的原理：

- 假定我们修改的是产品，例如将产品由 PD01 改为 PD02，3 到 14 行会从被修改行开始重算所有产品为 PD02 的行的余额；显然被修改行之后的所有产品为 PD01 的行的余额，也是应该重算的，于是就有了 15 到 29 行的代码， 首先第 16 行代码找出被修改行的上一个产品为 PD01 的行，保存在变量 dr 中，如果找到的话(变量 dr 不等于 Nothing)，则执行第 24 代码，找出从此行开始的所有产品为 PD01 的行，第 25 到 27 行重算此行之后的所有产品为 PD01 的行的余额。

- 假定第 16 行代码并未找到被修改行上一个产品为 PD01 的行，说明被修改的行就是第一个产品为 PD01 的行，现在该行的产品已经由 PD01 改为 PD02，原来第二个产品为 PD01 的行现在变成了第一个产品为 PD01 的行，于是第 18 行代码找出现在第一个产品为 PD01 的行，并保存在变量 dr 中，如果找到的话，由于这是第一个产品为 PD01 的行，其库存直接等于入库减去出库，于是第 20 行代码计算出此行的余额；然后执行第 24 行代码，找出从此行开始所有产品为 PD01 的行，保存在集合 drs 中，第 25 到 27 行重算此行之后的所有产品为 PD01 的行的余额。

表事件 AfterMoveRow 的代码保持不变：

```
Dim Key As Decimal
Dim Index As Integer
Dim Filter As String
Dim r As Row
Index = Math.Min(e.OldIndex, e.NewIndex)
Key = e.Table.Rows(Index)("_SortKey")
r = e.Table.Rows(e.NewIndex)
Filter = "[_SortKey] >= " & Key & " And [产品] = '" & r("产品") & "'"
e.Table.DataTable.DataCols("入库").RaiseDataColChanged(Filter)
```

表事件 DataRowDeleting 的代码同样保持不变：

```
e.DataRow("入库") = 0
e.DataRow("出库") = 0
```

如果多用户同时录入数据，可以在项目事件 AfterOpenproject 中加入代码：

```
Dim drs As New List(of DataRow)
With DataTables("例子二")
    For Each nm As String In .GetValues("产品") '找出每个产品的第一行数据，添加到集合drs中
        drs.Add(.Find("产品  = '" & nm & "'", "[_SortKey]"))
    Next
    For Each r As DataRow In drs
        .DataCols("入库").RaiseDataColChanged(r) '重置每个产品的第一行
    Next
End With
```

这样打开项目之后，会自动重置每个产品的第一行，刷新每个产品的库存。

4.4.2.4.7.3 流水账与重置列

假定下图的流水账有 10000 行数据：

	日期	收入	支出	余额
1	1999-02-01	1		1
2	1999-02-03	7	5	3
3	1999-02-05	2		5
4	1999-02-07	4	6	3
5	1999-02-09	8		11
6	1999-02-11	4		15
7	1999-02-13	6	4	17
8	1999-02-15	3		20
9	1999-02-17	9	6	23
10	1999-02-19	4		27

如果重置收入列，当我们重置第一行的时候，会重算所有 10000 行的余额，当重置第二行的时候，会重算剩下的 9999 行的余额，同样重置第三行的时候，回重算余下的 9998 行的余额，这样算下来，重置一次收入列，等于重算了 50005000 行的余额，也许一个昼夜也无法完成这样一次重置，而我们实际只有 10000 行数据，也只需重算 1 万行的余额。

显然我们只需重置第一行的收入列，即可重算所有 10000 行的余额，所以我们应该针对上面的流水账加上一个自定义的重置命令：

```
Dim dr As DataRow
dr = DataTables("表名").Find("","[_SortKey]") '找出第一行
If dr IsNot Nothing Then
    '模拟第一行的支出发生变化，刷新已加载行的余额.
    DataTables("表名").DataCols("支出").RaiseDataColChanged(dr)
End If
```

上述代码的效率比直接重置收入列快 5000 倍。

对于下图这种区分产品的流水账:

	日期	产品	入库	出库	库存
1	1999-02-01	PD01	1		1
2	1999-02-03	PD02	7	5	2
3	1999-02-05	PD03	2	2	0
4	1999-02-07	PD01	4	3	2
5	1999-02-09	PD02	8		10
6	1999-02-11	PD03	4		4
7	1999-02-13	PD01	6	4	4
8	1999-02-15	PD02	3		13
9	1999-02-17	PD03	9	6	7
10	1999-02-19	PD01	4		8
11	1999-02-17	PD02	7	4	16

显然也只需重置每个产品的第一行的入库列，即可重算所有行的库存，所以我们也应该加上一个自定义的重置按钮：

```
Dim nms As New List(Of String)
Dim drs As New List(of DataRow)
With DataTables("表名")
    nms = .GetValues("产品")
```

```
    For Each nm As String In nms '找出每个产品的第一行数据，添加到集合 drs 中
        drs.Add(.Find("产品 = '" & nm & "'", "[_SortKey]"))
    Next
    For Each dr As DataRow In drs
        .DataCols("入库").RaiseDataColChanged(dr)
    Next
End With
```

4.4.2.4.8 其他常见任务

4.4.2.4.8.1 关联与增加行

有的时候，希望父表增加一行后，能够在子表自动对应增加一行，以提高输入效率。
例如订单和订单明细表，通过订单编号建立关联，希望新增一订单，并输入订单编号后，能自动为该订单增加一个订单明细。
为此可以设置订单表的 DataColChanged 事件为：

```
Select Case e.DataCol.Name
    Case "订单编号"
        If e.DataRow.IsNull("订单编号") = False Then '如果已经输入订单编号
            If e.DataRow.GetChildRows("订单明细").Count = 0 Then '而且无订单明细
                Dim dr As DataRow = DataTables("订单明细").AddNew()
                dr("订单编号") = e.DataRow("订单编号")
            End If
        End If
End Select
```

4.4.2.4.8.2 关联与删除行

假定订单表和订单明细表通过订单编号建立了关联，通常在建立关联的时候，我们会选择选项“同步删除关联行”，这样在删除某个订单的时候，会同步删除此订单的所有订单明细，防止出现孤立的订单明细。
显然如果我们误删除了某订单，那么该订单的所有订单明细也会被误删除，增加了恢复此订单的难度。
所以最好的方式是在删除某个订单之前，先判断此订单是否存在订单明细，如果存在的话，就给用户以提示，如果不存在，就直接删除：
为此可以将订单表的 BeforeDeleteDataRow 事件代码设置为：

```
If e.DataRow.GetChildRows("订单 明细").Count > 0 Then
    Dim s As String = "此订单存在订单明细，是否确定要删除此订单及其订单明细?"
        If  MessageBox.Show(s," 提 示 ",MessageBoxButtons.YesNo,MessageBoxIcon.Question)  =
DialogResult.No
            e.Cancel= True
    End If
End If
```

4.4.2.4.8.3 到期提醒设计

假定有下图所示的一个表：

	货品名称	生产日期	保质天数	到期日期	数量	单位
1	5卡油漆	2012-02-01	60	2012-04-01	10	桶
2	玻璃胶	2012-01-23	120	2012-05-22	5	支
3	中性笔	2012-03-01	100	2012-06-09	12	支
4	5卡油漆	2012-02-03	90	2012-05-03	5	桶

现在希望：

1、对于 10 天内即将到期的产品，用红低白字显示到期日期。

2、每次打开项目，如果有 10 天内即将到期的产品，就自动提示用户，并筛选出这些即将到期的产品。

设计步骤

1、首先增加一个自定义样式，名称为“到期”，红底白字。

2、设置表的 DataColChanged 事件代码，以便根据生产日期和到期天数，计算出到期日期：

```
Select Case e.DataCol.Name
    Case "生产日期","保质天数"
        If e.DataRow.IsNull("生产日期") Then
            e.DataRow("到期日期") = Nothing
        Else
            e.DataRow("到期日期") = e.DataRow("生产日期").AddDays(e.DataRow("保质天数"))
        End If
End Select
```

3、设置表的 DrawCell 事件代码，标出 10 天内即将到期的产品：

```
If e.col.Name = "到期日期" Then
    If e.Row.IsNull("到期日期") = False
        If e.Row("到期日期") < Date.Today.AddDays(10) Then
            e.Style = "到期"
        End If
    End If
End If
```

4、最后设置项目事件 AfterOpenProject 的代码，以便打开项目的时候，能够自动提示并显示即将到期的产品：

```
Dim dt As Date = Date.Today.AddDays(10)
Dim dr As DataRow = DataTables("库存表").Find("到期日期 <= #" & dt & "#")
If dr IsNot Nothing Then
        MessageBox.Show("有产品即将到期!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Tables("库存表").Filter = "到期日期 <= #" & dt & "#"
    MainTable = Tables("库存表")
End If
```

4.4.2.4.8.4 显示动态提示信息

示例一

假定有个员工表，希望将鼠标移到备注列的时候，能够自动显示所有的备注内容，鼠标离开备注列的时候，能自动隐藏提示，如图所示：



要实现这个功能非常的简单:

1、在 MouseEnterCell 事件设置代码：

```
If e.Col.Name = "备注" AndAlso e.Row.IsNull("备注") = False Then
    e.Table.ShowToolTip(e.Row("备注"),e.Row,e.Col)
End If
```

2、在MouseLeaveCell事件设置代码：

```
If e.Col.Name = "备注" Then
    e.Table.HideToolTip()
End If
```

示例二

假定希望在输入编码列的内容的时候，能自动列出各编码代表的内容，如下图所示：



要实现这个功能同样很简单：

1、在StartEdit(注意不是PrePareEdit事件)事件中设置代码：

```
If e.Col.Name = "编码" Then
    Dim v As String = "1: 初中"
    v = v & vbcrlf & "2: 高中"
```

```
    v = v & vbcrlf & "3: 大专"
    v = v & vbcrlf & "4: 本科"
    v = v & vbcrlf & "5: 硕士"
    v = v & vbcrlf & "6: 博士"
    e.Table.ShowToolTip(v, e.Row, e.Col)
End If
```

2、在AfterEdit事件中设置代码：

```
If e.Col.Name = "备注" Then
    e.Table.HideToolTip()
End If
```

4.4.2.4.8.5 显示动态错误信息

如果发现用户输入的重要数据不符合规范，通常我们会显示一个对话框，给用户以提示。
这种纠正输入错误的方法较为“粗暴”，在用户输入的过程中，应该尽量不要打断用户操作的连贯性，我们可以用较为“平和”的方式对用户做出提示。

示例一

假定订单表中的折扣列不能超过 0.2，如果输入的折扣超过 0.2，就给以提示。
为此可以在 AfterEdit 事件中设置代码：

```
If e.Col.name = "折扣" Then
    If e.Row("折扣") > 0.2 Then
        e.Row.DataRow.SetError("折扣", "折扣不能超过0.2!")
    Else
        e.Row.DataRow.SetError("折扣","")
    End If
End If
```

这样用户输入超过 0.2 的折扣后，会在折扣单元格显示一个红色的图标，表示这里输入的内容有误，当鼠标移到这个红色图标上时，会显示具体的错误信息：

	产品	客户	雇员	单价	折扣	数量	金额
1	PD05	CS03	EP04	17	0.1	650	9945
2	PD02	CS01	EP01	20	0.1	400	7200
3	PD05	CS04	EP02	17	0	320	5440
4	PD01	CS03	EP04	18	0.15	80	1224
5	PD03	CS03	EP04	10	0.21	490	3871
6	PD04	CS04	EP05	17.6	0	240	4224
7	PD05	CS03	EP04	17	0	111	1887

示例二

只有直接在表中输入数据，AfterEdit 事件才会生效，如果希望任何情况导致折扣超出 0.2，都会出现折扣提示，可以将 DataColChanged 事件的代码设置为：

```
If e.DataCol.name = "折扣" Then
    If e.DataRow("折扣") > 0.2 Then
```

```
        e.DataRow.SetError("折扣", "折扣不能超过0.2!")
    Else
        e.DataRow.SetError("折扣","")
    End If
End If
```

提示：上述方式并非只能用于错误提示，实际上你可以用于任何提示，只是这种方式多数时候用于错误提示而已。

4.4.2.4.8.6 在状态栏显示动态信息

如果你需要针对不同的表、不同的列、甚至不同状态的行，在状态栏显示动态信息，可以用全局表事件 AfterSelChange。
可以参考下面的代码：

```
Select Case e.Table.Name
    Case "表名一"
        StatusBar.Message1 = "信息一"
    Case "表名二"
        StatusBar.Message1 = "信息二"
    Case "表名三"
        Dim ColName As String
        ColName = e.Table.Cols(e.Table.ColSel).Name
        Select Case ColName
            Case "列名 1"
                StatusBar.Message1 = "信息 1"
            Case "列名 2"
                StatusBar.Message1 = "信息 2"
            Case Else
                StatusBar.Message1 = "就绪"
        End Select
    Case "表名四"
        Dim Val As String = e.Table.Current("某列名")
        Select Case Val
            Case "值 1"
                StatusBar.Message1 = "信息 1"
            Case "值 2"
                StatusBar.Message1 = "信息 2"
            Case Else
                StatusBar.Message1 = "就绪"
        End Select
    Case Else
        StatusBar.Message1 = "就绪"
End Select
```

如果你只是想针对不同的表、不同的行显示不同的提示信息，那么应该设置在全局表事件

CurrentChanged 中：

```
Select Case e.Table.Name
    Case "表名一"
        StatusBar.Message1 = "信息一"
    Case "表名二"
        StatusBar.Message1 = "信息二"
    Case "表名三"
        Dim Val As String = e.Table.Current("某列名")
        Select Case Val
            Case "值 1"
                StatusBar.Message1 = "信息 1"
            Case "值 2"
                StatusBar.Message1 = "信息 2"
            Case Else
                StatusBar.Message1 = "就绪"
        End Select
    Case Else
        StatusBar.Message1 = "就绪"
End Select
```

全局表事件 AfterSelChange 和 CurrentChanged 默认是对所有表开启的，不需要另外编写开启代码。

如果你只是想针对不同的表显示不同的信息，可以设置在项目事件 CurrentTableChanged 中：

```
Select Case CurrentTable.Name
    Case "表名一"
        StatusBar.Message1 = "信息一"
    Case "表名二"
        StatusBar.Message1 = "信息二"
    Case "表名三"
         StatusBar.Message1 = "信息三"
    Case Else
        StatusBar.Message1 = "就绪"
End Select
```

如果你只是想针对不同的主表显示不同的信息，可以设置在项目事件 MainTableChanged 中：

```
Select Case MainTable.Name
    Case "表名一"
        StatusBar.Message1 = "信息一"
    Case "表名二"
        StatusBar.Message1 = "信息二"
    Case "表名三"
         StatusBar.Message1 = "信息三"
    Case Else
        StatusBar.Message1 = "就绪"
End Select
```

4.4.2.4.8.7 事件与权限管理

权限管理会有专门的一章介绍，这里先简单介绍一下。
为便于权限管理，我们强列建议你为每一个用户指定一个分组，这样我们在编码进行权限管理的时候，就可以针对同一组的用户进行编码，而不用针对每一个用户进行编码。
权限管理有两种方式：

方式一

打开系统或者切换用户之后，都会触发 LoadUserSetting 事件，我们可以在这里设置代码，根据登录用户的分组进行权限设置。
示例
例如只有经理级别的用户，才能编辑表 A，可以在 LoadUserSetting 事件中加入下面的代码：

```
If User.Group = "经理" Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

如果你要求订单表的折扣列只能经理编辑，可以在 LoadUserSetting 事件中加入下面的代码：

```
If User.Group = "经理" Then
    DataTables("订单").DataCols("折扣").AllowEdit = True
Else
    DataTables("订单").DataCols("折扣").AllowEdit   = False
End If
```

方式二

在 Foxtable 中进行任何一项数据管理操作，例如增加行、删除行、锁定行、编辑数据等等，都会触发事件； 通过在事件中编码，可以进行和登录用户相关的条件判断，符合条件则允许操作，不符合条件则禁止操作。 这样我们可以进行更复杂的权限管理。
示例
例如公司规定，对于已经结帐的行，只有“经理”级别的用户才能删除，你可以在 BeforeDeleteDataRow 设置如下代码：

```
'如果当前用户不是经理级别，而且即将被删除的行已经结帐，那么取消删除。
If User.Group <> "经理" And e.DataRow("结帐") = True Then
    MessageBox.Show("对于已经结帐的行，只有经理才能删除！ ", "提示")
    e.Cancel = True
End If
```

如果在该表的 PrepareEdit 事件中加入代码：

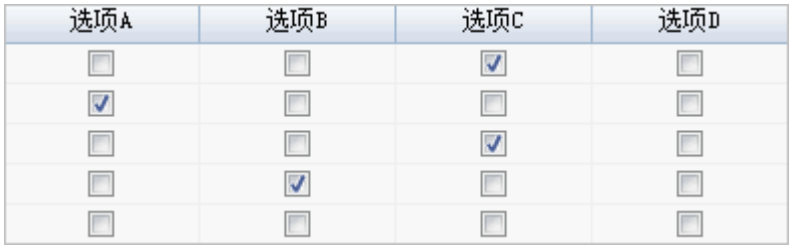
```
If e.Row("已结帐") = True AndAlso User.Group <> "经理" Then
    e.Cancel = True
End If
```

那么就只有经理级别的用户才能编辑已经结帐的行。
如果要求只有经理才能修改折扣列的内容，可以在 PrepareEdit 事件中加入代码：

```
If e.Col.Name = "折扣" AndAlso User.Group <> "经理" Then
    e.Cancel = True
End If
```

4.4.2.4.8.8 逻辑列的排他式选择

假定表中有四个逻辑列，要求每行只能选中一个，例如选中选项 A，会自动去掉选项 B、选项 C 和选项 D 的选中标记。



要实现此目的，只需将表的 DataColchanging 事件代码设置为：

```
Select Case e.DataCol.Name
    Case "选项A","选项B","选项C","选项D"
        If e.NewValue = True Then
            Dim nms() As String = {"选项A","选项B","选项C","选项D"}
            For Each nm As String In nms
                If nm <> e.DataCol.Name AndAlso e.DataRow(nm) = True Then
                    e.DataRow(nm) = False
                End If
            Next
        End If
End Select
```

4.4.2.4.8.9 自动复制行

假定表 A 有个逻辑列，希望在某行选中此逻辑列时，自动将此行复制到表 B。
为此可以将表 A 的 DataColChanged 事件代码设置为：

```
If e.DataCol.Name = "逻辑列名" AndAlso e.DataRow("逻辑列名") = True Then
    Dim dr As DataRow = DataTables("表B").AddNew
    For Each dc As DataCol In DataTables("表B").DataCols
        dr(dc.Name) = e.DataRow(dc.Name)
    Next
End If
```

上面的代码假定表A和表B的结构相同，且列名相同。

如果列名不同，或者只需复制部分列，可以参考下面的代码：

```
If e.DataCol.Name = "逻辑列名" AndAlso e.DataRow("逻辑列名") = True Then
    Dim nma() As String = {"A1","A2","A3","A4"} 'A表数据来源列
    Dim nmb() As String = {"B1","B2","B3","B4"} 'B表数据接收列
    Dim dr As DataRow = DataTables("表B").AddNew
    For i As Integer = 0 To nma.Length - 1
```

```
        dr(nmb(i)) = e.DataRow(nma(i))
    Next
End If
```

4.4.2.4.8.10 常用的编号生成方法

以下例子，实际测试的时候，如果表中已经有不符合规范的编号，请先删除之，否则会影响编号的生成结果。

一、按月生成编号

假定有个表，需要按月自动生成编号，四位年，两位月，最后三位是顺序号，如下图所示：

	日期	编号
1	2011-05-02	201105-001
2	2011-05-02	201105-002
3	2011-05-20	201105-003
4	2011-06-16	201106-001
5	2011-06-25	201106-002
6	2011-06-29	201106-003

要自动生成上述编号，只需将表事件 DataColChanged 的代码设置为：

```
If e.DataCol.Name = "日期" Then
    If e.DataRow.IsNull("日期") Then
        e.DataRow("编号") = Nothing
    Else
        Dim d As Date = e.DataRow("日期")
        Dim y As Integer = d.Year
        Dim m As Integer = d.Month
        Dim Days As Integer = Date.DaysInMonth(y,m)
        Dim fd As Date = New Date(y,m,1) '获得该月的第一天
        Dim ld As Date = New Date(y,m,Days) '获得该月的最后一天
        Dim bh As String = Format(d,"yyyyMM") '生成编号的前6位,4位年,2位月.
        If e.DataRow("编号").StartsWith(bh) = False '如果编号的前6位不符
            Dim max As String
            Dim idx As Integer
            max = e.DataTable.Compute("Max(编号)","日期 >= #" & fd & "# And 日期 <= #" & ld & "# And
[_Identify] <> " & e.DataRow("_Identify")) '取得该月的最大编号
            If max > "" Then '如果存在最大编号
                idx = CInt(max.Substring(7,3)) + 1 '获得最大编号的后三位顺序号,并加1
            Else
                idx = 1 '否则顺序号等于1
            End If
            e.DataRow("编号") = bh & "-" & Format(idx,"000")
        End If
    End If
End If
```

说明：

我们知道，上图中编号的前六位表示年月，后三位为顺序号，是需要递增的，递增顺序号的代码如下:

```
idx = CInt(max.Substring(7,3)) + 1 '获得最大编号的后三位顺序号,并加 1
```

以编号 201105-002 为例，从位置 7（也就是第 8 个字符，因为字符的顺序号是从 0 开始的）开始，取 3 个字符，就是取得最后的三个字符（位置分别为 7、8、9），得到的是“002”：

编号	2	0	1	1	0	5	-	0	0	2
位置	0	1	2	3	4	5	6	7	8	9

提示：本节的示例都假定主键列为_Identify，如果主键列不是_Identify，请将代码中的“_Identify”改为实际的主键列名。

二、按日生成编号

假定有个表，需要按天自动生成编号，四位年，两位月，两位日，最后三位是顺序号，如下图所示：

	日期	编号
1	2011-05-02	20110502-001
2	2011-05-02	20110502-002
3	2011-05-02	20110502-003
4	2011-05-03	20110503-001
5	2011-05-03	20110503-002
6	2011-05-03	20110503-003

要自动生成上述编号，代码更加简单：

```
If e.DataCol.Name = "日期" Then
    If e.DataRow.IsNull("日期") Then
        e.DataRow("编号") = Nothing
    Else
        Dim bh As String = Format(e.DataRow("日期"),"yyyyMMdd") '取得编号的8位前缀
        If e.DataRow("编号").StartsWith(bh) = False '如果编号的前8位不符
            Dim max As String
            Dim idx As Integer
            max = e.DataTable.Compute("Max(编号)","日期 = #" & e.DataRow("日期") & "# And [_Identify]
<> " & e.DataRow("_Identify")) '取得该天的最大编号
            If max > "" Then '如果存在最大编号
                idx = CInt(max.Substring(9,3)) + 1 '获得最大编号的后三位顺序号,并加1
            Else
                idx = 1 '否则顺序号等于1
            End If
            e.DataRow("编号") = bh & "-" & Format(idx,"000")
        End If
    End If
End If
```

三、按类别编号

假定有下图所示的一个表，编号根据类别生成，前两位为类别，后三位为顺序号：

	类别	编号
1	HA	HA001
2	HD	HD001
3	FX	FX001
4	FX	FX002
5	HA	HA002
6	HA	HA003
7	FX	FX003
8	HA	HA004

要自动生成上面的编号，可以将 DataColChanged 事件代码设置为：

```
Select e.DataCol.Name
    Case "类别"
        If e.DataRow.IsNull("类别") Then
            e.DataRow("编号") = Nothing
        Else
            Dim lb As String = e.DataRow("类别")
            If e.DataRow("编号").StartsWith(lb) = False '如果单据编号前缀不符
                Dim max As String
                Dim idx As Integer
                max = e.DataTable.Compute("Max(编号)","类别 = '" & lb & "' And [_Identify] <> '" & e.DataRow("_Identify")) '取得该类别的最大编号
                If max > "" Then '如果存在最大编号
                    idx = CInt(max.Substring(2,3)) + 1 '获得最大编号的后三位顺序号,并加 1
                Else
                    idx = 1 '否则顺序号等于 1
                End If
                e.DataRow("编号") = lb & Format(idx,"000")
            End If
        End If
End Select
```

四、按日期和类别编号

假定有个表，需要按月自动生成编号，根据工程代码按顺序编号，前 4 位是工程代码，然后是 4 位年，2 位月，最后 4 位是顺序号，如下图所示：

	工程代码	制单日期	单据编号
1	NGAY	2012-03-02	NGAY-201203-0001
2	NGAY	2012-03-03	NGAY-201203-0002
3	NGAY	2012-03-04	NGAY-201203-0003
4	NGAY	2012-03-05	NGAY-201203-0004
5	NGAY	2012-03-06	NGAY-201203-0005
6	NGAY	2012-03-07	NGAY-201203-0006
7	NGAY	2012-03-08	NGAY-201203-0007
8	NGAY	2012-03-09	NGAY-201203-0008
9	NGAY	2012-03-01	NGAY-201203-0009
10	QYGC	2012-03-02	QYGC-201203-0001
11	QYGC	2012-03-03	QYGC-201203-0002
12	QYGC	2012-03-04	QYGC-201203-0003
13	QYGC	2012-03-05	QYGC-201203-0004
14	QYGC	2012-03-06	QYGC-201203-0005
15	QYGC	2012-04-14	QYGC-201204-0001
16	QYGC	2012-04-14	QYGC-201204-0002
17	QYGC	2012-04-14	QYGC-201204-0003

要自动生成上面的编号，可以将 DataColChanged 事件代码设置为：

```
Select e.DataCol.Name
    Case "制单日期","工程代码"
        If e.DataRow.IsNull("制单日期") OrElse e.DataRow.IsNull("工程代码") Then
            e.DataRow("单据编号") = Nothing
        Else
            Dim d As Date = e.DataRow("制单日期")
            Dim y As Integer = d.Year
            Dim m As Integer = d.Month
            Dim Days As Integer = Date.DaysInMonth(y,m)
            Dim fd As Date = New Date(y,m,1) '获得该月的第一天
            Dim ld As Date = New Date(y,m,Days) '获得该月的最后一天
            Dim bh As String = e.DataRow("工程代码") & "-" & Format(d,"yyyyMM") & "-" '生成编号的前缀
            If e.DataRow("单据编号").StartsWith(bh) = False '如果单据编号前缀不符
                Dim max As String
                Dim idx As Integer
                Dim flt As String
                flt = "工程代码 = '"& e.DataRow("工程代码") & "' And 制单日期 >= #" & fd & "# And 制单日期 <= #" & ld & "# And [_Identify] <> " & e.DataRow("_Identify")
                max = e.DataTable.Compute("Max(单据编号)",flt) '取得该月的相同工程代码的最大单据编号
                If max > "" Then '如果存在最大单据编号
                    idx = CInt(max.Substring(12,4)) + 1 '获得最大单据编号的后四位顺序号,并加1
                Else
                    idx = 1 '否则顺序号等于1
                End If
```

```
            e.DataRow("单据编号") = bh & Format(idx,"0000")
        End If
    End If
End Select
```

4.4.2.4.8.11 保持列位置与列宽

我们可以将表的列位置和列宽调整好，并保存为一个表视图，然后在 AfterOpenProject 事件设置代码：

```
Tables("表名").OpenView("视图名")
```

这样不管运行过程中用户如何调整列位置或列宽，只要重新打开项目，即可恢复正常的设置。

视图对于关联表也有效，例如：

```
Tables("订单.订单明细").OpenView("视图名")
```

除了表视图外，也可以考虑用 SetColVisibleWidth 设置列位置与列宽 ，或者用 AutoSizeCols 方法自动调整列宽。

SetColVisibleWidth 可以和 GetColVisibleWidth 配合使用，使得两个表的显示列、列位置和列宽和保持一致，例如：

```
Dim cvs As String = Tables("订单").GetColVisibleWidth()
Tables("产品.订单").SetColVisibleWidth(cvs)
```

执行上述代码后，关联表 Tables("产品.订单")的显示列、列位置和列宽将和主表 Tables("订单")完全一致。

4.4.2.4.8.12 动态设置文件存储位置

如果没有使用 Foxtable 进行文件或图片管理，或者没有动态设置存储目录的需要，可以跳过本节内容。

我们知道在默认情况下，每当我们在文件型、多文件型、以及图片型列中插入一个文件时，程序会自动将该文件复制到项目文件所在目录的子目录 Attachments 下。

虽然我们通过设置列属性“存放目录”，可以指定其它存储位置，但是在需要大量文件的时候，我们可能并不希望文件全部存放在同一个目录下，而是根据文件名或其它条件将文件存放在不同的子目录下。

表事件 BeforeAttachFile 正是为了满足此类需要而提供的，该事件在向文件型、多文件型、以及图片型列中添加文件的时候执行。

e 参数属性:

DataTable： 触发事件的 DataTable

DataRow： 触发事件的 DataRow

DataCol: 触发事件的 DataCol

SourceFolder： 要添加文件的来源目录

FileName： 要添加文件的名称，你可以根据需要改名。

SubFolder： 指定存放文件的子目录

Cancel： 逻辑型，是否取消此次添加。

示例一

例如表中有部门和图片两列，希望不同部门的图片，放在各自的目录中，为此可在 BeforeAttachFile 事件加上下面的代码：

```
Select Case e.DataCol.Name
   Case "图片" '如果在图片列插入文件
      e.Subfolder = e.DataRow("部门")  '将此文件存放在目录名为部门名称的子目录中.
End Select
```

示例二

例如希望在添加文件的时候，根据文件的后缀名自动添加到不同的子目录，例如 BMP 文件自动添加到 BMP 子目录，DOC 文件添加到 DOC 子目录。

BeforeAttachFile 事件只需一行代码即可完成此任务：

```
e.SubFolder = e.FileName.SubString(e.FileName.LastIndexof(".") + 1)
```

示例三

假定希望上示例一的基础上，再增加判断文件类型的功能，如果不是图片文件，则拒绝插入：

```
Dim ext As String
ext = e.FileName.SubString(e.FileName.LastIndexof(".") + 1) '获得文件后缀名
Select Case ext
    Case "bmp","gif","jpg","png","tif", "wmf"
       e.SubFolder = ext
    Case Else
          MessageBox.Show("此列只能能插入图片文件!", "提示", MessageBoxButtons.OK,MessageBoxIcon.Information)
       e.Cancel = True '取消插入
End Select
```

示例四

如果表中有多个图片型或文件型列，那么应该在 BeforeAttachFile 判断列名，例如要求照片列只能插入 JPG 文件，而且存放在名为"照片"的子目录中，简历列只能插入 Word 文件，而且存放在名为"简历"的子目录中：

```
Dim ext As String
ext = e.FileName.SubString(e.FileName.LastIndexof(".") + 1)
Select Case e.DataCol.Name
    Case "照片"
       If ext = "jpg" Then
          e.SubFolder = ext
       Else
          MessageBox.Show("此列只能插入 jpg 文件!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
          e.Cancel = True
       End If
    Case "简历"
       If ext = "doc" Then
          e.SubFolder = ext
       Else
          MessageBox.Show("此列只能插入 Word 文件!", "提示", MessageBoxButtons.OK,
```

```
MessageBoxIcon.Information)
            e.Cancel = True
        End If
End Select
```

示例五

假定表中有三列，分别是作者、拍摄日期、照片，希望向照片列添加文件的时候，自动根据作者和拍摄日期创建子目录来存放文件，例如作者张三在 2010 年度拍摄的照片存放在 Attachments 的子目录"张三\2010"下：

```
Dim dr As DataRow = e.DataRow
If dr.IsNull("拍摄日期") Or Dr.IsNull("作者") Then
        MessageBox.Show("拍摄日期和作者不能为空!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True
Else
    e.SubFolder = dr("作者") & "\" & dr("拍摄日期").Year
End If
```

示例六

假定表中有两列，分别是身份证号码和照片，希望向照片列添加文件的时候，能自动根据身份证号码来重命名 此文件，为此可将 BeforeAttachFile 事件代码设置为：

```
Dim dr As DataRow = e.DataRow
If dr.IsNull("身份证号码")  Then
    MessageBox.Show("请先输入身份证号码!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True
Else
    e.FileName = dr("身份证号码") & ".jpg"
End If
```

4.4.2.4.8.13 禁止输入重复值

我们知道，列有禁止输入重复内容的属性，不过这个属性有局限，就是只有直接在表格中输入的时候才有效，如果你通过窗口输入或者是通过代码设置，同样可以输入重复内容。

禁止单列输入重复值

假定有个订单表，要求订单号列不能输入重复值，可以将这个表的 DataColChanging 事件代码设置为：

```
If e.DataCol.Name = "订单号" Then
    Dim dr As DataRow
    dr = e.DataTable.Find("订单号 = '" & e.NewValue & "'")
    If dr IsNot Nothing Then
        MessageBox.Show("此订单号已经存在!")
        e.Cancel = True
    End If
End If
```

上述代码设置完成后，不管以任何方式，都无法输入重复的订单号。

禁止多列输入重复值

假定产品表有型号和规格两列，禁止输入型号和规格相同的产品，可以将这个表的DataColChanging 事件代码设置为：

```
Select Case e.DataCol.name
    Case "型号","规格"
        Dim xh As String
        Dim gg As String
        If e.DataCol.Name= "型号" Then
            xh = e.NewValue
            gg = e.DataRow("规格")
        Else
            gg = e.NewValue
            xh = e.DataRow("型号")
        End If
        If xh > "" AndAlso gg  > "" Then
            Dim dr As DataRow = e.DataRow
            If e.DataTable.Find("型号 = '" & xh & "' And 规格 = '" & gg & "'") IsNot Nothing Then
                MessageBox.Show("已经存在相同型号和规格的行!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
                e.Cancel = True
            End If
        End If
End Select
```

上面的代码有点繁琐。

我们可以变通一下，改为 DataColChanged 事件事件代码 ，同样可以禁止输入型号和规格相同的行，而且代码会简洁一些：

```
Select Case e.DataCol.name
    Case "型号","规格"
        Dim dr As DataRow = e.DataRow
        If dr.IsNull("型号") = False AndAlso dr.IsNull("规格") = False Then
            If e.DataTable.Compute("Count([_Identify])","型号 = '" & dr("型号") & "' And 规格 = '" & dr("规格") & "'") > 1 Then
                MessageBox.Show("已经存在相同型号和规格的行!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
                e.DataRow(e.DataCol.Name) = e.OldValue '取消输入
            End If
        End If
End Select
```

不过两者还是有所差别的，DataColChanging 是在值发生变化前进行判断，属于提前拦截，而 DataColChanged 是在值发生变化后判断，属于事后弥补。

4.4.2.4.8.14 非关联表之间的数据同步

假定有一个产品表和一个库存表，两个表都有产品编号、产品名称、产品规格三列，但是没有建立关联，我们希望：

1、在产品表增加一个产品，库存表对应也增加此产品。

2、在产品表删除一个产品，库存表对应也删除此产品。

3、在产品表更改某产品的产品编号、产品名称或产品规格后，库存表也能同步修改。

设计步骤

1、将产品表的 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.name
    Case "产品编号"
        Dim dr As DataRow = DataTables("库存").Find("产品编号 = '" & e.OldValue & "'")
        If dr Is Nothing Then
            dr = DataTables("库存").AddNew()
            dr("产品编号") = e.DataRow("产品编号")
            dr("产品名称") = e.DataRow("产品名称")
            dr("产品规格") = e.DataRow("产品规格")
        Else
            dr("产品编号") = e.DataRow("产品编号")
        End If
    Case "产品名称","产品规格"
        Dim dr As DataRow = DataTables("库存").Find("产品编号 = '" & e.DataRow("产品编号") & "'")
        If dr IsNot Nothing Then
            dr(e.DataCol.Name) = e.DataRow(e.DataCol.Name)
        End If
End Select
```

2、将产品表的 DataRowDeleting 事件代码设置为：

```
Dim dr As DataRow = DataTables("库存").Find("产品编号 = '" & e.DataRow("产品编号") & "'")
If dr IsNot Nothing Then
    dr.Delete()
End If
```

4.4.2.4.8.15 编码计算环比

假定有下图所示的一个表，希望输入数量后，能自动计算出环比：

	日期	数量	环比
1	2012-03-01	63	0%
2	2012-03-02	66.2	5.08%
3	2012-03-03	65	-1.81%
4	2012-03-04	64.8	-0.31%
5	2012-03-05	67.3	3.86%
6	2012-03-06	66.3	-1.49%

要实现此目的，可以将 DataColChanged 事件代码设置为：

```
Select e.DataCol.Name
    Case "数量","日期"
        Dim dr1 As DataRow = e.DataRow '本日期的行
        Dim dr2 As DataRow = e.DataTable.Find("日期 < #" & e.DataRow("日期") & "#","日期 Desc") '前一日期的行
        Dim dr3 As DataRow = e.DataTable.Find("日期 > #" & e.DataRow("日期") & "#","日期") '下一日期的行
        If dr2 Is Nothing Then
            dr1("环比") = 0
        ElseIf dr2("数量") > 0
            dr1("环比") = (dr1("数量") - dr2("数量")) / dr2("数量")
        Else
            dr1("环比") = Nothing
        End If
        If dr3 IsNot Nothing
            If dr1("数量") > 0 Then
                dr3("环比") = (dr3("数量") - dr1("数量")) / dr1("数量")
            Else
                dr3("环比") = Nothing
            End If
        End If
End Select
```

通过这个例子，我们可以更加灵活地掌握 Find 的用法。

4.4.2.4.8.16 年龄和工龄的精确计算

方法一

假定表中有两列，分别是入职日期和工龄，需要根据入职日期精确计算出工龄。

可以如下设置 DataColChanged 事件：

```
Select Case e.DataCol.name
    Case "入职日期"
        If e.DataRow.IsNull("入职日期") Then
            e.DataRow("工龄") = Nothing
        Else
            Dim tp As TimeSpan = Date.today - CDate(e.DataRow("入职日期"))
            e.DataRow("工龄") = Math.Round(tp.TotalDays / 365.2422,2)
```

```
    End If
End Select
```

年龄也可以用同样的方法计算。

提示： 一年的平均天数是365.2422天。

方法二

如果要准确计算出工龄的年月日，例如：

	入职日期	工龄		
		年	月	日
1	1999-12-01	12	9	6
2	2011-03-21	1	5	17
3	2001-09-17	10	11	21
4	2006-05-21	6	3	17

可以考虑使用DateYMD函数，如下设置DataColChanged事件代码：

```
Select Case e.DataCol.name
  Case "入职日期"
    If e.DataRow.IsNull("入职日期") Then
      e.DataRow("工龄_年") = Nothing
      e.DataRow("工龄_月") = Nothing
      e.DataRow("工龄_日") = Nothing
    Else
      Dim y,m,d As Integer
      DateYMD(e.DataRow("入职日期"),Date.Today,y,m,d)
      e.DataRow("工龄_年") = y
      e.DataRow("工龄_月") = m
      e.DataRow("工龄_日") = d
    End If
End Select
```

最后，为了每次打开项目后，都能得到最新的工龄数据，可以在AfterOpenProject事件中，设置下面的代码：

```
DataTables("员工").DataCols("入职日期").RaiseDataColChanged()
```

4.4.2.4.8.17 根据身份证号码提取户籍

在 CaseStudy 目录下，有个 Access 文件“身份证籍贯编码.mdb”。
如果需要根据身份证号码自动提取户籍，可以将这个文件的数据导入到自己的系统中，然后设置 DataColChanged 事件代码：

```
Select Case e.DataCol.name
    Case "身份证"
        If e.DataRow.IsNull("身份证") Then
            e.DataRow("籍贯") = Nothing
        Else
            Dim bm As String = e.DataRow("身份证").SubString(0,6)
            Dim dr As DataRow = DataTables("身份证籍贯编码").Find("籍贯编码  = '" & bm & "'")
            If dr IsNot Nothing Then
                e.DataRow("籍贯") = dr("籍贯")
            End If
        End If
End Select
```

4.4.2.4.9 强行触发事件

Table 有两个特别的方法，分别是：
RaiseCurrentChanged： 强行触发此 Table 的 CurrentChanged 事件
RaisePositionChanged：强行触发此 Table 的 PositionChanged 事件。
示例
假定有个订单表，希望每次选择一个订单的时候，能够自动在状态栏显示此订单客户的总订单数、总订购数量，为此可以将订单表的 CurrentChanged 事件代码设置为：

```
Dim flt As String = "客户  = '" & Tables("订单").Current("客户") & "'"
Dim cnt As Integer = DataTables("订单").Compute("Count([_Identify])",flt)
Dim sum As Integer = DataTables("订单").Compute("Sum([数量])",flt)
StatusBar.Message1 = "订单数:" & cnt & "  订购数量:" & sum
```

现在有个新的要求，希望在订单表修改客户列或者数量列内容是，能够自动刷新上面统计结果，我们可以在DataColChanged加上类似的统计代码：

```
Select Case e.DataCol.name
    Case "客户","数量"
        If e.DataRow("_Identify") = Tables("订单").Current("_Identify") '如果发生变化的是当前行
            Dim flt As String = "客户  = '" & Tables("订单").Current("客户") & "'"
            Dim cnt As Integer = DataTables("订单").Compute("Count([_Identify])",flt)
            Dim sum As Integer = DataTables("订单").Compute("Sum([数量])",flt)
            StatusBar.Message1 = "订单数:" & cnt & "  订购数量:" & sum
        End If
End Select
```

这样的设计没有大的问题，但是同样的代码出现在两个地方，不仅代码冗余，而且不利于维护，因为如果要调整代码，就必须同时修改这两处的代码。

我们可以将DataColChanged事件代码改为：

```
Select Case e.DataCol.name
    Case "客户","数量"
        If e.DataRow("_Identify") = Tables("订单").Current("_Identify") '如果发生变化的是当前行
            Tables("订单").RaiseCurrentChanged() '触发订单表的CurrentChanged事件，相当于重新统计一次。
        End If
End Select
```

上述代码可以完成了同样的任务，但是代码精炼了很多，维护起来也方便了很多，因为统计代码只出现在CurrentChanged事件。

4.4.2.5 动态合成表达式之一

不少初次接触编程的用户，对于何时使用双引号，何时使用单引号，感到非常困惑。
其实很简单，代码中的字符串都是用双引号括起来的。
例如：

```
Dim s As String
s = "abcd"
```

而单引号主要用于表达式，表达式可以用于计算、筛选。
表达式本身是一个字符串，所以必须用双引号括起来；表达式内部的字符串，用单引号括起来。

例如：

```
Tables("订单").Filter = "[产品] = 'PD01'"
```

上面的代码意思是在订单表表中筛选出产品为 PD01 的行，筛选表达式本身是一个字符串，所以用双引号括起来，表达式中的 PD01 也是一个字符串，是表达式的一个内部字符串，所以用单引号括起来。

表达式中的日期用符号#括起来，数值则不需要任何符号括起来，这些和代码中的格式是一样的，唯一不同的是字符串用单引号括起来。

例如：

```
Tables("订单").Filter = "[产品] = 'PD01'"   '字符用单引号括起来
Tables("订单").Filter = "[日期] = #3/17/1999#"   '日期用#括起来
Tables("订单").Filter = "[折扣] = 0.1"   '数值直接使用
```

通过表达式列的Expression属性，可以动态设置其计算公式。

例如：

```
DataTables("表 A").DataCols("ID").Expression = "'CA' + [_Identify]"
```

实际编程的时候，经常要根据用户的输入，动态合成表达式。

例如要计算某个产品的销售数量，要计算的产品名称由用户输入，假定输入的结果保存在全局变量Vars("产品")中：

```
Dim Sum As Integer
Sum =  DataTables("订单").Compute("Sum(数量)", "[产品] = '" & Vars("产品") & "'")
```

Compute的第二个参数为计算条件表达式，该表达式分为三部分:

```
"[产品] = '" & Vars("产品") & "'"
```

假定全局变量Vars("产品")的值为PD01， 组合三部分的内容后，这个表达式就等效于：

```
"[产品] = 'PD01'"
```

假定表 A 和表 B 都有品名、型号两列，我们在表 A 中选择了一行，希望从表 B 中找出相同品名、规格的行：

```
Dim dr1 As DataRow = Tables("表A").Current.DataRow
Dim dr2 As DataRow
dr2 = DataTables("表B").Find("[品名]  = '" & dr1("品名") & "'And [规格] = '" & dr1("规格") & "'")
```

上面代码的最后一行，初看非常难以理解。
为便于理解，我们首先来看看正常的查找表达式：

```
"[品名] = '要找品名' And [规格] = '要找的规格'"
```

其中红色的部分要被 dr1 中的实际的品名和规格代替，所以最终的表达式成了:

```
"[品名]  = '" & dr1("品名") & "'And [规格] = '" & dr1("规格") & "'"
```

上面都是用字符型的值作为例子，表达式中字符型的值要用单引号括起来。
如果是日期型，要用符号#括起来。
例如我们要筛选出指定月份的订单，但是因为表达式没有求得日期月份的函数，我们只能这样设置代码：

```
Dim StartDate As Date '起始日期变量
Dim EndDate As Date '结束日期变量
Dim Month As Integer = 6 '指定月份
Dim Year As Integer = 1999 '指定年份
StartDate = New Date(Year,Month,1) '获得该月第一天
EndDate = New Date(Year,Month,Date.DaysInMonth(Year,Month)) '获得该月最后一天
Tables("订单").Filter = "[日期] >= #" & StartDate & "# And [日期] <= #" & EndDate & "#"
```

另外，如果是数值，则不需要括起来，直接使用即可，例如：

```
Dim v1 As Integer = 100
Dim v2 As Integer = 500
Tables("订单").Filter = "数量 >= " & v1 & " And 数量 <= " & v2
```

大家一定要记住表达式中的字符用单引号括起来，日期用符号#括起来，数值则不需要任何符号。例如 在编号为整数的情况下，有人写出了这样的代码：

```
Tables("表 A").Filter = "[编号] = '" & dr("编号") & "'"
```

然后问我为什么不能执行？我想原因你应该已经知道了，正确的代码是：

```
Tables("表 A").Filter = "[编号] = " & dr("编号")
```

DataTable 和 Table 都有很多属性和方法用到了表达式，我们必须掌握单引号和双引号的区别， 并学会动态合成表达式，才能设计出强大而又灵活的管理系统。

4.4.2.6 动态合成表达式之二

Foxtable 专门提供了一个名为 CExp 的函数，可以用更清晰的方式动态合成表达式。
语法：
CExp(StrExp, V0, V1, V2, ...)
StrExp: 要合成的表达式，包含位置标记，{0}表示第一个值，{1}表示第二个值，{2}表示第三个值...
V0,V1,V2： 用于替换位置标记的值，V0 替换{0}，V1 替换{1}，V0 替换{0}...
示例一
在命令窗口执行下面的代码：

```
Dim exp As String = "{0} + {1} + {2} = {3}"
exp = CExp(exp,1,3,5,9)
output.show(exp)
```

得到的结果为：1 + 3 + 5 = 9

可以看到CExp函数依次用1、3、5、9替换了{0}、{1}、{2}、{3}

示例二

定表A和表B都有产品、规格两列，我们在表A中选择了一行，希望从表B中找出相同产品、规格的行：

```
Dim dr1 As DataRow = Tables("表A").Current.DataRow
Dim dr2 As DataRow
Dim exp As String = CExp("[产品] = '{0}' And [规格] = '{1}'", dr1("产品") ,dr1("规格"))
dr2 = DataTables("表B").Find(exp)
```

示例三

同样要注意日期用符号#括起来，字符用符号'括起来，数字直接用，例如在命令窗口执行：

```
Dim cp As String = "PD01"
Dim sl As Integer = 500
Dim rq As Date = Date.Today
Dim exp As String
exp =  CExp("[产品] = '{0}' And [数量] > {1} And [日期] = #{2}#", cp,sl,rq)
output.show(exp)
```

得到的结果为：[产品] = 'PD01' And [数量] > 500 And [日期] = #2014/2/25#

如果我们不用CExp合成表达式，代码为：

```
Dim cp As String = "PD01"
Dim sl As Integer = 500
Dim rq As Date = Date.Today
Dim exp As String
exp =  "[产品]  = '" & cp & "' And [数量] > " & sl & "[日期] = #" & rq & "#"
output.show(exp)
```

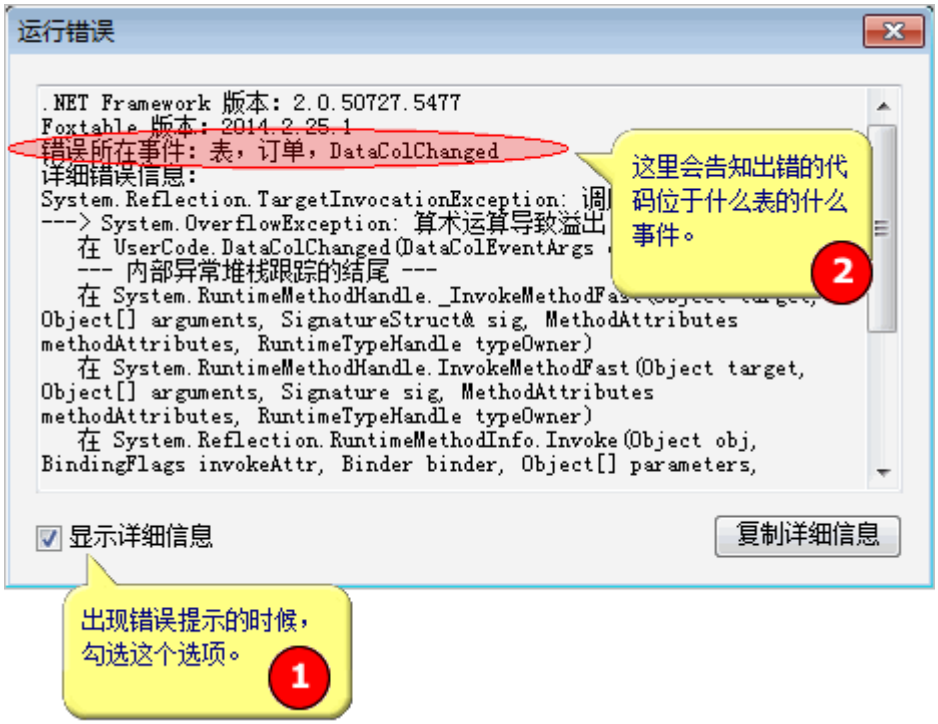
代码的长度差不多，但是使用CExp有好处，就是表达式始终是一个整体，无需分段拼接，所以代码更清晰，不容易出错。

尽管帮助文件中的例子大多使用分段拼接的方式，但我个人建议大家尽量用CExp函数，特别是新手。

4.4.2.7 调试技巧

很多错误只有在运行的时候，才会发现。
虽然对于运行中的错误，系统会明确提示是哪一个对象的哪一个事件发生了错误，以及错误的原因。

例如通过下图的错误提示，我们可以知道错误代码位于订单表的 DataColChanged 事件：



虽然错误提示会告诉我们是那一个事件的代码引发了错误，但是要具体定位是哪一行代码有问题，往往是一件艰巨的工作，特别是代码很长，逻辑较为复杂的时候。

用 MessageBox 跟踪

我们可以在代码段中插入 MessageBox.Show(x)，将代码分割成多段，x 是一个数字，例如：

```
代码段一
MessageBox.Show(1)
代码段二
MessageBox.Show(2)
代码段三
MessageBox.Show(3)
代码段四
MessageBox.Show(4)
代码段五
MessageBox.Show(5)
...
```

这样每执行一段代码，就会显示一个数字，在上面的例子中，如果显示 2 后发生了错误，那么说明问题发生在第三段代码，这样我们就可以重点分析该段代码了。

调试结束后，删除所有用于调试的 MessageBox.Show 语句即可。

在调试代码的时候，我们还可以利用 MessageBox.Show 来显示关键变量或数据，这对于分析错误原因，会有很大的帮助。

例如下面的代码用于加载雇员为当前登录用户的订单：

```
Dim fl As String
fl = "  雇员  = " & User.Name
DataTables("订单").LoadFilter = fl
```

```
DataTables("订单").Load()
```

执行上面的代码，会提示错误，显然最大的可能是我们合成的加载条件语法有误，为了验证，我们先屏蔽 Load 语句，暂时用 MessageBox.Show 显示一下合成的加载条件：

```
Dim fl As String
fl = "  雇员  = " & User.Name
DataTables("订单").LoadFilter = fl
MessageBox.Show(fl)
'DataTables("订单").Load()
```

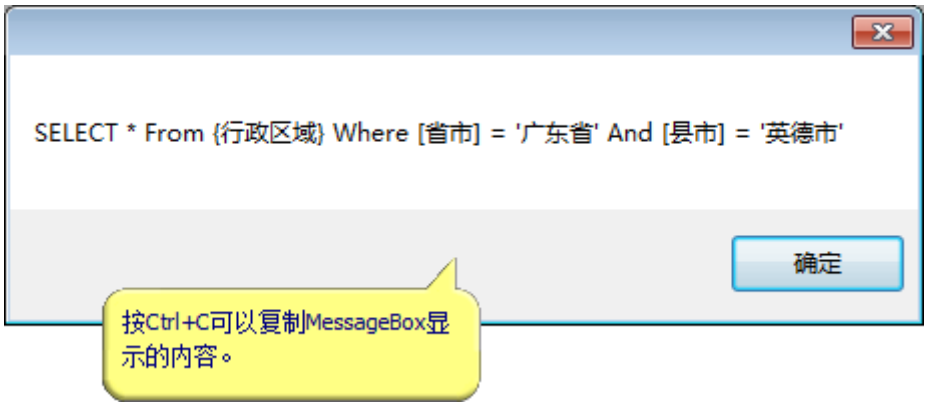
当登录用户为 EP01 的时候，执行上述代码，可以发现合成的条件为：雇员 = EP01。

显然正确的条件应该为：雇员 = 'EP01'，出错的原因在于表达式中的字符串没有用单引号括起来。

既然 知道了错误原因，我们就能写出正确代码：

```
Dim fl As String
fl = "  雇员  = '" & User.Name & "'"
DataTables("订单").LoadFilter = fl
DataTables("订单").Load()
```

在 MessaageBox 显示的对话框中，按"Ctrl+C"键，可以复制其显示的内容：



这在动态合成一些复杂的 SQL 语句或表达式时的时候很实用，例如我们可以从 MessageBox 中复制合成好的 SQL 语句，粘贴到 SQL 执行窗口进行执行调试，找出此 SQL 语句的问题后，再回头调整代码。

同样如果合成的是复杂的条件表达式，可以粘贴到对应表的表达式筛选窗口，根据此条件语句进行筛选测试，找出问题所在，然后回头调整代码。

可以在代码中多处嵌入 MessaageBox.Show 语句，在任何时候显示任何需要跟踪的信息，对代码执行结果进行全方位的跟踪。

用命令窗口跟踪

使用 MessageBox 跟踪简单直观，缺点是每次只能显示一个数据，无法查看之前显示的数据，不便于进行一些复杂的跟踪调试；而且 MessageBox 语句如果出现在一些特定的事件中，还可能会导致死循环，例如表事件 PrepareEdit。

如果不想采用 MessageBox 跟踪错误代码位置，或者由于特殊原因，无法使用 MessageBox，可以改用命令窗口来调试分析，例如：

```
Output.Clear() '清除命令窗口显示区现有内容
代码段一
Output.Show(1)
代码段二
Output.Show(2)
```

```
代码段三
Output.Show(3)
代码段四
Output.Show(4)
代码段五
Output.Show(5)
...
```

需要注意的是，如果要调试的是事件代码，必须先打开命令窗口，然后再调试。

命令窗口是非模式的，打开后可以继续进行任何操作，例如编辑数据、执行菜单命令等等，所以很适合调试代码。

需要的话，可以在调试过程中将命令窗口最小化，调试结束后再还原窗口，查看显示结果。

Output 的 Clear 方法我们是首次接触，此方法用于清除命令窗口信息显示区的内容。

同样，在调试代码的时候，我们可以利用 Output.Show 来显示关键变量或数据，这对于分析错误原因，会有很大的帮助 ，例如：

```
Dim dt As DataTable
Dim cmd As New SQLCommand
cmd.CommandText = "SELECT * From {订单} Where  日期  = #" & Date.Today & "#"
Output.show(cmd.CommandText) '显示合成的SQL语句
dt = cmd.ExecuteReader
Output.show(dt.DataRows.Count) '显示查询得到的行数
```

使用命令窗口跟踪的好处是不会打断程序的执行，而且历次显示的调试数据全部可查，但是执行前必须先打开命令窗口，也不够 MessageBox 来得简单直观。

代码是前后关联的

代码是前后关联的，有的时候某处代码执行时发生错误，并不是该处代码有问题，而是前面的代码造成的，例如：

```
Dim dr As DataRow
dr = DataTables("产品").Find("编号  = '03'") '找出编号为 03 的产品
dr("单价") = 0.1
```

如果产品表中不存在编号为 03 的记录，那么上面的代码执行到第三行：

```
dr("单价") = 0.1
```

将出现错误提示，但是显然该行代码本身并没有问题，我们需要的是加上判断语句：

```
Dim dr As DataRow
dr = DataTables("产品").Find("编号  = '03'") '找出编号为 03 的产品
If dr IsNot Nothing Then '如果找到的话
    dr("单价") = 0.1
End If
```

初学者对于错误代码位置的判断往往过于直接，我曾经遇到一个用户，他贴出了一段类似下面的代码：

```
'新加入的一段代码
Dim dr As DataRow
dr = DataTables("产品").Find("编号  = '03'") '找出编号为 03 的产品
dr("单价") = 0.1
```

然后问我什么会出错，我告诉他应该是最后这三行有问题，最好判断一下是否存在符合条件的行，

也就是 dr 是否为 Nothing。
可该用户并不这么认为，因为整段代码之前运行无误，在加入前面这一段代码后，运行才出错的，所以错误肯定是在前面这段新加入的代码。
他调试了一下午，最终没有找出问题，只好直接将项目发给我，我如下修改代码后，运行一切正常：

```
'新加入的一段代码
Dim dr As DataRow
dr = DataTables("产品").Find("编号  = '03'") '找出编号为 03 的产品
If dr IsNot Nothing Then '如果找到的话
    dr("单价") = 0.1
End If
```

此用户觉得很惊讶，明明是加入了前面这段代码后，运行才出错，为啥错误代码却在后面呢？
其实很好理解，因为前面新加入的代码可能修改了数据，导致产品表中不再存在编号为 03 的行，这样变量 dr 就等于 Nothing，从而导致运行出错。

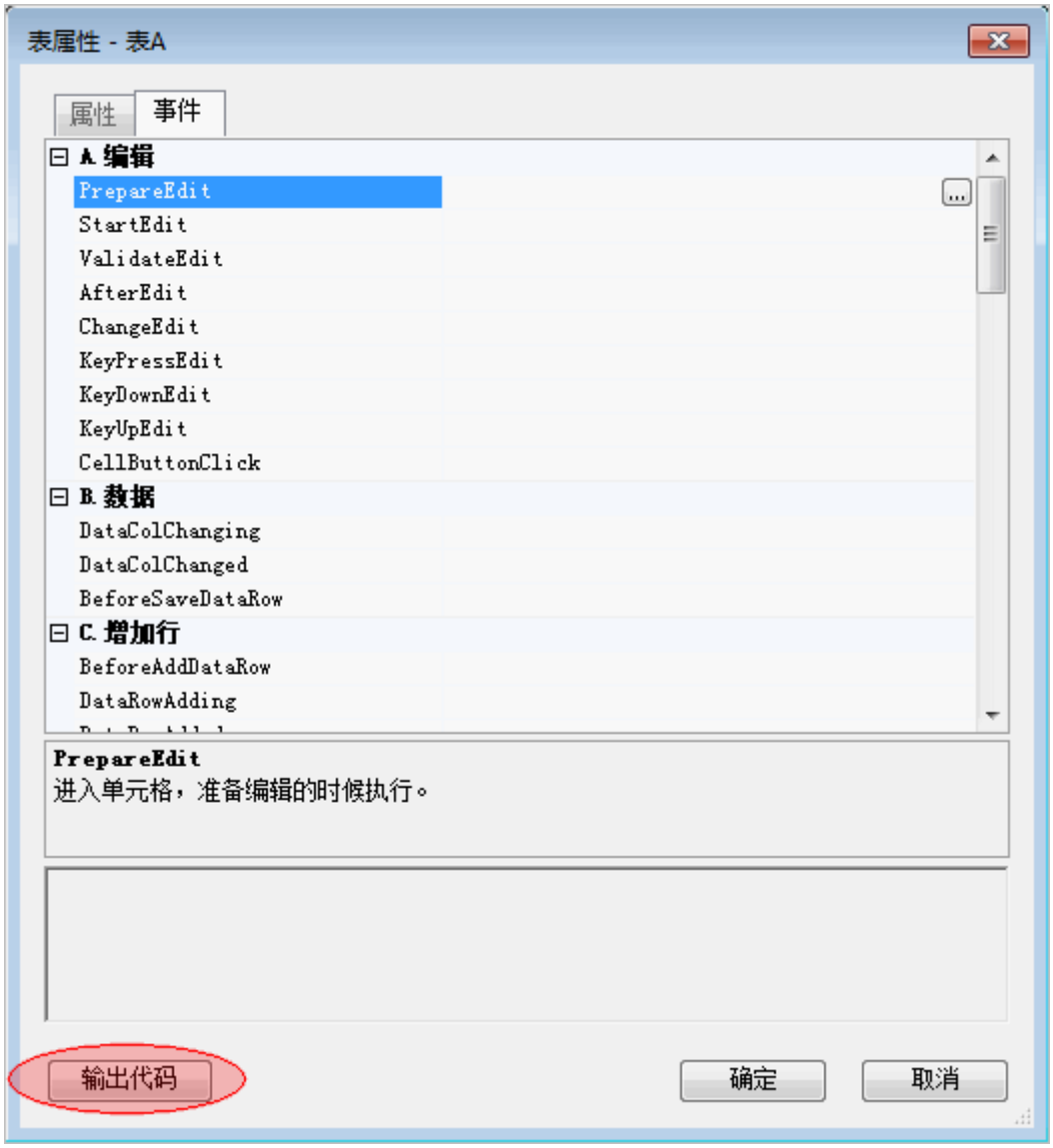
4.4.2.8 输出代码

在菜单的“杂项”功能区，单击下面的按钮：

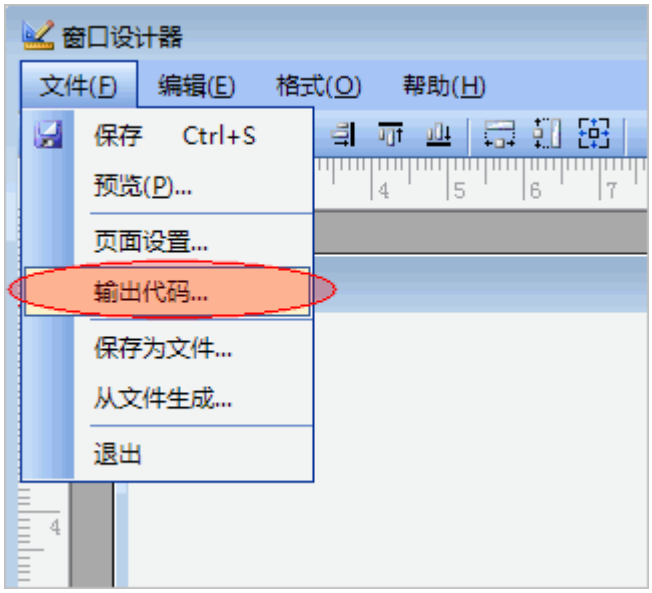


将自动生成并打开一个 rtf 文件，该文件包括所有的事件代码。
Foxtable 也可单独输出项目、菜单、自定义函数、某个表或某个窗口的事件代码。

例如要输出某个表的事件代码，只需在其属性设置窗口，单击“输出代码”按钮即可：



如果要输出某个窗口的代码，只需在窗口设计器中，执行“文件”菜单下的“输出代码”命令即可：



4.4.3 导入导出

4.4.3.1 Exporter

Exporter 用于导出数据。

属性

- **SourceTableName**
  字符型，指定导出表的名称。

- **Filter**
  字符型，指定导出条件表达式；请参考表达式的运算符和函数 和 条件表达式。

- **Fields**
  字符型，指定导出字段(列)，不同的字段，用逗号隔开，如果不指定，则导出所有字段。
  不能包括表达式列，如果导出文件一定要包括某一表达式列，那么请将该列改为普通数据列，并用代码进行计算。

- **Format**
  字符型，指定导出格式，包括 Access、Access2007、Excel、Excel2007、dBASE、Delimited、Fixed
  其中 Delimited 为符号分割的文本文件，Fixed 为固定列宽的文本文件。
  默认为 Access
  除 Excel2007 外，其余格式的文件均可自动创建，而 Excel2007 格式的目标文件必须事先已经存在。

- **PassWord**
  字符型。如果导出格式是 Access，可以用此属性指定目标文件的密码。
  如果目标文件是 Excel 格式，那么不能设置密码，否则无法正常导出。

- **Header**
  逻辑型，用于指定导出文本文件的时候，是否包括列名称。

- **FilePath**
  字符型，指定目标数据库文件名，含文件路径。
  如果导出格式是 dBASE 或者文本文件，那么只需指定路径。

- **NewTableName**
  字符型，指定导出后的表的名称。
  如果目标数据文件已经存在同名表，将不能导出。
  对于 dBASE 和文本格式，用文件名代替表名，不含扩展名，如果目标文件已经存在，也不能导出。

方法:

- **Export**
  开始导出数据。

请将下面的示例代码复制到命令窗口进行测试。

示例一

将订单表导出为 Access 格式：

```
Dim ex as New Exporter
ex.SourceTableName = "订单" '指定导出表
ex.filepath = "c:\Data\订单.mdb" '指定目标文件
ex.Export() '开始导出
```

示例二

如果希望打开一个保存文件对话框，导出到用户指定的文件中，可以参考下面的代码：

```
Dim dlg As New SaveFileDialog '定义一个新的SaveFileDialog
dlg.Filter= "Access文件|*.mdb" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then '如果用户单击了确定按钮
    Dim ex As New Exporter
    ex.SourceTableName = "订单" '指定导出表
    ex.filepath = dlg.FileName   '指定目标文件
    ex.Export() '开始导出
End If
```

示例三

将订单表导出为 Access 2007 格式：

```
Dim ex as New Exporter
ex.SourceTableName = "订单" '指定导出表
ex.FilePath = "c:\Data\订单.accdb" '指定目标文件
ex.Format = "Access2007" '导出格式为 Access 2007
ex.Export() '开始导出
```

示例四

将产品 PD01 的订单导出为 Excel 格式保存，并只导出指定的列：

```
Dim ex as New Exporter
ex.SourceTableName = "订单" '指定导出表
ex.FilePath = "c:\Data\订单.xls" '指定目标文件
ex.Format = "Excel" '导出格式为 Excel
ex.Fields = "日期,客户,数量,单价,产品" '指定导出字段
ex.Filter = "[产品] = 'PD01'" '指定导出条件
```

```
ex.Export() '开始导出
```

示例五

将订单表的数据导出为 dbf 格式：

```
Dim ex as New Exporter
ex.SourceTableName = "订单" '指定导出表
ex.FilePath = "c:\Data\" '指定目标文件路径
ex.Format = "dbase" '导出格式为 dBASE
ex.NewTableName = "订单数据" '指定文件名，注意无须扩展名
ex.Export() '开始导出
```

示例六

将订单表的数据导出为以逗号分割的文本文件：

```
Dim ex as New Exporter
ex.SourceTableName = "订单" '指定导出表
ex.FilePath = "c:\Data\" '指定目标文件路径
ex.Format = "Delimited" '导出格式为符号分割的文本文件
ex.NewTableName = "订单数据" '指定文件名，注意无须扩展名
ex.Export() '开始导出
```

执行之后，同时还会生成一个配置文件 schema.ini，里面有新生成文本文件的信息，还包括各列的名称、类型、宽度。

示例七

将订单表的数据导出为固定列宽的文本文件：

```
Dim ex as New Exporter
ex.SourceTableName = "订单" '指定导出表
ex.FilePath = "c:\Data\" '指定目标文件路径
ex.Format = "Fixed" '导出格式为固定宽度的文本文件
ex.NewTableName = "订单数据 2" '指定文件名，注意无须扩展名
ex.Export() '开始导出
```

执行之后，同时还会生成一个配置文件 schema.ini，里面有新生成文本文件的信息，还包括各列的名称、类型、宽度。

4.4.3.2 Importer

Importer 用于导入数据。

属性

- **SourcePath**
  指定要导入的数据文件，含路径。
  如果要导入的是 dBASE 或者 Text 文件，则只需指定路径。

- **SourceTableName**
  指定要导入的表的名称。
  如果要导入的是 dBase 或者 Text 文件，用文件名表示，无须扩展名。

- **Format**
  指定导入格式，包括 Access、Access2007、Excel、Excel2007、dBASE、Delimited、Fixed 等格式
  其中 Delimited 为符号分割的文本文件，Fixed 为固定列宽的文本文件。
  默认为 Access。
  导入文本文件的时候，最好配合有 schema.ini 文件，以便于指定列的类型和宽度。

- **PassWord**
  指定要导入的 Access 文件的密码。

- **Filter**
  指定导入条件。

- **Fields**
  指定要导入的字段(列)，不同的字段，用逗号给开，如果不指定，则导入所有字段。

- **NewTableName**
  指定导入后的表名。

- **NewTableCaption**
  指定导入后的表标题。

- **Header**
  逻辑型，如果导入的是文本文件，用此属性指定第一行是否是列名称。

方法

- **Import**
  开始导入数据。

示例一

导入 Access 文件：

```
Dim ip as New Importer
ip.SourcePath = "c:\Data\销售管理.mdb" '指定数据文件
ip.SourceTableName = "订单" '指定要导入的表
```

```
ip.NewTableName ="订单" '导入后的表名
ip.Import()
```

示例二

如果希望出现一个打开文件对话框，导入用户指定的文件，可以参考下面的代码：

```
Dim dlg As New OpenFileDialog '定义一个新的OpenFileDialog
dlg.Filter= "Access文件|*.mdb" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then '如果用户单击了确定按钮
    Dim ip As New Importer
    ip.SourcePath = dlg.FileName '指定数据文件
    ip.SourceTableName = "订单" '指定要导入的表
    ip.NewTableName ="订单" '导入后的表名
    ip.Import()
End If
```

示列三

导入 Excel 文件：

```
Dim ip as New Importer
ip.SourcePath = "c:\Data\销售管理.xls" '指定数据文件
ip.SourceTableName = "订单" '指定要导入的表
ip.NewTableName ="订单" '导入后的表名
ip.Format = "Excel" '指定导入格式
ip.Import()
```

一些 Excel 文件，可能需要在表名后面加上“$”，才能正常导入，例如：

```
Dim ip as New Importer
ip.SourcePath = "c:\Data\销售管理.xls" '指定数据文件
ip.SourceTableName = "订单$" '指定要导入的表
ip.NewTableName ="订单" '导入后的表名
ip.Format = "Excel" '指定导入格式
ip.Import()
```

示例四

导入 DBF 文件：

```
Dim ip as New Importer
ip.SourcePath = "c:\Data\" '指定目录
ip.SourceTableName = "订单" '指定要导入的文件“订单.DBF”
ip.NewTableName ="订单" '导入后的表名
ip.Format = "dbase" '指定导入格式
```

```
ip.Import()
```

示例五

导入文本文件：

```
Dim ip as New Importer
ip.SourcePath = "c:\Data\" '指定目录
ip.SourceTableName = "订单" '指定要导入的文件“订单.txt”
ip.NewTableName ="订单" '导入后的表名
ip.Format = "Delimited" '指定导入格式
ip.Import()
```

4.4.3.3 Merger

Merger 用于合并数据。
合并的速度要远低于导入，所以如果有大量数据要引入管理项目中，最好用导入。

属性

- **SourcePath**
  指定要要合并的数据库文件，含路径。
  如果要合并的是 dBASE 或者 Text 文件，则只需指定路径。

- **ConnectionName**
  指定数据源名称，从此数据源中合并数据。
  SourcePath 和 ConnectionName 属性只需指定一个，如果同时指定，则 SourcePath 属性失效。

- **SourceTableName**
  指定要合并的表的名称。
  如果要合并的是 dBase 或者 Text 文件，用文件名表示，无须扩展名。

- **Format**
  指定合并格式，包括 Access、Access2007、Excel、Excel2007、dBASE、Delimited、Fixed 等格式
  其中 Delimited 为符号分割的文本文件，Fixed 为固定列宽的文本文件。
  默认为 Access。
  合并文本文件的时候，最好配合有 schema.ini 文件，以便于指定列的类型和宽度。

- **PassWord**
  指定要合并的 Access 文件的密码。

- **Filter**
  指定合并条件。

- **Fields**
  指定要合并的字段(列)，不同的字段用逗号隔开，如果不指定，则合并所有字段。

- **DataTableName**
  指定接收合并数据的数据表的名称，这个表必须是当前项目中已经存在的表。

- **Header**
  逻辑型，如果合并的是文本文件，用此属性指定第一行是否是列名称。

- **FieldMaps**
  这是一个集合，要合并的表和接收数据的表的字段名称存在不一致的情况下，可以用此集合指定两个表之间字段的对应关系。
  假定要合并的表中某一字段的名称为“ID”，合并的时候，需要将该列的内容合并到接收表中的"编号"列，可以设置代码：
  Dim mg As New Merger
  mg.FieldMaps.Add("ID","编号")

方法

- **Merge**
  开始合并数据。

示例一

合并 access 文件：

```
Dim mg As New Merger
mg.SourcePath = "c:\data\销售管理.mdb"
mg.SourceTableName = "订单"
mg.DataTableName = "订单"
mg.Merge()
```

示例二

如果希望出现一个打开文件对话框，合并用户指定的文件，可以参考下面的代码：

```
Dim dlg As New OpenFileDialog '定义一个新的OpenFileDialog
dlg.Filter= "Access文件|*.mdb" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then '如果用户单击了确定按钮
    Dim mg As New Merger
    mg.SourcePath = dlg.FileName
    mg.SourceTableName = "订单"
```

```
    mg.DataTableName = "订单"
    mg.Merge()
End If
```

示例三

从指定的数据源中合并数据：

```
Dim mg As New Merger
mg.ConnectionName = "nwnd" '指定数据源名称
mg.SourceTableName = "订单" '指定要合并的表
mg.DataTableName = "订单" '指定接收数据的表
mg.Merge() '开始合并
```

示例四

FieldMaps 和 Filter 属性的用法示例：

```
Dim mg As New Merger
mg.SourcePath = "c:\data\销售管理.accdb"
mg.Format = "Access2007" '指定格式
mg.SourceTableName = "订单" '指定要合并的表
mg.DataTableName = "订单" '指定接收数据的表
mg.FieldMaps.Add("ID","编号") '将指定 ID 列的数据合并到编号列中
mg.Filter = "[日期] > #5/31/1999#" '指定合并条件
mg.Merge() '开始合并
```

示例五

合并 DBF 文件：

```
Dim mg As New Merger
mg.SourcePath = "c:\data\" '指定数据文件所在目录
mg.Format = "dbase" '指定格式
mg.SourceTableName = "订单" '指定要合并的 dbase 文件，无需扩展名
mg.DataTableName = "订单" '指定接收数据的表
mg.Merge() '开始合并
```

合并"c:\data\订单.dbf"文件的数据到当前项目的订单表

示例六

合并文本文件：

```
Dim mg As New Merger
mg.SourcePath = "c:\data\" '指定数据文件所在目录
```

```
mg.Format = "Delimited" '指定格式
mg.SourceTableName = "订单" '指定要合并的文件，无需扩展名
mg.DataTableName = "订单" '指定接收数据的表
mg.Merge() '开始合并
```

合并文本文件的时候，最好配合有 schema.ini 文件。

示例七

合并 Excel 文件:

```
Dim mg As New Merger
mg.SourcePath = "c:\data\销售管理.xls"
mg.Format = "excel" '指定格式
mg.SourceTableName = "订单" '指定要合并的表
mg.DataTableName = "订单" '指定接收数据的表
mg.Merge() '开始合并
```

一些 Excel 文件，可能需要在表名后加上符号$才能正常导入，例如：

```
Dim mg As New Merger
mg.SourcePath = "c:\data\销售管理.xls"
mg.Format = "excel" '指定格式
mg.SourceTableName = "订单$" '指定要合并的表
mg.DataTableName = "订单" '指定接收数据的表
mg.Merge() '开始合并
```

示例八

下面是一个批量合并的例子，假定目录 c:\data 下有多个 Excel 文件，这些文件都有一个名为订单的表，需要将这些表中的数据批量并到 Foxtable 的订单表中：

```
For Each fl As String In FileSys.GetFiles("c:\Data")
    Dim mg As New Merger
    mg.SourcePath = fl
    mg.Format = "excel" '指定格式
    mg.SourceTableName = "订单" '指定要合并的表
    mg.DataTableName = "订单" '指定接收数据的表
    mg.Merge() '开始合并
Next
```

示例九

也可以利用打开文件对话框一次选择多个文件来批量合并，例如：

```
Dim dlg As New OpenFileDialog
dlg.Filter = "Access文件|*.mdb"
```

```
dlg.MultiSelect = True '允许选择多个文件
If dlg.ShowDialog =DialogResult.OK Then
    For Each fl As String In dlg.FileNames
        Dim mg As New Merger
        mg.SourcePath = fl
        mg.SourceTableName = "订单"
        mg.DataTableName = "订单"
        mg.Merge()
    Next
End If
```

4.4.3.4 CompactToAccess

这是一个函数，用于将当前项目中的内部表全部保存为一个 Access 文件。
语法：
CompactToAccess(File, PassWord)
File:　　目标文件名
PassWord:　为目标文件设置的密码。

例如在命令窗口执行：

CompactToAccess("c:\data\test.mdb", "")

即可将当前项目保存为一个 Access 文件，密码为空。

4.4.3.5 CompactAccessFile

压缩指定的 Access 文件。
语法：
CompactAccessFile(File, PassWord)
File:　　要压缩的 Access 文件名，含路径。
Password: 可选参数，用于设置 Access 文件的密码。
示例
CompactAccessFile("C:\Data\订单.mdb","123")

4.4.3.6 编码合并文本文件

假定有个文本文件，有四列，不同的列之间用 Tab 符号分割，现在需要将其合并到表 A 中，可以参考下面的代码：

```
1 Dim dlg As new OpenFileDialog
2 dlg.Filter = "文本文件|*.txt"
```

```
3  Tables("表A").StopRedraw
4  If dlg.ShowDialog = DialogResult.OK Then
5      Dim strs As String = FileSys.ReadAllText(dlg.FileName)
6      strs = strs.Replace(vblf,"")
7      Dim rs() As String = strs.Split(vbcr)
8      For i As Integer = 0 To rs.Length - 1
9          Dim cs() As String = rs(i).Split(vbtab)
10         If cs.Length = 4 Then
11             Dim dr As DataRow = DataTables("表A").Addnew()
12             dr("第一列") = cs(0)
13             dr("第二列") = cs(1)
14             dr("第三列") = cs(2)
15             dr("第四列") = cs(3)
16         End If
17     Next
18 End If
19 Tables("表A").ResumeRedraw
```

代码说明：

1、第3行和第19行代码是配合使用的，在合并数据的过程中，停止绘制表A，提高运行效率，也避免屏幕闪烁。
2、文本文件中每行内容的结尾其实是两个字符，分别是换行(vblf)和回车(vbcr),第6行代码将读取的文本文件中的换行符(vblf)清除掉，仅保留回车符。
3、第7行代码将读取的文本文件内容拆分成一个数组，数组中每个元素对应中文本文件中的一行内容。
4、第9行代码将每行文本内容拆分成单个的值。
5、有的文本文件，第一行是标题，此时第8行代码应该改为：

```
For i As Integer = 1 To rs.Length - 1
```

4.4.3.7 保存为 Excel

Table 提供了一个 SaveExcel 方法，可以将数据保存为 Excel 格式，而且提供了很多选项，比 Exporter 更为方便。
请参考：SaveExcel

4.4.4 计划管理

4.4.4.1 计划管理

计划用于按照指定的时间间隔，执行特定的代码。

1、在“管理项目”功能区，单击下面的按钮：



2、出现计划管理窗口，可以在这里增加、删除、设置计划：



除了名称和代码外，计划的属性还包括：

间隔：　计划执行的时间间隔，单位为毫秒。

执行：　是否执行计划。

允许暂停：是否允许用户通过单击菜单中的“计划暂停”按钮，来暂停执行此计划。

3、在单击“确定”按钮退出计划管理窗口后，所设置的计划即刻生效，如果要临时暂停计划的执行，可单击菜单中的计划暂停按钮：



示例一

如果你希望每 5 分钟自动保存一次数据，可以增加一个计划，计划的执行间隔为 300000（毫秒），也就是 5 分钟，计划的代码设为：

```
For Each dt As DataTable In DataTables
    dt.Save()
Next
```

示例二

假定希望系统每十分钟自动备份一次项目，可以增加一个计划，计划的执行间隔 600000(毫秒)，也就是 10 分钟，计划的代码为：

```
Static dt As Date '静态变量,用于保存上次备份的时间
Dim tp As TimeSpan
tp = Date.Now - dt
If tp.TotalHours >= 1 Then '如果距离上次备份已经超过一个小时
    dt = Date.Now
    Syscmd.Project.Backup("c:\data", True) '自动备份到c:\data目录
End If
```

4.4.4.2 MyTimers

利用 MyTimers 集合，可以获得指定名称的计划，例如：

```
MyTimers("计划 1")
```

计划的属性有：

Interval：执行间隔，单位为毫秒。

Enabled: 是否执行，设为 False，暂停执行该计划。

例如：

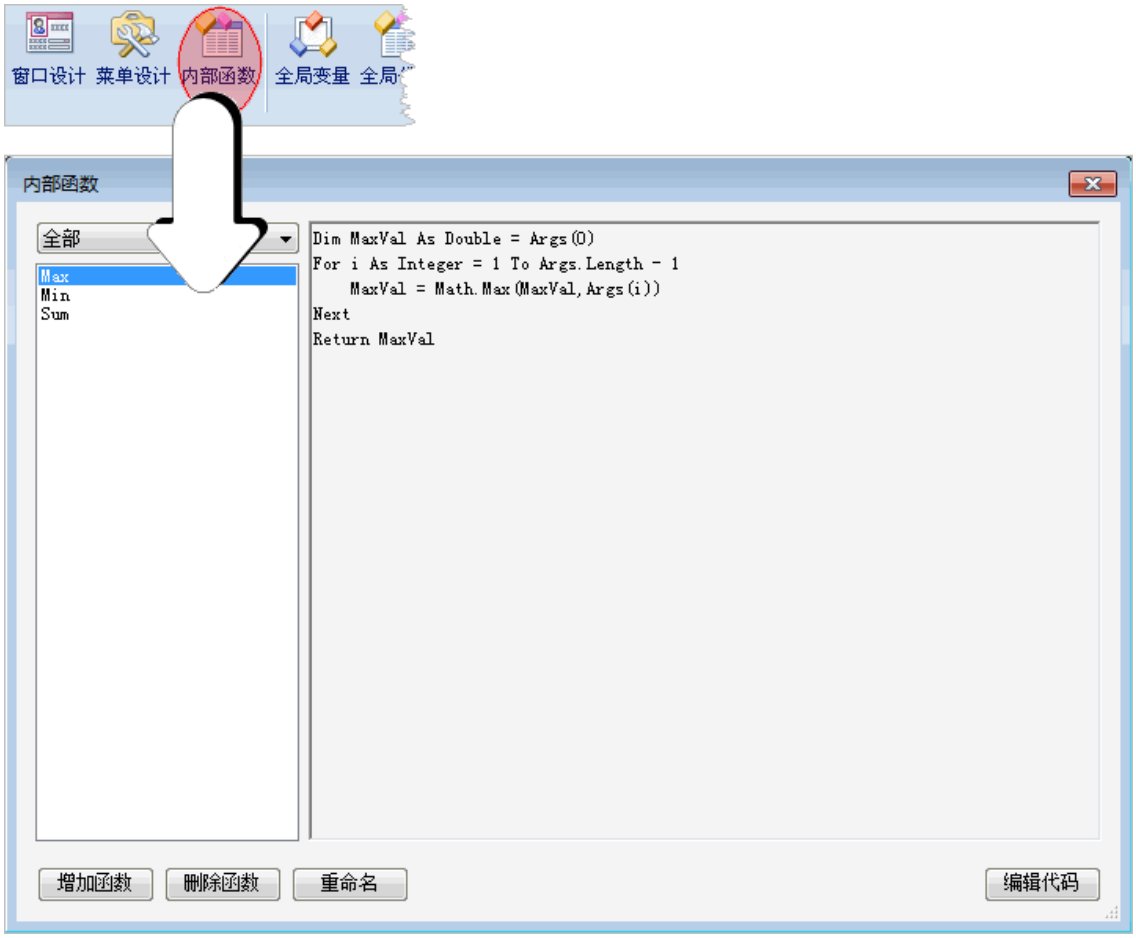
```
MyTimers("计划 1").Enabled = False '暂停计划 1
MyTimers("计划 2").Interval = 4000 '计划 2 的执行间隔改为 4 秒
```

4.4.5 自定义函数

4.4.5.1 自定义函数

1、定义函数

在菜单的“管理项目”功能区，单击“内部函数”按钮，即可定义自己的函数：



自定义函数有一个 Args 参数，这是一个 Object 类型的数组，用于获得传递给该函数的参数，例如 Args(0)表示第一个参数，Args(1)表示第二个参数。

参数的个数不限，而且因为是 Object 类型，所以任何类型的数据都可以作为参数传递给自定义函数。

需要注意的是，自定义函数中的 Return 语句必须有返回值，否则会报错，如果要中断函数的执行，又不需要返回任何值，可以：

```
Return Nothing
```

2、执行函数

Foxtable 提供了一个 Functions 类型，用于管理和执行自定义函数，执行自定义函数的语法为：

Functions.Execute(函数名，参数 1，参数 2，参数 3....)

示例一

增加一个自定义函数，名称为 Max，代码为：

```
Dim MaxVal As Double = Args(0)
For i As Integer = 1 To Args.Length - 1
    MaxVal = Math.Max(MaxVal,Args(i))
Next
Return MaxVal
```

该自定义函数可以比较任意个数的参数，并返回最大的一个，例如在命令窗口执行：

```
Output.Show(Functions.Execute("Max",1,3,5,7,9))
```

即可显示 1、3、5、7、9 几个数值中的最大数 9。

示例二

增加一个自定义函数，名称为 ClearDataRow，代码为：

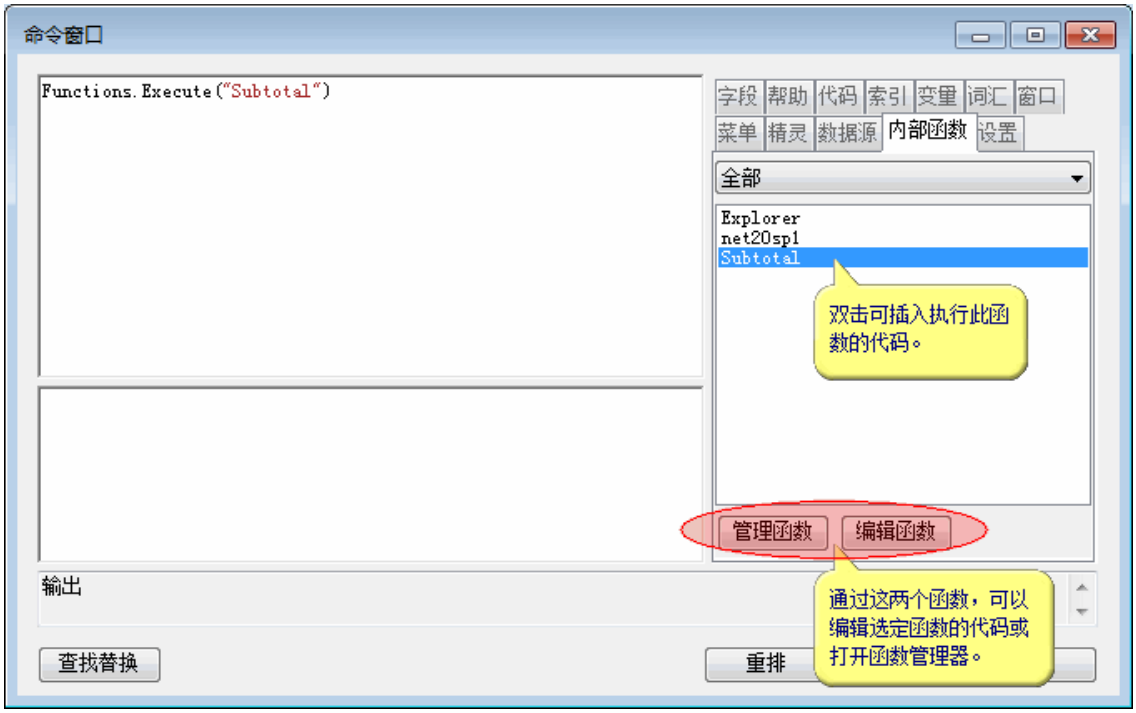
```
Dim dr As DataRow = Args(0)
For Each dc As DataCol In dr.DataTable.DataCols
    If dc.Expression = "" Then '有表达式的列,不能直接赋值
        dr(dc.Name) = Nothing
    End If
Next
```

以后就可以利用该函数清空任意一行的数据，例如清空当前表选定行的数据：

```
Functions.Execute("ClearDataRow",CurrentTable.Current.DataRow)
```

3、代码编辑器与函数

代码编辑器可以自动列出已经定义好的内部函数，双击函数名可以直接插入执行此函数的代码，还提供了两个按钮，分别用于编辑选定函数的代码和打开函数管理器：



4.4.5.2 动态管理函数

普通用户可以忽略本节内容。

Functions 不仅可以用来执行自定义函数，还可以动态管理函数。

Functions 提供了下述方法管理函数：

- Add

  增加自定义函数。

  语法：

  Add(Name,Code)

  Name：函数名称

Code：代码

- Remove
  删除用 Add 方法增加的自定义函数，通过系统菜单定义的函数，是无法删除的。

  语法：

  Remove(Name)

  Name：要删除的函数名称

- Complie
  重新编译函数。
  增加或删除函数后，必须执行此方法才能生效。

编译函数是需要时间的，所以动态函数一般用于对性能要求不高，但是对灵活性要求很高的场合。
我们的建议是尽量不要使用动态函数，而应该事先定义好函数。
示例
下面的代码定义了一个简单的 Sum 函数：

```
Dim Code As string
Code = "Dim Val1 As Integer = 1" & vbcrlf
Code = Code & "Dim Val2 As Integer = 2" & vbcrlf
Code = Code & "Dim Sum As Integer = Val1 + Val2" & vbcrlf
Code = Code & "Return Sum"
Functions.Add("Sum",Code)
Functions.Complie()
```

这样我们就可以在其余地方调用此函数：

```
Functions.Execute("Sum")
```

4.4.6 杂谈

4.4.6.1 再谈 Table 和 DataTable

有了前面的知识，我们可以更深入地理解 Table 和 DataTable 的区别。
示例一
假定有一个订单表，这个表有 1000 行，假定其中 100 个订单是订购产品 PD01 的，我们在该表筛选出 PD01 的订单，现在你在订单表看到的行数是 100 行。
我们在窗口执行下面的代码：

```
For Each r As Row in Tables("订单").Rows
    r("折扣") = 0.2
Next
```

上面的代码，将你看到的 100 行(订购 PD01 的订单)的折扣设为 0.2。
我们再来测试下面的代码：

```
For Each dr As DataRow in DataTables("订单").DataRows
    dr("折扣") = 0.2
```

```
Next
```

上面的代码将所有 1000 行(全部订单)的折扣设为 0.2

总之，如果你要对看得见的行进行操作，你就用 Table，如果你要对所有行进行操作，你就用 DataTable。

示例二

DataTable 中的行用 DataRow 表示，Table 中的行用 Row 表示。

Table 的数据来自于 DataTable，所以每一个 Row 都对应一个 DataRow，Row 有一个名为 DataRow 的属性，用于返回该 Row 所对应的 DataRow。

例如判断订单表的选定行是否是新增行，可以用下面的代码：

```
If Tables("订单").Current.DataRow.RowState = DataRowState.Added Then
End If
```

提示：Row 并没有 RowState 方法，所以要 判断其是否是新增行，必须使用 DataRow 的 RowState 属性。

同样 DataTable 中的列用 DataCol 表示，Table 中的列用 Col 表示，每一个 Col 都对应一个 DataCol，Col 有一个名为 DataCol 的属性，用于返回该 Col 对应的 DataCol。

例如要重置当前表选定列，但是 Col 并没有重置列的方法，我们只能调用 DataCol 的 RaiseDataColChanged 方法：

```
With CurrentTable
    .Cols(.ColSel).DataCol.RaiseDataColChanged()
End With
```

对应的，Table 也有一个名为 DataTable 的属性，用于返回该 Table 所属的 DataTable。

例如要保存当前表，因为 Table 没有保存命令，只能调用 DataTable 的 Save 方法：

```
CurrentTable.DataTable.Save()
```

示例三

上面的例子提到，Table、Row、Col 都有对应的属性，用于返回其所属的 DataTable、DataRow、DataCol。

但是却不可能反过来操作，因为一个 DataTable，可能对应有多个 Table。

不过通过 Table 的 FindRow 方法，我们可以获得某一个 DataRow 在 Table 中的位置。

例如我们希望在订单表中，将光标定位到最近一次订购 PD01 的行上：

```
Dim dr As DataRow
Dim po As Integer
dr = DataTables("订单").Find("产品 = 'PD01'","日期 Desc") '找出最后一次订购 PD01 产品的 DataRow
po = Tables("订单").FindRow(dr)'找出该 DataRow 在 Table 中的位置
If po >= 0 Then '如果找到
    Tables("订单").Position = po '选定
End If
```

因为 Table 可能会进行筛选，所以一个 DataRow 可能不会出现在 Table 中，此时 FindRow 方法会返回-1，表示没有找到指定的 DataRow。

再例如正在订单表操作，希望在订单表中选定一行时，客户表的光标能够自动定位到该客户，这样我们一旦从订单表回到客户表，即可看到刚刚所选订单的客户资料。

为此我们在订单表的 CurrentChanged 事件中设置如下代码：

```
Dim wz As Integer
Dim dr As DataRow
dr = DataTables("客户").Find("[客户 ID] = '" & e.Table.Current("客户 ID") & "'")
If dr IsNot Nothing Then
    wz = Tables("客户").FindRow(dr)
    If wz >= 0 Then
        Tables("客户").Position = wz
    End If
```

```
EndIf
```

这种在不同表之间联动的技巧应该掌握，也许你觉得通过关联，不就可以在订单表中以关联表的形式显示客户信息吗？

但是：

1、有的时候我们并不希望建立太多的关联。

2、即使建立了关联，我们也可能关闭了 Foxtable 的双向关联特性，这样就无法以关联表的形式查看父表数据了。

示例四

其实 Table 也有 Find 方法，不过 Table 的 Find 方法是根据内容查找，而不是根据表达式来查询，返回的是行的位置。

例如：

```
With CurrentTable
    Dim r As Integer
    r = .Find("PD01", 0, "产品", False, False, True)
    Ifr > - 1 Then '如果找到符合条件的行
        .Position = r
    End If
End With
```

Table 的 Find 方法功能较弱，但是使用简单，通常用于进行简单的查找和定位。

Table 还有一个 FindRow 方法，可以使用表达式进行查找，例如订单表的公司名称列中，查找包括“湛江”二字的单位：

```
With Tables("订单")
    Dim r As Integer
    r = .FindRow("[公司名称] Like '%湛江%'", .Position + 1, True ) '从当前行开始查找
    If r >= 0 Then '如果找到的话
        .Position = r'定位到找到的行。
    End If
End With
```

显然 Table 的 Find 和 FindRow 方法的查找结果，会受筛选的影响，因为作为 Table 的方法，只是在可见的行中进行查找； 此外因为没有排序参数，这两个方法无法按照特定的顺序进行查找，只能从上往下地顺序查找。

相对来说，DataTable 的 Find 方法则要强大很多，不仅可以用表达式查询，有排序参数，还可以指定返回第几个符合条件的行，而且可以不受筛选的影响，只要有符合条件的行，始终都能找出来。

一些复杂的查找，用 Table 的 Find 和 FindRow 方法是无法实现的，我们可以先用 DataTable 的 Find 方法，找出符合条件的 DataRow， 然后利用 Table 的 FindRow 方法 ，找出该 DataRow 在 Table 中的位置。

例如我们希望定位到客户 CS01 第二次订购 PD01 产品的行：

```
Dim dr As DataRow
Dim Index As Integer
dr = DataTables("订单").Find("客户  = 'CS01' And  产品  = 'PD01'","日期",1)
Index = Tables("订单").FindRow(dr)
If Index > 0 Then
    Tables("订单").Position = Index
End If
```

提示：Table 的 FindRow 有两个语法，一个是根据表达式查询，一个是查找指定的 DataRow 在表中的位置。

示例五

除非需要定位 到符合条件的行，否则是不需要使用 Table 的 Find 和 FindRow 方法的。

例如希望在订单表输入产品编号，能够从产品表提取该产品的名称、型号、规格，自动输入到订单表的相关列中。

为实现此目的,可在订单表的 DataColChanged 事件中输入如下代码：

```
If e.DataCol.Name ="产品编号" Then '发生变化的是产品名称吗?
    '在产品表找出该产品
    Dim dr As DataRow = DataTables("产品").Find("编号  = " & """ & e.DataRow("产品编号") & """ )
    If dr IsNot Nothing'如果找到,则设置各列内容
        e.DataRow("品名")= dr("品名")
        e.DataRow("型号")= dr("型号")
        e.DataRow("规格")= dr("规格")
    End If
EndIf
```

在上面的代码中，我们是在 DataTable 中查找所输入编号的产品，这是显然的， 首先我们只是引用数据，并不需要定位，其次如果在 Table 中查找，一旦 Table 进行了筛选，而你输入的产品编号恰好不符合筛选条件，那么就无法实现自动输入。

示例六

DataTable 有一个 Select 方法，以集合的形式，返回所有符合指定条件的行。

例如对于 1999 年 1 月 4 日订购 PD01 的订单，希望将其折扣统一设置为 0.12，代码为：

```
Dim drs As List(Of DataRow)
drs = DataTables("订单").Select("[产品] = 'PD01' And [日期]= #1/4/1999#")
For Each dr As Datarow In drs
    dr("折扣") = 0.12
Next
```

Table 也有 Select 方法，不过作用是完全不同的，该方法用于选定指定位置的单元格或区域。

例如选定当前表的第 1 行第 1 列:

```
CurrentTable.Select(0,0)
```

我们经常需要选定整行、整列、甚至整个表，可以参考下面的代码：

```
'选定第二列
With Tables("订单")
    .Select(0,1, .Rows.Count - 1,1)
End With
'选定数量列
With Tables("订单")
  Dim c As Integer= .Cols("数量").Index
    . Select(0, c, .Rows.Count - 1, c)
End With
'选定第三行
With Tables("订单")
    .Select(2,0,2, .Cols.Count - 1)
End With
'选定第三行的数量单元格
With Tables("订单")
  Dim c As Integer = .Cols("数量").Index
    .Select(2,c,2,c)
End With
```

```
'选定整个表
With Tables("订单")
    .Select(0,0, .Rows.Count -1, .Cols.Count -1)
End With
```

示例七

DataTable 侧重于数据管理，而 Table 侧重于数据展示，所以他们各自有自己独有的方法，基本上没有什么重复的。

例如 DataTable 的 DeleteFor 方法，用于按条件批量删除行，Table 就没有这样的方法。

DataTable 的方法针对的是所有行，可是有的时候，我们只是希望对可见的行进行操作，可是 Table 却没有对应的办法，需要使用 DataTable 的方法才行，那么该怎么办呢？

其实很简单，DataTable 的方法大多是可以设置条件的，我们可以将 Table 的 Filter 属性值传递给这些方法，即可让这些方法仅对可见的行生效。

例如进行筛选后，希望删除所有筛选出的行，只需：

```
CurrentTable.DataTable.DeleteFor(CurrentTable.Filter)
```

我们将 Table 的 Filter 属性传递给 DataTable 的 DeleteFor 方法，显然删除的就是我们筛选出来的行，也就是可见的行。

反过来，如果在筛选后，要删除不符合条件的行，也就是被隐藏的行，只需：

```
With CurrentTable
    .DataTable.DeleteFor("(" & .Filter &") = False")
End With
```

假定原来的筛选条件：[产品] = 'PD01'，那么上面的代码传递给 DeleteFor 的条件就是：([产品] = 'PD01') = False。

删除的自然就是条件不成立的行，也就是被隐藏的行。

再例如，希望将筛选出来的订单的折扣，替换为 0.15，代码为：

```
DataTables("订单").ReplaceFor("折扣", 0.15, Tables("订单").Filter)
```

示例八

前面提到，DataTable 和 Table 各自有自己独有的方法，基本上没有什么重复的。

不过 DataTable 和 Table 都有 Compute 方法，用于统计数据；DataTable 的 Compute 方法统计所有行，而 Table 的 Compute 方法只统计 Table 中的行，也就是经过筛选后见到的行。

所以如果没有进行筛选，那么两者的统计结果是一样的，如果进行了筛选，就需要根据具体情况选用了。

例如你在订单表中筛选出了客户 CS01 的所有行，那么：

```
Dim Sum1 As Double = Tables("订单").Compute("Sum(数量)")
Dim Sum2 As Double = DataTables("订单").Compute("Sum(数量)")
```

Sum1 将等于客户 CS01 的订购数量，而 Sum2 将等于所有客户的订购数量。

同样：

```
Dim Sum1 As Double = Tables("订单").Compute("Sum(数量)", "产品  = 'PD01'")
Dim Sum2 As Double = DataTables("订单").Compute("Sum(数量)", "产品  = 'PD01'")
```

Sum1 将等于客户 CS01 订购产品 PD01 的数量，Sum2 将等于所有客户订购产品 PD01 的数量。

示例九

很多类型，例如常用的 CrossTableBuilder 和 GroupTableBuilder ，只能对 DataTable 进行操作。

如果你希望对筛选后的数据，也就是 Table 中的数据进行统计，那么该怎么办呢？

其实一样，将这些类型的 Filter 属性的值，设为 Table 的 Filter 属性值即可。

例如：

```
Dim g As New GroupTableBuilder("统计表 1",DataTables("订单"))
g.Groups.AddDef("产品")
g.Totals.AddDef("数量")
g.Filter =Tables("订单").Filter
g.Build()
MainTable =Tables("统计表 1")
```

示例十

Foxtable 提供了四个增加行的命令，例如你要在订单表增加一行，下面四个方法都是有效的：

```
DataTables("订单").AddNew()
DataTables("订单").DataRows.AddNew()
Tables("订单").AddNew()
Tables( "订单").Rows.AddNew()
```

下面的代码是完全等效的，都是在 DataTable 中增加一行，前者不过是后者的简写:

```
DataTables("订单").AddNew()
DataTables("订单").DataRows.AddNew()
```

同样，下面的代码是也是完全等效的，都是在 Table 中增加一行，前者是后者的简写而已:

```
Tables("订单").AddNew()
Tables( "订单").Rows.AddNew()
```

所以其实只有两种增加行的方法，分别是在 DataTable 和 Table 中增加行。

那么在 DataTable 和 Table 中增加行，会有什么差别呢？差别有两个:

1、在 Table 中增加行，光标会自动移到新增加的行，使得新增行始终可以见。

2、在 Table 中增加行，而且该 Table 是一个关联子表，那么新增行会自动从父表中取得关联列的内容，并填入到新增行的关联列中。

例如订单表和订单明细表通过订单编号建立关联，如果要给当前选定订单增加一个订单明细，只能通过下面的代码：

```
Tables("订单.订单明细").AddNew()
```

示例十一

DataTable 和 Table 有很多同名的属性，例如 AllowEdit 属性，在 Table 和 DataTable 都有，那么两者如何配合使用呢？

原则是：Table 的同名属性优先。

如果 Table 没有设置过该属性的值，那么从 DataTable 继承，如果 Table 已经设置了该属性的值，那么以 Table 的设置为准。

例如在一个简单的订单管理系统中，产品表和订单表通过产品编号建立关联，客户表和订单表通过客户编号建立关联。

现在希望产品表内容只能在 Tables(″产品″)修改，也就是只能在主表修改，不能在关联表修改，可以这样设置代码：

```
DataTables("产品").AllowEdit = False
Tables("产品").AllowEdit = True
```

由于 Tables("产品")的 AllowEdit 属性被设置为 True，所以它是可以编辑的。

其它基于 DataTables("产品")的 Table，例如 Tables("订单.产品")、Tables("客户.订单.产品")，因为他们没有设置 AllowEdit 属性，所以其 AllowEdit 属性的值从 DataTables("产品")继承，而 DataTables("产品")的 AllowEdit 属性被设为 False，所以这些 Table 是不能被编辑的，也就是被锁定了。

显然，只有不同的 Table 需要不同的设置时，才需要使用 Table 的设置属性，否则直接用 DataTable 的设置属性即可。

实际上 DataTable 本身是不能编辑的，因为用户根本就看不到 DataTable，用户能编辑的只能是 Table，所以 AllowEdit 属性本应该只出现在 Table 中， 而 DataTable 之所以也提供这个属性，只是为了统一设置的方便。

例如，假定不管是作为主表还是关联表，都要禁止编辑订单表，只需：

```
DataTables("订单").AllowEdit = False
```

如果 DataTable 没有 AllowEdit 属性，那么就必须逐个 Table 设置，代码显得繁琐很多：

```
Tables("订单").AllowEdit = False
Tables("产品.订单").AllowEdit = False
Tables("客户.订单").AllowEdit = False
```

我们这里只是以 AllowEdit 属性为例说明，所有 DataTable 和 Table 同时具备的属性，都有和 AllowEdit 属性一样的作用原理。

示例十二

前面已经提到，DataTable 和 Table 有各自不同的作用，所以他们基本上不会有重复的属性和方法，即使有同名的属性或方法，其作用也是有所区别，例如上面介绍的 AddNew 方法和 AllowEdit 属性。

不过有一个例外，DataRow 和 Row 有七个同名的方法和属性：Save、Reject、Load、Delete、Remove、Locked、IsNull， 它们的作用是完全相同的；调用 Row 上述成员时，Foxtable 实际上调用的是 DataRow 的同名成员。

也就是说，下面的代码：

```
If Tables("表 A").Current.IsNull("第一列") = False Then
    Tables("表 A").Current.Locked = True
End If
Tables("表 B").Rows(0).Delete()
```

在执行之前，会自动转换为下面的代码：

```
If Tables("表 A").Current.DataRow.IsNull("第一列") = False Then
    Tables("表 A").Current.DataRow.Locked = True
End If
Tables("表 B").Rows(0).DataRow.Delete()
```

你也许会问，为什么单单这七个成员被特殊处理，为什么不考虑通过 Table、Col、Row 直接调用 DataTable、DataCol、DataRow 的全部成员？

我也不能解释，也许 Foxtable 的策划人员感觉这七个成员使用的频率特别高，所以给了特别待遇，让我们编写代码的时候，能够更简洁一点。

4.4.6.2 再谈_Identify

我们知道内部表有一个_Identify，这是一个整数型列，每增加一行，该列的值就会自动加 1。

该列的值是只读的，我们没有办法使用常规的代码去修改_Identify 值。

可以将_Identify 值看作是行的身份证号码，因为新增行一旦保存，其_Identify 值将终生不变。

这个“身份证号码”就是主键，系统就是根据这个值来识别不同的行的。

实际使用的时候，我们是看不到_Identify 列的，但是可以在代码中使用该列，例如：

```
Dim id1 As Integer = Tables("订单").Current("_Identify")
Dim id2 As Integer = Tables("订单").Rows(0)("_Identify")
Dim id3 As Integer = DataTables("订单").DataRows(1)("_Identify")
```

也可以在表达式中使用该列，例如可以增加一个表达式列，将其表达式设为：

```
[_Identify]
```

这样即可通过此列查看_Identify 的值。

可以看出_Identify 列除了不可见，使用起来和普通的列并无差别。

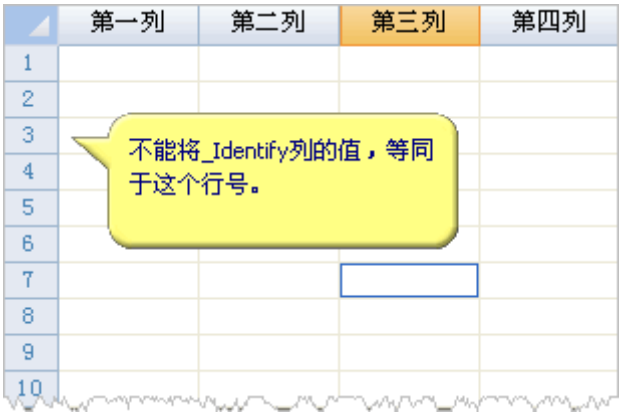
_Identify 值类似行的编号，这个编号表示数据录入的顺序，也就是某个 DataRow 在 DataTable 中的顺序。

下面是一个根据输入顺序进行流水账计算的例子，设置在 DataColChanged 事件中即可：

```
Dim Val1 As Double
```

```
Dim Val2 As Double
Select Case e.DataCol.Name
Case "收入","支出"
    For Each dr As DataRow In e.DataTable.Select("[_Identify] >= " & e.DataRow("_Identify") )
        Val1 = e.DataTable.compute("Sum(收入)","[_Identify] <= " & dr("_Identify"))
        Val2 = e.DataTable.compute("Sum(支出)","[_Identify] <= " & dr("_Identify"))
        dr("余额") = Val1 - Val2
    Next
End Select
```

有一个问题需要明确，很多初学的用户将_Identify 等同于下图的行号：



其实这两者没有任何关系；显然，同一行数据，随着筛选和排序，其行号是不断地变化的，而_Identify 列的值是终生不变的。

我们可以这样理解：

1、_Identify 表示增加行的顺序，也就是 DataRow 在 DataTable 中的顺序，同时也是行的“身份证号码”，终生不变。

2、而上图中的行号，表示 Row 在 Table 中的顺序，Row 有一个 Index 属性，返回的就是这个行号，这个行号是动态的，因为同一行数据，在排序和筛选后，其在 Table 中的位置是会变化的。

4.4.6.3 再谈空值

我们已经知道，空值是用 Nothing 表示的。 例如需要删除订单表某行客户列的内容，可以：

```
Tables("订单").Rows(1)("客户") = Nothing
```

或

```
DataTables("订单").DataRows(1)("客户") = Nothing
```

如果要判断某一列的内容为空，可以：

```
If Tables("订单").Rows(1).IsNull("客户") Then
  '代码
End If
```

或

```
If DataTables("订单").DataRows(1).IsNull("客户") Then
  '代码
End If
```

你也许会感到奇怪，为什么要用 IsNull 判断，而不能用下面的方法判断呢：

```
If Tables("订单").Rows(1)("客户") Is Nothing Then
  '代码
End If
```

这是因为即使某列的值为空，其返回的值也不会是 Nothing。

在某列内容为空的时候，会根据不同的列类型，返回不同的值，规则如下：

字符列：""

数值列：0

逻辑列：False

日期列：#0001-01-01 0:00:00#

很多方法，例如 Compute、Find、DeleteFor、Select，都可以使用表达式来作为执行条件，那么表达式中如何针对空值和非空值来设置条件呢？

我们还是举例说明，例如删除表 A 中第一列内容为空的行：

```
DataTables("表 A").DeleteFor("第一列  Is Null")
```

再例如，计算订购数量大于 500，且发货日期为空的行数：

```
Dim Count As Integer
Count = DataTables("订单").Compute("Count(编号)","数量> 500 And  发货日期  Is Null")
Output.Show(Count)
```

如果要表示非空，可以加上 Not 关键词，例如要得到订单表中折扣不为空的行数：

```
Dim Val As Integer
Val = DataTables("订单").Compute("Count(产品编号)","折扣  Is Not Null")
```

4.4.6.4 空值与比较

假定有下面这么一个表，

	第一列	第二列	第三列	第四列
1	a			
2	b			
3	a			
4	b			
5				
6				

如果希望筛选出不等于 a 的行，通常我们会用如下的代码：

```
Tables("表 A").Filter = "第一列  <> 'a'"
```

因为 b 和空值都不等于 a，所以按道理能得到期待的正确结果：

	第一列	第二列	第三列	第四列
1	b			
2	b			
3				
4				

可实际上我们得到的结果会是：

	第一列	第二列	第三列	第四列
1	b			
2	b			

这是因为空值是不参与比较的，不管条件是等于某个值还是不等于某个值，对于空值来说，条件

都是不成立的 ，空值只能用 Is Null 或者 Is Not Null 进行判断。
所以对于上图中的表，要筛选出第一列不等于 a 的行，正确的代码应该是：

```
Tables("表 A").Filter = "第一列 <> 'a' Or 第一列 Is Null"
```

意思说是筛选出等一列不等于 a 或者第一列为空的行。

4.4.6.5 保存设置信息

一个完整的系统，通常会提供一些自定义的设置给用户，以便满足不同的用户需要。
Foxtable 提供了一些函数，分别用于保存、读取和维护设置信息。

SaveConfigValue

保存设置信息
语法：
SaveConfigValue(Name, Value)
Name： 设置名称
Value： 设置值
设置信息经过加密储存，安全性比一般的方式更有保证。

GetConfigValue

读取设置信息。
语法：
GetConfigValue(Name, DefaultValue)
Name： 设置名称
DefaultValue：默认设置值，如果指定名称的设置不存在，则返回此值。

RemoveConfigItem

删除指定名称的设置。
语法：
RemoveConfigItem(Name)
Name: 要删除的设置名称。

ClearConfigItem

删除所有的设置，无参数。

示例

假定你提供给客户的是一个试用版本，希望最多使用 30 次，可以在项目事件 AfterOpenProject 中加入如下代码：

```
Dim n As Integer
n = GetConfigValue("Count",1) '假定保存试用次数的设置名称为 Count
If n > 30 Then
    Messagebox.Show("您正在使用的产品已经超出试用次数!")
    Syscmd.Project.Exit()
Else
    n = n + 1
    SaveConfigValue("Count",n)
End If
```

提示：如果启动项目而不保存，是不会计算试用次数的。

辅助输入

为方便大家准确输入设置名称，在代码编辑器中提供了一个辅助功能：



4.4.6.6 读写注册表

设置信息也可以保存在注册表中，Registry 类专门用于读写注册表。

Registry 的方法有：

方法	说明
SetValue	向注册表写入值，语法： SetValue(keyName ,valueName ,value) keyName： 要写入的项的名称 valueName：要写入的值的名称 value： 要写入的值
GetValue	从注册表中读取值，语法： GetValue(keyName ,valueName ,DefaultValue) keyName: 要读取的项的名称 ValueName：要读取的值的名称 defaultValue：默认值，如果不存在指定名称的项或值，则返回此默认值。

示例

```
Dim Count As Integer
Count = Registry.GetValue("HKEY_CURRENT_USER\Software\MyApp","Count",0)
Registry.SetValue("HKEY_CURRENT_USER\Software\MyApp","Count",Count + 1)
Output.Show(Count)
```

在命令窗口反复执行上面的代码，可以发现每执行一次，Count 的值就会增加 1。

4.4.6.7 突破表关联限制

本节的内容，可以参考 CaseStudy 目录下的文件：同表同列关联.Table

我们知道，Foxtable 的表关联有两大限制：

1、表达式列不能用于建立关联。

2、建立同表关联的时候，父表和子表的关联列必须不同。

不过我们可以用代码突破这两个限制。

假定有下图所示的表：

	班级	姓名	语文				数学				英语			
			分数	最低	最高	平均	分数	最低	最高	平均	分数	最低	最高	平均
1	1	陈锦标	132	71	135	117.45	147	38	147	105.26	145	44	146	115.6
2	2	伍景珠	134	63	137	118.66	140	48	150	111.43	142	33	142	107.52
3	1	邓玉婷	132	71	135	117.45	136	38	147	105.26	146	44	146	115.6
4	2	李晓燕	137	63	137	118.66	140	48	150	111.43	136	33	142	107.52
5	1	何列琦	120	71	135	117.45	140	38	147	105.26	139	44	146	115.6
6	2	林璐韵	132	63					150	111.43	110	33	142	107.52
7	1	冯 勃	135	71					147	105.26	129	44	146	115.6
8	2	蒋广裕	131	63					150	111.43	122	33	142	107.52
9	2	黄 图	116	63					150	111.43	141	33	142	107.52
10	1	叶卜源	135	71	135	117.45	125	38	147	105.26	144	44	146	115.6
11	1	何思敏	129	71	135	117.45	143	38	147	105.26	134	44	146	115.6

除了各科目的分数，还同时显示同班级同科目的最低分、最高分、和平均分。

要实现这样的统计很简单：

1、新建一个表达式列，列名为“班级 A”，表达式为：[班级]

2、将"班级 A"这一列隐藏起来，因为这一列仅仅用于建立关联。

3、新建好各科目的最低、最高、平均分列，这些列都是表达式列，但是新建的时候，不需要设置表达式：

列名	类型	长度	标题	表达式
班级	整数			
班级A	整数			[班级]
姓名	字符型	12		
语文_分数	双精度小数			
语文_最低	双精度小数			
语文_最高	双精度小数			
语文_平均	双精度小数			
数学_分数	双精度小数			
数学_最低	双精度小数			
数学_最高	双精度小数			
数学_平均	双精度小数			
英语_分数	双精度小数			
英语_最低	双精度小数			
英语_最高	双精度小数			
英语_平均	双精度小数			

4、最后在项目事件 AfterOpenProject 中加上代码，用于创建关联并设置表达式：

```
Dim dt As DataTable = DataTables("成绩表")
If Relations.Contains("cj") = False Then '如果还没有创建关联
    Relations.Add("cj",dt.DataCols("班级"),dt.DataCols("班级A"),RelationPathEnum.None)
End If
dt.DataCols("语文_最低").Expression = "Min(Child(cj).语文_分数)"
dt.DataCols("语文_最高").Expression = "Max(Child(cj).语文_分数)"
dt.DataCols("语文_平均").Expression = "Avg(Child(cj).语文_分数)"
dt.DataCols("数学_最低").Expression = "Min(Child(cj).数学_分数)"
dt.DataCols("数学_最高").Expression = "Max(Child(cj).数学_分数)"
dt.DataCols("数学_平均").Expression = "Avg(Child(cj).数学_分数)"
```

```
dt.DataCols("英语_最低").Expression = "Min(Child(cj).英语_分数)"
dt.DataCols("英语_最高").Expression = "Max(Child(cj).英语_分数)"
dt.DataCols("英语_平均").Expression = "Avg(Child(cj).英语_分数)"
```

4.4.6.8 Select 方法的灵活使用

普通用户可以忽略本节内容。

DataTable 的 Select 是一个非常重要的方法，灵活使用，可以轻松解决很多看似复杂的难题。

示例一

下面的示例可以参考 CaseStudy 目录下的文件： 成绩排名.Table

例如在一个成绩表中，包括班级、姓名、总分、总分排名几列数据，希望按班级自动生成总分排名：

```
'获得所有班级名称,保存在集合中
Dim bjs As List(Of String) = DataTables("成绩表").GetValues("班级")
For Each bj As String In bjs
    '获得该班级的全部行,按总分降序排序
    Dim drs As List(Of DataRow) = DataTables("成绩表").Select("[班级] = " & bj, "总分  DESC")
    For n As integer = 0 To drs.Count - 1 '遍历所有行
        If n > 0 AndAlso drs(n)("总分") = drs(n-1)("总分") Then '如果总分和上一行相同
            drs(n)("总分排名") = drs(n-1)("总分排名") '则排名等于上一行
        Else
            drs(n)("总分排名") = n + 1 '设置排名
        End If
    Next
Next
```

示例二

下面的示例可以参考 CaseStudy 目录下的文件： 等级划分.Table

假定有一个表，有分数和等级两列，要根据分数从高到低进行等级划分，其中 A 等占 10%，B 等占 25%，C 等占 40%，D 等占 20%，E 等占 5%。

代码为：

```
Dim vals() As Double = {0.1,0.35,0.75,0.95}
Dim cnt As Integer = DataTables("表A").DataRows.Count '得到总行数
Dim drs As List (of DataRow) = DataTables("表A").Select("", "分数  Desc") '得到按分数高低排列的行的集合.
For i As Integer = 0 To drs.Count - 1
    Dim v As Double   = i /cnt   '计算  位置比例
    If v > vals(3) Then
```

```
        drs(i)("等级") = "E"
    ElseIf v > vals(2) Then
        drs(i)("等级") = "D"
    ElseIf v > vals(1) Then
        drs(i)("等级") = "C"
    ElseIf v > vals(0) Then
        drs(i)("等级") = "B"
    Else
        drs(i)("等级") = "A"
    End If
Next
```

4.4.6.9 高效随机抽取记录

随机抽取少量记录

如果要从数据表中随机抽取一定数量的记录，例如从员工表随机抽取 5 个员工，可以参考下面的代码：

```
Dim ids As String
Dim lst As new List(of String)
Dim cnt As Integer = DataTables("员工").DataRows.Count
Do
    Dim Id As Integer = rand.Next(0,cnt)
    id = DataTables("员工").DataRows(id)("_Identify")
    If lst.Contains(id) =False Then
        lst.Add(id)
    End If
    ids = ids & id & ","
Loop While lst.count < 5    '5是要抽取的行数
Tables("员工").Filter = "[_Identify] In (" & ids.Trim(",") & ")"
```

上面的代码适合随机抽取少量的记录。

随机抽取大量记录

如果随机抽取的记录较多，可以考虑增加一个逻辑列，假定逻辑列的名称为“选择”，可以参考下面的代码：

```
Dim cnt As Integer = DataTables("订单").DataRows.Count
Dim nds As Integer = 100 '要抽取的记录数
Tables("订单").StopRedraw()
DataTables("订单").ReplaceFor("选择",False)
Do
```

```
    Dim idx As Integer = rand.Next(0,cnt)
    Dim dr As DataRow = DataTables("订单").DataRows(idx)
    If dr("选择") = False Then
        dr("选择") = True
        nds = nds - 1
    End If
Loop While nds > 0
Tables("订单").Filter = "[选择] = True"
Tables("订单").ResumeRedraw()
```

上面的代码从订单表中随机抽取100个订单。
建议逻辑列“选择”采用表达式列，不要给其设置表达式即可，因为：没有设置表达式的表达式列，可以通过代码设置此列的值。

用洗牌法抽取记录

上面的代码在抽取的记录数接近于总的记录数时，性能会下降，虽然可以用逆排除的方法提高性能，但是毕竟要多写一半的代码。
我们可以换个思路，用类似洗牌的方法，将记录顺序打乱后再来抽取，性能会相对比较稳定：

```
Dim cnt As Integer = DataTables("订单").DataRows.Count
Dim ids1 As New List(of Integer) '用于存储洗牌前的位置
Dim ids2 As New List(of Integer) '用于存储洗牌后的位置
For i As Integer = 0 To cnt -1 '准备初始的牌
    ids1.add(i)
Next
For i As Integer = 0 To cnt - 1 '开始洗牌
    Dim idx As Integer = ids1(rand.Next(0,ids1.count))
    ids2.Add(idx)
    ids1.Remove(idx)
Next
Tables("订单").StopRedraw()
DataTables("订单").ReplaceFor("选择",False)
For i As Integer = 0 To 5 - 1 '100为要抽取的行数
    DataTables("订单").DataRows(ids2(i))("选择") = True
Next
Tables("订单").Filter = "[选择] = True"
Tables("订单").ResumeRedraw()
```

4.4.6.10 行的筛选与定位

Foxtable 筛选前后定位行的默认方式为尽量保持物理位置不变，例如：

假定筛选前，用户选择的是第 10 行，那么筛选后，Foxtable 会自动定位到第 10 行，如果符合条件的行不足 10 行，那么就会定位到最后 1 行。
假定撤销筛选前，用户选择的是第 10 行，那么撤销筛选后，Foxtable 同样会自动定位到第 10 行。当然此第 10 行，并非彼第 10 行。
有的用户并不喜欢默认处理方式，而是希望：
1、筛选后，如果之前选择的行符合筛选条件，那么筛选后依然选择此行。
2、撤销筛选后，自定定位到之前选择的行。
以订单表为例，要实现此功能非常简单：
1、BeforeFilter 事件代码：

```
If Tables("订单").Position >= 0 Then
    Vars("ddr") = Tables("订单").current.DataRow
End If
```

2、AfterFilter事件代码：

```
If   Vars("ddr") IsNot Nothing Then
     Tables("订单").Position = Tables("订单").FindRow(Vars("ddr"))
End If
```

参考：

Var变量　BeforeFilter　AfterFilter

4.4.6.11 可以编辑的表达式列

我们知道表达式列是不能编辑的，也不能通过代码设置值。

但是没有设置表达式的表达式列，是可以编辑的，也可以通过代码设置值。
这个特性可以给我们的开发带来一些便利。
一个例子
例如我们知道，Foxtable 可以在行号列显示一个复选框，用于勾选行：

http://www.foxtable.com/webhelp/scr/1776.htm
但这种方式也有不足，首先占用了行号列，其次这毕竟不是真正的列，在日常操作(例如筛选)和编码处理上会有一些不足。
我们可以用代码增加一个逻辑型表达式列，但是不设置其表达式，利用这一列来实现勾选行的功能，会方便很多。
AfterOpenProject 事件代码：

```
DataTables("订单").DataCols.Add("选",Gettype(Boolean))
Tables("订单").Cols("选").Move(0)
Tables("订单").Cols("选").Width = 25
```

这样打开项目后，订单表的左边就会出现一个逻辑列，用于勾选行：

	选	产品	客户	雇员	单价	折扣	数量	金额
1	☐	PD05	CS03	EP04	17	0.1	650	9945
2	☐	PD02	CS01	EP01	20	0.1	400	7200
3	☐	PD05	CS04	EP02	17	0	320	5440
4	☐	PD01	CS03	EP04	18	0.15	80	1224
5	☐	PD03	CS03	EP04	10	0	490	4900
6	☐	PD04	CS04	EP05	17.6	0	240	4224
7	☐	PD05	CS03	EP04	17	0	111	1887
8	☐	PD01	CS04	EP05	14.4	0	200	2880
9	☐	PD03	CS02	EP02	8	0.1	200	1440
10	☐	PD01	CS02	EP01	18	0.2	800	11520

4.4.6.12 日期列的一个意外

大多数时候，我们可以直接使用日期列的值。

例如假定当前表有一个日期列，可以在命令窗口正常执行下面的代码：

```
Dim m As Integer = CurrentTable.Current("日期").Month
Dim d As Date = CurrentTable.Current("日期").AddYears(360)
```

但是如果你在命令窗口执行下面的代码，却会出现错误提示：

```
Dim t As TimeSpan
t = Date.Today - CurrentTable.Current("日期")
OutPut.Show("距今天数:" & t.TotalDays)
```

要使上面的代码正确执行，可以修改为：

```
Dim t As TimeSpan
t = Date.Today - CDate(CurrentTable.Current("日期"))
OutPut.Show("距今天数:" & t.TotalDays)
```

上面的代码用 CDate 将列中的值转换为日期，你也许会奇怪，本来就是日期，何来转换之说，这是因为我们从某列取值的时候，不管列类型如何，得到的始终是一个 Object 类型的值，但是 Visual Basic 会自动判断真实的值类型，所以我们直接使用即可。唯一的例外就是上面这个例子：日期之间相减的时候，用 CDate 函数明确地转换一下。

除了用 CDate 转换外，还可以：

```
Dim t As TimeSpan
Dim d As Date = CurrentTable.Current("日期")
```

```
t = Date.Today - d
OutPut.Show("距今天数:" & t.TotalDays)
```

4.4.6.13 关于生日提醒

假定有个员工表，有姓名和出生日期列，希望能够筛选出今天生日的行，代码为：

```
Dim nms As String
For Each dr As DataRow In DataTables("员工").DataRows
    Dim dt As Date = dr("出生日期")
    If dt.Month = Date.Today.Month AndAlso dt.Day = Date.Today.Day Then
        nms = nms & ",'" & dr("姓名") & "'"
    End If
Next
If nms > "" Then
    nms = nms.Trim(",")
    Tables("员工").filter = "[姓名] In (" & nms &")"
End If
```

如果希望筛选出未来七天内生日的员工，代码为：

```
Dim nms As String
Dim n As Integer = 6 '未来天数,含今天,所以实际是未来 7 天
For Each dr As DataRow In DataTables("员工").DataRows
    Dim dt As Date = dr("出生日期")
    Dim dt1 As New Date(Date.Today.Year,dt.Month,dt.Day)
    Dim dt2 As  New Date(Date.Today.AddDays(n).Year,dt.Month,dt.Day)
    If (dt1 >= Date.Today AndAlso dt1 < Date.Today.AddDays(n)) OrElse (dt2 >= Date.Today AndAlso dt2 < Date.Today.AddDays(n)) Then
        nms = nms & ",'" & dr("姓名") & "'"
    End If
Next
If nms > "" Then
    nms = nms.Trim(",")
    Tables("员工").filter = "[姓名] In (" & nms &")"
End If
```

同名问题

上面的代码有漏洞，假定有两个张三，其中一个张三今天生日，那么执行上述代码之后，两个张三都会显示出来，尽管另一个张三并不是今天生日。

要解决这个问题，可以参考下面的代码，同样以筛选出今天生日的员工为例：

```
Dim ids As String
Dim n As Integer = 6 '未来天数,含今天,所以实际是未来 7 天
For Each dr As DataRow In DataTables("员工").DataRows
    Dim dt As Date = dr("出生日期")
```

```
    Dim dt1 As New Date(Date.Today.Year,dt.Month,dt.Day)
    Dim dt2 As   New Date(Date.Today.AddDays(n).Year,dt.Month,dt.Day)
    If (dt1 >= Date.Today AndAlso dt1 < Date.Today.AddDays(n)) OrElse (dt2 >= Date.Today AndAlso dt2 < Date.Today.AddDays(n)) Then
        ids = ids & "," & dr("_Identify")
    End If
Next
If ids > "" Then
    ids = ids.Trim(",")
    Tables("员工").filter = "[_Identify] In (" & ids &")"
End If
```

由于每一行的_Identify 列的内容都是不同的，同名带来的问题不复存在。
在表达式中，因为姓名列是字符型，所以值要用单引号括起来，例如：[姓名] In ('张三', '李四', '王五')；而_Identify 是整数型，值不需要用单引号括起来，例如：[_Identify] In (1, 3, 7)
如果希望打开项目后，能自动筛选出当天生日的员工，可以将上面的代码设置在 AfterOpenProject 事件中。

4.4.6.14 系统样式设置

DataTable 有个 SysStyles 集合，用于返回指定名称的系统样式，例如：
DataTables("成绩表").SysStyles("Alternate")
表示成绩表交替行的样式。
系统样式包括：
Normal：数据区
Frozen：冻结区
Alternate:交替行
EmptyArea：空白区
Fixed：标题行
Highlight:选定区域
Focus：焦点单元格
ModifiedCell:已修改单元格
SelectedRowHeader：选定行的行标题
SelectedColumnHeader：选定列的列标题
CurrentRow:当前行
GrandTotal：总计行
Subtotal0：小计行 0
Subtotal1：小计行 1
Subtotal2：小计行 2
Subtotal3：小计行 3
Subtotal4：小计行 4
Subtotal5：小计行 5

样式包括以下属性：

BackColor： 背景颜色
ForeColor： 字体颜色
FontBold： 逻辑型，设为 True，字体加粗
FontItalic： 逻辑型，设为 True，字体倾斜
FontStrikeout：逻辑型，设为 True，字体带删除线
FontUnderline：逻辑型，设为 True，字体带下划线

示例

希望将空白区的背景颜色设为白色：

```
DataTables("表 A").SysStyles("EmptyArea").BackColor = Color.White
```

4.4.6.15 变量的作用范围

在代码的任何位置，我们都可以用 Dim 语句定义变量，但是不同位置定义的变量，其作用范围是不同的。

变量的作用范围原理如下：如果在流程语句之内定义的，则只在该流程语句之内以及嵌套的子流程语句内有效，如果在流程语句之外定义的，则在该段代码的任何位置有效。

下面我们通过一些例子来说明上述原理，这些代码本身没有意义，甚至显得非常冗余，但是可以很好地说明问题：

示例一

下面的代码能够正常执行：

```
Dim m As Integer 'm 在流程语句之外,在任何位置可以使用
If m > 0 Then
    Dim n As integer 'n 在流程语句之内定义,只能在本流程内使用
    n = 1 '使用 n
    m = n + 1 '使用 m 和 n
End If
m = m * 2 '使用 m
```

示例二

下面的会报错，问题出在最后一行代码，因为变量 n 是在 If 语句内定义的，不能在 If 语句之外使用：

```
Dim m As Integer 'm 在流程语句之外,在任何位置可以使用
If m > 0 Then
    Dim n As integer 'n 在流程语句之内定义,只能在本流程内使用
    n = 1 '使用 n
    m = n + 1 '使用 m 和 n
End If
m = m * 2 '使用 m
n = n * 2 '这会导致代码出错，因为变量 n 不能在这里使用
```

下面的代码也会报错，问题同样出在最后一行：

```
Dim m As Integer = 100
If m > 100 Then
    Dim n As Integer
    n = m * 2
Else
```

```
    Dim n As Integer = 20
    n = n + m
End If
n = n + 1 '这里会出错,因为变量 n 不能在这里使用。
```

如果要在 If 语句之外使用变量，必须在 If 语句之前定义好：

```
Dim m As Integer = 100
Dim n As Integer
If m > 100 Then
    n = m * 2
Else
    n = 20
    n = n + m
End If
n = n + 1
m = n + 100
```

上面的代码，由于变量 n 和 m 是在 if 语句之前定义的，所以在 if 语句之内和之外，都能使用这两个变量。

示例三

下面的语句在执行过程中会报错：

```
Dim Sum As Integer
For i As Integer = 0 To 100
    Sum = Sum + i
Next
Output.Show(Sum / i) '这里会出错,变量 i 不能在这里出现
```

这是因为变量 i 是在语句 For 中定义的，所以只能在循环语句(For...Next)中使用，最后一行代码位于循环语句之外，却使用了变量 i，所以会报错，错误提示为“变量 i 未申明”。

如果某个变量在循环语句之内和之外都要使用，那么应该在循环语句之前定义好变量，例如：

```
Dim Sum As Integer
Dim i As Integer
For i = 0 To 100
    Sum = Sum + i
Next
Output.Show(Sum / i)
```

上面的代码中，变量 Sum 和 i 是在循环语句之前定义的，所以在循环语句之内和之外都可以使用。

示例四

不同的流程语句内，可以定义同名的变量，例如在下面的代码中，我们在四个地方定义了名称为 n 的变量：

```
Dim m As Integer = 5
If m > 0 Then
    Dim n As Integer
    n = 1
Else
    Dim n As Integer
    n = 10
End If
```

```
Select Case m
    Case 1,3,5,7,8
        Dim n As Integer = 1
        m = n
    Case 2,4,6,8,10
        Dim n As Integer = 1
        m = 2
End Select
Messagebox.Show(IIF(m = 1,"奇数","偶数"))
```

实际使用的时候，如果某一个变量需要在不同的流程中使用，应该直接在流程外定义，如果某个变量只在流程内使用，那么就应该在该流程内定义。

示例五

在流程语句之内定义的变量，可以在子流程内使用，例如下面的代码中，变量 n 是在 If 语句中定义的，在嵌套的子流程 Select Case 语句中也使用了该变量：

```
Dim m As Integer = 1
If m > 0 Then
    Dim n As Integer
    n = 1
    Select Case m
        Case 1,3,5,7,8
            n = 1
        Case 2,4,6,8,10
            n = 2
    End Select
    Messagebox.Show(IIF(n =1,"奇数","偶数"))
End If
```

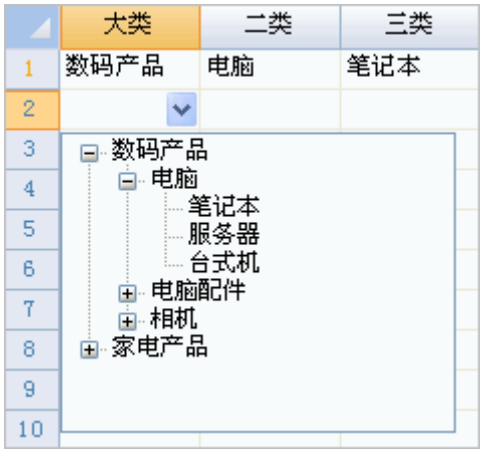
示例六

流程内和流程外，不能有同名的变量，例如下面的代码会出现错误，这是因为变量 n 同时在流程内和流程外定义了：

```
Dim m As integer
Dim n As Integer
If n > 0 Then
    Dim n As integer '这里会出错
    m = 1
End If
```

4.4.6.16 用代码生成下拉目录树

用于给列生成下拉目录树，下拉目录树能够非常高效地输入数据：



DropTreeBuilder 成员介绍：

属性

TreeFile：　字符型，指定目录树文件

TreeString:　字符型，指定用于生成目录树的字符串

SourceTable：　DataTable 类型，指定目录树表

TreeFilter:　字符型，用于设置从 SourceTable 取数据生成目录树的过滤条件。

TreeSort:　字符型，用于设置生成目录树的排序列。

ReceiveCols：　字符型，指定数据接收列

TreeCols：　字符型，指定目录树列

SourceCols：　字符型，指定数据来源列

PathSeparator：　字符型，路径分割符号

ContentSeparator：字符型，指定内容分割符号

SeparateContent：　逻辑型，是否分割内容

TreeWidth:　整数型，指定目录树宽度

TreeHeight：　整数型，指定目录树高度

方法

Build：　生成下拉目录树。

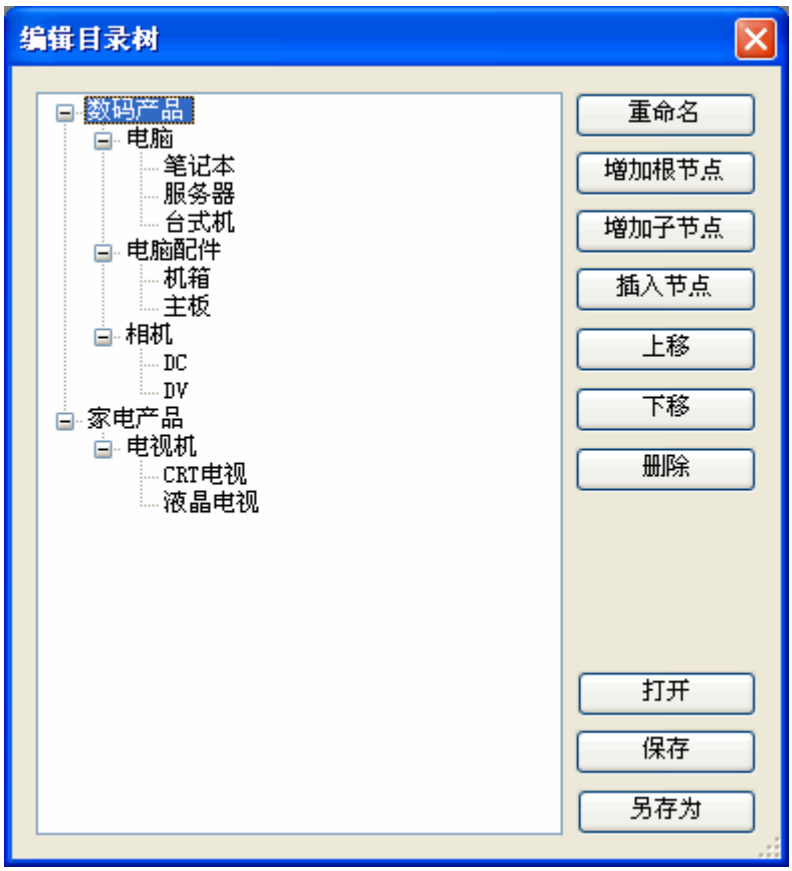
创建目录树文件

在使用下拉目录树之前，通常需要先创建一个目录树文件。

在菜单的“杂项”功能区，单击下面的按钮：



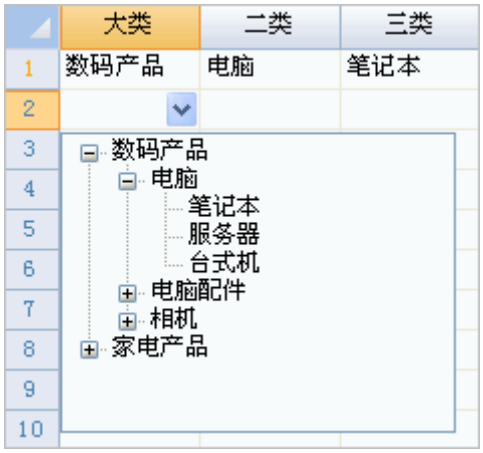
可以打开目录树编辑窗口，可以在这里创建目录树文件，也可以打开一个现有的目录树文件修改：



示例一

创建了目录树文件后，就可以使用目录树录入数据了。

假定有一个表，希望使用下图所示的目录树输入数据：



可以在项目事件 AfterOpenProject 事件中加入下面的代码：

```
Dim tb As New DropTreeBuilder
tb.TreeFile = "product.foxtr" '指定目录树文件
tb.ReceiveCols = "大类|二类|三类" '指定数据接收列
Tables("订单").Cols("大类").DropTree = tb.Build()
```

提示：目录树文件最好放在项目的 Attachments 文件夹中，否则必须指定明确的路径。

示例二

在网络环境下使用文件型目录树不太方便，更方便的是将目录树存储在数据表中，用于存储目录

树的列最好是备注型

虽然目录树文件的后缀名为 foxtr，但其实是一个纯文本文件，可以用记事本打开，将其内容复制粘贴到数据表中的用于存储目录树的备注列中。

```
Dim s As String
'请在这里设置代码从数据表中获得目录树字符串,并赋值给变量 s
Dim tb As New DropTreeBuilder
tb.TreeString = s '设置目录树字符串
tb.ReceiveCols = "大类|二类|三类" '指定数据接收列
Tables("订单").Cols("大类").DropTree = tb.Build()
```

提示：在窗口设计的章节，我们还会介绍如何自己编写一个目录树编辑器，这样可以将目录树编辑和自己的系统更为紧密地结合起来。

示例三

有的时候，目录树一个节点的内容，可能包括多项内容。

例如对于下面的目录树，二级节点就包括县市、区号、邮编三项内容，三项内容用符号“|”隔开了：

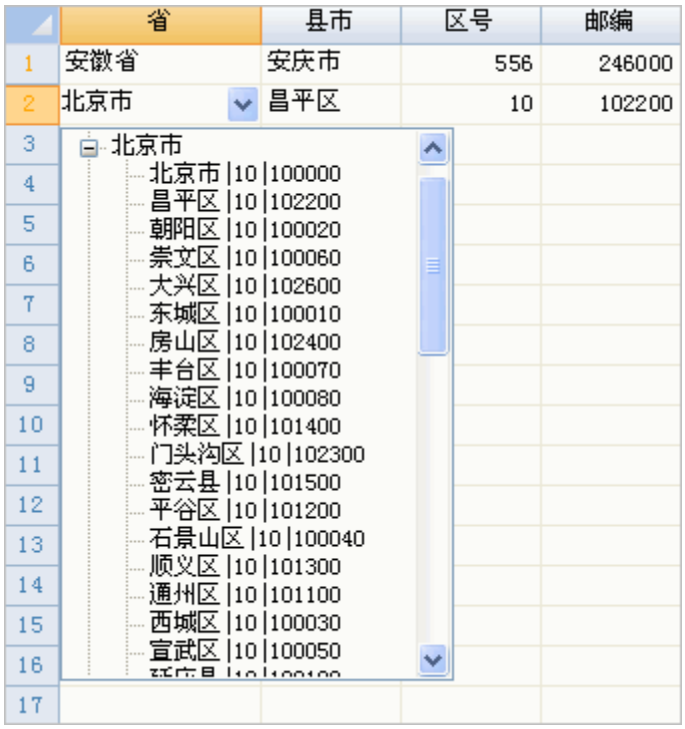


在使用目录数列表的时候，为了将二级节点的三项内容对应地填入县市、区号、邮编三列中，可以如下设置代码：

```
Dim tb As New DropTreeBuilder
tb.TreeFile = "Region.foxtr" '指定目录树文件
tb.ReceiveCols = "省|县市|区号|邮编" '指定数据接收列
tb.SeparateContent = True '启用内容分割功能
tb.ContentSeparator = "|" '指定内容分割符号
```

```
Tables("客户").Cols("省").DropTree = tb.Build()
```

这样当我们选择某个节点的时候，二级节点的内容能够对应地输入到县市、区号、邮编三列中：



示例四

除了根据文件，还可以根据数据表生成目录树。

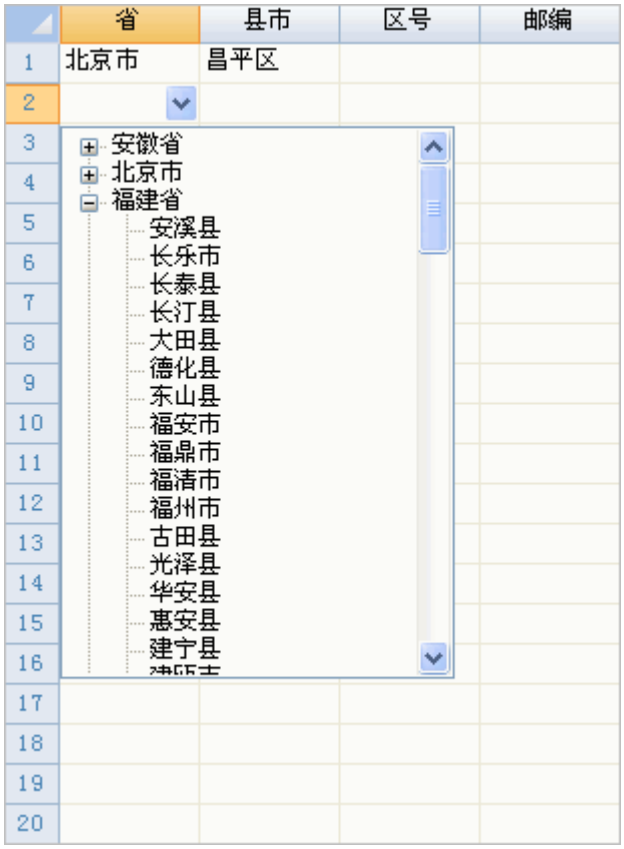
例如有一个行政区域表，结构如下：

	省	县市	区号	邮编
1	北京市	北京市	10	100000
2	北京市	东城区	10	100010
3	北京市	西城区	10	100030
4	北京市	崇文区	10	100060
5	北京市	宣武区	10	100050
6	北京市	朝阳区	10	100020

我们希望在输入数据的时候，能够根据该数据表自动生成目录树，用于输入省和县市列的内容，可以如下设置代码：

```
Dim tb As New DropTreeBuilder
tb.SourceTable = DataTables("行政区域") '指定目录树表
tb.TreeCols = "省|县市" '指定用于生成目录树的列
tb.ReceiveCols = "省|县市" '指定数据接收列
Tables("客户").Cols("省").DropTree = tb.Build()
```

即可得到下图所示的下拉目录树：



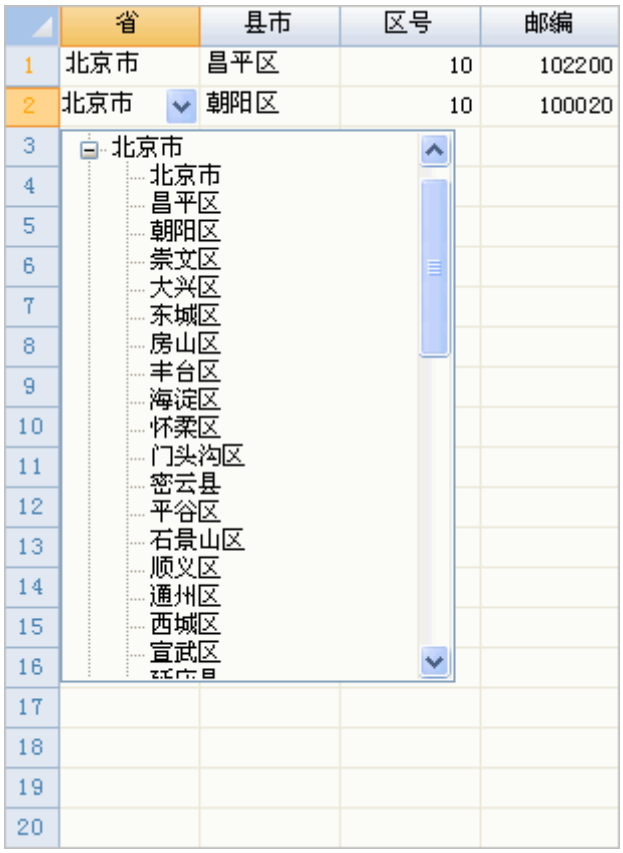
示例五

上面的下拉目录树只能输入省、县市两列的内容，而区号和邮编由于没有参与目录树的生成，所以无法输入到数据表中。

不过我们可以通过明确指定数据来源列，来输入区号和邮编两列的数据：

```
Dim tb As New DropTreeBuilder
tb.SourceTable = DataTables("行政区域") '指定目录树表
tb.TreeCols = "省|县市" '指定用于生成目录树的列
tb.SourceCols = "省|县市|区号|邮编" '指定数据来源列
tb.ReceiveCols = "省|县市|区号|邮编" '指定数据接收列
Tables("客户").Cols("省").DropTree = tb.Build()
```

通过上面的设置后，可以正常输入区号和邮编，尽管这两列并没有用于生成目录树：



示例六

有的时候，用于生成目录树的表，其中一列的内容包括多个栏目的内容。

例如下面的县市列，就包括县市、区号、邮编三个栏目的内容：



为了在目标表中，能够将上图中县市列的内容拆分成 3 个部分，对应地输入到县市、区号、邮编三列中，可以如下设置代码：

```
Dim tb As New DropTreeBuilder
tb.SourceTable = DataTables("行政区域 2") '指定目录树表
tb.TreeCols = "省|县市" '指定用于生成目录树的列
tb.ReceiveCols = "省|县市|区号|邮编" '指定数据接收列
tb.SeparateContent = True '启用内容分割功能
tb.ContentSeparator = "|" '指定内容分割符号
Tables("客户").Cols("省").DropTree = tb.Build()
```

示例七

有的时候，目录树表的所有分类内容都在同一列中，如下图：

	分类
1	数码产品|电脑|笔记本
2	数码产品|电脑|台式机
3	数码产品|电脑|服务器
4	数码产品|相机|DC
5	数码产品|相机|DV
6	家电产品|电视机|液晶电视
7	家电产品|电视机|CRT电视
8	数码产品|电脑配件|机箱
9	数码产品|电脑配件|主板

如果需要根据这一列的内容，来生成一个多层的目录树，指定目录树列的时候，应该将列名用大括号括起来：

```
Dim tb As New DropTreeBuilder
tb.SourceTable = DataTables("分类") '指定目录树表
tb.TreeCols = "{分类}" '指定用于生成目录树的列,用大括号括起来
tb.ReceiveCols = "大类|二类|三类" '指定数据接收列
tb.PathSeparator = "|" '指定路径分割符号
Tables("订单").Cols("大类").DropTree = tb.Build()
```

4.4.6.17 单元格多行内容的处理

我们知道，多文件列可以管理多个文件，图片列可以管理多个图片。

也就是说，这些列的单元格，可能会有多行的内容，每行对应一个文件。

DataRow 提供了一个 Lines 属性，该属性以字符集合的形式返回或设置单元格内容。

Lines 属性的出现，可以让我们以非常轻松的方式处理单元格的多行文本内容。

示例一

假定表中有两列，列名分别为“文件”和“数量”，前者是多文件型，希望在每行的数量列统计出文件的数量。

为此，可将 DataColChanged 事件代码设置为：

```
If e.DataCol.name = "文件" Then
    Dim fls As List(of String)
    fls = e.DataRow.Lines("文件")
    e.DataRow("数量") = fls.Count
End If
```

示例二

如果希望每行的文件列至少保留一个文件，也就是当只剩下一个文件的时候，禁止删除文件。

为此可以将 BeforeDeleteFile 事件代码设置为：

```
If e.DataCol.name = "文件" Then
    If e.DataRow.Lines("文件").Count = 1 Then
```

```
        MessageBox.Show("至少要保留一个文件!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
        e.cancel = True
    End If
End If
```

示例三

也可以直接以字符集合的形式给单元格赋值，例如从目录“c:\data”下取得所有文件，赋值给文件列：

```
Dim lst As New List(of String)
For Each fl As String In FileSys.GetFiles("c:\data")
    lst.Add(fl)
Next
Tables("表A").Current.DataRow.Lines("文件") = lst
```

再例如，从文件列中删除一个文件：

```
Dim lst As New List(of String)
lst = Tables("表A").Current.DataRow.Lines("文件")
lst.Remove("明细.xls")
Tables("表A").Current.DataRow.Lines("文件") = lst
```

或者增加一个文件：

```
Dim lst As New List(of String)
lst = Tables("表A").Current.DataRow.Lines("文件")
lst.Add("明细.xls")
Tables("表A").Current.DataRow.Lines("文件") = lst
```

有了Lines属性，我们可以非常灵活地处理文件列和图片列了。

示例四

Foxtable通过文件列和图片列，可以很方便地管理文件和图片，如果想自己设计一个按钮，单击此按钮可以 向列中增加文件，可以参考下面的代码：

```
Dim dlg As New OpenFileDialog '定义一个新的OpenFileDialog
dlg.Filter= "图形文件|*.bmp;*.jpg;*.gif" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok  Then  '如果用户单击了确定按钮
    Dim dr As DataRow = Tables("表A").Current.DataRow
    Dim lst As List(of String) =  dr.Lines("文件")  '获取文件集合
    lst.Add(Filesys.GetName(dlg.FileName))  '给集合增加一个文件
    dr.Lines("文件") = lst  '将新的集合赋值给文件列
    If  dlg.FileName.ToLower.StartsWith((ProjectPath  &  "Attachments").ToLower)  =  False  Then '如果文件不在Attachments目录
```

```
    FileSys.CopyFile(dlg.FileName,ProjectPath & "Attachments\" & Filesys.GetName(dlg.FileName),True)  '那么复制到Attachments目录
  End If
End If
```

4.4.6.18 按时段筛选数据

我们经常需要筛选某个时段的数据，下面列出了各种情况的筛选代码给大家参考。

筛选指定年份数据

例如筛选 2011 年的订单：

```
Dim y As Integer = 2011 '指定年份
Dim dt1 As New Date(y, 1, 1)
Dim dt2 As New Date(y, 12, 31)
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

筛选指定月份数据

例如筛选 2011 年 6 月份的订单：

```
Dim y As Integer = 2011 '指定年份
Dim m As Integer = 6 '指定月份
Dim dt1 As New Date(y, m, 1)
Dim dt2 As New Date(y, m, Date.DaysInMonth(y,m)) '获取该月的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

筛选指定季度的数据

例如筛选 2011 年 2 季度的订单：

```
Dim y As Integer = 2011 '指定年份
Dim q As Integer = 2 '指定季度
Dim dt1 As New Date(y, 3 * (q - 1) + 1, 1) '获取该季度的第一天
Dim dt2 As New Date(y, 3 * q, Date.DaysInMonth(y,3 * q)) '获取该季度的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

筛选本年数据

例如筛选本年的订单：

```
Dim y As Integer = Date.Today.Year
Dim dt1 As New Date(y, 1, 1)
Dim dt2 As New Date(y, 12, 31)
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

筛选本季度数据

例如筛选本季度的订单：

```
Dim y As Integer = Date.Today.Year
Dim q As Integer = (Date.Today.Month - 1) \ 3 + 1 '计算现在是第几个季度
Dim dt1 As New Date(y, 3 * (q - 1) + 1, 1) '获取本季度的第一天
Dim dt2 As New Date(y, 3 * q, Date.DaysInMonth(y,3 * q)) '获取本季度的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

注意第二行代码，用的是整除运算符（\），而不是通常的除法运算符（/），整除运算符会直接对结果取整。

筛选本月数据

例如筛选本月的订单：

```
Dim y As Integer = Date.Today.Year
Dim m As Integer = Date.Today.Month
Dim dt1 As New Date(y, m, 1)
Dim dt2 As New Date(y, m, Date.DaysInMonth(y, m)) '获取本月的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

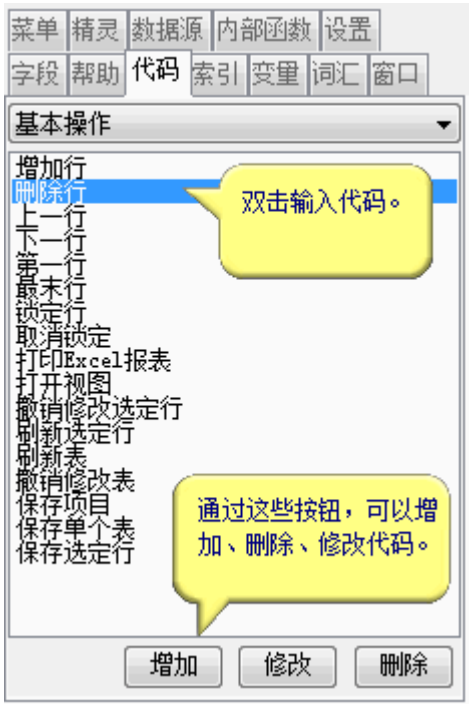
筛选本周数据

例如筛选本周的订单：

```
Dim y As Integer = Date.Today.Year
Dim w As Integer = Date.Today.DayOfWeek '算出今天是星期几
Dim dt1 As Date = Date.Today.AddDays(-w) '获取本周的第一天
Dim dt2 As Date = Date.Today.AddDays(6 - w) '获取本周的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

4.4.6.19 管理自己的代码库

代码编辑器内置了一个代码库功能，你可以将一些常用代码添加到这里，需要的时候双击即可输入这些代码：



提示：

代码库保存在 foxtable 安装目录下的 Code.mdb 文件中，每次修改，Foxtable 都会自动备份此文件，备份文件名为 code.mdb.bak。

尽管有自动备份功能，但如果你在代码库中加入了大量的代码，我们还是建议您手工备份此文件，以确保安全。

4.4.6.20 长代码换行

如果某一行代码特别长，会给编辑和查阅带来不便。

实际上，我们可以将一行代码分成多行书写，例如：

```
If a = 1 And b =1 Then
```

等效于：

```
If a = 1 _
And b = 1 _
Then
```

后者分成了 3 行书写，注意符号"_"的前面必须有个空格，也就是说在代码中间换行输入的时候，必须以" _"结尾。

如果是长字符串，可以拆分成多个字符串，然后用符号"&"连接起来，例如：

```
Dim s As String
s = "当我们运行发布后的程序时,Publish 目录相当于原来的 Foxtable 安装目录,而 Publish 目录下的子目录 Project,则相当于原来的项目文件所在目录"
```

可以改写为:

```
Dim s As String
s = "当我们运行发布后的程序时," & _
"Publish 目录相当于原来的 Foxtable 安装目录," & _
"而 Publish 目录下的子目录 Project,则相当于" & _
"原来的项目文件所在目录"
```

4.5 统计与查询

4.5.1 设置合并模式

下面是 Table 类型和合并模式相关的属性：

属性名	说明
MergeMode	获得或者设置 Table 的合并方式
MergeCols	一个字符集合，包括需要合并列的名称
MergeSort	返回或设置启用合并模式后的排序方式
AllowMerge	逻辑型，用于切换合并模式

MergeMode 属性是 MergeModeEnum 型枚举，包括以下可选值：

值	说明
None	不合并
Standard	标准合并模式，左边的单元格必须处于合并状态下，内容相同的单元格才会合并。
Free	自由合并，不管左边单元格是否处于合并模式，只要相邻单元格内容相同，均会合并

示例

下面的代码合并订单表中的客户、雇员两列，因为是标准合并模式，请在执行之前，将客户、雇员两列移到最左边。

完整的合并模式设置代码：

```
With Tables("订单")
```

```
    .MergeMode = MergeModeEnum.Standard '标准合并模式
    .MergeCols.Clear() '清除原来的合并列
    .MergeCols.Add("客户") '加入要合并的列
    .MergeCols.Add("雇员")
    .MergeSort = "客户,雇员" '设置合并模式下的排序方式
    .AllowMerge = True '启用合并模式
End With
```

如果要关闭合并模式：

```
Tables("订单").AllowMerge = False
```

如果要恢复上次合并：

```
Tables("订单").AllowMerge = True
```

自动生成代码

要学习合并模式设置的方法和属性，最好的办法是在合并模式窗口设置好合并模式，然后单击按钮"查看代码"，即可根据现有设置生成等效的代码。



两种合并模式

下图说明了两种合并模式的不同：



显然，对于标准合并方式，我们应该将需要合并的列移到最左边。

4.5.2 设置汇总模式

下面是 Table 类型和汇总模式相关的属性和方法。

属性

属性名	说明
GroupAboveData	逻辑型，该属性决定汇总模式下，分组行是否位于数据行之前，默认为 False，设为 True，分组行将位于数据行之前
TreeVisible	逻辑型，该属性决定在汇总模式下是否显示目录树，默认为 False，设为 True，将显示目录树

IsSubtotal	逻辑型，判断 Table 是否处于汇总模式
SubtotalGroups	集合，包括汇总模式下的各个分组设置

方法

方法名	说明
Subtotal	根据现有的分组设置生成汇总模式
ClearSubtotal	退出汇总模式

最重要的属性是 SubtotalGroups，这是一个 SubtotalGroup 型集合，包括汇总模式下的各个分组设置。

SubtotalGroup 指的是汇总模式下的分组，包括以下属性：

属性	说明
GroupOn	指定分组列，如果是总计分组，用星符号(*)表示
TotalOn	指定统计列，如果要对多列统计，列与列之间用逗号分隔。
Aggregate	AggregateEnum 型枚举，指定统计类型，包括以下可选值： Average：平均值 Count：记录数 Max：最大值 Min：最小值 Percent：百分比 Std：标准偏差 StdPop：总体标准偏差 Sum：累积值 Var：方差 VarPop：总体方差
DateGroup	DateGroupEnum 型枚举，指定日期列的分组方式，包括以下可选值： None：无 Year: 按年 Quarter：按季度 Month：按月 Week：按周 Day：按天 Hour：按小时 Minute：按分 Second：按秒
Caption	字符型，指定标题，标题中使用"{0}"表示分组名称
Upto	逻辑型，是否是截止统计。

例如：

```
Dim g As New SubtotalGroup '定义一个新的分组
g.Aggregate = AggregateEnum.Sum '统计类型为求和
g.GroupOn = "客户" '分组列为客户列
```

```
g.TotalOn = "数量,金额" '对数量和金额进行统计
g.Caption = "{0} 小计" '设置标题
With Tables("订单")
    .SubtotalGroups.Clear() '清除原有的分组
    .SubtotalGroups.Add(g) '加入刚刚定义的分组
    .Subtotal() '生成汇总模式
End With
```

在进行新的分组统计之前，应该先清除原来的分组。

一个汇总模式可能有多个分组，添加分组的先后顺序会影响统计结果，原则是：首先添加小的分组，然后添加大的分组，最后总计分组。

此外，根据日期型列进行分组的时候，日期分组必须位于所有分组之前。

示例一

下面的代码对订单表设置汇总模式，根据客户和产品分组，累积数量和金额列。

```
Dim t As Table = Tables("订单")
Dim g As SubtotalGroup
t.SubtotalGroups.Clear()
'定义客户分组
g = New SubtotalGroup '定义一个新的分组
g.Aggregate = AggregateEnum.Sum '统计类型为求和
g.GroupOn = "客户" '分组列为客户
g.TotalOn = "数量,金额" '统计数量和金额列
g.Caption = "{0} 小计" '设置标题
t.SubtotalGroups.Add(g) '加到分组集合中
'定义产品分组
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "产品"
g.TotalOn = "数量,金额"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)
'定义总计分组
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "*" '注意总计分组用符号*表示。
g.TotalOn = "数量,金额"
g.Caption = "总计"
t.SubtotalGroups.Add(g)
t.Subtotal() '生成汇总模式
```

代码虽然比较长，但是每一段几乎都一样，不同的只是列名。

示例二

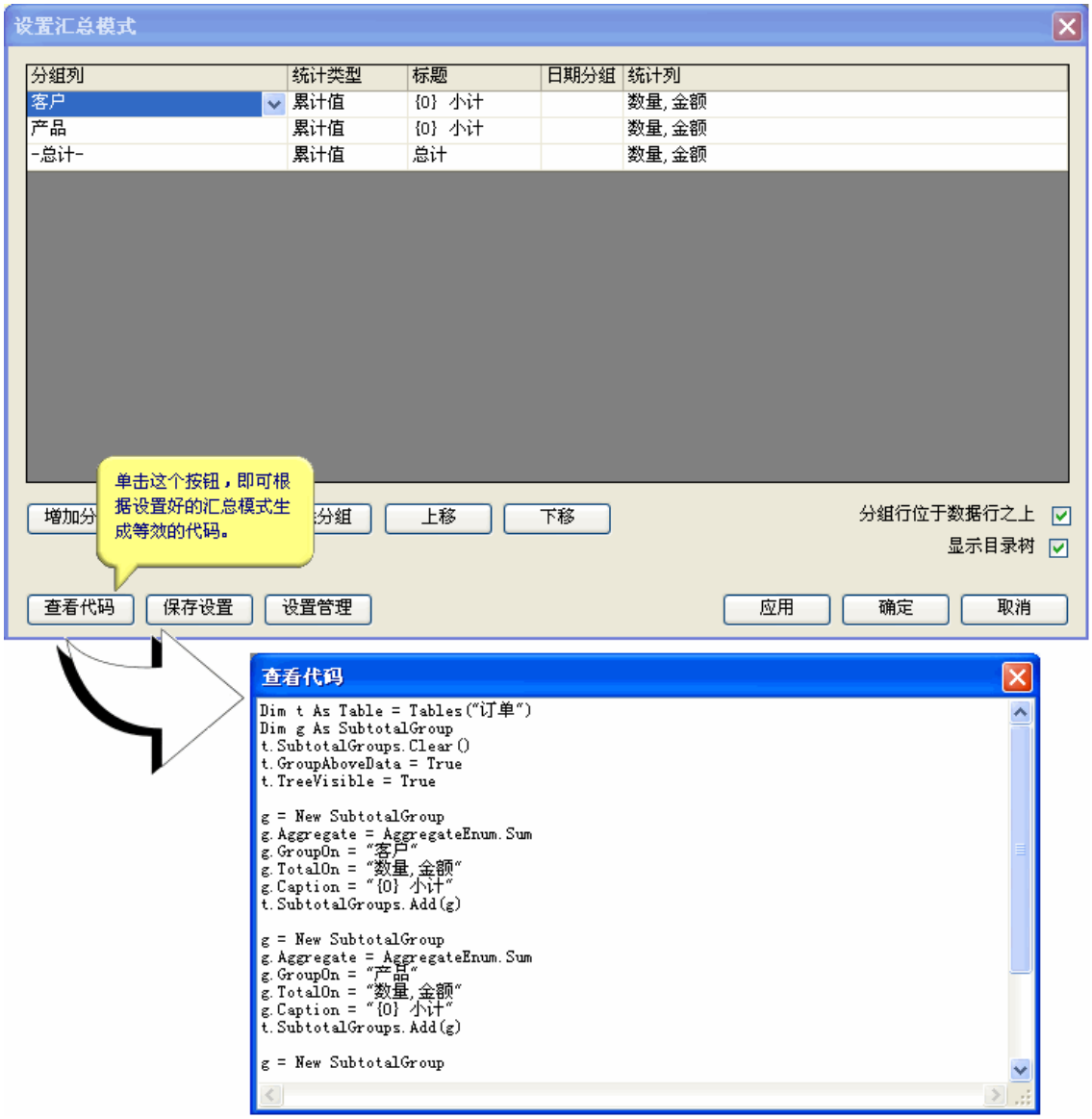
按月统计不同产品的销售数量：

```
Dim t As Table = Tables("订单")
Dim g As SubtotalGroup
t.SubtotalGroups.Clear()
t.GroupAboveData = False
```

```
t.TreeVisible = False
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "日期"
g.TotalOn = "数量"
g.Caption = "{0}月 小计"
g.DateGroup = DateGroupEnum.Month
t.SubtotalGroups.Add(g)
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "产品"
g.TotalOn = "数量"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)
t.Subtotal()
```

自动生成代码

要学习汇总模式设置的方法和属性，最好的办法是在汇总模式设置窗口中，先设置好汇总模式，然后单击按钮"查看代码"，即可根据现有设置生成等效的代码。



接下来的内容，一般用户可以忽略。

隐藏汇总模式下的非相关列

如果一个表有很多列，可能希望执行汇总模式后，能够隐藏非相关列，只显示分组列和统计列。要实现这个目的，可以参考下面的代码：

```
Dim lst As New List(of String)
Dim tbl As Table = Tables("订单")
For each g As SubtotalGroup In tbl.SubtotalGroups
    For Each s As string In g.Totalon.Split(",")
        If lst.Contains(s) = False Then
            lst.Add(s)
        End If
    Next
    If lst.Contains(g.Groupon) = False Then
        lst.Add(g.GroupOn)
```

```
    End If
Next
For Each c As Col In tbl.Cols
    c.Visible = lst.Contains(c.Name)
Next
```

上面的代码很有意思的，这是帮助中少有的真正利用集合来编程的例子。

如果希望重新显示所有列，可以利用表视图，例如建立了一个包括所有列的表视图，名称为“默认视图”，可以直接执行下面的代码来显示所有列：

```
Tables("订单").OpenView("默认视图")
```

如果不想建立表视图，也可以使用下面的代码来显示所有列：

```
For Each c As Col In Tables("订单").Cols
    c.Visible = True
Next
```

自定义排序

在默认情况下，Subtotal 方法会根据分组列进行排序，然后再生成汇总模式。

Subtotal 有一个可选参数，如果将其设置为 True，那么在生成汇总模式时，可以自定义排序参数，而不是根据分组列排序。

例如根据日期和客户排序，分组列为客户，统计列为数量与金额，生成汇总模式：

```
Dim t As Table = Tables("订单")
Dim g As Subtotalgroup
t.SubtotalGroups.Clear()
g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "客户"
g.TotalOn = "数量,金额"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)
g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "*"
g.TotalOn = "数量,金额"
g.Caption = "总计"
t.SubtotalGroups.Add(g)
t.Sort = "日期,客户" '指定排序字段
t.Subtotal(True) '生成汇总模式
```

4.5.3 设置合计模式

和汇总模式相比，本节介绍的合计模式设置简单，功能单一，只有一个总的合计行。

但合计模式有个极大的优点，在此模式下，你可以正常编辑数据、增加行、删除行，还可以进行筛选和排序等操作，合计结果会即时刷新，而之前介绍的汇总模式，数据是只读的。

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
2	PD01	CS03	EP04	14.4	0.25	200	2160	1999-01-14
3	PD01	CS03	EP01	14.4	0	240	3456	1999-01-21
4	PD01	CS03	EP04				720	1999-02-07
5	PD01	CS03	EP01				5760	1999-03-03
6	PD01	CS03	EP01				1728	1999-03-14
7	PD01	CS03	EP01				2025	1999-03-16
8	PD01	CS03	EP05				1800	1999-03-24
9	PD01	CS03	EP01				1620	1999-05-27
10	PD01	CS03	EP01	18	0.25	350	4725	1999-06-02
	合计					1780	25218	

在合计模式下，可以直接编辑数据、增加、删除行，也可以筛选、排序数据，合计结果会自动更新。

合计模式和汇总模式各有优势，一个可以编辑数据，一个功能强大，我们可以根据需要选用。要设置合计模式很简单，以上图的合计模式为例，只需三行代码：

```
Tables("订单").Cols("数量").GrandTotal = True '指定要合计的列
Tables("订单").Cols("金额").GrandTotal = True
Tables("订单").GrandTotal = True '显示合计模式
```

如果要关闭合计模式，将GrandTotal属性设置为False即可：

```
Tables("订单").GrandTotal = False
```

合计模式经常用于在关联子表显示一个合计行，自动合计子表数据：



以上图为例，要在关联子表中显示一个合计行，代码为：

```
With Tables("产品.订单")
    .Cols("数量").GrandTotal = True '指定要合计的列
    .Cols("金额").GrandTotal = True
    .GrandTotal = True '显示合计模式
End With
```

4.5.4 分组统计

4.5.4.1 分组统计表生成器

```
b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("产品") '根据产品分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

执行上述代码，可以得到不同产品的销售数量:

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

再例如要统计 CS01 客户订购各种产品的数量：

```
'定义一个 GroupTableBuilder，名称为"统计表 1"，对订单表进行统计。
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Filter = "[客户] = 'CS01'" '仅统计客户为 CS01 的记录
b.Groups.AddDef("产品") '根据产品分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

代码很简单，几乎具备自我描述性，无非就是定义一个生成器，指定分组列和统计列，然后生成统计表。

GroupTableBuilder 是专门针对分组统计而设计的，定义一个 GroupTableBuilder 的语法如下：

```
New GroupTableBuilder(TableName, SourceDataTable)
```

参数说明：

- TableName
  字符型， 指定即将生成的统计表的名称
- SourceDataTable
  DataTable 型，指定要统计的数据表。

详解 GroupTableBuilder

下面有些内容比较抽象，在接触具体的示例前，会比较难以理解，没有关系，先大概了解一下即可。

GroupTableBuilder 包括以下属性：

Caption： 字符型，指定新生成统计表的标题。
Groups: 集合，用于添加分组列
Totals: 集合，用于添加统计列。
VerticalTotal： 逻辑型，设为 True，垂直方向自动增加汇总行。
Subtotal: 逻辑型，设为 True，将对分组统计结果进行二次统计，生成汇总模式。
SubtotalLevel: 整数型，根据多少个分组生成汇总模式。
Decimals： 整数型，用于设置统计结果要保留的小数位数，默认为 4 位。
Filter: 字符型，如果只是对 SourceDataTable 中的部分数据进行统计，可用 Filter 属性设置条件。

请参考表达式的运算符和函数 和 条件表达式。

在 Groups 集合中添加一个分组列的语法是：

```
Groups.AddDef(Name)
Groups.AddDef(Name, NewName)
Groups.AddDef(Name, NewName, NewCaption)
Groups.AddDef(Name, DateGroup)
Groups.AddDef(Name, DateGroup, NewName)
Groups.AddDef(Name, DateGroup, NewName, NewCaption)
```

参数说明

Name: 字符型，指定分组列。
NewName: 字符型，用于给分组列指定一个新的名称，这个名称在统计表中显示。
NewCaption: 字符型，用于给分组列设置标题。
DateGroup： DateGroupEnum 型枚举，如果分组列是日期列，可用此属性指定如何根据日期进行分组，默认按月分组。

DateGroupEnum 枚举的可选值：

None ： 不分组
Year： 按年分组
Quarter： 按季度分组
Month： 按月分组
Week： 按周分组
Day： 按天分组，月中的某一天。
Hour： 按小时分组
Minute： 按分钟分组
Second： 按秒分组

在 Totals 集合中添加一个统计列的语法如下：

```
Totals.AddDef(Name)
```

Totals.AddDef(Name, NewName)
Totals.AddDef(Name, NewName, NewCaption)
Totals.AddDef(Name, Aggregate)
Totals.AddDef(Name, Aggregate, NewName)
Totals.AddDef(Name, Aggregate, NewName, NewCaption)

参数说明

Name: 字符型，指定统计列。
NewName: 字符型，用于给统计列指定一个新的名称，这个名称在统计表中显示。
NewCaption: 字符型，用于给统计列设置标题。
Aggregate： AggregateEnum 枚举型，用于指定统计类型，默认是求和。

AggregateEnum 型枚举的可选值：

Average：平均值
Count： 记录数
Max： 最大值
Min： 最小值
Sum： 累积值

实战演练

示例一，统计不同客户和产品的销售数量和金额

将下面的代码复制到命令窗口：

```
'定义一个 GroupTableBuilder，名称为"统计表 1"，对订单表进行统计。
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("客户") '添加客户列用于分组
b.Groups.AddDef("产品") '添加产品列用于分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加金额列用于统计
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

执行后，得到的统计结果：

	客户	产品	数量	金额
1	CS01	PD01	260	3777.12
2	CS01	PD02	2250	34814.6
3	CS01	PD03	2180	17720
4	CS01	PD04	230	3951.2
5	CS01	PD05	600	12338.7
6	CS02	PD01	4120	60292.8
7	CS02	PD02	4300	64634.2
8	CS02	PD03	2250	18940
9	CS02	PD04	1830	32414.8
10	CS02	PD05	2550	45418.05
11	CS03	PD01	1980	28638
12	CS03	PD02	6300	106352.5
13	CS03	PD03	970	9680

示例二，统计不同月份的销售数量和金额

将下面的代码复制到命令窗口：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("日期","月份") '添加日期列用于分组，并用"月份"代替原名称
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加金额列用于统计
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

执行结果：

	月份	数量	金额
1	1	7560	104923.5
2	2	8990	150142.4
3	3	8420	132185.05
4	4	10910	168037.2
5	5	10530	170795
6	6	5960	99568.42

示例三，按年统计不同月份的销售数量和金额

在示例二中，不同年但同月的数据，会作为同一个分组统计在一起，除非你的订单表只有一年的数据，或者你要的就是这样的统计结果，否则我们还需要加上一个按年统计的分组。
复制以下代码至命令窗口：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于分组，并用"年"代替原名称
b.Groups.AddDef("日期","月") '添加日期列用于分组，并用"月"代替原名称
b.Totals.AddDef("数量") '添加数量列用于统计
```

```
b.Totals.AddDef("金额") '添加金额列用于统计
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

执行结果：

	年	月	数量	金额
1	1999	1	7560	104923.5
2	1999	2	8990	150142.4
3	1999	3	8420	132185.05
4	1999	4	10910	168037.2
5	1999	5	10530	170795
6	1999	6	5960	99568.42

示例四，按月统计指定产品的销售数量和金额

复制以下代码至命令窗口执行：

```
Dim b As New GroupTableBuilder("统计表 1", DataTables("订单"), "产品 = 'PD01'")
b.Groups.AddDef("日期", DateGroupEnum.Year, "年") '添加日期列用于分组，并用"年"代替原名称
b.Groups.AddDef("日期","月") '添加日期列用于分组，并用"月"代替原名称
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加金额列用于统计
b.Build '生成统计
MainTable = Tables("统计表 1") '打开生成的统计表
```

即可统计出 PD01 在各月的销售数量。

示例五，统计不同客户在 1999 年各季度下的订单数，以及总的订购数量

复制以下代码至命令窗口：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Filter = "[日期] >= #1/1/1999# And [日期]<= #12/31/1999#"
b.Groups.AddDef("客户")'添加客户列用于分组
'添加日期列用于分组，按季度分组，并用"季度"代替原名称
b.Groups.AddDef("日期",DateGroupEnum.Quarter,"季度")
'添加客户列用于统计，其实对于 Count 类型的统计，可用任何列作为统计列。
b.Totals.AddDef("客户",AggregateEnum.Count,"订单数")
b.Totals.AddDef("数量","订购数量") '添加数量列用于统计
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

执行结果：

	客户	季度	订单数	订购数量
1	CS01	1	11	2500
2	CS01	2	15	3020
3	CS02	1	25	6540
4	CS02	2	26	8510
5	CS03	1	30	6510
6	CS03	2	26	6360
7	CS04	1	23	5460
8	CS04	2	33	6500
9	CS05	1	15	3960
10	CS05	2	12	3010

示例六，垂直方向自动汇总

统计不同客户的订购数量和金额，并在垂直方向增加一个自动汇总行:

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("客户") '添加客户列用于分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加数量列用于统计
b.VerticalTotal = True '自动垂直汇总
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

执行结果：

	客户	数量	金额
1	CS01	5520	72601.62
2	CS02	15050	221699.85
3	CS03	12870	211458.25
4	CS04	11960	204947
5	CS05	6970	114944.85
6	合计	52370	825651.57

提示： 垂直方向自动汇总现在比较少用了，因为 GroupTableBuilder 已经可以对生成的统计结果进行二次统计，直接生成汇总模式了，下面会有专门一节对此进行介绍。

4.5.4.2 分组统计之数据分析

在学习本课的内容之前，请打开 CaseStudy 目录下的示例文件"数据分析.Table"。
GroupTableBuilder 有以下属性，用于对统计结果进行进一步的分析：

- SamePeriodGrowth
  逻辑型，是否生成同比增长率

- CircleGrowth
  逻辑型，是否生成环比增长率

- GrandProportion
  逻辑型，是否生成总占比

- GroupProportion
  逻辑型，是否生成分组占比

示例一

```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
g.Totals.AddDef("数量")
g.GrandProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表：

	产品	数量	占比
1	PD01	11290	21.58%
2	PD02	18200	34.79%
3	PD03	7000	13.38%
4	PD04	5480	10.47%
5	PD05	10351	19.78%

示例二

```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
g.Groups.AddDef("客户")
g.Totals.AddDef("数量")
g.GroupProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表：

	产品	客户	数量	占比
1	PD01	CS01	260	2.30%
2	PD01	CS02	4120	36.49%
3	PD01	CS03	1980	17.54%
4	PD01	CS04	3390	30.03%
5	PD01	CS05	1540	13.64%
6	PD02	CS01	2250	12.36%
7	PD02	CS02	4300	23.63%
8	PD02	CS03	6300	34.62%
9	PD02	CS04	3110	17.09%
10	PD02	CS05	2240	12.31%
11	PD03	CS01	2180	31.14%
12	PD03	CS02	2250	32.14%
13	PD03	CS03	970	13.86%

示例三

```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
```

```
g.Groups.AddDef("客户")
g.Totals.AddDef("数量")
g.GroupProportion = True
g.GrandProportion= True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表：

	产品	客户	数量	占比	
				产品	总
1	PD01	CS01	260	2.30%	0.50%
2	PD01	CS02	4120	36.49%	7.87%
3	PD01	CS03	1980	17.54%	3.78%
4	PD01	CS04	3390	30.03%	6.48%
5	PD01	CS05	1540	13.64%	2.94%
6	PD02	CS01	2250	12.36%	4.30%
7	PD02	CS02	4300	23.63%	8.22%
8	PD02	CS03	6300	34.62%	12.04%
9	PD02	CS04	3110	17.09%	5.94%
10	PD02	CS05	2240	12.31%	4.28%

示例四

如果同时统计多列的占比,通过适当地设计统计列标题，可以让生成的统计表更为美观，例如：

```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
g.Totals.AddDef("数量", "", "数量_值")
g.Totals.AddDef("金额", "", "金额_值")
g.GrandProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

这是生成的统计表：

	产品	数量		金额	
		值	占比	值	占比
1	PD01	11290	21.58%	171754.92	20.82%
2	PD02	18200	34.79%	296522.5	35.95%
3	PD03	7000	13.38%	61490	7.45%
4	PD04	5480	10.47%	102861	12.47%
5	PD05	10351	19.78%	192190.15	23.30%

示例五

```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("日期", "月")
g.Totals.AddDef("数量")
g.CircleGrowth = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表：

	月	数量	环比
1	1	7511	
2	2	8990	19.69%
3	3	8420	-6.34%
4	4	10910	29.57%
5	5	10530	-3.48%
6	6	5960	-43.40%

示例六

```
Dim g As New GroupTableBuilder("统计表 2", DataTables("订单"))
g.Groups.AddDef("日期", DateGroupEnum.Year, "年")
g.Groups.AddDef("日期", "月")
g.Totals.AddDef("数量")
g.SamePeriodGrowth = True
g.Build()
MainTable = Tables("统计表 2")
```

生成的统计表：

	年	月	数量	同比
1	1999	1	7552	
2	1999	2	9136	
3	1999	3	9134	
4	1999	4	11078	
5	1999	5	11331	
6	1999	6	5253	
7	1999	7	7310	
8	1999	8	9194	
9	1999	9	7847	
10	1999	10	11260	
11	1999	11	10525	
12	1999	12	6019	
13	2000	1	7305	-3.27%
14	2000	2	9577	4.83%
15	2000	3	9173	0.43%
16	2000	4	11048	-0.27%
17	2000	5	9660	-14.75%
18	2000	6	6688	27.32%
19	2000	7	7017	-4.01%
20	2000	8	8511	-7.43%
21	2000	9	9138	16.45%
22	2000	10	10609	-5.78%
23	2000	11	11125	5.70%
24	2000	12	5527	-8.17%

示例七

```
Dim g As New GroupTableBuilder("统计表 2", DataTables("订单"))
g.Groups.AddDef("日期", DateGroupEnum.Year, "年")
g.Groups.AddDef("日期", "月")
g.Totals.AddDef("数量")
g.SamePeriodGrowth = True
g.CircleGrowth= True
g.Build()
```

```
MainTable = Tables("统计表 2")
```

生成的统计表：

	年	月	数量	环比	同比
1	1999	1	7552		
2	1999	2	9136	20.97%	
3	1999	3	9134	-0.02%	
4	1999	4	11078	21.28%	
5	1999	5	11331	2.28%	
6	1999	6	5253	-53.64%	
7	1999	7	7310	39.16%	
8	1999	8	9194	25.77%	
9	1999	9	7847	-14.65%	
10	1999	10	11260	43.49%	
11	1999	11	10525	-6.53%	
12	1999	12	6019	-42.81%	
13	2000	1	7305	21.37%	-3.27%
14	2000	2	9577	31.10%	4.83%
15	2000	3	9173	-4.22%	0.43%
16	2000	4	11048	20.44%	-0.27%
17	2000	5	9660	-12.56%	-14.75%
18	2000	6	6688	-30.77%	27.32%
19	2000	7	7017	4.92%	-4.01%
20	2000	8	8511	21.29%	-7.43%
21	2000	9	9138	7.37%	16.45%
22	2000	10	10609	16.10%	-5.78%
23	2000	11	11125	4.86%	5.70%
24	2000	12	5527	-50.32%	-8.17%

示例八

在对多列进行统计的时候，通过适当地设置标题，可以让生成的统计表更为美观，例如:

```
Dim g As New GroupTableBuilder("统计表 2", DataTables("订单"))
g.Groups.AddDef("日期", DateGroupEnum.Year, "年")
g.Groups.AddDef("日期", "月")
g.Totals.AddDef("数量", "", "数量_值")
g.Totals.AddDef("金额", "", "金额_值")
g.SamePeriodGrowth = True
g.CircleGrowth = True
g.Build()
MainTable = Tables("统计表 2")
```

这是生成的统计表：

	年	月	数量			金额		
			值	同比	环比	值	同比	环比
1	1999	1	7552			105629.17		
2	1999	2	9136		20.97%	151020.067		42.97%
3	1999	3	9134		-0.02%	143293.87		-5.12%
4	1999	4	11078		21.28%	171847.637		19.93%
5	1999	5	11331		2.28%	183151.207		6.58%
6	1999	6	5253		-53.64%	86066.03		-53.01%
7	1999	7	7310		39.16%	100822.49		17.15%
8	1999	8	9194		25.77%	155207.07		53.94%
9	1999	9	7847		-14.65%	124687.985		-19.66%
10	1999	10	11260		43.49%	175934.742		41.10%
11	1999	11	10525		-6.53%	171353.722		-2.60%
12	1999	12	6019		-42.81%	102384.326		-40.25%
13	2000	1	7305	-3.27%	21.37%	99036.85	-6.24%	-3.27%
14	2000	2	9577	4.83%	31.10%	159381.51	5.54%	60.93%
15	2000	3	9173	0.43%	-4.22%	143529.2	0.16%	-9.95%
16	2000	4	11048	-0.27%	20.44%	168608.34	-1.88%	17.47%
17	2000	5	9660	-14.75%	-12.56%	155910.525	-14.87%	-7.53%
18	2000	6	6688	27.32%	-30.77%	113760.398	32.18%	-27.03%
19	2000	7	7017	-4.01%	4.92%	94971.32	-5.80%	-16.52%
20	2000	8	8511	-7.43%	21.29%	143204.222	-7.73%	50.79%

4.5.4.3 分组统计之截止统计

前面已经讲到分组统计的时候，在 Totals 集合中添加一个统计列的语法如下：

Totals.AddDef(Name)
Totals.AddDef(Name, NewName)
Totals.AddDef(Name, NewName, NewCaption)
Totals.AddDef(Name, Aggregate)
Totals.AddDef(Name, Aggregate, NewName)
Totals.AddDef(Name, Aggregate, NewName, NewCaption)

如果要进行截止统计，只需在上述语法后面，加上一个参数 True：

Totals.AddDef(Name, True)
Totals.AddDef(Name, NewName, True)
Totals.AddDef(Name, NewName, NewCaption, True)
Totals.AddDef(Name, Aggregate, True)
Totals.AddDef(Name, Aggregate, NewName, True)
Totals.AddDef(Name, Aggregate, NewName, NewCaption, True)

示例

统计每月销售数量以及截止到该月的累计销售数量：

```
Dim g As New GroupTableBuilder("统计表1", DataTables("订单"))
g.Groups.AddDef("日期", "月")
g.Totals.AddDef("数量")
g.Totals.AddDef("数量", "累计", True)
g.Build()
MainTable = Tables("统计表1")
```

得到的统计结果为：

	月	小计	累计
1	1	1240	1240
2	2	450	1690
3	3	1200	2890
4	4	1390	4280
5	5	830	5110
6	6	900	6010

4.5.4.4 分组统计之汇总模式

在学习本课的内容之前，请打开 CaseStudy 目录下的示例文件"数据分析.Table"。
如果将 GroupTableBuilder 的 Subtotal 属性设置为 True，那么就可以对分组统计结果进行二次统计，生成汇总模式。

示例一

按年月统计销售数量和金额，并生成汇总模式：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于分组，并用"年"代替原名称
b.Groups.AddDef("日期","月") '添加日期列用于分组，并用"月"代替原名称
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加金额列用于统计
b.Subtotal = True '生成汇总模式
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

执行结果：

	年	月	数量
1	1999	1	41426
2	1999	2	44734
3	1999	3	42236
4	1999	4	43971
5	1999	5	39369
6	1999	6	39201
7	1999	7	39295
8	1999	8	51104
9	1999	9	51236
10	1999	10	33347
11	1999	11	41818
12	1999	12	37859
	小计 1999		505596
13	2000	1	44005
14	2000	2	29402
15	2000	3	39801
16	2000	4	41544
17	2000	5	42099
18	2000	6	34300
19	2000	7	43653
20	2000	8	46453
21	2000	9	37710
22	2000	10	47441
23	2000	11	38726
24	2000	12	39758
	小计 2000		484892
	总计		990488

示例二

在默认情况下，系统会根据所有分组生成汇总模式，不过我们可以精确控制。

GroupTableBuilder 有个名为 SubtotalLevel 的整数型属性，用于设置生成汇总模式的层数。

SubtotalLevel 的默认值为-1，根据所有分组生成汇总模式；设为 0，只生成总计行，设为 1，根据第一个分组生成汇总模式；设为 2，根据前两个分组生成汇总模式，其余类推。

例如按年月统计销售数量和金额，并生成汇总模式，但只生成总计行：

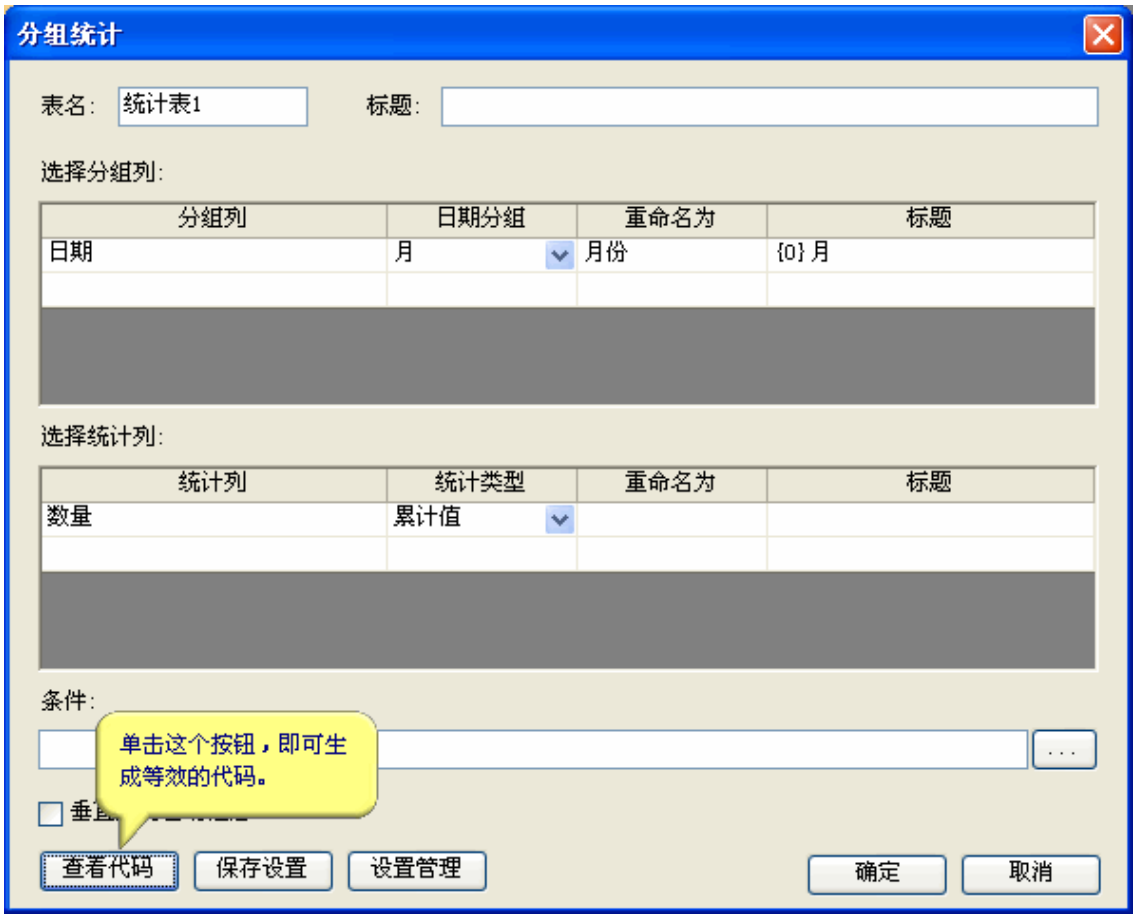
```
Dim b As New GroupTableBuilder("统计表1",DataTables("订单"))
b.Groups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于分组,并用"年"代替原名称
b.Groups.AddDef("日期","月") '添加日期列用于分组,并用"月"代替原名称
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加金额列用于统计
b.Subtotal = True '生成汇总模式
B.SubtotalLevel = 0 '只生成总计行
b.Build '生成统计表
MainTable = Tables("统计表1") '打开生成的统计表
```

得到的统计结果：

	年	月	数量	金额
1	1999	1	41426	931199.5
2	1999	2	44734	1022233
3	1999	3	42236	934657
4	1999	4	43971	955745.5
5	1999	5	39369	889428
6	1999	6	39201	876667.5
7	1999	7	39295	895298
8	1999	8	51104	1151191
9	1999	9	51236	1102699.5
10	1999	10	33347	720452
11	1999	11	41818	933640
12	1999	12	37859	901345.5
13	2000	1	44005	932355
14	2000	2	29402	685339.5
15	2000	3	39801	815941
16	2000	4	41544	944243.5
17	2000	5	42099	919495.5
18	2000	6	34300	708997
19	2000	7	43653	948762
20	2000	8	46453	1072998.5
21	2000	9	37710	840816
22	2000	10	47441	1019797
23	2000	11	38726	874931.5
24	2000	12	39758	863164
	总计		990488	21941397

4.5.4.5 统计代码的自动生成

如果你不想手工编写分组统计代码，可以在分组统计设置窗口中，先设置好相关设置，然后单击按钮"查看代码"，即可根据现有设置生成等效的代码：



4.5.5 交叉统计

4.5.5.1 交叉统计表生成器

参数说明

TableName： 字符型， 指定即将生成的统计表的名称 。

SourceDataTable： DataTable 型，指定要统计的数据表。

下面有些内容比较抽象，在接触具体的示例前，会比较难以理解，没有关系，先大概了解一下即可。

CrossTableBuilder 包括以下属性：

Caption： 字符型，指定新生成统计表的标题。

HGroups: 集合，用于添加水平分组列。

VGroups: 集合，用于添加垂直分组列。

Totals: 集合，用于添加统计列。

OrderByTotal： 逻辑型，在有多个统计列时，是否将同一个统计列的数据排放在一起。

HorizontalTotal: 逻辑型，水平方向是否自动汇总。

VerticalTotal： 逻辑型，垂直方向是否自动汇总。

Subtotal: 逻辑型，设为 True，将对交叉统计结果进行二次统计，生成汇总模式。

SubtotalLevel: 整数型，根据多少个分组生成汇总模式。

Decimals： 整数型，用于设置统计结果要保留的小数位数。

Filter:　　字符型，如果只是对 SourceDataTable 中的部分数据进行统计，可用 Filter 属性设置条件。
请参考表达式的运算符和函数 和 条件表达式。
添加水平分组到 HGroups 的语法是：
HGroups.AddDef(Name)
HGroups.AddDef(Name, NewName)
HGroups.AddDef(Name, NewName, NewCaption)
HGroups.AddDef(Name, DateGroup)
HGroups.AddDef(Name, DateGroup, NewName)
HGroups.AddDef(Name, DateGroup, NewName, NewCaption)
参数说明

Name:　　字符型，指定分组列。
NewName:　字符型，用于给分组列指定一个新的名称，这个名称在统计表中显示。
NewCaption:　字符型，用于给分组列设置标题。
DateGroup:　DateGroupEnum 型枚举，如果分组列是日期列，可用此属性指定如何根据日期进行分组，默认按月分组。
DateGroupEnum 枚举的可选值：

None ：　　不分组
Year:　　按年分组
Quarter:　　按季度分组
Month:　　按月分组
Week:　　按周分组
Day:　　按天分组，月中的某一天。
Hour:　　按小时分组
Minute:　　按分钟分组 Second:　　按秒分组
添加垂直分组到 VGroups 中的语法如下：
VGroups.AddDef(Name)
VGroups.AddDef(Name, Pattern)
VGroups.AddDef(Name, DateGroup)
VGroups.AddDef(Name, DateGroup, Pattern)
参数说明
Name:　　字符型，指定分组列
Pattern:　指定生成标题的模式，用字符"{0}"表示分组值；假定分组值是 10，模式是"第{0}周"，生成的分组标题将是"第 10 周"
DateGroup：DateGroupEnum 枚举型，如果分组列是日期列，可用此属性指定如何根据日期进行分组，默认按月分组。
添加统计列到 Totals 集合中的语法是：
Totals.AddDef(Name)
Totals.AddDef(Name, Caption)
Totals.AddDef(Name, Aggregate)
Totals.AddDef(Name, Aggregate,Caption)
参数说明

Name:　字符型，指定统计列
Caption:　字符型，指定统计列的标题
Aggregate: AggregateEnum 枚举型，用于指定统计类型，默认是求和。

AggregateEnum 型枚举的可选值：

Average：平均值
Count:　记录数
Max:　最大值
Min:　最小值
Sum:　累积值

示例一，统计同一客户订购不同产品的数量

将下面的代码复制到命令窗口执行：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("产品") '添加产品列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	PD01	PD02	PD03	PD04	PD05
1	CS01	260	2250	2180	230	600
2	CS02	4120	4300	2250	1830	2550
3	CS03	1980	6300	970	1200	2420
4	CS04	3390	3110	700	1410	3350
5	CS05	1540	2240	900	810	1480

示例二，美化示例一

将示例一的代码修改为：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("产品","产品_{0}") '添加产品列用于垂直分组，并设置了 Pattern 参数
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	产品				
		PD01	PD02	PD03	PD04	PD05
1	CS01	260	2250	2180	230	600
2	CS02	4120	4300	2250	1830	2550
3	CS03	1980	6300	970	1200	2420
4	CS04	3390	3110	700	1410	3350
5	CS05	1540	2240	900	810	1480

示例三，统计同一个客户订购不同产品的数量和金额

将下面的代码复制到命令窗口执行：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("产品") '添加产品列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	PD01		PD02		PD03		PD04		PD05	
		数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	CS01	260	3777.12	2250	34814.6	2180	17720	230	3951.2	600	12338.7
2	CS02	4120	60292.8	4300	64634.2	2250	18940	1830	32414.8	2550	45418.05
3	CS03	1980	28638	6300	106352.5	970	9680	1200	23320	2420	43467.75
4	CS04	3390	52155	3110	52706	700	6550	1410	27280	3350	66256
5	CS05	1540	26892	2240	38015.2	900	8600	810	15895	1480	25542.65

示例四，重复示例三，同种类型的统计排列在一起

示例三中，同一种产品的统计数据判列在一起，如果希望同种类型的统计数据排列在一起，以便于比较，可将代码修改为：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("产品") '添加产品列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Totals.AddDef("金额") '添加数量列用于统计
b.OrderByTotal = True '同种类型的统计数据排列在一起
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	数量					金额				
		PD01	PD02	PD03	PD04	PD05	PD01	PD02	PD03	PD04	PD05
1	CS01	260	2250	2180	230	600	3777.12	34814.6	17720	3951.2	12338.7
2	CS02	4120	4300	2250	1830	2550	60292.8	64634.2	18940	32414.8	45418.05
3	CS03	1980	6300	970	1200	2420	28638	106352.5	9680	23320	43467.75
4	CS04	3390	3110	700	1410	3350	52155	52706	6550	27280	66256
5	CS05	1540	2240	900	810	1480	26892	38015.2	8600	15895	25542.65

示例五：统计同一客户不同月份的订购数量

将下面的代码复制到命令窗口执行：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("日期","{0}月") '添加日期列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	1月	2月	3月	4月	5月	6月
1	CS01	800	720	980	1440	1150	430
2	CS02	2000	1590	2950	2620	3660	2230
3	CS03	2250	1800	2460	2670	2060	1630
4	CS04	1350	3100	1010	3350	2090	1060
5	CS05	1160	1780	1020	830	1570	610

示例六，按年月统计每个客户的订购数量

将下面的代码复制到命令窗口执行：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("日期",DateGroupEnum.Year,"{0}年") '添加日期列用于垂直分组，按年分组
b.VGroups.AddDef("日期","{0}月") '添加日期列用于垂直分组，按月分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	1999年					
		1月	2月	3月	4月	5月	6月
1	CS01	800	720	980	1440	1150	430
2	CS02	2000	1590	2950	2620	3660	2230
3	CS03	2250	1800	2460	2670	2060	1630
4	CS04	1350	3100	1010	3350	2090	1060
5	CS05	1160	1780	1020	830	1570	610

示例七，重复示例六，在垂直和水平方向加上汇总

```
Dim b As New CrossTableBuilder("统计表1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("日期",DateGroupEnum.Year,"{0}年") '添加日期列用于垂直分组,按年分组
b.VGroups.AddDef("日期","{0}月") '添加日期列用于垂直分组,按月分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.VerticalTotal= True '垂直汇总
b.HorizontalTotal = True '水平汇总
b.Build '生成统计表
MainTable = Tables("统计表1") '打开生成的统计表
```

统计结果：

	客户	1999年						合计
		1月	2月	3月	4月	5月	6月	
1	CS01	800	720	980	1440	1150	430	5520
2	CS02	2000	1590	2950	2620	3660	2230	15050
3	CS03	2201	1800	2460	2670	2060	1630	12821
4	CS04	1350	3100	1010	3350	2090	1060	11960
5	CS05	1160	1780	1020	830	1570	610	6970
6	合计	7511	8990	8420	10910	10530	5960	52321

提示： 垂直方向自动汇总现在比较少用了，因为 CrossTableBuilder 已经可以对生成的统计结果进行二次统计，直接生成汇总模式了，下面会有专门一节对此进行介绍。

示例八，重复示例六，但是水平分组和垂直分组对调

将下面的代码复制到命令窗口执行：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于水平分组，按年分组
b.HGroups.AddDef("日期","月") '添加日期列用于水平分组，按月分组
b.VGroups.AddDef("客户","客户_{0}") '添加客户列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	年	月	客户				
			CS01	CS02	CS03	CS04	CS05
1	1999	1	800	2000	2250	1350	1160
2	1999	2	720	1590	1800	3100	1780
3	1999	3	980	2950	2460	1010	1020
4	1999	4	1440	2620	2670	3350	830
5	1999	5	1150	3660	2060	2090	1570
6	1999	6	430	2230	1630	1060	610

示例九，多个水平分组

统计不同产品和客户，在各月份的销售数量，将下面的代码复制到命令窗口执行：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.HGroups.AddDef("产品") '添加客户列用于水平分组
b.VGroups.AddDef("日期",DateGroupEnum.Year,"{0}年") '添加日期列用于垂直分组，按年分组
b.VGroups.AddDef("日期","{0}月") '添加日期列用于垂直分组，按月分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

统计结果：

	客户	产品	1999年					
			1月	2月	3月	4月	5月	6月
1	CS01	PD01		100		150		10
2	CS01	PD02	500	380		400	970	
3	CS01	PD03	300	240	800	840		
4	CS01	PD04				50	180	
5	CS01	PD05			180			420
6	CS02	PD01	900	350	1200	820	550	300
7	CS02	PD02		240	760	1340	810	1150
8	CS02	PD03	1100	250	300		600	
9	CS02	PD04			210	460	500	660
10	CS02	PD05		750	480		1200	120
11	CS03	PD01	520	40	770	200	100	350
12	CS03	PD02	200	1350	680	1860	1480	730
13	CS03	PD03	490	300			180	
14	CS03	PD04	230	10	550	60	100	250
15	CS03	PD05	810	100	460	550	200	300
16	CS04			430				

4.5.5.2 交叉统计之数据分析

在学习本课的内容之前，请打开 CaseStudy 目录下的示例文件"数据分析.Table"。
CrossTableBuilder 有以下属性，用于对统计结果做进一步的分析：

- HorizontalProportion
  逻辑型，是否生成水平份额

- VerticalProportion
  逻辑型，是否生成垂直占比

- HorizontalProportionCaption
  字符型，用于设置水平份额标题，默认为“份额”

- VerticalProportionCaption
  字符型，用于设置垂直占比标题，默认为“占比”

示例一

```
Dim g As New CrossTableBuilder("统计表 1", DataTables("订单"))
g.HGroups.AddDef("产品")
g.VGroups.AddDef("客户")
g.Totals.AddDef("数量", "数量")
g.HorizontalTotal = True
g.VerticalTotal = True
g.VerticalProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表为：

	产品	CS01	CS01占比	CS02	CS02占比	CS03	CS03占比	CS04
1	PD01	260	4.71%	4120	27.38%	1980	15.44%	33
2	PD02	2250	40.76%	4300	28.57%	6300	49.14%	3
3	PD03	2180	39.49%	2250	14.95%	970	7.57%	7
4	PD04	230	4.17%	1830	12.16%	1200	9.36%	14
5	PD05	600	10.87%	2550	16.94%	2371	18.49%	3
6	合计	5520	100.00%	15050	100.00%	12821	100.00%	119

示例二

通过适当设置标题，可以美化统计表的结构。

例如：

```
Dim g As New CrossTableBuilder("统计表 1", DataTables("订单"))
g.HGroups.AddDef("产品")
g.VGroups.AddDef("客户", "{0}_数量")
g.Totals.AddDef("数量", "数量")
g.HorizontalTotal = True
g.VerticalTotal = True
g.VerticalProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表为：

	产品	CS01		CS02		CS03		CS04		CS05		合计
		数量	比例	数量	比例	数量	比例	数量	比例	数量	比例	
1	PD01	260	4.71%	4120	27.38%	1980	15.44%	3390	28.34%	1540	22.09%	11290
2	PD02	2250	40.76%	4300	28.57%	6300	49.14%	3110	26.00%	2240	32.14%	18200
3	PD03	2180	39.49%	2250	14.95%	970	7.57%	700	5.85%	900	12.91%	7000
4	PD04	230	4.17%	1830	12.16%	1200	9.36%	1410	11.79%	810	11.62%	5480
5	PD05	600	10.87%	2550	16.94%	2371	18.49%	3350	28.01%	1480	21.23%	10351
6	合计	5520	100.00	15050	100.00	12821	100.00	11960	100.00	6970	100.00	52321

示例三

```
Dim g As New CrossTableBuilder("统计表 1", DataTables("订单"))
g.HGroups.AddDef("产品")
g.VGroups.AddDef("日期", DateGroupEnum.Year, "{0}年_数量")
g.Totals.AddDef("数量", "数量")
g.HorizontalTotal = True
g.VerticalTotal = True
g.HorizontalProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表为：

	产品	1999年		2000年		合计
		数量	份额	数量	份额	
1	PD01	21791	48.83%	22837	51.17%	44628
2	PD02	37148	50.09%	37012	49.91%	74160
3	PD03	14448	50.42%	14207	49.58%	28655
4	PD04	12210	52.68%	10967	47.32%	23177
5	PD05	20042	49.57%	20392	50.43%	40434
6	合计	105639		105415		211054

示例四

```
Dim g As New CrossTableBuilder("统计表 1", DataTables("订单"))
g.HGroups.AddDef("产品")
g.VGroups.AddDef("日期", DateGroupEnum.Year, "{0}年_数量")
g.Totals.AddDef("数量", "数量")
g.HorizontalTotal = True
g.VerticalTotal = True
g.HorizontalProportion = True
g.VerticalProportion = True
g.Build()
MainTable = Tables("统计表 1")
```

生成的统计表为：

	产品	1999年			2000年			合计
		数量	份额	占比	数量	份额	占比	
1	PD01	21791	48.83%	20.63%	22837	51.17%	21.66%	44628
2	PD02	37148	50.09%	35.17%	37012	49.91%	35.11%	74160
3	PD03	14448	50.42%	13.68%	14207	49.58%	13.48%	28655
4	PD04	12210	52.68%	11.56%	10967	47.32%	10.40%	23177
5	PD05	20042	49.57%	18.97%	20392	50.43%	19.34%	40434
6	合计	105639		100.00%	105415		100.00%	211054

4.5.5.3 交叉统计之截止统计

前面已经讲到交叉统计的时候，在 Totals 集合中添加一个统计列的语法如下：

```
Totals.AddDef(Name)
Totals.AddDef(Name, Caption)
Totals.AddDef(Name, Aggregate)
Totals.AddDef(Name, Aggregate,Caption)
```

如果要进行截止统计，只需在上述语法后面，加上一个参数 True：

```
Totals.AddDef(Name, True)
Totals.AddDef(Name, Caption, True)
Totals.AddDef(Name, Aggregate, True)
Totals.AddDef(Name, Aggregate,Caption, True)
```

示例

按产品统计每月销售数量以及截止到该月的累计销售数量：

```
Dim g As New CrossTableBuilder("统计表1", DataTables("订单"))
g.HGroups.AddDef("日期", "月")
g.VGroups.AddDef("产品")
g.Totals.AddDef("数量", "数量")
g.Totals.AddDef("数量", "累计", True)
g.Build()
MainTable = Tables("统计表1")
```

得到的统计结果为：

	月	PD01		PD02		PD03		PD04		PD05	
		数量	累计	数量	累计	数量	累计	数量	累计	数量	累计
1	1	2120	2120	1300	1300	1930	1930	970	970	1191	1191
2	2	1600	3720	3170	4470	1290	3220	510	1480	2420	3611
3	3	2430	6150	2040	6510	1100	4320	1010	2490	1840	5451
4	4	2000	8150	5350	11860	1400	5720	940	3430	1220	6671
5	5	1870	10020	4140	16000	980	6700	1140	4570	2400	9071
6	6	1270	11290	2200	18200	300	7000	910	5480	1280	10351

4.5.5.4 交叉统计之汇总模式

在学习本课的内容之前，请打开 CaseStudy 目录下的示例文件"数据分析.Table"。
如果将 CrossTableBuilder 的 Subtotal 属性设置为 True，那么就可以对交叉统计结果进行二次统计，生成汇总模式。

示例一

按年月统计各产品销售数量，并生成汇总模式：

```
Dim b As New CrossTableBuilder("统计表 1",DataTables("订单"))
b.HGroups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于水平分组，按年分组
b.HGroups.AddDef("日期","月") '添加日期列用于水平分组，按月分组
b.VGroups.AddDef("产品","产品_{0}") '添加产品列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Subtotal = True '生成汇总模式
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

执行结果：

	年	月	产品				
			PD01	PD02	PD03	PD04	PD05
1	1999	1	5569	12720	8638	7325	7174
2	1999	2	9685	10004	6029	8633	10383
3	1999	3	9533	10753	8109	5759	8082
4	1999	4	9026	5337	7644	13885	8079
5	1999	5	4751	7840	7768	8903	10107
6	1999	6	7555	6951	6389	9841	8465
7	1999	7	5204	6807	6088	11864	9332
8	1999	8	8103	9755	8579	13565	11102
9	1999	9	11014	8657	10107	12863	8595
10	1999	10	6001	6504	7644	7302	5896
11	1999	11	7346	10433	8843	6165	9031
12	1999	12	6124	5120	4929	7714	13972
	小计 1999		89911	100881	90767	113819	110218
13	2000	1	8548	6409	11585	9142	8321
14	2000	2	5449	7677	3445	6328	6503
15	2000	3	11845	4223	9943	7001	6789
16	2000	4	8228	11208	7213	6194	8701
17	2000	5	10069	9154	8709	6439	7728
18	2000	6	8519	4516	9555	5244	6466
19	2000	7	6861	12682	12709	3821	7580
20	2000	8	6830	8774	9137	7926	13786
21	2000	9	9805	8897	5660	6915	6433
22	2000	10	13623	7286	8444	9471	8617
23	2000	11	6448	10598	7529	6231	7920
24	2000	12	7807	8205	9322	6658	7766
	小计 2000		104032	99629	103251	81370	96610
	总计		193943	200510	194018	195189	206828

示例二

在默认情况下，系统会根据所有分组生成汇总模式，不过我们可以精确控制。

CrossTableBuilder 有个名为 SubtotalLevel 的整数型属性，用于设置生成汇总模式的层数。

SubtotalLevel 的默认值为-1，根据所有分组生成汇总模式；设为 0，只生成总计行，设为 1，根据第一个分组生成汇总模式；设为 2，根据前两个分组生成汇总模式，其余类推。

例如按年月统计各产品的销售数量和金额，并生成汇总模式，但只生成总计行：

```
Dim b As New CrossTableBuilder("统计表1",DataTables("订单"))
b.HGroups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于水平分组,按年分组
b.HGroups.AddDef("日期","月") '添加日期列用于水平分组,按月分组
b.VGroups.AddDef("产品","产品_{0}") '添加产品列用于垂直分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Subtotal = True '生成汇总模式
b.SubtotalLevel = 0 '只生成总计行
b.Build '生成统计表
MainTable = Tables("统计表1") '打开生成的统计表
```

得到的统计结果：

	年	月	产品				
			PD01	PD02	PD03	PD04	PD05
1	1999	1	5569	12720	8638	7325	7174
2	1999	2	9685	10004	6029	8633	10383
3	1999	3	9533	10753	8109	5759	8082
4	1999	4	9026	5337	7644	13885	8079
5	1999	5	4751	7840	7768	8903	10107
6	1999	6	7555	6951	6389	9841	8465
7	1999	7	5204	6807	6088	11864	9332
8	1999	8	8103	9755	8579	13565	11102
9	1999	9	11014	8657	10107	12863	8595
10	1999	10	6001	6504	7644	7302	5896
11	1999	11	7346	10433	8843	6165	9031
12	1999	12	6124	5120	4929	7714	13972
13	2000	1	8548	6409	11585	9142	8321
14	2000	2	5449	7677	3445	6328	6503
15	2000	3	11845	4223	9943	7001	6789
16	2000	4	8228	11208	7213	6194	8701
17	2000	5	10069	9154	8709	6439	7728
18	2000	6	8519	4516	9555	5244	6466
19	2000	7	6861	12682	12709	3821	7580
20	2000	8	6830	8774	9137	7926	13786
21	2000	9	9805	8897	5660	6915	6433
22	2000	10	13623	7286	8444	9471	8617
23	2000	11	6448	10598	7529	6231	7920
24	2000	12	7807	8205	9322	6658	7766
	总计		193943	200510	194018	195189	206828

4.5.5.5 逻辑列与交叉统计

假定订单表有个逻辑列，名为“已付款”，用于标记某个订单是否已经付款。
现在要统计每个客户已经付款的金额、未付款的金额以及总的金额，通常会这样变编写代码：

```
Dim g As New CrossTableBuilder("统计表1", DataTables("订单"))
g.HGroups.AddDef("客户")
g.VGroups.AddDef("已付款")
g.Totals.AddDef("金额", "金额")
g.HorizontalTotal = True
g.VerticalTotal = True
g.Build()
MainTable = Tables("统计表1")
```

得到的统计结果是：

	客户	金额		合计
		1	2	
1	CS01	16520	56081.62	72601.62
2	CS02	23307.75	198392.1	221699.85
3	CS03	29845.75	180119.5	209965.25
4	CS04	39966	164981	204947
5	CS05	11620	103324.85	114944.85
6	合计	121259.5	702899.07	824158.57

虽然得到了期望的统计结果，但是出现“1”和“2”这样的列标题，是不够“友善”的。
我们可以调整一下代码，给逻辑列指定标题：

```
Dim g As New CrossTableBuilder("统计表1", DataTables("订单"))
g.HGroups.AddDef("客户")
g.VGroups.AddDef("已付款", "已付款|未付款")
g.Totals.AddDef("金额", "金额")
g.HorizontalTotal = True
g.VerticalTotal = True
g.Build()
MainTable = Tables("统计表1")
```

现在得到的统计结果是：

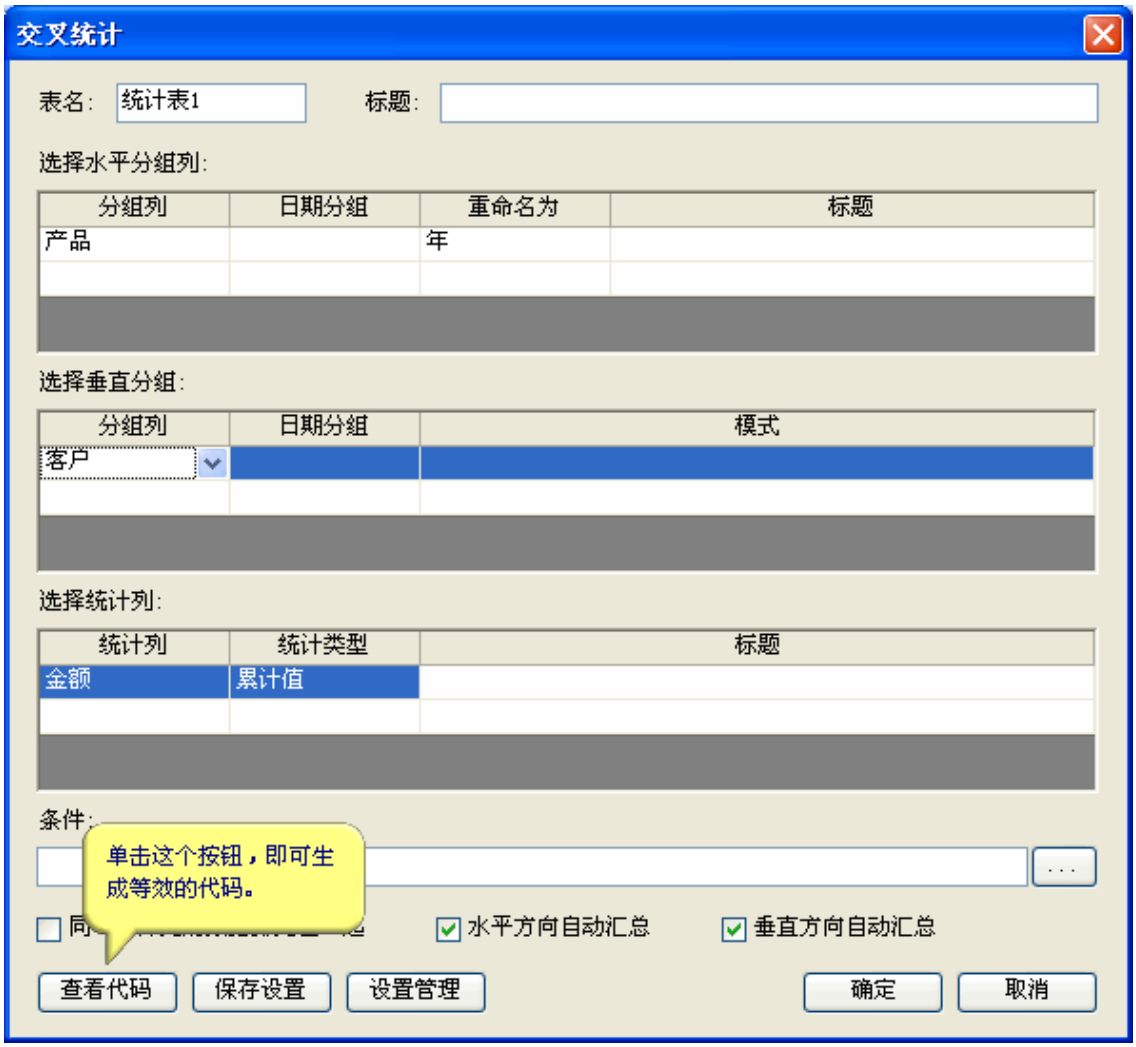
	客户	未付款	已付款	合计
1	CS01	16520	56081.62	72601.62
2	CS02	23307.75	198392.1	221699.85
3	CS03	29845.75	180119.5	209965.25
4	CS04	39966	164981	204947
5	CS05	11620	103324.85	114944.85
6	合计	121259.5	702899.07	824158.57

注意这行代码，第个参数用于设置逻辑列的标题，标题用符号“|”格式，第一部分表示值为True时的标题，第二部分表示值为False时的标题:

```
g.VGroups.AddDef("已付款", "已付款|未付款")
```

4.5.5.6 统计代码的自动生成

如果你不想手工编写交叉统计代码，可以在交叉统计设置窗口中，先设置好相关设置，然后单击按钮"查看代码"，即可根据现有设置生成等效的代码：



4.5.6 直接统计后台数据

如果数据量非常大，例如上百万甚至千万级别的记录，那么全部加载到内存是不可能的，所以 Foxtable 提供了动态加载数据的功能，使得在程序的运行过程中，也能随时根据需要加载符合条件的数据，这让 Foxtable 管理超大型数据库成为可能。

但是这有一个问题，动态加载部分数据是没问题的，但是很多时候，我们的统计是针对所有数据的，而 Foxtable 的分组统计和交叉统计默认都是针对已经加载的数据，如果我要统计所有数据，难道需要将一百万甚至一千万记录全部加载到 Foxtable？显然这是不可能的。

对于熟练掌握 SQL 语言的用户来说，这不是问题，直接自己编写 SQL 语句来统计即可；但实际上，SQL 语言虽然入门容易，但是精通是很难的，对于一些复杂的统计，即使是一些非常资深的程序员，也不见得能够在短时间内写出合理高效的 SQL 语句来，何况是普通用户呢？

为此 GroupTableBuilder 和 CrossTableBuilder 都提供了一个名为 FromServer 的逻辑属性，只需将此属性设为 True，即可直接统计后台数据。

例如：

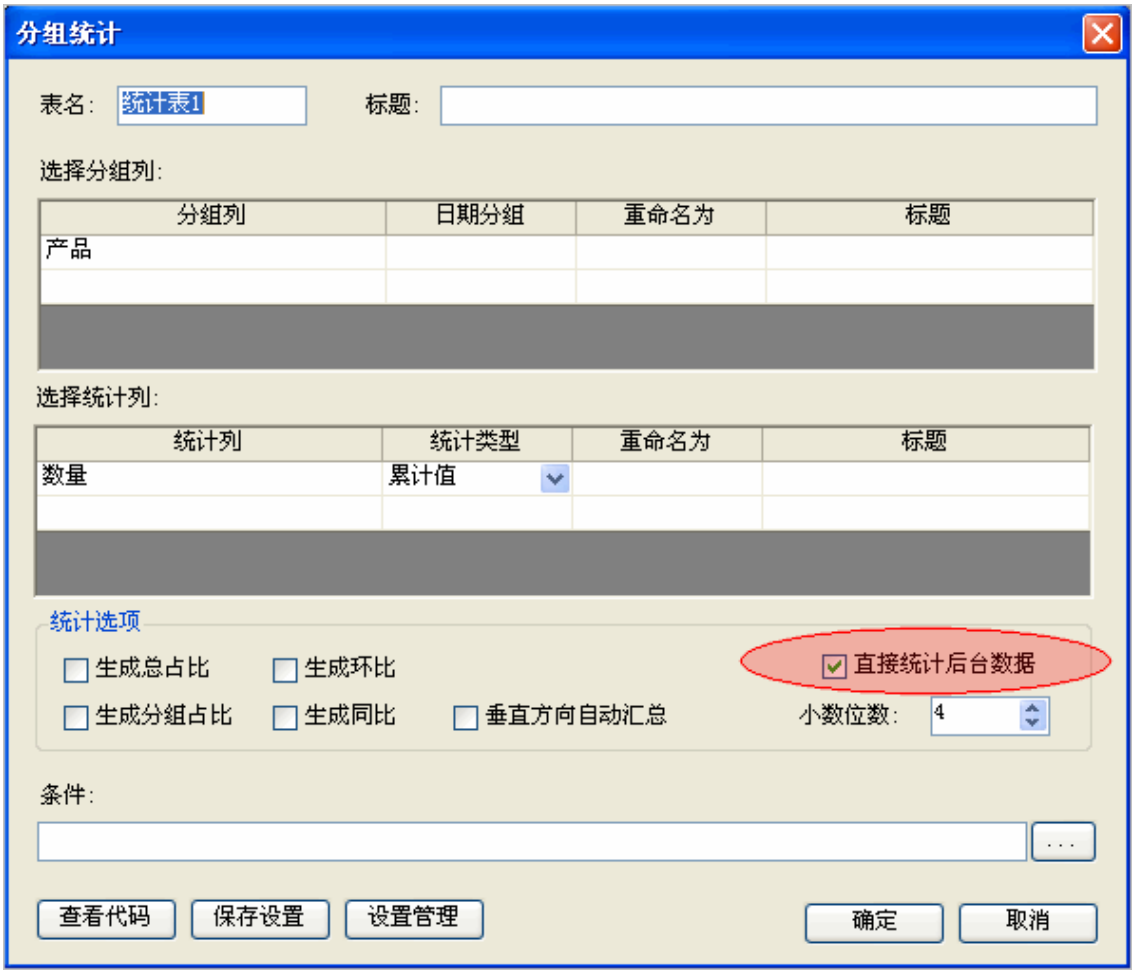
```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
g.Totals.AddDef("数量")
```

```
g.FromServer = True
g.Build()
MainTable = Tables("统计表 1")
```

上述代码能够自动生成最高效的 SQL 语句，来对后台数据进行统计。

对于一些复杂的统计，GroupTableBuilder 和 CrossTableBuilder 生成的 SQL 语句，会比最资深的程序员写出来的 SQL 语句还要高效。

如果通过菜单统计，且需要直接统计后台数据，可以选中下图所示的选项：



对于某些表，可能希望禁止分组统计和交叉统计窗口的“直接统计后台数据”选项，为此可将 Table 的 AllowBackgroundStatistics 属性设置为 False：

```
Tables("订单").AllowBackgroundStatistics = False
```

另 GroupTableBuilder 和 CrossTableBuilder 还提供一个名为 CommandTimeOut 的属性，用于设置统计后台数据的超时时限，默认为 30 秒，一般不需要设置此属性，除非需要进行非常耗时超过 30 秒的统计，以至于出现超时错误，此时可适当地加大 CommandTimeOut 属性的值。

提示：

统计后台数据时，GroupTableBuilder 和 CrossTableBuilder 的 Filter 属性采用 SQL 语法，和常规表达式有所差别，请参考：SQL 表达式的语法差异。

4.5.7 高级后台统计工具

4.5.7.1 还要更专业吗？

GroupTableBuilder 和 CrosssTableBuilder 使用简单，看上去无所不能，几乎能得到任何复杂的统计结果。
实际上在应对复杂应用的时候，这两个类型还有不足，体现在：
1、只能统计一个表的数据，无法组合多个 关联表的数据进行统计，如果不能解决此问题，有时将不得不添加一些冗余列，以保证统计的顺利进行。
2、只能统计单个列，不能自定义表达式进行统计。
所以 Foxtable 提供了两个更专业的统计工具，分别是 SQLGroupTableBuilder 和 SQLCrossTableBuilder，专门用于对后台数据进行统计，而且没有上述不足，当然使用上也会稍微繁琐一点。

4.5.7.2 SQLGroupTableBuilder

用于对后台数据进行分组统计。

定义一个 SQLGroupTableBuilder 的语法如下：

New SQLGroupTableBuilder(TableName, BaseTable)

参数说明：

- TableName
  字符型，指定即将生成的统计表的名称

- BaseTable
  字符型，指定默认的要统计的数据表，简称为基表
  可以是未加载的数据表，只要数据源存在此表即可。

在定义 SQLGroupTableBuilder 的时候，只能指定一个统计表，也就是基表。
如果要同时针对多个表进行统计，可以使用 AddTable 方法添加表，参与统计的表相互之间必须存在某种联系。

添加统计表的语法为：

AddTable(Table1,Col1,Table2,Col2)

Table1：基表，或者已经添加的数据表 ，称之为左表。
Col1： Table1 通过此列和 Table2 关联。
Table2：要添加的统计表，称之为右边。
Col2： Table2 通过此列和 Table1 关联。

提示：

1、Table1(左表)和 Table2(右表)并非一定要事先建立关联，只需两者的数据可以通过指定的列关联起来即可。
3、如果两个表存在一对多的父子关系，那么建议将多方(子表)作为左表，一方(父表)作为右表。
3、参与统计的表可以是未加载的数据表，只要数据源存在此表即可。

例如：

```
Dim sg As New SQLGroupTableBuilder("统计表 1","订单明细")
sg.AddTable("订单明细","产品 ID","产品","产品 ID")
```

表示基表是订单明细，此外添加产品表参与统计，产品表和订单明细通过产品 ID 列关联。
由于产品表和订单表是一对多的父子关系，所以用 AddTable 添加表的时候，订单表(子表)作为左表，产品表(父表)作为右表。
如果要根据多列关联，请参考：

```
Dim b As New SQLGroupTableBuilder("统计表 1","产品")
Dim Cols1() As String = {"型号","规格"}
Dim Cols2() As String = {"型号","规格"}
b.AddTable("产品",Cols1,"订单",Cols2)
```

上面的代码，产品表作为基表，添加订单表参与统计，两个表之间通过型号、规格两列建立关联。

SQLGroupTableBuilder 包括以下属性：

ConnectionName： 指定数据源名称，如果省略，表示使用内部数据表。
Caption： 字符型，指定新生成统计表的标题。
Groups: 集合，用于添加分组列。
Totals: 集合，用于添加统计列。
VerticalTotal： 逻辑型，设为 True，垂直方向自动增加汇总行。
Subtotal： 逻辑型，设为 True，将对分组统计结果进行二次统计，生成汇总模式。
Decimals： 整数型，用于设置统计结果要保留的小数位数，默认为 4 位。
SamePeriodGrowth：逻辑型，是否生成同比增长率。
CircleGrowth： 逻辑型，是否生成环比增长率。
GrandProportion： 逻辑型，是否生成总占比。
GroupProportion： 逻辑型，是否生成分组占比。
Filter: 字符型，设置统计条件。
CommandTimeOut： 整数型，用于设置统计后台数据的超时时限，默认为 30 秒，一般不需要设置此属性，除非需要进行非常耗时超过 30 秒的统计，以至于出现超时错误，此时可适当地加大 CommandTimeOut 属性的值。

在 Groups 集合中添加一个分组列的语法是：

```
Groups.AddDef(Name)
Groups.AddDef(Name, NewName)
Groups.AddDef(Name, NewName, NewCaption)
Groups.AddDef(Name, DateGroup)
```

Groups.AddDef(Name, DateGroup, NewName)
Groups.AddDef(Name, DateGroup, NewName, NewCaption)

参数说明

Name: 字符型，指定分组列。
NewName: 字符型，用于给分组列指定一个新的名称，这个名称在统计表中显示。
NewCaption: 字符型，用于给分组列设置标题。
DateGroup： DateGroupEnum 型枚举，如果分组列是日期列，可用此属性指定如何根据日期进行分组，默认按月分组。

DateGroupEnum 枚举的可选值：

None ： 不分组
Year： 按年分组
Quarter： 按季度分组
Month： 按月分组
Week： 按周分组
Day： 按天分组，月中的某一天。
Hour： 按小时分组
Minute： 按分钟分组
Second： 按秒分组

可以向 Groups 增加表达式分组列，语法为：

Groups.AddExp(Name, Expression)
Groups.AddExp(Name, Expression，Caption)

Name： 字符型，指定列名
Expression： 字符型，指定表达式。
Caption： 字符型，指定标题

在 Totals 集合中添加一个统计列的语法如下：

Totals.AddDef(Name)
Totals.AddDef(Name, NewName)
Totals.AddDef(Name, NewName, NewCaption)
Totals.AddDef(Name, Aggregate)
Totals.AddDef(Name, Aggregate, NewName)
Totals.AddDef(Name, Aggregate, NewName, NewCaption)

参数说明

Name: 字符型，指定统计列。
NewName: 字符型，用于给统计列指定一个新的名称，这个名称在统计表中显示。

NewCaption: 字符型，用于给统计列设置标题。
Aggregate： AggregateEnum 枚举型，用于指定统计类型，默认是求和。

AggregateEnum 型枚举的可选值：

Average：平均值
Count： 记录数
Max： 最大值
Min： 最小值
Sum： 累积值

可以向 Totals 添加表达式统计列，语法是：

Totals.AddExp(Name, Expression)
Totals.AddExp(Name, Expression, Caption)
Totals.AddExp(Name, Expression, Aggregate)
Totals.AddExp(Name, Expression, Caption, Aggregate)
Name： 字符型，指定列名
Expression：字符型，指定表达式。
Caption： 字符型，指定列标题
Aggregate： AggregateEnum 枚举型，用于指定统计类型，默认是求和。
例如：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.Totals.AddExp("金额","数量 * 单价 * (1-折扣)")
```

可以看出 SQLGroupTableBuilder 和 GroupTableBuilder 的使用方法基本相似，前者只是在后者的基础上增加了：
1、可以用 ConnectionName 指定数据源
2、可以用 AddTable 方法增加多个数据表
3、可以用 AddExp 方法增加表达式分组列或统计列
除此之外 SQLGroupTableBuilder 专门统计后台数据，而 GroupTableBuilder 默认统计的是已经加载的数据，如果需要统计后台数据，需将其 FromServer 属性设为 True。
提示：由于 SQLGroupTableBuilder 是根据用户的设置，自动生成 SQL 语句进行统计，所以 添加表或列时，名称必须和后台数据库保持一致，此外条件表达式的语法和函数与采用的数据源有关，我们会在《SQL 相关》这章对此进行专门的介绍。

统计单个数据表

在运行下面的示例之前，请打开 CaseStudy 目录下的文件：统计演示.Table
示例一
普通的统计，SQLGroupTableBuilder 和 GroupTableBuilder 用起来基本一样。
例如统计每个产品的销售数量：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.Groups.AddDef("产品") '根据产品分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

示例二

SQLGroupTableBuilder 可以添加表达式统计列，例如：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.Groups.AddDef("产品")
b.Totals.AddDef("数量")
b.Totals.AddExp("金额","数量 * 单价 * (1 - 折扣)")
b.Build
Maintable = Tables("统计表 1")
```

示例三

不仅统计列可以是表达式，分组列也可以是表达式，例如：

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.Groups.AddExp("年","Year(日期)")
b.Totals.AddDef("数量")
b.Totals.AddExp("金额","数量 * 单价 * (1 - 折扣)")
b.Build
MainTable = Tables("统计表1")
```

注意这样代码：

```
b.Groups.AddExp("年","Year(日期)")
```

是通过表达式列来实现按年分组的，等效于：

```
b.Groups.AddDef("日期",DateGroupEnum.Year,"年")
```

统计多个数据表

在运行下面的示例之前，请打开 CaseStudy 目录下的文件：多表统计.Table

这个文件的所有表都来自外部数据源，外部数据源的名称为 Sale，外部数据源是一个 Access 文件：多表统计.mdb

整个系统包括三个表，分别是客户、产品和订单：

	产品ID	产品名称
1	P01	运动饮料
2	P02	温馨奶酪
3	P03	三合一麦片
4	P04	浓缩咖啡
5	P05	盐水鸭

产品表

	客户ID	客户名称
1	C01	红阳事业
2	C02	威航货运有限公司
3	C03	浩天旅行社
4	C04	立日股份有限公司
5	C05	福星制衣厂股份有限公司

客户表

	产品ID	客户ID	单价	折扣	数量	日期
1	P05	C03	17	0.1	690	2000-01-02
2	P02	C01	20	0.1	414	2000-01-03
3	P05	C04	17	0	332	2000-01-03
4	P01	C03	18	0.15	-9	2000-01-04
5	P03	C03	10	0	445	2000-01-04
6	P04	C04	17.6	0	246	2000-01-07

订单表

订单表通过产品 ID 和产品表建立关联，通过客户 ID 和客户表建立关联。

示例一

假定要统计不同产品的销售数量，由于订单表没有产品名称列，如果只根据订单表生成统计表，只能显示产品 ID 及对应的销售数量，显然这样的统计表太不友好了，毕竟没有人会记得每个产品 ID 对应的产品名称是什么。

所以我们需要从产品表提取产品名称来进行统计：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.Groups.AddDef("产品名称") '根据产品名称分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果为：

	产品名称	数量
1	浓缩咖啡	23177
2	三合一麦片	28655
3	温馨奶酪	74160
4	盐水鸭	40397
5	运动饮料	44628

在使用外部数据源的时候，一定要记得设置 ConnectionName 属性：

```
b.ConnectionName = "数据源名称"
```

示例二

假定要统计每个产品在不同年份的销售数量：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.Groups.AddDef("产品名称") '根据产品名称分组
b.Groups.AddDef("日期", DateGroupEnum.Year, "年") '根据日期按年分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果为：

	产品名称	年	数量
1	浓缩咖啡	1999	12210
2	浓缩咖啡	2000	10967
3	三合一麦片	1999	14448
4	三合一麦片	2000	14207
5	温馨奶酪	1999	37148
6	温馨奶酪	2000	37012
7	盐水鸭	1999	20042
8	盐水鸭	2000	20355
9	运动饮料	1999	21791
10	运动饮料	2000	22837

示例三

统计结果可以来自任意多个表，例如要按年统计每个客户订购不同产品的数量：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.AddTable("订单","客户 ID","客户","客户 ID") '添加统计表
b.Groups.AddDef("客户名称") '根据产品名称分组
b.Groups.AddDef("日期", DateGroupEnum.Year, "年") '根据日期按年分组
b.Groups.AddDef("产品名称") '根据产品名称分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果：

	客户名称	年	产品名称	数量
1	福星制衣厂	1999	浓缩咖啡	1822
2	福星制衣厂	1999	三合一麦片	1670
3	福星制衣厂	1999	温馨奶酪	4706
4	福星制衣厂	1999	盐水鸭	3349
5	福星制衣厂	1999	运动饮料	3093
6	福星制衣厂	2000	浓缩咖啡	1954
7	福星制衣厂	2000	三合一麦片	1884
8	福星制衣厂	2000	温馨奶酪	4696
9	福星制衣厂	2000	盐水鸭	2614
10	福星制衣厂	2000	运动饮料	3526
11	浩天旅行社	1999	浓缩咖啡	2694
12	浩天旅行社	1999	三合一麦片	2403
13	浩天旅行社	1999	温馨奶酪	12030

示例四

SQLGroupTableBuilder 同样可以生成汇总模式，只需将 Subtotal 属性设置为 True 即可，例如：

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品ID","产品","产品ID") '添加统计表
b.AddTable("订单","客户ID","客户","客户ID") '添加统计表
b.Groups.AddDef("客户名称") '根据产品名称分组
b.Groups.AddDef("日期", DateGroupEnum.Year, "年") '根据日期按年分组
```

```
b.Groups.AddDef("产品名称") '根据产品名称分组
b.Totals.AddDef("数量") '对数量进行统计
b.Subtotal = True '生成汇总模式
b.Build '生成统计表
MainTable = Tables("统计表1") '打开生成的统计表
```

得到的统计结果为：

	客户名称	年	产品名称	数量
1	福星制衣厂股份	1999	浓缩咖啡	1822
2	福星制衣厂股份	1999	三合一麦片	1670
3	福星制衣厂股份	1999	温馨奶酪	4706
4	福星制衣厂股份	1999	盐水鸭	3349
5	福星制衣厂股份	1999	运动饮料	3093
	小计 1999			14640
6	福星制衣厂股份	2000	浓缩咖啡	1954
7	福星制衣厂股份	2000	三合一麦片	1884
8	福星制衣厂股份	2000	温馨奶酪	4696
9	福星制衣厂股份	2000	盐水鸭	2614
10	福星制衣厂股份	2000	运动饮料	3526
	小计 2000			14674
	小计 福星制衣厂股份有限公司			29314

示例五

SQLGroupTableBuilder 同样可以进行占比、环比、同比等数据分析，例如：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.Groups.AddDef("产品名称") '根据产品名称分组
b.Groups.AddDef("日期", DateGroupEnum.Year, "年") '根据日期按年分组
b.Totals.AddDef("数量") '对数量进行统计
b.CircleGrowth = True '生成环比增长率
b.GrandProportion = True '生成总站比
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计表：

	产品名称	年	数量	环比	占比
1	浓缩咖啡	1999	12210		5.79%
2	浓缩咖啡	2000	10967	-10.18%	5.20%
3	三合一麦片	1999	14448		6.85%
4	三合一麦片	2000	14207	-1.67%	6.73%
5	温馨奶酪	1999	37148		17.60%
6	温馨奶酪	2000	37012	-0.37%	17.54%
7	盐水鸭	1999	20042		9.50%
8	盐水鸭	2000	20355	1.56%	9.65%
9	运动饮料	1999	21791		10.33%
10	运动饮料	2000	22837	4.80%	10.82%

同名列的处理

在命令窗口执行下面的代码:

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品ID","产品","产品ID") '添加统计表
b.Groups.AddDef("产品ID")  '根据产品ID分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
MainTable = Tables("统计表1") '打开生成的统计表
```

你会发现Foxtable会报错，出问题的是这行代码：

```
b.Groups.AddDef("产品ID")  '根据产品ID分组
```

因为订单表和产品表，都有产品ID列，Foxtable无法判断应该用哪个表的产品ID列，所以报错。

对于这种多个表都存在的列，我们应该明确指定此列来自于哪个表，下面是正确的代码：

```
Dim b As New SQLGroupTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.Groups.AddDef("{订单}.产品 ID") '根据订单表产品 ID 分组
b.Totals.AddDef("数量") '对数量进行统计
b.Build '生成统计表
MainTable = Tables("统计表 1") '打开生成的统计表
```

注意这行代码中，表名必须用大括号括起来：

```
b.Groups.AddDef("{订单}.产品 ID") '根据订单表产品 ID 分组
```

再例如，假定产品表和订单表都有单价列，那么统计金额的时候，就必须明确指定单价列的来源表：

```
b.Totals.AddExp("金额","数量 * {订单}.单价")
```

同样在设置统计条件的时候，遇上同名列，也必须指定来源表，例如：

```
b.Filter = "{订单}.客户 ID = 'C01'"
```

4.5.7.3 SQLCrossTableBuilder

用于对后台数据进行交叉统计。
定义一个 SQLCrossTableBuilder 的语法如下：

```
New SQLCrossTableBuilder(TableName, BaseTable)
```

参数说明：

- TableName
  字符型，指定即将生成的统计表的名称

- BaseTable
  字符型，指定默认的要统计的数据表，简称为基表
  可以是未加载的数据表，只要数据源存在此表即可。

在定义 SQLCrossTableBuilder 的时候，只能指定一个统计表，也就是基表。
如果要同时针对多个表进行统计，可以使用 AddTable 方法添加表，参与统计的表相互之间必须存在某种联系。

添加统计表的语法为：

```
AddTable(Table1,Col1,Table2,Col2)
```

Table1：基表，或者已经添加的数据表，称之为左表。
Col1：　Table1 通过此列和 Table2 关联。
Table2：要添加的统计表，称之为右边。
Col2：　Table2 通过此列和 Table1 关联。

提示：

1、Table1(左表)和 Table2(右表)并非一定要事先建立关联，只需两者的数据可以通过指定的列关联起来即可。
3、如果两个表存在一对多的父子关系，那么建议将多方(子表)作为左表，一方(父表)作为右表。
3、参与统计的表可以是未加载的数据表，只要数据源存在此表即可。

例如：

```
Dim sg As New SQLCrossTableBuilder("统计表 1","订单明细")
sg.AddTable("订单明细","产品 ID","产品","产品 ID")
```

表示基表是订单明细，此外添加产品表参与统计，产品表和订单明细通过产品 ID 列关联。
由于产品表和订单表是一对多的父子关系，所以用 AddTable 添加表的时候，订单表(子表)作为左表，产品表(父表)作为右表。
如果要根据多列关联，请参考：

```
Dim b As New SQLCrossTableBuilder("统计表 1","产品")
```

```
Dim Cols1() As String = {"型号","规格"}
Dim Cols2() As String = {"型号","规格"}
b.AddTable("产品",Cols1,"订单",Cols2)
```

上面的代码，产品表作为基表，添加订单表参与统计，两个表之间通过型号、规格两列建立关联。

SQLCrossTableBuilder 包括以下属性：

ConnectionName： 指定数据源名称，如果省略，表示使用内部数据表。

Caption： 字符型，指定新生成统计表的标题。

HGroups: 集合，用于添加水平分组列。

VGroups: 集合，用于添加垂直分组列。

Totals: 集合，用于添加统计列。

OrderByTotal： 逻辑型，在有多个统计列时，是否将同一个统计列的数据排放在一起。

HorizontalTotal: 水平方向自动汇总。

VerticalTotal： 垂直方向自动汇总。

Subtotal： 逻辑型，设为 True，将对交叉统计结果进行二次统计，生成汇总模式。

Decimals： 整数型，用于设置统计结果要保留的小数位数。

HorizontalProportion： 逻辑型，是否生成水平份额

VerticalProportion： 逻辑型，是否生成垂直占比

HorizontalProportionCaption：字符型，用于设置水平份额标题，默认为“份额”

VerticalProportionCaption： 字符型，用于设置垂直占比标题，默认为“占比”

Filter: 字符型，设置统计条件。

CommandTimeOut： 整数型，用于设置统计后台数据的超时时限，默认为 30 秒，一般不需要设置此属性，除非需要进行非常耗时超过 30 秒的统计，以至于出现超时错误，此时可适当地加大 CommandTimeOut 属性的值。

添加水平分组到 HGroups 的语法是：

```
HGroups.AddDef(Name)
HGroups.AddDef(Name, NewName)
HGroups.AddDef(Name, NewName, NewCaption)
HGroups.AddDef(Name, DateGroup)
HGroups.AddDef(Name, DateGroup, NewName)
HGroups.AddDef(Name, DateGroup, NewName, NewCaption)
```

参数说明

Name: 字符型，指定分组列。

NewName: 字符型，用于给分组列指定一个新的名称，这个名称在统计表中显示。

NewCaption: 字符型，用于给分组列设置标题。

DateGroup： DateGroupEnum 型枚举，如果分组列是日期列，可用此属性指定如何根据日期进行分组，默认按月分组。

DateGroupEnum 枚举的可选值：

None ： 不分组

Year： 按年分组

Quarter： 按季度分组

Month： 按月分组

Week： 按周分组

Day：　　按天分组，月中的某一天。
Hour：　　按小时分组
Minute：　　按分钟分组 Second：　　按秒分组

可以向 HGroups 增加表达式分组列，语法为：

HGroups.AddExp(Name, Expression)
HGroups.AddExp(Name, Expression, Caption)

参数说明：

Name：　　字符型，指定列名。
Expression：字符型，指定表达式。
Caption：　　字符型，指定标题。

添加垂直分组到 VGroups 中的语法如下：

VGroups.AddDef(Name)
VGroups.AddDef(Name, Pattern)
VGroups.AddDef(Name, DateGroup)
VGroups.AddDef(Name, DateGroup, Pattern)

参数说明
Name:　字符型，指定分组列。
Pattern:　指定生成标题的模式，用字符"{0}"表示分组值；假定分组值是 10，模式是"第{0}周"，生成的分组标题将是"第 10 周"。
DateGroup：DateGroupEnum 枚举型，如果分组列是日期列，可用此属性指定如何根据日期进行分组，默认按月分组。
可以向 VGroups 增加表达式分组列，语法为：
VGroups.AddExp(Name, Expression)
VGroups.AddExp(Name, Expression，Pattern)
参数说明：
Name：　字符型，指定列名。
Expression：字符型，指定表达式。
Pattern:　指定生成标题的模式，用字符"{0}"表示分组值；假定分组值是 10，模式是"第{0}周"，生成的分组标题将是"第 10 周"

添加统计列到 Totals 集合中的语法是：

Totals.AddDef(Name)
Totals.AddDef(Name, Caption)
Totals.AddDef(Name, Aggregate)
Totals.AddDef(Name, Aggregate,Caption)

参数说明
Name:　字符型，指定统计列。
Caption：　字符型，指定统计列的标题。
Aggregate: AggregateEnum 枚举型，用于指定统计类型，默认是求和。

AggregateEnum 型枚举的可选值：

Average：平均值
Count：　记录数
Max：　　最大值
Min：　　最小值
Sum：　　累积值

在 Exprs 添加一个表达式统计列的语法是：

Totals.AddExp(Name, Expression)
Totals.AddExp(Name, Expression, Caption)
Totals.AddExp(Name, Expression, Aggregate)
Totals.AddExp(Name, Expression, Caption, Aggregate)
Name：　字符型，指定列名。
Expression：字符型，指定表达式。
Caption：　字符型，指定列标题。
Aggregate：AggregateEnum 枚举型，用于指定统计类型，默认是求和。
例如：

```
Dim b As New SQLCrossTableBuilder("统计表 1","订单")
b.Exprs.Adddef("金额","[数量] * [单价] * (1-[折扣])")
```

可以看出 SQLCrossTableBuilder 和 CrossTableBuilder 的使用方法基本相似，前者只是在后者的基础上增加了：
1、可以用 ConnectionName 指定数据源
2、可以用 AddTable 方法增加多个数据表
3、可以用 AddExp 方法增加表达式分组列或统计列
提示：由于 SQLCrossTableBuilder 是根据用户的设置，自动生成 SQL 语句进行统计， 所以添加表或列时，名称必须和后台数据库保持一致，此外条件表达式的语法和函数与采用的数据源有关，我们会在《SQL 相关》这章对此进行专门的介绍。

统计单个数据表

在运行下面的示例之前，请打开 CaseStudy 目录下的文件：统计演示.Table

示例一

普通的统计，SQLCrossTableBuilder 和 CrossTableBuilder 的用法基本一样。
例如统计同一客户订购不同产品的数量：

```
Dim b As New SQLCrossTableBuilder("统计表 1","订单")
b.HGroups.AddDef("客户") '添加客户列用于水平分组
b.VGroups.AddDef("产品","产品_{0}") '添加产品列用于垂直分组,并设置了 Pattern 参数
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果：

	客户	产品				
		PD01	PD02	PD03	PD04	PD05
1	CS01	260	2250	2180	230	600
2	CS02	4120	4300	2250	1830	2550
3	CS03	1980	6300	970	1200	2420
4	CS04	3390	3110	700	1410	3350
5	CS05	1540	2240	900	810	1480

示例二

SQLCrossTableBuilder 可以添加表达式统计列，例如：

```
Dim b As New SQLCrossTableBuilder("统计表 1","订单")
b.HGroups.AddDef("客户")
b.VGroups.AddDef("产品")
b.Totals.AddDef("数量")
b.Totals.AddExp("金额","数量 * 单价 *(1 - 折扣)")'
b.Build
Maintable = Tables("统计表 1")
```

	客户	PD01		PD02		PD03		PD04		PD05	
		数量	金额	数量	金额	数量	金额	数量	金额	数量	金额
1	CS01	260	3777.12	2250	34814.6	2180	17720	230	3951.2	600	12338.7
2	CS02	4120	60292.8	4300	64634.2	2250	18940	1830	32414.8	2550	45418.05
3	CS03	1980	28638	6300	106352.5	970	9680	1200	23320	2420	43467.75
4	CS04	3390	52155	3110	52706	700	6550	1410	27280	3350	66256
5	CS05	1540	26892	2240	38015.2	900	8600	810	15895	1480	25542.65

示例三

不仅是统计列，水平分组列和垂直分组列也可以是表达式：

```
Dim b As New SQLCrossTableBuilder("统计表1","订单")
b.HGroups.AddExp("月", "Month(日期)")
b.VGroups.AddExp("年", "Year(日期)")
b.Totals.AddDef("数量")
b.Totals.AddExp("金额","数量 * 单价") '
b.Build
MainTable = Tables("统计表 1")
```

注意这两行代码：

```
b.HGroups.AddExp("月", "Month(日期)")
b.VGroups.AddExp("年", "Year(日期)")
```

等效于：

```
b.HGroups.AddDef("日期",DateGroupEnum.Month,"月")
b.VGroups.AddDef("日期",DateGroupEnum.Year)
```

统计多个数据表

在运行下面的示例之前，请打开 CaseStudy 目录下的文件：多表统计.Table

这个文件的所有表都来自外部数据源，外部数据源的名称为 Sale，外部数据源是一个 Access 文件：

多表统计.mdb

整个系统包括三个表，分别是客户、产品和订单：

	产品ID	产品名称
1	P01	运动饮料
2	P02	温馨奶酪
3	P03	三合一麦片
4	P04	浓缩咖啡
5	P05	盐水鸭

产品表

	客户ID	客户名称
1	C01	红阳事业
2	C02	威航货运有限公司
3	C03	浩天旅行社
4	C04	立日股份有限公司
5	C05	福星制衣厂股份有限公司

客户表

	产品ID	客户ID	单价	折扣	数量	日期
1	P05	C03	17	0.1	690	2000-01-02
2	P02	C01	20	0.1	414	2000-01-03
3	P05	C04	17	0	332	2000-01-03
4	P01	C03	18	0.15	-9	2000-01-04
5	P03	C03	10	0	445	2000-01-04
6	P04	C04	17.6	0	246	2000-01-07

订单表

示例一

假定要统计同一客户订购不同产品的数量，由于订单表没有产品名称和客户名称列，如果只根据订单表生成统计表，只能显示产品 ID 和客户 ID，显然这样的统计表太不友好了，毕竟没有人会记得每个 ID 对应的 具体名称是什么。

所以我们需要从产品表提取产品名称，从客户表提取客户名称列：

```
Dim b As New SQLCrossTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.AddTable("订单","客户 ID","客户","客户 ID") '添加统计表
b.HGroups.AddDef("客户名称") '添加客户列用于水平分组
b.VGroups.AddDef("产品名称","产品名称_{0}") '添加产品列用于垂直分组,并设置了 Pattern 参数
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果：

	客户名称	产品名称				
		浓缩咖啡	三合一麦片	温馨奶酪	盐水鸭	运动饮料
1	福星制衣厂	3776	3554	9402	5963	6619
2	浩天旅行社	5223	4301	24944	9038	7494
3	红阳事业	1241	9072	9407	2459	1100
4	立日股份有	5592	2738	13287	12934	13630
5	威航货运有	7345	8990	17120	10003	15785

在使用外部数据源的时候，一定要记得设置 ConnectionName 属性：

```
b.ConnectionName = "数据源名称"
```

示例二

例如要按年统计同一客户订购不同产品的数量：

```
Dim b As New SQLCrossTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.AddTable("订单","客户 ID","客户","客户 ID") '添加统计表
b.HGroups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于垂直分组,按年分组
b.HGroups.AddDef("产品名称") '添加产品列用于垂直分组
b.VGroups.AddDef("客户名称","客户_{0}") '添加客户列用于水平分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果：

	年	产品名称	客户				
			福星制衣厂	浩天旅行社	红阳事业	立日股份有	威航货运有
1	1999	浓缩咖啡	1822	2694	602	3215	3877
2	1999	三合一麦片	1670	2403	4325	1449	4601
3	1999	温馨奶酪	4706	12030	4700	7189	8523
4	1999	盐水鸭	3349	4414	1087	6631	4561
5	1999	运动饮料	3093	3472	654	7040	7532
6	2000	浓缩咖啡	1954	2529	639	2377	3468
7	2000	三合一麦片	1884	1898	4747	1289	4389
8	2000	温馨奶酪	4696	12914	4707	6098	8597
9	2000	盐水鸭	2614	4624	1372	6303	5442
10	2000	运动饮料	3526	4022	446	6590	8253

示例三

SQLCrossTableBuilder 同样可以生成汇总模式，只需将 Subtotal 属性设置为 True 即可，例如：

```
Dim b As New SQLCrossTableBuilder("统计表1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品ID","产品","产品ID") '添加统计表
b.AddTable("订单","客户ID","客户","客户ID") '添加统计表
b.HGroups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于垂直分组,按年分组
b.HGroups.AddDef("产品名称") '添加产品列用于垂直分组
b.VGroups.AddDef("客户名称","客户_{0}") '添加客户列用于水平分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.Subtotal = True '生成汇总模式
b.Build '生成统计表
MainTable = Tables("统计表1") '打开生成的统计表
```

得到的统计结果为：

	年	产品名称	客户				
			福星制衣厂	浩天旅行社	红阳事业	立日股份有	威航货运有
1	1999	浓缩咖啡	1822	2694	602	3215	3877
2	1999	三合一麦片	1670	2403	4325	1449	4601
3	1999	温馨奶酪	4706	12030	4700	7189	8523
4	1999	盐水鸭	3349	4414	1087	6631	4561
5	1999	运动饮料	3093	3472	654	7040	7532
	小计 1999		14640	25013	11368	25524	29094
6	2000	浓缩咖啡	1954	2529	639	2377	3468
7	2000	三合一麦片	1884	1898	4747	1289	4389
8	2000	温馨奶酪	4696	12914	4707	6098	8597
9	2000	盐水鸭	2614	4624	1372	6303	5442
10	2000	运动饮料	3526	4022	446	6590	8253
	小计 2000		14674	25987	11911	22657	30149
	总计		29314	51000	23279	48181	59243

示例四

SQLCrossTableBuilder 同样可以进行垂直占比和水平份额的分析：

```
Dim b As New SQLCrossTableBuilder("统计表 1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品 ID","产品","产品 ID") '添加统计表
b.AddTable("订单","客户 ID","客户","客户 ID") '添加统计表
b.HGroups.AddDef("日期",DateGroupEnum.Year,"年") '添加日期列用于垂直分组,按年分组
b.HGroups.AddDef("产品名称") '添加产品列用于垂直分组
b.VGroups.AddDef("客户名称","{0}_数量") '添加客户列用于水平分组
b.Totals.AddDef("数量") '添加数量列用于统计
b.HorizontalProportion = True '生成水平份额
b.Build '生成统计表
Maintable = Tables("统计表 1") '打开生成的统计表
```

生成的统计结果：

	年	产品名称	福星制衣厂股份有限公		浩天旅行社		红阳事业	
			数量	份额	数量	份额	数量	份额
1	1999	浓缩咖啡	1822	14.92%	2694	22.06%	602	4.93%
2	1999	三合一麦片	1670	11.56%	2403	16.63%	4325	29.93%
3	1999	温馨奶酪	4706	12.67%	12030	32.38%	4700	12.65%
4	1999	盐水鸭	3349	16.71%	4414	22.02%	1087	5.42%
5	1999	运动饮料	3093	14.19%	3472	15.93%	654	3.00%
6	2000	浓缩咖啡	1954	17.82%	2529	23.06%	639	5.83%
7	2000	三合一麦片	1884	13.26%	1898	13.36%	4747	33.41%
8	2000	温馨奶酪	4696	12.69%	12914	34.89%	4707	12.72%
9	2000	盐水鸭	2614	12.84%	4624	22.72%	1372	6.74%
10	2000	运动饮料	3526	15.44%	4022	17.61%	446	1.95%

同名列的处理

在命令窗口执行下面的代码:

```
Dim b As New SQLCrossTableBuilder("统计表1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品ID","产品","产品ID")
```

```
b.HGroups.AddDef("产品ID")
b.VGroups.Adddef("日期",DateGroupEnum.Year,"年")
b.Totals.AddDef("数量")
b.Build()
MainTable = Tables("统计表1")
```

你会发现Foxtable会报错，出问题的是这行代码：

```
b.HGroups.AddDef("产品ID")
```

因为订单表和产品表，都有产品ID列，Foxtable无法判断应该用哪个表的产品ID列，所以报错。对于这种多个表都存在的列，我们应该明确指定此列来自于哪个表，下面是正确的代码：

```
Dim b As New SQLCrossTableBuilder("统计表1","订单")
b.ConnectionName = "Sale"
b.AddTable("订单","产品ID","产品","产品ID")
b.HGroups.AddDef("{订单}.产品ID")
b.VGroups.Adddef("日期",DateGroupEnum.Year,"年")
b.Totals.AddDef("数量")
b.Build
MainTable = Tables("统计表 1")
```

注意这行代码中，表名必须用大括号括起来：

```
b.HGroups.AddDef("{订单}.产品ID")
```

再例如，假定产品表和订单表都有单价列，那么统计金额的时候，就必须明确指定单价列的来源表：

```
b.Totals.AddExp("金额","数量 * {订单}.单价")
```

同样在设置统计条件的时候，遇上同名列，也必须指定来源表，例如：

```
b.Filter = "{订单}.客户 ID = 'C01'"
```

4.5.7.4 灵活运用自定义表达式

由于 SQLGroupTableBuilder 和 SQLCrossTableBuilder 可以自定义表达式，可以给我们的统计带来额外的灵活性。

示例一

例如有下图所示的一个名单表：

	姓名	居民组	签约期
1	张琳	第01小组	2015-04-29
2	魏晨	第02小组	
3	马可	第03小组	
4	马雪阳	第02小组	
5	孔融	第02小组	2015-04-23
6	袁文康	第01小组	
7	张曼玉	第02小组	
8	叶全真	第02小组	
9	玉米提	第03小组	2015-04-07
10	安琥	第02小组	
11	右小祖	第02小组	
12	张勋杰	第01小组	

如果要得到每个小组已经签约和未签约数量，如下图：

	居民组	总人数	已签约	未签约
1	第01小组	8	1	7
2	第02小组	16	4	12
3	第03小组	6	3	3

用 GroupTableBuilder 是没有办法完成这样的统计的，但是用 SQLGroupTableBuilder 却很简单：

```
Dim b As New SQLGroupTableBuilder("统计表1","名单")
b.Groups.Adddef("居民组")
b.Totals.Adddef("姓名",AggregateEnum.Count,"总人数")
b.Totals.Addexp("已签约","iif(签约期  Is null,0,1)")
b.Totals.Addexp("未签约","iif(签约期  Is null,1,0)")
b.Build
MainTable = Tables("统计表1")
```

示例二

假定订单表的结构如下：

	产品	客户	雇员	单价	折扣	数量	金额	日期	已付款
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02	☑
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03	☑
3	PD05	CS04	EP02	17	0	320	5440	1999-01-03	☑
4	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04	☑
5	PD03	CS03	EP04	10	0	490	4900	1999-01-04	☑
6	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07	☐
7	PD05	CS03	EP04	17	0	111	1887	1999-01-07	☑
8	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08	☐
9	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08	☑
10	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10	☑
11	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10	☑

如果要分别统计出每个客户的已付款金额和未付款金额，以及总的金额：

	客户	已付款	未付款	合计
1	CS01	46980.92	25620.7	72601.62
2	CS02	130192.6	91507.25	221699.85
3	CS03	139689.25	70936	210625.25
4	CS04	138665.5	66281.5	204947
5	CS05	89570.45	25374.4	114944.85

同样只能用 SQLGroupTableBuilder 完成：

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.Groups.Adddef("客户")
b.Totals.Addexp("已付款","iif(已付款 = True,数量 * 单价 * (1-折扣),0)")
b.Totals.Addexp("未付款","iif(已付款 <> True,数量 * 单价 * (1-折扣),0)")
b.Totals.Addexp("合计","数量 * 单价 * (1-折扣)")
b.Build()
MainTable = Tables("统计表1")
```

需要注意的是，如果你的数据源是SQL Server，那么上面的代码是无法执行的，因为这是后台统计，采用的是后台数据源的函数，而SQL Server:

1、逻辑值不用False和True表示，用0和1表示。
2、没有iif函数,只能用Case When语句表示。

这些差异会在后面的《SQL 相关》这一章介绍，目前可以有所了解。
基于上面的原因，如果是 SQL Server 数据源，代码需要改为：

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.ConnectionName = "数据源名称"
b.Groups.Adddef("客户")
b.Totals.Addexp("已付款", "Case When 已付款 = 1 Then 数量 * 单价 * (1-折扣) Else 0 End")
b.Totals.Addexp("未付款", "Case When 已付款 <> 1 Then 数量 * 单价 * (1-折扣) Else 0 End")
b.Totals.Addexp("合计","数量 * 单价 * (1-折扣)")
b.Build()
MainTable = Tables("统计表1")
```

关于 SQL Server 和 Access 的语法差异，帮助文件《SQL 相关》这一章会有较为详细的介绍。

4.5.7.5 非自然月的统计

部分企业在统计数据的时候，并不使用自然月，例如有的从上月的 26 日到本月的 25 日算做一个月，如何完成种非自然月的统计呢？
我们下面介绍两个两个方法。

增加一个统计日期列

我们可以在表中再增加一个统计日期列，然后将表的 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.name
    Case "日期"
        If e.NewValue Is Nothing Then
            e.DataRow("统计日期") = Nothing
        Else
            Dim dt As Date = e.NewValue
            If dt.Day > 25 Then '如果为25日以后的日期
                dt = new Date(dt.year,dt.month,1) '改为下个月的第一天
                dt= dt.AddMonths(1)
            End If
            e.DataRow("统计日期") = dt
        End If
End Select
```

之后我们根据这个新增加的统计日期列进行统计即可。

利用自定义表达式

如果使用 SQLGroupTableBuilder 或 SQLCrossTableBuilder 进行统计，那么我们无需额外增加一列，可以直接用自定义表达式作为分组，实现非自然月的统计。

例如按月统计销售数量和金额：

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.Groups.AddExp("年","Year(iif(Day(日期)> 25,DateAdd('d',6,日期),日期))")
b.Groups.AddExp("月","Month(iif(Day(日期)> 25,DateAdd('d',6,日期),日期))")
b.Totals.AddDef("数量")
b.Totals.AddExp("金额","数量 * 单价")
b.Build()
MainTable = Tables("统计表1")
```

上述代码利用表达式，如果如果是 25 日之后的日期，则将此日期增加 6 天，使其成为下个月的日期，然后再取年月参与统计。

需要注意的是，SQL Server 的数据源并不支持 iif 函数，需要用 Case When 语句代替，此外 DateAdd 的第一个参数也不需要用单引号括起来，所以代码需要改为：

```
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.ConnectionName = "数据源名称"
b.Groups.AddExp("年","Year(Case When Day(日期) > 25 Then DateAdd(d,6,日期) Else 日期 End)")
b.Groups.AddExp("月","Month(Case When Day(日期) > 25 Then DateAdd(d,6,日期) Else 日期 End)")
b.Totals.AddDef("数量")
b.Totals.AddExp("金额","数量 * 单价")
b.Build()
MainTable = Tables("统计表1")
```

关于 Access 和 SQL Server 的语法和函数差异，我们的《SQL 相关》这一章，有较为详细的介绍。

4.5.7.6 数值分布统计的方法

统计数值的分布范围，是很常见的一种统计。

示例一

假定有下图所示的成绩表：

	班级	姓名	科目	分数
1	1班	陈成进	语文	133
2	1班	林景阳	语文	129
3	1班	苏　捷	语文	132
4	1班	谢晓霞	语文	129
5	2班	谢杰立	语文	129
6	2班	谢晓霞	语文	134
7	2班	杨景亮	语文	132
8	2班	张李华	语文	133
9	4班	谢杰立	语文	129
10	4班	谢晓霞	语文	134
11	4班	杨景亮	语文	132
12	4班	张李华	语文	133
13	1班	陈成进	数学	136
14	1班	苏　捷	数学	136
15	1班	谢杰立	数学	133
16	2班	林景阳	数学	133
17	2班	杨景亮	数学	136
18	4班	林景阳	数学	133

我们需要统计每一个科目的成绩分布范围，例如 60 分以下，60 分以上，70 分以上等等范围的人数，以及各个科目的最高分、最低分、平均分等：

	科目	最高分	最低分	平均分	分布									
					60以下	60-70	70-80	80-90	90-100	100-110	110-120	120-130	130-140	140以上
1	化学	100	18	77.3636	5	4	10	15	9	1	0	0	0	0
2	数学	136	38	104.2045	6	0	3	4	2	3	10	9	7	0
3	物理	93	36	74.1591	9	1	17	14	3	0	0	0	0	0
4	英语	146	74	117.9091	0	0	1	0	10	5	5	6	10	7
5	语文	134	100	119.1364	0	0	0	0	0	5	21	10	8	0

采用自定义表达式，可以方便地统计出来：

```
Dim b As New SQLGroupTableBuilder("统计表 1","学生成绩")
b.Groups.AddDef("科目")
b.Totals.AddDef("分数",AggregateEnum.Max,"最高分")
b.Totals.AddDef("分数",AggregateEnum.Min,"最低分")
b.Totals.AddDef("分数",AggregateEnum.Average,"平均分")
b.Totals.AddExp("分布_60 以下","iif(分数 < 60,1,0)")
b.Totals.AddExp("分布_60-70","iif(分数 >= 60 And 分数 < 70,1,0)")
b.Totals.AddExp("分布_70-80","iif(分数 >= 70 And 分数 < 80,1,0)")
b.Totals.AddExp("分布_80-90","iif(分数 >= 80 And 分数 < 90,1,0)")
b.Totals.AddExp("分布_90-100","iif(分数 >= 90 And 分数 < 100,1,0)")
b.Totals.AddExp("分布_100-110","iif(分数 >= 100 And 分数 < 110,1,0)")
b.Totals.AddExp("分布_110-120","iif(分数 >= 110 And 分数 < 120,1,0)")
```

```
b.Totals.AddExp("分布_120-130","iif(分数 >= 120 And 分数 < 130,1,0)")
b.Totals.AddExp("分布_130-140","iif(分数 >= 130 And 分数 < 140,1,0)")
b.Totals.AddExp("分布_140 以上","iif(分数 >= 140,1,0)")
b.build()
MainTable = Tables("统计表 1")
```

示例二

本示例的内容涉及到 SQL 语句的知识，帮助文件《SQL 相关》这章有具体介绍，如果好没有学习到 SQL 的知识，可以暂时忽略本示例。

实际上多数的原始成绩表，并非我们前面介绍的格式，而是下图这样的格式:

	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136	140	71	75	555
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529
11	1班	张李华	124	118	124	83	88	537
12	2班	陈成进	110	58	106	36	34	344
13	2班	陈诗淇	113	123	117	78	78	509
14	2班	陈思达	113	55	92	55	46	361

由于这个表本身是一个交叉表，用常规的统计工具我们无法直接得到下面的统计表，除非手工编码进行统计：

	科目	最高分	最低分	平均分	分布								
					60分以下	60分以上	70分以上	80分以上	90分以上	100分以上	110分以上	120分以上	130分以上
1	语文	134	100	119	0	0	0	0	0	5	21	10	8
2	英语	146	74	117	0	0	1	0	10	5	5	6	17
3	数学	136	38	104	6	0	3	4	2	3	10	9	7
4	物理	93	36	74	9	1	17	14	3	0	0	0	0
5	化学	100	18	77	5	4	10	15	9	1	0	0	0

不过，如果我们能够通过 SQL 语句将原始成绩表转转为示例一的那种格式：

	班级	姓名	科目	分数
1	1班	陈成进	语文	133
2	1班	林景阳	语文	129
3	1班	苏　捷	语文	132
4	1班	谢晓霞	语文	129
5	2班	谢杰立	语文	129
6	2班	谢晓霞	语文	134
7	2班	杨景亮	语文	132
8	2班	张李华	语文	133
9	4班	谢杰立	语文	129
10	4班	谢晓霞	语文	134
11	4班	杨景亮	语文	132
12	4班	张李华	语文	133
13	1班	陈成进	数学	136
14	1班	苏　捷	数学	136
15	1班	谢杰立	数学	133
16	2班	林景阳	数学	133
17	2班	杨景亮	数学	136
18	4班	林景阳	数学	133

那么就可以直接用统计工具进行统计了，关于这种转换，可以参考：水平表和垂直表的相互转换。

由于 SQLGroupTableBuilder 和 SQLCrossTableBuilder，都可以直接对 SQL 语句进行统计，所以我们并不需要对转换后的表进行统计（这样会影响效率），只需将合成的 SQL 语句传递给 SQLGroupTableBuilder 即可，具体代码：

```
Dim nms() As String = {"语文","数学","英语","化学","物理"}
Dim sql As String
For Each nm As String In nms
    If sql  > "" Then
        sql = sql & " Union All "
    End If
    sql = sql & "Select 班级,姓名,'" & nm & "' As 科目," & nm & " As 分数 From {学生成绩}"
Next
Dim b As New SQLGroupTableBuilder("统计表1",sql,"")
b.Groups.AddDef("科目")
b.Totals.AddDef("分数",AggregateEnum.Max,"最高分")
b.Totals.AddDef("分数",AggregateEnum.Min,"最低分")
b.Totals.AddDef("分数",AggregateEnum.Average,"平均分")
b.Totals.AddExp("分布_60以下","iif(分数 < 60,1,0)")
b.Totals.AddExp("分布_60-70","iif(分数 >= 60 And 分数 < 70,1,0)")
b.Totals.AddExp("分布_70-80","iif(分数 >= 70 And 分数 < 80,1,0)")
b.Totals.AddExp("分布_80-90","iif(分数 >= 80 And 分数 < 90,1,0)")
b.Totals.AddExp("分布_90-100","iif(分数 >= 90 And 分数 < 100,1,0)")
b.Totals.AddExp("分布_100-110","iif(分数 >= 100 And 分数 < 110,1,0)")
b.Totals.AddExp("分布_110-120","iif(分数 >= 110 And 分数 < 120,1,0)")
```

```
b.Totals.AddExp("分布_120-130","iif(分数  >= 120 And  分数  < 130,1,0)")
b.Totals.AddExp("分布_130-140","iif(分数  >= 130 And  分数  < 140,1,0)")
b.Totals.AddExp("分布_140以上","iif(分数  >= 140,1,0)")
b.build()
MainTable = Tables("统计表1")
```

提醒：SQL Server没有iif函数，需要用Case When语句代替，上一节已经用具体的例子说明了，这里就不再重复了。

4.5.7.7 一个经典的统计示例

假定有下图的成绩表：

成绩表 统计表1

	班级	学号	姓名	语文	数学	英语	物理	化学
1	1	121	陈锦标	132	147	145	97	97
2	2	256	伍景珠	134	140	142	94	96
3	1	110	邓玉婷	132	136	146	93	94
4	2	211	李晓燕	137	140	136	87	90
5	1	140	何列琦	120	140	139	98	90
6	2	227	林璐韵	132	128	139	90	95
7	1	136	冯　勃	135	141	129	94	91
8	2	253	蒋广裕	131	150	122	96	90
9	2	206	黄　图	116	147	141	92	90
10	1	128	叶卜源	135	125	144	82	91
11	1	124	何思敏	129	143	134	80	84
12	2	235	黄秀东	135	134	127	87	94
13	2	240	蒋春泽	133	126	134	98	85
14	1	158	宋妙倩	130	122	143	86	87
15	2	213	徐思远	124	150	122	86	95
16	2	212	陈春花	126	138	122	89	94
17	1	144	罗　迪	127	133	137	85	90
18	1	109	陈诗淇	129	118	139	88	80
19	1	152	吴晓梅	127	131	142	70	87

希望按班级统计每科的及格人数、及格率、优秀人数和优秀率：

成绩表 统计表1

	班级	人数	语文				英语				数学				物理				化学			
			及格	及格率	优秀	优秀率	及格	及格率	优秀	优秀率	及格	及格率	优秀	优秀率	及格	及格率	优秀	优秀率	及格	及格率	优秀	优秀率
1	1	65	62	95.38%	60	92.31%	61	93.85%	55	84.62%	56	86.15%	49	75.38%	44	67.69%	5	7.69%	49	75.38%	10	15.38%
2	2	62	59	95.16%	57	91.94%	51	82.26%	46	74.19%	57	91.94%	46	74.19%	41	66.13%	9	14.52%	41	66.13%	13	20.97%
3	合计	127	121	95.28%	117	92.13%	112	88.19%	101	79.53%	113	88.98%	95	74.80%	85	66.93%	14	11.02%	90	70.87%	23	18.11%

这是一个比较经典的统计示例，需要灵活运用我们之前掌握的多种知识，但代码并不复杂：

```
Dim b As New SQLGroupTableBuilder("统计表1","成绩表")
b.Groups.AddDef("班级")
b.Totals.AddExp("人数","学号","",AggregateEnum.Count) '统计人数
Dim nms() As String = {"语文","英语","数学","物理","化学"}
For Each nm As String In nms '利用表达式统计及格人数和优秀人数
    b.Totals.Addexp(nm & "_及格", "iif(" &  nm & " >= 60, 1, 0)")
```

```
    b.Totals.Addexp(nm & "_优秀", "iif(" &  nm & " >= 90, 1, 0)")
Next
b.VerticalTotal = True
b.Build()
For Each nm As String In nms '对生成的统计表增加表达式列,计算及格率和优秀率
    DataTables("统计表1").DataCols.Add(nm & "_及格率",Gettype(Double), nm & "_及格/人数" )
    Tables("统计表1").Cols(nm & "_及格率").Move(Tables("统计表1").Cols(nm & "_及格").Index + 1)
    DataTables("统计表1").DataCols.Add(nm & "_优秀率",Gettype(Double), nm & "_优秀/人数" )
    Tables("统计表1").Cols(nm & "_优秀率").Move(Tables("统计表1").Cols(nm & "_优秀").Index + 1)
    DataTables("统计表1").DataCols(nm & "_及格率").SetFormat("0.00%")
    DataTables("统计表1").DataCols(nm & "_优秀率").SetFormat("0.00%")
Next
MainTable = Tables("统计表1")
```

4.5.8 查询表生成器

SQLGroupTableBuilder 和 SQLCrossTableBuilder 可以从多个相关表中提取数据进行统计。
有的时候，我们并不想进行统计，只是希望从多个表中提取相关列组合成一个查询表而已。
当然我们可以用 Select 语句完成任务，但是通晓 SQL 语句的用户毕竟是少数，所以这样的任务这对于多数人来说，可不那么容易实现。
Foxtablet 提供了一个查询表生成器：SQLJoinTableBuilder，无需编写任何 SQL 语句，即可从多个表中提取相关列组合成一个查询表。

定义一个 SQLJoinTableBuilder 的语法如下：

New SQLJoinTableBuilder(TableName, BaseTable)

参数说明：

- TableName
  字符型，指定即将生成的查询表的名称

- BaseTable
  字符型，指定用于生成查询表的第一个数据表，我们称之为基表

既然要从多个表中提取数据，除基表之外，我们还需要添加关联表。

添加关联表的语法为：

AddTable(Table1,Col1,Table2,Col2)

Table1：基表，或者已经添加的数据表 ，称之为左表。
Col1： Table1 通过此列和 Table2 关联。
Table2：要添加的关联表，称之为右表。
Col2： Table2 通过此列和 Table1 关联。

提示：

1、左表(Table1)和右表（Table2）并非一定要事先建立关联，只需两者的数据可以通过指定的列关联起来即可。
3、如果两个表存在一对多的父子关系，那么建议将多方(子表)作为左表，一方(父表)作为右表。
3、不管是基表，还是关联表，都可以是未加载的数据表，只要数据源存在此表即可。

例如：

```
Dim jb As New SQLJoinTableBuilder("统计表 1","订单")
jb.AddTable("订单","产品 ID","产品","产品 ID")
```

表示基表是订单，此外添加产品表参与查询，产品表和订单表通过产品 ID 列关联起来。
由于产品表和订单表是一对多的父子关系，所以用 AddTable 添加表的时候，订单表(子表)作为左表，产品表(父表)作为右表。
如果要根据多列关联，请参考：

```
Dim jb As New SQLJoinTableBuilder("统计表 1","产品")
Dim Cols1() As String = {"型号","规格"}
Dim Cols2() As String = {"型号","规格"}
jb.AddTable("产品",Cols1,"订单",Cols2)
```

上面的代码，产品表作为基表，添加订单表参与统计，两个表之间通过型号、规格两列关联起来。
添加了数据表之后，还要用 AddCols 方法来给查询表添加列，AddCols 的语法为：
AddCols(Col1, Col2, Col3...)
Col1，Col2，Col3：要添加的多列。
不管列是来自于基表，还是来自关联表，都需要用 AddCols 方法加入。
如果存在同名列，添加列的时候，必须指定列的来源表，例如

```
Dim jb As New SQLJoinTableBuilder("查询表 1","订单")
jb.AddTable("订单","产品 ID","产品","产品 ID")
jb.AddCols("{订单}.产品 ID","产品名称","数量","单价")
```

由于产品 ID 在产品表和订单表都存在，所以必须指定其来源表，而产品名称只存在于产品表，数量和单价只存在于订单表，所以不需要指定来源表。
注意指定来源表的时候，表名必须用大括号括起来,例如：{订单}.产品 ID
如果要将某个表所有列加入查询表，可以用这样的格式：{表名).*
例如：

```
Dim jb As New SQLJoinTableBuilder("查询表 1","订单")
jb.AddTable("订单","产品 ID","产品","产品 ID")
jb.AddCols("产品名称", "{订单}.*")
```

表示将产品表的产品名称列及订单表的所有列加入到查询表中。
我们还可以用 AddExp 方法添加表达式列，语法为：
AddExp(Name, Expression)
Name:　　列名
Expression：表达式
例如：

```
jb.AddExp("金额", "数量 * 单价")
```

表示给查询表添加一个名为“金额”的列，此列通过数量和单价两列计算得出，计算公式为：[数量] * [单价]

同样参与表达式计算的列，如果存在对多个表中，必须指定列的来源表。
例如，假定产品表和订单表都存在单价列，那么就必须如下设置金额的表达式：

```
jb.AddExp("金额", "数量 * {订单}.单价")
```

注意表名要用大括号括起来。

利用 AddExp 方法，可以间接实现给列改名的功能，例如：
jb.AddExp("品名", "产品名称")
表示添加一个名为“品名”的表达式列，其表达式为“产品名称”，这等于就是将“产品名称”列添加到查询表，并改名为“品名”。
SQLJoinTableBuilder 的属性有：
ConnectionName： 指定数据源名称，如果省略，表示使用内部数据表。
Caption： 字符型，指定新生成查询表的标题。
Filter: 字符型，设置统计条件。
Order： 字符型，指定排序列。
Top： 整数型，指定要加载的行数。
Distinct: 逻辑型，是否排除重复值。
CommandTimeOut： 整数型，用于设置统计后台数据的超时时限，默认为 30 秒，一般不需要设置此属性，除非需要进行非常耗时超过 30 秒的统计，以至于出现超时错误，此时可适当地加大 CommandTimeOut 属性的值。
提示：
1、例如你有 100 万甚至更多的行，那么用 SQLJoinTableBuilder 生成查询表的唯一后果就是计算机崩溃，所以数据量大的时候，记得设置 SQLJoinTableBuilder 的 Filter 属性。
2、添加表达式列或设置统计条件时，具体语法和所采用的数据源有关，我们会在 SQL 章节对此进行具体介绍。
接下来我们来实战演练一下 SQLJoinTableBuilder，请先打开 CaseStudy 目录下的文件：多表统计.Table。
这个文件的所有表都来自外部数据源，外部数据源的名称为 Sale，外部数据源是一个 Access 文件：多表统计.mdb
整个系统包括三个表，分别是客户、产品和订单：

	产品ID	产品名称
1	P01	运动饮料
2	P02	温馨奶酪
3	P03	三合一麦片
4	P04	浓缩咖啡
5	P05	盐水鸭

产品表

	客户ID	客户名称
1	C01	红阳事业
2	C02	威航货运有限公司
3	C03	浩天旅行社
4	C04	立日股份有限公司
5	C05	福星制衣厂股份有限公司

客户表

	产品ID	客户ID	单价	折扣	数量	日期
1	P05	C03	17	0.1	690	2000-01-02
2	P02	C01	20	0.1	414	2000-01-03
3	P05	C04	17	0	332	2000-01-03
4	P01	C03	18	0.15	-9	2000-01-04
5	P03	C03	10	0	445	2000-01-04
6	P04	C04	17.6	0	246	2000-01-07

订单表

可以直接在命令窗口执行下面的示例代码。

示例一

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddCols("产品名称", "日期", "数量","单价")
jb.Build()
MainTable = Tables("查询表1")
```

生成的查询表如下图，其中产品名称列来自于产品表，日期、数量和单价三列来自于订单表：

	产品名称	日期	数量	单价
1	盐水鸭	1999-01-02	753	17
2	温馨奶酪	1999-01-03	348	20
3	盐水鸭	1999-01-03	305	17
4	运动饮料	1999-01-04	202	18
5	三合一麦片	1999-01-04	492	10
6	盐水鸭	1999-01-07	144	17

示例二

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单", "客户ID","客户","客户ID")
jb.AddCols("产品名称","客户名称","日期", "数量","单价")
jb.AddExp("金额","数量 * 单价")
jb.Build()
MainTable = Tables("查询表1")
```

生成的查询表如下图，产品名称列来自于产品表，客户名称列来自客户表，日期、数量、单价和金额来自于订单表，其中金额列是通过数量和单价计算得出的：

	产品名称	客户名称	日期	数量	单价	金额
1	盐水鸭	浩天旅行社	1999-01-02	753	17	12801
2	温馨奶酪	红阳事业	1999-01-03	348	20	6960
3	盐水鸭	立日股份有限公	1999-01-03	305	17	5185
4	运动饮料	浩天旅行社	1999-01-04	202	18	3636
5	三合一麦片	浩天旅行社	1999-01-04	492	10	4920
6	盐水鸭	浩天旅行社	1999-01-07	144	17	2448
7	运动饮料	立日股份有限公	1999-01-08	232	14.4	3340.8

示例三

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单", "客户ID","客户","客户ID")
jb.AddExp("产品","产品名称")
jb.AddExp("客户","客户名称")
jb.AddCols("日期", "数量","单价")
jb.AddExp("金额","数量 * 单价")
jb.Build()
MainTable = Tables("查询表1")
```

生成的查询表如下图，产品列由产品表的产品名称列改名得来，客户列由客户表的客户名称列改名得来，日期、数量、单价和金额来自于订单表，其中金额列是通过数量和单价计算得出的：

	产品	客户	日期	数量	单价	金额
1	盐水鸭	浩天旅行社	1999-01-02	753	17	12801
2	温馨奶酪	红阳事业	1999-01-03	348	20	6960
3	盐水鸭	立日股份有限公	1999-01-03	305	17	5185
4	运动饮料	浩天旅行社	1999-01-04	202	18	3636
5	三合一麦片	浩天旅行社	1999-01-04	492	10	4920
6	盐水鸭	浩天旅行社	1999-01-07	144	17	2448
7	运动饮料	立日股份有限公	1999-01-08	232	14.4	3340.8

示例四

遇上同名列的时候，记得要指定来源表，例如下面的产品ID列：

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddCols("{产品}.产品ID","产品名称", "日期", "数量","单价")
jb.Build()
MainTable = Tables("查询表1")
```

注意表名要用大括号括起来。

示例五

在命令窗口执行下面的代码，可以得到所有订购过浓缩咖啡的客户的名单：

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单", "客户ID","客户","客户ID")
jb.AddCols("客户名称")
jb.Filter = "产品名称  = '浓缩咖啡'"
jb.Distinct = True
jb.Build()
MainTable = Tables("查询表1")
```

代码的关键在于：

```
jb.Filter = "产品名称  = '浓缩咖啡'"
jb.Distinct = True
```

第一行代码设置了条件，第二行代码排除了重复值。

示例六

在命令窗口执行下面的代码，可得到订购数量最多的前10个订单：

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单", "客户ID","客户","客户ID")
jb.AddCols("产品名称","客户名称","日期", "数量","单价")
jb.AddExp("金额","数量 * 单价")
jb.Top = 10
jb.Order = "数量  Desc"
jb.Build()
MainTable = Tables("查询表1")
```

示例七

SQLJoinTableBuilder也可以进行简单的分组统计。
用AddCols增加列时，可以将最后一个参数设置为True，以表示添加的是分组列。
用AddExp增加表达式列时，可以将最后一个参数设置为True，以表示添加的是分组列。

例如按年统计各客户订购每种产品的数量和金额：

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
```

```
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单","客户ID","客户","客户ID")
jb.AddExp("年","Year(日期)",True)
jb.AddCols("产品名称","客户名称",True)
jb.AddExp("数量","Sum(数量)")
jb.AddExp("金额","Sum(数量 * 单价)")
jb.Build()
MainTable = Tables("查询表1")
```

得到的统计表为：

	年	产品名称	客户名称	数量	金额
1	1999	浓缩咖啡	福星制衣厂股份	1822	40084
2	1999	浓缩咖啡	浩天旅行社	2694	55470.8
3	1999	浓缩咖啡	红阳事业	602	11888.8
4	1999	浓缩咖啡	立日股份有限公	3215	65326.8
5	1999	浓缩咖啡	威航货运有限公	3877	83494.4
6	1999	三合一麦片	福星制衣厂股份	1670	15978
7	1999	三合一麦片	浩天旅行社	2403	24030
8	1999	三合一麦片	红阳事业	4325	36170

提示：

SQLJoinTableBuilder毕竟不是专门的统计工具，只能进行简单的分组统计，不能交叉统计，更不具备数据分析功能，我们建议数据统计用专门的统计工具，而数据查询用SQLJoinTableBuilder。
所以本示例只是让大家知道SQLJoinTableBuilder也具备统计功能而已，并无太多实质意义。

示例八

除Build方法外，SQLJoinTableBuilder还有一个BuildSQL方法，用于生成Select语句，例如在命令窗口执行：

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
Dim sl As String
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单", "客户ID","客户","客户ID")
jb.AddCols("产品名称","客户名称","日期", "数量","单价")
jb.AddExp("金额","数量 * 单价")
sl = jb.BuildSQL()
Output.Show(sl)
```

可以看出生成的Select语句为：

Select 产品名称,客户名称,日期,数量,单价,数量 * 单价 As [金额] From ({订单} INNER JOIN {产

品} ON {产品}.[产品 ID] = {订单}.[产品 ID]) INNER JOIN {客户} ON {客户}.[客户 ID] = {订单}.[客户 ID]

似乎直接用 Select 语句更为简洁点，但是用 SQLJoinTableBuilder 不需要任何 SQL 语言知识，而且是傻瓜式的编写，简单直接，一点不伤脑筋；我个人是很喜欢这个工具的，因为 Select 语句中的 Join 一多，我就会头晕。

4.5.9 合并多个查询结果

有的时候，我们会有将多个查询结果组合在一起的需求。

组合同一数据源的查询结果

SQLJoinTableBuilder 有一个 Union 方法，可以组合多个其他 SQLJoinTableBuilder 的查询结果。

语法：

Union(Builder, All)

Builder：被组合的另一个 SQLJoinTableBuilder

All： 可选参数，组合的时候默认是排除重复值的，如果需要组合所有数据，请将此参数设置为 True

参与组合的多个 SQLJoinTableBuilder，其生成的查询表必须具备相同的结构。

示例

假设你有订单 2009、订单 2010、订单 2011 三个表，需要从这三个订单表中找出所有客户为“红阳事业”的订单：

```
Dim jb1 As New SQLJoinTableBuilder("查询表1","订单2009")
jb1.ConnectionName = "Sale"
jb1.AddTable("订单","产品ID","产品","产品ID")
jb1.AddTable("订单", "客户ID","客户","客户ID")
jb1.AddCols("产品名称","客户名称","日期", "数量","单价")
jb1.AddExp("金额","数量 * 单价")
jb1.Filter = "客户名称 = '红阳事业'"

Dim jb2 As New SQLJoinTableBuilder("查询表1","订单2010")
jb2.ConnectionName = "Sale"
jb2.AddTable("订单","产品ID","产品","产品ID")
jb2.AddTable("订单", "客户ID","客户","客户ID")
jb2.AddCols("产品名称","客户名称","日期", "数量","单价")
jb2.AddExp("金额","数量 * 单价")
jb2.Filter = "客户名称 = '红阳事业'"

Dim jb3 As New SQLJoinTableBuilder("查询表1","订单2011")
jb3.ConnectionName = "Sale"
jb3.AddTable("订单","产品ID","产品","产品ID")
jb3.AddTable("订单", "客户ID","客户","客户ID")
jb3.AddCols("产品名称","客户名称","日期", "数量","单价")
jb3.AddExp("金额","数量 * 单价")
jb3.Filter = "客户名称 = '红阳事业'"
```

```
jb1.Union(jb2) '组合jb2
jb1.Union(jb3) '组合jb3
jb1.Build()

MainTable = Tables("查询表1")
```

组合不同数据源的查询结果

如果查询结果来自于不同的数据源，需要采用Merge方法组合，Merge方法不如Union方法高效。

示例

假设你有三个数据源，分别为 Sale1、Sale2、Sale3，三个数据源都有订单表，需要从这三个订单表中找出所有客户为“红阳事业”的订单：

```
Dim jb1 As New SQLJoinTableBuilder("查询表1","订单")
jb1.ConnectionName = "Sale1"
jb1.AddTable("订单","产品ID","产品","产品ID")
jb1.AddTable("订单", "客户ID","客户","客户ID")
jb1.AddCols("产品名称","客户名称","日期", "数量","单价")
jb1.AddExp("金额","数量 * 单价")
jb1.Filter = "客户名称 = '红阳事业'"

Dim jb2 As New SQLJoinTableBuilder("查询表1","订单")
jb2.ConnectionName = "Sale2"
jb2.AddTable("订单","产品ID","产品","产品ID")
jb2.AddTable("订单", "客户ID","客户","客户ID")
jb2.AddCols("产品名称","客户名称","日期", "数量","单价")
jb2.AddExp("金额","数量 * 单价")
jb2.Filter = "客户名称 = '红阳事业'"

Dim jb3 As New SQLJoinTableBuilder("查询表1","订单")
jb3.ConnectionName = "Sale3"
jb3.AddTable("订单","产品ID","产品","产品ID")
jb3.AddTable("订单", "客户ID","客户","客户ID")
jb3.AddCols("产品名称","客户名称","日期", "数量","单价")
jb3.AddExp("金额","数量 * 单价")
jb3.Filter = "客户名称 = '红阳事业'"

jb1.Merge(jb2) '组合jb2
jb1.Merge(jb3) '组合jb3
jb1.Build()

MainTable = Tables("查询表1")
```

4.5.10 指定连接类型

普通用户可以忽略本节内容。
向 SQLGroupTableBuilder、SQLCrossTableBuilder、SQLJoinTableBuilder 添加表的完整语法是：

AddTable(Table1,Col1,Table2,Col2,JoinMode)

Table1： 基表，或者已经添加的数据表 ，称之为左表。
Col1： Table1 通过此列和 Table2 关联。
Table2： 要添加的表，称之为右边。
Col2： Table2 通过此列和 Table1 关联。
JoinMode：可选参数，指定连接类型。JoinModeEnum 类型，可选值有：

- Inner
  内连接,左表和右表都具备的数据才会出现在查询表中。

- Left
  左连接,左表的所有数据出现在查询表中。

- Right
  右连接,右表的所有数据出现在查询表中。

示例:

```
Dim jb As New SQLJoinTableBuilder("查询表 1","订单")
jb.AddTable("订单","产品 ID","产品","产品 ID",JoinModeEnum.Left)
```

4.5.11 综合使用 SQL 语句和统计工具

如果你已经对 SQL 语言有所了解，可以看看本节内容，否则可以先跳过。
关于 SQL 语言，后面会用专门的一章给大家讲述。
FoxTable 的统计可以让用户无需掌握 SQL 语言，也能随心所欲地进行统计。
不过如果你熟练 SQL 语言，那么可以将二者双剑合璧，充分发挥各自的优势，完成一些二者难以独立承担的任务。
示例一
我们知道，通过 SQLCommand 的 ExecuteReader 方法，可以根据指定的 SQL 语句生成一个临时的 DataTable 对象，而 GroupTableBuilder 和 CrossTableBuilder 都可以对这种临时的 DataTable 进行统计。
打开 CaseStudy 目录下的文件"多表统计.Table"，然后在命令窗口测试下面的代码：

```
'用SQL语句生成临时表
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.ConnectionName = "Sale"
```

```
cmd.CommandText = "Select 产品名称,数量,日期 FROM {订单} INNER JOIN {产品} ON {订单}.产品ID = {产品}.产品ID"
dt = cmd.ExecuteReader()
'对临时进行分组统计
Dim b As New CrossTableBuilder("统计表1",dt)
b.HGroups.AddDef("产品名称")
b.VGroups.AddDef("日期","{0}月")
b.Totals.AddDef("数量")
b.HorizontalTotal = True
b.VerticalTotal = True
b.Build()
MainTable = Tables("统计表1")
```

可以得到下图所示的统计结果：

	产品名称	01月	02月	03月	04月	05月	06月	07月	08月	09月	10月	11月	12月	合计
1	浓缩咖啡	2061	1058	1908	2352	2226	1987	1786	1201	2194	2090	2238	2076	23177
2	三合一麦片	3996	2938	2145	2951	2092	722	4118	2444	2013	2635	2139	462	28655
3	温馨奶酪	2524	7070	5172	10926	8361	4405	2369	6499	3695	10436	8533	4170	74160
4	盐水鸭	2530	4623	3667	2053	4768	2420	2058	4626	4297	2540	4936	1879	40397
5	运动饮料	3746	3024	5415	3844	3544	2407	3996	2935	4786	4168	3804	2959	44628
6	合计	14857	18713	18307	22126	20991	11941	14327	17705	16985	21869	21650	11546	211017

代码的关键在于定义CrossTableBuilder的时候，指定用ExecuteReader生成的临时表dt作为要统计的数据表：

```
Dim b As New CrossTableBuilder("统计表1",dt)
```

当然用SQLCrossTableBuilder也可以直接完成同样的统计任务，本节只是用一个简单的例子，告诉你如何结合使用SQL语句和Foxtable特有的统计工具。

示例二

上面的方法并不是一个高效的方法，假定有10万个订单，上面的方法会将10万行加载到FoxTable再统计，效率极低；如果是100万行呢？ 那么后果就只有一个：系统崩溃。
所以我们还需要有更好的办法。

其实不管是GroupTableBuilder还是CrossTableBuilder，都可以直接使用Select语句作为统计数据来源。例如要完成示例一的统计任务，代码为：

```
Dim sql As String = "Select 产品名称,数量,日期 FROM {订单} INNER JOIN {产品} ON {订单}.产品ID = {产品}.产品ID"
Dim b As New CrossTableBuilder("统计表1",sql, "Sale")
b.HGroups.AddDef("产品名称")
b.VGroups.AddDef("日期","{0}月")
b.Totals.AddDef("数量")
```

```
b.HorizontalTotal = True
b.VerticalTotal = True
b.Build()
MainTable = Tables("统计表1")
```

不仅代码更为简洁， 效率更有天壤之别。

上述代码的关键在于：

```
Dim b As New CrossTableBuilder("统计表1",sql, "Sale")
```

第一个参数指定统计表名称，第二个参数指定Select语句，第三个参数指定数据源名称。

如果是内部数据源，代码为：

```
Dim b As New CrossTableBuilder("统计表1",sql)
```

4.5.12 SQLJoinTableBuilder 和统计工具

上一节已经讲到，不管是 GroupTableBuilder 还是 CrossTableBuilder，都可以直接使用 Select 语句作为统计数据来源。
不过这有个不足，就是需要我们自己编写 Select 语句，对于某些用户来说，这有些难度。
如果将 SQLJoinTableBuilder 配合统计工具使用，那么这个问题就不复存在了，因为 SQLJoinTableBuilder 可以用简单直观的方式生成 SQL 语句。
示例
打开 CaseStudy 目录下的文件"多表统计.Table"，然后在命令窗口测试下面的代码：

```
Dim jb As new SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddExp("产品","产品名称")
jb.AddCols("日期", "数量","单价")
jb.AddExp("金额", "[数量] * [单价]")

Dim cb As new CrossTableBuilder("统计表1",jb.BuildSQL,"Sale")
cb.HGroups.AddDef("产品")
cb.VGroups.AddDef("日期",DateGroupEnum.Year,"{0}年")
cb.Totals.Adddef("数量")
cb.Totals.AddDef("金额")
```

```
cb.Build()
MainTable = Tables("统计表1")
```

得到的统计结果为：

	产品	1999年		2000年	
		数量	金额	数量	金额
1	浓缩咖啡	12210	256264.8	10967	230885.6
2	三合一麦片	14448	132986	14207	129486
3	温馨奶酪	37148	659253	37012	653097.2
4	盐水鸭	20042	394915	20355	403768.2
5	运动饮料	21791	364172.4	22837	381873.6

当然用 SQLCrossTableBuilder 也可以直接完成同样的统计任务， 而且更加简单，本节只是用一个简单的例子，告诉你如何结合使用 SQLJoinTableBuilder 和统计工具。

4.5.13 记得设置数据源名称

SQLGroupTableBuilder，SQLCrossTableBuilder 和 SQLJoinTableBuilder，这三个以 SQL 开头的工具，都是统计和查询后台数据的。
所以当你使用外部数据源的时候，记得设置其 ConnectionName 属性，指定数据源名称，例如：

```
Dim b As New  SQLGroupTableBuilder("统计表1","订单")
b.ConnectionName = "数据源名称" '使用外部表，一定要设置数据源名称
b.AddTable("订单","产品ID","产品","产品ID")
b.Groups.AddDef("产品名称")
b.Totals.AddDef("数量")
b.Build()
MainTable = Tables("统计表1")
```

经常会有用户，包括我们自己，代码明明没有问题，表名和列名也肯定正确，却总是提示生成的 SQL 语句错误，或提示不存在指定表名的表，多数是因为没有设置数据源名称造成的。

4.5.14 组合统计结果

4.5.14.1 多表统计工具的局限性

SQLGroupTableBuilder 和 SQLCrossTableBuilder，包括 SQLJoinTableBuilder，都可以对多个表的数据进行统计，但是对于参与统计的表有两点要求:
1、参与统计的表必须是一对多或一对一的关系，不能是多对多的关系。
2、如果参与统计的表是一对多的关系，那么父表中的列只能作为分组列，而不能作为统计列。
如果参与统计的表无法满足上述要求 ，我们也就无法一次性地通过某个统计工具来得出统计结果，只能使用统计工具分别对各表进行统计，然后将多个统计结果组合起来。
接下来我们就介绍如何组合多个统计结果。

4.5.14.2 组合多个统计结果

本节内容可以参考 CaseStudy 目录下的文件：组合统计结果.Table
假定有下图所示三个表：

进货单

	型号	数量	单价	金额
1	NB-MS24	60	5	300
2	NB-XG118	80	5	400
3	NB-XG1	50	8	400
4	NB-XG1	70	8	560
5	NB-XXX	25	7	175
6	NB-XG1	100	8	800

销售单

	型号	数量	售价	金额
1	NB-MS24	12	20	240
2	NB-MS24	15	25	375
3	NB-XG118	8	15	120
4	NB-XG118	5	10	50
5	NB-XG118	10	15	150
6	NB-XG1	2	16	32
7	NB-XG118	20	20	400
8	NB-XG1	12	13	156
9	NB-XG1	6	15	90

退货单

	型号	数量	单价	金额
1	NB-XG1	3	8	24
2	NB-XG118	6	5	30

希望根据上述三个表得出下图所示的统计结果：

	型号	进货		销售		退货		库存	
		数量	金额	数量	金额	数量	金额	数量	金额
1	NB-MS24	60	300	27	615			33	165
2	NB-XG1	220	1760	20	278	3	24	197	1576
3	NB-XG118	80	400	43	720	6	30	31	155
4	NB-XXX	25	175					25	175

上述三个表之间，是一种多对多的关系，因为一个型号会有多次进货，也会有多次销售或多次退货，而且要统计的是在三个表中同名的数量列和金额列。
所以 SQLGroupTableBuilder 和 SQLCrossTableBuilder 对于这种统计是无能为力的，我们只能另想办法。
即使你精通 SQL，这样的统计也是很伤脑筋的，而且对于大多数普通用户来说，精通 SQL 语言只是一种奢望而已。

上面统计虽然看似复杂，但细想一下，不过就是分别统计出每个型号的进货、出货和退货数据，然后组合在一起而已。

如果有一个方法，能够将多个统计结果组合在一起，那么就可以将复杂问题简单化，只需分别统计进货、销售和退货数据，然后组合在一起即可。
我们知道所有的统计工具都有一个 BuildDataSource 方法，用此方法生成的统计结果可以动态绑定到现有的 Table 中，例如：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("日期","月份")
b.Totals.AddDef("数量")
Tables("窗口 1_Table1").DataSource = b.BuildDataSource()
```

BuildDataSource 方法生成的是一个 fxDataSource 类型的对象，此类型有一个 Combine 方法，用于 组合多个 fxDataSource 对象，语法为：

```
Combine(LeftColName, RightDataSource, RightColName)
```

LeftColName:　字符型，指定用于联接的列，可以是单个列名，也可以是数组。
RightDataSource：参与组合的另一个 fxDataSource 对象。
RightColName：　字符型，指定另一个 fxDataSource 用于联接的列，可以是单个列名，也可以是数组。
联接列的作用类似于关联列，将两个统计表联系起来。
有了这个 Combine 方法，我们就可以轻松完成之前的统计任务，新建一个窗口，在窗口中插入一个 Table 控件和一个按钮，将按钮的 Click 事件代码设为：

```
Dim bd1 As New GroupTableBuilder("统计表1",DataTables("进货单"))
Dim dt1 As fxDataSource
bd1.Groups.AddDef("型号") '根据型号分组
bd1.Totals.AddDef("数量","进货_数量") '对数量进行统计
bd1.Totals.AddDef("金额","进货_金额") '对金额进行统计
dt1 = bd1.BuildDataSource()

Dim bd2 As New GroupTableBuilder("统计表2",DataTables("销售单"))
Dim dt2  As fxDataSource
bd2.Groups.AddDef("型号") '根据型号分组
bd2.Totals.AddDef("数量","销售_数量") '对数量进行统计
bd2.Totals.AddDef("金额","销售_金额") '对金额进行统计
dt2 = bd2.BuildDataSource()

Dim bd3 As New GroupTableBuilder("统计表3",DataTables("退货单"))
Dim dt3 As fxDataSource
bd3.Groups.AddDef("型号") '根据型号分组
bd3.Totals.AddDef("数量","退货_数量") '对数量进行统计
bd3.Totals.AddDef("金额","退货_金额") '对金额进行统计
dt3 = bd3.BuildDataSource()

dt1.Combine("型号",dt2,"型号") '将销售统计数据组合到进货统计数据
dt1.Combine("型号",dt3,"型号") '将退货统计数据组合到进货统计数据

Tables("窗口 1_Table1").DataSource = dt1 '将统计结果绑定到 Table
```

```
With DataTables("窗口 1_Table1").DataCols  '用表达式列计算库存数据
    .Add("库存_数量",Gettype(Integer), "IsNull([进货_数量],0) - ISNULL([销售_数量],0) - ISNULL([退货_数量],0)")
    .Add("库存_金额",Gettype(Double), "[库存_数量] /[进货_数量] * [进货_金额]")
End With
```

这样单击此按钮，Table 控件即可显示我们要求的统计结果。

说明：

1、由于三个表都要统计数量和金额列，所以在添加统计列的时候，必须将数量和金额列改名，以便区分：

```
bd1.Totals.AddDef("数量","进货_数量") '对数量进行统计
bd1.Totals.AddDef("金额","进货_金额") '对金额进行统计
```

2、库存数据是无需直接统计的，直接用表达式列根据进货、销售、库存计算得出。

4.5.14.3 直接显示统计结果

如果想直接在主界面中显示组合统计结果，可以使用fxDataSource的Show方法，此方法的语法为：

```
Show(Name, Caption)
```

Name: 指定表名
Caption：可选参数，指定表标题

示例

打开CaseStudy目录下的文件“组合统计结果.Table”，然后在命令窗口执行执行下面的代码：

```
Dim bd1 As New GroupTableBuilder("统计表1",DataTables("进货单"))
Dim dt1 As fxDataSource
bd1.Groups.AddDef("型号") '根据型号分组
bd1.Totals.AddDef("数量","进货_数量") '对数量进行统计
bd1.Totals.AddDef("金额","进货_金额") '对金额进行统计
dt1 = bd1.BuildDataSource()

Dim bd2 As New GroupTableBuilder("统计表2",DataTables("销售单"))
Dim dt2  As fxDataSource
bd2.Groups.AddDef("型号") '根据型号分组
bd2.Totals.AddDef("数量","销售_数量") '对数量进行统计
bd2.Totals.AddDef("金额","销售_金额") '对金额进行统计
dt2 = bd2.BuildDataSource()
```

```
Dim bd3 As New GroupTableBuilder("统计表3",DataTables("退货单"))
Dim dt3 As fxDataSource
bd3.Groups.AddDef("型号") '根据型号分组
bd3.Totals.AddDef("数量","退货_数量") '对数量进行统计
bd3.Totals.AddDef("金额","退货_金额") '对金额进行统计
dt3 = bd3.BuildDataSource()

dt1.Combine("型号",dt2,"型号") '将销售统计数据组合到进货统计数据
dt1.Combine("型号",dt3,"型号") '将退货统计数据组合到进货统计数据
dt1.Show("统计表1") '显示统计结果
```

4.5.14.4 组合统计与后台统计

我们知道，GroupTableBuilder 和 CrossTableBuilder 有一个名为 FromServer 的属性，只需将此属性设为 True，即可直接统计后台数据。

此外 Foxtable 还提供了两个专门用于统计后台数据的工具：SQLGroubpTableBuilder 和 SQLCrossTableBuilder。

多个后台统计的统计结果，同样可以组合起来.

示例一

打开 CaseStudy 目录下的文件“组合统计结果.Table”，然后在命令窗口执行执行下面的代码：

```
Dim bd1 As New GroupTableBuilder("统计表 1",DataTables("进货单"))
Dim dt1 As fxDataSource
bd1.FromServer = True
bd1.Groups.AddDef("型号") '根据型号分组
bd1.Totals.AddDef("数量","进货数量") '对数量进行统计
dt1 = bd1.BuildDataSource()

Dim bd2 As New GroupTableBuilder("统计表 2",DataTables("销售单"))
Dim dt2 As fxDataSource
bd2.FromServer = True
bd2.Groups.AddDef("型号") '根据型号分组
bd2.Totals.AddDef("数量","销售数量") '对数量进行统计
dt2 = bd2.BuildDataSource()

Dim bd3 As New GroupTableBuilder("统计表 3",DataTables("退货单"))
Dim dt3 As fxDataSource
bd3.FromServer = True
bd3.Groups.AddDef("型号") '根据型号分组
bd3.Totals.AddDef("数量","退货数量") '对数量进行统计
dt3 = bd3.BuildDataSource()
dt1.Combine("型号",dt2,"型号") '将销售统计数据组合到进货统计数据
dt1.Combine("型号",dt3,"型号") '将退货统计数据组合到进货统计数据
dt1.Show("统计表 1") '显示统计结果
```

即可得到不同型号产品的进货数量、销售数量和退货数量。

示例二

由于使用 GroupTableBuilder 统计后台数据时，不能包括表达式列，所以如果需要同时统计数量和金额，只能改用 SQLGroupTableBuilder：

```
Dim bd1 As New SQLGroupTableBuilder("统计表 1","进货单")
Dim dt1 As fxDataSource
bd1.Groups.AddDef("型号") '根据型号分组
bd1.Totals.AddDef("数量","进货_数量") '对数量进行统计
bd1.Totals.AddExp("进货_金额","数量 * 单价") '对金额进行统计
dt1 = bd1.BuildDataSource()

Dim bd2 As New SQLGroupTableBuilder("统计表 2","销售单")
Dim dt2 As fxDataSource
bd2.Groups.AddDef("型号") '根据型号分组
bd2.Totals.AddDef("数量","销售_数量") '对数量进行统计
bd2.Totals.AddExp("销售_金额","数量 * 售价") '对金额进行统计
dt2 = bd2.BuildDataSource()

Dim bd3 As New SQLGroupTableBuilder("统计表 3","退货单")
Dim dt3 As fxDataSource
bd3.Groups.AddDef("型号") '根据型号分组
bd3.Totals.AddDef("数量","退货_数量") '对数量进行统计
bd3.Totals.AddExp("退货_金额","数量 * 单价") '对金额进行统计
dt3 = bd3.BuildDataSource()

dt1.Combine("型号",dt2,"型号") '将销售统计数据组合到进货统计数据
dt1.Combine("型号",dt3,"型号") '将退货统计数据组合到进货统计数据
dt1.Show("统计表 1") '显示统计结果
```

4.5.14.5 多个联接列

组合统计结果的联接列可以有多个。

假定进货单、销售单、退货单三个表，除了型号列之外，还有一个品名列，需要根据品名和型号分组，统计进货、销售、退货、库存数据，代码为：

```
Dim bd1 As New GroupTableBuilder("统计表1",DataTables("进货单"))
Dim dt1 As fxDataSource
bd1.Groups.AddDef("品名") '根据品名分组
bd1.Groups.AddDef("型号") '根据型号分组
bd1.Totals.AddDef("数量","进货_数量") '对数量进行统计
bd1.Totals.AddDef("金额","进货_金额") '对金额进行统计
dt1 = bd1.BuildDataSource()

Dim bd2 As New GroupTableBuilder("统计表2",DataTables("销售单"))
Dim dt2 As fxDataSource
```

```
bd2.Groups.AddDef("品名") '根据品名分组
bd2.Groups.AddDef("型号") '根据型号分组
bd2.Totals.AddDef("数量","销售_数量") '对数量进行统计
bd2.Totals.AddDef("金额","销售_金额") '对金额进行统计
dt2 = bd2.BuildDataSource()

Dim bd3 As New GroupTableBuilder("统计表3",DataTables("退货单"))
Dim dt3 As fxDataSource
bd3.Groups.AddDef("品名") '根据品名分组
bd3.Groups.AddDef("型号") '根据型号分组
bd3.Totals.AddDef("数量","退货_数量") '对数量进行统计
bd3.Totals.AddDef("金额","退货_金额") '对金额进行统计
dt3 = bd3.BuildDataSource()

Dim nms As String() = {"品名","型号"} '指定连接列
dt1.Combine(nms,dt2,nms) '将销售统计数据组合到进货统计数据
dt1.Combine(nms,dt3,nms) '将退货统计数据组合到进货统计数据
dt1.Show("统计表1") '显示统计结果
```

4.5.15 临时表生成器

DataTableBuilder 类型是专为创建临时表而提供, 定义 DataTableBuilder 变量的语法是：
New DataTableBuilder(Name)
New DataTableBuilder(Name, Caption)
Name:　临时表名称
Caption: 临时表标题
DataTableBuilder 类型有一个 AddDef 方法，用于定义新增列的属性，可以一次性的指定新增列的名称、类型、标题、长度、表达式，AddDef 的语法有六种，非常灵活：
AddDef(Name, Type)
AddDef(Name, Type, Expression)
AddDef(Name, Type, Expression，Caption)
AddDef(Name, Type, MaxLength)
AddDef(Name, Type, MaxLength，Expression)
AddDef(Name, Type, MaxLength，Expression，Caption)
Name:　　字符型，指定列的名称
Type:　　Type 类型，指定列的类型，用 GetType 函数获得指定的类型，例如:GetType(String),表示字符型
MaxLength:　整数型，指定字符列的长度
Expression：　字符型，指定计算表达式
Caption:　字符型，指定列标题
用 AddDef 方法添加所有列的定义后，最后用 Build 方法生成临时表。
示例一

```
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("产品", GetType(String), 32)
dtb.AddDef("客户", GetType(String), 32)
dtb.AddDef("数量", GetType(Integer))
dtb.AddDef("金额", GetType(Double))
dtb.Build()
MainTable= Tables("统计")
```

上述的代码增加了一个名为统计表的临时表，该表包括四列：
产品：字符型，长度为 32
客户：字符型，长度为 32
数量：整数型
金额：双精度小数型

	产品	客户	数量	金额
1				
2				
3				
4				

用代码生成的临时表，在关闭项目后，将不复存在。

示例二

假定表 A 的结构如下：

	姓名	课程	分数
1	张三	语文	74
2	张三	数学	83
3	张三	物理	93
4	李四	语文	74
5	李四	数学	84
6	李四	物理	94

希望将其转换为表 B，结构为：

	姓名	数学	物理	语文
1	李四	84	94	74
2	张三	83	93	74

转换代码为：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("姓名", Gettype(String), 32)
For Each v As String In DataTables("表A").GetValues("课程")
        dtb.AddDef(v, Gettype(Double))
Next
dtb.Build()
For Each v As String In DataTables("表A").GetValues("姓名")
        Dim dr1 As DataRow = DataTables("表B").AddNew()
        dr1("姓名") = v
        For Each dr2 As DataRow In DataTables("表A").DataRows
                If dr2("姓名") = v Then
                        dr1(dr2("课程")) = dr2("分数")
                End If
        Next
Next
MainTable = Tables("表B")
```

更多的转换示例，可以参考：水平表和垂直表的转换

4.5.16 不显示生成的表

以下所有的类型，都有一个 Build 方法
GroupTableBuilder
CrossTableBuilder
SQLGroupTableBuilder
SQLCrossTableBuilder
DataTableBuilder
SQLJoinTableBuilder
执行这些类型的 Build 方法，将生成一个临时表(包括 DataTable 和 Table)， 有的用于显示统计结果，有的用于显示查询结果，有的只是一个空白表。
在一些特殊的场合，可能只想在代码中使用这个临时表，而并不希望显示它。
Build 方法有一个可选的逻辑参数，如果设置为 True，将只生成一个 DataTable，这个 Datatable 只能在代码中使用，而且代码运行结束后，会自动销毁。
由于没有生成 Table，所以这个临时表并不会在界面中。
示例

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
Dim t As DataTable
Dim s As Double
b.Groups.AddDef("产品")
b.Totals.AddDef("数量")
t = b.Build(True) '生成一个 DataTable
For Each r As DataRow In t.DataRows
    Select Case r("产品")
        Case "PD01","PD02"
            s = s + r("数量")
        Case Else
            s = s + r("数量") / 100
    End Select
Next
Output.Show(s)
```

上述代码执行结束后，所生成的 DataTable 将自动销毁。

4.5.17 自动显示合计行

不管是合计模式，还是汇总模式，用户一旦进行排序或筛选，合计模式和汇总模式就会自动消失。
如果希望排序或筛选后，依然能够保持显示合计模式或汇总模式，可以在 AfterSort 和 AfterFilter 两个事件中，加上设置合计模式或汇总模式的代码。
例如对于订单表，希望在排序和筛选后，能够自动显示合计模式，在 AfterSort 和 AfterFilter 两个事件中同时加上下面的代码即可：

```
Tables("订单").Cols("数量").GrandTotal = True
```

```
Tables("订单").Cols("金额").GrandTotal = True
Tables("订单").GrandTotal = True
```

让用户自己决定

当然，更符合实际情况的是用户能够自动控制是否显示合计行，为此我们首先定义一个Public变量：

```
Public AutoTotal As Boolean
```

然后在AfterSort和AfterFilter两个事件中同时加上下面的代码：

```
If AutoTotal Then
    Tables("订单").Cols("数量").GrandTotal = True
    Tables("订单").Cols("金额").GrandTotal = True
    Tables("订单").GrandTotal = True
End If
```

最后在菜单或者窗口中加一个按钮，按钮的代码设置为：

```
If AutoTotal = True Then
    AutoTotal = False
    Tables("订单").GrandTotal = False
Else
    AutoTotal = True
    Tables("订单").Cols("数量").GrandTotal = True
    Tables("订单").Cols("金额").GrandTotal = True
    Tables("订单").GrandTotal = True
End If
```

这样用户单击这个按钮，既可在自动合计和非自动合计两种模式中切换。

4.5.18 数据填充器

Filler 用于从表中提取数据，并填充到另一个表中。

属性：

- SourceTable

  数据来源表。

- SourceCols

  数据来源列，不同的列用逗号分割。

- DataTable

  数据接收表。

- DataCols

  数据接收列，不同的列用逗号分割。

- ExcludeExistValue

  逻辑型，设为 True，排除接收表中已经存在的内容。

- ExcludeNullValue

  逻辑型，设为 True，排除空白值。

- Filter

  填充条件，请参考表达式的运算符和函数 和 条件表达式。

- Distinct

  逻辑型，是否排除重复内容，默认为 True；如果设为 False，将不会排除重复内容，也不会排除已有内容和空白值。

- Append

  逻辑型，默认为 True，在接收表中增加新行用于接收数据，设为 False，则在原有的行中填充，如果不够再增加新行。

提示：数据来源列和数据接收列是可以省略的，或者只设置其中一个，Filler 会自动查找同名列进行匹配填充。

方法：

Fill

开始填充数据

示例：

例如从订单表中提取不重复的客户、产品数据，填充到统计表中：

```
dim f As New Filler
f.SourceTable = DataTables("订单") '指定数据来源
f.SourceCols = "产品,客户" '指定数据来源列
f.DataTable = DataTables("统计") '指定数据接收表
f.DataCols = "产品,客户" '指定数据接收列
f.Fill() '填充数据
```

4.5.19 手工编码填充数据

前面介绍了用 Filler 进行数据填充，Filler 的设置很丰富，使用比较灵活。
不过填充数据并非一定要 Filler，我们也可以手工编码填充。
例如需要从表 A 中，将金额大于 100 的记录，复制到表 B 中，假定两个表的结构一样：

```
For Each dr1 As DataRow In DataTables("表 A").Select("[金额] > 100")
    Dim dr2 As DataRow = DataTables("表 B").AddNew()
    For Each dc As DataCol in DataTables("表 A").DataCols
        dr2(dc.Name) = dr1(dc.name)
    Next
Next
```

如果表 A 和表 B 的结构不同，或者只需填充部分列，就需要定义数据来源列和数据接收列，例如：

```
Dim Cols1() As String = {"来源列一","来源列二","来源列三"}
Dim Cols2() As String = {"接收列一","接收列二","接收列三"}
For Each dr1 As DataRow In DataTables("表 A").Select("[金额] > 100")
    Dim dr2 As DataRow = DataTables("表 B").AddNew()
    For i As Integer = 0 To Cols1.Length -1
        dr2(Cols2(i)) = dr1(Cols1(i))
    Next
Next
```

如果要排除重复值，可以用 GetValues 获得不重复的值，然后再填充：

```
Dim Vals As List(of String())
Vals = DataTables("来源表").GetValues("产品|客户")
For i As Integer = 0 To Vals.Count - 1
    Dim dr As DataRow = DataTables("接收表").AddNew()
    dr("产品") = Vals(i)(0)
    dr("客户") = Vals(i)(1)
Next
```

4.5.20 杂谈

4.5.20.1 手工编码进行统计

普通用户可忽略本节的内容。
GroupTableBuilder 可以让我们轻松地生成各种分组统计表。
CrossTableBuilder 可以让我们轻松地生成各种交叉统计表。
但是，他们也不是万能的：
1、一些特殊的统计要求，它们是无法胜任的，例如本课的成绩分布统计。
2、如果希望事先设计好统计表的结构，需要的时候通过单击某个按钮刷新统计表的结果，显然这样的统计要求通过生成器是无法实现的。
在学习本课的内容之前，请打开示例文件"统计演示.Table"。
示例一

假定有下面这样的一个成绩表：

	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136	140	71	75	555
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529
11	1班	张李华	124	118	124	83	88	537
12	2班	陈成进	110	58	106	36	34	344
13	2班	陈诗淇	113	123	117	78	78	509
14	2班	陈思达	113	55	92	55	46	361

我们需要统计每一个科目的成绩分布范围，例如 60 分以下，60 分以上，70 分以上等等范围的人数。

首先新建一个数据表，因为考试的科目是固定的，所以这里我们不用临时表，而用数据表，设计好表之后，手工输入科目：

	科目	分布								
		60分以下	60分以上	70分以上	80分以上	90分以上	100分以上	110分以上	120分以上	130分以上
1	语文									
2	英语									
3	数学									
4	物理									
5	化学									

然后将这个表的 DataColChanged 设为：

```
Dim dr As DataRow = e.DataRow
Dim dt As DataTable =DataTables("学生成绩")
If e.DataCol.Name = "科目" And dr.IsNull("科目") = False Then
    dr("分布_60 分以下") = dt.Compute("Count(姓名)", dr("科目") & " < 60")
    dr("分布_60 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 60 And " & dr("科目") & " < 70" )
    dr("分布_70 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 70 And " & dr("科目") & " < 80" )
    dr("分布_80 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 80 And " & dr("科目") & " < 90" )
    dr("分布_90 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 90 And " & dr("科目") & " < 100" )
    dr("分布_100 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 100 And " & dr("科目") & " < 110" )
    dr("分布_110 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 110 And " & dr("科目") & " < 120" )
    dr("分布_120 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 120 And " & dr("科目") & " < 130" )
    dr("分布_130 分以上") = dt.Compute("Count(姓名)", dr("科目") & " >= 130" )
End If
```

经过上述设置，在任何时候，你只需选择科目列，然后单击菜单中的重置按钮，即可得到最新的成绩分布数据。

如果希望能够自动计算，而不是通过单击按钮去刷新结果，假定这个表的名称是“成绩分布”，可

以在项目(数据库)事件的 MainTableChanged 事件中设置如下代码：

```
If Maintable.Name = "成绩分布" Then
    DataTables("成绩分布").DataCols("科目").RaiseDataColChanged()
End If
```

示例二

继续上面的例子，如果希望得到下面的成绩统计表：

	科目	最高分	最低分	平均分
1	语文	134	100	119
2	英语	146	74	117
3	数学	136	38	104
4	物理	93	36	74
5	化学	100	18	77

只需将该表的 DataColChanged 事件代码设为：

```
Dim dr As DataRow = e.DataRow
Dim dt As DataTable =DataTables("学生成绩")
If e.DataCol.Name = "科目" And dr.IsNull("科目") = False Then
    dr("最高分") = dt.Compute("Max(" & dr("科目") & ")")
    dr("最低分") = dt.Compute("Min(" & dr("科目") & ")")
    dr("平均分") = dt.Compute("Avg(" & dr("科目") & ")")
End If
```

示例三

统计不同产品的销售数量和金额，我们采用事先设计好的临时表来显示统计结果。

1、首先增加一个临时表，包括产品、数量、金额三列。

2、将统计表的 DataColChanged 事件设置为：

```
Dim dr As DataRow = e.DataRow
Dim dt As DataTable = DataTables("订单")
If e.DataCol.Name = "产品" Then
    dr("数量") = dt.Compute("Sum(数量)","[产品] = '" & dr("产品") & "'")
    dr("金额") = dt.Compute("Sum(金额)","[产品] = '" & dr("产品") & "'")
End If
```

3、MainTableChanged 事件中加入如下代码：

```
If Maintable.Name = "销售统计" Then
    dim f As New Filler
    DataTables("销售统计").DataRows.Clear() '清除原有数据
    f.SourceTable = DataTables("订单")
    f.SourceCols = "产品"
    f.DataTable = DataTables("销售统计")
    f.DataCols = "产品"
    f.Fill()
End If
```

这里假定这个统计表的名称为“销售统计”，以后我们每次选择这个表，即可得到最新的统计结果。

示例四

还是统计不同产品的销售数量和金额，不过我们这次全部用一段代码完成。

将下面的代码复制到命令窗口执行即可：

```
'生成统计表
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("产品", GetType(String), 32)
```

```
dtb.AddDef("数量", GetType(Integer))
dtb.AddDef("金额", GetType(Double))
dtb.Build()
'填充数据
dim f As New Filler
f.SourceTable = DataTables("订单")
f.SourceCols = "产品"
f.DataTable = DataTables("统计")
f.DataCols = "产品"
f.Fill()
'统计
For Each dr As Datarow In DataTables("统计").DataRows
    dr("数量") = DataTables("订单").Compute("Sum(数量)","[产品] = '" & dr("产品") & "'")
    dr("金额") = DataTables("订单").Compute("Sum(金额)","[产品] = '" & dr("产品") & "'")
Next
MainTable= Tables("统计")
```

当然，上面的代码如果改用 GroupTableBuilder 来实现，要容易很多。

之所以提供这样的例子，是为了让你知道，其实你有更多的选择，这样面对一些复杂的统计要求，你也能应付自如。

示例五

前面的统计，都使用了 Filler 来向统计表填充分组数据，实际上有时手工编码填充分组数据也是一个不错的选择。

还是以统计不同产品的销售数量和金额为例，如果手工编码填充分组数据，代码为：

```
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("产品", Gettype(String), 32)
dtb.AddDef("数量", Gettype(Integer))
dtb.AddDef("金额", Gettype(Double))
dtb.Build()
For Each nm As String In DataTables("订单").GetValues("产品")
    Dim dr As DataRow = DataTables("统计").AddNew()
    dr("产品") = nm
    dr("数量") = DataTables("订单").Compute("Sum(数量)","[产品] = '" & dr("产品") & "'")
    dr("金额") = DataTables("订单").Compute("Sum(金额)","[产品] = '" & dr("产品") & "'")
Next
MainTable= Tables("统计")
```

代码是不是比示例四简洁很多呢？

4.5.20.2 分组统计和交叉统计的实现原理

普通用户请忽略本节内容。

FoxTable 的分组统计和交叉统计非常强大，几乎无所不能，看上去很神秘。

其实我们自己完全可以编码实现类似的功能。

本节的内容没有太多实际的意义，只是为了让大家更好的融合所学到的知识。
在学习本课的内容之前，请打开示例文件"统计演示.Table"。
本节内容的关键知识在于 DataTableBuilder 类型和 GetValues 方法，如果你对他们已经生疏，请先温习一下。

示例一

统计不同产品的销售数量和金额：

```
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("产品", Gettype(String), 16)
dtb.AddDef("数量", Gettype(Integer))
dtb.AddDef("金额", Gettype(Double))
dtb.Build()
For Each nm As String In DataTables("订单").GetValues("产品")
    Dim dr As DataRow = DataTables("统计").AddNew()
    dr("产品") = nm
    dr("数量") = DataTables("订单").Compute("Sum(数量)","[产品] = '" & dr("产品") & "'")
    dr("金额") = DataTables("订单").Compute("Sum(金额)","[产品] = '" & dr("产品") & "'")
Next
MainTable= Tables("统计")
```

示例二

统计每个客户订购不同产品的数量和金额：

```
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("客户", Gettype(String), 16)
dtb.AddDef("产品", Gettype(String), 16)
dtb.AddDef("数量", Gettype(Integer))
dtb.AddDef("金额", Gettype(Double))
dtb.Build()
For Each nm() As String In DataTables("订单").GetValues("客户|产品")
    Dim dr As DataRow = DataTables("统计").AddNew()
    dr("客户") = nm(0)
    dr("产品") = nm(1)
    dr("数量") = DataTables("订单").Compute("Sum(数量)","客户 = '" & nm(0) & "' And [产品] = '" &
nm(1) & "'")
    dr("金额") = DataTables("订单").Compute("Sum(金额)","客户 = '" & nm(0) & "' And [产品] = '" &
nm(1) & "'")
Next
MainTable= Tables("统计")
```

示例三

以交叉统计的形式，统计每个客户订购不同产品的数量：

```
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("客户", Gettype(String), 16)
Dim prds As List(of String) = DataTables("订单").GetValues("产品")
For Each prd As String In prds
    dtb.AddDef(prd, Gettype(Double))
```

```
Next
dtb.Build()
For Each cus As String In DataTables("订单").GetValues("客户")
    Dim dr As DataRow = DataTables("统计").AddNew()
    dr("客户") = cus
    For Each prd As String In prds
        dr(prd) = DataTables("订单").Compute("Sum(数量)","客户 = '" & cus & "' And [产品] = '" & prd & "'")
    Next
Next
MainTable= Tables("统计")
```

示例四

以交叉统计的形式，统计每个客户订购不同产品的数量和金额：

```
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("客户", Gettype(String), 16)
Dim prds As List(of String) = DataTables("订单").GetValues("产品")
For Each prd As String In prds
    dtb.AddDef(prd & "_数量", Gettype(Integer))
    dtb.AddDef(prd & "_金额", Gettype(Double))
Next
dtb.Build()
For Each cus As String In DataTables("订单").GetValues("客户")
    Dim dr As DataRow = DataTables("统计").AddNew()
    dr("客户") = cus
    For Each prd As String In prds
        dr(prd & "_数量") = DataTables("订单").Compute("Sum(数量)","客户 = '" & cus & "' And [产品] = '" & prd & "'")
        dr(prd & "_金额") = DataTables("订单").Compute("Sum(金额)","客户 = '" & cus & "' And [产品] = '" & prd & "'")
    Next
Next
MainTable= Tables("统计")
```

4.5.20.3 多值字段与统计

普通用户请忽略本节内容。

本节知识的一个要点是 Dictionary，如果你对其已经生疏，请先温习一下。

这是一个很好的使用字典的例子，希望大家能细细体会。

假定有一个下图所示的产量登记表，其中员工工号列是多值字段：

	日期	产品	员工工号	数量
1	2011-11-01	BA001	0002,0003	60
2	2011-11-01	DA047	0002,0004	36
3	2011-11-02	BH266	0005,0006	90
4	2011-11-02	BA001	0002,0006	58
5	2011-11-02	BG101	0002	61
6	2011-11-02	HF379	0001,0002,0003	60
7	2011-11-03	BA001	0005,0004	60
8	2011-11-03	TH273	0006	30
9	2011-11-03	LBB541	0003,0004,0005,0006	32
10	2011-11-03	DE051	0003,0004	60

现在需要统计每个员工加工的产品数量，同一个产品，如果有多个员工参与加工，那么产量平分。显然这样的统计，用传统的统计工具是无法实现的，我们只能手工编码统计，不过代码也不算复杂：

```
'生成统计表的结构
Dim dt As DataTable
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("工号", Gettype(String), 16)
dtb.AddDef("数量", Gettype(Double))
dt = dtb.Build()
'开始逐行累加统计
Dim lst As New Dictionary(of String ,DataRow) '定义一个字典,用于检索每个员工在统计表中对应的行
For Each dr1 As DataRow In DataTables("产量登记").DataRows
    If dr1.IsNull("员工工号") = False Then '如果工号列不为空
        Dim nms() As String = dr1("员工工号").Split(",") '将工号列内容拆分成数组
        Dim v As Double = dr1("数量") / nms.Length '获得平均之后每个工号的生产数量
        For Each nm As String In nms '遍历参与加工此产品的每个工号
            Dim dr2 As DataRow
            If lst.ContainsKey(nm) '如果集合中包括此工号对应的行
                dr2= lst(nm) '将此行赋值给变量 dr2
            Else
                dr2 = dt.AddNew() '否则增加一行.
                dr2("工号") = nm '新增行的工号列设置为此工号
                lst.add(nm,dr2) '将新增行添加到字典中,以便接下来检索
            End If
            dr2("数量") = dr2("数量") + v '加上平均后的产量
        Next
    End If
Next
```

在命令窗口执行上面的代码，即可得到如下统计结果：

	工号	数量
1	0002	158
2	0003	88
3	0004	86
4	0005	83
5	0006	112
6	0001	20

如果要统计每个员工加工不同产品的数量，代码只需稍微调整一下：

```
'生成统计表的结构
Dim dt As DataTable
Dim dtb As New DataTableBuilder("统计")
dtb.AddDef("工号", Gettype(String), 16)
dtb.AddDef("产品", Gettype(String), 16)
dtb.AddDef("数量", Gettype(Double))
dt = dtb.Build()
'开始逐行累加统计
Dim lst As New Dictionary(of String ,DataRow) '定义一个字典,用于检索每个员工在统计表中对应的行
For Each dr1 As DataRow In DataTables("产量登记").DataRows
    If dr1.IsNull("员工工号") = False Then '如果工号列不为空
        Dim nms() As String = dr1("员工工号").Split(",") '将工号列内容拆分成数组
        Dim v As Double = dr1("数量") / nms.Length '获得平均之后每个工号的生产数量
       For Each nm As String In nms '遍历参与加工此产品的每个工号
          Dim dr2 As DataRow
          Dim key As String = nm & "|" & dr1("产品") '这里是关键,由工号和产品组成字典的关键词.
          If lst.ContainsKey(key) '如果集合中包括此关键词对应的行
             dr2= lst(key) '将此行赋值给变量 dr2
          Else
             dr2 = dt.AddNew() '否则增加一行.
             dr2("工号") = nm '新增行的工号列设置为此工号
             dr2("产品") = dr1("产品") '设置增行的产品
             lst.add(key,dr2) '将新增行添加到字典中,以便接下来检索
          End If
       dr2("数量") = dr2("数量") + v '加上平均后的产量
       Next
    End If
Next
MainTable = Tables("统计")
```

4.5.20.4 物料需求统计

假定有下面三个表。

生产计划表，用于输入每个产品计划的生产数量：

生产计划

	产品	生产数量
1	PD01	100
2	PD02	60
3	PD03	70

零件用量表，用户输入每单个产品使用各零件的数量：

零件用量

	产品	零件	用量
1	PD01	CP01	5
2	PD01	CP02	10
3	PD01	CP03	8
4	PD02	CP01	10
5	PD02	CP03	9
6	PD02	CP04	4
7	PD03	CP06	11
8	PD03	CP01	12
9	PD03	CP02	6

零件需求表，用于根据生产计划表和零件需求表，统计出各种零件总的需求数量：

零件需求

	零件	需求数量
1	CP01	2300
2	CP02	1600
3	CP03	1340
4	CP06	1100
5	CP04	240

这是一个很经典的跨表统计的实例，下面我们用三种不同的方法，来得到这个需求统计表。

方法一

用 SQLGroupTableBuilder，在窗口或者菜单新建一个按钮，将其代码设置为：

```
Dim gb As new SQLGroupTableBuilder("统计表","零件用量")
gb.AddTable("零件用量","产品","生产计划","产品")
gb.Groups.AddDef("零件")
gb.Totals.AddExp("需求数量","[生产数量] * [用量]")
gb.Build()
MainTable = Tables("统计表")
```

方法二

用 SQLJoinTableBuilder，在窗口或者菜单新建一个按钮，将其代码设置为：

```
Dim jb As new SQLJoinTableBuilder("统计表","零件用量")
jb.AddTable("零件用量","产品","生产计划","产品")
jb.AddCols("零件",True)
jb.AddExp("需求数量","sum([生产数量] * [用量])")
```

```
jb.Build()
MainTable = Tables("统计表")
```

方法三

我们也可以自己直接编码实现，步骤：

1、需要事先设计一个名为“零件需求”的表，包括零件和需求数量两列。

2、然后新建一个菜单或窗口按钮，将其代码设置为:

```
Dim dic As new Dictionary(of String,Integer)
DataTables("零件需求").DataRows.Clear()
For Each dr1 As DataRow In DataTables("生产计划").DataRows
    For Each dr2 As DataRow In DataTables("零件用量").Select("[产品] = '" & dr1("产品") & "'")
        Dim nm As String = dr2("零件")
        Dim vl As Integer = dr1("生产数量") * dr2("用量")
        If dic.ContainsKey(nm) = False
            dic.Add(nm,vl)
        Else
            dic(nm) = dic(nm)+ vl
        End If
    Next
Next
For Each key As String In dic.keys
    Dim dr As DataRow = DataTables("零件需求").AddNew()
    dr("零件") = key
    dr("需求数量") = dic(key)
Next
```

提示：

1、方法一和方法二是直接统计后台数据，所以新增或修改数据后，应该先保存，然后再统计，才能得到正确的结果。

2、方法三显然比方法一和方法二繁琐很多，之所以提供这个方法，目的是让大家再次熟悉一下字典（Dictionary）的用法。

4.5.20.5 百分比与统计

假定有下图所示的一个数据表：

	日期	产品	生产数	成品数	成品率
1	2012-06-05	PD01	100	90	90%
2	2012-06-05	PD02	120	80	66.67%
3	2012-06-05	PD03	90	75	83.33%
4	2012-06-12	PD01	100	76	76%
5	2012-06-12	PD02	20	16	80%
6	2012-06-12	PD03	180	150	83.33%

希望按日期生成汇总模式，统计出每日的生产数、成品数和成品率：

	日期	产品	生产数	成品数	成品率
1	2012-06-05	PD01	100	90	90%
2	2012-06-05	PD02	120	80	66.67%
3	2012-06-05	PD03	90	75	83.33%
	5日 小计		310	245	79.03%
4	2012-06-12	PD01	100	76	76%
5	2012-06-12	PD02	20	16	80%
6	2012-06-12	PD03	180	150	83.33%
	12日 小计		300	242	80.67%
	总计		610	487	79.84%

显然，生产数和成品数，可以按常规统计出来，但是成品率却没有没办法直接统计得出，只能在得到生产数和成品数的统计结果后，据此计算得出成品率，代码如下：

```
Dim t As Table = Tables("订单")
Dim g As Subtotalgroup
t.SubtotalGroups.Clear()
t.GroupAboveData = False
t.TreeVisible = False
t.SpillNode = True

g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "日期"
g.TotalOn = "生产数,成品数"
g.Caption = "{0}日 小计"
g.DateGroup = DateGroupEnum.Day
t.SubtotalGroups.Add(g)

g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "*"
g.TotalOn = "生产数,成品数"
g.Caption = "总计"
t.SubtotalGroups.Add(g)

t.Subtotal()

Dim r As Row
For i As Integer = 0 To t.Rows.Count(True) - 1
    r = t.Rows(i,True)
    If r.IsGroup Then
        r("成品率") = r("成品数") / r("生产数")
    End If
Next
```

同样，如果要得到下图所示的分组统计结果：

	日期	生产数	成品数	成品率
1	2012-06-05	310	245	79.03%
2	2012-06-12	300	242	80.67%
3	合计	610	487	79.84%

也是没有办法直接分组统计得出的，但是可以利用分组统计得出生产数和成品数，然后在生成的统计表中增加一个表达式列，计算出成品率：

```
Dim g As New GroupTableBuilder("统计表1", DataTables("订单"))
g.Groups.AddDef("日期", DateGroupEnum.None)
g.Totals.AddDef("生产数")
g.Totals.AddDef("成品数")
g.VerticalTotal = True
g.Build()
DataTables("统计表1").DataCols.Add("成品率",Gettype(Double),"成品数/生产数")
DataTables("统计表1").DataCols("成品率").SetFormat("#0.00%")
MainTable = Tables("统计表1")
```

4.5.20.6 汇总模式下的行

关于普通模式和汇总模式下的行，之前说得比较零散，现在来总结一下。

不考虑分组行

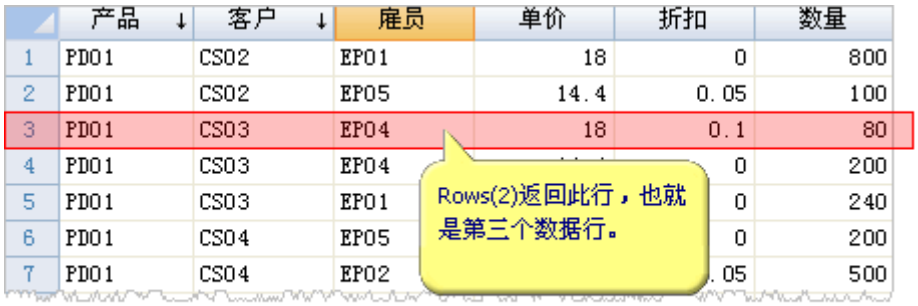
Foxtable 的代码默认是不考虑分组行的，我们不用担心汇总模式的出现，会影响代码的正常运行，下面我们逐个回顾一下这方面的特性。

首先看看 Rows 集合，该集合可以返回 Table 中指定位置的行，例如：

```
Dim r As Row = Tables("订单").Rows(2)
```

变量 r 将等于订单表的第 3 个数据行，因为行号是从 0 开始计算的，所以 2 表示第 3 个数据行。

如果 Table 处于汇总模式下，Rows 在计算行位置的时候，并不会包括分组行。



	产品	客户	雇员	单价	折扣	数量
1	PD01	CS02	EP01	18	0	800
2	PD01	CS02	EP05	14.4	0.05	100
3	PD01	CS03	EP04	18	0.1	80
4	PD01	CS03	EP04		0	200
5	PD01	CS03	EP01		0	240
6	PD01	CS04	EP05		0	200
7	PD01	CS04	EP02		.05	500



	产品	客户	雇员	单价	折扣	数量
1	PD01	CS02	EP01	18	0	800
2	PD01	CS02	EP05	14.4	0.05	100
	CS02 小计					900
3	PD01	CS03	EP04	18	0.1	80
4	PD01	CS03	EP04			200
5	PD01	CS03	EP01			240
	CS03 小计					520
6	PD01	CS04	EP05			200
7	PD01	CS04	EP02			500

Rows 的 Count 属性返回的是数据行的行数，也是排除分组行的，你可以在命令窗口执行下面的代码：

```
Output.Show(Tables("订单").Rows.Count)
```

不管订单表是在汇总模式还是普通模式下，上述代码的执行结果都是一样的，因为 Count 返回的始终是数据行的行数。

因为上述特性，所以下面的代码，不管在何种模式下，都能够正常地运行：

```
Dim r As Row
With Tables("订单")
    For i As Integer = 0 To .Rows.Count - 1
        r = .Rows(i)
        If r("数量") > 200 And r("日期") >= #1/26/1999# Then
            r("折扣") = 0.13
        End If
    Next
End With
```

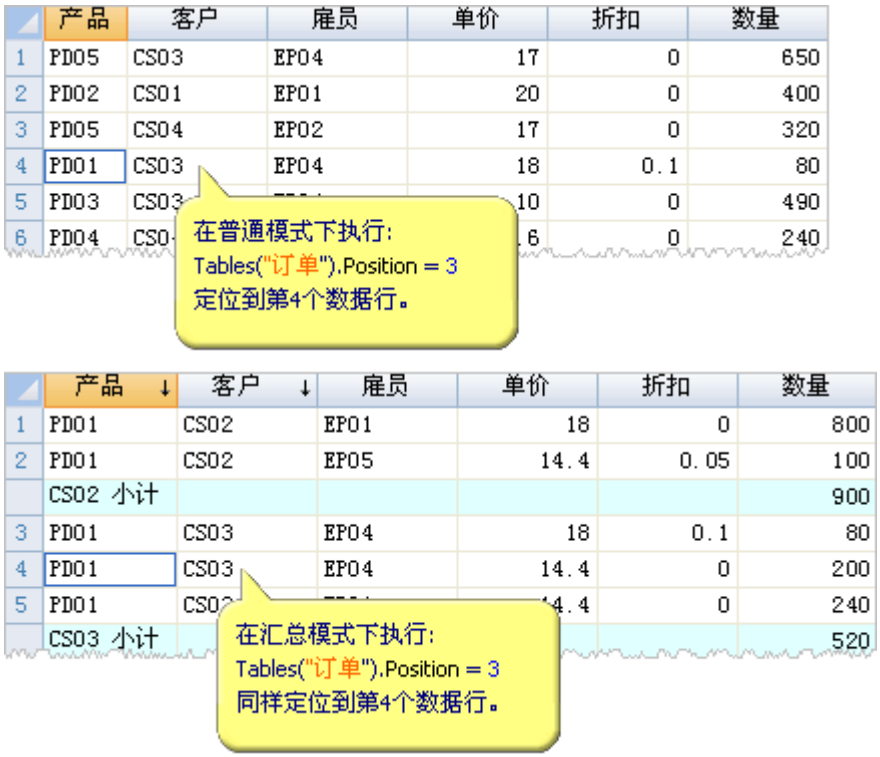
用 For Each 语句遍历 Rows 集合时，一样是忽略分组行的，所以上面的代码等效于：

```
For Each r As Row In Tables("订单").Rows
    If r("数量") > 200 And r("日期") >= #1/26/1999# Then
        r("折扣") = 0.13
    End If
Next
```

同样，表示选定行位置的 Position 属性是不考虑分组行的，下面的代码：

```
Tables("订单").Position = 3
```

执行后，不管 Table 是否处于汇总模式下，都能准确地定位到第 4 个数据行(行号从 0 开始计算，所以 3 表示第 4 个数据行)：



另两个重要的属性，TopPosition 和 BottomPosition，分别表示选定区域的第一个数据行位置和最后一个数据行的位置，它们和 Position 属性一样，在计算行位置的时候，是不考虑分组行的。

例如要锁定用户选定的一行或多行：

```
With Tables("订单")
    For i as Integer = .TopPosition To .BottomPosition
        .Rows(i).Locked = True
    Next
End With
```

上述代码，不管是在汇总模式下，还是普通模式下，都能正常地锁定选定的一行或多行，因为代码自动排除了分组行。

但是有一个例外，如果当前选定的是一行，而且这一行恰好是分组行，那么 Position、TopPosition、BottomPosition 三个属性都返回-1，导致上述代码运行出错。

所以比较“完美”的代码是：

```
With Tables("订单")
    If .TopPosition > -1 Then '如果选定区域包括数据行
        For i as Integer = .TopPosition To .BottomPosition
            .Rows(i).Locked = true
        Next
    End If
End With
```

此外，Table 两个用于查找的方法 Find 和 FindRow，其返回的行位置，也是不考虑分组行的；这样我们可以直接用 Position 定位 于查找到的行。

例如在第一列查找“abc”，不区分大小写，不用完全匹配：

```
With CurrentTable
    Dim r As Integer
    r = .Find("abc", 0, 0, False, False, True)
```

```
    If r > - 1 Then '如果找到符合条件的行
        .Position = r '则选择该行
    End If
End With
```

同样不管是在汇总模式下，还是普通模式下，上述的代码都能准确地查找定位，因为 Find 方法和 Position 都是不考虑分组行的。

绝大多数时候，我们的代码都是用于处理数据行，而不是分组行的，所以 Foxtable 这种自动忽略分组行的特性，能够大大简化我们的编码工作量，我们只需按照常规方式编写代码，无需担心汇总模式的存在，会影响 了代码的正常运行，更无需为兼容汇总模式，而另外编写代码。

考虑分组行

尽管实际编写代码的时候，要处理分组行的情况很少，但是我们还是介绍一下用代码处理分组行的方法。

前面提到，在汇总模式下，Rows 集合在返回指定位置的行的时候，是不考虑分组行的。

如果要包括分组行，可以给 Rows 加上一个参数 True，例如：

```
Dim r As Row = Tables("订单").Rows(2, True)
```

为了测试这种特性，我们首先让订单表进入汇总模式：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
2	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
	CS02 小计					900	12960	
3	PD01	CS03	EP04	18				1999-01-04
4	PD01			14.4				1999-01-14
5	PD01			14.4				1999-01-21
	CS03					520	6840	
6	PD01	CS04	EP05	14.4	0	900	2880	1999-01-02

Rows(2)表示这一行。

Rows(2,True) 表示这个分组行。

然后在命令窗口执行下面的代码：

```
Dim r1 As Row = Tables("订单").Rows(2)
Dim r2 As Row = Tables("订单").Rows(2,True)
Output.Show("r1: " & r1("产品"))
Output.Show("r2: " & r2("产品"))
```

执行结果：

```
r1: PD01
r2: CS02  小计
```

可以看出 Rows(2)本来应该返回第三行，但是因为第三行是分组行，所以被排除在外，实际返回的是第四行，也就是第三个数据行。

Rows(2,True)在计算行位置的时候，包括了分组行，所以返回的是第三行数据，尽管第三行是分组行。

Rows 的 Count 属性返回总的行数，同样默认是不包括分组行的，如果要包括分组行，可以给 Count 加上一个参数 True，例如：

```
Dim c1 As Integer = Tables("订单").Count
Dim c2 As Integer = Tables("订单").Count(True)
Output.show("c1 = " & c1)
Output.show("c2 = " & c2)
```

在汇总模式下执行上述代码，可以发现 c1 等于数据行的行数，没有包括分组行；而 c2 包括了分组行，等于数据行的行数加上分组行的行数，也就是总行数。

Row 有两个和汇总模式有关的属性：

名称	说明
IsGroup	逻辑型，判断该行是否是汇总模式下的分组行
Level	整数型，返回分组行的级别，总计行的 Level 属性为-1，第一个分组为 0，第二个分组为 1，其余依次类推。

例如：

```
If CurrentTable.Rows(4, True).IsGroup Then
    Messagebox.Show("第五行是分组行！")
End If
```

显然，在使用 IsGroup 属性的时候，Rows 必须加上参数 True，否则返回的永远是数据行，IsGroup 的值永远为 False，失去了判断的必要性。

下面举个处理分组行的例子，假定需要将分组行数量列的内容，全部除以 100：

```
Dim r As Row
With Tables("订单")
    For i As Integer = 0 To .Rows.Count(True) - 1 'Count 加上参数 True
        r = .Rows(i,True) 'Rows 也需要加上参数 True
        If r.IsGroup Then '如果是分组行
            r("数量") = r("数量") /100
        End If
    Next
End With
```

RowSel、TopRow、BottomRow 三个属性分别表示当前行位置、选定区第一行位置、选定区最后一行位置；这三个属性和 Position、TopPosition、BottomPosition 是对应的，不同的是前者在计算位置的时候，是考虑分组行的，而后者是不考虑分组行的。实际上 RowSel、TopRow、BottomRow 三个属性使用的场合非常有限，通常用于 Select 和 Aggregate 两个方法，因为只有这两个方法的参数，是考虑分组行的。

例如，我们利用 FindRow 找出某个符合条件的行后，可以用 Position 属性定位到这一行，如果我们希望不仅定位到该行，而且定位到该行的某一列，就必须使用 Select 方法和 RowSel 属性 了。下面的代码，找出订购产品 PD01 且数量大于 100 的行，然后定位到该行的日期列：

```
With CurrentTable
    Dim r As Integer
    Dim p As Integer = .Cols("日期").Index '获得日期列的位置
    r = .FindRow("产品  = 'PD02' And  数量  > 100", 0, False)
    If r > - 1 Then '如果找到符合条件的行
        .Position = r '则选择该行
        .Select(.RowSel, p) '再定位到日期列
    End If
End With
```

上面的代码不管是汇总模式，还是普通模式，都能正常查找和定位。

如果直接将代码写为：

```
With CurrentTable
    Dim r As Integer
    Dim p As Integer = .Cols("日期").Index '获得日期列的位置
```

```
    r = .FindRow("产品  = 'PD02' And  数量  > 100", 0, False)
    If r > - 1 Then '如果找到符合条件的行
        .Select(.Position, p)
    End If
End With
```

在普通模式下，上面的代码没有问题，因为此时 Position 和 RowSel 的值是相等的，但是在汇总模式下， 因为 Position 的值并不一定等于 RowSel，所以无法正常定位。

再来看看选定整列的代码，例如选定数量列，之前介绍的代码是：

```
With Tables("订单")
    Dim c As Integer= .Cols("数量").Index
    .Select(0, c, .Rows.Count - 1, c)
End With
```

这个代码是有缺陷的，要保证在汇总模式下也能正常选定这个数量列，代码应该改为：

```
With Tables("订单")
    Dim c As Integer= .Cols("数量").Index
    .Select(0, c, .Rows.Count(True) - 1, c)
End With
```

Aggregate 方法的参数，同样是考虑分组行的，例如要统计当前表选定区域的累积值:

```
Dim Sum As Double
With CurrentTable
    Sum = .Aggregate(AggregateEnum.Sum, .TopRow, .LeftCol, .BottomRow, .RightCol)
End With
OutPut.Show(Sum)
```

4.5.20.7 篡改汇总模式的结果

有了上一节“关于汇总模式下的行”的知识，我们就可以轻松地“篡改”汇总模式的统计结果。

示例一

例如希望汇总模式的统计结果中，金额以“万元”为单位显示：

```
'首先要生成汇总模式
Dim r As Row
For i As Integer = 0 To Tables("订单").Rows.Count(True) - 1
    r = Tables("订单").Rows(i,True)
    If r.IsGroup
        r("金额") = r("金额") / 10000 & "万元"
    End If
Next
```

示例二

假定要生成下图所示的汇总模式，显然常规的方法是不可能的：

	分组1	分组2	金额
1	A	东	1
2	A	南	2
3	A	西	3
4	A	北	4
5	A	东	9
	A 小计		19
6	B	东	5
7	B	南	6
8	B	西	7
9	B	北	8
10	B	南	10
	B 小计		36
	总计	北	12
	总计	东	15
	总计	南	18
	总计	西	10

有了上一节的知识，我们可以用如下的代码实现，望大家仔细体会：

```
Dim t As Table = Tables("表A")
Dim g As Subtotalgroup
t.SubtotalGroups.Clear()

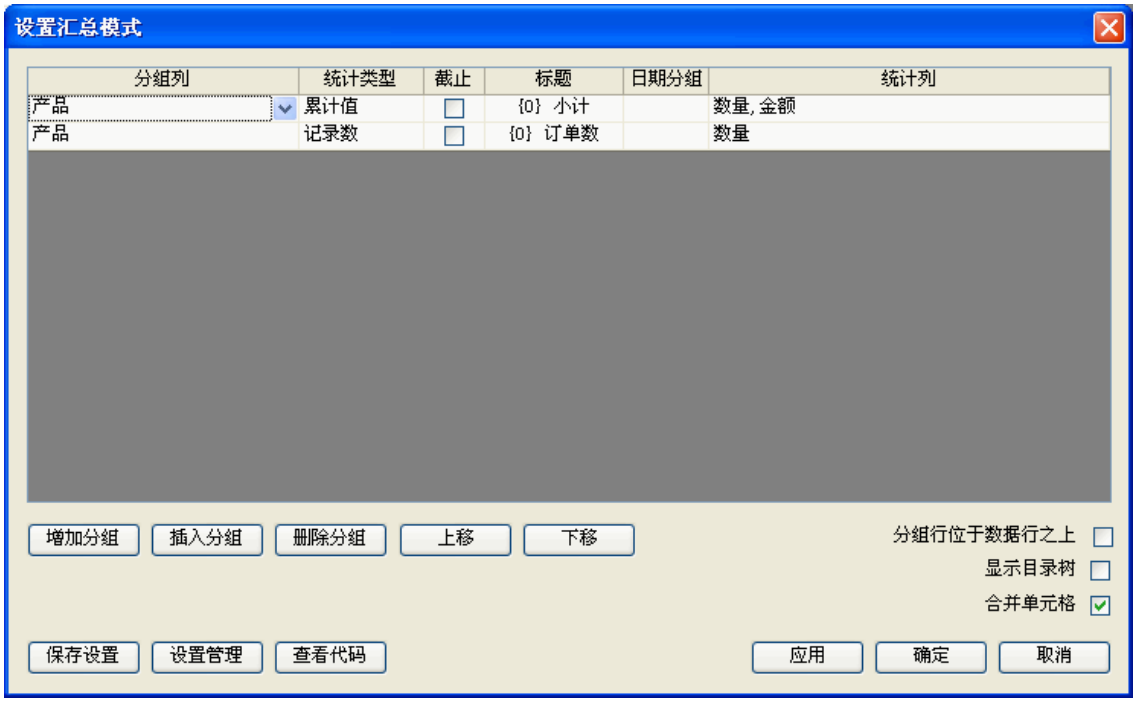
g = New Subtotalgroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "分组1"
g.TotalOn = "金额"
g.Caption = "{0} 小计"
t.SubtotalGroups.Add(g)

Dim groups As List(of String) = DataTables("表A").GetValues("分组2")
For Each grp As String In groups
    g = New Subtotalgroup
    g.Aggregate = AggregateEnum.Sum
    g.GroupOn = "*"
    g.Caption = "总计"
    t.SubtotalGroups.Add(g)
Next
t.Subtotal()
```

```
Dim cnt As Integer = Tables("表A").Rows.Count(True) - groups.count
For Each group As String In groups
    Tables("表A").Rows(cnt,True)("分组2") = group
    Tables("表A").Rows(cnt,True)("金额") = Tables("表A").Compute("Sum(金额)","分组2 = '" &group & "'")
    cnt = cnt +1
Next
```

4.5.20.8 自己定义汇总模式

假定我们需要根据产品进行分组，统计每个产品的订单数，以及订购数量和订购金额。
按照常规，我们应该这样设置：



得到的统计结果为：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
2	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
3	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
4	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10
5	PD01	CS03	EP04	14.4	0.25	200	2160	1999-01-14
6	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
7	PD01	CS03	EP01	14.4	0	240	3456	1999-01-21
	PD01 小计					2120	29520	
	PD01 订单数					7		
8	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
9	PD02	CS05	EP04	19	0.1	200	3420	1999-01-13

过程很简单，但是得到的结果不是很美观。
下图的统计结果可能更符合大多数人的习惯：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
2	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
3	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
4	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10
5	PD01	CS03	EP04	14.4	0.25	200	2160	1999-01-14
6	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
7	PD01	CS03	EP01	14.4	0	240	3456	1999-01-21
	产品:PD01	订单数:7	订购数量:2120	订购金额:29520				
8	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
9	PD02	CS05	EP04	19	0.1	200	3420	1999-01-13
10	PD02	CS01	EP04	15.2	0	100	1520	1999-01-14
11	PD02	CS05	EP04	15.2	0	400	6080	1999-01-14
12	PD02	CS03	EP01	15.2	0.1	200	2736	1999-01-17
	产品:PD02	订单数:5	订购数量:1300	订购金额:20956				
13	PD03	CS03	EP04	10	0	490	4900	1999-01-04
14	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08
15	PD03	CS01	EP04	8	0	300	2400	1999-01-11

要实现上面的统计结果，代码并不复杂：

```
Dim t As Table = Tables("订单")
Dim g As SubtotalGroup
t.SubtotalGroups.Clear()
t.GroupAboveData = False
t.TreeVisible = False
g = New SubtotalGroup
g.Aggregate = AggregateEnum.Sum
g.GroupOn = "产品"
g.Caption = "{0}"
t.SubtotalGroups.Add(g)
t.Subtotal()
Dim r As Row
For i As Integer = 0 To t.Rows.Count(True) - 1
    r = t.Rows(i,True)
    If r.IsGroup '如果是分组行
        Dim f As String = "产品  = '" & r("产品") & "'"
        Dim v As String = "  订单数:" & t.Compute("Count(日期)",f)
        v = v & "  订购数量:" & t.Compute("Sum(数量)",f)
        v = v & "  订购金额:" & t.Compute("Sum(金额)",f)
        r("产品") = "产品:" & r("产品") & v
    End If
Next
```

上面的代码分成两部分，第一段设置汇总模式，需要注意的是，这里只是指定了分组列，没有指定统计列，同时分组标题简单设置为“{0}”，这样执行这段代码后，会得到下图所示的汇总模式：

	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD01	CS03	EP04	18	0.15	80	1224	1999-01-04
2	PD01	CS04	EP05	14.4	0	200	2880	1999-01-08
3	PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
4	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-10
5	PD01	CS03	EP04	14.4	0.25	200	2160	1999-01-14
6	PD01	CS02	EP05	14.4	0	100	1440	1999-01-20
7	PD01	CS03	EP01	14.4	0	240	3456	1999-01-21
	PD01							
8	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
9	PD02	CS05	EP04	19	0.1	200	3420	1999-01-13
10	PD02	CS01	EP04	15.2	0	100	1520	1999-01-14
11	PD02	CS05	EP04	15.2	0	400	6080	1999-01-14
12	PD02	CS03	EP01	15.2	0.1	200	2736	1999-01-17
	PD02							
13	PD03	CS03	EP04	10	0	490	4900	1999-01-04
14	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08
15	PD03	CS01				300		1999-01-1

然后第二段代码遍历所有行，如果是分组行，则计算出该分组的订单数、订购数量和订购金额，并填入到表中。

4.6 Excel 报表

4.6.1 本章示例文件

本章课程使用的示例文件为“Excel 报表.Table”，位于安装目录的子目录“CaseStudy”下。
请在学习本章接下来的内容之前，先打开此文件。
需要注意的是，你的电脑必须安装有 Excel，才可以正常使用 Excel 报表功能。

4.6.2 基础应用

4.6.2.1 关于 Excel 报表

Excel 报表能够以现成的 Excel 文件作为模板，通过设置一些特殊的设置，自动从 Foxtable 的数据表中提取数据填入到 Excel 表格中，这样我们就可以利用 Excel 强大的制表和格式设置功能， 轻松设计出复杂的报表：



Excel 报表使用起来很轻松，效率远远超过传统的可视化报表设计器，而且无需任何专业知识，非常适合普通用户掌握应用。

Foxtable 还提供了 Word 报表功能，同样可以直接利用 Word 可视化设置报表模板，自动根据模板的设置从 Foxtable 提取数据填入到 Word 中，批量生成报表或文章。

Excel 报表和 Word 报表各有优势，两者结合使用，基本上能满足绝大多数场合的需要，大家可以根据需要选用。

为了照顾更专业用户的需求，Foxtable 还提供了专业报表功能，专业报表的功能更为强大，使用也相对复杂一些，所以专业报表的内容将完全在开发篇讲述。

对于一般用户来说，掌握 Excel 报表和 Word 报表足以应付日常工作的需要，无需再去研究专业报表。

我们接下来首先介绍 Excel 报表，然后是 Word 报表，至于专业报表，则在开发指南讲述。

4.6.2.2 细节区和数据引用

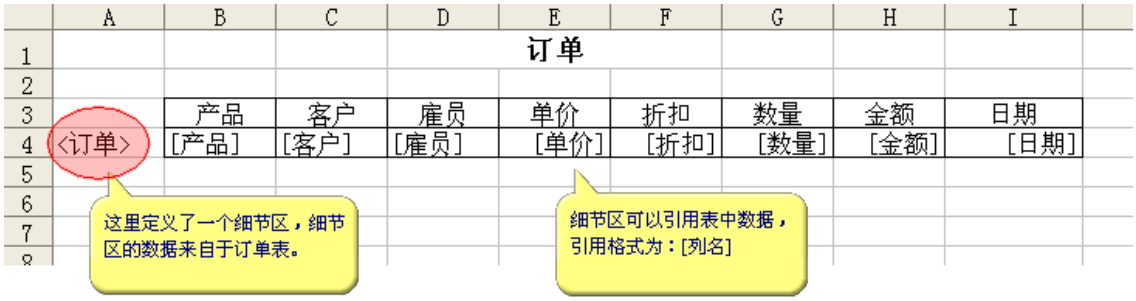
Excel 报表模板的第一列用于定义细节区，细节区的定义格式为：

〈表名〉

在细节区，可以引用表中的数据，引用数据的格式为：

[列名]

例如：

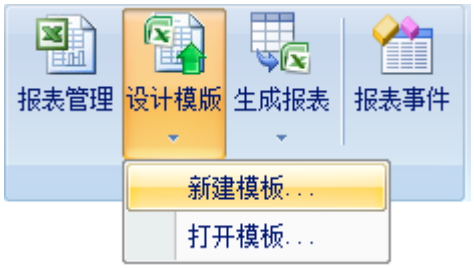


对于每一个数据行，都会重复生成一个细节区，例如上图的模板，可以生成下图所示的报表：

	A	B	C	D	E	F	G	H
1			订单					
2								
3	产品	客户	雇员	单价	折扣	数量	金额	日期
4	PD05	CS03	EP04	17	0.1	650	9945	36162
5	PD02	CS01	EP01	20	0.1	400	7200	36163
6	PD05	CS04	EP02	17	0	320	5440	36163
7	PD01	CS03	EP04	18	0.15	80	1224	36164
8	PD03	CS03	EP04	10	0	490	4900	36164
9	PD04	CS04	EP05	17.6	0	240	4224	36167
10	PD05	CS03	EP04	17	0	160	2720	36167
11	PD01	CS04	EP05	14.4	0	200	2880	36168
12	PD03	CS02	EP02	8	0.1	200	1440	36168
13	PD01	CS02	EP01	18	0.2	800	11520	36170
14	PD01	CS04	EP02	14.4	0.05	500	6840	36170
15	PD03	CS01	EP04	8	0	300	2400	36171
16	PD04	CS05	EP04	22	0.05	500	10450	36171
17	PD04	CS03	EP04	22	0	30	660	36172
18	PD02	CS05	EP04	19	0.1	200	3420	36173
19	PD01	CS03	EP04	14.4	0.25	200	2160	36174
20	PD02	CS01	EP04	15.2	0	100	1520	36174
21		CS05	EP04	15				74

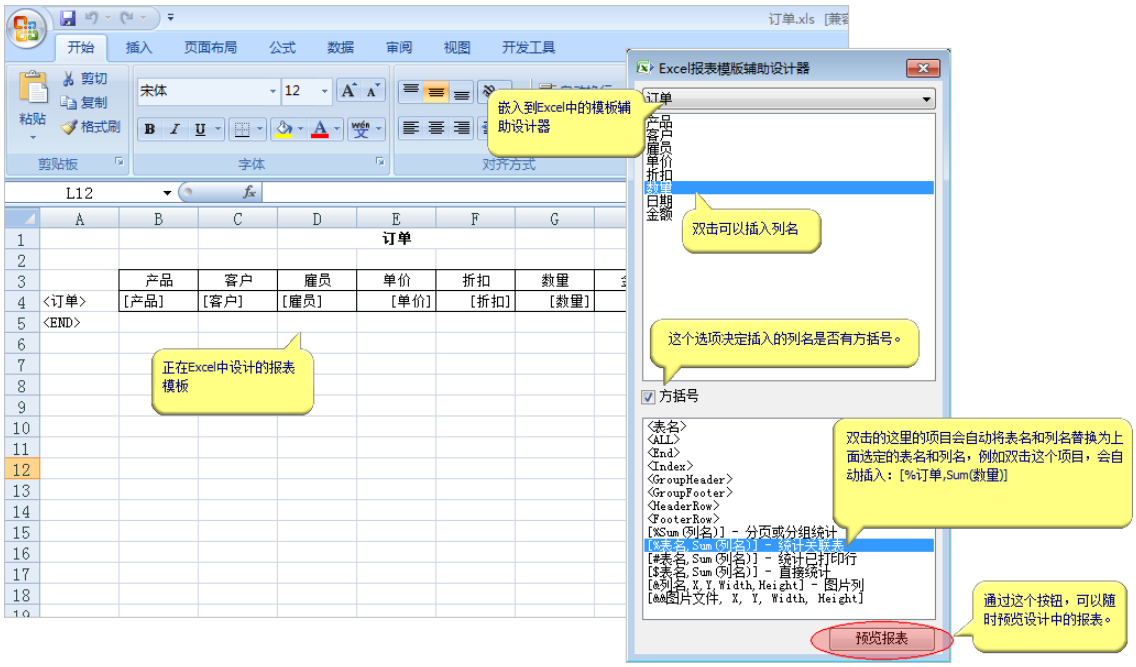
4.6.2.3 设计报表模板

在菜单的“打印输出”功能区的“Excel 报表”功能组，单击“设计模板”，可以新建报表模板，也可以打开现有模板修改：



所有的模版默认保存在项目所在目录的 Attachments 子目录中，为方便管理，最好不要改变这个存储目录。

模版设计器会启动 Excel，并在 Excel 中附加一个下图所示的辅助设计器，通过这个设计器，可以快速向 Excel 的活动单元格插入列名和各种标记，还可以随时预览设计中的报表：



有了这个辅助设计器，Excel 报表模板的设计将变得非常轻松，效率会提高 10 倍以上，准确度也会大大提高。

当然你也可以直接启动 Excel 来设计模版，将设计好的模板保存在 Attachments 目录就行，只是没有了辅助设计器，设计效率会大打折扣。

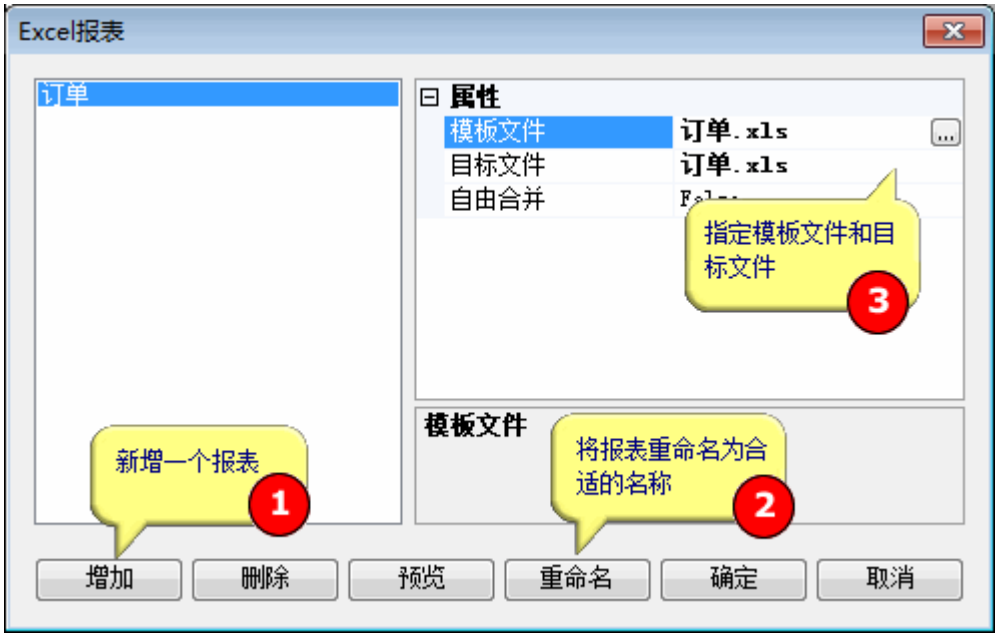
4.6.2.4 添加报表模板

设计好报表模版之后，我们需要将报表模版添加到菜单中。

在菜单的“打印输出”功能区的“Excel 报表”功能组，单击“报表管理”按钮：



会出现下图所示的窗口，通过这个窗口，可以添加已经设计好的报表模版：

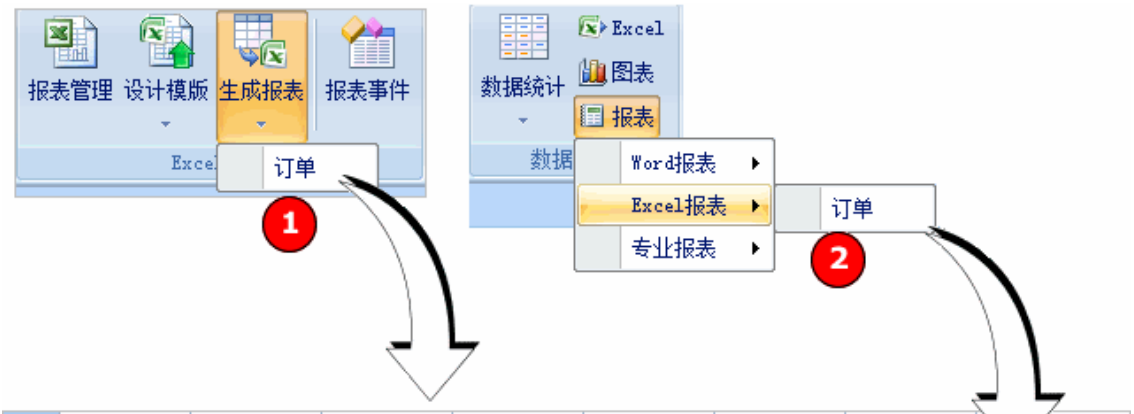


目标文件指的是根据模板文件生成的报表文件的文件名。

4.6.2.5 生成 Excel 报表

设计好 Excel 报表模版，并将模版添加到菜单后，就可以通过菜单来生成 Excel 报表了。
菜单中有两处地方可以生成 Excel 报表：
1、在菜单的“打印输出”功能区的“Excel 报表”功能组，单击“生成报表”按钮 ，会列出已经添加的模版，单击其中一个，即可生成报表。

2、在菜单的“日常工作”功能区的“数据统计”功能组，单击“报表”按钮，指向“Excel 报表”,也会列出已经添加的模版，单击其中一个，即可生成报表。



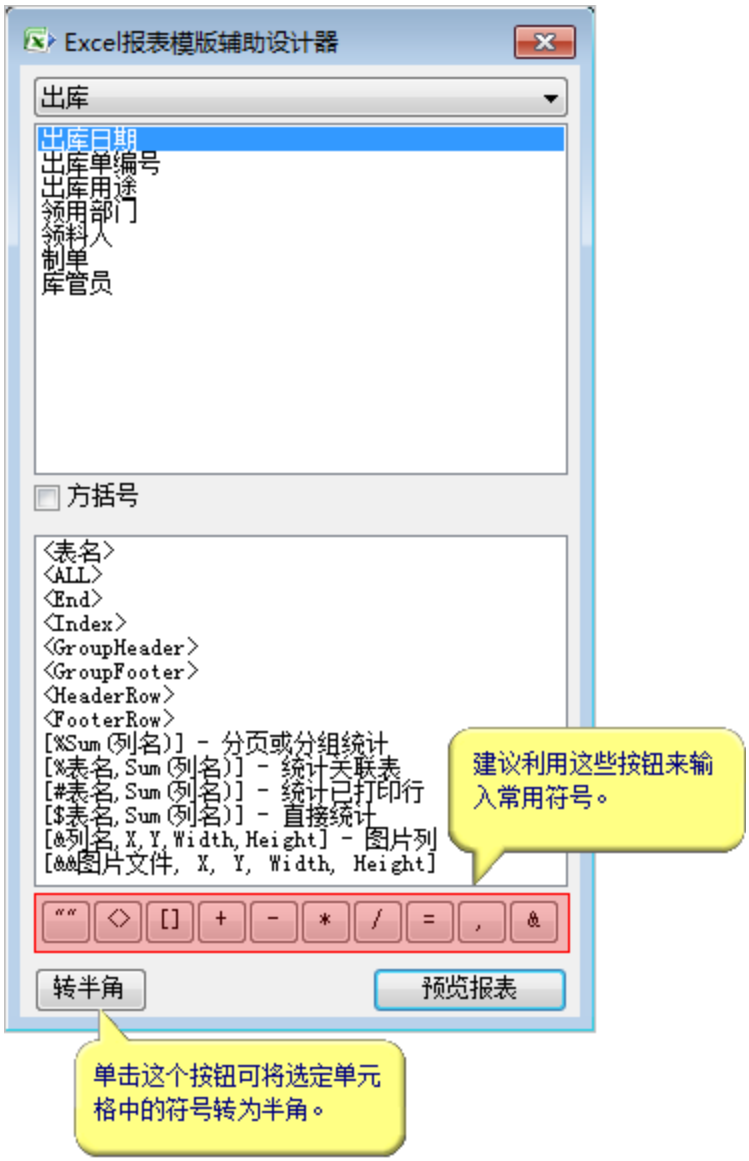
	A	B	C	D	E	F	G	H
1				订单				
2								
3	产品	客户	雇员	单价	折扣	数量	金额	日期
4	PD05	CS03	EP04	17	0.1	650	9945	1999/1/2
5	PD02	CS01	EP01	20	0.1	400	7200	1999/1/3
6	PD05	CS04	EP02	17	0	320	5440	1999/1/3
7	PD01	CS03	EP04	18	0.15	80	1224	1999/1/4
8	PD03	CS03	EP04	10	0	490	4900	1999/1/4
9	PD04	CS04	EP05	17.6	0	240	4224	2012/8/11
10	PD05	CS03	EP04	17	0	160	2720	2012/8/11
11	PD01	CS04	EP05	14.4	0	200	2880	1999/1/8
12	PD03	CS02	EP02	8	0.1	200	1440	1999/1/8
13	PD01	CS02	EP01	18	0.2	800	11520	1999/1/10
14	PD01	CS04	EP02	14.4	0.05	500	6840	2012/8/11
15	PD03	CS01	EP04	8	0	300	2400	2012/8/11
16	PD04	CS05	EP04	22	0.05	500	10450	1999/1/11
17	PD04	CS03	EP04	22	0	30	660	1999/1/12
18	PD02	CS05	EP04	19	0.1	200	3420	1999/1/13

系统会自动打开生成的报表文件，这些文件默认存放在目录目录的 Reports 子目录下。
小提示： 模版和表是有对应关系的，例如你在表 A 添加的报表模版，必须先选择表 A，才能生成报表。

4.6.2.6 符号要半角的

在设计 Excel 模板的时候，我们需要插入很多数据引用标记和表达式。

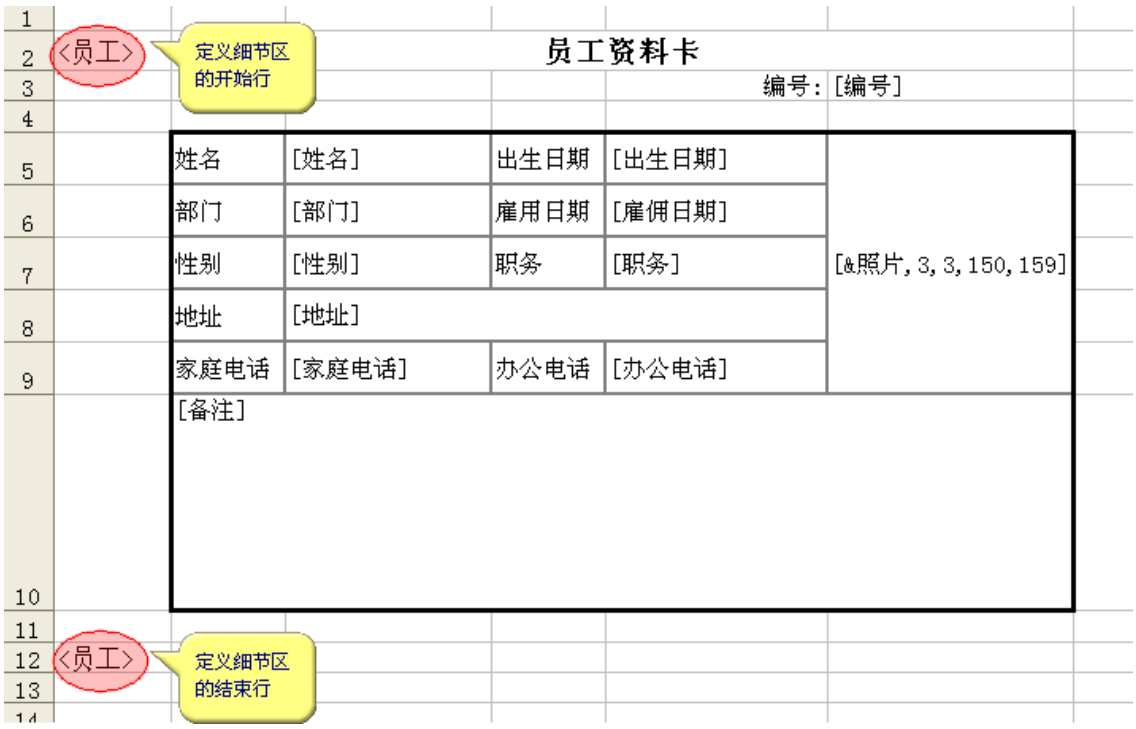
需要注意的是，标记中使用的符号必须是半角的，不能是全角的。
例如下面的表达式不会正确计算：
〈CLNum([金额])〉
正确的写法是：
<CLNum([金额])>
二者的差别在于前者首尾的尖括号时全角的，后者则是半角的。
使用汉字的我们，很难察觉这种错误，也很难避免这种错误，设计模板的一定要小心。
为了减少这种错误，Excel 模板辅助设计器后来加上了两个按钮，分别用于快速输入常用的半角符号 ，以及将选定单元格中的符号转换为半角：



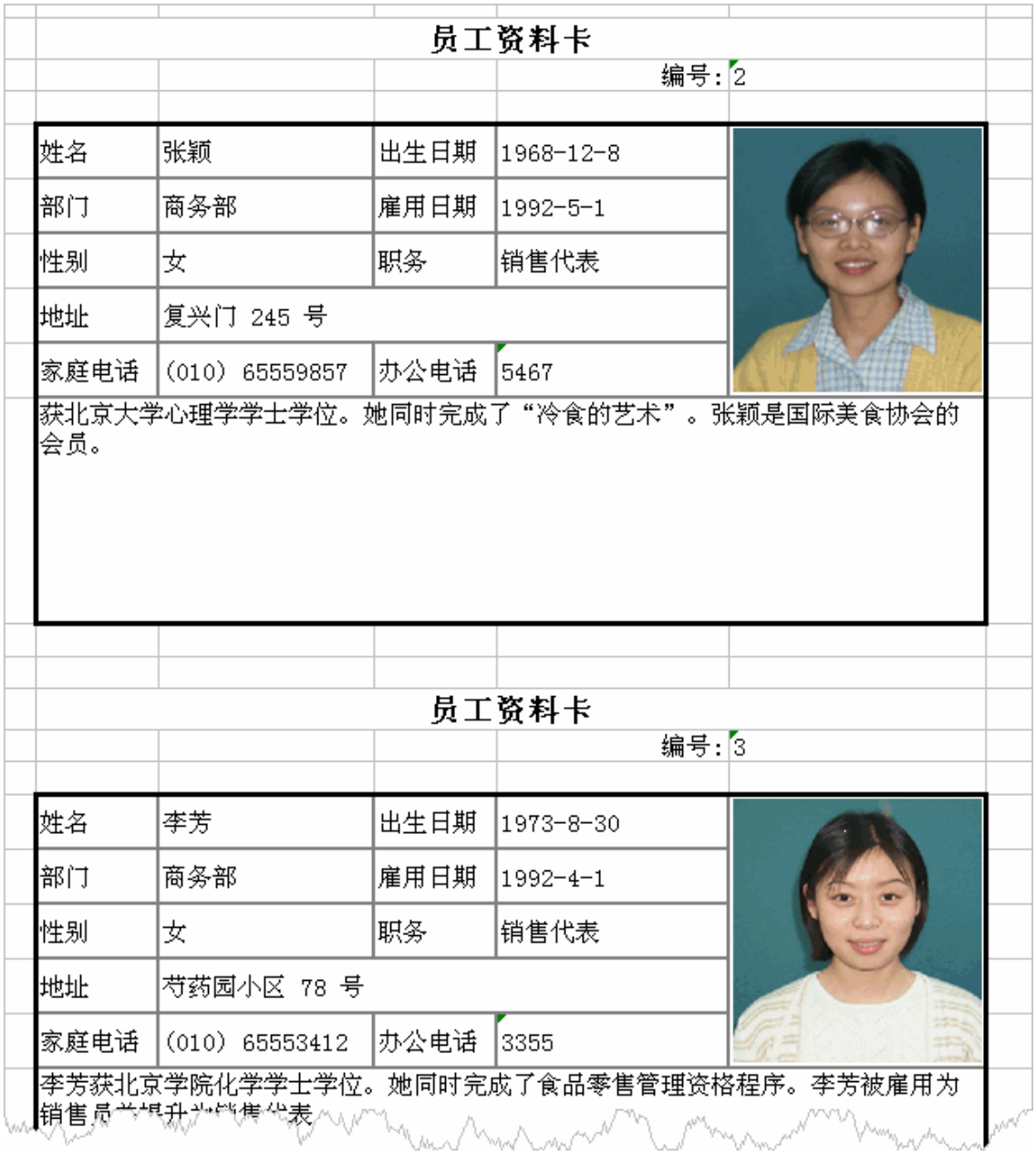
4.6.2.7 多行细节区

Excel 报表的细节区可以包括多行，而且可以任意复杂。

例如我们要 Excel 报表打印员工资料卡，可以按照下图所示设计模板：


<员工>
定义细节区的开始行
员工资料卡
编号：[编号]
姓名
[姓名]
出生日期
[出生日期]
部门
[部门]
雇用日期
[雇佣日期]
性别
[性别]
职务
[职务]
[&照片,3,3,150,159]
地址
[地址]
家庭电话
[家庭电话]
办公电话
[办公电话]
[备注]
<员工>
定义细节区的结束行

打印效果



员工资料卡

编号：2

姓名	张颖	出生日期	1968-12-8
部门	商务部	雇用日期	1992-5-1
性别	女	职务	销售代表
地址	复兴门 245 号		
家庭电话	(010) 65559857	办公电话	5467

获北京大学心理学学士学位。她同时完成了“冷食的艺术”。张颖是国际美食协会的会员。

员工资料卡

编号：3

姓名	李芳	出生日期	1973-8-30
部门	商务部	雇用日期	1992-4-1
性别	女	职务	销售代表
地址	芍药园小区 78 号		
家庭电话	(010) 65553412	办公电话	3355

李芳获北京学院化学学士学位。她同时完成了食品零售管理资格程序。李芳被雇用为销售员并提升为销售代表。

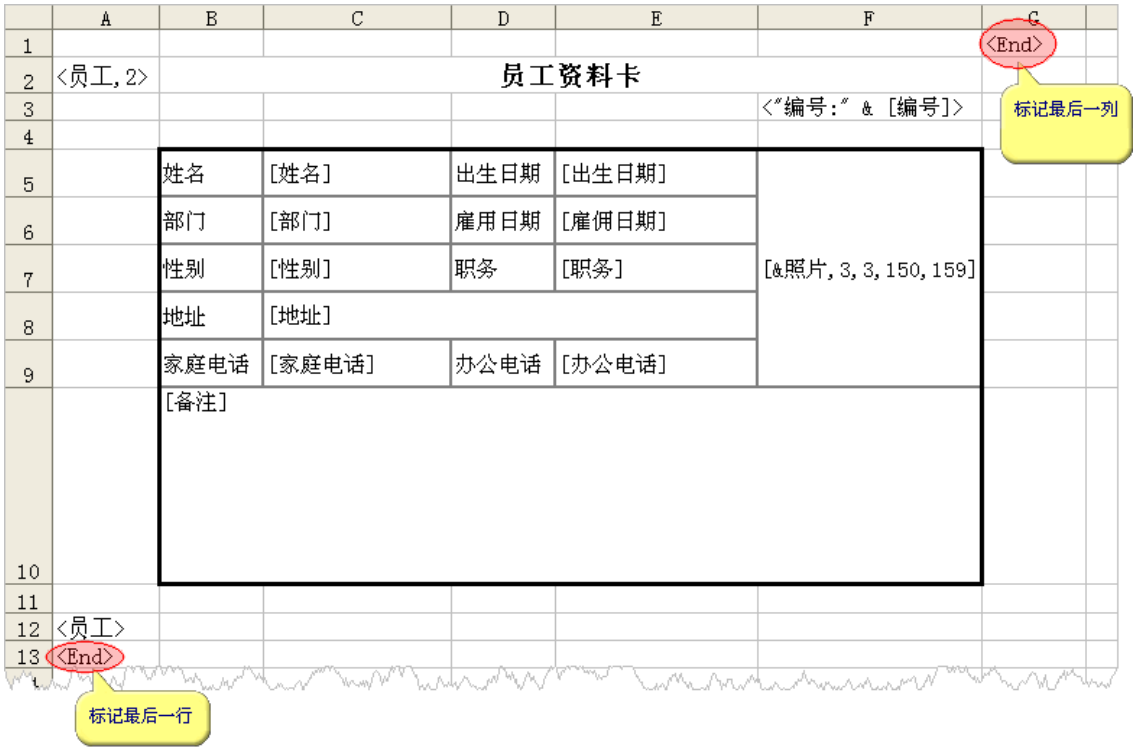
4.6.2.8 指定有效区域

在 Excel 报表的测试阶段，我们总是遇到一些用户反馈：在生成 Excel 报表的过程中死机。

其实并不是死机，而是他们在设计 Excel 报表模板的过程中，无意中设计了一个超大的表格，有的甚至达到 Excel 2007 的上限：16384 列。

为了避免这种意外的发生，你可以用<End>标记指定报表的有效区域。

在第一行的最后一列，以及第一列的最后一行，分别输入<End>标记即可：

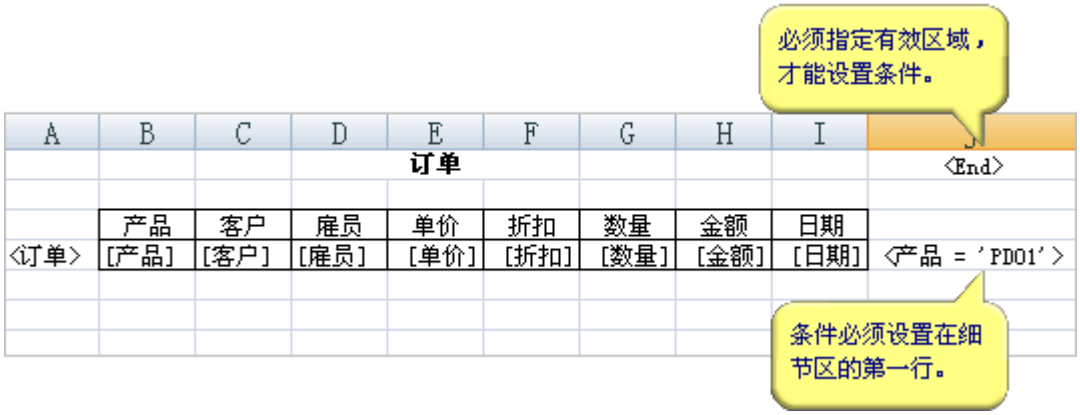


生成报表的时候，<End>标记之外的行列，包括<End>标记所在的行列，将会被忽略。

4.6.2.9 设置打印范围

需要注意的是，Excel 报表默认只会打印选定行(记录)，如果你只选定 1 行，就会打印 1 行，如果你选定了 10 行，就会打印 10 行。

可以设置一个条件表达式，例如只打印产品为 PD01 的行：



请参考表达式的 运算符和函数 以及 条件表达式

特别强调一下：必须设置有效区域，打印条件才会生效。

如果要打印所有行，可以按下图所示设置：

A	B	C	D	E	F	G	H	I	J
				订单					<End>
	产品	客户	雇员	单价	折扣	数量	金额	日期	
<订单>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>

ALL表示打印所有行。

注意这个 ALL 不是一个条件表达式，只是告诉系统：没有条件，全部打印。

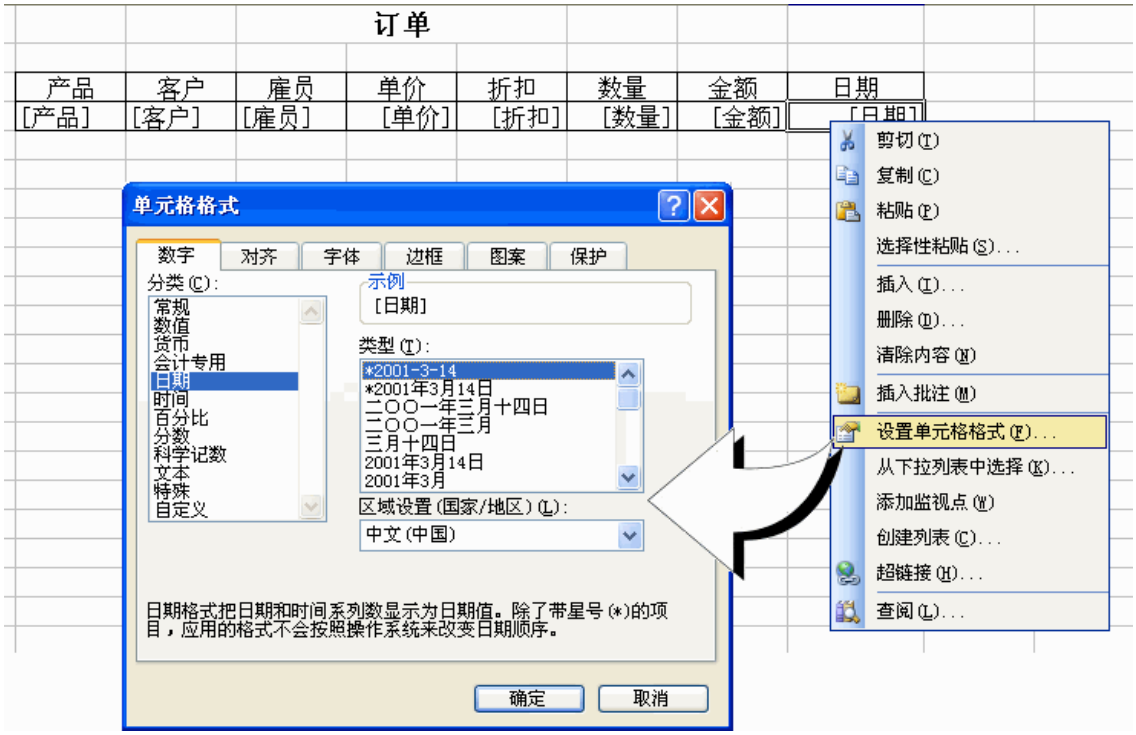
4.6.2.10 关于日期列

也许你已经留意到，前面打印出来的报表，日期列的内容全部变成了数字：

订单							
产品	客户	雇员	单价	折扣	数量	金额	日期
PD05	CS04	EP02	17	0	320	5440	36163
PD01	CS03	EP04	18	0.15	80	1224	36164
PD03	CS03	EP04	10	0	490	4900	36164
PD04	CS04	EP05	17.6	0	240	4224	36167
PD05	CS03	EP04	17	0	160	2720	36167
PD01	CS04	EP05	14.4	0	200	2880	36168
PD03	CS02	EP02	8	0.1	200	14	3
PD01	CS02	EP01	18	0.2	800	115	0

日期变成了数字

为避免这种情况的发生，在设计报表模板的时候，右击日期单元格，从快捷菜单中执行“单元格格式”命令，然后选择合适的日期格式即可：



4.6.2.11 换页控制

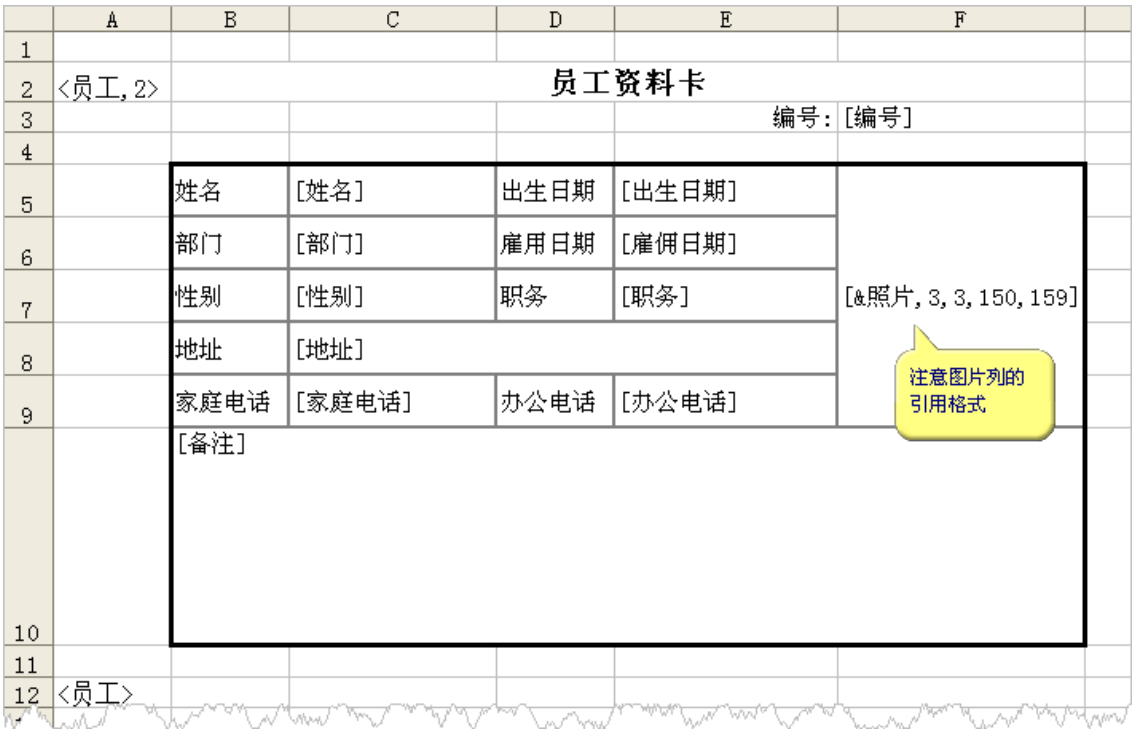
前一节已经讲述如何利用多行细节区来打印员工资料卡。
由于一个页面可以打印多个资料卡，这样同一个资料卡可能跨越前后两个页面，显然这是我们不希望看到的。
我们可以在定义细节区开始行的时候，增加一个参数，用于指定单个页面可打印细节区的个数，格式为：
<表名,n>
一旦某一页面的细节区数量达到参数 n 的设定值，将自动插入一个换页符。
例如：



在上图中，我们指定每页打印资料卡 2 个，这样就不会出现资料卡因为换页而一分为二的情况。

4.6.2.12 引用图片列

也许你已经留意到，前面的员工资料卡，正确地打印出了每个员工的照片。
引用图片列的格式为：
[&列名,X,Y,Width,Height]
X: 左边距
Y： 上边距
Width： 图片宽度
Height：图片高度
上述参数的单位为像素。
例如：



还有另外两种简化引用格式：

[&列名]

或

[&列名,X,Y]

我们知道，图片列的一个单元格可能包括多个图片，上述的引用格式打印的是第一个图片。

我们可以指定打印第几个图片，方法是在列名后面加上“&X”，X 是一个整数，表示要打印的图片的位置，从 0 开始编号。

例如：

[&照片&1,3,3,150,159]

表示打印第二个图片。

再例如打印第三个图片：

[&照片&2,3,3,150,159]

4.6.2.13 按比例缩放图片

我们已经知道，引用图片列的格式为：

[&列名,X,Y,Width,Height]

X: 左边距

Y： 上边距

Width： 图片宽度

Height：图片高度

有的时候，我们可能不需要同时指定高度或宽度，而是指定其中一个值，另一个值按比例生成。

例如：

[&照片,3,3,120,-1]

表示宽度固定为 120，高度按比例生成，如果图片原本的宽度和高度分别为 180 和 150，那么打印出来的图片，宽度是 120，高度是 100。

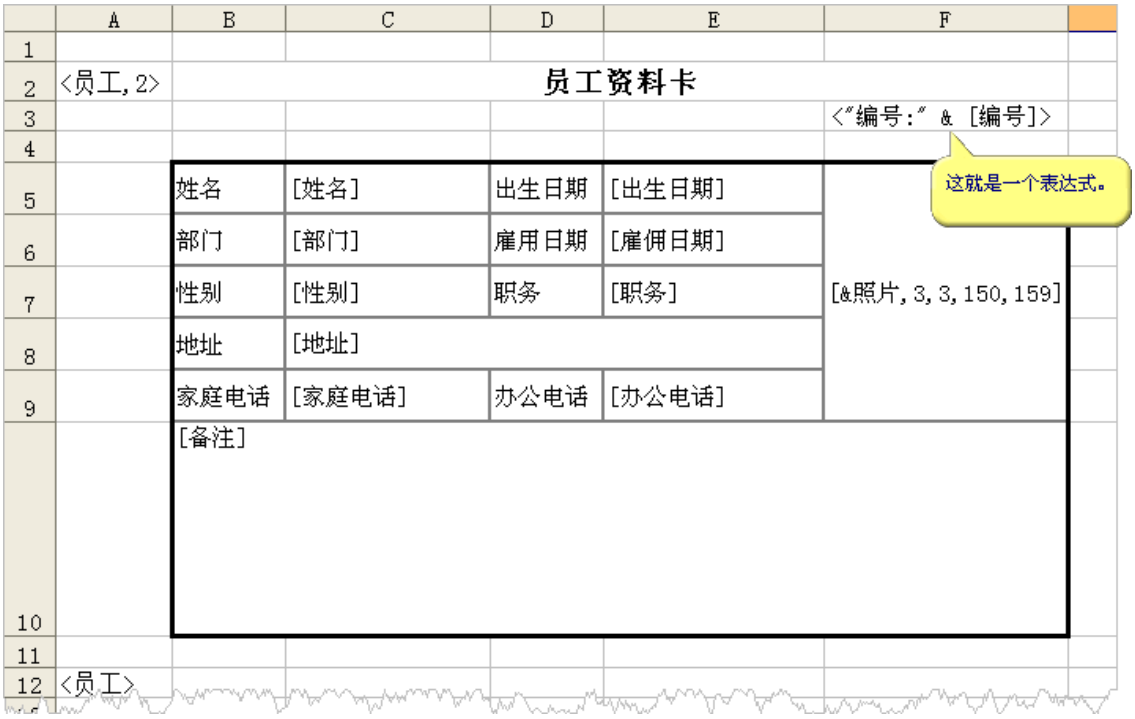
再例如：
[&照片,3,3,-1,100]
表示高度固定为 100，宽度按比例生成，如果图片原本的宽度和高度分别为 300 和 200，那么打印出来的图片，宽度是 150，高度是 100。

4.6.2.14 使用表达式

可以在 Excel 报表中直接进行表达式计算，表达式必须用符号"<"和">"括起来，例如：
<[数量] * [单价]>

<[语文] + [英语] + [数学]>

<"编号:" & [编号]>
员工资料卡中的编号，就是利用表达式来打印的：



表达式中的列名必须用方括号括起来，例如：<"编号：" & [编号]>，如果没有方括号：<"编号：" & 编号>，表达式将不能计算。

表达式采用 VBScript 语法，而且 Foxtable 对 VBScript 进行了扩展，提供了很多非常实用的函数。例如你要将出生日期以中文大写的形式打印出来，可以这样设置模板：<CUDATE([出生日期])>
再例如，假定已付款是个逻辑列，希望打印的时候，已付款的用符号"√"，可以这样设置模板：IIF([已付款], "√", "")
下面介绍一些常用的函数，更多的函数和更具体的用法，请参考：VBSCript 参考

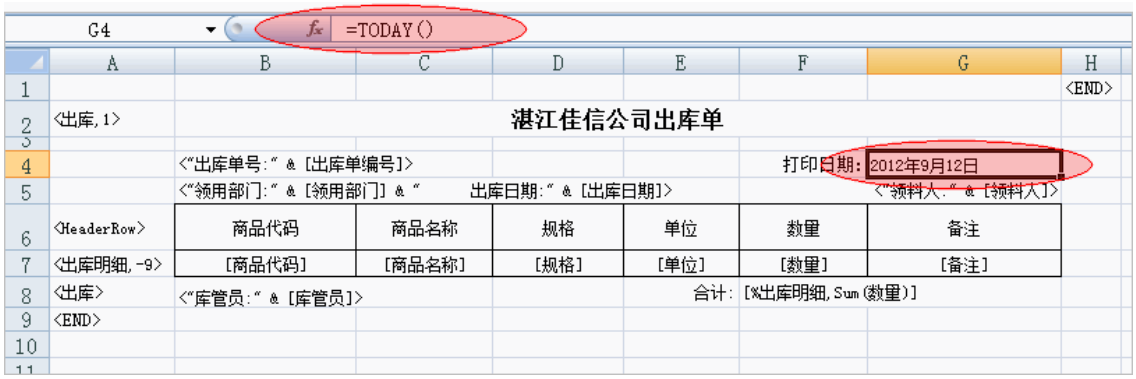
函数名	说明
Format	格式化日期或数字。

	格式化数字的时候，用 0 表示占位符，如果指定位置有数字，则显示数字，如果没有数字，则用 0 补充。 例如需要金额保留两位小数显示： <Format([金额],"#0.00")> 如果金额的值是 123.567，则显示为 123.57，如果金额的值是 123.5，则显示为 123.50，而不是 123.5。 如果希望以百分比的形式显示，可以加上符号%，例如希望以百分比形式显示合格率，保留一位小数： <Format([合格率],"#0.0%")> 格式化日期的时候，yyyy 表示四位年，yy 表示两位年，MM 表两 2 位月,M 表示一位月，dd 表示两位日,d 表示一位日，例如： <Format([出库日期],"yyyy 年 MM 月 dd 日")> 显示的日期格式为：2013 年 09 月 08 日 <Format([出库日期],"yyyy 年 M 月 d 日")> 显示的日期格式为：2013 年 9 月 8 日 <Format([出库日期],"yy 年 M 月 d 日")> 显示的日期格式为：13 年 9 月 8 日 注意 M 是大写。
IIF	语法为： IIF(expression, truevalue, falsevalue) 如果逻辑表达式 expression 的计算结果为 True，将返回 truevalue，否则返回 falsevalue。 例如： IIF([已付款], " √ ", "")
Round	对数值进行四舍五入，例如：<Round([金额],2)>
Date	返回当前日期，例如：<Format(Date,"yyyy 年 MM 月 dd 日")> 日期可以直接和数字进行加减，例如： <Date - 2> 表示前天的日期 <Date + 1> 表示明天的日期 <Format(Date + 1,"yyyy 年 MM 月 dd 日")> 表示以四位年两位月两位日的格式显示明天的日期。
Now	放回当前时间，例如：<Now>
Year	返回日期的年份，例如: <Year([出生日期])>
Month	返回日期的月份，例如: <Month([出生日期])>
Day	返回日期的日部分，例如: <Day([出生日期])>
Weekday	用数字形式返回星期几，例如: <WeekDay([出生日期])>
WeekDayName	用名称形式返回星期几，例如: <WeekDayName([出生日期])>
CUDate	将日期 Date 转换为大写中文格式。

	例如： <CUDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：贰零壹叁年拾贰月玖日
CLDate	将日期 Date 转换为小写中文格式。 例如： <CLDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：二〇一三年十二月九日
CUMoney	将数值转换为中文货币格式。 例如： <CUMoney([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍圆整
CUNumber	将数值转换为中文大写格式。 例如： <CUNumber([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍
EUNumber	将整数转换为英文大写。 例如： <EUNumber([数量])> 假定数量的值是 12345，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five
EUMoney	将数值转换为英文货币格式。 例如： <EUMoney([金额])> 假定金额的值是 12345.45，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five Dollars And Forty Five Cents

4.6.2.15 设置单元格公式

Excel 本身有非常丰富的函数，我们在在设计报表模版的时候，使用 Excel 的函数和语法来设置单元格公式，基于模版生成的报表，会保留之前的单元格公式。
例如在设计出库单模版的时候，打印日期需要调用当天的日期，可以直接将这个单元格的公式设置为： =Today()

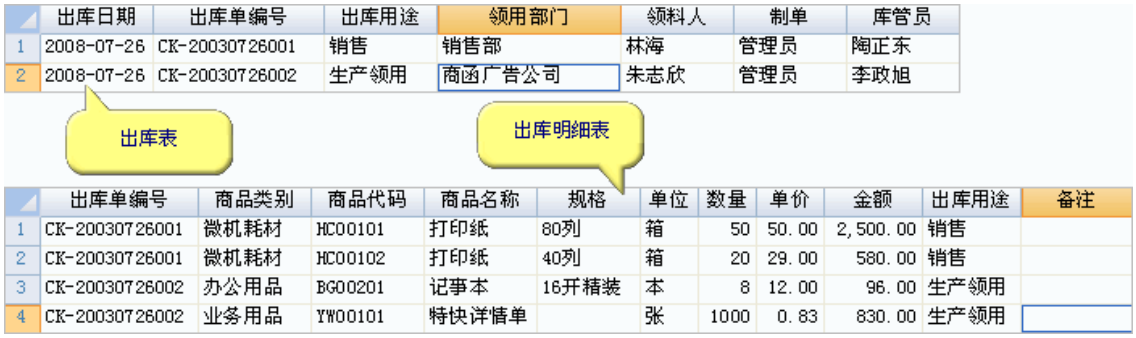


Excel 报表功能是非常灵活的，你既可以使用表达式，也可以使用 Excel 原有的单元格公式，充分利用各自的优势。

4.6.2.16 打印关联表

Excel 报表的细节区可以嵌套，这样可以非常方便地打印关联表数据。

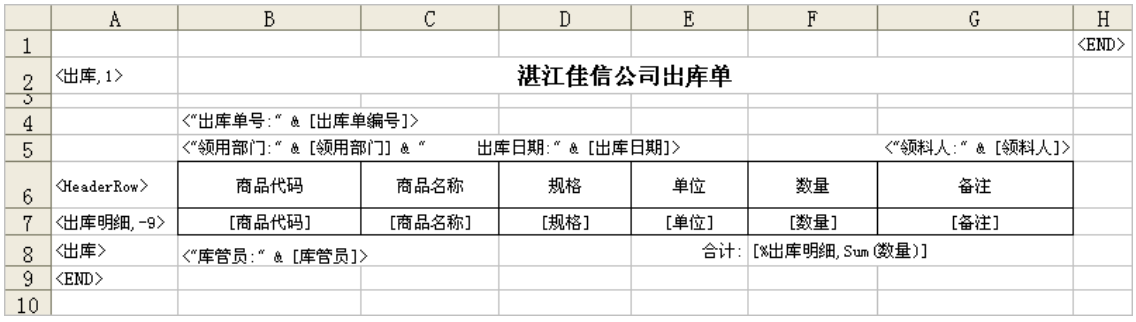
打开示例文件“Excel 报表.Table”，可以看到两个表，分别是出库和出库明细。



出库表和出库明细表通过出库单号建立了关联，其中出库表为父表，出库明细表为子表。

现在我们要针对出库表设计一个 Excel 报表，用于打印出库单。

启动 Excel，如下图所示设计好出库单模板，请留意是如何引用关联表数据的：



在项目文件目录的“CaseStudy\Attachments”子目录下，有一个已经按上图所示设计好的 Excel 模板文件，名为“出库单.XLS”，可以直接 参考使用。

需要注意：

1、通常不会将两个出库单打印在同一张纸上，所以出库单细节区的首行定义为：<出库,1>

2、出库单的细节区有多行，所以不能省略第 8 行(结束行)的定义：<出库>

3、第 7 行的数据来自于出库明细表，所以在第 7 行加入一个嵌套的细节区定义：<出库明细>

根据上面设计好的模板，生成的出库单：

湛江佳信公司出库单

出库单号:CK-20030726001

领用部门:销售部　　出库日期:2008/7/26　　领料人:林海

商品代码	商品名称	规格	单位	数量	备注
HC00101	打印纸	80列	箱	50	
HC00102	打印纸	40列	箱	20	

库管员:陶正东　　合计：70

4.6.2.17 统计关联表

上一节以出库单为例，我们学会了如何打印关联表。

这次我们还是以出库单为例，看看如何在报表中累计出库单数据。

统计某列数据的格式为：

[%表名,统计表达式,统计条件]

统计表达式使用聚合函数，例如：Sum(数量)

统计条件是可选的，语法采用我们之前介绍的表达式，请参考表达式的运算符和函数 和条件表达式。

为了给出库单底部加上数量合计，我们按下图修改出库单模板文件：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & "　　出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

在这里对数量进行合计

这是打印结果：

	B	C	D	E	F	G
		湛江佳信公司出库单				
	出库单号:CK-20030726001					
	领用部门:销售部	出库日期:2008-7-26				领料人:林海
	商品代码	商品名称	规格	单位	数量	备注
	HC00101	打印纸	80列	箱	50	
	HC00102	打印纸	40列	箱	20	
	库管员:陶正东			合计:	70	

4.6.2.18 打印多个关联表

Excel 报表中可以同时嵌入多个关联表细节区。
假定表 A 同时和表 B 和表 C 建立了关联，在设计报表的时候，可以在表 A 的细节区中，同时嵌入表 B 和表 C 的细节区。

	A
1	<表A>
2	
3	<表B>
4	
5	<表C>
6	
7	<表A>

关联表的细节区，也可以包括多行，例如下图中，作为关联表的表 C，其细节区就有四行:

	A
1	<表A>
2	
3	<表B>
4	
5	<表C>
6	
7	
8	<表C>
9	<表A>

4.6.2.19 自动添加空行

有的时候，我们不仅希望设置每页最多可打印的细节区数，而是希望在细节区数达不到设定值的时候，能够自动补充空行。

例如下面的出库单，希望每张出库单至少 9 行出库明细，在不够的情况下，自动补充空行：

湛江佳信公司出库单

出库单号:CK-20030726001

领用部门:销售部　　出库日期:2008-7-26　　领料人:林海

商品代码	商品名称	规格	单位	数量	备注
HC00101	打印纸	80列	箱	50	
HC00102	打印纸	40列	箱	20	

库管员:陶正东　　合计: 70

实现的方法很简单，只需在细节区后面增加一个参数，我们只需用负数来设置最多可打印的细节区数即可：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & "　　出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>				合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

必须用负数来设置每页最多可打印的细节区数，才能自动补空行。

4.6.2.20 设置打印顺序

在定义细节区时，可以加入排序参数，格式为：

<表名,排序列>

默认排序方式是升序，如果希望降序排序，格式为：

<表名,排序列 Desc>

对于主表，必须指定打印范围，排序参数才会发生作用，例如下图按日期顺序打印订购产品 PD01 的行：

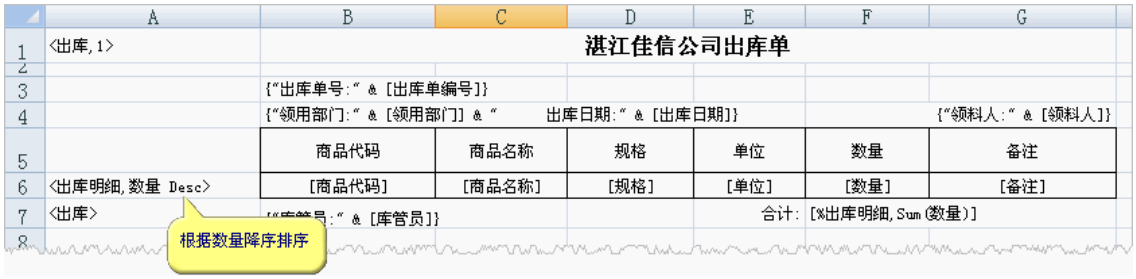
A	B	C	D	E	F	G	H	I	J
				订单					<End>
	产品	客户	雇员	单价	折扣	数量	金额	日期	
<订单,日期>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<产品 = 'PD01'>

按日期顺序打印

必须设置打印范围，排序参数才会发生作用

如果要打印所有行，将条件单元格的内容设为"<ALL>"即可。

对于关联表，不需要设置打印范围，排序参数也能发生作用：



还可以同时加入排序和细节区数量设置，例如：

<出库明细,数量,-9>

4.6.2.21 根据多列排序

打印顺序可以指定多个排序列，不同的排序列之间用符号"|"隔开。

例如：

产品|客户

表示根据产品和客户列排序

产品|日期 Desc

表示根据产品和日期排序，其中日期排序方式为降序。



4.6.2.22 指定页首行

如果指定了每页最大可打印的细节区数，我们就可以将某一行或某几行设为页首行，这样每次换页的时候，将自动插入这几行作为页首使用。

指定页首行的格式为：

<HeaderRow>

需要注意的是，必须明确指定每页可打印的细节区数量，页首行的设置才会生效。

例如：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>				合计:	[%出库明细.Sum(数量)]	
9	<END>							
10								

必须明确指定每页最多可打印的细节区数，页首行的设置才有效。

如果页首行有多行，请分别在首行和尾行加上标记：

	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4	<HeaderRow>	<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>				合计:	[%出库明细.Sum(数量)]	
9	<END>							
10								

4.6.2.23 页尾行与分页汇总

如果指定了每页最大可打印的细节区数，我们就可以将某一行或某几行设为页尾行，这样每次换页的时候，将自动插入这几行作为页尾使用。

指定页尾行的格式为:

<FooterRow>

如果页尾行有多行，请分别在首行和尾行加上上述标记。

可以在页尾行对本页数据进行统计，格式为：

[%统计表达式]

统计表达式使用聚合函数，例如：Sum(数量)。

必须明确指定每页可打印的细节区数量，页尾行的设置才会生效。

示例

下面的模板用于打印订单表，在每页的结尾加上金额小计，最后加上合计:

同样必须明确指定每页打印的细节区数，页尾行的设置才有效。

	D	E	F	G	H	I	J
1	雇员	单价	折扣	数量	金额	日期	
2	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	
3	小计				[%Sum(金额)]		
4	总计				[%Sum(金额)]		
5							

打印结果：

产品	客户	雇员	单价	折扣	数量	金额	日期
PD05	CS03	EP04	17	0	300	5100	1999-6-27
PD05	CS01	EP02	17	0	20	340	1999-6-28
PD02	CS02	EP02	15.2	0	400	6080	1999-6-29
PD02	CS03	EP02	19	0	30	570	1999-6-29
PD01	CS02	EP01	18	0	300	5400	1999-6-30
PD01	CS05	EP03	18	0	60	1080	1999-6-30
小计						18570	
总计						825651.57	

4.6.2.24 汇总已经打印的行

如果需要统计已经打印行，可以采用以下格式：
[#表名,统计表达式]
统计表达式使用聚合函数，例如：Sum(数量)
假定我们不仅要汇总出每一页的小计，而且需要得截止到当前页为止的累计，可以按下图所示设计报表模板：

	A	B	C	D	E	F	G	H	I
1	<HeaderRow>	产品	客户	雇员	单价	折扣	数量	金额	日期
2	<订单,30>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]
3									
4	<FooterRow>	小计:	[%Sum(金额)]		当前累计:	[#订单,Sum(金额)]			
5									

打印效果：

PD03	CS04	EP05	19	0	100	1900	1999-2-21
PD02	CS05	EP03	19	0	100	1900	1999-2-21
PD03	CS05	EP02	10	0	300	3000	1999-2-21
PD05	CS02	EP02	17	0.05	160	2584	1999-2-21
PD01	CS02	EP05	14.4	0.15	150	1836	1999-2-22
PD01	CS04	EP02	14.4	0	400	5760	1999-2-23
PD01	CS05	EP03	14.4	0	180	2592	1999-2-23
PD02	CS03	EP02	19	0.2	150	2280	1999-2-23
PD05	CS02	EP03	21.35	0.05	240	4867.8	1999-2-24
PD05	CS02	EP02	17	0.2	200	2720	1999-2-25
小计:		107775.9	当前累计:	233794.4			

4.6.2.25 分组统计报表

Excel 报表也提供了分组统计功能。
定义分组头的格式为：
<GroupHeader,表名,列名>

或

<GroupHeader,表名,列名,1>

不同的是，后者在换页之后，会重复打印一次分组头。

定义分组尾的格式为：

<GroupFooter,表名,列名>

或者

<GroupFooter,表名,列名,1>

不同的是，后者打印完分组尾之后，会自动换页。

在分组头和分组尾可以引用数据，引用数据的格式为：

[列名]

在分组尾可以统计数据，统计数据格式为：

[%统计表达式]

统计表达式使用聚合函数，例如：Sum(数量)。

需要特别提示的是： 分组报表的细节区必须指定排序参数，且排序列必须为分组列。

示例

希望根据产品分组打印订单，并统计每个产品的销售数量和金额，并自动换页，模板可按下图所示设计：



这是打印效果：

PD03	CS01	EP01	10	0	240	2400	1999-02-[illegible]
PD03	CS05	EP03	10	0	200	2000	1999-02-10
PD03	CS05	EP02	10	0	300	3000	1999-02-21
PD03	CS03	EP02	10	0	300	3000	1999-02-26
小计PD03					1290	12900	
产品	客户	雇员	单价	折扣	数量	金额	日期
PD04	CS04	EP02	22	0.1	500	9900	1999-02-14
PD04	CS03	EP02	22	0	10	220	1999-02-26
小计PD04					510	10120	
产品	客户	雇员	单价	折扣	数量	金额	日期
PD05	CS02	EP03	17	0	150	2550	1999-02-01
PD05	CS04	EP04	21.35	0	700	14945	1999-02-01
PD05	CS04	EP04	21.35	0	10	213.5	1999-02-04
PD05	CS04	EP03	21.35	0	700	14945	1999-02-07
PD05	CS03	EP03	17	0	100	1700	1999-02-20
PD05	CS02	EP02	17	0.05	160	2584	1999-02-21
PD05	CS02	EP03	21.35	0.05	240	4867.8	1999-02-24
PD05	CS02	EP02	17	0.2	200	2720	1999-02-25
PD05	CS04	EP02	21.35	0	160	3416	1999-02-28
小计PD05					2420	47941.3	

可以同时使用分组头和分组尾，而且不管是分组头，还是分组尾，都可以是多行的，例如下图的分组头包括两行：

A	B	C	D	E	F	G	H	I
	订单							<END>
<GroupHeader,订单,产品>						<"产品："&[产品]>		
<GroupHeader,订单,产品>	客户	雇员	单价	折扣	数量	金额	日期	
<订单,产品>	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
<GroupFooter,订单,产品,1>	<"小计"&[产品]>				[%Sum(数量)]	[%Sum(金额)]		
<END>								

这是打印结果：



订单

产品：PD01

客户	雇员	单价	折扣	数量	金额	日期
CS03	EP04	18	0.15	80	1224	1999-1-4
CS04	EP05	14.4	0	200	2880	1999-1-8
CS02	EP01	18	0.2	800	11520	1999-1-10
CS04	EP02	14.4	0.05	500	6840	1999-1-10
CS03	EP04	14.4	0.25	200	2160	1999-1-14
CS02	EP05	14.4	0	100	1440	1999-1-20
CS03	EP01	14.4	0	240	3456	1999-1-21
CS05	EP05	18	0	200	3600	1999-2-1
CS01	EP04	18	0	20	360	1999-2-7
CS01	EP02	18	0	80	1440	1999-2-7

很多时候，我们希望换页之后，能够重复打印一次分组头，此时可以将 GroupHeader 的第四个参数设为 1，并指定每页的细节区数。

例如：

A	B	C	D	E	F	G	H	I
	订单							<END>
<GroupHeader,订单,产品,1>						<"产品："&[产品]>		
<GroupHeader,订单,产品>	客户	雇员	单价	折扣	数量	金额	日期	
<订单,产品,30>	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
<GroupFooter,订单,产品,1>	<"小计"&[产品]>				[%Sum(数量)]	[%Sum(金额)]		
<END>								

不管是分组头还是分组尾，都可以多行，分别在第一行和最后一行加上对应的格式定义即可，例如下图中，分组头有三行，分组尾有四行：

	A
1	
2	
3	<GroupHeader,订单,产品>
4	
5	<GroupHeader,订单,产品>
6	<订单,产品>
7	<GroupFooter,订单,产品>
8	
9	
10	<GroupFooter,订单,产品,1>
11	<END>

4.6.2.26 多个分组列

上一节介绍的分组统计报表，都只有一个分组列。

其实 Excel 报表一样可以有多个分组列，例如下面的报表模板，有两个分组尾，分别是客户和产品，也就是根据产品和客户进行分组统计，并生成报表：

A	B	C	D	E	F	G	H	I	J
				订单					<END>
<HeaderRow>	产品	客户	雇员	单价	折扣	数量	金额	日期	
<订单,产品|客户>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
<GroupFooter,订单,客户>		<"小计" & [客户]>				[%Sum(数量)]	[%Sum(金额)]		
<GroupFooter,订单,产品,1>	<"小计" & [产品]>					[%Sum(数量)]	[%Sum(金额)]		
<END>									

需要特别注意的是，细节区的排序参数，必须加上所有的分组列，而且大分组列应该位于小分组列之前，例如上面的报表，产品列是一级分组，在产品相同的情况下，再根据客户分组，所以排序参数必须是“产品|客户”，而不能是“客户|产品”。

4.6.2.27 引用父表数据

引用父表数据的格式为：

[父表路径,列名]

例如有一个订单管理系统，其产品表通过产品编号和订单明细表建立关联，客户表通过客户编号和订单表建立关联，而订单表通过订单编号和订单明细表建立关联。

其中订单明细表的结构如下：

	订单编号	产品编号	单价	数量	折扣
1	10248	17	14	12	0
2	10248	42	9.8	10	0
3	10248	72	34.8	5	0
4	10249	14	18.6	9	0
5	10249	51	42.4	40	0
6	10250	41	7.7	10	0
7	10250	51	42.4	35	0.15
8	10250	65	16.8	15	0.15
9	10251	22	16.8	6	0.05
10	10251	57	15.6	15	0.05

现在希望通过 Excel 报表来打印订单明细表，为了便于阅读，希望能够包括产品名称和公司名称两列，为此我们可如下设置 Excel 报表模板：

	A	B	C	D	E	F	G
1		订单ID	产品	客户	单价	数量	折扣
2	<订单明细>	[订单ID]	[产品.产品名称]	[订单.客户.客户名称]	[单价]	[数量]	[折扣]
3							
4							

因为订单明细表和产品表之间有直接的关联，所以从产品表引用产品名称的方法很简单：

[产品,产品名称]

订单明细表没有直接和客户表建立关联，而是通订单表间接地关联到客户表，所以从客户表引用客户名称，就稍微复杂一点，需要给出完整的路径：

[订单.客户,客户名称]

这是打印结果：

	A	B	C	D	E	F
1	订单ID	产品	客户	单价	数量	折扣
2	10248	猪肉	山泰企业	14	12	0
3	10248	糙米	山泰企业	9.8	10	0
4	10248	酸奶酪	山泰企业	34.8	5	0
5	10249	沙茶	东帝望	18.6	9	0
6	10249	猪肉干	东帝望	42.4	40	0
7	10250	虾子	实翼	7.7	10	0
8	10250	猪肉干	实翼	42.4	35	0.15
9	10250	海苔酱	实翼	16.8	15	0.15
10	10251	糯米	千固	16.8	6	0.05
11	10251	小米	千固	15.6	15	0.05
12	10251	海苔酱	千固	16.8	20	0
13	[illegible]	桂花糕	[illegible]	[illegible]	10	[illegible]

如果被引用的是一个图片列，引用格式为：

[父表路径,&列名]

[父表路径,&列名,x,y]

[父表路径,&列名,x,y,width,height]
x:　左边距
y：　上边距
width：　图片宽度
height：图片高度

4.6.2.28 直接引用数据

前面我们已经提供了引用父表和子表数据的方法，但是有的时候，我们希望从一个毫无关系的表中引用数据。
这样的引用同样简单，格式为:
[@表名,列名]
或
[@表名,列名,行位置]
[@表名,列名,+行位置]
利用直接引用，我们可以在细节区之外引用表中数据。
行位置是可选的，如果省略，则引用的是表中选定行的数据。
如果在行位置之前加上符号“+”，则表示相对位置，也就是距离选定行的位置。
例如：
[@产品,名称]
[@产品,名称,+1]
[@产品,名称,+2]
[@产品,名称,9]

假定产品表选定行的位置为 3，那么上述标示分别表示：
第三行的名称列内容
第四行的名称列内容
第五行的名称列内容
第十行的名称列内容(行号从 0 开始编号，9 表示第十行)

直接引用对于关联表也有效，例如引用关联表“出库.出库明细”中第二行商品代码列的内容：
[@出库.出库明细,商品代码,+1]
如果直接引用的是图片列，可以按照以下格式：
[@表名,&列名,行位置]
[@表名,&列名,行位置,x,y]
[@表名,&列名,行位置,x,y,width,height]
或者
[@表名,&列名,+行位置]
[@表名,&列名,+行位置,x,y]
[@表名,&列名,+行位置,x,y,width,height]
x:　左边距
y:　上边距
width:　图片宽度
height:图片高度
需要注意的是，对于图片列，即使是当前行，也必须明确指定行位置，例如：
[@员工,&照片,+0]

4.6.2.29 直接统计

如果要直接统计某个表的数据，格式为：
[$表名,统计表达式]
或
[$表名,统计表达式,统计条件]
统计表达式使用聚合函数，例如：Sum(数量)
请参考表达式的 运算符和函数 以及 条件表达式
示例
例如要用 Excel 报表统计不同产品的销售数量和金额，可以按照下图所示设计模板：

A	B	C	D
	产品	数量	金额
	PD01	[$订单,Sum(数量),产品 = 'PD01']	[$订单,Sum(金额),产品 = 'PD01']
	PD02	[$订单,Sum(数量),产品 = 'PD02']	[$订单,Sum(金额),产品 = 'PD02']
	PD03	[$订单,Sum(数量),产品 = 'PD03']	[$订单,Sum(金额),产品 = 'PD03']
	PD04	[$订单,Sum(数量),产品 = 'PD04']	[$订单,Sum(金额),产品 = 'PD04']
	PD05	[$订单,Sum(数量),产品 = 'PD05']	[$订单,Sum(金额),产品 = 'PD05']

这是生成的报表：

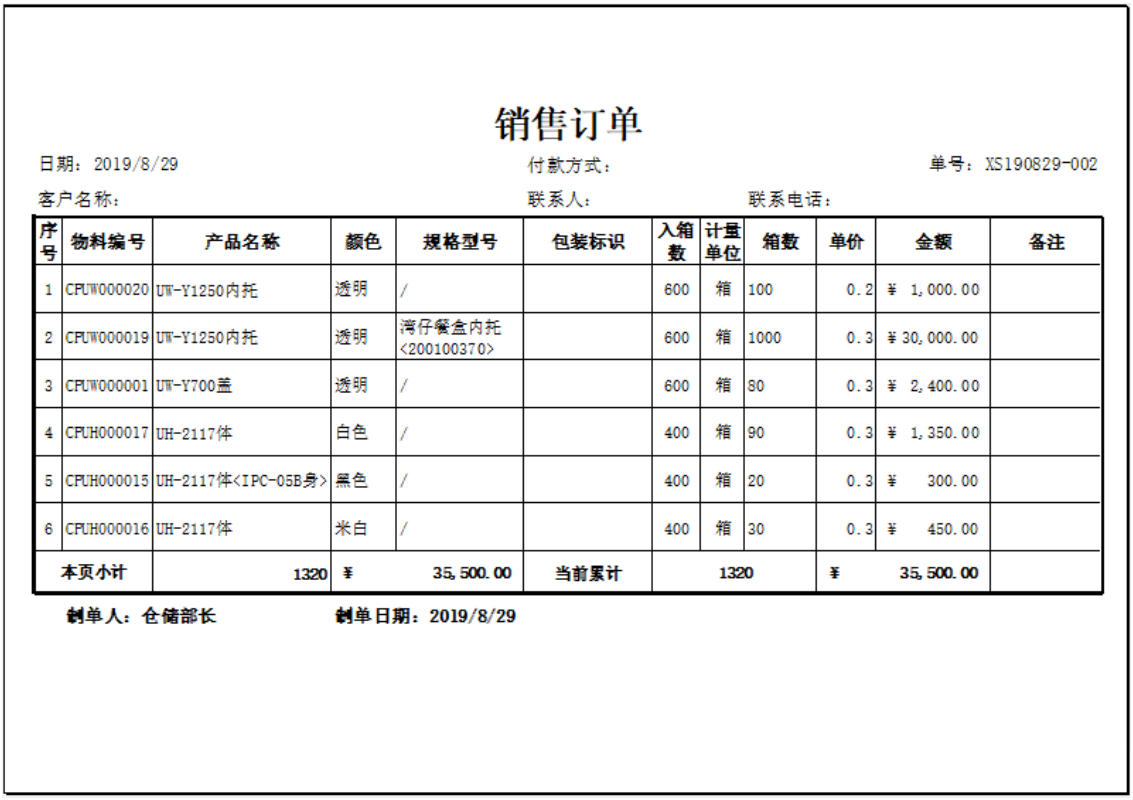
A	B	C
产品	数量	金额
PD01	11290	171754.92
PD02	18200	296522.5
PD03	7000	61490
PD04	5480	102861
PD05	10400	193023.15

4.6.2.30 引用顺序号

顺序号的初始值为 1，每打印一个细节区，顺序号就会增加 1。
可以在 Excel 报表中引用顺序号，引用格式为：
<Index>

4.6.2.31 又一个关联表打印示例

假定有两个表，分别是订单表和订单明细表，二者之间已经建立关联，希望通过 Excel 报表实现下图所示的订单打印功能：



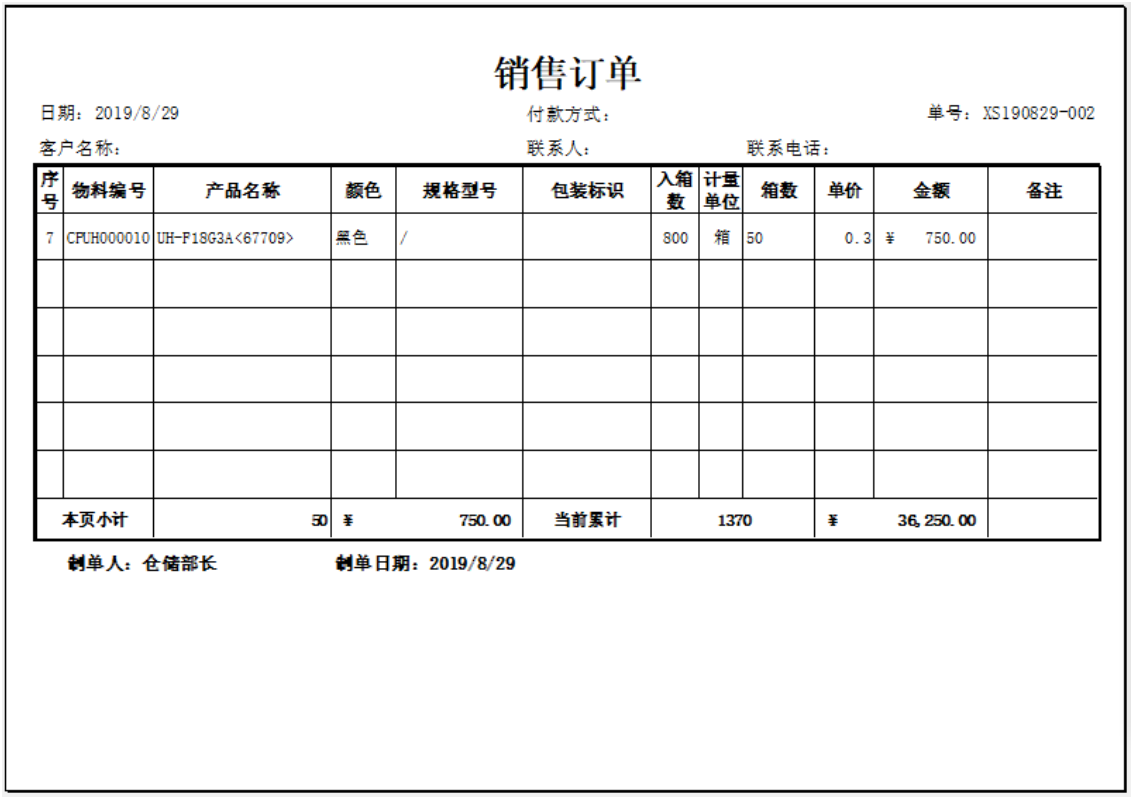
销售订单

日期：2019/8/29　　付款方式：　　单号：XS190829-002

客户名称：　　联系人：　　联系电话：

序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单位	箱数	单价	金额	备注
1	CPUW000020	UW-Y1250内托	透明	/		600	箱	100	0.2	¥ 1,000.00	
2	CPUW000019	UW-Y1250内托	透明	湾仔餐盒内托<200100370>		600	箱	1000	0.3	¥ 30,000.00	
3	CPUW000001	UW-Y700盖	透明	/		600	箱	80	0.3	¥ 2,400.00	
4	CPUH000017	UH-2117体	白色	/		400	箱	90	0.3	¥ 1,350.00	
5	CPUH000015	UH-2117体<IPC-05B身>	黑色	/		400	箱	20	0.3	¥ 300.00	
6	CPUH000016	UH-2117体	米白	/		400	箱	30	0.3	¥ 450.00	
本页小计		1320	¥	35,500.00	当前累计	1320			¥	35,500.00	

制单人：仓储部长　　制单日期：2019/8/29



销售订单

日期：2019/8/29　　付款方式：　　单号：XS190829-002

客户名称：　　联系人：　　联系电话：

序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单位	箱数	单价	金额	备注
7	CPUH000010	UH-F18G3A<67709>	黑色	/		800	箱	50	0.3	¥ 750.00	
本页小计		50	¥	750.00	当前累计	1370			¥	36,250.00	

制单人：仓储部长　　制单日期：2019/8/29

这个报表比之前的关联表打印稍微复杂一点，页头和也尾都有多行，有分页汇总和总计，但设计的原则是一样的：

首尾是父表细节区定义，中间是子表细节区定义，HeaderRow 和 FooterRow 加在需要的地方。

把握了这个原则，报表模板设计起来一样的轻松：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P
1															<End>
2	<订单,1>														
3	<HeaderRow>			销售订单											
4				<"日期："&[日期]>				<"付款方式："&[付款方式]>					<"单号："&[单号]>		
5				<"客户名称："&[客户名称]>				<"联系人："&[联系人]>			<"联系电话："&[联系电话]>				
6	<HeaderRow>		序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单	箱数	单价	金额	备注	
7	<订单明细,-6>	Inde	[物料编号]		[产品名称]	[颜色]	[规格型号]	[包装标识]	入箱数	量单	[箱数]	[单价]	[金额]	[备注]	
8	<FooterRow>		本页小计		[%Sum(箱数)]		[%Sum(金额)]	当前累计	订单明细,Sum(箱数		订单明细,Sum(金额)				
9	<FooterRow>			<"制单人："&[制单人]>		<"制单日期："&[制单日期]>									
10	<订单>														
11	<End>														

隐藏无需打印的标记行

实际预览或打印报表的时候，你会发现上述模板有问题：第一页的顶部页边距明显大过第二页。本节头两个图片也说明了这个问题。

导致这个问题的出现，是因为 Excel 报表需要一些没有实质内容的标记行，例如上述模板中的第 1、2、10、11 四行，而这些行也会参与打印。正式因为模板中第 1 行和第 2 行，导致了第一页的顶部页边距比其他页面大，所以解决这个问题很简单，模板设计完成之后，隐藏无需参与打印的标记行：



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	P
3	<HeaderRow>			销售订单											
4				<"日期："&[日期]>				<"付款方式："&[付款方式]>					<"单号："&[单号]>		
5				<"客户名称："&[客户名称]>				<"联系人："&[联系人]>			<"联系电话："&[联系电话]>				
6	<HeaderRow>		序号	物料编号	产品名称	颜色	规格型号	包装标识	入箱数	计量单	箱数	单价	金额	备注	
7	<订单明细, -6>	Inde	[物料编号]	[产品名称]		[颜色]	[规格型号]	[包装标识]	入箱数	量单	[箱数]	[单价]	[金额]	[备注]	
8	<FooterRow>		本页小计		[%Sum(箱数)]		[%Sum(金额)]	当前累计	订单明细.Sum(箱数		订单明细.Sum(金额)				
9	<FooterRow>			<"制单人："&[制单人]>		<"制单日期："&[制单日期]>									
12															

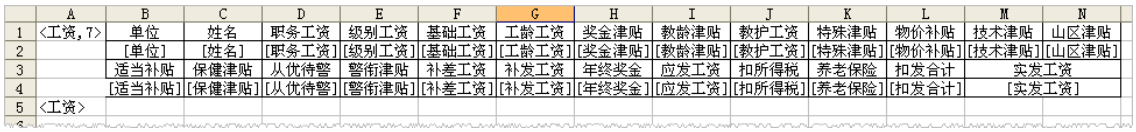
重要提示：

不要在设计过程隐藏标记行，应该等报表设计完全达到要求之后再隐藏。

4.6.2.32 打印工资条

有了 Excel 报表，打印工资条是非常轻松的了。

我们只需设计一个这样简单的模板：



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	<工资, 7>	单位	姓名	职务工资	级别工资	基础工资	工龄工资	奖金津贴	教龄津贴	教护工资	特殊津贴	物价补贴	技术津贴	山区津贴
2		[单位]	[姓名]	[职务工资]	[级别工资]	[基础工资]	[工龄工资]	[奖金津贴]	[教龄津贴]	[教护工资]	[特殊津贴]	[物价补贴]	[技术津贴]	[山区津贴]
3		适当补贴	保健津贴	从优待警	警衔津贴	补差工资	补发工资	年终奖金	应发工资	扣所得税	养老保险	扣发合计	实发工资	
4		[适当补贴]	[保健津贴]	[从优待警]	[警衔津贴]	[补差工资]	[补发工资]	[年终奖金]	[应发工资]	[扣所得税]	[养老保险]	[扣发合计]	[实发工资]	
5	<工资>													

然后将这个模板加入到工资表的 Excel 报表管理器中即可，就是这么简单，谁都可以完成。

上图设置每页打印 7 个工资条，具体请根据需要设定。

4.6.2.33 标签打印

如果同一细节区包括多个标签，而且这些标签基于同一个数据行，那么模板设计很简单。

例如每一个订单打印四个标签，可以按下图所示设计模板：



	A	B	C	D	E	F	G	H	I	J	K	L
1												
2	<订单, 11>	客户:	[客户]		客户:	[客户]		客户:	[客户]		客户:	[客户]
3		产品:	[产品]		产品:	[产品]		产品:	[产品]		产品:	[产品]
4		数量:	[数量]		数量:	[数量]		数量:	[数量]		数量:	[数量]
5		日期:	[日期]		日期:	[日期]		日期:	[日期]		日期:	[日期]
6	<订单>											
7												

注意为了避免换页导致标签被分割，应该设置每页可打印的细节区数，上面设置的是 11。

标签打印结果：

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS03
产品:	PD05
数量:	650
日期:	1999-1-2

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS01
产品:	PD02
数量:	400
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS04
产品:	PD05
数量:	320
日期:	1999-1-3

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

客户:	CS03
产品:	PD01
数量:	80
日期:	1999-1-4

可是很多时候，同一个细节区的多个标签，可能来自不同的数据行；例如一个细节区包括四个标签，每个标签来自于不同的数据行。

我们知道，每一个细节区针对一个数据行，之前的引用方式只能引用该数据行的数据，如果要引用其它行的数据，必须使用相对定位的方式：

[列名+X]

X 是一个整数，表示细节区数据行之后的第几行，例如：

[客户+2]

由于同一个细节区不再是对应一个数据行，而是对应多个数据行，所以在定义细节区的时候，要增加一个参数，用于指定每个细节区对应的数据行数，这个参数以符号@开头：

<订单,@4>

综上所述，为了一个细节区打印四个标签，每个标签来自不同数据行，可以按下图所示设计模板：

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													<END>
2	<订单,@4,11>	客户:	[客户]		客户:	[客户+1]		客户:	[客户+2]		客户:	[客户+3]	<ALL>
3		产品:	[产品]		产品:	[产品+1]		产品:	[产品+2]		产品:	[产品+3]	
4		数量:	[数量]		数量:	[数量+1]		数量:	[数量+2]		数量:	[数量+3]	
5		日期:	[日期]		日期:	[日期+1]		日期:	[日期+2]		日期:	[日期+3]	
6	<订单>												
7	<END>												
8													

打印结果：



客户:	CS03	客户:	CS01	客户:	CS04	客户:	CS03
产品:	PD05	产品:	PD02	产品:	PD05	产品:	PD01
数量:	650	数量:	400	数量:	320	数量:	80
日期:	1999-1-2	日期:	1999-1-3	日期:	1999-1-3	日期:	1999-1-4
客户:	CS03	客户:	CS04	客户:	CS03	客户:	CS04
产品:	PD03	产品:	PD04	产品:	PD05	产品:	PD01
数量:	490	数量:	240	数量:	160	数量:	200
日期:	1999-1-4	日期:	1999-1-7	日期:	1999-1-7	日期:	1999-1-8
客户:	CS02	客户:	CS02	客户:	CS04	客户:	CS01
产品:	PD03	产品:	PD01	产品:	PD01	产品:	PD03
数量:	200	数量:	800	数量:	500	数量:	300
日期:	1999-1-8	日期:	1999-1-10	日期:	1999-1-10	日期:	1999-1-11
客户:	CS05	客户:	CS03	客户:	CS05	客户:	CS03
产品:	PD04	产品:	PD04	产品:	PD02	产品:	PD01
数量:	500	数量:	30	数量:	200	数量:	200
日期:	1999-1-11	日期:	1999-1-12	日期:	1999-1-13	日期:	1999-1-14

同样，如果希望一个细节区打印四个标签，前两个标签来自一个数据行，后两个标签来自一个数据行，可以这样设计模板：

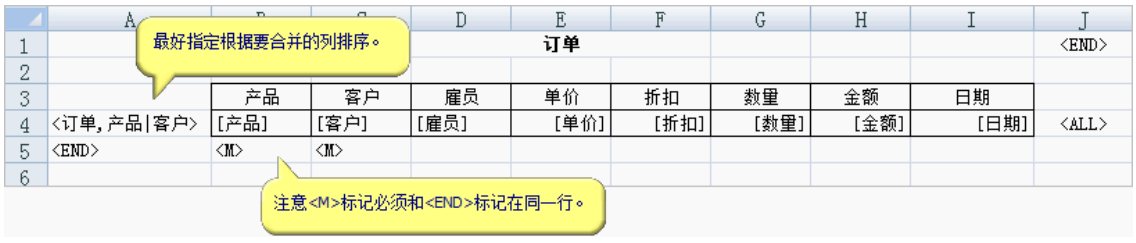
	A	B	C	D	E	F	G	H	I	J	K	L	M
1													<END>
2	<订单,@2,11>	客户:	[客户]		客户:	[客户]		客户:	[客户+1]		客户:	[客户+1]	<ALL>
3		产品:	[产品]		产品:	[产品]		产品:	[产品+1]		产品:	[产品+1]	
4		数量:	[数量]		数量:	[数量]		数量:	[数量+1]		数量:	[数量+1]	
5		日期:	[日期]		日期:	[日期]		日期:	[日期+1]		日期:	[日期+1]	
6	<订单>												
7	<END>												
8													

4.6.2.34 自动合并单元格

Excel 报表提供了自动合并单元格的功能，可以将同一列内容相同的单元格自动合并为一个。

上一节提到用<End>标记指定报表的有效区域，如果在有效区标记行的某一列加上标记<M>，会自动合并该列的相同内容。

例如：



	A	D	E	F	G	H	I	J
1			订单					<END>
2								
3		产品	客户	雇员	单价	折扣	数量	金额	日期	
4	<订单,产品|客户>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<ALL>
5	<END>	<M>	<M>							
6										

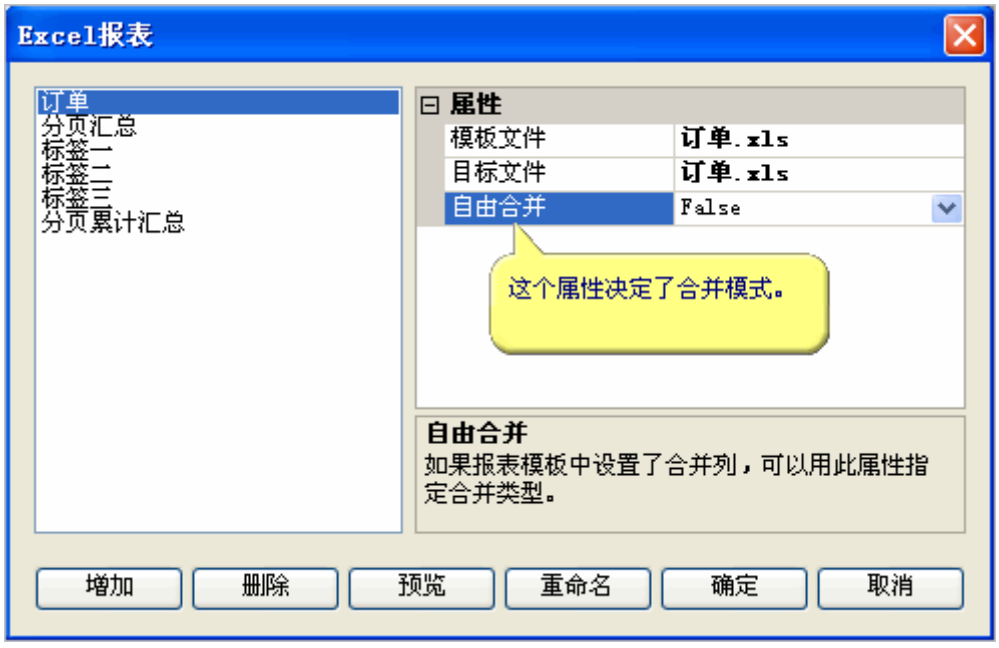
提示：因为只有内容相同且相邻的单元格才会合并，所以设置模板的时候，通常会将排序列设置为要合并的列。

上图的报表模板指定要合并产品列和客户列，这是输出结果：

A	B	C	D	E	F	G	H
			订单				
产品	客户	雇员	单价	折扣	数量	金额	日期
PD01	CS02	EP01	18	0.2	800	11520	1999-01-10
		EP05	14.4	0	100	1440	1999-01-20
	CS03	EP04	18	0.15	80	1224	1999-01-04
		EP04	14.4	0.25	200	2160	1999-01-14
		EP01	14.4	0	240	3456	1999-01-21
	CS04	EP05	14.4	0	200	2880	1999-01-08
		EP02	14.4	0.05	500	6840	1999-01-10
PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
		EP04	15.2	0	100	1520	1999-01-14
	CS03	EP01	15.2	0.1	200	2736	1999-01-17
	CS05	EP04	19	0.1	200	3420	1999-01-13
		EP04	15.2	0	400	6080	1999-01-14
PD03	CS01	EP04	8	0	300	2400	1999-01-11
	CS02	EP02	8	0.1	200	1440	1999-01-08
		EP05	8	0.25	700	4200	1999-01-26
		EP03	10	0.1	200	1800	1999-01-28
	CS03	EP04	10	0	490	4900	1999-01-04
	CS04	EP04	10	0	40	400	1999-01-17
		EP04		0		660	1999-01-12

合并类型有两种，分别是自由合并和标准合并，标准合并必须是前一列和本列内容都相同才合并，自由合并只需本列内容相同即可合并。

Excel 报表默认是标准合并，你可以通过下面的属性来选择合并类型：



4.6.2.35 引用图片文件

引用图片文件的完整格式为：

[&&文件名, X, Y, Width, Height]

X:　　左边距

Y： 上边距
Width： 图片宽度
Height：图片高度
还有另外两种简化引用格式：
[&&文件名]
或
[&&文件名, X, Y]
示例
如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可：
[&&myfox.gif, 2, 2]
否则需要包括路径，例如：
[&&c:\Images\myfox.gif, 2, 2]
对于固定的图片文件，不如在设计模板的时候直接插入，无需采用引用的方式。
如果图片是动态生成的，则只能采用引用的方式，通常是事先用代码生成图片文件，并保存在项目的 Images 子目录下，然后再生成 Excel 报表。

4.6.2.36 引用 Var 变量

可以在 Excel 报表中引用 Var 变量，格式为：
[!Var 变量名称]
Var 变量可以参与表达式计算，例如：
<"起始日期:" & [!起始日期]>

4.6.2.37 嵌入表达式

我们已经知道，Excel 报表模板采用"[内容]"格式来引用数据，采用"<表达式>"格式来计算表达式。
通常 表达式自身也会引用数据，例如以下格式中的"[编号]"表示引用编号列的数据：
<"编号：" & [编号]>
反过来，我们也可以在"[内容]"这种格式中嵌入表达式 ，需要注意的是，嵌入表达式的格式为"<*表达式*>"，而不是"<表达式>"。
示例一
下面的格式可以直接统计订单表中产品为 PD01 的销售数量：
[$订单,Sum(数量),产品 = 'PD01']
如果要统计的产品不是固定的，而是存储在 Var 变量 tp 中，一些用户可能会如下设置格式：
[$订单,Sum(数量),"产品 = '" & [!tp] & "'"]
这样的格式是不对的，常规格式中嵌入的表达式，要用符号"<*"和"*>"括起来，所以正确的格式为：
[$订单,Sum(数量),<*"产品 = '" & [!tp] & "'"*>]
实际上，常规格式的任何一部分，都可以用嵌入表达式来动态合成。
示例二
假定表中有一名为“姓名”的列，用于输入员工姓名，如张三、李四、王五等等。
在项目文件夹的 Images 目录下，有员工的照片文件，如: 张三.bmp、李四.bmp、王五.bmp
如果要在 Excel 报表中打印员工照片，由于员工表没有照片列，无法以引用图片列的方法实现。

所以只能采用引用图片文件的方式来打印员工照片。
我们知道，引用图片文件的格式为：[&&文件名]。
但是这里的文件名不是固定的，需要根据员工姓名来合成。
我们可以用下面的格式来打印员工照片：
[<*"&&" & [姓名] & ".bmp"*>]
上面的格式中，嵌入了一个表达式："&&" & [姓名] & ".bmp"
假定姓名是“张三”，在求出表达式的值后，整个格式变为：
[&&张三.bmp]
也就是直接引用图片文件“张三.bmp”。

4.6.2.38 再谈表达式计算

前面的章节已经提到，可以在 Excel 报表中直接进行表达式计算，表达式用符号"<"和">"括起来。
例如：
<"编号：" & [编号]>

<CUMoney([金额])>
不过下面两个表达式却不能正常计算：
<[%Sum(金额)] & "元">
<CUMoney([%Sum(金额)])>
[%Sum(金额)]这种用于数据统计的格式，在表达式中出现的时候，必须用符号*括起来，所以上面两个表达式应该修改为：
<*[%Sum(金额)]* & "元">
<CUMoney(*[%Sum(金额)]*)>
实际上除了**[列名]**这种标准数据引用格式外，所有其他格式，在表达式中出现的时候，都必须用符号*括起来。
下面我们来总结一下这些非标准的格式。
正如前面所言，数据统计格式在表达式中出现的时候，必须用符号*括起来，例如下面的表达式是错误的：
<Round([%Sum(金额)] /10000,2) & "万元">
正确的表达式为：
<Round(*[%Sum(金额)]* /10000,2) & "万元">
我们在学习标签打印的时候，已经知道下面的格式：
[列名+X]
表示细节区数据行之后的第 X 行，X 是个整数，例如：
[数量+2]
表示细节区数据行之后第 2 行的数量列。
这种跨行引用数据的格式参与表达式计算的时候，也必须用符号*括起来，例如：
<IIF(*[数量+1]* > 100, *[数量+1]*, 0)>
此外，直接引用格式在表达式中出现的时候，也必须用符号*括起来，例如：
<"单号:" & *[@任务,单号,+0]*>
同样，表达式中引用父表数据的格式，也必须用符号*括起来，例如要在订单表的细节区的表达式中，引用父表产品表中的产品名称：

<"品名" & *[产品,产品名称]*>

4.6.2.39 金额的按位打印

如果要用 Excel 打印或套打一些财务报表和单据，会涉及到金额的中文大写和分位打印：



金额的大小写比较简单，设计模板的时候，只需此单元格的内容设置为：<CUMoney([金额])>
金额分位打印稍微复杂的，需要用 GetDigit 函数逐位设置，下表列出了设计模板的时候，每位金额对应的公式：

位	公式
分	<GetDigit([金额],-2,"￥")>
角	<GetDigit([金额],-1,"￥")>
元	<GetDigit([金额],0,"￥")>
十	<GetDigit([金额],1,"￥")>
百	<GetDigit([金额],2,"￥")>
千	<GetDigit([金额],3,"￥")>
万	<GetDigit([金额],4,"￥")>
十万	<GetDigit([金额],5,"￥")>
百万	<GetDigit([金额],6,"￥")>
千万	<GetDigit([金额],7,"￥")>
亿	<GetDigit([金额],8,"￥")>

正如上一节所介绍，如果需要对合计的金额大写或分位打印，记得加上*，例如：
<CUMoney(*[%出库明细,Sum(数量)]*)>
<GetDigit(*[%出库明细,Sum(数量)]*,3,"￥")>

4.6.2.40 使用条形码字体

通过条形码字体，可以在 Excel 报表中批量生成条形码。
77 种条形码字体的下载地址：

http://www.foxtable.com/download/codebarfont.rar
解压后，双击某个字体文件，可以看到字体名称和对应的条形码。
使用方法：
如果是 Windows 7/vista，右击某条形码字体文件，从弹出的快捷菜单中执行“安装”命令。
如果是 Windows xp 系统，则进入控制面板，切换到经典视图，再进入字体文件夹，将需要使用的条形码字体文件复制粘贴到此文件夹中。
安装条形码字体后,在 Excel 报表批量生成条形码就非常简单了，可以按常规引用某字段，然后将其字体设置为条形码字体，并设置合适的字体大小即可。
提示：
为了提高条形码扫描器的识别率，有些条形码需要加上首尾识别符号，例如 Code39 条形码，就建议在首尾各附加一个*号，为此在 Excel 模板可以这样引用条码列：
<"*" & [条码列] & "*">

4.6.2.41 再谈有效区域的设置

在编写 Excel 报表示例文件的时候，我按照下图所示设置了列的有效区域：

A	B	C	D	E	F	G	H
<出库,1>	湛江佳信公司出库单						<END>
	<"出库单号:" & [出库单编号]>						
	<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
<END>							

表面上看没有任何问题，但是结果却让我大吃一惊，生成报表的时候提示错误“未找到列[END]”。
原来出库表的细节区是从第一行开始的，而细节区的此位置是用于定义打印条件的，而不是用于定义有效区域的，但是有效区域的定义必须出现在表格的第一行，如何来解决这种冲突呢？
我们可以另外插入一行，用于定义有效区域，如下图所示：

A	B	C	D	E	F	G	H
							<END>
<出库,1>	湛江佳信公司出库单						
	<"出库单号:" & [出库单编号]>						
	<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
<END>							

这样处理虽然解决了冲突的问题，但是却浪费了一行的打印空间，如果你在意这个问题，可以将第一行隐藏起来，这样既定义了有效区域，同时又不会对打印效果带来任何影响。

4.6.2.42 分割打印

前面提到，我们可以通过设置细节区标示的参数，来指定每页可打印的细节区数，例如：
<学生,-30>
表示每页最多打印 30 个学生，在不够 30 个的情况下，自动补充空行。

有的时候，我们会有一些特殊的要求，例如我们希望分组打印学生花名册，每组 9 个学生，每页最多打印 3 组，可以这样设置

<学生,(-9|3)>

4.6.3 报表编程

4.6.3.1 快速入门

一、测试快速入门代码

将下面的代码复制到命令窗口执行，你现在无需理解这些代码，接下来我们会逐行分析。

```
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
For i As Integer = 0 To 9 '向工作表的单元格中设置值
    Sheet(i, 0).Value = (i + 1) * 10
    Sheet(i, 1).Value = (i + 1) * 100
    Sheet(i, 2).Value = (i + 1) * 1000
Next
Book.Save("c:\reports\test.xls") '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

二、定义一个新的工作簿

工作簿的类型为 XLS.Book，定义一个新工作簿的代码为：

```
Dim Book As New XLS.Book
```

也可以根据现有的文件生成一个工作簿:

```
Dim Book As New XLS.Book("c:\reports\test.xls")
```

三、获得指定位置的工作表

工作表的类型为 XLS.Sheet，工作簿有一个属性 Sheets，返回该工作簿所有工作表的集合，例如：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
```

变量 Sheet 引用的是第一个工作表。

每一个工作簿默认包括一个工作表，可以通过 Sheets 的 Add 方法来增加工作表。

例如：

```
Dim Book As New XLS.Book()
Dim Sheet1 As XLS.Sheet = Book.Sheets(0)
Book.Sheets.Add() '增加一个工作表
Dim Sheet2 As XLS.Sheet = Book.Sheets(1) '引用新增加的工作表
```

如果没有事先用 Add 方法增加一个工作表，下面的代码会报错：

```
Dim Sheet2 As XLS.Sheet = Book.Sheets(1) '引用新增加的工作表
```

Add 方法可以带一个参数，指定新增表的名称，例如：

```
Book.Sheets.Add("统计") '增加一个名为“统计”的工作表。
```

同样我们可以通过名称来引用工作表，例如：

```
Dim Sheet2 As XLS.Sheet = Book.Sheets("统计")
```

四、获得指定位置的单元格

单元格的类型为 XLS.Cell，通过行号和列号可以获得指定位置的单元格，例如

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Dim Cell As XLS.CEll = Sheet(1,2)
```

变量 Cell 引用的是第二行第三列位置的单元格。

单元格会自动创建，例如：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Dim Cell As XLS.CEll = Sheet(99,99)
```

变量 Cell 引用的是第 100 行第 100 列位置的单元格，如果不存在该单元格，则工作表自动扩展到 100 行 100 列

五、为单元格指定值

单元格有一个 Value 属性，用于获得或设置单元格的值。

例如：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
For i As Integer = 0 To 9
    Sheet(i, 0).Value = (i + 1) * 10
    Sheet(i, 1).Value = (i + 1) * 100
    Sheet(i, 2).Value = (i + 1) * 1000
Next
```

六、保存工作簿

XLS.Book 有一个方法 Save，可以将工作簿保存为指定的文件。

将下面的代码复制到命令窗口执行：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
For i As Integer = 0 To 9
    Sheet(i, 0).Value = (i + 1) * 10
    Sheet(i, 1).Value = (i + 1) * 100
    Sheet(i, 2).Value = (i + 1) * 1000
Next
Book.Save("c:\reports\test.xls")
```

执行后，将在 C 盘的 reports 目录下生成一个 Excel 文件：test.xls

七、打开工作簿

我们利用 Process 类来打开工作簿：

```
Dim Proc As New Process '打开工作簿
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

请参考: Process

4.6.3.2 格式化单元格

XLS.Cell 类型除了有 Value 属性用于设置单元格的值外，还有一个 Style 属性用于设置单元格的样

式。
样式的类型为 XLS.Style，该类型有很多属性，例如 ForeColor 用于设置单元格字体的颜色。
XLS.Book 有一个 NewStyle 方法，用于创建一个新的样式。
复制下面的代码到命令窗口执行，可以看到第一列偶数行单元格的字体颜色被设成红色：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Dim Style As XLS.Style = Book.NewStyle() '定义新样式
Style.ForeColor = Color.Red '设置样式的字体颜色
For i As Integer = 0 To 9
    Sheet(i, 0).Value = i
    If i Mod 2 =0 Then '如果是偶数行
        Sheet(i,0).Style =Style '设置单元格样式
    End If
Next
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

样式的其它属性将在后面介绍。

4.6.3.3 最简单的统计报表

统计订单表中不同产品的销售额，用 Excel 形式输出：

```
Dim dt As DataTable = DataTables("订单")
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Names As List(Of String) = dt.GetValues("产品")
Sheet(0,1).Value = "产品销售统计"
For i As Integer = 0 to Names.Count - 1
    Sheet(i + 1, 0).Value = Names(i)
    Sheet(i + 1, 1).Value = dt.Compute("Sum(数量)","[产品] = '" & Names(i) & "'")
Next
'下面的代码用于将首行字体加粗
Dim Style As XLS.Style = Book.NewStyle
Style.FontBold = True
Sheet(0,1).Style = Style
'打开工作簿
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

直接在命令窗口执行上面的代码，可得到：

	A	B	C	D
1		产品销售统计		
2	PD01	11290		
3	PD02	18200		
4	PD03	7000		
5	PD04	5480		
6	PD05	10400		
7				
8				

4.6.3.4 导出数据

虽然 Foxtable 已经有导出 Excel 的功能，但是不够灵活。
如果你有更多的要求，可以用 Excel 报表来实现导出功能。

示例一

例如下面的代码，导出订单表，用红色背景标出折扣超过 0.15 的单元格：

```
Dim dt As Table = Tables("订单")
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Style As Xls.Style = Book.NewStyle '新建一个样式
Style.BackColor = Color.Red '样式的背景颜色设为红色
For c As Integer = 0 To dt.Cols.Count -1 '添加列标题
    Sheet(0, c).Value = dt.Cols(c).Name
Next
For r As Integer = 0 To dt.Rows.Count - 1 '填入数据
    For c As Integer = 0 To dt.Cols.Count -1
        Sheet(r +1, c).Value = dt.rows(r)(c)
    Next
    If dt.rows(r)("折扣") >= 0.15 Then '如果折扣大于等于 0.15
        Sheet(r + 1,dt.Cols("折扣").Index).Style = Style '设置折扣单元格的样式
    End If
Next
'打开工作簿
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

在命令窗口执行上面的代码，得到：

	A	B	C	D	E	F	G
1	产品	客户	雇员	单价	折扣	数量	日期
2	PD05	CS03	EP04	17	0.1	650	36162
3	PD02	CS01	EP01	20	0.1	400	36163
4	PD05	CS04	EP02	17	0	320	36163
5	PD01	CS03	EP04	18	0.15	80	36164
6	PD03	CS03	EP04	10	0	490	36164
7	PD04	CS04	EP05	17.6	0	240	36167
8	PD05	CS03	EP04	17	0	160	36167
9	PD01	CS04	EP05	14.4	0	200	36168
10	PD03	CS02	EP02	8	0.1	200	36168
11	PD01	CS02	EP01	18	0.2	800	36170
12	PD01	CS04	EP02	14.4	0.05	500	36170

你如果细心的话，你会发现生成的 Excel 文件中，日期列内容全变成了数字。
你可以在 Excel 中手工将日期列的格式设为日期型，但这不是一个好方法。
最好是修改上面的导出代码，在打开工作簿的代码之前插入下面几行：

```
Dim St2 As XLS.Style = Book.NewStyle
St2.Format = "yyyy-MM-dd"
Sheet.Cols(dt.Cols("日期").Index).Style = st2
```

工作表 Sheet 由行和列组成，Rows 返回其行的集合，Cols 返回其列的集合，行和列都可以设置样式(Style)，行和列将在后面详细介绍。

示例二

如果要导出部分列，或者要自定义列宽，或者要自定义列标题，可以参考下面的代码：

```
Dim dt As Table = Tables("订单")
Dim nms() As String = {"产品","客户","数量","单价","金额","日期"} '要导出的列名
Dim caps() As String = {"产品名称","用户名称","数量","单价","金额","日期"} '对应的列标题
Dim szs() As Integer = {100,100,80,80,80,120}'对应的列宽
Dim Book As New XLS.Book '定义一个Excel工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim st As XLS.Style = Book.NewStyle '日期列的显示格式
st.Format = "yyyy-MM-dd"
For c As Integer = 0 To nms.length -1
    Sheet(0, c).Value = caps(c) '指定列标题
    Sheet.Cols(c).Width = szs(c) '指定列宽
    If dt.Cols(nms(c)).IsDate Then '如果是日期列
        Sheet.Cols(c).Style = st '设置显示格式
    End If
Next
For r As Integer = 0 To dt.Rows.Count - 1 '填入数据
    For c As Integer = 0 To nms.length -1
        Sheet(r +1, c).Value = dt.rows(r)(nms(c))
    Next
Next
```

```
Dim dlg As New SaveFileDialog '定义一个新的SaveFileDialog
dlg.Filter= "Excel文件|*.xls" '设置筛选器
If dlg.ShowDialog = DialogResult.Ok Then
    Book.Save(dlg.FileName)
    Dim Proc As New Process
    Proc.File = dlg.FileName
    Proc.Start()
End If
```

4.6.3.5 合并数据

假定有一个下图所示的 Excel 格式的订单表：

	A	B	C	D	E	F	G	H
1	编号	产品	客户	雇员	单价	折扣	数量	日期
2	HO1	PDO5	CSO3	EPO4	17	0.1	650	1999-01-02
3	HO2	PDO2	CSO1	EPO1	20	0.1	400	1999-01-03
4	HO3	PDO5	CSO4	EPO2	17	0	320	1999-01-03
5	HO4	PDO1	CSO3	EPO4	18	0.15	80	1999-01-04
6	HO5	PDO3	CSO3	EPO4	10	0	490	1999-01-04
7	HO6	PDO4	CSO4	EPO5	17.6	0	240	2012-08-11
8	HO6	PDO5	CSO3	EPO4	17	0	160	2012-08-11

需要将其数据合并到 Foxtable 的订单表中，可以参考以下代码：

```
Dim Book As New XLS.Book("c:\test\订单.xls")
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Tables("订单").StopRedraw()
'注意以下数组中列名称的顺序,必须和 Excel 表中的列顺序一致
Dim nms() As String = {"编号","产品","客户","雇员","单价","折扣","数量","日期"}
'注意下面的循环变量从 1 开始，而不是从 0 开始，因为 Excel 表的第一行是标题
For n As Integer = 1 To Sheet.Rows.Count -1
    Dim r As Row = Tables("订单").AddNew()
    For m As Integer = 0 To nms.Length - 1
        r(nms(m)) = Sheet(n,m).Value
    Next
Next
Tables("订单").ResumeRedraw()
```

如果订单表中已经存在相同编号的订单，那么就跳过此订单，可以将代码改为：

```
Dim Book As New XLS.Book("c:\test\订单.xls")
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Tables("订单").StopRedraw()
'注意以下数组中列名称的顺序,必须和Excel表中的列顺序一致
```

```
Dim nms() As String = {"编号","产品","客户","雇员","单价","折扣","数量","日期"}
'注意下面的循环变量从1开始,而不是从0开始,因为Excel表的第一行是标题
For n As Integer = 1 To Sheet.Rows.Count -1
    Dim bh As String = sheet(n,0).Text
    If DataTables("订单").Find("编号 = '" & bh & "'") Is Nothing Then '如果不存在同编号的订单
        Dim r As Row = Tables("订单").AddNew()
        For m As Integer = 0 To nms.Length - 1
            r(nms(m)) = Sheet(n,m).Value
        Next
    End If
Next
Tables("订单").ResumeRedraw()
```

再进一步，如果存在相同编号的订单，则覆盖原来的订单，否则新增一个订单，代码为：

```
Dim Book As New XLS.Book("c:\test\订单.xls")
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Tables("订单").StopRedraw()
Dim nms() As String = {"编号","产品","客户","雇员","单价","折扣","数量","日期"}
For n As Integer = 1 To Sheet.Rows.Count -1
    Dim bh As String = sheet(n,0).Text
    Dim dr As DataRow = DataTables("订单").Find("编号 = '" & bh & "'")
    If dr Is Nothing Then '如果不存在同编号的订单
        dr =  DataTables("订单").AddNew()
    End If
    For m As Integer = 0 To nms.Length - 1
        dr(nms(m)) = Sheet(n,m).Value
    Next
Next
Tables("订单").ResumeRedraw()
```

提示：其实我们用 Merger 合并数据会更加简单，上述代码的目的是给大家提供另一种思路，因为在一些特殊情况下，Merger 可能无法正常合并数据。

4.6.3.6 自动生成多层表头

在编码生成 Excel 工作表(Sheet)的时候，如果遇到多层表头，这对于大多数人是一个难题。Table 有一个 CreateSheetHeader 方法，可以为 Sheet 自动生成表头，包括多层表头。

语法：

CreateSheetHeader(Sheet, StartRow, StartCol, VisibleOnly)

Sheet：　工作表

StartRow：　可选参数，表头在工作表中的起始行，默认为 0。

StartCol：　可选参数，表头在工作表中的起始列，默认为 0。

VisibleOnly：可选参数，是否只包括可见列，默认为 True。

示例一

假定有下面这样一个表：

	产品	东部				南部				总计
		一季度	二季度	三季度	四季度	一季度	二季度	三季度	四季度	
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										

执行代码：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Tables("产品").CreateSheetHeader(Sheet) '生成表头
Book.Save("c:\reports\test.xls")
```

即可在工作表中生成下图所示的多层表头：

A	B	C	D	E	F	G	H	I	J
第一列	一季度				二季度				总计
	东部	南部	西部	北部	东部	南部	西部	北部	

示例二

如果导出具有多层表头的 Table 的数据，可以参看下面的代码：

```
Dim tbl As Table = Tables("产品")
Dim hdr As Integer = tbl.HeaderRows '获得表头的层数
Dim cnt As Integer
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
tbl.CreateSheetHeader(Sheet) '生成表头
For c As Integer = 0 To tbl.Cols.Count - 1
    If tbl.Cols(c).Visible Then
        For r As Integer = 0 to tbl.Rows.Count - 1
            sheet(r + hdr,cnt).value = tbl(r,c)
        Next
        cnt = cnt + 1
    End If
Next
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
```

```
Proc.Start()
```

这一段代码只导出可见列，如果是导出所有列，代码会更加简单：

```
Dim tbl As Table = Tables("产品")
Dim hdr As Integer = tbl.HeaderRows '获得表头的层数
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
tbl.CreateSheetHeader(Sheet,0,0,False) '生成表头
For c As Integer = 0 To tbl.Cols.Count - 1
    For r As Integer = 0 to tbl.Rows.Count - 1
        sheet(r + hdr,c).value = tbl(r,c)
    Next
Next
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.3.7 从 Excel 报表提取数据

Foxtable 可以根据 Excel 模板批量生成 Excel 表格，反过来，我们也可以从 Excel 表格提取数据写入 Foxtable。

假定有一个下图所示的 Excel 文件，发给所有员工自行填写，然后将所有填写好的文件放在同一个目录，现在要求编写代码从这些 Excel 文件中提取数据，写入到 Foxtable 的 员工表中：



员工资料卡

编号:1

姓名	王伟	出生日期	1962/2/19
部门	商务部	雇用日期	1992/8/14
性别	男	职务	副总裁(销售)
地址	罗马花园 890 号		
家庭电话	(010) 65559482	办公电话	3457

王伟获南京大学商业学士学位，获该校国际营销博士学位。他能说流利的法语和意大利语并能阅读德语。他加入公司时是销售代表，被提拔为销售经理并升任销售副总裁。王伟是销售管理圆桌协会，北京商业总会和太平洋周边进口协会的成员。

代码是很简单的，只是将 Excel 各单元格的内容写入新增行对应列而已：

```
For Each file As String In filesys .GetFiles("c:\Data")
    If file.EndsWith(".xls") OrElse file.EndsWith(".xlsx") Then
        Dim Book As New XLS.Book(file)
        Dim Sheet As XLS.Sheet = Book.Sheets(0)
        Dim dr As DataRow = DataTables("员工").AddNew
        dr("姓名") = sheet(4,1).Text
        dr("部门") = sheet(5,1).text
        dr("出生日期") = sheet(4,3).text
        dr("雇佣日期") = sheet(5,3).text
        dr("性别") = sheet(6,1).text
        dr("职务") = sheet(6,3).text
        dr("地址") = sheet(7,1).text
        dr("家庭电话") = sheet(8,1).text
        dr("办公电话") = sheet(8,3).text
        dr("备注") = sheet(9,0).text
    End If
Next
```

如果要同时提取照片，可以将代码改为：

```
For Each file As String In filesys .GetFiles("c:\Data")
    If file.EndsWith(".xls") OrElse file.EndsWith(".xlsx") Then
        Dim Book As New XLS.Book(file)
        Dim Sheet As XLS.Sheet = Book.Sheets(0)
        Dim dr As DataRow = DataTables("员工").AddNew
        dr("姓名") = sheet(4,1).Text
        dr("部门") = sheet(5,1).text
        dr("出生日期") = sheet(4,3).text
        dr("雇佣日期") = sheet(5,3).text
        dr("性别") = sheet(6,1).text
        dr("职务") = sheet(6,3).text
        dr("地址") = sheet(7,1).text
        dr("家庭电话") = sheet(8,1).text
        dr("办公电话") = sheet(8,3).text
        dr("备注") = sheet(9,0).text
        Dim fl As String = ProjectPath & "Attachments\" & dr("姓名") & ".jpg"
        If Sheet(4,4).SaveImage(fl) Then '如果照片保存成功
            dr("照片") = fileSys.GetName(fl)
        End If
    End If
Next
```

4.6.3.8 基于模板

在基础应用篇，我们已经介绍了利用 Excel 文件作为模板生成 Excel 报表的方法。
利用 Foxtable 提供的数据引用方法，可以大大减轻设计 Excel 报表的工作量。
设计好报表模板后，可能需要通过窗口中的按钮来生成 Excel 报表，而不是通过菜单，此时可以参考下面的代码：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls")
Dim fl As String = ProjectPath & "Reports\出库单.xls"
Book.Build() '生成细节区
Book.Save(fl) '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = fl
Proc.Start()
```

如果要直接打印，代码为：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls")
Dim fl As String = ProjectPath & "Reports\出库单.xls"
Book.Build() '生成细节区
Book.Save(fl) '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = fl
Proc.Verb = "Print" '指定动作
Proc.Start()
```

Book 的 Build 方法用于生成细节区，如果模板文件中包含细节区定义，务必调用此方法。
此外，模板只能引用单元格的数据，并进行一些简单的计算统计，所以某些时候并不能满足我们的要求。
我们可以预先设计好模板，简单的部分通过数据引用来实现，然后利用代码动态修改模板，满足一些特殊的需求。
例如在基础应用的示例二中，我们通过模板生成了下图所示的出库单：

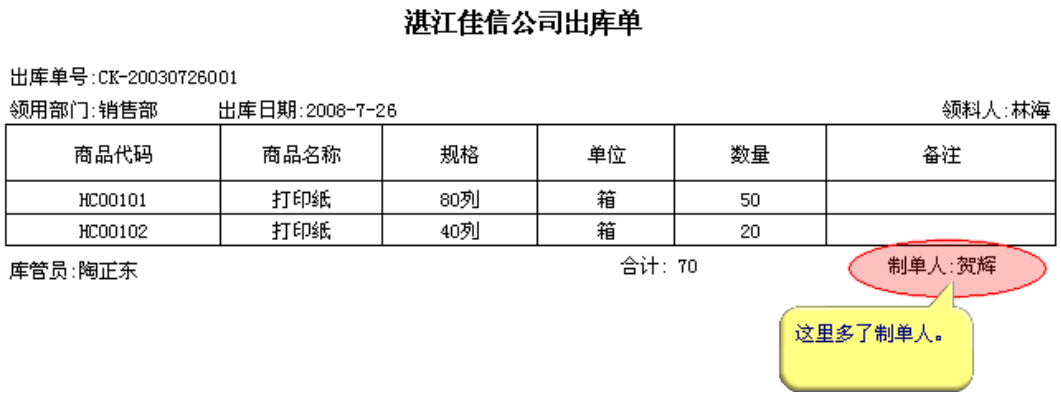
B	C	D	E	F	G
	湛江佳信公司出库单				
出库单号:CK-20030726001					
领用部门:销售部	出库日期:2008-7-26				领料人:林海
商品代码	商品名称	规格	单位	数量	备注
HC00101	打印纸	80列	箱	50	
HC00102	打印纸	40列	箱	20	
库管员:陶正东			合计:	70	

现在我们希望在出库单的后面加上制单人信息，制单人等于当前登录用户名， 代码很简单：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls") '打开模板
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet(6,6).Value = "制单人:" & User.Name '修改模板,加入制单人信息
Book.Build() '生成细节区
Book.Save("c:\reports\test.xls") '保存工作簿
Dim Proc As New Process '打开工作簿
```

```
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

在命令窗口执行上面的代码，会得到下图所示的出库单：



湛江佳信公司出库单

出库单号:CK-20030726001

领用部门:销售部　　出库日期:2008-7-26　　领料人:林海

商品代码	商品名称	规格	单位	数量	备注
HC00101	打印纸	80列	箱	50	
HC00102	打印纸	40列	箱	20	

库管员:陶正东　　合计：70

制单人:贺辉

这里多了制单人。

通过动态修改模板文件，我们可以完成更多复杂的功能，例如可以修改细节区定义，使得同一个模板文件能够用于不同的表；也可以在 Build 方法执行完成后，再根据需要向生成的报表中写入内容；甚至可以完全用代码定义一个模板，然后用 Build 方法来生成报表。

4.6.3.9 基于后台数据生成报表

从 Foxtable 2017 开始，Excel 报表可以直接基于后台数据生成。

一个例子

如果你设计了一个下图所示的 Excel 报表模板：



	A	B	C	D	E	F	G
1							<End>
2	<员工, 2>			员工资料卡			
3						<"编号:" & [编号]>	
4							
5		姓名	[姓名]	出生日期	[出生日期]		
6		部门	[部门]	雇用日期	[雇佣日期]		
7		性别	[性别]	职务	[职务]	[&照片, 3, 3, 150, 159]	
8		地址	[地址]				
9		家庭电话	[家庭电话]	办公电话	[办公电话]		
10		[备注]					
11							
12	<员工>						
13	<End>						

假定系统并没有加载员工表，或者员工表只加载了部分数据，我们用下面的代码，一样可以从后台提取未加载的数据来生成报表，而且非常简单：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\资料卡.xls")
Dim fl As String = ProjectPath & "Reports\资料卡.xls"
book.AddDataTable("员工","数据源名称","Select * from {员工} where  姓名  = '王伟'")
Book.Build()
Book.Save(fl)
Dim Proc As New Process
Proc.File = fl
Proc.Start()
```

提示：

AddDataTable方法用于从后台提取数据生成一个临时表，由Excel报表模板根据此临时表生成报表。例如上面的代码中，AddDataTable利用Select语句生成了一个临时表，表名为"员工"。
系统在生成Excel报表的过程中，优先调用AddDataTable生成的临时表，无需担心和现有同名表产生冲突。

AddDataTable的语法为：

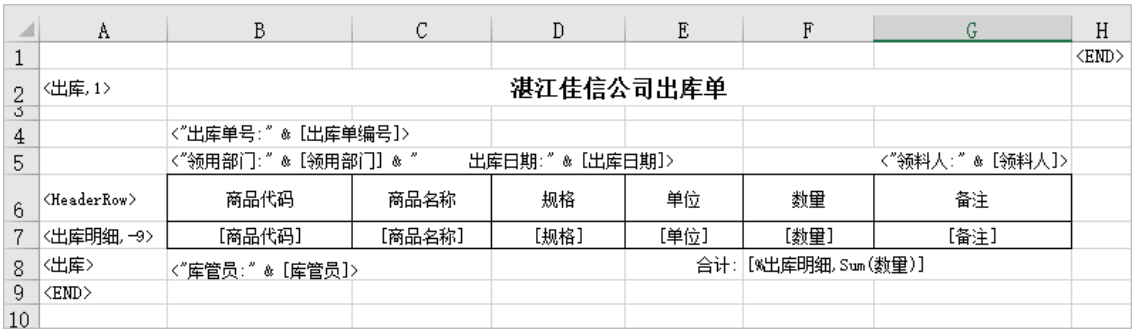
```
AddDataTable(Name,DataSouce,SelectString)
AddDataTable(Name,DataTable)
```

Name	字符型，临时表的名称，必须和报表模板中的表名保持一致。
DataSouce	字符型，用于指定数据源名称。
SelectString	字符型，用于指定 Select 语句
DataTable	DataTable 型，用于直接指定一个临时 DataTable 作为报表的数据来源，此 DataTable 不能是 DataTables 中的一员。 GroupTableBuilder 、 CrossTableBuilder 、 SQLGroupTableBuilder 、 SQLCrossTableBuilder、SQLJoinTableBuilder 和 DataTableBuilder 的 Build 方法，都有一个逻辑参数，将此参数设置为 True，将生成一个 DataTable，此 DataTable 将不会包括在 DataTables 中。 例如： Dim Book As New XLS.Book(ProjectPath & "Attachments\test.xls") Dim fl As String = ProjectPath & "Reports\test.xls" Dim g As New SQLGroupTableBuilder("统计表 1", "订单") g.ConnectionName = "数据源名称" g.Groups.AddDef("产品") g.Totals.AddDef("数量") book.AddDataTable("统计表 1", g.Build(True)) '添加临时表 Book.Build() Book.Save(fl) Dim Proc As New Process Proc.File = fl Proc.Start()

多个数据表和关联

我们可以添加多个临时表，而且可以在临时表之间建立关联。

假定设计了下图所示的报表模版，用于生成出库单。



	A	B	C	D	E	F	G	H
1								<END>
2	<出库,1>	湛江佳信公司出库单						
4		<"出库单号:" & [出库单编号]>						
5		<"领用部门:" & [领用部门] & " 出库日期:" & [出库日期]>					<"领料人:" & [领料人]>	
6	<HeaderRow>	商品代码	商品名称	规格	单位	数量	备注	
7	<出库明细,-9>	[商品代码]	[商品名称]	[规格]	[单位]	[数量]	[备注]	
8	<出库>	<"库管员:" & [库管员]>			合计:	[%出库明细,Sum(数量)]		
9	<END>							
10								

此模版涉及到出库和出库明细两个表，这两个表通过出库单号建立关联的。

不管系统是否已经加载了这两个表，我们都可以通过下面的代码生成一个出库单：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls")
Dim fl As String = ProjectPath & "Reports\出库单.xls"
book.AddDataTable("出库","数据源名称","Select * from {出库} where 出库单编号='CK-20030726001'") '添加父表
book.AddDataTable("出库明细","数据源名称","Select * from {出库明细} where 出库单编号='CK-20030726001'") '添加子表
book.AddRelation("出库","出库单编号","出库明细","出库单编号") '建立关联
Book.Build() '生成细节区
Book.Save(fl) '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = fl
Proc.Start()
```

提示： AddRelation 方法用于在临时表之间建立关联。

AddRelation 的语法为：

AddRelation(ParentTable,ParentCol,ChildTable,ChildCol)

ParentTable	字符型，父表名称。
ParentCol	字符型，父表关联列名称，如果有多个关联列，可以用一个字符型数组表示。
ChildTable	字符型，子表名称。
ChildCol	字符型，子表关联类名称，如果有多个关联列，可以用一个字符型数组表示。

4.6.3.10 保存为 PDF 文件

提示：此功能需要安装 Office 2007 或更高的版本。

为了避免用户修改生成好的 Excel 报表，我们可以将报表保存为 PDF 文件再打开，例如：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls")
```

```
Dim fl As String = ProjectPath & "Reports\出库单.pdf"
Book.Build() '生成细节区
Book.SaveToPDF(fl) '保存为PDF文件
Dim Proc As New Process '打开PDF文件
Proc.File = fl
Proc.Start()
```

提示：为了能打开生成的 PDF 文件，请先安装 PDF 阅读软件，例如：Adobe Reader X

4.6.3.11 动态设置打印条件

我们知道，Excel 报表默认只会打印选定行(记录)，如果你只选定 1 行，就会打印 1 行，如果你选定了 10 行，就会打印 10 行。
虽然可以设置一个条件表达式，例如只打印产品为 PD01 的行：

	A	B	C	D	E	F	G	H	I	J
1					订单					<END>
2										
3		产品	客户	雇员	单价	折扣	数量	金额	日期	
4	<订单>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<产品 = 'PD01'>
5	<END>									
6										

但是在模版设置的条件是固定的，而实际工作中，打印条件是动态的，不可能永远只打印符合同一条件的行，难道我们要为每种打印条件单独设置一个模版？显然这是不现实的。
通常我们可以在设计模版的时候，将打印条件设置为: <All>，然后筛选出符合条件的行，再打印报表，不过这还不够“自动化”。

再想一下，上面这个模版中，打印条件设置在第三行第九列这个单元格，如果我们在生成报表之前，动态合成打印条件，然后加载模版，将合成的条件写入到这个单元格中，再生成报表， 这样不就可以动态设置打印条件吗？
方案完全可行，例如我们要打印今天的订单:

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\订单.xls") '打开模板
Dim fl As String = ProjectPath & "Reports\订单.xls"
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet(3,9).Value = "<日期 = # " & Date.Today & "#>"  '写入打印条件
Book.Build() '生成报表
Book.Save(fl)
Dim Proc As New Process
Proc.File = fl
Proc.Start()
```

必须将打印条件写入准确的位置，模版中行和列的编号都是从0开始的额，切记。

4.6.3.12 报表模版复用

多个结构相同的数据表，需要同一个报表模版打印，我们只需在生成报表之前，将模版中定义细节区的表名设置为要打印的表名即可。

例如我们设置下图所示的一个模版，显然默认情况下，这个模版只能打印订单表的数据：

	A	B	C	D	E	F	G	H	I	J
1					订单					<END>
2										
3		产品	客户	雇员	单价	折扣	数量	金额	日期	
4	<订单>	[产品]	[客户]	[雇员]	[单价]	[折扣]	[数量]	[金额]	[日期]	<产品 = 'PD01'>
5	<END>									
6										

假定我们要用这个模版，打印一个名为“窗口 1_Table1”的表中的数据，可以参考下面的代码：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\订单.xls") '打开模板
Dim fl As String = ProjectPath & "Reports\订单.xls"
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet(3,0).Value = "<窗口1_Table1>"  '写入表名
Book.Build() '生成报表
Book.Save(fl)
Dim Proc As New Process
Proc.File = fl
Proc.Start()
```

必须将表名写入准确的位置，模版中行和列的编号都是从0开始的额，切记。

如果是多行细节区，例如：



记得两处细节区的定义都需要替换：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\资料卡.xls") '打开模板
Dim fl As String = ProjectPath & "Reports\资料卡.xls"
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet(1,0).Value = "<窗口1_Table1, 1>"  '细节区定义一
Sheet(11,0).Value = "<窗口1_Table1>"  '细节区定义二
Book.Build() '生成报表
Book.Save(fl)
Dim Proc As New Process
Proc.File = fl
Proc.Start()
```

4.6.3.13 跳过指定的表

有的时候 Excel 模板文件中的表，并非全部是作为模板使用的，部分表只是作为常规的表存在，不希望 Build 方法处理这些表。

Book 的 Build 方法可以带多个参数，用于指定哪些位置的表需要忽略。

示例

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\员工档案.xls")
Dim fl As String = ProjectPath & "Reports\员工档案.xls"
Book.Build(0) '生成细节区,忽略第一个表
Book.Save(fl) '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = fl
```

```
Proc.Start()
```

上面的代码忽略了模板文件中的第一个表。

4.6.3.14 直接插入图片

单元格的 Value 属性，不仅可以设置为数据，还可以设置为一个 XLS.Picture 对象。

XLS.Picture 用于定义一个可插入到 Excel 单元格的图片，语法为：

```
New XLS.Picture(Image)
```

或

```
New XLS.Picture(Image, Left, Top, Width, Height)
```

参数：

Image: 要插入的图片。

Left: 左边距，单位为像素。

Top: 上边距，单位为像素。

Width： 宽度，单位为像素。

Height: 高度，单位为像素。

可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

例如：

```
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
For i As Integer = 0 To 9 '向工作表的单元格中设置值
    Sheet(i, 0).Value = (i + 1) * 10
    Sheet(i, 1).Value = (i + 1) * 100
    Sheet(i, 2).Value = (i + 1) * 1000
Next
Sheet(10,2).Value = New XLS.Picture(GetImage("c:\Foxtable.Ico"))
Book.Save("c:\reports\test.xls") '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.3.15 一个小问题

在编写帮助的时候，我曾经写过一段简单的代码：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls")
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Book.Build()
Sheet(Sheet.Rows.Count, 0).Value = "制单人:" & User.Name
Book.Save("c:\reports\test.xls")
```

执行的结果很奇怪，并没有如我所料在最后一行出现制单人，也就是说第四行代码没有任何效果，这一度让我百思不得其解。

这是因为执行 Build 方法之后，原来的模板工作表会被全部删除，取而代之的是基于模板生成的新的工作表，而变量 sheet 在引用的是模板工作表，所以第四行代码是无效的。
因此，如果要在 Build 方法生成后修改生成的报表，必须使变量 Sheet 指向新生成的工作表：

```
Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls")
Dim Sheet As XLS.Sheet
Book.Build()
Sheet = Book.Sheets(0)
Sheet(Sheet.Rows.Count, 0).Value = "制单人:" & User.Name
Book.Save("c:\reports\test.xls")
```

4.6.3.16 使用预定义的报表

我们知道，通过下图所示的管理器按钮，可以预先定义好一些 Excel 报表：



XlS.Reports 是一个集合，可以返回定义好的、指定名称的 Excel 报表，类型为 XLS.Report。
例如：

```
Dim rp As XLS.Report
rp = XLS.Reports("员工资料卡")
```

XLS.Report 的属性和方法有：
Name： 字符型，返回报表名称
TempLate： 字符型，返回模板文件名称
FileName： 字符型，返回目标文件名称
TableName：字符型，返回所属表的表名
Build： 方法，生成 Excel 报表
示例一
列出所有定义好的报表及其模板文件:

```
For Each rp As XLS.Report In XLS.Reports
    Output.Show(rp.Name)
    Output.Show(rp.TempLate)
Next
```

示例二
生成员工资料卡报表：

```
XLS.Reports("员工资料卡").Build()
```

4.6.4 类型介绍

4.6.4.1 Book(工作簿)

4.6.4.1.1 属性

Book 的主要属性有：

- **Sheets**

  返回工作表集合，通过此集合获得指定位置的工作表。

  示例

```
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
```

  每一个工作簿默认包括一个工作表，可以通过 Sheets 的 Add 方法来增加工作表。
  例如：

```
Dim Book As New XLS.Book("c:\reports\test.xls")
Dim Sheet1 As XLS.Sheet = Book.Sheets(0)
Book.Sheets.Add() '增加一个工作表
Dim Sheet2 As XLS.Sheet = Book.Sheets(1) '引用新增加的工作表
```

  如果没有事先用 Add 方法增加一个工作表，下面的代码会报错：

```
Dim Sheet2 As XLS.Sheet = Book.Sheets(1) '引用新增加的工作表
```

- **DefaultFont**

  返回或者设置工作簿的默认字体。

  示例

```
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Book.DefaultFont = New Font("宋体",9) '设置默认字体
```

- **MergeFree**

  如果报表模板中设置了合并列，可以用此属性指定合并类型，默认为标准合并，如果需要自由合并，可以将此属性设为 True。

4.6.4.1.2 方法

方法	说明
Save	保存工作簿，语法为: Save(File) File: 文件名称，含路径。 示例 Dim Book As New XLS.Book '定义一个 Excel 工作簿 Book.Save("c:\reports\test.xls") '保存工作簿
SaveToPdf	保存工作簿为 PDF 文件，语法为： SaveToPdf(File) File: 文件名称，含路径。
NewStyle	创建并返回一个新的样式。 示例 Dim Book As New XLS.Book Dim Sheet As XLS.Sheet = Book.Sheets(0) Dim Style As XLS.Style = Book.NewStyle() '定义新样式 Style.ForeColor = Color.Red '设置样式的字体颜色 For i As Integer = 0 To 9 Sheet(i, 0).Value = i If i Mod 2 =0 Then '如果是偶数行 Sheet(i,0).Style =Style '设置单元格样式 End If Next Book.Save("c:\reports\test.xls")
Build	如果 Book 来自于一个模板文件，必须先调用方法，才能生成细节区。 示例： Dim Book As New XLS.Book(ProjectPath & "Attachments\出库单.xls") Book.Build() '生成细节区 Book.Save("c:\reports\test.xls") '保存工作簿 Dim Proc As New Process '打开工作簿 Proc.File = "c:\reports\test.xls" Proc.Start()

4.6.4.2 Sheets(工作表集合)

4.6.4.2.1 属性

Sheets 表示工作表集合，主要属性有：

- Count
  整数型，返回总的工作表数量。

- SelectedIndex
  整数型，设置打开工作簿时，默认选择哪一个工作表。

4.6.4.2.2 方法

Sheets 表示工作表集合，主要方法有：

- **Add**

  增加一个工作表。
  新创建的工作簿，默认只有一个工作表，可以通过 Add 方法来增加工作表。

  语法:

  Add()
  Add(Name)
  Add(Sheet)

  Name: 指定工作表的名称
  Sheet：一个现有的工作表

  例如：

```
Dim Book As New XLS.Book()
Dim Sheet1 As XLS.Sheet = Book.Sheets(0)
Book.Sheets.Add() '增加一个工作表
Dim Sheet2 As XLS.Sheet = Book.Sheets(1) '引用新增加的工作表
```

  Add 方法可以带一个参数，指定新增表的名称，例如：

```
Book.Sheets.Add("统计") '增加一个名为“统计”的工作表。
```

  同样我们可以通过名称来引用工作表，例如：

```
Dim Sheet2 As XLS.Sheet = Book.Sheets("统计")
```

- **Insert**

  在指定位置插入一个工作表。

  语法：

  ```
  Insert(Index)
  Insert(Index,Name)
  Insert(Index,Sheet)
  ```

  Index: 指定插入位置
  Name: 指定工作表的名称
  Sheet：一个现有的工作表

  例如：

  ```
  Dim Book1 As New XLS.Book("C:\001.xls")
  Book1.Sheets.Insert(0,"订单")
  Book1.Save("C:\001.xls")
  ```

- **Remove**

  删除工作表。

  语法:

  ```
  Remove(Name)
  Add(Sheet)
  ```

  例如：

  ```
  Dim Book As New XLS.Book
  Book.Sheets.Add("统计")
  Book.Sheets.Remove("统计")
  ```

- **RemoveAt**

  删除指定位置的工作表。

  例如删除第一个工作表：

  ```
  Dim Book As New XLS.Book("c:\reports\test.xls")
  ```

```
Book.Sheets.RemoveAt(0)
```

示例

Add、Insert 和 Remove 方法也可以直接增加或删除一个现有的工作表，例如下面的代码，将订单工作表由文件 001.xls 移动到 002.xls：

```
Dim Book1 As New XLS.Book("C:\001.xls")
Dim Book2 As New XLS.Book("C:\002.xls")
Dim Sheet = Book1.Sheets("订单")
Book1.Sheets.Remove(Sheet)
Book2.Sheets.Add(Sheet)
Book1.Save("C:\001.xls")
Book2.Save("C:\002.xls")
```

4.6.4.3 Sheet(工作表)

4.6.4.3.1 属性

Sheet 表示工作表，主要属性有：

属性	说明
DefaultColumnWidth	返回或设置默认列宽，单位为像素。
DefaultRowHeight	返回或设置默认行高，单位为像素。
GridColor	返回或设置网格线的颜色。
ShowGridLines	逻辑型，是否显示网格线。
ShowHeaders	逻辑型，是否显示表头。
Visible	逻辑型，是否显示工作表。
Name	字符型，返回或设置工作表的名称。
Locked	逻辑型，是否锁定此工作表，禁止用户编辑。
Rows	返回行的集合，可以通过此集合获得指定位置的行。
Cols	返回列的集合，可以通过此集合获得指定位置的列。

示例：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet.GridColor = Color.Pink '设置网格颜色
Sheet.DefaultColumnWidth = 75 '设置默认列宽
Sheet.Locked = True '禁止编辑
Book.Save("c:\reports\test.xls") '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.4.3.2 方法

Sheet 有两个方法：

MergeCell

用于合并单元格。

语法：

MergeCell(RowFrom, ColFrom, RowCount, ColCount)

RowFrom： 合并起始行
ColFrom: 合并起始列
RowCount：合并行数
ColCount: 合并列数

示例：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet.MergeCell(2,2,10,10)
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

ClearMergedCells
用于取消所有合并单元格，无参数。

4.6.4.4 Rows(行集合)

4.6.4.4.1 属性

Rows 表示工作表的行集合。
主要属性：

- **Count**
  返回总的行数

- **Frozen**
  返回或设置冻结行数。
  例如要冻结前两行：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet.Rows.Frozen = 2
Book.Save("c:\reports\test.xls")
```

4.6.4.4.2 方法

Rows 表示工作表的行集合。
主要方法:

- **Add**
  增加一行。
  由于行可以自动增加，所以实际上很少用 Add 方法去增加行的。

- **Insert**
  在指定位置插入一行。

  语法：

  Insert(Index)

  Index：要插入行的位置。

- **RemoveAt**
  删除指定位置的行。

  语法：

  RemoveAt(Index）

  Index: 删除行的位置。

示例：

```
Dim Book As New XLS.Book("c:\reports\test.xls") '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Sheet.Rows.Insert(0) '在最前面插入一行
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.4.5 Cols(列集合)

4.6.4.5.1 属性

Cols 表示工作表的列集合。
主要属性：

- **Count**
  返回总的列数。

- **Frozen**
  返回或设置冻结列数。
  例如要冻结前两列：

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Sheet.Cols.Frozen = 2
Book.Save("c:\reports\test.xls")
```

4.6.4.5.2 方法

Cols 表示工作表的列集合。
主要方法:

- **Add**
  增加一列。
  由于列可以自动增加，所以实际上很少用 Add 方法去增加列的。

- **Insert**
  在指定位置插入一列。

  语法：

  Insert(Index)

  Index：要插入列的位置。

- **RemoveAt**
  删除指定位置的列。

  语法：

RemoveAt(Index）

Index: 删除列的位置。

示例

```
Dim Book As New XLS.Book("c:\reports\test.xls") '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Sheet.Cols.Insert(0) '在最前面插入一列
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.4.6 Row(行)

4.6.4.6.1 属性

Row 的主要属性有：

属性	说明
Height	返回或设置行高，单位为像素。
PageBreak	逻辑型，是否在该行强制换页。
Visible	逻辑型，是否显示该行。
Style	设置行的样式。行、列、单元格都可以设置样式，优先顺序是：单元格>行>列

示例：

```
Dim Book As New XLS.Book("c:\reports\test.xls") '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Style As XLS.Style = Book.NewStyle() '定义新样式
Style.ForeColor = Color.Red '设置样式的字体颜色
Sheet.Rows(0).Height = 40 '设置第 1 行的行高
Sheet.Rows(0).Style = Style '设置第 1 行的样式
Sheet.Rows(20).PageBreak = True '在第 21 行处强制换页
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.4.7 Col(列)

4.6.4.7.1 属性

Col 的主要属性有：

- **Width**
  返回或设置列宽，单位为像素。

- **PageBreak**
  逻辑型，是否在该列强制水平分页。

- **Visible**
  逻辑型，是否显示该列

- **Style**
  设置列的样式。行、列、单元格都可以设置样式，优先顺序是：单元格>行>列

示例

```
Dim Book As New XLS.Book("c:\reports\test.xls") '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Style As XLS.Style = Book.NewStyle() '定义新样式
Style.ForeColor = Color.Red '设置样式的字体颜色
Sheet.Cols(0).Width = 120 '设置第 1 列的宽度
Sheet.Cols(0).Style = Style '设置第 1 列的样式
Sheet.Cols(4).PageBreak = True '在第 5 列处强制分页
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.4.8 Cell(单元格)

4.6.4.8.1 Cell

Cell 的属性有：

- **Value**
  返回或设置单元格的值。

- **Formula**
  返回或设置单元格公式。

- **Text**
  只读属性，返回单元格的文本内容。

- **Style**
  设置单元格的样式。行、列、单元格都可以设置样式，优先顺序是：单元格>行>列

- **Hyperlink**
  字符型，设置单元格的超链接。
  可以是某个网页，例如“http://www.egrid20000.com”，也可以发送邮件，例如“mailto:joe@foo.com”，还可以指向某个文件，例如“c:\readme.doc”

Cell 的方法有：

- **Save**
  将单元格中的图片保存为文件。

示例：

```
Dim Book As New XLS.Book() '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Style As XLS.Style = Book.NewStyle() '定义新样式
Style.ForeColor = Color.Red '设置样式的字体颜色
Sheet(0,0).Style = Style
Sheet(0,0).Value = "邮件"
Sheet(0,0).Hyperlink = "Mailto:zjtdr@21cn.net"
Sheet(1,0).Formula = "=1+3"
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.4.9 Style(样式)

4.6.4.9.1 对齐数据

下面两个属性用于对齐数据。

AlignHorz

设置水平对齐方式，XLS.AlignHorzEnum 型枚举，主要可选值有：

Center： 居中

General：默认

Left： 靠左

Right： 靠右

AlignVert
设置垂直对齐方式，XLS.AlignVertEnum，主要可选值有：
Bottom： 靠底
Center： 居中
Top： 靠上
Undefined：默认
示例

```
Dim Book As New XLS.Book() '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Style As XLS.Style = Book.NewStyle() '定义新样式
Style.ForeColor = Color.Red '设置样式的字体颜色
Style.AlignHorz = XLS.AlignHorzEnum.Center
Style.AlignVert = XLS.AlignVertEnum.Center
Sheet.Rows(0).Height = 50
Sheet.Cols(0).Width = 120
Sheet(0,0).Value = "邮件"
Sheet(0,0).Style = Style
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

执行结果：



4.6.4.9.2 颜色相关

Style 中和颜色相关的属性有：

- **BackColor**
  返回或设置背景颜色。

- **ForeColor**
  返回或设置字体颜色。

之前大量的例子使用了这两个属性，这里就不再重复了。

4.6.4.9.3 边框设置

和边框设置相关的属性：

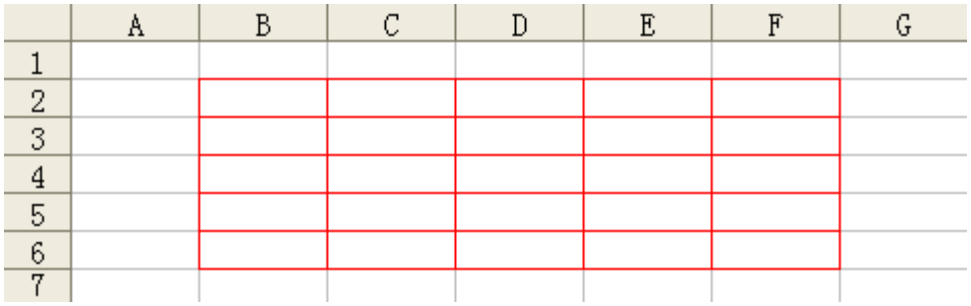
属性	说明	备注
BorderTop	返回或设置上边框的类型。	XLS.LineStyleEnum 型枚举，主要可选值有： Dashed: 虚线 Dotted：点线 Double：双实线 Hair：头发丝一样的细线(一个像素) Medium：中等实线 MediumDashDotDotted：中等的(短线+点+点) MediumDashDotted: 中等的(短线+点) MediumDashed: 中等虚线 None：无 Thick:粗线 Thin：细线 ThinDashDotDotted：细(短线+点+点) ThinDashDotted：细(短线+点)
BorderBottom	返回或设置下边框的类型。	
BorderLeft	返回或设置左边框类型。	
BorderRight	返回或设置右边框类型。	
BorderColorTop	返回或设置上边框颜色	
BorderColorBottom	返回或设置下边框颜色	
BorderColorLeft	返回或设置左边框颜色	
BorderColorRight	返回或设置上边框颜色	

示例

```
Dim Book As New XLS.Book() '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
Dim Style As XLS.Style = Book.NewStyle() '定义新样式
Style.BorderTop = XLS.LineStyleEnum.Thin
Style.BorderBottom = XLS.LineStyleEnum.Thin
Style.BorderLeft = XLS.LineStyleEnum.Thin
Style.BorderRight = XLS.LineStyleEnum.Thin
Style.BorderColorTop = Color.Red
Style.BorderColorBottom = Color.Red
Style.BorderColorLeft = Color.Red
Style.BorderColorRight = Color.Red
For r As Integer = 1 to 5
    For c As Integer =1 To 5
        Sheet(r,c).Style = Style
```

```
    Next
Next
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

生成的工作表：



4.6.4.9.4 斜线设置

和斜线设置有关的属性：

- **Diagonal**
  返回或设置斜线的方向，XLS.DiagonalEnum 型枚举，可选值有：

  Backward：反斜线
  Forward：斜线
  None：无

- **DiagonalColor**
  返回或设置斜线颜色

- **DiagonalStyle**
  返回或设置斜线的类型，XLS.LineStyleEnum 型枚举，主要可选值有：

  Dashed：虚线
  Dotted：点线
  Double：双实线
  Hair：头发丝一样的细线(一个像素)
  Medium：中等实线
  MediumDashDotDotted：中等的(短线+点+点)
  MediumDashDotted：中等的(短线+点)
  MediumDashed：中等虚线
  None：无
  Thick:粗线
  Thin：细线

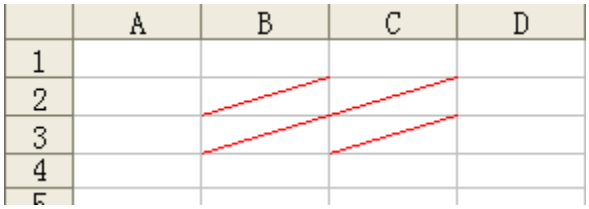
ThinDashDotDotted：细(短线+点+点)

ThinDashDotted：细(短线+点)

示例

```
Dim Book As New XLS.Book()
Dim Sheet As XLS.Sheet = Book.Sheets(0)
Dim Style As XLS.Style = Book.NewStyle()
Style.DiagonalStyle= XLS.LineStyleEnum.Thin
Style.DiagonalColor =Color.Red
Style.Diagonal = XLS.DiagonalEnum.Backward
For r As Integer = 1 to 2
    For c As Integer =1 To 2
        Sheet(r,c).Style = Style
    Next
Next
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

生成的工作表：



4.6.4.9.5 其他属性

Style 的其他属性：

- **Font**
  返回或设置字体。

- **Format**
  设置显示格式字符。

- **Locked**
  逻辑型，是否禁止编辑

- **WordWrap**
  逻辑型，单元格内容是否自动换行

- **Rotation**
  整数型，单元格内容旋转的角度。

4.6.5 打印设置

4.6.5.1 关于打印设置

工作表(Sheet)有一个 PrintSetting 属性，用于进行打印设置。
大多数 Excel 报表都是基于模板文件生成的，通常在设计模板文件的时候，就已经完成了打印设置，所以需要编码进行打印设置的时候并不多。
因此，一般的用户完全可以跳过本章节的全部内容。

4.6.5.2 页面设置

PrintSetting 有很多子属性，其中和页面设置相关的有：
PaperKind: 纸张类型代码
Landscape: 逻辑型，是否横向打印
MarginLeft: 左边距，单位为毫米
MarginRight: 右边距，单位为毫米
MarginTop: 上边距，单位为毫米
MarginBottom: 底边距，单位为毫米
示例

```
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
With Sheet.PrintSetting
    .PaperKind = 9 '设为 A4 纸
    .LandScape = True '横向打印
    .MarginLeft = 20 '左右边距设为 20 毫米
    .MarginRight = 20
    .MarginTop = 15 '上下边距设为 15 毫米
    .MarginBottom = 15
End With
```

下表列出了常用的纸张类型：

值	纸张类型
8	A3
9	A4
11	A5
12	B4
13	B5
66	A2

70 A6

纸张类型很多，全部纸张类型，请参考：纸张大全

4.6.5.3 缩放打印

PrintSetting 中和缩放打印有关的子属性有：

- AutoScale
  逻辑型，设为 False 按照比例缩放，设为 True，缩放到指定的页数。

- ScalingFactor
  整数型，指定缩放百分比，例如 50 表示缩小一半，100 表示不缩放，200 表示放大一半。
  必须将 AutoScale 属性设为 False，本属性才能发生作用。

- FitPagesAcross
  指定水平方向打印的页数，系统自动计算水平方向的缩放比例，以便水平方向能够在指定的页数内打印完。
  必须将 AutoScale 属性设为 True，本属性才能发生作用。

- FitPagesDown
  指定垂直方向打印的页数，系统自动计算垂直方向的缩放比例，以便垂直方向能够在指定的页数内打印完。
  必须将 AutoScale 属性设为 True，本属性才能发生作用。

示例
在命令窗口执行下面的代码，然后在 Excel 中打印预览，可以看到已经自动缩放为 1 页：

```
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
With Sheet.PrintSetting
    .AutoScale = True '自动缩放
    .FitPagesDown = 1 '垂直方向缩为 1 页
End With
For i As Integer = 0 To 50 '向工作表的单元格中设置值
    Sheet(i, 0).Value = (i + 1) * 10
    Sheet(i, 1).Value = (i + 1) * 100
    Sheet(i, 2).Value = (i + 1) * 1000
Next
Book.Save("c:\reports\test.xls") '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.5.4 页眉页脚

PrintSetting 中和页眉页脚有关的子属性有：
Header：字符型，指定页眉
Footer: 字符型，指定页脚
MarginHeader：指定页眉边距，单位为毫米
MarginFooter：指定页脚边距，单位为毫米
页眉页脚都分为左、中、右三部分，分别用字符串“&L”、 “&C”、 “&R”开头。
此外下列以字符“&”开始的字符串，都有着特殊的意义：

字符串	说明
&P	页号
&N	总页数
&D	当天日期
&T	当前时间
&A	工作表名称
&F	文件名，不含路径
&Z	文件名，含路径
&B	字体加粗切换
&I	字体倾斜切换
&U	下划线切换
&E	双下划线切换
&S	删除线切换
&X	上标切换
&Y	下标切换
&"[FontName]"	指定字体名称，例如： &"[黑体]"
&[FontSize]	指定字体大小，例如：&[8]

示例：

```
Dim Book As New XLS.Book '定义一个 Excel 工作簿
Dim Sheet As XLS.Sheet = Book.Sheets(0) '引用工作簿的第一个工作表
With Sheet.PrintSetting
    '页眉左边为日期,中间为时间,右边为表名
    .Header = "&L&D &C&T &R&A"
    '在页脚右边打印页号和总页数
    .Footer = "&R 第&P 页,总&N 页"
End With
For i As Integer = 0 To 100 '向工作表的单元格中设置值
    Sheet(i, 0).Value = (i + 1) * 10
    Sheet(i, 1).Value = (i + 1) * 100
```

```
    Sheet(i, 2).Value = (i + 1) * 1000
Next
Book.Save("c:\reports\test.xls") '保存工作簿
Dim Proc As New Process '打开工作簿
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.5.5 其他设置

PrintSetting 还包括以下子属性：
BlackAndWhite： 逻辑型，设为 True 将用黑白模式打印，即使你安装的是彩色打印机。
CenterHorizontal：逻辑型，表格是否在水平方向页面居中。
CenterVertical： 逻辑型，表格是否在垂直方向页面居中。
PrintGridlines： 逻辑型，是否打印网格线。
PrintHeaders： 逻辑型，是否打印列标题。

4.6.6 扩展 Excel 报表的功能

如果还没有接触窗口设计和图表生成的功能，可以暂时忽略本节的内容。
在窗口显示和打印 Excel 报表
为 Excel 报表增加绘图功能
为 Excel 报表增加图表功能
为 Exel 报表增加条形码功能
直接向 Excel 报表插入图片

4.6.7 标记与事件

4.6.7.1 使用标记

利用 Marks 集合，我们可以在报表模板中插入标记，然后在生成报表的时候，将标记替换为真正的值。
在 Excel 报表模板中插入一个标记的格式为：
<标记名称>
示例
假定报表中某个位置需要打印出当前登录用户的名称，我们可以在设计报表的时候，在该处插入标记：
<用户名>
标记的名称可以根据需要取。
然后在生成报表之前，利用 Marks 集合，将标记替换为真正的用户：

```
Dim Book As New XLS.Book("c:\book5.xls")
Book.Marks.Add("用户名",User.Name)
```

```
Book.Build()
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.7.2 报表事件

报表事件是针对高端用户的，一般用户可以忽略。
前面介绍的 Excel 报表功能，足以满足大多数场合的需要，需要配合事件代码使用的场合并不多。
为什么要有报表事件？
Excel 报表的表达式是基于 VBScript 的，功能有限，无法满足一些相对复杂的需求。
标记的出现，可以在一定程度上缓解这个问题，因为标记的值可以通过代码来设置。
如果能够更进一步，允许在报表生成过程中动态修改标记的值，那么就更加完美了。
为此 Foxtable 针对 Excel 报表提供了七个事件:
BeforeBuild： 生成某个报表之前执行
AfterBuild： 生成某个报表之后执行
BuildDetail： 生成某个细节区之前执行
BuildPageHeader： 生成某个页首之前执行
BuildPageFooter： 生成某个页尾之前执行
BuildGroupHeader：生成某个分组头之前执行
BuildGroupFooter：生成某个分组尾之前执行
通过这些事件，可以动态地修改标记的值，使得用模板生成的 Excel 报表，同样可以完成任意复杂的计算。
通常来说，只有遇到用常规方法和表达式无法实现的任务时，才可以考虑用标签加事件来解决。
通过菜单“打印输出”功能区，单击下面的按钮，可以为 Excel 报表设置事件代码：



区分模板
这六个事件是项目级别的，只要基于模板生成 Excel 报表，不管是通过代码还是菜单，都会触发这些事件。
显然大多数时候，不同的报表，需要不同的事件代码，所以我们需要在事件代码中知道正在基于哪一个模板文件生成报表。
所有这些事件的 e 参数，都有一个 e 参数 Book(工作簿)，我们可以利用 Book 的 TempLate 属性获得报表的模板文件名称。
这样我们就可以在事件中，为不同的报表，编写不同的代码：

```
Select e.Book.TempLate
    Case "模板 1"
        '代码
```

```
    Case "模板 2"
        '代码
    Case "模板 3"
        '代码
    Case "模板 4"
        '代码
End Select
```

需要注意的是，TempLate 返回的模板文件名称是不包括路径和后缀名的， 例如你使用的模板文件名称“出库单.xls”，那么 TempLate 属性的值等于“出库单”。

区分工作表

有的时候，一个模板文件可能会包括多个工作表，我们同样需要在代码中对此进行区分。

Book 有一个属性 CurrentSheetName 属性，用于返回正在生成的工作表的名称。

这样我们就可以在事件中，为不同的工作表，编写不同的代码：

```
Select e.Book.TempLate
    Case "模板 1"
        Select Case e.Book.CurrentSheetName
            Case "工作表 1"
                '代码
            Case "工作表 2"
                '代码
        End Select
    Case "模板 2"
        Select Case e.Book.CurrentSheetName
            Case "工作表 1"
                '代码
            Case "工作表 2"
                '代码
        End Select
    Case "模板"
        '代码
End Select
```

修改标记的值

在任何一个报表事件中，都可以通过 Marks 集合修改标记的值：

```
e.Book.Marks("标记名称") = 值
```

接下来我们将逐个介绍如何使用这七个报表事件。

4.6.7.3 事件介绍

4.6.7.3.1 BeforeBuild

在生成 Excel 报表前执行，通常在此事件中添加标记。

e 参数属性：

Book：工作簿

示例

```
Select e.Book.TempLate
    Case "模板 1"
        e.Book.Marks.Add("记录数",1)
        e.Book.Marks.Add("用户名",UserName)
    Case "模板 2"
        e.Book.Marks.Add("页号",1)
End Select
```

只有在直接通过系统菜单生成 Excel 报表的时候，才需要在 BeforeBuild 事件中添加标记，否则在报表生成代码中添加标记显得更为直接，例如：

```
Dim Book As New XLS.Book("c:\book5.xls")
Book.Marks.Add("用户名",User.Name) '添加标记
Book.Build()
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.6.7.3.2 BuildDetail

在生成某个细节区之前执行。

e 参数属性：

Book： 返回工作簿。

Region： 返回正在打印的细节区名称。

DataRow：返回正在打印的行。

示例一

假定基于客户表设计了一个 Excel 报表模板，希望能够在报表中统计每个客户的已结账和未结账的货款。

实现这个功能的步骤是：

1、首先在设计模板的时候，插入两个标签：

A	B	C	D
	客户	已结帐	未结帐
〈客户〉	[客户]	〈已结账〉	〈未结账〉

2、在 BeforeBuild 事件中，将这两个标签插入到 Marks 集合：

```
Select Case e.Book.TempLate
    Case "模板名称"
        e.Book.Marks.Add("已结账",0)
        e.Book.Marks.Add("未结账",0)
End Select
```

3、在 BuildDetail 事件中加入下面的代码：

```
Select Case e.Book.TempLate
    Case "模板名称"
        If e.DataRow IsNot Nothing Then
```

```
        Dim Filter1 As String = "客户 = '" & e.DataRow("客户") & "' And 已结账 = True"
        Dim Filter2 As String = "客户 = '" & e.DataRow("客户") & "' And 已结账 = False"
        e.Book.Marks("已结账") = DataTables("订单").Compute("Sum(金额)",Filter1)
        e.Book.Marks("未结账") = DataTables("订单").Compute("Sum(金额)",Filter2)
    End If
End Select
```

需要注意的是，如果定义细节区的时候，设置了自动添加空行，那么 e.DataRow 有可能返回 Nothing，所以我们需要在代码中对此进行判断，以免出错。

此外，如果报表模板中存在不止一个细节区，那么应该在 BuildDetail 事件代码中判断细节区的名称：

```
Select Case e.Book.TempLate
  Case "模板名称"
    If e.DataRow IsNot Nothing AndAlso e.Region = "客户" Then
      Dim Filter1 As String = "客户 = '" & e.DataRow("客户") & "' And 已结账 = True"
      Dim Filter2 As String = "客户 = '" & e.DataRow("客户") & "' And 已结账 = False"
      e.Book.Marks("已结账") = DataTables("订单").Compute("Sum(金额)",Filter1)
      e.Book.Marks("未结账") = DataTables("订单").Compute("Sum(金额)",Filter2)
    End If
End Select
```

示例二

如果还没有接触窗口设计，可以暂时忽略本示例。

我们之前介绍了一些扩展 Excel 报表功能的方法，例如利用 BarCodeBuilder 为 Excel 报表增加条码打印功能。

现在我们在此基础上进行扩展，为每一个细节区生成一个对应的条码图片。

实现这个功能的步骤为：

1、设计模版的时候，在细节区引用约定的图片文件"myimg.wmf"。

2、在 BuildDetail 事件中加入下面的代码：

```
Select Case e.Book.TempLate
  Case "模板名称"
    If e.DataRow IsNot Nothing Then
      Dim Bar As New BarCodeBuilder
      Bar.Symbology = Barpro.Symbology.Code39
      Bar.Code = e.DataRow("编码")
      Bar.BarRatio = 0.3
      Bar.BarHeight = 8
      bar.SaveImage(ProjectPath & "Images\myimg.wmf")
    End If
End Select
```

4.6.7.3.3 BuildPageHeader

Book： 返回工作簿。

请参考指定页首行

示例

假定需要在页首区域加入行号，格式为“第 X 页”。

实现步骤为：

1、设计模板的时候，在页首区域加入一个标签：<页号>

2、在 BeforeBuild 事件中，将两个标签插入到 Marks 集合：

```
Select Case e.Book.TempLate
    Case "模板名称"
        e.Book.Marks.Add("页号", "")
        e.Book.Marks.Add("pg", 0)
End Select
```

上面的代码其实加入了两个标签，模板只使用了“页号”标签，而标签“pg”则只用于在代码中记录页号。

3、在 BuildPageHeader 事件中加入代码：

```
Select Case e.Book.TempLate
    Case "模板名称"
        e.Book.Marks("pg") = e.Book.Marks("pg") + 1
        e.Book.Marks("页号") = "第" & e.Book.Marks("pg") & "页"
End Select
```

4.6.7.3.4 BuildPageFooter

在生成某个页尾前执行。

e 参数属性：

Book：　返回工作簿

PageRows：　返回一个 DataRows 集合，包括此页打印的所有行

TotalRows：　返回一个 DataRows 集合，包括截止到此页为止，所有已经打印的行。

示例

假定针对订单表设计了一个报表模板，希望在每页的页尾打印此页已结账和未结账金额小计，以及截止到此页为止，已结账和未结账金额合计。

实现步骤

1、在设计报表模版的时候，插入四个标签：<本页已结账>、<本页未结账>、<截止已结账>、<截止未结账>。

2、在 BeforeBuild 事件中，将四个标签插入到 Marks 集合：

```
Select Case e.Book.TempLate
    Case "模板名称"
        e.Book.Marks.Add("本页已结账", 0)
        e.Book.Marks.Add("本页未结账", 0)
        e.Book.Marks.Add("截止已结账", 0)
        e.Book.Marks.Add("截止未结账", 0)
End Select
```

3、在 BuildPageFooter 事件中加入代码：

```
Select Case e.Book.TempLate
    Case "模板名称"
```

```
    Dim Sum1, Sum2, Sum3, Sum4 As Double
    For Each dr As DataRow In e.PageRows
      If dr("已结账") = True
        Sum1 = Sum1 + dr("金额")
      Else
        Sum2 = Sum2 + dr("金额")
      End If
    Next
    For Each dr As DataRow In e.TotalRows
      If dr("已结账") = True
        Sum3 = Sum3 + dr("金额")
      Else
        Sum4 = Sum4 + dr("金额")
      End If
    Next
    e.Book.Marks("本页已结账") = Sum1
    e.Book.Marks("本页未结账") = Sum2
    e.Book.Marks("截止已结账") = Sum3
    e.Book.Marks("截止未结账") = Sum4
End Select
```

4.6.7.3.5 BuildGroupHeader

在生成某个分组头之前执行。

e 参数属性：

Book：　返回工作簿

TableName：字符型，返回分组的表名

ColumnName：字符型，返回分组的列名

DataRow：　DataRow 类型，返回此分组的第一个数据行

NewPage：　逻辑型，如果此分组头是因为换页而重复打印的，则返回 True，否则返回 False。

之前介绍的分组统计，统计结果只能在分组尾显示，有了 BuildGroupHeader 事件，我们就可以在分组头显示统计结果了。

示例

假定设计了一个报表模板，根据产品分组打印订单，希望在分组头打印金额和数量小计。

实现步骤

1、在设计模板的时候，在分组头插入两个标签：　<数量合计>、<金额合计>　。

2、在 BeforeBuild 事件中，将两个标签插入到 Marks 集合：

```
Select Case e.Book.TempLate
  Case "模板名称"
    e.Book.Marks.Add("数量合计", 0)
    e.Book.Marks.Add("金额合计", 0)
End Select
```

3、在 BuildGroupHeader 事件中加入代码：

```
Select Case e.Book.TempLate
```

```
    Case "模板名称"
        Dim Filter As String = "产品  = '" & e.Datarow("产品") & "'"
        e.Book.Marks("数量合计") = DataTables("订单").Compute("Sum(数量)",Filter)
        e.Book.Marks("金额合计") = DataTables("订单").Compute("Sum(金额)",Filter)
End Select
```

提示，如果模板中有多个分组，需要用 e 参数 TableName 和 ColumnName 来判断是哪一个分组，例如:

```
Select Case e.Book.TempLate
    Case "模板名称"
        If e.TableName = "订单" AndAlso e.ColumnName = "产品" Then
            Dim Filter As String = "产品  = '" & e.Datarow("产品") & "'"
            e.Book.Marks("数量合计") = DataTables("订单").Compute("Sum(数量)",Filter)
            e.Book.Marks("金额合计") = DataTables("订单").Compute("Sum(金额)",Filter)
        End If
End Select
```

4.6.7.3.6 BuildGroupFooter

生成某个分组尾后执行。

e 参数属性：

Book：　　返回工作簿

TableName:　字符型，返回分组的表名

ColumnName：字符型，返回分组的列名

GroupRows：　返回一个 DataRows 集合，包括此分组的所有行

TotalRows：　返回一个 DataRows 集合，包括截止到此分组为止，所有已经打印的行

示例

假定针对订单表设计了一个报表模板，根据客户进行分组，希望在每个分组尾打印该客户已结账和未结账金额小计。

实现步骤

1、设计报表模版的时候，在分组尾插入两个标签：<已结账>、<未结账>。

2、在 BeforeBuild 事件中，将两个标签插入到 Marks 集合：

```
Select Case e.Book.TempLate
    Case "模板名称"
        e.Book.Marks.Add("已结账", 0)
        e.Book.Marks.Add("未结账", 0)
End Select
```

3、在 BuildGroupFooter 事件中加入代码：

```
Select Case e.Book.TempLate
    Case "模板名称"
        Dim Sum1, Sum2 As Double
        For Each dr As DataRow In e.GroupRows
            If dr("已结账") = True
                Sum1 = Sum1 + dr("金额")
            Else
```

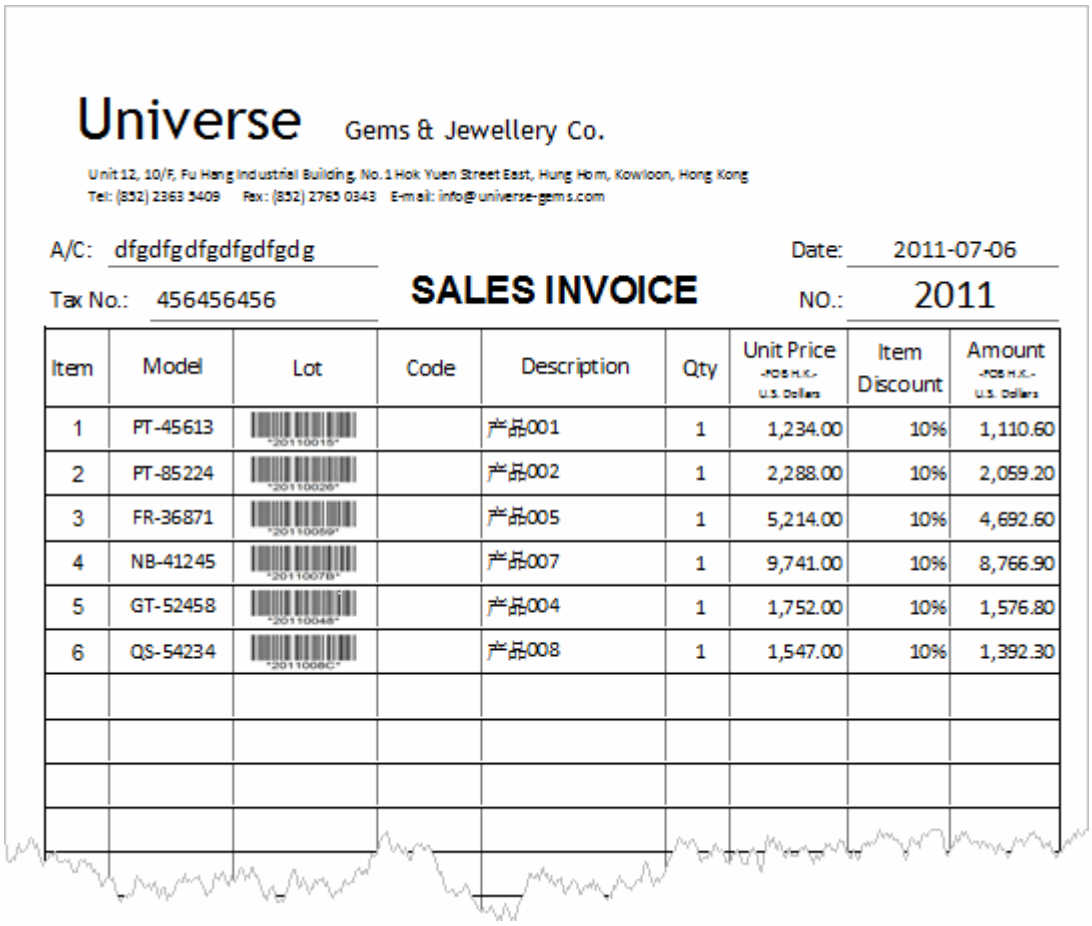
```
            Sum2 = Sum2 + dr("金额")
        End If
    Next
    e.Book.Marks("已结账") = Sum1
    e.Book.Marks("未结账") = Sum2
End Select
```

4.6.7.3.7 AfterBuild

在生成报表后执行，可以在这里对生成的报表进行进一步的加工。
e 参数属性：
Book：返回工作簿。

4.6.7.3.8 实战 - 用 Excel 报表批量生成条形码

除了使用条形码字体，我们还可以使用本节介绍的方法来批量生成更加专业的条形码。
本节内容可以参考 CaseStudy 目录下的文件：条形码与 Excel 报表.Table。
该文件包括两个表，分别是 SaleOrder(订单)和 SaleOrderDetail(订单明细)，希望用 Excel 报表打印订单，其中 Lot 列以条形码的形式打印：



Universe Gems & Jewellery Co.

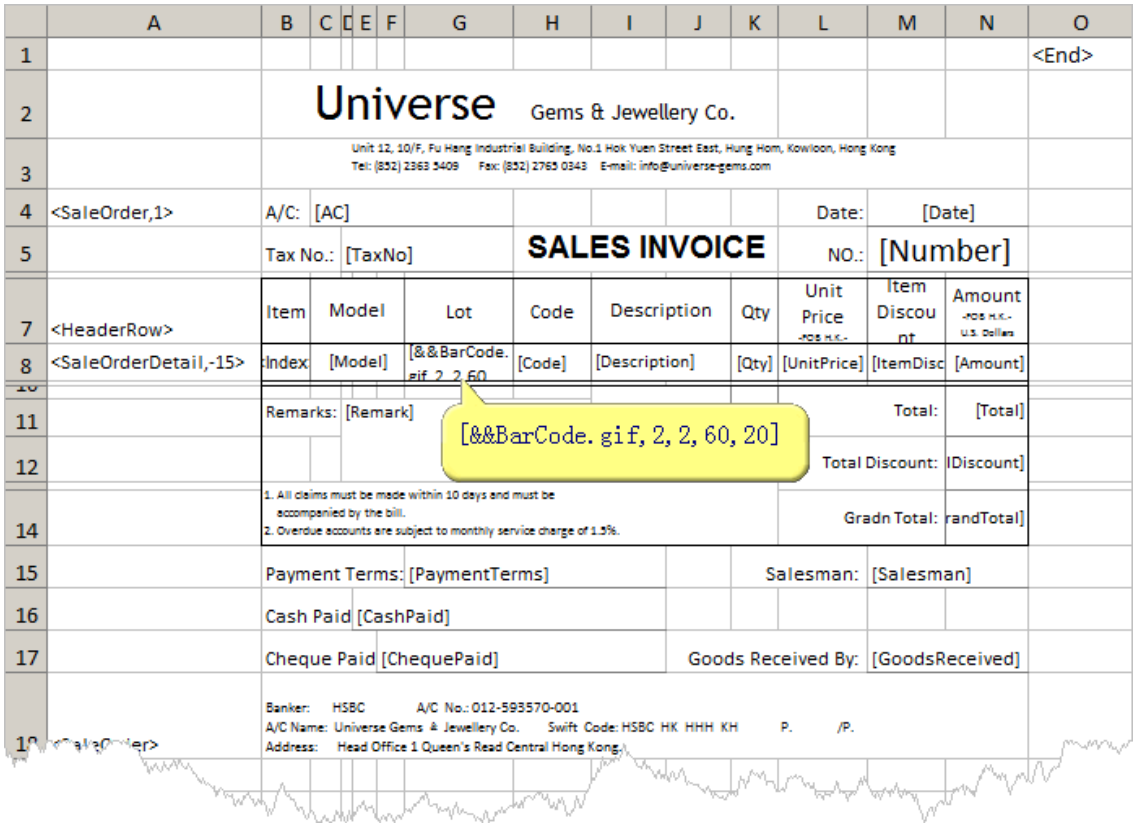
Unit 12, 10/F, Fu Hang Industrial Building, No. 1 Hok Yuen Street East, Hung Hom, Kowloon, Hong Kong
Tel: (852) 2363 3409 Fax: (852) 2765 0343 E-mail: info@universe-gems.com

A/C: dfgdfgdfgdfgdfgdg Date: 2011-07-06

Tax No.: 456456456 **SALES INVOICE** NO.: 2011

Item	Model	Lot	Code	Description	Qty	Unit Price -FOB H.K.- U.S. Dollars	Item Discount	Amount -FOB H.K.- U.S. Dollars
1	PT-45613			产品001	1	1,234.00	10%	1,110.60
2	PT-85224			产品002	1	2,288.00	10%	2,059.20
3	FR-36871			产品005	1	5,214.00	10%	4,692.60
4	NB-41245			产品007	1	9,741.00	10%	8,766.90
5	GT-52458			产品004	1	1,752.00	10%	1,576.80
6	QS-54234			产品008	1	1,547.00	10%	1,392.30

首先在设计 Excel 报表模板的时候，如下图所示将 Lot 列的内容设置为: [&&BarCode.gif,2,2,60,20]



最后将 BuildDetail 事件代码设置为:

```
If e.Book.TempLate = "SalesInvoice" Then
    If e.region = "SaleOrderDetail" Then
        If e.DataRow Is Nothing OrElse e.DataRow.IsNull("Lot") Then '如果是空行或者Lot列为空
            Dim img As New System.Drawing.Bitmap(1,1) '生成一个空白图片
            img.Save(ProjectPath & "Images\BarCode.gif")
        Else '否则生成条形码图片
            Dim Bar As New BarCodeBuilder
            Bar.Symbology = Barpro.Symbology.Code39
            Bar.Code = e.DataRow("Lot")
            Bar.BarRatio = 0.3
            Bar.BarHeight = 8
            bar.SaveImage(ProjectPath & "Images\BarCode.gif",300)
        End If
    End If
End If
```

提示：

1、上面用于生成空白图片的代码涉及Foxtable之外的知识，大家只需套用即可，无需刻意理解。
2、BarCodeBuilder会在条形码这一章专门讲述。

4.7 Word 报表

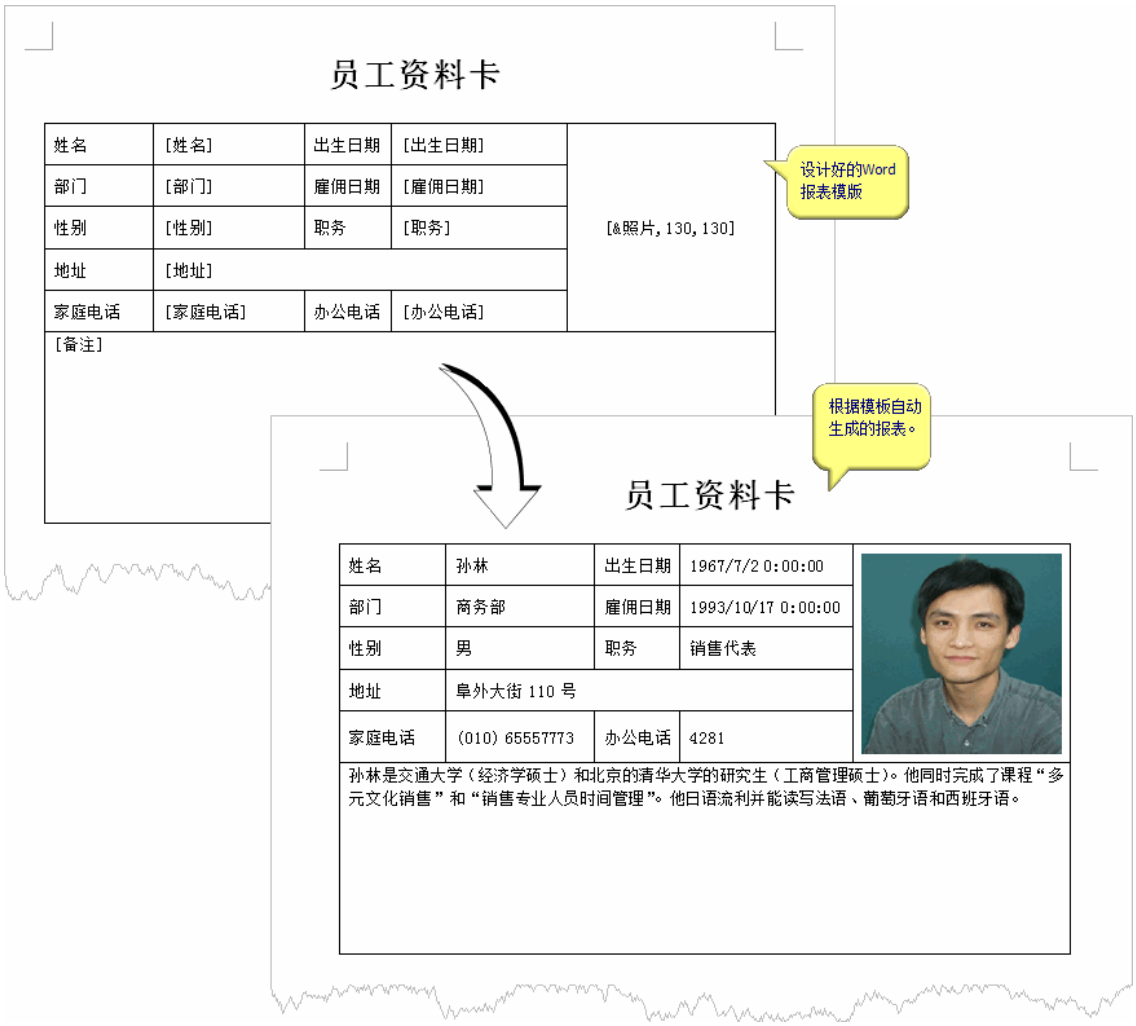
4.7.1 基础应用

4.7.1.1 关于 Word 报表

实际上，在 Foxtable 的规划之中，最初是没有 Word 报表的，报表主要通过 Excel 报表来实现。Word 作为字处理软件的王者，在复杂版面的设计方面具有无可比拟的优势和灵活性，所以 Foxtable 从 2014 版开始，提供了 Word 报表功能。

Foxtable 和 Excel 的长处是数据管理，而不是字处理，字处理的王者是微软的 Word。

有了 Word 报表，Foxtable 可以和 Word 完美地结合起来，你可以用 Word 可视化地设计一个报表模版，Foxtable 能根据这个模版的设定，自动填入数据，批量生成报表：



4.7.1.2 数据引用说明

标记	说明
[列名]	引用主表指定列数据，例如: [客户] 表示引用客户列的数据
[&列名,宽度,高度]	引用主表某列，以图片的形式显示，例如： [&照片,130,160] 表示引用照片列的内容，并以图片的形式显示，宽度是 130，高度是 160，单位为磅 如果照片列有多个图片，可以用下面的格式引用指定位置的图片： [&照片&x,100,120] x 数个数字，表示图片位置编号，0 表示第一个图片，1 表示第二个图片。
[@父表,列名]	引用父表指定列的数据，例如订单表和产品表建立了关联，产品表是父表，那么在订单表中设计模版的时候，下面的标记： [@产品,规格] 表示引用父表（产品表）的规格列的数据
[@子表,列名,行号]	引用子表指定行的指定列的数据，因为子表通常会存在多个对应行，所以要有行号参数表示引用子表第几行的数据。 例如假定出库和出库明细建立了关联，出库表是父表，在出库表设计模版的时候，下面的标记： [@出库明细,数量,2] 表示引用子表（出库明细表）第 2 行的数量列的内容。
[@&父表,列名,宽度,高度]	引用父表某列，以图片的形式显示，例如假定订单表和产品表建立了关联，产品表是父表，那么在订单表中设计模版的时候，下面的标记： [@&产品,照片,100,120] 表示引用父表（产品表）的照片列，并以图片的形式显示,宽度是 100，高度是 120，单位为磅。 如果照片列有多个图片，可以用下面的格式引用指定位置的图片： [@&产品,照片&x,100,120] x 数个数字，表示图片位置编号，0 表示第一个图片，1 表示第二个图片。
[@&子表,列名,行号,宽度,高度]	引用子表某列，以图片的形式显示，因为子表通常会存在多个对应行，所以要有行号参数表示引用子表第几行的数据。 例如假定员工表和家庭成员表建立了关联，员工表是父表，那么在员工表中设计模版的时候，下面的标记：

	[@&家庭成员,照片,1,60,80] 表示引用子表（家庭成员表）的第一行的照片列，并以图片的形式打印,宽度是 60，高度是 80，单位为磅 如果照片列有多个图片，可以用下面的格式引用指定位置的图片： [@&家庭成员,照片&x,1,60,80] x 数个数字，表示图片位置编号，0 表示第一个图片，1 表示第二个图片。
[%子表,Sum(列名)]	统计子表指定列的数据，如出库和出库明细建立了关联，出库表是父表，在出库表设计模版的时候，下面的标记： [%出库明细,Sum(数量)] 表示累计出库明细的数量列，Sum 是聚合函数，表示求和，可以使用其它聚合函数进行其他统计，参考：聚合函数
[&&图片文件,宽度,高度]	打印指定的图片文件，文件事先复制到 Images 或 Attachments 目录下，则只需指定文件名，例如： [&&Denmark.png,48,48] 否则需要包括路径，例如： [&&c:\data\Denmark.png,48,48]
<表达式>	对表达式进行计算，采用 VBScript 语法和函数，VBSCript 提供了很多函数，具体请参考：VBScript 参考 例如你要将出生日期以中文大写的形式打印出来，可以这样设置模板： <CUDATE([出生日期])>
<Index>	<Index>表示顺序号，主表每打印一行，顺序号就会增加 1

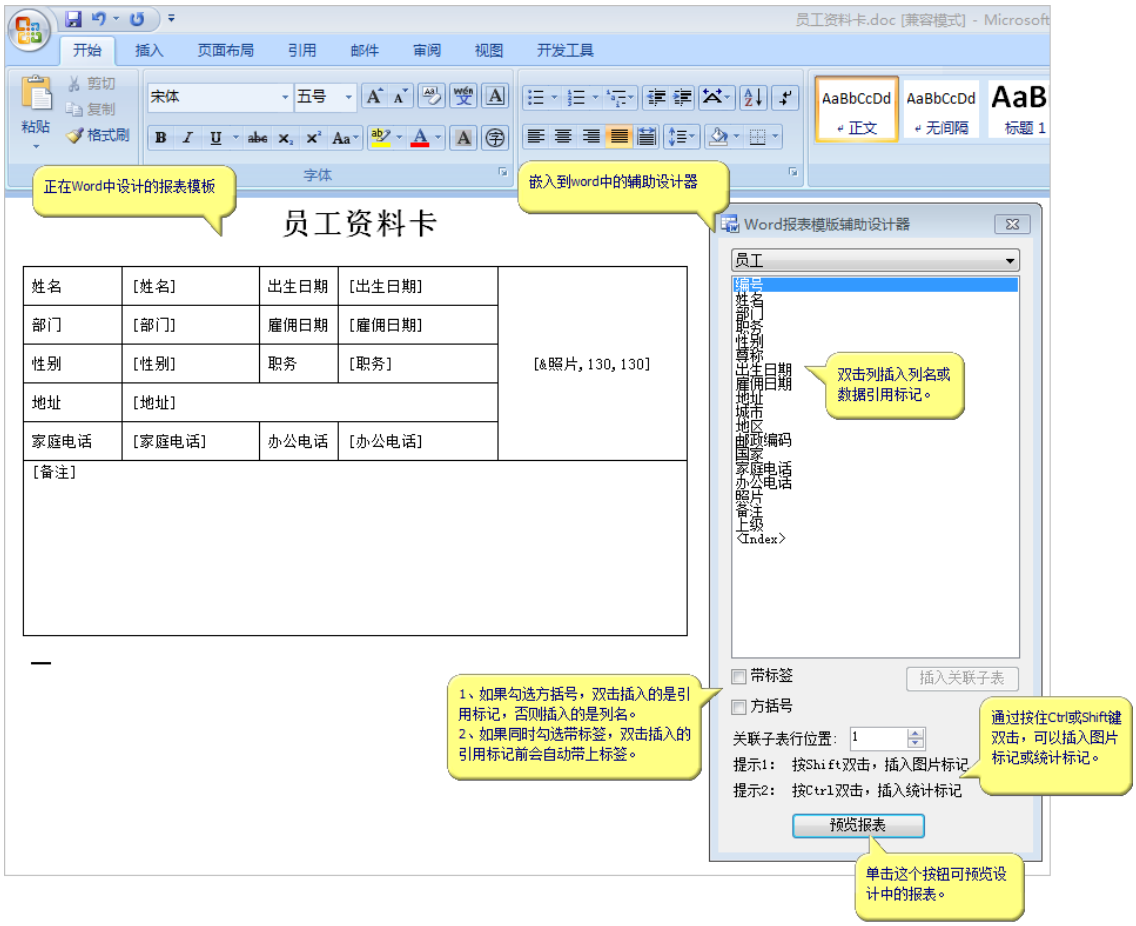
4.7.1.3 设计报表模版

在菜单的“打印输出”功能区的“Word 报表”功能组，单击“设计模版”，可以新建模版，或打开现有模版修改：



所有的模版默认保存在项目所在目录的 Attachments 子目录中，为方便管理，最好不要改变这个存储目录。

Foxtable 直接利用 Word 来设计模版，而且在 Word 中嵌入了一个辅助设计器，一切都是可视化，你无须手工输入，只需双击，即可插入列名和 各种数据引用标记，而且还可以一边设计，一边预览效果：



Word 先天具备无可比拟的设计优势，再加上 Foxtable 给 Word 嵌入的模版辅助设计器，可以说双剑合璧，如虎添翼，将大大提高我们的报表设计效率和能力 。

当然你也可以直接启动 Word 来设计报表模版，将设计好的模板文件保存在 Attachments 目录就行，只是没有了辅助设计器，设计效率会大打折扣。

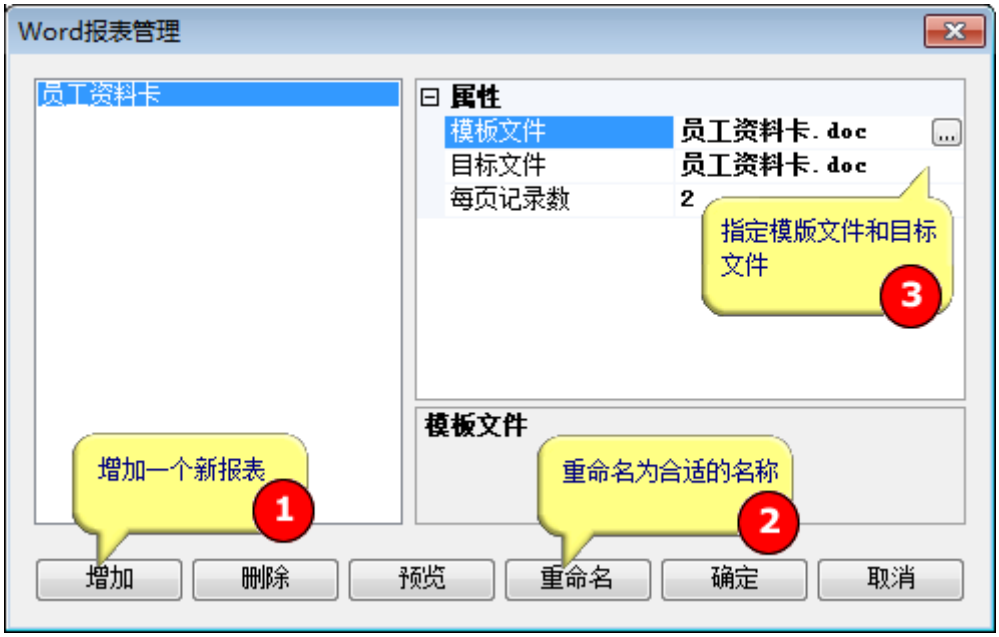
4.7.1.4 添加报表模版

设计好报表模版之后，我们需要将报表模版添加到菜单中。

在菜单的“打印输出”功能区的“Word 报表”功能组，单击“报表管理”按钮：



会出现下图所示的窗口，通过这个窗口，可以添加已经设计好的报表模版：



目标文件指的是根据模板文件生成的报表文件的文件名。

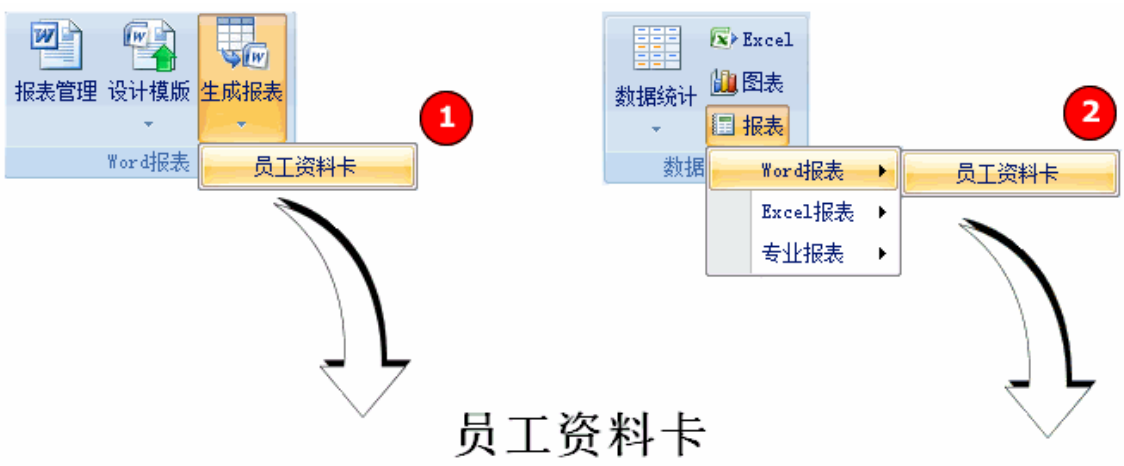
4.7.1.5 生成 Word 报表

设计好 Word 报表模版，并将模版添加到菜单后，就可以通过菜单来生成 Word 报表了。

菜单中有两处地方可以生成 word 报表：

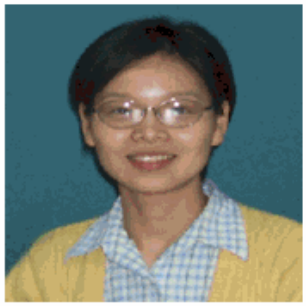
1、在菜单的“打印输出”功能区的“Word 报表”功能组，单击“生成报表”按钮 ，会列出已经添加的模版，单击其中一个，即可生成报表。

2、在菜单的“日常工作”功能区的“数据统计”功能组，单击“报表”按钮，指向“Word 报表”,也会列出已经添加的模版，单击其中一个，即可生成报表。



员工资料卡

姓名	张颖	出生日期	1968/12/8 0:00:00
部门	商务部	雇佣日期	1992/5/1 0:00:00
性别	女	职务	销售代表
地址	复兴门 245 号		
家庭电话	(010) 65559857	办公电话	5467

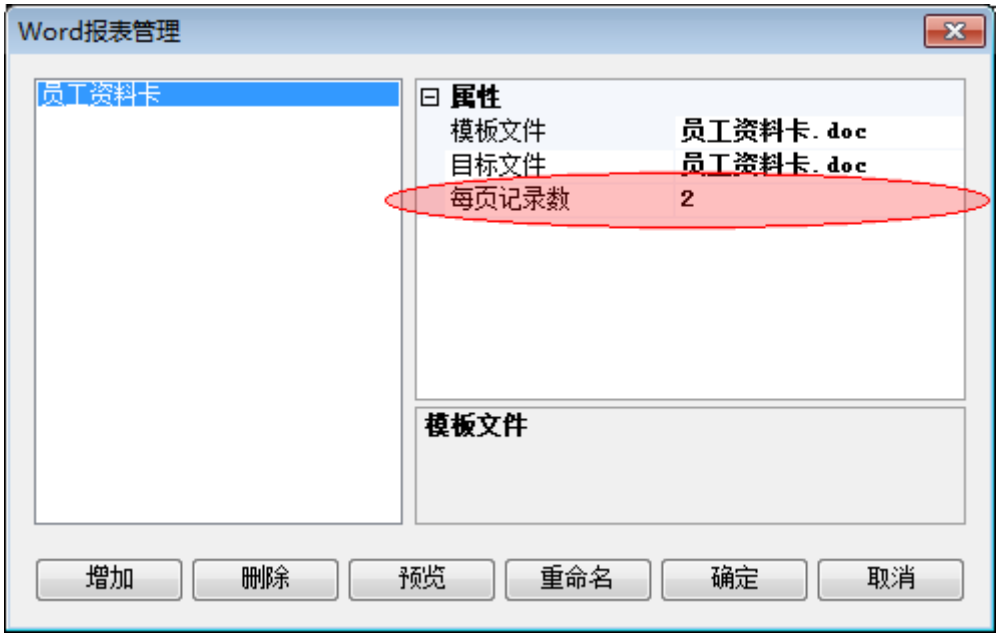


获北京大学心理学学士学位。她同时完成了“冷食的艺术”。张颖是国际美食协会的会员。

系统会自动打开生成的报表文件，这些文件默认存放在目录目录的 Reports 子目录下。

4.7.1.6 自动换页控制

在添加模板的时候，我们可以指定每页打印的记录数，例如下图意味着每页打印两个员工的资料卡：



这样生成报表的时候，每打印两个员工的资料卡，就会另起一页。
如果我们不指定每页打印的记录数（参数设为 0），由系统自动换页，那么某员工的资料卡可能跨越两页打印，显然这不是我们所希望的。

4.7.1.7 Word 文章套打

用 Word 报表，可以轻松实现文章套打，例如我们可以用 Word 设计一个下图所示的模版：

尊敬的[客户]：

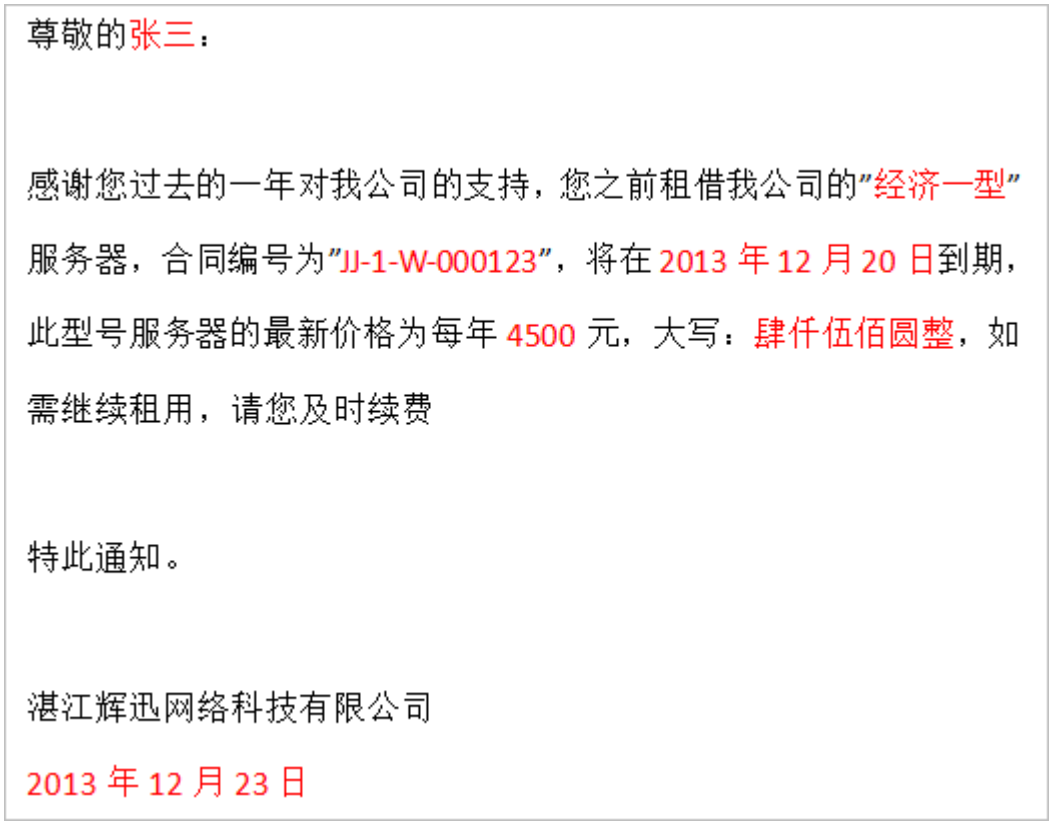
感谢您过去的一年对我公司的支持，您之前租借我公司的"[型号]"服务器，合同编号为"[合同编号]"，将在<Format([到期日], "yyyy 年 MM 月 dd 日")>到期，此型号服务器的最新价格为每年[价格]元，大写：<CUMoney([价格])>，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

<Format(Date, "yyyy 年 MM 月 dd 日")>

上图的模版中，我们用红色标出了数据引用标记，有直接引用，例如：[合同编号]，也有表达式，例如：<CUMoney([价格])>。
当我们根据这个模板生成报表时，Foxtable 会自动填入数据并计算表达式，得到正式的文章：


尊敬的张三：

感谢您过去的一年对我公司的支持，您之前租借我公司的"经济一型"服务器，合同编号为"JJ-1-W-000123"，将在 2013 年 12 月 20 日到期，此型号服务器的最新价格为每年 4500 元，大写：肆仟伍佰圆整，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

2013 年 12 月 23 日

4.7.1.8 打印关联子表

父表一行，在子表通常会存在多个对应行，就象一个父亲会有多个儿子一样，所以我们引用关联子表的时候，要有行号参数表示引用子表第几行的数据。

引用子表的标记格式为：

[@子表,列名,行号]
例如假定出库和出库明细建立了关联，出库表是父表，在出库表设计模版的时候，下面的标记：

[@出库明细,数量,5]

表示引用子表（出库明细表）第 5 行的数量列的内容，如果子表不够 5 行，则生成报表时此标记不会显示任何内容。
如果要统计子表，标记格式为：
[%子表,Sum(列名)]
例如:
[%出库明细,Sum(数量)]
表示累计出库明细子表的数量列。

如果 我们在出库单表设计报表时，按下图所示设计模板：

湛江辉迅公司出库单

出库单号：[出库单编号]					
领用部门：[领用部门]		出库日期：[出库日期]		领料人：[领料人]	
商品代码	商品名称	规格	单位	数量	备注
[@出库明细,商品代码,1]	[@出库明细,商品名称,1]	[@出库明细,规格,1]	[@出库明细,单位,1]	[@出库明细,数量,1]	[@出库明细,备注,1]
[@出库明细,商品代码,2]	[@出库明细,商品名称,2]	[@出库明细,规格,2]	[@出库明细,单位,2]	[@出库明细,数量,2]	[@出库明细,备注,2]
[@出库明细,商品代码,3]	[@出库明细,商品名称,3]	[@出库明细,规格,3]	[@出库明细,单位,3]	[@出库明细,数量,3]	[@出库明细,备注,3]
[@出库明细,商品代码,4]	[@出库明细,商品名称,4]	[@出库明细,规格,4]	[@出库明细,单位,4]	[@出库明细,数量,4]	[@出库明细,备注,4]
[@出库明细,商品代码,5]	[@出库明细,商品名称,5]	[@出库明细,规格,5]	[@出库明细,单位,5]	[@出库明细,数量,5]	[@出库明细,备注,5]
库管员：[库管员]			合计：	[%出库明细,Sum(数量)]	

可以生成下图所示的出库单：

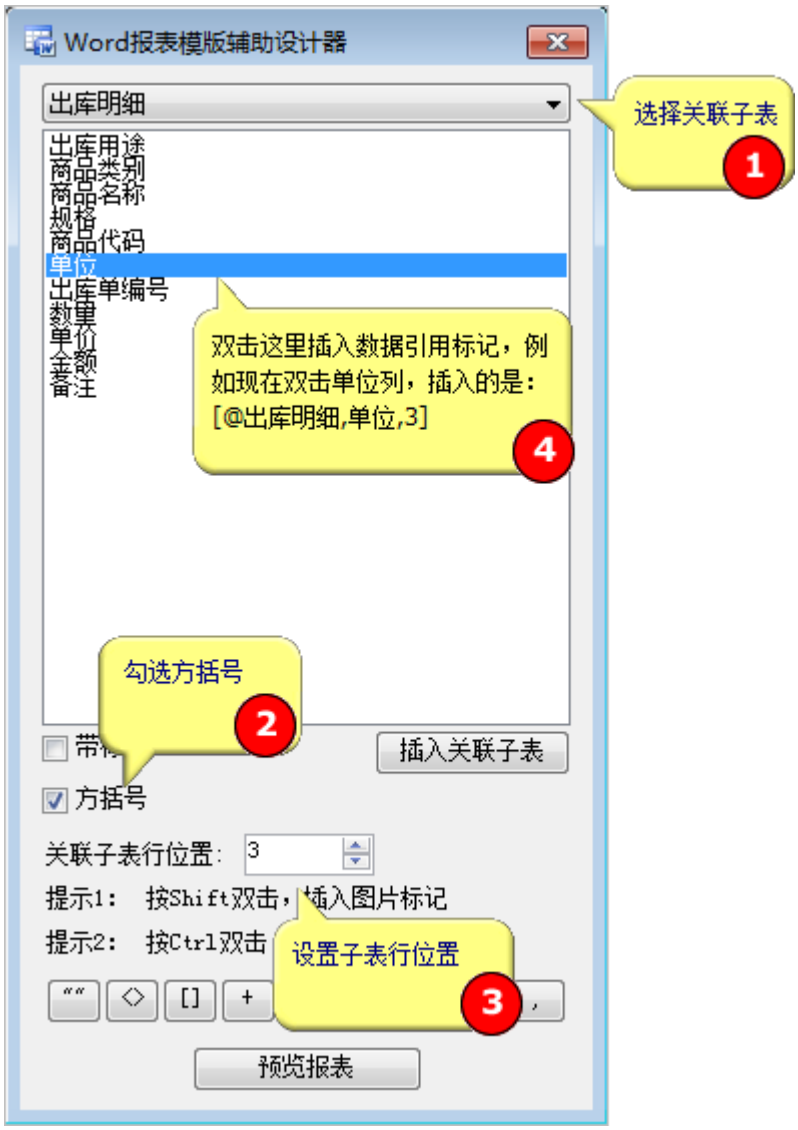
湛江辉迅公司出库单

出库单号：CK-20030726002
领用部门：商函广告公司　　出库日期：2008/7/26 0:00:00　　领料人：朱志欣

商品代码	商品名称	规格	单位	数量	备注
BG00201	记事本	16 开精装	本	8	
YW00101	特快详情单		张	1000	

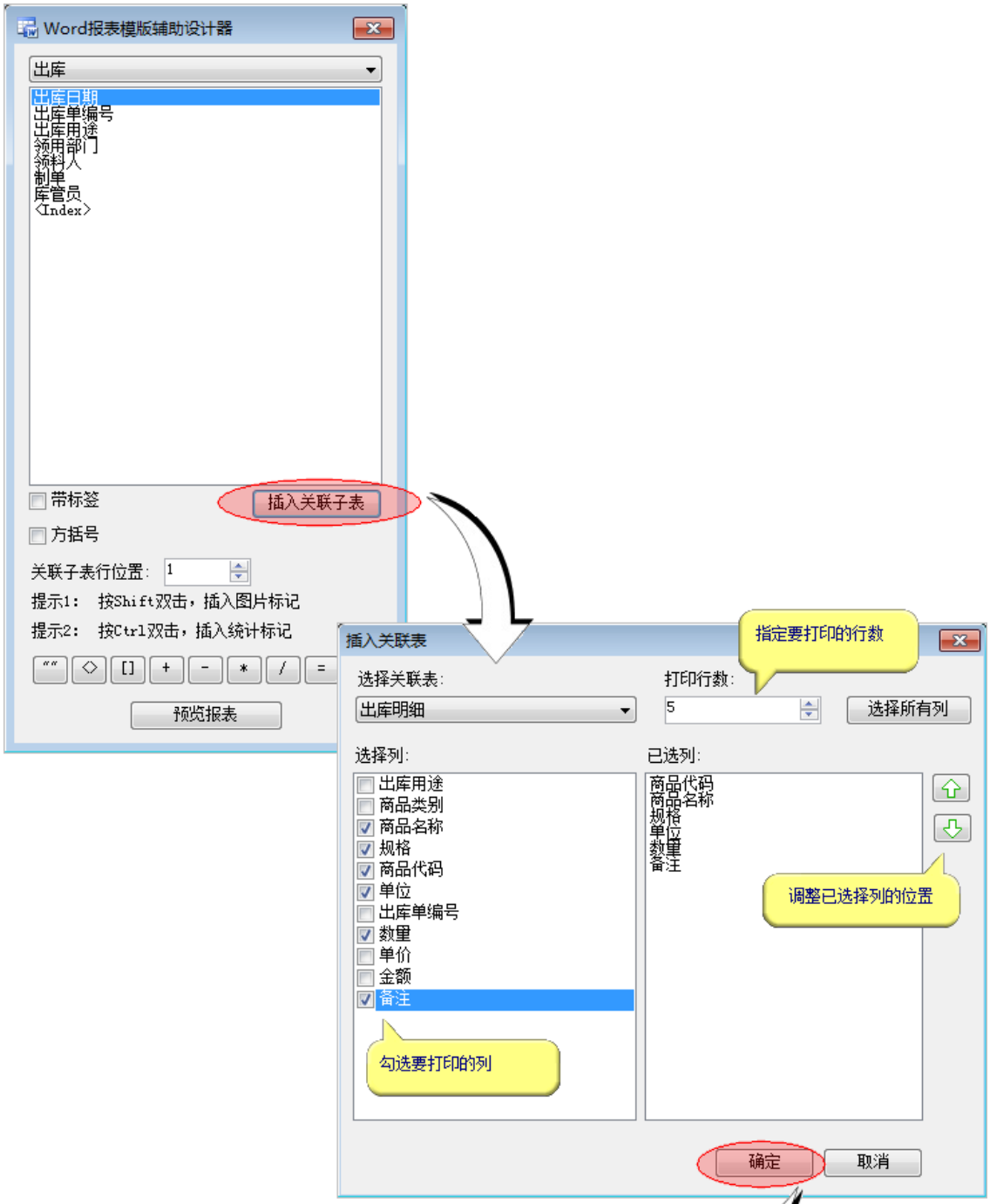
库管员：李政旭　　合计：　1008

在设计模板的时候，可以按照下面的提示快速插入关联子表的数据引用标记：



4.7.1.9 插入关联子表

在打印关联子表的时候，我们并不需要逐个单元格去插入引用标记，模板设计辅助器提供一个插入按钮，可以一次整个插入关联子表，轻松无比：



商品代码	商品名称	规格	单位	数量	备注
[@出库明细,商品代码,1]	[@出库明细,商品名称,1]	[@出库明细,规格,1]	[@出库明细,单位,1]	[@出库明细,数量,1]	[@出库明细,备注,1]
[@出库明细,商品代码,2]	[@出库明细,商品名称,2]	[@出库明细,规格,2]	[@出库明细,单位,2]	[@出库明细,数量,2]	[@出库明细,备注,2]
[@出库明细,商品代码,3]	[@出库明细,商品名称,3]	[@出库明细,规格,3]	[@出库明细,单位,3]	[@出库明细,数量,3]	[@出库明细,备注,3]
[@出库明细,商品代码,4]	[@出库明细,商品名称,4]	[@出库明细,规格,4]	[@出库明细,单位,4]	[@出库明细,数量,4]	[@出库明细,备注,4]
[@出库明细,商品代码,5]	[@出库明细,商品名称,5]	[@出库明细,规格,5]	[@出库明细,单位,5]	[@出库明细,数量,5]	[@出库明细,备注,5]

在Word中插入的关联子表

4.7.1.10 常用函数介绍

我们已经知道在设计 Word 模板的时候，可以插入表达式，格式为：

<表达式>
需要注意的是，表达式中的列名，必须用方括号括起来，例如：
<Round([数量] * [单价],2)>
表示数量乘以单价，并保留两位小数，如数量和单价不用方括号括起来，此表达式将不能正确计算。
表达式采用 VBScript 语法，而且 Foxtable 对 VBScript 进行了扩展，提供了很多非常实用的函数。
例如你要将出生日期以中文大写的形式打印出来，可以这样设置模板：<CUDATE([出生日期])>
再例如，假定已付款是个逻辑列，希望打印的时候，已付款的用符号"√"，可以这样设置模板：IIF([已付款], "√", "")
我们下面介绍一些常用的函数，更多的函数和更具体的用法，请参考：VBSCript 参考

函数名	说明
Format	格式化日期或数字。 格式化数字的时候，用 0 表示占位符，如果指定位置有数字，则显示数字，如果没有数字，则用 0 补充。 例如需要金额保留两位小数显示： <Format([金额],"#0.00")> 如果金额的值是 123.567，则显示为 123.57，如果金额的值是 123.5，则显示为 123.50，而不是 123.5。 如果希望以百分比的形式显示，可以加上符号%，例如希望以百分比形式显示合格率，保留一位小数： <Format([合格率],"#0.0%")> 格式化日期的时候，yyyy 表示四位年，yy 表示两位年，MM 表两 2 位月,M 表示一位月，dd 表示两位日,d 表示一位日，例如： <Format([出库日期],"yyyy 年 MM 月 dd 日")> 显示的日期格式为：2013 年 09 月 08 日 <Format([出库日期],"yyyy 年 M 月 d 日")> 显示的日期格式为：2013 年 9 月 8 日 <Format([出库日期],"yy 年 M 月 d 日")> 显示的日期格式为：13 年 9 月 8 日 注意 M 是大写。
IIF	语法为： IIF(expression, truevalue, falsevalue) 如果逻辑表达式 expression 的计算结果为 True，将返回 truevalue，否则返回 falsevalue。 例如：

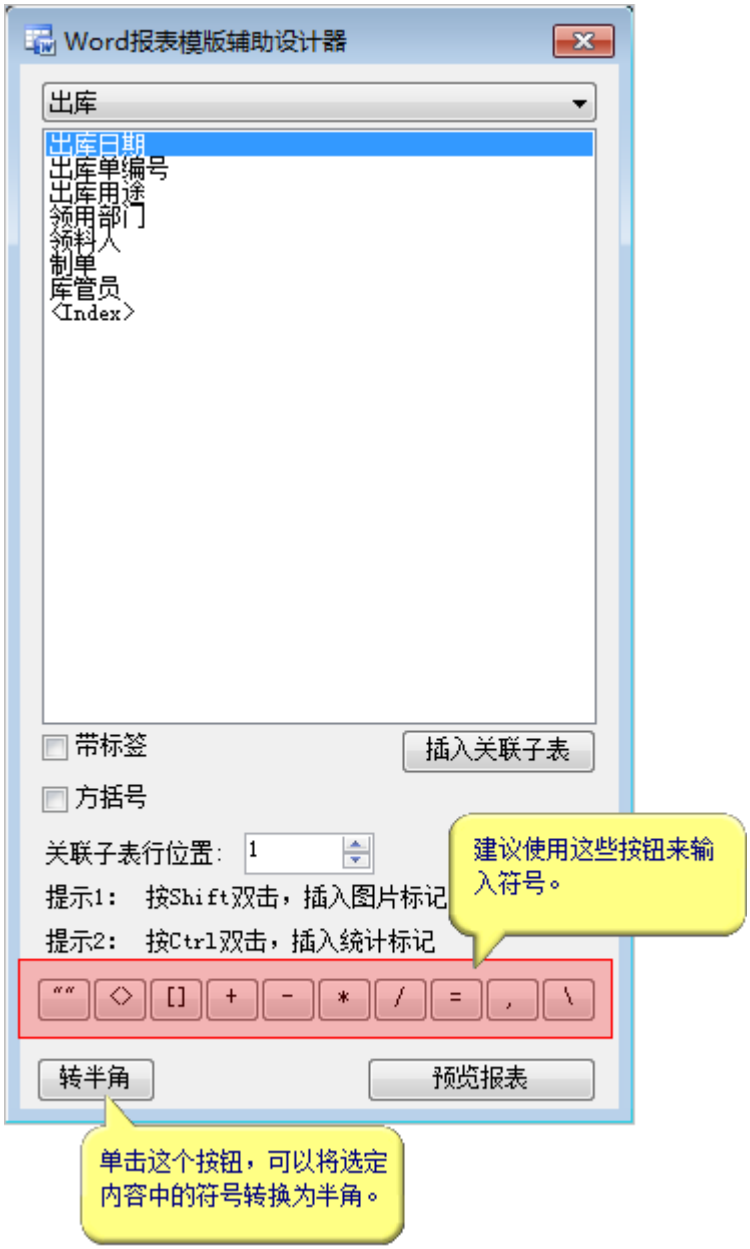
	IIF([已付款], " √ ", "")
Round	对数值进行四舍五入，例如：<Round([金额],2)>
Date	返回今天的日期，例如： <Format(Date,"yyyy 年 MM 月 dd 日")> 表示以四位年两位月两位日的格式显示今天的日期。 日期可以直接和数字进行加减，例如： <Date - 2> 表示前天的日期 <Date + 1> 表示明天的日期 <Format(Date + 1,"yyyy 年 MM 月 dd 日")> 表示以四位年两位月两位日的格式显示明天的日期。
Now	放回当前时间，例如：<Now>
Year	返回日期的年份，例如: <Year(出生日期)>
Month	返回日期的月份，例如: <Month(出生日期)>
Day	返回日期的日部分，例如: <Day(出生日期)>
Weekday	用数字形式返回星期几，例如: <WeekDay(出生日期)>
WeekDayName	用名称形式返回星期几，例如: <WeekDayName(出生日期)>
CUDate	将日期 Date 转换为大写中文格式。 例如： <CUDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：贰零壹叁年拾贰月玖日
CLDate	将日期 Date 转换为小写中文格式。 例如： <CLDate([日期])> 假定日期列的值是“2013-12-9”，上面的表达式显示的是：二〇一三年十二月九日
CUMoney	将数值转换为中文货币格式。 例如： <CUMoney([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍圆整
CUNumber	将数值转换为中文大写格式。 例如： <CUNumber([金额])> 假定金额的值是 12345，上面的表达式显示的是：壹万贰仟叁佰肆拾伍
EUNumber	将整数转换为英文大写。 例如：

	<EUNumber([数量])> 假定数量的值是 12345，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five
EUMoney	将数值转换为英文货币格式。 例如： <EUMoney([金额])> 假定金额的值是 12345.45，上面的表达式显示的是：Twelve Thousand Three Hundred Forty Five Dollars And Forty Five Cents

4.7.1.11 符号要半角的

在设计 Word 模板的时候，我们需要插入很多数据引用标记和表达式。

需要注意的是，标记中使用的符号必须是半角的，不能是全角的。
例如下面的表达式不会正确计算：
〈CLNum([金额])〉
正确的写法是：
<CLNum([金额])>
二者的差别在于前者首尾的尖括号时全角的，后者则是半角的。
使用汉字的我们，很难察觉这种错误，也很难避免这种错误，设计模板的一定要小心。
为了减少这种错误，Word 模板辅助设计器后来加上了两个按钮，分别用于快速输入常用的半角符号 ，以及将选定内容中的符号转换为半角：



4.7.1.12 报表与表的关系

显然报表的数据来自于表，所以每个报表是属于特定表的。
例如我们之前设计的"员工资料卡.doc"这个模版，生成报表的时候，会从员工表提取数据，所以这个报表是属于员工表的。

但是在设计 Word 报表模版的时候，并没有办法指定这个报表属于那个表；我们只能在设计“员工资料卡.doc”这个模板的时候，首先选择员工表，这样模板辅助设计器会自动列出员工表的所有列；同样将这个设计好的模板添加到菜单中，或者基于这个模板生成报表的时候，也必须确保当前表为员工表。
Foxtable 自动认为所有和 Word 报表相关的操作，都是基于当前表的。

4.7.1.13 表达式中的运算符冲突

在 Word 报表模版中，"<"和">"有特殊意义，分别用于标记表达式 的开始和结束，例如：
<CUDATE([出生日期])>
但是在表达式中，"<"和">"本身也是一个运算符，分别表示大于和小于，当然还有表示不等于的"<>"。
为了避免冲突，Foxtable 要求：
1、作为运算符的"<"、">"和"<>"，前后必须有空格。
2、用于表达式开始标记的"<"，后面不能有空格；作为表达式结束标记的">"，前面不能有空格。
下面列出一些表达式，有的正确，有的错误，请大家体会：

表达式	说明
<IIF([成绩] > 60, "及格","不及格")>	正确的表达式
<IIF([出生日期] <> #0:00#, CUDATE([出生日期]), "")>	正确的表达式
<IIF([成绩] >60, "及格","不及格")>	错误的表达式，运算符>的后面没有空格。
<IIF([成绩]> 60, "及格","不及格")>	错误的表达式，运算符>的前面没有空格。
<IIF([等级]<>"A", "录取","淘汰")>	错误表达式，运算符<>的前后都没有空格。
< IIF([成绩]> 60, "及格","不及格")>	错误表达式，开始标记<的后面有空格。
<IIF([成绩]> 60, "及格","不及格") >	错误表达式，结束标记>的前面有空格

需要特别注意的是，一个表达式编写错误，可能会导致后续其它表达式也无法正确计算。
例如假定模版中有连续的两个表达式：

是否及格：<IIF([成绩] > 60, "及格","不及格") >
是否录取：<IIF([等级] <> "A", "录取","淘汰")>

看起来只是第一个表达式有错，实际上两个表达式都无法计算，因为系统在分析模版的时候，由于第一个表达式的结尾标记">"的前面有空格，所以会继续往后查找，一直找到第一个符合要求的结尾标记(也就是第二个表达式的结尾标记)，这样生成的时候，Foxtable 会将两个表达式合并为一个，当然这个合并后的表达式完全不符合语法要求，自然也就无法计算 ，最终的结果就是两个表达式都没有结算，尽管其中一个表达式并没有错误。

所以在分析表达式错误的时候，应该从第一个没有正确计算的表达式开始分析，逐个往后处理。

4.7.1.14 Word 报表还是 Excel 报表

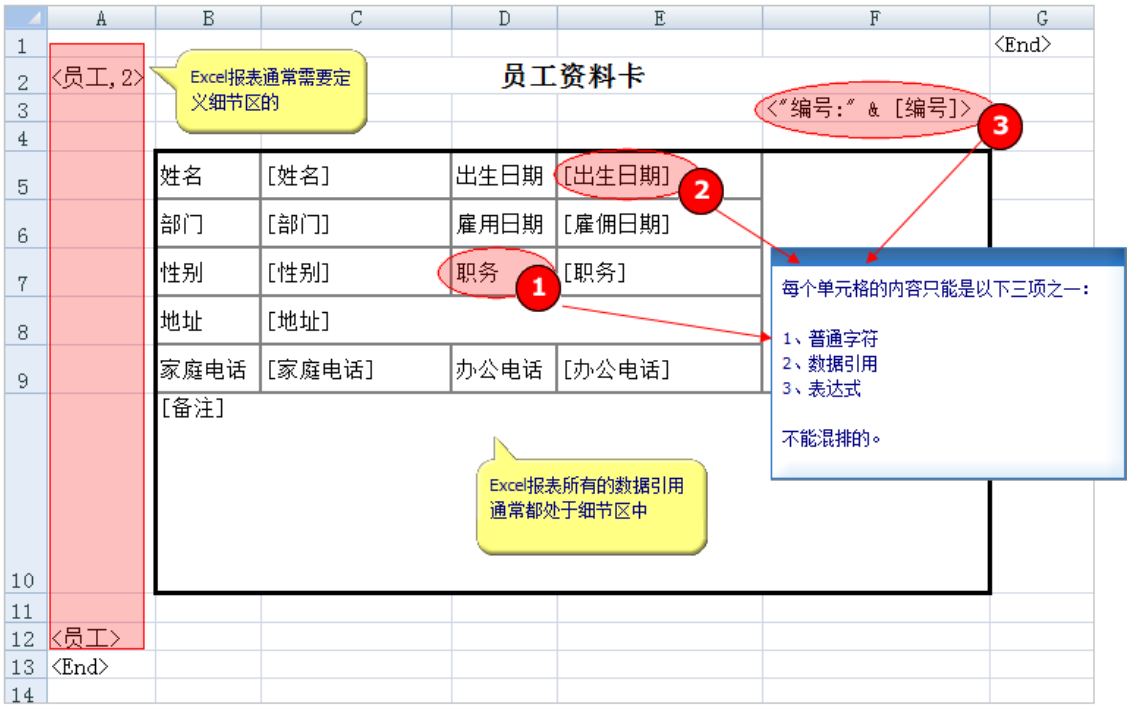
根据 Word 报表和 Excel 报表在帮助中所占的篇幅可知，Excel 报表是 Foxtable 的重点，确实 Excel 报表能给我们的， 要比 Word 报表多一些 。

但是 Word 报表也有自己的优势：

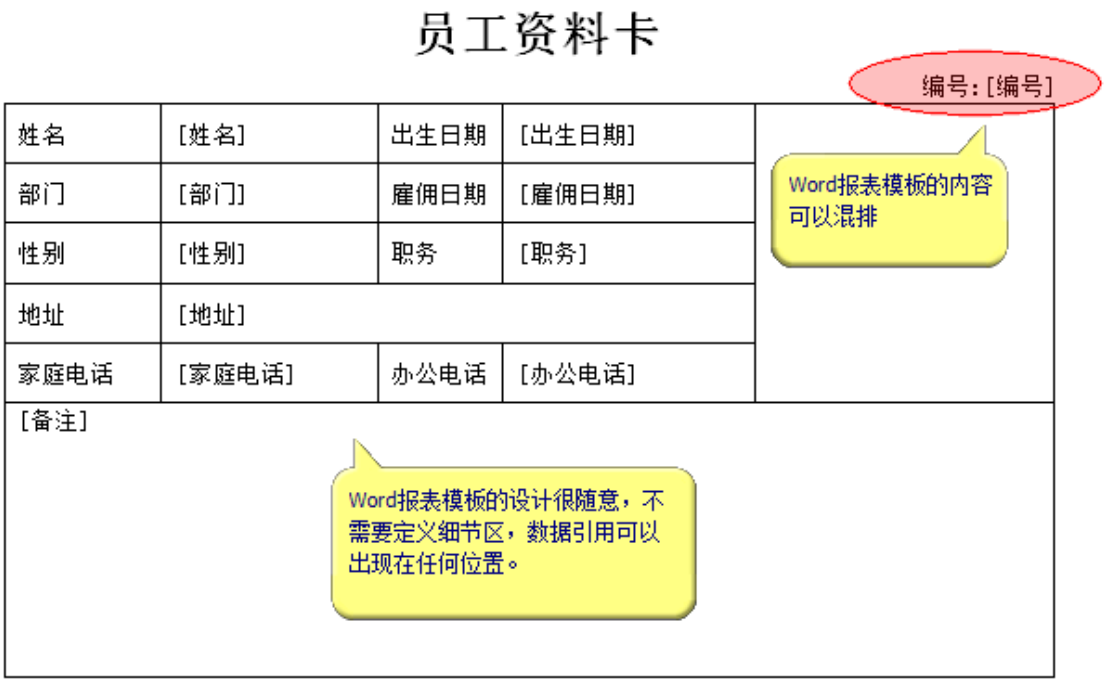
1、Word 在排版方面更具优势，更便于设计版面复杂的报表。

2、Word 报表在数据引用方面更随意。

Excel 报表在数据引用方面是比较严谨的，通常所有的数据引用标记都必须位于细节区中，而且每个单元格内容，只能是普通字符、数据引用或表达式三者中的一个，不能混排：



而 Word 报表在设计模板的时候，则“随意”很多，不需要定义细节区，单元格可以将多种内容混排在一起：



因为这种随意的特性，所以用 Word 模板可以很方便地实现文章套打，例如设计一个下图的模板：

尊敬的[客户]：

感谢您过去的一年对我公司的支持，您之前租借我公司的"[型号]"服务器，合同编号为"[合同编号]"，将在<Format([到期日], "yyyy 年 MM 月 dd 日")>到期，此型号服务器的最新价格为每年[价格]元，大写：<CUMoney([价格])>，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

<Format(Date, "yyyy 年 MM 月 dd 日")>

可以自动生成如下的文章：

尊敬的张三：

感谢您过去的一年对我公司的支持，您之前租借我公司的"经济一型"服务器，合同编号为"JJ-1-W-000123"，将在 2013 年 12 月 20 日到期，此型号服务器的最新价格为每年 4500 元，大写：肆仟伍佰圆整，如需继续租用，请您及时续费

特此通知。

湛江辉迅网络科技有限公司

2013 年 12 月 23 日

总结：

word 报表和 Excel 报表各有优势，大家可以根据需要选择；一般来说，根据单条记录生成一个格

式较为复杂但相对固定的表格（包括文章），或者虽然要打印关联子表，但是子表行数有限，可以在报表中 预留固定行数显示子表，此时 Word 报表较有优势，其他场合 Excel 报表优势明显。

4.7.2 编程指南

4.7.2.1 WordReport

WordReport 用于编程生成 Word 报表。
定义一个 WordReport 的语法为：
Dim wrt As New WordReport(Table, TemplateFile, ReportFile)
Table： 要打印的表(Table)或表的名称
TemplateFile：字符型，模板文件名称，含路径
ReportFile：字符型，生成的报表文件名称，含路径
WordReport 的成员有：

名称	说明
Build	方法，生成报表。
BuildOne	方法，根据指定的一行数据生成报表，语法： BuildOne(dr) dr: DataRow 或 Row 类型，根据此行数据生成报表。
Replace	方法，将指定的内容替换为新的内容，语法： Repalce(findstr,replacestr) findstr: 字符型，指定要查找的内容 replacestr: 字符型，指定用于替换的内容。 Replace 是全局有效。
ReplaceOne	方法，将指定的内容替换为新的内容，语法： RepalceOne(findstr,replacestr) findstr: 字符型，指定要查找的内容 replacestr: 字符型，指定用于替换的内容。 ReplaceOne 只对下一条记录生效，需要和 BuildOne 配合使用。
ReplaceWithImage	方法，将指定的内容替换为图片，语法： ReplaceWithImage(findstr,imagefile,width,height) findstr: 字符型，指定要查找的内容 imgefile：字符型，用于替换的图片文件

	width： 整数型，指定图片宽度 height：整数型，指定图片高度
Quit	方法，退出报表
Show	方法，启动 Word，显示生成的报表。
SaveToPdf	保存工作簿为 PDF 文件，语法为： SaveToPdf(File) File: 文件名称，含路径。
RecordsOnEachPage	属性，整数型，指定每页打印的记录数。

重要提示：

用 WordReport 生成报表之后，必须执行 Quit 方法退出，或者执行 Show 方法显示报表，否则内存将常驻一个 Word 进程， 影响后续报表的生成和设计，直到你中断这个进程，切记切记。

示例一

常规的根据模板生成报表：

```
Dim tm As String  = ProjectPath & "Attachments\出库单.doc" '指定模板文件
Dim fl As String = ProjectPath & "Reports\出库单.doc" '指定目标文件
Dim wrt As New WordReport(Tables("出库"),tm,fl) '定义一个WordReport
wrt.Build() '逐行生成报表
wrt.Show() '显示报表
```

提示，Build方法根据当前选定行生成报表。

示例二

利用 BuildOne 方法，可有根据指定条件的行来生成报表，例如：

```
Dim drs As List(of DataRow) = DataTables("出库").Select("出库日期 = #" & Date.Today & "#") '筛选出符合条件的行
If drs.Count > 0 Then '如果存在符合条件的行
    Dim tm As String  = ProjectPath & "Attachments\出库单.doc" '指定模板文件
    Dim fl As String = ProjectPath & "Reports\出库单.doc" '指定目标文件
    Dim wrt As New WordReport(Tables("出库"),tm,fl) '定义一个WordReport
    For Each dr As DataRow In drs '逐行生成报表
        wrt.BuildOne(dr)
    Next
    wrt.Show() '显示报表
End If
```

再例如根据已经勾选的行生成报表：

```
Dim rs As List(of Row) = Tables("出库").GetCheckedRows
If rs.Count > 0 Then '如果存在符合条件的行
```

```
    Dim tm As String   = ProjectPath & "Attachments\出库单.doc" '指定模板文件
    Dim fl As String = ProjectPath & "Reports\出库单.doc" '指定目标文件
    Dim wrt As New WordReport(Tables("出库"),tm,fl) '定义一个WordReport
    For Each r As Row In rs '逐行生成报表
      wrt.BuildOne(r)
    Next
    wrt.Show() '显示报表
End If
```

实例三

你可以在生成模板前，先用Repalce修改模板，这样我们可以用代码完成一些Word报表无法直接实现的统计，然后将得到的统计结果替换约定的标记，例如：

```
Dim tm As String   = ProjectPath & "Attachments\订单.doc" '指定模板文件
Dim fl As String = ProjectPath & "Reports\订单.doc" '指定目标文件
Dim wrt As New WordReport(Tables("订单"),tm,fl) '定义一个WordReport
wrt.Replace("[合计]",DataTables("订单").Compute("Sum(数量)")) '用代码计算出总的销售数量并替换模板中的合计标记
wrt.Build() '生成报表
wrt.Show() '显示报表
```

示例四

Replace 对模板的修改是全局的。
我们也可以用 ReplaceOne 修改模板，和 Repalce 不同，RepalceOne 仅对下一条记录生效，需要和 BuildOne 配合使用，例如：

```
Dim tm As String   = ProjectPath & "Attachments\订单.doc" '指定模板文件
Dim fl As String = ProjectPath & "Reports\订单.doc" '指定目标文件
Dim wrt As New WordReport(Tables("订单"),tm,fl) '定义一个WordReport
Dim sum As Integer
For Each r As Row In Tables("订单").rows '逐行生成报表
    sum = sum + r("数量") '逐行累计
    wrt.ReplaceOne("[合计]",sum)
    wrt.BuildOne(r)
Next
wrt.Show() '显示报表
```

这样生成报表的时候，每一条记录的合计，都是截止到此行为止的合计。

实例五

假定我们要给员工资料卡加上二维码，可以在设计模板的时候，引用一个约定的图片文件：



然后用下面的代码生成报表，在生成每个员工的资料卡之前，都先根据这个员工的资料合成一个二维码，并保存为约定的图片文件：

```
Dim tm As String   = ProjectPath & "Attachments\员工资料卡.doc" '指定模板文件
Dim fl As String = ProjectPath & "Reports\员工资料卡.doc" '指定目标文件
Dim mg As String = ProjectPath & "Images\二维码.wmf" '约定的图片文件
Dim wrt As New WordReport(Tables("员工"),tm,fl) '定义一个WordReport
For Each r As Row In Tables("员工").Rows '逐行生成报表
    Dim Bar As New BarCodeBuilder
    Bar.Symbology = Barpro.Symbology.QRCode
    Bar.QRCodeModuleSize = 2 '二维码的放大倍数
    Bar.Code = "姓名:" & r("姓名") & vbcrlf & "部门:" & r("部门") & vbcrlf & "电话:" & r("办公电话")
    bar.SaveImage(mg,300)
    wrt.BuildOne(r)
Next
wrt.Show() '显示报表
```

这是生成的报表，有兴趣的话，你可以扫描一下这个二维码：

员工资料卡

编号:1

姓名	王伟	出生日期	1962/2/19 0:00:00	
部门	商务部	雇佣日期	1992/8/14 0:00:00	
性别	男	职务	副总裁(销售)	
地址	罗马花园 890 号			
家庭电话	(010) 65559482	办公电话	3457	

王伟获南京大学商业学士学位，获该校国际营销博士学位。他能说流利的法语和意大利语并能阅读德语。他加入公司时是销售代表，被提拔为销售经理并升任销售副总裁。王伟是销售管理圆桌协会，北京商业总会和太平洋周边进口协会的成员。

总结：Word 报表有 BuildOne 和 RepalceOne 方法，可以逐行生成报表，可以逐行修改模板，而且用 BuildOne 生成报表的时候，行可以来自不同的表，只要模板中引用的列存在即可，非常灵活。

4.7.2.2 保存为 PDF 文件

提示：此功能需要安装 Office 2007 或更高的版本。
为了避免用户修改生成好的 Word 报表，我们可以将报表保存为 PDF 文件再打开，例如：

```
Dim tm As String   = ProjectPath & "Attachments\出库单.doc" '指定模板文件
Dim fl1 As String = ProjectPath & "Reports\出库单.doc" '指定目标文件
Dim fl2 As String = ProjectPath & "Reports\出库单.pdf" '指定目标PDF文件
Dim wrt As New WordReport(Tables("出库"),tm,fl1) '定义一个WordReport
wrt.Build() '逐行生成报表
wrt.SaveToPDF(fl2) '保存为PDF文件
wrt.Quit() '退出
Dim Proc As New Process '打开PDF文件
Proc.File = fl2
Proc.Start()
```

提示：为了能打开生成的 PDF 文件，请先安装 PDF 阅读软件，例如：Adobe Reader X

4.7.2.3 在窗口显示生成的 Word 报表

如果还没有接触过窗口设计，可以暂时忽略本节的内容。
在默认情况下，Foxtable 会直接启动 Word 来显示生成的 Word 报表。
借助 WebBrowser 控件，你可以在窗口中显示、编辑和打印生成的 Word 报表。
示例一
新建一个窗口（建议使用模式窗口或独立窗口），在窗口中加入一个 WebBrowser 控件，窗口的 AfterLoad 事件代码设置为：

```
Dim wbr As WinForm.WebBrowser = e.Form.Controls("WebBrowser1")
Dim tm As String  = ProjectPath & "Attachments\出库单.doc"
Dim fl As String = ProjectPath & "Reports\出库单.doc"
Dim wrt As New WordReport(Tables("出库"),tm,fl)
wrt.Build()
wrt.Quit()
wbr.AddRess = fl
```

这样一打开窗口，就会自动在窗口的 WebBrowser 控件显示生成的 Word 报表。
重要提示：
1、如果执行上面代码的时候，没有在 WebBrowser 显示 Word 报表，而是在一个新的 IE 窗口显示，那么请双击 Foxtable 目录下的 Office.reg 文件，将其添加到注册表中。
2、在 WebBrowser 显示的 Word 菜单缺少包括打印命令的程序菜单，你可以按快捷键“Ctrl+P”打印，或者自定义 word 的快速访问栏，把打印命令加上去。
示例二
为了避免用户修改生成后的报表，可以将报表保存为 PDF 文件，然后再显示在窗口中：

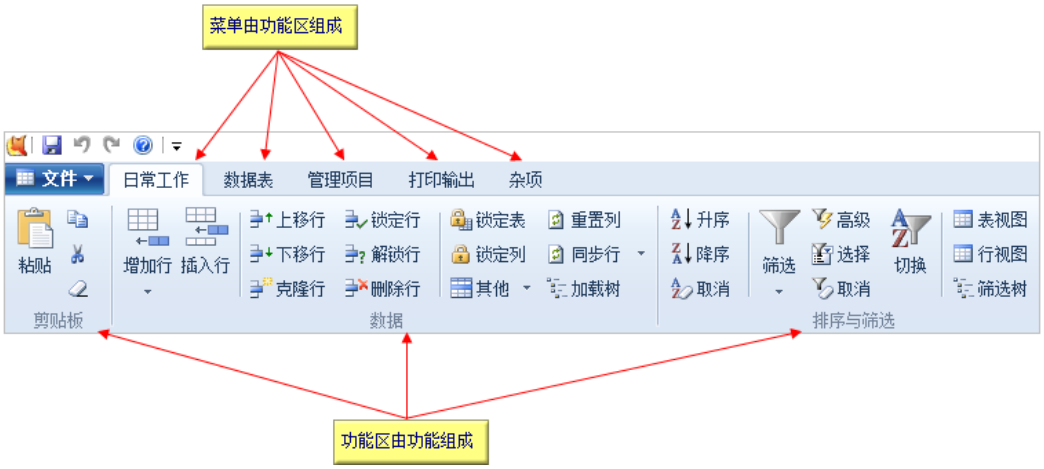
```
Dim wbr As WinForm.WebBrowser = e.Form.Controls("WebBrowser1")
Dim tm As String  = ProjectPath & "Attachments\出库单.doc" '指定模板文件
Dim fl1 As String = ProjectPath & "Reports\出库单.doc" '指定目标文件
Dim fl2 As String = ProjectPath & "Reports\出库单.pdf" '指定目标PDF文件
Dim wrt As New WordReport(Tables("出库"),tm,fl1) '定义一个WordReport
wrt.Build() '逐行生成报表
wrt.SaveToPDF(fl2) '保存为PDF文件
wrt.Quit() '退出
wbr.AddRess = fl2
```

提示：为了能打开生成的PDF文件，请先安装PDF阅读软件，例如：Adobe Reader X

4.8 菜单设计

4.8.1 主菜单组成

主菜单由功能区组成，例如系统菜单就包括日常工作、数据表、管理项目、打印输出、杂项等功能区，相似的功能放在同一个功能区中，例如和日常工作相关的命令都放在“日常工作”功能区中。功能区又由功能组组成，功能组用于对功能类似的命令进行进一步的区分，例如系统菜单的“日常工作”功能区，就被分成“剪贴板”、“数据”、“排序与筛选”、“数据统计”等功能组，其中和数据管理相关的命令放置在“数据”功能组，和数据统计相关的命令放置在“数据统计”功能组。



主菜单还包括程序菜单、快速访问栏、配置栏，传统的工具栏已经弱化成主菜单的一部分，后面我们会看到。

程序菜单类似于 Windows 的开始菜单，只是我们更多的命令是放在功能区，而不是程序菜单。窗口左上角的按钮为程序菜单按钮，单击会出现程序菜单。

程序菜单分为左、右、下三个分区：



快速访问栏在程序菜单按钮右侧，可以在这里放置一些常用的命令。
配置栏位于右上角，所以虽然名为配置栏，但是可以在这里放置任何你喜欢的常用命令。
快速访问栏和配置栏的命令一直可见，不用切换功能区也可以快速地访问执行。



4.8.2 主菜单组件

标准按钮

标准按钮是最常用的菜单组件，用于单击执行操作。
按钮位置是按顺序自动排列的，如果按钮设置的是大图标(32*32)，按钮将单独显示，如果设置的是小图标(16*16)或者没有设置图标，那么每三个按钮在垂直方向排成一列:



检查框

检查框通常用于启用或者禁用某个选项：



菜单按钮

菜单按钮本身不执行操作，单击后出现下拉菜单：



组合按钮

组合按钮实际上就是标准按钮和菜单按钮的组合，单击组合按钮的上半部分，执行操作，单击组合按钮的下半部分，出现下拉菜单。

例如下面的增加行按钮：



状态按钮

状态按钮外观类似于标准按钮，实际使用的时候类似于检查框。

例如下图中的计算按钮，用以打开和关闭自动计算功能：



如果有多个功能状态按钮，且设置了相同的“所属分组名”属性，那么这些状态按钮的状态将是互斥的，按下其中一个按钮，那么之前按下的按钮将会自动弹起。

标签

标签用于显示说明性的文字：



文本框

文本框用于输入内容:



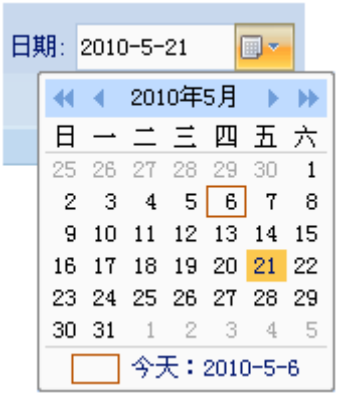
组合框

组合框不仅可以输入内容，还可以选择预先设置的内容：



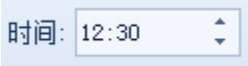
日期输入框

日期输入框专门用于输入日期，右边有一个下拉按钮，单击按钮可以显示一个日历，用于快速选择日期:



时间输入框

时间输入框用于输入时间，除了直接输入外，可以单击右边两个微调按钮，微调小时和分钟：



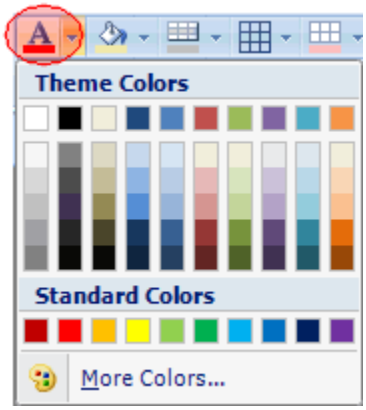
数值输入框

数值输入框专门用于输入数字：



颜色选取按钮

颜色选取按钮类似于组合按钮，只是其下拉的不是一个菜单，而是一个颜色列表：



字体选择框

字体选择框类似于组合框，只是单击下拉按钮的时候，会自动列出所有的字体供您选择:



进度条

通过设置进度条的最小值、最大值、当前值，可以表示某一任务的完成进度：



滑动条

滑动条用于可视化地调某一数值指标，中间的滑动块可以直接拖动，也可以单击左右的加减按钮进行微调：



工具栏

我们前面已经提过，功能组中的各个菜单组件是自动排列的，小图标或者没有图标的组件是在垂直方向顺序排列的，如果你希望按照水平方向顺序排列，那么就应该增加一个工具栏，将这些组件放置在工具栏中，例如下面的样式功能组就包括两个工具栏：



自动列表

自动列表会自动列出特定的项目，单击这些项目即可执行特定的操作。

例如窗口列表会自动列出当前表所有窗口的名称，单击其中一个，即可打开对应的窗口。



Foxtable 包括以下自动列表：
窗口列表
表视图列表
分组统计列表
交叉统计列表
汇总模式列表
Excel 报表列表
专业报表列表

分隔条

用于分割菜单组件：



4.8.3 如何返回系统菜单？

在自行设计菜单之前，首先必须了解如何回到系统菜单，因为自定义菜单生效后，通常你必须回到系统菜单才能重新设计自定义菜单。

回到系统菜单

在自定义菜单生效后，如果希望回到系统菜单，可以同时按 Ctrl+Shift+F12。
需要注意的是，只有目前登录用户是开发者，才可以回到系统菜单。
你也可以在自定义菜单中加入一个按钮，将其代码设置为：

```
Syscmd.Project.OpenSystemMenu()
```

这样只需单击此按钮，即可回到系统菜单。

回到自定义菜单

在系统菜单生效后，如果希望回到自定义菜单，可以同时按 Ctrl+Shift+F11。
你也可以在窗口中加入一个按钮，将其代码设置为：

```
Syscmd.Project.OpenUserMenu()
```

这样只需单击此按钮，即可回到自定义菜单。

判断当前菜单

Foxtable 提供了一个系统变量 SystemMenu，此变量为逻辑型，如果当前菜单为系统菜单，则返回 True，如果当前菜单为自定义菜单，则返回 False。

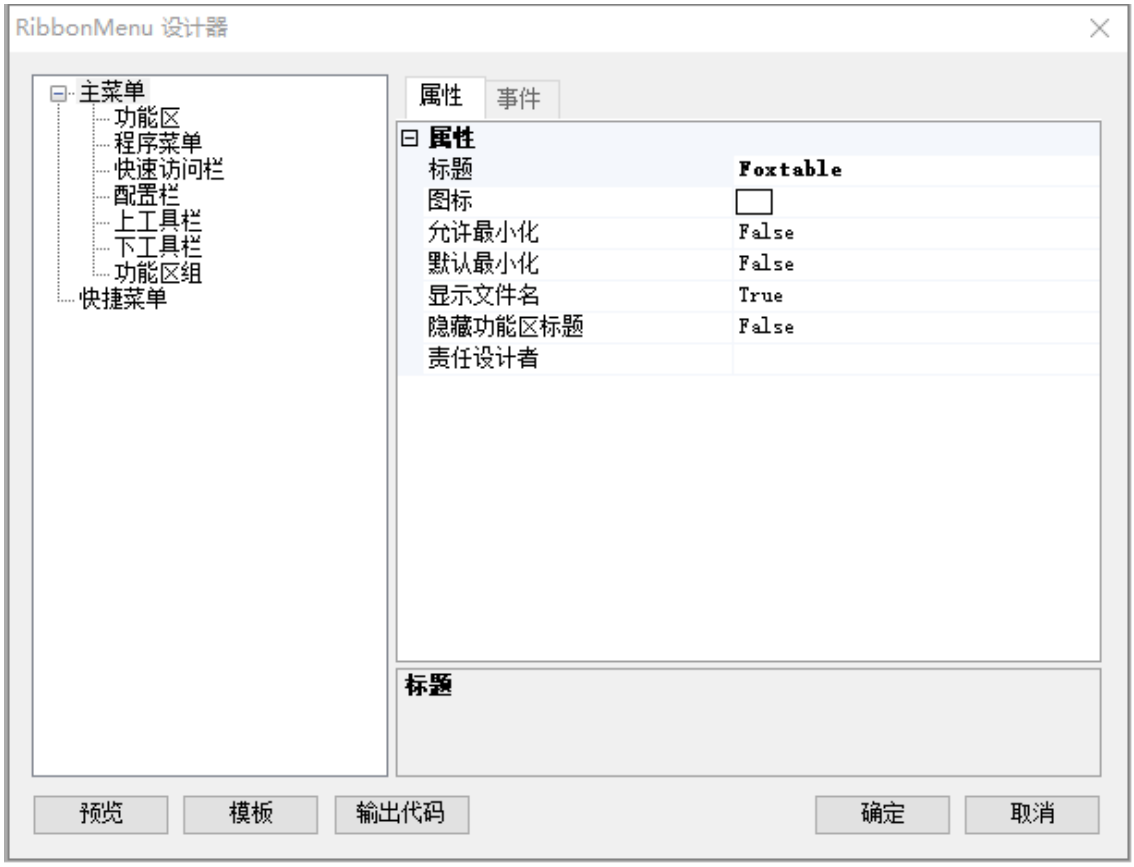
4.8.4 快速入门

4.8.4.1 A.设置主菜单属性

在“管理项目”功能区，单击下面的按钮，会出现菜单设计窗口：



菜单设计窗口中有两个根目录，分别是主菜单和快捷菜单，选择主菜单，窗口右边的属性设置窗口出现以下属性：



- **标题**
  指定启用此菜单后，Foxtable 程序窗口出现的标题，默认是"Foxtable"，你可以将其改为自己喜欢的名字，例如"XXX 数据管理系统"。

- **图标**
  指定程序菜单图标，也就是主窗口左上角的图标。

- **显示文件名**
  是否在窗口标题栏显示当前所打开文件的名称，默认显示。

- **允许最小化**
  是否允许最小化菜单，如果设为 True，运行过程中用户可以通过双击最小化菜单。

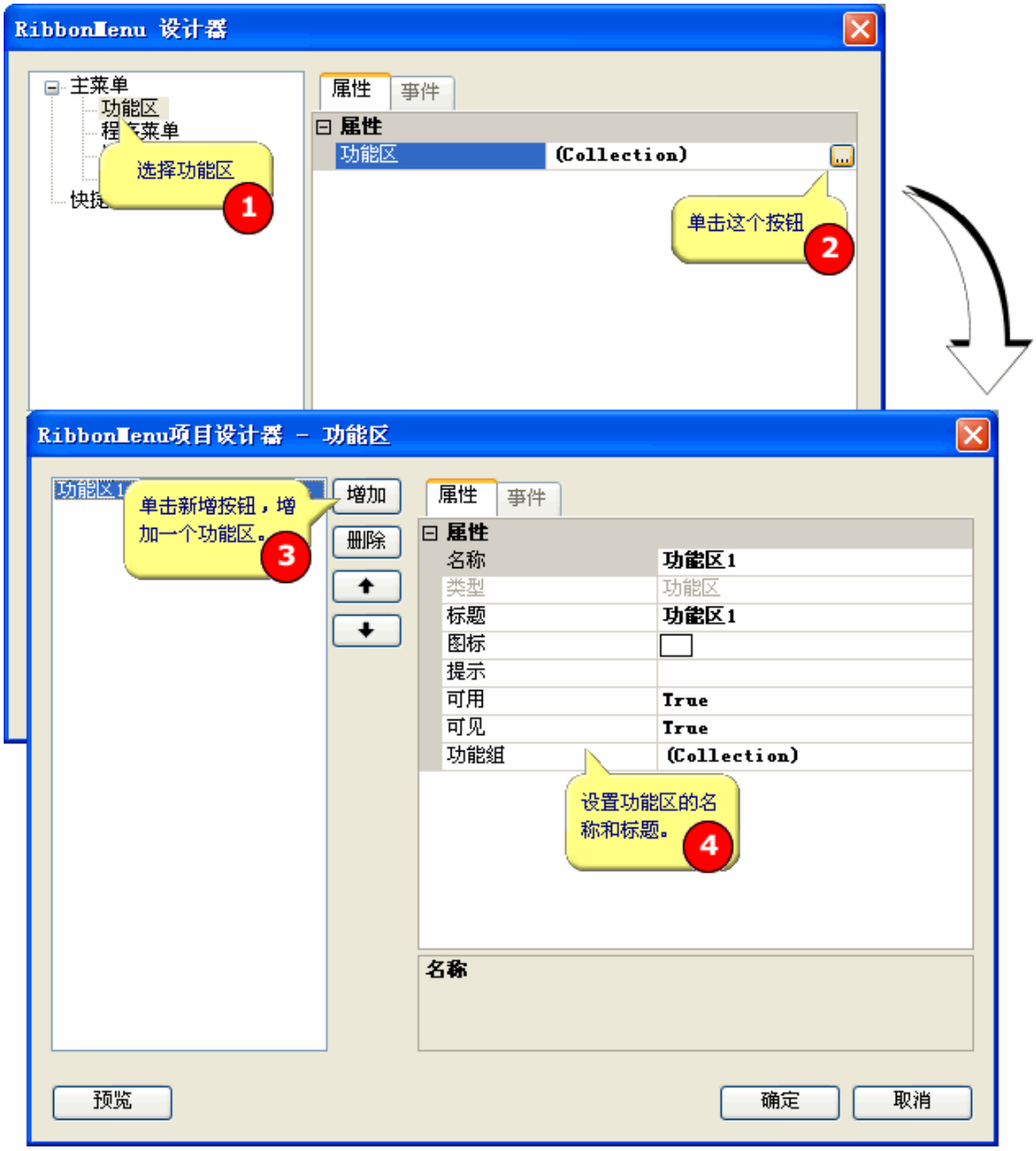
- **默认最小化**
  默认是否最小化菜单，只有前面的“允许最小化”属性为 True，本属性才会生效。

- **隐藏功能区标题**
  隐藏功能区的标题，只有一个功能区的时候，可以将此属性设置为 True，以节省屏幕空间。

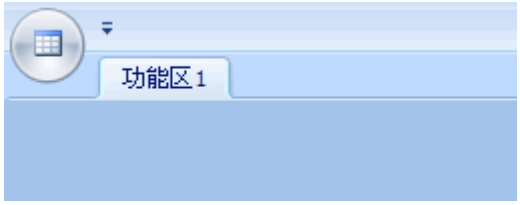


4.8.4.2 B.增加功能区

在菜单设计器左边的目录树中，选择功能区；然后单击属性页"功能区"项目右边的按钮，出现功能区管理窗口，单击“增加”按钮增加一个功能区，然后按需要设置新增功能区的名称和标题：

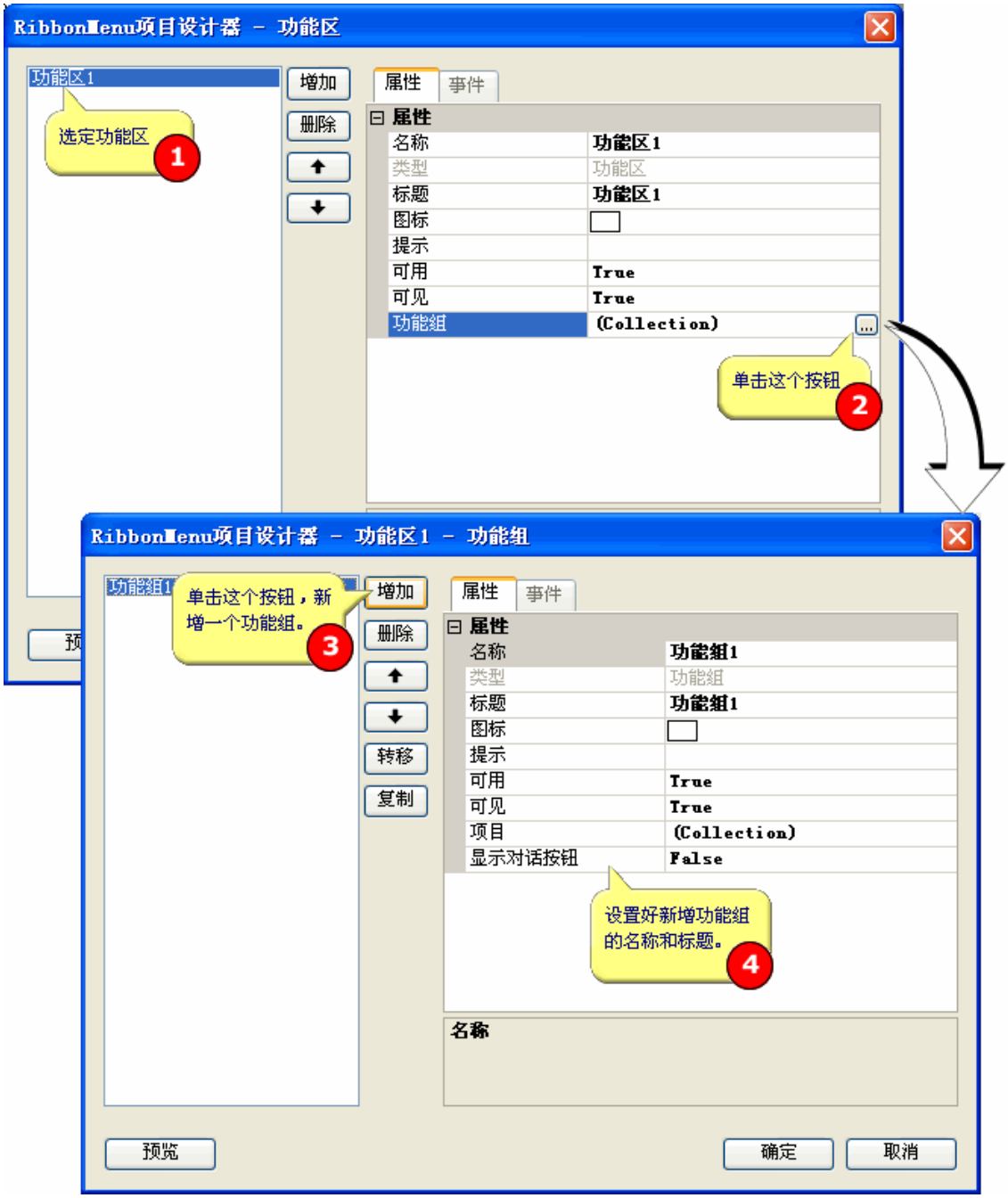


如果此时单击预览按钮，可以看到我们所增加的功能区：



4.8.4.3 C.增加功能组

选择要增加功能组的功能区，单击属性页“功能组”项目右侧的按钮，出现功能组管理窗口，单击“增加”按钮，增加一个功能组，在属性页设置好新增功能组的名称和标题。



现在单击“预览"按钮，可以看到我们刚刚增加的功能组：



4.8.4.4 D.添加标准按钮

选定新增的功能组，单击属性窗口中“项目”右边的按钮，出现项目管理窗口，单击“增加”按钮，会出现一个菜单，列出所有可以增加的菜单组件，我们选择增加一个标准按钮：



我们增加三个标准按钮，按下表所示，设置其名称、标题和图标：

名称	标题	大图标
Add	增加行	AddRow32.ico

Delete	删除行	DeleteRow32.ico
Clone	克隆行	Clone32.ico

名称也可以是中文，甚至可以和标题一样。

上表中的图标，包括今后菜单课程中使用的其它图标，均可以在 Foxtable 安装目录的 Images 子目录下找到。



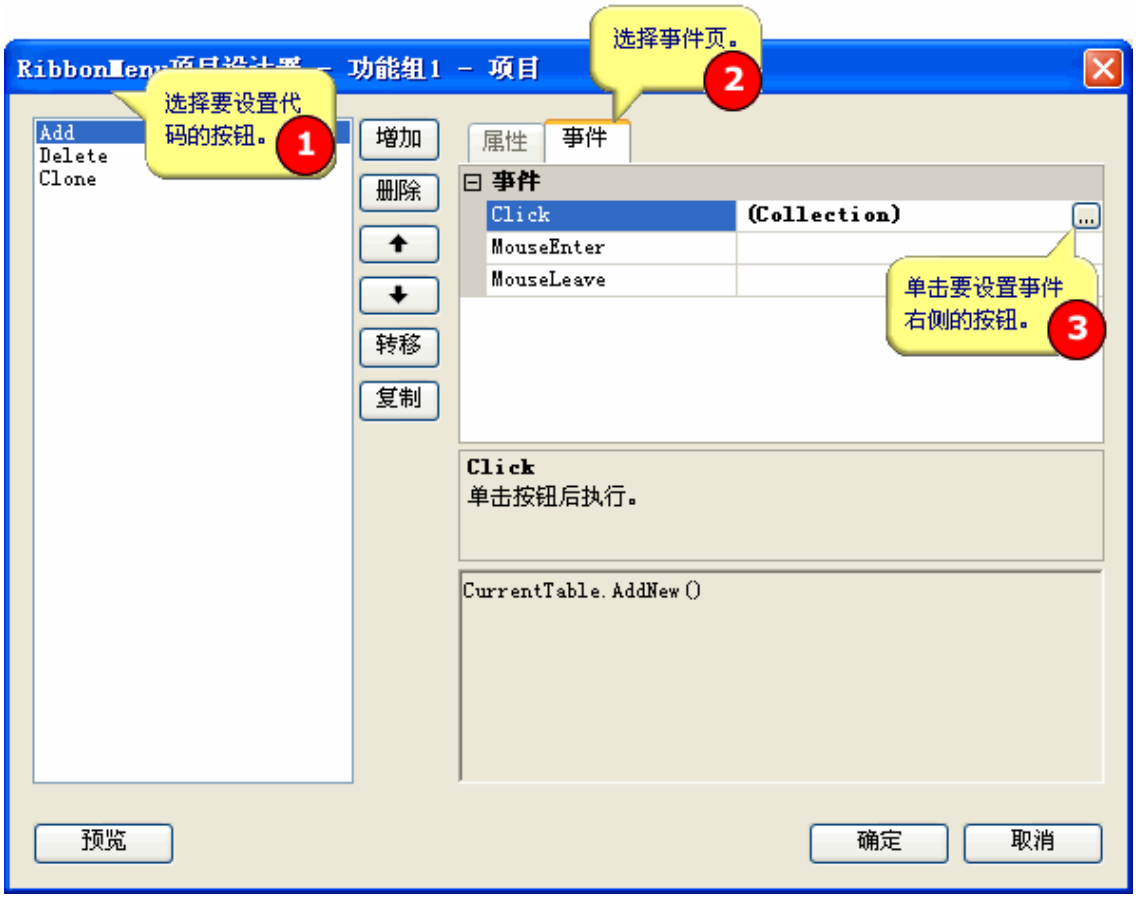
现在单击预览按钮，可以看到我们设计的菜单：



提示：如果要回到系统菜单，请同时按住：Ctrl+Shift+F12

4.8.4.5 E.给按钮加上代码

选择要设置代码的按钮，然后选择"事件"页，单击"Click"项目右边的按钮，为 Click 事件设置代码：



三个按钮的 Click 事件分别设为：

名称	标题	Click 事件代码
Add	增加行	CurrentTable.AddNew()

Delete	删除行	If CurrentTable.Current IsNot Nothing Then
		CurrentTable.Current.Delete
		End If

Clone	克隆行	If CurrentTable.Current IsNot Nothing Then
		CurrentTable.Current.Clone
		End If

最后逐次单击“确定”按钮退出菜单设计，可以看到我们设计的菜单将替换掉原来的系统菜单。

4.8.4.6 F:使用文本框

我们希望直接在工具栏中输入内容，然后单击某个按钮，在当前列查找所输入的内容。
首先在功能组中增加一个工具栏，在工具栏中增加以下项目：

类型	名称	小图标
文本框	StringToFind	
标准按钮	FindNext	find.ico

这里为什么要使用工具栏？因为功能组中的项目是按垂直顺序排列，只有工具栏中的项目是按水平顺序排列的。
将 FindText 的 Click 事件的代码设置如下：

```
'获得文本框中输入的内容,保存在变量 Val 中,实际编写代码的时候,组件名称应该改为真实的名称。
Dim Val As String = RibbonTabs("功能区 1").Groups("功能组 1").Items("工具栏 1").Items("StringToFind").Text
Dim Index As Integer
With CurrentTable
    Index = .FindRow(.Cols(.ColSel).Name & " = '" & Val & "'", .Position + 1,True)
    If Index >= 0 Then
        .Position = Index
    End If
End With
```

初次接触的代码只有下面这一行：

```
Dim Val As String = RibbonTabs("功能区 1").Groups("功能组 1").Items("工具栏 1").Items("StringToFind").Text
```

其实这行代码很好理解，就是获得指定功能区、指定功能组、指定工具栏中某一指定 文本框的 Text 属性值，并保存到变量 Val 中。
RibbonTabs 是一个全局集合，返回指定名称的变量；Goups 也是一个集合，返回指定名称的功能组；Items 也是一个集合，返回指定名称的组件，可用于功能组以及组合按钮、菜单按钮、工具栏等容器类组件；Text 是 文本框的一个属性，获得文本框中的内容。此外 Groups 和 Items 都是默认属性，所以这一行代码可以简化为：

```
Dim Val As String = RibbonTabs("功能区 1")("功能组 1")("工具栏 1")("StringToFind").Text
```

下图是我们设计的结果，在文本框中输入查询内容，单击右边的按钮即可进行查询：



4.8.4.7 G.使用组合框

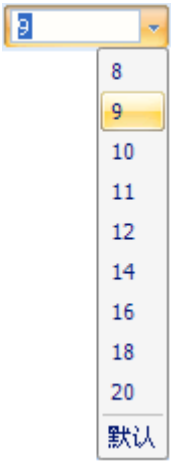
现在的任务在菜单中增加一个组合框，用以改变当前字体的字号。
设计步骤
1、在功能组中加入一个组合框，名称设为“FontSize”，内容设为“9”。

2、在列表项目集合增加 9 个标准按钮，按钮的标题分别为： 8、9、10、11、12、14、16、18、20。
3、将组合框的 ChangeCommitted 事件设置为：

```
Dim fnt As Font = CurrentTable.Font
Dim Size As Integer
'尝试将组合框的内容转换为整数,成功则将字体大小设为此整数
If Integer.TryParse(e.ComboBox.Text, Size) Then
    CurrentTable.Font = New Font(fnt.Name, Size, Fnt.Style)
End If
```

ComboBox 是组合框事件 e 参数的唯一属性，表示触发事件的组合框，Text 属性表示组合框中输入的内容。
有关事件和属性，我们会在菜单编程篇介绍。

下图就是我们的设计结果，既可以手工输入字号，也可以通过下拉列表来选择字号：



4.8.4.8 H.使用组合按钮

假定有一组非常类似的功能，例如增加行、增加多行、克隆行，我们可以用一个组合按钮来实现这三个功能，而不是三个按钮，因为这样太占地方。
我们需要得到一个下图所示的组合按钮：



设计过程很简单，首先在功能组中增加一个组合按钮，属性和事件代码设置如下：

属性	设置
名称	AddRow

标题	增加行
大图标	AddRow32.ico
Click 事件	CurrentTable.AddNew()

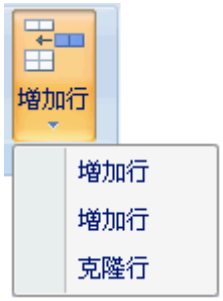
然后在组合按钮的项目集合中，增加三个标准按钮，按下表所示设置其属性和事件：

名称	标题	Click 事件
AddNew	增加行	CurrentTable.AddNew()
AddNews	增加多行	Syscmd.Row.AddNews() '这是一个系统命令，用于增加多行，接下来我们会介绍。

Clone	克隆当前行	If CurrentTable.Current IsNot Nothing Then
		CurrentTable.Current.Clone()
		End If

4.8.4.9 I.使用菜单按钮

菜单按钮的应用和组合按钮的应用，几乎完全一样，而且更简单。
假定要得到一个下图所示的菜单按钮：



设计过程很简单，首先在功能组中增加一个菜单按钮，属性设置如下：

属性	设置
名称	AddRow
标题	增加行
大图标	AddRow32.ico

然后在菜单按钮的项目集合中，增加三个标准按钮，按下表所示设置其属性和事件：

名称	标题	Click 事件
AddNew	增加行	CurrentTable.AddNew()
AddNews	增加多行	Syscmd.Row.AddNews() '这是一个系统命令，用于增加多行，接下来我们会介绍。
Clone	克隆当前行	If CurrentTable.Current IsNot Nothing Then

		CurrentTable.Current.Clone() End If

4.8.4.10 J.使用自动列表

自动列表是最复杂也是最简单的一个组件，复杂指的是自动按钮完成的功能复杂，而简单指的是应用简单，您只需插入指定的自动列表项目，设置一下其标题和图标，无需编写任何代码，系统会帮您自动完成剩余的工作。

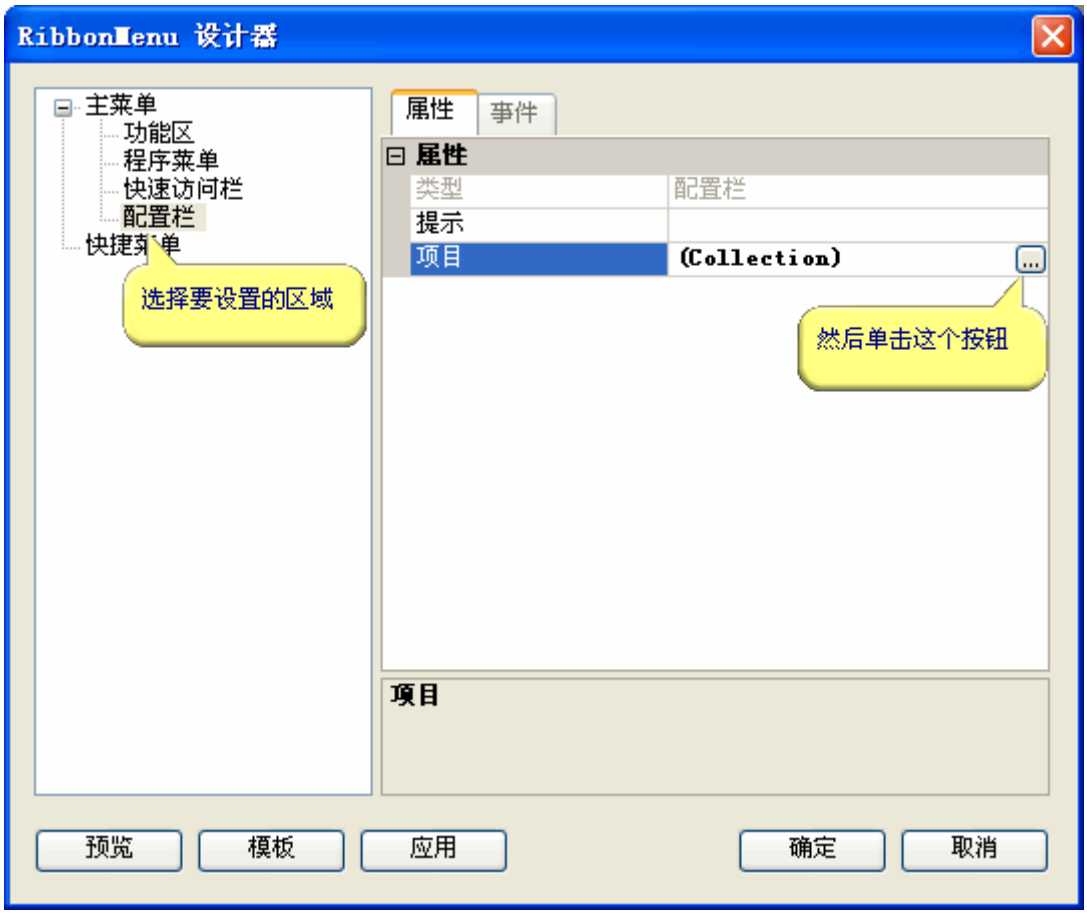
例如在功能组中插入窗口列表，运行的时候，单击窗口列表按钮，即可自动列出所有的工作窗口、关联表、列窗口，单击其中一个，即可打开对应的窗口:



4.8.4.11 K.设置其它区域

我们已经知道，除了功能区，主菜单还包括程序菜单、快速访问栏，配置栏。

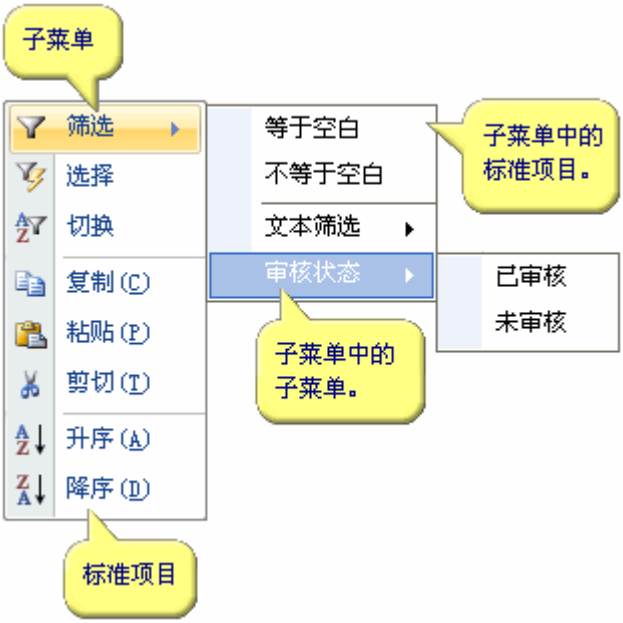
这些区域可以直接加入标准按钮这样的菜单组件，首先在设计器左侧选择相关的区域，然后在属性页单击“项目”右边的按钮即可：



4.8.4.12 M.使用快捷菜单

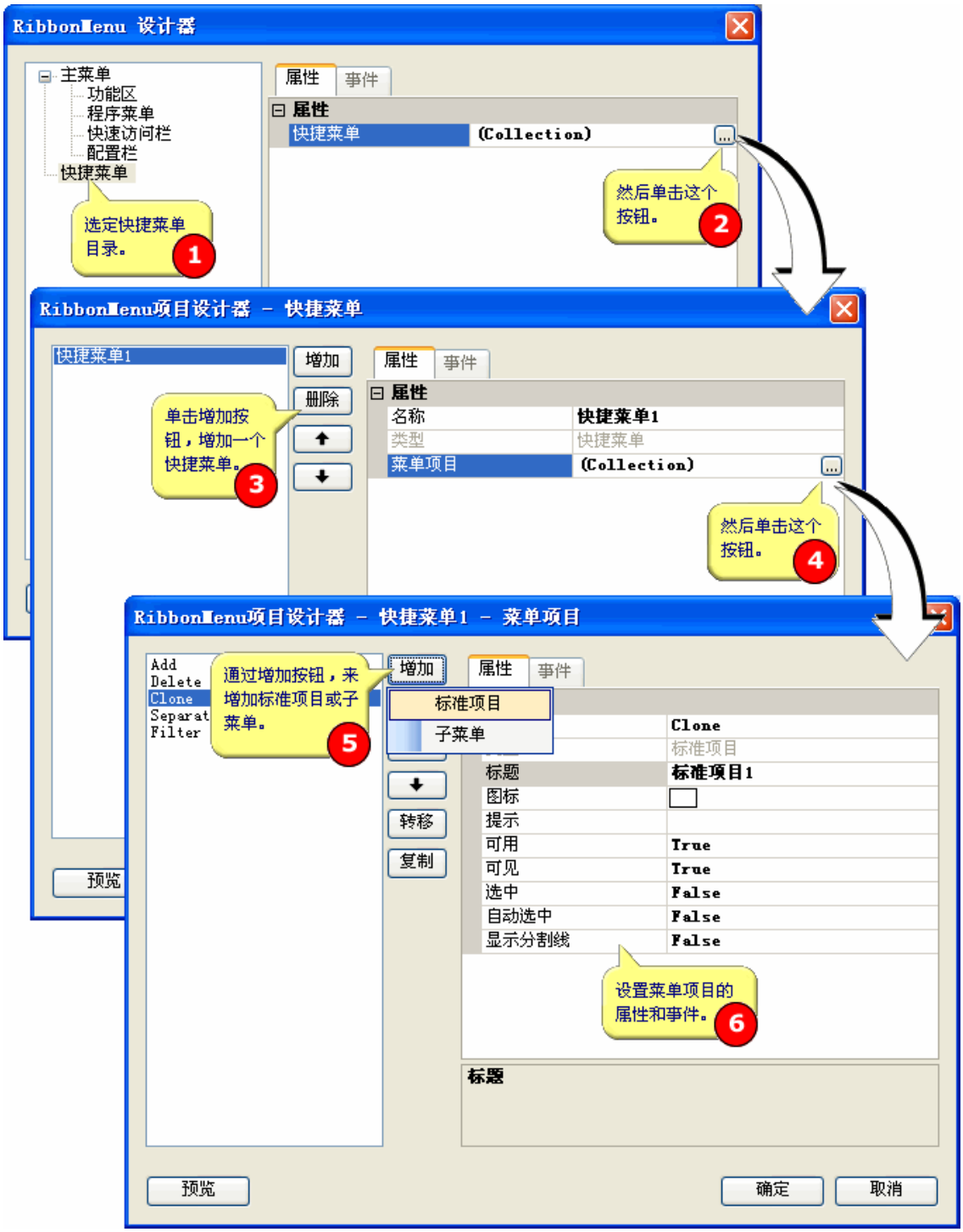
4.8.4.12.1 快捷菜单组件

快捷菜单只有两种组件，分别是“标准项目”和“子菜单”，标准项目就是一个命令，单击执行指定的操作，而子菜单是一个容器，可以包含标准项目和子菜单，这样可以设计出多层快捷菜单：



4.8.4.12.2 设计快捷菜单

快捷菜单的设计和主菜单设计几乎一样，不过要简单很多，下图是设计一个快捷菜单的基本流程：



4.8.4.12.3 默认快捷菜单

你也许已经发现，右击表格的不同区域，出现的快捷菜单并不相同。

那么在右击不同区域的时候，系统如何知道应该显示哪一个快捷菜单呢？

很简单，通过菜单名称来判断，例如你单击某个单元格的时候，系统会自动查找并显示名为“Cell”的快捷菜单。

这种具备特殊名称，且右击某一区域，会默认显示的快捷菜单，我们称之为默认快捷菜单。

全部默认快捷菜单名：

Column：右击列标题的时候出现
Row:　右击行号列的时候出现
Cell:　选定一个单元格后右击显示
Range:　选定多个单元格后右击显示
Empty:　右击空白区域的时候显示
提示：上述名称是区分大小写的。
Column 快捷菜单：



Row 快捷菜单:



Cell 快捷菜单:



Range 快捷菜单：



4.8.4.12.4 更改默认菜单

前面已经提到，右击表格某一区域，会自动显示和该区域对应的默认快捷菜单。

但是有的时候，你可能不希望显示默认菜单，而要求不同的表，不同状态的行列，显示不同的快捷菜单。

实现这样的要求很简单，我们早在菜单设计之前就已经讲述过，请参考：BeforeShowContextMenu

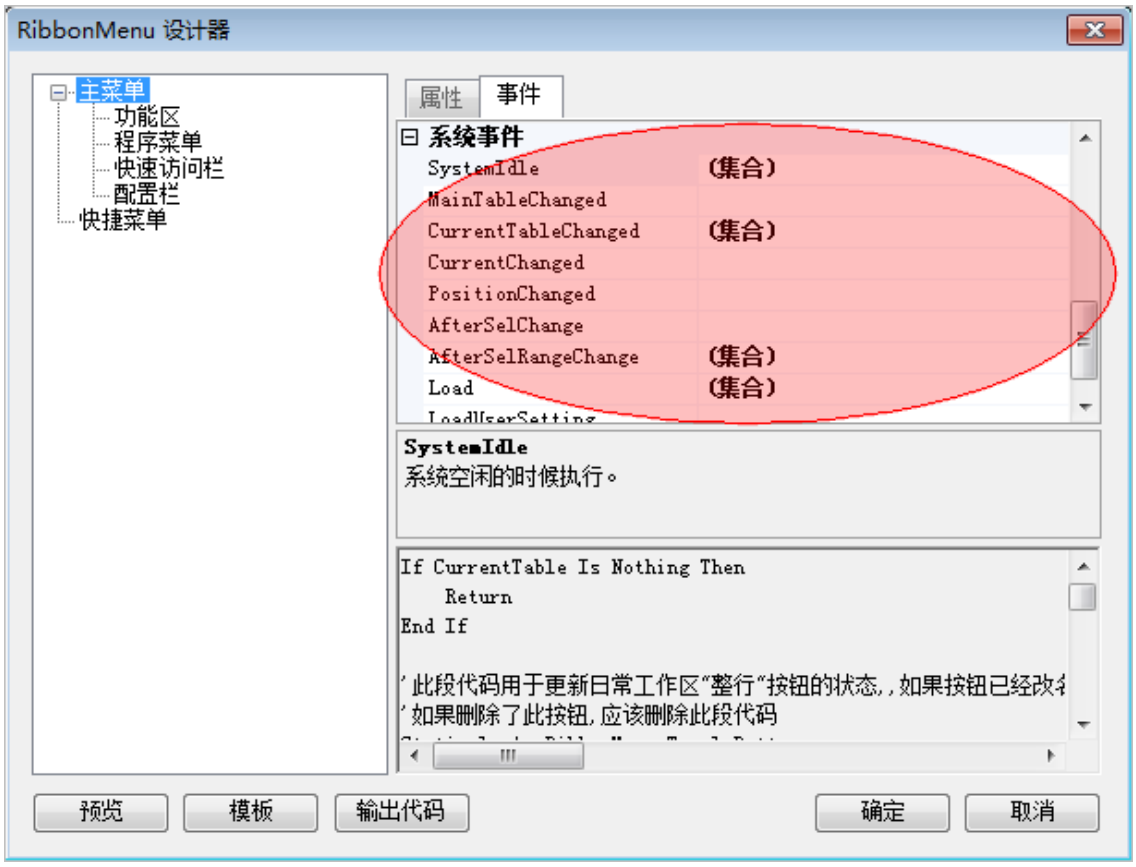
4.8.4.13 N.其他内容

4.8.4.13.1 菜单模板

在菜单设计窗口，单击模板按钮，可以将你设计的菜单方案保存为一个模板文件，也可以根据现有的模板文件，生成你的菜单。
需要注意的是，菜单模板只包括设计，不包括图标。我们在菜单设计过程中使用的图标，都会自动复制到项目文件的 Images 子目录下，如果要在新的项目中应用你设计好的菜单模板，必须手工将图标文件复制到新项目的 Images 子目录下

4.8.4.13.2 系统菜单

其实系统菜单也是通过菜单设计器设计的，系统菜单是一个菜单模板文件，位于 Foxtable 安装目录下，文件名为 System.Ribbon；系统菜单中的图标文件位于安装目录的 Images 子目录下。
大家在学习菜单的过程中，可以在设计窗口将该文件作为模板应用，在此基础上进行修改，设计出自己的个性化菜单。
如果你愿意，你可以将自己的菜单保存为模板文件，并替换掉 System.Ribbon 文件和 Images 目录下的图标，将 Foxtable 改头换面。需要注意的是，在这么做之前，最好将 System.Ribbon 文件和 Images 目录做个备份，这样替换后，一旦发现问题，还可以恢复到初始状态。
不单单是系统菜单，Foxtable 的很多功能，都是利用 Foxtable 自己来设计的。
需要注意的是，很多用户在系统菜单模板 System.Ribbon 的基础上设计菜单后，运行的过程中往往会出现错误提示，这是因为该菜单模板原来已经设置了系统事件代码:



例如系统事件 Load 中有代码：

```
If SysInfo.WorkingAreaWidth < 1280 '如果屏幕分辨率小于 1280
    RibbonTabs("Common").Groups("QuickStatistic").Visible = False ' 隐藏日常工作区的快速统计功能组
End If
```

这段代码的意思是：当屏幕的水平分辨率小于 1280 时，隐藏 Common(日常工作)功能区的 QuickStatistic(快速统计）功能组。

如果将 Common 功能区中的 QuickStatistic 功能组移到了别的功能区，或者修改了这二者的名称，显然上述代码的存在会引发错误。

所以修改系统菜单后，必须对应地调整菜单的系统事件代码，特别是 SystemIdle 和 Load 这两个系统事件中的代码。

SystemIdle 和 Load 这两个事件中的代码，已经根据功能分段并做了详细说明，大家可以根据需要进行删除和修改。

4.8.4.13.3 系统命令

如果你打开系统菜单模板，你会发现，里面的代码和我们之前的大不一样，例如系统菜单中增加行的代码为：

```
Syscmd.Row.AddNew()
```

这是一个系统命令，等效于：

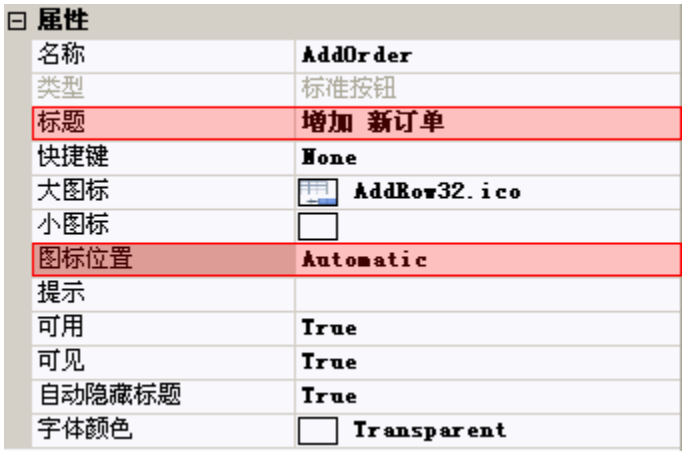
```
CurrentTable.AddNew()
```

在系统菜单中，绝大多数按钮使用的都是系统命令，那么系统命令和自己编写代码有何不同呢？

1、使用系统命令可以简化代码。虽然上面增加行的例子，系统命令和自己编写代码的繁简程度差不多，但是很多时候代码会比系统命令复杂很多，例如对于删除选定行，可不是一句代码能够完成的。
2、最重要的一点，使用系统命令完成的操作，可以撤销和重做，而自己编写的代码，虽然能完成同样的操作，但是不可以撤销和重做的。
3、系统命令只能完成最常用的操作，如果你有自己的特殊需要，那么可能就不得不自己编写代码了。
请参考：系统命令大全

4.8.4.13.4 分两行显示按钮标题

如果按钮的标题比较长，可以分两行显示标题，用空格分隔按钮的标题即可，并将图标位置设置属性为 Automatic
例如：



效果：



4.8.4.14 O.StatusBar(状态栏)

StatusBar 表示状态栏，可以在状态栏显示文字信息和进度条。
StatusBar 包括下述属性：

- Message1: 左侧的文字信息。
- Message2: 中间的文字信息。
- Message3: 右侧的文字信息，系统在这里显示自动计算结果。
- DefaultMessage: 设置左侧文字的默认信息，默认为“就绪”。
- ProgressBar: 返回进度条。

ProgressBar(进度条)包括四个属性：

- Visible：逻辑型，默认为 False，如果设为 True，将显示进度条。
- Minimum：进度条的起始值，默认为 0。
- Maximum：进度条的最大值，默认为 100。
- Value：　进度条的当前值。

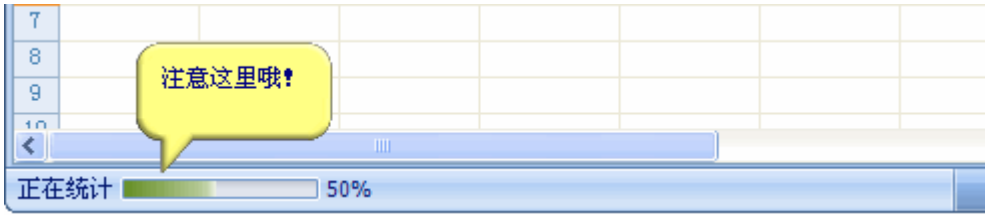
StatusBar 的方法：

- Refresh：默认情况下，在代码中设置状态栏的文字信息或进度条进度后，状态栏并不会立即显示最新的设置，而是等代码全部执行完毕后才会刷新，如果要在代码执行过程中更新状态栏的显示，执行此方法即可。
- Reset：初始化状态栏。

示例
将下面的代码复制到命令窗口：

```
StatusBar.Message1= "正在统计"
StatusBar.ProgressBar.Visible =true
StatusBar.ProgressBar.Value = 50
StatusBar.Message2= "50%"
```

执行之后，状态栏会显示如下内容：



如果要状态栏回到默认设置，可执行：

```
StatusBar.Reset()
```

4.8.4.15 P.我还需要继续吗？

对于大多数人来说，没有必要继续了，目前的知识对于普通的菜单设计已经足够.
关于菜单设计的学习可以到此为止，如果你希望更上一层楼，而且你有时间和耐心，那么请继续。

4.8.5 组件参考

4.8.5.1 容器类

4.8.5.1.1 Tab(功能区)

4.8.5.1.1.1 关于 Tab

Tab 表示功能区，类型为 RibbonMenu.Tab。
实际上和菜单有关的类都在 RibbonMenu 空间中，在定义的时候，需要加上前缀 RibbonMenu。
例如 RibbonMenu.Group 表示功能组，RibbonMenu.Button 表示标准按钮。

通过集合 RibbonTabs，可以获得指定名称的功能区。
示例

```
Dim rt As RibbonMenu.Tab
rt = RibbonTabs("Common")
```

4.8.5.1.1.2 属性

属性名	说明
Name	字符型，返回功能区的名称。
Text	字符型，返回或设置功能区的标题。
Image	返回或者设置功能区的图标，图标大小最好是 16*16 个像素。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
ToolTip	字符型，返回或者设置功能区的提示。
Enabled	逻辑型，返回或者设置功能区是否可用。
Selected	逻辑型，返回或设置此功能区是否为选定功能区。 示例： RibbonTabs("Project").Selected = True
Visible	逻辑型，返回或者设置功能区是否可见。 示例 Dim rt As RibbonMenu.Tab rt = RibbonTabs("Project") rt.Visible = False
Groups	集合， 可获得功能区中指定名称的功能组。 示例 Dim gp As RibbonMenu.Group gp = RibbonTabs("Common").Groups("Data") 表示 Common 功能区中名为 Data 的功能组，Groups 是功能区的默认属性，所以上面的代码可以简化为： Dim gp As RibbonMenu.Group gp = RibbonTabs("Common")("Data")

4.8.5.1.1.3 事件

下表是 Tab 的全部事件

事件名称	事件说明

SelectTab	选择功能区后执行。
DeselectTab	离开功能区后执行。
MouseEnter	鼠标进入功能区的时候执行。
MouseLeave	鼠标离开功能区的时候执行。

事件 e 参数属性：
Tab：返回触发事件的功能区。
示例
例如你希望选择某一个功能区后，能够自动打开指定的表，可以将该功能区的 SelectTab 事件代码设为：

```
Maintable = Tables("表名")
```

4.8.5.1.1.4 关于 RibbonTabs

通过集合 RibbonTabs，可以获得指定名称的功能区。
示例

```
RibbonTabs("Table").Visible = False
RibbonTabs("Project").Visible = False
```

在命令窗口执行上述代码，会隐藏 Table 和 Project 两个功能区。
RibbonTabs 还包括以下成员：

- AllowMinimize
  是否允许双击菜单标题后最小化菜单。

- Minimized
  是否最小化菜单，只有 AllowMinimize 属性为 True 的时候，才属性才有效。

- SelectedTab
  返回或设置选定的功能区。

- HideTabHeaderRow
  是否隐藏功能区标题。

示例
例如希望打开项目后，菜单能够自动最小化，可以在 AfterOpenProjec 事件中设置代码：

```
RibbonTabs.AllowMinimize = True
```

```
RibbonTabs.Minimized = True
```

4.8.5.1.2 Group(功能组)

4.8.5.1.2.1 关于 Group

Group 表示功能组。
功能区有一个 Groups 属性，表示该功能区的功能组集合，通过该集合，我们可以获得指定名称的功能组。
示例

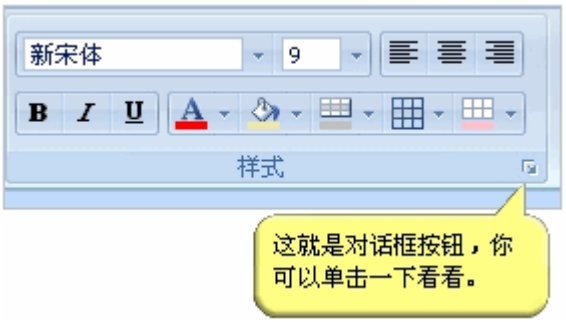
```
Dim gp As RibbonMenu.Group
gp = RibbonTabs("Common").Groups("Data")
```

表示 Common 功能区中名为 Data 的功能组，Groups 是功能区的默认属性，所以上面的代码可以简化为：

```
Dim gp As RibbonMenu.Group
gp = RibbonTabs("Common")("Data")
```

4.8.5.1.2.2 属性

属性名	说明
Name	字符型，返回功能 组的名称。
Text	字符型，返回或设置功能 组的标题。
Image	返回或者设置功能组的图标，图标大小最好是 16*16 个像素。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。 只有在屏幕控件不够，功能组收缩的时候，功能组的图标才会显示： 
ToolTip	字符型，返回或设置功能组的提示。
Enabled	逻辑型，返回或设置功能组是否可用。
Visible	逻辑型，返回或者设置功能组是否可见。 示例：

	Dim gp As RibbonMenu.Group gp = RibbonTabs("Common").Groups("Data") gp.Visible = False
Items	返回一个项目集合，包括此功能组中的所有菜单组件，请参考：项目集合
HasLauncherButton	逻辑型，如果设为 True，会在功能组右下角显示一个对话按钮，单击该按钮，会触发功能组的 DialogLauncherClick 事件。 

4.8.5.1.2.3 事件

下表是 Group 的全部事件：

事件名称	事件说明
DialogLauncherClick	单击对话按框按钮后执行。
MouseEnter	鼠标进入功能区的时候执行。
MouseLeave	鼠标离开功能区的时候执行。

e 参数属性：

Group：返回触发事件的功能组。

4.8.5.1.3 ApplicationMenu(程序菜单)

4.8.5.1.3.1 关于 ApplicationMenu

Foxtable 有一个系统变量 ApplicationMenu，用以返回程序菜单。

示例

ApplicationMenu.Tooltip = "这是 FoxTable 的程序菜单按钮"

4.8.5.1.3.2 属性

下表是 ApplicationMenu 的全部属性：

属性名	说明

Text	字符型，返回或设置程序菜单的标题。
ToolTip	字符型，返回或设置程序菜单按钮的提示。
Enabled	逻辑型，返回或设置程序菜单是否可用。
Visible	逻辑型，返回或设置程序菜单是否可见。
LeftItems	返回程序菜单左侧的项目集合，请参考：项目集合
RightItems	返回程序菜单右侧的项目集合，请参考：项目集合
BottomItems	返回程序菜单底部的项目集合，请参考：项目集合

4.8.5.1.3.3 事件

下表是 ApplicationMenu 的全部事件

事件名称	事件说明
MouseEnter	鼠标进入程序菜单按钮的时候执行。
MouseLeave	鼠标离开程序菜单按钮的时候执行。
DropDown	展开程序菜单的时候执行。
DropDownClosed	关闭程序菜单的时候执行。

事件无 e 参数。

4.8.5.1.4 QAT(快速访问栏)

4.8.5.1.4.1 关于 QAT

Foxtable 有一个系统变量 QAT，用以返回快速访问栏。
示例

```
QAT.BelowRibbon = True
```

4.8.5.1.4.2 属性

下表是 QAT 的全部属性：

属性名	说明
ToolTip	字符型，返回或设置快速访问栏的提示。
Enabled	逻辑型，返回或设置快速访问栏是否可用。
Visible	逻辑型，返回或设置快速访问栏是否可见。
BelowRibbon	逻辑型，返回或设置快速访问栏的位置。快速访问栏默认在功能区之上，将 BelowRibbon 设为 True 的话，快速访问栏将位于功能区之下。

Items	返回快速访问栏的项目集合，请参考：项目集合

4.8.5.1.4.3 事件

下表是 QAT 的全部事件：

事件名称	事件说明
MouseEnter	鼠标进入快速访问栏的时候执行。
MouseLeave	鼠标离开快速访问栏的时候执行。

事件无 e 参数。

4.8.5.1.5 ConfigBar(配置栏)

4.8.5.1.5.1 关于 ConfigBar

Foxtable 有一个系统变量 ConfigBar，用以返回配置栏。

4.8.5.1.5.2 属性

下表是 ConfigBar 的全部属性：

属性名	说明
ToolTip	字符型，返回或设置配置栏的提示。
Items	返回配置栏的项目集合，请参考：项目集合

4.8.5.1.5.3 事件

下表是 ConfigBar 的全部事件：

事件名称	事件说明
MouseEnter	鼠标进入配置栏的时候执行。
MouseLeave	鼠标离开配置栏的时候执行。

事件无 e 参数。

4.8.5.1.6 ToolBar(工具栏)

4.8.5.1.6.1 属性

工具栏的属性有：

- Visible
  逻辑型，返回或设置工具栏是否可见。

- Enabled
  逻辑型，返回或设置工具栏是否可用。

- Items
  返回工具栏的项目集合，请参考：项目集合

4.8.5.1.6.2 事件

工具栏有以下事件：

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
ToolBar: 返回触发事件的工具栏。

4.8.5.1.7 项目集合

RibbonTabs 是个集合，包括菜单的所有功能区。
功能区(Tab)的 Groups 属性是个集合，包括此功能区的所有功能组。
功能组(Group)、工具栏(ToolBar)、配置栏(ConfigBar)、组合按钮(SplitButton)、菜单按钮(MenuButton)、组合框(ComboBox)等都是容器，它们都有 Items 属性，此属性返回一个集合，包括此容器中的所有菜单组件。
程序菜单（ApplicationMenu）也是个容器，不过它有三个集合，用三个属性返回，分别是 LeftItems(左侧菜单组件集合)、RightItems(右侧菜单组件集合)、BottomItems(底部菜单组件集合)。
利用项目集合，不仅可以获得指定名称的菜单组件，我们还可以向其中增加、插入和删除菜单组件。
项目集合的方法：

方法	说明
Contains	判断是否包括指定名称的组件。 语法： Contains(Name) Name: 组件名称
Clear	清除所有组件

Add	增加一个组件，语法： Add(Item) Item: 要增加的项目
Insert	在指定位置插入一个组件。 语法： Insert(Index, Item) Index: 整数型，插入位置 Item: 要插入的组件。
Remove	删除指定的组件，或者删除指定名称的组件。 语法： Remove(Name) Remove(Item) Name: 组件名称 Item:组件

通过项目集合，可以返回集合中指定名称的菜单组件，例如：

```
Dim btn As RibbonMenu.Button
btn = RibbonTabs("Common").Groups("Data").Items("DeleteRow")
btn.Enabled = False
```

表示禁用 Common 功能区、Data 功能组中的 DeleteRow 组件，也就是禁用删除行按钮。

由于 Groups 和 Items 都是默认属性，所以可简化为：

```
Dim btn As RibbonMenu.Button
btn = RibbonTabs("Common")("Data")("DeleteRow")
btn.Enabled = False
```

如果喜欢，用一行代码也可以：

```
RibbonTabs("Common")("Data")("DeleteRow").Enabled = False
```

利用项目集合，可以动态生成菜单，后面的实战演练会介绍。

4.8.5.2 功能类

4.8.5.2.1 通用属性

所有的菜单组件都派生自 RibbonMenu.RibbonItem 类型，所以所有的菜单组件都会具备此类型的全部属性。

下表是 RibbonMenu.RibbonItem 的属性：

属性名	说明
Name	字符型，返回组件名称
ToolTip	字符型，返回或者设置组件的提示内容
Enabled	逻辑型，返回或者设置组件是否可用。
Visible	逻辑型，返回或者设置组件是否可见。
AutoHideText	逻辑型，是否允许自动隐藏标题。
LargeImage	返回或者设置组件的大图标，大图标最好是 32*32 个像素。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
SmallImage	返回或者设置组件的小图标，小图标最好是 16*16 个像素。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
TextImageRelation	设置标题和图标的相对位置，TextImageRelationEnum 枚举，可选值有： Automatic：　自动 ImageAboveText：　图片在文本之上 ImageBeforeText：图片在文本之前

上述属性对所有的菜单组件有效，例如：

```
Dim btn As RibbonMenu.ToggleButton
btn = RibbonTabs("Common").Groups("SortAndFilter").Items("Toggle")
btn.Text = "切换筛选"
btn.LargeImage = GetImage("Filter32.ico")
```

4.8.5.2.2 遍历组件

```
Dim itm1 As RibbonMenu.RibbonItem
Dim itm2 As RibbonMenu.RibbonItem
itm1 = RibbonTabs("Common").Groups("ClipBoard").Items("Paste")
itm2 = RibbonTabs("Print").Groups("QuickSetting").Items("OnlySelected")
```

如果要遍历容器中的所有控件，只能使用 RibbonMenu.RibbonItem 类型的变量，例如：

```
Dim itm As RibbonMenu.RibbonItem
For Each itm In RibbonTabs("Table").Groups("Table").Items
    itm.Enabled = False
Next
```

如果要遍历特定类型的控件，可以用 Typeof 判断控件类型，例如：

```
Dim itm As RibbonMenu.RibbonItem
For Each itm In RibbonTabs("Table").Groups("Column").Items
   If Typeof itm Is RibbonMenu.Button Then
      itm.Enabled = False
   End If
```

Next

有关派生和判断类型的知识，在窗口设计的遍历控件一节有详细的介绍，大家可以温习一下。

4.8.5.2.3 Button(标准按钮)

4.8.5.2.3.1 属性

标准按钮的属性:

- Text
  字符型，返回或者设置标准按钮的标题。

示例

```
Dim btn As RibbonMenu.Button
btn = RibbonTabs("Common").Groups("Data").Items("DeleteRow")
btn.Text = "删除行"
```

4.8.5.2.3.2 事件

标准按钮有以下事件：

- Click
  单击标准按钮后执行。

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
Button: 返回触发事件的标准按钮。

4.8.5.2.3.3 方法

标准按钮的方法：

- PerformClick
  模拟单击该按钮。

示例

```
Dim btn As RibbonMenu.Button
```

```
btn = RibbonTabs("Common").Groups("Data").Items("DeleteRow")
btn.PerformClick()
```

如果你用的是系统菜单，执行上面的代码会删除选定行，因为这段代码模拟单击删除行按钮。

4.8.5.2.4 MenuButton(菜单按钮)

4.8.5.2.4.1 属性

菜单按钮的属性有：

- Text
  返回或者设置菜单按钮的标题。

- Items
  返回菜单按钮的项目集合。

4.8.5.2.4.2 事件

菜单按钮的事件有：

- DropDown
  展开下拉菜单的时候执行。

- DropDownClosed
  关闭下拉菜单的时候执行。

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
MenuButton：返回触发事件的菜单按钮

4.8.5.2.5 SplitButton(组合按钮)

4.8.5.2.5.1 属性

组合按钮的属性有：

- Text
  返回或者设置组合按钮的标题。

- Items
  返回组合按钮的项目集合。

4.8.5.2.5.2 事件

组合按钮有以下事件：

- Click
  单击组合按钮后执行。

- DropDown
  展开下拉菜单的时候执行。

- DropDownClosed
  关闭下拉菜单的时候执行。

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
SplitButton: 返回触发事件的组合按钮。

4.8.5.2.5.3 方法

组合按钮的方法：

- PerformClick
  模拟单击该按钮。

4.8.5.2.6 ToggleButton(状态按钮)

4.8.5.2.6.1 属性

ToggleButton 的属性有：

- Text
  返回或者设置状态按钮的标题

- Pressed
  逻辑型，返回或者设置状态按钮是否处于按下状态。

- GroupName
  字符型，用于设置状态按钮的分组名
  具有相同分组名的状态按钮，相互之间的状态是互斥的，按下其中一个按钮，那么之前按下的按钮将会自动弹起。

4.8.5.2.6.2 事件

ToggleButton 的事件有：

- Click
  单击状态按钮后执行。

- PressedChanged
  按下或者弹起状态按钮后执行。

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
ToggleButton：返回触发事件的状态按钮

4.8.5.2.7 TextBox(文本框)

4.8.5.2.7.1 属性

TextBox 的属性有：

- Text
  返回或者设置文本框中的内容。

- Label
  返回或者设置文本框的标题。

- TextAreaWidth
  整数型，返回或设置文本框的宽度。

4.8.5.2.7.2 事件

事件名称	说明	e 参数属性
ChangeCommitted	提交修改后执行，这个事件使用频率最高。	TextBox：表示触发事件的文本框
TextChanged	内容改变后执行。	
ChangeCanceled	取消修改后执行。	
GotFocus	获得输入焦点后执行。	
LostFocus	失去输入焦点后执行。	
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.2.8 DatePicker(日期框)

4.8.5.2.8.1 属性

DatePicker 的属性有：

- Value
  日期型，返回或者设置日期框的值。

- Label
  字符型，返回或者设置日期框的标题。

- TextAreaWidth
  整数型，返回或设置日期框的宽度。

4.8.5.2.8.2 事件

事件名称	说明	e 参数属性
ChangeCommitted	提交修改后执行，这个事件使用频率最高。	DatePicker：表示触发事件的日期框。
ValueChanged	内容改变后执行。	
ChangeCanceled	取消修改后执行。	
GotFocus	获得输入焦点后执行。	
LostFocus	失去输入焦点后执行。	
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.2.9 TimePicker(日期输入框)

4.8.5.2.9.1 属性

TimePicker 的属性有：

- Value
  TimeSpan(时段)型，返回或者设置时间输入框的值。

- Label
  字符型，返回或者设置时间输入框的标题。

- TextAreaWidth
  整数型，返回或设置时间输入框的宽度。

- LongTimeFormat
  逻辑型，是否为长时间格式，长时间格式包括秒。

4.8.5.2.9.2 事件

事件名称	说明	e 参数属性
ChangeCommitted	提交修改后执行，这个事件使用频率最高。	TimePicker：表示触发事件的时间输入框。
ValueChanged	内容改变后执行。	
ChangeCanceled	取消修改后执行。	
GotFocus	获得输入焦点后执行。	

LostFocus	失去输入焦点后执行。	
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.2.10 NumericBox(数值框)

4.8.5.2.10.1 属性

NumericBox 的属性有：

- Value
  数值型，返回或者设置数值框的值。

- Minimum
  数值型，返回或设置数值框允许输入的最小值

- Maximum
  数值型，返回或设置数值框允许输入的最大值

- Increment
  数值型，返回或者是单击数值输入框的微调按钮时，数值改变的幅度。

- Label
  字符型，返回或者设置日期框的标题。

- TextAreaWidth
  整数型，返回或设置日期框的宽度。

4.8.5.2.10.2 事件

事件名称	说明	e 参数属性
ChangeCommitted	提交修改后执行，这个事件使用频率最高。	NumericBox：表示触发事件的数值框。
ValueChanged	内容改变后执行。	
ChangeCanceled	取消修改后执行。	
GotFocus	获得输入焦点后执行。	
LostFocus	失去输入焦点后执行。	
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.2.11 ComboBox(组合框)

4.8.5.2.11.1 属性

ComboBox 的属性有：

- Text
  返回或者设置组合框中的内容。

- Label
  返回或者设置组合框的标题。

- TextAreaWidth
  整数型，返回或设置组合框的宽度。

- Items
  返回列表项目集合。

4.8.5.2.11.2 事件

事件名称	说明	e 参数属性
ChangeCommitted	提交修改后执行，这个事件使用频率最高。	ComboBox：表示触发事件的组合框
TextChanged	内容改变后执行。	
ChangeCanceled	取消修改后执行。	
GotFocus	获得输入焦点后执行。	
LostFocus	失去输入焦点后执行。	
DropDown	展开下拉菜单的时候执行。	
DropDownClosed	关闭下拉菜单的时候执行。	
SelectedIndexChanged	择不同列表项目后执行。	
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.2.12 Label(标签)

4.8.5.2.12.1 属性

标签的属性:

- Text
  字符型，返回或者设置标签的文本内容。

4.8.5.2.12.2 事件

标签有以下事件：

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
Label: 返回触发事件的标签。

4.8.5.2.13 CheckBox(检查框)

4.8.5.2.13.1 属性

检查框的属性:

- Text
  字符型，返回或设置检查框的标题。

- Checked
  逻辑型，是否选中检查框。

4.8.5.2.13.2 事件

检查框的事件有：

- CheckedChanged
  检查框的选中状态发生改变后执行。

e 参数属性：
CheckBox：表示触发事件的检查框

4.8.5.2.14 Separator(分割条)

4.8.5.2.14.1 属性

除通用属性外，Separator(分割条)并无其余属性。

4.8.5.2.14.2 事件

分割条有以下事件：

- MouseEnter
  鼠标进入组件区域的时候执行。

- MouseLeave
  鼠标离开组件区域的时候执行。

e 参数属性：
Separator: 返回触发事件的分割条。

4.8.5.2.15 ColorPicker(颜色拾取按钮)

4.8.5.2.15.1 属性

ColorPicker 的属性有：

- Text
  返回或设置颜色拾取按钮的标题。

- Color
  Color 型枚举，返回或设置选定的颜色。

4.8.5.2.15.2 事件

ColorPicker 的事件有：

- SelectedColorChanged
  选定不同颜色后执行。

e 参数属性：
ColorPicker：返回触发事件的颜色拾取按钮。

4.8.5.2.16 FontComboBox(字体选择框)

4.8.5.2.16.1 属性

FontComboBox 的属性有：

- Text
  返回或设置字体组合框中的内容。

- Label
  返回或设置字体组合框的标题。

- MenuItems
  返回菜单项目集合。

- TextAreaWidth
  整数型，返回或设置组合框的宽度。

4.8.5.2.16.2 事件

事件名称	说明	e 参数属性
ChangeCommitted	提交修改后执行，这个事件使用频率最高。	FontComboBox：表示触发事件的字体选择框
TextChanged	内容改变后执行。	
ChangeCanceled	取消修改后执行。	
GotFocus	获得输入焦点后执行。	
LostFocus	失去输入焦点后执行。	
DropDown	展开下拉菜单的时候执行。	
DropDownClosed	关闭下拉菜单的时候执行。	
SelectedIndexChanged	择不同列表项目后执行。	
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	
KeyDown	按下某按键后执行。	FontComboBox：表示触发事件的字体选择框。 Alt： 逻辑型，判断是否同时按下了 Alt 键 Control： 逻辑型，判断是否同时按下了 Ctrl 键 Shift： 逻辑型，判断是否同时按下了 Shift 键 KeyCode： Keys 枚举型，表示按键的代码。 Cancel： 逻辑型，设为 True，取消此次按键操作
KeyUp	松开某按键后执行。	
KeyPress	按下某字符键后执行。	FontComboBox：表示触发事件的字体选择框。

		KeyChar： 字符型，返回所按下的字符键。 Cancel： 逻辑型，设为True，取消本次输入操作。

4.8.5.2.17 ProgressBar(进度条)

4.8.5.2.17.1 属性

ProgressBar 的属性有：

- Value
  数值型，返回或设置进度条的当前值。

- Minimum
  数值型，返回或设置进度条的最小值，

- Maximum
  数值型，返回或设置进度条的最大值

- Label
  字符型，返回或者设置日期框的标题。

- Width
  整数型，返回或设置进度条的宽度。

- Height
  数值型，返回或设置进度条的高度。

4.8.5.2.17.2 事件

进度条的事件有：

事件名称	说明	e 参数属性
ValueChanged	当前值改变后执行。	ProgressBar：触发事件的进度条
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.2.18 TrackBar(滑动条)

4.8.5.2.18.1 属性

TrackBar 的属性有：

- Value
  数值型，返回或设置进度条的当前值。
- Minimum
  数值型，返回或设置进度条的最小值，
- Maximum
  数值型，返回或设置进度条的最大值
- Label
  字符型，返回或者设置日期框的标题。
- Width
  整数型，返回或设置进度条的宽度。
- TickFrequency
  整数型，刻度间隔值
- StepFrequency
  整数型，每次单击加减按钮值的改变幅度。

4.8.5.2.18.2 事件

滑动条的事件有：

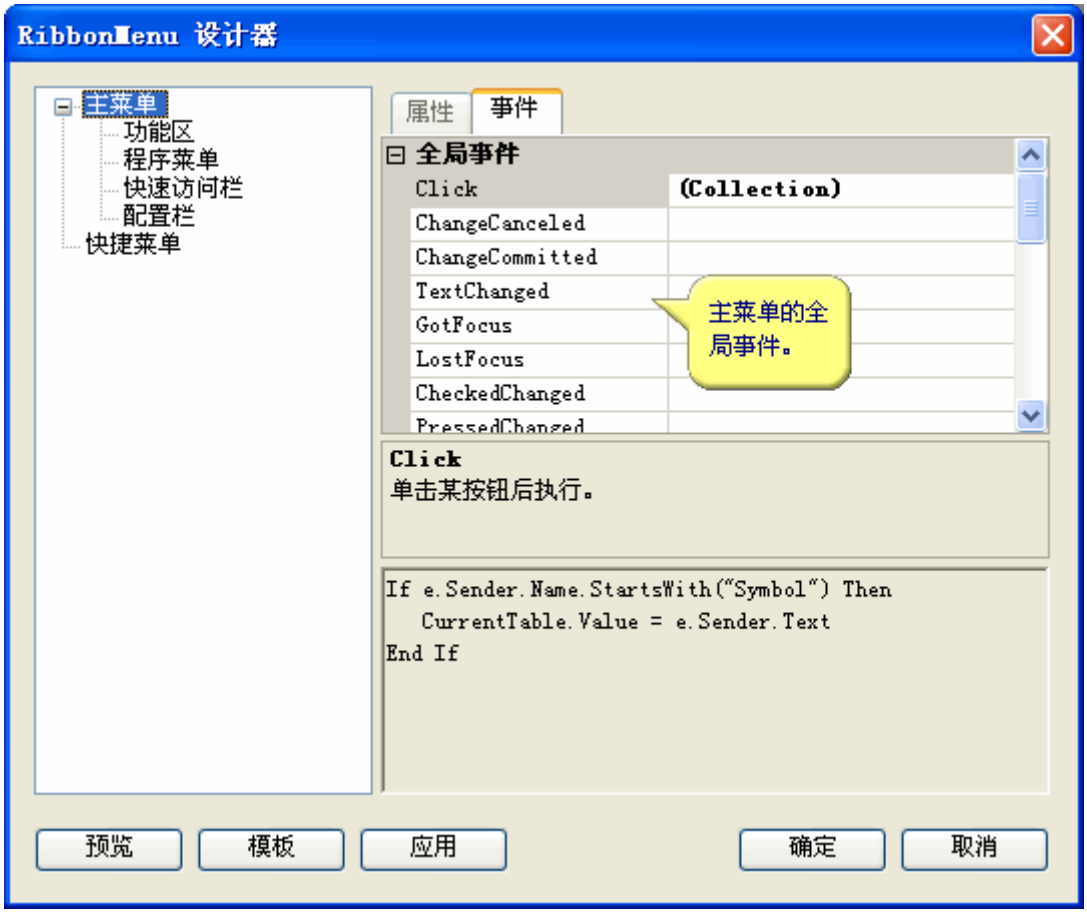
事件名称	说明	e 参数属性
ValueChanged	当前值改变后执行。	TrackBar：触发事件的进度条
MouseEnter	鼠标进入组件区域的时候执行。	
MouseLeave	鼠标离开组件区域的时候执行。	

4.8.5.3 全局事件

通常我们会针对每一个菜单组件编写事件代码，因为这些组件各自要完成不同的功能。但是有的时候有一些组件会用来完成同样的功能，例如你增加了 30 个按钮，用来向表格插入 30 个不同的符号，难道我们要针对这 30 个按钮编写 30 次相同的代码吗？ 如果代码有变动，难道我门又要进

行 30 次同样的修改吗？

考虑到这种特殊的情况，主菜单提供了全局事件处理程序，我们前面介绍的菜单组件事件，都有同名的全局事件。但我们单击任何一个按钮，首先会执行该按钮的 Click 事件，然后再执行全局 Click 事件，其他事件的执行过程亦如此。



所有的全局事件的 e 参数均只有一个属性：

Sender：返回触发事件的组件。

有了全局事件，那么前面的问题就迎刃而解了，首先增加 30 个按钮，为了便于代码的编写，这些代码分别命名为：Symbol1、Symbol2、Symbol3....，然后将这些按钮的标题设置为要插入的符号，最后将全局 Click 事件的代码设置为：

```
If e.Sender.Name.StartsWith("Symbol") Then
    CurrentTable.Value = e.Sender.Text
End If
```

很简单，对么？

4.8.5.4 系统事件

主菜单的系统事件有：

- Load
  生成菜单后执行，通常用来做一些初始化工作。

- SystemIdle
  系统空闲的时候执行

- MainTableChanged
  选择不同主表后执行。

- CurrentTableChanged
  选择不同表后执行。

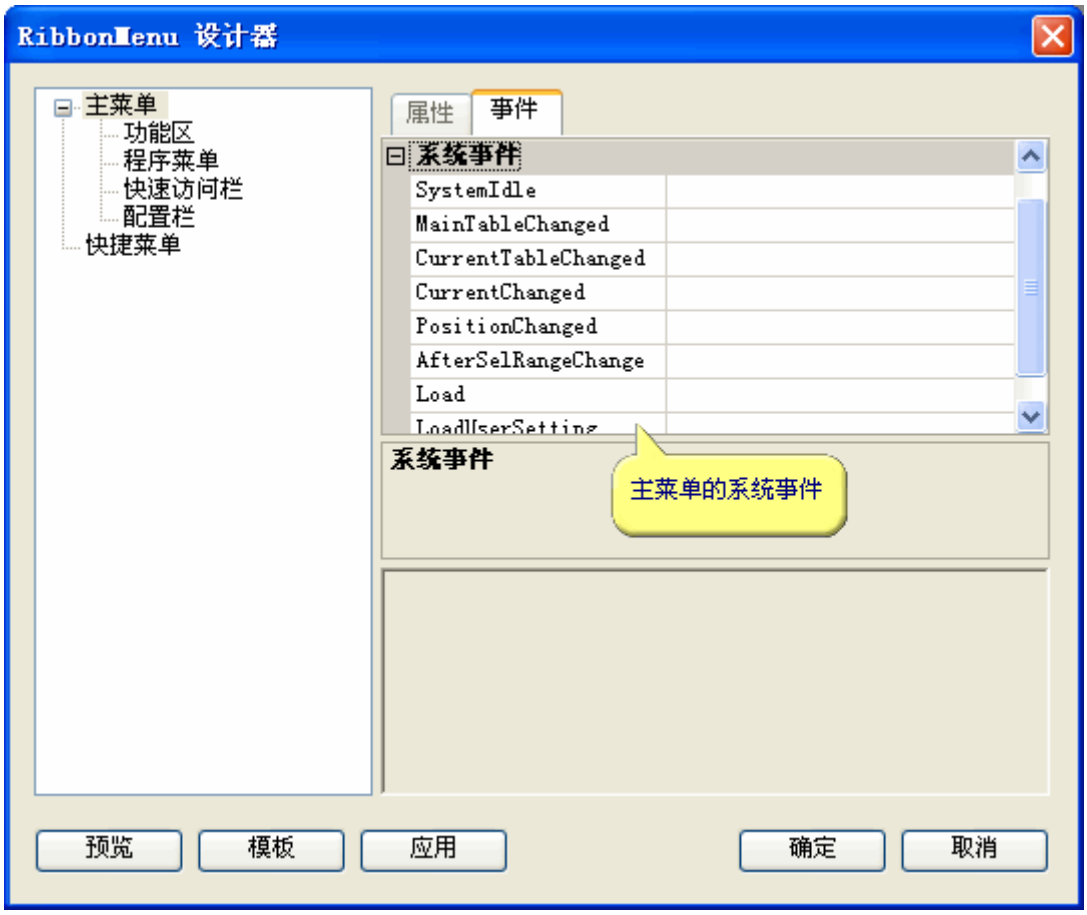
- CurrentChanged
  选择不同行后执行。

- PositionChanged
  选择不同行位置后执行

- AfterSelChange
  选择不同单元格后执行

- AfterSelRangeChange
  在表中选择不同区域后执行。

- LoadUserSetting
  打开系统或者切换用户之后执行，可以在此针对用户进行权限方面的设置。



除了 Load 事件，上述事件我们已经在表事件和项目事件的课程中讲过，使用方法完全一样。

如果事件代码和菜单有关，应该使用菜单事件，如果和菜单无关，和多个表有关，应该使用项目级别的事件，如果代码只是针对单个表，那么应该使用表事件。

例如你希望在菜单中的某个文本框显示当前行的位置和总的行数，那么就应该在菜单设计中，为 PositionChanged 事件编写代码。

这样的代码如果编写在项目级别中，并不是不可以，但是如果项目应用了新的菜单方案，那么代码就有可能出错，因为代码中引用的文本框已经不存在了。

同样如果你希望某个表成为主表后，立即执行一些操作，那么就应该使用项目级别的 MainTableChanged 事件来跟踪，因为这样的操作和菜单没有任何关系。

还有，如果希望你的代码是在某一个表中选择不同的行后执行，那么代码就应该设置在这个表的 CurrentChanged 或者 PositionChanged 事件中。

同样的事件在两个甚至三个地方出现，是为了提供最大的灵活性，具体使用哪一个位置的事件，并没有固定的准则，请自己灵活掌握。

4.8.5.5 快捷菜单参考

4.8.5.5.1 ContextMenu(快捷菜单)

4.8.5.5.1.1 关于 ContextMenu

ContextMenu 表示快捷菜单，类型为 RibbonMenu.ContextMenu.
通过集合 ContextMenus，可以获得指定名称的快捷菜单。
例如：

```
Dim mnu As RibbonMenu.ContextMenu
mnu = ContextMenus("Cell")
```

4.8.5.5.1.2 属性

ContextMenu 包括下述属性：

属性名	说明
Name	返回快捷菜单的名称
Items	获得快捷菜单中指定名称的项目，例如： Dim mnu As RibbonMenu.ContextMenu mnu = ContextMenus("Cell") mnu.Items("Copy").Enabled = False 上面的代码禁用了快捷菜单中名为"Copy"的项目。 Items 是默认属性，所以上面代码的最后一行，可以简化为： mnu("Copy").Enabled = False

4.8.5.5.1.3 方法

快捷菜单就一个方法 Show，用于在指定的控件上显示快捷菜单。
语法：
Show(Ctl, X, Y)
Ctl: 要显示快捷菜单的控件
X： 水平显示位置，从控件的左上角开始计算。
Y: 垂直显示位置，从空间的左上角开始计算。
示例
假定希望在某个控件上右击时，能够显示某一快捷菜单，可以将该控件的 MouseDown 事件设为：

```
If e.Button = MouseButtons.Right Then '假定按下的是鼠标右键
    ContextMenus("快捷菜单名").Show(e.Sender,e.x,e.y) '则在鼠标位置显示快捷菜单
End If
```

4.8.5.5.1.4 事件

4.8.5.5.1.4.1 Popup

展开快捷菜单的时候执行。
e 参数属性：
ContextMenu：要展开的快捷菜单。

4.8.5.5.1.4.2 Closed

关闭快捷菜单的时候执行。
e 参数属性：
ContextMenu：要关闭的快捷菜单。

4.8.5.5.2 ContextCommand(快捷菜单命令)

4.8.5.5.2.1 关于 ContextCommand

ContextCommand 表示快捷菜单的命令，类型为 RibbonMenu.ContextCommand
通过快捷菜单的 Items 集合，可以获得其中指定名称的命令。
例如：

```
ContextMenus("Cell").Items("Copy").Enabled = False
```

上面的代码禁用了快捷菜单中名为"Copy"的项目(命令)。
Items 是快捷菜单的默认属性，所以上面代码的最后一行，可以简化为：

```
ContextMenus("Cell")("Copy").Enabled = False
```

4.8.5.5.2.2 属性

- Name
  返回命令的名称

- Text
  返回或设置命令的标题

- Image
  返回或者设置命令的图标，小图标最好是 16*16 个像素。
  可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的

Images 子目录下,则只需指定文件名即可,否则需要包括路径。

- ToolTip
  返回或设置命令的提示

- Enabled
  逻辑型，返回或设置命令是否可用。

- Visible
  逻辑型，返回或设置命令是否可见。

- Checked
  逻辑型，返回或设置是否在命令左边显示一个选中标记

- CheckAutoToggle
  逻辑型，单击命令时，是否可以自动切换其选中标记

- Delimiter
  逻辑型，是否在该命令前显示一条分割线。

4.8.5.5.2.3 事件

4.8.5.5.2.3.1 Click

单击快捷菜单命令的时候执行。
e 参数属性：
Command：返回触发事件的快捷菜单命令

4.8.5.5.2.3.2 CheckedChanged

改变快捷菜单命令选中状态的时候执行。
e 参数属性：
Command：返回触发事件的快捷菜单命令

4.8.5.5.3 ContextCommandMenu(子菜单)

4.8.5.5.3.1 关于 ContextCommandMenu

ContextCommandMenu 表示子菜单，类型为 RibbonMenu.ContextCommandMenu。
子菜单同样通过快捷菜单的 Items 属性来获得，例如：

```
Dim fm As RibbonMenu.ContextCommandMenu = ContextMenus("Cell").Items("Filter")
```

子菜单中本身也有一个 Items，用来返回子菜单中指定名称的项目，例如：

```
Dim ft As RibbonMenu.ContextCommand = ContextMenus("Cell").Items("Filter").Items("FilterTrue")
```

Items 是子菜单的默认属性，所以上面的代码也可以简化为：

```
Dim ft As RibbonMenu.ContextCommand = ContextMenus("Cell")("Filter")("FilterTrue")
```

4.8.5.5.3.2 属性

ContextCommandMenu 包括下述属性：

- Name
  返回子菜单的名称

- Text
  返回或设置子菜单的标题

- Image
  返回或者设置子菜单的图标，小图标最好是 16*16 个像素。
  可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

- Enabled
  逻辑型，返回或设置命令是否可用。

- Visible
  逻辑型，返回或设置命令是否可见。

- Delimiter
  逻辑型，是否在该命令前显示一条分割线。

- Items
  子菜单的项目集合，用于返回指定名称的项目。

4.8.5.5.3.3 事件

4.8.5.5.3.3.1 Popup

展开子菜单的时候执行。
e 参数属性：
CommandMenu：要展开的子菜单。

4.8.5.5.3.3.2 Closed

关闭子菜单的时候执行。
e 参数属性：
CommandMenu：要关闭的子菜单。

4.8.6 Gallery

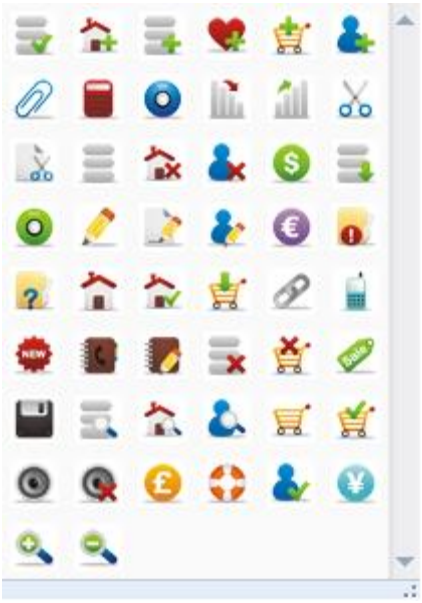
4.8.6.1 关于 Gallery

Gallery 相对于其它菜单组件，要复杂一些，所以我们单独用一章来介绍。
Gallery 英文的意思是画廊，就像画廊会挂很多画一样，菜单中 Galley 组件也可以挂很多项目(GalleryItem)。

下图是一个典型的 Gallery:

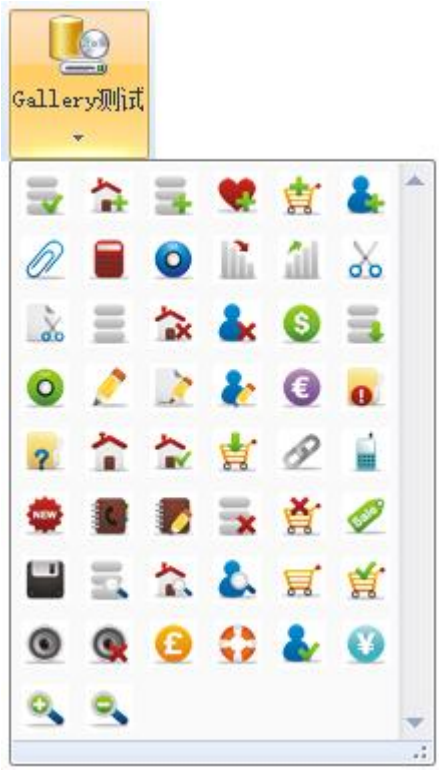


单击下拉按钮，可以显示所有项目：



在设计菜单的时候，通过设置 Gallery 的“项目宽度”和"项目高度"属性，可以决定项目的大小。

Gallery 的宽度由“可见项目数”属性决定，注意：这个属性其实设置的是横向项目可见数，而不是总的项目可见数。
如果将“默认展开”属性设置为 False，Gallery 会显示为一个按钮，单击这个按钮，会显示所有项目：



Gallery 可以显示文字，或文字和图片的组合:

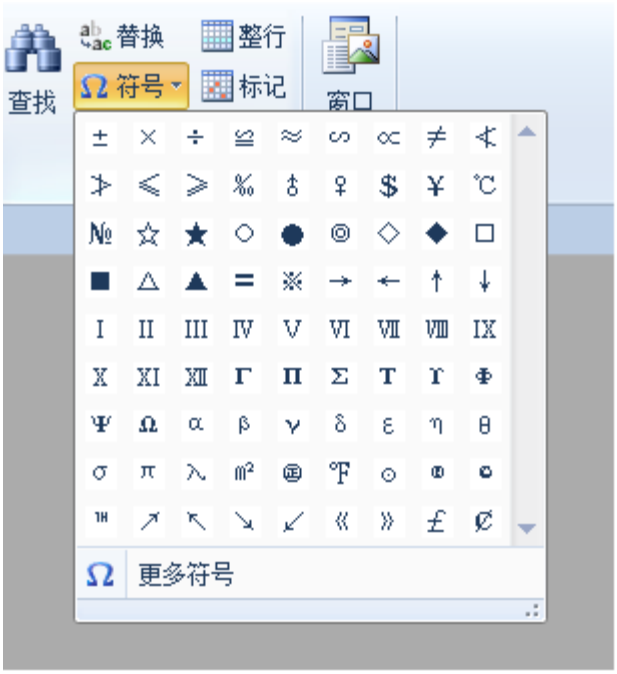




项目标题默认位于图标的下面， 如果希望标题靠右，可以将“项目标题居下”属性设置为 False。

重要提示： 必须合理设置项目宽度和项目高度属性，否则项目可能显示不完整。

Gallery 还可以包括菜单项目，例如系统菜单的插入符号功能，就是用 Gallery 设计的：



这个 Gallery 的上面用一些项目显示常用符号，下面的“更多符号”则是一个菜单命令。

4.8.6.2 两种工作模式

Gallery 有两种工作模式，一种是命令模式，另一种为非命令模式。
在命令模式下，Gallery 中的项目类似于菜单按钮，单击项目会触发 Gallery 的 ItemClick 事件。
在非命令模式下，Gallery 就像一个列表框，选择不同项目后，会触发 Gallery 的 ItemSelected 事件。
在非命令模式下，当前选定的项目会突出显示：



提示： Gallery 默认工作模式为命令模式，设计菜单的时候，将其属性“命令模式”设置为 False，可以将工作模式改为非命令模式。
ItemClick 和 ItemSelected 事件的 e 参数是相同的：
Gallery: 返回触发事件的 Gallery。
GalleryItem：返回所单击的项目。
通过在代码中判断 GalleryItem 的 Name(名称)或 Text(标题)，来执行相应的操作。

4.8.6.3 类型参考

4.8.6.3.1 Gallery

4.8.6.3.1.1 属性

属性名	说明
Name	字符型，返回组件名称
Text	字符型,返回组件标题。
ToolTip	字符型，返回或者设置组件的提示内容
Enabled	逻辑型，返回或者设置组件是否可用。
Visible	逻辑型，返回或者设置组件是否可见。
AutoHideText	逻辑型，是否允许自动隐藏标题。
LargeImage	返回或者设置组件的大图标，大图标最好是 32*32 个像素。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
SmallImage	返回或者设置组件的小图标，小图标最好是 16*16 个像素。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
TextImageRelation	设置标题和图标的相对位置，TextImageRelationEnum 枚举，可选值有： Automatic： 自动 ImageAboveText： 图片在文本之上 ImageBeforeText：图片在文本之前
Expanded	逻辑型，返回或设置组件是否处于展开状态，默认为 True。
VerticalLayout	逻辑型，返回或设置 Gallery 中项目的标题是否位于图标之下，默认为 True，如果设置为 False，则标题位于图标右边。
ItemSize	Size 类型，返回或设置 Gallery 中项目的大小。
Items	GalleryItem 集合类型，返回 Gallery 的所有项目。
MenuItems	RibbonItem 集合类型，返回 Gallery 的所有菜单项目。

4.8.6.3.1.2 事件

4.8.6.3.1.2.1 ItemClick

当 Gallery 的工作模式为命令模式时，单击 Gallery 的项目会触发 ItemClick 事件。

e 参数：
Gallery: 返回触发事件的 Gallery。
GalleryItem：返回所单击的项目。

4.8.6.3.1.2.2 ItemSelected

当 Gallery 工作方式为非命令模式时，在 Gallery 中选择不同项目时会触发 ItemSelected 事件。
e 参数：
Gallery: 返回触发事件的 Gallery。
GalleryItem：返回所单击的项目。

4.8.6.3.2 GalleryItem

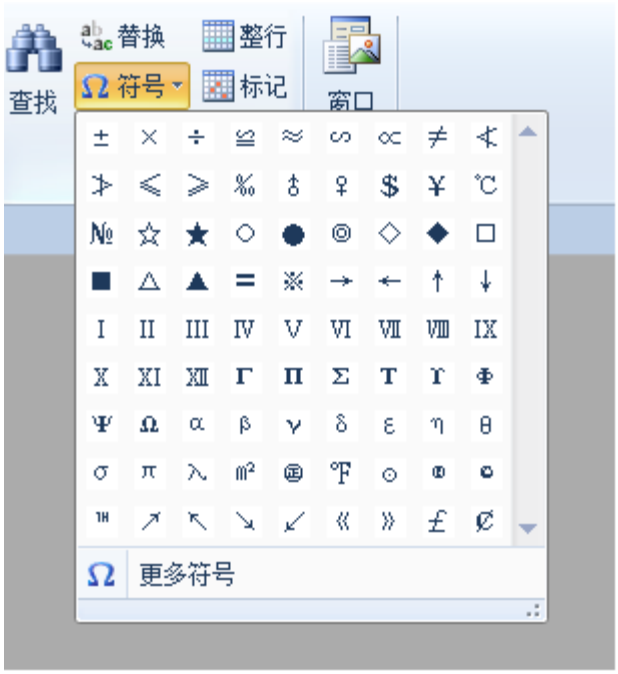
4.8.6.3.2.1 属性

GalleryItem 表示 Gallery 中的项目。
下面是 GalleryItem 的全部属性：

属性名	说明
Name	字符型，返回组件名称
Text	字符型,返回组件标题。
ToolTip	字符型，返回或者设置组件的提示内容
Enabled	逻辑型，返回或者设置组件是否可用。
Visible	逻辑型，返回或者设置组件是否可见。
Image	返回或者设置组件的图标。 可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

4.8.6.4 动态生成项目

我们可以在设计菜单的时候，事先添加好 Gallery 的项目。
我们也可以在运行过程中，动态生成 Gallery 的项目。
示例一
系统菜单中的插入符号按钮，其实就是一个 Gallery（默认展开设置成 False 了）：



这个 Gallery 的项目就是根据 Foxtable 安装目录下的文件 Symbol.txt 动态生成的。

我们看一下系统菜单是如何实现这个功能的，步骤：

1、在菜单中插入一个 Gallery，“默认展开”设置为 False，标题设置为“符号”，项目宽度和项目高度均设置为 24，并指定好小图标。

2、设置 Gallery 的 MouseEnter 事件代码为：

```
If e.Gallery.Items.Count = 0 Then '如果还没有生成项目
    Dim SymbolFile As String =  ApplicationPath & "symbol.txt"
    If FileSys.FileExists(SymbolFile) Then
        Dim Str1 As String = FileSys.ReadAllText(SymbolFile, System.Text.Encoding.Default)
        For i As Integer = 0 To Str1.Length -1
            Dim itm As New RibbonMenu.GalleryItem("Symbol" & i)
            itm.Text = Str1(i)
            e.Gallery.Items.Add(itm)
        Next
    End If
End If
```

需要注意的是，如果Gallery的“默认展开”属性为True，则不能在MouseEnter事件生成项目，应该在菜单的Load事件编写代码生成项目：

```
Dim Gallery As RibbonMenu.Gallery = RibbonTabs("Common").Groups("Edit").Items("Symbol")
Dim SymbolFile As String =  ApplicationPath & "symbol.txt"
If FileSys.FileExists(SymbolFile) Then
    Dim Str1 As String = FileSys.ReadAllText(SymbolFile, System.Text.Encoding.Default)
    For i As Integer = 0 To Str1.Length -1
        Dim itm As New RibbonMenu.GalleryItem("Symbol" & i)
        itm.Text = Str1(i)
```

```
    Gallery.Items.Add(itm)
  Next
End If
```

3、设置Gallery的ItemClick事件代码为：

```
SendKeys.Send(e.GalleryItem.Text)
```

示例二

希望在菜单中动态生成下面的国家列表，单击某个国家，即可筛选出这个国家的数据：



设计步骤：

1、首先我们应该准备好所有的国旗图标，为了避免和图片混淆，在项目文件夹中新建一个名为"Flags"的文件夹。
2、所有国旗图标的文件名都和对应的国家一致，例如： 中国.ico，美国.ico
3、在菜单中新建一个Gallery，“项目标题居下”属性设置为False，项目宽度设置为62，高度设置为20，这里假定国旗图标的尺寸是16*16
4、在菜单的Load事件加上代码：

```
Dim Gallery As RibbonMenu.Gallery = RibbonTabs("功能区1")("功能组1")("Gallery1")
For Each File As String In FileSys.GetFiles(ProjectPath & "Flags")
    Dim itm As New RibbonMenu.GalleryItem(File)
    Dim txt As String = Filesys.GetName(File)
    itm.Image = GetImage(File)
    itm.Text = txt.SubString(0,txt.Length - 4)
    Gallery.Items.Add(itm)
Next
```

5、最后将Gallery的ItemClick事件代码设置为：

```
Tables("表 A").Filter = "国家  = '" & e.GalleryItem.Text & "'"
```

4.8.7 实战演练

4.8.7.1 关于实战演练

普通的菜单设计，其实很简单，没有必要再进行什么实战演练，所以接下来的任务都较为特殊。
CaseStudy 目录下有一个“实战菜单.table”文件，该文件的菜单是基于系统菜单模板“System.Ribbon”生成的，我们接下来的任务就是给这个菜单加上一些特殊的功能。
增加的功能都大部分位于“实战演练”和“实战演练二”两个功能区中。
在设计菜单之前，应用 System.Ribbon，然后在此基础上增加自己的功能，是个不错的主意，毕竟从头设计一个菜单可不容易。
需要注意的是，大家在查看“实战菜单.table”的菜单时候，其系统事件中的代码既有因为实战演练需要而增加的，也有从 System.Ribbon 中继承的，为便于大家学习，我将实战演练的代码放在最前面，并注明每一段代码针对的是第几个任务。
System.Ribbon 作为系统菜单模板，实现了很多特殊的功能，例如自动计算，自动筛选、打印设置等等，所以其系统事件的代码较为复杂，大家无需去研究。

4.8.7.2 任务一

我们需要一个如下图所示的插入符号功能组，当然你可以通过菜单设计器来一个一个设计按钮，但是本次任务不是学习设计按钮，而是学习如何通过代码来生成菜单组件。



既然要动态生成菜单组件，我们先介绍定义一个新的菜单组件的语法：

```
Dim 变量名 As 组件类型(组件名称)
```

例如：

```
Dim g As New RibbonMenu.Group("Symbol")
```

表示定义一个新的功能组，功能组的名称为"Symbol"。

本任务的设计步骤：

首先在菜单的系统事件 Load 中加入如下代码：

```
Dim s As String = "±×÷≦≈∽∝≠≮≯≧‰♂♀＄￥℃№☆★○●◎◇◆□■△▲"
Dim g As New RibbonMenu.Group("Symbol") '定义一个功能组，功能组的名称设为"Symbol"
g.Text = "符号" '设置功能组的标题
```

```
RibbonTabs("实战演练").Groups.Add(g) '将功能组添加到实战演练功能区
For i As Integer = 0 To s.Length - 1
    Dim b As New RibbonMenu.Button("Symbol" & i) '定义一个标准按钮
    b.Text= s.Chars(i) '设置按钮的标题
    g.Items.Add(b) '将按钮添加到功能组中
Next
```

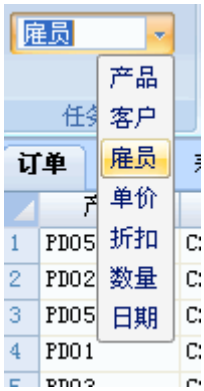
然后在全局 Click 事件中加入如下代码：

```
'如果组件的名称是以 Symbol 开头
If e.Sender.Name.StartsWith("Symbol") Then
    '那么将组件的标题插入到选定单元格中
    CurrentTable.Value = e.Sender.Text
End If
```

这样符号功能组和按钮都是通过代码自动生成的，单击某个符号按钮，即可在当前单元格插入该符号。

4.8.7.3 任务二

我们希望在菜单中增加一个组合框，该组合框要具备两个功能：
1、自动显示当前列的名称
2、自动列出当前表所有的列名供选择定位。



如果某个数据表有很多列，那么这个功能是很实用的，因为可以快速定位到某一列，而无需左右滚动。

设计步骤

首先在实战演练功能区中增加一个功能组"任务二"，在该功能组中增加一个组合框"ColList"。
然后设置该组合框的 ChangeCommitted 事件代码为：

```
Dim ColName As String = e.ComboBox.Text '获得组合框的内容
With CurrentTable
    If .Cols.Contains(ColName) Then '如果当前表包括输入名称的列
        .Select(.RowSel, .Cols(ColName).Index) '那么选择该列。
    End If
End With
```

然后在系统事件 CurrentTableChanged 加入如下代码:

```
With RibbonTabs("实战演练")("任务二")("ColList")
    .Items.Clear '清除原有项目
    For Each c As Col In CurrentTable.Cols '遍历当前表中的列
        '定义一个标准按钮，按钮的名称等于列名称。
        Dim b As New RibbonMenu.Button(c.Name)
        b.Text = c.Name '按钮的标题也等于列名称
        .Items.Add(b)
    Next
End With
```

最后在系统事件 AfterSelChange 加入如下代码：

```
RibbonTabs("实战演练")("任务二")("ColList").Text = CurrentTable.Cols(CurrentTable.Colsel).Name
```

完成上述工作后，切换一下表或者重新打开文件，本功能即可生效。
为什么要切换表？ 很简单，因为更新组合框的代码在 CurrentTableChanged 事件中。

4.8.7.4 任务三

本次任务的目的是限制某些用户的权限。

我们增加一个用户“录入员”，如果该用户登录，隐藏“数据表”和“管理项目”两个功能区，因为这两个区有涉及结构修改的命令。
第一步当然是增加一个名为“录入员”的用户，然后在系统事件 LoadUserSetting 加入如下代码：

```
If User.Name = "录入员" Then
    RibbonTabs("Table").Visible = False
    RibbonTabs("Project").Visible = False
Else
    RibbonTabs("Table").Visible = True
    RibbonTabs("Project").Visible = True
End If
```

设计完成后，请切换到录入员用户，看看是否已经实现设计要求。

4.8.7.5 任务四

设计一个根据需要动态加载数据的按钮。
我们用“订单 2”表来测试，希望打开项目的时候，该表不加载任何数据，由用户选择加载哪一个时间段的数据。

首先要做的是在项目的 BeforeLoadInnerTable 事件中设置如下代码：

```
If e.DataTableName = "订单 2" Then
    e.Filter = "False"
End If
```

这样打开项目之后，“订单 2”表不会加载任何数据。

然后在菜单中增加一个的菜单按钮，标题设为“加载数据”，给该菜单按钮的项目集合中增加两个标准按钮：

名称	标题
Q1	一季度
Q2	二季度

Q1 的代码设为：

```
DataTables("订单 2").LoadFilter = "[日期] Between #1999-1-1# And #1999-2-28#"
DataTables("订单 2").Load()
```

Q2 的代码设为：

```
DataTables("订单 2").LoadFilter = "[日期] Between #1999-3-1# And #1999-5-31#"
DataTables("订单 2").Load()
```

这样打开项目后，单击 Q1 和 Q2，分别会加载第一季度和第二季度的数据。

4.8.7.6 任务五

本次任务的目的是给菜单加入日期筛选功能：



单击筛选按钮，即可筛选出开始日期和结束日期之间的数据。

设计步骤

在菜单中，增加一个功能组，名称为"任务五"，在功能组中增加一个工具栏，名称为"工具栏一"，然后在工具栏增加两个日期输入框(DatePicker)，一个标准按钮：

类型	名称	标题	图标

日期输入框	StartDate	开始日期：	
日期输入框	EndDate	结束日期：	
标准按钮	Filter		Filter.Ico

将 Filter 按钮的 Click 事件设为：

```
Dim StartDate As Date
Dim EndDate As Date
With RibbonTabs("实战演练")("任务五")
    StartDate = .Items("StartDate").Value
    EndDate = .Items("EndDate").Value
End With
Tables("订单").Filter = "[日期] >= #" & StartDate & "# And [日期] <= #" & EndDate & "#"
```

4.8.7.7 任务六

给实战演练功能区加上所有筛选与排序功能。

系统菜单在日常工作中功能区中，已经提供了非常完善的筛选与排序功能，要自己在"实战演练"功能区实现同样的功能，可不容易。
既然有现成的，我们没必要重新设计，只需在系统事件 Load 中加入如下代码：

```
RibbonTabs("实战演练").Groups.Add(RibbonTabs("Common").Groups("SortAndFilter"))
```

是不是出乎意料的简单？ 没错，所有的功能组和单个菜单组件都可以随意移动，甚至可以同时在多个地方出现。
利用这个功能，你可以在 Load 事件中，将常用的按钮添加到快速访问栏和配置栏中，而无需重新设计。
需要注意的是，快速访问栏和配置栏中只能显示小图标，所以如果某个按钮计划加入到其中，请同时设置该按钮大小图标。

如果并不想将“排序与筛选”整个功能组加入到“实战演练”功能区，而只希望加入其中部分组件，可以参考下面的代码：

```
Dim ng As New RibbonMenu.Group("Filter") '定义新的功能组
ng.Text = "筛选" '设置功能组标题
RibbonTabs("实战演练").Groups.Add(ng) '将功能组加入实战演练功能区
With RibbonTabs("Common").Groups("SortAndFilter")
    ng.Items.Add(.Items("Filter")) '添加筛选按钮
    ng.Items.Add(.Items("SelectFilter")) '添加选择按钮
```

```
    ng.Items.Add(.Items("Advanced")) '添加编辑按钮
    ng.Items.Add(.Items("UnFilter")) '添加撤销按钮
    ng.Items.Add(.Items("Separator")) '添加分隔条
    ng.Items.Add(.Items("Toggle")) '添加切换按钮
End With
```

你可以将上面的代码复制到命令窗口执行，看看会发生什么。

4.8.7.8 任务七

在开始菜单中始终列出几个重要文件。

在默认情况下，开始菜单的右分区会按时间顺序列出最近打开过的文件，如果你有几个非常重要的文件，希望始终能够列出来，不管最近是否使用过，而且列出位置固定，该怎么办？

很简单,在程序菜单的右侧项目集合中，加入几个标准按钮：

名称	标题	Click 事件
Button1	工资管理系统	Syscmd.Project.Open("C:\Data\工资管理.Table")
Button2	人事管理系统	Syscmd.Project.Open("C:\Data\人事管理.Table")
Button3	销售管理系统	Syscmd.Project.Open("C:\Data\销售管理.Table")
Button4	客户管理系统	Syscmd.Project.Open("C:\Data\客户管理.Table")

Click 事件请根据具体的文件名称和路径编写，这是本例的设计效果：



4.8.7.9 任务八

尝试使用状态按钮和检查框。

实战演练功能区已经满了，我们增加一个新的功能区"实战演练二"。

首先在实战演练二功能区增加一个状态按钮，用以切换查阅模式，同时增加一个检查框，实现同样的功能。
用两个组件实现同样的功能，没有实际意义，只是为了便于大家学习。
两个菜单组件的属性设置如下：

类型	名称	标题
检查框	chkBrowse	查阅模式
状态按钮	staBrowse	查阅模式

然后将两个组件的 Click 事件代码均设为：

```
DataTables.AllowEdit = Not DataTables.AllowEdit
```

最后在系统事件 SystemIdle 中增加如下代码：

```
RibbonTabs("实战演练二")("任务八")("chkBrowse").Checked = Not DataTables.AllowEdit
RibbonTabs("实战演练二")("任务八")("staBrowse").Pressed = Not DataTables.AllowEdit
```

为什么在 SystemIdle 事件中更新两个组件的状态，而不是在 Click 事件中？ 这是因为 DataTables 的 AllowEdit 属性还可能通过其他方式修改，例如配制栏中的查阅模式按钮。
而且 SystemIdle 是在系统空闲的时候执行，不会影响运行效率。

现在单击两个组件中的任何一个，即可切换项目的查阅模式，而且两个组件包括配置栏中的"查阅模式"按钮，都会自动更新状态。

本任务设计结果：



4.8.7.10 任务九

在菜单中加上简单的交叉统计功能。

增加一个功能组"任务九"，在该功能组中增加四个组件：

类型	名称	标题	大图标
组合框	HGroup	水平分组	
组合框	VGroup	垂直分组	
组合框	TotalOn	统计列	

标准按钮	Build	统计	CrossTable32.ico

分别给 HGroup 和 VGroup 两个组合框增加三个列表项目：产品、客户、雇员
给 TotalOn 组合框设置两个列表项目：数量、金额。
Build 按钮的 Click 事件代码：

```
Dim HGroup As String
Dim VGroup As String
Dim TotalOn As String
'获取输入内容
With RibbonTabs("实战演练二")("任务九")
    HGroup = .Items("HGroup").Text
    VGroup = .Items("VGroup").Text
    TotalOn = .Items("TotalOn").Text
End With
With Tables("订单").Cols
    If HGroup >"" AndAlso .Contains(HGroup) Then '如果已经输入有效水平分组
        If VGroup >"" AndAlso .Contains(VGroup) Then '如果已经输入有效垂直分组
            If TotalOn >"" AndAlso .Contains(TotalOn) Then '而且统计列有效
                Dim b As New CrossTableBuilder("统计表",DataTables("订单"))
                b.HGroups.AddDef(HGroup) '添加水平分组
                b.VGroups.AddDef(VGroup) '添加垂直分组
                b.Totals.AddDef(TotalOn) '添加统计列
                b.Build '生成统计表
                MainTable = Tables("统计表")
            End If
        End If
    End If
End With
```

本次任务的设计效果：



分别选择水平分组、垂直分组、统计列，单击右侧统计按钮，即可自动生成一个交叉统计表。
其实完全可以做得更好一些，例如组合框自动列出当前表的列，而不是事先设定，可以选择统计类型以及日期列的分组模式等等。
只是本课的任务是学习菜单设计，所以全部用最简单的例子来说明，以免分散大家的注意力。

4.8.7.11 任务十

下拉菜单根据条件显示不同项目。

大家肯定用过系统菜单中的筛选功能，这个按钮很智能，能够根据当前列的数据类型以及当前单元格的内容自动显示最合适的筛选项目。
要实现这样的功能很容易，我们通过一个最简单的例子来实际操作一下：

首先新增一个菜单按钮，给菜单按钮的项目集合中加上四个标准按钮，分别是：

名称	标题
Equal	等于
String	字 符筛选
Numeric	数 值筛选
Date	日 期筛选

然后给菜单按钮的 MouseEnter 事件设置如下代码：

```
Dim m As RibbonMenu.MenuButton = e.MenuButton
Dim t As Table = CurrentTable
If t.Current Is Nothing Then
    Return
End If
If t.Current.IsNull(t.ColSel) Then
    m.Items("Equal").Text = "等于空白"
Else
    m.Items("Equal").Text = "等于" & t.Text
End If
m.Items("String").Visible = t.Cols(t.Colsel).IsString
m.Items("Numeric").Visible = t.Cols(t.Colsel).IsNumeric
m.Items("Date").Visible = t.Cols(t.Colsel).IsDate
```

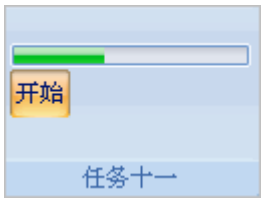
经过上述设置，Equal 按钮会自动根据单元格内容调整标题，而 String 按钮只有在当前列是字符型的时候才显示，Numeric 按钮只有在当前列是数值型的时候显示，Date 当然是当前列是日期型的时候才显示。

4.8.7.12 任务十一

```
Dim vl As Double
```

```
Dim pb As RibbonMenu.ProgressBar
pb = RibbonTabs("实战演练二").Groups("任务十一").Items("进度条 1")
For i As Integer = 1 To 10000
    vl = i / 3
    If i Mod 100 = 0 Then
        pb.Value = i / 10000 * 100
        Application.DoEvents
    End If
Next
```

单击开始按钮，进度条将显示任务完成进度：



4.8.7.13 任务十二

在前面的任务六中，我们已经知道功能组和单个菜单组件可以被随意移动，甚至可以同时在多个地方出现。
快捷菜单命令也有类似的特性，同一个快捷菜单命令可以同时出现在多个快捷菜单中。

例如打开示例文件“实战菜单.Table”，进入菜单设计器后，可以发现快捷菜单 Range 是空的，里面并没有任何命令。
可实际使用的时候，选定多个单元格，右击后弹出的快捷菜单 Range 是有命令的，这是为什么呢？
这是因为我们在菜单的 Load 事件中加入了下面一段代码：

```
'任务十二
Dim mnu1 As RibbonMenu.ContextMenu = ContextMenus("Cell")
Dim mnu2 As RibbonMenu.ContextMenu = ContextMenus("Range")
mnu2.Items.Add(mnu1.Items("Cut"), False)
mnu2.Items.Add(mnu1.Items("Copy"))
mnu2.Items.Add(mnu1.Items("Paste"))
mnu2.Items.Add(mnu1.Items("Ascending"))
mnu2.Items.Add(mnu1.Items("Descending"))
mnu2.Items.Add(mnu1.Items("UnSort"))
```

上面的代码在菜单生成后，会将 Cell 快捷菜单中的部分命令加入到 Range 快捷菜单中，避免了重复设计。
不仅仅是快捷菜单命令，子菜单也可以被同时加入到多个快捷菜单或其它子菜单中。
提示：
Add 方法的第二个参数是可选的，用于决定是否在新加入的命令前显示一条分割线，如果省略此参数，则保持现有设置不变。

4.8.8 一点改进

以前 Foxtable 的菜单设计器需要逐层打开和关闭，如果菜单层数较多，对开发效率影响会很大。Foxtable 2018 对此做出了改进，在任意层级的设计窗口，你可以单击预览按钮，设计中的菜单将直接在主窗口生效，方便用户进行测试，同时会出现一个浮动窗口，你可以：

1、单击浮动窗口中的“设计菜单”按钮，将直接回到之前的菜单设计窗口进行修改，无需重新逐层打开。

2、单击浮动窗口中的“保存菜单”按钮，将直接保存所有修改，无需回到菜单设计器保存。

此外，在任意层级的菜单设计窗口，单击“应用”按钮将关闭所有层级的菜单设计窗口，并保存菜单，同时在主窗口应用设计好的菜单。

4.8.9 关于 TopToolBar

TopToolBar(上工具栏)在主窗口的窗口标题栏右侧显示。

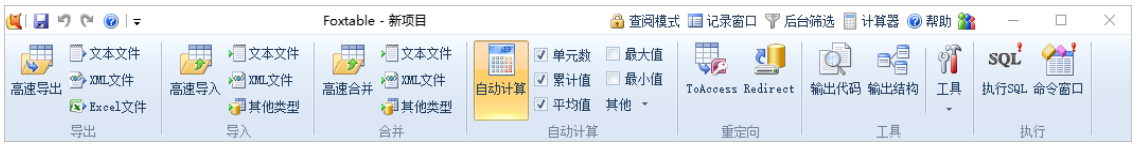
TopToolBar 使用的场合不多，多数时候我们会选择用 ConfigBar(配置栏)。

但是如果隐藏了功能区标题，配置栏同样会被隐藏，此时可以考虑用 TopToolBar。

例如新建一个项目，执行下面的代码：

```
RibbonTabs.HideTabHeaderRow = True '隐藏功能区标题
'将ConfigBar的项目复制到TopToolBar
For Each itm As RibbonMenu.RibbonItem In ConfigBar.Items
    TopToolBar.Items.Add(itm)
Next
```

执行后的菜单显示为：



当然你也可以在设计菜单的时候，就将配置栏中的项目转移到上工具栏中。

提示：TopToolBar的属性和方法与ConfigBar完全一样，这里就不重复介绍了。

4.8.10 关于 BottomToolBar

BottomToolBar(下工具栏)位于功能区的右下方：



在有功能区的情况下，在这个位置显示工具栏，看不出有什么优势，所以 BottomToolBar 比 TopToolBar 更不常用。
但是后一个特殊场合，当你不想设计 Ribbon 风格的菜单，想用传统菜单时，BottomToolBar 就派上用场了：



关于传统菜单的设计，参考：设计传统菜单
你也许会奇怪，BottomToolBar 不是显示在功能区的右下方吗？ 上面的 BottomToolBar 怎么显示在左侧呢？
很简单，我们可以用“挤”的方式，在菜单的 Load 事件中加上代码：

```
Dim lb As new RibbonMenu.Label("Space")
lb.Text = new String(" ",500)
BottomToolBar.Items.Add(lb)
Qat.Visible = False
```

上述代码会在BottomToolBar的右侧插入一个包括500个空格的标签，这样其他所有项目都被挤到最左侧去了。

提示：BottomToolBar的属性和方法与ConfigBar完全一样，这里就不重复介绍了。

4.8.11 关于 TabGroup

TabGroup(功能区组)可以对功能区进行分组，例如下图的菜单包括两个 TabGroup，分别是“开发设计”和“其 它”：



开发设计这个 TabGroup 包括两个功能区，分别是数据表和管理项目两个功能区，其它这个 TabGroup 则包括打印输出和杂项这两个功能区，而日常工作这个功能区则是独立的，不属于任何 TabGroup.
使用 TabGroup(功能区组)很简单：

1、在菜单设计窗口，定义好功能区组，不同的功能区组可以设置不同的背景颜色，以示区分。
2、选择某个功能区，在属性设置中，为该功能区选择一个功能区组。
TabGroup 的概念来自于 Windows 10，其真正意义并不是对功能区进行分组，而是用于突出显示动态出现的功能区。

TabGroup 编程

RibbonTabGroups 集合可以返回指定名称的 TabGroup，例如：

```
For Each rg As RibbonMenu.TabGroup In RibbonTabGroups
    Output.Show(rg.Name)
Next
RibbonTabGroups("功能区组1").Visible = False
```

TabGroup 有个 RibbonTabs 集合，用于返回该 TabGroup 包含的所有功能区，例如：

```
For Each tab As RibbonMenu.Tab In RibbonTabGroups("功能区组1").RibbonTabs
    Output.Show(tab.Name)
Next
RibbonTabGroups("功能区组1").RibbonTabs("Table").Visible = False
```

4.8.12 菜单和代码编辑器

为菜单编写事件代码的时候，免不了要引用某个菜单组件的。
这有一点麻烦，首先一个菜单可能由很多组件组成，要记住每个组件的名称不容易，而且菜单是层层嵌套的，你得记住每一层组件的名称，才可能引用改组件。
其次组件类型也很多，要记住每个类型的名称才能正确地定义组件变量，显然这也不太容易。
不过代码编辑器的菜单页，能够让你轻松解决这些问题。



菜单页列出了所有的组件，在默认情况下，当我们双击某个组件的时候，会在编辑器插入组件的名称。

页面下方的两个选项，可以改变这种默认的方式，以双击上图中的“Reset”为例：

定义引用变量

如果选择此选项，则插入：

```
Dim 变量名 As RibbonMenu.Button = RibbonTabs("Table").Groups("Column").Items("Reset")
```

显然这样我们可以快速定一个变量来引用组件“Reset”，要做的只是将字符“变量名”改为你需要的名称而已。

直接引用

如果选择此选项，则插入：

```
RibbonTabs("Table").Groups("Column").Items("Reset").
```

提高开发效率

通常我们要回到系统菜单才能进行下一步的开发，但是一旦回到系统菜单，代码编辑器列出的菜单项目就是系统菜单的项目，而不是自定义的菜单项目，这給我们的编码会带来一些麻烦，因为无法通过代码编辑器自动生成某个菜单组件的引用代码。

我们可以在设计自定义菜单的时候，另外增加一个功能区(假定名称为"设计")，这个功能区包括一些开发方面的按钮，例如项目属性、表属性、菜单设计、窗口设计等等，这样我们在使用自定义菜单的时候，依然可以进行开发，但是这个和开发有关的功能区显然不应该让开发者之外的用户使用，为此我们可以在项目事件 LoadUserSetting 加上代码：

```
If RibbonTabs.Contains("设计") Then '如果包含名为"设计"的功能区
    If user.Type = UserTypeEnum.Developer Then
        RibbonTabs("设计").Visible = True
    Else
        RibbonTabs("设计").Visible = False
    End If
End If
```

上面的代码还告诉了我们一个技巧：通过判断菜单是否存在特定名称的功能区，就能知道项目当前使用的是系统菜单还是自定义菜单；在技术支持过程中，经常会有用户问这个问题，希望大家能举一反三。

4.8.13 设计传统菜单

Foxtable 的菜单为 Office 2007 风格，虽然有不少优势，但也有相当多的用户还是喜欢传统菜单。

Foxtable 同样可以设计传统菜单的，下图是在快速访问栏（QAT）设计的传图菜单：

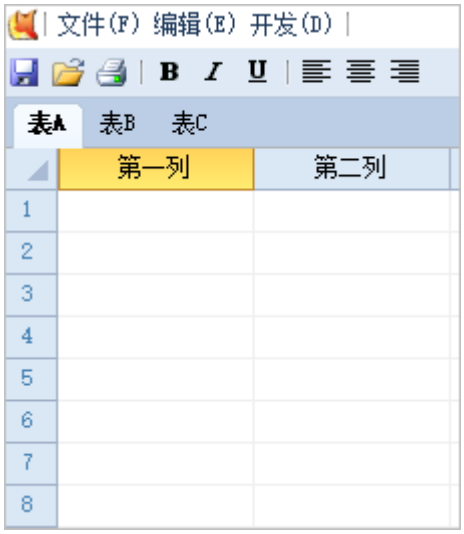


我们也可以在 TopToolBar(上工具栏)、BottomToolBar(下工具栏)或 ConfigBar(配置栏)设计传统菜单 。

例如下图是利用配置栏设计的：



我们可以组合利用各区域，例如下图中，主菜单位于快速访问栏，工具栏则位于配置栏中：



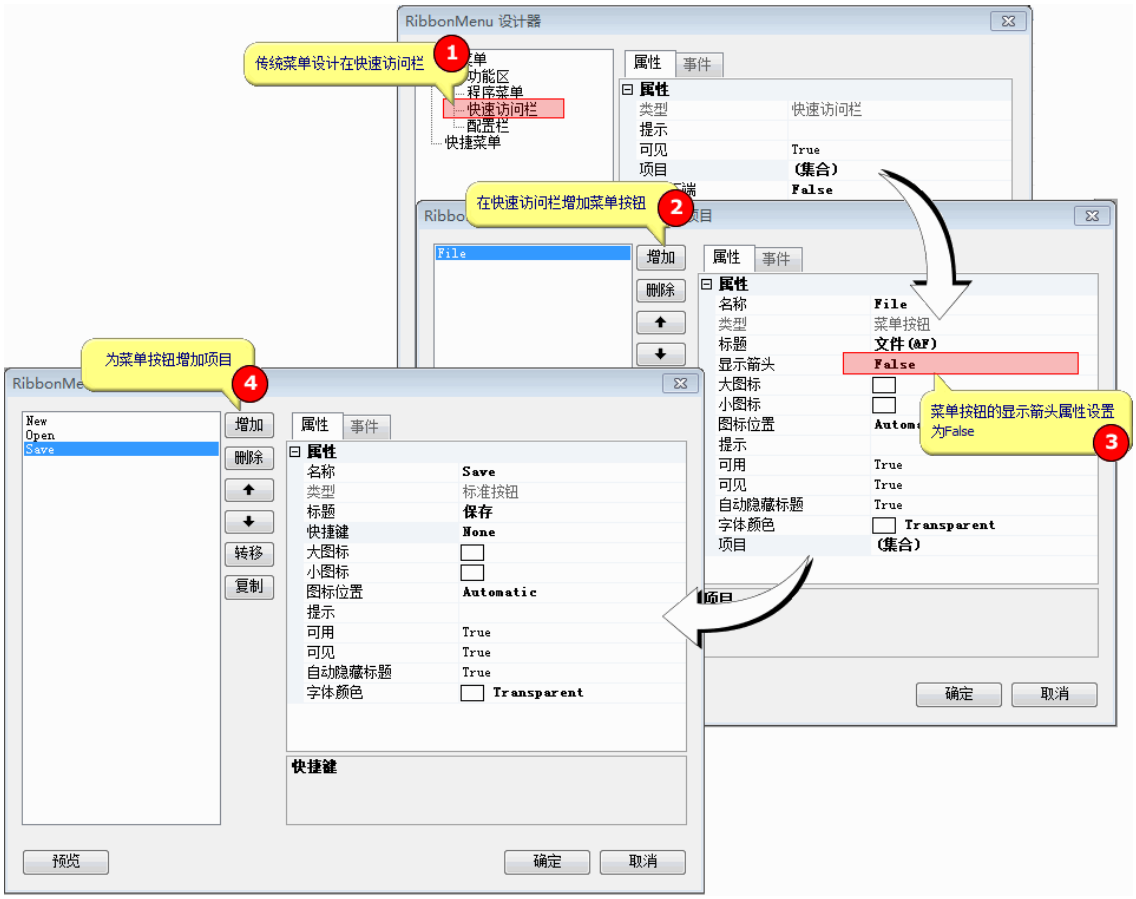
如果在 TopToolBar、BottomToolBar 或 ConfigBar 设计传统菜单，由于其默认显示在右侧，所以需要在菜单的 Load 事件中加上代码 调整其显示位置，以 ConfigBar 为例：

```
Dim lb1 As new RibbonMenu.Label("Space")
lb1.Text = new String(" ",500)
ConfigBar.Items.Add(lb1)
```

上述代码会在 ConfigBar 的右侧插入一个包括 500 个空格的标签，这样其他所有项目都被挤到最左侧去了。

传统菜单设计步骤

以在快速访问栏设计为例：

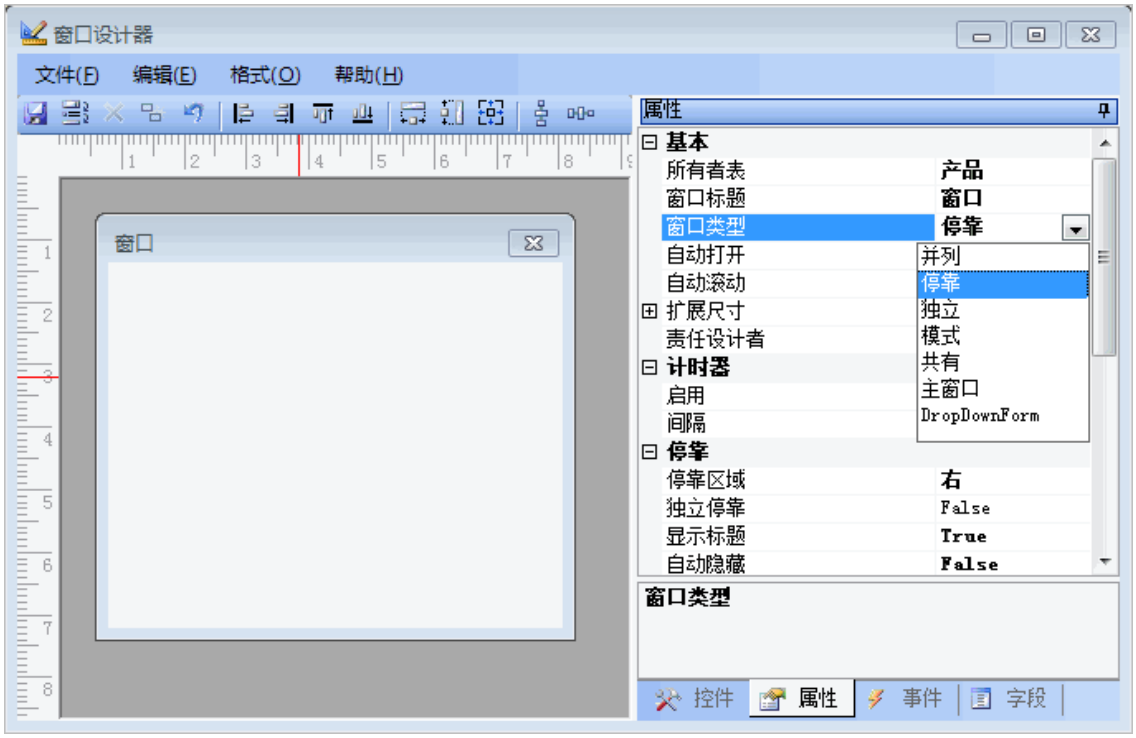


4.9 窗口设计

4.9.1 窗口基础

4.9.1.1 窗口类型

窗口有六种类型，窗口类型默认是“停靠”，在设计窗口的时候，可以通过指定窗口类型：



需要注意的是，在设计和预览的时候，各种类型的窗口是看不出差异的，只有你打开或者应用它们，才能看出差别。

建议你新建几个不同类型的窗口，然后分别打开它们，以了解一下他们的不同。

下面我们分别介绍各种类型的窗口。

主窗口

主窗口用于代替主表，最常用的信息应该在主窗口中显示，最常使用的功能也应该在主窗口中设计:



主窗口使用的频率较高，其余窗口用于辅助主窗口，用于显示不常用的信息，以及进行一些不常用的操作。

主窗口属于特定的表，通过窗口的“所有者表”属性，可以指定主窗口所属的表，属于表 A 的主窗

口，只能在表 A 打开，切换到其它表后，主窗口将不可见。

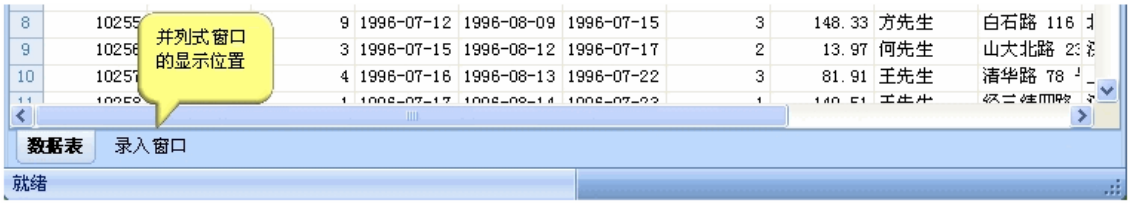
每个表只能打开一个主窗口，打开一个新的主窗口，之前的主窗口会自动关闭。

我们没有办法直接关闭已经打开的主窗口，只能通过代码关闭：

Forms("窗口名").Close()

并列窗口

并列式窗口和主表并列显示，如下图所示：



如上图所示，并列窗口的窗口标题默认显示在下方，你可以在设计窗口的时候，通过设置“标题位置”属性来改变默认的显示位置。

只有窗口类型为并列窗口的时候，才可以设置“标题位置”属性。

一个表可以打开多个并列窗口， 如果一个表打开了多个并列窗口，标题位置由最后一个打开的窗口决定。

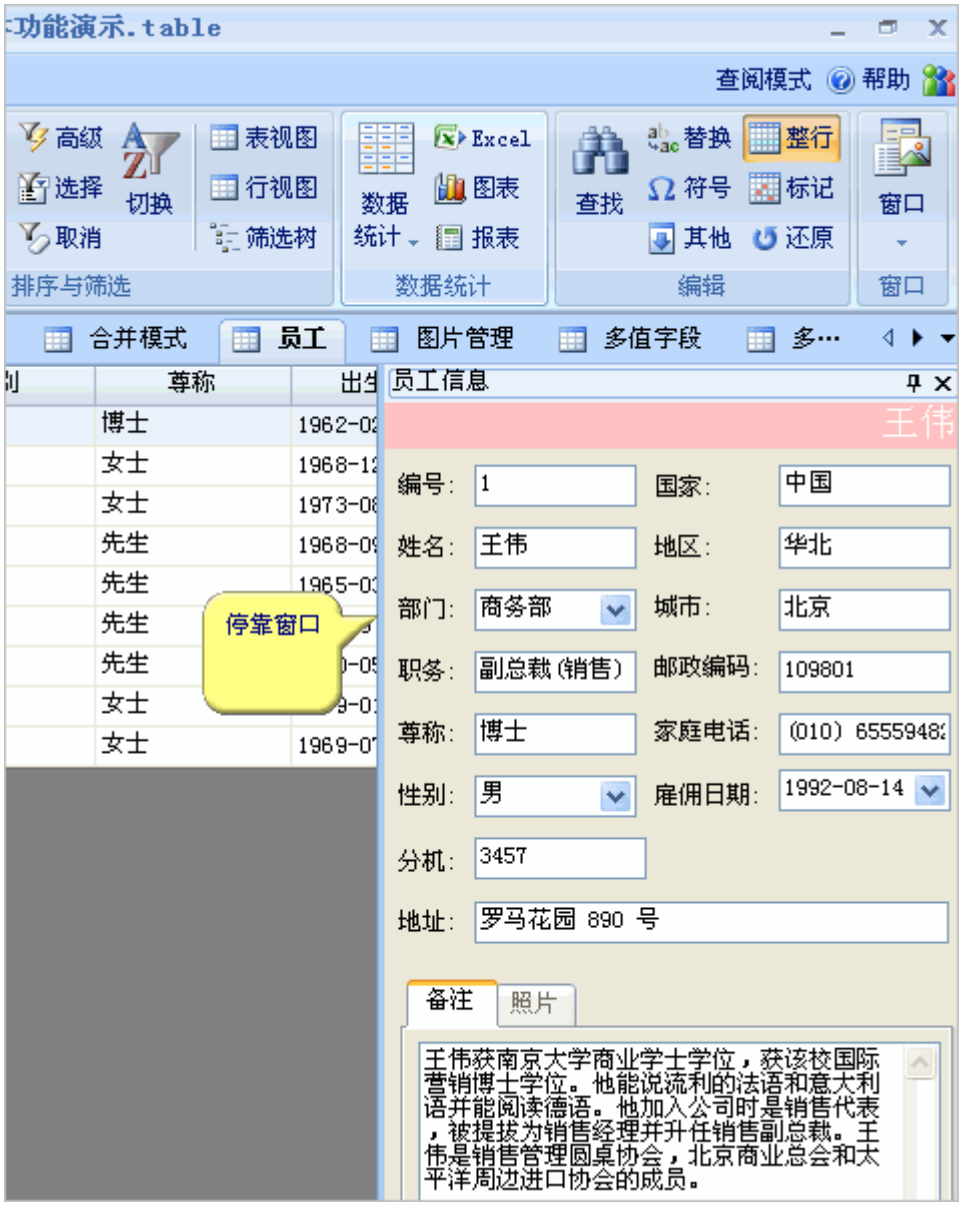
并列窗口也属于特定的表，通过窗口的“所有者表”属性，可以指定并列窗口所属的表，属于表 A 的并列窗口，只能在表 A 打开，切换到其它表后，并列窗口将不可见。

我们没有办法直接关闭已经打开的并列窗口，只能通过代码关闭：

Forms("窗口名").Close()

停靠窗口

停靠窗口与关联表和列窗口类似，停靠在主表周围：

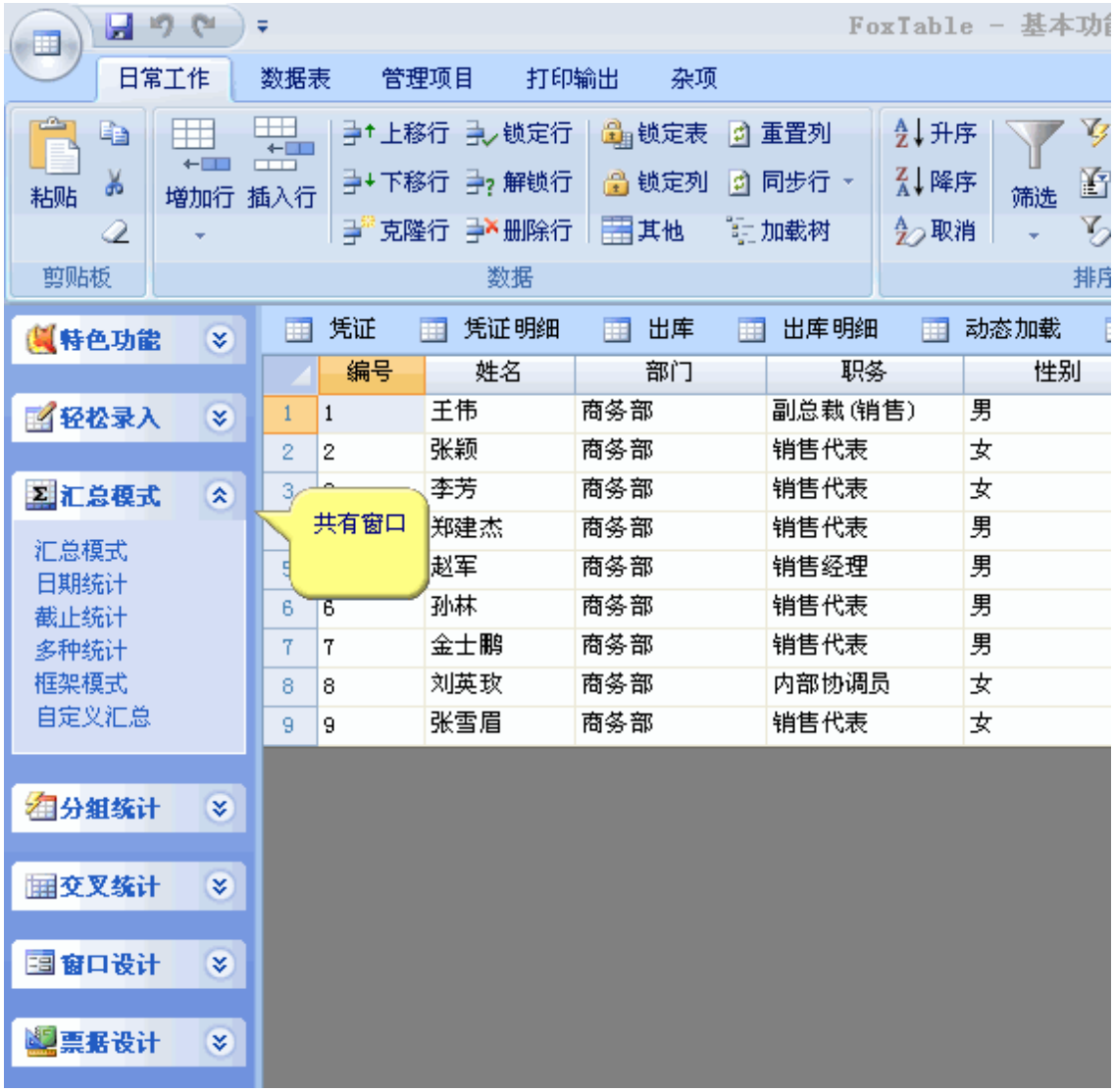


停靠窗口属于特定的表，通过窗口的“所有者表”属性，可以指定停靠窗口所属的表，属于表 A 的停靠窗口，只能在表 A 打开，切换到其它表后，停靠窗口将不可见。

共有窗口

共有窗口和停靠窗口类似，也是停靠在主表周围，但是在切换表的时候，共有窗口始终可见。

基本功能演示文件中的导航窗口，就是一个共有窗口，这样选择不同的表时，导航窗口始终可见：



独立窗口

并列、停靠和共有窗口，都是和主界面融合在一起，而独立窗口与此不同，是作为一个独立的窗口显示的:

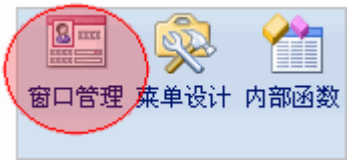


模式窗口

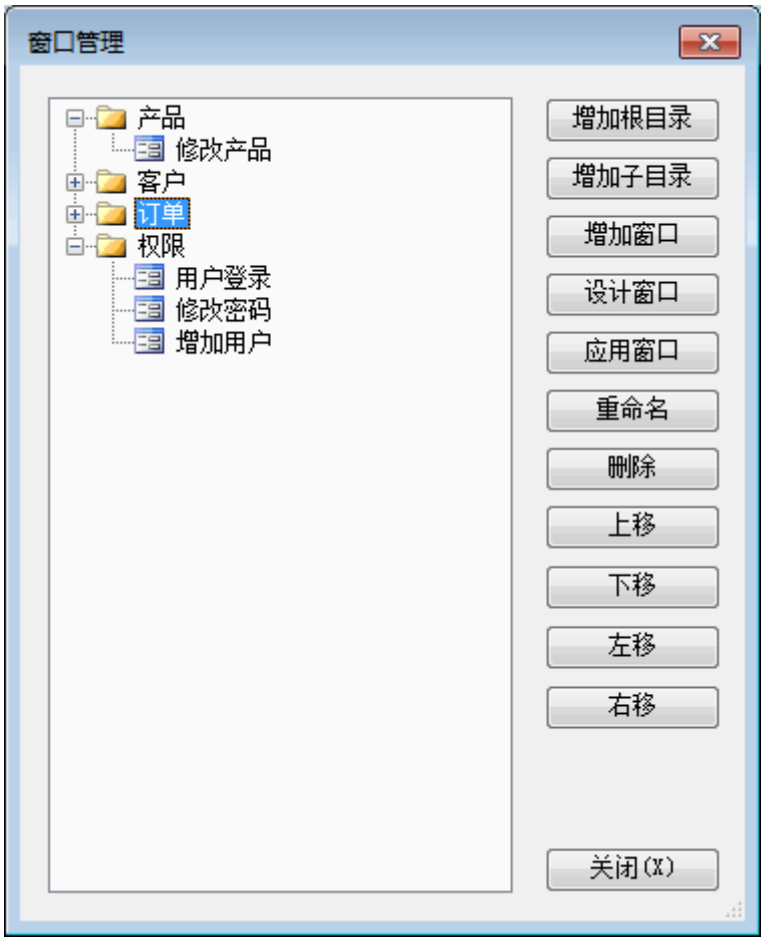
模式窗口其实也是独立窗口，只不过在关闭模式窗口之前，你不能进行任何其它的操作。

4.9.1.2 窗口管理

1、在“管理项目”功能区，单击下面的“窗口管理”按钮，可以打开窗口管理器。

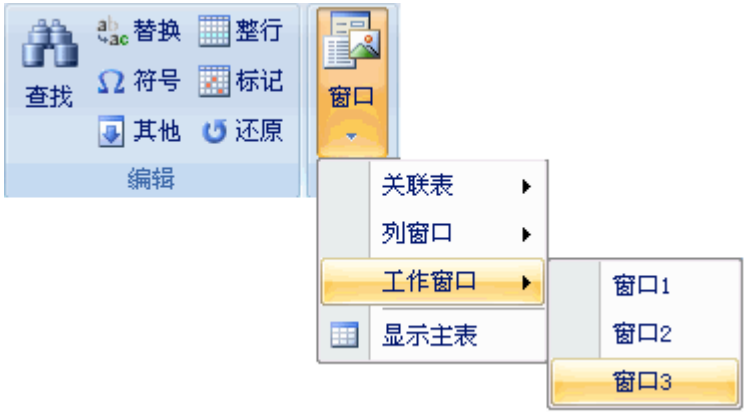


2、下图是窗口管理器，在这里可以增加、删除、重命名和设计窗口，还可以用目录树对窗口进行分类管理：



4.9.1.3 打开窗口

1、单击菜单“日常工作”中的“窗口”按钮，然后指向“工作窗口”，会列出当前表的所有窗口名，单击其中一个即可手动打开对应的窗口：



提示：窗口按钮只会列出在当前表设计的窗口，例如你在表 A 中设计的窗口，并不会在表 B 中列出来，必须回到表 A，窗口按钮才会列出这些窗口。

2、主窗口显示在主表位置，如果希望重新显示主表，可单击上图中的“显示主表”按钮。

3、如果希望打开管理项目后，能够自动打开某窗口，可以在项目事件 AfterOpenProject 中设置代码：

Forms("窗口名").Open()

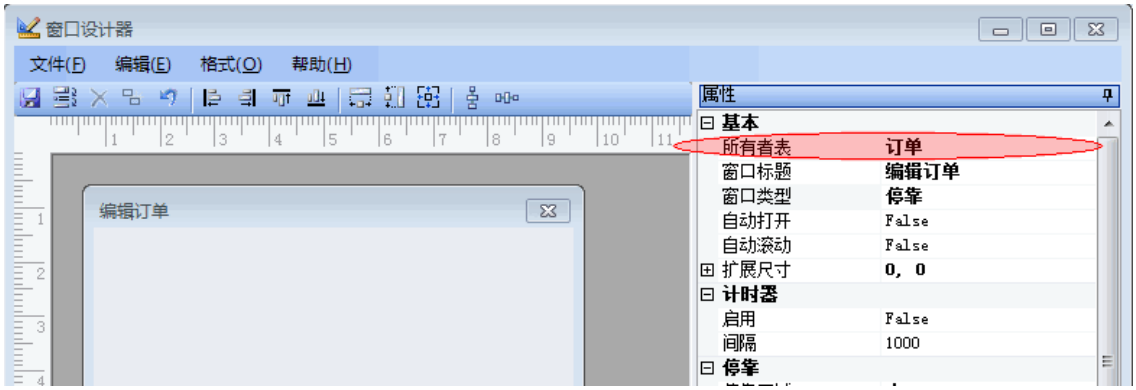
或者在设计窗口的时候，将该窗口的“自动打开”属性设为 True：



推荐在 AfterOpenProject 事件中使用代码来打开窗口。

4.9.1.4 重要概念:所有者表

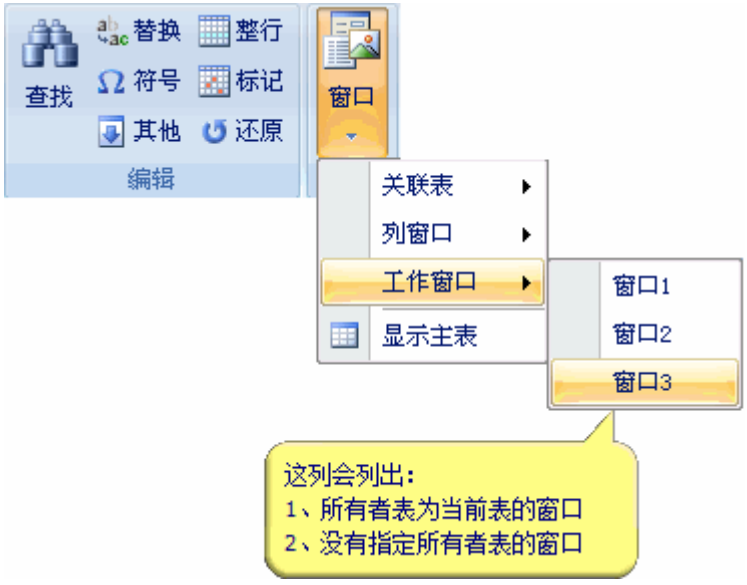
主窗口、停靠窗口、并列窗口在实际使用的时候，是属于特定表的， 所以在设计窗口的时候，必须给这类窗口指定一个所有者表：



例如将某停靠窗口的所有者表设置为“订单”，那么此窗口打开后，将停靠在“订单”表的周围。
如果你要通过代码打开主窗口、停靠窗口、并列窗口，需要在打开窗口后，切换到所有者表，否则将看不到所打开的窗口，例如：

```
Forms("窗口名").Open()
MainTable = Tables("所有者表")
```

共有窗口、独立窗口、模式窗口在实际使用的时候，不属于任何表；设计窗口的时候，可以给此类窗口指定一个所有者表，也可以不指定。
不过是否指定所有者表，会影响打开窗口菜单的显示结果：



所以对于共有窗口、独立窗口、模式窗口，通常无需指定所有者表，但是如果希望此窗口只能在选择某表后才能通过菜单打开，那么就应该将此表设置为该窗口的所有者表。

同样，如果你设计了一个窗口，但是在菜单中却找不到这个窗口，那么通常是因为这个窗口指定了一个所有者表，而当前表并非这个所有者表。

如果是通过代码，而不是通过菜单打开共有窗口、独立窗口、模式窗口，那么是否指定所有者表都不受影响。

4.9.1.5 窗口位置

本节内容仅对独立窗口和模式窗口有效。

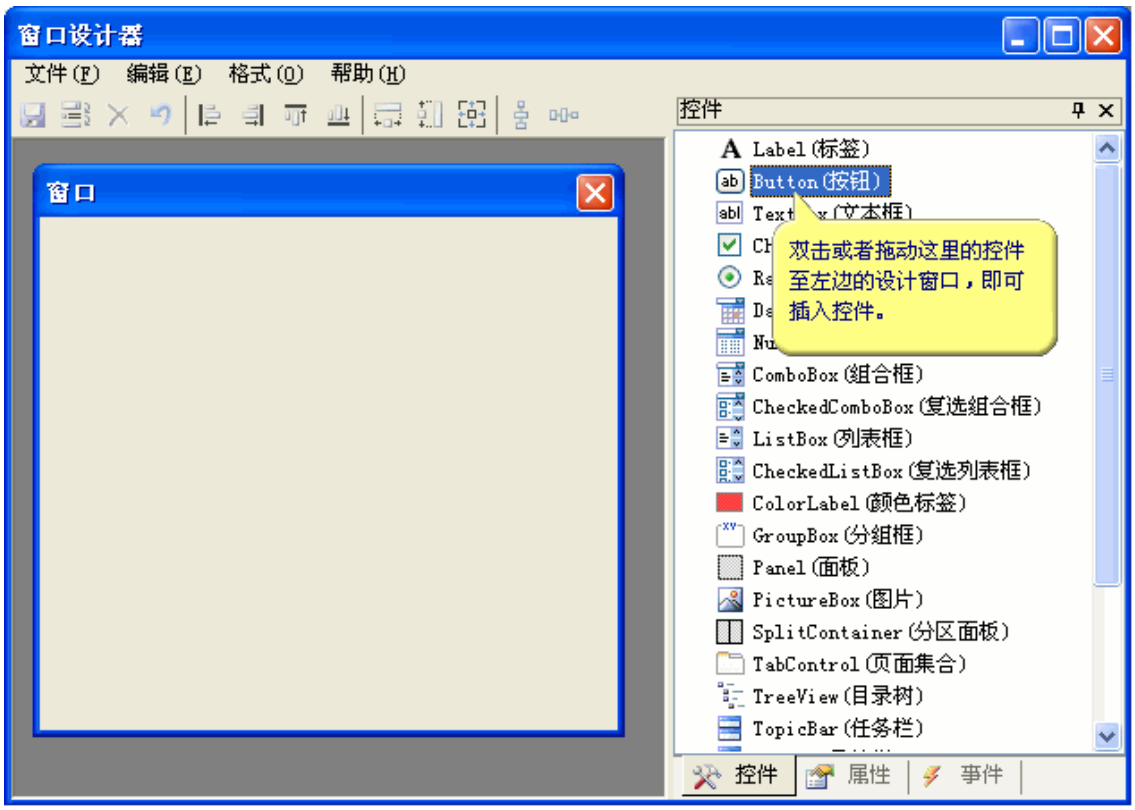
独立窗口和模式窗口的默认 位置是位于屏幕中央，如果要改变默认的位置，可以在设计窗口的时候，将窗口位置指定为“Manual”，然后设置其左上角位置：

位置	
初始位置	Manual
左	100
上	100
宽	404
高	334

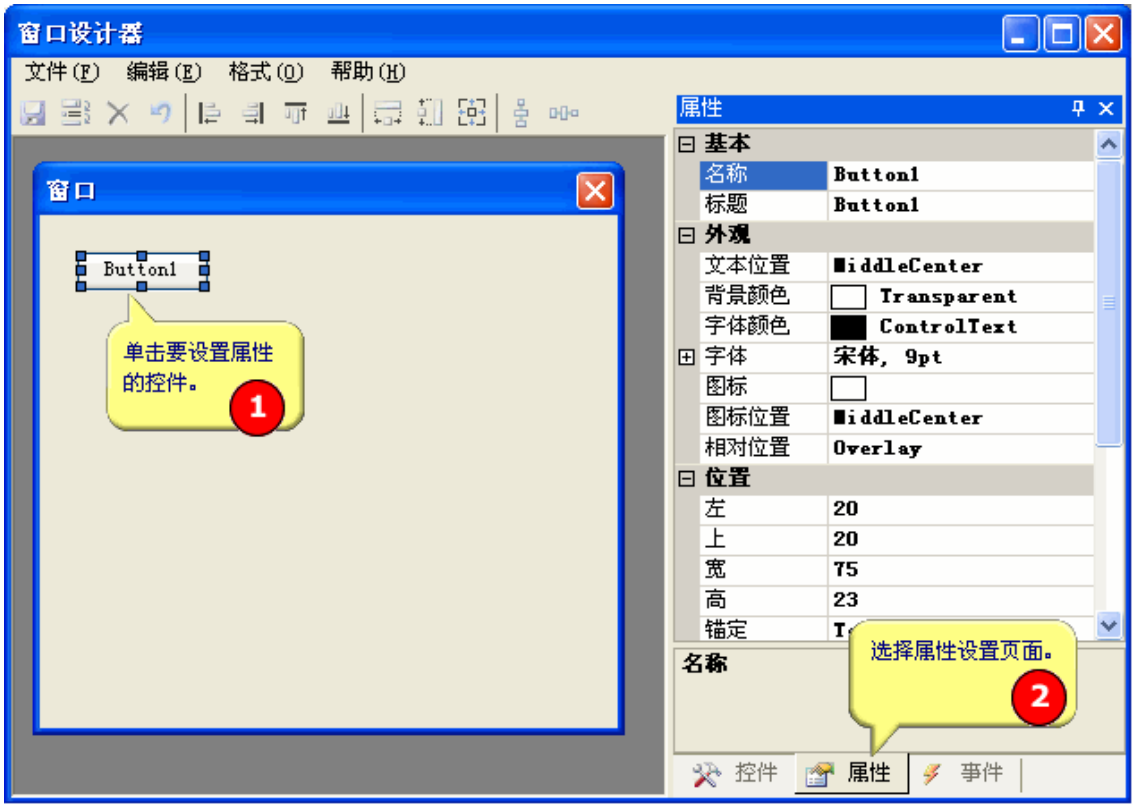
提示：只有先将窗口类型设置为独立窗口或模式窗口，才会出现以上用于设置窗口位置的属性。

4.9.1.6 设计窗口

插入控件

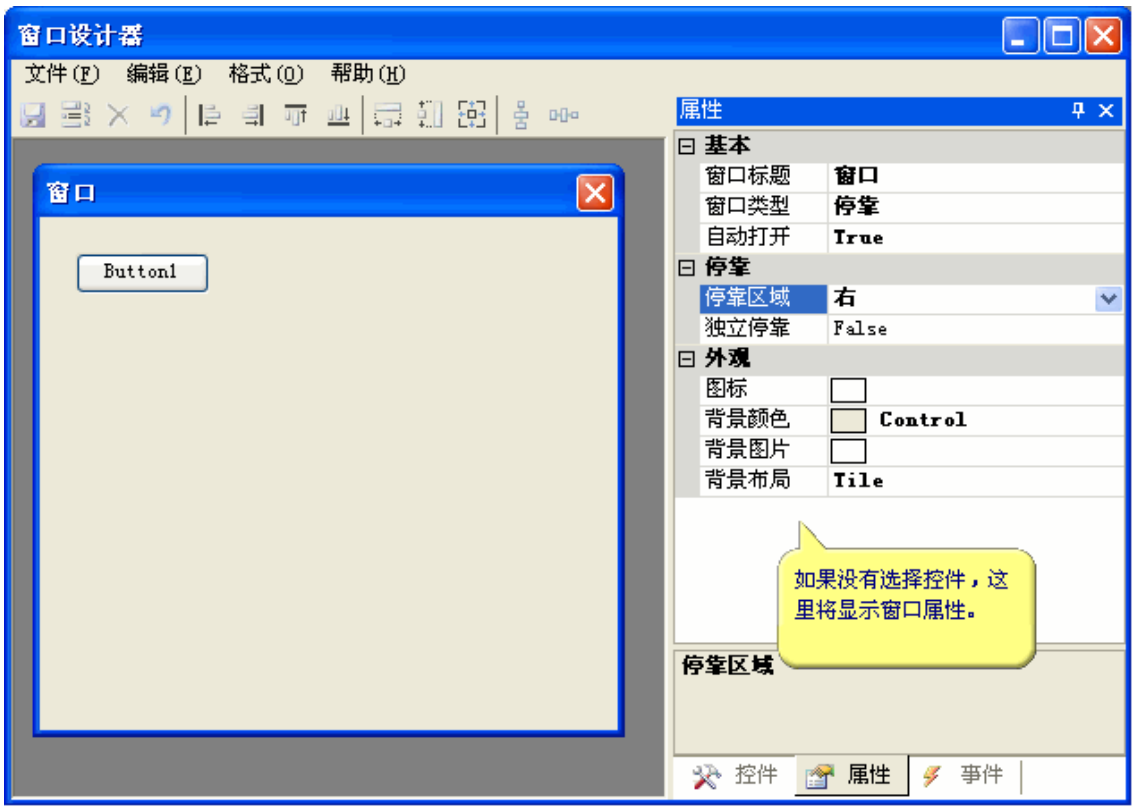


设置控件属性

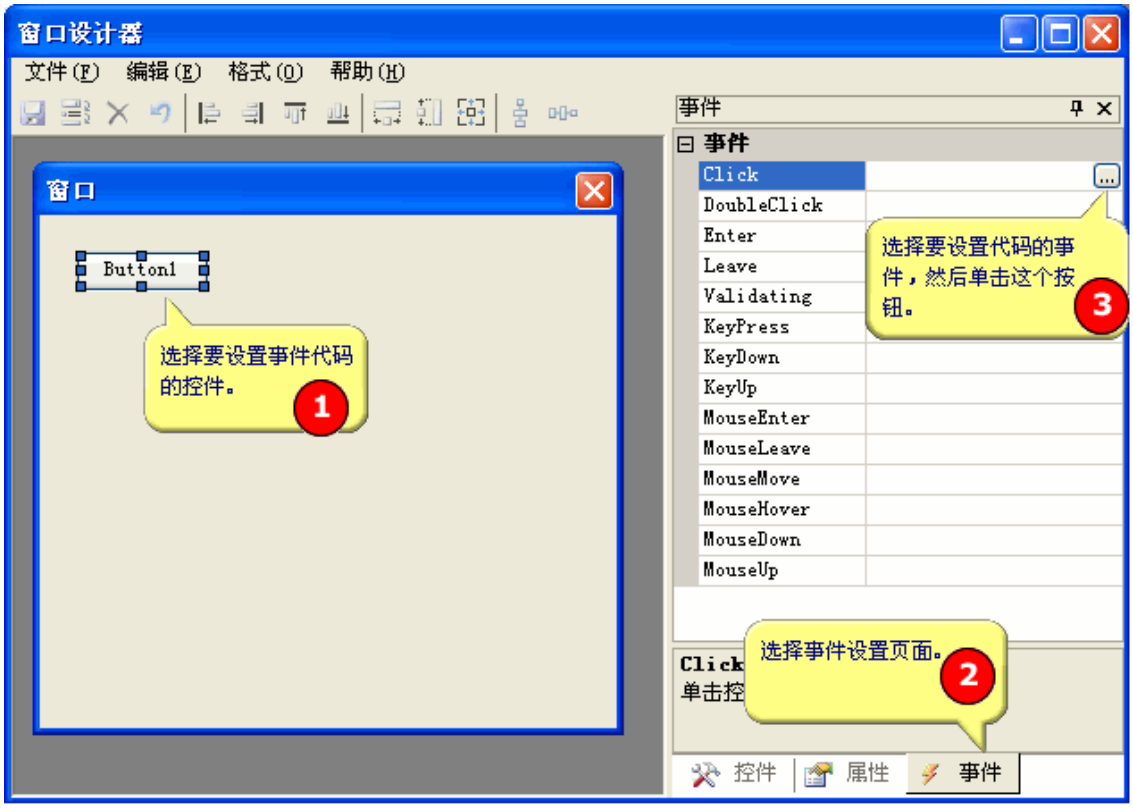


设置窗口属性

如果单击窗口的空白区域，将撤销选择控件，此时属性页面显示的是窗口属性：



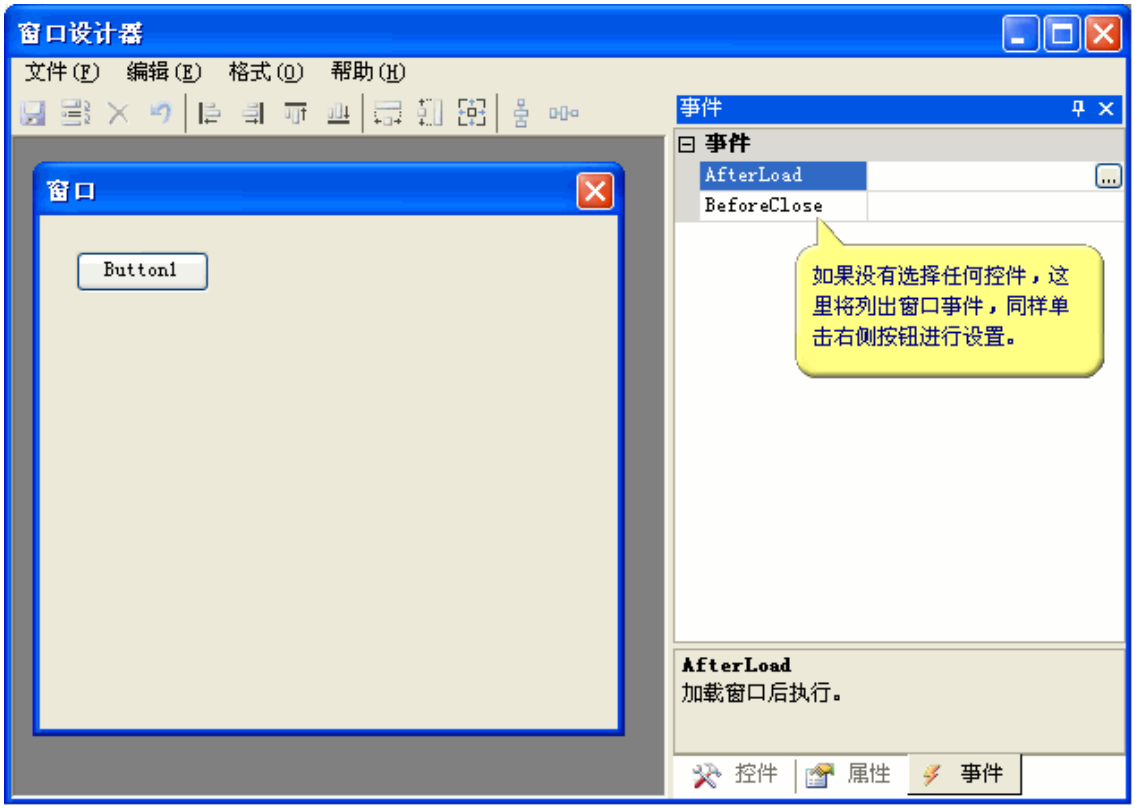
设置控件事件代码



按钮的 Click 事件是最常使用的，所以 Foxtable 提供了一个快捷设置方式：直接双击按钮，即可设置按钮的 Click 事件代码。

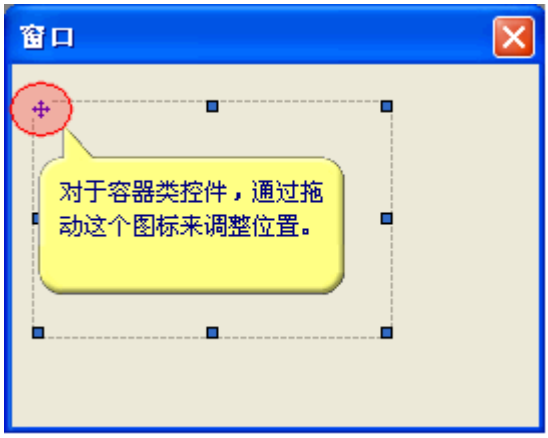
设置窗口事件代码

如果单击窗口的空白区域，将撤销选择控件，此时事件页面显示的是窗口事件：



调整控件位置



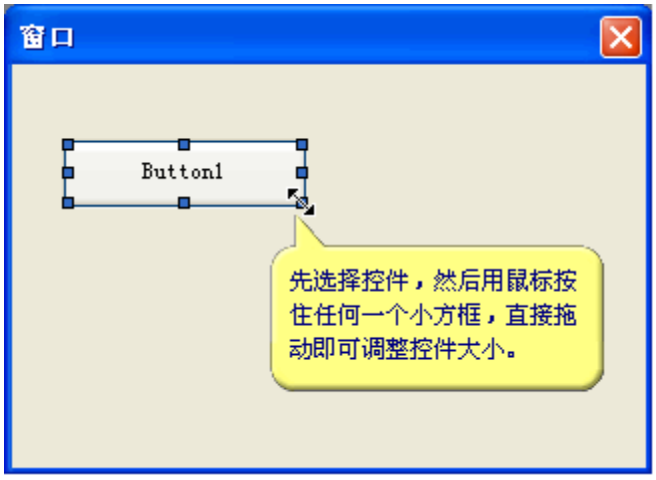


微调控件位置

按住 Ctrl 键的同时，按箭头键，可以微调选定控件的位置。

当然也可以通过属性设置窗口，精确调整控件位置。

调整控件大小



微调控件大小

按住 Shift 键的同时，按箭头键，可以微调选定控件的大小。

当然也可以通过属性设置窗口，精确调整控件大小。

选择多个对象

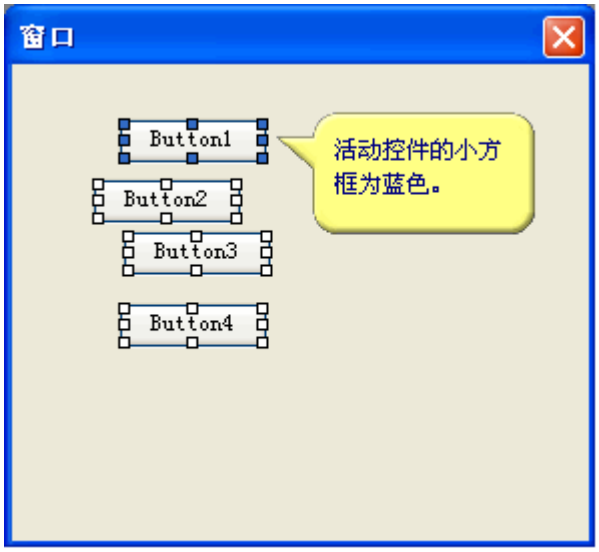
按住 Ctrl 键，依次单击要选择的控件即可，如果再次单击已经选择的控件，则取消选择该控件。

如果要选择某一区域的所有控件，在该区域的起始位置按住鼠标，拖动至结束位置松开鼠标即可。

关于活动控件

在同时选择多个控件的时候，其中一个控件为活动控件，活动控件四周的小方框为蓝色，非活动控件为白色。

如果要使某一选定控件成为活动控件，单击该控件即可。



格式菜单

在同时选择多个控件的时候，可以利用格式菜单，让所有选定控件向活动控件对齐，或者尺寸等于活动控件，还可以让选定控件的间距相等。

通过格式菜单，可以快速地对控件进行整体布局。



可视化设置 Tab 顺序

所谓 Tab 顺序，就是按 Tab 键时，依次选择控件的顺序。

在“编辑”菜单中单击“设置 Tab 顺序”，每一个控件的左上角都会出现一个蓝色方框，方框中的数字表示该控件的 Tab 顺序。

通过单击这些蓝色方框，可以调整 Tab 顺序，第一次被单击的控件 Tab 顺序会被设为 0，然后是 1，依次类推。

设置完成后，再次单击编辑菜单中的“设置 Tab 顺序”，可结束设置。



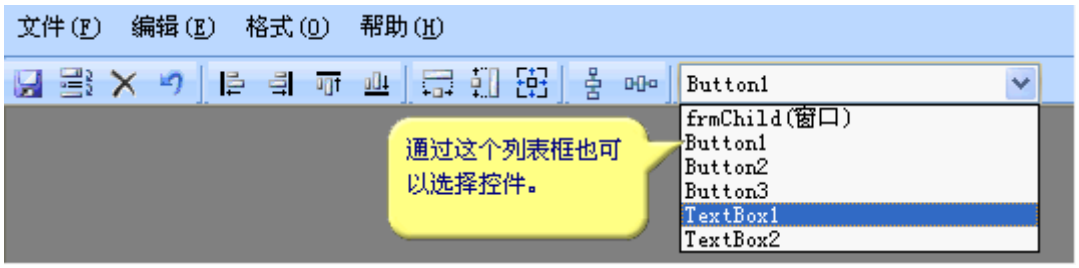
如果希望按 Tab 键选择不同控件时，能跳过某控件，可以将其“跳过 Tab 键”属性设为 True。

复制控件

选择要复制的控件，然后在“编辑”菜单中，单击“复制”命令，即可生成一个新控件，新控件的属性和事件代码和原控件完全一致。

通过工具栏选择控件

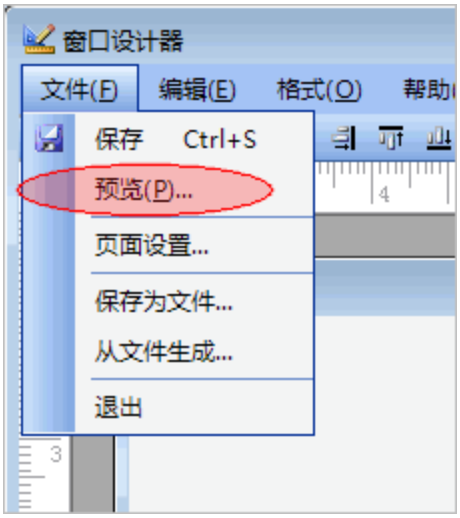
有的时候可能因为失误，导致控件超出窗体范围，以致无法选择该控件，此时可以通过工具栏进行选择。



此外， 如果某个控件的停靠属性设置为 Fill，此时控件会布满整个窗口，导致无法通过单击窗口空白区域来选择窗口，我们只能通过上面这个列表来选择窗口，然后在设置窗口的属性和事件。

4.9.1.7 预览窗口

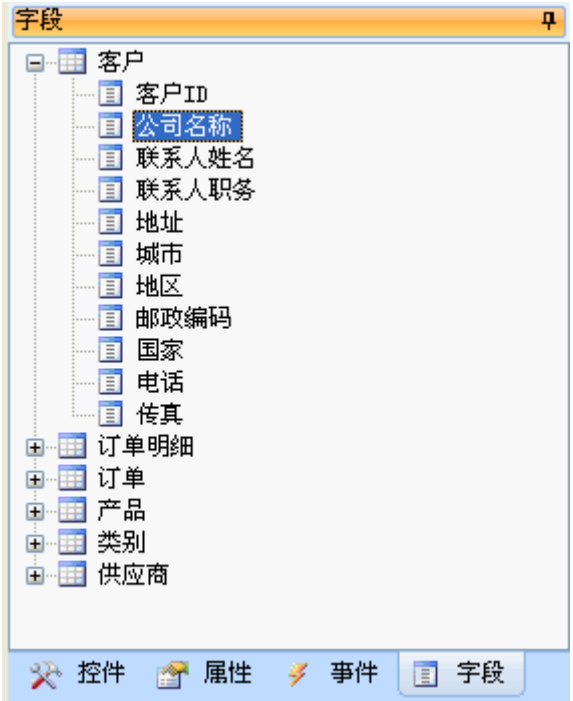
在窗口设计过程中，可以单击文件菜单下的预览命令，预览设计效果：



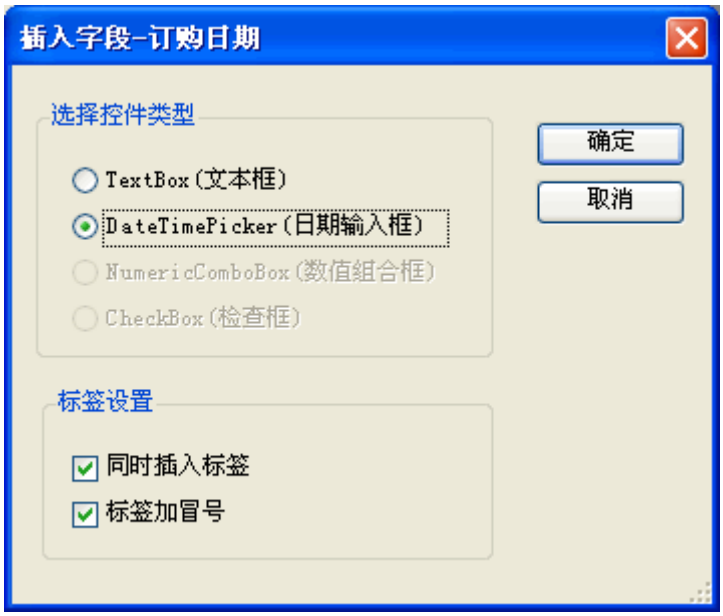
提示：不管是什么类型的窗口，预览的时候，都会以模式窗口的形式出现，只有实际应用的时候，窗口的类型才会得以体现。

4.9.1.8 快速插入字段

在设计窗口的时候，选择字段页，会列出所有表的字段(列)供选择。
拖动字段至希望插入的位置，松开鼠标，即可插入字段。
这样就能非常轻松地设计数据录入窗口。



在插入字段之前，你可以选择控件类型，以及是否同时插入标签：



并不是所有的控件类型都是可选的，系统会自动根据字段的数据类型，列出可用的控件类型。

4.9.1.9 锚定控件

本节的内容可以参考 CaseStudy 目录下的文件：锚定与停靠.Table

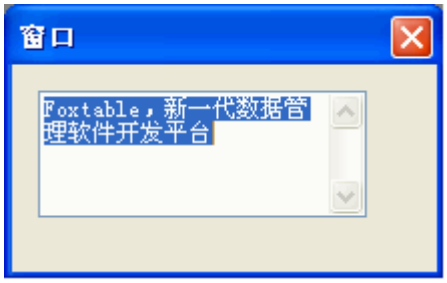
所谓锚定，就是将控件锚定于其容器(通常就是窗口)的指定边界，窗口大小发生变化时，控件和指定边界的位置始终保持不变。

控件默认是锚定上边界和左边界，这样调整窗口大小时，控件和容器左上角的距离始终保持不变。

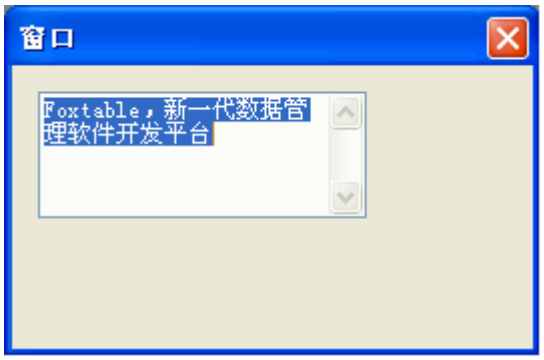
锚定属性的设置是可视化的，直接通过单击来指定所要锚定或取消锚定的边界，深色表示所要锚定的边界，如下图所示：



为了测试和学习锚定属性，在窗口中加入一个 TextBox(文本框)，将其“多行文本”属性设为 True，如下图所示：

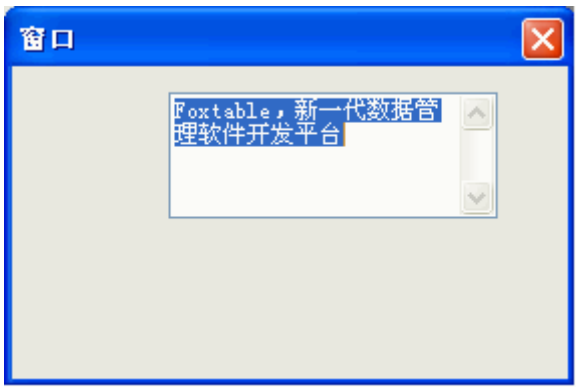


1、在默认的情况下，所有控件都是锚定于其容器的左上角，这意味着窗口大小改变时，控件并不会随之移动或改变大小：

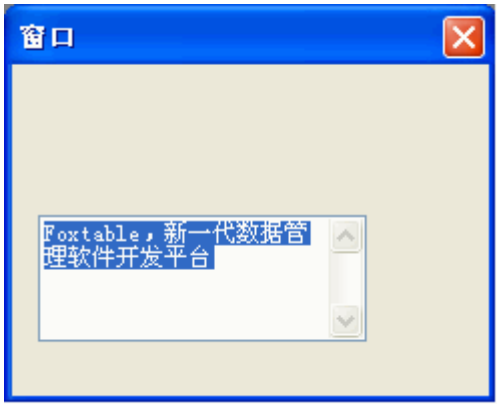


然而，当我们选择锚定到不同边界时，会发生一些有趣的情况：

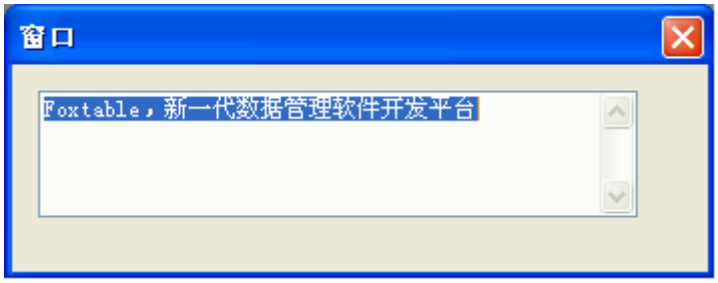
2、如果锚定于右上角，也就是同时锚定于右边和上边，控件将始终粘附于窗口的右上角，当我们改变窗口的大小时，控件也会随之移动，保持和窗口右上角的距离不变：



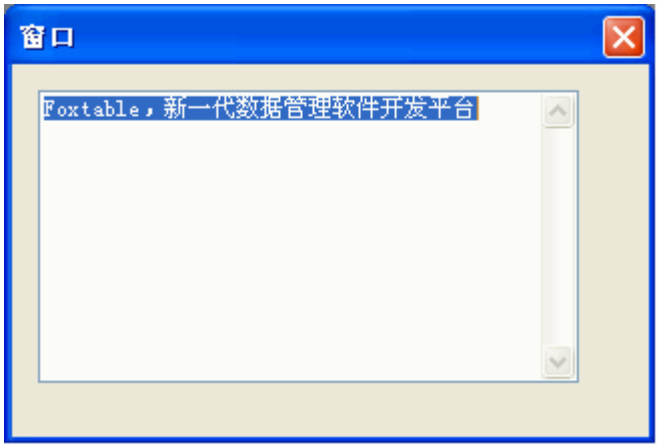
3、如果锚定于左下角，当窗口大小改变时，控件会自动移动，以便和窗口左下角的距离保持不变：



4、如果同时锚定于上边、左边和右边，那么窗口大小改变时，控件将自动调整宽度，以便保持和窗口左右两个边界的距离不变：



5、如果同时锚定四边，那么窗口大小改变时，控件将同时调整高度和宽度，以便和窗口四个边界的距离保持不变：



6、如果没有锚定到任何一个边界，或者只锚定到上下或者左右两边，那么窗口大小改变后回发生什么呢？ 请您自行测试一下。

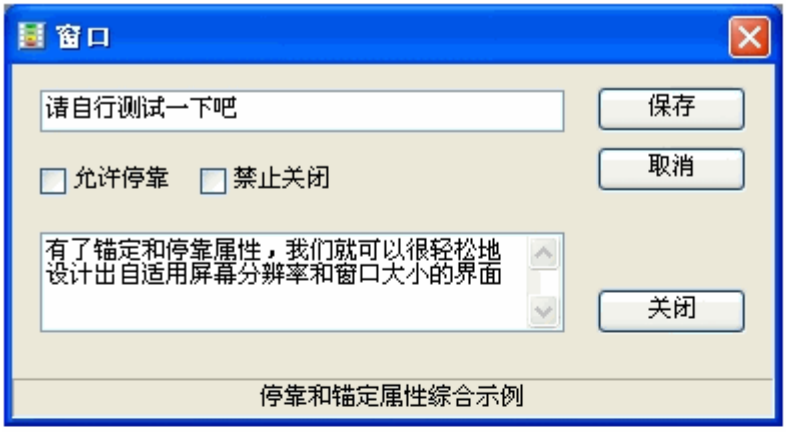
4.9.1.10 停靠控件

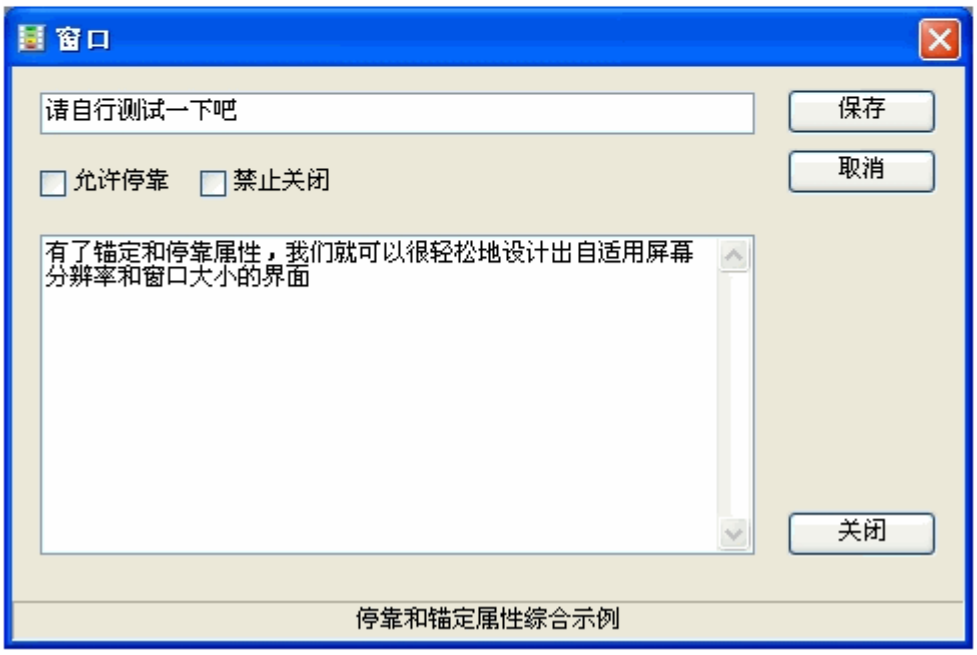
本节的内容可以参考 CaseStudy 目录下的文件：锚定与停靠.Table
停靠属性允许将控件停靠到容器(通常是窗口)指定的边界上，停靠属性的设置是可视化的：



例如要将控件停靠在容器右边，只需单击上图右侧按钮即可。
如果单击上图矩形的中心区域，则将控件停靠到四个边界上，在这种情况下，控件将覆盖容器的整个工作区。

如果要取消停靠，请单击上图矩形底部的“None”区域。
有了锚定和停靠属性，我们就可以很轻松地设计出自适应屏幕分辨率和窗口大小的界面：



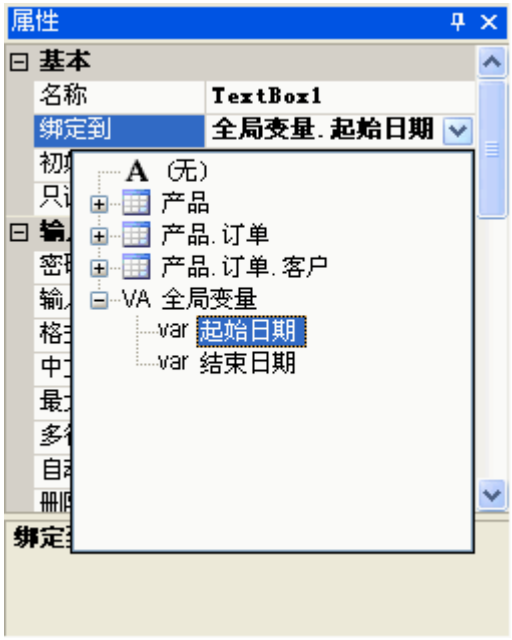


4.9.1.11 数据绑定

为了利用窗口输入数据，你必须将控件绑定到表中的某一列，绑定列可以来自于主表，也可以来自于关联表。
操作很简单，在属性窗口中设置“绑定到”属性即可:



如果定义了 Var 变量，在设置绑定字段的时候，可以选择将控件绑定到 Var 变量:



4.9.1.12 锁定控件

很多控件都有一个名为“只读”的属性，该属性决定是否允许编辑控件的内容。
只读属性包括三个可选值：

- Default
  如果绑定列、绑定列所在的表，以及该表当前行，这三者任何一个处于锁定状态，则控件的内容不能被编辑，否则可正常编辑。

- True
  禁止编辑控件内容。

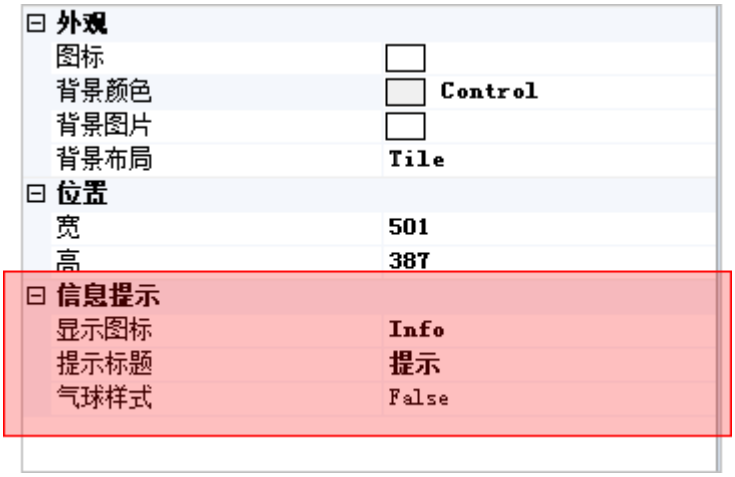
- False
  可以编辑控件内容。

4.9.1.13 提示信息

在设计窗口的时候，如果给控件的“提示”属性设置了内容，那么打开窗口后，当鼠标移到此控件位置并停留时，将出现提示，例如：



窗口本身有一组属性，用于统一设置各控件提示信息的外观：



4.9.1.14 快速入门

窗口的控件很多，每一个控件都有很多属性、方法和事件，看上去很复杂。
实际上，对于普通的应用来说，窗口设计是非常简单的。
这节我们就提供一个快速设计窗口的例子，目的是为下面的表格设计一个录入窗口。

	产品	客户	业务员	单价	折扣	数量	金额	日期	结帐
1	PD05	CS03	EP04	17	0.1	650	9945	1999-01-02	☐
2	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03	☐
3	PD05	CS04	EP02	14	0	320	4480	1999-01-03	☐
4	PD03	CS03	EP04	10	0	490	4900	1999-01-04	☑
5	PD04	CS04	EP05	17.6	0	240	4224	1999-01-07	☐
6	PD05	CS03	EP04	17	0	160	2720	1999-01-07	☐
7	PD03	CS02	EP02	8	0.1	200	1440	1999-01-08	☐
8	PD03	CS01	EP04	8	0	300	2400	1999-01-11	☐
9	PD04	CS05	EP04	22	0.05	500	10450	1999-01-11	☐

输入部分的设计：

1、插入一个 Label(标签)，将其文本属性设为“产品”。

2、插入一个 TextBox(文本框)，将其绑定到产品列，这样就为产品列提供了窗口输入功能。

3、按同样的步骤，为其余各列提供输入功能；对于数值型列，可以用 NumericComboBox(数值输

入框)代替 TextBox(文本框),而日期型列用 DateTimePicker(日期输入框)会更方便一些。

4、最后插入一个检查框，文本属性设为“已经结帐”，并将其绑定到结帐列。



操作部分的设计

1、在窗口中插入一个按钮，按钮的标题改为“增加行”，将其 Click 事件代码设为：

```
Tables("订单").AddNew()
```

2、同样增加另外五个按钮，分别是：

按钮标题	Click 事件代码

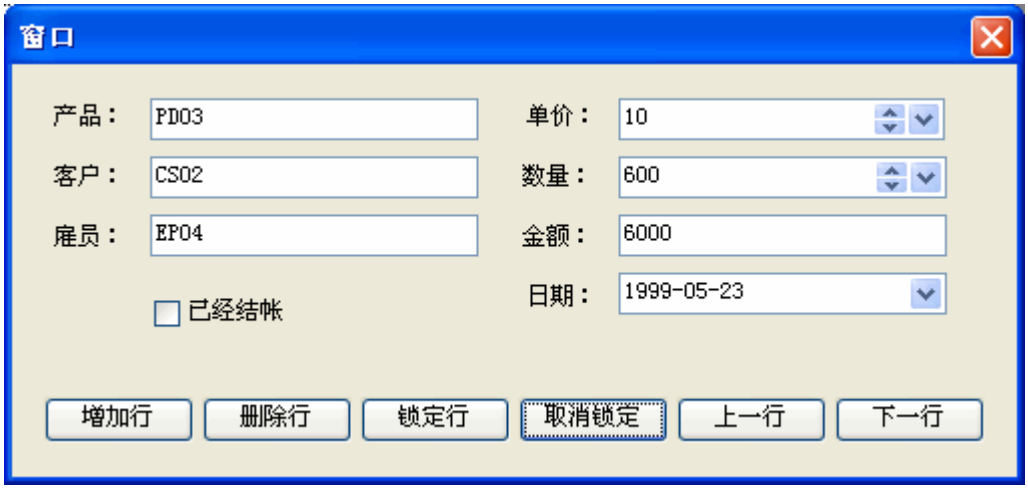
删除行	With Tables("订单")
	  If .Current IsNot Nothing Then
	  .Current.Delete
	  End If
	End With

锁定行	With Tables("订单")
	  If .Current IsNot Nothing Then
	  .Current.Locked = True
	  End If
	End With

取消锁定	With Tables("订单")
	  If .Current IsNot Nothing Then
	  .Current.Locked = False
	  End If
	End With

上一行	With Tables("订单") .Position = .Position - 1 End With
下一行	With Tables("订单") .Position = .Position + 1 End With

至此，我们的窗口不仅具有录入功能，而且具备常用的操作功能：



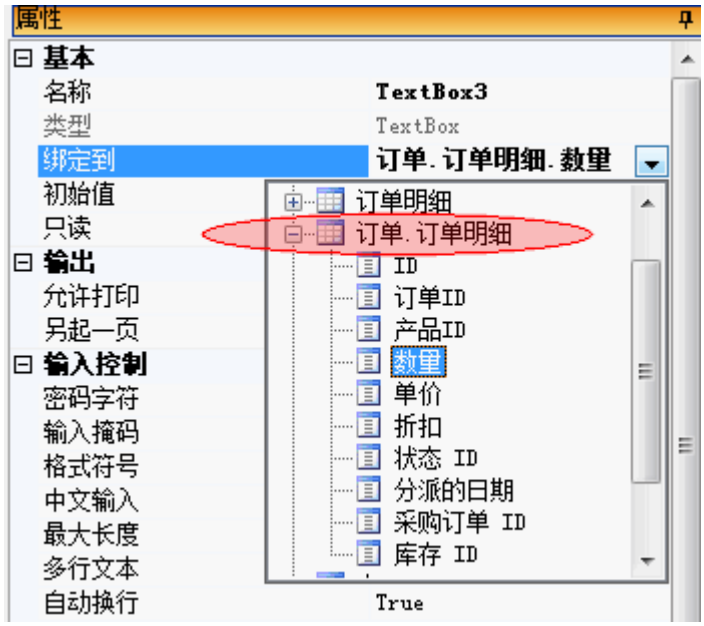
4.9.1.15 为关联表设计录入窗口

假定订单(父表)和订单明细表(子表)已经建立关联，这样选择某个订单，关联表(订单.订单明细)即可显示此订单的订单明细。

如果希望在关联表(订单.订单明细)双击某个订单明细，能够显示一个录入窗口，用于编辑此订单明细。

此录入窗口的设计和普通录入窗口的设计基本类似

1、首先设计窗口的时候，各文本框需要绑定到关联表(订单.订单明细)的各列，例如下图将 TextBox3 绑定到此关联表的数量列：



2、窗口中各按钮的代码也应该有所调整：

按钮标题	Click 事件代码

删除行	With Tables("订单.订单明细")
	  If .Current IsNot Nothing Then
	  .Current.Delete
	  End If

	End With

锁定行	With Tables("订单.订单明细")
	  If .Current IsNot Nothing Then
	  .Current.Locked = True
	  End If
	End With

取消锁定	With Tables("订单.订单明细")
	  If .Current IsNot Nothing Then
	  .Current.Locked = False
	  End If
	End With

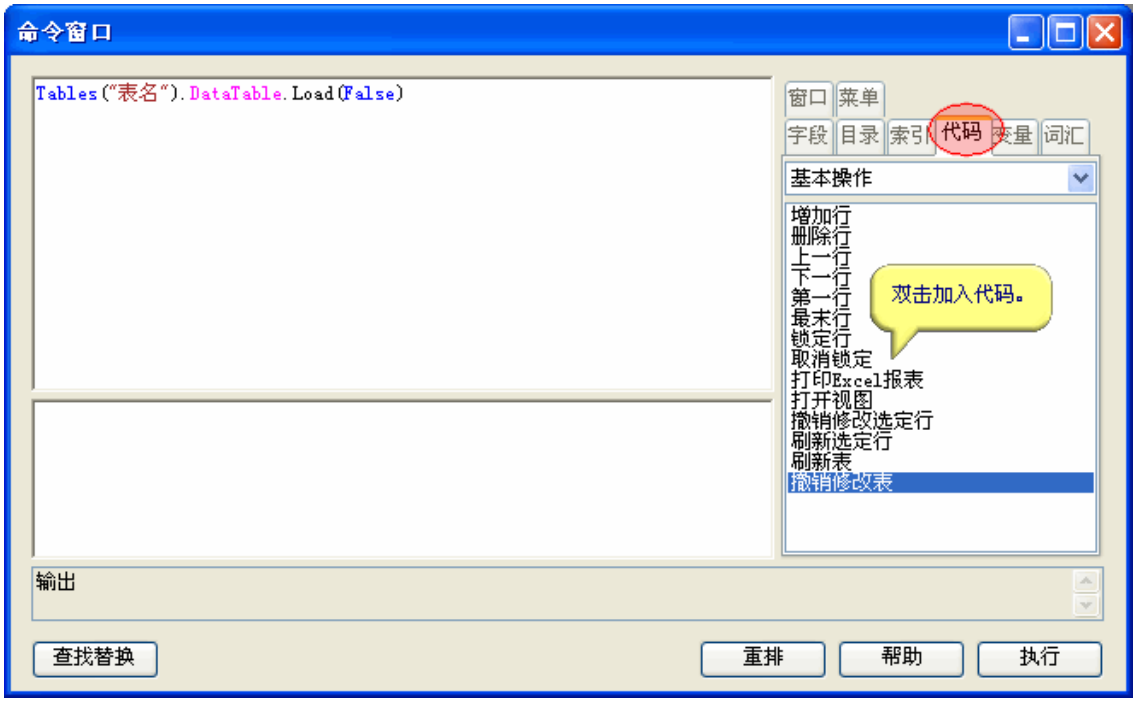
上一行	With Tables("订单.订单明细") .Position = .Position - 1 End With
下一行	With Tables("订单.订单明细") .Position = .Position + 1 End With

3、最后将订单明细表的DoubleClick事件代码设置为：

```
If e.Table.Name = "订单.订单明细" Then
   e.Cancel = True '禁止进入编辑状态
   Forms("窗口1").Open()
End If
```

4.9.1.16 代码库

在代码编辑器的右侧的代码页，提供了一些常用的简单代码，双击选定的项目，即可将代码加入到代码编辑器中：



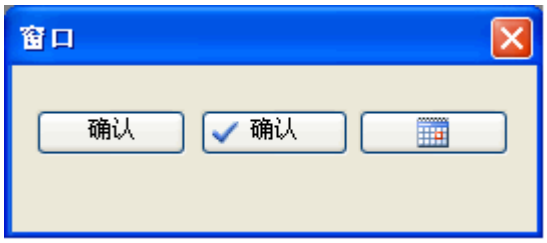
Foxtable 目录下有一个 Access 文件 Code.mdb，该文件用于管理代码库，各位可以向其中增加自己常用的代码。

4.9.1.17 常用控件

4.9.1.17.1 Button

Button 表示按钮，用于单击执行某一操作。

通过设置按钮的标题和图标，可以设计出各种风格的按钮：

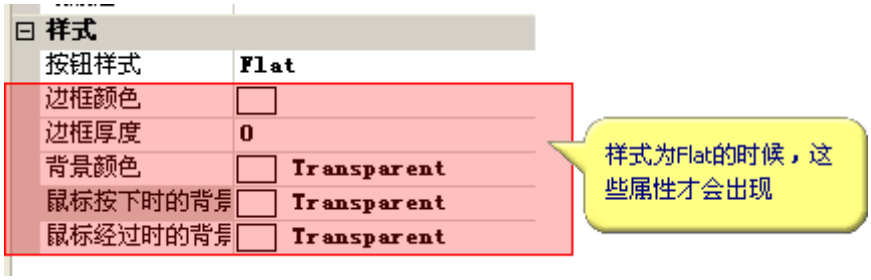




按钮样式

有四种类型的按钮样式，分别是 Flat、Popup、System、Standard。

其中样式 Flat 较为特别，选择此样式后，还可以设置按钮的背景、边框等属性，这样就可以设计出一些较为特殊的按钮，例如没有边框的透明按钮。



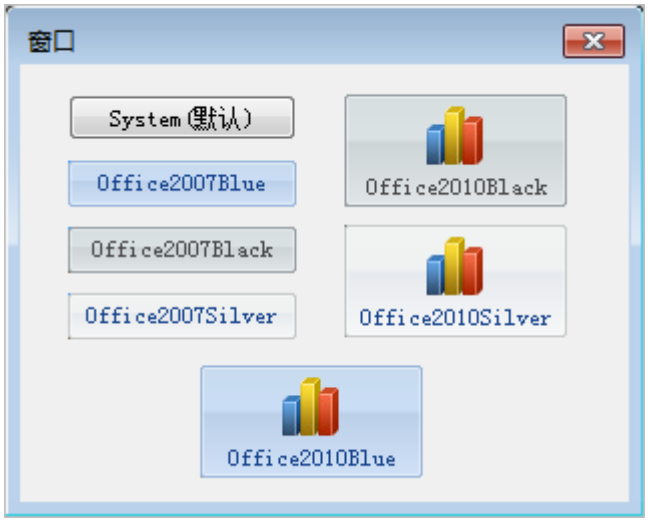
提示：要使某个控件的背景透明，应该将其背景颜色设置为 Transparent

按钮风格

如果按钮的外观是 Standard，那么还可以设置按钮的样式，默认的样式为 System：



下图是不同样式的按钮：

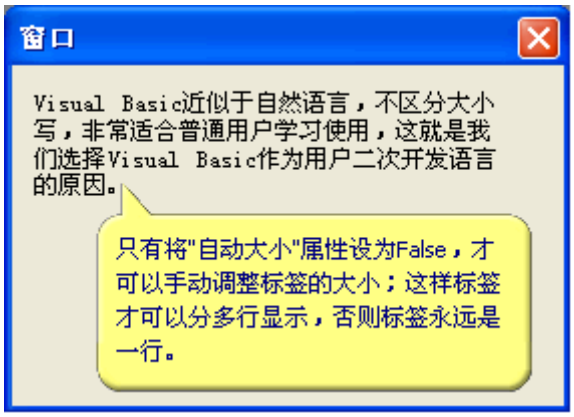


4.9.1.17.2 Label

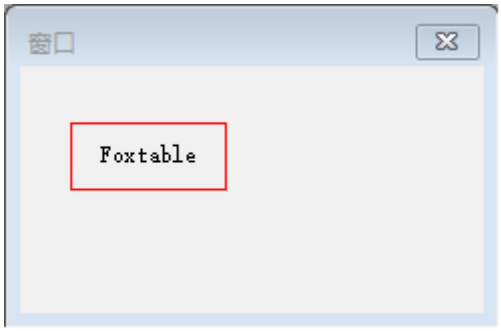
Label 表示标签，通常用于显示不可编辑的字符型内容。

标签有一个很重要的属性，名为“自动大小”，默认为 True，此时标签自动根据其内容调整大小，而不可以手动调整或通过代码设置。

如果需要设置 Label 的大小，请将“自动大小”属性设为 False。



可以给 Label 设置边框和边框颜色，还可以设置文本的位置(自动大小为 False 时)，例如：



Label 有格式符号属性，可以用于设置显示格式，例如下图三个 Label 都绑定到了统一日期列，但是由于格式符号属性值不同，所以显示结果也不同：



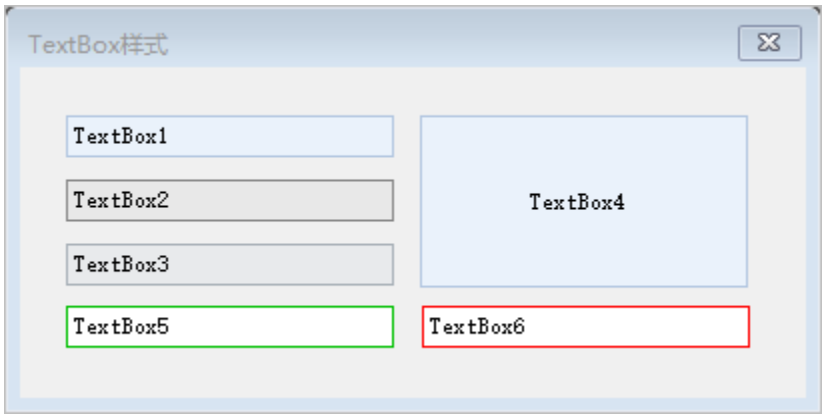
有关格式化符号的规范，和 Format 函数相同，请参考：
格式化数字
格式化日期和时间
如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.1.17.3 TextBox

TextBox 表示用于输入字符型内容的文本框。
在默认的情况下，文本框只能输入一行内容，且不能调整高度，如果要输入多行文本，请在设计窗口的时候，将“多行文本”属性设为 True。



TextBox 可以设置样式和边框颜色，以及水平和垂直方向的文本位置：



TextBox 可以设置格式符号属性，同一个值不同的格式符号，会有不同的显示结果，如：



有关格式化符号的规范，和 Format 函数相同，请参考：

格式化数字

格式化日期和时间

如果有更复杂的显示要求，可以参考事件：DataFormat

提示：

1、设计窗口的时候，对于单行文本框，我们无法直接通过鼠标拖动调整其高度，但是可以通过属性设置窗口设置其高度属性。

2、如果想多行文本框出现滚动条以显示更多的内容，可以设置其“滚动条”属性为 Vertical。

3、设计窗口的时候，对于日期输入，尽量用 DateTimePicker 代替 TextBox，对于数值输入，尽量用 NumericComboBox 代替 TextBox。

4.9.1.17.4 DateTimePicker

DateTimePicker 表示日期输入框，专门用于输入日期和时间。

可直接在日期输入框输入日期或时间，如果单击右边的下拉按钮，将出现一个日历用于选择输入日期：



通过设置“日历维数”属性，可以让下拉日历显示多个月份的日历：



通过“日期时间格式”属性，可以选择输入格式：



该属性有五个可选值：

Date：　　标准日期格式，如"2008-12-31"

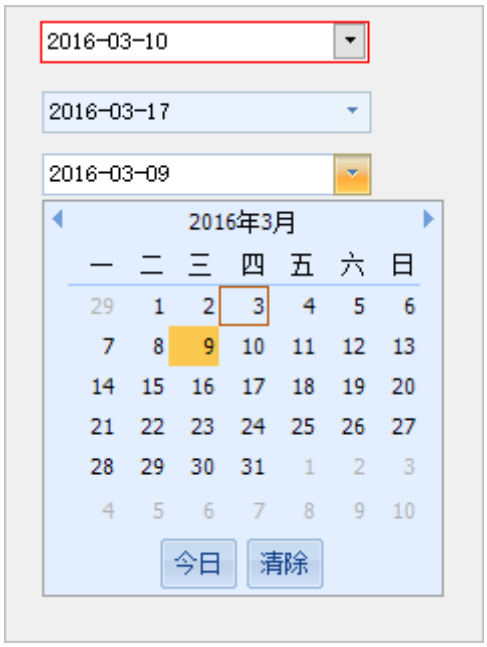
DateTime：　　短日期时间格式，包括日期和时分，如"2008-12-31 9:45"

LongDateTime：　　长日期时间格式，包括日期和时分秒，如"2008-12-31 9:45:30"

Time：　　短时间格式，包括时分，如“12:59”

LongTime：　　长时间格式，包括时分秒，如“12:59:21”

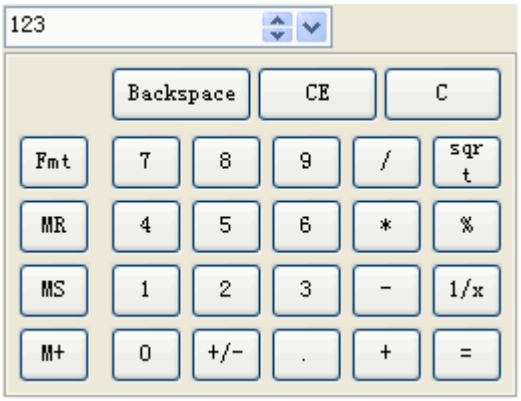
DateTimePicker 同样可以设置样式和边框颜色：



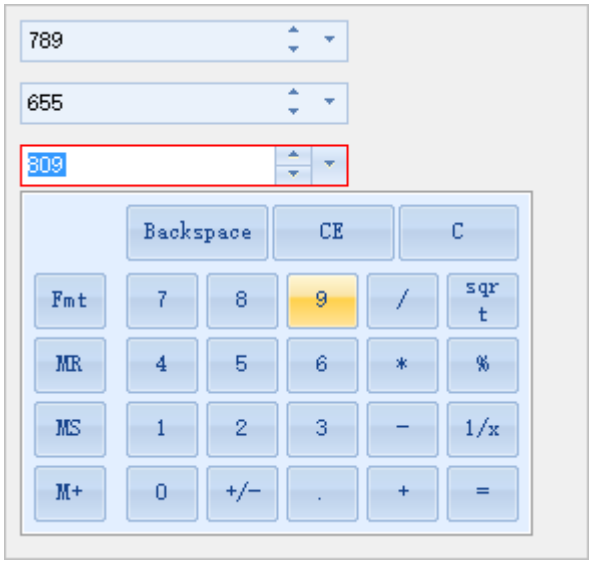
提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 DateTimePicker 的高度，但是可以通过属性设置窗口设置其高度属性。

4.9.1.17.5 NumericComboBox

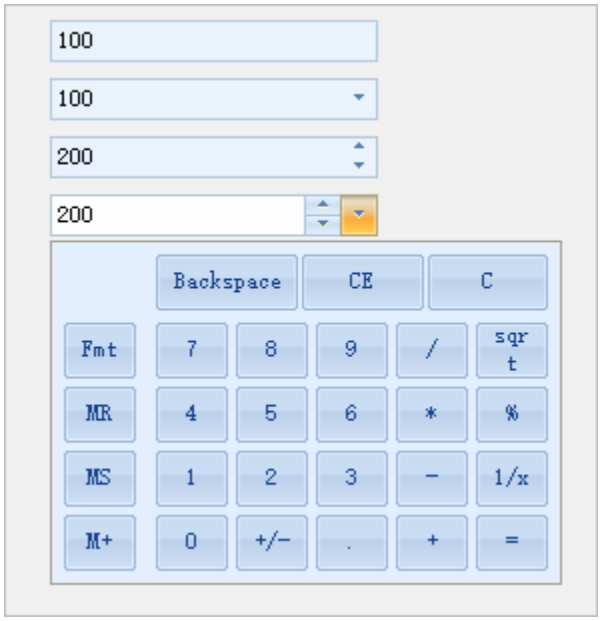
NumericComboBox 表示数值输入框，专门用于输入数值。
可直接在数值输入框中输入数值，或利用上下微调按钮调整已经输入的数值，如果单击右边的下拉按钮，将出现一个计算器用于计算和输入数值。



提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 NumericComboBox 的高度，但是可以通过属性设置窗口设置其高度属性。
NumericComboBox 可以设置边框颜色和样式：



NumericComboBox 右侧的微调按钮和下拉按钮是可以关闭的，例如下图四个输入框都是NumericComboBox，一个只有微调按钮，一个只有下拉按钮，一个则没有显示任何按钮：



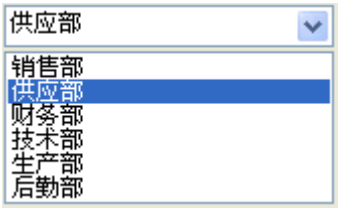
NumericComboBox 可以设置格式符号属性，同一个值不同的格式符号，会有不同的显示结果，如：



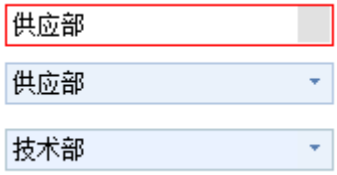
有关格式化符号的规范，和 Format 函数相同，请参考：格式化数字
如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.1.17.6 ComboBox

ComboBox 表示组合框。
组合框可以直接输入内容，也可以单击右侧的箭头，从预先设置好的列表项目中选择值：



请参考：
设置列表项目
辅助输入设置
ComboBox 可以设置样式和边框颜色：



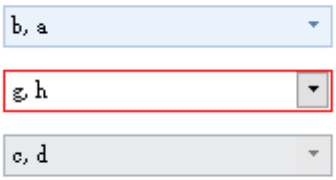
提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 ComboBox 的高度，但是可以通过属性设置窗口设置其高度属性。

4.9.1.17.7 CheckedComboBox

CheckedComboBox 表示复选组合框，可以同时从列表项目中选择多项：



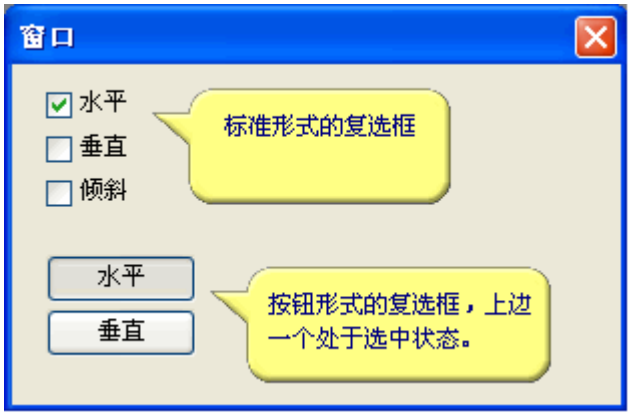
值与值之间默认的分隔符号是逗号，如果需要改为其他符号，在属性设置中修改“分隔符”属性即可。
复选组合框其余的设置，和组合框是完全一样的。
请参考：
设置列表项目
辅助输入设置
CheckedComboBox 同样可以设置边框颜色和样式：



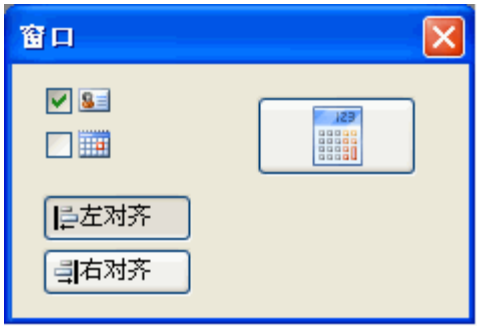
提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 CheckedComboBox 的高度，但是可以通过属性设置窗口设置其高度属性。

4.9.1.17.8 CheckBox

CheckBox 表示复选框，通常用于输入逻辑值，可以在两种状态之间反复切换。
复选框默认为方框形式，如果在设计窗口的时候，将其“外观”属性设为“Button”，复选框看上去就像一个按钮，按钮可以在按下和弹起两种状态中切换，按下表示选中，弹起表示未选中。



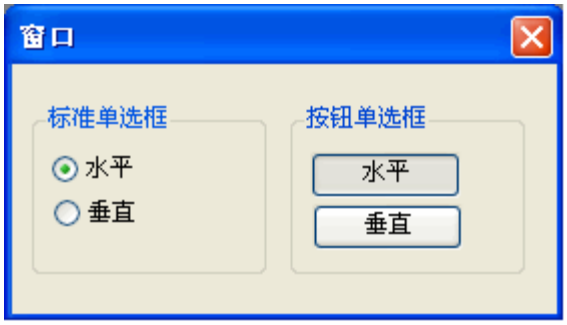
和按钮一样，复选框也可以设置图标：



4.9.1.17.9 RadioButton

RadioButton 表示单选框。
单选框和复选框非常类似，不同的是，同一个容器中的多个单选框，每次只能有一个处于选中状态，如果你选中一个单选框，之前选中的单选框将被自动撤销选中状态。

如果在设计窗口的时候，将单选框的“外观”属性设为“Button”，单选框看上去就像一个按钮，按钮可以在按下和弹起两种状态中切换，按下表示选中，弹起表示未选中。



和按钮一样，单选框也可以设置图标：



4.9.1.17.10 GroupBox

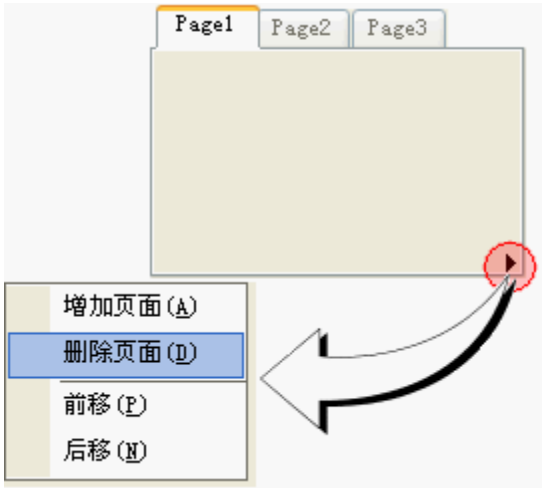
GroupBox 表示分组框。
窗口中控件较多的时候，可以将功能类似的控件放在同一分组中，使得窗口看上去更有条理。
分组框有一个标题，用于对该分组框中的控件进行说明。



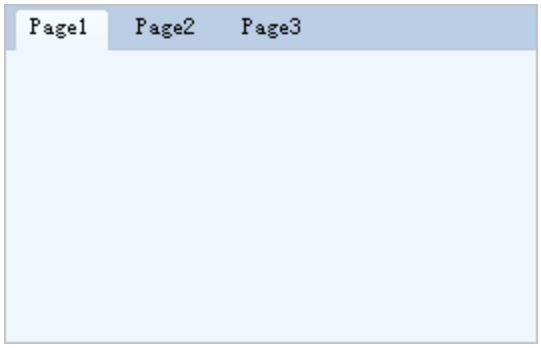
4.9.1.17.11 TabControl

TabControl 表示页面集合，可以包括多个页面，每一个页面就是一个面板，可以容纳任意类型的控件。

在设计模式下的 TabControl，右下角会显示一个黑色箭头，单击该箭头会出现一个菜单，通过该菜单增加、删除、移动页面：



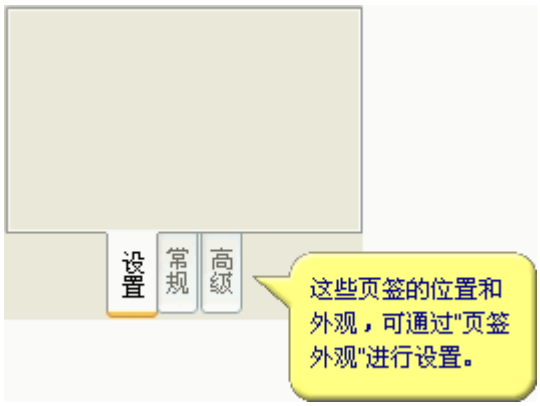
TabControl 提供了多种样式供选择，以下是其中几种：







TabControl 有一组“页签外观”属性，通过这些属性，可以很灵活地设置页签的外观，例如：



此外还有一组“选定页面”的属性，用于对选定页面进行设置：



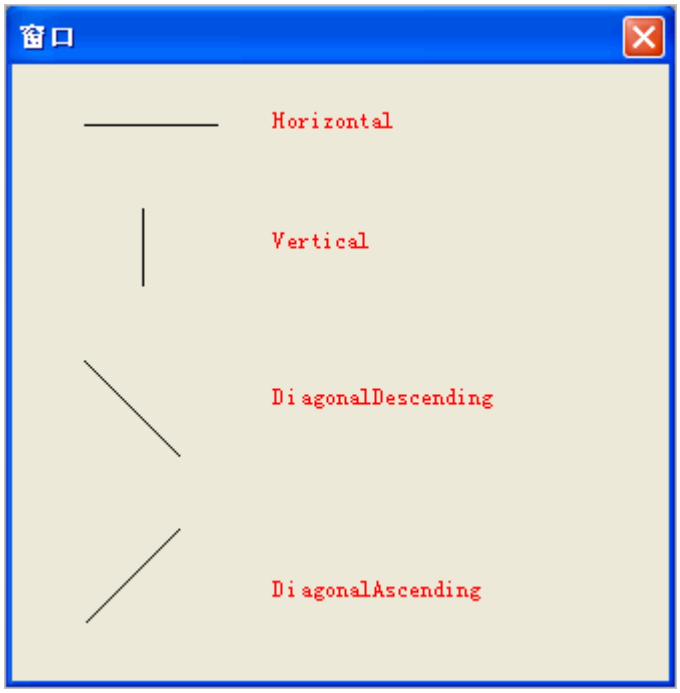
这些属性都是自我描述性的，而且设置之后，可以即刻看到效果，所以就不再一一介绍。

4.9.1.17.12 Line

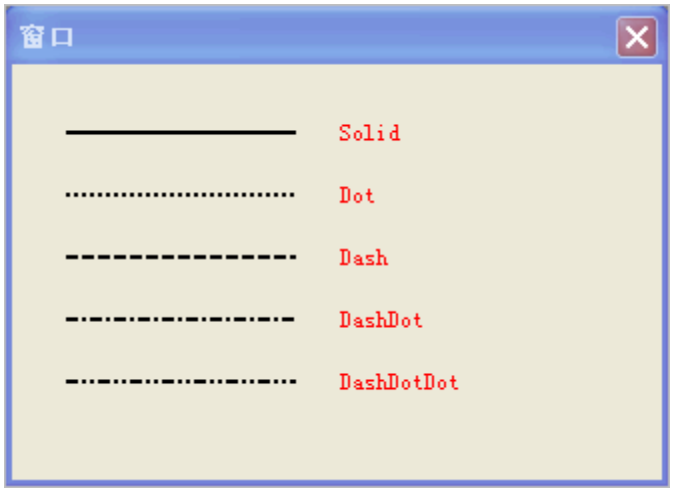
Line 表示线条。

颜色、线宽、方向、线形决定了一个线条的外观。

方向有四个：



线形有以下几种：



4.9.1.17.13 PictureViewer

PictureViewer 表示图片浏览器，可以同时管理多个图片，非常方便。



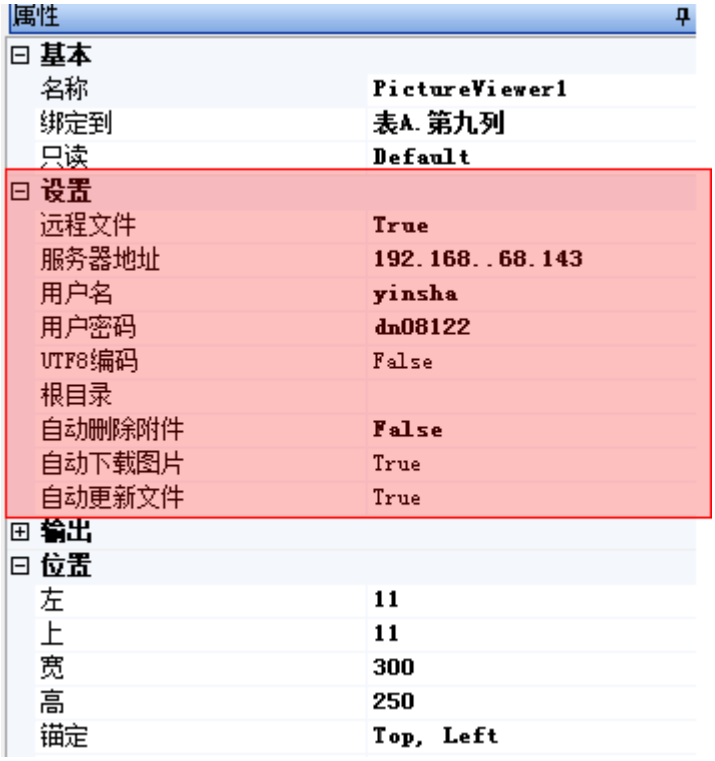
PictureViewer 不能单独使用，必须绑定字段。
PictureViewer 较重要的属性有：

- 显示命令
  是否在 PictureViewer 的底端显示增加、删除、清除、打开、全屏等命令。

- 允许打开
  是否允许用户单击打开命令调用第三方程序来打开图片文件。

- 自动缩放
  是否自动缩放图片来适用 PictureViewer 的尺寸。

PictureViewer 提供了一组和图片管理相关的属性：

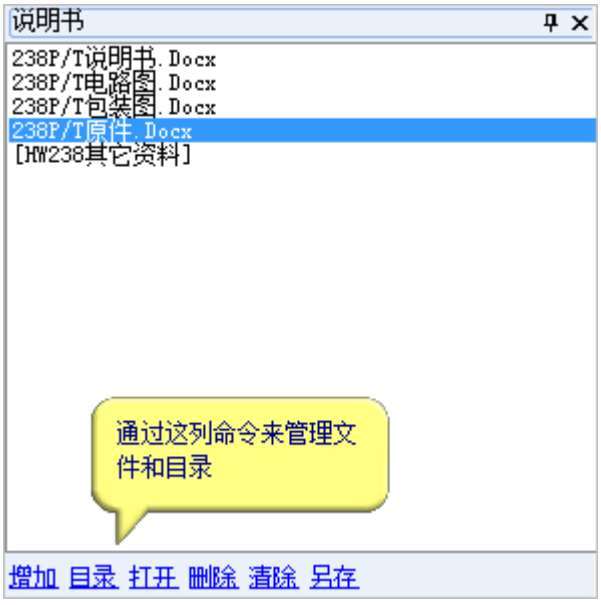


上述属性在《使用指南》之《列相关》之《扩展列列类型》之《文件与图片的管理》中已经讲述，这里就不重复了。

需要注意的是，即使 PictureViewer 绑定到某一列，也不会从这一列继承上述属性设置，PictureViewer 需要单独设置上述属性，这样虽然有所不便，但也带来了更大的灵活性。

4.9.1.17.14 FileManager

FileManager 表示文件管理器。



FileManager 不能单独使用，必须绑定字段。
FileManager 较重要的属性有：

- 显示命令
  是否在 FileManager 的底端显示增加、删除、清除、打开等命令。

- 允许打开
  是否允许用户单击打开命令调用第三方程序来打开文件。

FileManager 提供了一组和文件管理相关的属性：



上述属性在《使用指南》之《列相关》之《扩展列列类型》之《文件与图片的管理》中已经讲述，这里就不重复了。
需要注意的是，即使 FileManager 绑定到某一列，也不会从这一列继承上述属性设置，FileManager 需要单独设置上述属性，这样虽然有所不便，但也带来了更大的灵活性。

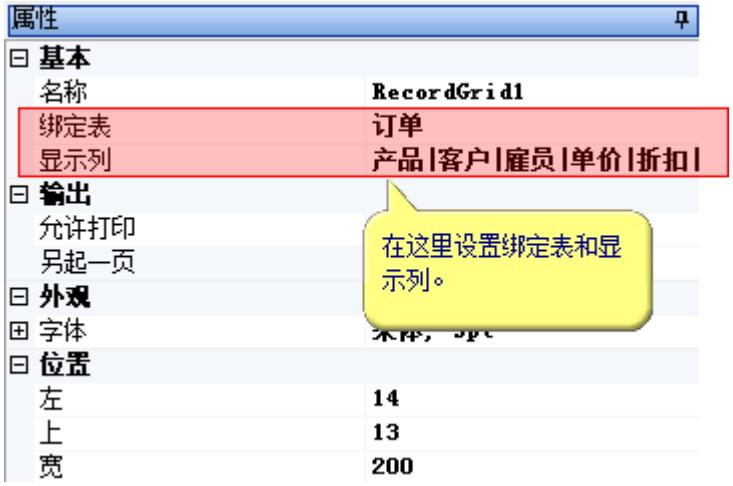
4.9.1.17.15 RecordGrid

关于记录窗口，大家应该很熟悉了，因为在使用指南中已经讲述了，参考：记录窗口
RecordGrid 表示记录窗口，当然这并非是一个窗口，而是一个控件，有了 RecordGrid，可以在普通窗口中实现记录窗口的功能，一次显示某一记录的全部内容：



更重要的是，常规的记录窗口只对主表有效，而 RecordGrid 可以绑定到任何类型的表，包括关联表，以及 SQLTable、QueryTable 和副本 Table。

在窗口设计器中，可以选择 RecordGrid 要绑定的表和显示列：



如果一个表的列很多，你可以插入多个 RecordGrid 控件，全部绑定到这个表，然后给不同的 RecordGrid 控件设置不同的显示列。

4.9.1.18 格式符号

大部分用于输入数据的控件，包括 TextBox（文本框）、DateTimePicker(日期输入框)和 NumericComboBox(数值输入框),都有一个名为“格式符号”的属性，此属性通常用于设置数值 和日期的显示格式。

例如希望按照百分比格式显示数值，最多两位整数，保留两位小数，可以将格式符号设置为：#0.00%

格式符号只影响数值的显示格式，例如将 NumericComboBox 的格式符号设置为：#0.00%，如果 其值为 0.123，那么进入输入框编辑数据的时候，显示的是：0.123，而输入完成离开后，显示的是：

12.30%
同样如果 DateTimePicker 绑定到了日期列，格式符号设置为：yyyy 年 MM 月 dd 日，如果输入 2016-2-29，那么 输入完成离开后，其内容会显示为：2016 年 02 月 29 日
有关格式化符号的规范，和 Format 函数相同，请参考：
格式化数字
格式化日期和时间
如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.1.19 输入掩码

一般用户可以忽略本节内容。
一些输入控件，如 TextBox、ComboBox 和 NumericComboBox 等，可以设置输入掩码属性。

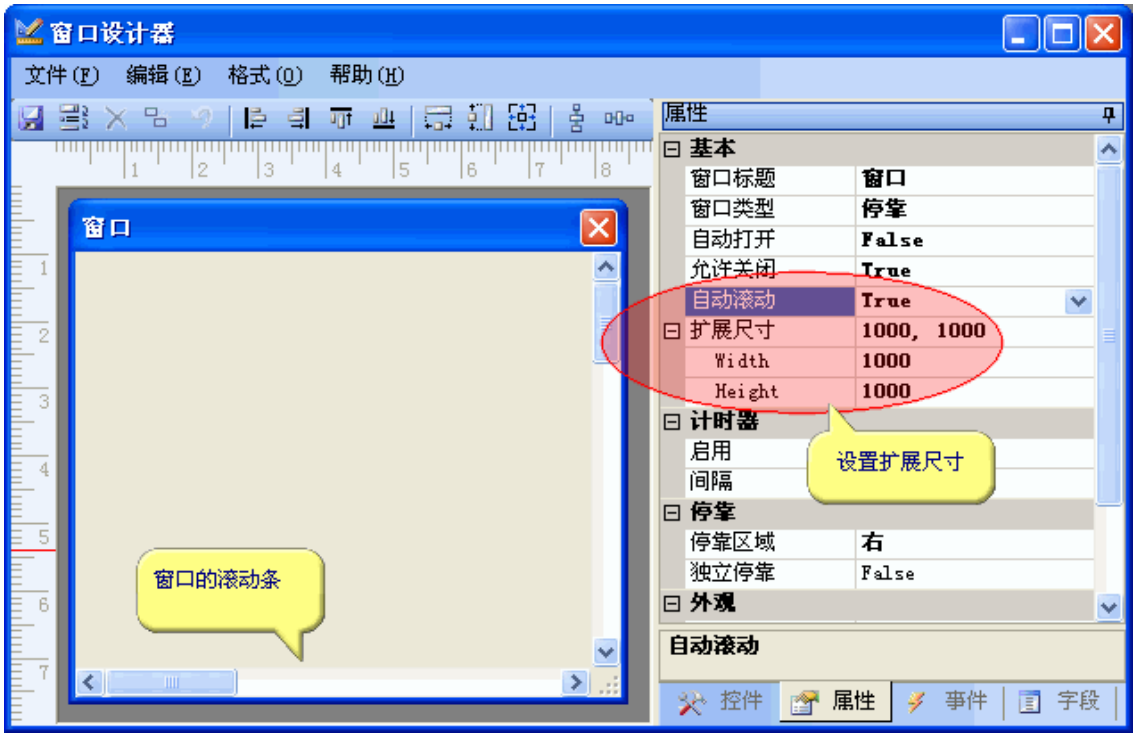
输入掩码可以规范和校验用户所输入的数据。
可作为输入掩码使用的字符有：

字符	说明
0	数字（0 到 9，必需项，不允许使用加号 [+] 与减号 [－]）
9	数字或空格（可选项，不允许使用加号和减号）
#	数字或空格（可选项，允许使用加号和减号）
L	字母（A 到 Z，必需项）
?	字母（A 到 Z，可选项）
A	字母或数字（必需项）
a	字母或数字（可选项）
&	任一字符或空格（必需项）
C	任一字符或空格（可选项）。
. , : ; - /	小数点占位符及千位、日期与时间的分隔符
<	将所有字符转换为小写
>	将所有字符转换为大写
\	使接下来的字符以表意字符显示（例如，\A 只显示为 A）

例如某 TextBox 绑定到了产品型号列，产品型号由三个字母、一个横线、两个数字组成，例如“KLU-73”，可以将该 TextBox 的输入掩码设为“AAA-00”；如果要求三个字母必须为大写，那么掩码可以设为“>AAA-00”。

4.9.1.20 扩展尺寸

如果窗口内容太多，我们可以使用 TabControl 控件分页显示。
如果不想分页显示，想一次性地显示所有内容，可以设置扩展尺寸。
扩展尺寸的单位为像素，一般都会大于窗口本身的尺寸。
设置扩展尺寸的同时，最好也将“自动滚动”属性设为 True，这样窗口就会出现滚动条，便于用户滚动查看窗口内容。



另外如果将“自动滚动”属性设为 True，当窗口中的控件位置或大小超出窗口范围时，将自动出现滚动条，不管是否设置了扩展尺寸。

4.9.1.21 计时器

如果希望打开窗口，就定时执行某一特定的任务，可以启用窗口的计时器功能。

启用计时器很简单：

1、将“启用”属性设为 True，将“间隔”属性设为任务执行的时间间隔，单位为毫秒。



2、设置 TimerTick 事件代码，一旦打开窗口，就会按照设定的时间间隔触发该事件，达到定时执行的目的。

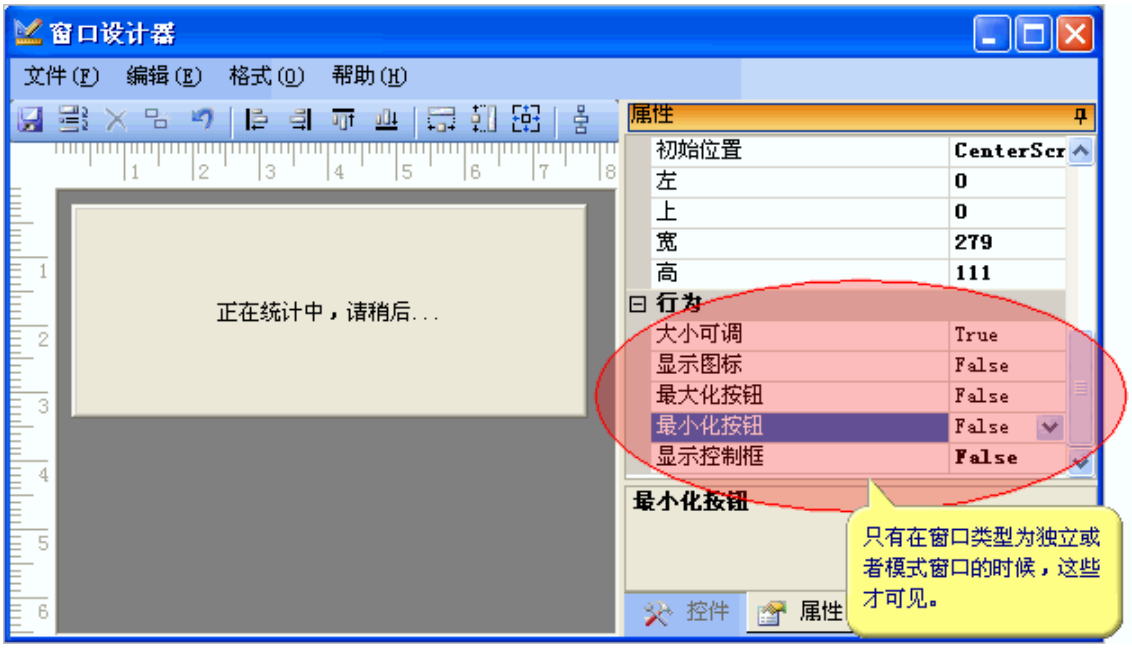
4.9.1.22 关于窗口标题栏

独立与模式窗口

对于独立窗口和模式窗口，窗口标题栏默认会显示窗口标题和关闭按钮，你可以在窗口标题栏加上最大化和最小化按钮，只需将“最大化按钮”和“最小化按钮”两个属性设置为 True 即可。

有时候可能希望标题栏不要显示关闭按钮，此时只需将“显示控制框”属性设置为 False 即可。

如果要隐藏整个标题栏，只需删除窗口标题，并将“显示控制框”属性设置为 False 即可，下图正在设计的就是一个无标题栏的窗口：



停靠窗口和共有窗口

对于停靠窗和共有窗口，如果希望不显示窗口标题，可以将“显示标题属性”设置为 False，如果希望标题栏不要显示关闭按钮，可以将“允许关闭”属性设置为 False。



下图左边的窗口没有显示标题栏：



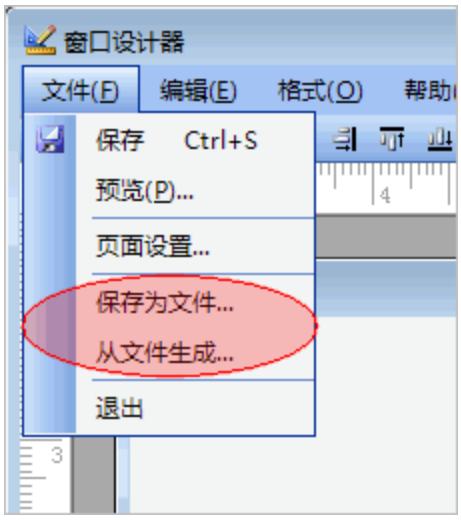
需要注意的是，同一个停靠位置可以停靠多个窗口，而这些窗口是共享标题栏的，所以有关标题栏的设置，是以最后一个打开窗口的属性设置为准。

4.9.1.23 窗口模板

在窗口设计器的文件菜单中：

1、单击“保存为文件”命令，可以将正在设计的窗口保存为一个模板文件。

2、单击“从文件生成”命令，可以选择一个现有窗口模板文件来生成窗口。



4.9.1.24 我需要窗口吗？

很多初次接触编程的人，总是将过多的精力放在窗口和界面中，恨不得所有的输入和操作都在窗口完成。
其实对于数据管理和输入，最方便的形式还是表格。
软件的设计，应该以方便实用为主，不要太刻意追求界面。
我的建议是：
1、输入尽可能在表格中完成，Foxtable 为表格提供了最完善的输入设置和控制功能。
2、简单的操作通过菜单完成，例如增加、删除、锁定行这样的操作。
3、需要用户交互的操作，可以考虑通过窗口来完成，例如对于某一时段的数据进行统计，需要用户输入起始日期和终止日期。
当然，这只是建议而已，实际开发的时候，还需要针对具体的情况进行分析。

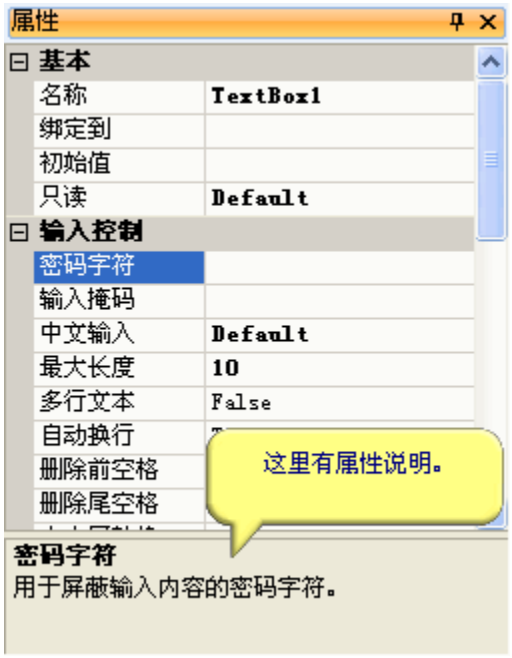
4.9.1.25 我需要继续吗？

如果您只是将窗口用于简单的录入和操作,那么“窗口基础”的内容已经足够了，“控件参考”部分是多余的，您关于窗口的学习可以到此为止。
如果你想更上一层楼，而且你有时间和耐心，那么请继续。
接下来的“控件参考”篇，会详细介绍每一个控件，内容繁多，不过这些内容都是参考性质的，在学习上并无难度可言。
再次提醒大家：不要尝试去记住所有的内容，没有人能做到的，“控件参考”篇其实就是一个参考手册，是提供给大家在编程时根据需要查阅参考的。

4.9.2 控件参考

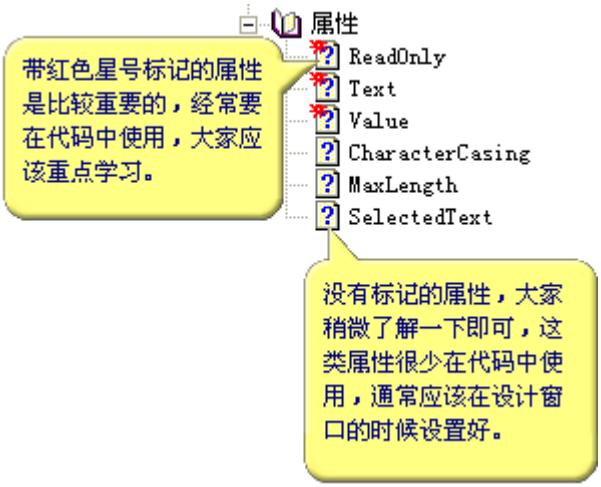
4.9.2.1 抓住重点

在设计窗口的时候，属性的名称已经翻译为中文，这些名称都是自我描述性的，而且部分属性还会提供说明，使用起来很方便：



绝大多数属性，都可以在设计窗口的时候设置好，很少在编码中使用。

尽管如此，大部分能够在代码中应用的属性，接下来都会介绍给大家，由于属性众多，而且相当部分基本上是用不着的，所以为便于大家的学习，帮助文件用两种不同的图标来区分重要性不同的属性：



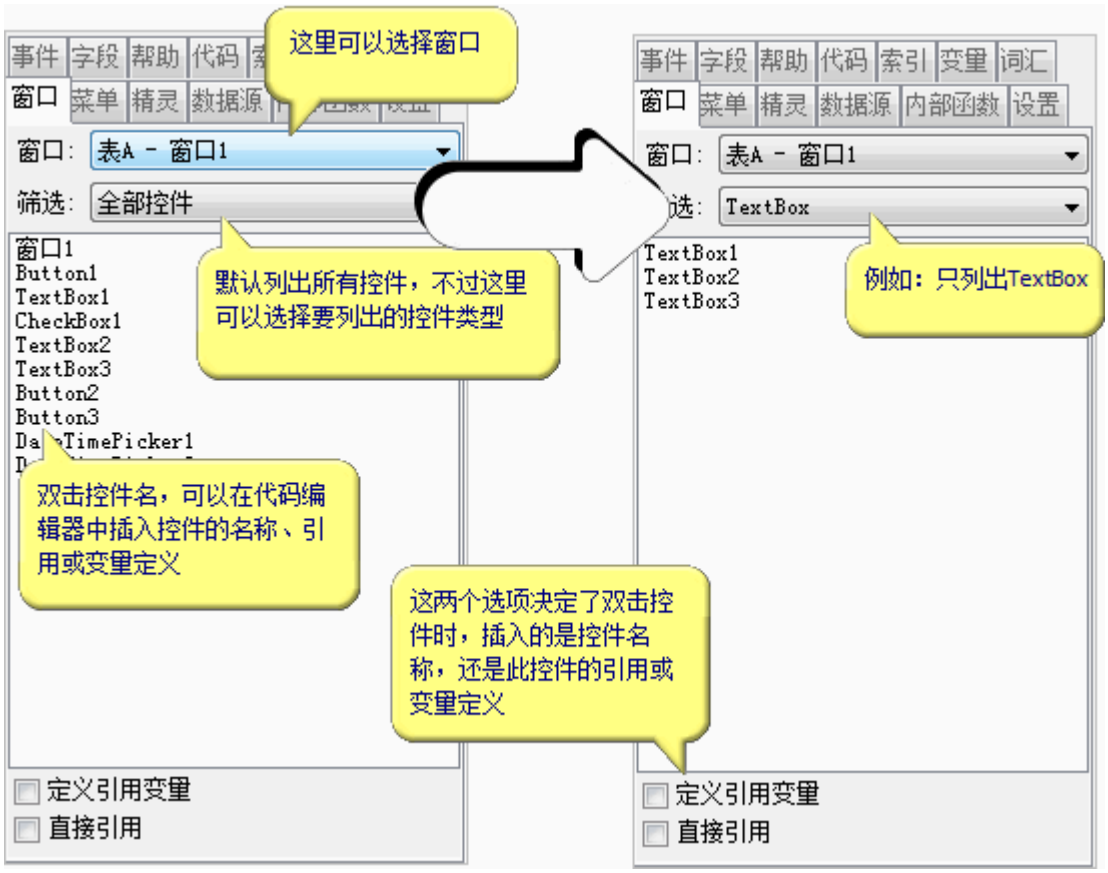
同样,较为重要的方法和事件,我们也会用红色星号标出。

4.9.2.2 代码编辑器和窗口

在窗口中编写事件代码的时候，免不了要引用某个控件的。

这有一点麻烦，首先如果窗口控件很多，要记住每个控件的名称不容易，其次控件类型也很多，要记住每个类型的名称才能正确地定义控件变量，显然这也不太容易。

不过代码编辑器的窗口页，能够让你轻松解决这些问题。



在窗口页选择某个窗口，会自动列出该窗口的全部控件，当我们双击某个控件的时候，默认在编辑器插入控件的名称。

页面下方的两个选项，可以改变这种默认的方式，以双击“Button1”为例：

定义引用变量

如果选中这个选项，则插入：

```
Dim 变量名 As WinForm.Button = e.Form.Controls("Button1")
```

显然这样我们可以快速定义一个变量来引用按钮“Button1”，要做的只是将字符“变量名”改为你需要的名称而已。

直接引用

如果选中这个选项，则插入：

```
e.Form.Controls("Button1")
```

另外，合理地为控件设置名称很重要，显然"ButtonAdd"和“ButtonDelete”这样的名称，比起“Button1”和“Button2”，要友善得多。

智能输入

在命令窗口或代码编辑器中输入：

```
Dim btn As WinForm.Button = e.Form.Controls("Button1")
btn.
```

按 Tab 键，会自动列出 Button 所有成员供选择，按上下箭头选择合适的成员，会出现所选成员的说明，选定某个成员，按回车即可自动输入该成员。

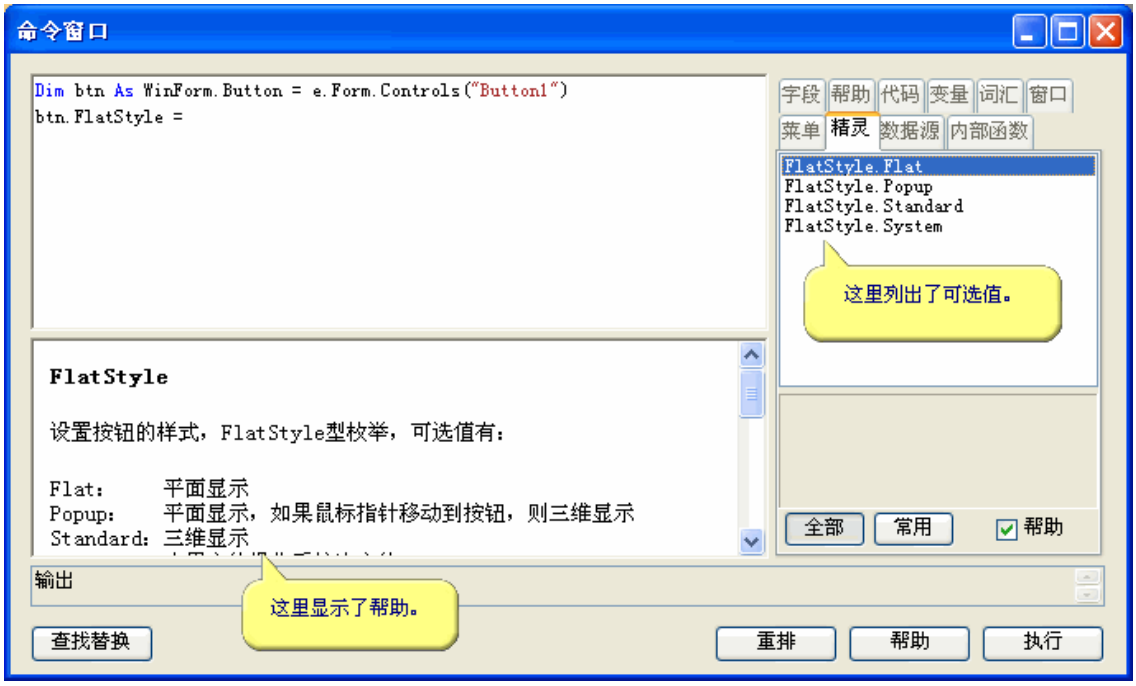
如果输入：

```
Dim btn As WinForm.Button = e.Form.Controls("Button1")
btn.e
```

然后按 Tab 键，会自动输入：

```
Dim btn As WinForm.Button = e.Form.Controls("Button1")
```

btn.Enabled
这是因为 Button 以字母 e 开头的成员只有 Enabled。
如果输入：
Dim btn As WinForm.Button = e.Form.Controls("Button1")
btn.F
然后按 Tab 键，会自动列出以 F 开头的成员供选择，选择 FlatStyle，按 Tab 键，会自动输入：
Dim btn As WinForm.Button = e.Form.Controls("Button1")
btn.FlatStyle
接着输入：
Dim btn As WinForm.Button = e.Form.Controls("Button1")
btn.FlatStyle =
然后按 Tab 键，会自动列出 FlatStyle 的所有可选值供选择：



可以看出，Foxtable 中虽然有很多控件，每一种控件都有很多属性和方法，但是我们并不需要特意去记忆这些属性和方法，只需对其有所了解即可；其实上面的智能提示和输入功能，之前我们已经介绍过，这里再次重复一次，是希望大家巩固一下，不要被窗口众多的控件、属性、方法所吓倒。

4.9.2.3 Form(窗口)

4.9.2.3.1 关于 Form

Form 表示一个窗口，类型为 WinForm.Form。
实际上所有和窗口有关的类都在命名空间 WinForm 中，在定义的时候需要加上前缀 WinForm。
例如 WinForm.Button 表示按钮类，WinForm.TextBox 表示文本框类。
我们通过集合 Forms 来获得指定名称的 Form。
每一个 Form 都属于一个表，我们可以通过其 TableName 属性，获得 Form 所属表的名称。

通过 Form 的 Open 方法，可以打开窗口。
例如打开名为“窗口 1”的窗口：

```
Dim frm As WinForm.Form
frm = Forms("窗口 1")
frm.Open()
```

打开订单表的所有窗口：

```
For Each frm As WinForm.Form In Forms
    If frm.TableName = "订单" Then
        frm.Open()
    End If
Next
```

Form 还有一个 Show 方法，这个方法和 Open 一样，可以用于打开窗口，但是 Show 会确保打开的窗口可见。
例如：

```
Forms("窗口 1").Show()
```

例如对于并列式窗口，Open 方法只是打开窗口而已，而 Show 不仅会打开，而且会自动切换到该窗口。
此外如果需要显示已经打开，但处于自动隐藏状态的停靠窗口时，只能用 Show 方法。
停靠窗口和并列窗口都是属于特定表的，不过利用 OpenTo 方法，可以在另一个表打开窗口。
例如假定窗口 1 是一个停靠窗口，属于表 A，可以利用 OpenTo 方法在表 B 打开窗口 1，使其停靠在表 B 的周围：

```
Forms("窗口 1").OpenTo("表 B")
```

如果要关闭窗口，可以用 Form 的 Close 方法。
例如：

```
Forms("窗口 1").Close()
```

Form 最常用的属性是 Controls，通过该属性可以返回指定名称的控件，例如假定窗口 1 中有一个按钮，名称为"OK"，要获得该按钮的代码是：

```
Dim btn As WinForm.Button
Btn = Forms("窗口 1").Controls("OK")
```

Form 还提供了两个属性，用于控制定时器

- TimerEnabled
  逻辑型，是否启用计时器。

- TimerInterval
  整数型，设定计时器的执行间隔，单位为毫秒。

4.9.2.3.2 快速参考

属性名	说明
Name	字符型，返回窗口名称
Text	字符型，返回或设置窗口标题

BackColor	返回或设置窗口的背景颜色，请参考：关于颜色
ExtendSize	返回或设置窗口的扩展尺寸
AutoScroll	逻辑型，当窗口中的控件位置或大小超出窗口范围时，是否自动出现滚动条
Image	返回或设置窗口的背景图片
ImageLayout	返回或设置窗口背景图片的排列方式
Controls	控件集合，包括窗口所有的控件，可以返回指定名称的控件
Children	返回顶层控件集合。
Opened	逻辑型，如果窗口已经打开，则返回 True，否则返回 False
Visible	逻辑型，仅对已经打开的独立型窗口有效，设为 False，临时隐藏窗口，设为 True，重新显示已经隐藏的窗口
TableName	字符型，返回窗口的所有者表的表名
TimerEnabled	逻辑型，是否启用计时器
TimerInterval	整数型，设定计时器的执行间隔，单位为毫秒
ActiveControl	返回有输入焦点的控件，如果没有任何控件获得输入焦点，则返回 Nothing
Parent	返回包含此控件的容器
AcceptButton	返回或设置“接受”按钮的名称
CancelButton	返回或设置“取消”按钮的名称
Category	字符型，返回窗口所属分类

方法

方法名	说明
Open	打开窗口
Show	打开并显示窗口
Close	关闭窗口
SetBounds	用于设置窗口位置和大小，仅对独立窗口和模式窗口有效。
OpenTo	在非所有者表打开窗口。
DisableXButton	禁用模式窗口和独立窗口的标题栏的关闭按钮。
StopRedraw	暂停绘制窗口
ResumeRedraw	恢复绘制窗口
ClearErrors	清除所有控件的错误信息。

4.9.2.3.3 ActiveForm

这是 Forms 集合的一个属性，用于返回活动窗口，如果没有活动窗口，则返回 Nothing。

示例

```
Dim frm As WinForm.Form = Forms.ActiveForm
If frm IsNot Nothing Then
    MessageBox.Show("活动窗口为:" & frm.Name)
Else
    MessageBox.Show("无活动窗口!" )
End If
```

提示：上面的代码在命令窗口执行的时候，会一直提示无活动窗口，理由是什么，大家可以自己想想。

4.9.2.3.4 事件

4.9.2.3.4.1 AfterLoad

生成窗口后执行，通常在此对窗口和控件进行初始化设置。

e 参数属性：

Form：表示触发事件的窗口

示例

请首先在窗口中加入一个 Label(标签)，并改名为“Welcome”。

然后将 AfterLoad 事件设为：

```
Dim lbl As WinForm.Label
lbl = e.Form.Controls("Welcome")
lbl.Text = "今天是" & Date.Today() & "，Foxtable 开发团队欢迎您！"
```

提示：

在事件代码中引用本窗口，请尽量用 e 参数：

```
lbl = e.Form.Controls("Welcome")
```

而不要直接使用窗口名称，例如：

```
lbl = Forms("窗口 1").Controls("Welcome")
```

如果万一之后因为某种原因，需要修改窗口名称，那么你将不得不修改所有事件代码。

当然如果引用的不是本窗口，将不得不通过窗口名称引用。

4.9.2.3.4.2 BeforeClose

在关闭窗口前执行。

e 参数属性：

Form：　　表示要关闭的窗口

Cancel：　　逻辑型，设为 True，将禁止关闭窗口。

CloseMode：　整数型，如果是通过 Form 的 Close 方法关闭窗口，则返回 1，否则返回 0。

如果任何时候 Cancel 参数都返回 True 的话，那么窗口将永远不能关闭，直到强行中止进程，所以在代码中使用 Cancel 参数的时候，一定要慎重。

示例

请在窗口中加入两个 DateTimePicker(日期输入框)，分别命名为 StartDate 和 EndDate。

然后将 BeforeClose 事件代码设为：

```
Dim sd As WinForm.DateTimePicker
Dim ed As WinForm.DateTimePicker
sd = e.Form.Controls("StartDate")
ed = e.Form.Controls("EndDate")
If sd.Value Is Nothing OrElse ed.Value Is Nothing Then
    MessageBox.Show("请输入起始日期和终止日期！","提示",MessageBoxButtons.OK ,MessageBoxIcon.Information)
  e.Cancel = True
End If
```

假定两个日期输入框分别绑定到两个 var 变量：StartDate 和 EndDate，那么上述代码可以简化为：

```
If Vars("StartDate") Is Nothing OrElse Vars("EndDate") Is Nothing Then
    MessageBox.Show("请输入起始日期和终止日期！","提示",MessageBoxButtons.OK ,MessageBoxIcon.Information)
  e.Cancel = True
End If
```

4.9.2.3.4.3 TimerTick

窗口计时器代码，根据设定的间隔执行。

e 参数属性：

Form：表示触发事件的窗口。

示例

本示例可以参考 CaseStudy 目录下的示例文件：窗口计时器.table

窗口中加入一个标签，要求标签不停地闪烁，而且能够通过按钮控制闪烁速度，以及暂停和开始闪烁。

首先在窗口中加入一个标签，标签文本为“警告”，另外加入四个按钮，分别是暂停、开始、慢速、快速。

然后将窗口的 TimerClick 事件代码设为：

```
Dim Lbl As WinForm.Label = e.Form.Controls("Label1")
If Lbl.ForeColor = Color.Red Then
  Lbl.ForeColor = Color.Black
Else
  Lbl.ForeColor = Color.Red
End If
```

四个按钮的 Click 事件代码分别设为：

按钮名称	代码
暂停	e.Form.TimerEnabled = False
开始	e.Form.TimerEnabled = True
慢速	e.Form.TimerInterval = 1000
快速	e.Form.TimerInterval = 500

4.9.2.3.4.4 AfterClose

在关闭窗口时执行，此时还可以引用窗口中的控件。
e 参数属性：
Form：表示触发事件的窗口。

4.9.2.3.4.5 SizeChanged

在窗口大小发生变化后执行。
e 参数属性：
Form：表示触发事件的窗口。

4.9.2.3.5 遍历控件

一般用户可以忽略本节的内容。
遍历控件
所有的控件都是派生自 WinForm.Control 类型，所以这个类型的变量可以引用任何类型的控件，例如：

```
Dim c1 As Winform.Control
Dim c2 As Winform.Control
c1 = e.Form.Controls("Button1")
c2 = e.Form.Controls("TextBox1")
```

如果要遍历窗口中的所有控件，只能使用 WinForm.Control 类型的变量，例如：

```
For Each c As Winform.Control In e.Form.Controls
    c.BackColor = Color.Green
Next
```

关于派生
前面提到所有控件都是派生自 WinForm.Control 类型，初学者可能不能理解“派生”的意义。
以汽车作为例子：汽车是最基本的类型，它定义了汽车最基本的属性和方法，然后在汽车这个类型的基础上，派生出轿车、火车、救护车、消防车等类型，这些类型定义继承了汽车这个基本类型的所有属性，同时增加了自己独有的属性和方法。
对于窗口中的控件来说，WinForm.Control 是最基本的类型，它定义了作为控件最基本的属性和方法，其它所有控件都派生于这个基本类型，在 WinForm.Control 的基础上增加了自己独有的属性和方法。
我们很少需要直接使用 WinForm.Control 类型的变量，而是直接使用 WinForm.Button 这种直接表明控件类型的变量，因为前者无法访问 WinForm.Button 专有的属性和方法。
示例：
下面的代码在编译的时候，会提示 WinForm.Control 没有 Value 属性：

```
Dim b As WinForm.Control
b = e.Form.Controls("TextBox1")
b.BackColor = Color.Red
b.Value = "123" '这里不能通过编译
```

如果改用 WinForm.TextBox 类型的变量，则一切正常：

```
Dim b As WinForm.TextBox
b = e.Form.Controls("TextBox1")
b.BackColor = Color.Red
b.Value = "123"
```

这是因为 BackColor 属性是 WinForm.Control 类型定义好的，既可以通过 WinForm.Control 类型的变量访问，也可以通过 WinForm.TextBox 类型的变量访问，因为后者继承了前者的全部属性和方法。

而 Value 属性是 WinForm.TextBox 类型定义的，所以不能通过 WinForm.Control 类型的变量访问。

唯一的例外就是需要遍历控件的时候，此时只有使用 WinForm.Control 类型的遍历变量，才能正常引用各种类型的控件；如果你用 WinFom.Button 这种代表特定控件类型的变量，如果遇到 其它控件类型，例如 WinForm.TextBox 的时候，肯定要出错。

判断控件类型

如果我希望清除窗口中所有文本框的内容，显然只能使用遍历才能完成这样的任务。

但是遍历控件的时候，只能使用 WinForm.Control 类型的变量，但是这样的变量又不能访问文本框的 Value 属性。

所以只能在遍历控件的时候判断控件类型，然后用具体的控件类型变量来引用控件：

```
For Each c As WinForm.Control In e.Form.Controls
    If Typeof c Is WinForm.TextBox Then '判断控件是否是文本框
        Dim t As WinForm.TextBox = c '使用特定类型的变量引用控件
        t.Value = Nothing
    End If
Next
```

判断变量是否是某类型的语法是：

Typeof 变量 Is 类型

如果变量是指定的类型则返回 True，否则返回 False。

本节介绍的派生原理，不仅适合窗口设计，也适合今后要介绍的菜单设计。

4.9.2.4 控件参考

4.9.2.4.1 通用成员

4.9.2.4.1.1 关于通用成员

前面提到，所有控件都派生自 WinForm.Control，所以不管何种类型的控件，都具备 WinForm.Control 类型的全部成员，包括属性、方法和事件。

所谓通用成员，就是在 WinForm.Control 类型中定义好的属性、方法和事件，这些成员对于任何类型的控件都有效。

为了便于大家学习，避免不必要的重复，我们先介绍通用成员，然后再逐个介绍每个控件独有的属性、方法和事件。

4.9.2.4.1.2 通用属性

4.9.2.4.1.2.1 Enabled

逻辑型，返回或者设置控件是否可用。
默认为 True，如果设为 False，控件将不会对用户的操作作出响应，也就是不可用。
示例
例如窗口中有一个 Button(按钮)的名称为“btnAdd”，希望该 Button 对于普通用户不可用，可以在窗口的 AfterLoad 事件中加入如下的代码：

```
If User.Type = UserTypeEnum.User Then
    e.Form.Controls("btnAdd").Enabled = False
End If
```

4.9.2.4.1.2.2 Visible

逻辑型，返回或者设置控件是否可见。
默认为 True，设为 False，控件将被隐藏。
示例
例如窗口中有一个 TextBox(文本框)的名称为“txtName”，如果当前用户分组不是经理级别，希望打开窗口时，自动隐藏该 TextBox，可在窗口的 AfterLoad 事件中加入如下的代码：

```
If User.Group <> "经理" Then
    e.Form.Controls("txtName").Visible = False
End If
```

4.9.2.4.1.2.3 ReadOnly

是否锁定控件，禁止用户编辑其内容，也就是只读。
BooleanEnum 型枚举，包括以下可选值：

- Default
  如果绑定的表、列、行，这三者有任何一个处于锁定状态，则控件的内容不能被编辑，否则可正常编辑。
  也就是由绑定数据源的锁定状态决定控件是否能编辑，这也是默认的设置。

- True
  禁止编辑控件的内容。

- False
  可以编辑控件的内容。

示例
例如窗口中有一文本框，如果用户分组为录入员，则禁止编辑其内容，否则由绑定数据源的锁定状态决定是否能编辑。

假定文本框的名称为"txtSale"，为实现此要求，可以在窗口的 AfterLoad 事件中设置如下代码：

```
Dim txtBox As WinForm.TextBox
txtBox = e.Form.Controls("txtSale")
If User.Group = "录入员" Then
    txtBox.ReadOnly = BooleanEnum.True
Else
    txtBox.ReadOnly = BooleanEnum.Default
End If
```

提示：不是所有的控件都是可以编辑的，此类控件也有 ReadOnly 属性，但是不会发生任何作用。

4.9.2.4.1.2.4 BindingField

字符型，返回或设置控件的绑定字段。

示例一

```
Forms("窗口 1").Controls("TextBox1").BindingField = "产品.产品名称"
Forms("窗口 1").Controls("TextBox2").BindingField = "产品.订单.数量"
```

上述代码将 TextBox1 绑定到“产品”表的“产品名称”列，将 TextBox2 绑定到关联表“产品.订单”的“数量”列。

示例二

```
Dim s As String = e.Form.Controls("TextBox1").BindingField
If s > "" Then
    Dim p As Integer = s.LastIndexOf(".")
    Dim t As String = s.SubString(0, p)
    Dim c As String = s.SubString(p + 1)
    MessageBox.Show("绑定表: " & t & "  绑定列: " & c)
End If
```

上述代码获得 TextBox1 绑定的表名和列名。

提示：并非所有的控件都可以绑定的，此类控件虽然也有 BindingField 属性，但是不会发生任何作用。

示例三

如果绑定到 Var 变量，格式为“全局变量.变量名”，例如要绑定到名为“起始日期”的 Var 变量：

```
e.Form.Controls("TextBox1").BindingField = "全局变量.起始日期"
```

4.9.2.4.1.2.5 ToolTip

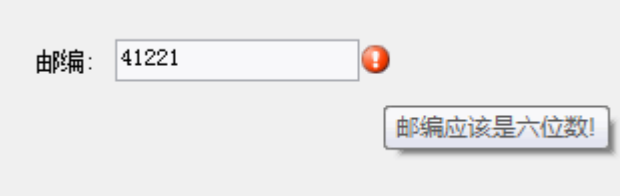
字符型，返回或设置控件的提示。

如果控件设置了提示信息，当鼠标移到控件位置时，将自动出现一个浮动窗口，用于显示提示信息：



4.9.2.4.1.2.6 Error

字符型，返回或设置控件的错误提示。
如果给控件设置了错误提示，会在控件右边显示一个红色小图标，当用户单击或鼠标停留在上面的时候，会显示该控件的错误信息。



虽然 Error 的意思是错误提示，但是并非说只能用于错误提示，只要你愿意，作为普通的信息提示也是可以的。

4.9.2.4.1.2.7 Left

整数型，返回或设置控件水平位置，单位为像素。

4.9.2.4.1.2.8 Top

整数型，返回或设置控件的垂直位置，单位为像素。

4.9.2.4.1.2.9 Width

整数型，返回或设置控件的宽度，单位为像素。

4.9.2.4.1.2.10 Height

整数型，返回或设置控件的高度，单位为像素。

4.9.2.4.1.2.11 BackColor

返回或者设置控件的背景颜色。
请参考：关于颜色

4.9.2.4.1.2.12 ForeColor

返回或设置控件的字体颜色。
请参考：关于颜色

4.9.2.4.1.2.13 Font

返回或者设置控件的字体。
请参考：关于字体

4.9.2.4.1.2.14 TopMost

当多个控件的位置重叠时，将此属性设为 True，可以确保控件处于最顶端，不会被其他控件遮挡。

4.9.2.4.1.2.15 BackMost

当多个控件的位置重叠时，将此属性设为 True，可以确保控件处于最底端，不会遮挡其他控件。

4.9.2.4.1.2.16 ImeMode

设置控件的中文输入法状态，BooleanEnum 型枚举，有三个可选值：
Default：保持当前输入法状态不变
True： 自动打开中文输入法
False： 自动关闭中文输入法

4.9.2.4.1.2.17 Focused

逻辑型，如果输入焦点位于此控件上，则返回 True，否则返回 False。

4.9.2.4.1.2.18 CausesValidation

逻辑型，是否触发 Validateing 事件，默认为 True，如果设置为 False，会导致如下结果：
1、离开此控件时，不会触发此控件的 Validating 事件。
2、离开其它控件选择此控件时，不会触发其它控件的 Validating 事件。
在窗口设计器中，选择某个控件，通常会有一个"执行验证"的属性，对应的就是本属性。

4.9.2.4.1.3 通用方法

4.9.2.4.1.3.1 Select

选择该控件。
示例
窗口中有一个按钮，希望单击该按钮能够增加一行，然后将输入焦点移到一个名为“txtName”的文本框中。
为此，可将该按钮的 Click 事件代码设为：

```
Tables("订单").AddNew()
e.Form.Controls("txtName").Select()
```

4.9.2.4.1.3.2 ShowToolTip

在此控件位置显示一个提示信息，直到执行 HideToolTip 方法，否则这个提示信息会一直显示。
我们知道通过 ToolTip 属性可以设置控件的提示信息，但是这个提示信息只有鼠标经过并停留在这个控件的时候，才会显示。

有的时候我们希望强行在某个控件显示一个提示信息，直到操作结束后才隐藏这个提示信息，此时就需要用 ShowToolTip 方法。
语法：
ShowToolTip(ToolTip, Duration)
ToolTip: 要显示的提示信息
Duration：可选参数，显示持续的时间，单位为毫秒，如果省略，则一直显示，直到执行 HideToolTip 方法。
示例：

```
Dim txt As WinForm.TextBox = e.Form.Controls("TextBox1")
txt.ShowToolTip("请输入六位数的邮政编码!")
```

4.9.2.4.1.3.3 HideToolTip

隐藏此控件的提示信息。
示例

```
Dim txt As WinForm.TextBox = e.Form.Controls("TextBox1")
txt.HideToolTip()
```

4.9.2.4.1.3.4 SetBounds

统一设置控件的位置和大小。
语法
SetBounds(x, y, Width, Height)
x: 控件的水平位置
y: 控件的垂直位置
Width: 控件宽度
Height: 控件高度
上述参数的单位为像素。

4.9.2.4.1.3.5 ReadValue

从绑定字段或者变量中读取数据到控件中。
此方法很少使用。

4.9.2.4.1.3.6 WriteValue

将控件中的内容写入到绑定字段或者变量中。
如果控件已经绑定到某字段，那么当输入焦点离开此控件的时候，其内容会自动写入到字段中。
所以这个方法是不常用的，除非需要提前将控件内容写入字段。

4.9.2.4.1.4 通用事件

4.9.2.4.1.4.1 Click

用鼠标单击控件的时候执行。
e 参数属性:
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
示例：
希望单击某个 Button(按钮)，能够在当前表增加一行，可以将该 Button 的 Click 事件代码设为:

```
CurrentTable.AddNew()
```

如果希望单击该按钮移到下一行：

```
With CurrentTable
    .Position = .Position + 1
End With
```

4.9.2.4.1.4.2 Enter

进入控件后执行。
e 参数属性:
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
示例
假定有一个 TextBox(文本框)，希望进入该 TextBox 的时候，判断当前用户分组是否是经理，如果不是，则不允许编辑。
可将该 TextBox 的 Enter 事件设为：

```
If User.Group = "经理" Then
    e.Sender.ReadOnly = BooleanEnum.False
Else
    e.Sender.ReadOnly = BooleanEnum.True
End If
```

Readonly 属性我们会在接下来的章节介绍。

4.9.2.4.1.4.3 Leave

离开控件的后执行。
e 参数属性:
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.1.4.4 Validating

离开控件之前执行，通常用于验证控件的内容。
e 参数属性:
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
Cancel: 逻辑型，设为 True 禁止离开控件。
示例
如果将某个 DateTimePicker(日期输入框)的 Validating 事件设置为：

```
If e.Sender.Value Is Nothing Then
    e.Cancel = True
End If
```

这样直到你在该 DateTimePicker 中输入内容，否则你将不能离开它去选择其他控件。
但是验证也有例外的时候，例如窗口中通常有一个取消按钮，单击这个按钮，最好不用执行任何控件的 Validating 事件，否则我明明需要取消操作，却还得在取消之前输入一些内容，这不符合通常的操作习惯。为解决这个问题，只需在窗口设计的时候，将这个按钮的"执行验证"属性设为 False 即可，该属性位于“杂项”分组中。

4.9.2.4.1.4.5 DataFormat

从绑定字段(列)或变量中读取数据的时候执行。
e 参数属性:
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
Value：　返回控件的值，也就是从字段(列)或变量中读取的值。
Text：　用于设置控件要显示的内容。
DataFormt 事件主要应用在票据设计中，后面会有专门的章节讲述。
示例
假定某个 TextBox 控件绑定到了金额列，希望能够按照中文大写形式显示金额，只需将该控件的 DataFormat 事件设为：

```
e.Text = CUMoney(e.Value)
```

如果金额是 14250，那么 TextBox 显示的内容为：壹万肆仟贰佰伍拾元整，如果控件获得焦点，那么显示的值依然为 14250，这样就不会影响正常的数据录入。

4.9.2.4.1.4.6 DataParse

向绑定字段(列)或者变量中写入数据的时候执行。
e 参数属性:
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
Text:　返回用户输入的内容
Value：　返回或设置写入的值

示例

假定在某个文本框中，希望通过输入代码来输入国家名称，可将 DataParse 事件代码设为：

```
If e.Text = "cn" Then
    e.Value = "中国"
ElseIf e.Text = "usa" Then
    e.Value = "美国"
ElseIf e.Text = "jp" Then
    e.Value = "日本"
Else
    e.Value = e.Text
End If
```

这样用户在文本框输入"cn"，离开此文本框时，"cn"会被自动替换为"中国"。

4.9.2.4.1.4.7 DoubleClick

用鼠标双击控件后执行。

e 参数属性:

Sender：返回触发事件的控件

Form:　返回控件所在的窗口

4.9.2.4.1.4.8 KeyPress

按下某一个字符键的时候执行。

e 参数属性:

Sender：　返回触发事件的控件

Form:　返回控件所在的窗口

KeyChar: 返回所按下的字符。

Cancel：逻辑型，设为 True，取消此次按键输入。

示例一

例如对于某个 TextBox(文本框)，希望禁止输入符号和标点，同时希望将输入的小写字母自动转换为大写，可设置 KeyPress 代码如下：

```
If Char.IsPunctuation(e.KeyChar) OrElse Char.IsSymbol(e.KeyChar) Then
    e.Cancel = True
ElseIf Char.IsLower(e.KeyChar) Then
    e.Cancel = True
    e.Sender.SelectedText = e.KeyChar.ToUpper
End If
```

参考：SelectedText

示例二

例如对于某个 TextBox(文本框)，希望只能输入数字和小数点，可设置 KeyPress 代码如下：

```
If Char.IsNumber(e.KeyChar) OrElse e.KeyChar = "." OrElse Char.IsControl(e.keychar) Then
```

```
Else
    e.Cancel   = True
End If
```

4.9.2.4.1.4.9 KeyDown

按下任何一个按键的时候执行。
e 参数属性:
Sender： 返回触发事件的控件
Form: 返回控件所在的窗口
KeyCode: Keys 枚举型，表示按键的代码
Alt: 逻辑型，判断是否同时按下了 Alt 键
Control: 逻辑型，判断是否同时按下了 Ctrl 键
Shift: 逻辑型，判断是否同时按下了 Shift 键
Cancel: 逻辑型，设为 True，取消此次按键
示例
例如希望在某个 TextBox 中，同时按下 Ctrl 和 F11 键时，会在当前位置插入符号“Ω”，可设置 KeyDown 事件如下：

```
If e.Control = True AndAlso e.KeyCode = Keys.F11 Then
    e.Sender.SelectedText = "Ω"
    e.Cancel = True
End If
```

参考：Keys 枚举

4.9.2.4.1.4.10 KeyUp

松开一个按键的时候执行。
e 参数属性:
Sender： 返回触发事件的控件
Form: 返回控件所在的窗口
KeyCode: Keys 枚举型，表示按键的代码
Alt: 逻辑型，判断是否同时按下了 Alt 键
Control: 逻辑型，判断是否同时按下了 Ctrl 键
Shift: 逻辑型，判断是否同时按下了 Shift 键
Cancel: 逻辑型，设为 True，取消此次按键

4.9.2.4.1.4.11 MouseEnter

鼠标进入控件的时候执行。
e 参数属性:
Sender：返回触发事件的控件
Form: 返回控件所在的窗口

4.9.2.4.1.4.12 MouseLeave

鼠标离开控件的时候执行。
e 参数属性:
Sender：返回触发事件的控件
Form: 返回控件所在的窗口

4.9.2.4.1.4.13 MouseHover

鼠标停留在控件上的时候执行。
e 参数属性:
Sender：返回触发事件的控件
Form: 返回控件所在的窗口

4.9.2.4.1.4.14 MouseDown

按下鼠标按键的时候执行
e 参数属性:

- Sender
  返回触发事件的控件

- Form
  返回控件所在的窗口

- X
  鼠标的水平位置

- Y
  鼠标的垂直位置。

- Button
  MouseButtons 型枚举，用于判断按下的是哪一个按钮，可能的值有：
  Left: 鼠标左按钮。
  Middle: 鼠标中按钮。
  Right: 鼠标右按钮。

4.9.2.4.1.4.15 MouseUp

松开鼠标按键的时候执行
e 参数属性:

- Sender
  返回触发事件的控件

- Form
  返回控件所在的窗口

- X
  鼠标的水平位置

- Y
  鼠标的垂直位置。

- Button
  MouseButtons 型枚举，用于判断松开的是哪一个按钮，可能的值有：
  Left:　鼠标左按钮。
  Middle:　鼠标中按钮。
  Right:　鼠标右按钮。

4.9.2.4.1.4.16 MouseMove

在控件上移动鼠标的时候执行
e 参数属性:

- Sender
  返回触发事件的控件

- Form
  返回控件所在的窗口

- X
  鼠标的水平位置

- Y
  鼠标的垂直位置。

- Button
  MouseButtons 型枚举，用于判断按下的是哪一个按钮，可能的值有：
  Left:　鼠标左按钮。
  Middle:　鼠标中按钮。
  Right:　鼠标右按钮。

4.9.2.4.1.5 引用窗口和控件

在窗口或控件的事件代码中，如果要引用本窗口，请尽量用 e 参数：e.Form。

例如窗口中有一个按钮，希望单击这个按钮能够关闭本窗口，该按钮的 Click 事件代码应该设置为：

```
e.Form.Close()
```

如果通过窗口名称来引用：

```
Forms("窗口 1").Close()
```

一旦之后因为某种原因，需要修改窗口名称，那么你将不得不修改相关事件代码。

同样在控件的事件代码中，如果要引用控件本身，也应该使用 e 参数：e.Sender，例如：

```
If User.Group = "经理" Then
    e.Sender.ReadOnly = BooleanEnum.False
Else
    e.Sender.ReadOnly = BooleanEnum.True
End If
```

如果采用控件名称来引用，那么一旦需要修改控件名称，就必须同步修改代码

4.9.2.4.1.6 通用成员和代码编辑器

假定窗口有一个名为“ComboBox1”的组合框，在代码编辑器输入：

```
e.Form.Controls("ComboBox1").
```

然后按 Tab 键，你会发现精灵只会列出通用成员，而不会列出 ComboBox 特有的成员供选择，这是因为 Controls 集合返回的是基本类型 WinForm.Control。

为了能够列出 ComboBox 的全部成员，我们可以用一个 ComboBox 类型的变量引用 Controls 返回的控件，例如：

```
Dim cmb As WinForm.ComboBox
cmb = e.Form.Controls("ComboBox1")
cmb.
```

现在按 Tab 键，将正常列出 ComboBox 的全部成员供选择。

上面的代码是以 ComboBox 为例，实际上对于任何类型的都是如此，只有用特定控件类型的变量引用 Controls 集合返回的控件，代码编辑器才会列出其全部成员。

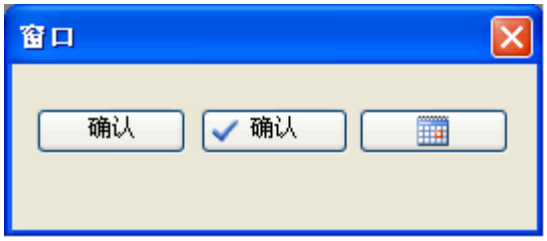
4.9.2.4.2 常用类

4.9.2.4.2.1 Button(按钮)

4.9.2.4.2.1.1 关于 Button

Button 表示按钮，用于单击执行某一操作。

通过设置按钮的标题和图标，可以设计出各种风格的按钮：

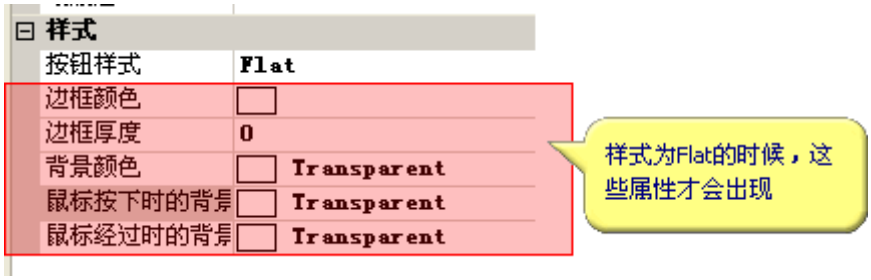




按钮样式

有四种类型的按钮样式，分别是 Flat、Popup、System、Standard。

其中样式 Flat 较为特别，选择此样式后，还可以设置按钮的背景、边框等属性，这样就可以设计出一些较为特殊的按钮，例如没有边框的透明按钮。



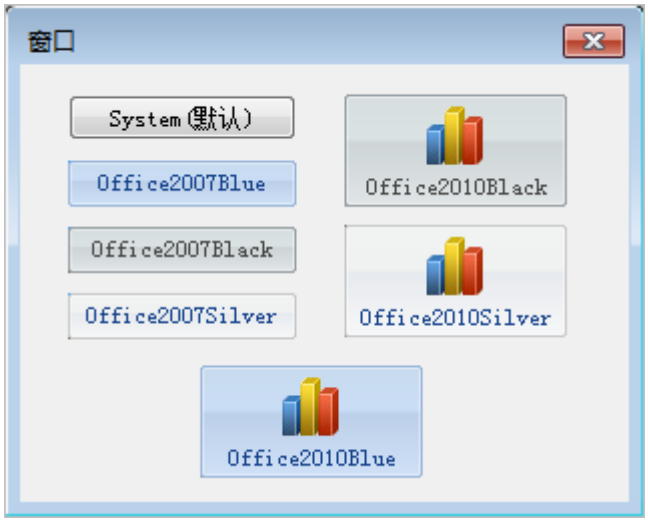
提示：要使某个控件的背景透明，应该将其背景颜色设置为 Transparent

按钮风格

如果按钮的外观是 Standard，那么还可以设置按钮的样式，默认的样式为 System：



下图是不同样式的按钮：



4.9.2.4.2.1.2 属性

4.9.2.4.2.1.2.1 Text

返回或设置按钮的标题。

4.9.2.4.2.1.2.2 Image

返回或设置按钮的图标。
可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
示例

```
Dim btn As WinForm.Button
btn = Forms("窗口 1").Controls("Button1")
btn.Image = GetImage("Accept.Ico")
```

4.9.2.4.2.1.2.3 TextAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.1.2.4 ImageAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.1.2.5 TextImageRelation

设置按钮标题和图标的相对位置，TextImageRelation 型枚举，可选值有：
Overlay： 重叠
ImageAboveText： 图像在文本的上方
ImageBeforeText： 图像在文本的左方
TextAboveImage： 文本在图像的上方
TextBeforeImage： 文本在图像的左方

4.9.2.4.2.1.2.6 VisualStyle

用于设置按钮的样式，IVisualStyle 型枚举，可选值：
Custom
Office2007Black
Office2007Blue
Office2007Silver
Office2010Black
Office2010Blue
Office2010Silver
System
示例

```
Dim btn As WinForm.Button = e.Form.Controls("Button1")
btn.VisualStyle = IVisualStyle.Office2007Blue
```

4.9.2.4.2.1.2.7 FlatStyle

设置按钮的样式，FlatStyle 型枚举，可选值有：
Flat： 平面显示
Popup： 平面显示，如果鼠标指针移动到按钮，则三维显示
Standard：三维显示

System： 由用户的操作系统决定的
如果样式设置为 Flat，还可以通过以下属性进一步设置按钮的外观：
BackColor： 设置按钮的背景颜色
MouseDownBackColor：设置按下鼠标时，按钮的背景颜色
MouseOverBackColor：设置鼠标经过时，按钮的背景颜色
BorderColor： 设置按钮边框颜色
BorderSize： 设置按钮边框厚度，单位为像素
示例

```
With e.Form.Controls("Button1")
    .FlatStyle = FlatStyle.flat
    .BackColor = Color.Transparent
    .BorderSize = 0
End With
```

4.9.2.4.2.1.3 方法

4.9.2.4.2.1.3.1 PerformClick

模拟单击此按钮。
示例
假定窗口有两个按钮，单击按钮 1 执行 A 操作，单击按钮 2 执行 A、B 两项操作，我们可以将按钮 2 的 Click 事件代码设置为：

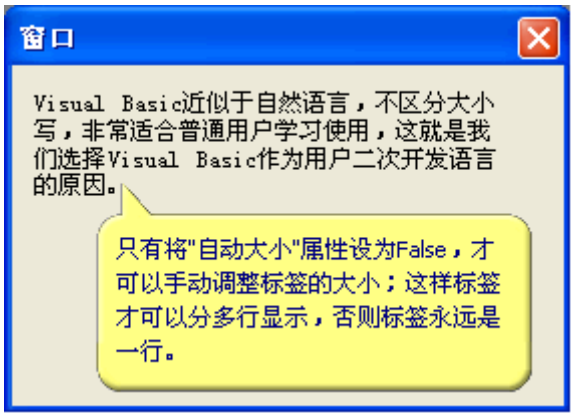
```
Dim btn As WinForm.Button = Forms("窗口 1").Controls("Button1")
btn.PerformClick()
'B 操作代码
```

这样就无需重复在两个按钮中编写 A 操作的代码，只需在按钮 1 中编写一次即可。

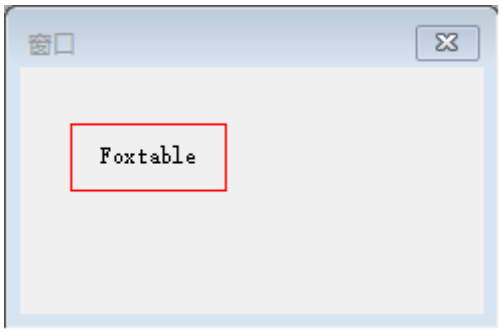
4.9.2.4.2.2 Label(标签)

4.9.2.4.2.2.1 关于 Label

Label 表示标签，通常用于显示不可编辑的字符型内容。
标签有一个很重要的属性，名为“自动大小”，默认为 True，此时标签自动根据其内容调整大小，而不可以手动调整或通过代码设置。
如果需要设置 Label 的大小，请将“自动大小”属性设为 False。



可以给 Label 设置边框和边框颜色，还可以设置文本的位置(自动大小为 False 时)，例如：



Label 有格式符号属性，可以用于设置显示格式，例如下图三个 Label 都绑定到了统一日期列，但是由于格式符号属性值不同，所以显示结果也不同：



有关格式化符号的规范，和 Format 函数相同，请参考：

格式化数字

格式化日期和时间

如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.2.4.2.2.2 属性

4.9.2.4.2.2.2.1 Text

返回或设置标签的内容。

4.9.2.4.2.2.2.2 AutoSize

逻辑型，标签是否自动调整大小，默认为 True。

4.9.2.4.2.2.2.3 TextAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.2.2.4 FormatString

字符型，返回或设置 Label 的格式符号。
示例
例如希望按照百分比格式显示数值，最多两位整数，保留两位小数：
e.Form.Controls("Label1").FormatString = "#0.00%"
有关格式化符号的规范，和 Format 函数相同，请参考：
格式化数字
格式化日期和时间
如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.2.4.2.2.2.5 BorderColor

用于设置 Label 控件的边框颜色。
例如：

```
Dim lbl As WinForm.Label = e.Form.Controls("Label1")
lbl.BorderStyle = BorderStyle.FixedSingle
lbl.TextAlign = ContentAlignment.MiddleCenter
lbl.BorderColor = Color.Red
```

提示：只有边框类型为FixedSingle的时候，才能设置边框颜色。

4.9.2.4.2.2.2.6 BorderStyle

设置分 Label 边框类型，BorderStyle 型枚举，可选值有：

Fixed3D： 立体边框

FixedSingle：单线条边框
None： 无边框
例如：

```
Dim lbl As WinForm.Label = e.Form.Controls("Label1")
lbl.BorderStyle = BorderStyle.FixedSingle
lbl.TextAlign = ContentAlignment.MiddleCenter
lbl.BorderColor = Color.Red
```

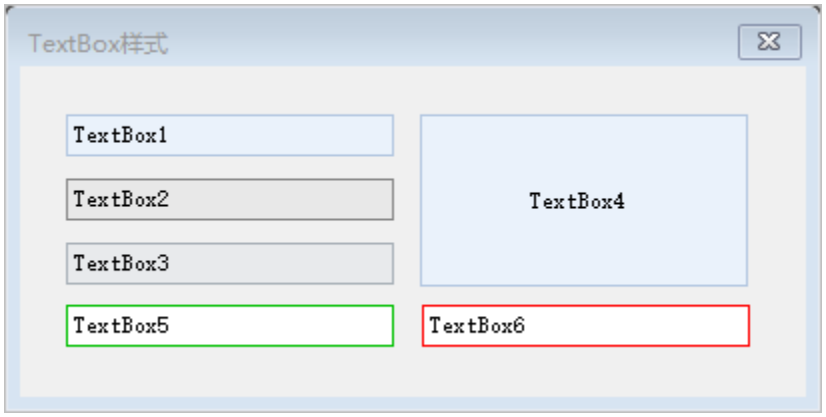
4.9.2.4.2.3 TextBox(文本框)

4.9.2.4.2.3.1 关于 TextBox

TextBox 表示用于输入字符型内容的文本框。
在默认的情况下，文本框只能输入一行内容，且不能调整高度，如果要输入多行文本，请在设计窗口的时候，将“多行文本”属性设为 True。



TextBox 可以设置样式和边框颜色，以及水平和垂直方向的文本位置：



TextBox 可以设置格式符号属性，同一个值不同的格式符号，会有不同的显示结果，如：



有关格式化符号的规范，和 Format 函数相同，请参考：

格式化数字

格式化日期和时间

如果有更复杂的显示要求，可以参考事件：DataFormat

提示：

1、设计窗口的时候，对于单行文本框，我们无法直接通过鼠标拖动调整其高度，但是可以通过属性设置窗口设置其高度属性。

2、如果想多行文本框出现滚动条以显示更多的内容，可以设置其“滚动条”属性为 Vertical。

3、设计窗口的时候，对于日期输入，尽量用 DateTimePicker 代替 TextBox，对于数值输入，尽量用 NumericComboBox 代替 TextBox。

4.9.2.4.2.3.2 属性

4.9.2.4.2.3.2.1 Value

字符型，返回或设置文本框的值。

示例一

假定窗口 1 中有一个名为"txtSerial"的文本框， 要求编号的长度必须等于 6 个字符：

```
With Forms("窗口1").Controls("txtSerial")
    Dim txt As String = .Value
    If txt = "" OrElse txt.Length <> 6  Then
        Messagebox.Show("必须输入六位数的编号!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End With
```

示例二

如果要判断输入框是否已经输入值，可以参考下面的代码：

```
With Forms("窗口1").Controls("txtSerial")
    If .Value Is Nothing Then
        Messagebox.Show("请输入编号!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End With
```

有的用户会这样编写代码：

```
With Forms("窗口1").Controls("txtSerial")
    If .Value = "" Then
        Messagebox.Show("必须输入六位数的编号!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End With
```

这种写法对于文本框(TextBox)是可以的，但是对于日期输入框(DateTimePicker)和数值输入框(NumericComboBox)，这种写法是无效的。
所以在判断输入框内容是否为空的时候，最好用第一种写法，切记切记。

4.9.2.4.2.3.2.2 MultiLine

逻辑型，是否为多行文本框。
文本框默认为单行，如果要显示多行文本，将 MultiLine 属性设置为 True 即可。

4.9.2.4.2.3.2.3 Text

字符型，返回文本框当前显示的文本内容。
文本框很少使用此属性，一般是使用 Value 属性。
注意 Text 和 Value 属性两个属性的不同：
通常两者返回的内容是相同的，但是在输入的过程中，两者会有所差别。例如有一个文本框，它最初的内容是"中国"，此时其 Text 和 Value 属性的内容都是“中国”，现在选择该文本框进行编辑，输入“中华”，此时 Text 的内容是“中华”，而 Value 的还是“中国”,输入结束，选择其它控件，此时该控件的 Text 和 Value 属性都是“中华”。通俗一点：Value 返回的是控件已经输入完成的值，而 Text 返回的是控件中显示的文本内容，它会随着编辑的进行，而不断地变化。
示例：
例如希望某文本框在输入的过程中，能够检测输入内容的长度，如果输入内容达到 4 个字符，就自动结束输入，跳转到下一个控件。
为此将文本框的 TextChanged 事件代码设置为：

```
Dim tx As WinForm.TextBox = e.Sender
If tx.Text.Length = 4 Then
    Sendkeys.Send("{TAB}") '模拟按 Tab 键，选择下一个控件
End If
```

由于 Value 属性只能返回文本框输入结束后的内容，所以上面的代码使用了 Text 属性。
总结：
实际编码的时候，都应该使用 Value 属性，而不是 Text 属性，除非想在用户输入的过程中，获得

正在输入的内容，才可以使用 Text 属性。

4.9.2.4.2.3.2.4 TextAlign

设置文本的对齐方式，HorizontalAlignment 型枚举，可选值有：
Center：居中
Left： 靠左
Right： 靠右

4.9.2.4.2.3.2.5 VisualStyle

用于设置 TextBox 的样式，IVisualStyle 型枚举，可选值：
Custom
Office2007Black
Office2007Blue
Office2007Silver
Office2010Black
Office2010Blue
Office2010Silver
System
示例

```
Dim txt As WinForm.TextBox = e.Form.Controls("TextBox1")
txt.VisualStyle = IVisualStyle.Office2007Blue
```

4.9.2.4.2.3.2.6 BorderColor

用于设置 TextBox 的边框颜色。
例如：

```
Dim tbx As WinForm.TextBox = e.Form.Controls("TextBox1")
tbx.BorderStyle = BorderStyle.FixedSingle
tbx.BorderColor = Color.Red
```

提示：只有边框类型为 FixedSingle 的时候，才能设置边框颜色。

4.9.2.4.2.3.2.7 BorderStyle

设置分 TextBox 边框类型，BorderStyle 型枚举，可选值有：

Fixed3D： 立体边框
FixedSingle：单线条边框
None： 无边框
例如：

```
Dim tbx As WinForm.TextBox = e.Form.Controls("TextBox1")
tbx.BorderStyle = BorderStyle.FixedSingle
tbx.BorderColor = Color.Red
```

4.9.2.4.2.3.2.8 SelectedText

字符型，返回或设置输入框中选定的文本。
示例
例如希望在某个 TextBox 中，同时按下 Ctrl 和 F11 键时，会在当前位置插入符号“Ω”，可设置 KeyDown 事件如下：

```
If e.Control = True AndAlso e.KeyCode = Keys.F11 Then
    e.Sender.SelectedText = "Ω"
    e.Cancel = True
End If
```

4.9.2.4.2.3.2.9 SelectionStart

整数型，返回文本框中选定字符的起始位置。

4.9.2.4.2.3.2.10 SelectionLength

整数型，返回或设置文本框中选定字符的长度。

4.9.2.4.2.3.2.11 Lines

一个字符型数组，可以返回文本框中指定行号的内容。
显然，此属性仅对多行文本框才有效。
示例：
假定窗口 1 中有一个名为"txtMemo"的多行文本框，分别显示该文本框每一行的内容：

```
With Forms("窗口 1").Controls("txtMemo")
    For i As Integer = 0 To .Lines.Length - 1
        Messagebox.Show(.Lines(i))
    Next
End With
```

4.9.2.4.2.3.2.12 MaxLength

整数型，返回或设置文本框允许输入内容的最大长度。

4.9.2.4.2.3.2.13 PasswordChar

字符型，返回或设置文本框的密码字符。
如果文本框用于输入密码，请将此属性设置为一个符号，例如“*”，这样将用“*”代替显示所输入的

密码。
示例:

```
Forms("窗口 1").Controls("TextBox2").PasswordChar = "*"
```

4.9.2.4.2.3.2.14 EditMask

字符型，返回或设置文本框的输入掩码。
可作为输入掩码使用的字符有：

字符	说明
0	数字（0 到 9，必需项，不允许使用加号 [+] 与减号 [－]）
9	数字或空格（可选项，不允许使用加号和减号）
#	数字或空格（可选项，允许使用加号和减号）
L	字母（A 到 Z，必需项）
?	字母（A 到 Z，可选项）
A	字母或数字（必需项）
a	字母或数字（可选项）
&	任一字符或空格（必需项）
C	任一字符或空格（可选项）。
. , : ; - /	小数点占位符及千位、日期与时间的分隔符
<	将所有字符转换为小写
>	将所有字符转换为大写
\	使接下来的字符以表意字符显示（例如，\A 只显示为 A）

示例
假定要求输入三个字母、一个横线、两个数字，例如“KLU-73”，可以将输入掩码设为“AAA-00”，如果要求这三个字母必须为大写，那么掩码可以设为“>AAA-00”。

4.9.2.4.2.3.2.15 FormatString

字符型，返回或设置文本框的格式符号。
示例
例如希望按照百分比格式显示数值，最多两位整数，保留两位小数：

```
Forms("窗口 1").Controls("TextBox1").FormatString = "#0.00%"
```

有关格式化符号的规范，和 Format 函数相同，请参考：
格式化数字
格式化日期和时间
如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.2.4.2.3.3 方法

4.9.2.4.2.3.3.1 Copy

复制文本框中选定内容复制到剪贴板。

4.9.2.4.2.3.3.2 Cut

复制文本框中选定内容移到剪贴板。

4.9.2.4.2.3.3.3 Paste

将文本框中选定的内容替换为剪贴板中的内容。

4.9.2.4.2.3.3.4 Undo

撤销文本框最近的一次编辑操作。

4.9.2.4.2.3.3.5 SelectAll

选定文本框中的所有内容。

4.9.2.4.2.3.3.6 ResetText

根据 Value 属性的值重置 Text 属性。

4.9.2.4.2.3.4 事件

4.9.2.4.2.3.4.1 TextChanged

控件的 Text 属性发生变化后执行。
在控件中每输入或删除一个字符，都会触发 TextChanged 事件。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
示例
假定某个文本框绑定到一个字段，我们在该文本框输入内容的时候，并不会即刻刷新字段的内容，直到你输入结束并离开该文本框，字段的内容才会刷新。
如果你希望在文本框中进行的任何编辑，均能即时显示在字段中，可以将其 TextChanged 事件代码设为：
e.Sender.WriteValue()
请参考：WriteValue

4.9.2.4.2.3.4.2 ValueChanged

控件的 Value 属性发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

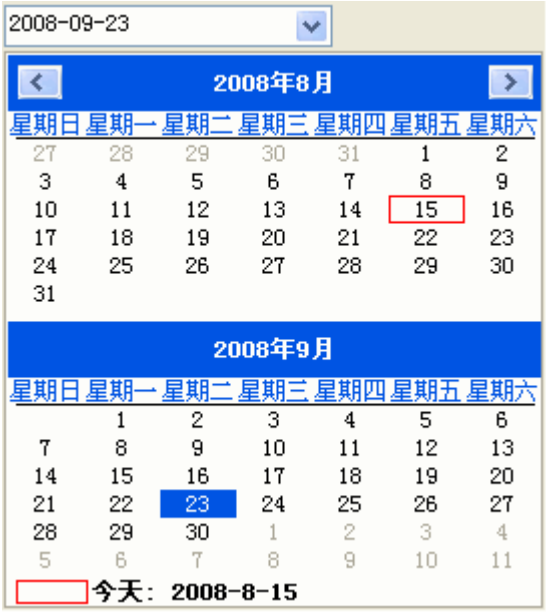
4.9.2.4.2.4 DateTimePicker(日期输入框)

4.9.2.4.2.4.1 关于 DateTimePicker

DateTimePicker 表示日期输入框，专门用于输入日期和时间。
可直接在日期输入框输入日期或时间，如果单击右边的下拉按钮，将出现一个日历用于选择输入日期：



通过设置“日历维数”属性，可以让下拉日历显示多个月份的日历：



通过“日期时间格式”属性，可以选择输入格式：



该属性有五个可选值：

Date： 标准日期格式，如"2008-12-31"

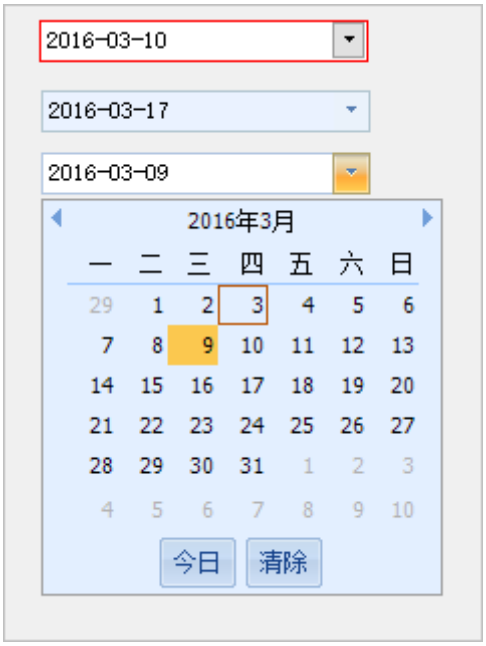
DateTime： 短日期时间格式，包括日期和时分，如"2008-12-31 9:45"

LongDateTime： 长日期时间格式，包括日期和时分秒，如"2008-12-31 9:45:30"

Time： 短时间格式，包括时分，如“12:59”

LongTime： 长时间格式，包括时分秒，如“12:59:21”

DateTimePicker 同样可以设置样式和边框颜色：



提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 DateTimePicker 的高度，但是可以通过属性设置窗口设置其高度属性。

4.9.2.4.2.4.2 属性

4.9.2.4.2.4.2.1 Value

日期型，返回或设置日期输入框的值。

4.9.2.4.2.4.2.2 DateTimeFormat

设置日期时间格式，DateTimeFormatEnum 型枚举，有五个可选值：
Date：　　标准日期格式，如"2008-12-31"
DateTime：　　短日期时间格式，包括日期和时分，如"2008-12-31 9:45"
DateLongTime：　长日期时间格式，包括日期和时分秒，如"2008-12-31 9:45:30"
Time：　　短时间格式，包括时分，如“12:59”
LongTime：　　长时间格式，包括时分秒，如“12:59:21”
示例

```
Dim dp1 As WinForm.DateTimePicker = e.Form.Controls("DateTimePicker1")
dp1.DateTimeFormat = DateTimeFormatEnum.DateTime
```

4.9.2.4.2.4.2.3 TextAlign

设置文本的对齐方式，HorizontalAlignment 型枚举，可选值有：
Center：居中
Left：　靠左
Right：　靠右

4.9.2.4.2.4.2.4 ShowDropDownButton

逻辑型，是否显示下拉日历按钮。

4.9.2.4.2.4.3 事件

4.9.2.4.2.4.3.1 ValueChanged

控件的 Value 属性发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.2.4.4 方法

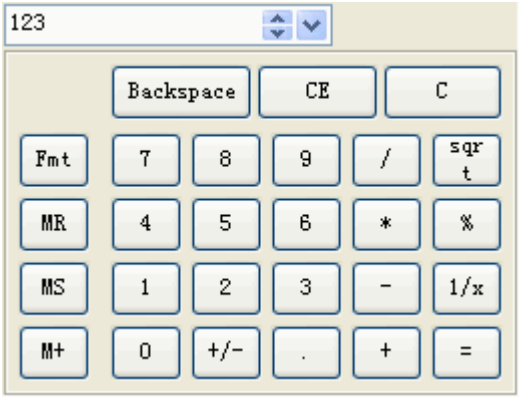
4.9.2.4.2.4.4.1 SelectAll

选定日期输入框中的所有内容。

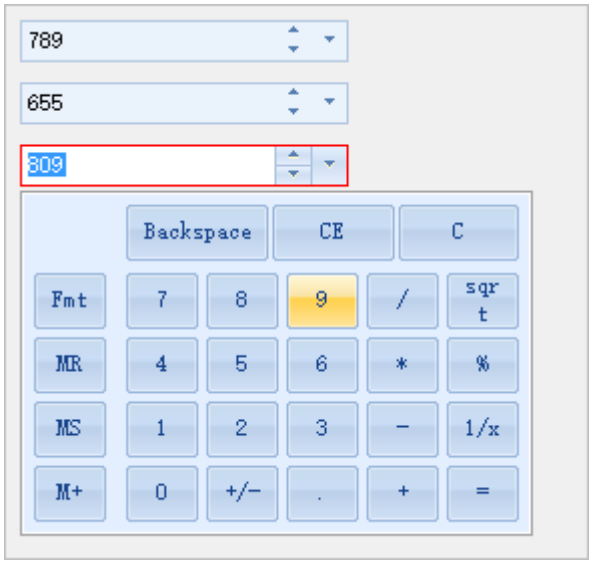
4.9.2.4.2.5 NumericComboBox(数值组合框)

4.9.2.4.2.5.1 关于 NumericComboBox

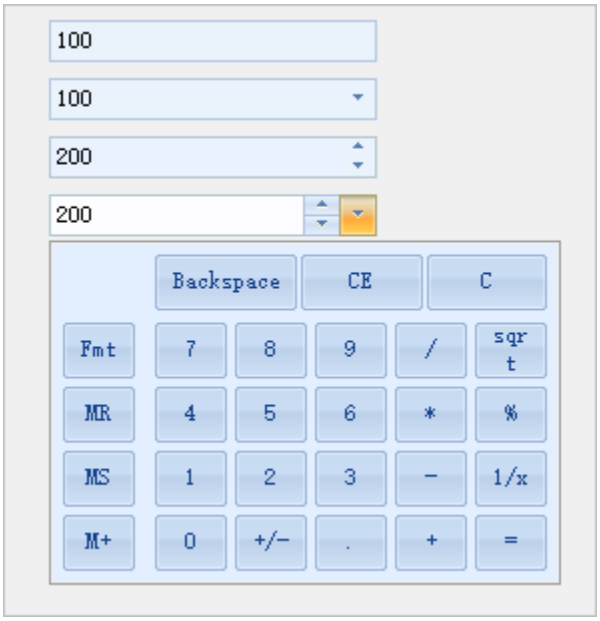
NumericComboBox 表示数值输入框，专门用于输入数值。
可直接在数值输入框中输入数值，或利用上下微调按钮调整已经输入的数值，如果单击右边的下拉按钮，将出现一个计算器用于计算和输入数值。



提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 NumericComboBox 的高度，但是可以通过属性设置窗口设置其高度属性。
NumericComboBox 可以设置边框颜色和样式：



NumericComboBox 右侧的微调按钮和下拉按钮是可以关闭的，例如下图四个输入框都是 NumericComboBox，一个只有微调按钮，一个只有下拉按钮，一个则没有显示任何按钮：



NumericComboBox 可以设置格式符号属性，同一个值不同的格式符号，会有不同的显示结果，如：



有关格式化符号的规范，和 Format 函数相同，请参考：格式化数字
如果有更复杂的显示要求，可以参考事件：DataFormat

4.9.2.4.2.5.2 属性

4.9.2.4.2.5.2.1 Value

返回或设置数值输入框的值。

4.9.2.4.2.5.2.2 TextAlign

设置文本的对齐方式，HorizontalAlignment 型枚举，可选值有：
Center：居中
Left： 靠左
Right： 靠右

4.9.2.4.2.5.2.3 FormatString

字符型，返回或设置数值输入框的格式符号。

有关格式化符号的规范，和 Format 函数相同，请参考：
格式化数字

4.9.2.4.2.5.2.4 ShowUpDownButtons

逻辑型，是否显示数值输入框的上下微调按钮。

4.9.2.4.2.5.2.5 ShowDropDownButton

逻辑型，是否显示数值输入框的下拉计算器按钮。

4.9.2.4.2.5.2.6 Increment

数值型，设置单击上下微调按钮的时候，数值改变的幅度。

4.9.2.4.2.5.3 事件

4.9.2.4.2.5.3.1 ValueChanged

控件的 Value 属性发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.2.5.4 方法

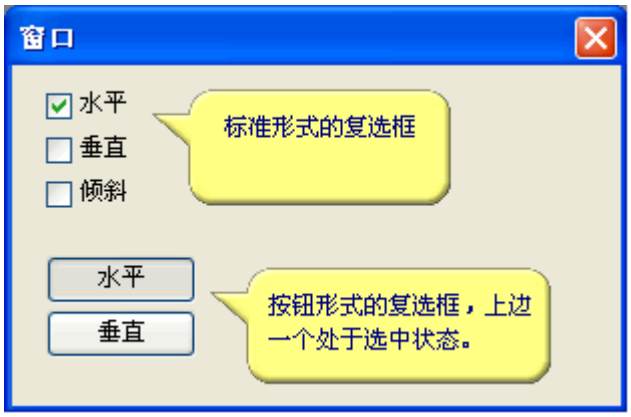
4.9.2.4.2.5.4.1 SelectAll

选定数值输入框中的所有内容。

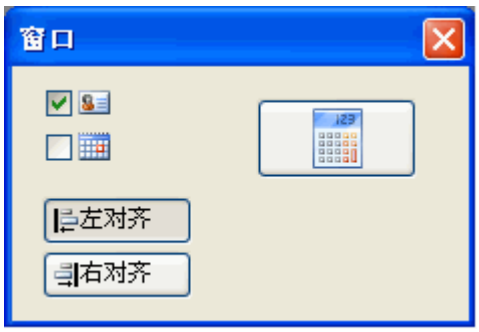
4.9.2.4.2.6 CheckBox(复选框)

4.9.2.4.2.6.1 关于 CheckBox

CheckBox 表示复选框，通常用于输入逻辑值，可以在两种状态之间反复切换。
复选框默认为方框形式，如果在设计窗口的时候，将其“外观”属性设为“Button”，复选框看上去就像一个按钮，按钮可以在按下和弹起两种状态中切换，按下表示选中，弹起表示未选中。



和按钮一样，复选框也可以设置图标：



4.9.2.4.2.6.2 属性

4.9.2.4.2.6.2.1 Checked

逻辑型，返回或设置复选框的选中状态，True 表示选中，False 表示未选中。

示例

```
Dim chk As WinForm.CheckBox
chk = Forms("窗口 1").Controls("CheckBox1")
chk.Checked = True
```

4.9.2.4.2.6.2.2 Text

字符型，返回或设置复选框的标题。

4.9.2.4.2.6.2.3 Image

返回或设置复选框的图标。

可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

示例

```
Dim chk As WinForm.CheckBox
```

```
chk = Forms("窗口 1").Controls("CheckBox1")
chk.Image = GetImage("Accept.Ico")
```

4.9.2.4.2.6.2.4 Appearance

设置复选框的外观样式，Appearance 型枚举，可选值有：
Button: 按钮形式
Normal：标准形式
示例

```
Forms("窗口 1").Controls("CheckBox1").Appearance = Appearance.Button
```

4.9.2.4.2.6.2.5 TextAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.6.2.6 CheckAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.6.2.7 ImageAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中

TopLeft　　垂直靠上，水平靠左
TopRight　　垂直靠上，水平靠右

4.9.2.4.2.6.2.8 TextImageRelation

设置文本和图标的相对位置，TextImageRelation 型枚举，可选值有：
Overlay：　　重叠
ImageAboveText：　图标在文本的上方
ImageBeforeText：　图标在文本的左方
TextAboveImage：　文本在图标的上方
TextBeforeImage：　文本在图标的左方

4.9.2.4.2.6.3 事件

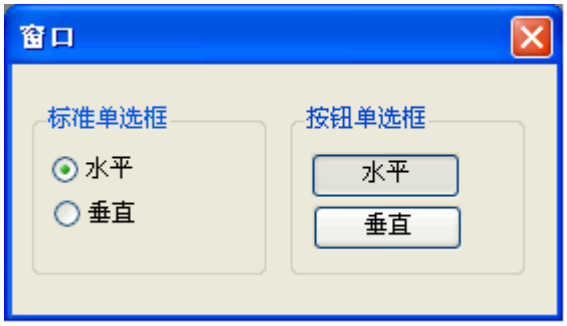
4.9.2.4.2.6.3.1 CheckedChanged

Checked 属性发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.2.7 RadioButton(单选框)

4.9.2.4.2.7.1 关于 RadioButton

RadioButton 表示单选框。
单选框和复选框非常类似，不同的是，同一个容器中的多个单选框，每次只能有一个处于选中状态，如果你选中一个单选框，之前选中的单选框将被自动撤销选中状态。
如果在设计窗口的时候，将单选框的“外观”属性设为“Button”，单选框看上去就像一个按钮，按钮可以在按下和弹起两种状态中切换，按下表示选中，弹起表示未选中。



和按钮一样，单选框也可以设置图标：



4.9.2.4.2.7.2 属性

4.9.2.4.2.7.2.1 Checked

逻辑型，返回或设置单选框的选中状态，True 表示选中，False 表示未选中。
示例

```
Dim rdo As WinForm.RadioButton
rdo = Forms("窗口 1").Controls("RadioButton1")
rdo.Checked = True
```

4.9.2.4.2.7.2.2 Text

字符型，返回或设置单选框的标题。

4.9.2.4.2.7.2.3 Image

返回或设置单选框的图标。
可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
示例

```
Forms("窗口 1").Controls("RadioButton1").Image = GetImage("Accept.Ico")
```

4.9.2.4.2.7.2.4 Appearance

设置单选框的外观样式，Appearance 型枚举，可选值有：
Button：按钮形式
Normal：标准形式
示例

Forms("窗口 1").Controls("RadioButton1").Appearance = Appearance.Button

4.9.2.4.2.7.2.5 TextAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.7.2.6 CheckAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.7.2.7 ImageAlign

BottomCenter 垂直靠下，水平居中
BottomLeft 垂直靠下，水平靠左
BottomRight 垂直靠下，水平靠右
MiddleCenter 垂直居中，水平居中
MiddleLeft 垂直居中，水平靠左
MiddleRight 垂直居中，水平靠右
TopCenter 垂直靠上，水平居中
TopLeft 垂直靠上，水平靠左
TopRight 垂直靠上，水平靠右

4.9.2.4.2.7.2.8 TextImageRelation

设置文本和图标的相对位置，TextImageRelation 型枚举，可选值有：
Overlay： 重叠
ImageAboveText： 图标在文本的上方
ImageBeforeText： 图标在文本的左方
TextAboveImage： 文本在图标的上方

TextBeforeImage： 文本在图标的左方

4.9.2.4.2.7.3 事件

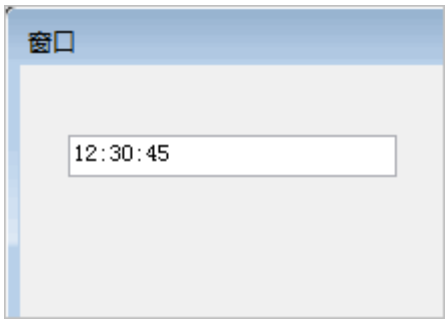
4.9.2.4.2.7.3.1 CheckedChanged

Checked 属性发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form: 返回控件所在的窗口

4.9.2.4.2.8 TimeSpanBox(时段输入框)

4.9.2.4.2.8.1 关于 TimeSpanBox

TimeSpanBox 表示时段输入框，专门用于为时段列输入数据：



关于时段列的更多信息，请参考：时段列 时段列与计算
TimeSpanBox 虽然以“时:分:秒 毫秒”格式显示和输入数据，但是其值本质上是一个双精度小数，以秒为单位。
例如上图中的“12:30:45”，表示 12 个小时 30 分钟 45 秒，其值为：12 * 3600 + 30 * 60 + 45，也就是 45045。
你可以用下面的代码来验证：

```
Dim tp As WinForm.TimeSpanBox = e.Form.Controls("TimeSpanBox1")
MessageBox.show(tp.Value)
```

我们可以给TimeSpanBox设置输入掩码，以提高输入效率：

时段格式	输入掩码
时:分	00:00
时:分:秒	00:00:00
时 : 分 : 秒	00:00:00

毫秒	0##

输入掩码中的 0 表示必填的数字，所以如果输入掩码是"00:00",不能输入"12:9"，必须输入"12:09"

4.9.2.4.2.8.2 属性

4.9.2.4.2.8.2.1 Value

返回 TimeSpanBox(时段输入框)的值，以秒为单位。

4.9.2.4.2.8.2.2 TextAlign

设置文本的对齐方式，HorizontalAlignment 型枚举，可选值有：
Center：居中
Left： 靠左
Right： 靠右

4.9.2.4.2.8.3 事件

4.9.2.4.2.8.3.1 ValueChanged

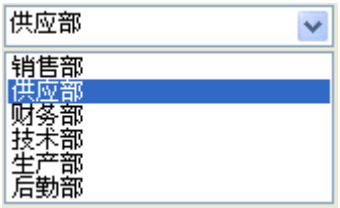
控件的 Value 属性发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form: 返回控件所在的窗口

4.9.2.4.3 列表类

4.9.2.4.3.1 ComboBox(组合框)

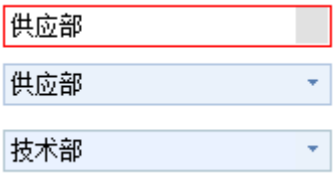
4.9.2.4.3.1.1 关于 ComboBox

ComboBox 表示组合框。
组合框可以直接输入内容，也可以单击右侧的箭头，从预先设置好的列表项目中选择值：



请参考：
设置列表项目
辅助输入设置

ComboBox 可以设置样式和边框颜色：



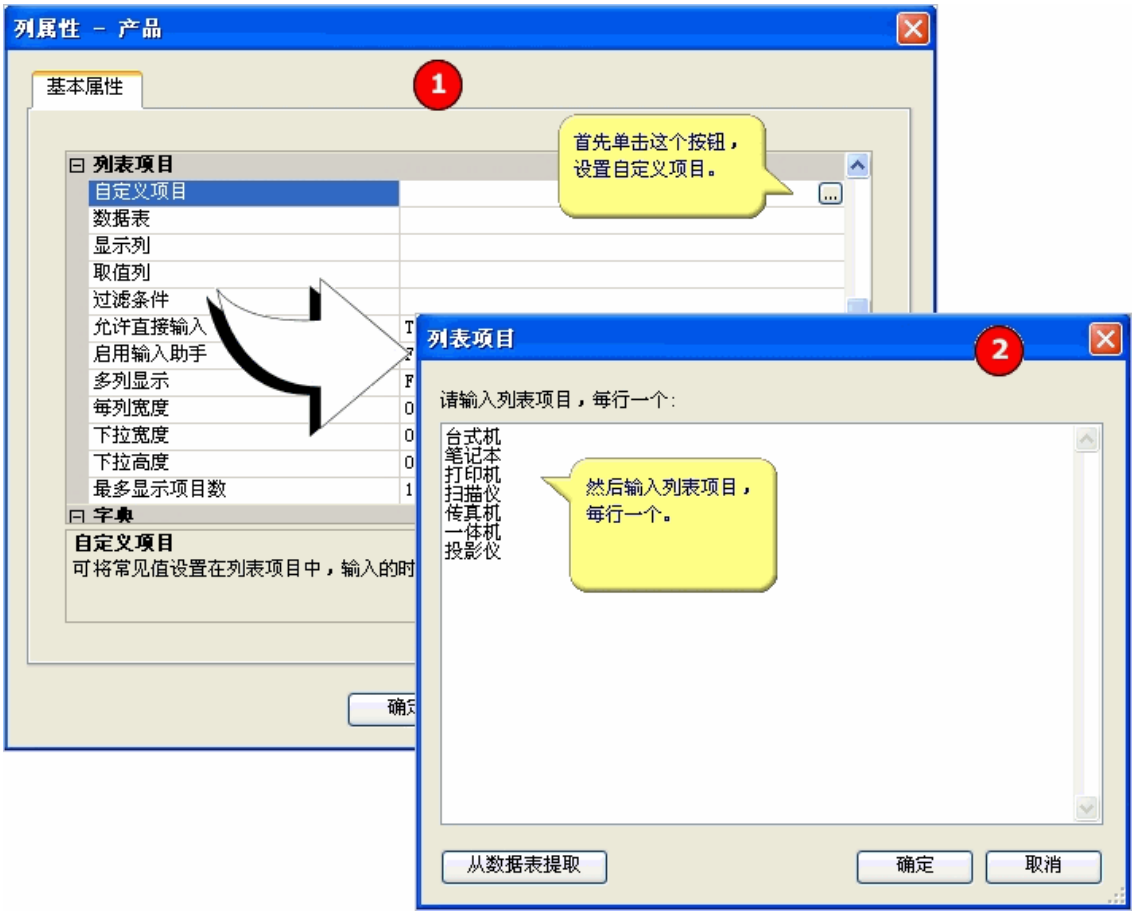
提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 ComboBox 的高度，但是可以通过属性设置窗口设置其高度属性。

4.9.2.4.3.1.2 设置列表项目

此节的内容虽然针对的是组合框，但是同样适合复选组合框、列表框、复选列表框。

自定义列表项目

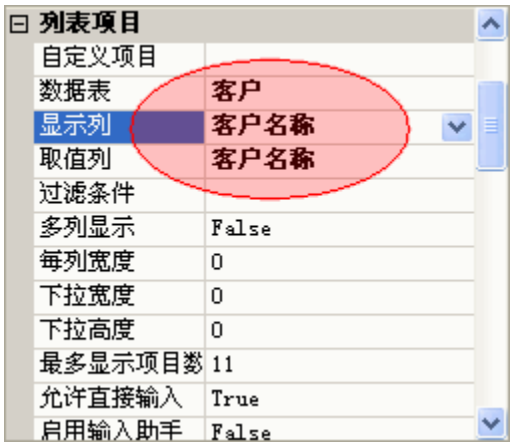
自定义列表项目的设置很简单：



列表项目来自于列

列表项目不仅可以自定义，还可以直接用某一列中已经输入的内容作为列表项目。

例如假定客户表中已经输入了所有的客户资料，某列表框希望从客户表中提取客户名称作为列表项目，设置很简单，只需将组合框的“数据表”属性设为客户表，“显示列”和“取值列”都设为客户名称列：



显示列和取值列

列表项目来自于列时，“显示列”和“取值列”属性有着不同的意义。

展开下拉列表时，显示哪一列的内容，由“显示列”属性决定。

单击某个列表项目时，将哪一列的内容填入到单元格中，由“取值列”属性决定。

假定组合框输入的不是客户名称，而是客户编号，其列表项目自然来自于客户表的客户编号列，如果“显示列”和“取值列”属性都设为客户编号列，那么从下拉列表中选择正确的客户编号可不容易，你也许知道“威航货运有限公司”的客户编号是“CACTU”，但是你能知道所有客户的编号吗？

最好的办法是将“显示列”属性设为“客户名称”，“取值列”属性设为“客户编号”，这样下拉列表显示的是客户名称，而单击某一客户名称，组合框即可自动填入该客户对应的客户编号。

过滤条件

列表项目来自于列的时候，你可以用“过滤条件”属性设置选择哪些行用于列表项目。

例如来自于客户表的时候，你希望只列出华北地区的客户，你可以设置条件为：

```
[地区] = '华北'
```

4.9.2.4.3.1.3 输入辅助设置

此节的内容虽然针对的是组合框,但是同样适合复选组合框。

启用输入助手

如果列表项目上百个，甚至更多，那么从中找出自己需要的也不容易，可能还不如直接输入来得快。

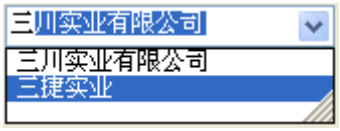
可是直接输入的话，又难免会出现输入错误，对于电脑来说，“北京市公安局”和“北京公安局”可是完全不同的两个单位。

对于这种情况，可以将属性“启用输入助手”设为 True(是)，这将启动两个辅助性的输入功能:

- **自动完成输入**

  输入一个字符，系统会自动输入第一个以该字符开始的项目，同时筛选出所有以该字符开始的项目供选择;你可以按 Tab 键接受自动输入的项目,也可以选择筛选出来的候选项目;随着输入字符的增多，会更精确地定位出您可能要输入的项目。

  例如在下图中，用户只是输入了一个字符“三”，但是程序却自动输入了第一个以“三”开头的项目“三川实业有限公司”，同时列出了所有以“三”开头的项目供选择，你可以按 Tab 键接受自动输入的内容，也可以从列出的候选项目中选择一个。

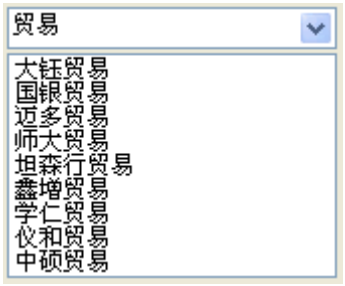


- **高级筛选功能**

  前面提到的自动筛选功能，只能筛选出以输入字符开始的项目。

  如果需要筛选出包括输入字符的项目，可以按“Ctrl + Enter"键。

  例如下图中，我们输入“贸易”二字，然后按“Ctrl + Enter"键，即可自动列出所有包括“贸易”二字的项目供选择。



禁止直接输入

在默认情况下，用户不仅可以从列表项目中选择内容，还可以直接输入不在列表项目中的内容。如果你要求用户只能从列表项目中选择内容，不能直接输入，可以将属性“允许直接输入”设为False(否)。

用键盘选择列表项目

大多数时候，我们是通过鼠标从下拉列表中选择要输入的项目。

不过也可以通过键盘来操作，选择一个空白单元格，按空格键进入编辑状态，再次按空格键会自动展开下拉列表，按上下箭头选择项目，按空格键或者回车键确认选择。

4.9.2.4.3.1.4 多列下拉列表

此节的内容虽然针对的是组合框,但是同样适合复选组合框。

默认情况下，下拉列表只显示一列，如果下拉列表的项目很多，也可以分多列显示。

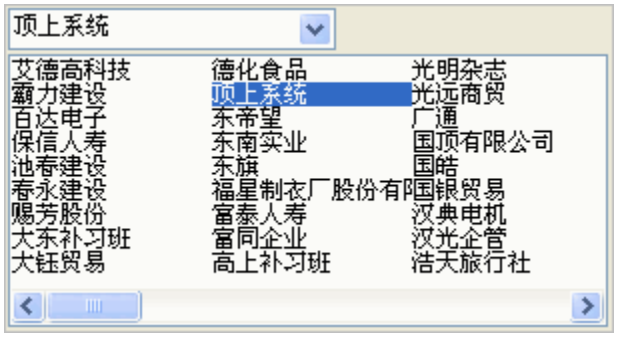
要分多列显示下拉列表，需要进行如下设置：

1、将“多列显示”属性设为 True(是)。

2、设置“每列宽度”属性。

3、设置“下拉宽度”属性，下拉宽度应该等于：每列宽度 * N，N 为你希望显示的列数。

例如下图，每列宽度等于 100，下拉宽度等于 300，下拉列表分三列显示：



4.9.2.4.3.1.5 与 TextBox 通用的成员

Combobox(组合框)是在 TextBox(文本框)的基础上增加了下拉列表功能，所以具备 TextBox(文本框)的很多成员。

属性

Text
Value
TextAlign
SelectedText
SelectionStart
SelectionLength

方法

Copy
Cut
Paste
Undo
ResetText
SelectAll

事件

TextChanged
ValueChanged

上述成员，请参考 TextBox 中的介绍，这里就不再重复。
接下来将介绍 ComboBox 特有的成员。

4.9.2.4.3.1.6 属性

4.9.2.4.3.1.6.1 ComboList

字符型，用于设置自定义列表项目，项目之间用符号“|”隔开。

示例一

```
Dim cmb As WinForm.ComboBox
cmb = Forms("窗口 1").Controls("ComboBox1")
cmb.ComboList = "计算机|打印机|传真机|电话机"
```

示例二

```
Dim cmb As WinForm.ComboBox
cmb = Forms("窗口 1").Controls("ComboBox1")
cmb.ComboList = DataTables("客户").GetComboListString("公司名称")
```

上述的代码如果设置在窗口的 AfterLoad 事件中，即可在打开窗口的时候，自动从客户表提取公司名称，作为组合框 Combox1 的列表项目。

4.9.2.4.3.1.6.2 Items

字符集合，表示所有的自定义列表项目。
只有使用自定义列表项目时，此属性才有效。
示例
假定希望单击某个按钮，组合框能够选中"上海"，可以将该按钮的 Click 事件代码设为：

```
Dim cmb As WinForm.ComboBox
cmb = e.Form.Controls("ComboBox1")
For i As Integer = 0 To cmb.Items.Count - 1
    If cmb.Items(i) = "上海" Then
        cmb.SelectedIndex = i
    End If
Next
```

4.9.2.4.3.1.6.3 DataSource

字符型，返回或设置列表项目来源表的名称。
必须先设置显示列和取值列的名称，然后再设置数据来源表的名称，设置才可生效。
示例

```
Dim cmb As WinForm.ComboBox
cmb = Forms("窗口 1").Controls("ComboBox1")
cmb.DisplayMember = "客户名称" '设置显示列
cmb.ValueMember = "客户编号" '设置取值列
cmb.DataSource = "客户" '列表项目来自于客户表
```

4.9.2.4.3.1.6.4 DisplayMember

返回或设置显示列的名称。
请参考：DataSource

4.9.2.4.3.1.6.5 ValueMember

返回或设置取值列的名称。
请参考：DataSource

4.9.2.4.3.1.6.6 DataFilter

列表项目来自于数据表时，可以用此属性返回或设置过滤条件。

4.9.2.4.3.1.6.7 SelectedIndex

整数型，返回或设置所输入或选择的值，在列表项目中的位置。

如果没有输入或选择值，或者输入的值不包括在列表项目中，则返回-1。
示例
从下拉列表中选择第四个项目：

```
Dim cmb As WinForm.ComboBox
cmb = Forms("窗口 1").Controls("ComboBox1")
cmb.SelectedIndex = 3
```

4.9.2.4.3.1.6.8 SelectedItem

列表项目来自于数据表时，可用此属性返回或设置所选择的行(DataRow 类型)。
如果是自定义列表项目,此属性等同于 SelectedValue。
示例
某组合框绑定到客户表，希望从该组合框选择一个客户后，能够将该客户的电话和姓名填入另两个文本框中。
为实现此目的，可将该组合框的 SelectedIndexChanged 事件代码设为：

```
Dim cm As WinForm.ComboBox = e.Form.Controls("ComboBox1")
Dim dr As DataRow = cm.SelectedItem
If dr IsNot Nothing Then
    e.Form.Controls("txtName").Value = dr("姓名")
    e.Form.Controls("txtTel").Value = dr("电话")
End If
```

4.9.2.4.3.1.6.9 SelectedValue

和 Value 属性一样，此属性用于返回或设置组合框的值。
但是两者也有所不同：
1、如果组合框中的值，并非来自于列表项目，那么 SelectedValue 返回 Nothing，而 Value 会正常返回该值。
2、不属于列表项目的值，不能通过 SelectedValue 设置，只能通过 Value 设置。

4.9.2.4.3.1.6.10 AllowEditValue

逻辑型，是否允许用户输入不在列表项目中的内容。

4.9.2.4.3.1.6.11 DroppedDown

逻辑型，判断下拉列表或者下拉窗口是否已经打开。

4.9.2.4.3.1.6.12 AutoComplete

逻辑型，是否启用输入助手。

4.9.2.4.3.1.7 方法

4.9.2.4.3.1.7.1 OpenDropDown

打开下拉列表或者下拉窗口。

4.9.2.4.3.1.8 事件

4.9.2.4.3.1.8.1 SelectedIndexChanged

选择不同列表项目后执行。
e 参数属性:
Form:　返回触发事件的窗口
Sender:　返回触发事件的控件
示例
某组合框绑定到客户表，希望从该组合框选择一个客户后，能够将该客户的电话和姓名填入另两个文本框中。
为实现此目的，可将该组合框的 SelectedIndexChanged 事件代码设为：

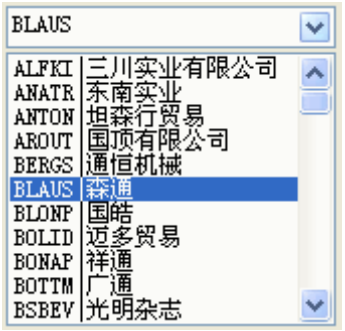
```
Dim cm As WinForm.ComboBox = e.Form.Controls("ComboBox1")
Dim dr As DataRow = cm.SelectedItem
If dr IsNot Nothing Then
    e.Form.Controls("txtName").Value = dr("姓名")
    e.Form.Controls("txtTel").Value = dr("电话")
End If
```

4.9.2.4.3.1.8.2 ListItemFormat

如果列表项目来自于数据表，此事件将在生成每一个列表项目时执行。
e 参数属性:
Form:　触发事件的窗口
Sender:　触发事件的控件
ListItem:　准备显示的行，DataRow 类型
Value:　列表项目准备显示的内容
示例
列表项目来自于数据表时，下拉的列表项目只能显示一列的内容，如果希望显示多列内容，可以通过事件 ListItemFormat 来实现。
例如某个 ComboBox 的列表项目来自于客户表，取值列为客户编号，显示列为客户名称，在默认情况下下拉列表只显示客户名称，选择某客户后，自动填入对应的客户编号。
如果希望下拉列表同时显示客户编号和客户名称两列的内容，可以在 ListItemFormat 事件中设置如下代码：

```
e.Value = e.ListItem("客户编号") & "|" & e.ListItem("客户名称")
```

现在的效果是：



4.9.2.4.3.1.9 窗口中的动态列表项目

在 Foxtable 编程部分，我们已经学会了如何在表中实现动态列表项目。

现在我们同样看看如何在窗口中实现列表项目。

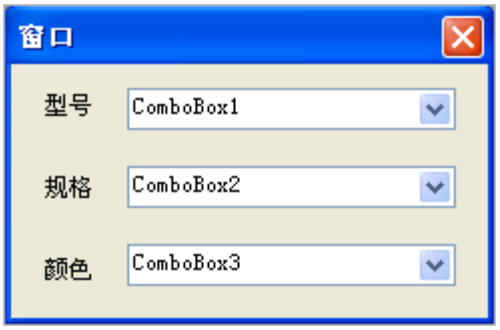
本节示例可以参考 CateStydy 目录下的文件：窗口动态列表项目.table

示例一

假定有一个表：

	型号	规格	颜色
1	VV	4	红
2	VV	6	红
3	YJV	10	黄
4	YJV	16	绿
5	VV	10	黄
6	VV	10	蓝
7	VV	10	黑
8	YJV	16	白

和下面这样的一个窗口:



希望第一个组合框能够自动列出所有型号供选择，而第二个组合框能够列出所选型号的全部规格供选择，同样第三个组合框能够列出所选规格的全部颜色供选择。

设计的步骤为：

1、在窗口的 AfterLoad 事件中加入代码：

```
Dim cmb As WinForm.ComboBox = e.form.Controls("ComboBox1")
cmb.ComboList = DataTables("表 A").GetComboListString("型号")
```

2、在 ComboBox2 的 Enter 事件中加入代码：

```
Dim cmb As WinForm.ComboBox = e.Sender
Dim str As string = e.Form.Controls("ComboBox1").Value
cmb.ComboList = DataTables("表 A").GetComboListString("规格", "型号 = '" & str & "'")
```

3、最后在 ComboBox3 的 Enter 事件中加入代码：

```
Dim cmb As WinForm.ComboBox = e.Sender
Dim str As String = e.Form.Controls("ComboBox2").Value
cmb.ComboList = DataTables("表 A").GetComboListString("颜色", "规格  = '" & str & "'")
```

示例二

假定组合框的列表项目并非动态的，随着应用的深入，这些列的列表项目可能会需要调整，虽然通过窗口设计，可以方便地重新设置列表项目。但是对于一个已经交付使用的系统来说，开放窗口设计功能给最终用户，是一个非常危险的做法。有没有安全一些的做法呢？有，通用的做法是：

1、新建一个表(假定表名为项目)，包括三列，分别是表、列、列表项目，用于输入某表某列的列表项目:

	表	列	列表项目
1	表A	第一列	项目1|项目2
2	表A	第二列	项目3|项目4
3	表B	第一列	项目5|项目6|项目7

2、将组合框的 Enter 事件代码设为：

```
Dim cmb As WinForm.ComboBox = e.Sender
If cmb.Items.Count > 0 Then '如果已经设置了项目
    Return '则返回
End If
Dim t As Table = cmb.BindTable '获得绑定的 Table
Dim c As Col = cmb.BindCol '获得绑定的 Col
Dim dr As DataRow = DataTables("项目").Find("[表] = '" & t.Name & "' And [列] = '" & c.Name & "'")
If dr IsNot Nothing Then '如果在项目表找到对应的行
    cmb.Combolist = dr("列表项目") '则设置列表项目
End If
```

代码原理很简单，根据代码注释，大家应该都能理解。

一个窗口可能有很多个组合框，针对每个组合框重复设置这样的代码，显得太累赘，为此我们可以使用窗口的全局事件 Enter，这样只需设置一次即可。但是全局事件对所有类型的控件有效，而我们的代码是针对组合框的，即使是组合框，也不见得都要执行上述代码。我们可以利用控件名称进行标识，例如对于需要执行上述代码的组合框，名称统一用字符"_cmb"开始，这样只需将全局事件 Enter 设为：

```
If e.Sender.Name.SubString(0,4) = "_cmb" Then '如果名称符合
    Dim cmb As WinForm.ComboBox = e.Sender
    If cmb.Items.Count > 0 Then '如果已经设置了项目
        Return '则返回
    End If
    Dim t As Table = cmb.BindTable '获得绑定的 Table
    Dim c As Col = cmb.BindCol '获得绑定的 Col
    Dim dr As DataRow = DataTables("项目").Find("[表] = '" & t.Name & "' And [列] = '" & c.Name & "'")
    If dr IsNot Nothing Then '如果在项目表找到对应的行
        cmb.Combolist = dr("列表项目") '则设置列表项目
    End If
End If
```

4.9.2.4.3.2 CheckedComboBox(复选组合框)

4.9.2.4.3.2.1 关于 CheckedComboBox

CheckedComboBox 表示复选组合框，可以同时从列表项目中选择多项：



值与值之间默认的分隔符号是逗号，如果需要改为其他符号，在属性设置中修改“分隔符”属性即可。

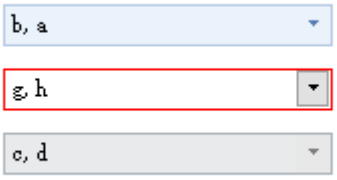
复选组合框其余的设置，和组合框是完全一样的。

请参考：

设置列表项目

辅助输入设置

CheckedComboBox 同样可以设置边框颜色和样式：



提示：设计窗口的时候，我们无法直接通过鼠标拖动调整 CheckedComboBox 的高度，但是可以通过属性设置窗口设置其高度属性。

4.9.2.4.3.2.2 与 ComboBox 通用的成员

CheckedComboBox(复选组合框)本身也是一个组合框(ComboBox)，所以它具备组合框全部的属性、方法、事件。

这些通用的成员，已经在组合框章节中介绍，这里就不重复。

实际上，和组合框相比，复选组合框就多了一个属性：Separator

4.9.2.4.3.2.3 属性

4.9.2.4.3.2.3.1 Separator

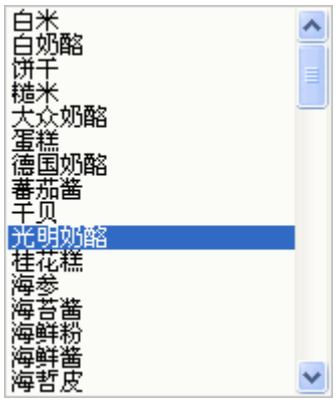
字符型，返回或设置值与值之间的分隔符号。
默认的分隔符号符号是逗号。
示例

```
Dim cmb As WinForm.CheckedComboBox
cmb = Forms("窗口 1").Controls("CheckedComboBox1")
cmb.Separator = "#"
```

4.9.2.4.3.3 ListBox(列表框)

4.9.2.4.3.3.1 关于 ListBox

ListBox 表示列表框，列表框直接显示一些候选项目供用户选择，因为占用空间比较大，所以较少使用。
一般的用户可以忽略这个控件。



列表项目的设置和组合框完全相同，请参考：
设置列表项目

4.9.2.4.3.3.2 属性

4.9.2.4.3.3.2.1 ComboList

字符型，用于设置自定义列表项目，项目之间用符号“|”隔开。
示例

```
Dim lst As WinForm.ListBox
Lst = Forms("窗口 1").Controls("ListBox1")
lst.ComboList = "计算机|打印机|传真机|电话机"
```

4.9.2.4.3.3.2.2 Items

字符集合，表示所有的自定义列表项目。
只有使用自定义列表项目时，此属性才有效。
示例
假定希望单击某个按钮，列表框能够选中"上海"，可以将该按钮的 Click 事件代码设为：

```
Dim lst As WinForm.ListBox
lst = e.Form.Controls("ListBox1")
For i As Integer = 0 To lst.Items.Count - 1
    If lst.Items(i) = "上海" Then
        lst.SelectedIndex = i
    End If
Next
```

4.9.2.4.3.3.2.3 DataSource

字符型，返回或设置列表项目来源表的名称。
必须先设置显示列和取值列的名称，然后再设置数据来源表的名称，设置才可生效。
示例

```
Dim lst As WinForm.ListBox
Lst = Forms("窗口 1").Controls("ListBox1")
lst.DisplayMember = "客户名称" '设置显示列
lst.ValueMember = "客户编号" '设置取值列
lst.DataSource = "客户" '列表项目来自于客户表
```

4.9.2.4.3.3.2.4 DisplayMember

返回或设置显示列的名称。
请参考：DataSource

4.9.2.4.3.3.2.5 ValueMember

返回或设置取值列的名称。
请参考：DataSource

4.9.2.4.3.3.2.6 DataFilter

列表项目来自于数据表时，可以用此属性返回或设置过滤条件。

4.9.2.4.3.3.2.7 DataSort

列表项目来自于数据表时，可以用此属性返回或设置排序字段。

4.9.2.4.3.3.2.8 SelectedIndex

整数型，返回选定项目的位置。
示例

```
Dim lst As WinForm.ListBox
Lst = Forms("窗口 1").Controls("ListBox1")
MessageBox.Show("你选择的是" & lst.SelectedIndex & "项。")
```

4.9.2.4.3.3.2.9 SelectedItem

列表项目来自于数据表时，可用此属性返回或设置所选择的行(DataRow 类型)。
如果是自定义列表项目,此属性等同于 SelectedValue 。

4.9.2.4.3.3.2.10 SelectedValue

返回或设置选定的值。
示例
假定某列表框绑定到客户表，显示列为“客户名称”，取值列为“客户编号”，希望选择客户编号为“BOLID”的项目：

```
Dim lst As WinForm.ListBox
Lst = Forms("窗口 1").Controls("ListBox1")
lst.SelectedValue = "BOLID"
```

4.9.2.4.3.3.3 事件

4.9.2.4.3.3.3.1 SelectedIndexChanged

选择不同项目后执行。
e 参数属性:
Form: 返回触发事件的窗口
Sender： 返回触发事件的控件

4.9.2.4.3.3.3.2 ListItemFormat

如果列表项目来自于数据表，此事件将在生成每一个列表项目时执行。
e 参数属性:
Form: 触发事件的窗口
Sender: 触发事件的控件
ListItem: 准备显示的行，DataRow 类型
Value: 列表项目准备显示的内容
示例
列表项目来自于数据表时，列表项目只能显示一列的内容，如果希望显示多列内容，可以通过事

件 ListItemFormat 来实现。

假定列表项目来自于客户表，取值列为客户编号，显示列为客户名称，如果希望列表项目能同时显示客户编号和客户名称两列的内容，可以在 ListItemFormat 事件中设置如下代码：

```
e.Value = e.ListItem("客户编号") & "|" & e.ListItem("客户名称")
```

现在的效果是：

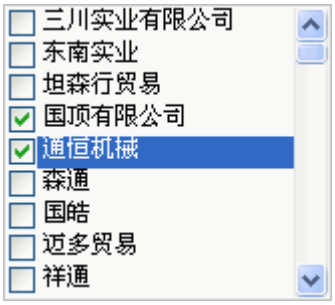


4.9.2.4.3.4 CheckedListBox(复选列表框)

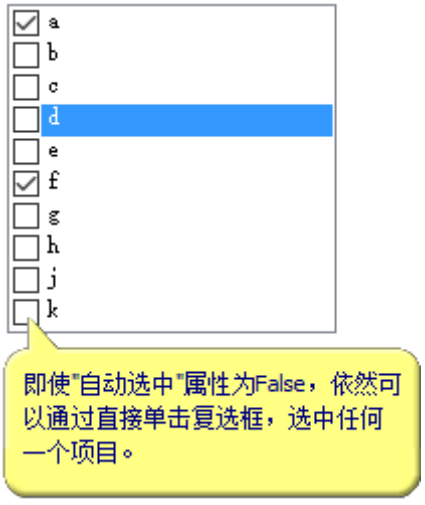
4.9.2.4.3.4.1 关于 CheckedListBox

CheckedListBox 表示复选列表框。

复选列表框直接显示一些候选项目供用户选择，可以多选，因为占用空间比较大，所以较少使用。一般的用户可以忽略这个控件。



CheckedListBox 有个自动选中(CheckOnClick)的属性，默认为 True，用户单击某个项目会自动勾选此项目，如果设置为 False，只有单击项目的复选框，才会勾选此项目。



列表项目的设置和组合框完全相同，请参考：
设置列表项目

4.9.2.4.3.4.2 与 ListBox 通用的成员

CheckedListBox(复选列表框)本身也是一个列表框(ListBox)，所以它具备列表框全部的属性、方法、事件。
这些通用的成员，已经在列表框章节中介绍，这里就不重复。
接下来我们介绍复选列表框在列表框基础上增加的成员。

4.9.2.4.3.4.3 属性

4.9.2.4.3.4.3.1 Value

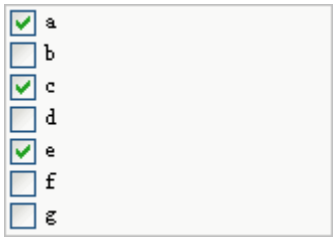
返回或设置复选列表框的值。
对于复选列表框，Value 和 SelectedValue 是不同的。
Value 表示选中(打勾)项目的值，如果有多个项目被选中，则用分隔符隔开。
SelectedValue 表示选定项目的值，所谓选定项目，就是光标所在项目。
例如对于下图，Value 属性的值为“c,e”，而 SelectedValue 属性的值为“b”：



示例

```
Dim lst As WinForm.CheckedListBox
lst = Forms("窗口 1").Controls("CheckedListBox1")
lst.Value = "a,c,e"
```

执行上述代码，会在列表项目中选中三个项目，分别是 a、c、e：



4.9.2.4.3.4.3.2 CheckedIndices

整数型集合，返回所有选定项目的位置。
示例

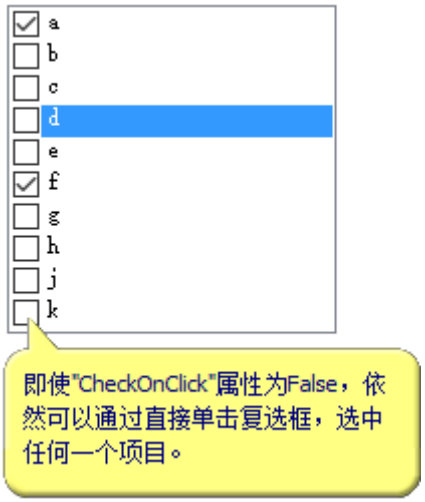
```
Dim lst As WinForm.CheckedListBox
Dim msg As String
```

```
lst = Forms("窗口 1").Controls("CheckedListBox1")
msg = "总共选中了" & lst.CheckedIndices.Count & "项，分别是："
For Each Index As Integer In lst.CheckedIndices
    msg = msg & Index & " "
Next
Messagebox.Show(msg)
```

4.9.2.4.3.4.3.3 CheckOnClick

用鼠标单击列表项目的标题时，是否自动勾选此项目。

如果此属性设置为 True，不管单击的是列表项目的标题还是复选框，都会勾选此项目。
如果此属性设置为 False，单击标题只会选中此项目，并不会勾选此项目，而单击复选框，依然则会选中并勾选此项目。



4.9.2.4.3.4.4 方法

4.9.2.4.3.4.4.1 SetItemChecked

设置指定项目的选中状态。
语法
SetItemChecked(Index, Checked)
Index： 整数型，指定项目位置
Checked：逻辑型，是否选中项目。
示例
清除所有项目的选中标记，然后选中第三项：

```
Dim lst As WinForm.CheckedListBox
lst = Forms("窗口 1").Controls("CheckedListBox1")
For Index As Integer = 0 To lst.Items.Count - 1
```

```
    lst.SetItemChecked(Index, False)
Next
lst.SetItemChecked(2, True)
```

4.9.2.4.3.4.4.2 GetItemChecked

判断指定项目是否选中，如果选中，返回 True，否则返回 False。
语法
GetItemChecked(Index)
Index： 整数型，指定项目位置
示例

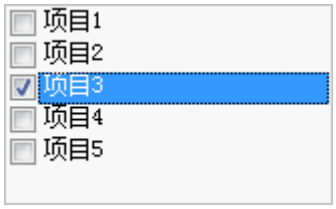
```
Dim lst As WinForm.CheckedListBox
lst = Forms("窗口 1").Controls("CheckedListBox1")
If lst.GetItemChecked(2) Then
    MessageBox.Show("第三项已经选中！")
End If
```

4.9.2.4.3.4.5 事件

4.9.2.4.3.4.5.1 ItemCheck

某一项目的选中状态即将发生变化时执行。
e 参数属性:
Form: 返回触发事件的窗口
Sender： 返回触发事件的控件
Index： 整数型，返回项目位置
Checked: 逻辑型，返回或设置项目新的选中状态。

示例：
假定对于下图的 CheckedListBox，要求每次只能选中一项，也就是选中一项后，将自动清除其他各项的选中标记：



为此，只需将其 ItemCheck 事件代码设置为：

```
If e.Checked Then
    Dim cst As WinForm.CheckedListBox = e.Form.Controls("CheckedListBox1")
```

```
    For idx As Integer = 0 To cst.Items.Count - 1
        If idx <> e.Index Then
            cst.SetItemChecked(idx, False)
        End If
    Next
End If
```

4.9.2.4.3.4.6 应用实例一

假定要设计下图所示的导出窗口，让用户选择要导出的列，单击确定按钮后导出为 Excel 文件：



设计方法很简单：

1、将窗口的 AfterLoad 事件代码设置为：

```
Dim ckl As WinForm.CheckedListBox = e.Form.Controls("CheckedListBox1")
For Each dc As DataCol In DataTables("订单").DataCols
    ckl.Items.Add(dc.name)
Next
```

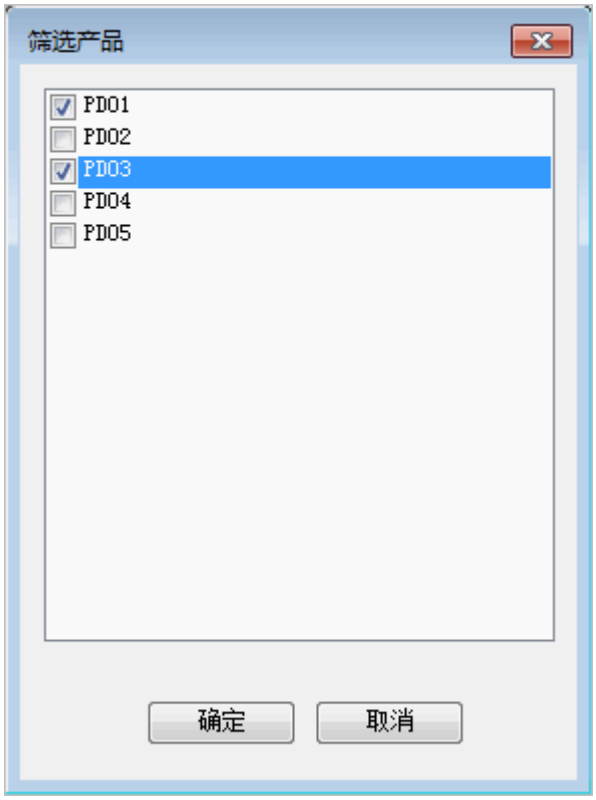
2、将导出按钮的Clik事件代码设置为：

```
Dim ckl As WinForm.CheckedListBox = e.Form.Controls("CheckedListBox1")
Dim nms As New List(of String)
For i As Integer = 0 To ckl.Items.count - 1 '获取已经勾选的列
    If ckl.GetItemChecked(i) Then
        nms.Add(ckl.Items(i))
```

```
    End If
Next
If nms.Count = 0 Then
    MessageBox.Show("至少要选择一列","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
'生成Excel表
Dim dt As Table = Tables("订单")
Dim Book As New XLS.Book
Dim Sheet As XLS.Sheet = Book.Sheets(0)
For i As Integer = 0 To nms.count -1
    Sheet(0, i).Value = nms(i)
Next
For r As Integer = 0 To dt.Rows.Count - 1 '填入数据
    For i As Integer = 0 To nms.count -1
        Sheet(r + 1, i).Value = dt.rows(r)(nms(i))
    Next
Next
'保存并打开excel文件
Book.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

4.9.2.4.3.4.7 应用实例二

假定要设计一个下图所示的窗口，自动列出订单表的所有产品供选择，单击确定按钮后，筛选出所选各产品的订单：



设计这样的窗口很简单，窗口插入一个 CheckedListBox 控件，一个“确定”按钮，一个“取消”按钮。窗口的 AfterLoad 事件代码设置为：

```
Dim lb As WinForm.CheckedListBox = e.Form.Controls("CheckedListBox1")
lb.ComboList = DataTables("订单").GetComboListString("产品")
```

“确定”按钮的代码设置为：

```
Dim s As String
Dim l As WinForm.CheckedListBox = e.Form.Controls("CheckedListBox1")
For Each Index As Integer In l.CheckedIndices
    If s > "" Then
        s = s & ","
    End If
    s = s & "'" & l.Items(Index) & "'"
Next
If s > "" Then
    Tables("订单").Filter = "产品  In (" & s & ")"
End If
e.Form.Close()
```

4.9.2.4.4 容器类

4.9.2.4.4.1 GroupBox(分组框)

4.9.2.4.4.1.1 关于 GroupBox

GroupBox 表示分组框。
窗口中控件较多的时候，可以将功能类似的控件放在同一分组中，使得窗口看上去更有条理。
分组框有一个标题，用于对该分组框中的控件进行说明。



4.9.2.4.4.1.2 属性

4.9.2.4.4.1.2.1 Text

字符型，返回或设置分组框的标题。

4.9.2.4.4.1.2.2 Children

返回一个集合，包括所有顶层子控件。

4.9.2.4.4.2 Panel(面板)

4.9.2.4.4.2.1 关于 Panel

Panel 表示面板。
和分组框一样，面板同样是用于对功能相近的控件进行分组，不过没有标题，默认也没有边框。

如果内容较多，面板不足以容纳，可以在设计窗口的时候，设置面板的“扩展尺寸”属性，扩展尺寸大于实际尺寸时，将自动出现滚动条：



另外如果将“自动滚动”属性设为 True，当面板中的控件位置或大小超出面板范围时，将自动出现滚动条，不管是否设置了扩展尺寸。

4.9.2.4.4.2.2 属性

4.9.2.4.4.2.2.1 Children

返回一个集合，包括所有顶层子控件。

4.9.2.4.4.2.2.2 ExtendSize

Size 类型，返回或设置面板的扩展尺寸。
示例
设置面板的扩展尺寸，宽度为 500，高度为 400：

```
Dim Pane As WinForm.Panel
Pane = Forms("窗口 1").Controls("Panel1")
Pane.ExtendSize = New Size(500,400)
```

4.9.2.4.4.2.2.3 AutoScroll

逻辑型，当面板中的控件位置或大小超出面板范围时，是否自动出现滚动条。

4.9.2.4.4.2.2.4 BorderStyle

设置面板的边框类型，BorderStyle 型枚举，可选值有：
Fixed3D：　三维边框
FixedSingle：单行边框
None：　无边框
示例：

```
Forms("窗口 2").Controls("Panel1").BorderStyle = BorderStyle.Fixed3D
```

4.9.2.4.4.2.2.5 ScrollPosition

返回或设置面板的滚动位置，Point 类型。
Point 类型有两个成员，分别为 X 和 Y，X 表示水平滚动位置，Y 表示垂直滚动位置。

4.9.2.4.4.2.3 事件

4.9.2.4.4.2.3.1 AfterScroll

滚动面板后执行。
e 参数属性：
Sender： 返回触发事件的控件
Form: 返回控件所在的窗口
AfterScroll 事件一般和 ScrollPosition 属性一起配合使用。

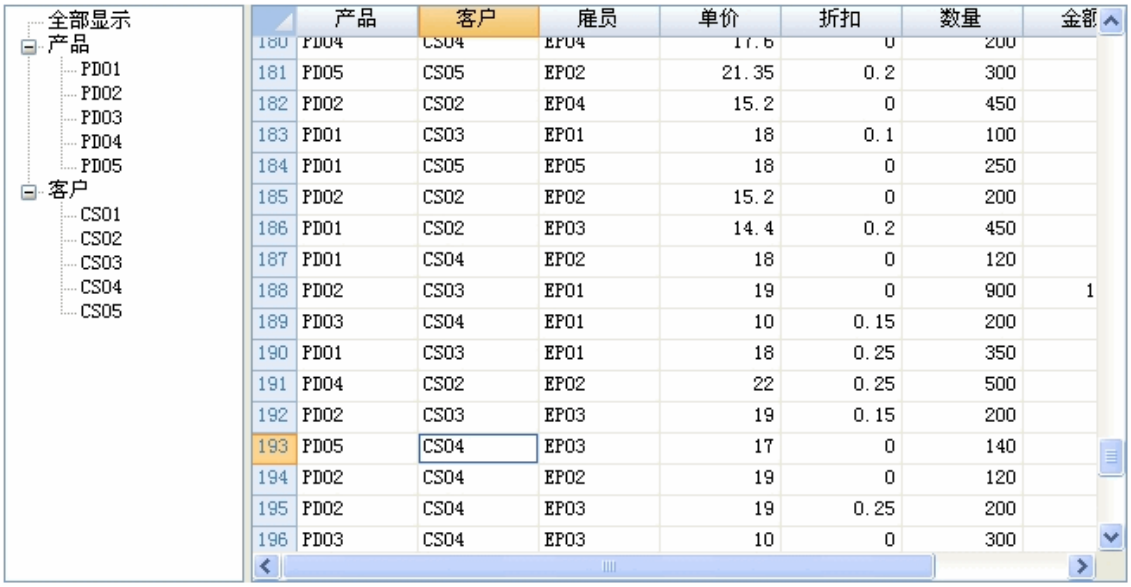
4.9.2.4.4.3 SplitContainer(分区面板)

4.9.2.4.4.3.1 关于 SplitContainer

SplitContainer 表示分区面板。
分区面板被一个分割条分割成左右两个分区，拖动该分割条可以调整两个分区的大小。
在设计窗口的时候，应该先设置好分区面板的“方向”属性，该属性决定分区面板是水平分割，还是垂直分割。
例如下图有两个分区面板，第一个为为垂直分割，第二个为水平分割。



下面是我们用分区面板设计的一个主窗口，类似于 Windows 的资源管理器：



	产品	客户	雇员	单价	折扣	数量	金额
180	PD04	CS04	EP04	17.6	0	200	
181	PD05	CS05	EP02	21.35	0.2	300	
182	PD02	CS02	EP04	15.2	0	450	
183	PD01	CS03	EP01	18	0.1	100	
184	PD01	CS05	EP05	18	0	250	
185	PD02	CS02	EP02	15.2	0	200	
186	PD01	CS02	EP03	14.4	0.2	450	
187	PD01	CS04	EP02	18	0	120	
188	PD02	CS03	EP01	19	0	900	1
189	PD03	CS04	EP01	10	0.15	200	
190	PD01	CS03	EP01	18	0.25	350	
191	PD04	CS02	EP02	22	0.25	500	
192	PD02	CS03	EP03	19	0.15	200	
193	PD05	CS04	EP03	17	0	140	
194	PD02	CS04	EP02	19	0	120	
195	PD02	CS04	EP03	19	0.25	200	
196	PD03	CS04	EP03	10	0	300	

设计的步骤很简单：

1、新建一个窗口，窗口类型设为主窗口，在主窗口中插入一个分区面板。

2、在右分区插入一个 Table(表)，将其绑定到主表，并将其停靠属性设置为 Fill。

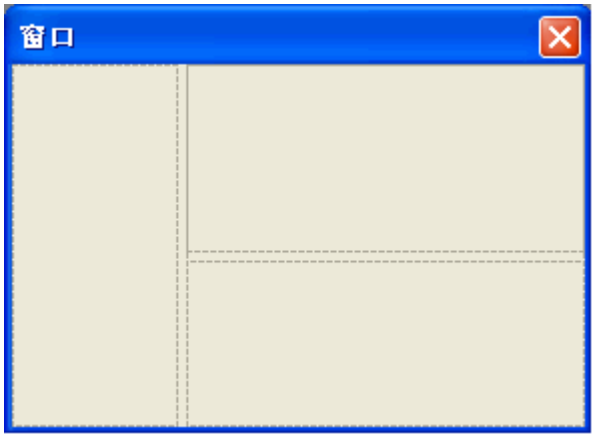
3、在左分区插入一个 TreeView(目录树)，并将其停靠属性设置为 Fill。

4、将分区面板的停靠属性也设为 Fill。

5、最后给 TreeView(目录树)加上节点。

关于目录树，后面的章节会介绍。

分区面板可以嵌套，这样我们就可以进行各种复杂的布局，例如我们在右分区中再插入一个水平方向分割的分区面板，就可以形成下图所示的布局：



主要属性：

- **方向**
  指定分区面板的分割方向。

- **固定分区**
  指定当分区面板的大小发生变化时，哪一个分区的大小保持不变。

- **固定分割条**
  指定是否禁止通过拖动分割条来调整两个分区的大小

属性列表中有两组属性，是用于单独设置某一个分区的：



4.9.2.4.4.3.2 属性

4.9.2.4.4.3.2.1 Panel1

返回第一个分区，类型为 WinForm.SplitPanel。

4.9.2.4.4.3.2.2 Panel2

返回第二个分区，类型为 WinForm.SplitPanel。

4.9.2.4.4.3.2.3 FixedPanel

是否固定分区面板中某一分区的大小，使得其大小不随分区面板大小的调整而调整。
该属性为整数型，设为 1 固定第一个分区，设为 2 固定第二个分区，设为 0 不固定任何分区，这也是默认设置。

4.9.2.4.4.3.2.4 SplitterDistance

整数型，返回或设置分割条的位置，单位为像素。
通常用于编程调整两个分区的大小。

4.9.2.4.4.3.2.5 IsSplitterFixed

逻辑型，是否禁止通过拖动分割条来调整两个分区的大小

4.9.2.4.4.3.2.6 SplitterWidth

整数型，返回或设置分割条的宽度，单位为像素。

4.9.2.4.4.3.2.7 BorderStyle

设置分区面板的边框类型，BorderStyle 型枚举，可选值有：
Fixed3D：　三维边框
FixedSingle：单行边框
None：　无边框

4.9.2.4.4.3.3 事件

4.9.2.4.4.3.3.1 SplitterMoved

拖动分割条之后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.4.3.4 SplitterPanel

SplitterPanel 表示分区面板的两个分区。
其属性有：

- AutoScroll
  逻辑型，是否自动出现滚动条。
- ExtendSize
  Size 类型，设置分区的扩展尺寸。
- Collapsed
  逻辑型，是否隐藏此分区。
- BackColor
  Color 类型，设置分区的背景颜色。
- Children
  返回一个集合，包括所有顶层子控件。

示例一
设置第一个分区的扩展尺寸，宽度为 500，高度为 400：

```
Dim p As WinForm.SplitPanel = Forms("窗口 2").Controls("SplitContainer1").Panel1
p.ExtendSize = New Size(500,400)
```

示例二
隐藏第二个分区：

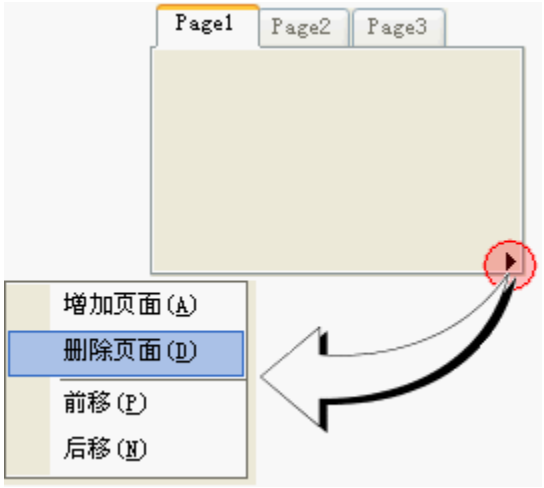
```
Forms("窗口 1").Controls("SplitContainer1").Panel2.Collapsed = True
```

4.9.2.4.4.4 TabControl(页面集合)

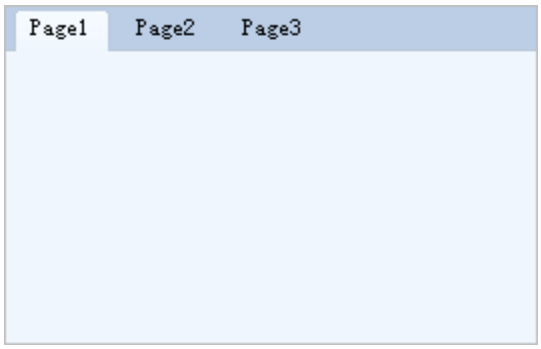
4.9.2.4.4.4.1 关于 TabControl

TabControl 表示页面集合，可以包括多个页面，每一个页面就是一个面板，可以容纳任意类型的控件。

在设计模式下的 TabControl，右下角会显示一个黑色箭头，单击该箭头会出现一个菜单，通过该菜单增加、删除、移动页面：



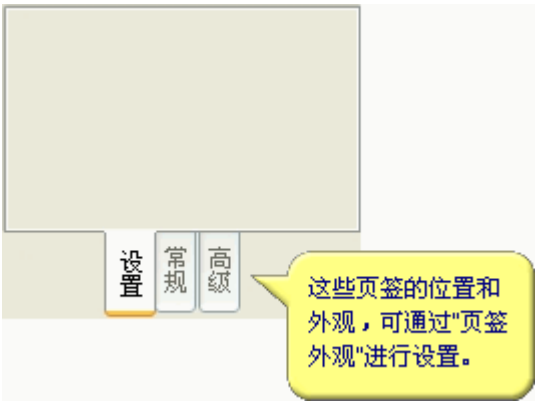
TabControl 提供了多种样式供选择，以下是其中几种：







TabControl 有一组“页签外观”属性，通过这些属性，可以很灵活地设置页签的外观，例如：



此外还有一组“选定页面”的属性，用于对选定页面进行设置：



这些属性都是自我描述性的，而且设置之后，可以即刻看到效果，所以就不再一一介绍。

4.9.2.4.4.4.2 属性

4.9.2.4.4.4.2.1 TabPages

TabPage 集合，可通过此集合访问指定位置或名称的页面。
TabPage 表示一个页面，接下来会介绍 TabPage 的属性。
示例
隐藏第二个页面：

```
Dim pg As WinForm.TabPage
pg = Forms("窗口 1").Controls("TabControl1").TabPages(1)
pg.Visible = False
```

TabPages 有以下属性和方法：

Add

增加新的页面，并返回所增加的页面。
语法：

```
Add(Name, Text)
```

Name: 页面名称
Text： 页面标题

Insert
插入新的页面，并返回所插入的页面。
语法：
Insert(Name, Text, Index)

Name: 页面名称
Text： 页面标题
Index：插入位置

Delete
删除指定名称或位置的页面。
语法
Delete(Name)
或
Delete(Index)
Name： 要删除页面的名称
Index：要删除页面的位置
Clear
清除所有页面。
Count
返回总的页面数

4.9.2.4.4.4.2.2 SelectedIndex

整数型，返回或设置选定页面编号。
示例
选择第二个页面：

```
Forms("窗口 1").Controls("TabControl1").SelectedIndex = 1
```

4.9.2.4.4.4.2.3 SelectedPage

TabPage 类型，返回或设置选定的页面。
示例
隐藏选定的页面：

```
Forms("窗口 1").Controls("TabControl1").SelectedPage.Visible = False
```

4.9.2.4.4.4.2.4 ShowTabs

逻辑型，是否显示页签。
例如：

```
e.Form.Controls("TabControl1").ShowTabs = False
```

参考：
用代码切换页面

4.9.2.4.4.4.3 事件

4.9.2.4.4.4.3.1 SelectedIndexChanged

选择不同页面后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.4.4.3.2 SelectedIndexChanging

选择不同页面前执行。
e 参数属性：
Sender：　返回触发事件的控件
Form:　　返回控件所在的窗口
NewIndex：　新页面的位置
Cancel：　逻辑型，设为 True，禁止选择新页面。

4.9.2.4.4.4.4 TabPage

4.9.2.4.4.4.4.1 关于 TabPage

TabPage 表示一个页面，我们可以通过页面集合的 TabPages 属性获得指定位置或名称的页面。
示例

```
Dim pg As WinForm.TabPage
pg = Forms("窗口 1").Controls("TabControl1").TabPages(1)
pg.Visible = False
```

4.9.2.4.4.4.4.2 属性

4.9.2.4.4.4.4.2.1 Name

返回页面的名称。

4.9.2.4.4.4.4.2.2 Text

返回或设置页面标题。

4.9.2.4.4.4.4.2.3 Image

返回或设置页面图标。
用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
示例

```
Dim pg As WinForm.TabPage
pg = Forms("窗口 1").Controls("TabControl1").TabPages(0)
pg.Image = GetImage("c:\clone.ico")
```

4.9.2.4.4.4.4.2.4 Visible

逻辑型，是否显示页面，默认为 True，设为 False 隐藏页面。

4.9.2.4.4.4.4.2.5 Children

返回一个集合，包括所有顶层子控件。

4.9.2.4.4.4.4.2.6 BackColor

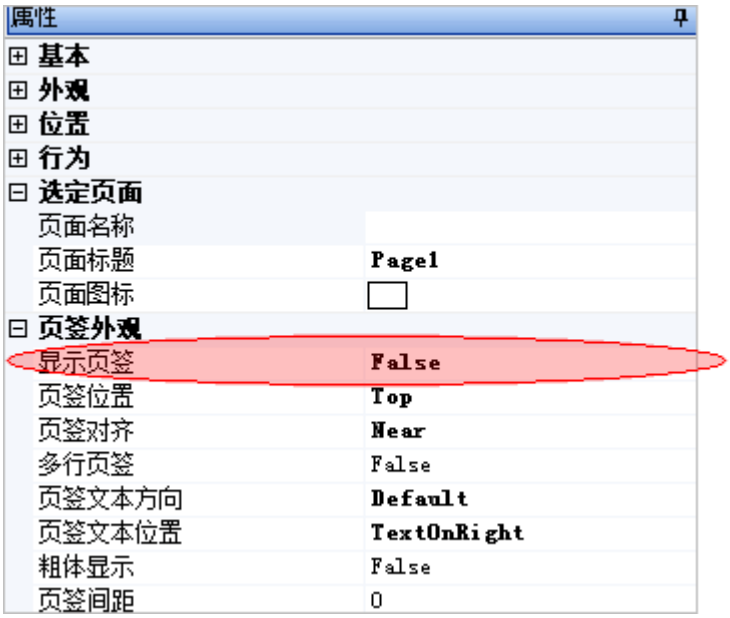
返回或设置页面背景颜色。

4.9.2.4.4.4.4.2.7 IsSelected

逻辑型，如果页面被选中，返回 True，否则返回 False。

4.9.2.4.4.4.5 用代码切换页面

在默认请况下，TabControl 会显示页签，单击某页签，可以选择对应的页面。
如果不希望用户能直接单击页签选择页面，而是通过导航栏、任务栏或者按钮来选择页面，可以在设计窗口的时候，如果将 TabControl 的“显示页签”属性设置为 False：



至于选择页面的代码，是很简单的，例如希望单击某个按钮能选择第二个页面，只需将其代码设置为：

```
Dim tab As WinForm.TabControl = e.Form.Controls("TabControl1")
tab.SelectedIndex = 1
```

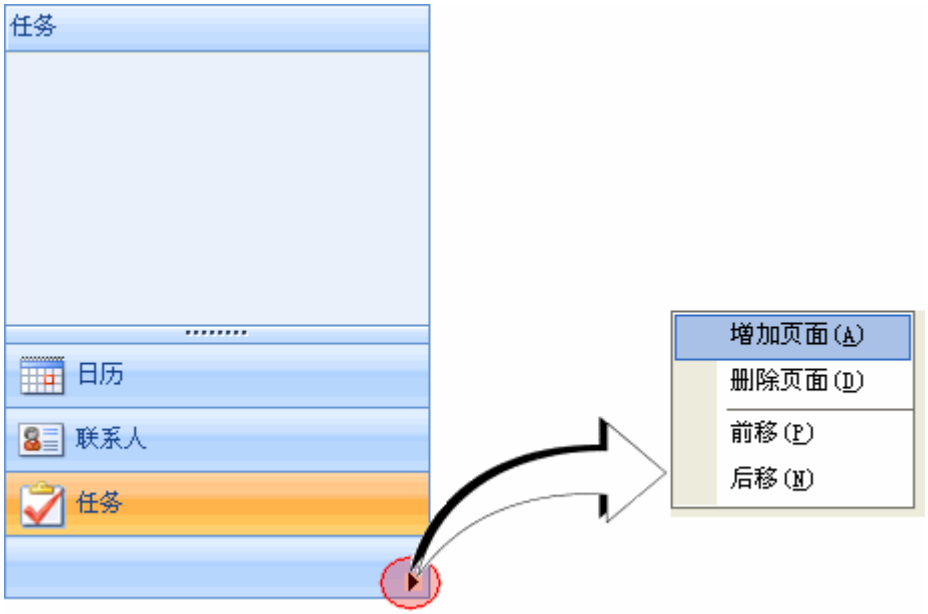
4.9.2.4.4.5 NavBar(导航栏)

4.9.2.4.4.5.1 关于 NavBar

NavBar 表示导航栏。

NavBar 类似 TabControl，由多个页面组成，页面可以接受任意类型的控件，通过单击导航栏下方的按钮来切换页面。

导航栏默认已经有三个页面，在设计窗口的时候，单击导航栏右下角的黑色箭头，会出现一个菜单，利用该菜单可以增加、删除、移动页面。



设置选定页面

首先单击导航栏下方的按钮，选择要设置的页面，然后利用下面这一组属性进行设置：



按钮显示位置

前面已经知道导航栏下方的按钮用于切换页面，这里默认最多显示 5 个按钮，你可以通过设置“显示按钮数”属性来调整这个值，多余的切换会以小图标的形式显示在水平栏：



这也就是为什么在设置选定页面的时候，有两个图标需要设置，如果数量不多，不需要在水平栏显示按钮，页面的小图标就没有必要设置，同时可以将“水平栏高度”属性设为 0，以节省空间。

图标的大小

图标默认的大小为 24*24，小图标默认为 16*16，图标大小在设计窗口的时候可以修改，如果修改“图标大小”属性，最好同时修改“按钮高度”属性，通常按钮高度应该比图标高度大 8 个像素左右，才比较协调。

样式

NavBar 提供了多种样式供选择，以下是其中几种：



4.9.2.4.4.5.2 属性

4.9.2.4.4.5.2.1 NavPages

返回导航栏的页面集合，通过此集合，可以访问指定名称或位置的页面。

导航栏页面的类型为 NavPage，接下来我们会介绍。

示例

```
Dim Bar As WinForm.NavBar
```

```
Dim Page1 As WinForm.NavPage
Dim Page2 As WinForm.NavPage
Bar = Forms("窗口 1").Controls("NavBar1")
Page1 = Bar.NavPages("Task")
Page2 = Bar.NavPages(0)
```

NavPages 有以下属性和方法：

Add

增加新的导航页面，并返回所增加的页面。

语法：

Add(Name, ButtonText, ButtonImageFile)

Name:　　页面名称

ButtonText：　　页面按钮的标题

ButtonImageFile：页面按钮的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

Insert

插入新的导航页面，并返回所插入的页面。

语法：

Insert(Name, ButtonText, ButtonImageFile, Index)

Name:　　页面名称

ButtonText：　　页面按钮的标题

ButtonImageFile：页面按钮的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

Index:　　插入位置

Delete

删除指定名称或位置的页面。

语法

Delete(Name)

或

Delete(Index)

Name：　要删除页面的名称

Index：要删除页面的位置

Clear

清除所有页面。

Count

返回总的页面数

4.9.2.4.4.5.2.2 SelectedIndex

整数型，返回或者设置导航栏选定页面的位置。

示例

选定第一个页面：

```
Dim Bar As WinForm.NavBar
Bar = e.Form.Controls("NavBar1")
```

Bar.SelectedIndex = 0

4.9.2.4.4.5.2.3 SelectedPage

返回或者设置导航栏选定的页面
示例
选定名称为"Task"的页面：

```
Dim Bar As WinForm.NavBar
Bar = e.Form.Controls("NavBar1")
Bar.SelectedPage = Bar.NavPages("Task")
```

4.9.2.4.4.5.3 事件

4.9.2.4.4.5.3.1 SelectedIndexChanged

选择不同页面后执行。
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口
示例
假定导航栏有三个页面，标题分别是页面 1、页面 2、页面 3，希望选择这些页面时，能同时切换到对应的表 A、表 B、表 C。
为此可以将 SelectedIndexChanged 事件代码设置为：

```
Dim bar As WinForm.NavBar = e.sender
Select Case bar.SelectedPage.ButtonText
    Case "页面1"
        MainTable = Tables("表A")
    Case "页面2"
        MainTable = Tables("表B")
    Case "页面3"
        MainTable = Tables("表C")
End Select
```

4.9.2.4.4.5.4 NavPage

4.9.2.4.4.5.4.1 关于 NavPage

NavPage 表示导航栏的页面。
通过导航栏的 NavPages，可以访问指定名称或位置的页面。
示例

```
Dim Bar As WinForm.NavBar
Dim Page1 As WinForm.NavPage
```

```
Dim Page2 As WinForm.NavPage
Bar = Forms("窗口 1").Controls("NavBar1")
Page1 = Bar.NavPages("Task")
Page2 = Bar.NavPages(0)
```

4.9.2.4.4.5.4.2 属性

4.9.2.4.4.5.4.2.1 Name

返回页面的名称。

4.9.2.4.4.5.4.2.2 Children

返回一个集合，包括所有顶层子控件。

4.9.2.4.4.5.4.2.3 Visible

逻辑型，默认为 True，设为 False，将隐藏页面。
示例
隐藏第一个页面：

```
Dim Bar As WinForm.NavBar
Bar = e.Form.Controls("NavBar1")
Bar.NavPages(0).Visible = False
```

4.9.2.4.4.5.4.2.4 ButtonText

字符型，返回或者设置页面的按钮标题。

4.9.2.4.4.5.4.2.5 HeaderText

字符型，返回或者设置页面标题。



4.9.2.4.4.5.4.2.6 ButtonImageFile

字符型，返回或设置页面的按钮图标文件。
如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。
示例

```
Dim Bar As WinForm.NavBar
Bar = e.Form.Controls("NavBar1")
Bar.NavPages(0).ButtonImageFile = "1601.Ico"
```

4.9.2.4.4.5.4.2.7 ButtonSmallImageFile

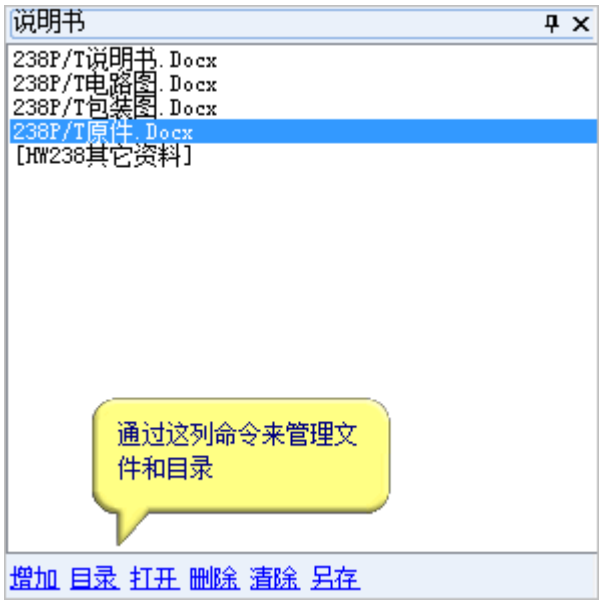
字符型，返回或设置页面的按钮小图标文件。
如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

4.9.2.4.5 界面类

4.9.2.4.5.1 FileManager(文件管理器)

4.9.2.4.5.1.1 关于 FileManager

FileManager 表示文件管理器。



FileManager 不能单独使用，必须绑定字段。
FileManager 较重要的属性有：

- 显示命令
  是否在 FileManager 的底端显示增加、删除、清除、打开等命令。

- 允许打开
  是否允许用户单击打开命令调用第三方程序来打开文件。

FileManager 提供了一组和文件管理相关的属性：



上述属性在《使用指南》之《列相关》之《扩展列列类型》之《文件与图片的管理》中已经讲述，这里就不重复了。

需要注意的是，即使 FileManager 绑定到某一列，也不会从这一列继承上述属性设置，FileManager 需要单独设置上述属性，这样虽然有所不便，但也带来了更大的灵活性。

4.9.2.4.5.1.2 属性

4.9.2.4.5.1.2.1 CommandVisible

逻辑型，是否在 FileManager 的底端显示增加、删除、清除、打开等命令。

4.9.2.4.5.1.2.2 OpenEnabled

逻辑型，是否允许用户单击打开命令调用第三方程序来打开文件。

4.9.2.4.5.1.2.3 SelectedItem

字符型，返回或设置用户在 FileManager 中选择的文件。

4.9.2.4.5.1.2.4 FTPclient

如果 FileManager 用于远程文件管理，那么可以用此属性返回 或设置其对应的 FTPClient。

示例

用代码动态设置 FileManager 的 FTPClient：

```
Dim flm As WinForm.FileManager = e.Form.Controls("FileManager1")
Dim ftp As New FTPClient
ftp.host="192.168.128.143"
ftp.Account = "yinsha"
ftp.password = "128168909"
flm.FTPclient = ftp
```

4.9.2.4.5.1.2.5 DoubleClickOpen

是否允许用户双击打开文件，默认为 True，也就是允许双击打开。

示例

```
Dim flm As WinForm.FileManager = e.Form.Controls("FileManager1")
flm.DoubleClickOpen = False
```

4.9.2.4.5.1.3 方法

4.9.2.4.5.1.3.1 AddFile

向 FileManager 中添加文件。
语法：
AddFile(FileName)
FileName: 要添加的文件名。
示例一
添加本地文件：

```
Dim dlg As New OpenFileDialog
Dim flm As WinForm.FileManager = e.Form.Controls("FileManager1")
dlg.Filter= "Excel文件|*.xls|Word文件|*.doc|Access文件|*.mdb"
If dlg.ShowDialog = DialogResult.OK Then
    flm.AddFile(dlg.FileName)
End If
```

示例二
如果想将添加的文件自动复制到 Attachments 目录，就像内置的增加按钮一项，可以将代码改为：

```
Dim dlg As New OpenFileDialog
Dim flm As WinForm.FileManager = e.Form.Controls("FileManager1")
dlg.Filter= "Excel文件|*.xls|Word文件|*.doc|Access文件|*.mdb"
If dlg.ShowDialog = DialogResult.OK Then
    Dim pth As String = ProjectPath & "Attachments\"
    If dlg.FileName.ToLower.StartsWith(pth.ToLower) Then '如果文件来自Attachments目录
        flm.AddFile(dlg.FileName.SubString(pth.Length))
    Else '如果不是
        FileSys.CopyFile(dlg.FileName, pth & FileSys.GetName(dlg.FileName))
        flm.AddFile(FileSys.GetName(dlg.FileName))
    End If
End If
```

示例三
如果使用 FTP 管理远程文件，可以参考下面的代码添加文件：

```
Dim flm As WinForm.FileManager = e.Form.Controls("FileManager1")
Dim Files As List(of String) = flm.FtpClient.SelectFile()
For Each File As String In Files
    flm.AddFile(File)
Next
```

4.9.2.4.5.1.3.2 DeleteFile

从 FileManager 中删除指定的文件。
语法：
DeleteFile(FileName)
FileName：要删除的文件名。
示例一
删除指定的文件：

```
Dim fm As WinForm.FileManager = e.Form.Controls("FileManager1")
fm.DeleteFile("abc.jpg")
```

如果添加的文件包括路径，那么删除的时候，也必须包括路径：

```
Dim fm As WinForm.FileManager = e.Form.Controls("FileManager1")
fm.DeleteFile("\photo\abc.jpg")
```

示例二

删除选定的文件：

```
Dim fm As WinForm.FileManager = e.Form.Controls("FileManager1")
Dim fl As String = fm.SelectedItem
If fl > "" Then
    fm.DeleteFile(fl)
End If
```

4.9.2.4.5.1.4 独立设置各命令

FileManager 提供了以下属性，用于返回底端的各命令：
AddCommand ：增加
AddFolderCommand：目录
DeleteCommand：删除
ClearCommand：清除
OpenCommand：打开
ManagerCommand：管理
SaveAsCommand：另存
ConnectCommand：重连
命令的属性有：
Enabled: 逻辑型，用于设置命令是否可用。
Visible: 逻辑型，用于设置命令是否可见。
Text: 字符型，用于设置命令标题。
BackColor：Color 类型，用于设置命令的背景颜色。

ForeColor：Color 类型：用于设置命令的字体颜色。

示例

希望隐藏文件管理器中的删除和清除两个命令：

```
Dim fm As WinForm.FileManager = e.Form.Controls("FileManager1")
fm.DeleteCommand.Visible = False
fm.ClearCommand.Visible = False
```

4.9.2.4.5.1.5 设置内置的文件管理器

我们知道，如果将某列的扩展列类型设置为多文件型，默认会自动打开一个文件管理器，可以通过 Col 的 FileManager 属性获得此文件管理器，进行细节设置。

示例

假定员工表的附件列是一个多文件型，我们可以通过下面的代码打开此列的文件管理器，并禁用其文件管理器的“删除”和“清除”两个命令：

```
Tables("员工").Cols("附件").OpenWindow()
Dim fl As WinForm.FileManager = Tables("员工").Cols("附件").FileManager
fl.DeleteCommand.Enabled = False
fl.ClearCommand.Enabled = False
```

4.9.2.4.5.2 PictureViewer(图片浏览器)

4.9.2.4.5.2.1 关于 PictureViewer

PictureViewer 表示图片浏览器，可以同时管理多个图片，非常方便。



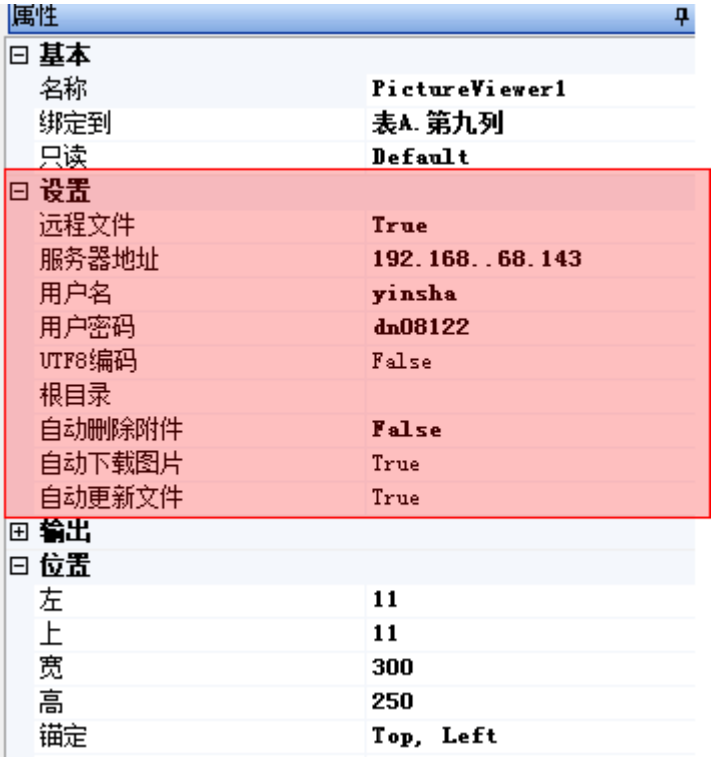
PictureViewer 不能单独使用，必须绑定字段。
PictureViewer 较重要的属性有：

- 显示命令
  是否在 PictureViewer 的底端显示增加、删除、清除、打开、全屏等命令。

- 允许打开
  是否允许用户单击打开命令调用第三方程序来打开图片文件。

- 自动缩放
  是否自动缩放图片来适用 PictureViewer 的尺寸。

PictureViewer 提供了一组和图片管理相关的属性：



上述属性在《使用指南》之《列相关》之《扩展列列类型》之《文件与图片的管理》中已经讲述，这里就不重复了。

需要注意的是，即使 PictureViewer 绑定到某一列，也不会从这一列继承上述属性设置，PictureViewer 需要单独设置上述属性，这样虽然有所不便，但也带来了更大的灵活性。

4.9.2.4.5.2.2 属性

4.9.2.4.5.2.2.1 CommandVisible

逻辑型，是否在 PictureViewer 的底端 显示增加、删除、清除、打开等命令。

4.9.2.4.5.2.2.2 OpenEnabled

逻辑型，是否允许用户单击打开命令调用第三方程序来打开图片。

4.9.2.4.5.2.2.3 AutoSize

逻辑型，是否自动缩放图片

4.9.2.4.5.2.2.4 SelectedItem

字符型，返回用户在 PictureViewer 中选择的图片文件。

4.9.2.4.5.2.2.5 FTPclient

如果 PictureViewer 用于远程文件管理，那么可以用此属性返回 或设置其对应的 FTPClient。

示例

用代码动态设置 PictureViewer 的 FTPClient：

```
Dim piv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
Dim ftp As New FTPClient
ftp.host="192.168.128.143"
ftp.Account = "yinsha"
ftp.password = "128168909"
piv.FTPclient = ftp
```

4.9.2.4.5.2.3 方法

4.9.2.4.5.2.3.1 AddFile

向 PictureViewer 中添加图片。

语法：

AddFile(FileName)

FileName: 要添加的图片文件名。

示例一

添加本地图片：

```
Dim dlg As New OpenFileDialog
Dim piv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
dlg.Filter= "图形文件|*.bmp;*.jpg;*.gif"
If dlg.ShowDialog = DialogResult.OK Then
    piv.AddFile(dlg.FileName)
End If
```

示例二

如果想将添加的图片文件自动复制到 Attachments 目录，就像内置的增加按钮一项，可以将代码改为：

```
Dim dlg As New OpenFileDialog
Dim piv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
dlg.Filter= "图形文件|*.bmp;*.jpg;*.gif"
If dlg.ShowDialog = DialogResult.OK Then
    Dim pth As String = ProjectPath & "Attachments\"
    If dlg.FileName.ToLower.StartsWith(pth.ToLower) Then '如果文件来自Attachments目录
        piv.AddFile(dlg.FileName.SubString(pth.Length))
    Else '如果不是
        FileSys.CopyFile(dlg.FileName, pth & FileSys.GetName(dlg.FileName))
```

```
        piv.AddFile(FileSys.GetName(dlg.FileName))
    End If
End If
```

示例三

如果使用 FTP 管理远程图片，可以参考下面的代码添加图片：

```
Dim piv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
Dim Files As List(of String) = piv.FtpClient.SelectFile()
For Each File As String In Files
    piv.AddFile(File)
Next
```

4.9.2.4.5.2.3.2 DeleteFile

从 PictureViewer 中删除指定的图片文件。

语法：

DeleteFile(FileName)

FileName：要删除的图片文件名。

示例一

删除指定的图片文件：

```
Dim pv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
pv.DeleteFile("abc.jpg")
```

如果添加的图片文件包括路径，那么删除的时候，也必须包括路径：

```
Dim pv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
pv.DeleteFile("\photo\abc.jpg")
```

示例二

删除选定的图片文件：

```
Dim pv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
Dim fl As String = pv.SelectedItem
If fl > "" Then
    pv.DeleteFile(fl)
End If
```

4.9.2.4.5.2.4 独立设置各命令

PictureViewer 提供了以下属性，用于返回底端的各命令：

AddCommand ：增加

DeleteCommand：删除

ClearCommand：清除
OpenCommand：打开
FullViewCommand：全屏
ManagerCommand：管理
SaveAsCommand：另存
StretchCommand：缩放
UpdateCommand：更新
ConnectCommand：重连
命令的属性有：
Enabled: 逻辑型，用于设置命令是否可用。
Visible: 逻辑型，用于设置命令是否可见。
Text： 字符型，用于设置命令标题。
BackColor：Color 类型，用于设置命令的背景颜色。
ForeColor：Color 类型：用于设置命令的字体颜色。
示例
希望隐藏图片浏览器中的删除和清除两个命令：

```
Dim pv As WinForm.PictureViewer = e.Form.Controls("PictureViewer1")
pv.DeleteCommand.Visible = False
pv.ClearCommand.Visible = False
```

4.9.2.4.5.2.5 设置内置的图片管理器

我们知道，如果将某列的扩展列类型设置为图片型，默认会自动打开一个图片浏览器，可以通过 Col 的 PictureViewer 属性获得此图片浏览器，进行细节设置。
示例
假定员工表的照片列是一个多文件型，我们可以通过下面的代码代开此列的图片浏览器，并禁用其图片浏览器的“删除”和“清除”两个命令：

```
Tables("员工").Cols("照片").OpenWindow()
Dim fl As WinForm.PictureViewer = Tables("员工").Cols("照片").PictureViewer
fl.DeleteCommand.Enabled = False
fl.ClearCommand.Enabled = False
```

4.9.2.4.5.3 TopicBar(任务栏)

4.9.2.4.5.3.1 关于 TopicBar

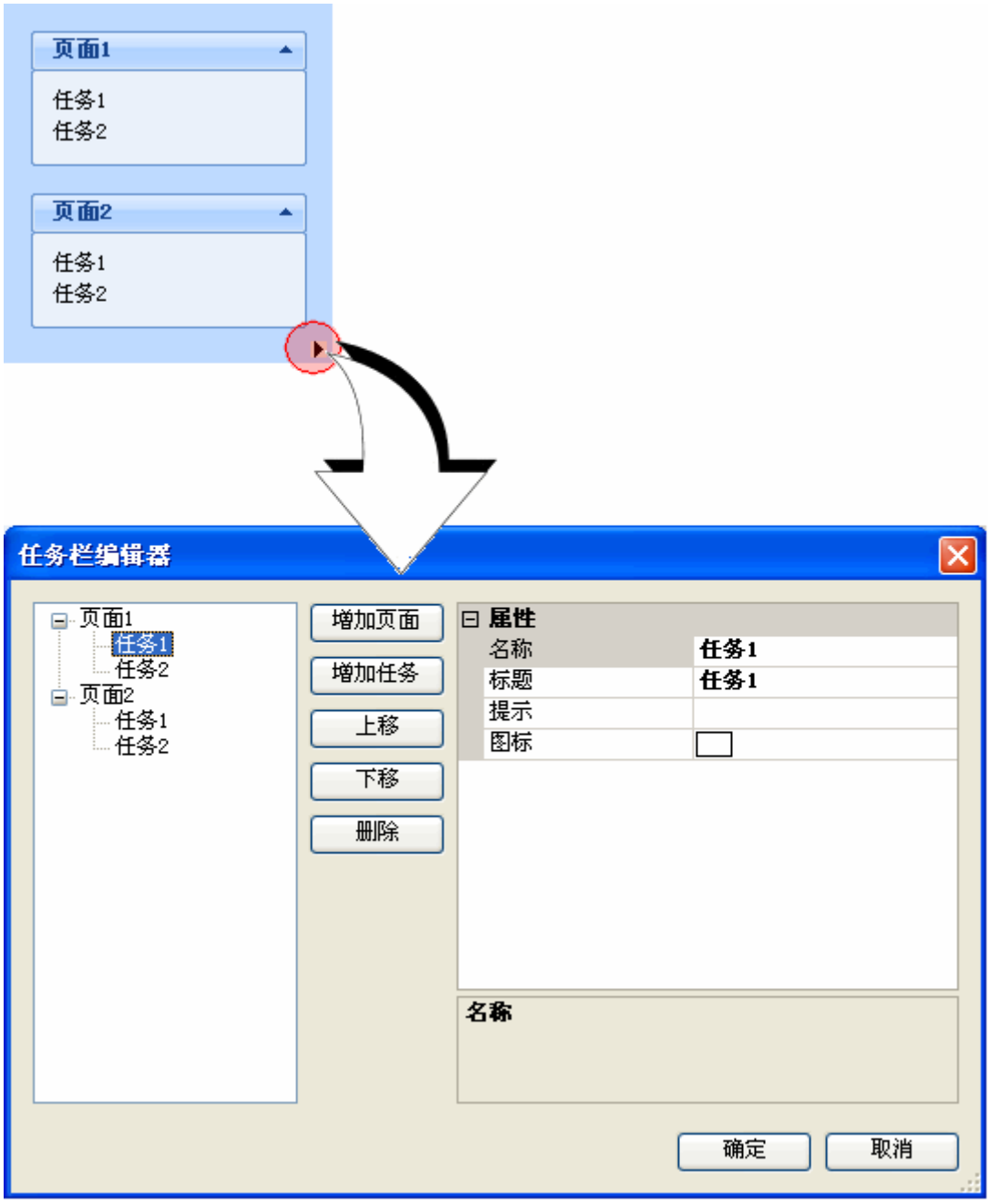
TopicBar 表示任务栏，任务栏将各种任务组织在不同的任务页面中：



TopicBar 提供了多种样式供选择，以下是其中几种：



在设计窗口的时候，任务栏的右下角有一个黑色三角形，单击该箭头出现任务栏编辑器：



可以在任务栏编辑器增加、删除、设置页面及任务，使用很简单。

运行过程中，单击任何一个任务都会产生 TopicLinkClick 事件，通过设置该事件代码，执行相应的操作，例如：

```
Select Case e.Link.Text
    Case "增加"
        '在此加入相应的代码
    Case "移动"
        '在此加入相应的代码
    Case "删除"
        '在此加入相应的代码
    Case "主页"
        '在此加入相应的代码
End Select
```

除非你希望在运行的时候，通过代码动态添加、删除、设置页面和任务，否则关于任务栏的学习就可以到此为止了。

4.9.2.4.5.3.2 类型参考

4.9.2.4.5.3.2.1 Pages

4.9.2.4.5.3.2.1.1 关于 Pages

TopicPage 集合，可以通过此集合访问指定位置或名称的任务栏页面。
TopicPage 表示任务栏页面，接下来我们会介绍。
示例

```
Dim tp1 As WinForm.TopicPage
Dim tp2 As WinForm.TopicPage
tp1 = Forms("窗口 1").Controls("TopicBar1").Pages(0)
tp2 = Forms("窗口 1").Controls("TopicBar1").Pages("项目")
```

Pages 提供了一些方法，用于动态增加和删除页面。

4.9.2.4.5.3.2.1.2 方法

4.9.2.4.5.3.2.1.2 Add

用于增加页面。
如果增加成功，则返回增加的页面。
语法：

```
Add(Name, Text)
Add(Name, Text, ImageFile)
```

Name: 页面名称
Text: 页面标题
ImageFile: 页面显示的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。
示例
增加两个页面:

```
With Forms("窗口 1").Controls("TopicBar1").Pages
    .Add("Project","项目","1601.ico")
    .Add("Plan","计划","1602.ico")
End With
```

4.9.2.4.5.3.2.1.2 Insert

用于在指定位置插入一个页面。
如果插入成功，则返回插入的页面。
语法：

```
Insert(Name, Text, Index)
Insert(Name, Text, ImageFile, Index)
```

Name: 页面名称
Text: 页面标题
Index: 页面位置
ImageFile: 页面显示的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。
示例
在最前面插入一个页面：

```
Forms("窗口 1").Controls("TopicBar1").Pages.Insert("Project", "项目", 0)
```

4.9.2.4.5.3.2.1.2 Remove

删除指定的页面。
语法

```
Rmove(Page)
```

Page：指定要删除的页面。
示例
删除名为"Project"的页面：

```
Dim pg As WinForm.TopicPage
With Forms("窗口 1").Controls("TopicBar1")
    pg = .Pages("Project")
    .Pages.Remove(pg)
End With
```

4.9.2.4.5.3.2.1.2 RemoveAt

删除指定位置的页面。
语法

```
RemoveAt(Index)
```

Index: 要删除页面的位置。
示例
删除第一个页面：

```
Dim tpb As WinForm.TopicBar
tpb = Forms("窗口 1").Controls("TopicBar1")
tpb.Pages.RemoveAt(0)
```

4.9.2.4.5.3.2.1.2 Clear

清除所有页面。
示例

```
Dim tpb As WinForm.TopicBar
tpb = Forms("窗口 1").Controls("TopicBar1")
tpb.Pages.Clear
```

4.9.2.4.5.3.2.2 TopicPage

4.9.2.4.5.3.2.2.1 关于 TopicPage

表示任务栏页面。
通过任务栏的 Pages 属性，可以访问指定位置或名称的任务栏页面。
示例

```
Dim tp1 As WinForm.TopicPage
Dim tp2 As WinForm.TopicPage
tp1 = Forms("窗口 1").Controls("TopicBar1").Pages(0)
tp2 = Forms("窗口 1").Controls("TopicBar1").Pages("项目")
```

4.9.2.4.5.3.2.2.2 属性

4.9.2.4.5.3.2.2.2 Name

返回页面的名称。

4.9.2.4.5.3.2.2.2 Text

返回或设置页面标题。

4.9.2.4.5.3.2.2.2 ToolTip

返回或设置页面提示。
鼠标停留在页面标题时，将显示此提示。

4.9.2.4.5.3.2.2.2 Visible

逻辑型，默认设为 True，设为 False，将隐藏页面。

4.9.2.4.5.3.2.2.2 Collapsed

逻辑型，设为 True，折叠页面，设为 False，展开页面。
示例
展开名为"项目"的页面：

```
Dim tp As WinForm.TopicPage
tp = Forms("窗口 1").Controls("TopicBar1").Pages("项目")
tp.Collapsed = False
```

4.9.2.4.5.3.2.2.2 ImageFile

返回或设置页面显示的图标文件，如果实现复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

4.9.2.4.5.3.2.3 Links

4.9.2.4.5.3.2.3.1 关于 Links

TopicLink 集合，通过此集合可以访问指定位置或名称的任务。
TopicLink 表示任务，接下来我们会介绍。
示例

```
Dim tl1 As WinForm.TopicLink
Dim tl2 As WinForm.TopicLink
tl1 = Forms("窗口 1").Controls("TopicBar1").Pages(0).Links(0)
tl2 = Forms("窗口 1").Controls("TopicBar1").Pages("项目").Links("打开")
```

Links 提供了一些方法，用于动态增加和删除任务。

4.9.2.4.5.3.2.3.2 方法

4.9.2.4.5.3.2.3.2 Add

用于增加任务。

如果增加成功，则返回增加的任务。

语法：

```
Add(Name, Text)
Add(Name, Text, ImageFile)
```

Name: 任务名称

Text: 任务标题

ImageFile: 任务显示的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

例如:

```
Dim Page As WinForm.TopicPage
Page = Forms("窗口 1").Controls("TopicBar1").Pages.Add("Project","项目")
Page.Links.Add("Open","打开")
Page.Links.Add("Help","帮助")
```

4.9.2.4.5.3.2.3.2 Insert

用于在指定位置插入一个任务。

如果插入成功，则返回插入的任务。

语法：

```
Insert(Name, Text, Index)
Insert(Name, Text, Image, Index)
```

Name: 任务名称

Text: 任务标题

ImageFile: 任务显示的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

Index：插入位置

4.9.2.4.5.3.2.3.2 Remove

删除指定的任务。

语法

```
Rmove(Link)
```

Page：指定要删除的任务。

示例

```
With Forms("窗口 1").Controls("TopicBar1").Pages("项目")
    .Links.Remove(.Links("打开"))
End With
```

4.9.2.4.5.3.2.3.2 RemoveAt

删除指定位置的任务。

语法

RemoveAt(Index)

Index: 要删除任务的位置。

示例

```
With Forms("窗口 1").Controls("TopicBar1").Pages("项目")
    .Links.RemoveAt(0)
End With
```

4.9.2.4.5.3.2.3.2 Clear

清除所有任务。

示例

```
Dim tp As WinForm.TopicPage
tp = Forms("窗口 1").Controls("TopicBar1").Pages("项目")
tp.Links.Clear()
```

4.9.2.4.5.3.2.4 TopicLink

4.9.2.4.5.3.2.4.1 关于 TopicLink

表示一个任务栏中的任务。

通过页面的 TopicLinks 属性，可以访问指定位置或名称的任务。

示例

```
Dim tl1 As WinForm.TopicLink
Dim tl2 As WinForm.TopicLink
tl1 = Forms("窗口 1").Controls("TopicBar1").Pages(0).Links(0)
tl2 = Forms("窗口 1").Controls("TopicBar1").Pages("项目").Links("打开")
```

4.9.2.4.5.3.2.4.2 属性

4.9.2.4.5.3.2.4.2 Name

返回任务的名称。

4.9.2.4.5.3.2.4.2 Text

返回或设置任务的标题。

4.9.2.4.5.3.2.4.2 ToolTip

返回或设置任务提示。

鼠标停留在任务时，将显示此提示。

4.9.2.4.5.3.2.4.2 Visible

逻辑型，默认设为 True，设为 False，将隐藏任务。

4.9.2.4.5.3.2.4.2 ImageFile

返回或设置任务显示的图标文件，如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

```
Dim tl As WinForm.TopicLink
tl = Forms("窗口 1").Controls("TopicBar1").Pages(0).Links(0)
tl.ImageFile = "1601.ico"
```

4.9.2.4.5.3.2.5 TopicBar

4.9.2.4.5.3.2.5.1 事件

4.9.2.4.5.3.2.5.1 TopicLinkClick

单击任务栏中某个任务后执行。

e 参数属性：

Sender：返回触发事件的控件

Form: 返回控件所在的窗口

Link: 所单击的任务

Page: 任务所在的页面。

通常利用此事件，判断单击的是哪个页面中的哪个任务，然后执行相应的操作。

示例

```
Select Case e.Link.Name
    Case "增加"
        '在此加入相应的代码
    Case "移动"
        '在此加入相应的代码
    Case "删除"
        '在此加入相应的代码
    Case "主页"
        '在此加入相应的代码
End Select
'MessageBox.Show(e.Page.Name & "|" & e.Link.Name)
```

4.9.2.4.5.3.2.5.1 TopicPageExpanded

展开某个页面后执行。

e 参数属性：

Sender：返回触发事件的控件

Form: 返回控件所在的窗口

Page: 返回展开的页面。

示例

例如希望展开一个页面的同时，关闭其它页面，为此可将 TopicPageExpanded 事件代码设置为：

```
Dim tpb As WinForm.TopicBar = e.Form.Controls("TopicBar1")
For Each page As WinForm.TopicPage In tpb.Pages
    If page.Name <> e.Page.Name Then
        page.Collapsed = True
    End If
Next
```

4.9.2.4.5.3.2.5.1 TopicPageCollapsed

关闭某个页面后执行。

e 参数属性：

Sender：返回触发事件的控件

Form: 返回控件所在的窗口

Page: 返回关闭的页面。

4.9.2.4.5.4 TreeView(目录树)

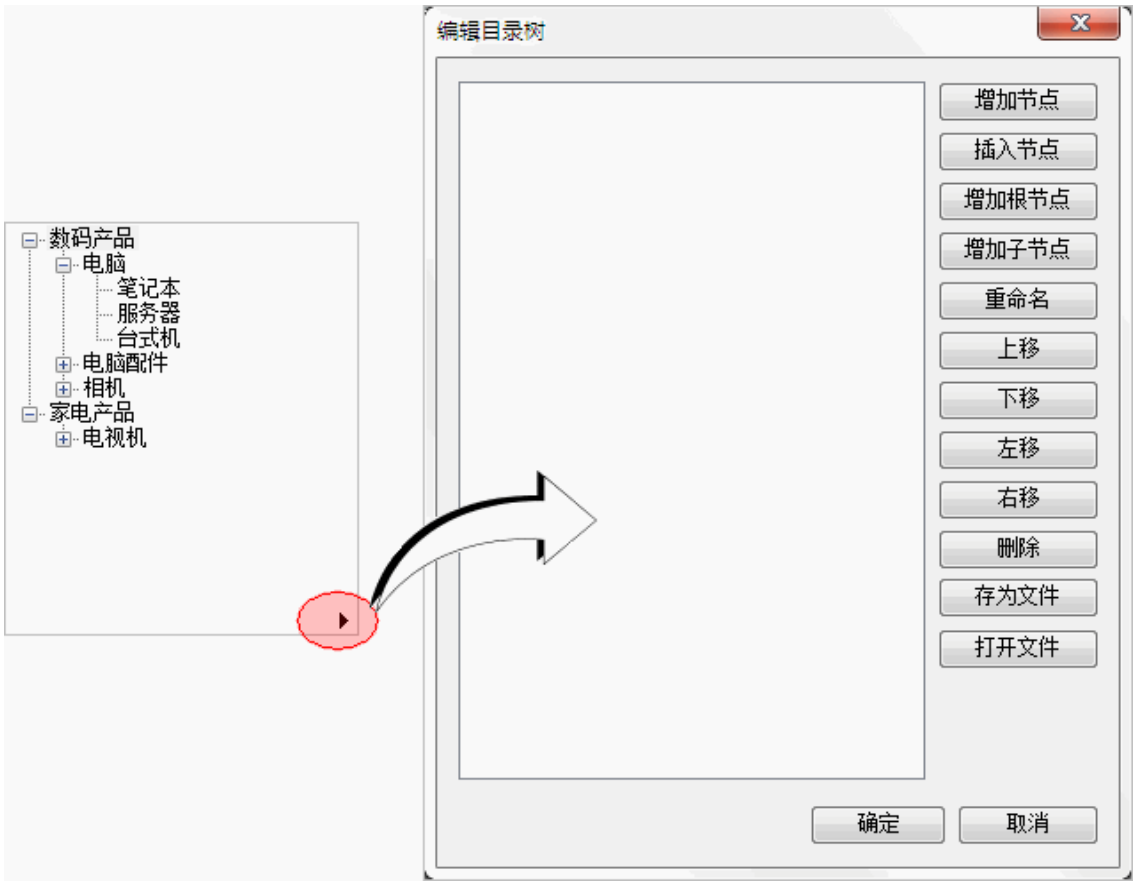
4.9.2.4.5.4.1 关于 TreeView

如果有大量的命令需要组织，可以考虑 TreeView。

Tree 表示目录树，目录树由节点组成，节点还可以包括子节点，层数不受限制：



在设计窗口的时候，单击目录树右下角的黑色箭头，会出现一个窗口，利用该窗口，可以增加、删除、移动节点：



属性窗口提供了一组属性，用于对选定节点进行更具体的设置:



在运行过程中，单击节点会执行 NodeMouseClick 事件，双击节点会执行 NodeMouseDoubleClick 事件。

例如你希望双击节点后执行特定的代码，可以设置 NodeMouseDoubleClick 事件的代码为：

```
'如果希望通过节点标题来判断，请改为：
'Select Case e.Node.Text
Select Case e.Node.Name
```

```
    Case "名称 1"
        '在此加入相应的代码
    Case "名称 2"
        '在此加入相应的代码
    Case "名称 3"
        '在此加入相应的代码
    Case "名称 4"
    '在此加入相应的代码
End Select
```

如果你没有更多的要求，那么关于目录树的学习，就可以到此为止了。

4.9.2.4.5.4.2 自动生成目录树

4.9.2.4.5.4.2.1 根据多列内容生成目录树

目录树有一个 BuildTree 方法，可以根据指定的数据表内容生成目录树节点。

语法：

BuildTree(DataTableName, Columns, Filter, Sort)

或

BuildTree(DataTable, Columns, Filter, Sort)

DataTableName: 数据表名

DataTable： 数据表

Columns: 指定数据列，不同的列之间用符号“|”隔开，例如：产品|客户|雇员”

Filter: 可选参数，指定过滤条件，请参考表达式的运算符和函数 和 条件表达式。

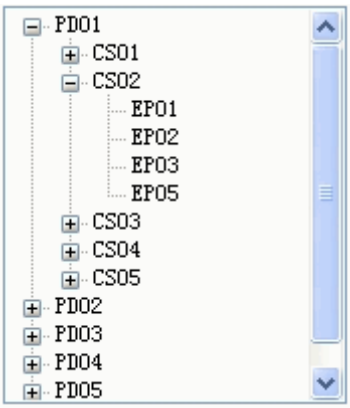
Sort： 可选参数，指定排序列。

示例一

在窗口中加入一个目录树(TreeView)，再加入一个按钮(Button)，按钮的 Click 事件代码设置为：

```
Dim tv As WinForm.TreeView
tv =e.Form.Controls("TreeView1")
tv.BuildTree("订单", "产品|客户|雇员")
```

在运行的时候，单击该按钮，即可生成目录树节点：



示例二

生成目录树的时候，不仅可以指定生成条件，还可以根据需要指定排序列，需要注意的时，你根

据多少列生成目录树，就得指定多少个排序列，例如：

```
Dim tv As WinForm.TreeView
tv =e.Form.Controls("TreeView1")
tv.BuildTree("产品", "类别|品名", "", "类别,价格  DESC")
```

表示根据产品表的类别和品名生成目录树，相同类别的产品，按价格由高到低的顺序排列。
如果你单单指定根据价格排序:

```
Dim tv As WinForm.TreeView
tv =e.Form.Controls("TreeView1")
tv.BuildTree("产品", "类别|品名", "", "价格  DESC")
```

生成的目录树有可能会出现错乱。

4.9.2.4.5.4.2.2 根据单列内容生成目录树

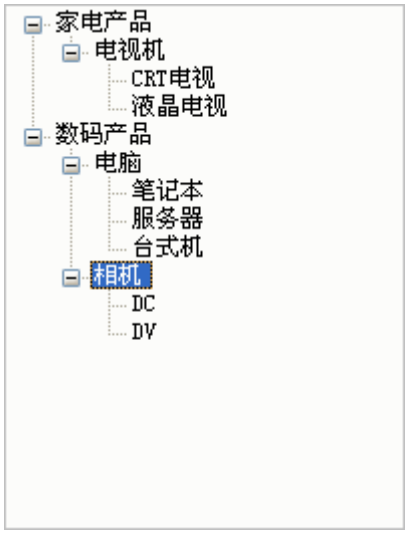
BuildTree 能够根据指定的某几列生成目录树，但是有的时候，目录树的内容并非分开放在不同的列中，而是用符号分割放在同一列中。
根据单列内容生成目录树需要使用 CreateTree 方法。
语法：
CreateTree(DataTableName, ColumnName, Separator,Filter,Sort)
或
CreateTree(DataTable, ColumnName, Separator,Filter,Sort)
DataTableName: 数据表名
DataTable： 数据表
ColumnName： 列名
Separator： 可选参数，设置分隔符，默认分隔符为“\”
Filter: 可选参数，设置过滤条件，请参考表达式的运算符和函数 和 条件表达式。
Sort: 可选参数，设置排序，默认根据 ColumnName 参数指定的列排序
提示：如果分割符号不是“\”，则 Sort 参数无效。
示例
例如有下表中的数据：

	类别
1	数码产品\电脑\笔记本
2	数码产品\电脑\台式机
3	数码产品\电脑\服务器
4	数码产品\相机\DC
5	数码产品\相机\DV
6	家电产品\电视机\液晶电视
7	家电产品\电视机\CRT电视

在窗口中加入一个目录树(TreeView)，再加入一个按钮(Button)，按钮的 Click 事件代码设置为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
tr.CreateTree("表 A","类别")
```

单击此按钮，即可生成下图所示的目录数：



4.9.2.4.5.4.2.3 根据路径列和标题列生成目录树

有的时候，目录树要根据两列内容生成，一列指定路径，一列指定标题，根据两列内容生成目录树需要使用 GenerateTree 方法。

语法：

GenerateTree(DataTableName, PathCol, TextCol, Separator, Filter, Sort)

或

GenerateTree(DataTable, PathCol, TextCol, Separator, Filter, Sort)

DataTableName: 数据表名

DataTable： 数据表

PathCol： 路径列名

TextCol： 标题列名

Separator： 可选参数，设置分隔符，默认分隔符为“\”

Filter: 可选参数，设置过滤条件，请参考表达式的运算符和函数 和 条件表达式。

Sort: 可选参数，设置排序，默认根据 ColumnName 参数指定的列排序

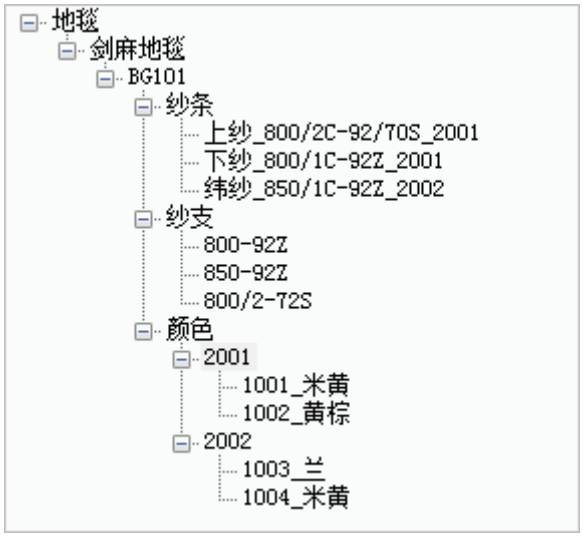
提示：如果分割符号不是“\”，则 Sort 参数无效。

示例

假定有下图所示的物料编码表：

	级码	描述	备注
1	1	地毯	
2	1.1	剑麻地毯	
3	1.1.1	BG101	BG101
4	1.1.1.1	纱条	
5	1.1.1.1.1	上纱_800/2C-92/70S_2001	800/2C-92/70S_2001
6	1.1.1.1.2	下纱_800/1C-92Z_2001	800/1C-92Z_2001
7	1.1.1.1.3	纬纱_850/1C-92Z_2002	850/1C-92Z_2001
8	1.1.1.2	纱支	
9	1.1.1.2.1	800-92Z	800-92Z
10	1.1.1.2.2	850-92Z	850-92Z
11	1.1.1.2.3	800/2-72S	800/2-72S
12	1.1.1.3	颜色	
13	1.1.1.3.1	2001	2001
14	1.1.1.3.1.1	1001_米黄	1001
15	1.1.1.3.1.2	1002_黄棕	1002
16	1.1.1.3.2	2002	2002
17	1.1.1.3.2.1	1003_兰	1003
18	1.1.1.3.2.2	1004_米黄	1004

希望根据“级码”列生成一个目录树，但是目录树节点的标题来自“描述”列：



代码非常简单，只有两行：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.GenerateTree("物料编码表","级码","描述", ".")
```

提示： 由于级码列的路径分隔符是".”，而不是不是默认的"\”，所以上面的代码需要明确指定分割符为".”。

4.9.2.4.5.4.3 类型参考

4.9.2.4.5.4.3.1 Nodes

4.9.2.4.5.4.3.1.1 关于 Nodes

节点集合，可以通过此集合访问指定名称或位置的节点。
节点的类型为 TreeNode，后面会介绍。
示例一
下面的代码获得目录树的第一个节点，并将其标题设为“筛选”：

```
Dim tr As WinForm.TreeView
Dim td As WinForm.TreeNode
tr = Forms("窗口 1").Controls("TreeView1")
td = tr.Nodes(0)
td.Text = "筛选"
```

不仅目录树有节点属性，节点本身也有 Nodes 属性，返回其子节点集合
示例二
获得目录树第一个节点的第一个子节点，并将其标题设为“筛选”：

```
Dim tr As WinForm.TreeView
Dim td As WinForm.TreeNode
tr = Forms("窗口 1").Controls("TreeView1")
td = tr.Nodes(0)
td = td.Nodes(0)
td.Text = "筛选"
```

你可以将上面的代码改写为一行：

```
Forms("窗口 1").Controls("TreeView1").Nodes(0).Nodes(0).Text = "筛选"
```

Nodes 还提供了一些方法，用于增加、插入、删除节点。

4.9.2.4.5.4.3.1.2 方法

4.9.2.4.5.4.3.1.2 Add

增加节点。
如果增加成功，则返回所增加的节点。
语法

```
Add(Name)
Add(Name, Text)
Add(Name, Text, IconFile)
```

参数

- Name
  新增节点的名称。

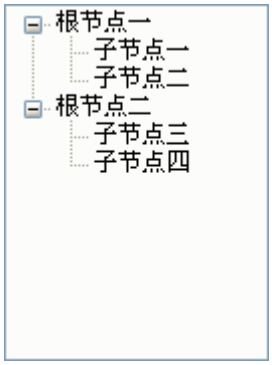
- Text
  新增节点的标题，如果省略，则自动用名称作为标题。

- IconFile
  新增节点显示的图标文件，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

示例
在窗口中插入一个目录树、一个按钮，将按钮的 Click 事件代码设为：

```
Dim tr As WinForm.TreeView
Dim nd As WinForm.TreeNode
tr = Forms("窗口 1").Controls("TreeView1")
nd = Tr.Nodes.Add("Root1","根节点一")
nd.Nodes.Add("Sub1","子节点一")
nd.Nodes.Add("Sub2","子节点二")
nd = Tr.Nodes.Add("Root2","根节点二")
nd.Nodes.Add("Sub3","子节点三")
nd.Nodes.Add("Sub4","子节点四")
```

运行时单击按钮，会生成目录树：



4.9.2.4.5.4.3.1.2 Insert
在指定位置插入一个节点。
如果插入成功，则返回所插入的节点。
语法

```
Insert(Name, Index)
Insert(Name, Text, Index)
Insert(Name, Text, Icon, Index)
```

参数说明

- Name
  表示新增节点的名称。

- Text
  表示新增节点的标题，如果省略，则自动用名称作为标题。

- Icon
  新增节点显示的图标文件，如果文件已经事先复制到管理项目的 Images 子目录下,则只需

指定文件名即可,否则需要包括路径。

- Index
  表示新增节点插入的位置。

示例

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes.Insert("Node0","节点 1",0)
```

4.9.2.4.5.4.3.1.2 Remove

删除指定的节点。

示例

删除名为"Filter"的节点：

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes.Remove(tr.Nodes("Filter"))
```

4.9.2.4.5.4.3.1.2 RemoveAt

删除指定位置的节点。

语法

RemoveAt(Index)

Index：要删除节点的位置。

示例

删除目录树的第二个节点：

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes.RemoveAt(1)
```

4.9.2.4.5.4.3.1.2 Contains

判断是否存在指定名称的节点。

示例

```
Dim tr As WinForm.TreeView
tr = e.Form.Controls("TreeView1")
If tr.Nodes.Contains("北京") Then
    MessageBox.Show("节点已经存在")
Else
    tr.Nodes.Add("北京")
End If
```

4.9.2.4.5.4.3.1.2 Clear

清除所有节点。

示例

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes.Clear
```

4.9.2.4.5.4.3.2 TreeNode

4.9.2.4.5.4.3.2.1 关于 TreeNode

表示目录树节点。
通过目录树的 Nodes 属性，可以获得目录树指定名称或位置的节点。
通过节点的 Nodes 属性，可以获得指定名称或位置的子节点。
示例一
下面的代码获得目录树的第一个节点，并将其标题设为“筛选”：

```
Dim tr As WinForm.TreeView
Dim td As WinForm.TreeNode
tr = Forms("窗口 1").Controls("TreeView1")
td = tr.Nodes(0)
td.Text = "筛选"
```

示例二
获得目录树第一个节点的第一个子节点，并将其标题设为“筛选”：

```
Dim tr As WinForm.TreeView
Dim td As WinForm.TreeNode
tr = Forms("窗口 1").Controls("TreeView1")
td = tr.Nodes(0)
td = td.Nodes(0)
td.Text = "筛选"
```

4.9.2.4.5.4.3.2.2 属性

4.9.2.4.5.4.3.2.2 Name

字符型，返回节点名称
提示，目录树允许同名节点的存在。

4.9.2.4.5.4.3.2.2 Text

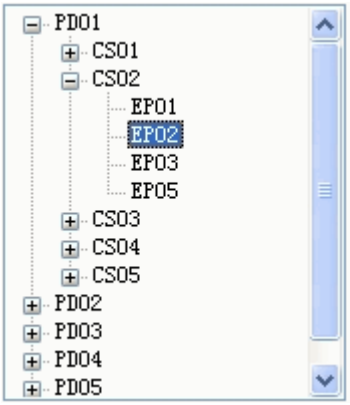
字符型，返回或设置节点标题。

4.9.2.4.5.4.3.2.2 Checked

逻辑型，是否选中目录节点的复选框。

4.9.2.4.5.4.3.2.2 FullPath

字符型，返回节点路径。
例如对于下图选定的节点，其 FullPath 属性为“PD01\CS02\EP02”。



4.9.2.4.5.4.3.2.2 FullName

字符型，返回节点的全路径名称。

注意 FullName 和 FullPah 的差别，FullName 由各级节点的名称组成，而 FullPath 由各级节点的标题组成。

多数时候，节点的标题和名称是一致的，所以 FullName 和 FullPah 多数时候是相同的。

示例

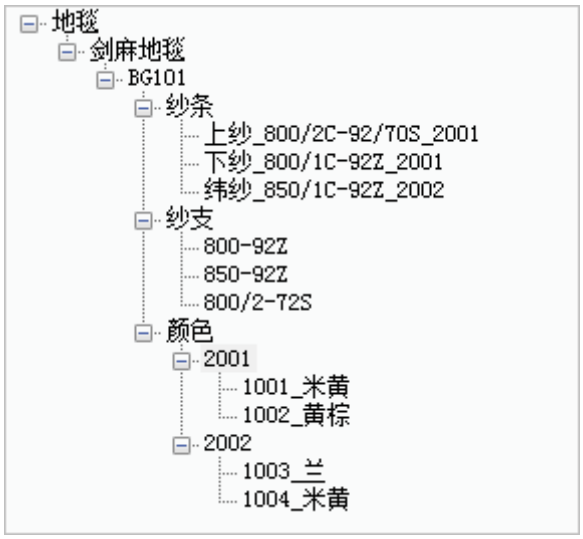
假定有下图所示的物料编码表：

	级码	描述	备注
1	1	地毯	
2	1.1	剑麻地毯	
3	1.1.1	BG101	BG101
4	1.1.1.1	纱条	
5	1.1.1.1.1	上纱_800/2C-92/70S_2001	800/2C-92/70S_2001
6	1.1.1.1.2	下纱_800/1C-92Z_2001	800/1C-92Z_2001
7	1.1.1.1.3	纬纱_850/1C-92Z_2002	850/1C-92Z_2001
8	1.1.1.2	纱支	
9	1.1.1.2.1	800-92Z	800-92Z
10	1.1.1.2.2	850-92Z	850-92Z
11	1.1.1.2.3	800/2-72S	800/2-72S
12	1.1.1.3	颜色	
13	1.1.1.3.1	2001	2001
14	1.1.1.3.1.1	1001_米黄	1001
15	1.1.1.3.1.2	1002_黄棕	1002
16	1.1.1.3.2	2002	2002
17	1.1.1.3.2.1	1003_兰	1003
18	1.1.1.3.2.2	1004_米黄	1004

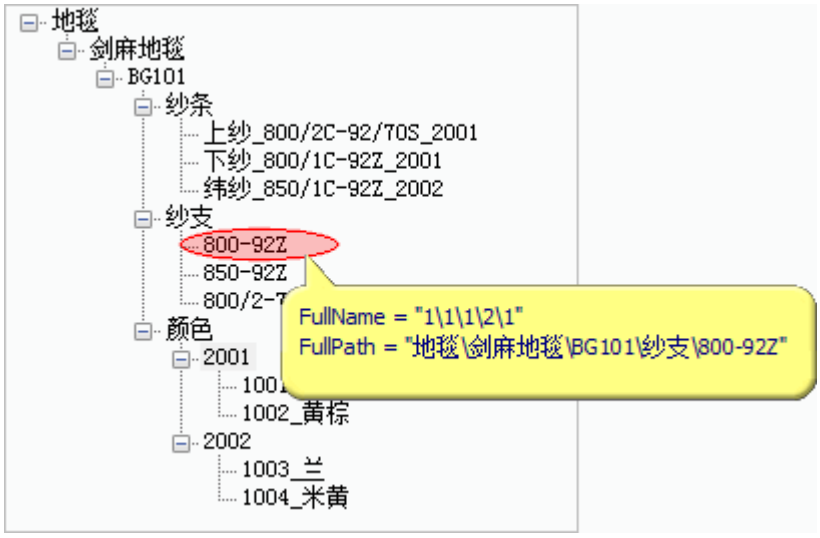
通过下面的代码：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.GenerateTree("物料编码表","级码","描述", ".")
```

可以根据“级码”列生成一个目录树，节点的标题来自“描述”列：



下图可以说明 FullName 和 FullPath 的差别：



如果希望双击某个节点，能够从物料编码表中筛选出此节点(含子节点)下的所有行，可以将 NodeMouseDoubleClick 事件代码设置为：

```
Dim bm As String = e.Node.FullName
Tables("物料编码表").Filter = "级码 = '" & bm & "' Or 级码 Like '" & bm & ".%'"
```

4.9.2.4.5.4.3.2.2 ParentNode

返回父节点。

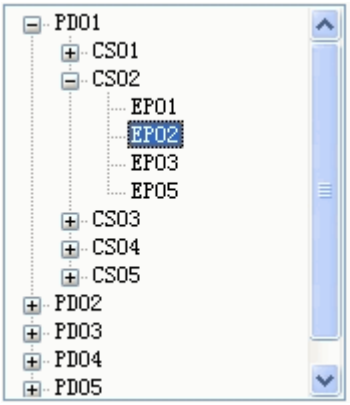
示例：

```
Dim tr As WinForm.TreeView
Dim pd As WinForm.TreeNode
tr = Forms("窗口 1").Controls("TreeView1")
pd = tr.SelectedNode.ParentNode
pd.Nodes.Add("新节点")
```

4.9.2.4.5.4.3.2.2 Level

整数型，返回节点所的层数，根节点是 0。

例如对于下图所选节点，它的 Level 属性为 2：



4.9.2.4.5.4.3.2.2 Index

```
tr As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim nd As WinForm.TreeNode = tr.SelectedNode
Dim id As Integer = nd.index
If nd.ParentNode Is Nothing Then
    nd = tr.Nodes.Insert("新节点","新节点",id)
Else
    nd = nd.ParentNode.Nodes.Insert("新节点","新节点",id)
End If
tr.SelectedNode = nd '选择新插入的节点
tr.Select
```

4.9.2.4.5.4.3.2.2 DataRow

如果目录树是通过 BuildTree、CreateTree、GenerateTree 方法根据某数据表内容自动生成的，那么可以通过节点的 DataRow 属性，返回此节点在数据表中对应的行。

需要注意的是，一个节点可能对应有多个行，DataRow 属性将只返回其中一行。

有的时候，这个属性可以大大简化我们的编码。

4.9.2.4.5.4.3.2.2 Tooltip

字符型，返回或设置节点提示。

4.9.2.4.5.4.3.2.2 IconFile

返回或者设置节点显示的图标文件。

如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

示例

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes(0).IconFile = "C:\Icons\Clone.ico"
```

4.9.2.4.5.4.3.2.2 SelectedIconFile

返回或者设置节点处于选中状态时，其显示的图标文件。

如果事先复制到项目文件的 Images 子目录下，则无需路径，否则必须包括路径。

4.9.2.4.5.4.3.2.2 Backcolor

返回或设置节点的背景颜色。

4.9.2.4.5.4.3.2.2 ForeColor

返回或设置节点的字体颜色。

示例

将第二个节点的颜色设为红色：

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes(1).ForeColor = Color.Red
```

4.9.2.4.5.4.3.2.2 IsExpanded

逻辑型，判断节点是否已经展开，如果节点已经展开返回 True，否则返回 False。

4.9.2.4.5.4.3.2.2 IsSelected

逻辑型，判断节点是否已经被选中，如果选中，返回 True，否则返回 False。

4.9.2.4.5.4.3.2.3 方法

4.9.2.4.5.4.3.2.3 Collapse

关闭节点。

示例

关闭第二个节点：

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes(1).Collapse()
```

4.9.2.4.5.4.3.2.3 CollapseAll

关闭节点及其子节点。

4.9.2.4.5.4.3.2.3 Expand

展开节点。

示例：

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.Nodes(1).Expand()
```

4.9.2.4.5.4.3.2.3 ExpandAll

展开节点及其子节点。

4.9.2.4.5.4.3.2.3 Toggle

切换次节点的展开和关闭状态。

4.9.2.4.5.4.3.2.3 EnsureVisible

确保节点可见。

示例

假定有一个下图所示的行政区域表：

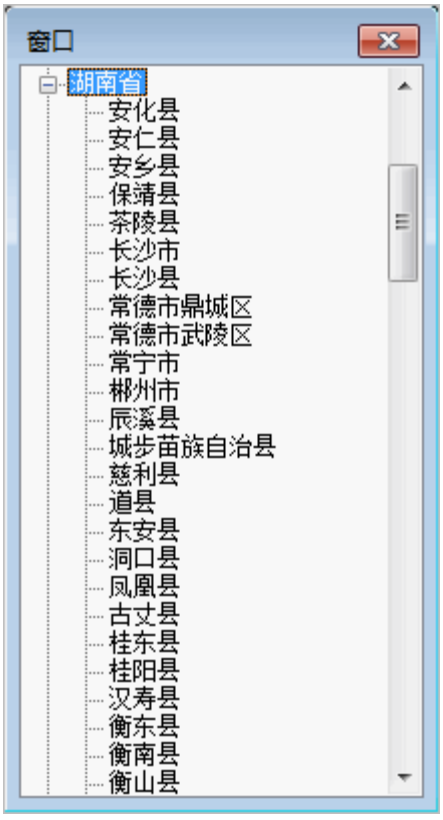
	省	县市	区号	邮编
1	北京市	北京市	10	100000
2	北京市	东城区	10	100010
3	北京市	西城区	10	100030
4	北京市	崇文区	10	100060
5	北京市	宣武区	10	100050
6	北京市	朝阳区	10	100020

窗口有个目录树，根据此表生成，希望打开窗口的时候，能够自动展开并滚动“湖南省”这个节点至顶端，使得其子节点可见。

为此可以将窗口的 AfterLoad 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市") '生成目录树
Dim nd As WinForm.TreeNode = trv.Nodes("湖南省") '获得名为“湖南省”的节点
Dim idx As Integer = Math.MIn(20, nd.Nodes.Count - 1) '获得“湖南省”的第 20 个或最后一个子节点的位置
trv.SelectedNode = nd '选择"湖南省"节点
nd.Nodes(idx).EnsureVisible '使得湖南省的第 20 个或最后一个子节点可见.
```

上述代码的原理很简单，通过使“湖南省”的第 20 个子节点可见，间接达到展开并滚动“湖南省”这个节点至顶端的目的：



4.9.2.4.5.4.3.2.3 MoveUp

向上移动目录树节点。

4.9.2.4.5.4.3.2.3 MoveDown

向下移动目录树节点。

4.9.2.4.5.4.3.2.3 MoveLeft

向左移动目录树节点。

4.9.2.4.5.4.3.2.3 MoveRight

向右移动目录树节点。

4.9.2.4.5.4.3.2.3 Delete

删除目录树节点。

4.9.2.4.5.4.3.3 TreeView

4.9.2.4.5.4.3.3.1 属性

4.9.2.4.5.4.3.3.1 SelectedNode

TreeNode 类型，返回或者设置选定的节点。

示例

选定目录树第一个节点的第二个子节点：

```
Dim tr As WinForm.TreeView
tr = Forms("窗口 1").Controls("TreeView1")
tr.SelectedNode = tr.Nodes(0).Nodes(1)
```

4.9.2.4.5.4.3.3.1 CheckBoxes

逻辑型，是否显示复选框。

默认为 False，设为 True，将显示复选框。

下图是一个显示了复选框的目录树：



4.9.2.4.5.4.3.3.2 方法

4.9.2.4.5.4.3.3.2 BuildTree

根据指定的数据表内容生成目录树节点。

语法：

BuildTree(DataTableName, Columns, Filter, Sort)

或

BuildTree(DataTable, Columns, Filter, Sort)

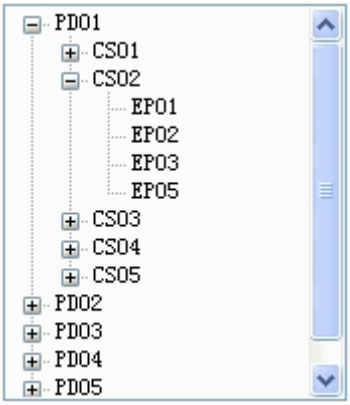
DataTableName：	数据表名
DataTable：	数据表
Columns:	指定数据列，不同的列之间用符号“|”隔开，例如：产品|客户|雇员”
Filter:	可选参数，指定过滤条件，请参考表达式的运算符和函数 和 条件表达式
Sort:	可选参数，设置排序，默认根据 ColumnName 参数指定的列排序

实例：

在窗口中加入一个目录树(TreeView)，再加入一个按钮(Button)，按钮的 Click 设置为：

```
Dim tv As WinForm.TreeView
tv =e.Form.Controls("TreeView1")
tv.BuildTree("订单", "产品|客户|雇员")
```

在运行的时候，单击该按钮，即可生成目录树节点：



4.9.2.4.5.4.3.3.2 CreateTree

根据单列内容生成目录树。

语法：

```
CreateTree(DataTableName, ColumnName, Separator,Filter,sort)
```

或

```
CreateTree(DataTable, ColumnName, Separator,Filter,sort)
```

DataTableName: 数据表名

DataTable： 数据表

ColumnName： 列名

Separator： 可选参数，设置分隔符，默认分隔符为“\”

Filter: 可选参数，设置过滤条件，请参考表达式的运算符和函数 和 条件表达式。

Sort: 可选参数，设置排序，默认根据 ColumnName 参数指定的列排序

提示：如果分割符号不是“\”，则 Sort 参数无效。

示例

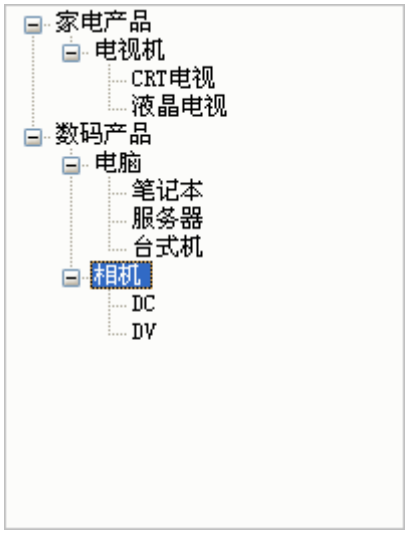
例如有下表中的数据：

	类别
1	数码产品\电脑\笔记本
2	数码产品\电脑\台式机
3	数码产品\电脑\服务器
4	数码产品\相机\DC
5	数码产品\相机\DV
6	家电产品\电视机\液晶电视
7	家电产品\电视机\CRT电视

在窗口中加入一个目录树(TreeView)，再加入一个按钮(Button)，按钮的 Click 事件代码设置为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
tr.CreateTree("表 A","类别")
```

单击此按钮，即可生成下图所示的目录数：



4.9.2.4.5.4.3.3.2 GenerateTree

有的时候，目录树要根据两列内容生成，一列指定路径，一列指定标题，根据两列内容生成目录树需要使用 GenerateTree 方法。

语法：

GenerateTree(DataTableName, PathCol, TextCol, Separator, Filter, Sort)

或

GenerateTree(DataTable, PathCol, TextCol, Separator, Filter, Sort)

DataTableName: 数据表名

DataTable： 数据表

PathCol： 路径列名

TextCol： 标题列名

Separator： 可选参数，设置分隔符，默认分隔符为“\”

Filter: 可选参数，设置过滤条件，请参考表达式的运算符和函数 和 条件表达式。

Sort: 可选参数，设置排序，默认根据 ColumnName 参数指定的列排序

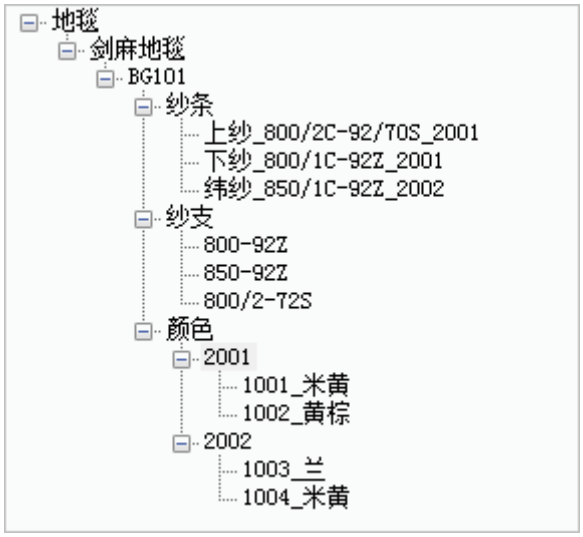
提示：如果分割符号不是“\”，则 Sort 参数无效。

示例

假定有下图所示的物料编码表：

	级码	描述	备注
1	1	地毯	
2	1.1	剑麻地毯	
3	1.1.1	BG101	BG101
4	1.1.1.1	纱条	
5	1.1.1.1.1	上纱_800/2C-92/70S_2001	800/2C-92/70S_2001
6	1.1.1.1.2	下纱_800/1C-92Z_2001	800/1C-92Z_2001
7	1.1.1.1.3	纬纱_850/1C-92Z_2002	850/1C-92Z_2001
8	1.1.1.2	纱支	
9	1.1.1.2.1	800-92Z	800-92Z
10	1.1.1.2.2	850-92Z	850-92Z
11	1.1.1.2.3	800/2-72S	800/2-72S
12	1.1.1.3	颜色	
13	1.1.1.3.1	2001	2001
14	1.1.1.3.1.1	1001_米黄	1001
15	1.1.1.3.1.2	1002_黄棕	1002
16	1.1.1.3.2	2002	2002
17	1.1.1.3.2.1	1003_兰	1003
18	1.1.1.3.2.2	1004_米黄	1004

希望根据“级码”列生成一个目录树，但是目录树节点的标题来自“描述”列：



代码非常简单，只有两行：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.GenerateTree("物料编码表","级码","描述", ".")
```

提示： 由于级码列的路径分隔符是".”，而不是不是默认的"\”，所以上面的代码需要明确指定分割符为".”。

4.9.2.4.5.4.3.3.2 Save

将目录树保存为一个文件。

语法

Save(FileName, All)

FileName：文件名，含路径。

All: 可选参数，逻辑型，是否保存节点的所有属性，默认为 False，只保存节点的名称和标题属

性。

4.9.2.4.5.4.3.3.2 Load

根据指定的文件生成目录树。

语法

Load(FileName)

FileName：文件名，含路径。

示例

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.Nodes.Clear() '清除现有节点
trv.Load("c:\data\product.tree")
```

4.9.2.4.5.4.3.3.2 SaveToString

将目录树保存为一个字符串，可以将这个字符串存储在数据表中，在需要的时候再用 LoadFromString 方法根据这个字符串生成目录树。

语法：

SaveToString(All)

All: 可选参数，逻辑型，是否保存节点的所有属性，默认为 False，只保存节点的名称和标题属性。

示例

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim val As String
val = trv.SaveToString()
```

4.9.2.4.5.4.3.3.2 LoadFromString

根据指定的字符串生成目录树。

通常用 SaveToString 方法将目录树保存为一个字符串，然后存储在数据表中，在需要的时候用 LoadFromString 方法根据这个字符串生成目录树。

4.9.2.4.5.4.3.3.2 CollapseAll

关闭目录树所有节点。

4.9.2.4.5.4.3.3.2 ExpandAll

展开目录树所有节点。

4.9.2.4.5.4.3.3.2 BeginEdit

编辑选定节点的标题。

4.9.2.4.5.4.3.3.2 StopRedraw

停止绘制目录树。

运行过程中，需要向目录树大量增加和删除节点时，应该首先执行 StopRedraw 方法，停止绘制目录树，增加或删除节点结束后，再执行 ResumeRedraw 方法，恢复绘制目录树。

示例：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
tr.StopRedraw()
'添加或删除节点的代码
tr.ResumeRedraw()
```

4.9.2.4.5.4.3.3.2 ResumeRedraw

恢复绘制目录树，请参考 StopRedraw。

4.9.2.4.5.4.3.3.3 事件

4.9.2.4.5.4.3.3.3 AfterSelectNode

选择节点后执行。

e 参数属性

Form: 触发事件的窗口

Sender: 触发事件的目录树

Node: 所选择的节点

4.9.2.4.5.4.3.3.3 AfterCheckNode

改变节点复选框状态后执行。

e 参数属性

Form: 触发事件的窗口

Sender: 触发事件的目录树

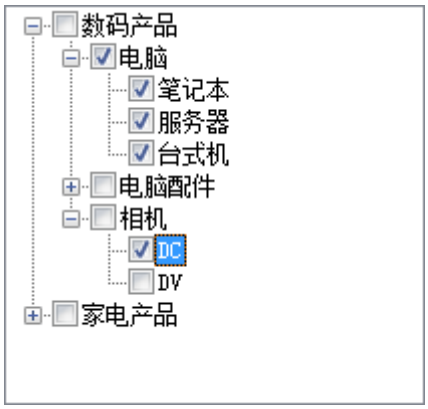
Node: 触发事件的节点

示例

假定有下图所示的目录树，希望：

1、勾选某节点后，能同步勾选所有子节点，反之亦然。

2、清除某节点勾选状态后，其父节点的勾选状态也能同步被清除。



为此，可以设置 AfterCheckNode 事件代码为：

```
For Each nd As WinForm.TreeNode In e.node.Nodes  '同步子节点选中状态
    nd.Checked = e.node.Checked
Next
If e.node.Checked = False Then '如果本次操作为清除节点选中状态
    Dim pd As WinForm.TreeNode = e.node.ParentNode '获取父节点
    If pd IsNot Nothing Then
        pd.Checked = False '清除父节点选中状态
    End If
End If
```

上面的代码只能处理一级子节点，表现在：

1、如果我们勾选数码产品，那么电脑、电脑配件、相机会同步勾选，但是笔记本、服务器等二级子节点并不会同步勾选。

2、如果我们清除台式机的勾选状态，那么只能同步清除电脑的勾选状态，再上一级的数码产品的勾选状态则不会发生变化。

要实现跨层同步的目的，可以将代码改为：

```
For Each nd As WinForm.TreeNode In e.node.AllNodes   '同步子节点选中状态
    If nd.FullPath.StartsWith(e.node.FullPath) Then
        nd.Checked = e.node.Checked
    End If
Next
If e.node.Checked = False Then '如果本次操作为清除节点选中状态
    Dim pd As WinForm.TreeNode = e.node.ParentNode '获取父节点
    Do While pd IsNot Nothing
        pd.Checked = False '清除父节点选中状态
        pd   = pd.ParentNode '继续获取上一层级的父节点
    Loop
End If
```

注意这里使用的是AllNodes.

4.9.2.4.5.4.3.3.3 BeforeSelectNode

选择节点前执行。

e 参数属性：

Form:　触发事件的窗口

Sender:　触发事件的目录树

Node:　即将选择的节点

Cancel:　逻辑型，设为 True 取消选择该节点。

4.9.2.4.5.4.3.3.3 AfterExpandNode

展开节点后执行。

e 参数属性

Form:　触发事件的窗口

Sender:　触发事件的目录树

Node:　所展开的节点

参考：AfterCollapseNode

4.9.2.4.5.4.3.3.3 AfterCollapseNode

收缩节点后执行。

e 参数属性

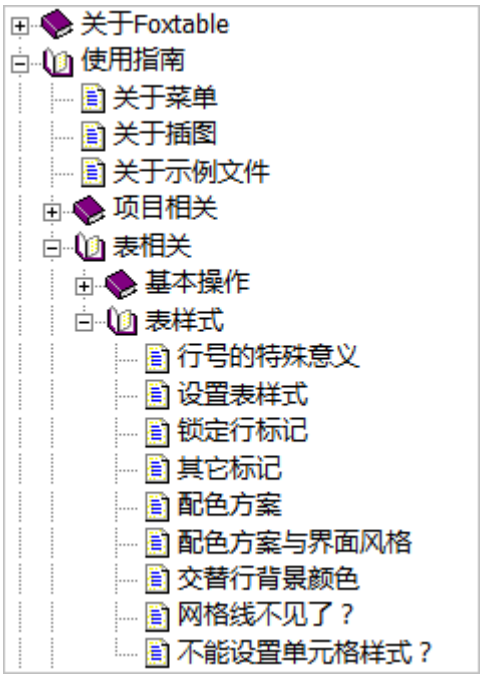
Form:　触发事件的窗口

Sender:　触发事件的目录树

Node:　所展开的节点

示例

假定希望如下图所示，目录节点在展开和收缩状态下，能显示不同的图标：



这里假定展开时的图标文件为"002.ico"，收缩时的图标文件为"001.ico"，注意图标文件要事先复制到项目的 Images 子目录中。

设计步骤：

1、在 AfterExpandNode 事件中设置代码：

```
e.Node.IconFile="002.ico"
```

2、在 AfterCollapseNode 事件中设置代码：

```
e.Node.IconFile="001.ico"
```

4.9.2.4.5.4.3.3.3 BeforeExpandNode

展开节点前执行。

e 参数属性：

Form: 触发事件的窗口

Sender: 触发事件的目录树

Node: 即将展开的节点

Cancel: 逻辑型，设为 True 取消展开该节点。

4.9.2.4.5.4.3.3.3 AfterEditNode

编辑节点标题后执行。

e 参数属性：

Form: 触发事件的窗口

Sender: 触发事件的目录树

Node: 所选择的节点

NewText：新的标题

Cancel： 逻辑型，设为 True 取消此次编辑

4.9.2.4.5.4.3.3.3 BeforeEditNode

编辑节点标题前执行。

e 参数属性：

Form: 触发事件的窗口
Sender: 触发事件的目录树
Node: 所选择的节点
Cancel： 逻辑型，设为 True 取消此次编辑

4.9.2.4.5.4.3.3.3 NodeMouseClick

用鼠标单击节点后执行。
e 参数属性
Form: 触发事件的窗口
Sender: 触发事件的目录树
Node: 单击的节点

4.9.2.4.5.4.3.3.3 NodeMouseDoubleClick

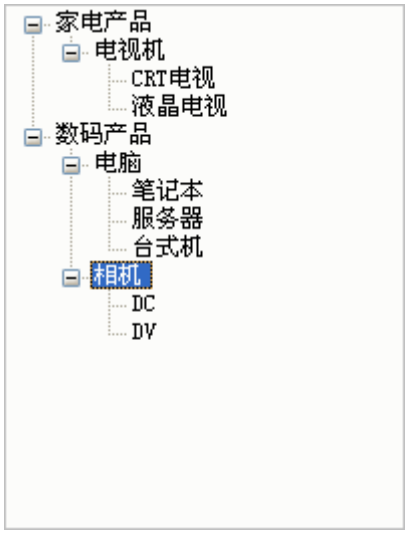
用鼠标双击节点后执行。
e 参数属性
Form: 触发事件的窗口
Sender: 触发事件的目录树
Node: 双击的节点

4.9.2.4.5.4.3.3.3 NodeMouseHover

鼠标停留在某个节点上时执行。
e 参数属性
Form: 触发事件的窗口
Sender: 触发事件的目录树
Node: 鼠标停留位置的节点

4.9.2.4.5.4.3.4 AllNodes

我们知道，目录树有 Nodes 属性，用于返回目录树的节点集合，节点自身也有 Nodes 属性，用于返回其子节点集合。
需要注意的是，Nodes 只包括一级子节点，不包括更深层级的子节点。
以下图为例，目录树的 Nodes 属性包括两个节点，分别是"家电产品"和"数码产品"，而"家电产品"这个节点的 Nodes 属性，则只包括一个节点"电视机"：



但是有的时候，我们需要处理的不仅仅是一级节点，例如所有层级的子节点，例如希望遍历所有层级的节点，选中符合特定条件的节点的复选框，利用 Nodes 属性是无法完成的。

所以除 Nodes 属性外，目录树和节点还有另一个名为 AllNodes 的属性，用于返回所有层级的节点的集合。

示例一

遍历所有节点，选中内容包含"上海"的节点的复选框：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
For Each nd As WinForm.TreeNode In trv.AllNodes
    If nd.Text.IndexOf("上海") >= 0 Then
        nd.Checked = True
    End If
Next
```

示例二

如果希望选中某个节点的复选框，其子节点能够同步选中，反之亦然。

为此只需将目录树的 AfterCheckNode 事件代码设置为：

```
For Each nd As WinForm.TreeNode In e.Node.AllNodes
    nd.Checked   = e.Node.Checked
Next
```

可以看出，AllNodes 属性可以大大提高目录树编码的灵活型。

示例三

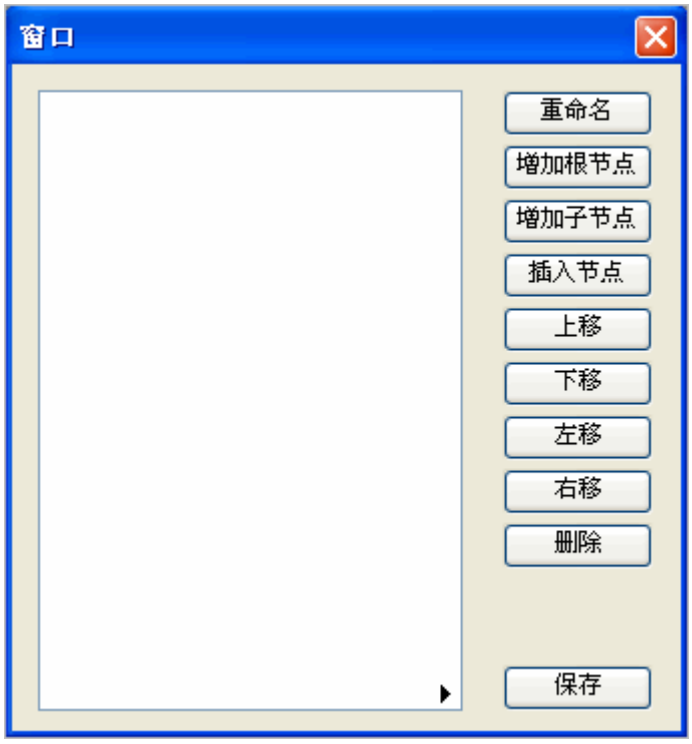
目录树显示复选框之后，默认是可以多选的，如果希望只能单选，可以在 AfterCheckNode 事件中加上代码：

```
If e.node.Checked Then
    For Each nd As WinForm.TreeNode   In e.Sender.AllNodes
        If nd.Checked   AndAlso nd.FullPath <> e.node.FullPath Then
            nd.Checked = False
        End If
```

```
    Next
End If
```

4.9.2.4.5.4.4 编辑目录树之一

本节内容可以参考 CaseStudy 目录下的文件：目录树管理.Table
假定并不想用表来存储目录结构，而是希望通过一个文本文件存储，并希望能够让修改并保存目录树结构。
实现这样的任务并不复杂，我们假定存储目录树结构的文件名“product.foxtr”,位于项目文件目录下。
设计的步骤为：
1、首先设计一个下图所示的窗口：



2、窗口的 AfterLoad 事件设置为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim fn As String = ProjectPath & "Product.foxtr"
If Filesys.FileExists(fn) Then
    tr.Load(fn)
End If
```

3、按下表所示设置各按钮的 Click 事件代码：

名称	代码
重命名	Dim tr As WinForm.TreeView tr = e.Form.Controls("TreeView1") tr.Select() tr.BeginEdit()
增加根节点	Dim tr As WinForm.TreeView Dim nd As WinForm.TreeNode

	tr = e.Form.Controls("TreeView1") nd = tr.Nodes.Add("新节点") tr.SelectedNode = nd tr.Select() tr.BeginEdit()

增加子节点	Dim tr As WinForm.TreeView
	Dim nd As WinForm.TreeNode
	tr = e.Form.Controls("TreeView1")
	nd = tr.SelectedNode
	If nd IsNot Nothing Then
	  nd = nd.Nodes.Add("新节点")
	  tr.SelectedNode = nd
	  tr.Select
	  tr.BeginEdit
	End If

插入节点	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
	Dim nd As WinForm.TreeNode = tr.SelectedNode
	Dim id As Integer = nd.index
	If nd.ParentNode Is Nothing Then
	  nd = tr.Nodes.Insert("新节点",id)
	Else
	  nd = nd.ParentNode.Nodes.Insert("新节点",id)
	End If
	tr.SelectedNode = nd
	tr.Select
	tr.BeginEdit

上移	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
	Dim nd As WinForm.TreeNode = tr.SelectedNode
	If nd IsNot Nothing Then
	  nd.MoveUp()
	End If
	tr.Select

下移	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
	Dim nd As WinForm.TreeNode = tr.SelectedNode
	If nd IsNot Nothing Then
	  nd.MoveDown()
	End If
	tr.Select

左移	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
	Dim nd As WinForm.TreeNode = tr.SelectedNode
	If nd IsNot Nothing Then
	  nd.MoveLeft()
	End If

	tr.Select

右移	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
	Dim nd As WinForm.TreeNode = tr.SelectedNode
	If nd IsNot Nothing Then
	  nd.MoveRight()
	End If
	tr.Select

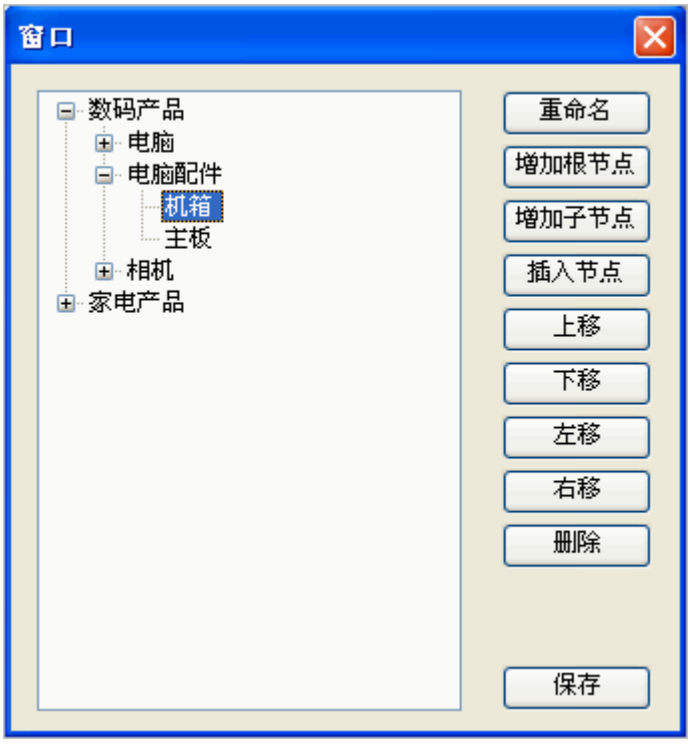
删除	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
	Dim nd As WinForm.TreeNode = tr.SelectedNode
	If nd IsNot Nothing Then
	  nd.Delete
	End If
	tr.Select

保存	Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1") Dim fn As String = ProjectPath & "Product.foxtr" tr.Save(fn)

4、实际上，重命名节点修改的只是节点标题，而不是节点名称，所以我们还需要在 AfterEditNode 事件中设置代码，使得节点名称和标题保持一致：

e.Node.Name = e.NewText

5、下图是设计效果，用户可以并保存修改目录树结构：

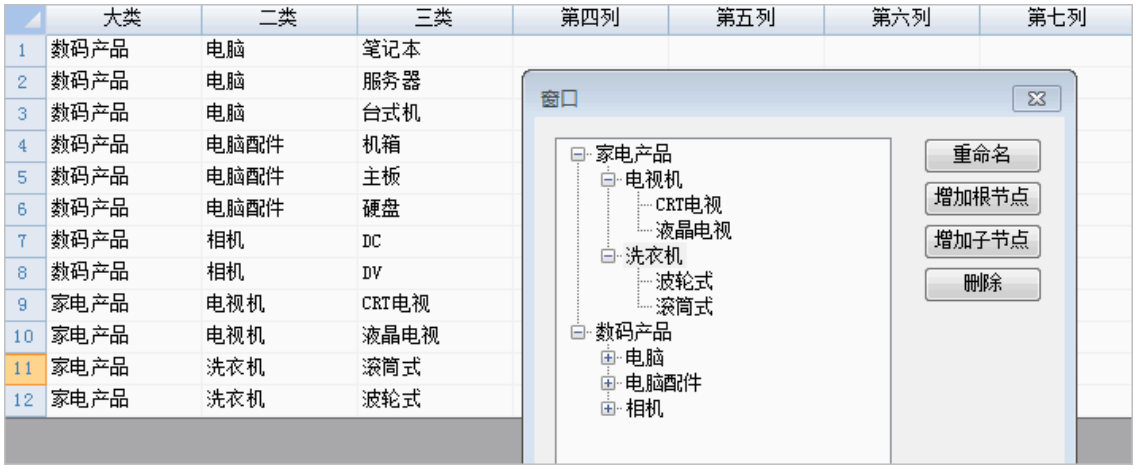


4.9.2.4.5.4.5 编辑目录树之二

本节的内容，可以参考 CaseStudy 目录下的文件：目录树与数据表.Table

本节的任务和上一节类似，不过目录树根据表生成，而且：

1、在目录树重命名节点，能够自动修改表中的相关数据。
2、在目录树删除节点，能够自动删除表中对应的行。
3、在目录树增加节点，能够自动在表中增加一行。



设计步骤：

1、首先设计一个上图所示的窗口。
2、窗口的 AfterLoad 事件代码为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
tr.BuildTree("表 A", "大类|二类|三类")
```

3、“重命名”按钮的 Click 事件代码：

```
Dim tr As WinForm.TreeView
tr = e.Form.Controls("TreeView1")
tr.Select()
tr.BeginEdit()
```

4、“增加根节点”按钮的 Click 事件代码：

```
Dim dr As DataRow = DataTables("表 A").AddNew
dr("大类") = "新节点"
Dim tr As WinForm.TreeView
Dim nd As WinForm.TreeNode
tr = e.Form.Controls("TreeView1")
nd = tr.Nodes.Add("新节点")
tr.SelectedNode = nd
tr.Select()
tr.BeginEdit()
```

5、“增加子节点”按钮的 Click 事件代码：

```
Dim tr As WinForm.TreeView
Dim nd As WinForm.TreeNode
tr = e.Form.Controls("TreeView1")
nd = tr.SelectedNode
If nd IsNot Nothing Then
    If nd.Level = 2 Then
            MessageBox.Show("最多允许三层节点!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
```

```
        Dim dr As DataRow = DataTables("表 A").AddNew
        If nd.Level = 0 Then
            dr("大类") = nd.Text
            dr("二类") = "新节点"
        ElseIf nd.Level = 1
            dr("大类") = nd.ParentNode.Text
            dr("二类") = nd.text
            dr("三类") = "新节点"
        End If
        nd = nd.Nodes.Add("新节点")
        tr.SelectedNode = nd
        tr.Select()
        tr.BeginEdit()
    End If
End If
```

6、“删除”按钮的 Click 事件代码：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim nd As WinForm.TreeNode = tr.SelectedNode
If nd IsNot Nothing Then
    Dim pth() As String = nd.FullPath.Split("\")
    Select Case nd.Level
    Case 0
        DataTables("表 A").DeleteFor("大类 = '" & pth(0) & "'")
    Case 1
        DataTables("表 A").DeleteFor("大类 = '" & pth(0) & "' And 二类 = '" & pth(1) & "'")
    Case 2
        DataTables("表 A").DeleteFor("大类 = '" & pth(0) & "' And 二类 = '" & pth(1) & "' And 三类 = '" & pth(2) & "'")
    End Select
    nd.Delete()
End If
tr.Select()
```

7、目录树 AfterEditNode 事件代码：

```
If e.NewText = "" Then
    e.Cancel = True
    Return
End If
Dim pth() As String = e.Node.FullPath.Split("\")
Select Case e.node.Level
    Case 0
        DataTables("表A").ReplaceFor("大类",e.NewText,"大类 = '" & pth(0) & "'")
    Case 1
        DataTables("表A").ReplaceFor("二类",e.NewText,"大类 = '" & pth(0) & "' And 二类 = '" & pth(1) & "'")
```

```
    Case 2
        DataTables("表A").ReplaceFor("三类",e.NewText,"大类 = '" & pth(0) & "' And 二类 = '" & pth(1) & "' And 三类 = '" & pth(2) & "'")
End Select
e.Node.Name = e.NewText
```

4.9.2.4.5.4.6 自行设计筛选树

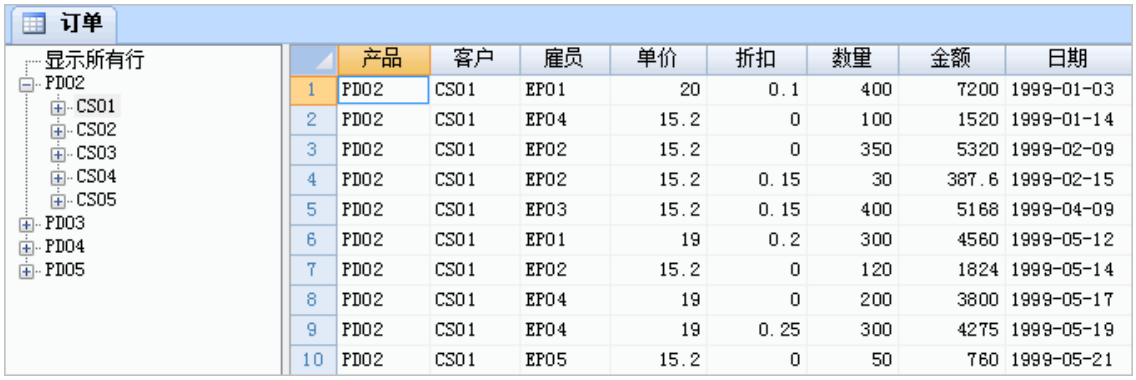
Foxtable 提供的筛选树，能够让我们快速便捷地筛选出所需要的数据。

本节我们将自己编码生成一个筛选树，这并非多此一举，毕竟有时我们会有一些特殊的要求，内置的筛选树不一定能满足我们的需要。

本节的示例文件为 CaseStudy 目录下的"目录树.Table"。

目的：

设计一个主窗口，在主表左边显示一个数据树，双击某个节点，自动筛选出对应的数据，如下图所示：



	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
2	PD02	CS01	EP04	15.2	0	100	1520	1999-01-14
3	PD02	CS01	EP02	15.2	0	350	5320	1999-02-09
4	PD02	CS01	EP02	15.2	0.15	30	387.6	1999-02-15
5	PD02	CS01	EP03	15.2	0.15	400	5168	1999-04-09
6	PD02	CS01	EP01	19	0.2	300	4560	1999-05-12
7	PD02	CS01	EP02	15.2	0	120	1824	1999-05-14
8	PD02	CS01	EP04	19	0	200	3800	1999-05-17
9	PD02	CS01	EP04	19	0.25	300	4275	1999-05-19
10	PD02	CS01	EP05	15.2	0	50	760	1999-05-21

设计步骤：

1、新建一个窗口，窗口类型设为主窗口，在窗口中插入一个分区面板。

2、在右分区插入一个 Table(表)，将其绑定到主表(订单表)，并将其停靠属性设置为 Fill。

3、在左分区插入一个 TreeView(目录树)，并将其停靠属性设置为 Fill。

4、将分区面板的停靠属性也设为 Fill。

5、将窗口的 AfterLoad 事件设为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("订单", "产品|客户|雇员")
trv.Nodes.Insert("显示所有行",0)
```

6、将目录树的 NodeMouseDoubleClick 事件设为：

```
Dim Filter As String
Dim dr As DataRow = e.Node.DataRow '获取生成此节点的行
If e.Node.Text <> "显示所有行" Then
    Select Case e.Node.Level
        Case 0
            Filter = "[产品] = '" & dr("产品") & "'"
        Case 1
            Filter = "[产品] = '" & dr("产品") & "' And [客户] = '" & dr("客户") & "'"
```

```
    Case 2
        Filter = "[产品] = '" & dr("产品") & "' And [客户] = '" & dr("客户") & "' And [雇员] = '" & dr("雇员") & "'"
    End Select
End If
Tables("订单").Filter = Filter
```

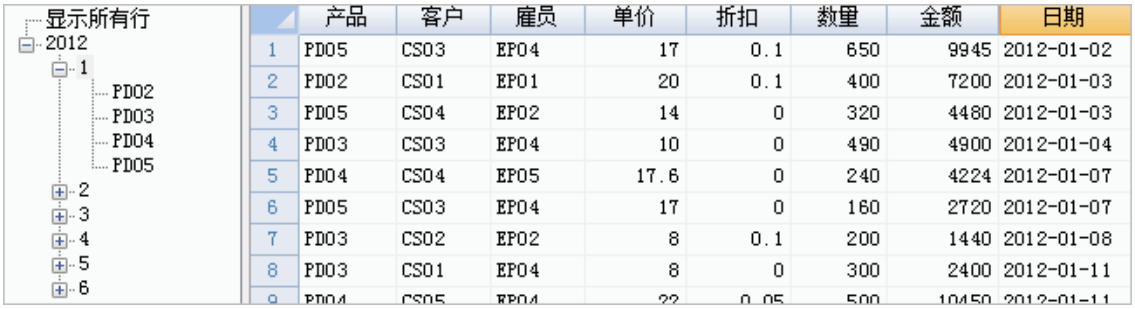
7、将窗口的“自动打开”属性设为 True。

总共不过十来行代码，即可完成这样一个看似复杂的系统，很简单对不对？

日期列与筛选树

下面的例子使用了 SQL 语句，如果你还没有接触过 SQL 语句，可以暂时忽略。

如果需要根据年月来筛选不同的产品：



	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	2012-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	2012-01-03
3	PD05	CS04	EP02	14	0	320	4480	2012-01-03
4	PD03	CS03	EP04	10	0	490	4900	2012-01-04
5	PD04	CS04	EP05	17.6	0	240	4224	2012-01-07
6	PD05	CS03	EP04	17	0	160	2720	2012-01-07
7	PD03	CS02	EP02	8	0.1	200	1440	2012-01-08
8	PD03	CS01	EP04	8	0	300	2400	2012-01-11
9	PD04	CS05	EP04	22	0.05	500	10450	2012-01-11

可以如下修改代码：

1、窗口的 AfterLoad 事件代码改为：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.CommandText = "Select Distinct Year(日期) As 年, Month(日期) As 月, 产品 From {订单}"
dt = cmd.ExecuteReader()
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt, "年|月|产品")
trv.Nodes.Insert("显示所有行",0)
```

上面的代码首先利用SQLCommand生成一个临时表，然后根据这个临时表生成目录树。

临时表包括年、月、产品三列，由于Select语句加上了Distinct关键词，所以提取出来的年、月、日是不重复的。

2、目录树的 NodeMouseDoubleClick 事件代码改为：

```
Dim Filter As String = ""
If e.node.Name <> "显示所有行" Then
    Dim d1 As Date
    Dim d2 As Date
    Dim Year As Integer = e.Node.DataRow("年")
    Dim Month As Integer = e.Node.DataRow("月")
    Dim Product As String = e.Node.DataRow("产品")
    Select Case e.Node.Level
        Case 0
```

```
        d1 = New Date(Year,1,1) '取得该年的第一天
        d2 = new Date(Year,12,31) '取得该年的最后一天
        Filter = "日期 >= #" & d1 & "# And 日期 <= #" & d2 & "#"
    Case 1
        d1 = New Date(Year, Month, 1) '取得该月的第一天
        d2 = new Date(Year, Month, Date.DaysInMonth(Year,Month)) '取得该月的最后一天
        Filter = "日期 >= #" & d1 & "# And 日期 <= #" & d2 & "#"
    Case 2
        d1 = New Date(Year, Month, 1) '取得该月的第一天
        d2 = new Date(Year, Month, Date.DaysInMonth(Year,Month)) '取得该月的最后一天
        Filter = "日期 >= #" & d1 & "# And 日期 <= #" & d2 & "# And 产品 = '" & Product & "'"
End Select
End If
Tables("订单").Filter = Filter
```

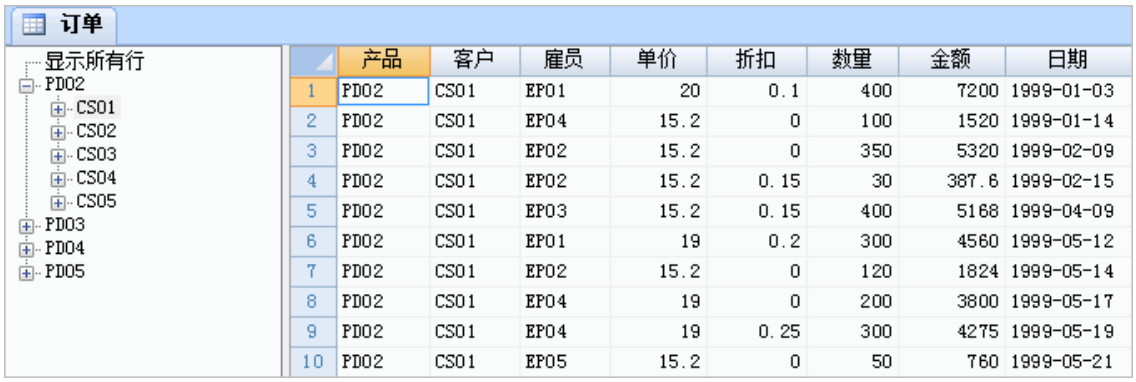
4.9.2.4.5.4.7 节点与行的关系

如果目录树是通过 BuildTree、CreateTree、GenerateTree 方法根据某数据表内容自动生成的，那么可以通过节点的 DataRow 属性，返回此节点在数据表中对应的行。

需要注意的是，一个节点可能对应有多个行，DataRow 属性将只返回其中一行。

示例一

我们在上一节已经讲述，要设计一个下图所示的筛选树：



	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD02	CS01	EP01	20	0.1	400	7200	1999-01-03
2	PD02	CS01	EP04	15.2	0	100	1520	1999-01-14
3	PD02	CS01	EP02	15.2	0	350	5320	1999-02-09
4	PD02	CS01	EP02	15.2	0.15	30	387.6	1999-02-15
5	PD02	CS01	EP03	15.2	0.15	400	5168	1999-04-09
6	PD02	CS01	EP01	19	0.2	300	4560	1999-05-12
7	PD02	CS01	EP02	15.2	0	120	1824	1999-05-14
8	PD02	CS01	EP04	19	0	200	3800	1999-05-17
9	PD02	CS01	EP04	19	0.25	300	4275	1999-05-19
10	PD02	CS01	EP05	15.2	0	50	760	1999-05-21

设计步骤为：

1、将窗口的 AfterLoad 事件设为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("订单", "产品|客户|雇员")
trv.Nodes.Insert("显示所有行",0)
```

2、将目录树的 NodeMouseDoubleClick 事件设为：

```
Dim Filter As String
Dim dr As DataRow = e.Node.DataRow '获取生成此节点的行
If e.Node.Text <> "显示所有行" Then
    Select Case e.Node.Level
        Case 0
            Filter = "[产品] = '" & dr("产品") & "'"
        Case 1
```

```
        Filter = "[产品] = '" & dr("产品") & "' And [客户] = '" & dr("客户") & "'"
    Case 2
        Filter = "[产品] = '" & dr("产品") & "' And [客户] = '" & dr("客户") & "' And [雇员] = '" & dr("雇员") & "'"
  End Select
End If
Tables("订单").Filter = Filter
```

如果不用DataRow属性，则需要直接拆分节点的FullPath，代码为：

```
Dim Filter As String = ""
Dim Value() As String
Value = e.Node.FullPath.Split("\")
If e.Node.Text <> "显示所有行" Then
  Select Case e.Node.Level
    Case 0
      Filter = "[产品] = '" & Value(0) & "'"
    Case 1
      Filter = "[产品] = '" & Value(0) & "' And [客户] = '" & Value(1) & "'"
    Case 2
      Filter = "[产品] = '" & Value(0) & "' And [客户] = '" & Value(1) & "' And [雇员] = '" & Value(2) & "'"
  End Select
End If
Tables("订单").Filter = Filter
```

两段代码的功能一样，但是第二种方法只能获取参与生成目录树的列的值，而第一种方法可以获取任意列的值，而且代码的可读性更好，所以帮助多数时候采用第一种方法。

示例二

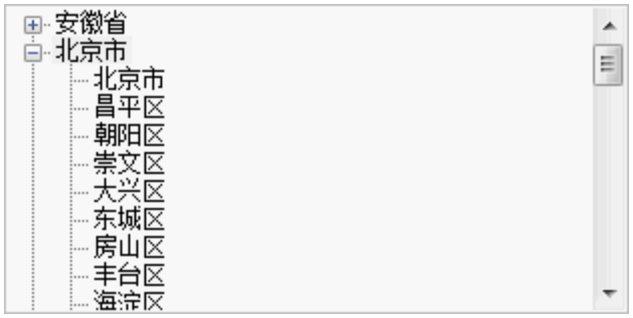
假定有下图所示的数据表：

	省	县市	区号	邮编
1	北京市	北京市	10	100000
2	北京市	东城区	10	100010
3	北京市	西城区	10	100030
4	北京市	崇文区	10	100060
5	北京市	宣武区	10	100050
6	北京市	朝阳区	10	100020

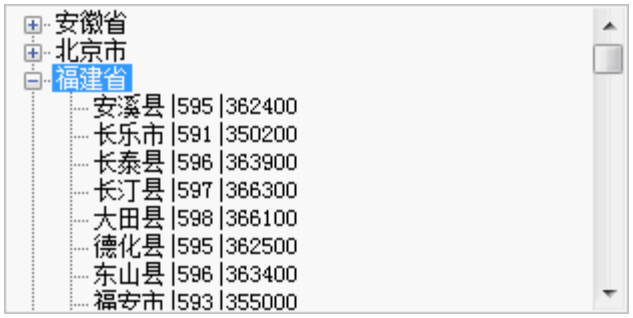
通过下面的代码：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省|县市")
```

生成的目录树是：



如果希望在目录树节点中同时显示出区号和邮编，如下图：



可以将代码改为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省|县市")
trv.StopRedraw '停止绘制目录树
For Each nd As WinForm.TreeNode In trv.AllNodes '遍历所有节点
    If nd.Level = 1 Then '如果是二级节点
        nd.Text = nd.text & "|" & nd.DataRow("区号") & "|" & nd.DataRow("邮编")
    End If
Next
trv.ResumeRedraw '恢复绘制目录树
```

示例三

如果节点和行是一一对应的，那么我们可以在表中快速定位到节点对应的行，例如只需将 NodeMouseDoubleClick 事件设置为：

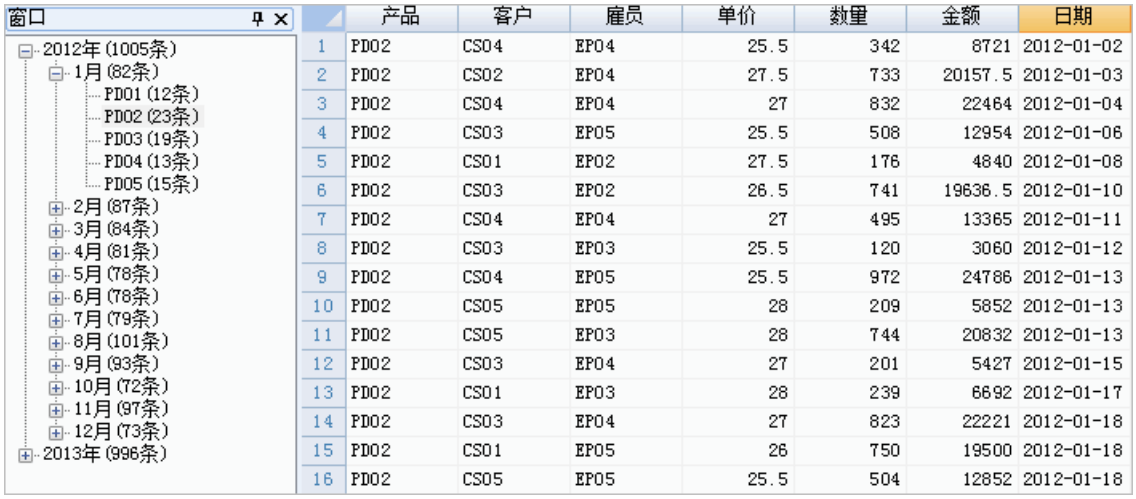
```
Tables("订单").Position = Tables("订单").FindRow(e.Node.DataRow)
```

这样双击节点，即可快速定位到节点所对应的行。
如果节点和行不是一对多的关系，那么也会定位到其中一个符合条件的行。

4.9.2.4.5.4.8 在筛选树显示统计数据

本节的示例，可以参考 CaseStudy 目录下的文件：统计筛选树.Table
假定我们要设计一个下图所示的筛选树，和普通的筛选树不同，这个筛选树 的年、月节点不是纯

数字的，会显示"年"、"月"字样，而且每个节点会显示该节点下的订单总数：



	产品	客户	雇员	单价	数量	金额	日期
1	PD02	CS04	EP04	25.5	342	8721	2012-01-02
2	PD02	CS02	EP04	27.5	733	20157.5	2012-01-03
3	PD02	CS04	EP04	27	832	22464	2012-01-04
4	PD02	CS03	EP05	25.5	508	12954	2012-01-06
5	PD02	CS01	EP02	27.5	176	4840	2012-01-08
6	PD02	CS03	EP02	26.5	741	19636.5	2012-01-10
7	PD02	CS04	EP04	27	495	13365	2012-01-11
8	PD02	CS03	EP03	25.5	120	3060	2012-01-12
9	PD02	CS04	EP05	25.5	972	24786	2012-01-13
10	PD02	CS05	EP05	28	209	5852	2012-01-13
11	PD02	CS05	EP03	28	744	20832	2012-01-13
12	PD02	CS03	EP04	27	201	5427	2012-01-15
13	PD02	CS01	EP03	28	239	6692	2012-01-17
14	PD02	CS03	EP04	27	823	22221	2012-01-18
15	PD02	CS01	EP05	26	750	19500	2012-01-18
16	PD02	CS05	EP05	25.5	504	12852	2012-01-18

设计步骤：

1、新建一个窗口，窗口类型设为主窗口，在窗口中插入一个分区面板。

2、在右分区插入一个 Table(表)，将其绑定到主表(订单表)，并将其停靠属性设置为 Fill。

3、在左分区插入一个 TreeView(目录树)，并将其停靠属性设置为 Fill。

4、将分区面板的停靠属性也设为 Fill。

5、将窗口的 AfterLoad 事件设为：

```
'首先生成统计表
Dim dt As DataTable
Dim b As New GroupTableBuilder("统计表1",DataTables("订单"))
b.Groups.AddDef("日期",DateGroupEnum.Year,"年")
b.Groups.AddDef("日期",DateGroupEnum.Month,"月")
b.Groups.AddDef("产品")
b.Totals.AddDef("客户",AggregateEnum.Count,"订单数") '根据客户列来统计记录数,也就是订单数
dt = b.Build(True) '参数设置为True,生成一个临时表,不在主界面显示
'然后根据统计表生成目录树
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt, "年|月|产品")
trv.StopRedraw
For Each nd As WinForm.TreeNode In trv.AllNodes
    Dim Year As Integer = nd.DataRow("年")
    Dim Month As Integer = nd.DataRow("月")
    Dim Product As String = nd.DataRow("产品")
    Select Case nd.Level
        Case 0
            nd.Text = nd.text & "年(" & dt.Compute("Sum(订单数)","年 = " & Year) & "条)"
        Case 1
            nd.Text = nd.text & "月(" & dt.Compute("Sum(订单数)","年 = " & Year & " And 月 = " & Month) & "条)"
        Case 2
            nd.Text = nd.text & "(" & nd.DataRow("订单数") & "条)"
```

```
    End Select
Next
trv.Nodes.Insert("显示所有行","显示所有行(" & dt.Compute("Sum(订单数)") & "条)",0)
trv.ResumeRedraw
```

上面的代码中，请留意第 8 行代码：

dt = b.Build(True) '参数设置为 True,生成一个临时表,不在主界面显示

加上参数 True 后，Build 生成的表，不会在主界面显示，当不再有地方引用这个表或这个表中的行和列时，这个表就会被自动销毁，参考不显示生成的表。

以本节的例子来说，由于根据这个表生成了目录树，目录树节点的 DataRow 属性会引用这个表中的行，所以直到关闭窗口，这个表才会被自动销毁。

6、将目录树的 NodeMouseDoubleClick 事件设为：

```
Dim Filter As String = ""
If e.node.Name <> "显示所有行" Then '要用Name属性,不能用Text属性
    Dim d1 As Date
    Dim d2 As Date
    Dim Year As Integer = e.Node.DataRow("年")
    Dim Month As Integer = e.Node.DataRow("月")
    Dim Product As String = e.Node.DataRow("产品")
    Select Case e.Node.Level
      Case 0
        d1 = New Date(Year,1,1) '取得该年的第一天
        d2 = new Date(Year,12,31) '取得该年的最后一天
        Filter = "日期 >= #" & d1 & "# And 日期 <= #" & d2 & "#"
      Case 1
        d1 = New Date(Year, Month, 1) '取得该月的第一天
        d2 = new Date(Year, Month, Date.DaysInMonth(Year,Month)) '取得该月的最后一天
        Filter = "日期 >= #" & d1 & "# And 日期 <= #" & d2 & "#"
      Case 2
        d1 = New Date(Year, Month, 1) '取得该月的第一天
        d2 = new Date(Year, Month, Date.DaysInMonth(Year,Month)) '取得该月的最后一天
        Filter = "日期 >= #" & d1 & "# And 日期 <= #" & d2 & "# And 产品 = '" & Product & "'"
    End Select
End If
Tables("订单").Filter = Filter
```

4.9.2.4.5.4.9 用目录树管理表

如果表比较多，那么通过表标题选择目标表可能会让人感到不够便捷。

你可以考虑采用目录树来分类管理表，单击某个节点即可快速选择某个表：



要实现这样的功能很简单，设计步骤：

1、新建一个窗口，窗口类型设置为共享，停靠区域设置为左。

2、在窗口增加一个 TreeView 控件，停靠属性设置为 Fill。

3、窗口的 AfterLoad 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim nd As WinForm.TreeNode
nd = Trv.Nodes.Add("分类 1","分类 1")
nd.Nodes.Add("表 A")
nd.Nodes.Add("表 B")
nd.Nodes.Add("表 C")
nd = Trv.Nodes.Add("分类 2","分类 2")
nd.Nodes.Add("表 D")
nd.Nodes.Add("表 E")
```

4、TreeView 控件的 NodeMouseClick 事件代码设置为：

```
If e.Node.Level = 1 Then
    MainTable = Tables(e.Node.Text)
End If
```

4.9.2.4.5.5 RecordGrid(记录窗口)

4.9.2.4.5.5.1 关于 RecordGrid

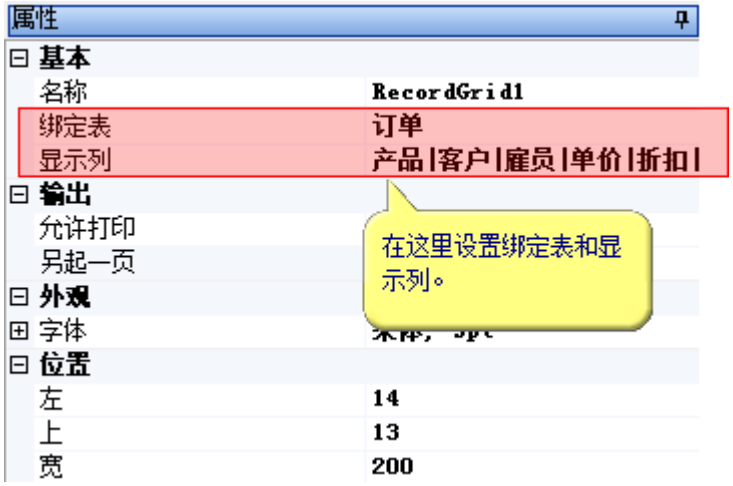
关于记录窗口，大家应该很熟悉了，因为在使用指南中已经讲述了，参考：记录窗口

RecordGrid 表示记录窗口，当然这并非是一个窗口，而是一个控件，有了 RecordGrid，可以在普通窗口中实现记录窗口的功能，一次显示某一记录的全部内容：



更重要的是，常规的记录窗口只对主表有效，而 RecordGrid 可以绑定到任何类型的表，包括关联表，以及 SQLTable、QueryTable 和副本 Table。

在窗口设计器中，可以选择 RecordGrid 要绑定的表和显示列：



如果一个表的列很多，你可以插入多个 RecordGrid 控件，全部绑定到这个表，然后给不同的 RecordGrid 控件设置不同的显示列。

4.9.2.4.5.5.2 动态绑定 RecordGrid

在设计窗口的时候，可以直接将 RecordGrid 绑定到主表和关联表。

如果要绑定到副本 Table、SQLTable 和 QueryTable，那么只能通过代码实现，例如：

```
Dim rgd As WinForm.RecordGrid = e.Form.Controls("RecordGrid1")
rgd.Table = Tables("窗口 1_Table1") '指定绑定表
rgd.Build() '重新生成记录窗口
```

上面的代码通常设置在窗口的 AfterLoad 事件中。

设置绑定表之后，还可以设置要显示的列，例如将 RecordGrid 绑定到员工表，并设置显示列：

```
Dim rgd As WinForm.RecordGrid = e.Form.Controls("RecordGrid1")
rgd.Table = Tables("员工")
rgd.Columns = "编号|姓名|部门|职务|性别|尊称|出生日期|地址|家庭电话|办公电话"
rgd.Build()
```

提示：必须先设置绑定表，再设置显示列，顺序不能错。

在绑定之前，也可以先进行一些细节设置：

```
Dim rgd As WinForm.RecordGrid = e.Form.Controls("RecordGrid1")
Tables("员工").Cols("邮政编码").ShortCaption = "邮编" ' 指定邮政编码列的缩写标题
Tables("员工").Cols("照片").ShowInRecordGrid = False '照片列不在记录窗口显示
Tables("员工").Cols("备注").RecordRowHeight = 3 '备注列的高度为 3 倍行高
rgd.Table = Tables("员工") '绑定到员工表
rgd.Build() '重新生成记录窗口
```

对于系统内置的记录窗口，也可以用同样的方法进行设置：

```
Tables("员工").Cols("邮政编码").ShortCaption = "邮编" ' 指定邮政编码列的缩写标题
Tables("员工").Cols("照片").ShowInRecordGrid = False '照片列不在记录窗口显示
Tables("员工").Cols("备注").RecordRowHeight = 3 '备注列的高度为3倍行高
Tables("员工").OpenRecordGrid()
```

4.9.2.4.5.5.3 属性

4.9.2.4.5.5.3.1 Table

返回或设置 RecordGrid 绑定的 Table，可以是任意类型的 Table。

示例

```
Dim rgd As WinForm.RecordGrid = e.Form.Controls("RecordGrid1")
rgd.Table = Tables("窗口 1_Table1") '指定绑定表
rgd.Build() '重新生成记录窗口
```

4.9.2.4.5.5.3.2 Columns

返回或设置 RecordGrid 显示的列。

示例

```
Dim rgd As WinForm.RecordGrid = e.Form.Controls("RecordGrid1")
rgd.Table = Tables("员工")
rgd.Columns = "编号|姓名|部门|职务|性别|尊称|出生日期|地址|家庭电话|办公电话"
rgd.Build()
```

提示：必须先设置绑定表，再设置显示列，顺序不能错。

4.9.2.4.5.5.3.3 DroppedDown

逻辑型，判断下拉列表或者下拉窗口是否已经打开。

4.9.2.4.5.5.4 方法

4.9.2.4.5.5.4.1 Build

重新生成记录窗口。
设置绑定表和显示列之后，必须立即执行此方法。
示例

```
Dim rgd As WinForm.RecordGrid = e.Form.Controls("RecordGrid1")
rgd.Table = Tables("员工")
rgd.Columns = "编号|姓名|部门|职务|性别|尊称|出生日期|地址|家庭电话|办公电话"
rgd.Build()
```

4.9.2.4.5.5.4.2 Refresh

强制刷新记录窗口中的数据。

4.9.2.4.5.5.4.3 OpenDropDown

打开下拉列表或者下拉窗口，只有正在编辑的列设置了列表项目或下拉窗口，此方法才有效。

4.9.2.4.5.5.4.4 CloseDropDown

关闭下拉列表或者下拉窗口。

4.9.2.4.5.5.5 记录窗口与表事件

不管是在表中编辑数据，还是在记录窗口编辑数据，都会触发以下和编辑相关的表事件：

PrepareEdit	准备编辑单元格的时候执行。
StartEdit	开始编辑单元格的时候执行。
ValidateEdit	结束编辑单元格前执行，通常用于验证输入的内容
AfterEdit	结束编辑单元格后执行，通常用于执行一些特定的操作和计算
CellButtonClick	单击窗口按钮后执行，通常用于实现一些复杂的输入功能
ChangeEdit	编辑框中内容发生变化时执行
KeyPressEdit	在编辑框每输入一个字符时执行
KeyDownEdit	在编辑框中按下某一按键时执行
KeyUpEdit	在编辑框中松开某一按键时执行。

那么在这些事件中，如何判断是在表中输入数据，还是在记录窗口输入数据呢？
所有这些事件的 e 参数都有一个属性：
RecordGrid：如果是在记录窗口编辑数据，则返回此记录窗口，如果是在表中输入数据，则返回

Nothing。

示例一

假定有个表，只允许在记录窗口输入数据，那么可以将 PrepareEdit 事件代码设置为：

```
If e.RecordGrid Is Nothing Then '如果不是在记录窗口输入数据
   e.Cancel = True '则取消输入
End If
```

示例二

假定不管是在录入窗口输入数据，还是在表中输入数据，只要正在编辑的是“第一列”，就自动打开下拉窗口：

```
If e.Col.Name = "第一列" Then
   If e.RecordGrid Is Nothing Then '如果是在表中输入数据
      If e.Col.DroppedDown = False Then
         e.Col.OpenDropDown()
      End If
   Else '如果是在记录窗口输入数据
      If e.RecordGrid .DroppedDown = False Then
         e.RecordGrid.OpenDropDown()
      End If
   End If
End If
```

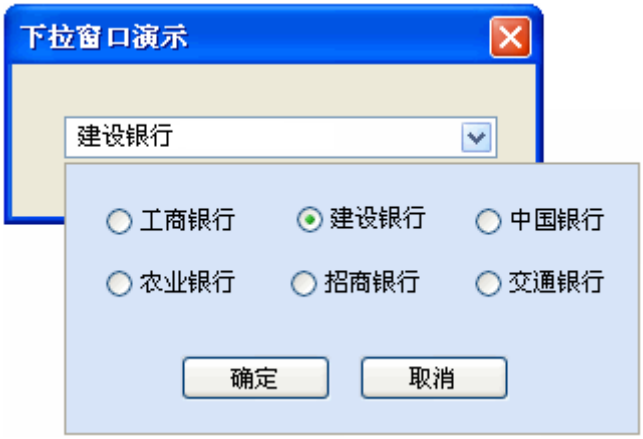
可以看出，记录窗口有自己独立的一套处理下拉列表和下拉窗口的方法，必须在相关事件中进行判断，以便采用合适的属性和方法。

4.9.2.4.5.6 DropDownBox(下拉组合框)

4.9.2.4.5.6.1 关于 DropDownBox

DropDownBox 是一个功能非常强大的控件，可以下拉显示一个窗口，我们可以基于此设计出一些复杂的录入界面，实际上我们前面接触的 DateTimePicker、NumericComboBox、ComboBox、CheckedComboBox 等控件都是在 DropDownBox 基础上设计出来的。

DropDownBox 既可以作为一个简单的控件独立使用，也可以配合一个下拉窗口使用:



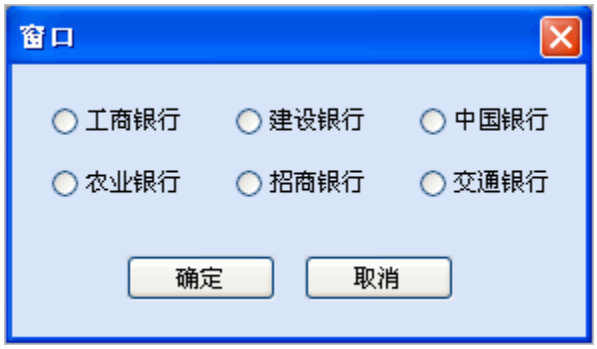
4.9.2.4.5.6.2 使用下拉窗口

我们了解一下如何设计一个带有下拉窗口的 DropDownBox。

本节的内容可以参考 CaseStudy 目录下的文件：下拉窗口演示.Table

一、设计下拉窗口

1、首先新建一个窗口，假定窗口名称为"窗口 1"，窗口布局如下：



窗口类型设置为"DropDownForm":



“确定”按钮的 Click 事件代码为：

```
For Each ctl As WinForm.Control In e.Form.Controls '遍历所有控件
    If TypeOf ctl Is WinForm.RadioButton Then '如果此控件是 RadionButton(单选框)
        Dim rdo As WinForm.RadioButton = ctl
        If rdo.Checked Then '如果已经选中
            e.Form.DropDownBox.Value = rdo.Text '将此单选框的文本赋值给下拉组合框
            e.Form.DropDownBox.CloseDropdown() '关闭下拉窗口
            Exit For
```

```
        End If
    End If
Next
```

“取消”按钮的 Click 事件代码为：

```
e.Form.DropDownBox.CloseDropDown() '关闭下拉窗口
```

窗口的 DropDownOpened 事件代码为：

```
Dim Val As String = e.Form.DropDownBox.Text
For Each ctl As WinForm.Control In e.Form.Controls '遍历所有控件
    If TypeOf ctl Is WinForm.RadioButton Then '如果此控件是 RadionButton(单选框)
        Dim rdo As WinForm.RadioButton = ctl
        If rdo.Text = Val Then '如果此单选框的文本等于下拉组合框的文本
            rdo.Checked = True '则选中此单选框
            Exit For
        End If
    End If
Next
```

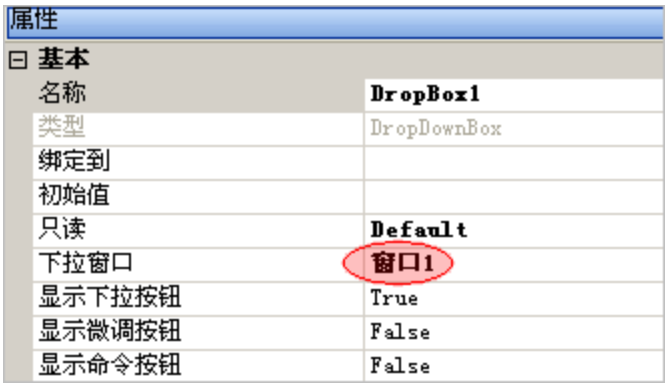
快速提示：

1、Form(窗口)的 DropDownBox 属性返回打开此窗口的下拉组合框，DropDownOpened 事件在显示此下拉窗口后执行。

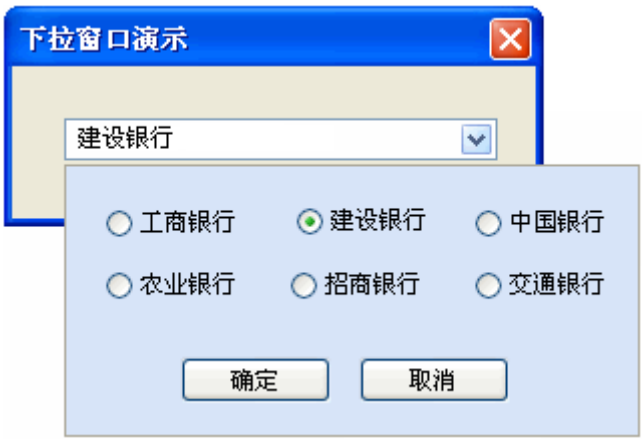
2、“确定”和“取消”两个按钮的代码只有作为下拉窗口打开时才能正常执行，否则会出错，这是因为只有此时 DropDownBox 才会返回打开此窗口的下拉组合框。

二、使用下拉窗口

2、新增一个窗口，插入一个下拉组合框(DropDowmBox)控件，将其“下拉窗口”属性设置为“窗口 1”：



现在打开此窗口，即可实现如下效果：



4.9.2.4.5.6.3 作为普通控件

DropDownBox 也可以作为普通控件使用。

由于文本、数值、日期都有自己专用的输入控件，分别是 TextBox、NumericBox 和 DataTimePicker，所以除非有特殊的需要，以致上述控件不能满足要求，才可以考虑将 DropDownBox 作为普通控件使用。

DropDownBox 除了文本输入框，还可以显示一些按钮：



上图从左到右依次为文本输入框、微调按钮、下拉按钮、命令按钮，这些按钮都有对应的属性决定其是否显示。

提示：

1、单击微调按钮、或者在键盘按上下箭头键，会触发 UpDownButtonClick 事件。

2、单击命令按钮，会触发 ModalButtonClick 事件。

示例一

假定希望定义一个文本输入框，文本框内显示一个命令按钮，单击此按钮可以打开另一个窗口。

首先在窗口插入一个 DropDownBox 控件，如下设置属性：

显示下拉按钮：False

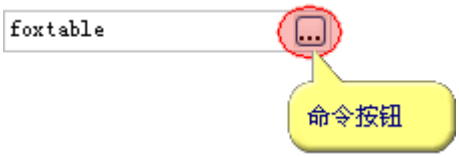
显示微调按钮：False

显示命令按钮：True

将 ModalButtonClick 事件设置为：

```
Forms("窗口 2").Open()
```

这样我们可以在 DropDownBox 中直接输入数据，也可以单击命令按钮打开窗口 2：



示例二

假定希望定义一个文本输入框， 在文本框显示微调按钮，单击微调按钮，或者按上下箭头按键，能够在“工商银行”、“建设银行”、“农业银行”、“中国银行”四个值之间来回切换。

首先在窗口插入一个 DropDownBox 控件，如下设置属性：

显示下拉按钮：False
显示微调按钮：True
显示命令按钮：False
将 UpDownButtonClick 事件代码设置为：

```
Dim drp As WinForm.DropDownBox = e.Sender
Dim vals As String() = {"工商银行","建设银行","农业银行","中国银行"}
Dim val As String = drp.value
Dim index As Integer = -1
For i As Integer = 0 To vals.Length - 1
    If val = vals(i) Then
        index = i
        Exit For
    End If
Next
index = index + 1
If index > 3 Then
    index = 0
End If
drp.Value = vals(Index)
```

这样我们可以在 DropDownBox 中直接输入数据，也可以单击微调按钮或者按箭头键选择不同的值。



4.9.2.4.5.6.4 类型参考

4.9.2.4.5.6.4.1 DropDownBox

4.9.2.4.5.6.4.1.1 与 TextBox 通用的成员

DropDownBox(组合框)本身也是一个 TextBox(文本框)，所以具备 TextBox(文本框)的全部属性、方法和事件。
这些通用的成员请参考 TextBox 中的介绍，这里就不再重复。
下面介绍 DropDownBox 特有的成员。

4.9.2.4.5.6.4.1.2 属性

4.9.2.4.5.6.4.1.2 DropFormName

字符型，用于设置下拉窗口的名称。
有了此属性，我们可以在运行的过程中视条件给 DropDownBox 指定不同的下拉窗口。

注意只有类型为“DropDownForm”的窗口，才能作为下拉窗口使用。

示例

```
e.Form.Controls("DropBox1").DropFormName= "窗口 1"
```

4.9.2.4.5.6.4.1.2 ShowDropDownButton

逻辑型，是否显示下拉按钮。

示例

```
e.Form.Controls("DropBox1").ShowDropDownButton = False
```

4.9.2.4.5.6.4.1.2 ShowUpDownButtons

逻辑型，是否显示微调按钮。

示例

```
e.Form.Controls("DropBox1").ShowUpDownButtons = True
```

4.9.2.4.5.6.4.1.2 ShowModalButton

逻辑型，是否显示命令按钮。

示例

```
e.Form.Controls("DropBox1").ShowModalButton = True
```

4.9.2.4.5.6.4.1.2 DroppedDown

逻辑型，判断 DropDownBox 是否已经打开下拉窗口。

4.9.2.4.5.6.4.1.2 AllowEditValue

逻辑型，是否允许用户直接输入内容。

4.9.2.4.5.6.4.1.3 方法

4.9.2.4.5.6.4.1.3 CloseDropdown

关闭已经显示的下拉窗口。

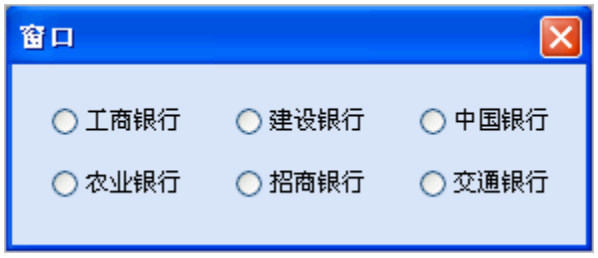
语法

```
ClosDropDown(Selected)
```

Selected: 可选参数，默认为 True，用于指定 DropDownClosed 事件的 e 参数 Selected 的值，一般不需要使用此参数。

示例

假定下拉窗口的布局为：



希望单击任何一个 RadionButton（单选框）即可将其标题复制给下拉组合框，并关闭下拉窗口。为此可以将此下拉窗口的全局事件 Click 的代码设置为：

```
With  e.Form.DropDownBox
        If  TypeOf  e.Sender  Is  WinForm.RadioButton  Then  '如果此控件是
RadionButton(单选框)
                .Value  =  e.Sender.Text
                .CloseDropDown()
        End  If
```

```
End  With
```

4.9.2.4.5.6.4.1.3 OpenDropDown

强制打开下拉窗口。

示例：

假定希望在 DropDownBox 中按下下箭头键，能够自动打开下拉窗口，可以将其 KeyDown 事件设置为：

```
If e.KeyCode = Keys.Down
    e.Sender.OpenDropDown()
    e.Cancel= True
End If
```

4.9.2.4.5.6.4.1.4 事件

4.9.2.4.5.6.4.1.4 UpDownButtonClick

单击微调按钮的时候执行。

e 参数属性:

Sender： 返回触发事件的控件

Form: 返回控件所在的窗口

Direction: 整数型，1 表示按的是向上的微调按钮，-1 表示按的是向下的微调按钮。

4.9.2.4.5.6.4.1.4 ModalButtonClick

单击命令按钮的时候执行。

e 参数属性:

Sender： 返回触发事件的控件

Form: 返回控件所在的窗口

4.9.2.4.5.6.4.2 下拉窗口

4.9.2.4.5.6.4.2.1 下拉窗口

在 Form(窗口)作为下拉窗口使用的时候，会有一些特殊的成员，我们接下来介绍这些成员。

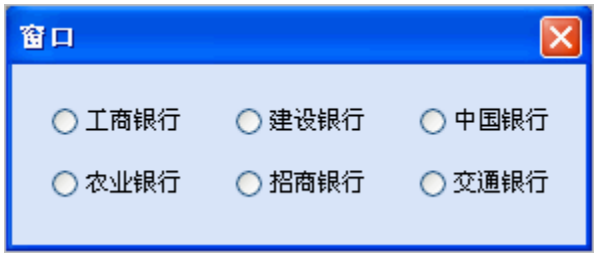
4.9.2.4.5.6.4.2.2 属性

4.9.2.4.5.6.4.2.2 DropDownBox

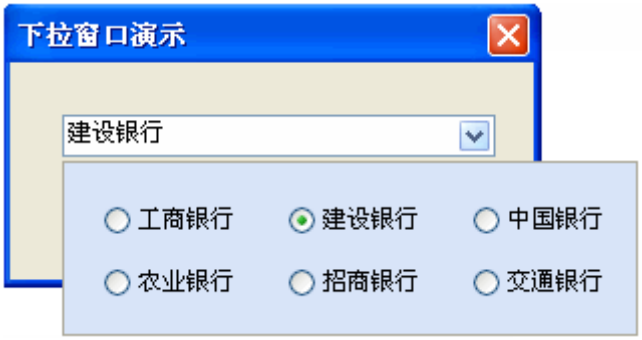
返回展开此下拉窗口的 DropDownBox。

示例

假定下拉窗口的布局为：



希望单击任何一个 RadionButton（单选框）即可将其标题复制给下拉组合框，并关闭下拉窗口:



为此可以此下拉窗口的全局表事件 CheckedChanged 代码设置为：

```
With e.Form.DropDownBox
    .Value = e.Sender.Text
    .CloseDropDown()
End With
```

本节的内容可以参考 CaseStudy 目录下的文件：下拉窗口演示.Table

4.9.2.4.5.6.4.2.2 DropTable

如果从表中的某列打开下拉窗口，可以通过 DropTable 返回对应的表(Table 类型)，通过 DropCol 返回对应的列(Col 类型)。

有了这两个属性，不同的表或不同的列，共用一个下拉窗口，会更加方便；对下拉窗口进行条件设置也会更加容易。

4.9.2.4.5.6.4.2.2 DropCol

参考 DropTable

4.9.2.4.5.6.4.2.3 事件

4.9.2.4.5.6.4.2.3 DropDownOpened

展开下拉窗口后执行。

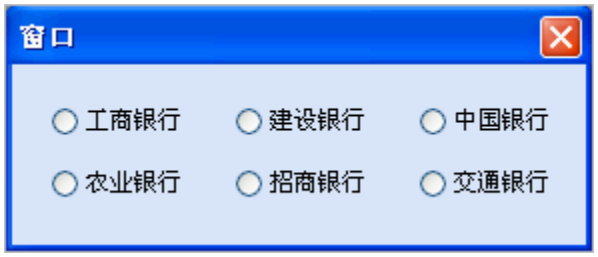
可以在这里对下拉窗口进行设置。

e 参数属性：

Form：表示触发事件的窗口。

示例

假定下拉窗口的布局如下：



希望在展开下拉窗口后，能够根据 DropDownBox 的内容自动选中对应的 RadionButton。
为此可以将下拉窗口的 DropDownOpened 事件的代码设置为：

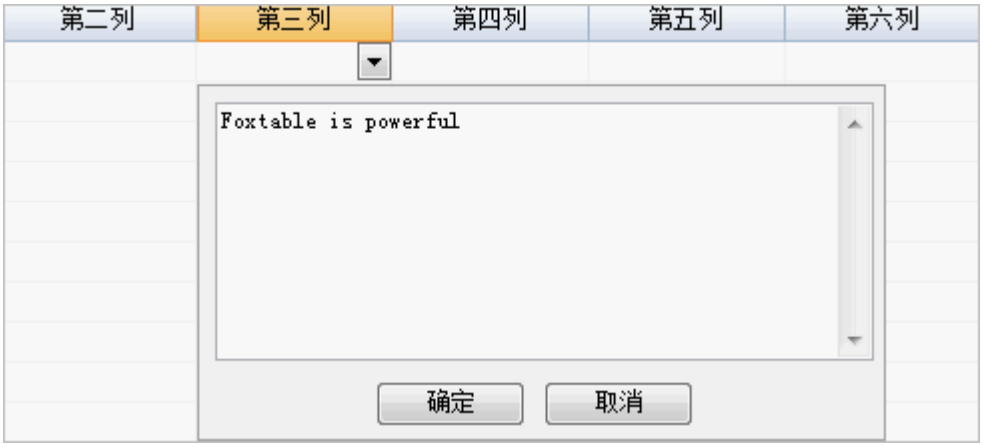
```
Dim Val As String = e.Form.DropDownBox.Text
For Each ctl As WinForm.Control In e.Form.Controls '遍历所有控件
    If TypeOf ctl Is WinForm.RadioButton Then '如果此控件是 RadionButton(单选框)
        Dim rdo As WinForm.RadioButton = ctl
        If rdo.Text = Val Then '如果此单选框的文本等于下拉组合框的文本
            rdo.Checked = True '则选中此单选框
            Exit For
        End If
    End If
Next
```

4.9.2.4.5.6.4.2.3 DropDownOpening

展开下拉窗口前执行。
可以在这里对下拉窗口进行设置。
e 参数属性：
Form：表示触发事件的窗口。
DropDownOpening 和 DropDownOpened 两个事件分别在展开窗口 的前后执行，都可以用来完成对下拉窗口的设置工作。
不过如果需要对下拉窗口的尺寸进行调整，或者要对其中的控件调整位置或大小，就应该使用 DropDownOpening 事件，以免展开下拉窗口的时候出现闪烁。

4.9.2.4.5.6.4.2.3 DropDownClosed

下拉窗口关闭后执行。
可以在这里对下拉窗口进行设置。
e 参数属性：
Form： 表示触发事件的窗口。
Selected：逻辑型，是否在下拉窗口选择了值。
说明：
是否在下拉窗口选择了值，并没有一个明确的标准，所以 Selected 参数通常返回 True，只有以下三种情况，系统才会认为用户没有在下拉窗口选择值，Selected 参数返回 False：
1、按 Esc 键关闭下拉窗口。
2、用鼠标单击其他控件或窗口空白区域，导致下拉窗口被关闭。
3、执行 CloseDropDown 方法关闭下拉窗口的时候，明确指定 Selected 参数为 False。
示例
下面用一个简单的例子来说明如何使用 DropDownClosed，任务是做一个下图所示的下拉输入框，不过采用的方法和前面有所不同：



设计步骤：

1、新建一个下拉窗口，窗口名为“窗口 1”，插入一个文本框和两个下拉按钮。

2、设置窗口的 DropDownOpened 事件代码为：

```
Dim tx As WinForm.TextBox = e.Form.Controls("TextBox1")
tx.Value = e.Form.DropDownBox.value
tx.Select()
```

3、设置窗口的 DropDownClosed 事件代码为：

```
If e.Selected Then
    Dim tx As WinForm.TextBox = e.Form.Controls("TextBox1")
    e.Form.DropDownBox.Value = tx.Value
End If
```

4、设置“确定”按钮的 Click 事件代码为：

```
e.Form.DropDownBox.CloseDropdown()
```

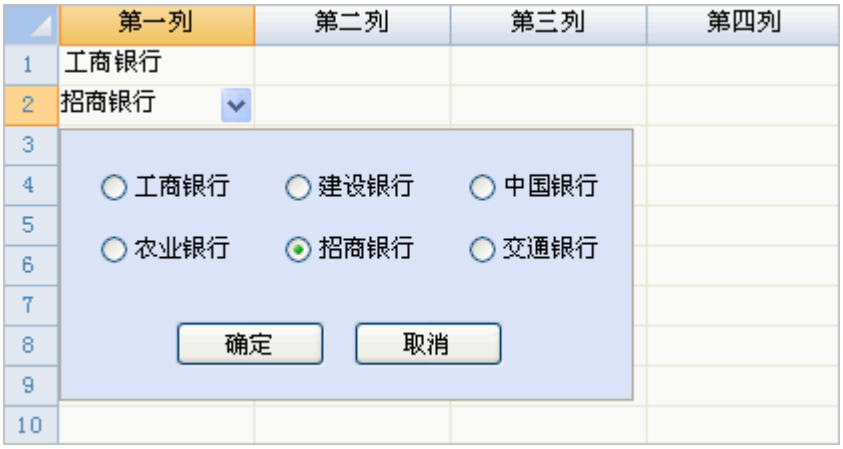
5、设置“取消”按钮的 Click 事件代码为：

```
e.Form.DropDownBox.CloseDropdown(False)
```

4.9.2.4.5.6.5 在表格中使用下拉窗口

```
Tables("表 A").Cols("第一列").DropForm = "窗口 1"
```

下图的第一列使用了下拉窗口来输入数据：



如果要取消使用下拉窗口输入数据，只需：

```
Tables("表 A").Cols("第一列").DropForm = ""
```

4.9.2.4.5.6.6 实战：自行设计下拉目录树

我们知道，在 Foxtable 只需经过简单的设置，即可在表中使用下拉目录树高效录入数据，无需编写任何代码。

我们也可以抛开内置的功能，自己设计一个下拉目录树来输入数据，不要以为这是多余的，因为：

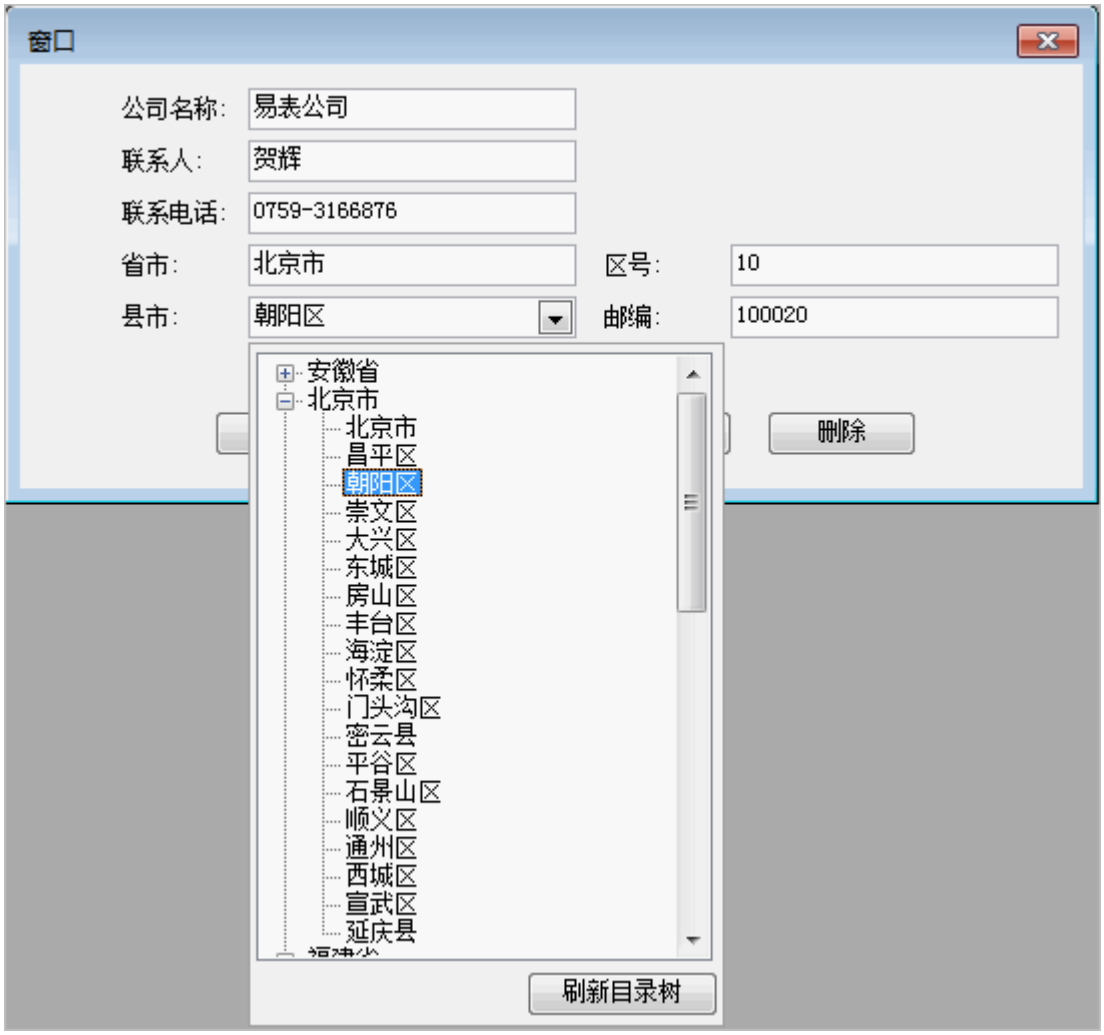
1、内置的下拉目录树只能在表中使用，不能在窗口使用。

2、内置的下拉目录树有固定的套路，一些特殊需求可能无法满足，例如本例的下拉目录树，增加了刷新功能。

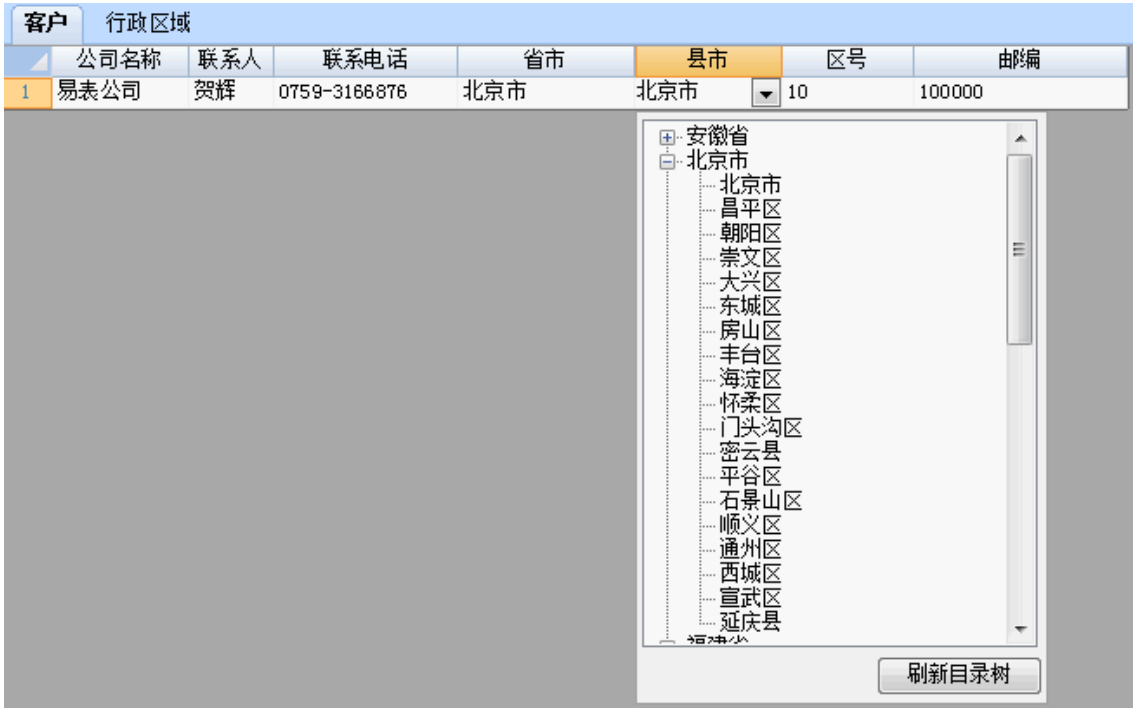
本示例请参考 CaseStudy 目录下的示例文件“下拉目录树.Table”。

此项目有两个表，分别为客户表和行政区域表，行政区域表为基础数据表，这个表已经输入了全国所有县级行政区域的资料，包括省市、市县、区号、邮编四列 。

我们希望在客户表设计一个下图所示的输入窗口，单击县市列这个文本框的下拉按钮，能出现一个下拉目录树，此目录树会列出所有的省市和县市，选择某个县市，会自动输入省市、县市、区号、邮编四列的内容：



同时希望在表中的县市列，也能通过下拉目录树来输入这四列的内容：



设计步骤

一、设计下拉窗口

1、新建一个窗口，名为“窗口 1”，窗口类型设置为“DropDownForm”。

2、窗口中插入一个 TreeView(目录树)控件。

3、窗口的 AfterLoad 事件设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市")
```

4、将刷新目录树按钮的代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市")
```

5、TreeView(目录树)的NodeMouseClick事件代码设置为：

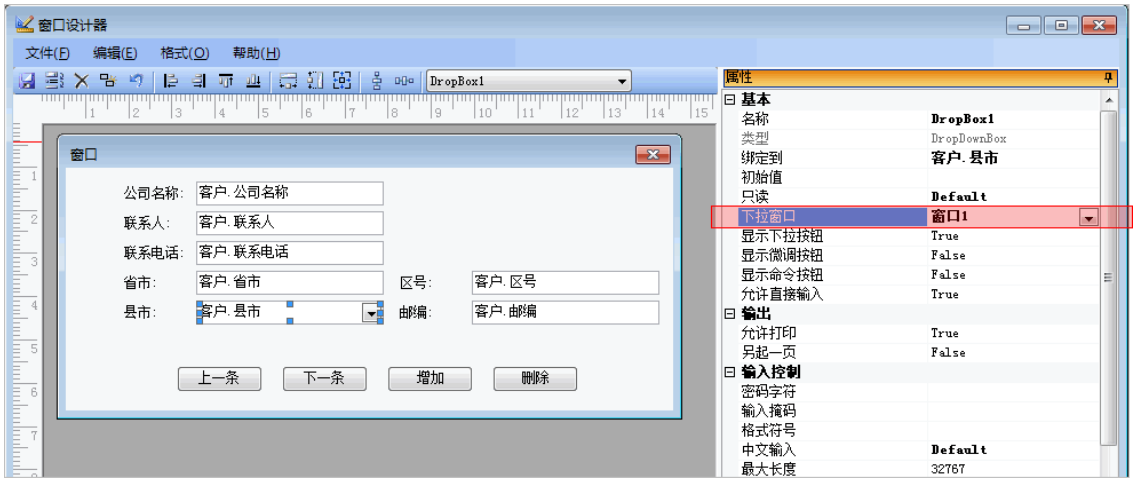
```
If e.Node.Level = 1 Then '如果单击的是第二层节点
    Dim tr As Row = Tables("客户").Current
    Dim dr As DataRow = e.Node.DataRow '获取生成此节点的DataRow
    tr("省市") = dr("省市")
    tr("县市") = dr("县市")
    tr("区号") = dr("区号")
    tr("邮编") = dr("邮编")
    e.Form.DropDownBox.Value = tr("县市") '这行不是多余的,省略此行,无法在表中正常使用此下拉窗口输入数据.
    e.Form.DropDownBox.CloseDropdown()
End If
```

二、设计录入窗口

1、新建一个窗口，名为“窗口2”，窗口类型为“独立”。

2、插入几个输入框，分别绑定到各字段，其中县市列输入框是DropDownBox，其余各列是TextBox。

2、选择县市输入框，将其下拉窗口设置为“窗口1”：



三、在项目事件AfterOpenProject加入代码：

```
Tables("客户").Cols("县市").DropForm = "窗口1" '客户表的县市列也采用我们设计的下拉窗口输入数据。
```

4.9.2.4.5.7 WebBrowser(网页浏览器)

4.9.2.4.5.7.1 关于 WebBrowser

WebBrowser 用于显示网页。

属性

- Address

  表示要显示的网页地址。
  地址必须以 http 开头，例如：http://www.foxtable.com/
  地址也可以是本地的一个网页文件。

方法

- GoBack
  后退一页。

- GoForward
  前进一页。

- GoHome
  回到首页，首页地址由计算机的 Internet 选项决定。

- Print
  打印显示的内容

- ShowSaveAsDialog
  显示另存为对话框。

- OfficeToolBar
  逻辑型，如果设置为 True，打开 Word 或 Excel 文件后，会显示菜单和工具栏。

示例一

在窗口中加一个 WebBrowser 和一个按钮，希望单击该按钮可以显示 Foxtable 主页，可以将按钮的 Click 事件代码设为：

```
Dim brw As WinForm.WebBrowser = e.Form.Controls("WebBrowser1")
brw.Address = "http://www.foxtable.com/"
```

示例二

在 WebBrowser 打开一个 Excel 文件，并显示 Excel 的菜单和工具栏：

```
Dim web As WinForm.WebBrowser = e.Form.Controls("WebBrowser1")
web.OfficeToolBar = True
web.Address = "c:\data\ddd.xls"
```

4.9.2.4.5.7.2 在窗口显示和打印 Excel 报表

示例一

在窗口中加入一个 WebBrowser 控件，窗口的 AfterLoad 事件代码设置为：

```
Dim wbr As WinForm.WebBrowser = e.Form.Controls("WebBrowser1")
Dim tmp As String = ProjectPath & "Attachments\资料卡.xls"
Dim rpt As String = ProjectPath & "Reports\资料卡.xls"
Dim Book As New XLS.Book(tmp)
Book.Build()
Book.Save(rpt)
wbr.OfficeToolBar = True '如果不需要显示工具栏，可删除此行代码
wbr.AddRess = rpt
```

这样一打开窗口，就会自动在窗口的 WebBrowser 控件显示生成的 Excel 报表。

重要提示：

1、如果执行上面代码的时候，没有在 WebBrowser 显示 Excel 报表，而是在一个新的 IE 窗口显示，那么请双击 Foxtable 目录下的 Office.reg 文件，将其添加到注册表中。

2、在 WebBrowser 显示的 Excel 菜单缺少包括打印命令的程序菜单，你可以按快捷键“Ctrl+P”打印，或者自定义 Excel 的快速访问栏，把打印命令加上去。

示例二

为了避免用户修改生成好的 Excel 报表，我们可以将报表保存为 PDF 文件，然后再显示在窗口中，例如：

```
Dim wbr As WinForm.WebBrowser = e.Form.Controls("WebBrowser1")
Dim tmp As String = ProjectPath & "Attachments\资料卡.xls"
Dim rpt As String = ProjectPath & "Reports\资料卡.pdf"
Dim Book As New XLS.Book(tmp)
Book.Build()
Book.SaveToPDF(rpt) '保存为 pdf 文件
wbr.AddRess = rpt
```

提示：为了能在 WebBrowser 显示 pdf 文件，请先安装 PDF 阅读软件，例如：Adobe Reader X

4.9.2.4.6 其它

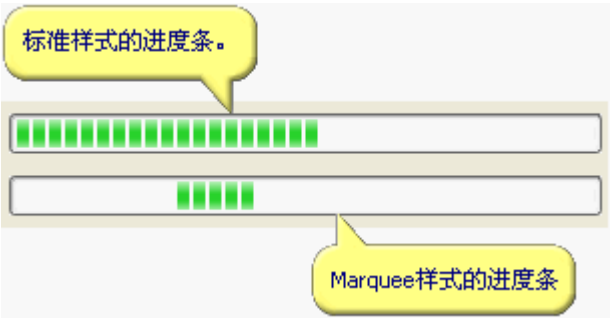
4.9.2.4.6.1 ProgressBar(进度条)

4.9.2.4.6.1.1 关于 ProgressBar

ProgressBar 表示进度条。

通过设置进度条的最小值、最大值和当前值，来表示某一任务的完成进度。

如果在设计窗口的时候，将进度条的样式属性设为“Marquee”,将显示一个连续滚动的方块，表示正在执行任务，类似于 Windows 启动时出现的滚动条。



示例

在表中增加一万行，在窗口中加入一个进度条和一个按钮，按钮的 Click 事件代码设为：

```
Dim p As WinForm.ProgressBar
p = e.Form.Controls("ProgressBar1")
p.Maximum = MainTable.Rows.Count '设置最大值
p.Minimum = 0 '设置最小值
```

```
p.Value = 0 '设置当前值
For i As integer = 0 To MainTable.Rows.Count - 1
    MainTable.Rows(i)("第一列") = i
    If i Mod 100 = 0 Then
        p.Value = i '当前值为已经完成的行数
    End If
Next
```

上面的代码中，每 100 行我们才更新一次进度条的进度，以避免频繁更新进度条而影响代码运行效率，这一点很多初学者容易忽略的。

4.9.2.4.6.1.2 属性

4.9.2.4.6.1.2.1 Maximum

整数型，返回或设置进度条的最大值。

4.9.2.4.6.1.2.2 Minimum

整数型，返回或设置进度条的最小值。

4.9.2.4.6.1.2.3 Value

整数型，返回或设置进度条的当前值。

4.9.2.4.6.2 PictureBox(图片框)

4.9.2.4.6.2.1 关于 PictureBox

PictureBox 表示图片框，通常用来显示一个指定的图片。
图片框提供了多种缩放模式：



PictureBox 提供了一组属性，用于设置文件来源位置：



上述属性在《使用指南》之《列相关》之《扩展列列类型》之《文件与图片的管理》中已经讲述，这里就不重复了。

需要注意的是，即使 PictureBox 绑定到某一列，也不会从这一列继承上述属性设置，PictureBox 需要单独设置上述属性，这样虽然有所不便，但也带来了更大的灵活性。

4.9.2.4.6.2.2 属性

4.9.2.4.6.2.2.1 Image

返回或设置图片框显示的图片。

可用 GetImage 函数从指定的图标文件中获得图标，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。
示例

```
Dim pbx AS WinForm.PictureBox
pbx = Forms("窗口 1").Controls("PictureBox1")
pbx.Image = GetImage("C:\Sample.bmp")
```

4.9.2.4.6.2.2.2 ImageFile

字符型，用于设置要现实的图片文件。
示例一

```
Dim pbx As WinForm.PictureBox = e.Form.Controls("PictureBox1")
pbx.ImageFile = "c:\data\玫瑰.png"
```

示例二

图片文件也可以来自web:

```
Dim pbx As WinForm.PictureBox = e.Form.Controls("PictureBox1")
pbx.ImageFile = "http://www.baidu.com/img/bdlogo.gif"
```

4.9.2.4.6.2.2.3 SizeMode

指定图片的缩放模式，ImageSizeMode 型枚举，可选值有：
Normal：标准
CenterImage：居中
StretchImage：拉伸
Zoom：等比拉伸
示例

```
Dim pbx AS WinForm.PictureBox
pbx = Forms("窗口 1").Controls("PictureBox1")
pbx.SizeMode = ImageSizeMode.Zoom
```

4.9.2.4.6.2.2.4 BorderStyle

设置图片框的边框类型，BorderStyle 型枚举，可选值有：
Fixed3D：　三维边框
FixedSingle：单行边框
None：　无边框
示例：

```
Forms("窗口 2").Controls("Panel1").BorderStyle = BorderStyle.Fixed3D
```

4.9.2.4.6.2.2.5 FTPclient

如果 PictureBox 用于远程图片显示，那么可以用此属性返回或设置其对应的 FTPClient。

示例

用代码动态设置 PictureBox 的 FTPClient：

```
Dim piv As WinForm.PictureBox = e.Form.Controls("PictureBox1")
Dim ftp As New FTPClient
ftp.host="192.168.128.143"
ftp.Account = "yinsha"
ftp.password = "128168909"
piv.FTPclient = ftp
```

4.9.2.4.6.2.2.6 Children

返回一个集合，包括所有顶层子控件。

4.9.2.4.6.2.3 事件

4.9.2.4.6.2.3.1 BeforeGetImage

在显示图片前执行。

e 参数属性:

Sender：返回触发事件的控件

Form：返回控件所在的窗口

File：字符型，返回或设置即将显示的图片文件名称。

利用此事件，可以灵活将数据和图片联动起来。

示例一

假定订单表中有一个名为“审核”的逻辑列，现在设计一个录入窗口，希望对于已经审核的订单，录入窗口能够自动显示一个审核印章：



设计步骤：

1、首先设计一个背景透明的印章图片，假定文件名为“已审核.png”。

2、将设计好的印章图片复制到项目文件目录下的 Images 子目录中。

3、在窗口中插入一个 PictureBox 控件，将其绑定到订单表的审核列，并将其背景颜色设置为 Transparent(透明)。

4、将 PictureBox 控件的 BeforeGetImage 事件代码设置为：

```
Dim pbx As WinForm.PictureBox = e.Sender
```

```
Dim r As Row = Tables("订单").Current
If r IsNot Nothing Then
    If r("审核") = True Then
        e.File = "已审核.png" '显示已审核印章图片
    Else
        e.File = "" '不显示图片
    End If
Else
     e.File = "" '不显示图片
End If
```

示例二

假定库存表中有一个到期日期列，希望在录入窗口中能够用图片提示已经到期和即将到期的产品。

设计步骤：

1、将设计好的图片复制到项目文件目录下的 Images 子目录中，假定文件名分别是“已过期.png”和“未过期.png"

2、在窗口中插入一个 PictureBox 控件，将其绑定到库存表的到期日期列，并将其背景颜色设置为 Transparent(透明)。

3、将 PictureBox 控件的 BeforeGetImage 事件代码设置为：

```
Dim pbx As WinForm.PictureBox = e.Sender
Dim r As Row = Tables("订单").Current
If r IsNot Nothing Then
    If r.IsNull("到期日期") = False Then
        Dim dt As Date = r("到期日期")
        Dim n As Integer = (dt - Date.Today).TotalDays
        If n <= 0   Then '如果已经过期
          e.File = "已过期.png"
        ElseIf n <= 10 Then '如果10天内到期
           e.File = "将过期.png"
        Else
           e.File = "" '不显示图片
        End If
    Else
        e.File = ""
    End If
Else
     e.File = "" '不显示图片
End If
```

4.9.2.4.6.2.4 作为容器

显然，PictrueBox 是用于显示图片的，不过 PictureBox 同样可以作为容器使用。

通常添加到 PictrueBox 中的子控件，其背景颜色应该设置为 Transparent(透明)，以避免遮挡住 PictrueBox 中原有的图片。

CaseStudy 目录下提供了两个示例文件，其中“透明叠加.Table”演示了如何实现图片之间以及图片和其它控件之间的透明叠加；另一个“中国地图.Table”则演示了如何在图片中加上隐形按钮。
隐形按钮一般用 Label 控件实现，将 Label 控件的自动大小设置为 False，鼠标形状设置为 Hand；将所有类似性质的 Label 控件的名称用约定的相同字符开始，并在名称中融合其代表的数据，例如在“中国地图.Table”这个文件的窗口中，名为“sh 黑龙江”、"sh 江苏"的 Label 控件，表示单击这两个 Label，分别显示黑龙江和江苏的数据，这样可以我们统一在窗口的全局 Click 事件中设置代码：

```
If TypeOf e.Sender Is WinForm.Label Then '如果单击的是标签
    If e.Sender.Name.ToLower.StartsWith("sh") Then '而且名称是以sh开头
        Dim sh As String = e.Sender.Name.SubString(2) '获取省份
        Dim ps As Integer = Tables("中国地图").FindRow("省份  = '" & sh & "'") '找出此省份所在行的位置
        If ps >= 0 Then '如果找到
            Tables("中国地图").Position = ps '定位到此行
        End If
    End If
End If
```

4.9.2.4.6.3 ColorLabel(颜色标签)

4.9.2.4.6.3.1 关于 ColorLabel

ColorLabel 表示颜色标签。
如果不会接触到颜色管理，那么 ColorLabel 是没有意义的，可以跳过去。
在运行的时候，双击颜色标签，会弹出一个颜色选择对话框，用于设置其颜色。

4.9.2.4.6.3.2 属性

4.9.2.4.6.3.2.1 Color

Color 类型，返回或设置颜色标签的颜色。
示例

```
Dim clb As WinForm.ColorLabel
clb = Forms("窗口 2").Controls("ColorLabel1")
clb.Color = Color.Red
```

4.9.2.4.6.3.2.2 Value

整数型，返回或者设置 ColorLabel 的颜色。
Value 和 Color 属性都表示 ColorLable 的颜色，不过前者用整数形式表示，后者直接用 Color(颜色)类型表示。
每一个颜色都可以转换为一个整数，具体请参考：关于颜色

4.9.2.4.6.3.3 事件

4.9.2.4.6.3.3.1 ValueChanged

颜色发生改变后执行。
e 参数属性：
e 参数属性：
Sender：返回触发事件的控件
Form:　返回控件所在的窗口

4.9.2.4.6.4 Calendar(月历)

4.9.2.4.6.4.1 Calendar

Calendar 是月历控件，可以显示一个或多个月：



通过 Value 属性可以设置或返回 Calendar 选择的日期。
提示：
1、在不同的操作系统下，Calendar 的外观和操作方式会有所不同。
2、在 Windows Vista/7 下，Calendar 的配色由操作系统决定，所以对于其颜色的设置是无效的。

4.9.2.4.6.4.2 属性

Calendar 的主要属性有：

- Value
  日期型，返回或设置 Calendar 选择的日期。

- ShowToday
  逻辑型，是否在月历的底部显示今天的日期。

- ShowTodayCircle
  逻辑型，是否在月历中用一个圆圈标出今天的日期。

- ShowWeekNumbers
  整数型，是否在月历的左边显示周数。

- XMonths
  整数型，设置月历水平方向显示的月数。

- YMonths
  整数型，设置月历垂直方向显示的月数。

- TitleBackColor
  Color 类型，设置月历标题区域的背景颜色。

- TitleForeColor
  Color 类型，设置月历标题的字体颜色。

- TrailingForeColor
  Color 类型，在月历中出现的非本月日期的字体颜色。

示例

```
Dim cal As WinForm.Calendar = e.Form.Controls("Calendar1")
cal.ShowWeekNumbers = True
cal.ShowToday = False
cal.YMonths = 2
cal.TitleBackColor = Color.Salmon
```

提示：在 Windows Vista/7 下，Calendar 的配色由操作系统决定，所以对于其颜色的设置是无效的。

4.9.2.4.6.4.3 事件

Calendar 的事件有：

- DateChanged
  选定日期发生变化后执行

- DateSelected
  用鼠标选定某个日期后执行

e 参数属性:
Sender：返回触发事件的控件

Form: 返回控件所在的窗口

两个事件很相似，DateChanged 在选定日期发生变化后执行，不管是单击上一月、下一月按钮，还是单击某个日期，都会触发，而 DateSelected 只有在用鼠标单击某个日期后才会触发。

示例一

例如希望单击某个日期，能够从订单表自动筛选出此日期的订单，可以将 DateSelected 事件设置为：

```
Tables("订单").Filter = "日期 = #" & e.Sender.Value & "#"
```

示例二

这次希望能够筛选选择的一个或多个日期的订单，如果单击某个日期，直接筛选此日期的订单，如果按住 Ctrl 键单击多个日期，则筛选多个日期的订单。

为实现此目的，可以如下设置 DateSelected 事件的代码，这也是一个很好的使用静态变量的例子：

```
Static lst As List(of Date) '定义一个静态集合变量,用于存储选择的日期
If lst Is Nothing Then
    lst = New List(of Date)
End If
If ModifierKey <> Keys.Control Then '如果没有按下 Ctrl 键
    lst.Clear() '清除以前的日期
End If
If lst.Contains(e.Sender.Value) = False Then '如果集合不包括按下的日期
    lst.Add(e.Sender.Value) '则向集合添加此日期
End If
If lst.Count = 1 Then '如果只选择了一个日期,则直接筛选此日期的订单
    Tables("订单").Filter = "日期 = #" & e.Sender.Value & "#"
Else '否则用 In 运算符筛选选定的多个日期的订单
    Dim v As String
    For i As Integer = 0 To lst.Count - 1
        v = v & "#" & lst(i) & "#,"
    Next
    v = v.Trim(",")
    Tables("订单").Filter = "日期 In (" & v & ")"
End If
```



4.9.2.4.6.5 Painter(绘图板)

4.9.2.4.6.5.1 关于 Painter

Painter 表示绘图板，我们可以通过代码在其中绘制图形。
Painter 采用 GDI+绘制图形，GDI+的内容很多，可以写成一本厚厚的书，我们不可能在这里详细地介绍 GDI+，但是会介绍其中最基础的部分。

属性

- Graphics

  返回一个 Graphics 对象，我们所有的图形绘都是通过这个对象来完成的。

- Image

  返回绘制好的图形。

方法

- Repaint

  显示绘制好的图形。

- Save

  保存绘制好的图形，语法：

  Save(FileName)

  FielName: 字符型，指定文件名。 系统会根据文件的扩展名选择合适的格式保存文件，例如文件名是“c:\data\001.gif”，系统会以 gif 格式保存文件，如果文件名是"c:\data\001.jpg"，则以 jpg 格式保存文件。

- Copy

  将绘制好的图形复制到剪贴板。

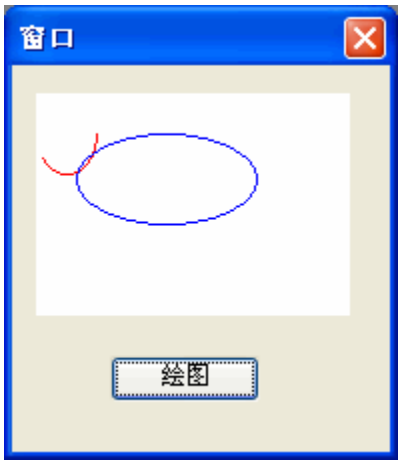
示例

在窗口中插入一个绘图板和一个按钮，将按钮的 Click 事件设为：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.DrawArc(Pens.Red, 0, 0, 30, 40, 0, 135)
g.DrawEllipse(Pens.Blue, 20, 20, 90, 45)
```

```
p.Repaint() '显示绘制好的图片
```

以后单击该按钮，即可绘制出下图所示的图片：



4.9.2.4.6.5.2 GDI+快速入门

GDI+的内容很多，可以写成一本厚厚的书，我们不可能在这里详细介绍 GDI+，只能介绍其中最基础的部分。

如果你能掌握 这些基础的内容，也足以应付大多数场合需要。

画笔(Pen)

绘图当然要用笔(Pen)来画，通过 Pens 可以返回各种颜色的默认画笔，例如：

```
Dim pn As Pen = Pens.Red '获得红色的默认画笔
```

默认画笔的宽度都是一个向度，如果要获得一个指定宽度的画笔，可以采用下面的语法：

```
Dim 变量名 As New Pen(颜色, 宽度)
```

例如：

```
Dim pn As New Pen(Color.Green, 5) '获得一个绿色的宽度为 5 的画笔
```

Pen 有一个 DashStyle 属性，用于定义通过此画笔绘制出的线条类型，该属性是一个 DashStyle 型枚举，包括以下可选值：

Dash 由线段构成的直线。

DashDot 由线段和点间隔构成的直线。

DashDotDot 由线段和连续两个点间隔构成的直线

Dot 由点构成的直线。

Solid 实线，这是默认值。

例如下面的代码定义一个红色的画笔，该画笔画出来的线条将是虚线，而不是实线：

```
Dim pn As New Pen(Color.Red)
pn.DashStyle = DashStyle.Dash
```

画刷(Brush)

画图除了要用 Pen(画笔)，还要用 Brush(画刷)。

通过 Brushes 可以返回各种颜色的默认画刷，例如：

```
Dim br As Brush = Brushes.Red '获得红色的画刷
```

可以用自定义的颜色来定义画刷，例如：

```
Dim br1 As Brush = New SolidBrush(Color.FromARGB(255,255,192,203))
Dim br2 As Brush = New SolidBrush(Color.FromARGB(-16181))
```

点(Point)

一个点(Point)表示一个位置，定义一个点的语法是：

Dim 变量名 As New Point(x, y)

X 表示点的水平坐标位置，y 表示点的垂直位置，单位为像素，坐标原点为左上角。

区域()

Graphics 成员

Grpahics 提供了很多方法用于绘制各种各样的图形和文字，我们下面逐一介绍这些方法。

在介绍的过程中，我们会提供一些示例代码，为了便于测试这些代码，请新建一个窗口，在窗口中分别加入一个绘图板(Painter)和一个按钮(Button)，将按钮的 Click 事件代码设置为示例代码。

DrawLine(pen,x1,y1,x2,y2)

绘制一条直线，参数 pen 指定画笔，x1 和 y1 指定直线的起始位置，x2 和 y2 指定直线的结束位置，单位为像素。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.DrawLine(Pens.Red,30,0,30,60)
g.DrawLine(Pens.Red,0,30,60,30)
p.Repaint()
```

绘制结果:



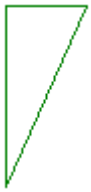
DrawLines(pen,Points)

绘制一个线条，将一系列的点连接起来，参数 Points 为一个数组，指定所有要连接的点。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
Dim points(3) As Point
points(0) = New Point(10, 10)
points(1) = New Point(10, 100)
points(2) = New Point(50, 10)
points(3) = New Point(10,10)
g.DrawLines(Pens.Green, points)
p.Repaint()
```

绘制结果：



DrawRectangle(pen,x,y,width,height)

绘制一个四边形，参数 x 和 y 指定四边形的起始位置，width 和 height 分别用于指定四边形的宽度和高度。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
Dim n As New Pen(Color.Red,5)
g.DrawRectangle(n,10,10,100,50)
p.Repaint()
```

绘制结果：



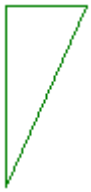
DrawPolygon(Pen,Points)

绘制一个多边型，参数 Points 为一个数组，指定多边形的各个顶点。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
Dim points(2) As Point
points(0) = New Point(10, 10)
points(1) = New Point(10, 100)
points(2) = New Point(50, 10)
g.DrawPolygon(Pens.Green, points)
p.Repaint()
```

绘制结果：



DrawEllipse(pen,x,y,width,height)

绘制一个椭圆，x 和 y 指定椭圆边框的左上角位置，width 和 height 指定椭圆边框的宽度和高度。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.DrawEllipse(Pens.Blue,10,10,100,50)
p.Repaint()
```

绘制结果：



DrawArc(pen,x,y,width,height,startAngle,sweepAngle)

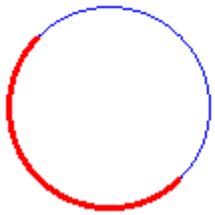
绘制一段弧型，弧形是椭圆周边的一部分，所以前五个参数和 DrawEllipse 方法一致，startAngle 指定弧形的起始角度，sweepAngle 指定该弧形所扫过的角度值，也就是幅度，方向为顺时钟。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
```

```
Dim g As Graphics = p.Graphics
Dim pn As New Pen(Color.Red,3)
g.DrawEllipse(Pens.Blue,10,10,100,100)
'绘制一个弧形,从 45 度角开始,幅度为 180 度
g.DrawArc(pn,10,10,100,100,45,180)
p.Repaint()
```

绘制结果：



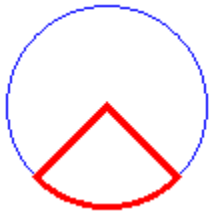
DrawPie(pen,x,y,width,height,startAngle,sweepAngle)

绘制一个扇形，扇形是椭圆的一部分，所以前五个参数和 DrawEllipse 方法一致，startAngle 指定扇形的起始角度，sweepAngle 指定该扇形所扫过的角度值，也就是幅度，方向为顺时钟。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
Dim pn As New Pen(Color.Red,3)
g.DrawEllipse(Pens.Blue,10,10,100,100)
'绘制一个扇形,从 45 度角开始,幅度为 90 度
g.DrawPie(pn,10,10,100,100,45,90)
p.Repaint()
```

绘制结果：



FillRectangle(Brush,x,y,width,height)

填充一个四边形，参数 Brush 指定填充所使用的画刷，x 和 y 指定四边形的起始位置，width 和 height 分别用于指定四边形的宽度和高度。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.FillRectangle(Brushes.Red,10,10,100,50)
p.Repaint()
```

绘制结果：



FillPolygon(Pen,Points)

填充一个多边型，参数 Points 为一个数组，指定多边形的各个定点。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
Dim points(2) As Point
points(0) = New Point(10, 10)
points(1) = New Point(10, 100)
points(2) = New Point(50, 10)
g.FillPolygon(Brushes.Green, points)
p.Repaint()
```

绘制结果：



FillEllipse(pen,x,y,width,height)

填充一个椭圆型，x 和 y 指定椭圆边框的左上角位置，width 和 height 指定椭圆边框的宽度和高度。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.FillEllipse(Brushes.Blue,10,10,100,60)
p.Repaint()
```

绘制结果：



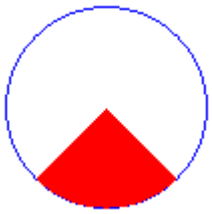
FillPie(pen,x,y,width,height,startAngle,sweepAngle)

填充一个扇形，扇形是椭圆的一部分，所以前五个参数和 FillEllipse 方法一致，startAngle 指定扇形的起始角度，sweepAngle 指定该扇形所扫过的角度值，也就是幅度，方向为顺时钟。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.DrawEllipse(Pens.Blue,10,10,100,100)
'填充一个扇形,从 45 度角开始,幅度为 90 度
g.FillPie(Brushes.Red,10,10,100,100,45,90)
p.Repaint()
```

绘制结果：



DrawString(s,font,brush,x,y)

在指定的位置，用指定的画刷和字体，绘制指定的文本。

参数 s 指定要绘制的文本，font 指定字体，brush 指定画刷，x 和 y 指定起始位置。

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
Dim fnt As New Font("宋体",16)
Dim msg As String = "I Like Foxtable"
g.DrawString(msg,fnt,Brushes.Red,10,10)
p.Repaint()
```

绘制结果：



DrawImage(Image,x,y,width,height)

在指定位置按指定大小绘制指定的图形。

参数 Image 为要绘制的图形(Image)，可用 GetImage 函数从指定的图形文件中获得图形，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

参数 x 和 y 指定图形的起始位置，width 和 height 指定大小。

其中 width 和 height 参数是可选的，如果省略，则按图片的原始大小绘制。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.DrawImage(getImage("c:\logo.jpg"),10,10)
p.Repaint()
```

绘制结果：



Clear(Color)

清除所有内容，并将背景色设为参数 Color 指定的颜色。

例如：

```
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.Clear(Color.White) '清除所有内容,并将背景颜色设为白色
p.Repaint()
```

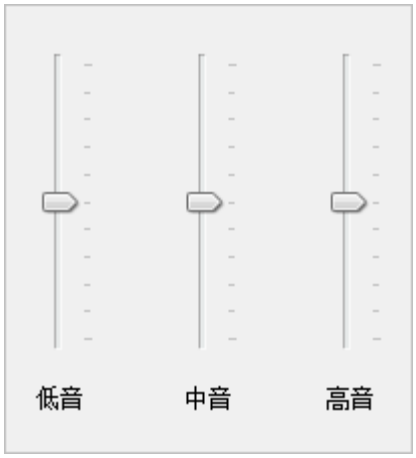
4.9.2.4.6.5.3 为 Excel 报表增加绘图功能

Excel 报表并没有绘图功能，但是有引用图片文件的功能。
如果我们将 Painter 绘制的图形保存为一个约定的文件，并在 Excel 报表中引用此约定的文件，那么就等于间接地为 Excel 报表加上了绘图功能。
示例
1、设计模版的时候，引用约定的图片文件"myimg.jpg"。
2、在窗口中加入一个 Painter 控件，一个 Button 控件。
3、Button 控件的代码设置为：

```
'绘制并保存图形
Dim p As WinForm.Painter = e.Form.Controls("Painter1")
Dim g As Graphics = p.Graphics
g.DrawEllipse(Pens.Blue,10,10,100,100)
g.FillPie(Brushes.Red,10,10,100,100,45,90)
p.Repaint()
p.Save(ProjectPath & "Images\myimg.jpg")
'生成 Excel 报表
Dim Book As New XLS.Book(ProjectPath & "Attachments\资料卡.xls")
Book.Build()
Book.Save(ProjectPath & "Reports\资料卡.xls")
```

4.9.2.4.6.6 TrackBar(滑动条)

TrackBar 表示滑动条，通常用于以可视化的形式调整数字属性设置，最常见的就是 Windows 的音量调节：



TrackBar 属性有：

- **Maximum**

  整数型，返回或设置滑动条的最大值。

- **Minimum**

整数型，返回或设置滑动条的最小值。

- **Value**

  整数型，返回或设置滑动条的当前值。

TrackBar 的事件有：

- **Scroll**

  在用户滚动滑块后，会触发 Scroll 事件，该事件 e 参数属性有：

  Sender：返回触发事件的控件。
  Form: 返回控件所在的窗口。

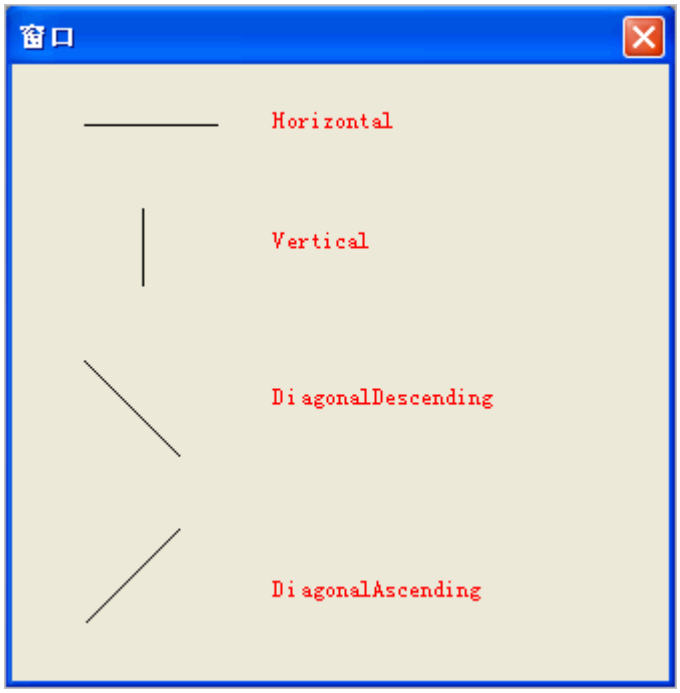
4.9.2.4.6.7 Line(线条)

Line 表示线条。
主要属性:

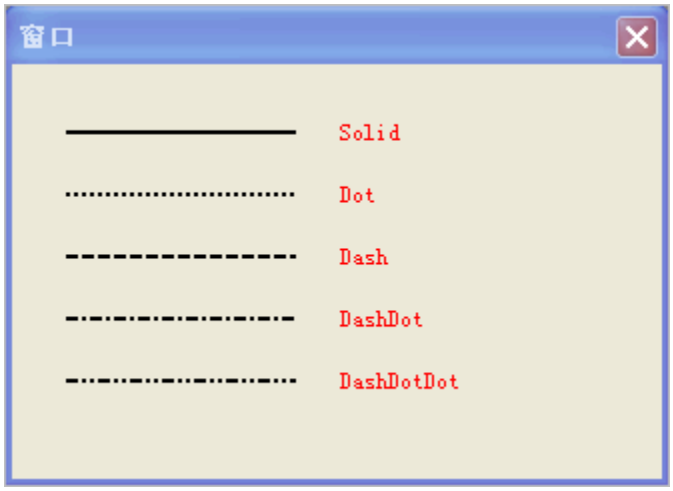
- **LineColor**
  返回或设置线条颜色

- **Thickness**
  返回或设置线宽

- **Ditection**
  返回或设置线条方向，LineDirectionEnum 型枚举，包括以下可选值。



- **DashStyle**
  返回或设置线条类型，DashStyleEnum 型枚举，包括以下可选值：

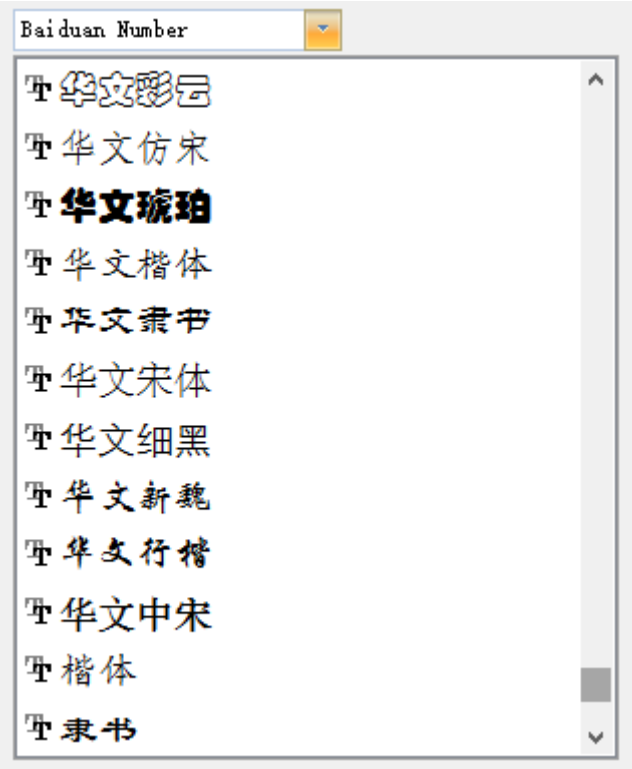


示例

```
Dim ln As WinForm.Line
ln = e.Form.Controls("Line1")
ln.SetBounds(0,0,200,200)
ln.ThickNess = 5
ln.DashStyle = DashStyleEnum.Dot
ln.Ditection = LineDirectionEnum.DiagonalDescending
ln.LineColor = Color.Red
```

4.9.2.4.6.8 FontPicker

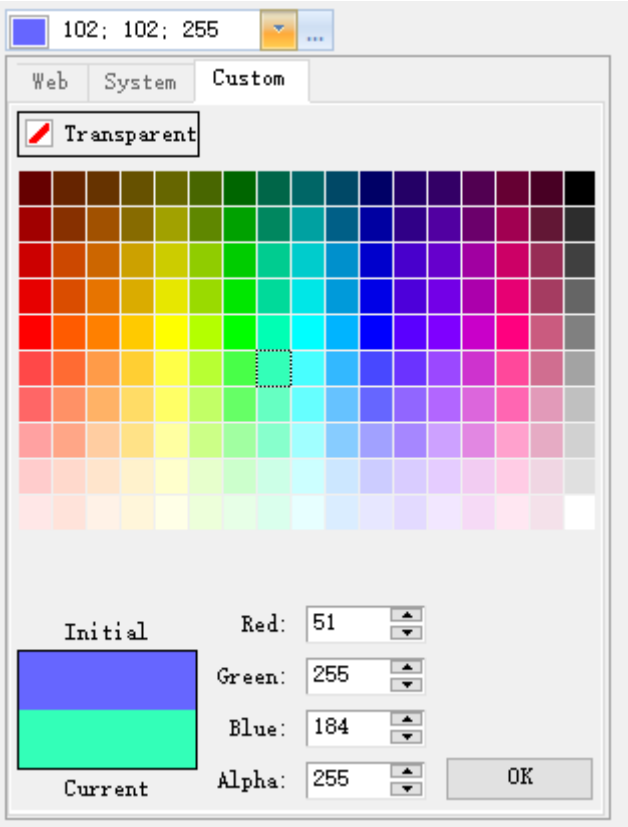
FontPicker 表示字体选择框，专门用于选择字体。



FontPicker 的 Value 属性，是一个字符型属性，用于返回或设置 FontPicker 选择的字体名称。通过 FontPicker 选择不同的字体后，会触发其 ValueChanged 事件。

4.9.2.4.6.9 ColorPicker

ColorPicker 表示颜色选择框，专门用于选择颜色：



ColorPicker 的 Value 属性，是一个 Color 型的属性，用于返回或设置 ColorPicker 选择的颜色。

通过 ColorPicker 选择不同的颜色后，会触发其 ValueChanged 属性。

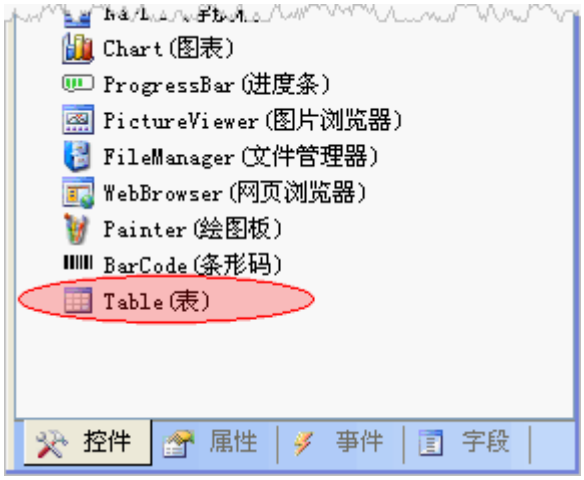
4.9.3 窗口中的 Table

4.9.3.1 Table 的类型

窗口可以插入三种类型的 Table，分别是：
Normal
SQLTable
SQLQuery
实际上是四种类型的 Table，因为 Normal 类型的 Table 还有副本和非副本之分。
下面分别介绍这四种类型的 Table。

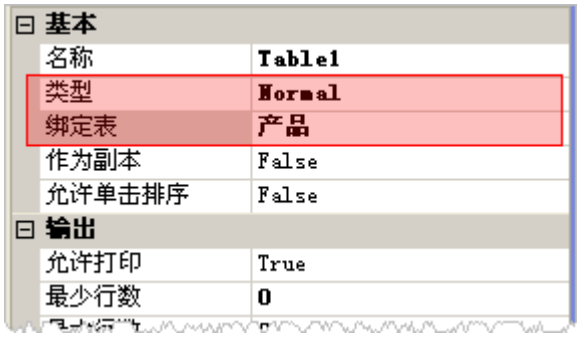
4.9.3.2 Normal 型 Table

在默认的情况下，Table 是显示在主界面中的。
你可以将 Table 移到窗口中显示，而且同一个窗口可以显示任意多个 Table。
将 Table 移到窗口的步骤非常简单：
1、在窗口中插入 Table 控件。



2、Table 类型选择 Normal。

3、选择要绑定的 Table。



显示窗口后，Table 将不在主界面中显示，转而在窗口显示，如果关闭窗口，Table 将重新显示在主界面中。

既然 Normal 型 Table 只是将原来的 Table 移到窗口中，所以包括名称等所有的一切都是不变的，例如对于订单表来说，不管是显示在主界面中，还是显示在窗口中，它始终都是 Tables("订单")，它还是它，没有任何改变，只是出现在不同的地方而已。

需要注意的是，此时的 Table 控件并没有事件，道理很简单，因为这个 Table 虽然换了个位置显示，但还是原来的 Table，之前按常规方法在表属性窗口中设置的事件代码继续有效，无需重新设置。

不过也只有这种 Table 控件是没有事件的，接下来介绍的几种 Table 控件，都会有自己独立的整套或部分事件。

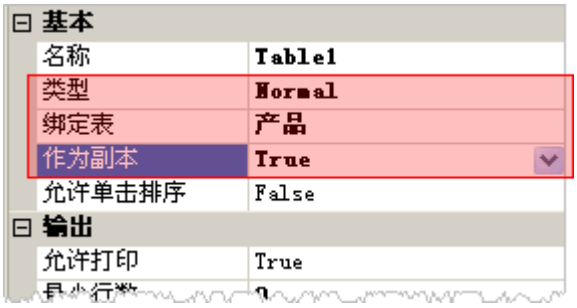
4.9.3.3 作为副本的 Normal 型 Table

前面提到，在窗口插入 Normal 型的 Table 控件，可以将 Table 从主界面移到窗口中。

在默认的情况下，Normal 型的 Table 是不能“分身”的，在一个地方出现，就会在另一个地方消失，无法同时出现在主界面或多个窗口中。

基本概述

如果将 Normal 型 Table 的“作为副本”属性设为 True，将不影响原 Table，而是创建一个原 Table 的副本插入到窗口中，这样同一个 Table 可以同时出现在主界面或多个窗口中，而且可以各自进行独立的操作，互不影响。



既然副本 Table 是独立的，所以它必须有自己的名称，这样才能和原 Table 区分开来，其名称为 ：窗口名称_控件名称

假定副本 Table 控件的名称为“Table1”，所在窗口的名称为“窗口 1”，那么不管其绑定到哪一个表，它的名称都是“窗口 1_Table1”，我们可以通过这个名称来引用它 ，例如：

```
Tables("窗口 1_Table1").Filter = "产品  = 'PD01'"
```

假定此副本 Table 控件绑定到订单表，那么 Tables("窗口 1_Table1")的数据就来自于 DataTables("订单")，此时 DataTables("订单")对应有两个 Table， 分别是作为副本的 Tables("窗口 1_Table1")和作为正本的 Tables("订单")。

你可以打开这个窗口，然后在命令窗口执行：

```
MessageBox.Show(Tables("窗口 1_Table1").DataTable.Name)
MessageBox.Show(Tables("订单").DataTable.Name)
```

可以看到两次显示的都是“订单”，说明 Tables("窗口 1_Table1")和 Tables("订单")对应的 DataTable 都是 DataTables("订单")。

副本和正本是基于同一个 DataTable 但各自独立的 Table，可以分别进行操作，例如在窗口的 AfterLoad 事件加上代码：

```
Tables("窗口 1_Table1").Filter = "日期  = #" & Date.Today & "#"
```

这样打开窗口后，副本 Table 显示的是今天的订单。

这个副本 Table 在打开窗口的时候自动生成，在关闭窗口后自动删除，所以如果窗口 1 已经关闭，那么执行上述代码是会出错的，因为此时并不存在名为"窗口 1_Table1"的 Table。

共用事件

首先，在表属性中设置的表事件，对于副本 Table 一样有效。

例如我们在订单表的 PrePareEdit 事件设置代码：

```
e.Cancel = True
```

那么不管是正本的 Tables("订单")，还是副本的 Tables("窗口 1_Table1")，都将是不可编辑的。

其次， 我们可以在表事件中，通过判断触发事件的 Table 的名称，可以知道是正本还是副本触发了此事件，这样就可以实现差异化处理。

例如在订单表的 PrePareEdit 事件设置代码：

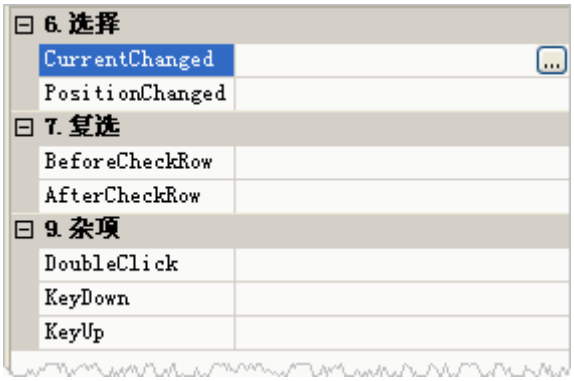
```
Select Case e.Table.Name
    Case "订单" '主表
      e.Cancel = True
    Case "产品.订单" '关联表
      e.Cancel = False
    Case "窗口 1_Table1" '副本 Table
```

```
    If User.Name <> "张经理" Then
        e.Cancel = True
    End If
End Select
```

这样 Tables("订单")是不能编辑的，关联表 Tables("产品.订单")是可以直接编辑的，而副本 Tables("窗口 1_Table1")只能张经理编辑。

独立事件

下图中的表事件有些特殊，设计窗口时，你可以为副本 Table 中单独设置这些表事件的代码：



如果在副本 Table 没有设置这些表事件，那么在原 Table 中设置的这些表事件对副本 Table 一样有效；如果你希望这些事件触发时，副本 Table 和原 Table 有不一样的行为，那么就可以给副本 Table 单独设置这些表事件代码。

副本 Table 能够将这些常用表事件独立出来设置，会给大家的开发工作带来很多便利。

例如希望在副本 Table 中双击某行后打开一个窗口，可以将副本 Table 的 DoubleClick 事件代码设置为：

```
e.Cancel = True '禁止进入编辑状态
Forms("窗口名").Open()
```

这样在副本中双击某行即可打开指定的窗口，在正本中双击某行则不会动作。

什么时候使用副本

既然 Normal 型的 Table 有副本和非副本(姑且称为正本)之分，那么应该什么时候使用正本，什么时候使用副本呢？

很简单，如果一个表，要同时在多个窗口出现，那么只有使用副本才能实现；如果一个表虽然在多个窗口出现，但不是同时出现，那么全部使用正本就行。

例如一个表，要同时在两个窗口出现，那么一个窗口使用正本，一个窗口使用副本；如果要在三个窗口同时出现，那么一个窗口使用正本，另外两个窗口使用副本即可。

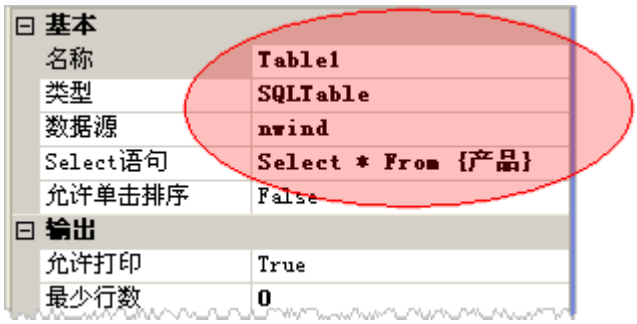
例如一个表，虽然要在三个不同的窗口出现，但是这些窗口永远不会同时打开，那么全部使用正本就行。

提示：只有主表才可以有副本，关联表不可以，如果将一个 Table 控件绑定到关联表，且作为副本属性设置为 True，创建的仍然是主表的副本。

4.9.3.4 SQLTable 型 Table

如果将窗口中 Table 控件的类型设为 SQLTable，那么就可以用 SQl 语句从指定数据源中提取数据

生成 Table。
有关 SQL 语句，后面会有专门的一章讲述。
例如按照下图所示设置 Table 控件的属性：



即可从外部数据源 nwind 中提取产品数据，显示在 Table 中。
需要注意的是，SQLTable 的 Select 语句只能取一个表的数据，否则出错。
可以不指定数据源，此时从内部数据源中取数据。
需要特别注意的是,如果只想取表的部分列生成 SQLTable，那么 Select 语句必须包括主键列在内，例如：

```
Select [_Identify], [编号], [名称], [单价], [库存] From {产品}
```

这里假定产品表是内部数据表，内部数据表的主键列是_Identify，如果是外部表，请使用实际的主键列名称。
SQLTable 不仅会生成一个 Table，还会生成一个 DataTable，名称都是：
窗口名称_控件名称
例如窗口 1 中插入了一个名为 Table1 的 Table，类型为 SQLTable，那么窗口打开后，可以分别通过：

```
Tables("窗口 1_Table1")
DataTables("窗口 1_Table1")
```

来引用其生成的 Table 和 DataTable，这些对象在打开窗口时自动生成，在关闭窗口后将自动删除。
SQLTable 可以设置所有的表事件，这是理所当然的，因为它会生成一个全新的 DataTable。

4.9.3.5 SQLQuery 型 Table

如果 Table 控件的类型选择为 SQlQuery，同样可以使用 Select 语句从后台提取数据生成 Table。
不过和 SQLTable 类型相比，有三点不同：
1、Select 语句可以从多个表取数据。
2、可以不包括主键列
3、生成的 Table 不能编辑的。
所以 SQLQuery 和 SQLTable 的差别，其实就是查询表和数据表的差别。

4.9.3.6 按条件加载数据

对于 SQLTable 和 SQLQuery，如果数据量不大，可以将数据全部加载进来，然后根据需要进行筛选显示，这和普通的表没有不同。
如果数据量比较大，可以根据需要加载部分数据，以提高效率。
假定窗口中有个 SQLTable，希望加载已经结账的订单，可以在设计窗口的时候，将 Select 语句设

置为：

Select * From {订单} Where 已结账 = True

一些条件加载可能没有办法在设计窗口的时候直接通过固定的 Select 语句完成。

例如希望打开窗口后加载本月的订单，就只能分两步完成：

1、在设计窗口的时候，Table 控件的类型根据需要设置为 SQLTable 或 SQLQuery，并将 Select 语句设置为：

Select * From {订单} Where [_Identify] Is Null

这样默认不会加载任何订单，是一个空表。

2、在窗口的 AfterLoad 事件加上代码：

```
Dim y As Integer = Date.Today.Year
Dim m As Integer = Date.Today.Month
Dim dt1 As New Date(y, m, 1)
Dim dt2 As New Date(y, m, Date.DaysInMonth(y, m)) '获取本月的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("窗口 1_Table1").LoadFilter = Filter '设置加载条件
DataTables("窗口 1_Table1").Load() '加载数据
```

这样打开窗口后，就会自动加载本月的订单。

有了 LoadFilter 和 Load，我们可以在打开窗口后，随时从后台加载不同条件的数据进来。

例如窗口中有个“最近七天”按钮，希望单击此按钮，加载最近七天的订单，可以将按钮的代码设置为：

```
Dim dt As Date = Date.Today.AddDays(-7)
DataTables("订单").LoadFilter = "日期 > #" & dt & "#"
DataTables("订单").Load()
```

LoadFilter 和 Load，涉及到动态加载的知识，后面会有专门的一章进行系统的讲述，目前只需有所了解即可。

4.9.3.7 Fill 方法

假定窗口中某 SQLTable，目前显示的是订单表的数据，希望单击某按钮后，能切换显示产品表的数据，显然这无法利用上一节介绍的 LoadFilter 和 Load 来实现的。

这种跨表的数据加载，可以使用 Table 的 Fill 方法，该方法可以根据指定的 SQL 语句，从指定数据源的指定表加载数据 ，新的数据源和数据表可以和当前不同。

SQLTable 和 SQLQuery 类型的 Table，以及副本型的 Table，可以通过 Fill 方法动态加载数据 ，非副本的 Normal 型 Table 不能使用此方法。

语法：

Fill(SelectString, IsQuery)

或者

Fill(SelectString, ConnectionName, IsQuery)

SelectString： Select 语句

ConnectionName：可选参数，指定数据源名称

IsQuery：　　逻辑型，设为 True，将生成 SQLQuery 型 Table，否则生成 SQLTable 型 Table。
SQLTable 是数据表，默认是可以编辑和保存的。
SQLQuery 是查询表，默认不能编辑，通过将其 AllowEdit 属性设置为 True，可以编辑查询结果，但不能保存编辑结果。

加载不同表的数据

我们可以在窗口打开后，使用 Fill 方法让 Table 控件加载另一个表的数据，例如：

```
Tables("窗口 1_Table1").Fill("Select * From {客户} Where 地区 = '华北'","nwnd",True)
```

不管 Table 原来是什么类型，显示的是什么表的数据，执行上述代码后，Table 将成为 SQLQuery 类型，并从 nwnd 数据源的客户表中 加载华北地区的客户显示。
需要特别注意的是,如果只想取部分列生成 SQLTable，那么必须包括主键列在内，例如：

```
Tables("窗口 1_Table1").Fill("Select [_Identify],[客户],[数量],[单价] From {订单}",False)
```

这里假定订单表是内部数据表，内部数据表的主键列是_Identify，如果是外部表，请使用实际的主键列名称。
如果要生成 SQLQuery，则没有这个限制：

```
Tables("窗口 1_Table1").Fill("Select [客户],[数量],[单价] From {订单}",True)
```

一定要注意，如果要生成 SQLTable，第二个参数要设置为 False，如果要生成 SQLQuery，第二个参数则要设置为 True，切不可搞混。

另一种使用场合

我们也可以在设计窗口的时候，将窗口的类型设置为 SQLTable 或 SQLQuery，但并不设置 Select 语句，而是在窗口的 AfterLoad 事件中，通过代码动态合成 Select 语句，然后再用 Fill 方法加载数据。

例如上一节介绍了让 Table 控件只加载本月数据的方法，使用的 Load 和 LoadFilter，如果采用 Fill 方法，步骤如下：

1、在设计窗口的时候，Table 控件的类型根据需要设置为 SQLTable 或 SQLQuery，但 Select 语句留空，不需要设置。
2、在窗口的 AfterLoad 事件加上代码：

```
Dim y As Integer = Date.Today.Year
Dim m As Integer = Date.Today.Month
Dim dt1 As New Date(y, m, 1)
Dim dt2 As New Date(y, m, Date.DaysInMonth(y, m)) '获取本月的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("窗口 1_Table1").Fill("Select * From {订单} Where " & Filter, False)
```

采用 Load 和 LoadFilter 实现的好处是我们可以在设计阶段，就可以将其它控件以可视化的方式绑定到 Table 控件的各列，参考：绑定到 Table 控件的列

采用 Fill 实现的好处是灵活，不单单是加载条件可以动态设置，要加载那些列，从什么数据源的什么表加载，都可以用代码动态指定，但也有坏处，就是设计窗口的时候，其他控件没办法以可视化的形式绑定到此 Table 控件的列。
我们只能在 Fill 执行之后，通过设置其它控件的 BindingField 属性，动态将这些控件绑定到 Table 控件的各列，例如在窗口的 AfterLoad 事件加上代码：

```
Tables("窗口1_Table1").Fill("Select * From {订单} Where 日期 = #" & Date.Today & "#", False)
e.Form.Controls("DateTimePicker1").BindingField = "窗口1_Table1.日期"
e.Form.Controls("TextBox1").BindingField = "窗口1_Table1.客户"
```

这样打开窗口后，Table 控件就会显示今天的订单，并将两个控件分别绑定到日期列和客户列。

请不要滥用 Fill

通过 Fill 生成的 Table，在打开窗口后，如果需要从同一个表加载其它数据进来，请采用 LoadFilter 和 Load，切不可再用 Fill(除非需要从另一个表加载数据)，因为 Fill 会重新生成 DataTable 和 Table，不仅效率低，还会导致设置好的绑定失效。

例如窗口中有个“最近七天”按钮，希望单击此按钮，加载最近七天的订单，可以将按钮的代码设置为：

```
Dim dt As Date = Date.Today.AddDays(-7)
DataTables("订单").LoadFilter = "日期 > #" & dt & "#"
DataTables("订单").Load()
```

总之， 只有需要从不同的表中加载数据时，使用 Fill 方法，如果从同一个表中加载数据，应该使用 LoadFilter 和 Load。

4.9.3.8 DataSource 属性

Table 有一个 DataSource 属性，可以通过此属性动态设置其绑定的表。

除了非副本的 Normal 类型 Table，其它类型都可以在运行过程中动态绑定到指定的表。

例如：

```
Tables("窗口 1_Table1").DataSource = DataTables("订单")
```

执行上述代码后，不管原来的 Table 是何种类型(非副本的 Normal 类型 Table 除外)，都将成为订单表的一个副本。

我们知道 Foxtable 是通过以下类型生成统计表或临时表的：

GroupTableBuilder

CrossTableBuilder

SQLGroupTableBuilder

SQLCrossTableBuilder

SQLJoinTableBuilder

DataTableBuilder

这些类型都有一个 BuildDataSource 方法，可以生成一个对象，此对象可以直接绑定到 SQLTable 和 SQLQuery 类型的 Table，以及副本型的 Table。

例如将某按钮的 Click 事件代码设为：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("日期","月份")
b.Totals.AddDef("数量")
Tables("窗口 1_Table1").DataSource = b.BuildDataSource()
```

那么单击此按钮，窗口中的 Table1 将显示各月销售数量。
提示：
1、上面的代码中生成的统计表的名称并非"统计表 1"，而是"窗口 1_Table1"，也就是说 GroupTableBuilder 指定的表名是无效的，切记切记。
2、如果某 Table 控件计划绑定到 BuildDataSource 方法生成的对象，那么在设计窗口的时候，其类型最好设置为 SQLQuery 或 SQLTable，这样此 Table 控件就拥有所有的表事件，便于进行精细设计。

4.9.3.9 设置显示列

在默认情况下，窗口控件 Table 会显示所有的列。
有三种方法可以设置显示部分列，以及显示列的顺序及宽度。
1、使用表视图
如果是 Normal 类型的 Table，不管是副本还是非副本，都可以使用原表的表视图。
例如可以在窗口的 AfterLoad 事件中加入下面的代码：

```
Tables("窗口 1_Table1").OpenView("精简")
```

这样打开窗口后，Table 控件将自动调用绑定表的名为"精简"的表视图。
2、运行时调整
如果没有定义表视图，或者因为是 SQLTable 或者 SQLQuery 型 Table，不能使用表视图的话，则可以使用 SetColVisibleWidth 方法。
例如：

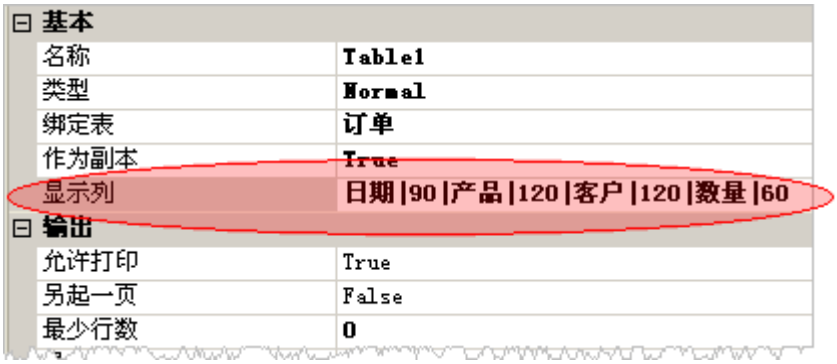
```
Tables("窗口 1_Table1").SetColVisibleWidth("日期|90|产品|120|客户|120|数量|60")
```

执行上述代码后，订单表将按顺序显示日期、产品、客户、数量四列，宽度分别为 90、120、120、60，而其余各列将被隐藏。
SetColVisibleWidth 对所有类型的 Table 都有效，相对于表视图来说更为灵活。
3、同步显示列
假定窗口 1 的 Table1 绑定到订单表，作为副本；希望窗口打开后，此副本 Table 和原订单表的列顺序和列宽保持一致，为此可在窗口的 AfterLoad 事件中设置代码：

```
Tables("窗口 1_Table1").SetColVisibleWidth(Tables("订单").GetColVisibleWidth())
```

参考：GetColVisibleWidth

4、设计时指定
在窗口设计的时候，Table 控件有一个名为“显示列”的属性，可以一次性地设置要显示的列及其列宽：



按上图所示设置，打开窗口后，Table 将按顺序显示日期、产品、客户、数量四列，宽度分别为 90、120、120、60，而其余各列将被隐藏。

4、单独调整某一列

如果要单独调整某一列的位置，可以使用 Col 的 Move 方法，例如将日期列移到最前面：

```
Tables("窗口 1_Table1").Cols("日期").Move(0)
```

也可以单独隐藏某一列，例如:

```
Tables("窗口 1_Table1").Cols("折扣").Visible = False
```

4.9.3.10 作为列表框

窗口设计的时候，通过适当地设置 Table 控件的属性，可以使其表现得像一个列表框：



效果：

客户	雇员	产品	单价	折扣	数量
CS03	EP04	PD05	17	0.1	650
CS01	EP01	PD02	20	0.1	4
CS04	EP02	PD05	17	0	
CS03	EP04	PD01	18	0.15	
CS03	EP04	PD03	10	0	49
CS04	EP05	PD04	17.6	0	

如果要复选列表框的效果，可以按下图所示设置：



效果：



	客户	雇员	产品	单价	折扣	数量
☐	CS03	EP04	PD05	17	0.1	65
☐	CS01	EP01	PD02	20	0.1	4
☐	CS04	EP02	PD05	17	0	
☐	CS03	EP04	PD01	18	0.15	6
☐	CS03	EP04	PD03	10	0	490
☐	CS04	EP05	PD04	17.6	0	

相关属性：

ListMode

RowHeaderVisible

ShowCheckBox

FocusRect

请参考：

处理复选框

4.9.3.11 Table 的双重身份

窗口中的 Table 具备双重身份，使用不同身份时，类型不同，名称也不同。

作为控件

当作为窗口中的控件时，其类型为 WinForm.Table，名称就是设计窗口时赋予的名称。

示例

```
Dim t As WinForm.Table = e.Form.Controls("Table1")
t.SetBounds(0,0,400,300)
```

作为 Table

当作为普通的 Table 使用的时候，其类型自然是 Table。

至于名称则要区分两种情况。

如果类型为 Normal，且非副本，此时窗口并不会创建新的 Table，只是将原来的 Table 移到窗口中显示，所以其名称自然就等于绑定的 Table 名称。

例如将订单表移到主窗口后，对其的调用还是和原来一样：

```
Tables("订单").AllowEdit = True
```

如果类型为 Normal，且为副本，或者类型为 SQLTable、SQLQuery，此时窗口会创建一个新的 Table，此新 Table 的名称为：

窗口名称_控件名称

假定这个 Table 的名称为 Table1，且在窗口 1 中，那么代码中必须使用“窗口 1_Table1”的名称来调用此 Table：

```
Tables("窗口 1_Table1").AllowEdit = False
```

获得对应的 DataTable

对于 Normal 型 Table，不管是副本还是非副本，其对应的 DataTable 的名称，就是其绑定的表的名称，例如：

```
DataTables("订单").AllowEdit = True
```

对于 SQLTable、SQLQuery 型 Table，对应的 DataTable 名称为：

窗口名称_控件名称

例如：

```
DataTables("窗口 1_Table1").AllowEdit = False
```

也可以通过 Table 的 DataTable 属性返回其对应的 DataTable，例如：

```
Tables("窗口 1_Table1").DataTable.Load()
```

通过控件的 Table 属性获得对应的 Table 和 DataTable

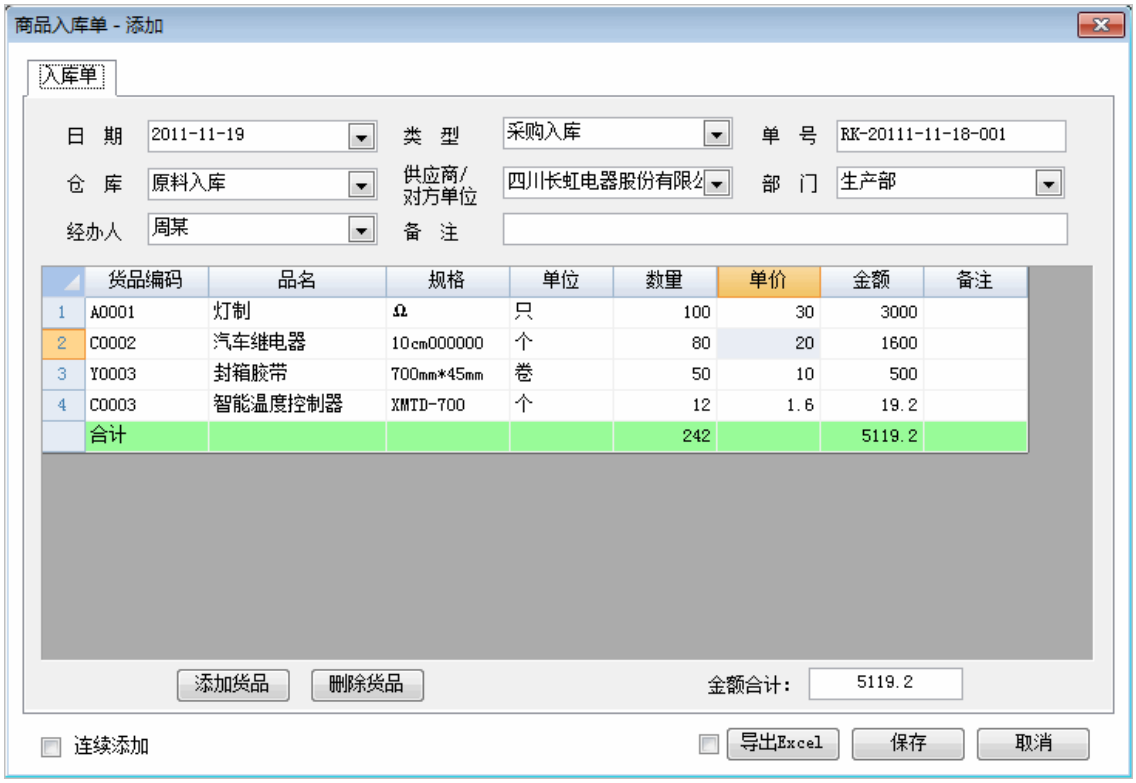
Table 控件本身有一个 Table 属性，用于返回其绑定的 Table，例如：

```
Dim wbl As WinForm.Table = e.Form.Controls("Table1")
Dim tbl As Table = wbl.Table
tbl.DataTable.Save()
```

上面的代码用于保存Table控件绑定的表，代码是通用的，和Table控件的类型无关，所以无需为不同类型的Table控件编写不同的代码。

4.9.3.12 窗口中的关联表

窗口中的 Table 控件也可以绑定到关联表，这样我们可以轻松地设计出下图所示的录入界面：



设计的方法很简单，上面的输入框绑定到父表(入库单)的各列，下面的 Table 控件绑定到关联子表(入库单.入库明细)：



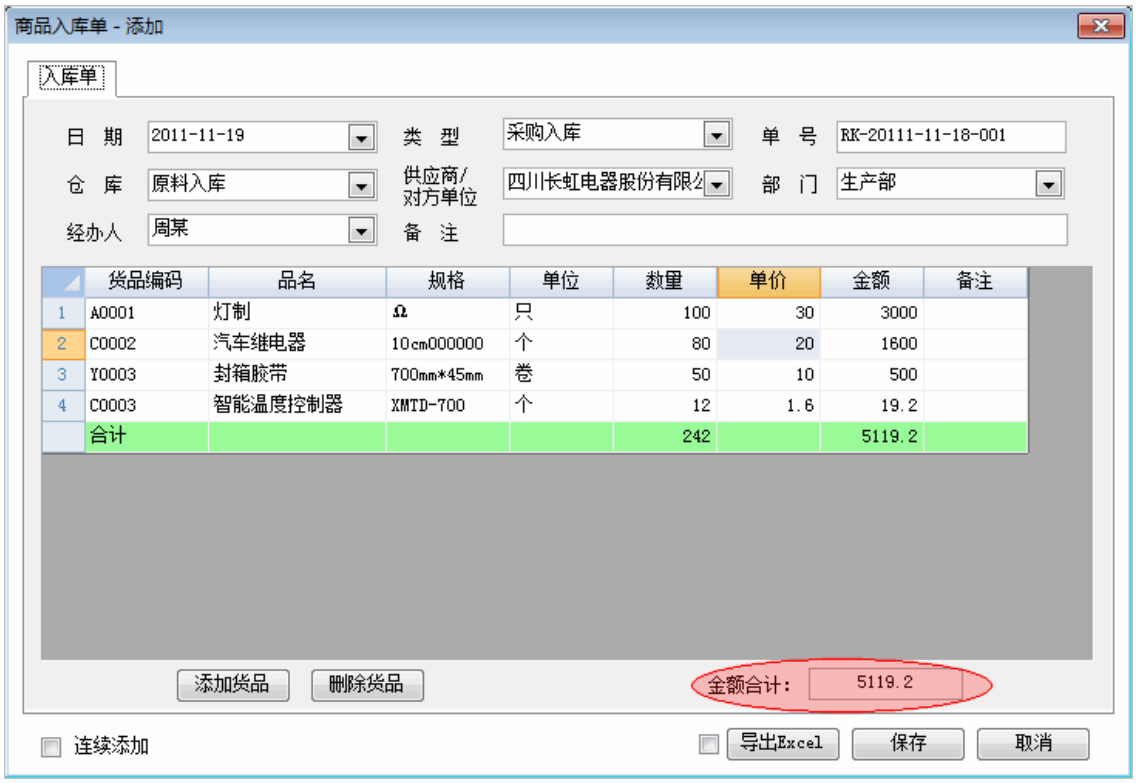
显示自动合计行

为了在关联表中显示自动合计行，可以在窗口的 AfterLoad 事件中加上代码：

```
With Tables("入库单.入库明细")
    .Cols("数量").GrandTotal = True
    .Cols("金额").GrandTotal = True
    .GrandTotal = True
End With
```

在文本框中显示合计结果

经常有用户会提问：下面的窗口中，如何在金额合计位置的文本框，动态汇总显示总金额？



其实很简单，就是利用使用指南中统计子表数据的知识而已。

假定入库单和入库明细之间的关联的名称为“rkmx”，在入库表中增加一个名为“金额”的表达式列，将其表达式设置为：Sum(Child(rkmx).金额)。

然后在设计窗口的时候，将上面窗口中金额合计位置的文本框绑定到这个表达式列即可。

关于增加行

上面的窗口同时包括两个表的数据，分别为入库单和入库明细，通常的操作是增加一个入库单，然后在此入库单下增加若干明细，那么怎样在这个窗口中分别给这两个表增加行呢？

很简单，我们可以在窗口中插入一个名为“新增入库单”的按钮，将其代码设置为：

```
Tables("入库单").AddNew()
```

这样单击此按钮，就可以在父表（入库单）中增加一行，即增加一个入库单。

然后再插入一个名为“新增入库明细”的按钮，将其代码设置为：

```
Tables("入库单.入库明细").AddNew()
```

这样单击此按钮，就可以在关联子表(入库单.入库明细)增加一行，既在当前入库单下增加一个入库明细。

当然也可以用鼠标右击窗口中的 Table，从弹出的快捷菜单中，执行“行相关 → 增加行”命令，来在对应的 Table 中增加行。

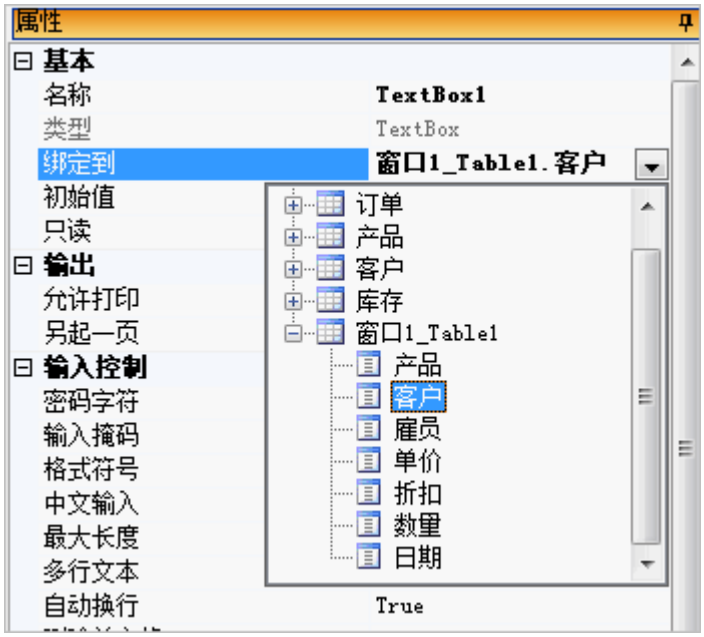
4.9.3.13 绑定到 Table 控件的列

虽然副本、SQLTable 和 SQLQuery 型的 Table 控件，只有在打开窗口的时候，才会生成对应的 Table，但是在设计阶段，我们依然可以用可视化的形式将各种控件绑定到这些 Table 的列。

在绑定到这些 Table 列之前，我们必须先将 Table 控件设置好；例如对于副本型 Table，要设置其绑定表；对于 SQLTable 和 SQLQuery，要设置好 Select 语句以及数据源(如果使用的不是内部表)。

例如有个窗口，名为“窗口 1”，窗口中有个 Table 控件，名为“Table1”，并绑定到订单表，作为副

本，在运行阶段，该 Tabble 控件生成的 Table 名称为“窗口 1_Table1”。
在设计阶段，我们可以用可视化的形式，直接将各种控件绑定到这个 Table 的列：



也可以直接拖动列到窗口中：



4.9.3.14 用 Table 控件模拟关联表

其实不管是双向生成还是单向生成，都会产生关联表，对于性能有一定程度的影响，只是双向生成的时候尤甚。
所以要追求极致的性能，最好是在设置关联的时候，将关联表生成模式选择为“无”。
但是关联表确实给我们带来了很大的便利，我们岂能因噎废食？
可否鱼和熊掌兼得，既可以得到关联表的好处，又不影响性能呢？
完全可以，这就是本节要介绍的内容。

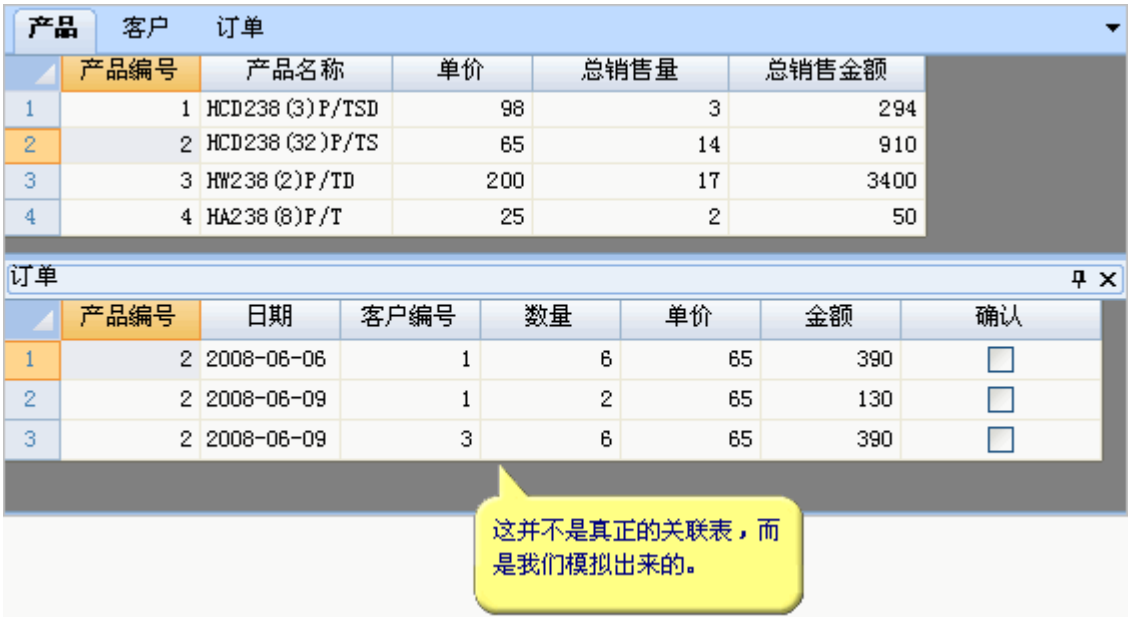
请首先请打开 CaseStudy 目录下的文件：模拟关联表.Table

该文件的产品表和订单表之间正常建立了关联， 所以可以充分享受关联带来的好处，例如使用表达式来统计和引用数据，在代码中使用 GetParentRow 和 GetChildRows 方法快捷定位或筛选出关联行。

由于在建立这个关联的时候，关联表的生成模式选择了“无”，不会像常规那样自动创建和显示关联表，所以此关联的存在，对于性能几乎没有影响。

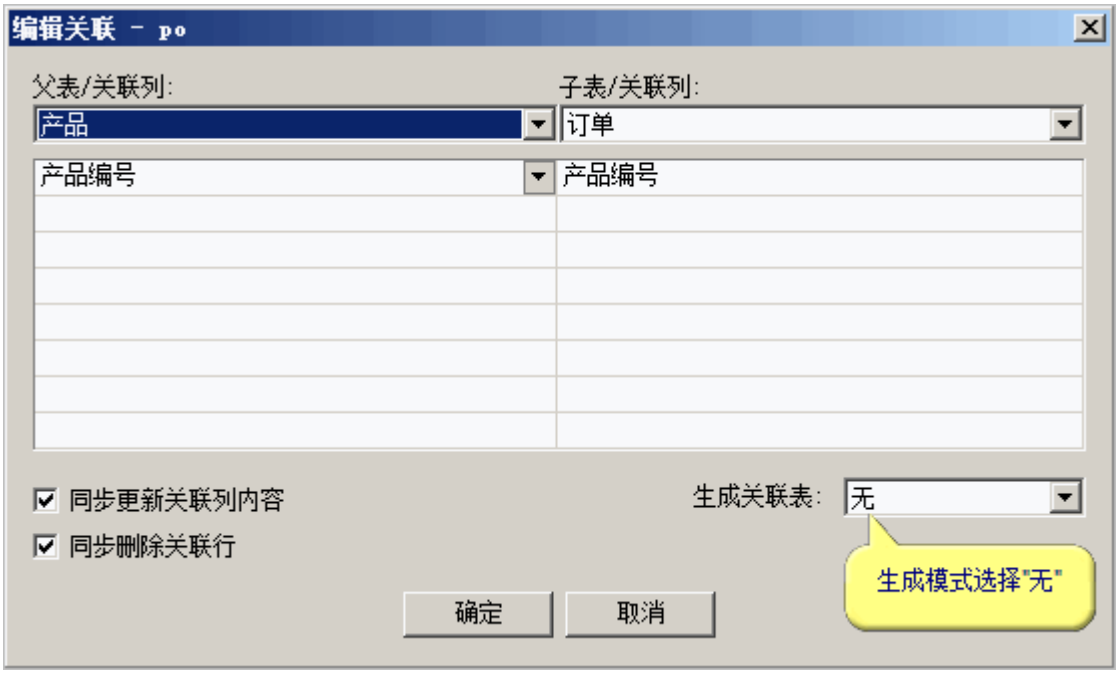
但是我们用窗口和 Table 控件模拟了一个关联表，打开此窗口后，如果在产品表选择某产品，窗口中的 Table 控件可以自动显示出该产品的所有订单，表现得就像真的关联表一样。

由于这个“关联表”是模拟的，平时并不存在，打开窗口的时候出现，关闭窗口的时候自动消失，所以这个“关联表”对于性能的影响极微。



设计步骤：

1、在产品表和订单表之间建立关联，关联名为"po"，注意关联表生成模式要选择为“无”：



2、在产品表新建一个窗口，窗口类型为停靠，停靠位置为底端。

3、插入一个 Table 控件，绑定到订单表，将其“作为副本”属性设置为 True，“允许编辑”属性设置为 True，“停靠”属性设为 Fill。
4、窗口的 AfterLoad 事件设为：

```
Dim t As Table = Tables("窗口 1_Table1")
With Tables("产品")
    If .Current Is Nothing Then
        t.Filter = "False"
    Else
        t.Filter = "产品编号 = " & .Current("产品编号")
    End IF
End With
```

此段代码确保打开窗口后，Tables("窗口 1_Table1")只显示当前选定产品的订单。
5、关闭窗口设计器，回到产品表，打开表属性设置窗口，将其 CurrentChanged 事件代码设为：

```
If Forms("窗口 1").Opened() '一定要判断用于模拟关联表的窗口是否已经打开
    Dim t As Table = Tables("窗口 1_Table1")
    With Tables("产品")
        If .Current Is Nothing Then
            t.Filter = "False"
        Else
            t.Filter = "产品编号 = " & .Current("产品编号")
        End IF
    End With
End If
```

当我们在产品表选择不同的产品时，上述代码会判断窗口是否已经打开， 如果已经打开，则更新Tables("窗口 1_Table1")的 Filter 属性，使其显示新选定产品的订单。
6、将订单表的 DataRowAdding 事件代码设置为：

```
If CurrentTable.Name = "窗口1_Table1" Then '如果当前表是模拟的关联表
   Dim r As Row = Tables("产品").Current
   If r IsNot Nothing Then
      e.DataRow("产品编号") = r("产品编号")
   End If
End If
```

当我们选择这个模拟的关联表，然后单击菜单中的增加行按钮时，上述代码会检查产品表中是否有选定行，如果有则新增订单自动继承此选定产品的产品编号。
如果没有这段代码，在模拟的关联表增加行时，新增行是看不见的，因为新增行的产品编号为空，而模拟的关联表只显示产品编号为选定产品的订单。

至此，我们用窗口的Table控件模拟出来的关联表，和真正的关联表一样，能够动态显示选定产品的订单，新增行也能自动继承选定产品的产品编号，不同的只是这个“关联表”的名称是Tables("窗口1_Table1")，而不是Tables("产品.订单")。

7、在产品表增加两个表达式列，分别为总销售量和总销售金额，总销售量的表达式：Sum(Child(po).数量)，总销售金额的表达式为：Sum(Child(po).金额)

8、将订单表的 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.Name
    Case "产品编号"
        Dim pr As DataRow = e.DataRow.GetParentRow("产品")
        If pr IsNot Nothing Then
            e.DataRow("单价") = pr("单价")
        End If
End Select
```

这样在订单表输入产品编号，会自动从产品表中找出对应的产品，然后取其单价填入到订单表的单价列中。
9、将产品表的 DataColChanged 事件代码设置为：

```
If e.DataCol.Name = "单价" Then '更新未确认订单的单价
    Dim drs As List(of DataRow)
    drs = e.DataRow.GetChildRows("订单")
    For Each dr As DataRow In drs
        If dr("确认") = False Then
            dr("单价") = e.DataRow("单价")
        End If
    Next
End If
```

这样在产品表修改某产品的单价，系统会找出订购此产品的全部订单，然后并将未确认订单的单价更新为新的单价。
一点提示：
我们没有必要将关联表视为洪水猛兽，只有确实因为关联表过多，对性能有严重影响时，才可以考虑编码模拟关联表。

4.9.3.15 不同类型的 Table 共用录入窗口

通过控件的 BindingField 属性，可以在运行过程中将控件动态绑定到 Table 的列。
假定表 A 和表 B 建立了关联，另外窗口 1 中插入了一个 Table 控件，此 Table 控件绑定到表 B，“作为副本”属性设置为 True。
这样系统中将存在三个表 B:

主表： Tables("表 B")
关联表： Tables("表 A.表 B")
窗口中： Tables("窗口 1_Table1")
假定我们为表 B 设计了一个录入窗口，希望在上述三个 Table 中双击任何一行，即可打开这个录入窗口来编辑修改此行的数据。
设计步骤：

1、在全局代码中加上以下代码：

```
Public btName As String
```

2、将表 B 的 DoubleClick 事件代码设置为：

```
btName = e.Table.Name '将Table名称保存在全局标量btName中
e.Cancel = True '禁止进入编辑状态
Forms("录入窗口名").Open()
```

3、按常规设计好这个录入窗口，各个字段如常绑定到表B的各列。

4、将窗口的AfterLoad事件代码设置为：

```
If btName   > "" AndAlso btName <> "表B" Then
    For Each c As Winform.Control In e.Form.Controls
        If c.Bindingfield > "" AndAlso c.Bindingfield.StartsWith("表B.")   Then
            c.BindingField= c.BindingField.Replace("表B.", btName & ".")
        End If
    Next
End If
```

5、如果窗口中有增加行的按钮，此按钮的代码应该为：

```
Tables(btName).AddNew()
```

6、同样删除行按钮的代码为：

```
Tables(btName).Current.Delete()
```

整个设计思路很简单:

1、双击基于表B的某个Table时，将此Table的名称保存在全局变量btName中,然后在窗口的Afteload事件中修改各控件的BindingField属性，绑定到这个Table。
2、窗口中增加行、删除行这些按钮的代码，通过Tables(btName)来获得要操作的表。

4.9.3.16 窗口 Table 与数据标记

如果窗口 Table 是 Normal 型，不管是副本还是非副本，都可以按常规增加自定义样式，然后设置 DrawCell 事件代码标记数据。

如果窗口 Table 是 SQLTable 或 SQLQuery，虽然同样可以设置 DrawCell 事件代码，但是却无法通过菜单增加自定义样式，只能通过 AddUserStyle 方法用代码增加自定义样式。

示例

假定窗口有个 Table 控件，类型 SQLTable，希望用红底白字在数量列标出值超过 500 的单元格。

设计步骤

1、在窗口的 AfterLoad 事件中加入代码：

```
DataTables("窗口 1_Table1").AddUserStyle("a", Color.Red, Color.White)
```

上述代码增加了一个红底白字的样式，样式名为"a"。

2、然后设置 Table 控件的 DrawCell 事件代码：

```
If e.Col.Name = "数量" Then
    If e.Row("数量") >= 500 Then
       e.Style = "a"
    End If
End If
```

4.9.3.17 副本 Table 之行位置联动

假定窗口中插入了一个 Table 控件，绑定到"表 A"，作为副本。

希望在这个副本 Table 中选定一行时，表 A 也能自动选定这一行，实现这个任务很简单，只需将副本 Table 的 CurrentChanged 事件代码设置为：

```
Dim dr As DataRow = e.Table.Current.DataRow
Dim wz As Integer = Tables("表A").FindRow(dr)
If wz >=0 Then
     Tables("表A").Position = wz
End If
```

反过来，如果你想在表A选定某行时，副本Table也能同步选定同一样，可以讲表A的CurrentChanged事件代码设置为：

```
If Forms("窗口1").Opened Then '如果窗口已经打开
    Dim dr As DataRow = e.Table.Current.DataRow
    Dim wz As Integer = Tables("窗口1_Table1").FindRow(dr)
    If wz >=0 Then
       Tables("窗口1_Table1").Position = wz
    End If
End If
```

4.9.3.18 在表事件中引用窗口

在给窗口中的 Table 控件编写事件代码的时候，可以通过 e.Form 来引用此窗口，例如：

```
If e.Table.Current("审核") = True
    e.Form.Controls("Button1").Enabled = False
```

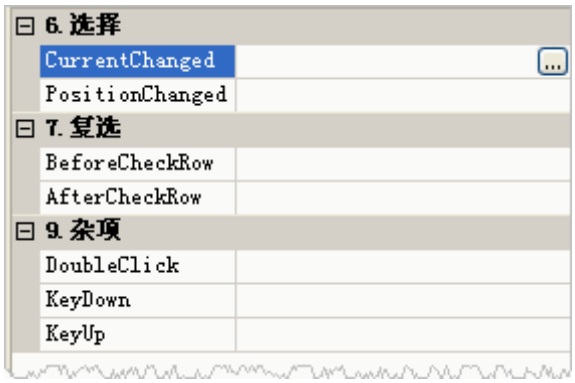
```
Else
    e.Form.Controls("Button1").Enabled = True
End If
```

此外 Table 有个 Form 属性，当此 Table 位于某个窗口中时，可以用此属性返回该窗口。
这样我们可以在任何代码中获取 Table 所在的窗口（如果此 Table 位于窗口中的话），例如：

```
Dim tb As Table = Tables("表A")
If tb.Form IsNot Nothing Then
    If tb.Current("审核") = True
        tb.Form.Controls("Button1").Enabled = False
    Else
        tb.Form.Controls("Button1").Enabled = True
    End If
End If
```

4.9.3.19 再谈窗口 Table 与表事件

其实本节的内容，前面的章节都有介绍。
但可能是由于比较分散的原因，我们在技术支持过程中，还是经常会遇到客户提出类似这样的问题：窗口 Table 为什么没有事件？
所以这里再集中复述一遍。
窗口 Table 有四种类型，这四种 Table 有的没有表事件，有的有部分表事件，有的有全部表事件。
Normal 型 Table（正本）
没有任何表事件，但是会完全继承表属性中设置的事件代码。这是因为非副本的 Normal 型 Table 只是将原来的 Table 移到窗口中，例如对于订单表来说，不管是显示在主界面中，还是显示在窗口中，它始终都是 Tables("订单")，它还是它，没有任何改变，只是出现在不同的地方而已。
Normal 型 Table（副本）
会完全继承表属性中设置的所有事件代码，但可以单独设置以下事件代码：



如果副本 Table 没有设置上图这些表事件，那么表属性中设置的这些事件代码对副本 Table 一样有效；如果你希望这些事件触发时，副本 Table 和正本 Table 有不一样的行为，那么就可以给副本 Table 单独设置这些表事件代码。
副本 Table 能够将这些常用表事件独立出来设置，会给大家的开发工作带来很大的便利。
SQLTable

SQLTable 可以设置所有的表事件，这是理所当然的，因为它会生成一个全新的 DataTable。

SQLQuery

SQLQuery 同样可以设置所有的表事件，因为它也会生成一个全新的 DataTable，它和 SQLTable 的差别，只是数据表和查询表的差别。

4.9.3.20 判断 Table 的类型

如果需要判断 Table 的类型，以及是否是副本，请参考:

TableType

IsCopy

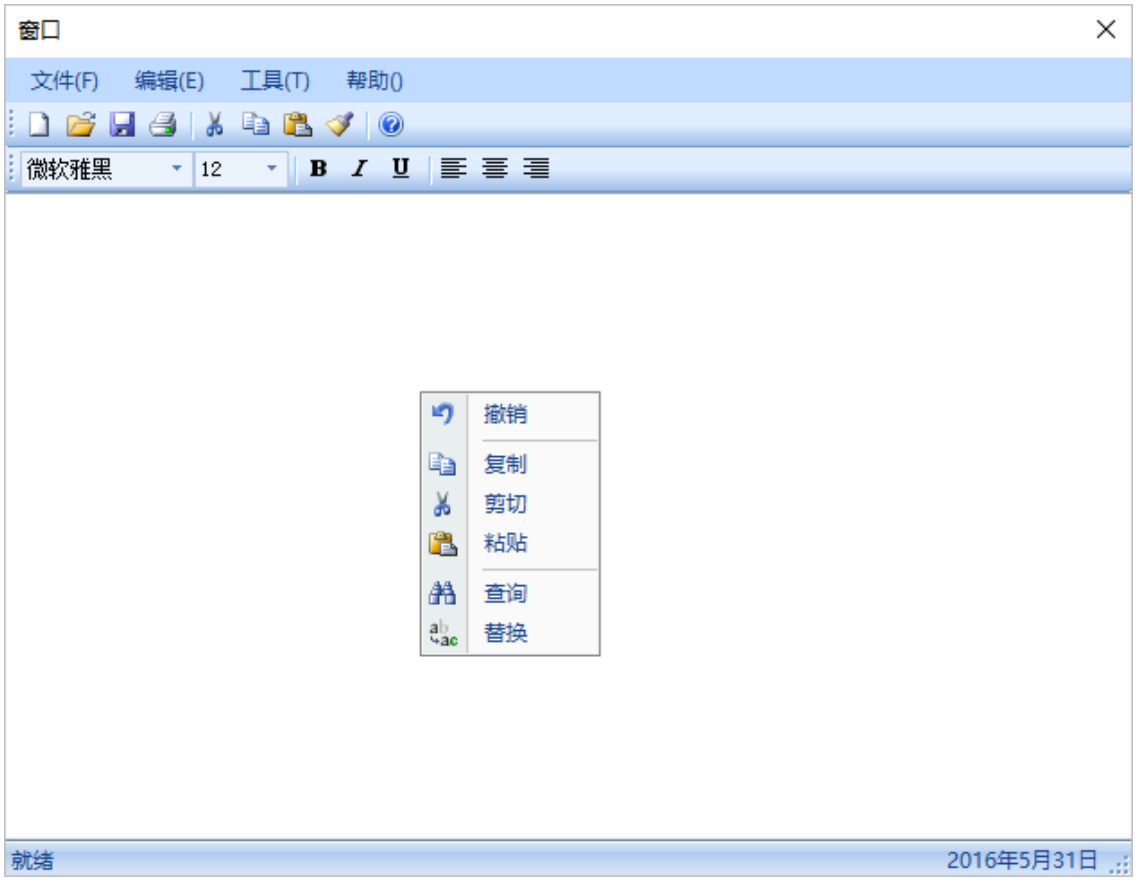
4.9.4 窗口中的菜单

4.9.4.1 关于窗口菜单

可以在窗口中设计主菜单、工具栏以、状态栏以及快捷菜单。

由于 Foxtable 的系统菜单是可以重新设计的（参考《菜单设计》这一章），且主窗口、停靠窗口以及共享窗口，均显示在主界面中，可以和系统菜单融合在一起（这也是我们推荐的），至于独立窗口和模式窗口，多数用于完成录入、查询或统计等单项任务， 通常也不需要菜单。所以实际开发的时候，需要单独在窗口中再设计一个菜单的机会并不多，因此 Foxtable 早期的版本并没有提供窗口菜单设计功能。

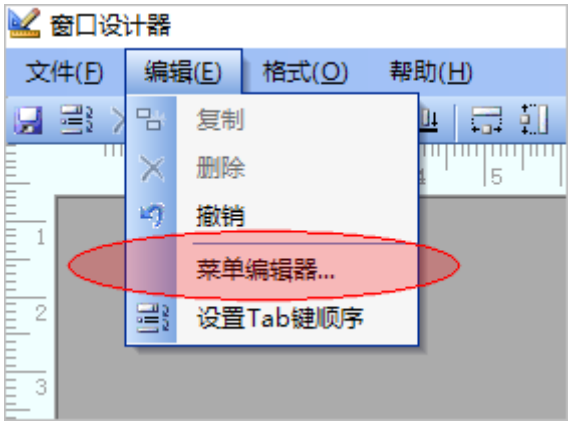
窗口菜单和系统菜单不同，系统菜单是 Ribbon 样式，类似 Office 2007，而窗口菜单则是传统样式，如下图所示：



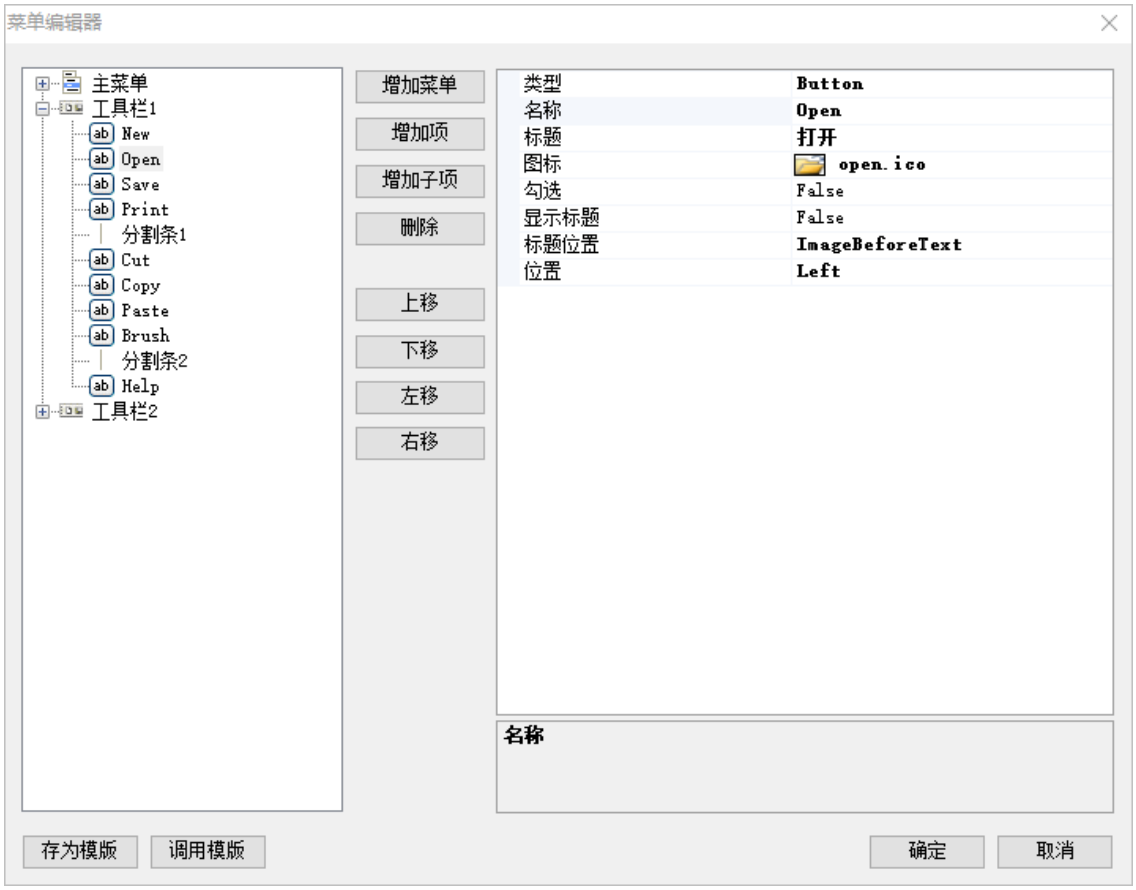
上图的主菜单、工具栏、快捷菜单以及状态栏都是用窗口的菜单编辑器设计的。

4.9.4.2 窗口菜单编辑器

在窗口设计器的“编辑”菜单中，单击“菜单编辑器”，可以打开窗口菜单编辑器：

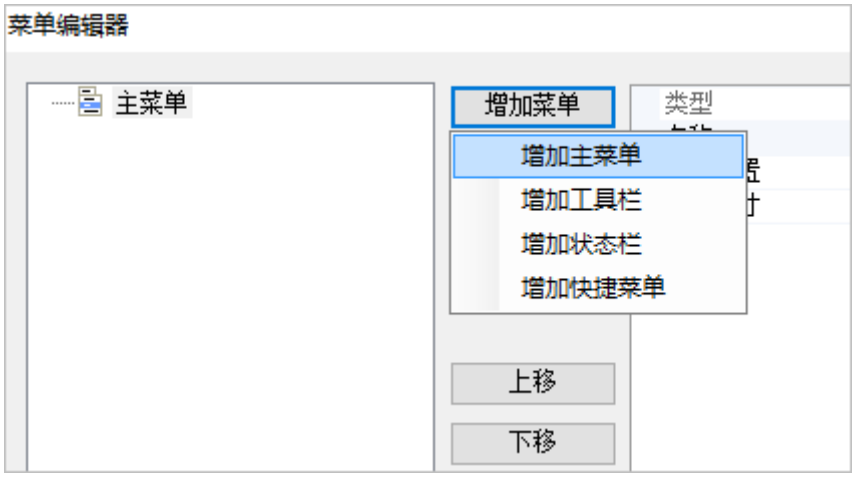


菜单编辑器：

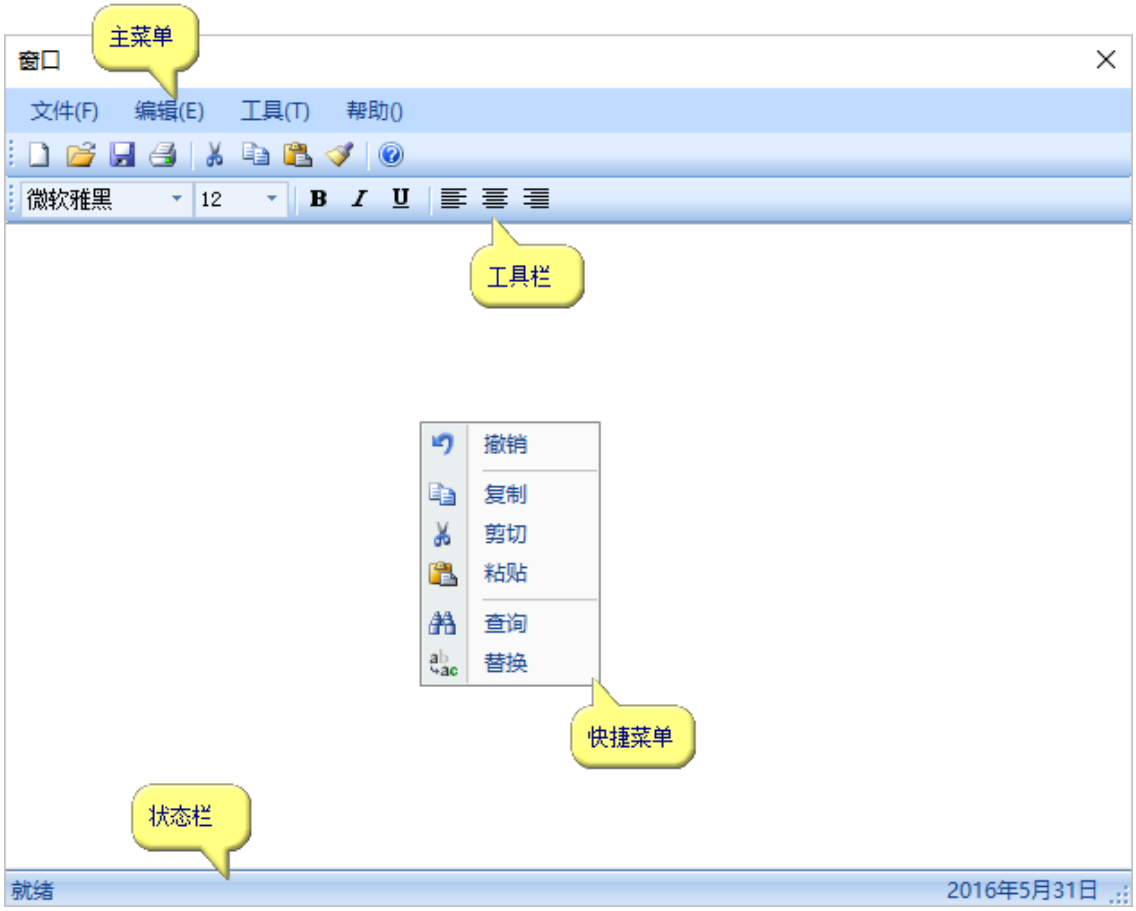


菜单设计步骤：

1、单击"增加菜单"按钮，会出现一个下拉菜单，用于选择增加主菜单、工具栏、状态栏还是快捷菜单，我们将这四种类型统称为菜单：



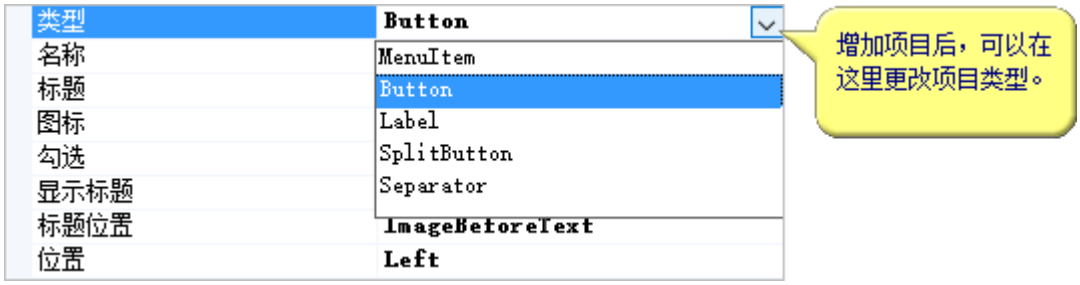
菜单的四种形态：



提示：一个窗口只能增加一个主菜单，只能增加一个状态栏。

2、在窗口左侧选择某一个菜单，单击“增加项”按钮，可以在这个菜单下增加菜单项目。

3、每次增加菜单项目，Foxtable 会自动根据当前菜单选择合适的项目类型，如果默认的项目类型不符合要求，可以在增加项目后，在属性窗口修改 菜单项目的类型：



菜单项目有五种类型，分别是：

MenuItem： 标准的菜单项目，通常用于主菜单和快捷菜单，MenuItem 可以包括子项目。

Button： 按钮，通常用于工具栏。

SplitButton：组合按钮，通常用于工具栏；组合按钮是 Button 和 MenuItem 的组合，可以包括子项目，单击按钮左侧触发单击事件，单击按钮右侧会出现一个菜单。

Label： 标签。

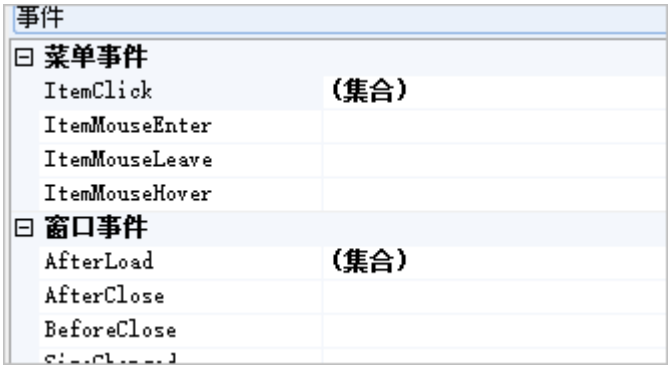
Separator： 分割条

我们之前介绍系统菜单设计的时候，已经接触过这些菜单项目类型，这里就不赘述了。

除了上述五种菜单项目类型，菜单还能容纳各种标准控件，例如 TextBox、ComboBox、DateTimePicker、NumericComboBox 等等，所以窗口菜单的功能可以变得很丰富 ，我们会在后面的章节中介绍如何向菜单中添加标准控件。

4.9.4.3 编写菜单事件代码

我们并不能给单个的菜单或菜单项目设置代码，只能在窗口事件中统一设置菜单代码，菜单事件有四个：



其中最常用的是 ItemClick 事件，该事件在单击某个菜单项目后执行。

所有菜单事件的 e 参数都是相同的：

Form： 触发事件的窗口

Strip： 触发事件的菜单

StripItem：触发是事件的菜单项目。

通常在 ItemClick 中，判断触发事件的 StripItem 的名称，来执行对应的操作，例如：

```
Select Case e.StripItem.Name
  Case "Open"
    '打开代码
  Case "Save"
    '保存代码
  Case "New"
    '新建代码
  Case  "Print"
    '打印代码
  Case "Exit"
    '退出代码
End Select
```

注意编写代码的时候，不要混线了名称和标题，例如下图中名称是“Open”，不是“打开”，这是新手很容易犯的错误：



4.9.4.4 图标文件位置和大小

文件位置

窗口菜单所使用的图标文件，请事先复制到项目的 Images 子目录中。

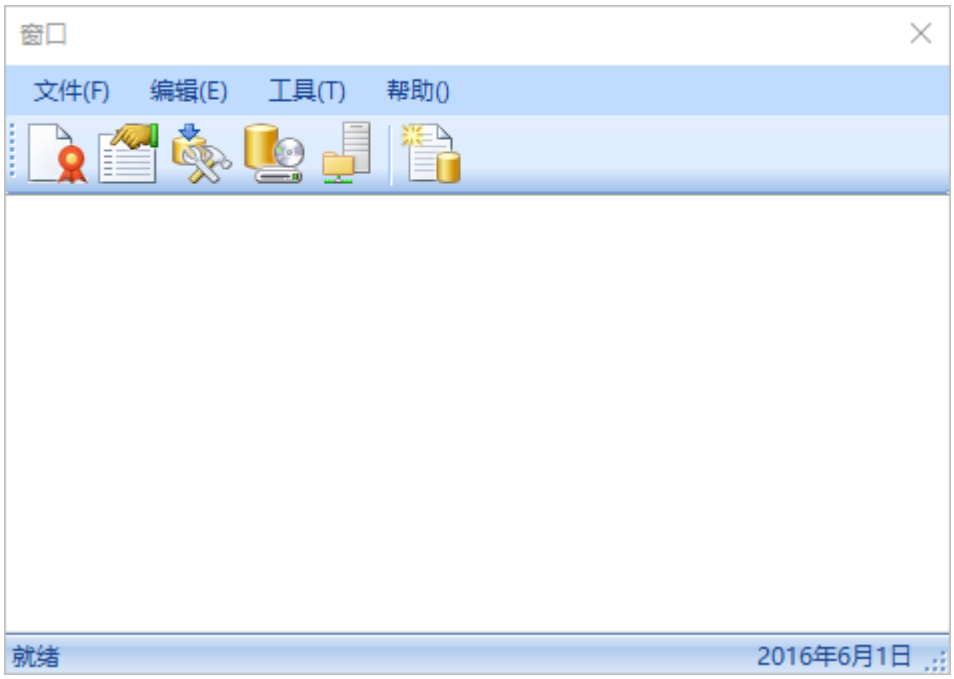
如果设计菜单的时候，从其它目录选择图标文件，系统会自动将所选择的图标文件复制到 Images 子目录中。

图标大小

默认的菜单图标尺寸为 16*16 像素，可以选择某个菜单，在右侧属性窗口更改图标尺寸：

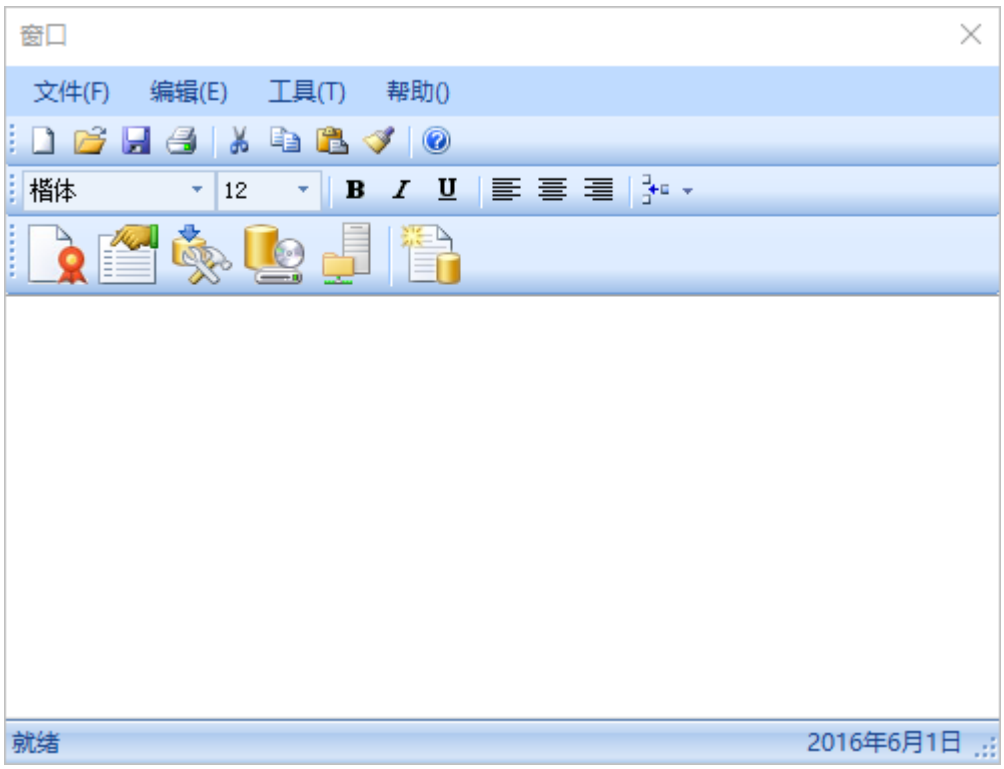


下图的工具栏的图标尺寸是 32*32 像素：



和 Ribbon 风格的系统菜单不一样，窗口中同一菜单的按钮图标尺寸必须是相同的。

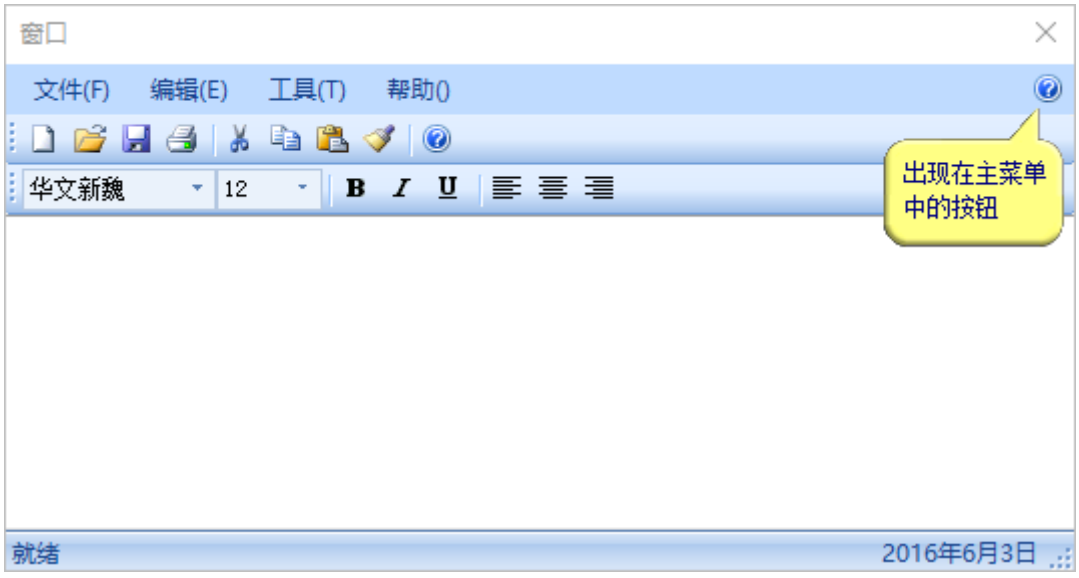
但不同的菜单可以分别指定图标尺寸，例如下图中，前两个工具栏的图标尺寸是 16*16 像素，第三个工具栏的图标尺寸是 32*32 像素：



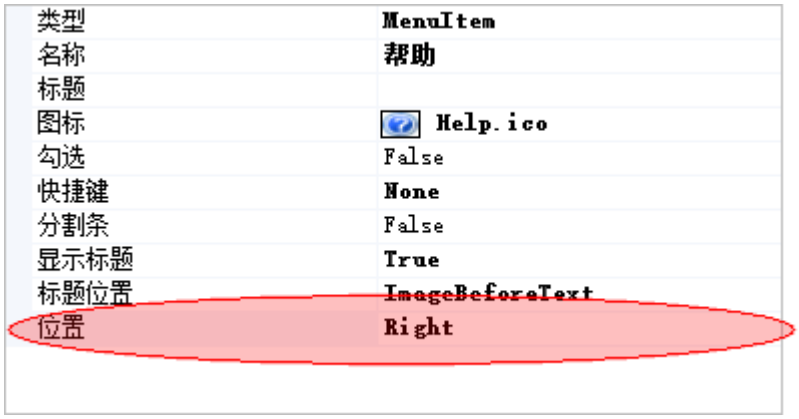
4.9.4.5 菜单项目类型和位置

窗口菜单有四种项目类型，分别是：MenuItem、Button、SplitButton、Label。
MenuItem 通常用于主菜单和快捷菜单，Button 和 SplitButton 通常用于工具栏，而 Label 通常用于状态栏，这样菜单的显示和使用效果更好。

这并不是规定，只要你愿意，主菜单也可以加上按钮，工具栏也可以显示菜单，例如下图中，主菜单就有一个按钮：

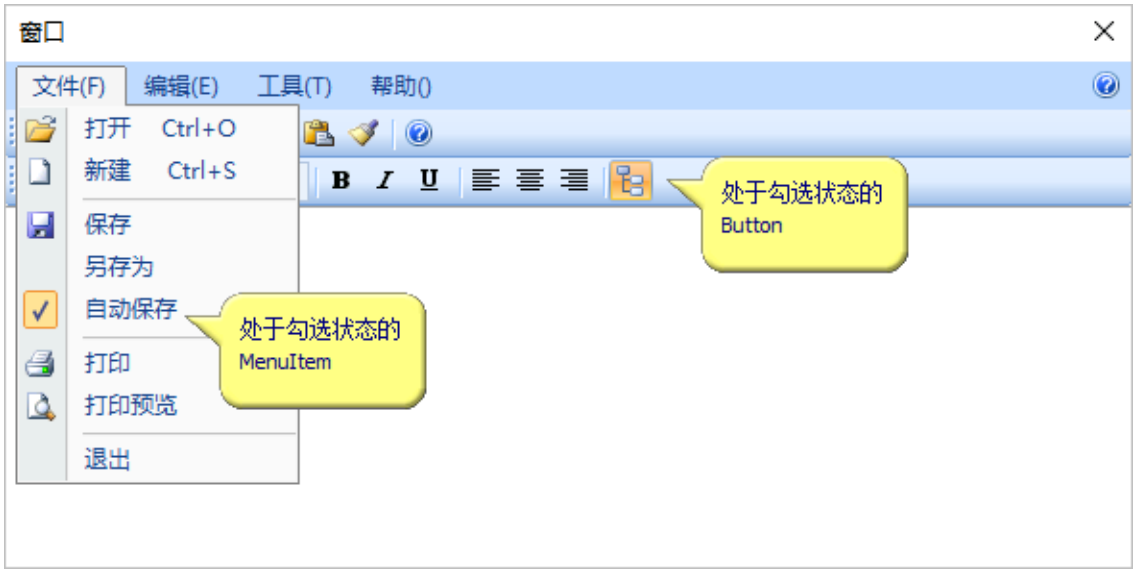


你也许会问，这个按钮怎么会出现在右侧呢？ 很简单，设计菜单的时候，将按钮的属性“位置”设为“Right”即可：



4.9.4.6 菜单项目的勾选状态

MenuItem 和 Buton 类型的菜单项目，都有一个勾选(Checked)属性，它们都可以作为状态按钮使用：



代码的编写

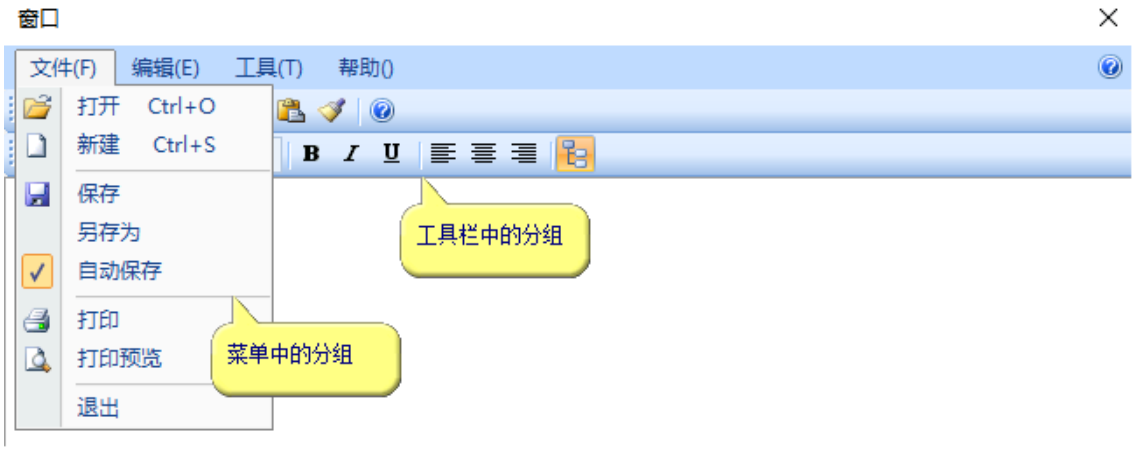
假定希望单击某个菜单项目，能够改变其勾选状态，并执行对应的操作，可以在 ItemClick 事件这样编写代码：

```
Select Case e.StripItem.Name
    Case "AutoSave"
        If e.StripItem.Checked = True '如果之前处于勾选状态
            e.StripItem.Checked = False   '取消勾选
            '执行相关代码
        Else '如果之前处于未勾选状态
            e.StripItem.Checked = True '勾选此项目
```

```
        '执行相关代码
    End If
End Select
```

4.9.4.7 菜单项目的分组显示

为了让菜单和工具栏看起来更有条理，有必要对其中的项目进行分组，在不同的分组之间插入分割条(Separator)，如：



实现分组功能很简单，只需在需要分组的位置插入一个项目，然后将项目的类型改为 Separator 即可：

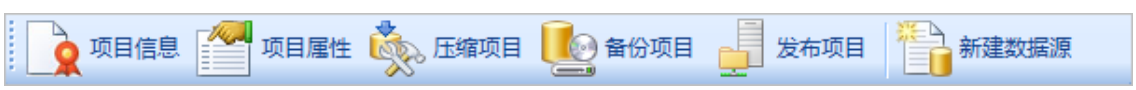


4.9.4.8 图标和标题的关系

在默认情况下，窗口工具栏中的按钮只显示图标，不会显示标题：

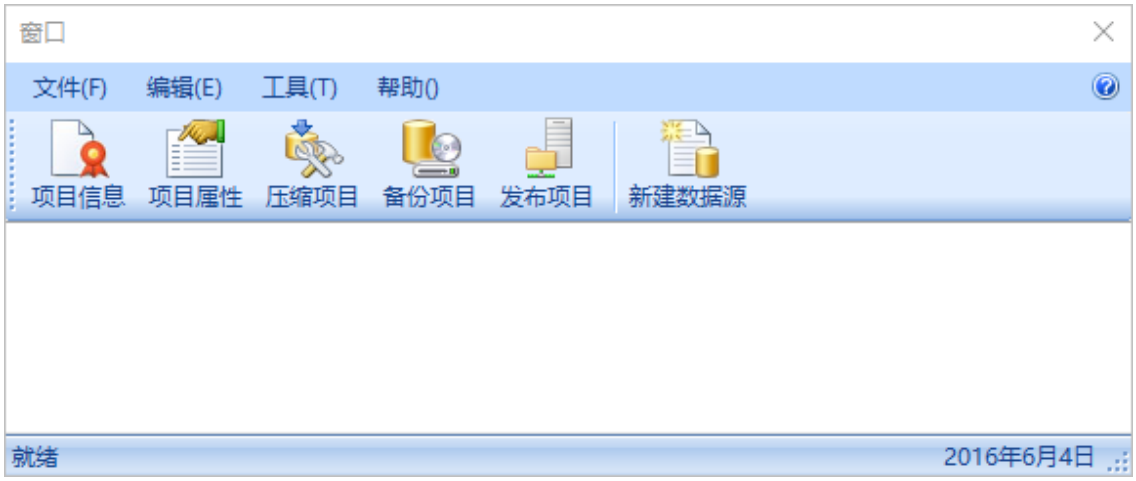


如果希望同时显示图标和标题，在设计菜单的时候，将菜单项目的“显示标题”属性设置为 True 即可：



标题默认显示在图标的右边，可以通过“标题位置”属性改变标题和图标的相对位置，例如将“标题

位置”属性设置为“Bottom”后：



需要注意的是，每个按钮都要分别设置“显示标题”和“标题位置”属性。

4.9.4.9 使用快捷菜单

给控件指定快捷菜单

如果希望右击某个控件，能弹出一个快捷菜单，可以先设计一个快捷菜单，将其指定给此控件。例如要给控件 TextBox1 指定快捷菜单，可以在窗口的 AfterLoad 事件加上代码：

```
e.Form.Controls("TextBox1").ContextStrip = e.Form.Strips("快捷菜单 1")
```

在指定位置显示快捷菜单

所有控件都有一个 ShowContextStrip 方法，用于在该控件的指定位置显示指定的快捷菜单，语法为：

```
ShowContextStrip(x,y,Strip)
```

x： 水平位置，控件左上角为原点。
y： 垂直位置，控件左上角为原点。
Strip： 要显示的快捷菜单。

例如窗口中有一个按钮，希望单击这个按钮，能出现一个快捷菜单：



可以将该按钮的 Click 事件代码设置为：

```
With e.Sender
    .ShowContextStrip(0, .Height, e.Form.Strips("快捷菜单1"))
End With
```

模拟SplitButton

假定希望做一个下图所示的按钮，单击按钮左侧直接增加一行，单击按钮右侧则显示一个菜单，用于选择是增加一行还是增加多行:



为实现这样的功能，我们可以设计两个并排的按钮，左侧按钮用于增加行，右侧按钮用于弹出下来菜单，由于两个按钮紧挨在一起，实际使用的时候看起来就像是一个按钮的两个分区。

给 Table 指定快捷菜单

假定窗口中有一个名为 Table1 的 Table 控件，希望在窗口中重新设计一个快捷菜单，代替系统默认的快捷菜单。
首先我们要取消系统默认的快捷菜单，在对应的表事件 BeforeShowContextMenu 加上代码：

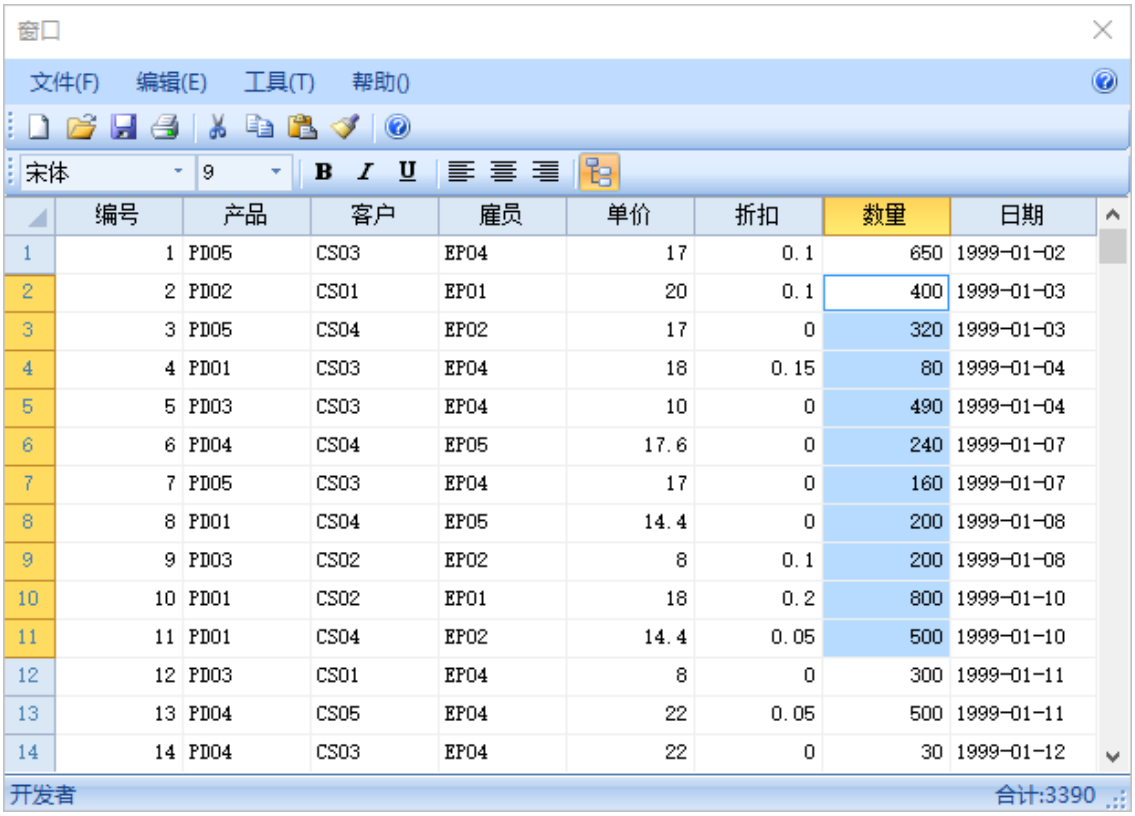
```
e.Cancel = True
```

然后在窗口的 AfterLoad 事件中加上代码：

```
e.Form.Controls("Table1").ContextStrip = e.Form.Strips("快捷菜单 1")
```

4.9.4.10 使用状态栏

假定有个窗口，窗口中有一个 SQLTable，底部有个状态栏，希望：
1、打开窗口后，在状态栏左侧显示当前用户名。
2、在 SQLTable 中选择某数值列的多个单元格后，能在状态栏右侧显示合计值。



	编号	产品	客户	雇员	单价	折扣	数量	日期
1	1	PD05	CS03	EP04	17	0.1	650	1999-01-02
2	2	PD02	CS01	EP01	20	0.1	400	1999-01-03
3	3	PD05	CS04	EP02	17	0	320	1999-01-03
4	4	PD01	CS03	EP04	18	0.15	80	1999-01-04
5	5	PD03	CS03	EP04	10	0	490	1999-01-04
6	6	PD04	CS04	EP05	17.6	0	240	1999-01-07
7	7	PD05	CS03	EP04	17	0	160	1999-01-07
8	8	PD01	CS04	EP05	14.4	0	200	1999-01-08
9	9	PD03	CS02	EP02	8	0.1	200	1999-01-08
10	10	PD01	CS02	EP01	18	0.2	800	1999-01-10
11	11	PD01	CS04	EP02	14.4	0.05	500	1999-01-10
12	12	PD03	CS01	EP04	8	0	300	1999-01-11
13	13	PD04	CS05	EP04	22	0.05	500	1999-01-11
14	14	PD04	CS03	EP04	22	0	30	1999-01-12

设计步骤：

1、在菜单设计器中，新增一个状态栏

2、在状态栏增加一个项目，类型为 Label，名称为“User”，用于显示当前用户名

3、在状态栏增加一个项目，类型为 Label，名称为“Stat”，位置属性设置为"Right"，用于显示合计值。

4、在窗口的 AfterLoad 事件加上代码：

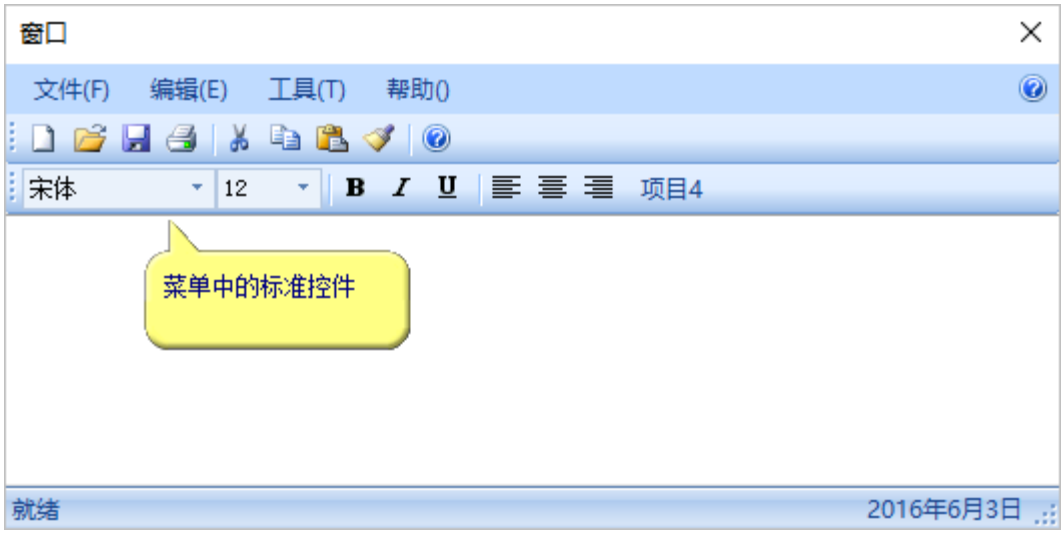
```
e.Form.Strips("状态栏").Items("User").Text =  User.Name
```

5、在 SQLTable 的 AfterSelRangeChange 事件加上代码：

```
If e.Form IsNot Nothing Then
    Dim it As Winform.StripItem = e.Form.Strips("状态栏").Items("Stat")
    Dim t As Table = e.Table
    it.Text = ""
    If  t.Cols(t.ColSel).IsNumeric Then '如果当前列是数值列
       it.Text = "合计:" & t.Aggregate(AggregateEnum.Sum,t.TopRow,t.ColSel,t.BottomRow,t.ColSel)
    End If
End If
```

4.9.4.11 插入标准控件

菜单可以插入标准控件，例如下图中选择字体名称和字体大小的两个下拉列表框就是标准控件来的：



我们没有办法直接在菜单设计器中插入标准控件，只能通过代码，以上图的例子为例，步骤：
1、在窗口中插入一个 FontPicker 控件，名称为 FontPicker1，设置好相关属性和事件。
2、在窗口中插入一个 ComboBox 控件，名称为 ComboBox1，设置好相关属性和事件。
3、在窗口的 AfterLoad 事件中加上代码：

```
e.Form.Strips("工具栏2").Items.InsertControl("FontName",e.Form.Controls("FontPicker1"),0)
e.Form.Strips("工具栏2").Items.InsertControl("FontSize",e.Form.Controls("Combobox1"),1)
```

这里假定工具栏的名称为"工具栏2"。

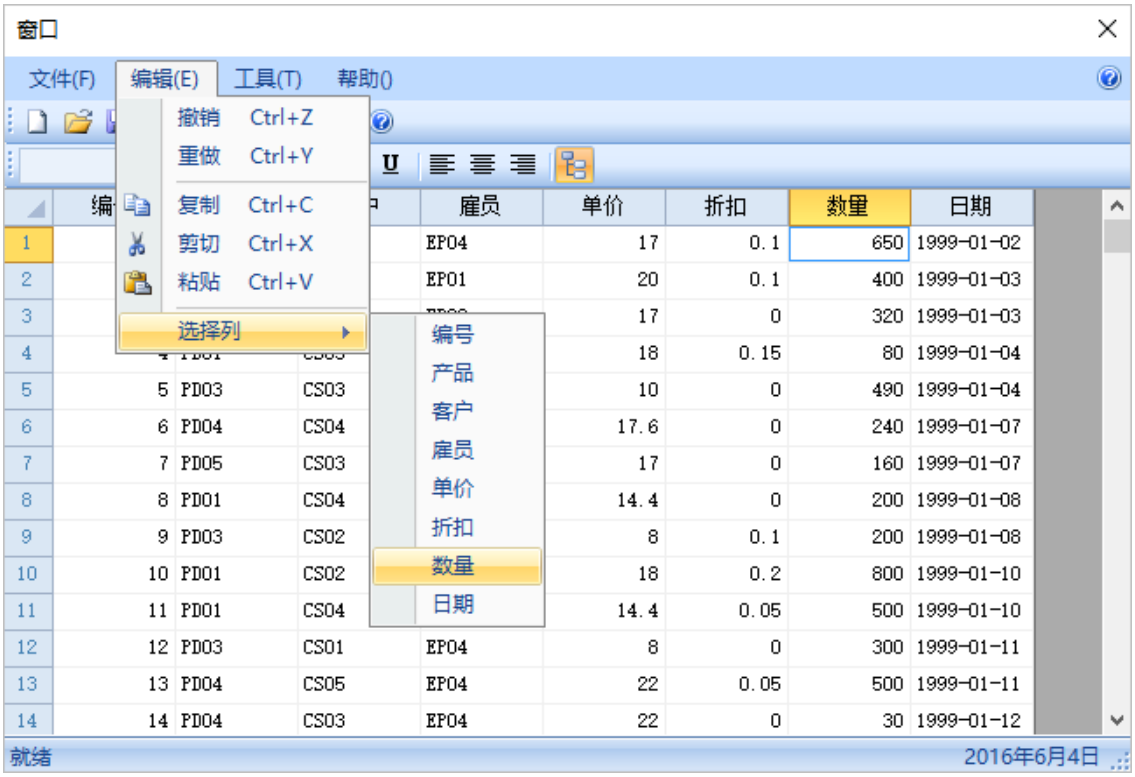
InsertControl的语法：

InsertControl(ItemName,Control,Index)

ItemName：项目名称，控件是作为一个特殊项目插入到菜单中的，所以要指定项目名称。
Control： 要插入的控件。
Index： 插入位置。

4.9.4.12 动态生成菜单项目

假定在窗口的编辑菜单中，有一个选择列的命令，单击该命令能自动列出当前表所有的列名，单击某个列名，能自动定位到对应的列：



设计步骤：

1、在窗口的菜单事件 ItemMouseEnter 中加上代码：

```
If e.StripItem.Text = "选择列" Then
    e.StripItem.Items.Clear() '清除原来的项目
    For Each c As Col In CurrentTable.Cols
        e.StripItem.Items.Add("Col_" & c.Name, c.Name, "", StripItemTypeEnum.MenuItem)
    Next
End If
```

ItemMouseEnter事件在鼠标进入菜单项目时执行，这样鼠标每次进入“选择列”项目时，就会自动针对当前表的所有列，分别生成一个菜单项目。

Items的Add方法语法是：

Add(Name, Text, ImageFile, Type, Delimiter)

Name：　指定项目名称
Text：　指定项目标题
ImageFile：　指定项目的图标文件
Type：　指定项目的类型，StripItemTypeEnum类型，包括MenuItem、Button、 Label、SplitButton四个候选值。
Delimiter：　可选参数，逻辑型，是否在该项目前面插入一个分割条。

提示：为了便于编写代码，动态增加的项目通常应该统一的约定字符开头，我们这里约定的开头字符是“Col_”

2、在窗口的菜单事件ItemClick中加上代码：

```
If e.StripItem.Name.StartsWith("Col_") Then '如果菜单项目用约定的字符开头
    Dim idx As Integer = CurrentTable.Cols(e.StripItem.Text).Index
    CurrentTable.Select(CurrentTable.RowSel,idx)
End If
```

任务完成，整个过程不过10行代码而已。

4.9.4.13 编程参考

4.9.4.13.1 Strip

WinForm.Strip 表示窗口菜单，窗口的主菜单、工具栏、快捷菜单和状态栏，其类型都是WinForm.Strip。
Form 有一个 Strips 属性，包括窗口的所有菜单，通过这个集合可以获得指定名称的菜单。

例如：

```
Dim Strip As WinForm.Strip = e.Form.Strips("状态栏")
Strip.Visible = False
```

Strip 的属性有：

属性	说明
Name	字符型，返回菜单名称
Visible	逻辑型，返回或设置菜单是否可见
Enabled	逻辑型，返回或设置菜单是否可用
Items	StripItem 集合，包括此菜单的所有菜单项目。
BaseControl	返回对应的底层.net 对象，一般用户用不着。

Strip 的方法有：

属性	说明
StopRedraw	停止绘制菜单，用代码在菜单大量增加项目前，可以先执行此方法。
ResumeRedraw	恢复绘制菜单，用 StopRedraw 停止绘制菜单

	后，必须执行ResumeRedraw，菜单才能恢复正常显示。
Close	关闭菜单，仅对快捷菜单有效。 例如希望单击快捷菜单中中某命令，能显示一个模式窗口，通常应该这样编写代码： e.Strip.Close() Forms("窗口1").Show() 否则直到模式窗口关闭，快捷菜单都会一直显示。

4.9.4.13.2 Strips

Form 有个 Strips 属性，这是一个集合，返回窗口的所有菜单，包括主菜单、工具栏、快捷菜单和状态栏。
通过 Strips 集合，可以获取指定名称的菜单，例如：

```
Dim Strip As WinForm.Strip = e.Form.Strips("工具栏 1")
Strip.Visible = False
```

Strips 的主要成员有：

- Count
  属性，整数型，返回窗口菜单的数量

- Contains
  方法，判断窗口是否存在指定名称的菜单，例如：

```
If e.Form.Strips.Contains("工具栏 1") Then
    MessageBox.Show("工具栏 1 已经存在")
End If
```

- Remove
  删除指定名称的菜单，例如：

```
If e.Form.Strips.Contains("工具栏 1") Then
    e.Form.Strips.Remove("工具栏 1")
```

```
End If
```

- Clear
  清除所有菜单，例如：

  ```
  e.Form.Strips.Clear()
  ```

- Add
  增加菜单，语法：

  Add(Name,Type)

  Name：新增菜单的名称
  Type：新增菜单的类型，StripTypeEnum 型枚举，可选值有 MainMenu，ToolBar，StatusBar，ContextMenu

  例如在窗口的 AfterLoad 事件加上代码：

  ```
  Dim Strip As WinForm.Strip
  Strip = e.Form.Strips.Add("工具栏1", StripTypeEnum.ToolBar)
  Strip.Items.Add("New", "新建", "new.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Open", "打开", "Open.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Save", "保存", "Save.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Print", "打印", "Print.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Separator1", "分割条", "", StripItemTypeEnum.Separator)
  Strip.Items.Add("Cut", "剪切", "Cut.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Copy", "复制", "Copy.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Brush", "格式刷", "Brush.ico", StripItemTypeEnum.Button)
  Strip.Items.Add("Separator2", "分割条", "", StripItemTypeEnum.Separator)
  Strip.Items.Add("Help", "帮助", "Help.ico", StripItemTypeEnum.Button)
  ```

  可以自动生成下图所示的工具栏：

  提示：相关图标文件要事先复制到项目的 Images 子目录下。

4.9.4.13.3 StripItem

WinForm.StripItem 表示菜单项目，MemuItem，Button，SplitButton 和 Label 等菜单项目的类型同为 WinForm.StripItem。
Strip 有个 Items 属性，这是一个集合，包括该菜单中的所有项目，通过该集合，可以获取菜单中指定名称的项目，例如：

```
Dim sit As WinForm.StripItem = e.Form.Strips("工具栏1").Items("Print")
sit.Enabled = False
```

再例如，禁用工具栏 1 中除保存按钮之外的所有按钮：

```
For Each sit As WinForm.StripItem In e.Form.Strips("工具栏1").Items
    If sit.Name <> "Save" Then
        sit.Enabled = False
    End If
Next
```

StripItem的属性：

属性	说明
Name	字符型，返回项目名称
Text	字符型，返回或设置项目标题
ToolTip	字符型，返回或设置项目的提示内容
Visible	逻辑型，返回或设置项目是否可见
Enabled	逻辑型，返回或设置项目是否可用
Checked	逻辑型，返回或设置是否勾选项目，仅对 MenuItem 和 Button 有效。
Index	整数型，返回项目的位置
ParentItem	返回父菜单项目，如果不存在父项目，则返回 Nothing。
Items	StripItem 集合，返回 MenuItem 或 SplitButton 的子项目集合
DisplayText	逻辑型，是否显示标题。
TextPosition	返回或设置标题位置，TextPositionEnum 枚举，候选值有 Right 和 Bottom。
ShortcutKeys	返回或设置项目的快捷键，仅对 MenuItem 有效。
DockToRight	逻辑型，项目是否靠右停放。

Image	Image 类型，返回或设置项目的图标。
BaseControl	返回对应的底层.net 对象，一般用户用不着。

StripItem 只有一个方法：

- PerformClick

  模拟单击此菜单项目

4.9.4.13.4 StripItems

Strip 有一个 Items 属性，这是一个集合，包括该菜单的所有项目。
StripItem 也有一个 Items 属性，这也是一个集合，包括所有子菜单项目，只有 MenuItem 和 SplitButton 类型的 StripItem，才可以包括子项目，其 Items 属性才有意义。
Items 的主要成员有：

- Count
  属性，整数型，返回菜单项目的数量。

- Contains
  方法，判断是否存在指定名称的菜单项目。

- Remove
  方法，用于删除指定名称的菜单项目

- Clear
  清除所有菜单项目

- Add
  增加菜单项目，语法：

Add(Name, Text, ImageFile, Type)

Name：　指定菜单项目名称
Text：　指定 菜单项目标题
ImageFile：　指定菜单项目的图标文件
Type：　指定菜单项目的类型，StripItemTypeEnum类型，包括MenuItem、Button、 Label、SplitButton、Separator五个候选值。

- InsertControl
  用于插入标准控件，语法：

InsertControl(ItemName,Control,Index)

ItemName：项目名称，控件是作为一个特殊项目插入到菜单中的，所以要指定项目名称。
Control： 要插入的控件。
Index： 插入位置。

参考：插入标准控件

示例一

删除文件菜单中的打印命令：

```
If e.Form.Strips("主菜单").Items("File").Items.Contains("Print") Then
    e.Form.Strips("主菜单").Items("File").Items.Remove("Print")
End If
```

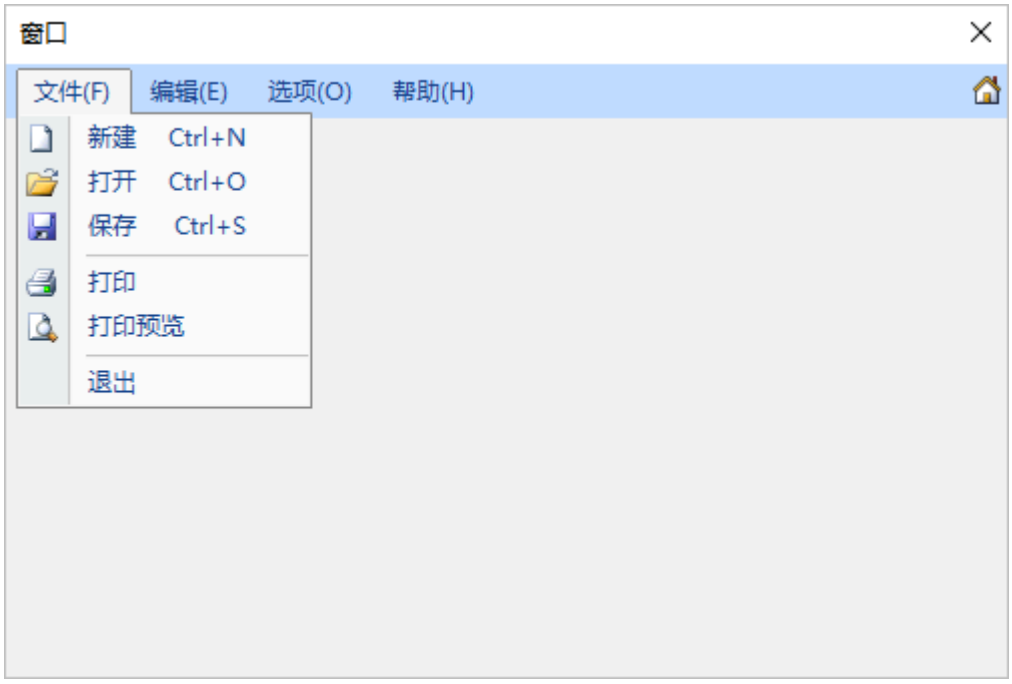
示例二

完全不用窗口的菜单设计器，直接用代码动态创建一个简单的主菜单，在窗口的AfterLoad事件加上代码：

```
Dim mnu As Winform.Strip
Dim itm As WinForm.StripItem
mnu = e.Form.Strips.Add("主菜单",StripTypeEnum.MainMenu)
mnu.Items.Add("File","文件(&F)","",StripItemTypeEnum.MenuItem)
mnu.Items.Add("Edit","编辑(&E)","",StripItemTypeEnum.MenuItem)
mnu.Items.Add("Option","选项(&O)","",StripItemTypeEnum.MenuItem)
mnu.Items.Add("Help","帮助(&H)","",StripItemTypeEnum.MenuItem)
'在主菜单右侧增加一个按钮
mnu.Items.Add("Home","主页","home.ico",StripItemTypeEnum.Button)
mnu.Items("Home").DisplayText = False
mnu.Items("Home").DockToRight = True '靠右停放
'为文件菜单增加项目
itm = mnu.Items("File")
itm.Items.Add("New","新建","new.ico",StripItemTypeEnum.MenuItem)
itm.Items.Add("Open","打开","Open.ico",StripItemTypeEnum.MenuItem)
itm.Items.Add("Save","保存","Save.ico",StripItemTypeEnum.MenuItem)
itm.Items.Add("Separator1", "分割条", "", StripItemTypeEnum.Separator)
itm.Items.Add("Print","打印","Print.ico",StripItemTypeEnum.MenuItem)
itm.Items.Add("Preview","打印预览","Preview.ico",StripItemTypeEnum.MenuItem)
itm.Items.Add("Separator2", "分割条", "", StripItemTypeEnum.Separator)
itm.Items.Add("Exit","退出","",StripItemTypeEnum.MenuItem)
'为常用项目增加快捷键
itm.Items("New").ShortcutKeys = Keys.Control Or Keys.N
itm.Items("Open").ShortcutKeys = Keys.Control Or Keys.O
```

```
itm.Items("Save").ShortcutKeys = Keys.Control Or Keys.S
'这里只给文件菜单加上了内容,其它菜单的内容请自行编写.
```

这是生成的主菜单：



提示：相关图标文件要事先复制到项目的Images子目录下。

4.9.4.13.5 菜单事件

窗口菜单事件有四个，分别为：
ItemClick：　单击菜单项目后执行。
ItemMouseEnter：鼠标进入菜单项目后执行。
ItemMouveLeave：鼠标离开菜单项目后执行。
ItemMouseHover：鼠标停留在菜单项目上时执行。
所有菜单事件的 e 参数都是相同的：
Form：　触发事件的窗口
Strip：　触发事件的菜单
StripItem：触发是事件的菜单项目
参考：
编写菜单事件代码

动态生成菜单项目

4.9.5 动态增加控件

4.9.5.1 动态增加控件

普通用户，可忽略本节内容。

Form 提供了四个方法，用于动态增加控件。

CreateControl

用于创建控件，并返回所创建的控件。

语法为：

CreateControl(Name,ControlType)

Name：　　控件名称

ControlType：控件类型，ControTypeEnum 型枚举，包括以下可选值：

Button

Chart

CheckBox

CheckedComboBox

CheckedListBox

ColorLabel

ComboBox

DateTimePicker

GroupBox

Label

Line

ListBox

NavBar

NumericComboBox

Panel

PictureBox

ProgressBar

RadioButton

SplitContainer

TabControl

TextBox

WebBrowser

Painter

BarCode

FileManager

PictureViewer

RecordGrid

AddControl

用于将控件加入到窗口中，语法为

AddControl(Control)

Control：要加入到窗口的控件。

RemoveControl

用于从窗口删除指定名称的控件，语法为：

RemoveControl(Name)

Name：字符型，要删除的控件地名称。

ExistControl

返回一个逻辑值，用于判断窗口是否存在指定名称的控件，语法为：

ExistControl(Name)

Name：字符型，要判断的控件名称

示例一

例如希望单击某个按钮，能够向窗口中加入一个标签，可以将按钮的 Click 代码设为：

```
Dim lbl As WinForm.Label
lbl = e.Form.CreateControl("lable1", ControlTypeEnum.Label)
lbl.Text = "Foxtable"
lbl.Left = 100
lbl.Top = 100
e.Form.AddControl(lbl)
```

示例二

可以利用 BindingField 给动态增加的控件设置绑定字段，例如:

```
Dim txt As WinForm.TextBox
txt = e.Form.CreateControl("TextBox1", ControlTypeEnum.TextBox)
txt.Left = 100
txt.Top = 100
e.Form.AddControl(txt)
txt.BindingField = "订单.数量" '必须先将控件加入到窗口或容器中，才能设置绑定字段
```

上面的代码将创建一个文本框，并将其绑定到订单表的数量列。

提示：动态生成控件的时候，必须先将控件加入到窗口或容器中，才能设置其绑定字段，切记切记。

4.9.5.2 动态添加 Table

窗口中的 Table 不能用 CreateControl 来创建，Form 使用另外三个方法来创建 Table。

CreateSQLTable

用于创建 SQLTable 类型的 Table，语法：

CreateSQLTable(Name, SqlText , ConnectionName)

参数说明:

Name：　　控件名称
SQLText：　　Select 语句
ConnectionName：数据源名称

CreateSQLQuery

用于创建 SQLQuery 类型的 Table，语法：
CreateSQLQuery(Name, SqlText , ConnectionName)
参数说明:
Name：　　控件名称
SQLText：　　Select 语句
ConnectionName：数据源名称

CreateTable

用于创建 Normal 型 Table，语法:
CreateTable(Name, TableName, IsCopy)
Name：　　控件名称
TableName：　数据表名称
IsCopy：　　逻辑型，是否作为副本，设为 True 作为副本。

示例一

创建一个 SQLTable 型 Table 控件，从 nwind 数据源中加载产品表的数据：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateSQLTable("Table1", "Select * From {产品}","nwind")
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

示例二

创建一个 SQLQuery 型 Table 控件，加载内部数据源中的产品表：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateSQLQuery("Table1", "Select * From {产品}","")
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

示例三

创建一个 Normal 型 Table 控件，将订单表移到窗口中：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateTable("Table1", "订单", False)
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

示例四

创建订单表的副本：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateTable("Table1", "订单", True)
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

4.9.5.3 动态增加 RecordGrid

```
rgd As WinForm.RecordGrid
rgd = e.Form.CreateControl("lable1", ControlTypeEnum.RecordGrid)
Tables("员工").Cols("邮政编码").ShortCaption = "邮编" ' 指定邮政编码列的缩写标题
Tables("员工").Cols("照片").ShowInRecordGrid = False '照片列不在记录窗口显示
rgd.Table = Tables("员工") '绑定到员工表
rgd.Build() '生成记录窗口
rgd.SetBounds(10,10,200,300) '指定位置和尺寸
```

4.9.5.4 将控件加入到容器中

GroupBox、Panel、NavPage、TabPage、SplitterPanel 等类型也有 AddControl 和 RemoveControl 方法，也可以通过代码动态地将控件加入其中。
至于 CreateControl 和 ExistControl 方法，只有窗口(Form)才有。

示例一

向 Panel 中加入一个文本框：

```
Dim txt As WinForm.TextBox
txt = e.Form.CreateControl("TextBox1", ControlTypeEnum.TextBox)
txt.Left = 100
txt.Top = 100
e.Form.Controls("Panel1").AddControl(txt)
txt.BindingField = "订单.数量"
```

提示：动态生成控件的时候，必须先将控件加入到窗口或容器中，才能设置其绑定字段，切记切记。

示例二

向页面集合的第一个页面加入一个标签：

```
Dim txt As WinForm.TextBox
txt = e.Form.CreateControl("TextBox1", ControlTypeEnum.TextBox)
txt.Left = 100
txt.Top = 100
e.Form.Controls("TabControl1").Tabpages(0).AddControl(txt)
txt.BindingField = "订单.数量"
```

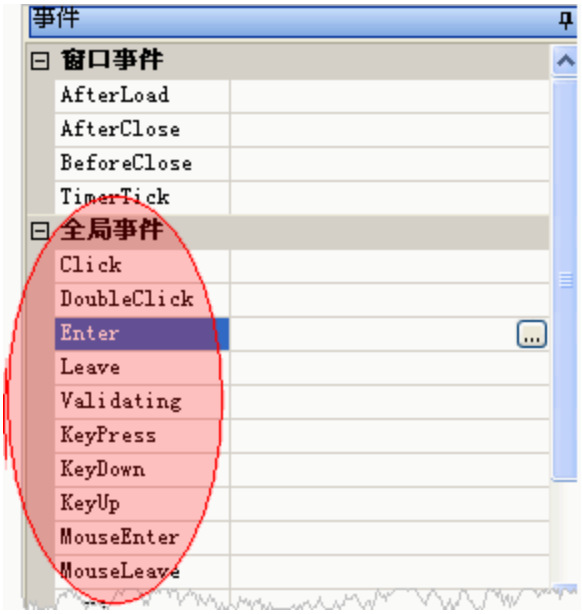
示例三

向分区面板的第二个分区加入一个 Table：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateTable("Table1", "订单", True)
tbl.SetBounds(10, 10, 600, 400)
e.Form.Controls("SplitContainer1").Panel2.AddControl(tbl)
```

4.9.5.5 全局事件

当我们选择窗口的事件页时，会发现除了窗口事件外，还有一组全局事件：



全局事件不是窗口事件，而是控件事件。

全局事件对所有的控件有效，例如你在窗口单击任何一个控件，都会触发全局事件 Click。

全局事件的作用有两个：

1、集中编写代码

例如窗口有四个按钮，分别是上一条、下一条、第一条、最末条，假定这四个按钮的名称分别为：btnPrev、btnNext、btnFirst、btnLast

如果按照之前的方法，我们需要分别为这四个按钮编写代码，代码分布在四个位置。

如果采用全局代码，只需在全局事件 Click 中加入以下代码即可：

```
Select Case e.Sender.Name
    Case "btnPrev"
        CurrentTable.Position = CurrentTable.Position - 1
    Case "btnNext"
        CurrentTable.Position = CurrentTable.Position + 1
    Case "btnFirst"
        CurrentTable.Position = 0
    Case "btnLast"
        CurrentTable.Position = CurrentTable.Rows.Count -1
End Select
```

很难说分别编写和统一编写哪个更合理，这个纯粹看开发者的习惯。

提示：如果要使用全局事件，最好为每一个控件取一个有意义的名称，例如在维护的过程中，btnNext 显然比 Button1 更容易让人理解。

2、为动态增加的控件编写事件代码

控件本身的事件代码只能在设计窗口的时候编写，所以如果需要在运行过程中动态增加控件，且需要为这些控件编写事件代码，那么就只能使用全局事件了。

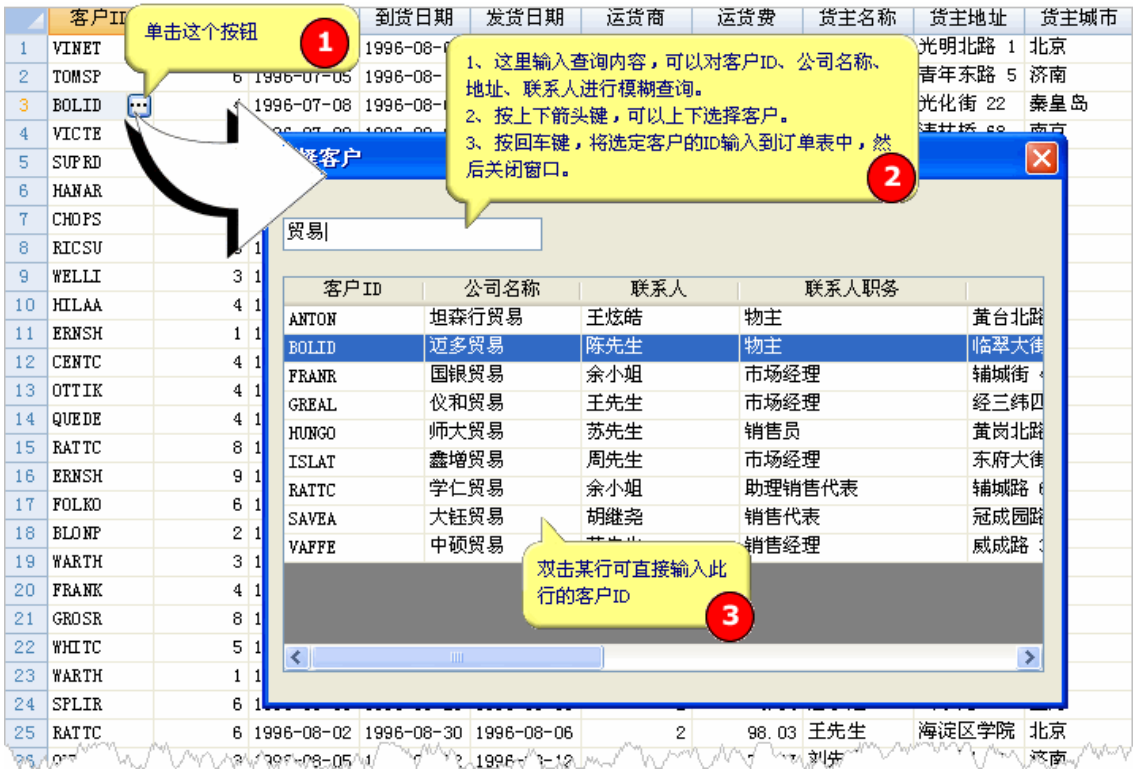
提示：如果是动态增加的 Table，就需要使用全局表事件。

4.9.6 自定义录入界面

4.9.6.1 自定义录入界面之一

本节的内容可以参考 CaseStudy 目录下的文件：自定义录入界面之一.Table

假定客户表已经输入所有的客户资料，订单表有一个客户 ID 列，我们可以利用列表项目来输入客户 ID，也可以利用数据字典来输入，前者利用输入助手可以模糊筛选，后者则可以同时显示多列数据，但是显然两者都不够完美。能不能自己设计一个输入界面， 就像下图所示一样，做到既可以模糊查询，又可以显示多列数据呢：



完全可以，接下来我们就看看如何来实现。

1、首先在订单表新增一个窗口，窗口类型为模式，窗口插入一个 TextBox 和一个 Table 控件，Table 控件的“作为副本”属性设置为 True，“允许编辑”属性设置为 False，并绑定到客户表。

2、将 TextBox1 的 TextChanged 事件代码设为:

```
Dim txt As String = e.Form.Controls("TextBox1").Text
Dim tbl As Table = Tables("窗口 1_Table1")
If txt = "" Then
```

```
    tbl.Filter = ""
Else
    txt = "'%" & txt & "%'"
    tbl.Filter = "客户 ID Like " & txt & " Or 公司名称 Like " & txt & " Or 地址 Like " & txt & " Or 联系人 Like " & txt
End If
```

这样我们在文本框中输入任何内容，Table 就会自动进行模糊筛选，显示客户 ID、公司名称、地址、联系人这四列中，任何一列包括输入内容的行。

注意只有字符列才能使用 Like 进行模糊筛选，其它列类型是不可以的。

3、将 TextBox1 的 KeyDown 事件代码设为：

```
Dim tbl As Table = Tables("窗口 1_Table1")
If e.KeyCode = Keys.Up Then
    tbl.Position = tbl.Position - 1
    e.Cancel = True
ElseIf e.KeyCode = Keys.Down Then
    tbl.Position = tbl.Position + 1
    e.Cancel = True
ElseIf e.KeyCode = Keys.Enter Then
    If tbl.Current IsNot Nothing Then
        Tables("订单").Current("客户 ID") = tbl.Current("客户 ID")
    End if
    e.Cancel = True
    e.Form.Close()
End If
```

这样在文本框中按上下箭头键，可以在 Table 中选择不同的行，按回车键会将选定行的客户 ID 输入到订单表的当前行中，并关闭窗口。

为了避免按上下箭头键的时候，文本框的光标会移动，下面的代码是必须的：

```
e.Cancel = True
```

4、将 Table 控件的 DoubleClick 事件设置为：

```
Dim tbl As Table = Tables("窗口 1_Table1")
If tbl.Current IsNot Nothing Then
    Tables("订单").Current("客户 ID") = tbl.Current("客户 ID")
End If
Forms("窗口 1").Close()
```

这样在窗口的 Table 中双击某行，即可将该行的客户 ID 输入到订单表的当前行中，并关闭窗口。

提示：由于 Table 的 DoubleClick 事件的 e 参数不包括 Form 属性，所以代码中不能使用 e.Form 来引用窗口，只能直接使用窗口名：Forms("窗口 1")。

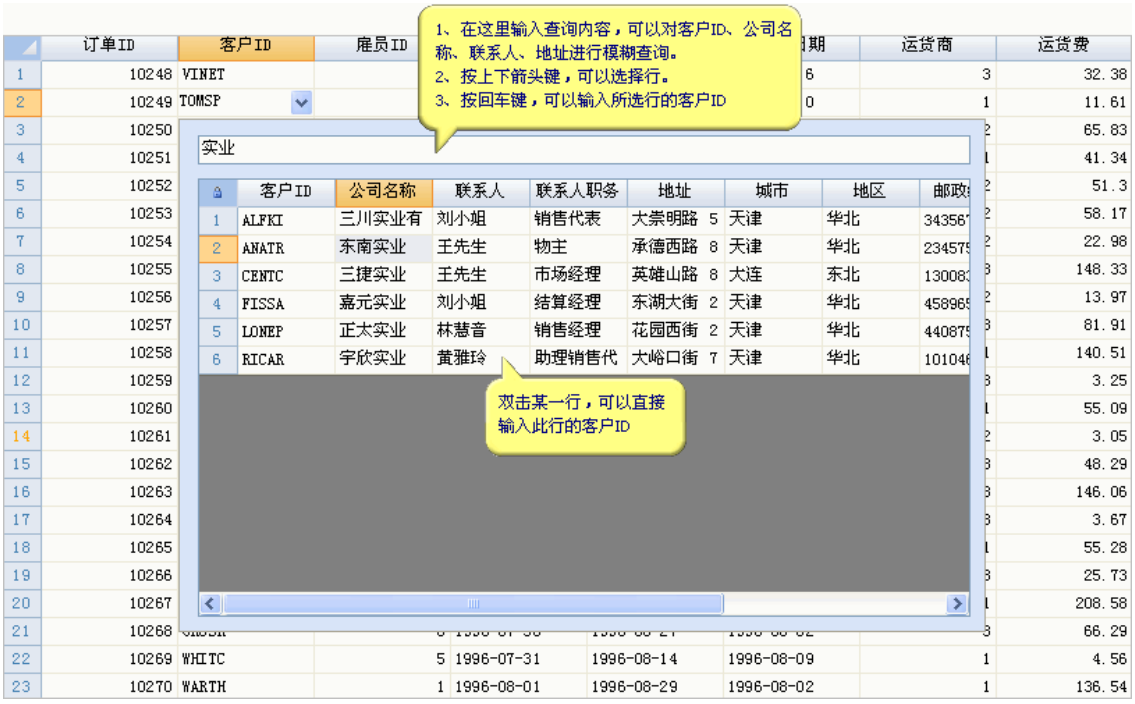
5、将订单表客户 ID 列的列表项目设为“|...”最后并将订单表的 CellButtonClick 事件设为：

```
If e.Col.Name = "客户 Id" Then
    e.Cancel = True
    Forms("窗口 1").Open()
End If
```

下图是我们最终的设计效果，显然用这个窗口来输入客户 ID，其便利性不是列表项目和数据字典能比拟的：

4.9.6.2 自定义录入界面之二

本节的内容可以参考 CaseStudy 目录下的文件：自定义录入界面之二.Table
上一节内容讲述了如何自定义录入界面来实现快速的模糊查询录入，使用的是模式窗口。
本节来完成同样的任务，不同的是使用下拉窗口来实现：



显然用这个下拉窗口来输入客户 ID，其便利性不是列表项目和数据字典能比拟的，下面我们来讲述设计步骤：

1、首先在订单表新增一个窗口，窗口类型为"DropDownForm"，窗口插入一个 TextBox 和一个 Table 控件，Table 控件的“作为副本”属性设置为 True，“允许编辑”属性设置为 False，并绑定到客户表。

2、将 TextBox1 的 TextChanged 事件代码设为:

```
Dim txt As String = e.Form.Controls("TextBox1").Text
Dim tbl As Table = Tables("窗口 1_Table1")
If txt = "" Then
    tbl.Filter = ""
Else
    txt = "'%" & txt & "%'"
    tbl.Filter = "客户 ID Like " & txt & " Or 公司名称 Like " & txt & " Or 地址 Like " & txt & " Or 联系人 Like " & txt
End If
```

这样我们在文本框中输入任何内容，Table 就会自动进行模糊筛选，显示客户 ID、公司名称、地址、联系人这四列中，任何一列包括输入内容的行。

3、将 TextBox1 的 KeyDown 事件代码设为：

```
Dim tbl As Table = Tables("窗口 1_Table1")
If e.KeyCode = Keys.Up Then
    tbl.Position = tbl.Position - 1
```

```
    e.Cancel = True
Elseif e.KeyCode = Keys.Down Then
    tbl.Position = tbl.Position + 1
    e.Cancel = True
Elseif e.KeyCode = Keys.Enter Then
    If tbl.Current IsNot Nothing Then
        e.Form.DropDownBox.Value = tbl.Current("客户 ID")
    End If
    e.Cancel = True
    e.Form.DropDownBox.CloseDropDown()
End If
```

这样在文本框中按上下箭头键，可以在 Table 中选择不同的行，按回车键会将选定行的客户 ID 输入到订单表的当前行中，并关闭下拉窗口

为了避免按上下箭头键的时候，文本框的光标会移动，下面的代码是必须的：

```
e.Cancel = True
```

4、将 Table 控件的 DoubleClick 事件设置为：

```
Dim tbl As Table = Tables("窗口1_Table1")
If tbl.Current IsNot Nothing Then
    e.Form.DropDownBox.Value = tbl.Current("客户ID")
End If
e.Form.DropDownBox.CloseDropDown()
```

这样在窗口的 Table 中双击某行，即可将该行的客户 ID 输入到订单表的当前行中，并关闭下拉窗口。

5、将窗口的 DropDownClosed 事件代码设置为：

```
If e.Form.DropTable IsNot Nothing Then '如果是通过表下拉的
    e.Form.DropTable.FinishEditing()
Else '如果是通过窗口下拉的
    e.Form.DropDownBox.WriteValue()
End If
```

上面的不是必须的，加上这个代码的好处是下拉窗口关闭后，输入的数据即时生效。

6、在项目事件 AfterOpenProject 事件中加入代码：

```
Tables("订单").Cols("客户 ID").DropForm = "窗口 1"
```

4.9.6.3 自定义录入界面之三

本节的内容可以参考 CaseStudy 目录下的文件：自定义录入界面之三.Table

这一节还是完成同样的任务，不过要更加“专业”一点，可以直接在单元格中输入筛选内容，进行模糊筛选，而不是在下拉窗口中：



设计步骤

一、设计下拉窗口

1、首先在订单表新增一个窗口，名称为"窗口 1"，窗口类型为"DropDownForm"，窗口插入一个 Table 控件，Table 控件的“作为副本”属性设置为 True，“允许编辑”属性设置为 False，并绑定到客户表。

2、将窗口的 DropDownOpened 事件代码设置为:

```
Dim txt As String = e.Form.DropDownBox.Text
Dim tbl As Table = Tables("窗口 1_Table1")
If txt = "" Then
    tbl.Filter = ""
Else
    txt = "'%" & txt & "%'"
    tbl.Filter = "客户 ID Like " & txt & " Or 公司名称 Like " & txt & " Or 地址 Like " & txt & " Or 联系人 Like " & txt
End If
e.Form.DropDownBox.Select() '将输入焦点返回下拉列表框
```

这样下拉窗口显示后，会自动根据下拉列表框的内容，筛选出可能的客户，并将输入焦点返回到下拉列表框。

3、将 Table 控件的 DoubleClick 事件代码设置为：

```
Forms("窗口 1").DropDownBox.CloseDropDown()
```

这样在下拉窗口双击某个客户，就会关闭下拉窗口。

4、将 Table 控件的 KeyDown 事件代码设置为：

```
If e.KeyCode = Keys.Enter Then
    Forms("窗口 1").DropDownBox.CloseDropdown()
End If
```

这样在下拉窗口中选择某个客户，按回车键，也会关闭下拉窗口。

5、将窗口的 DropDownClosed 事件代码设置为：

```
If e.Selected Then '如果选择了值
    Dim tbl As Table = Tables("窗口1_Table1")
    If tbl.Current IsNot Nothing Then
```

```
        e.Form.DropDownBox.Value = tbl.Current("客户ID")
        If e.Form.DropTable IsNot Nothing Then '如果是通过表下拉的
            e.Form.DropTable.FinishEditing()
        Else '如果是通过窗口下拉的
            e.Form.DropDownBox.WriteValue()
        End If
    End If
End If
```

这样关闭下拉窗口后，可以自动将选定客户的客户 ID 输入到下拉列表框中。

如果除了客户 ID 列，还要同时从客户表选择其它多列内容输入到订单表，可以将 DropDownClosed 事件代码改为：

```
If e.Selected Then '如果选择了值
    Dim tbl As Table = Tables("窗口 1_Table1")
    If tbl.Current IsNot Nothing Then
        e.Form.DropDownBox.Value = tbl.Current("客户 ID")
        Tables("订单").Current("其它列 1") = tbl.Current("其它列 1")
        Tables("订单").Current("其它列 2") = tbl.Current("其它列 2")
        Tables("订单").Current("其它列 3") = tbl.Current("其它列 3")
        If e.Form.DropTable IsNot Nothing Then '如果是通过表下拉的
            e.Form.DropTable.FinishEditing()
        Else '如果是通过窗口下拉的
            e.Form.DropDownBox.WriteValue()
        End If
    End If
End If
```

二、设计订单表事件代码

下拉窗口设计完成，现在设计订单表的事件代码。

1、将订单表的 KeyPressEdit 事件代码设置为：

```
If e.Col.Name = "客户 ID" Then '如果编辑的是客户 ID 列
    If e.Col.DroppedDown = False '且下拉窗口没有打开
        e.Col.OpenDropDown() '打开下拉窗口
    End If
End If
```

这样一旦用户在订单表的客户 ID 列输入内容，就会自动打开下拉窗口。

2、将订单表的 ChangeEdit 事件代码设置为：

```
If e.Col.Name = "客户 ID" Then '如果编辑的是客户 ID 列
    If e.Col.DroppedDown Then '如果下拉窗口已经打开
        Dim tbl As Table = Tables("窗口 1_Table1")
        If e.Text = "" Then '如果内容为空
            tbl.Filter = "" '显示所有客户
        Else '否则根据输入内容进行模糊筛选
            Dim txt As String = "'%" & e.Text & "%'"
            tbl.Filter = "客户 ID Like " & txt & " Or 公司名称 Like " & txt & " Or 地址 Like " & txt & "
Or 联系人 Like " & txt
```

```
        End If
    End If
End If
```

这样就能根据用户在客户 ID 列输入的内容，动态筛选出可能的客户。

3、将订单表的 KeyDownEdit 事件代码设置为：

```
If e.Col.Name = "客户 ID" Then '如果编辑的是客户 ID 列
    If e.Col.DroppedDown Then '如果下拉窗口已经打开
        Dim tbl As Table = Tables("窗口 1_Table1")
        If e.KeyCode = Keys.Up Then '如果按下的是上箭头按键
            tbl.Position = tbl.Position - 1 '向上移动一行
            e.Cancel = True
        ElseIf e.KeyCode = Keys.Down Then '如果按下的是下箭头按键
            tbl.Position = tbl.Position + 1 '向下移动一行
            e.Cancel = True
        End If
    End If
End If
```

这样用户在客户 ID 列输入内容的时候，可以按"上箭头"按键，选择上一个客户，按"下箭头"按键，选择下一个客户。

4、将订单表的 ValidateEdit 事件代码设置为：

```
If e.Col.DroppedDown Then '如果下拉窗口已经打开
    e.Col.CloseDropDown() '则关闭下拉窗口
End If
```

这样用户在编辑过程中，按Tab键离开单元格的时候，能自动关闭下拉窗口

5、最后在项目事件AfterOpenProject事件中加入代码：

```
Tables("订单").Cols("客户 ID").DropForm = "窗口 1"
```

4.9.6.4 自定义录入界面之四

本节的内容可以参考 CaseStudy 目录下的文件：自定义录入界面之四.Table

前面的自定义录入界面都是针对表的，这次针对窗口，可以在下拉输入框中输入内容进行模糊筛选，找出可能的客户供选择， 还可以直接在下拉窗口中增加或删除用户:



设计步骤

一、设计下拉窗口

1、首先在订单表新增一个窗口，名称为“窗口 1”，窗口类型为"DropDownForm"，窗口插入一个 Table 控件，Table 控件的“作为副本”属性设置为 True，“允许编辑”属性设置为 False，并绑定到客户表。

2、将窗口的 DropDownOpened 事件代码设置为:

```
Dim txt As String = e.Form.DropDownBox.Text
Dim tbl As Table = Tables("窗口 1_Table1")
If txt = "" Then
    tbl.Filter = ""
Else
    txt = "'%" & txt & "%'"
    tbl.Filter = "客户 ID Like " & txt & " Or 公司名称 Like " & txt & " Or 地址 Like " & txt & " Or 联系人 Like " & txt
End If
e.Form.DropDownBox.Select() '将输入焦点返回下拉列表框
```

这样下拉窗口显示后，会自动根据下拉列表框的内容，筛选出可能的客户，并将输入焦点返回到下拉列表框(这是很关键的一步)。

3、将 Table 控件的 DoubleClick 事件代码设置为：

```
Forms("窗口 1").DropDownBox.CloseDropDown()
```

这样在下拉窗口双击某个客户，就会关闭下拉窗口。

4、将 Table 控件的 KeyDown 事件代码设置为：

```
If e.KeyCode = Keys.Enter Then
    Forms("窗口 1").DropDownBox.CloseDropdown()
End If
```

这样在下拉窗口中选择某个客户，然后按回车键，也会关闭下拉窗口。

5、将窗口的 DropDownClosed 事件代码设置为：

```
If e.Selected Then '如果选择了值
    Dim tbl As Table = Tables("窗口1_Table1")
    If tbl.Current IsNot Nothing Then
        e.Form.DropDownBox.Value = tbl.Current("客户ID")
    End If
    If e.Form.DropTable IsNot Nothing Then '如果是通过表下拉的
        e.Form.DropTable.FinishEditing()
    Else '如果是通过窗口下拉的
        e.Form.DropDownBox.WriteValue()
    End If
End If
```

这样关闭下拉窗口后，可以自动将选定客户的客户 ID 输入到下拉列表框中。

6、将窗口的“增加客户”按钮的 Click 事件代码为：

```
e.Form.DropDownBox.CloseDropdown(False)
Dim r As Row = Tables("客户").AddNew()
Forms("客户编辑").Open()
r.Save()
e.Form.DropDownBox.Value = r("客户 ID")
```

这样单击“增加用户”按钮，可以增加一个用户，并打开“客户编辑”窗口编辑新增客户的资料，编辑完成后，会自动将新增客户的客户 ID 写入下拉列表中。

请使用上述代码之前，先在客户表建立一个“客户编辑”窗口，此窗口的设计就不再啰嗦了，记得将这个窗口的类型设置为模式窗口就行。

需要注意的是，上面的代码 CloseDropDown 方法使用了参数 False，这样关闭下拉窗口后，DropdownClosed 事件的 e 参数 Selected 等于 False，这样就只是简单地关闭窗口，不会向下拉列表框写入值。

7、将窗口的“删除用户”按钮的 Click 事件代码设置为：

```
Dim r As Row = Tables("窗口 1_Table1").Current
If r IsNot Nothing Then
    r.Delete()
End If
```

二、设计录入窗口

1、新建一个录入窗口，插入一个 DropDownBox 控件，将其下拉窗口设置为“窗口 1”。

2、将 DropDownBox 的 KeyPress 事件代码设置为：

```
Dim drp As WinForm.DropDownBox = e.Sender
If drp.DroppedDown = False '如果下拉窗口没有打开
    drp.OpenDropDown() '打开下拉窗口
End If
```

这样一旦开始在下拉列表框中输入内容，就能自动打开打开下拉窗口。

3、将 DropDownBox 的 TextChanged 事件代码设置为：

```
Dim drp As WinForm.DropDownBox = e.sender
If drp.DroppedDown Then '如果下拉窗口已经打开
    Dim tbl As Table = Tables("窗口 1_Table1")
```

```
    If drp.Text = "" Then '如果内容为空
        tbl.Filter = "" '显示所有客户
    Else '否则根据输入内容进行模糊筛选
        Dim txt As String = "'%" & drp.Text & "%'"
        tbl.Filter = "客户 ID Like " & txt & " Or 公司名称 Like " & txt & " Or 地址 Like " & txt & " Or 联系人 Like " & txt
    End If
End If
```

这样就能根据用户在下拉列表框输入的内容，动态筛选出可能的客户。

4、将 DropDownBox 的 KeyDown 事件代码设置为：

```
Dim drp As WinForm.DropDownBox = e.sender
If drp.DroppedDown Then '如果下拉窗口已经打开
    Dim tbl As Table = Tables("窗口 1_Table1")
    If e.KeyCode = Keys.Up Then '如果按下的是上箭头按键
        tbl.Position = tbl.Position - 1 '向上移动一行
        e.Cancel = True
    ElseIf e.KeyCode = Keys.Down Then '如果按下的是下箭头按键
        tbl.Position = tbl.Position + 1 '向下移动一行
        e.Cancel = True
    End If
End If
```

这样用户在下拉列表框输入内容的同时，还可以通过上下箭头按键来选择客户。

5、将 DropDownBox 的 Validating 事件代码设置为：

```
Dim drp As WinForm.DropDownBox = e.sender
If drp.DroppedDown Then '如果下拉窗口已经打开
    drp.CloseDropdown(False) '关闭下拉窗口
End If
```

这样就能确保输入焦点离开DropDownBox后，下拉窗口能自动关闭。

4.9.6.5 自行设计下拉目录树

我们知道，在 Foxtable 只需经过简单的设置，即可在表中使用下拉目录树高效录入数据，无需编写任何代码。

我们也可以抛开内置的功能，自己设计一个下拉目录树来输入数据，不要以为这是多余的，因为：

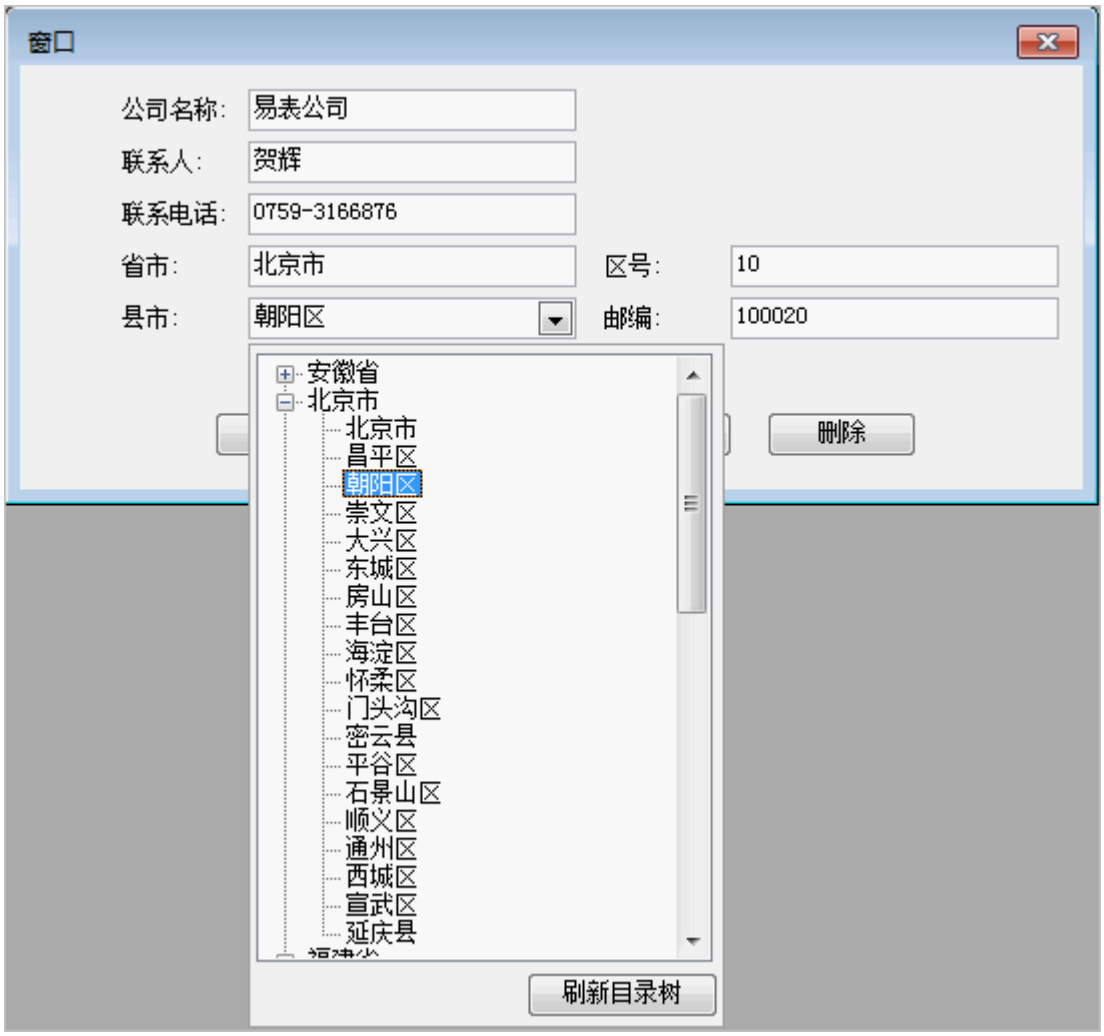
1、内置的下拉目录树只能在表中使用，不能在窗口使用。

2、内置的下拉目录树有固定的套路，一些特殊需求可能无法满足，例如本例的下拉目录树，增加了刷新功能。

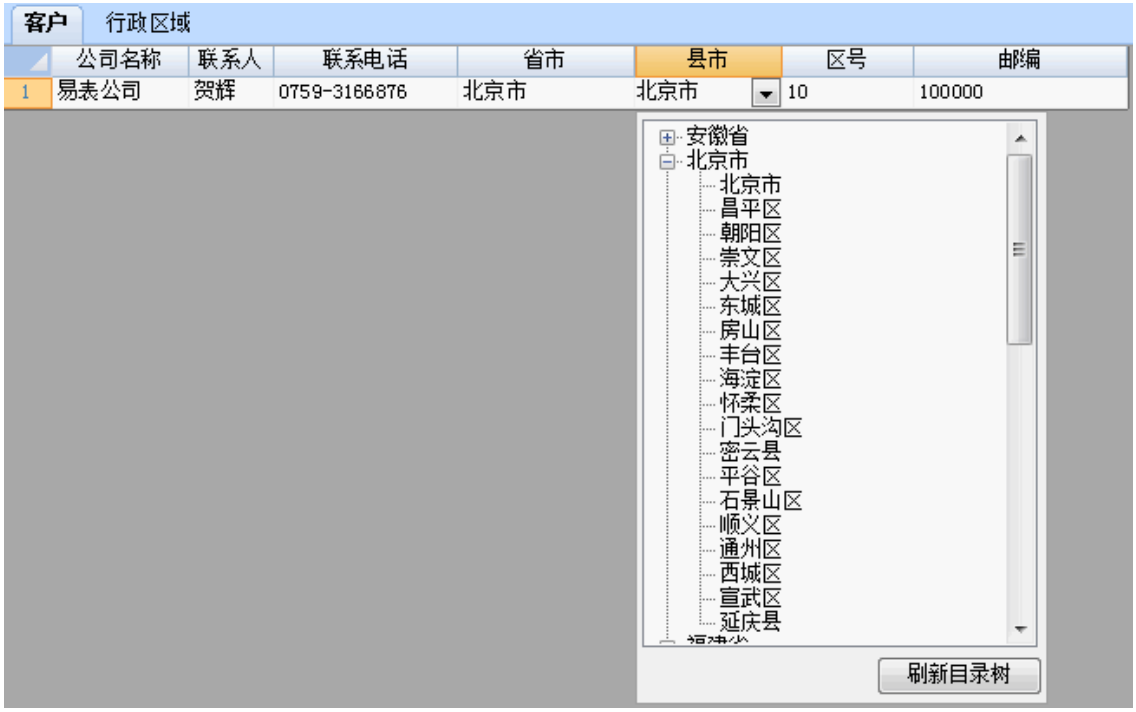
本示例请参考 CaseStudy 目录下的示例文件“下拉目录树.Table”。

此项目有两个表，分别为客户表和行政区域表，行政区域表为基础数据表，这个表已经输入了全国所有县级行政区域的资料，包括省市、市县、区号、邮编四列 。

我们希望在客户表设计一个下图所示的输入窗口，单击县市列这个文本框的下拉按钮，能出现一个下拉目录树，此目录树会列出所有的省市和县市，选择某个县市，会自动输入省市、县市、区号、邮编四列的内容：



同时希望在表中的县市列，也能通过下拉目录树来输入这四列的内容：



设计步骤

一、设计下拉窗口

1、新建一个窗口，名为“窗口 1”，窗口类型设置为“DropDownForm”。

2、窗口中插入一个 TreeView(目录树)控件。

3、窗口的 AfterLoad 事件设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市")
```

4、将刷新目录树按钮的代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市")
```

5、TreeView(目录树)的NodeMouseClick事件代码设置为：

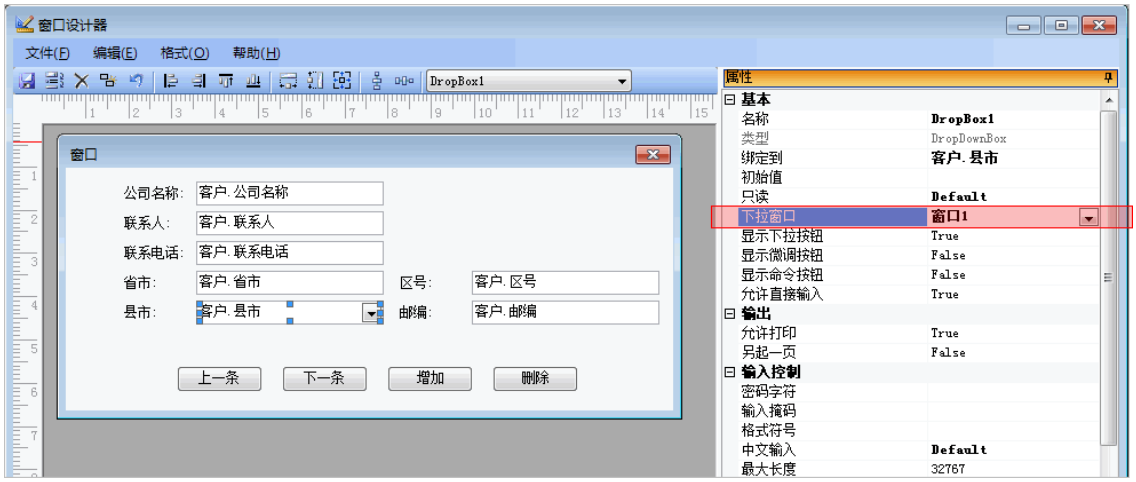
```
If e.Node.Level = 1 Then '如果单击的是第二层节点
    Dim tr As Row = Tables("客户").Current
    Dim dr As DataRow = e.Node.DataRow '获取生成此节点的DataRow
    tr("省市") = dr("省市")
    tr("县市") = dr("县市")
    tr("区号") = dr("区号")
    tr("邮编") = dr("邮编")
    e.Form.DropDownBox.Value = tr("县市") '这行不是多余的,省略此行,无法在表中正常使用此下拉窗口输入数据.
|   e.Form.DropDownBox.CloseDropdown()
End If
```

二、设计录入窗口

1、新建一个窗口，名为“窗口2”，窗口类型为“独立”。

2、插入几个输入框，分别绑定到各字段，其中县市列输入框是DropDownBox，其余各列是TextBox。

2、选择县市输入框，将其下拉窗口设置为“窗口1”：



三、在项目事件AfterOpenProject加入代码：

```
Tables("客户").Cols("县市").DropForm = "窗口1" '客户表的县市列也采用我们设计的下拉窗口输入数据。
```

4.9.6.6 可以通用的下拉目录树

本节的内容可以参考 CaseStudy 目录下的文件：通用下拉目录树.Table
本节的任务和上节类似，设计一个完全相同的下拉目录树用于快速输入数据，但不同的是，本节的下拉目录树是通用的，可以被多个表同时使用，而上一节设计的下拉目录树只能被客户表调用。
如果只是通过下拉目录树向某一列输入内容，那么下拉窗口自然就是通用的，只需在 NodeMouseClick 事件中写上代码：
e.Form.DropDownBox.Value =e.Node.Text
e.Form.DropDownBox.CloseDropdown()
如果要通过下拉目录树输入多列内容，而且要通用，就没办法如此简单的编码了，因为我们必须知道是什么表或什么列打开了此下拉目录树，才能向相关列写入数据。

获取打开下拉窗口的表和列的方法：

1、通过下拉窗口有 DropTable 和 DropCol 属性，可以知道是什么表的什么列打开了此下拉窗口，如果不是从表中打开的下拉窗口，那么这两个属性返回 Nothing。
2、如果 DropTable 返回 Nothing，那么通过下拉窗口的 DropDownBox 属性，可以获取打开此下拉窗口的 DropDownBox 控件，再通过此控件的 BindingDield 属性，就可以获取绑定的表和列。

示例

打开 CaseStudy 目录下的文件“通用下拉目录树.Table”，可以看到这个文件设计了一个名为“通用目录树”的下拉窗口，这个下拉窗口分别被两个表和两个窗口调用。

设置步骤：

1、新建一个窗口，名为“通用目录树”，窗口类型设置为“DropDownForm”。

2、窗口中插入一个 TreeView(目录树)控件。

3、窗口的 AfterLoad 事件设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市")
```

4、TreeView(目录树)的 NodeMouseClick 事件代码设置为：

```
If e.Node.Level = 1 Then '如果单击的是第二层节点
    Dim tr As Row
    Dim dr As DataRow = e.Node.DataRow '获取生成此节点的DataRow
    If e.Form.DropTable IsNot Nothing Then '如果下拉窗口是从表中打开
        tr = e.Form.DropTable.Current '获取此表的当前行
    Else '如果下拉窗口是通过DropdownBox打开
        Dim nm As String = e.Form.DropDownBox.BindingField '获取绑定的字段
        nm = nm.Split(".")(0) '获取绑定的表名
        tr = Tables(nm).Current '获取此表的当前行
    End If
    tr("省市") = dr("省市")
    tr("县市") = dr("县市")
    tr("区号") = dr("区号")
    tr("邮编") = dr("邮编")
    e.Form.DropDownBox.Value = tr("县市") '这行不是多余的,省略此行,无法在表中正常使用此下拉窗口输入数据.
    e.Form.DropDownBox.CloseDropdown()
End If
```

5、在项目事件 AfterOpenProject 加入代码：

```
Tables("客户").Cols("县市").DropForm = "通用目录树"
Tables("雇员").Cols("县市").DropForm = "通用目录树"
```

6、设计其它录入窗口，将相关 DropdownBox 控件的下拉窗口属性设置为“通用目录树”。

4.9.6.7 可以多选的下拉目录树

本节的任务是设计一个下图所示的下拉目录树，和之前的目录树不同，这个目录树可以选择多个

值：



本节的内容可以参考 CaseStudy 目录下的文件：多选目录树.Table

实现方法很简单：

1、新建一个窗口，名为“窗口 1”，窗口类型设置为“DropDownForm”。

2、窗口中插入一个 TreeView(目录树)控件和一个 Button(按钮)，TreeView 的"显示复选框"属性设置为 True。

3、窗口的 DropDownOpened 事件设置为：

```
Dim s As String   = e.Form.DropDownBox.Value
Dim nms As new List(of String)
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
If s > "" Then
    nms.AddRange(s.Split(","))
End If
For Each nd As WinForm.TreeNode In trv.AllNodes
    nd.Checked = nms.Contains(nd.text)
Next
```

上述代码的目的在于每次打开下拉窗口的时候，都能自动勾选已经输入的项目。

注意这段代码不能设置在窗口的 AfterLoad 事件中，下拉窗口只有在第一次打开的时候，才会执行 AfterLoad 事件，而 DropDownOpened 每次打开下拉窗口都会执行。

4、确定按钮的代码设置为：

```
Dim s As String
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
For Each nd As WinForm.TreeNode In trv.AllNodes
    If nd.Checked Then
```

```
        s = s & "," & nd.Text
    End If
Next
e.Form.DropDownBox.Value = s.trim(",")
e.Form.DropDownBox.CloseDropdown
```

5、最后将项目事件AfterOpenProject的代码设置为：

```
Tables("表A").Cols("第一列").DropForm   = "窗口1"
```

4.9.6.8 带查找功能的下拉目录树

本节内容可以参考 CaseStudy 目录下的文件：带查找功能的下拉目录树.Table。
本节内容的任务是设计一个下图所示的下拉窗口：



下面看一下如何设计。

设计下拉窗口

1、新建一个窗口，名为“窗口 1”，窗口类型设置为“DropDownForm”。
2、分别插入一个 TreeVew 和 Button 控件。
3、窗口的 AfterLoad 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree("行政区域","省市|县市")
```

4、窗口的DropDownOpened事件代码设置为：

```
e.Form.DropDownBox.Select()
```

5、TreeView控件的NodeMouseClick事件代码设置为：

```
If e.Node.Level = 1 Then '如果单击的是第二层节点
    Dim tr As Row = Tables("客户").Current
    Dim dr As DataRow = e.Node.DataRow '获取生成此节点的DataRow
    tr("省市") = dr("省市")
    tr("县市") = dr("县市")
    tr("区号") = dr("区号")
    tr("邮编") = dr("邮编")
    e.Form.DropDownBox.Value = tr("县市")
    e.Form.DropDownBox.CloseDropdown()
End If
```

6、Button控件的Click事件代码设置为：

```
Dim txt As String = e.Form.DropDownBox.Text
If txt = "" Then '如果下拉框没有输入内容,则无需查询
    Return
End If
Dim trv As WinForm.TreeView =   e.Form.Controls("TreeView1")
Dim Start As Integer
Dim idx As Integer = - 1
If trv.SelectedNode IsNot Nothing Then '获取当前节点位置.
    Dim FullName As String = trv.SelectedNode.FullName
    For i As Integer = 0 To trv.AllNodes.count - 1
        If trv.AllNodes(i).FullName = FullName   Then
            Start = i
            Exit For
        End If
    Next
End If
For i As Integer = Start + 1 To trv.AllNodes.count - 1   '从当前节点的下一个节点开始查找
    Dim nd As WinForm.TreeNode = trv.AllNodes(i)
    If nd.Text.IndexOf(txt) >= 0 Then
        trv.SelectedNode = nd
        nd.EnsureVisible
        idx = i '将找到的位置复制给变量idx
        Exit For
```

```
    End If
Next
If  idx = -1 Then 'idx的值如果等于初始值-1,表示没有找到下一个符合条件的节点,那么从第一个节点开始重新查找.
    For i As Integer = 0 To Start - 1
        Dim nd As WinForm.TreeNode = trv.AllNodes(i)
        If nd.Text.IndexOf(txt) >= 0 Then
            trv.SelectedNode = nd
            nd.EnsureVisible
            Exit For
        End If
    Next
End If
trv.Select()
```

设计录入窗口

1、新建一个录入窗口

2、在窗口中DropDownBox，将其绑定到“县市”列。

3、将DropDownBox的下拉窗口属性设置为“窗口1”

4、将DropDownBox的TextChanged事件代码设置为：

```
Dim drp As WinForm.DropDownBox = e.sender
If drp.DroppedDown Then '如果下拉窗口已经打开
    Dim trv As WinForm.TreeView = Forms("窗口1").Controls("TreeView1")
    If drp.Text > "" Then '如果内容不为空
        For Each nd As WinForm.TreeNode In trv.AllNodes
            If nd.Text.IndexOf(drp.Text) >= 0 Then
                trv.StopRedraw()
                trv.SelectedNode = nd
                nd.EnsureVisible()
                trv.ResumeRedraw()
                Exit For
            End If
        Next
    End If
End If
```

5、将DropDownBox的KeyPress事件代码设置为：

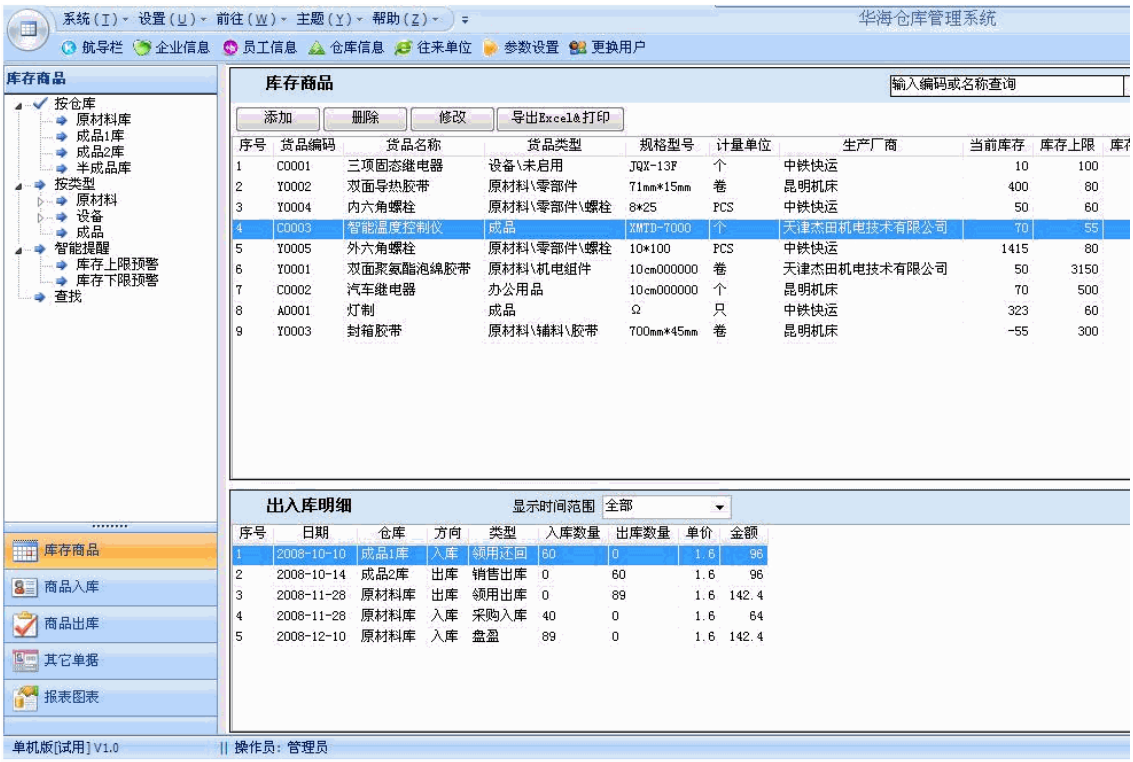
```
Dim drp As WinForm.DropDownBox = e.Sender
If drp.DroppedDown = False '如果下拉窗口没有打开
```

```
    drp.OpenDropDown() '打开下拉窗口
End If
```

4.9.7 其它

4.9.7.1 设计以窗口为主的界面

下图是基于 Foxtable 设计的一个仓管软件，和 Foxtable 本身以表为中心的界面不同，此软件的界面以窗口为中心。



设计这样的界面很简单：

1、设计窗口的时候，将窗口类型设为主窗口。

2、在项目事件 AfterOpenProject 中设置代码：

```
If User.Type <> UserTypeEnum.Developer
    TableCaptionVisible = False
End If
```

这样普通用户登录后，会隐藏显示下图所示的表标题，但开发者登录后，表标题则正常可见，不会影响开发工作：



TableCaptionVisible 是一个逻辑型的全局变量，用于设置是否显示表标题。

为便于调试，你可以在菜单栏中加一个按钮，将按钮的 Click 事件代码设置为：

```
TableCaptionVisible = Not TableCaptionVisible
```

这样单击此按钮，可以切换表标题的显示状态。

3、在 AfterOpenProject 事件中，还应该设置代码，打开第一个表的主窗口：

```
Forms("主窗口名").Open()
```

这样的系统，通常每个表都会有自己的主窗口，建议默认不要打开其他表的窗口，需要切换到某个表的时候，再打开此表的主窗口即可。

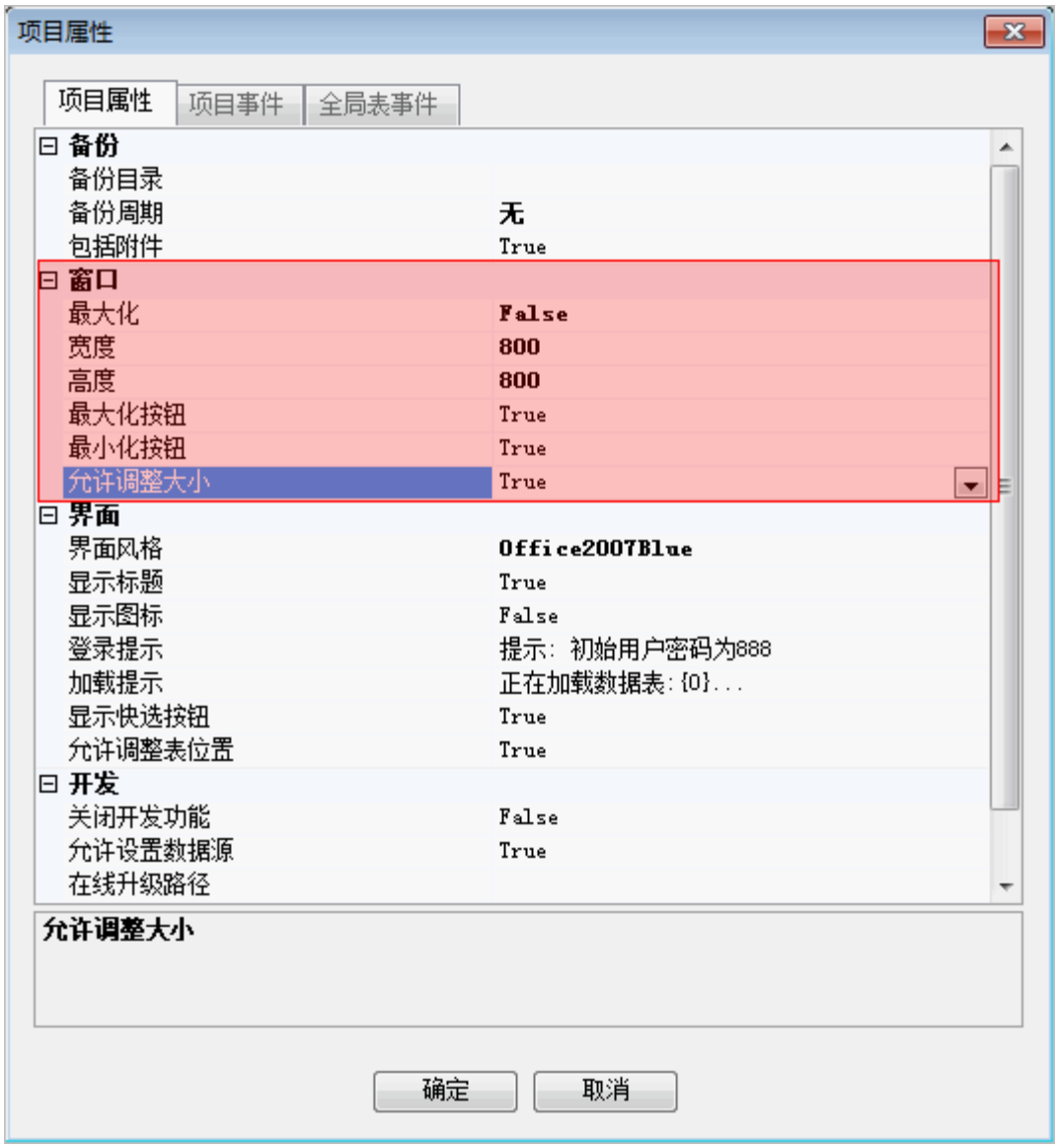
例如希望单击某个按钮，能够切换到表 B，并显示其主窗口，按钮代码可以设置为：

```
Forms("表 B 主窗口").Open()
MainTable = Tables("表 B")
```

上面的代码是先打开表 B 的主窗口，再切换到表 B，这样的顺序不能乱，否则屏幕会闪动。

设置主窗口样式

Foxtable 的主窗口默认是最大化的，在项目属性设置窗口，通过下面一组属性，可以对主窗口就行设置，这些属性不是即时生效的，设置完成后，需要重新打开项目：



提示：

1、只有在最大化属性为 False 的时候，宽度、高度、最大化按钮、最小化按钮、允许调整大小这五个属性的设置才会有效。
2、为了不影响正常开发，建议你在开发完成，准备发布或交付给用户使用的时候，才设置上述属性。

4.9.7.2 设计查询窗口

本课的示例文件为 CaseStudy 目录下的“窗口筛选.table”。
Foxtable 本身的筛选功能已经非常强大，不过如果自己设计一个筛选窗口，用起来也许更顺手。
本课的目的是为下表：

	产品	客户	雇员	单价	折扣	数量	金额	日期	已付款
1	PD01	CS03	EP04	18	0.15	80	1224	1999-01-	☑
2	PD01	CS04	EP05	14.4	0	200	2880	1999-01-	☐
3	PD01	CS02	EP01	18	0.2	800	11520	1999-01-	☑
4	PD01	CS04	EP02	14.4	0.05	500	6840	1999-01-	☑
5	PD01	CS03	EP04	14.4	0.25	200	2160	1999-01-	☑
6	PD01	CS02	EP05	14.4	0	100	1440	1999-01-	☑
7	PD01	CS03	EP01	14.4	0	240	3456	1999-01-	☑
8	PD01	CS05	EP05	18	0	200	3600	1999-02-	☐
9	PD01	CS01	EP04	18	0	20	360	1999-02-	☑
10	PD01	CS01	EP02	18	0	80	1440	1999-02-	☑

设计一个下图所示的筛选窗口：



重要提示：用于输入查询内容的各输入框，切不可绑定到任何列，这是初学者很容易犯的一个错误。

设计步骤

1、插入一个组合框，改名为“cmbProduct”，列表项目设为“PD01|PD02|PD03|PD04|PD05”，用于输入要筛选的产品。

2、插入一个组合框，改名为“cmbCustomer”，列表项目设为“CS01|CS02|CS03|CS04|CS05”，用于输入要筛选的客户。

3、插入一个组合框，改名为“cmbEmployee”，列表项目设为“EP01|EP02|EP03|EP04|EP05”，用于输入要筛选的雇员。

4、插入一个日期输入框，改为名“StartDate”，用于输入开始日期。

5、插入一个日期输入框，改为名“EndDate”，用于输入结束日期。

6、插入三个单选框，名称分别设为"rdoAll"、"rdoYifu"、"rdoWeifu"，标题分别设为"全部"、"已付"、"未付"。

7、插入三个按钮，标题和 Click 事件代码按下表所示设置。

标题	Click 事件代码
清除条件	e.Form.Controls("cmbProduct").Value = Nothing e.Form.Controls("cmbCustomer").Value = Nothing e.Form.Controls("cmbEmployee").Value = Nothing e.Form.Controls("StartDate").Value = Nothing e.Form.Controls("EndDate").Value = Nothing e.Form.Controls("rdoAll").Checked = True

撤销筛选	Tables("订单").ApplyFilter = False

开始筛选	Dim Filter As String
	With e.Form.Controls("cmbProduct")
	  If .Value IsNot Nothing Then
	  Filter = "产品 = '" & .Value & "'"
	  End If
	End With
	With e.Form.Controls("cmbCustomer")
	  If .Value IsNot Nothing Then
	  If Filter > "" Then
	  Filter = Filter & " And "
	  End If
	  Filter = Filter & "客户 = '" & .Value & "'"
	  End If
	End With
	With e.Form.Controls("cmbEmployee")
	  If .Value IsNot Nothing Then
	  If Filter >"" Then
	  Filter = Filter & " And "
	  End If
	  Filter = Filter & "雇员 = '" & .Value & "'"
	  End If
	End With
	With e.Form.Controls("StartDate")
	  If .Value IsNot Nothing Then
	  If Filter >"" Then
	  Filter = Filter & " And "
	  End If
	  Filter = Filter & "日期 >= #" & .Value & "#"
	  End If
	End With
	With e.Form.Controls("EndDate")
	  If .Value IsNot Nothing Then
	  If Filter >"" Then
	  Filter = Filter & " And "
	  End If
	  Filter = Filter & "日期 <= #" & .Value & "#"
	  End If
	End With
	If e.Form.Controls("rdoYifu").Checked = True ' 如果付款状态选择了"已付"
	  If Filter >"" Then
	  Filter = Filter & " And "
	  End If

```
    Filter = Filter & "已付款  = True"
End If
If e.Form.Controls("rdoWeifu").Checked = True ' 如果付款状态选择了"未付"
    If Filter >"" Then
        Filter = Filter & " And "
    End If
    Filter = Filter & "已付款  = False Or  已付款  Is Null"
End If
If Filter > "" Then
    Tables("订单").Filter = Filter
End If
```

4.9.7.3 窗口中的动态列表项目

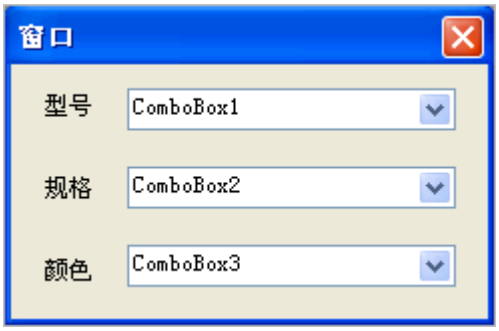
在 Foxtable 编程部分，我们已经学会了如何在表中实现动态列表项目。
现在我们同样看看如何在窗口中实现列表项目。
本节示例可以参考 CateStydy 目录下的文件：窗口动态列表项目.table

示例一

假定有一个表：

	型号	规格	颜色
1	VV	4	红
2	VV	6	红
3	YJV	10	黄
4	YJV	16	绿
5	VV	10	黄
6	VV	10	蓝
7	VV	10	黑
8	YJV	16	白

和下面这样的一个窗口：



希望第一个组合框能够自动列出所有型号供选择，而第二个组合框能够列出所选型号的全部规格供选择，同样第三个组合框能够列出所选规格的全部颜色供选择。
设计的步骤为：
1、在窗口的 AfterLoad 事件中加入代码：

```
Dim cmb As WinForm.ComboBox = e.form.Controls("ComboBox1")
cmb.ComboList = DataTables("表 A").GetComboListString("型号")
```

2、在 ComboBox2 的 Enter 事件中加入代码：

```
Dim cmb As WinForm.ComboBox = e.Sender
Dim str As string = e.Form.Controls("ComboBox1").Value
cmb.ComboList = DataTables("表 A").GetComboListString("规格", "型号 = '" & str & "'")
```

3、最后在 ComboBox3 的 Enter 事件中加入代码：

```
Dim cmb As WinForm.ComboBox = e.Sender
Dim str As String = e.Form.Controls("ComboBox2").Value
cmb.ComboList = DataTables("表 A").GetComboListString("颜色", "规格 = '" & str & "'")
```

示例二

假定组合框的列表项目并非动态的，随着应用的深入，这些列的列表项目可能会需要调整，虽然通过窗口设计，可以方便地重新设置列表项目。但是对于一个已经交付使用的系统来说，开放窗口设计功能给最终用户，是一个非常危险的做法。有没有安全一些的做法呢？有，通用的做法是：

1、新建一个表(假定表名为项目)，包括三列，分别是表、列、列表项目，用于输入某表某列的列表项目:

	表	列	列表项目
1	表A	第一列	项目1|项目2
2	表A	第二列	项目3|项目4
3	表B	第一列	项目5|项目6|项目7

2、将组合框的 Enter 事件代码设为：

```
Dim cmb As WinForm.ComboBox = e.Sender
If cmb.Items.Count > 0 Then '如果已经设置了项目
    Return '则返回
End If
Dim t As Table = cmb.BindTable '获得绑定的 Table
Dim c As Col = cmb.BindCol '获得绑定的 Col
Dim dr As DataRow = DataTables("项目").Find("[表] = '" & t.Name & "' And [列] = '" & c.Name & "'")
If dr IsNot Nothing Then '如果在项目表找到对应的行
    cmb.Combolist = dr("列表项目") '则设置列表项目
End If
```

代码原理很简单，根据代码注释，大家应该都能理解。

一个窗口可能有很多个组合框，针对每个组合框重复设置这样的代码，显得太累赘，为此我们可以使用窗口的全局事件 Enter，这样只需设置一次即可。但是全局事件对所有类型的控件有效，而我们的代码是针对组合框的，即使是组合框，也不见得都要执行上述代码。我们可以利用控件名称进行标识，例如对于需要执行上述代码的组合框，名称统一用字符"_cmb"开始，这样只需将全局事件 Enter 设为：

```
If e.Sender.Name.SubString(0,4) = "_cmb" Then '如果名称符合
    Dim cmb As WinForm.ComboBox = e.Sender
    If cmb.Items.Count > 0 Then '如果已经设置了项目
        Return '则返回
    End If
    Dim t As Table = cmb.BindTable '获得绑定的 Table
    Dim c As Col = cmb.BindCol '获得绑定的 Col
```

```
    Dim dr As DataRow = DataTables("项目").Find("[表] = '" & t.Name & "' And [列] = '" & c.Name & "'")
    If dr IsNot Nothing Then '如果在项目表找到对应的行
        cmb.Combolist = dr("列表项目") '则设置列表项目
    End If
End If
```

4.9.7.4 可以多选的筛选树

本节的内容可以参考 CaseStudy 目录下的文件: 多选筛选树.Table
本节的任务是生成一个可以多选的筛选树：



	产品	客户	雇员	单价	折扣	数量	日期
1	PD01	CS03	EP04	18	0.15	80	1999-01-04
2	PD01	CS02	EP01	18	0.2	800	1999-01-10
3	PD01	CS03	EP04	14.4	0.25	200	1999-01-14
4	PD01	CS02	EP05	14.4	0	100	1999-01-20
5	PD01	CS03	EP01	14.4	0	240	1999-01-21
6	PD01	CS03	EP04	18	0	40	1999-02-07
7	PD01	CS02	EP05	14.4	0.15	150	1999-02-22
8	PD01	CS02	EP03	14.4	0	200	1999-02-26
9	PD01	CS03	EP01	18	0.2	400	1999-03-03
10	PD01	CS02	EP02	18	0	100	1999-03-04
11	PD01	CS03	EP01	14.4	0	120	1999-03-14
12	PD01	CS03	EP01	18	0.25	150	1999-03-16
13	PD01	CS02	EP01	14.4	0	700	1999-03-17
14	PD01	CS02	EP03	18	0	400	1999-03-19

常规设计

设计步骤非常简单：

1、新建一个窗口，插入一个目录树控件，将目录树的“显示复选框”属性设置为 True。

2、将窗口的 AfterLoad 事件代码设置为：

```
e.Form.Controls("TreeView1").BuildTree("订单", "产品|客户")
```

3、将目录树的 AfterCheckNode 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Sender
Dim flt As String
For Each nd As WinForm.TreeNode In e.node.Nodes '清除子节点选中状态
    nd.Checked = False
Next
If e.node.ParentNode IsNot Nothing Then '去掉父节点选中状态
    e.node.ParentNode.Checked = False
End If
For Each nd As WinForm.TreeNode In trv.AllNodes
     If nd.Checked Then
      If flt > "" Then
```

```
            flt = flt & " Or "   '注意用or而不是And
        End If
        If nd.Level = 0 Then '注意下面的条件都要用括号括起来
            flt = flt & "( 产品 = '" & nd.Text & "')"
        Else
            flt = flt & "( 产品 = '" & nd.ParentNode.Text & "' And 客户 = '" & nd.Text & "')"
        End If
    End If
Next
Tables("订单").Filter = flt
```

通用设计

上述常规编码方式，目录树的层级越多，代码就越复杂，下面我们提供一个通用的筛选树代码，不过有多少层，代码都一样。

设计步骤同样很简单:

1、新建一个窗口，插入一个目录树控件，将目录树的“显示复选框”属性设置为 True。
2、将窗口的 AfterLoad 事件代码设置为：

```
e.Form.Controls("TreeView1").BuildTree("订单", "产品|客户|雇员")
```

3、将目录树的 AfterCheckNode 事件代码设置为：

```
Dim nms As String() = {"产品","客户","雇员"} '指定生成目录树的各列
Dim qts As String() = {"","",""} '指定将各列的值括起来的符号,这里都是字符型,所以都是单引号
Dim trv As WinForm.TreeView = e.Sender
Dim flt As String
Dim nd As WinForm.TreeNode
For Each nd In e.node.allNodes '清除子节点的选中标记
    nd.Checked = False
Next
nd = e.Node.ParentNode
Do While nd IsNot Nothing '清除父节点的选中标记
    nd.Checked = False
    nd = nd.ParentNode
Loop
For Each nd In trv.AllNodes
    If nd.Checked  Then
        Dim rts() As String = nd.FullPath.Split("\")
        Dim val As String = ""
        For i As Integer = 0 To rts.length - 1
            If val > "" Then
                val = val & " And "
```

```
            End If
            val = val & nms(i) & " = " & qts(i) & rts(i) & qts(i)
        Next
        If flt > "" Then
            flt = flt & " Or (" & val & ")"
        Else
            flt = val
        End If
    End If
Next
Tables("订单").Filter  = flt
```

至此这个多层的筛选树已经设计完成，而且代码是完全通用的，例如我们要根据产品、日期、折扣三列生成筛选树，只需稍作调整即可：

1、将窗口事件AfterLoad的代码改为：

```
e.Form.Controls("TreeView1").BuildTree("订单", "产品|日期|折扣")
```

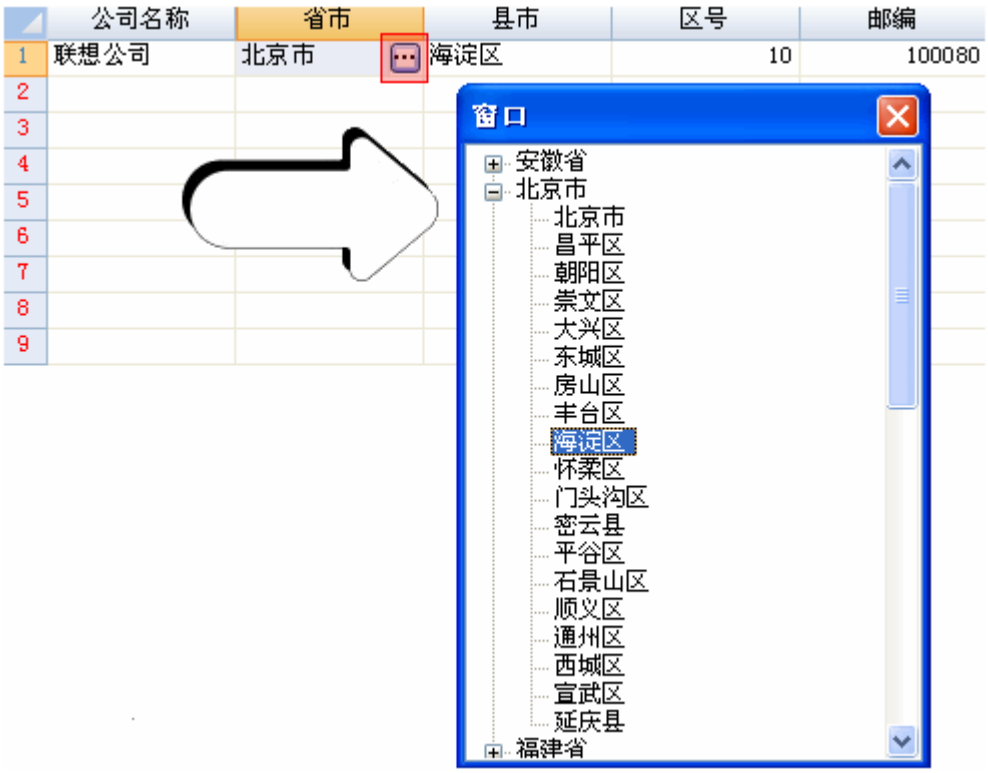
2、将目录树的 AfterCheckNode 事件的 前两行代码改为：

```
Dim nms As String() = {"产品","日期","折扣"} '指定生成目录树的各列
Dim qts As String() = {"'","#",""} '指定将各列的值括起来的符号
```

前后只需修改三行代码，唯独需要注意的是指定将各列的值括起来的符号时，记得字符列用单引号，日期列用符号#，数值列则不需要指定符号，例如上面的折扣列。

4.9.7.5 用目录树输入数据

我们已经知道，用 DropTreeBuilder 可以为列生成下拉目录树，用于快速输入多列数据。
本节我们看看如何自己编码实现用目录树输入数据，这并非多此一举，毕竟有时我们会有一些自己的特殊需要，DropTreeBuilder 并不一定能够满足我们的要求。
本节内容可以参考 CaseStudy 目录下的示例文件"用目录树输入数据.Table"
该示例文件有一个行政区域表，已经输入所有的县市及其区号、邮编，以及所属的省市。
我们希望选择客户资料表的省市列的时候，能够出现一个按钮，单击该按钮即可弹出一个窗口，以目录树的形式列出所有的行政区域，双击某个节点，即可自动输入对应的省市、县市、邮编和区号，如下图所示：



实现的过程很简单：

1、在行政区域表新建一个窗口，窗口类型设为模式型。

2、在窗口中加入一个目录树控件。

3、将窗口的 AfterLoad 事件设为：

```
Dim tv As WinForm.TreeView
tv = e.Form.Controls("TreeView1")
tv.BuildTree("行政区域","省市|县市")
```

4、将目录树的 NodeMouseDoubleClick 事件设为：

```
If e.Node.Level = 0 Then '如果是顶层节点
    Return '则返回
End If
Dim Vals() As String = e.Node.FullPath.Split("\")
Dim Filter As String = "[省市] = '" & Vals(0) & "' And [县市] = '" & Vals(1) & "'"
Dim dr As DataRow = DataTables("行政区域").Find(Filter)
Dim r As Row = Tables("客户资料").Current
r("省市") = dr("省市")
r("县市") = dr("县市")
r("区号") = dr("区号")
r("邮编") = dr("邮编")
e.Form.Close()
```

5、最后选择客户资料表，将其 CellButtonClick 事件设为：

```
Forms("窗口 1").Open()
e.Cancel = True
```

4.9.7.6 自动关闭与隐藏

本课的内容仅针对独立型窗口，因为其它类型的窗口并不存在自动隐藏的问题。
我们已经知道，独立窗口是可以共享的，独立窗口一旦打开，切换到其他主表后，窗口仍然是可见的。
有的时候我们并不希望这样，例如有一个设计好的窗口，名为“订单筛选”，我们希望进入订单表的时候，该窗口能够自动打开，而离开订单表的时候，能够自动关闭。
实现这个要求的代码很简单，只需将项目的 MainTableChanged 事件设为：

```
If MainTable.Name = "订单" Then
    Forms("订单筛选").Show()
Else
    Forms("订单筛选").Close()
End If
```

但是还有一点不足，因为离开订单表的时候，窗口被关闭了，窗口位置以及所输入的内容会全部丢失，能不能在离开订单表的时候，只是暂时隐藏窗口，而不关闭窗口？
当然可以，只需上面的代码改为：

```
If MainTable.Name = "订单" Then
    Forms("订单筛选").Show()
Else
    Forms("订单筛选").Visible = False
End If
```

还有一个问题，你可能并不希望自动打开窗口，只是希望对于已经打开的窗口，能够自动显示和隐藏，满足这种要求的代码为：

```
If MainTable.Name = "订单" Then
    Forms("订单筛选").Visible = True
Else
    Forms("订单筛选").Visible = False
End If
```

本课的内容，请参考 CaseStudy 目录下的示例文件“窗口筛选.table”。

4.9.7.7 InputValue 函数

InputValue 函数用于显示一个对话框，获得用户输入的单个值。
这样我们就无需为一些简单的输入工作而专门设计窗口，直接使用 InputValue 函数即可。
语法：
InputValue(Variant, Caption，Description)
Variant:　　用于接收数据的变量
Caption：　　字符型，设置对话框标题
Description：　字符型，设置对话框说明
InputValue 会返回一个逻辑值，如果用户在对话框中单击的是确定按钮，则返回 True，否则返回 False。
示例一

```
Dim Val As Date = Date.Today
If InputValue(Val, "筛选","请输入日期:") Then
    Output.Show(Val)
End If
```

示例二

```
Dim Val As Integer = 5
If InputValue(Val, "增加行", "请输入要增加的行数:") Then
    Output.Show(Val)
End If
```

4.9.7.8 InputPassword 函数

InputPassword 函数用于显示一个对话框，用于输入密码。
这样我们就无需为一些简单的密码输入工作而专门设计窗口，直接使用 InputPassword 函数即可。
语法：
InputPassword(Password, Caption，Description)
Password:　　用于接收密码的变量
Caption：　　字符型，设置对话框标题
Description：　字符型，设置对话框说明
InputPassword 会返回一个逻辑值，如果用户在对话框中单击的是确定按钮，则返回 True，否则返回 False。
示例

```
Dim pwd As String
If InputPassWord(pwd,"提示","请输入密码:") Then
    Messagebox.Show("您输入的密码是:" & pwd)
End If
```

4.9.7.9 PopMessage 方法

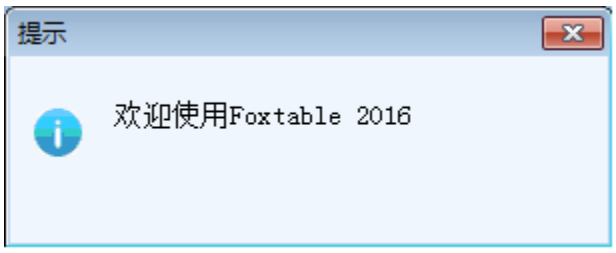
PopMessage 用于以弹窗形式，在屏幕右下角显示提示信息。
语法：
PopMessage(Message, Caption, Icon, Duration)
Message: 字符型，用于指定要显示的信息。
Caption：可选参数，字符型，用于指定窗口标题。
Icon:　可选参数，用于指定图标，PopIconEnum 型枚举，可选值有：
Infomation
OK
Alert
Error
Question
Duration: 整数型，单位为秒，用于设置信息窗口显示的时长，如果省略此参数，窗口会一直显示，

直到用户关闭窗口。

示例 1：

```
PopMessage("欢迎使用 Foxtable 2016")
```

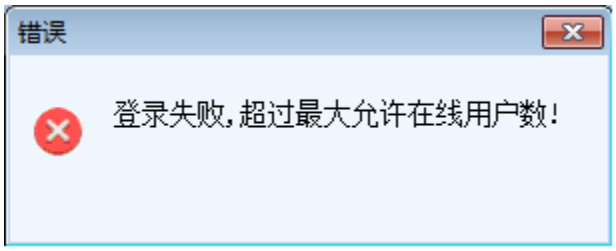
显示的窗口为：



示例 2：

```
PopMessage("登录失败,超过最大允许在线用户数!","错误", PopIconEnum.Error)
```

显示的窗口为：



示例 3

如果希望窗口显示 5 秒后，能自动关闭，可以将代码改为：

```
PopMessage("登录失败,超过最大允许在线用户数!","错误", PopIconEnum.Error, 5)
```

提示：

Foxtable 2016 新增了一个窗口类型 PopupForm，此类型的窗口在屏幕右下角以弹窗形式显示，当 PopMessage 无法满足要求的时候(例如需要在提示窗口加上按钮，让用户根据提示进行一些操作)，可以自行设计 PopupForm 类型的窗口来显示提示信息。

4.9.7.10 DoEvents 语句

新建一个窗口，增加一个标签和一个按钮，按钮的 Click 事件设为：

```
e.Form.Controls("Label1").Text = "正在增加行,请稍后..."
CurrentTable.AddNew(10000)
e.Form.Controls("Label1").Text = "已经完成."
```

打开窗口单击按钮，按道理在增加行的时候，标签应该显示“正在增加行,请稍后...”，可实际上标签只是在增加行结束的时候，显示“已经完成.”。这是为什么呢？这是因为 Windows 是多任务的，绘制控件属于优先级别较低的任务，直到我们执行完所有的代码，Windows 才会去绘制控件，从而出现上述异常。

有什么办法让绘制控件的代码即刻生效呢？有，我们只需在代码中间插入一行:

```
Application.DoEvents
```

即刻暂停代码的执行，重新绘制控件后，继续执行代码。

所以我们原来的代码应该改为：

```
e.Form.Controls("Label1").Text = "正在增加行,请稍后..."
Application.DoEvents()
CurrentTable.AddNew(10000)
e.Form.Controls("Label1").Text = "已经完成."
```

4.9.7.11 数据绑定与设置值

如果一个控件已经绑定到了某列，就不应该直接设置该控件的值，而是直接设置其绑定列的值。

假定有一个日期输入框(DateTimePicker)，绑定到了订单表的日期列，我们将某个按钮的代码设置为：

```
Dim dtp As WinForm.DateTimePicker = e.Form.Controls("DateTimePicker1")
dtp.Value = Date.Today
MessageBox.Show("输入框的值:" & dtp.Value)
MessageBox.Show("日期列的值:" & Tables("订单").Current("日期"))
```

单击这个按钮，你会发现日期输入框的值，确实等于今天的日期，但是订单表日期列的值，并不是今天，而是原来的值。

也就是说，在绑定状态下，设置控件的值，并不会同步更新其绑定列的值。

在绑定状态下，我们应该直接设置列的值，而不是控件的值。

例如我们将按钮的代码改为：

```
Tables("订单").Current("日期") = Date.Today()
MessageBox.Show("输入框的值:" & e.Form.Controls("DateTimePicker1").Value)
MessageBox.Show("日期列的值:" & Tables("订单").Current("日期"))
```

现在单击这个按钮，可以发现日期输入框的值和订单表的日期列，都等于今天的日期。

实际上，在绑定状态下，不管是写值还是取值，都应该通过列进行，而不是通过控件进行。

例如假定窗口有个 TextBox 控件，绑定到表 A 的第一列，在窗口的 AfterLoad 设置如下代码：

```
Dim txt1 As WinForm.TextBox = e.Form.Controls("TextBox1")
MessageBox.Show(txt1.Value)
```

然后打开窗口，可以发现显示的是一个空值，而不是第一列真正的值，因为窗口刚刚打开的时候，控件还没有来得及从表中取值。

在绑定状态下取值，一样应该直接读取列的值，而不是读取控件的值，所以正确的代码是：

```
Dim txt1 As WinForm.TextBox = e.Form.Controls("TextBox1")
MessageBox.Show(Tables("表A").Current("第一列"))
```

4.9.7.12 动态错误提示

通常我们会在 Validating 事件中设置代码来验证输入数据，例如：

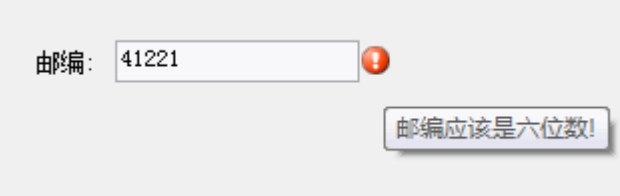
```
If e.Sender.Text.Length <> 6 Then
        MessageBox.Show("邮编应该是六位数!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True
End If
```

这样用户离开邮政编码输入框之前，就会判断输入内容的长度，如果长度不等于 6，将出现一个对话框提示用户，并禁止离开邮政编码输入框。

这种错误提示比较“粗暴”，我们可以换一个较为“平和”的提示方式，将 Validating 事件代码改为：

```
If e.Sender.Text.Length <> 6 Then
    e.Sender.Error = "邮编应该是六位数!"
Else
    e.Sender.Error = ""
End If
```

这样输入的邮政编码长度如果不等于 6，将在输入框的后边显示一个红色的小图标，表明有输入错误，将鼠标移到这个图标上时，就会显示错误信息：



如果想在保存的时候统一验证，可以将“保存”按钮的代码设置为：

```
Dim HasError As Boolean
Dim r As Row = Tables("客户").Current
If r.IsNull("邮编") OrElse r("邮编").Length <> 6 Then
    e.Form.Controls("TextBox1").Error = "邮政编码应该是六位数!"
    HasError = True
Else
    e.Form.Controls("TextBox1").Error =  ""
End If
If r.IsNull("分组") Then
    e.Form.Controls("TextBox2").Error = "必须输入用户分组!"
    HasError = True
Else
    e.Form.Controls("TextBox2").Error = ""
End If
```

```
If HasError = False Then '如果没有输入错误,则保存
   r.Save()
End If
```

同时将“取消”按钮的代码设置为：

```
Dim r As Row = Tables("客户").Current
r.Reject() '撤销修改
e.Form.ClearErrors() '清除所有控件的错误提示
```

4.9.7.13 动态信息提示

我们知道通过 ToolTip 属性可以设置控件的提示信息，但是这个提示信息只有鼠标经过并停留在这个控件的时候，才会显示。
有的时候我们希望强行在某个控件显示一个提示信息，直到操作结束后才隐藏这个提示信息，此时就需要用 ShowToolTip 和 HideToolTip 方法。
示例
假定窗口有个编码文本输入框(TextBox)，用于输入编码，希望焦点进入此输入框的时候，能够自动显示一个提示，离开此输入框的时候，能自动隐藏这个提示：



要实现这个功能很简单：
1、设置此 TextBox 的 Enter 事件代码为：

```
Dim v As String = "1: 初中"
v = v & vbcrlf & "2: 高中"
v = v & vbcrlf & "3: 大专"
v = v & vbcrlf & "4: 本科"
v = v & vbcrlf & "5: 硕士"
v = v & vbcrlf & "6: 博士"
e.Sender.ShowToolTip(v)
```

2、设置此 TextBox 的 Leave 事件代码为：

```
e.Sender.HideToolTip()
```

4.9.7.14 使用快捷菜单

如果要右击出现某个快捷菜单，首先要设计好快捷菜单，有关菜单设计，后面有专门的一章《菜单设计》进行讲述。

快捷菜单设计好之后，如果希望在某个控件上右击时，能够显示某一快捷菜单，可以将该控件的MouseDown 事件设为：

```
If e.Button = MouseButtons.Right Then '假定按下的是鼠标右键
    ContextMenus("快捷菜单名").Show(e.Sender,e.x,e.y) '则在鼠标位置显示快捷菜单
End If
```

4.9.7.15 主界面导航设计方案

一个管理系统会有多个功能模块，用户需要在不同的模块之间切换，也就是导航。

其实主界面的设计没有定式可言，本节介绍的只是一些常见的设计方法，如果你已经认真学习了前面的帮助，那么本节的内容是多余的，不过确实经常有用户会提出“怎样设计主界面？”的问题，所以我们还是专门用一节的帮助对此进行介绍。

设计主界面，首先要清楚了解各种窗口类型，因为界面设计需要各种窗口类型配合使用。

常规的导航

最常用的导航方式是在菜单中加入一些按钮，按钮的代码一般设置为：

```
Forms("窗口名").Open()
MainTable = Tables("表名")
```

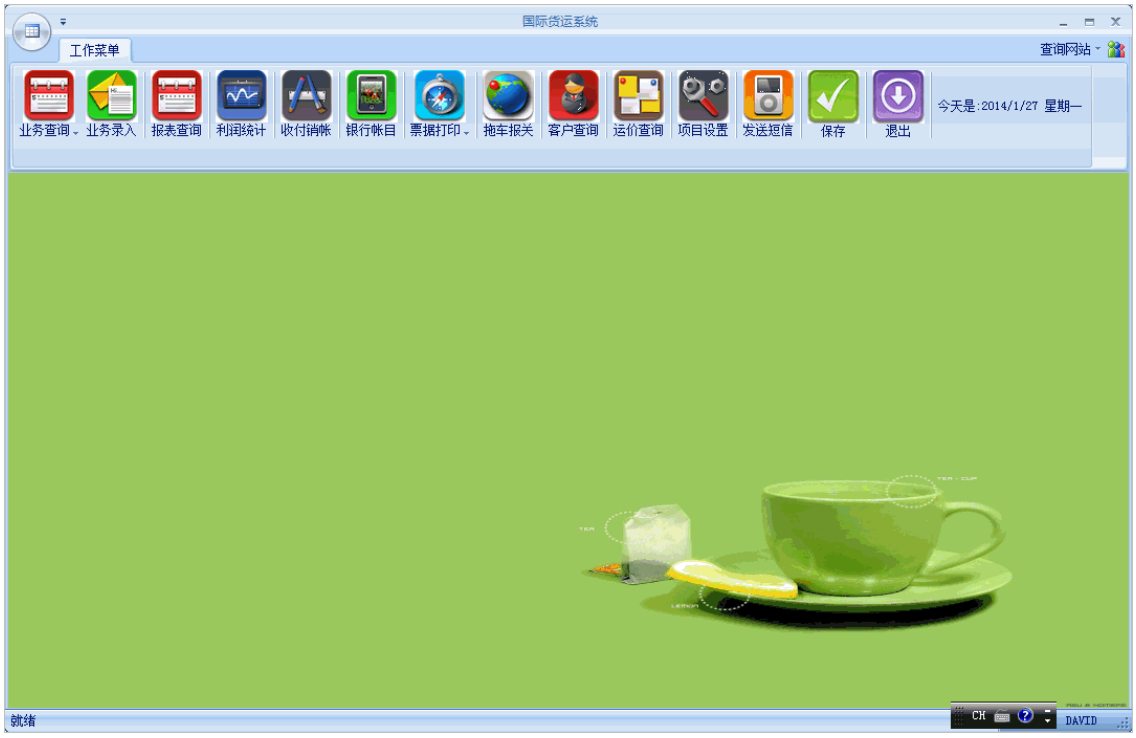
这样单击这些按钮就会打开对应的窗口，并切换到对应的表：



如果希望打开项目之后，系统默认显示一个背景图片，可以单独设计一个窗口，窗口类型设置为主窗口，将窗口的背景图片设置为希望的图片，在AfterOpenProject事件中设置代码打开此窗口即可：

```
Forms("背景窗口").Open()
MainTable = Tables("表A")
```

这样打开窗口后，系统就会打开背景图片窗口，并切换到对应的表 ：

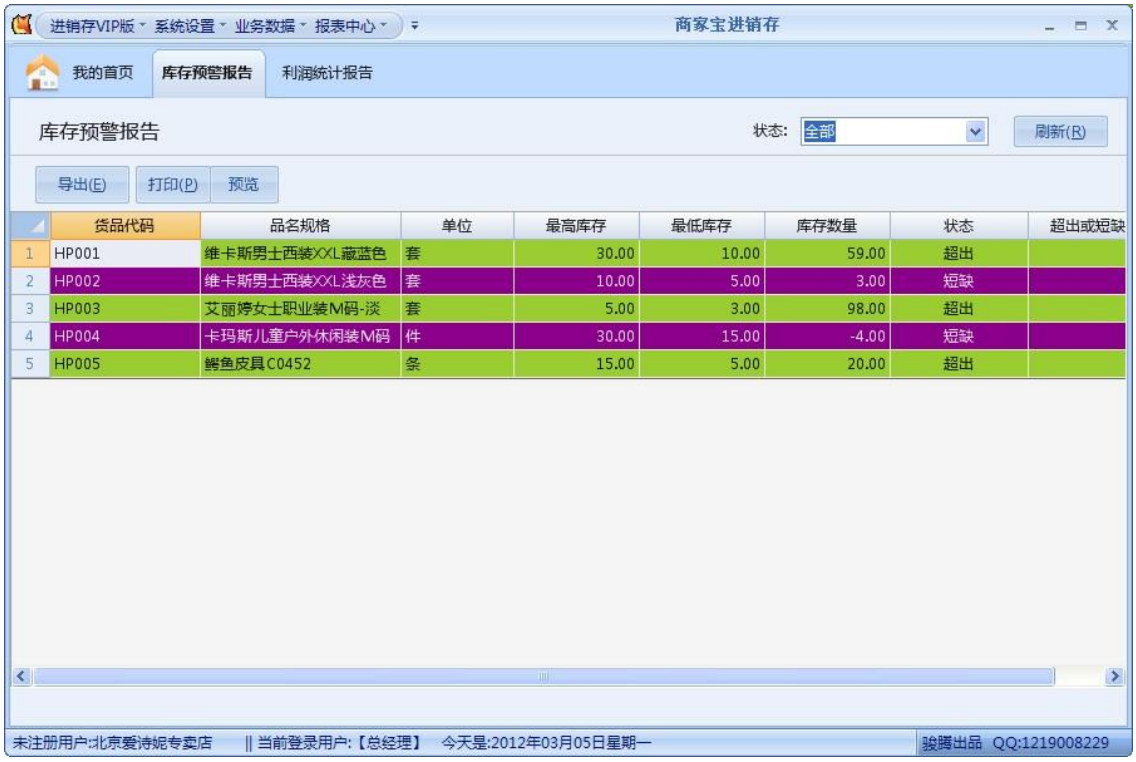


如果希望非开发者打开项目的时候，隐藏表标题，避免用户通过表标题切换表，可以将AfterOpenProject 事件代码改为：

```
Forms("背景窗口").Open()
MainTable = Tables("表 A")
If User.Type <> UserTypeEnum.Developer
    TableCaptionVisible = False
End If
```

用表标题导航

如果系统比较简单，一个表对应一个功能某块，我们可以直接用表标题切换，切换到某个表，也就是切换到某功能模块:



设计的方法很简单：

1、在项目事件 AfterOpenproject 加上代码：

```
Forms("窗口1").open()
MainTable = Tables("表A")
```

这样打开项目就会显示默认的表和窗口

2、在项目事件MainTableChaning中加上代码：

```
Select Case e.NewTableName
    Case "表A"
        Forms("窗口1").Open()
    Case "表B"
        Forms("窗口2").Open()
    Case "表C"
        Forms("窗口3").Open()
End Select
```

这样会在切换到对应的表之前，先打开相应的窗口，注意是在MainTableChanging事件设置代码，而不是在MainTableChanged事件设置代码，这样能避免屏幕闪烁。

3、不同的功能模块，有时也会对应不同的菜单，如果希望切换到对应的表之后，菜单也能同步切换到对应的功能 区，可以参考下面的代码：

```
Select Case e.NewTableName
    Case "表A"
        Forms("窗口1").Open()
        RibbonTabs("功能区1").Selected = True
    Case "表B"
        Forms("窗口2").Open()
        RibbonTabs("功能区2").Selected = True
    Case "表C"
        Forms("窗口3").Open()
        RibbonTabs("功能区3").Selected = True
End Select
```

4、既然用表标题切换功能模块，我们应该将表标题设置得漂亮一些，给不同的表加上一个不同的个性化图标是必须的，请参考：自定义表图标

5、建议将各个功能某块对应的窗口的类型设置为主窗口，这样表标题、窗口和菜单可以融为一体，当然这只是一个建议，并非绝对，窗口类型可根据需要选择。

用菜单功能区导航

如果使用自定义菜单，且一个功能区对应一个或一组功能，那么我们可以直接利用功能区来导航：



1、在项目事件 AfterOpenproject 加上代码：

```
Forms("窗口1").open()
MainTable = Tables("表A")
```

```
If User.Type <> UserTypeEnum.Developer
    TableCaptionVisible = False
End If
```

这样打开项目就会显示默认的表和窗口，同时如果当前用户不是开发者，则隐藏表标题，避免非开发者用户通过表标题切换表。

2、在菜单的全局事件 SelectTab 中设置代码：

```
Select Case e.Sender.Name
    Case "功能区A"
        Forms("窗口1").open()
        MainTable = Tables("表A")
    Case "功能区B"
        Forms("窗口2").open()
        MainTable = Tables("表B")
    Case "功能区C"
        Forms("窗口3").open()
        MainTable = Tables("表C")
    Case "功能区D"
        Forms("窗口4").open()
        MainTable = Tables("表C")
End Select
```

这样我们选择不同的菜单功能区，就会自动打开对应的功能窗口，并切换到对应的表。
提示：一个表可以设计多个主窗口，对应不同的功能，就像上面的表 C，有窗口 3 和窗口 4 两个主窗口，如果你愿意，你甚至可以将所有的主窗口都设计在同一个表中，这样就只需切换主窗口，而无需切换表了。
4、如果每个功能区对应有多个功能窗口，那么上面的代码只是打开默认的窗口，我们可以在设置这个功能区中加上其他功能的按钮，并相应设置代码，例如：

```
Forms("窗口 5").open()
MainTable = Tables("表 B")
```

使用导航窗口

如果系统比较大，功能模块比较多，我们可以设计一个导航窗口，窗口的类型设置为“共有”(必须的)，在窗口中插入一个 TopicBar 控件：
设置其 TopikLinkClick 事件代码为：

```
Select Case e.Link.Text
    Case "任务1"
        Forms("窗口1").Open()
        MainTable = Tables("表A")
    Case "任务2"
        Forms("窗口2").Open()
        MainTable = Tables("表B")
    Case "任务3"
```

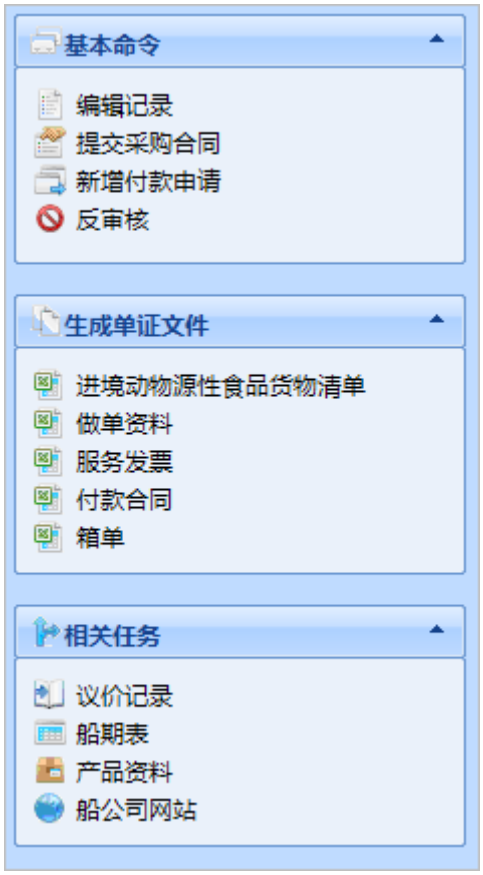
```
        Forms("窗口3").Open()
        MainTable = Tables("表C")
    '更多的任务
End Select
```

导航窗口的类型必须设置为共有，而窗口1、窗口2、窗口3这些功能窗口的类型通常为主窗口 ，这样导航窗口和工作窗口可以融为一体，看起来就是一个窗口。

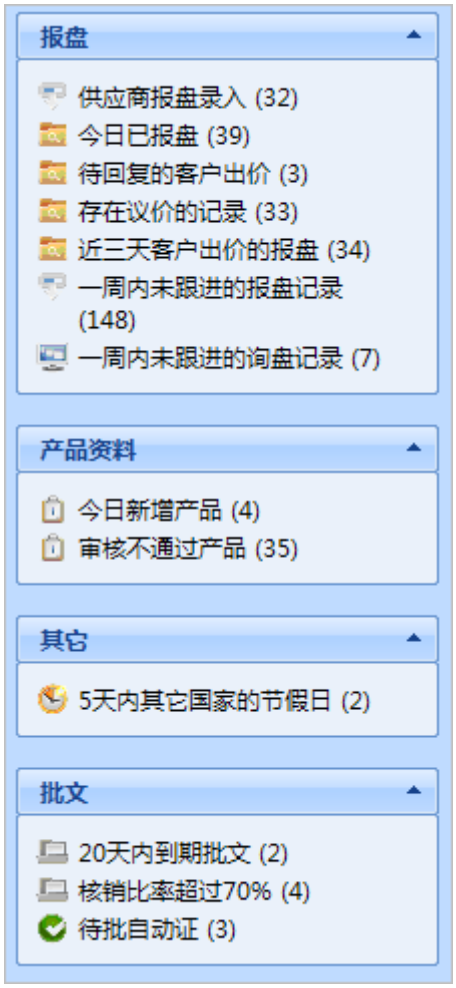
在AfterOpenproject事件中设置代码打开此导航窗口：

```
Forms("导航窗口").Open()
```

一个用TopicBar设计的导航窗口：



导航窗口并非只能进行导航，可以在导航的同时，显示一些提示和统计信息，例如：



要如上图所示，在TopicBar显示统计信息，通常在窗口的AfterLoad事件中加上类似的代码：

```
Dim Page As WinForm.TopicPage
Dim cnt1 As Integer = DataTables("产品").Compute("Count(主键列)","日期 = #"  & Date.Today & "#")
Dim cnt2 As Integer = DataTables("产品").Compute("Count(主键列)","审核 = False Or 审核 IsNull")
Page = e.Form.Controls("TopicBar1").Pages.Add("产品资料","产品资料")
Page.Links.Add("今日新增产品","今日新增产品(" & cnt1 & ")" )
Page.Links.Add("审核不通过产品","审核不通过产品(" & cnt2 & ")" )
```

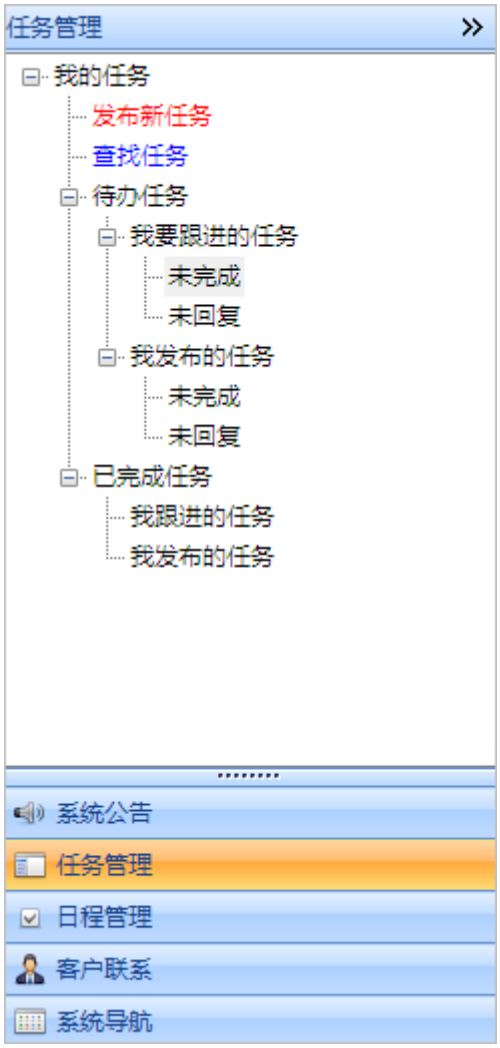
由于TopicLink的Text(标题)属性包括统计信息，其内容是动态变化的，而Name(名称)属性则始终保持不变，所以 给Topicbar的TopicLinkClick事件设置代码的时候，需要用TopicLink的Name属性来判断单击的是那个命令：

```
Select Case e.Link.Name
   Case "今日新增产品"
      Tables("产品").Filter = "日期 = #"  & Date.Today & "#"
   Case "审核不通过产品"
      Tables("产品").Filter = "审核 = False Or 审核 IsNull"
End Select
```

除了TopicBar，TreeView也常用于设计导航窗口，和TocpikBar相比，TreeView可以显示更多的内容。

双重导航的实现

如果你的系统很大，单单通过 TopicBar 或者 TreeView 导航还不够，可以靠用 NavBar 配合 TopicBar 和 TreeView 使用，同样设计以一个共享窗口作为导航窗口，导航窗口中插入一个 NavBar 控件，NavBar 的每个页面中插入一个 TopicBar 或者 TreeView，类似：



NavBar 的页面也可以直接插入其他任何控件，例如下面的窗口使用按钮来打开对应的功能：



通过设置按钮的风格，可以让按钮更美观，参考：关于 Button

如果不喜欢双重导航，我们也可以直接设计多个导航窗口，每个导航窗口对应一组类似的功能，通过菜单中的不同按钮分别打开对应的导航窗口。

多重导航的实现

如果需要更加清晰明了的导航界面，可以考虑多重导航，例如：



上图的导航窗口为一个主窗口，主窗口插入一个 SplitContainer 控件，SplitContainer 的左分页用 NavBar 控件导航，NavBar 各个页面插入一些按钮(当然也可以用 Treeview 或 TopicBar)，SplitContainer 的右分页插入一个 TabControl，在 TabControl 的各个页面中加上具体的功能按钮。

单击 NavBar 页面中的按钮切换 TabControl 的页面，参考：用代码切换 TabControl 的页面

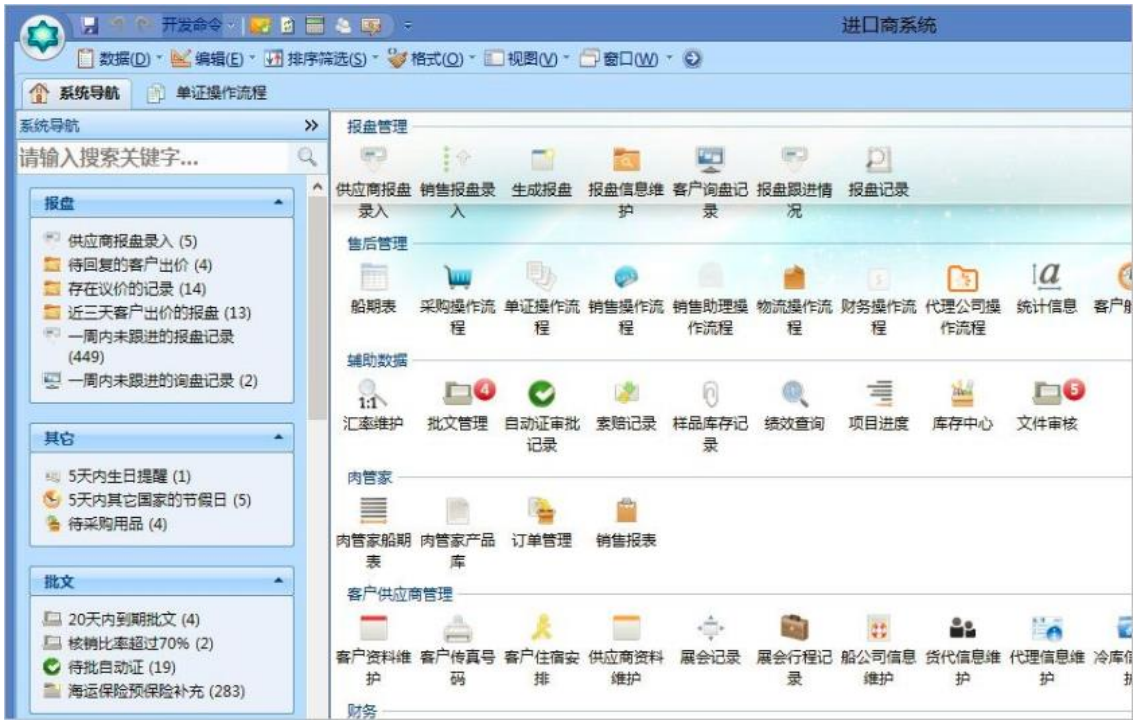
单击 TabControl 页面中的按钮打开具体的功能窗口:

```
Forms("窗口 5").open()
MainTable = Tables("表 B")
```

这些功能窗口可以是主窗口，也可以是独立窗口或模式窗口之类，如果功能窗口是主窗口，注意其所有者表不能和导航窗口相同，否则打开功能窗口就会自动关闭导航窗口，失去了导航的意义。

用 ListView 导航

如果你的系统非常大，可以考虑用 ListView 设计多重导航，例如：



上图的导航窗口为一个主窗口，主窗口插入一个 SplitContainer 控件，SplitContainer 的左分页用 TopicBar 控件导航，右分页用 ListView 控件对功能进行细分，单击 ListView 中项目，打开相应的功能窗口 。

使用并列窗口设计主界面

在设计主界面的时候，多数时候使用的是主窗口，其实并列窗口也是一个不错的选择，而且和主窗口相比，一个表可以同时打开多个并列窗口。

不过并列窗口的窗口标题默认显示在下方，如果用于设计主界面，可以在设计窗口的时候，将并列窗口的“标题位置”属性设置为“上”。

4.9.7.16 窗口级别的方法与变量

Foxtable 提供了内部函数和全局变量，用于在不同的事件之间复用代码和数据。

一些方法和变量，可能并不需要全局共享，例如在设计窗口的时候，某些方法和变量只需在本窗口共享，但是 Foxtable 并没有提供窗口级别的方法和变量，不过我们可以用变通的方法来实现。

我们可以在窗口中加入一个按钮(Button)，将其“可见”属性设置为 False，并将需要在该窗口中重复调用的方法代码，写在此按钮 Clik 的事件中。

窗口中的其它事件可以下面的方式执行此段代码：

```
e.Controls("按钮名称").PerformClick()
```

这样就间接实现了窗口级的方法共享。

同样我们可以在窗口中加入一个或多个隐藏的 Label，利用其 Text 属性作为窗口级别的变量。

当然你愿意的话，也可以全部采用内部函数和全局变量，这没有什么不好，看你的习惯，本节只是为了给大家多一个选择。

4.10 票据设计

4.10.1 用窗口设计票据

窗口可以直接打印，这样窗口可以成为一个既易用又强大的票据设计工具。

利用窗口设计票据的要点：

1、只有窗口中的控件才会打印，容器中的控件是不会打印的。

2、不是所有的控件都可以用于票据设计，目前可以用于票据设计的控件(也就是可打印的控件)有：Label、TextBox、ComboBox、DateTimePicker、NumericComboBox、Line、Table、PictureBox、Chart、CheckBox、RadioButton，Painter。

3、判断一个控件是否可以用于票据设计是很简单的，只要在属性窗口中包括“允许打印”属性，该控件都可以用于票据设计。



4、如果不希望某个控件在票据中输出，可以将其“允许打印”属性设为 False。

5、对于 Table，可以设置最小行数，行数不够的时候，会自动补空行 ；还可以设置是否打印行号列。

6、如果有多个 Table 要输出到票据中，请在垂直方向顺序排列，如果在水平方向排列，可能导致控件定位错误。

7、窗口设计器的文件菜单下，提供了页面设置功能：



8、窗口的票据设计功能是利用专业报表实现的，不过你无需研究专业报表，直接使用下面的代码即可。

预览票据：

```
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

打印票据：

```
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Print()
```

4.10.2 和打印相关的控件属性

窗口中所有可打印的控件，都有以下属性：

- **Printable**
  逻辑型属性，表示是否打印此控件。
  默认为 True，设为 False，将不打印此控件。

- **NewPage**
  逻辑型属性，在打印此控件之前，是否另起一页。

Table 控件除上述属性外，还有三个和打印相关的属性：

- **MinRows**
  整数型属性，表示每页最小应该打印的行数。
  默认为 0，按实际行数打印，
  如果设置为一个非零整数，实际行数达不到此设置值的时候，将自动补空行。

- **MaxRows**
  整数型属性，表示每页最多打印的行数，当 Table 的行数超过 MaxRows 的值时，将自动另起一页打印。
  默认为 0，表示不限制每页行数

- **PrintRowNumber**
  逻辑型属性，表示是否打印行号。
  默认为 True，如果设为 False，将不打印行号。

提示：一定要在执行 Form 的 GernatePrintDoc 方法前，设置控件的打印属性，否则对本次打印无效。

示例

```
e.Form.Controls("Table1").MaxRows = 5
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

实际上，我们很少需要通过代码设置打印属性，通常会在窗口设计的时候指定相关属性：



4.10.3 页面设置

前面提到，利用窗口设计器设计票据的时候，菜单中已经提供了页面设置功能。
但是不同的打印机，页边距的设置值可能会有所不同，对于交付使用的系统来说，我们通常不会开放窗口设计功能，让用户自己去调整票据的页边距。
能否自己设计一个打印窗口，让用户在打印票据之前，可以自己调整页边距呢？当然可以！
Form 有一个 Page 属性，专门用于页面设置，该属性包括以下子属性，我们会逐个介绍这些属性的使用方法：

PaperKind: 纸张类型代码
Landscape: 逻辑型型，是否横向打印
Height: 设置页面高度，单位为毫米
Width: 页面宽度
LeftMargin: 左边距，单位为毫米
RightMargin: 右边距，单位为毫米
TopMargin: 上边距，单位为毫米
BottomMargin: 底边距，单位为毫米

提示：页面设置代码，必须在 GernatePrintDoc 方法之前执行，否则可能无效。

设置纸张

PaperKind 属性是一个整数，用于设置纸张类型，所有的纸张类型都有一个整数表示。

示例

例如将纸张类型改为 B5:

```
e.Form.Page.PaperKind = 13 '纸张类型改为 B5
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

下表列出了常用的纸张类型：

值	纸张类型
8	A3
9	A4
11	A5
12	B4
13	B5
66	A2
70	A6

纸张类型很多，全部纸张类型，请参考：纸张大全

横向打印

将 Landscape 属性设为 True，即可横向打印。

示例

```
e.Form.Page.Landscape = True '横向打印
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

自定义纸张

要使用自定义纸张，只需直接设置纸张的宽度和高度即可，单位为毫米。

示例

例如使用一个 10 厘米宽，12 厘米高的纸张：

```
e.Form.Page.Width = 100 '纸张宽度为 100 毫米
e.Form.Page.Height = 120 '纸张高度为 120 毫米
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

设置页边距

下面的代码，左右边距设为 1 厘米，上下边距设为 2 厘米：

```
e.Form.Page.LeftMargin = 10 '设置左边距
e.Form.Page.RightMargin = 10 '设置右边距
e.Form.Page.TopMargin = 20 '设置上边距
e.Form.Page.BottomMargin = 20 '设置下边距
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

4.10.4 其它设置

GernatePrintDoc 生成一个 PrintDoc 对象，以后如果你学习了专业报表，就会知道 PrintDoc 其实就

是一个专业报表。

利用 PrintDoc 的一些属性，我们可以进行一些其他设置.

指定打印机

如果安装了多个打印机，且需要在非默认打印机上打印，可以设置 PrintDoc 的 PrinterName 属性，在指定的打印机上打印。

例如

```
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
Doc.PrinterName = "HP LaserJet 6L"
doc.Print()
```

可以用全局变量 InstalledPrinters 获得所有已经安装的打印机的名称。

禁止自动旋转

打印票据的时候，系统默认会自动根据实际纸张的大小，判断是否需要旋转打印内容，以便更好地配合纸张。

如果不需要自动旋转打印内容，可以将 AutoRotate 属性设为 False。

例如：

```
e.Form.Page.Width = 150
e.Form.Page.Height = 80
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.AutoRotate = False '禁止自动旋转打印内容
doc.Print()
```

你可以删除下面这一行代码，重新打印一张，看看结果有什么不同:

```
doc.AutoRotate = False '禁止自动旋转打印内容
```

4.10.5 DataFormat 事件和票据打印

利用窗口设计实现票据打印，最大的优势在于精确定位和数据绑定。

但是票据打印有自己的特殊要求，例如金额可能希望用大写中文格式输出，此时简单的数据绑定是无法达到目的的。

还记得 DataFormat 事件吗？这个事件在控件从表中取得数据的时候执行，利用该事件，我们可以“篡改”读取的数据，使得数据按照我们希望的格式显示。

DataFormat 事件的 e 参数属性有：

Sender：返回触发事件的控件

Form: 返回控件所在的窗口

Value： 返回控件的值，也就是从字段(列)或变量中读取的值。

Text： 用于设置控件要显示的内容。

示例

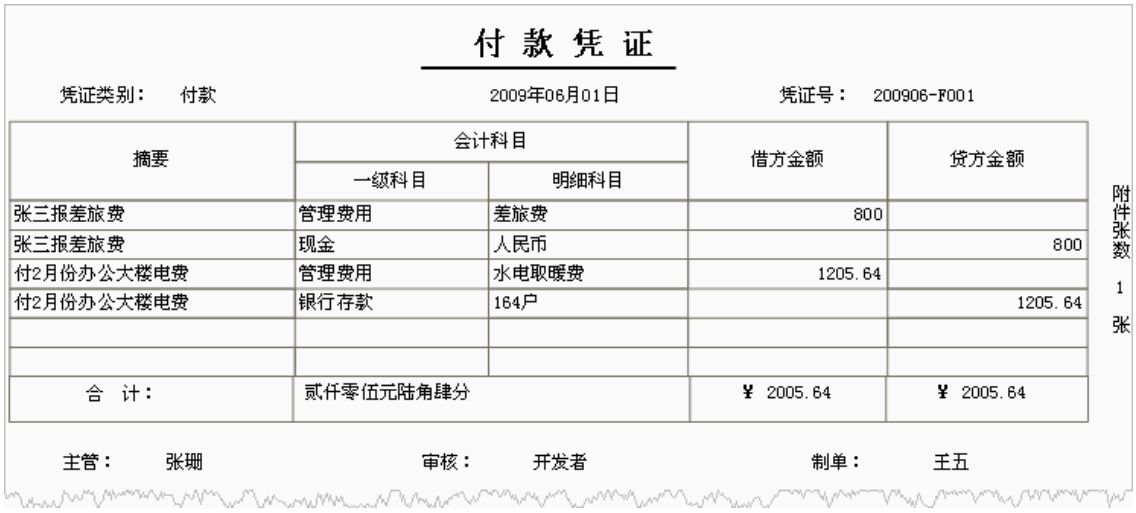
假定某个 Label 控件绑定到了金额列，希望能够按照中文大写形式显示金额，只需将该控件的 DataFormat 事件设为：

```
e.Text = CUMoney(e.Value)
```

4.10.6 实战演练一

本节的示例，可以参考 CaseStudy 目录下的文件：票据打印.Table。

该文件有两个表，分别为凭证和凭证明细，已经通过凭证 ID 建立关联。
本节的目的是设计一个窗口，用于打印一个下图所示的凭证：



付 款 凭 证

凭证类别： 付款　　2009年06月01日　　凭证号： 200906-F001

摘要	会计科目		借方金额	贷方金额
	一级科目	明细科目		
张三报差旅费	管理费用	差旅费	800	
张三报差旅费	现金	人民币		800
付2月份办公大楼电费	管理费用	水电取暖费	1205.64	
付2月份办公大楼电费	银行存款	164户		1205.64
合 计：	贰仟零伍元陆角肆分		¥ 2005.64	¥ 2005.64

附件张数 1 张

主管： 张珊　　审核： 开发者　　制单： 王五

设计步骤

1、首先新建一个窗口，然后在窗口设计器的“文件”菜单中，单击“页面设置”命令，设置好页面尺寸和边距。

2、在窗口插入一个 Table 控件，将其绑定到关联表“凭证.凭证明细”，并将其“最小行数”属性设置为 6，这样当订单明细不够 6 行时，会自动补空行。

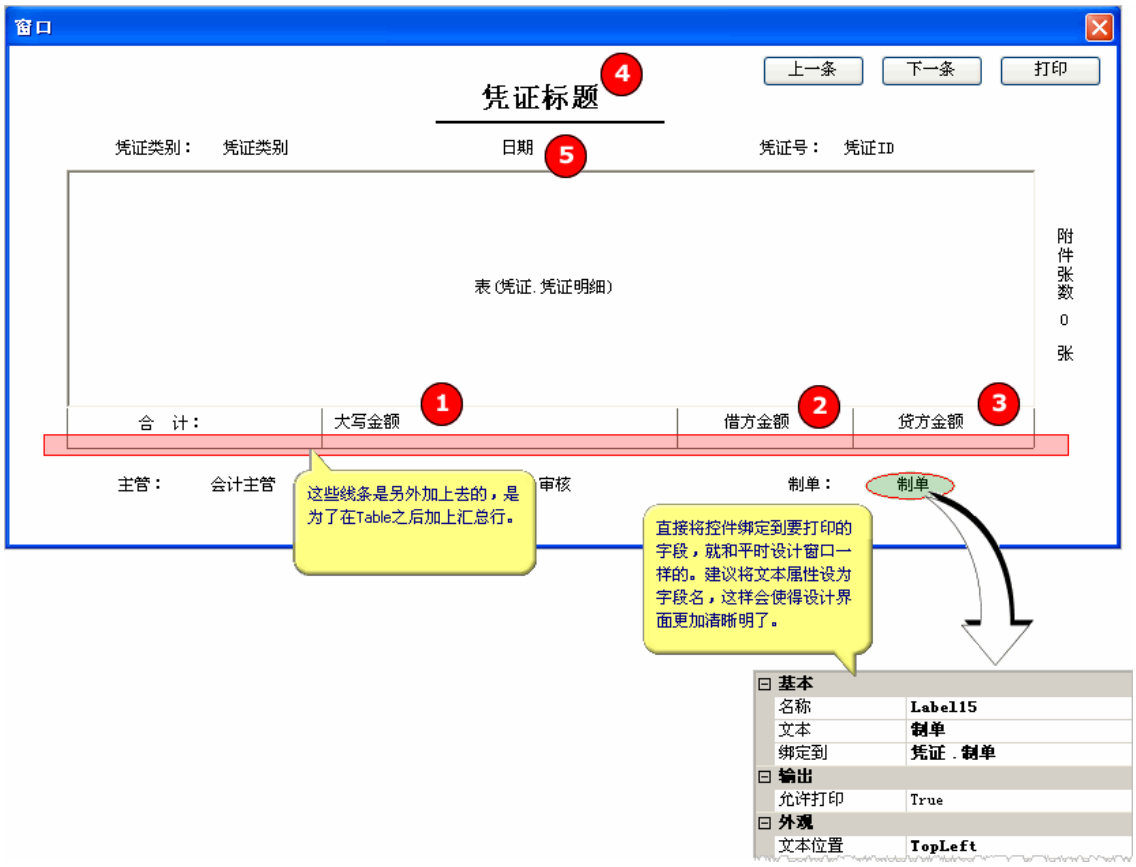
3、设置 Table 控件的“显示列”属性设置为：

摘要|186|会计科目_一级科目|126|会计科目_明细科目|130|借方金额|130|贷方金额|130



实际上很少能一次性地设置好"显示列"属性，上面的设置是反复调试后的结果。

4、按照常规方法，设计好窗口，并将相关控件绑定到字段：



5、上图中标有数字的控件，须在 DataFormat 事件中设置代码，使得打印格式符合我们的要求:

编号	说明	DataFormat 事件代码
1	大写金额	e.Text = CUMoney(e.Value)
2	借方金额	e.Text = "￥ " & e.Value
3	贷方金额	e.Text = "￥ " & e.Value

4	凭证标题	If e.Value IsNot Nothing Then
		e.Text = e.Value.Chars(0) & " " & e.Value.Chars(1) & " 凭 证"
		End If

5	日期	e.Text = Format(e.Value,"yyyy 年 MM 月 dd 日")

6、设置按钮的 Click 事件代码。

“上一条”按钮的 Click 事件代码设为：

```
With Tables("凭证")
    .Position = .Position - 1
End With
```

“下一条”按钮的 Click 事件代码设为：

```
With Tables("凭证")
    .Position = .Position + 1
End With
```

“打印”按钮的 Click 事件代码设置为：

```
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview()
```

多页凭证的设计

如果明细表的行数比较多，可能一份凭证需要分多页打印，要完成这个任务很简单，只需设置最大行数属性即可：



上图最小行数和最大行数属性都设置为 6，这样明细的行数不到 6 行时，会自动补充空行，超过 6 行时，会分页打印凭证，每页打印 6 行。

4.10.7 实战演练二

本节的示例，可以参考 CaseStudy 目录下的文件：票据打印.Table。

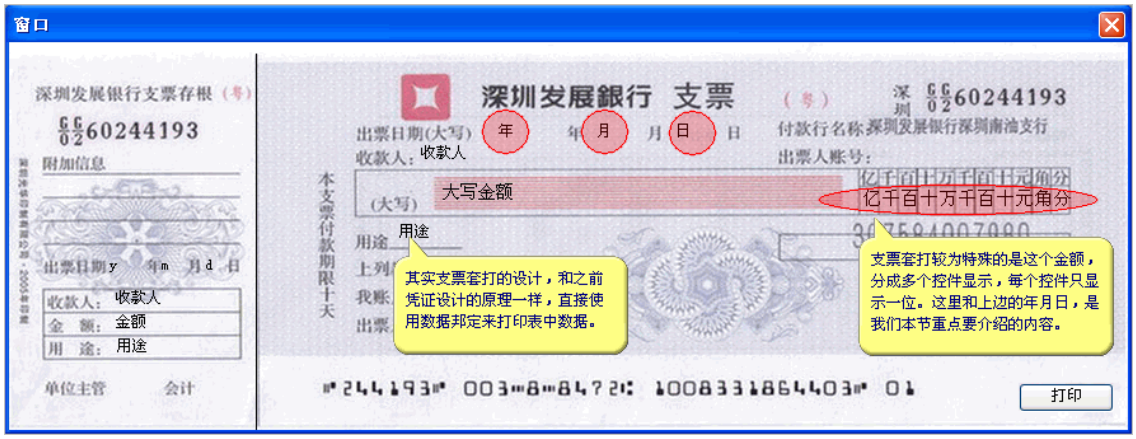
该文件有一个支票表，我们要在该表设计一个窗口，用于套打支票：



设计步骤

1、首先新建一个窗口，然后在窗口设计器的文件“菜单”中，单击“页面设置”命令，设置好页面尺寸和边距。

2、为便于定位，我们可以扫描一个空白的支票图片，作为窗口背景使用，然后再插入各个控件：



3、将各控件分别绑定到对应的字段；再次提示一下，控件绑定到某个字段后，我们应该将其文本值设为绑定字段名称，这样让窗口看上去会比较清晰。

4、现在介绍如何打印小写金额，Foxtable 提供了一个名为 GetDigit 的函数，专门用于解决这个问题。首先将用于打印小写金额的所有控件全部绑定到金额列，然后分别设置各控件的 DataFormat 事件代码：

分： e.Text = GetDigit(e.Value,-2)

角： e.Text = GetDigit(e.Value,-1)

元: e.Text = GetDigit(e.Value,0)

十： e.Text = GetDigit(e.Value,1)

百： e.Text = GetDigit(e.Value,2)

千： e.Text = GetDigit(e.Value,3)

....

5、大写日期的解决办法类似，同样将年、月、日三个控件绑定到日期列，然后分别设置其 DataFormat 事件代码：

年：

```
If e.Value IsNot Nothing Then
    e.Text = CCNumber(e.Value.Year)
End If
```

月：

```
If e.Value IsNot Nothing Then
    e.Text = CCNumber(e.Value.Month)
End If
```

日：

```
If e.Value IsNot Nothing Then
    e.Text = CCNumber(e.Value.Day)
End If
```

注意上面的代码中一定要先判断 e.Value 是不是空值，否则运行过程中，e.Value.Year 这样的代码会出错。

6、存根部分的小写日期和上面的处理方式一样，只是不需要用 CCNumber 函数转换为中文而已：
年：

```
If e.Value IsNot Nothing Then
    e.Text = e.Value.Year
End If
```

月：

```
If e.Value IsNot Nothing Then
    e.Text = e.Value.Month
End If
```

日：

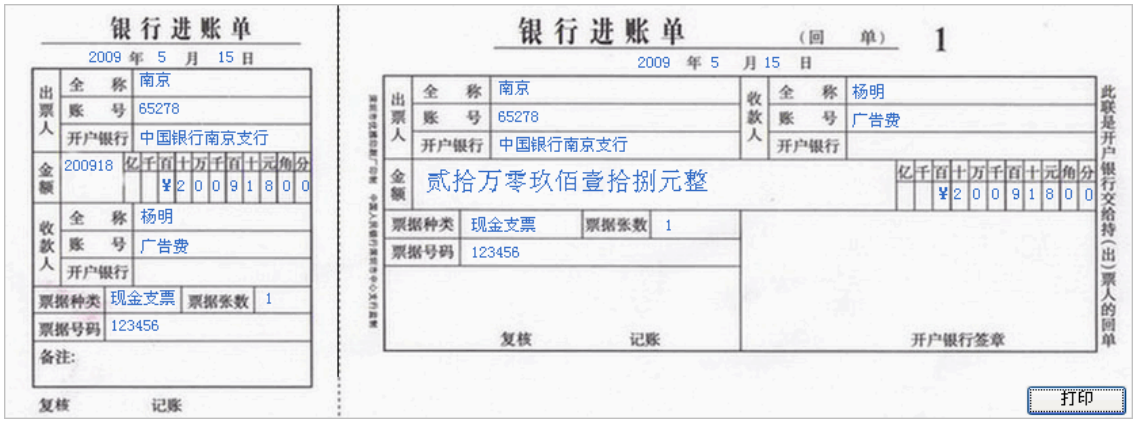
```
If e.Value IsNot Nothing Then
    e.Text = e.Value.Day
End If
```

7、最后将大写金额的 DataFormat 事件代码设为：

```
e.Text = CUMoney(e.Value)
```

4.10.8 实战演练三

本节的示例，可以参考 CaseStudy 目录下的文件：票据打印.Table。
该文件有一个进账单表，我们要在该表设计一个窗口，用于套打下面的进帐单：



示例文件已经设计好了窗口，您可以参考一下。设计方法和前面介绍的类似，这里就不重复了。

4.10.9 统一录入和打印窗口

前面介绍的利用窗口打印票据，都是用 Label 控件来实现的。
Label 控件无法用于输入数据，所以这些窗口仅仅只能用于票据打印。
如果希望一个窗口同时用于数据录入和票据打印，可以用 TextBox、ComboBox、CheckedComboBox、

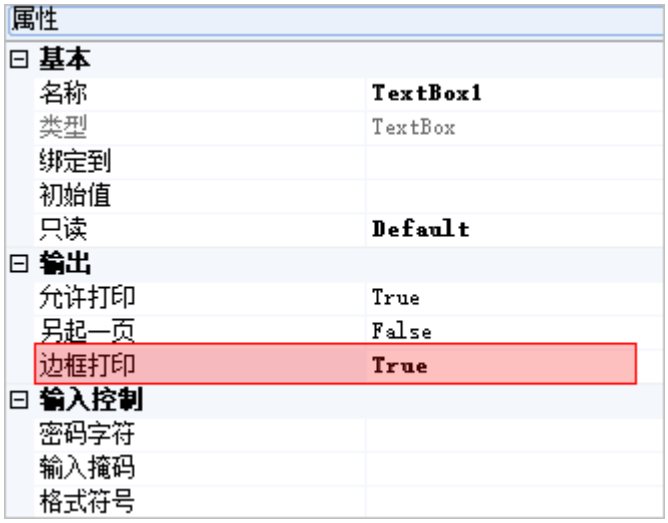
DateTimePicker 和 NumericComboBox 等录入控件代替 Label。
假定某个 TextBox 控件绑定到了金额列，希望能够按照中文大写形式显示金额，只需将该控件的 DataFormat 事件设为：
e.Text = CUMoney(e.Value)
如果金额是 14250，那么 TextBox 显示的内容为：壹万肆仟贰佰伍拾元整，如果控件获得焦点，那么显示的值依然为 14250，这样就不会影响正常的数据录入 ：



此功能需要最新版本的 Foxtable 支持。

4.10.10 关于边框打印

尽管 TextBox 和 ComboBox 等控件可以设置边框样式和边框颜色，但是默认情况下，边框是不会打印出来的。
如果希望将内容和边框同时打印出来，可以选择对应的控件，将边框打印属性设置为 True:



4.10.11 隐藏行号列

如果要利用 Table 控件来设计票据，可以在设计窗口的时候，将 Table 控件的“显示行号列”属性设置为 False。
通过下图可以看出，在设计票据的时候，隐藏行号列更便于定位：



4.10.12 背景图片和 DPI

前面在介绍支票和进账单设计的时候，都使用了背景图片来定位。

背景图片是通过扫描实际的支票和进账单得到的，在实际使用的时候，经常会有客户反应在使用背景图片定位的时候，打印位置和设计位置不一致，这时因为图片和屏幕的 DPI 参数不一致造成的。

在扫描图片的时候，应该在选项设置中，将图片的 DPI 设置为屏幕的 DPI，这样打印位置和设计位置才会一致。

屏幕的 DPI 通常是 96。

4.10.13 打印指定容器中的控件

Foxtable 默认是打印整个窗口，不过 Foxtable 也可以单单打印某个容器中的控件。

例如要打印一个 GroupBox 中的内容：

```
Dim grp As WinForm.GroupBox = e.Form.Controls("GroupBox1")
Dim doc As PrintDoc = e.Form.GernatePrintDoc(grp)
doc.Preview()
```

再例如，窗口中有一个 TabControl 控件，希望打印该控件选定页面的内容，可以参考下面的代码：

```
Dim tb As WinForm.TabControl = e.Form.Controls("TabControl1")
Dim pg As WinForm.TabPage = tb.SelectedPage
Dim doc As PrintDoc = e.Form.GernatePrintDoc(pg)
doc.Preview()
```

说明：GernatePrintDoc 方法有一个参数，用于指定要打印的容器，如果省略，则打印整个窗口。

4.10.14 用复选框模拟单选框

普通用户可以忽略本节问题。

我们知道通过将 RadioButton（单选框）放在不同的容器中，可以对其进行分组，同一容器中的多个单选框，其选中状态是互斥的，选中一个单选框，之前选中的同组单选框将被自动撤销选中状态。

这有一个问题，就是在打印窗口的时候，容器中的控件是不会打印的，所以所有的 RadioButton 必须直接置于窗口中，但只这样一来就不能对 RadioButton 分组。

这个问题是无解的，RadioButton 置于容器中可以分组，但是不能打印，置于窗口中可以打印，但是不能分组。

我们只能用折中的方案来解决这个问题，用 CheckBox(复选框)来模拟 RadioButton，通过设置代码，使得选择某个复选框后，取消同一分组其它复选框的选中状态。

例如在窗口加入 6 个 CheckBox，其中 CheckBox1、CheckBox2、CheckBox3 为一组，CheckBox4、CheckBox5、CheckBox6 为一组，为了实现 RadioButton 的效果，可以在窗口的全局事件 CheckedChanged 加入代码：

```
Dim nm As String = e.Sender.Name
Select Case nm
    Case "CheckBox1","CheckBox2","CheckBox3"
        If e.Sender.Checked Then
            e.Form.Controls("CheckBox1").Checked = (nm = "CheckBox1")
            e.Form.Controls("CheckBox2").Checked = (nm = "CheckBox2")
            e.Form.Controls("CheckBox3").Checked = (nm = "CheckBox3")
        End If
    Case "CheckBox4","CheckBox5","CheckBox6"
        If e.Sender.Checked Then
            e.Form.Controls("CheckBox4").Checked = (nm = "CheckBox4")
            e.Form.Controls("CheckBox5").Checked = (nm = "CheckBox5")
            e.Form.Controls("CheckBox6").Checked = (nm = "CheckBox6")
        End If
End Select
```

再次提示，上述代码必须设置在窗口的全局事件 CheckedChanged 中，而不是单个控件的 CheckedChanged 事件中。

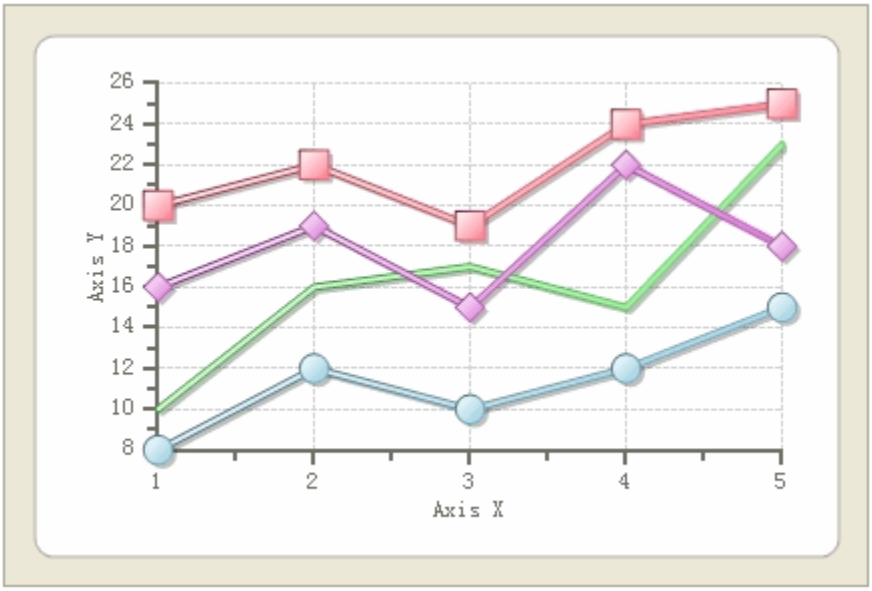
上述代码是针对非绑定状态的 CheckBox，如果 CheckBox 已经绑定到列，那么请参考：逻辑列的排他式选择

4.11 生成图表

4.11.1 图表基础

4.11.1.1 图表入门

要设计出一个吸引人的系统，图表是必不可少的。
在设计窗口的时候，插入一个 Chart(图表)，你会发现，竟然没有任何特殊的属性可以设置。
但是别小看它，有了它，你可以用短短的几行代码，就可设计出这样的图表：

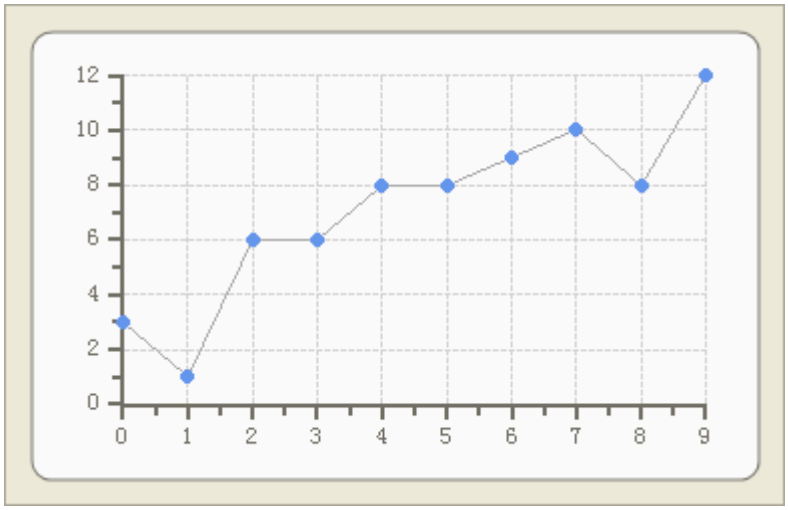


Foxtable 提供了 13 种图表，上图只是其中的一种(XYPlot)，你甚至可以用它来生成复杂的股票分析图。
一个图表(Chart)包括几个图系(CharSeries)，上图中每一条不同颜色的线就是一个图系，一个图系由多个数据点组成，而数据点的坐标位置就表示具体的数据。
要绘制一个图表，最简单的步骤就是给图表增加图系，然后给这些图系添加数据点。
示例一
我们先来生成一个最简单的图表，在窗口中分别插入一个图表和一个按钮，将按钮的 Click 事件代码设置为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= e.Form.Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '新增图系包括 10 个数据点
```

```
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
```

单击按钮，会生成下图所示的图表：

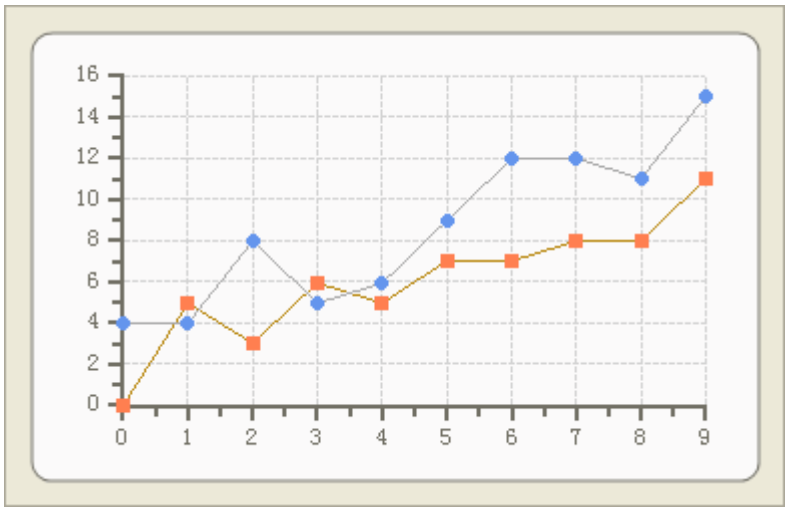


示例二

我们再改一下按钮的 Click 事件代码，这次不是一个图系，而是两个图系：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
For n As integer = 0 To 1
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 10 '图系包括 10 个数据点
    For i As integer = 0 to 9 '指定每个数据点的位置
        Series.X(i) = i '指定水平坐标
        Series.Y(i) = i + n * 2 + Rand.Next(5) '垂直坐标用随机数生成
    Next
Next
```

这次单击按钮生成的图表是：



你也许感到奇怪，怎么这两幅图片的效果和第一副图片不同呢？
其实这两幅图片是默认的效果，如果你希望得到第一副图片的效果，在代码中加上下面这一句即可：

```
Chart.VisualEffect = True
```

还有一件怪事，每单击一次按钮，数据线的颜色和标记点的形状都会发生变化？ 这是因为我们没有明确指定图系的属性。
实际上，只要你愿意，你可以精确控制图表的每一个方面，我们会在后面讲述。

4.11.1.2 绑定到表

实际应用的时候，图示的数据往往直接来自于数据表，例如对于下面这个表，如何来图示呢。

	月份	数量
1	1	7560
2	2	8990
3	3	8420
4	4	10910
5	5	10530
6	6	5960

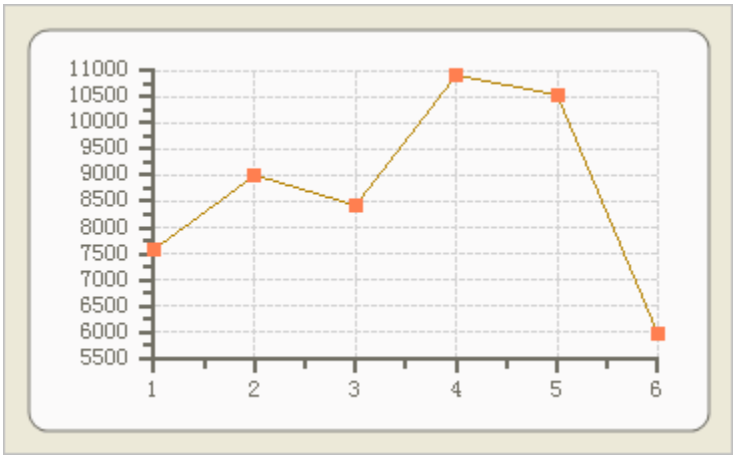
方法有两种，一种是编码增加图系和数据点，另一种是绑定。
同样在窗口分别中插入一个图表和一个按钮，将按钮的 Click 事件代码设置为示例代码。
接下来所有生成图表的示例代码，测试方法都一样，就不再重复提示了。

方法一

编码增加图系和数据点。
新建一个窗口，在窗口中分别插入一个图表和一个按钮，将按钮的 Click 事件代码设置为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart= e.Form.Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = t.Rows.Count '图系的数据点数等于表的行数
For i As integer = 0 to t.Rows.Count - 1 '指定每个数据点的位置
    Series.X(i) = t.Rows(i)("月份") '指定水平坐标
    Series.Y(i) = t.Rows(i)("数量") '指定垂直坐标
Next
```

这是生成的图表：



我在编写上面这段代码的时候，栽了一个小跟斗，运行的时候提示"月份"列不属于"统计表 1"，原来这个表是通过分组统计得到的，标题和列名有的时候并不会一致。

方法二

将图表绑定到数据表。
将按钮的 Click 事件代码设置为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= e.Form.Controls("Chart1") ' 引用窗口中的图表
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "月份" 'X 轴绑定到月份列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
```

执行效果和方法一完全一样，但是如果你修改表中的数据，生成的图表会立即自动更新，而方法一不会。

4.11.1.3 非数值字段

我们已经知道，指定一个数据点需要用两个数值，但是有的时候数据列并不是数值型的，例如下面的表：

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

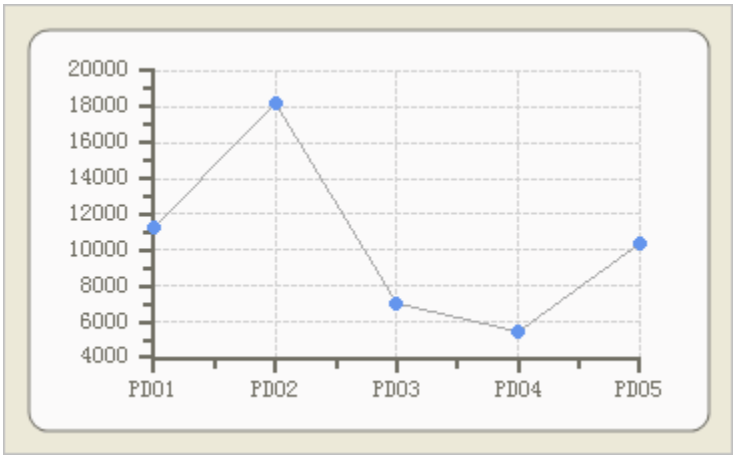
上表中产品列是字符型，并不是数值型，应该如何来图示这个数据表呢？

方法一

如果采用绑定的方法，那么很简单，和原来一样设置代码即可。
新建一个窗口，在窗口中分别插入一个图表和一个按钮，将按钮的 Click 事件代码设置为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart = e.Form.Controls("Chart1") ' 引用窗口中的图表
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
```

下图是生成的图表，你会发现，连 X 轴的刻度标记也自动转换为产品名称了：



方法二

下面采用编码的方式来处理非数值字段，其实原理很简单，为每一个数值指定一个字符型表示即可：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
```

```
Chart= e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = t.Rows.Count '设置图系的长度
For i As integer = 0 to t.Rows.Count - 1 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = t.Rows(i)("数量") '指定垂直坐标
    Chart.AxisX.SetValueLabel(i, t.Rows(i)("产品")) '指定字符表示
Next
Chart.AxisX.AnnoWithLabels = True '启用字符标示
```

代码比方法一复杂一点，而且首次接触到一个新的对象 AxisX。

AxisX 表示 X 坐标轴（就是水平方向的坐标轴），坐标轴类型有一个 SetValueLabel 方法，用于指定某一数值的字符型表示，例如：

```
Chart.AxisX.SetValueLabel(1, "一月份")
```

这样坐标轴标记刻度的时候，会用字符“一月份”代替数值 1。

除了设置数值的字符表示，还需要将坐标轴的 AnnoWithLabels 属性设置为 True，这样坐标轴才会用数字的字符表示来标记刻度。

4.11.1.4 图示多列

前面的例子都是图示一个数据列，这次复杂一点，我们来图示下个表中的所有数据：

	产品	EP01	EP02	EP03	EP04	EP05
1	PD01	3480	2840	2940	990	1040
2	PD02	2910	4080	3030	5270	2910
3	PD03	440	1340	2770	1590	860
4	PD04	250	1470	1380	1890	490
5	PD05	590	2440	4420	2240	710

方法一

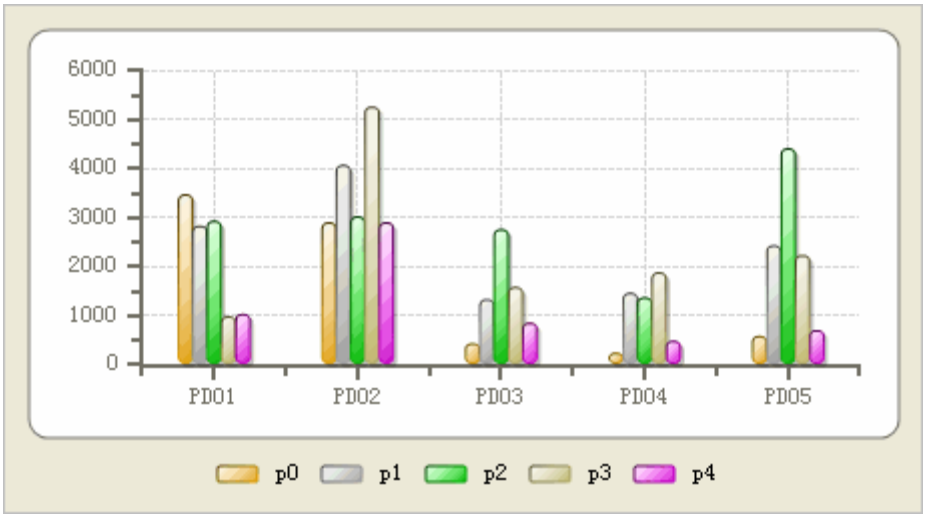
采用数据绑定的方式。

新建一个窗口，在窗口中分别插入一个图表和一个按钮，将按钮的 Click 事件代码设置为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart = e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型改为 Bar(条形)
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
For Each c As Col in t.Cols
    If c.Name <> "产品" Then
        Series = Chart.SeriesList.Add() '增加一个图系
        Series.Text = c.Name '设置图系的标题
        Series.X.DataField = "产品" 'X 轴绑定到产品列
        Series.Y.DataField = c.Name '设置 Y 轴的绑定列
    End If
```

```
Next
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South '图列显示在南方(底端)
```

生成的图表：



上面的代码接触到几点新的知识,首先通过下面的代码设置了新的图表类型：

```
Chart.ChartType = ChartTypeEnum.Bar
```

其次设置了每个图系的标题：

```
Series.Text = c.Name
```

在有多个图系的时候，最好设置每一个图系的标题，图系的标题显示在图例中，用于告诉用户该图系显示的是什么数据。

最后通过下面的代码，在指定的位置显示图例：

```
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South '图列显示在南方(底端)
```

不需要去深究上述新接触的属性，你只需要了解这些属性的作用，在后面的章节中，我们会详细介绍每一种属性。

方法二

直接编码生成：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart = e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型改为 Bar(条形)
Chart.SeriesList.Clear() '清除图表原来的图系
For Each c As Col in t.Cols
    If c.Name <> "产品" Then
        Series = Chart.SeriesList.Add() '增加一个图系
        Series.Text = c.Name '设置图系的标题
        Series.Length = t.Rows.Count '设置图系的长度
        For r As Integer = 0 To t.Rows.Count - 1
            Series.X(r) = r
            Series.Y(r) = t.Rows(r)(c.Name)
```

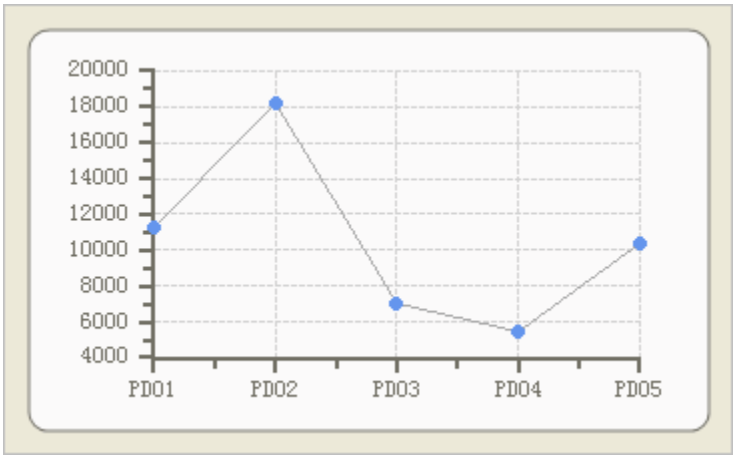
```
        Next
    End If
Next
For r As Integer = 0 to t.Rows.Count - 1
    Chart.AxisX.SetValueLabel(r, t.Rows(r)("产品")) '指定字符表示
Next
Chart.AxisX.AnnoWithLabels = True '启用字符标示
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South '图列显示在南方(底端)
```

4.11.1.5 跳过统计表

既然用数据绑定更简单，有时为什么要弃简从繁，采用手工编码呢？
原因很简单，用数据绑定，你必须预先生成统计表，而采用编码方式，可以直接在代码中加入统计代码，无需生成统计表。
同样是图示不同产品的销售数量，这次我们直接编码统计，而不是借助于统计表。
新建一个窗口，在窗口中分别插入一个图表和一个按钮，将按钮的 Click 事件代码设置为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim tbl As DataTable = DataTables("订单") '定义一个变量 tbl 引用数据表
Dim lst As List(Of String) = tbl.GetValues("产品") '获得产品名称集合
Chart = e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = lst.Count '设置图系的长度
For i As integer = 0 to lst.Count - 1 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = tbl.Compute("Sum(数量)","[产品] = '" & lst(i) & "'") '指定垂直坐标
    Chart.AxisX.SetValueLabel(i, lst(i)) '指定字符表示
Next
Chart.AxisX.AnnoWithLabels = True '启用字符标示
```

生成的图表：



4.11.1.6 启用双图表

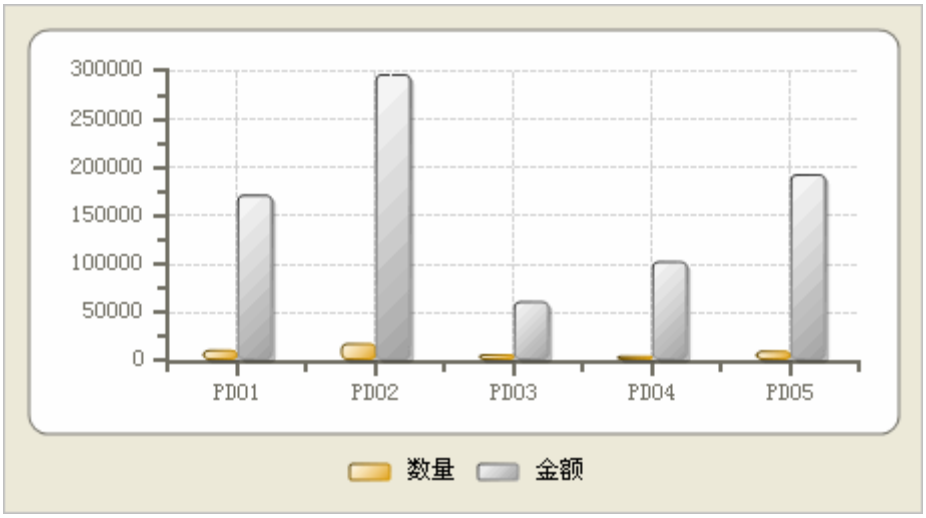
下表的是每一种产品的销售数量和金额:

	产品	数量	金额
1	PD01	11290	171754.92
2	PD02	18200	296522.5
3	PD03	7000	61490
4	PD04	5480	102861
5	PD05	10400	193023.15

首先按照常规方法，用下面的代码来图示该表：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart = e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加第一个图系
Series.Text = "数量"
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Series = Chart.SeriesList.Add() '增加第二个图系
Series.Text = "金额"
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "金额" 'Y 轴绑定到数量列
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South '图列显示在南方(底端)
```

生成的图表为：



可以看到，由于金额和数量的数值范围差距太大，上述的图表不能明显看出每种产品的销售数量差别。

为了改变这种效果，我们将代码稍微修改一下：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart = e.Form.Controls("Chart1") '  引用窗口中的图表
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.ChartType = ChartTypeEnum.Bar '图表 1 类型该为 Bar(条形)
Chart.ChartType2 = ChartTypeEnum.XYPlot '图表 2 类型该为线形
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '给图表 1 增加一个图系
Series.Text = "数量"
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Chart.SeriesList2.Clear() '清除图表原来的图系
Series = Chart.SeriesList2.Add() '给图表 2 增加一个图系
Series.Text = "金额"
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "金额" 'Y 轴绑定到数量列
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South '图列显示在南方(底端)
Chart.AxisY.Text = "数量"
Chart.AxisY2.Text = "金额"
```

生成的图表为：



最关键的变动是我们将下面的一行：

```
Series = Chart.SeriesList.Add() '增加第二个图系
```

改成了：

```
Series = Chart.SeriesList2.Add() '给图表 2 增加一个图系
```

为什么多了一个数字 2 会造成如此大差别呢？

实际上每一个 Chart 都有两个图表，他们的图系集合分别是 SeriesList 和 SeriesList2，所以上述的改动实际上就是将金额图系显示在第二个图表中。

两组图表共用 X 轴，而他们的 Y 轴是独立的，分别是 AxisY 和 AxisY2。

由于有两个图表，所以分别指定每个图表的类型：

```
Chart.ChartType = ChartTypeEnum.Bar '图表 1 类型该为 Bar(条形)
Chart.ChartType2 = ChartTypeEnum.XYPlot '图表 2 类型该为线形
```

同样因为有两个 Y 轴，我们需要明确指定每个 Y 轴的标题，这样才能知道每个图表图示的是那一列的数据：

```
Chart.AxisY.Text = "数量"
Chart.AxisY2.Text = "金额"
```

同时采用两个图表的情况极为少见，本来不打算讲述，但是客户总会有“不正常”的要求，大家了解一下也不错。

4.11.1.7 数据标示

一、动态标示

如果鼠标移到某一个数据点的时候，能够动态提示该点的一些信息(例如该点的值)，这样的图表会更友好一些。

利用图系的 ToolTipText 属性，可以很轻松地实现这样的功能。

ToolTipText 可以使用一些代码来标示数据，可使用的代码如下：

#XVAL： X 值

#YVAL： Y 值

#Y1VAL： Y1 值

#Y2VAL： Y2 值

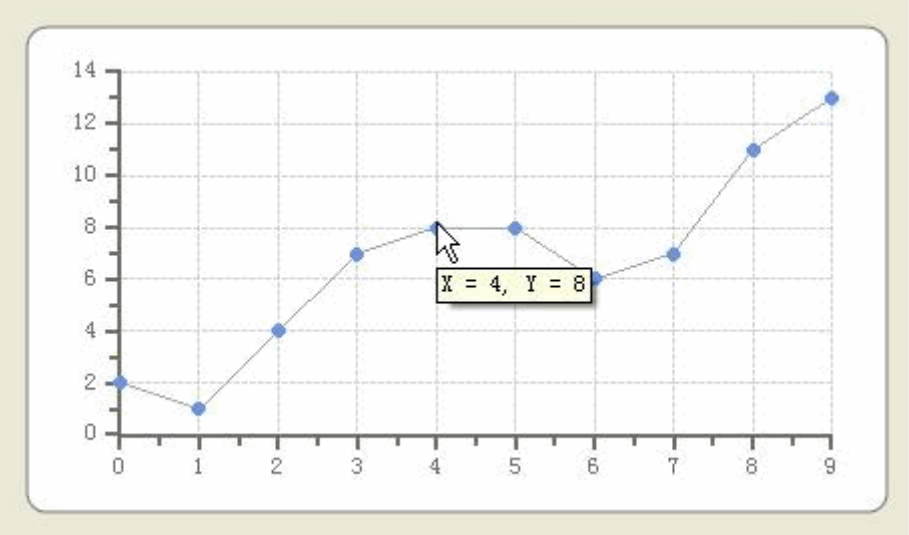
#Y3VAL： Y3 值

#ISERIES： 图序编号

#IPOINT: 数据点编号

代码必须用大括号“{}”括起来。

例如希望动态提示鼠标位置数据点的 X 值和 Y 值，如下图所示：



下面是完整的代码，和原来的代码相比，我们只是增加了加粗的那一行：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart = e.Form.Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '新增图系包括 10 个数据点
Series.TooltipText = "X = {#XVAL}, Y = {#YVAL}"
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
```

二、固定标示

图系还有一个 DataLabelText 属性，和 ToolTipText 一样，DataLabelText 也可以使用各种代码，不同的是，DataLabelText 生成的标示 会成为图表的一部分，始终可见。

还是用下面这个表来演示：

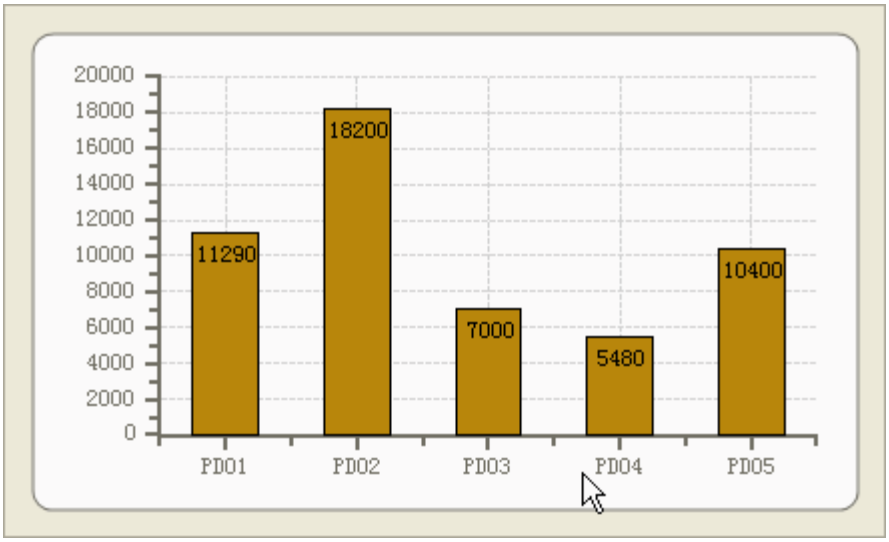
	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

代码为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart = e.Form.Controls("Chart1") ' 引用窗口中的图表
```

```
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Series.DataLabelText = "{#YVAL}"
```

下图是生成的图表，可以看到每一个产品的销量都被标出来了：



出现这种标示，是因为最后一行代码在发生作用：

Series.DataLabelText = "{#YVAL}"

除了 DataLabelText 属性用于设置标示内容外，Series 还提供了一些属性，让您精确控制标示的显示方式：

- **DataLabelVisible**
  逻辑型，返回或者设置标示是否可见

- **DataLabelBackColor**
  返回或者设置标示的背景颜色

- **DataLabelForeColor**
  返回或者设置标示的字体颜色

- **DataLabelBorder**
  逻辑型，返回或者设置是否显示标示的边框

- **DataLabelBorderColor**
  返回或者设置标示边框的颜色

- **DataLabelConnected**
  逻辑型，返回或者设置是否在标示和数据点之间显示一条连线。

- **DataLabelCompass**
  LabelCompassEnum 型枚举，用于设置标示的位置，通常用默认设置就行了。
  LabelCompassEnum 枚举的可选值有：

  Auto
  East
  North
  NorthEast
  NorthWest
  Orthogonal
  Radial
  South
  SouthEast
  SouthWest
  West

4.11.1.8 输出图表

Chart 提供了一些方法，用于输出图表。
请参考：
Image
CopyImage
SaveImage

Print

4.11.2 图表类型

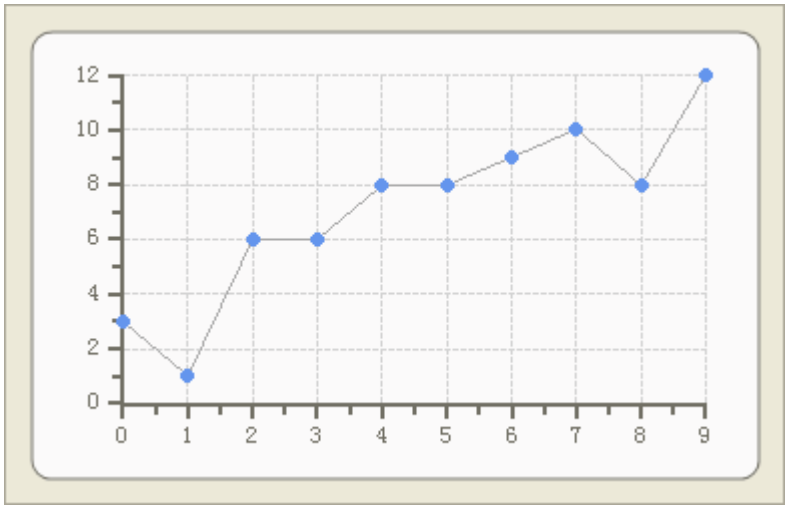
4.11.2.1 XYPlot

XYPlot(线形)图是默认的图表类型，所以无需明确指定。
例如:

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '新增图系包括 10 个数据点
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
```

```
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
```

生成的图表：



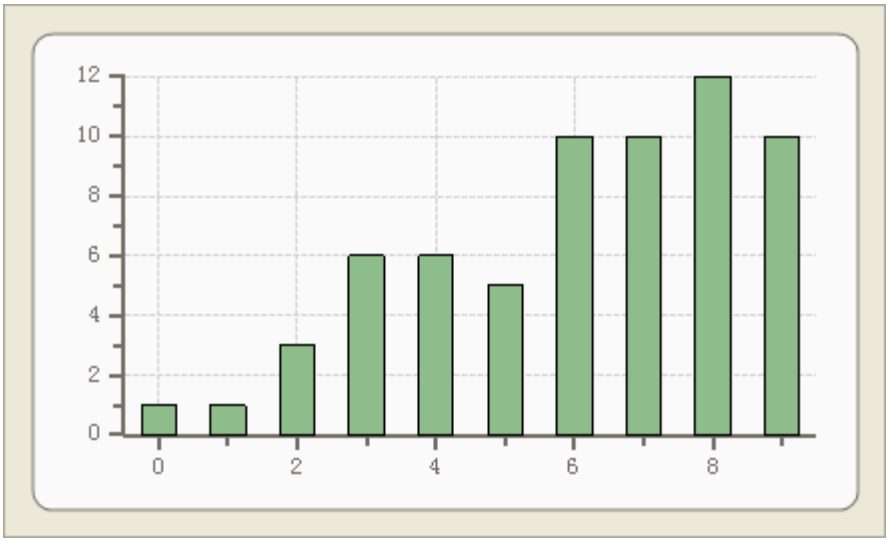
4.11.2.2 Bar

Bar(条形)图和线形图一样常用。

例如：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '设置图表类型
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '新增图系包括 10 个数据点
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
```

生成的图表为：



4.11.2.3 Pie

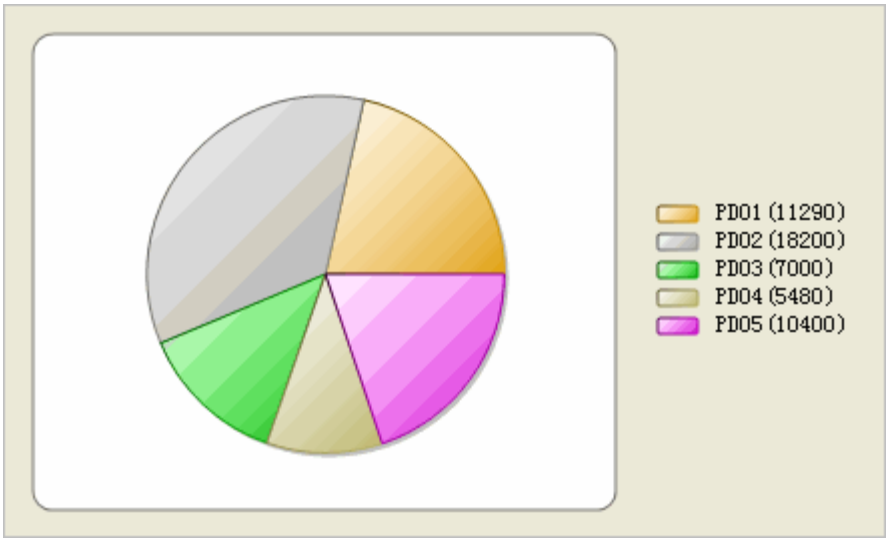
Pie(饼型)也是一种常用类型，Pie 型非常特殊，没有 X 轴，只使用 Y 轴设置数据；一个图系表示一个扇形，而且只能包括一个值，这个值的大小决定扇形的大小。

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

例如对于上面的表，如果需要用 Pie 型来图示，代码会和我们之前的代码有很大的不同：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.ChartType = ChartTypeEnum.Pie '图表 1 类型改为 Bar(条形)
For Each r As Row In t.Rows
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 1 '一个系列只能包括一个值
    Series.Text = r("产品") & "(" & r("数量") & ")" '设置图系的标题
    Series.Y(0) = r("数量") '指定值
Next
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.East '图列显示在东方(右方)
```

这是生成的图表：

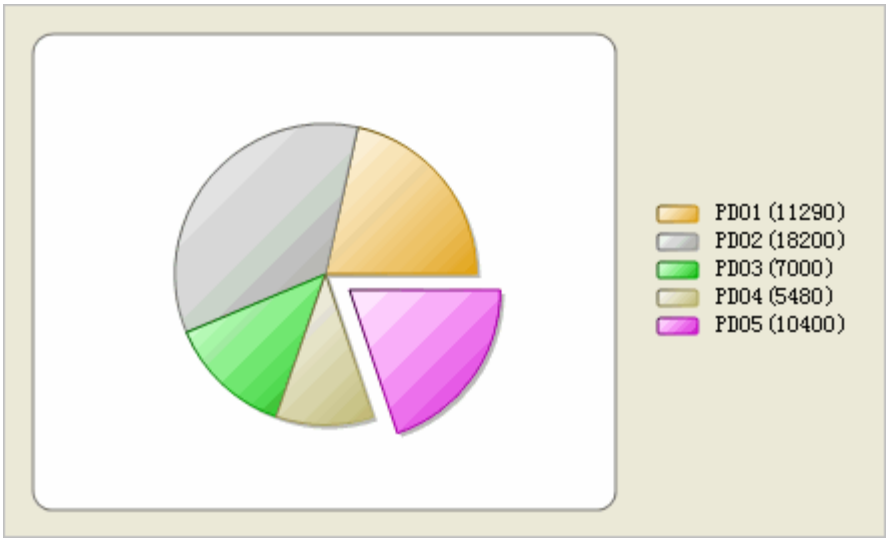


ChartSeries 有一个属性 Offset，它可以让图系所代表的扇形偏离中心点，所偏离的距离就由该属性指定。

例如，我们稍微改一下代码：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.ChartType = ChartTypeEnum.Pie '图表 1 类型改为 Bar(条形)
For Each r As Row In t.Rows
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 1 '一个系列只能包括一个值
    Series.Text = r("产品") & "(" & r("数量") & ")" '设置图系的标题
    Series.Y(0) = r("数量") '指定值
    If r("产品") = "PD05" Then '如果是产品 PD05
       Series.Offset = 20 '那么图系偏离中心点 20 个像素。
    End If
Next
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.East '图列显示在东方(右方)
```

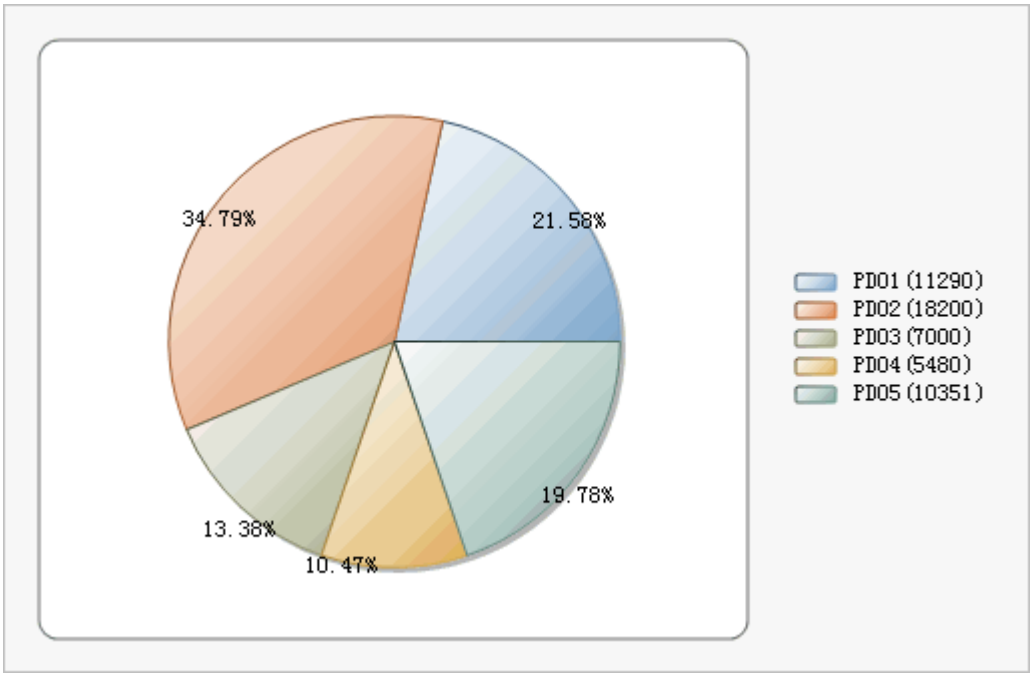
得到的图表为：



有的时候，可能希望能够以百分比的形式标出各数据，可以参考下面的代码：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表1") '定义一个变量t引用数据表
Dim sm As Integer = t.Compute("Sum(数量)") '计算总数量
Chart= Forms("窗口1").Controls("Chart1") ' 引用窗口中的图表
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.ChartType = ChartTypeEnum.Pie '图表1类型改为Bar(条形)
For Each r As Row In t.Rows
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 1 '一个系列只能包括一个值
    Series.Text = r("产品") & "(" & r("数量") & ")" '设置图系的标题
    Series.Y(0) = r("数量") '指定值
    Series.DataLabelText = Math.Round(r("数量")*100/sm,2) & "%" '计算百分比
Next
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.East '图列显示在东方(右方)
```

得到的图表为：



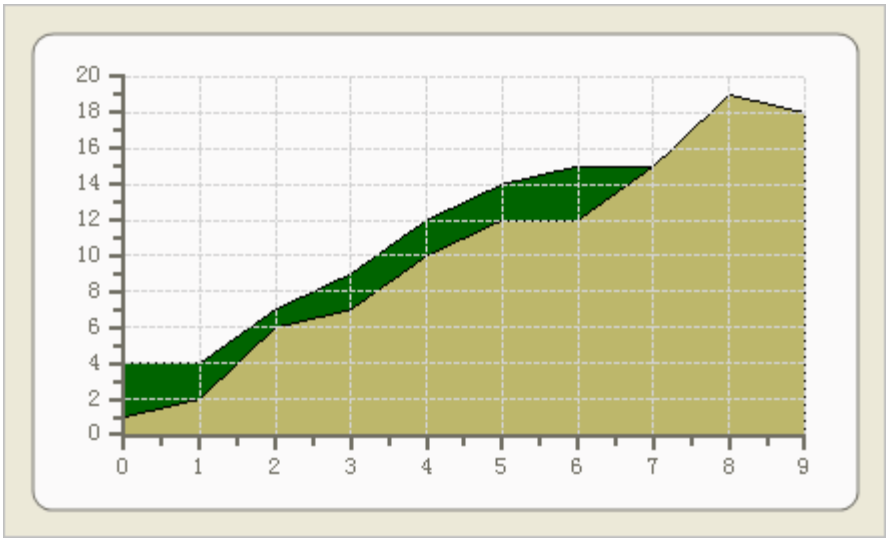
4.11.2.4 Area

Area(面积)型和线形非常相似，差别是线条和 X 轴之间的区域会被填充，而且新的图系绘制在之前图系之上。

例如：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Area '图表类型改为 Area
Chart.SeriesList.Clear() '清除图表原来的图系
For n As Integer = 0 To 1
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 10 '图系包括 10 个数据点
    For i As Integer = 0 to 9 '指定每个数据点的位置
        Series.X(i) = i '指定水平坐标
        Series.Y(i) = i * 2 + Rand.Next(5) '垂直坐标用随机数生成
    Next
Next
```

生成的图表为：



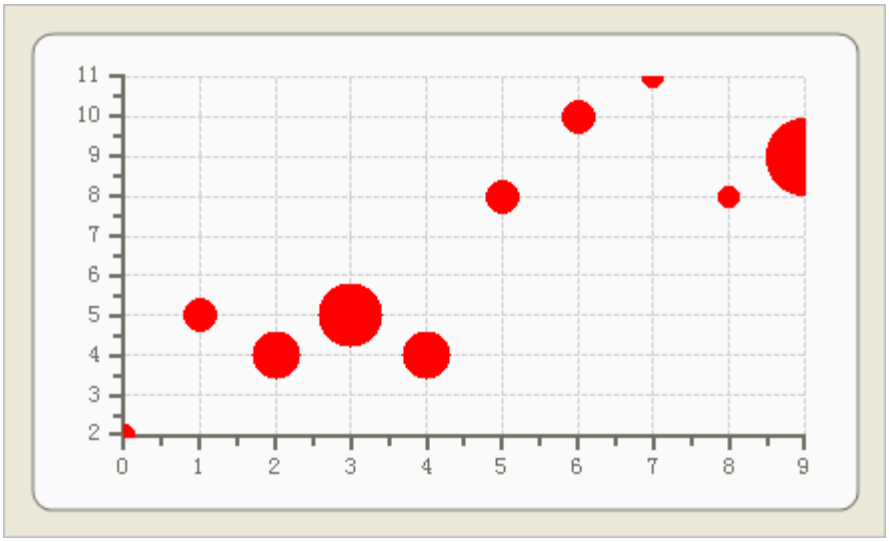
4.11.2.5 Bubble

Bubble(泡)用三个值来表示一个数据点，X 和 Y 确定点的坐标位置，Y1 确定该数据点标记大小，这样可以用平面图形表示三维的数据。

例如：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bubble
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '图系包括 10 个数据点
Series.MarkShape= MarkShapeEnum.Dot '指定数据标记为圆点。
Series.MarkColor = Color.Red '指定数据标记的颜色
Series.LinePattern = LinePatternEnum.None '不显示数据点之间的联线
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
    Series.Y1(i) = Rand.Next(5) '泡的大小用随机数生成
Next
```

生成的图表为：



上面的代码让我们接触到一些新的知识，我们可以设置连线和数据标记的颜色和形状，当然也可以禁止显示数据标记或连线，在后面的编程参考会详细介绍这方面的内容，现在不必深究。

4.11.2.6 Candle

我们需要四个数据才能完整表示一只股票在某天的价格，分别是开盘价、收盘价、盘中最高价、盘中最低价，而 Candle 主要是针对这种需求而设计。

Candle 图用五个数值来表示一个数据点：

X: 水平位置

Y: 最大值

Y1: 最小值

Y2: 开始值

Y3: 结束值

假定有一个表：

	日期	开盘价	最高价	最低价	收盘价	成交额
1	2008-04-01	1801.4	1801.6	1750	1752.6	6901204
2	2008-04-02	1750.8	1762.1	1718.8	1724	6674749
3	2008-04-03	1725.8	1740.9	1715	1720.6	5996402
4	2008-04-04	1722.6	1726.7	1695.5	1704.5	5420442
5	2008-04-05	1706.7	1725.9	1706.7	1724.5	4240450
6	2008-04-06	1727.4	1736.2	1722.7	1725.4	4295444
7	2008-04-07	1725.6	1758.5	1719.8	1758.5	6252500

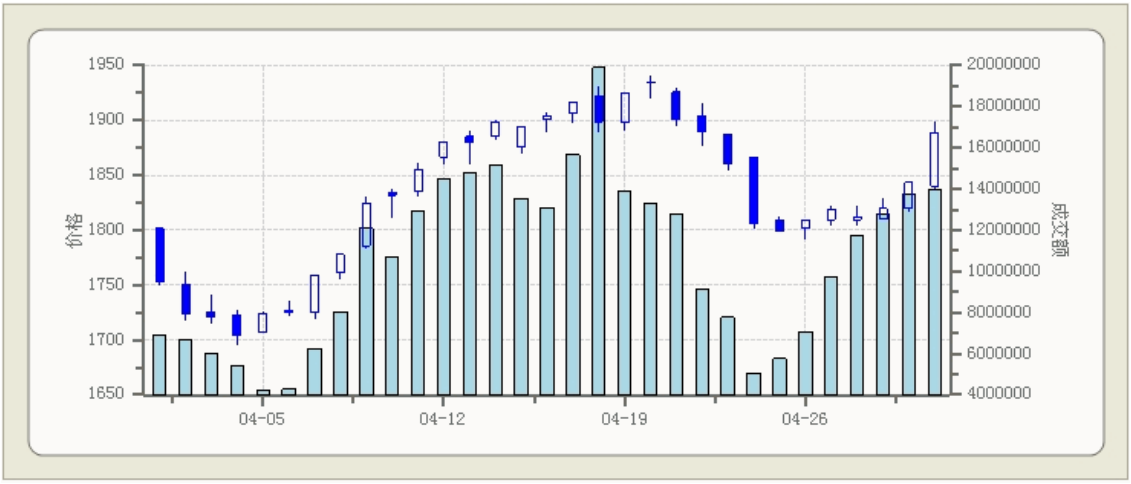
完整地图示该表的代码为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("stock") '定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Candle '设置图表类型
Chart.AxisX.DateType = True '明确指定 X 轴是日期型
Chart.AxisX.AnnoFormatString = "MM-dd" '制定日期表示格式
```

```
Chart.AxisY.Text = "价格"
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.LineColor = Color.Blue '设置该图系的颜色
Chart.DataSource = "Stock" '绑定设置
Series.X.DataField = "日期"
Series.Y.DataField = "最高价"
Series.Y1.DataField = "最低价"
Series.Y2.DataField = "开盘价"
Series.Y3.DataField = "收盘价"

'用第二个图表显示成交额
Chart.ChartType2 = ChartTypeEnum.Bar
Chart.SeriesList2.Clear()
Series = Chart.SeriesList2.Add()
Series.FillColor = Color.LightBlue '设置条形图的填充颜色
Series.X.DataField = "日期"
Series.Y.DataField = "成交额"
Chart.AxisY2.Text = "成交额"
Chart.AxisY2.GridMajorVisible = False '隐藏 Y2 轴的网格线
```

生成的图表：



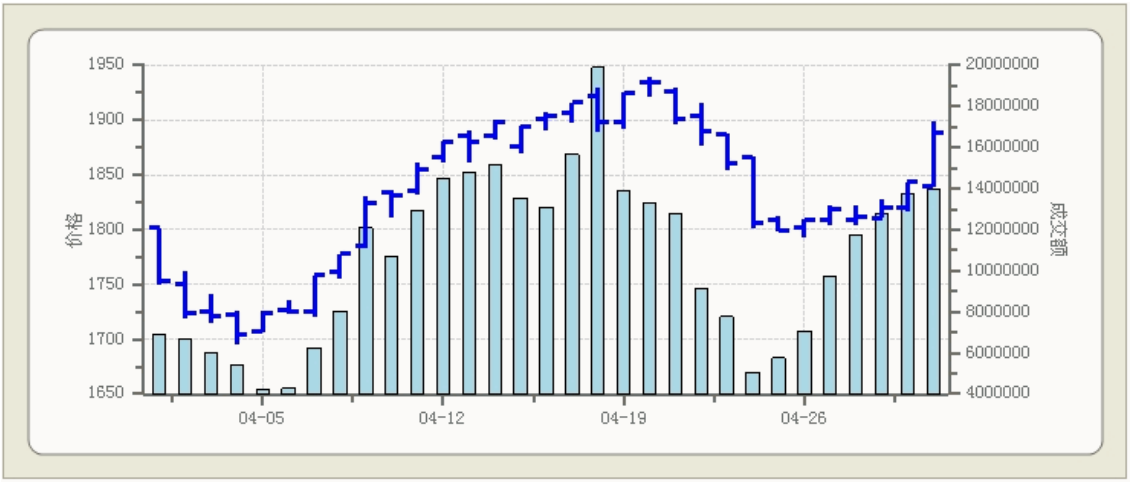
炒股的人应该都能看懂上面的图吧。

上面的代码还接触了几个新的属性，将坐标轴的 DateType 属性设为 True，表示坐标轴的数据是日期型，利用坐标轴的 AnnoFormatString 属性，可以设置日期或数值的刻度标记格式。

图系的 FillColor 属性用于设置条形图的填充颜色，LineColor 属性用于设置线形图的连线颜色，包括 XYPlot 和 Candle 等类型。

4.11.2.7 HiloOpenClose

HiloOpenClose 和 Candle 完全一样，不同的只是图表的形状稍有不同：



通过设置图系的 LineThickNess 属性，可以控制线条的宽度，单位为像素，上图的 LineThickNess 值为 3。

HiloOpenClose 图很少使用，因为 Candle 图表示数据更为明晰。

4.11.2.8 Hilo

Hilo 类似于 Candle，不过更简单，只有最高值和最低值，没有开始值和结束值。

Hilo 图用三个数值来表示一个数据点：

X:　水平位置

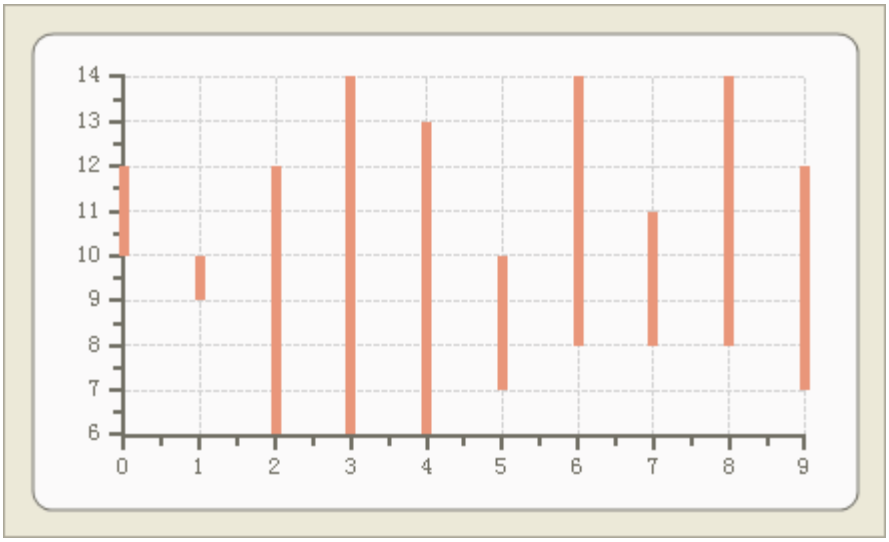
Y:　最大值

Y1:　最小值

示例

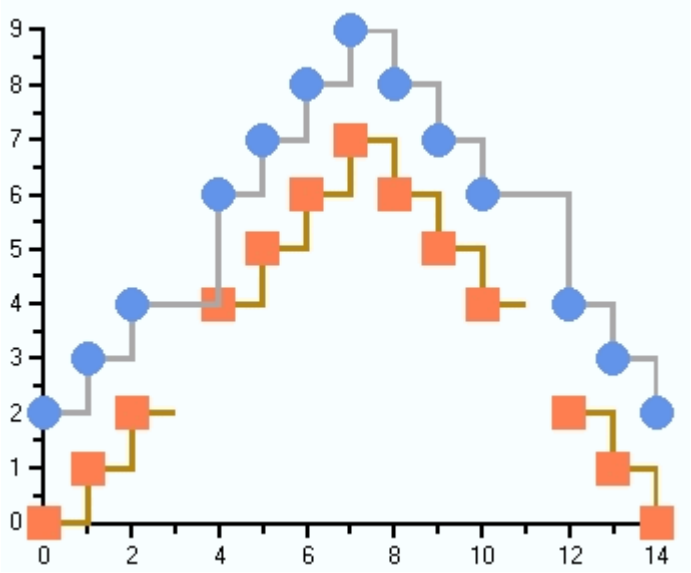
```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Hilo '设置图表类型
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.LineThickNess = 5 '设置线条宽度
Series.Length = 10 '图系包括 10 个数据点
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = 10 + Rand.Next(5) '生成最大值
    Series.Y1(i) = 10 - Rand.Next(5) '生成最小值
Next
```

生成的图表为：



4.11.2.9 Step

Step 和 XYPlot 完全一样，只是数据点不是用直线连接，而是用折线连接：



4.11.2.10 Histogram

Histogram 用于分析数值的分布范围，你可以将一段范围的数据分割成一个一个的区间，系统会自动显示 Y 值处于每一个区间的个数。

所以设计一个 Histogram 图标，需要指定数值范围的起始值，每一个分区的宽度，以及总的分区个数，至于数据点，并不会真正显示在图表上，图表显示的只是数值（Y 值）位于每一个区间的个数。

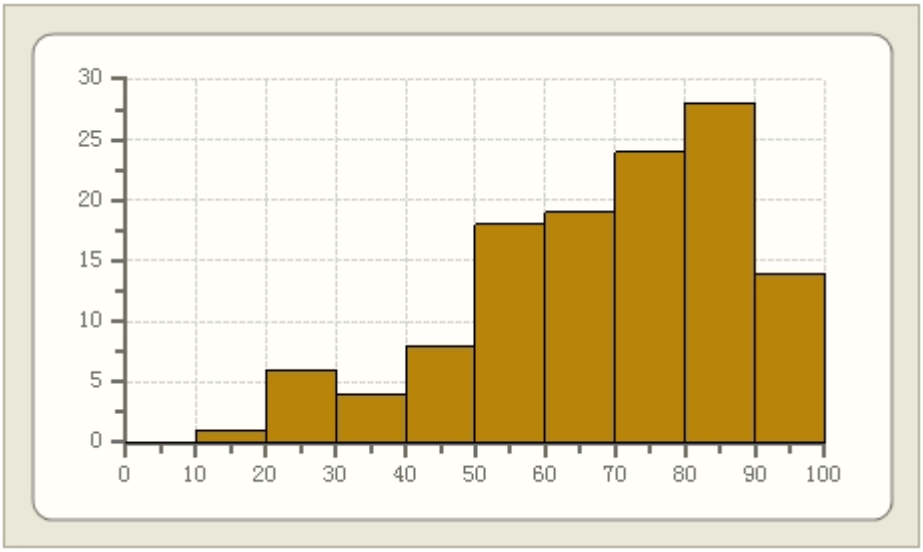
下面是一个学生考试成绩表，我们将 0 到 100 分成 10 个区间，每个区间的宽度为 10，图示每一个区间得个数。

	姓名	成绩
1	陈锦标	97
2	伍景珠	94
3	邓玉婷	93
4	李晓燕	87
5	何列琦	98
6	林璐韵	90
7	冯　勃	94
8	蒋广裕	96
9	黄　图	92
10	叶卜源	82
11	何思敏	80
12	黄秀东	87
13	蒋春泽	98
14	宋妙倩	86
15	徐思远	86

代码为：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Histogram ' 设置图表类型
Chart.DataSource = "成绩表" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Histogram.IntervalStart = 0 '指定起始值
Series.Histogram.IntervalWidth = 10 '指定分区宽度
Series.Histogram.IntervalNumber= 10 '指定分区个数
Series.Y.DataField = "成绩" 'Y 轴绑定到数量列
```

生成的图表：



可以看到，Series 有一个 Histogram 属性，利用该属性来设置分区的起始值、宽度、个数。

Histogram 有三种显示方式，通过图系的 HistogramType 属性来设置，分别是：

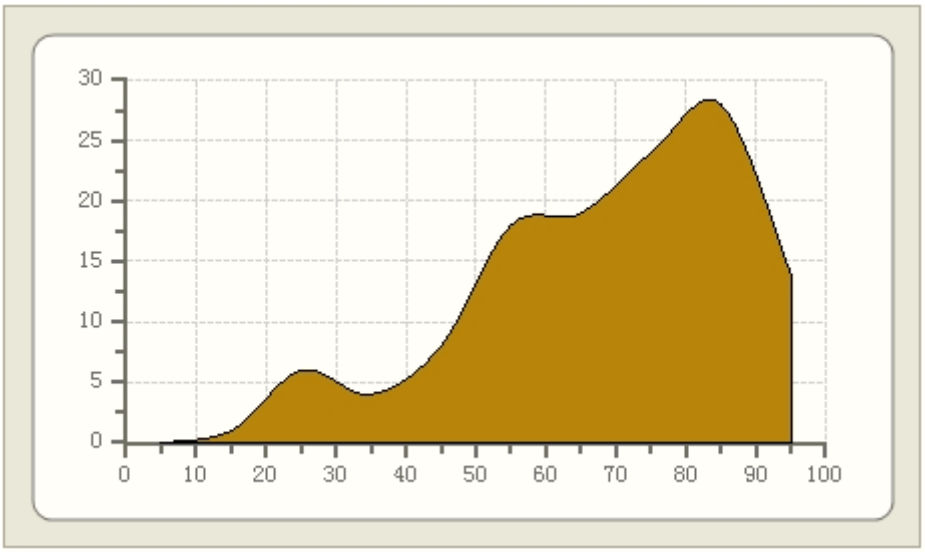
HistogramTypeEnum.Histogram

HistogramTypeEnum.FrequencyGraph

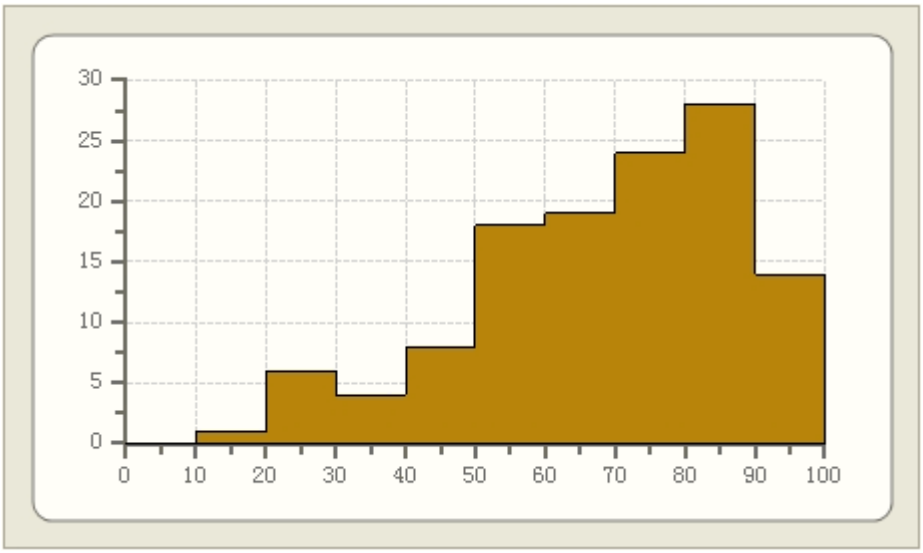
HistogramTypeEnum.SteppedFrequencyGraph
其中第一种是默认方式，就是我们前面看到的图表。
我们看看 FrequencyGraph 是如何显示的，在上面代码的最后加上两行：

```
Series.HistogramType = HistogramTypeEnum.FrequencyGraph '设置显示方式
Series.FitType = FitTypeEnum.Spline '采用平滑线条
```

现在的图表将显示为：



下图为 SteppedFrequencyGraph 方式的效果：



4.11.3 编程参考

4.11.3.1 Chart(图表)

4.11.3.1.1 属性

4.11.3.1.1.1 基本

4.11.3.1.1.1.1 DataSource

字符型，返回或者设置绑定数据表(DataTable)的名称。

4.11.3.1.1.1.2 DataTable

DataTable 类型，返回或者设置绑定的数据表。

4.11.3.1.1.1.3 DataFilter

绑定到数据表时，可用此属性设置条件表达式，只有符合此条件的数据，才会被图示。

4.11.3.1.1.1.4 ChartType

ChartTypeEnum 型枚举，用于设置图表类型。
ChartTypeEnum 常用的值有：
Area
Bar
Bubble
Candle
Hilo
HiloOpenClose
Histogram
Pie
Step
XYPlot
每一个值都代表着一个图表类型，这些图表类型我们都已经在前面的章节中一一介绍过了。

4.11.3.1.1.1.5 ChartType2

等同于 ChartType，只是此属性针对的是第二个图表。

4.11.3.1.1.1.6 SeriesList

返回图系集合，例如：
Chart.SeriesList(1)
表示第二个图系。
该集合包括以下方法：

- **Add**
  增加一个图系，并返回所增加的图系。

- **Clear**
  清除所有图系

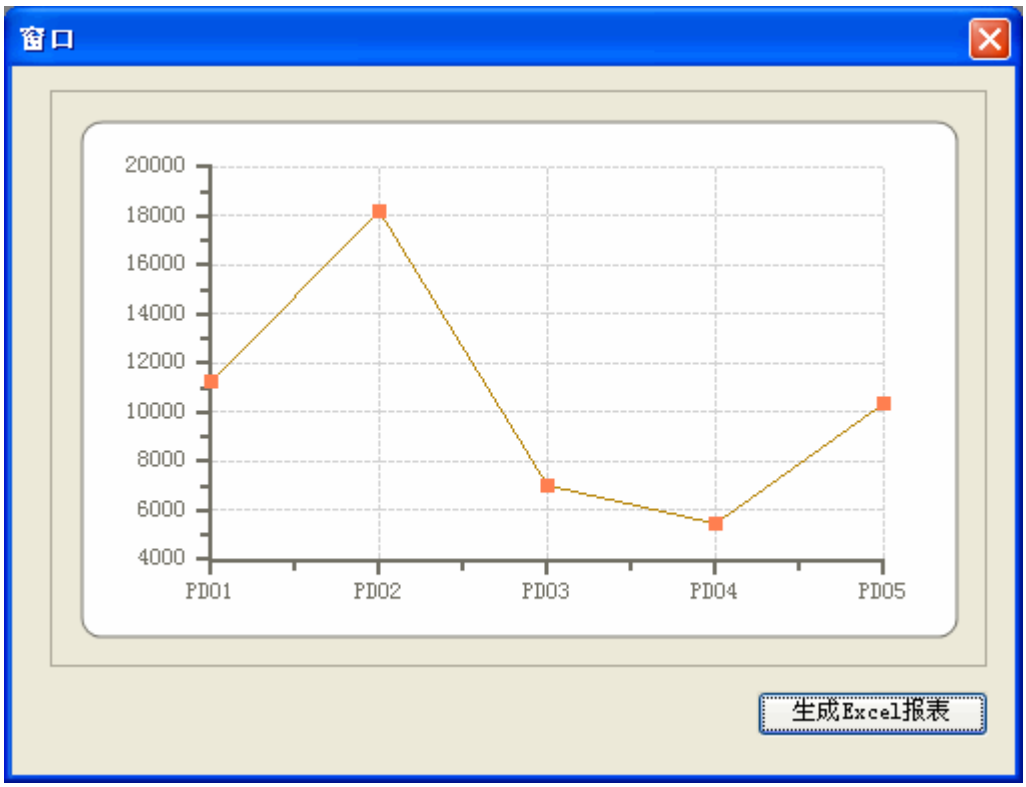
- **RemoveAt**
  删除指定编号的图系。

4.11.3.1.1.1.7 SeriesList2

返回第二个图表的图系。
请参考：SeriesList

4.11.3.1.1.1.8 Image

返回生成的图片，Image 类型。
示例
打开示例文件“Excel 报表.Table”，然后选择“统计表”，可以看到下面的窗口：

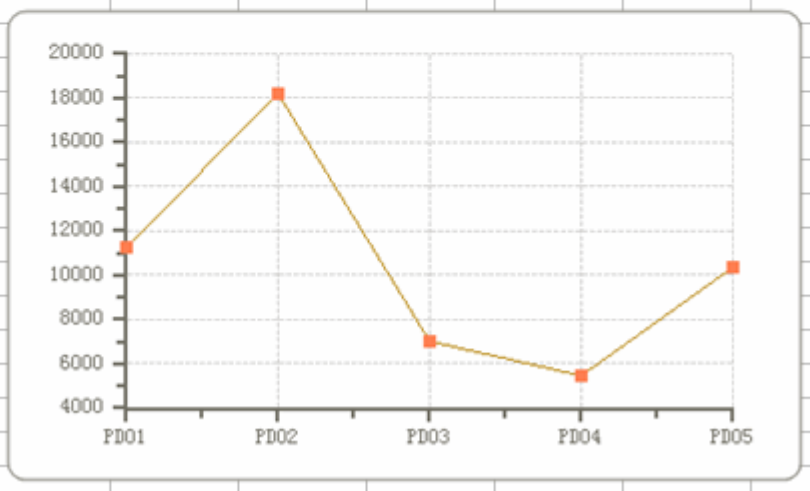


窗口中有一个“生成 Excel 报表”按钮，其 Click 事件代码如下：

```
Dim b As New XLS.Book
Dim t As Table = Tables("统计表")
Dim s As XLS.Sheet = b.Sheets(0)
Dim Chart As WinForm.Chart = Forms("窗口 1").Controls("Chart1")
For c As Integer = 0 To t.Cols.Count -1 '添加列标题
    s(0, c).Value = t.Cols(c).Name
Next
For r As Integer = 0 To t.Rows.Count - 1 '填入数据
    For c As Integer = 0 To t.Cols.Count -1
        s(r +1, c).Value = t.rows(r)(c)
    Next
Next
'插入图表
s(t.rows.Count + 2,1).Value = New XLS.Picture(Chart.Image,0,0,422,258)
b.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

单击该按钮，生成下图所示的 Excel 报表，报表中插入了 Chart 空间生成的图表：

	A	B	C	D	E	F	G	H
1	产品	数量						
2	PD01	11290						
3	PD02	18200						
4	PD03	7000						
5	PD04	5480						
6	PD05	10400						
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								



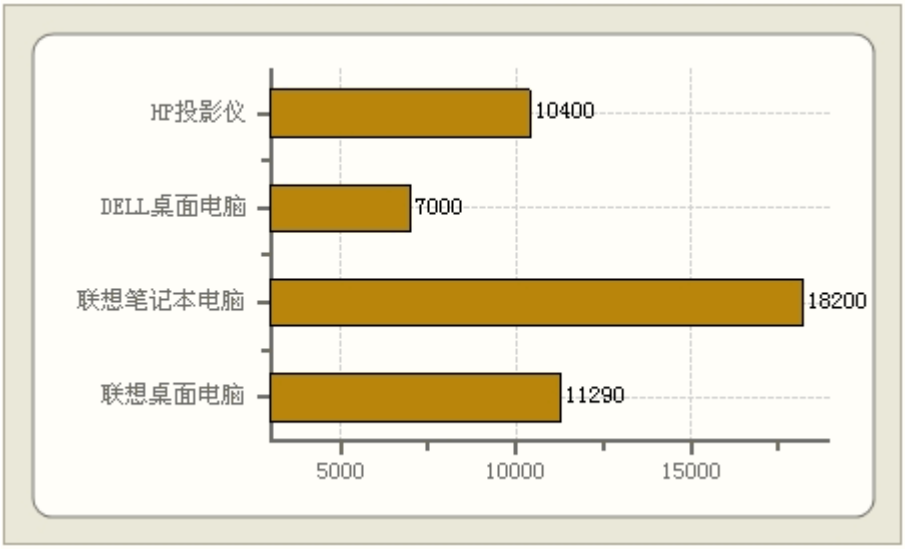
4.11.3.1.1.2 外观

4.11.3.1.1.2.1 VisualEffect

逻辑型，设为 True 将采用 Visual 风格绘制图表。

4.11.3.1.1.2.2 Inverted

逻辑型，对调 X 轴和 Y 轴的位置。
这个属性很有用，因为有的时候 X 轴的标示内容很长，对调一下位置可方便完整地显示标示，例如下面的图表：



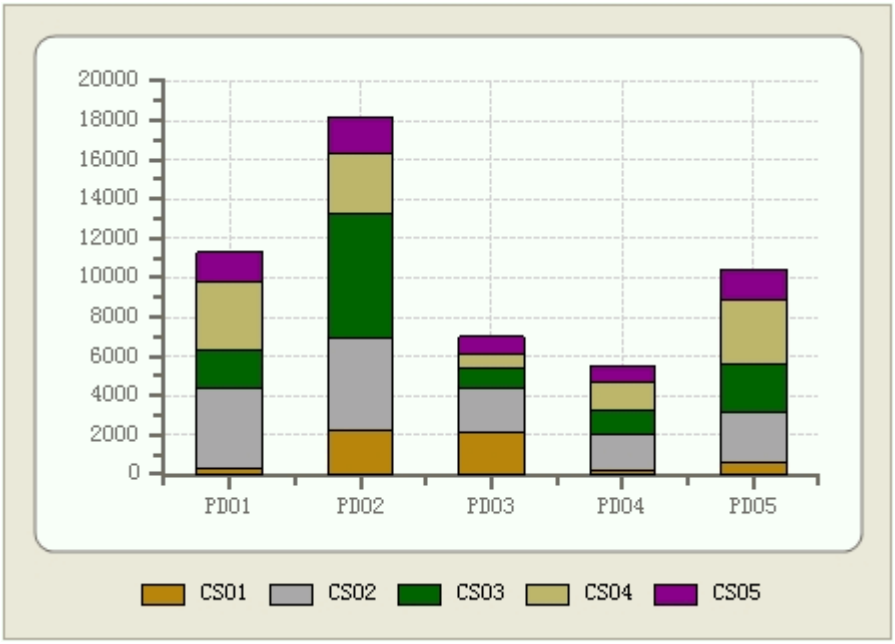
要实现这样的效果，只需加上一行代码即可：

Chart.Inverted = True

4.11.3.1.1.2.3 Stacked

逻辑型，设为 True，所有的图系将重叠在一起，而每一个图系所占的比重由该图系在此数据点的值决定。

下图是一个 Stacked 属性为 True 的图表：

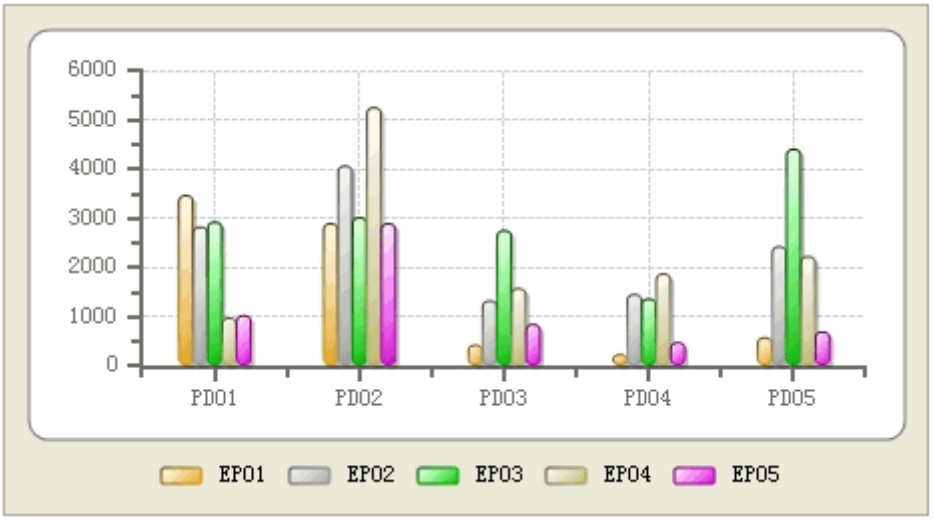


4.11.3.1.1.2.4 Stacked2

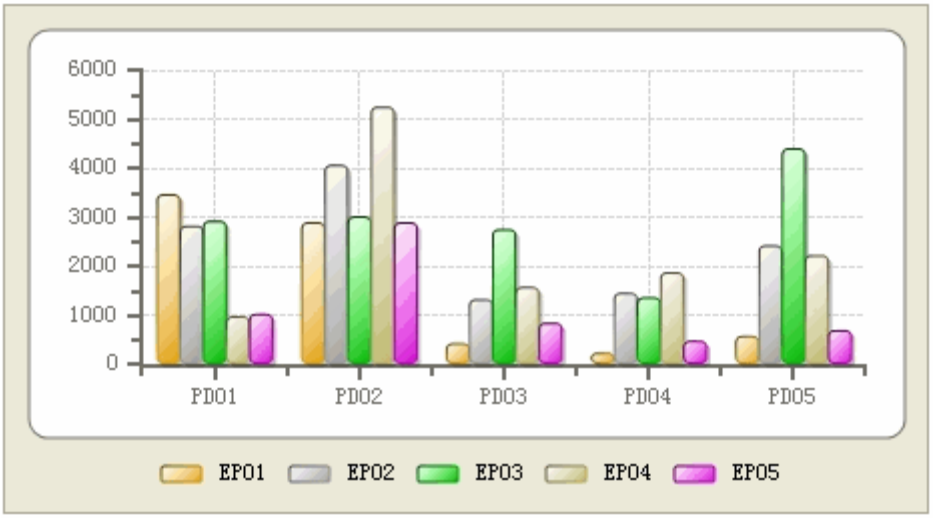
等同于 Stacked，不过针对的是第二个图表。

4.11.3.1.1.2.5 BarClusterWidth

返回或者设置条形图(Bar)所占水平空间的百分比，默认为 50。
这是默认情况(50)的情形：



如果加上一行：
Chart.BarClusterWidth = 80
同样的图表将显示为：



4.11.3.1.1.2.6 BarClusterWidth2

等同于 BarClusterWidth，不过针对的是第二个图表。

4.11.3.1.1.3 图例

4.11.3.1.1.3.1 LegendVisible

逻辑型，是否显示图例。



4.11.3.1.1.3.2 LegendCompass

CompassEnum 型枚举，返回或者图例的位置。
CompassEnum 枚举的可选值有：

West: 左
East: 右
South: 下
North: 上

4.11.3.1.1.3.3 LegendText

字符型，返回或者设置图例的标题。

4.11.3.1.1.3.4 LegendFont

返回或设置图例的字体。
请参考：关于字体

4.11.3.1.1.4 标题

4.11.3.1.1.4.1 HeaderText

字符型，返回或者设置图表的上标题。

4.11.3.1.1.4.2 HeaderFont

返回或者设置图表的上标题字体
请参考：关于字体

4.11.3.1.1.4.3 FooterText

字符型，返回或者设置图表的下标题。

4.11.3.1.1.4.4 FooterFont

返回或者设置图表的下标题字体。
请参考：关于字体

4.11.3.1.1.5 坐标轴

4.11.3.1.1.5.1 AxisX

返回 X 轴。

4.11.3.1.1.5.2 AxisY

返回 Y 轴

4.11.3.1.1.5.3 AxisY2

返回 Y2 轴，也就是第二个图表的 Y 轴。

4.11.3.1.2 方法

4.11.3.1.2.1 Print

打印图表
语法：
Print(ScaleToPage, ShowPrintDialog)

ScaleToPage:　逻辑型，设为 True，将自动调整图表大小至纸张大小。
ShowPrintDialog: 逻辑型，设为 True 显示打印对话框
例如：

```
Dim Chart As WinForm.Chart
Chart= Forms("窗口 1").Controls("Chart1")
Chart.Print(True, True)
```

4.11.3.1.2.2 PageSetup

设置页面
示例：

```
Forms("窗口 1").Controls("Chart1").PageSetup()
```

4.11.3.1.2.3 CopyImage

将图表复制到剪贴板。
例如：

```
Dim Chart As WinForm.Chart
Chart= Forms("窗口 1").Controls("Chart1")
Chart.CopyImage()
```

4.11.3.1.2.4 SaveImage

将图表保存为文件。
语法：

```
SaveImage(FileName)
```

FileName: 目标文件名。
系统会根据目标文件的扩展名选择合适的格式保存文件，例如文件名是“c:\data\001.bmp”，系统会以 BMP 格式保存文件，如果文件名是"c:\data\001.jpg"，则以 jpg 格式保存文件。
推荐采用 WMF 格式，这样缩放图片的时候，不至于变形。
例如：

```
Dim Chart As WinForm.Chart
Chart= Forms("窗口 1").Controls("Chart1")
Chart.SaveImage("C:\data\001.bmp")
Chart.SaveImage("c:\data\002.wmf")
```

4.11.3.1.2.5 CoordToDataIndex

获得指定坐标位置的图系编号和数据点编号。
坐标位置以像素为单位。
语法

```
CoordToDataIndex(CoordX, CoordY, SeriesIndex, PointIndex)
```

参数说明
CoodX: 水平坐标
CoodY: 垂直坐标
SeriesIndex：用于存储图系编号的变量，应该预先定义好
PointIndex: 用于存储数据点编号的变量，应该预先定义好。
如果指定坐标位置存在有效的数据点，那么 CoordToDataIndex 返回 True，否则返回 False
例如希望单击某个数据点，能够显示该数据点的位置和值，只需将 Chart 的 MouseUp 事件代码设置为：

```
Dim s As Integer '定义存储图系编号的变量
```

```
Dim p As Integer '定义存储数据点编号的变量
Dim Chart As WinForm.Chart = Forms("窗口 1").Controls("Chart1") '获得 Chart 对象的引用
If Chart.CoordToDataIndex(e.x, e.y ,s ,p) Then '如果鼠标位置存在有效的图序和数据点
    Dim Series As WinForm.ChartSeries = Chart.SeriesList(s) '获得鼠标位置的图序
    Dim msg as string
    msg = "单击的是第" & (s+1) & "个图系的第" & (p+1) & "个数据点，该点的值为 "
    msg = msg & "(X = " & Series.X(p) & ", Y = " & Series.Y(p) & ")"
    Messagebox.Show(msg)
End If
```

为什么不用 Click 事件而用 Mouseup 事件？ 很简单，MouseUp 的 e 参数包含鼠标位置信息，而 Click 事件没有。

4.11.3.1.2.6 CoordToDataIndex2

等同于 CoordToDataIndex，但针对的是第二个图表。

4.11.3.2 ChartAxis(坐标轴)

4.11.3.2.1 关于 ChartAxis

ChartAxis 表示坐标轴，图表(Chart)有三个坐标轴，分别是 X、Y、Y2。
通过 Chart 对应的属性，可以获得这三个坐标轴的引用。
示例

```
Dim Chart As WinForm.Chart = Forms("窗口 1").Controls("Chart1")
Dim Ax As WinForm.ChartAxis = Chart.AxisX
Dim Ay As WinForm.ChartAxis = Chart.AxisY
Dim Ay2 As WinForm.ChartAxis = Chart.AxisY2
```

4.11.3.2.2 属性

4.11.3.2.2.1 Min

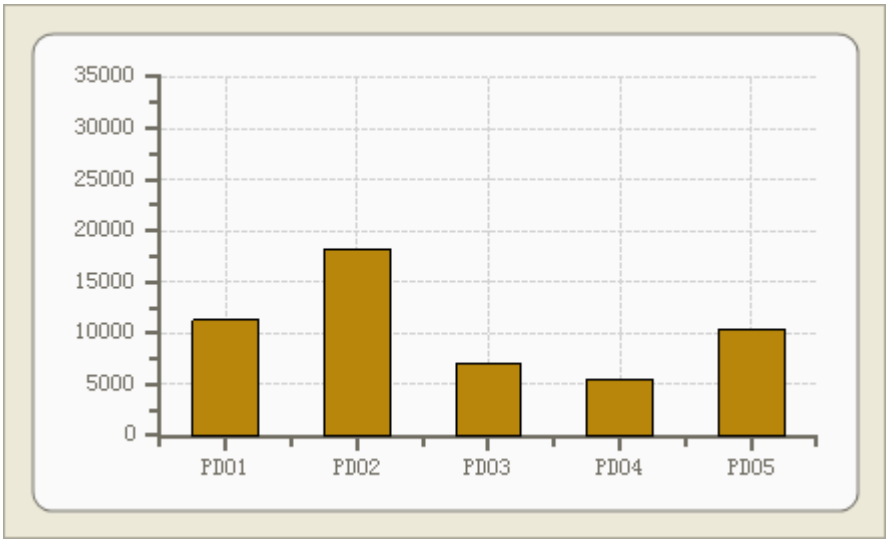
返回或指定坐标轴的最小值，此外还有一个 Max 属性，用于指定坐标轴的最大值。
在默认情况下，坐标轴的最大值和最小值都是程序根据图示数据的范围自动确定的，无需直接指定。
如果自动生成的最小值、最大值不符合要求，可以利用这两个属性指定坐标轴的数据范围。
示例
还是以下面的表格为例:

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

图示代码：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1")'定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1")' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource ="统计表 1" '设置绑定表
Chart.AxisY.Min = 0 '指定 Y 轴的最小值
Chart.AxisY.Max =35000 '指定 Y 轴的最大值
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField ="产品" 'X 轴绑定到产品列
Series.Y.DataField ="数量" 'Y 轴绑定到数量列
```

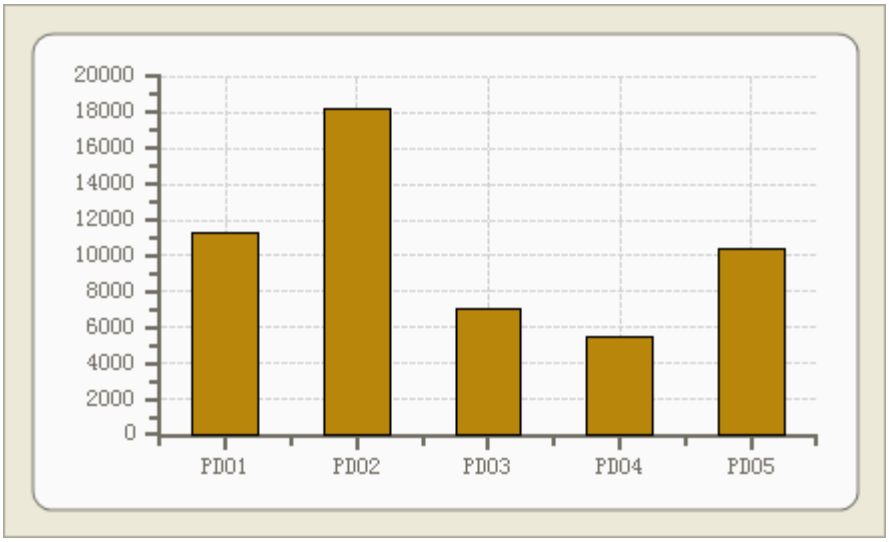
下面是生成的图表，可以看到 Y 轴的范围在 0 到 35000 之间：



如果我们删除下面两行：

```
Chart.AxisY.Min = 0 '指定 Y 轴的最小值
Chart.AxisY.Max =35000'指定 Y 轴的最大值
```

程序将自动决定最佳的 Y 轴范围：



4.11.3.2.2.2 Max

返回或指定坐标轴的最大值。
请参考: Min

4.11.3.2.2.3 Major

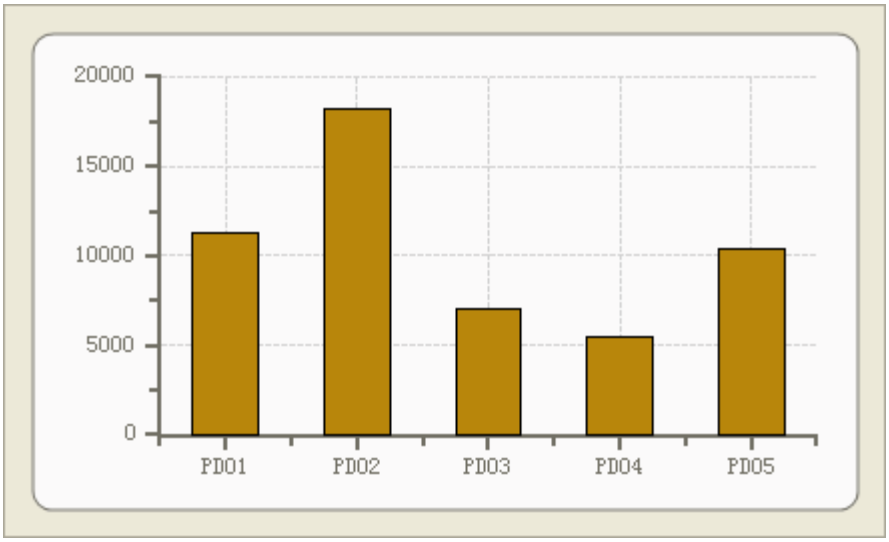
返回或设置主刻度的间隔值。
对应的还有一个 Minor 属性，用于设置副刻度的间隔值。
以下面的表格为例:

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

图示代码：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1")'定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1")'  引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource ="统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Chart.AxisY.Major = 5000 '主刻度间隔值为 5000
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField ="产品" 'X 轴绑定到产品列
Series.Y.DataField ="数量" 'Y 轴绑定到数量列
```

生成的图表为：



4.11.3.2.2.4 Minor

返回或设置副刻度的间隔值。

4.11.3.2.2.5 MajorTick

逻辑型，是否在坐标轴轴上显示主刻度。

4.11.3.2.2.6 MinorTick

逻辑型，是否在坐标轴轴上显示副刻度。

4.11.3.2.2.7 GridMajorVisible

逻辑型，是否显示主刻度的网格线。

4.11.3.2.2.8 GridMinorVisible

逻辑型，是否显示副刻度的网格线。

4.11.3.2.2.9 Compass

返回或者坐标轴的位置，CompassEnum 型枚举，可选值有：
West：左
East： 右
South：下
North：上

请参考：Reversed

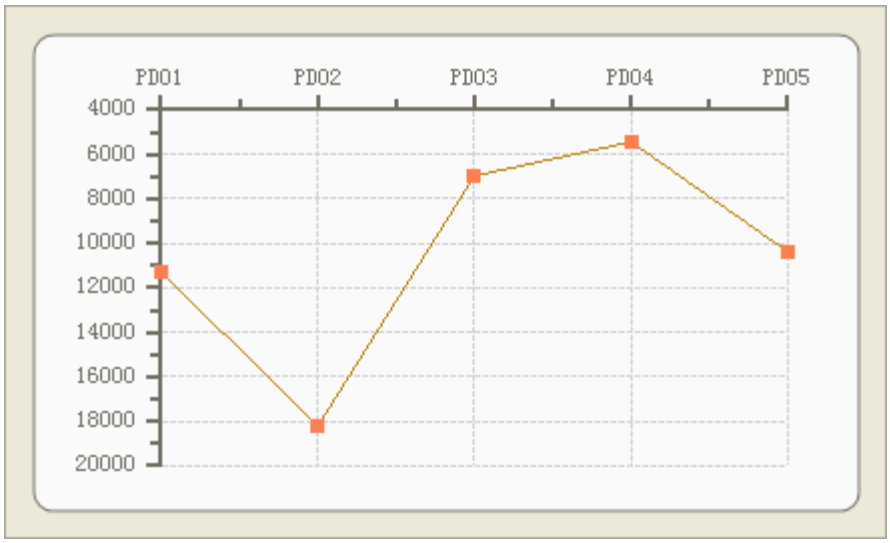
4.11.3.2.2.10 Reversed

逻辑型，是否翻转坐标轴。

示例：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1")'定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1")'  引用窗口中的图表
Chart.DataSource ="统计表 1" '设置绑定表
Chart.AxisY.Reversed = True '翻转 Y 轴
Chart.AxisX.Compass= CompassEnum.North 'X 轴位置设为北方(上方)
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField ="产品" 'X 轴绑定到产品列
Series.Y.DataField ="数量" 'Y 轴绑定到数量列
```

即可将坐标原点移到左上角：



4.11.3.2.2.11 Text

字符型，返回或者设置坐标轴的标题。

4.11.3.2.2.12 Font

返回或设置坐标轴的字体。

请参考：关于字体

4.11.3.2.2.13 AnnoWithLabels

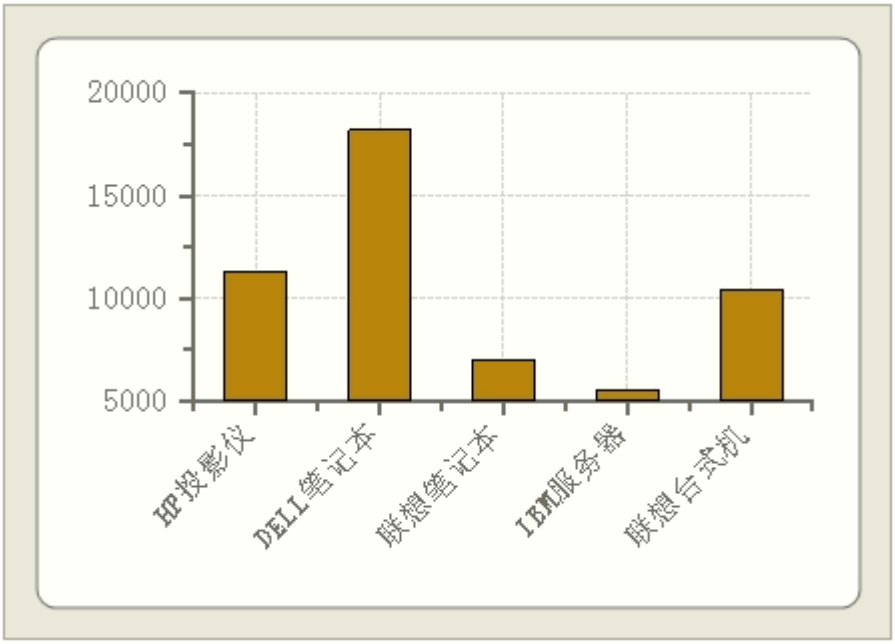
逻辑型，如果设为 True，将用 SetValueLabel 方法设置的字符来标记刻度。

4.11.3.2.2.14 AnnoRotation

数值型，设置坐标轴标示的旋转角度。
示例

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource = "统计表 1" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Chart.AxisX.AnnoRotation = - 45 'X 轴标示逆时针旋转 45 度
```

生成的图表：



4.11.3.2.2.15 AnnoFormatString

返回或者设置坐标轴的标示格式符号，类似于 Format 函数。
在之前的课程我们已经接触过：

```
Chart.AxisX.DateType = True '明确指定 X 轴是日期型
Chart.AxisX.AnnoFormatString = "MM-dd" '指定日期标示格式
```

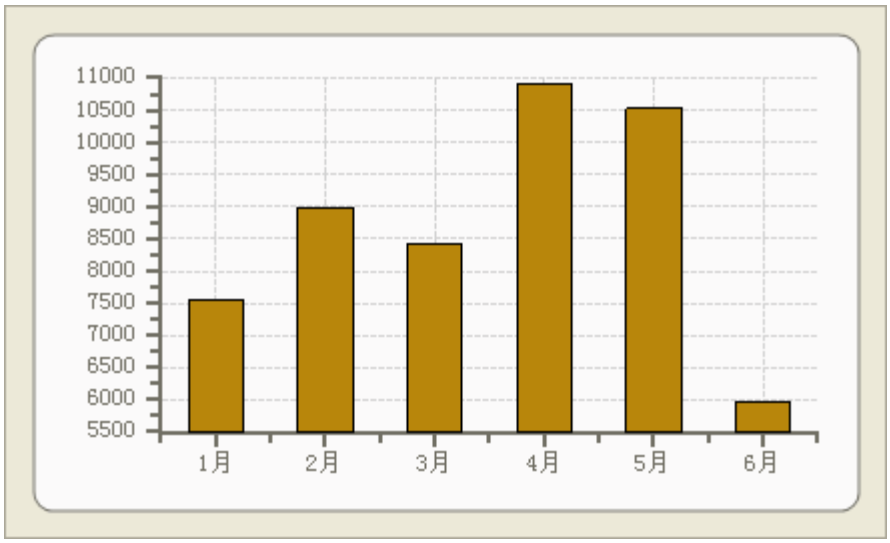
现在再来一个例子，例如对于下面的表格：

	月份	数量
1	1	7560
2	2	8990
3	3	8420
4	4	10910
5	5	10530
6	6	5960

为了让 X 轴的标示更友好一点，可以在图示代码中适用 AnnoFormatString，注意下面代码中加粗的一行：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource = "统计表 1" '设置绑定表
Chart.AxisX.AnnoFormatString = "0 月"
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "月份" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
```

这是生成的图表：



4.11.3.2.2.16 DateType

逻辑型，如果坐标轴的数据是日期型，请将此属性设为 True。

示例

假定有日期和数量两列，图表的 X 轴绑定到日期列，Y 轴绑定到数量列，希望 X 轴不显示年月，只显示日：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
```

```
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.DataSource = "表 A" '设置绑定表
Chart.AxisX.DateType = True 'X 轴是日期型
Chart.AxisX.AnnoFormatString = "dd" '只显示日
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "日期" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
```

4.11.3.2.3 方法

4.11.3.2.3.1 SetValueLabel

为给定的数值设置字符表示。

语法：

SetValueLabel(Value, Label)

Value: 数值

Label: 字符表示

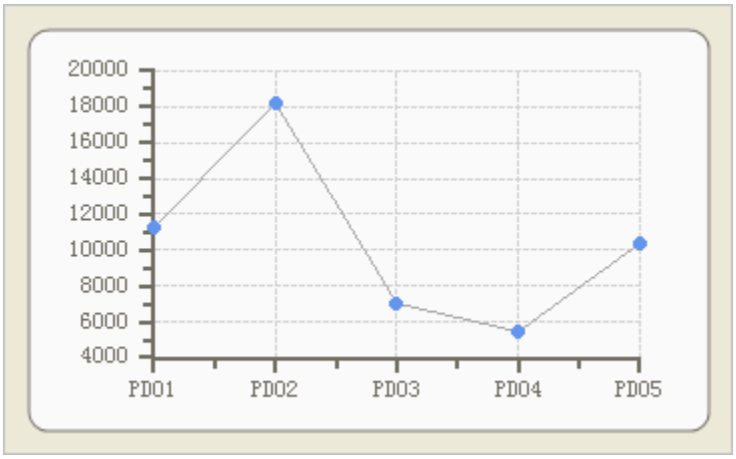
例如对于下面的表格：

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

上表中产品列是字符型，并不是数值型，如果手工编写图示代码，就必须使用 SetValueLabel 函数：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table = Tables("统计表 1") '定义一个变量 t 引用数据表
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = t.Rows.Count '设置图系的长度
For i As integer = 0 to t.Rows.Count - 1 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = t.Rows(i)("数量") '指定垂直坐标
    Chart.AxisX.SetValueLabel(i, t.Rows(i)("产品")) '指定字符表示
Next
Chart.AxisX.AnnoWithLabels = True '启用字符标示
```

生成的图表：



4.11.3.2.3.2 ClearValueLabel

清除所有通过 SetValueLabel 设置的数值字符标示。

4.11.3.3 ChartSeries(图系)

4.11.3.3.1 关于 ChartSeries

ChartSeries 表示图系，每一个 Chart 都有两个图系集合：分别是 SeriesList 和 SeriesList2，后者针对的是第二个图表，极少使用。

4.11.3.3.2 属性

4.11.3.3.2.1 基本

4.11.3.3.2.1.1 Length

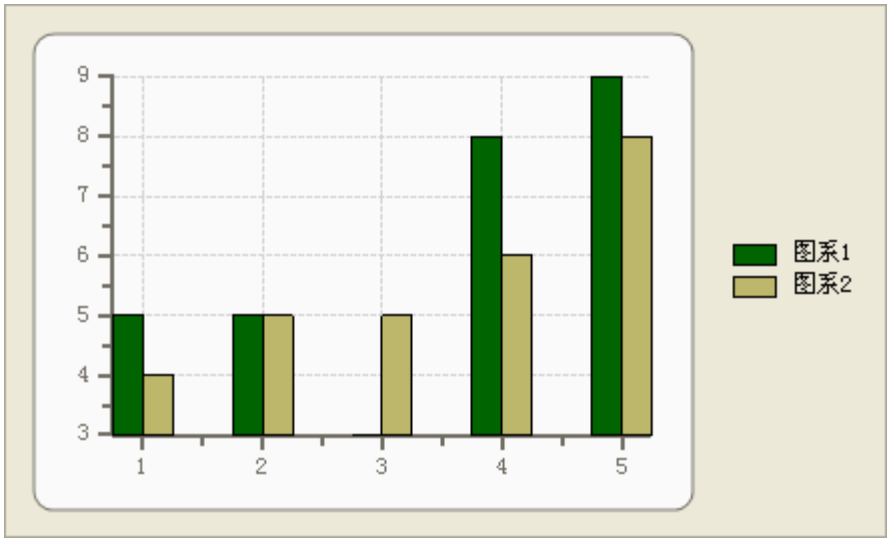
```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '新增图系包括 10 个数据点
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
```

4.11.3.3.2.1.2 Text

返回或者设置图系的标题，这个标题将显示在图例中。
示例

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.ChartType = ChartTypeEnum.Bar '图表类型该为 Bar(条形)
Chart.LegendVisible = True '显示图例
Chart.SeriesList.Clear() '清除图表原来的图系
For n As integer = 0 To 1
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Text = "图系" & (n + 1)
    Series.Length = 10 '图系包括 10 个数据点
    For i As integer = 1 to 5 '指定每个数据点的位置
        Series.X(i) = i '指定水平坐标
        Series.Y(i) = i + n * 2 + Rand.Next(5) '垂直坐标用随机数生成
    Next
Next
```

生成的图表：



4.11.3.3.2.1.3 Offset

图表类型为饼(Pie)形图的时候，可用此属性设置扇形偏离中心点的位置。
请参考：Pie

4.11.3.3.2.1.4 LegendEntry

逻辑型，是否在图例中包括此图系。
此属性极少使用。

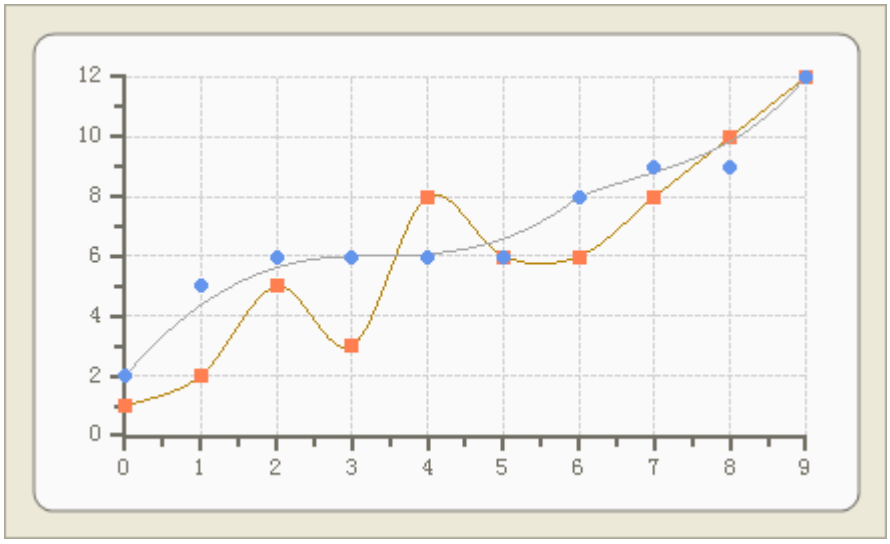
4.11.3.3.2.2 外观

4.11.3.3.2.2.1 FitType

FitTypeEnum 型枚举，用于设置数据点之间连线的类型，包括三个值：
Line： 直线，这是默认值
Spline： 平滑曲线
Beziers：贝塞尔曲线
示例

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
For n As integer = 0 To 1
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 10 '图系包括 10 个数据点
    For i As integer = 0 to 9 '指定每个数据点的位置
        Series.X(i) = i '指定水平坐标
        Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
    Next
Next
Chart.SeriesList(0).FitType = FitTypeEnum.Spline
Chart.SeriesList(1).FitType = FitTypeEnum.Beziers
```

生成的图表：



4.11.3.3.2.2.2 LineColor

返回或设置连线的颜色。
如果没有指定，连线的颜色将是随机的，这也是为什么测试示例代码的时候，每次单击按钮，生成的图形颜色都不一样的原因。

示例

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
For n As integer = 0 To 1
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Length = 10 '图系包括 10 个数据点
    For i As integer = 0 to 9 '指定每个数据点的位置
        Series.X(i) = i '指定水平坐标
        Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
    Next
Next
Chart.SeriesList(0).LineColor= Color.Red
Chart.SeriesList(1).LineColor = Color.Green
```

4.11.3.3.2.2.3 LineThickNess

返回或者设置连线的宽度，默认为 1，单位为像素。

4.11.3.3.2.2.4 LinePattern

返回或设置连线的类型，LinePatternEnum 型枚举，有以下可选值:

Dash： 虚线

DashDot: 虚线+点

DashDotDot：虚线+点+点

Dot： 点线

Solid： 实线

None： 不显示连线

4.11.3.3.2.2.5 FillColor

返回或设置填充颜色，该颜色用于绘制条形(Bar)或者饼形图(Pie)。

4.11.3.3.2.2.6 MarkColor

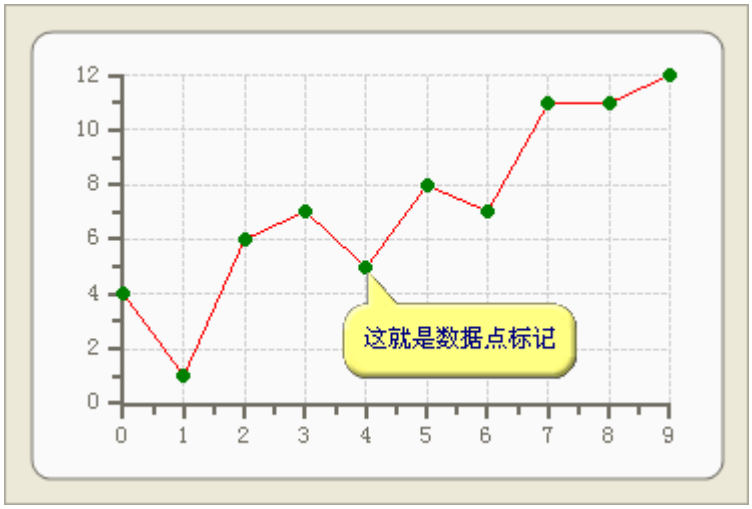
返回或设置数据点标记的颜色。

示例

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '图系包括 10 个数据点
```

```
For i As integer = 0 to 9 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
Series.LineColor = Color.Red
Series.MarkShape = MarkShapeEnum.Dot
Series.MarkColor = Color.Green
```

这是生成的图标，现在你反复单击按钮，生成的图表始终是相同的：



4.11.3.3.2.2.7 MarkShape

返回或设置数据点标记的形状，MarkShapeEnum 型枚举，可选值有：

Box：方形

Circle：圆型

Cross：十字形

DiagCross：斜十字

Diamond：菱形

Dot：原点型

HorizontalLine：水平线

InvertedTri：倒三角

None：不显示数据点标记

OpenDiamond：空心菱形

OpenInvertedTri：空心倒三角

OpenTri：空心三角形

Square：方框

Star：星型

Tri：三角形

VerticalLine：垂直线

4.11.3.3.2.2.8 MarkSize

返回或设置数据点标记的大小，单位为像素，默认是 5。

4.11.3.3.2.2.9 MarkBorderColor

返回或设置数据点标记的边框颜色。

4.11.3.3.2.2.10 MarkBorderWidth

返回或设置数据点标记的边框宽度，单位为像素，默认为 1。

4.11.3.3.2.3 标示

4.11.3.3.2.3.1 TooltipText

返回或设置数据标示字符。
ToolTipText 可以使用一些代码来表示数据，可使用的代码如下：
#XVAL： X 值
#YVAL： Y 值
#Y1VAL： Y1 值
#Y2VAL： Y2 值
#Y3VAL： Y3 值
#ISERIES： 图序编号
#IPOINT: 数据点编号
代码必须用大括号“{}”括起来。
请参考：数据标示

4.11.3.3.2.3.2 DataLabelText

返回或设置数据标示字符，和 ToolTipText 一样，可以使用各种代码。
请参考：数据标示

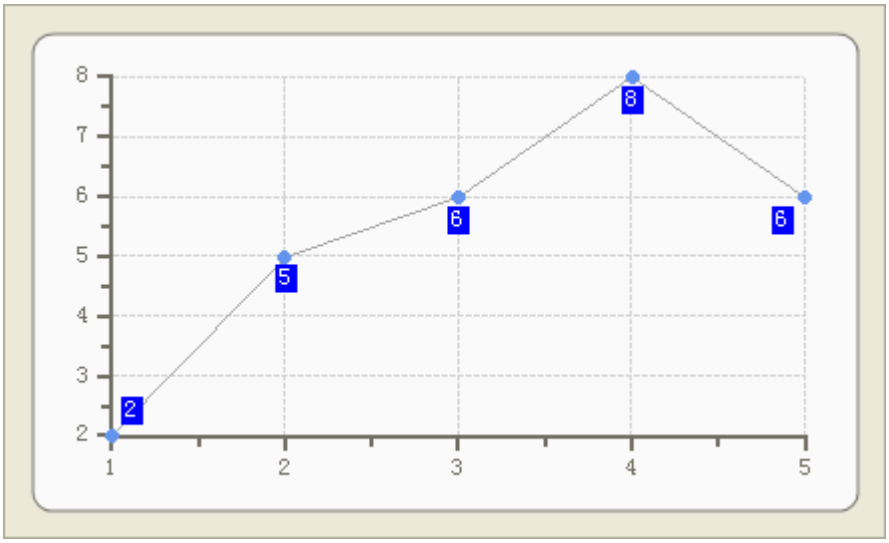
4.11.3.3.2.3.3 DataLabelBackColor

返回或设置标示的背景颜色。
示例
下面的代码，将数据标示设为蓝底白字：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= Forms("窗口 1").Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Length = 10 '新增图系包括 10 个数据点
For i As integer = 1 to 5 '指定每个数据点的位置
    Series.X(i) = i '指定水平坐标
    Series.Y(i) = i + Rand.Next(5) '垂直坐标用随机数生成
Next
```

```
Series.DataLabelText = "{#YVAL}"
Series.DataLabelBackColor = Color.Blue '蓝底
Series.DataLabelForeColor = Color.White '白字
```

生成的图表：



4.11.3.3.2.3.4 DataLabelForeColor

返回或设置标示的字体颜色。

4.11.3.3.2.3.5 DataLabelBorder

逻辑型，是否显示标示的边框。

4.11.3.3.2.3.6 DataLabelBorderColor

返回或设置标示边框的颜色。

4.11.3.3.2.3.7 DataLabelConnected

逻辑型，是否在标示和数据点之间显示一条连线。

4.11.3.3.2.3.8 DataLabelCompass

返回或设置标示的位置，LabelCompassEnum 型枚举，可选的值有：

Auto
East
North
NorthEast
NorthWest
Orthogonal

Radial
South
SouthEast
SouthWest
West
通常用默认的设置即可。

4.11.3.3.2.3.9 DataLabelVisible

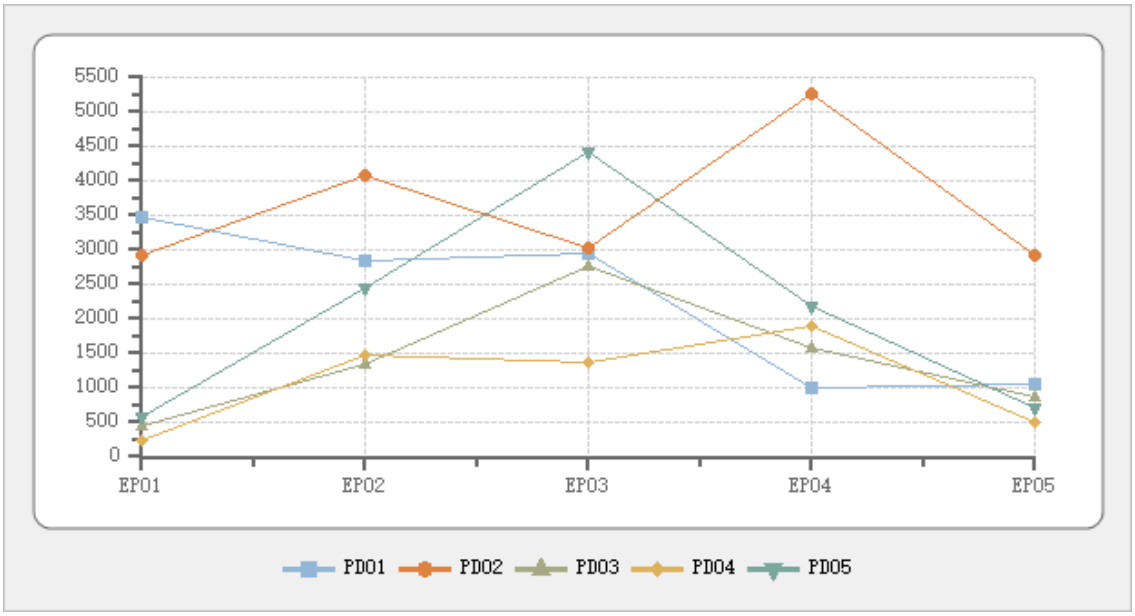
逻辑型，返回或者设置数据标示是否可见。

4.11.4 横向生成图表

前面的例子，都是一列一个图系，是纵向生成的。
假定有下图所示的一个数据表:

	产品	EP01	EP02	EP03	EP04	EP05
1	PD01	3480	2840	2940	990	1040
2	PD02	2910	4080	3030	5270	2910
3	PD03	440	1340	2770	1590	860
4	PD04	250	1470	1380	1890	490
5	PD05	590	2440	4420	2240	710

希望生成下面的图表，数据表中每行数据在图表中都对应一条线，也就是一行数据一个图系：



显然这个时候无法用绑定列的形式来生成上面的图表，只能手工编码添加数据点，大家可以参考下面的代码：

```
Dim Chart   As WinForm.Chart   '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Dim t As Table =   Tables("统计表1") '定义一个变量t引用数据表
Chart = e.Form.Controls("Chart1") '  引用窗口中的图表
```

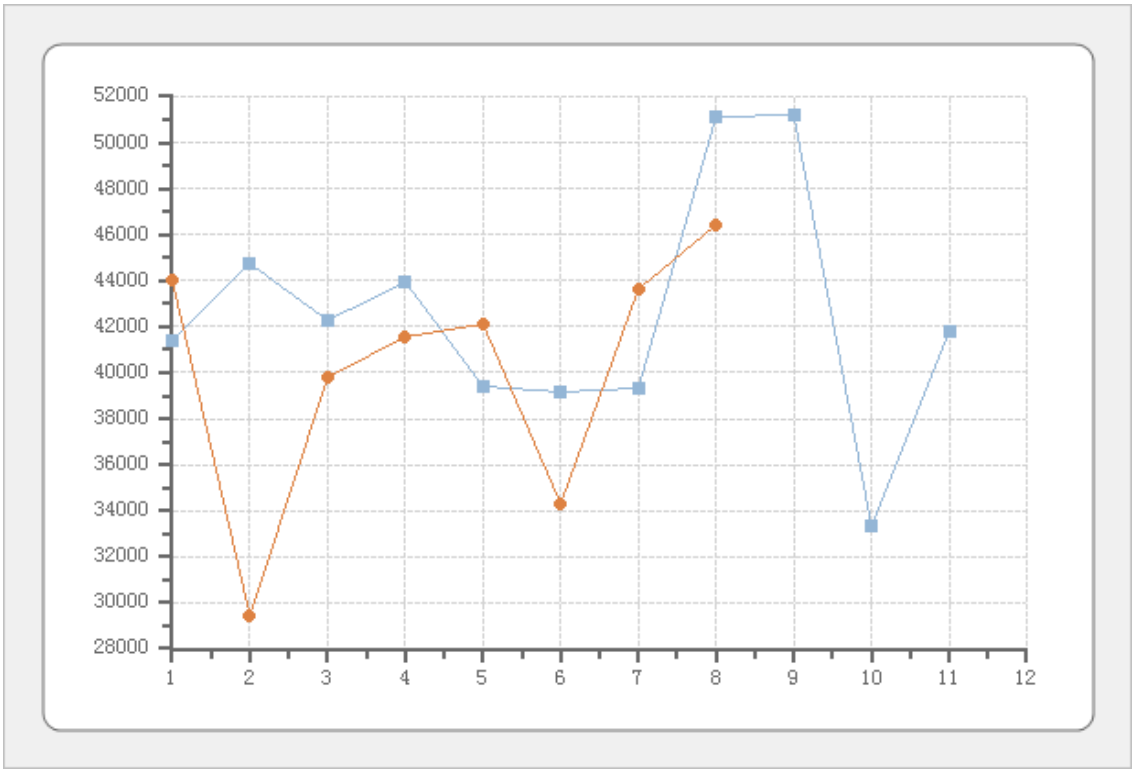
```
Chart.VisualEffect = False
Chart.SeriesList.Clear() '清除图表原来的图系
For r As Integer = 0 To t.Rows.count - 1
    Series = Chart.SeriesList.Add() '增加一个图系
    Series.Text = t.rows(r)("产品")  '设置图系的标题
    Series.Length = t.Cols.Count - 1  '设置图系的长度
    For c As Integer = 1 To t.Cols.count -1
        Series.X(c - 1) = c - 1
        Series.Y(c - 1) = t.Rows(r)(c)
        Chart.AxisX.SetValueLabel(c - 1, t.Cols(c).caption) '指定字符表示
    Next
Next
Chart.AxisX.AnnoWithLabels = True '启用字符标示
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South  '图列显示在南方(底端)
```

4.11.5 无法绑定的情况

假定有下图所示的数据表：

	年	月	数量
1	2014	1	41426
2	2014	2	44734
3	2014	3	42236
4	2014	4	43971
5	2014	5	39369
6	2014	6	39201
7	2014	7	39295
8	2014	8	51104
9	2014	9	51236
10	2014	10	33347
11	2014	11	41818
12	2014	12	37859
13	2015	1	44005
14	2015	2	29402
15	2015	3	39801
16	2015	4	41544
17	2015	5	42099
18	2015	6	34300
19	2015	7	43653
20	2015	8	46453

希望生成下面的图表：



显然我们无法使用绑定列的方法，只能手工编码增加数据点，不过代码也很简单：

```
Dim Chart As WinForm.Chart '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart= e.Form.Controls("Chart1") ' 引用窗口中的图表
Chart.SeriesList.Clear() '清除图表原来的图系
For Each Year As String In DataTables("统计表1").GetValues("年")
    Series = Chart.SeriesList.Add() '每年一个图系
    Series.Text = Year  '指定图系名称为年份
    Series.Length = 12  '指定图系的数据点数,1年12个月
    Chart.AxisX.Min = 1  '指定X轴的最小值,也就是从1月开始
    Chart.AxisX.Max = 12 '指定X轴的最大值,一年只有12个月
    For Month As Integer = 1 To 12 '逐月设置坐标
        Series.X(Month-1) = Month      '指定水平坐标为月份
        Dim r As DataRow = DataTables("统计表1").Find("年 = " & Year & " And 月 = " & Month)'找出对应年月的行
        If r IsNot Nothing '如果找到对应年月的行
            Series.Y(Month-1) = r("数量") '指定数量作为垂直坐标
        End If
    Next
Next
```

可以看到，只要将基础知识掌握，不管需求如何变化，都是万变不离其宗，都能轻松应对。

4.11.6 甘特图

本节的内容可以参考 CaseStudy 目录下的文件：甘特图.Table

Gantt(甘特图)用于表示任务的持续时间或进度，是一个很特殊的图表，设置方法和其他的图表有很大的不同。

甘特图用一个图系表示一个任务，用图系集合的 AddGanttSeries 方法来增加甘特图任务图系。

语法：

AddGanttSeries(taskName,startTime, endTime)

taskName： 任务名称

startTime：开始时间

endName: 结束时间

或者

AddGanttSeries(taskName,startTime(), endTime())

taskName： 任务名称

startTime：开始时间数组

endName: 结束时间数组

另外图系集合提供了一个 AddGanttLabels，用于给甘特图加上标示。

语法:

AddGanttLabels(FormatStr)

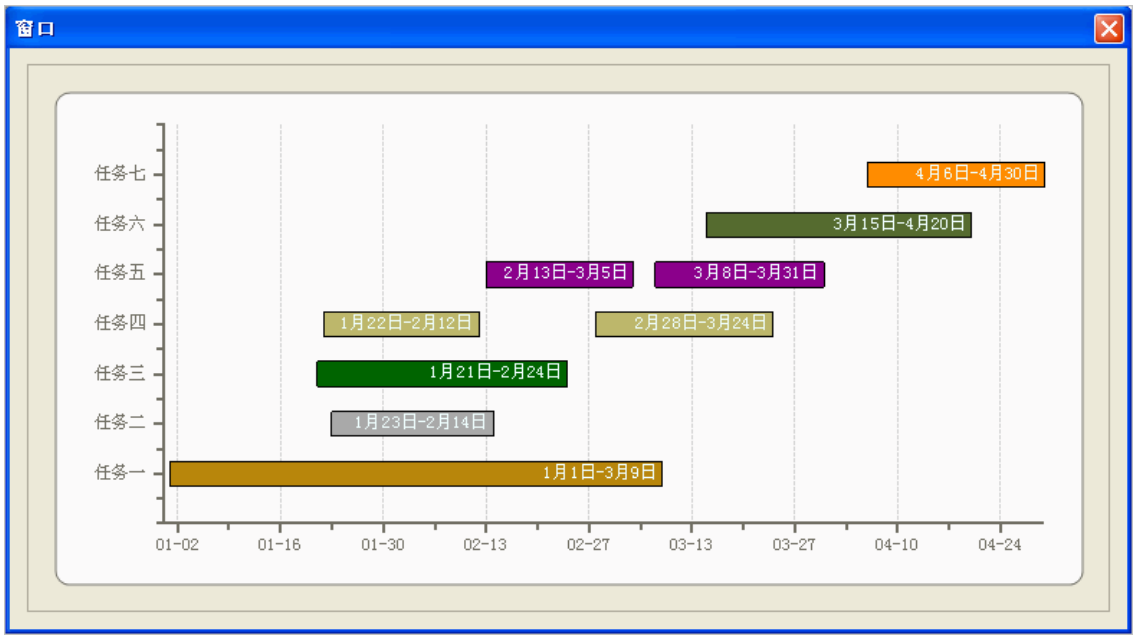
FormatStr: 格式字符串。

甘特图的 Y 轴刻度是时间，不是数值，需要用坐标轴的 MinDate 和 MaxDate 设置 Y 轴的起始日期和终止日期。

示例

```
Dim Chart As WinForm.Chart
Chart= Forms("窗口 1").Controls("Chart1")
Chart.ChartType = ChartTypeEnum.Gantt '图表类型为甘特图
Chart.Inverted = True '对调 X 轴和 Y 轴位置
Chart.AxisY.DateType= True 'Y 轴为日期型
Chart.AxisY.AnnoFormatString = "MM-dd" '设置 Y 轴的标示格式
Chart.AxisY.MinDate = #01/01/2010# '设置 Y 轴的起始日期
Chart.AxisY.MaxDate = #04/30/2010# '设置 Y 轴的终止日期
Chart.AxisX.GridMajorVisible = False '隐藏 X 轴的网格线
With Chart.SeriesList
    .AddGanttSeries("任务一", #1/1/2010#, #3/9/2010#)
    .AddGanttSeries("任务二", #1/23/2010#, #2/14/2010#)
    .AddGanttSeries("任务三", #1/21/2010#, #2/24/2010#)
    .AddGanttSeries("任务四", New DateTime() {#1/22/2010#, #2/28/2010#}, New DateTime() {#2/12/2010#, #3/24/2010#})
    .AddGanttSeries("任务五", New DateTime() {#2/13/2010#, #3/8/2010#}, New DateTime() {#3/5/2010#, #3/31/2010#})
    .AddGanttSeries("任务六", #3/15/2010#, #4/20/2010#)
    .AddGanttSeries("任务七", #4/6/2010#, #4/30/2010#)
    .AddGanttLabels("M 月 d 日")
End with
```

这是生成的甘特图：



4.11.7 ChartBuilder

ChartBuilder 除了具备 WinForm.Chart 的所有属性和方法外，还具备以下两个属性：

PrintWidth:　打印宽度，单位为毫米

PrintHeight:　打印高度，单位为

ChartBuilder 不需要加入到窗口中，即可生成图片，主要用于为报表生成图表。

示例一

我们希望能够为下面的表生成一个图表，生成的图表并不希望在窗口显示，而是直接保存为一个文件。

	产品	数量
1	PD01	11290
2	PD02	18200
3	PD03	7000
4	PD04	5480
5	PD05	10400

代码：

```
Dim Chart As New ChartBuilder '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart.DataSource = "统计表" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Chart.PrintWidth = 140 '宽度设为 140 毫米
Chart.PrintHeight = 100 '高度设为 100 毫米
Chart.SaveImage("c:\data\test.bmp") '将生成的图表保存为文件。
```

上面的代码可以直接在命令窗口执行，无须新建一个窗口并插入 Char 控件来测试

示例二

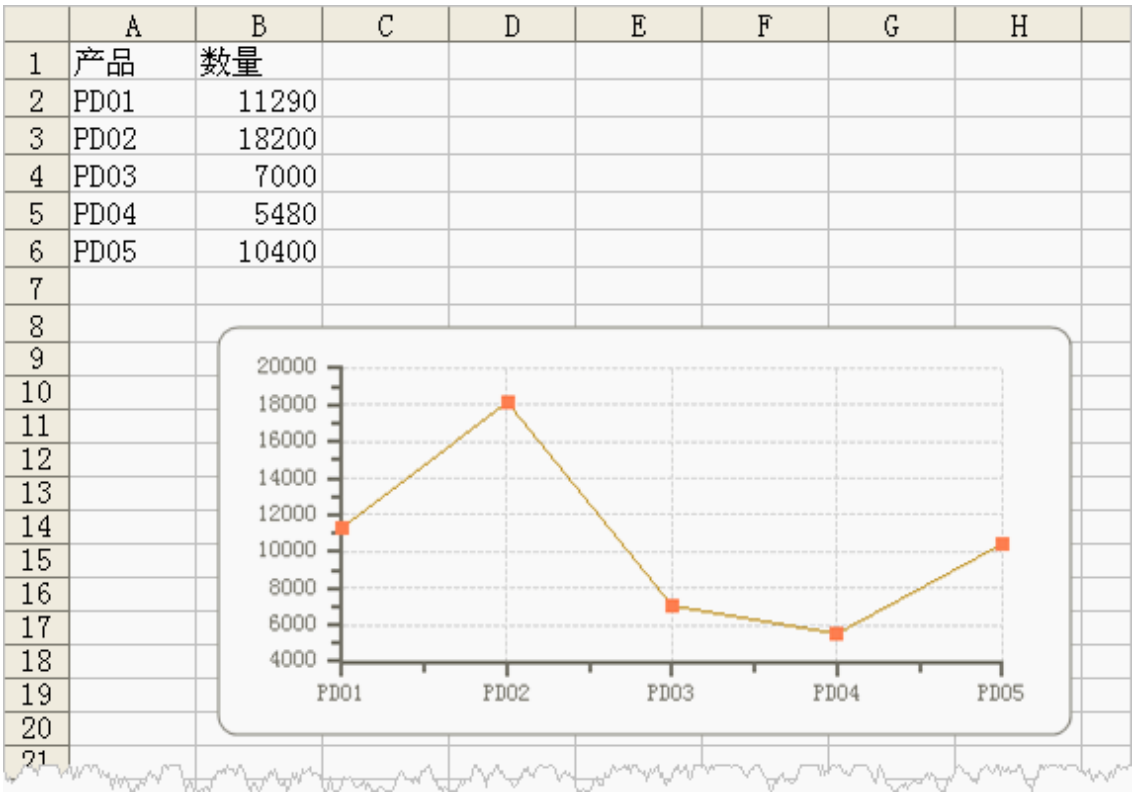
下面的代码利用 ChartBuilder 来生成图表，然后直接插入到 Excel 报表中：

例如：

```
'生成 Excel 报表
Dim b As New XLS.Book
Dim t As Table = Tables("统计表")
Dim s As XLS.Sheet = b.Sheets(0)
For c As Integer = 0 To t.Cols.Count -1 '添加列标题
    s(0, c).Value = t.Cols(c).Name
Next
For r As Integer = 0 To t.Rows.Count - 1 '填入数据
    For c As Integer = 0 To t.Cols.Count -1
        s(r +1, c).Value = t.rows(r)(c)
    Next
Next
'生成图表
Dim Chart As New ChartBuilder '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart.DataSource = "统计表" '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.X.DataField = "产品" 'X 轴绑定到产品列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Chart.PrintWidth = 120
Chart.PrintHeight = 60
s(t.rows.Count + 2,1).Value = New XLS.Picture(Chart.Image)
b.Save("c:\reports\test.xls")
Dim Proc As New Process
Proc.File = "c:\reports\test.xls"
Proc.Start()
```

上面的代码可以直接在命令窗口执行，无须新建一个窗口并插入 Char 控件来测试。

执行结果：



4.11.8 为 Excel 报表增加图表功能

Excel 报表并没有图表功能，但是有引用图片文件的功能。

如果我们将 Chart 控件或 ChartBuilder 类型生成的图表保存为一个约定的文件，并在 Excel 报表中引用此约定的文件，那么就等于间接地为 Excel 报表加上了图表功能。

示例

1、设计模版的时候，引用约定的图片文件"myimg.wmf"。

2、在窗口中加入一个 Button 控件。

3、Button 控件的代码设置为：

```
'生成图片文件：
Dim Chart As New ChartBuilder
Dim Series As WinForm.ChartSeries
Chart.DataSource = "统计表"
Chart.SeriesList.Clear()
Series = Chart.SeriesList.Add()
Series.X.DataField = "产品"
Series.Y.DataField = "数量"
Chart.PrintWidth = 140
```

```
Chart.PrintHeight = 100
Chart.SaveImage(ProjectPath & "Images\myimg.wmf")
'生成 Excel 报表：
Dim Book As New XLS.Book(ProjectPath & "Attachments\销售统计.xls")
Book.Build()
Book.Save(ProjectPath & "Reports\销售统计.xls")
```

提示：图片文件推荐采用 wmf 格式，这样缩放图片的时候，就不至于变形。

4.12 ListView

4.12.1 快速入门

4.12.1.1 关于 ListView

ListView 其实就是一个控件，但是这个控件非常强大，所以单独用一章讲述，就想我们单独用一章《生成图表》来介绍 Chart 控件一样。

ListView 可以用多种方式来展示数据，灵活且诱人。

大图标模式：



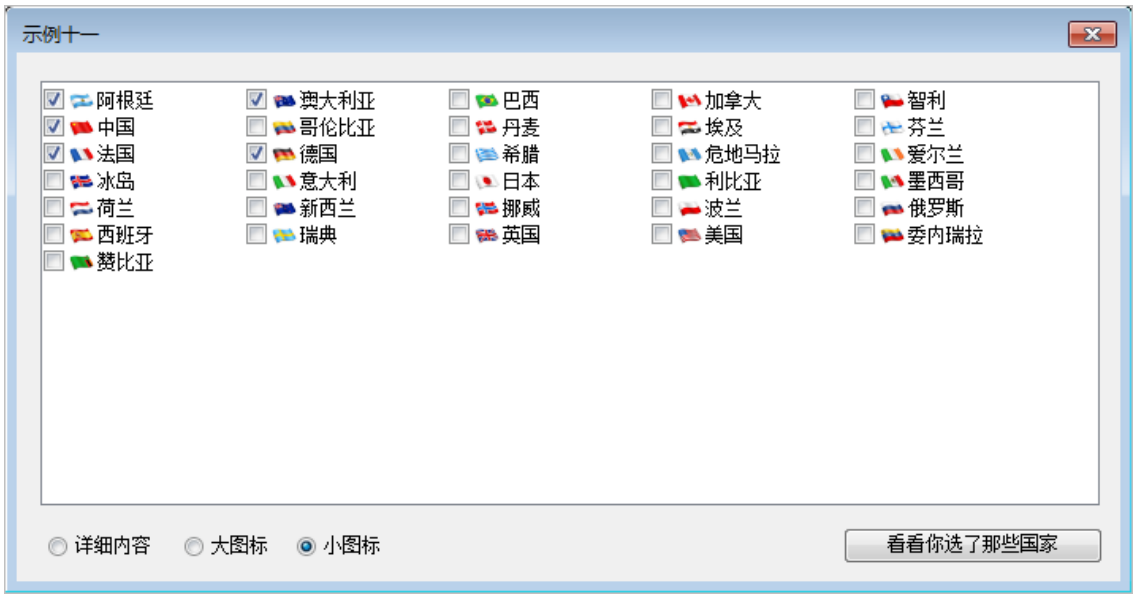
小图标模式：



详细信息模式：

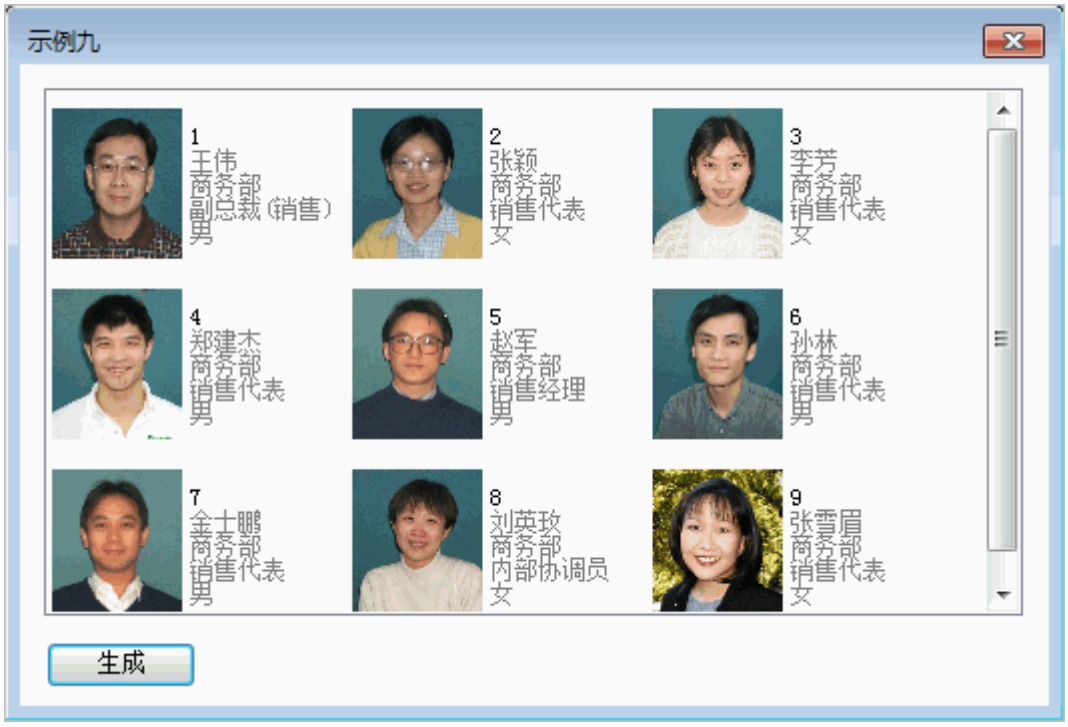


可以显示复选框：





甚至可以用来显示同时显示照片和信息：



4.12.1.2 开始第一步

本节的例子可以打开 CaseStudy 目录下的文件“ListView.Table”，有关代码在名为“示例一”的窗口中。本节的任务是显示一个下图所示的窗口:





设计步骤：

1、新建一个窗口，加入一个 ListView、一个 Button、两个 RadionButton。

2、按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片

lvw.Images.AddImage("China","China.Ico","China48.Ico") '增加代表中国的 一对图标,图标键值为"China"
lvw.Images.AddImage("USA","USA.Ico","USA48.Ico") ''增加代表美国的一对图标,图标键值为"USA"
lvw.Images.AddImage("UK","UK.Ico","UK48.Ico") ''增加代表英国的一对图标,图标键值为"UK"

Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
r.Text = "中国" '指定行的标题
r.ImageKey = "China" '指定行的图标键值

r = lvw.Rows.Add() '再增加一行
r.Text = "美国" '指定行的标题
r.ImageKey = "USA" '指定行的键值

r = lvw.Rows.Add() '再增加一行
r.Text = "英国" '指定行的标题
r.ImageKey = "UK" '指定行的键值
```

3、RadionButton1(大图标)的 CheckedChanged 事件代码设置为：

```
If e.Sender.Checked = True Then
    Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
    lvw.View = ViewMode.LargeIcon '显示模式切换为大图标
```

```
End If
```

4、RadionButton2(小图标)的 CheckedChanged 事件代码设置为：

```
If e.Sender.Checked = True Then
    Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
    lvw.View = ViewMode.SmallIcon '显示模式切换为小图标
End If
```

这样打开窗口后，单击按钮可以生成 ListView，单击两个 RadioButton，可以在大图标和小图标两个显示模式中切换。

提示：

1、 所有需要使用的图标，务必事先复制到项目文件所在目录的 Images 或 Attachments 子目录中。

2、其实按钮的代码可以简化为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
Dim nms() As String = {"中国","美国","英国"} '定义标题
Dim ics() As String = {"China","USA","UK"} '定义图标键值
For i As Integer = 0 To nms.Length - 1
    lvw.Images.AddImage(ics(i),ics(i) & ".ico", ics(i) & "48.ico") '添加代表这个国家的一对图标
    Dim r As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
    r.Text = nms(i) '指定标题
    r.ImageKey = ics(i) '指定图标键值
Next
```

采用这种方式编写代码的前提是，根据图标键值可以推导出对应的图标文件，例如前面的例子，假定键值为"UK"，对应的小图标为"UK.ico"，对应的大图标为"UK48.ico"。

前面之所有采用繁琐的方式编写代码，主要是为了方便初学者理解代码；但是为了简洁，今后的例子将采用上面简化后的方法编写。

4.12.1.3 图片的管理

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例二”的窗口，此窗口较为完整地演示了 ListView 是如何管理图片的：



ListView 诱人功能之一，就是可以显示图片，而且可以在大图片和小图片之间切换。
所以在使用 ListView 之前，通常应该先准备好图片，并将这些图片文件复制到项目文件所在目录的 Images 或 Attachments 子目录中。
ListView 有个 Images 属性，该属性类似一个集合(当然不是集合)，有自己的一套属性和方法，用于管理所有即将在 ListView 中使用的图片。

下面逐一介绍其属性和方法。
通过下面三个方法，可以向 Images 添加图片：

- **AddImage**

  用于添加指定键值对应的一对图片，语法：

  AddImage(Key, SmallImageFile, LargeImgeFile)

  Key：字符型，图片键值，ListView 通过这个键值来调用对应的图片。
  SmallImageFile：字符型，指定小图片对应的图片文件
  LargeImgeFile： 字符型，指定大图片对应的图片文件

- **AddSmallImage**

  用于添加指定键值对应的小图片，语法：

  AddSmallImage(Key, SmallImageFile)

Key：字符型，图片键值，ListView 通过这个键值来调用对应的图片。
SmallImageFile：字符型，指定小图片对应的图片文件

- **AddLargeImage**

  用于添加指定键值对应的大图片，语法：

  AddLargeImage(Key, LargeImgeFile)

  Key：字符型，图片键值，ListView 通过这个键值来调用对应的图片。
  LargeImgeFile： 字符型，指定大图片对应的图片文件

通过下面的方法，可以删除指定键值对应的图片，也可以一次清除所有图片：

- **RemoveImage**

  用于删除指定键值对应的图片，包括大图片和小图片，语法：

  RemoveImage(Key)

  Key：要删除的图片对应的键值

- **RemoveSmallImage**

  用于删除指定键值的小图片，语法：

  RemoveSmallImage(Key)

  Key：要删除的图片对应的键值

- **RemoveLargeImage**

  用于删除指定键值的大图片，语法：

  RemoveLargeImage(Key)

  Key：要删除的图片对应的键值

- **Clear**
  清除所有图片，包括大图片和小图片。

默认情况下，ListView 大图片的尺寸为 48*48 像素像素，小图片的尺寸为 16*16 像素，通过下面的属性可以改变默认的尺寸：

- **LargeSize**

  用于设置大图片尺寸，例如：

  lvw.Images.LargeSize = New Size(128,128)

- **SmallSize**

  用于设置小图片尺寸，例如：

  lvw.Images.SmallSize = New Size(48,48)

4.12.1.4 显示多列内容

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例三”的窗口。
前面的例子中，ListView 都只显示了图标和标题，实际上 ListView 可以像表一样，同时显示多列内容。
本节的任务是用 ListView 设计一个下图所示的窗口，显示各国更详细的信息：



设计步骤：
1、在窗口中分别插入一个 ListView 和 Button 控件，以及三个 RadionButton 控件：
2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Columns.Clear() '清除原来的列
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.Details '显示模式为详细信息
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '定义列名
Dim wds() As Integer = {100,100,150,80,80,80} '定义列宽
```

```
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i) '指定标题,这里标题和列名相同
    c.Width = wds(i) '指定列宽
Next
Dim nms() As String = {"中国","美国","英国","俄罗斯","法国"} '定义国名
Dim rks() As String = {"125,382","31,525","6,240", "14,410","6,580"} '各国人口数量
Dim mji() As String = {"96,000,000","9,629,091","244,820", "17,075,400","551,602"} '各国国土面积
Dim lan() As String = {"中文","英语","英语","俄语","法语"} '各国语言
Dim zhj() As String = {"佛教","基督教","基督教", "东正教","天主教"} '各国主要宗教
Dim szh() As String  = {"亚洲","美洲","欧洲","欧洲","欧洲"} '各国所在洲
Dim ics() As String = {"China","USA","UK","Russia","France"} '各国对应的图标键值
For i As Integer = 0 To nms.Length - 1
    lvw.Images.AddImage(ics(i),ics(i) & ".ico", ics(i) & "48.ico") '添加代表这个国家的一对图标
    Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    r.ImageKey = ics(i) '指定图标键值
    r("国家") =  nms(i) '设置各列的值
    r("人口") =  rks(i) & "  万"
    r("语言") = lan(i)
    r("面积") = mji(i) & "  平方公里"
    r("宗教") = zhj(i)
    r("洲") = szh(i)
Next
```

3、分别设置三个 RadionButton 控件的代码：

详细信息：

```
If e.Sender.Checked = True Then
    Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
    lvw.View = ViewMode.Details '显示模式切换为详细内容
End If
```

大图标：

```
If e.Sender.Checked = True Then
    Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
    lvw.View = ViewMode.LargeIcon '显示模式切换为大图标
End If
```

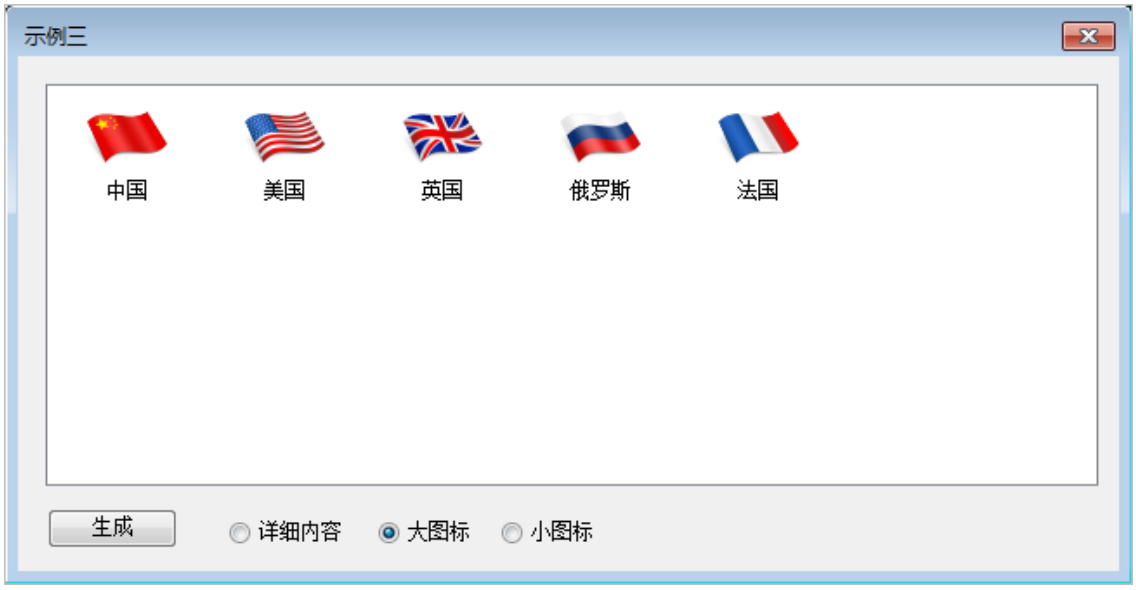
小图标：

```
If e.Sender.Checked = True Then
    Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
    lvw.View = ViewMode.SmallIcon '显示模式切换为小图标
End If
```

一些提示：

1、只有显示模式为详细信息的时候，ListView 才会显示多列数据。

2、当显示模式为大图标或者小图标的时候，第一列的内容作为标题使用:



4.12.1.5 显示表中数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例四”的窗口。

本节的任务是从一个 DataTable 中提取数据，生成下图所示的 ListView：



设计步骤：

1、在窗口中分别插入一个 ListView 和 Button 控件。

2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.Columns.Clear() '清除原来的列
```

```
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.Details '显示模式为详细信息
lvw.GridLines = True '显示网格线
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {100,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
For Each dr As DataRow In DataTables("国家").DataRows '从数据表中提取数据
    Dim Key As String = dr("图标")
    Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的一对图标
    r.ImageKey = Key '指定图标键值
    For Each cl As String In cls '逐列取值
      Select Case cl
        Case "人口","面积" '如果是人口列或面积列
          r(cl) = format(dr(cl),"#,000") '则显示千位分割符号
        Case Else
          r(cl) = dr(cl)
      End Select
    Next
Next
lvw.ResumeRedraw() '恢复绘制
```

一些提示：

1、需要向ListView中大量添加行的时候，为提高效率，应先执行StopRedraw方法，暂停绘制ListView，添加行结束后，再执行ResumeRedraw方法，恢复绘制。
2、可以将ListView的GridLines属性设置为True，这样ListView可以显示网格线，外观和普通的表类似。

4.12.1.6 分组显示数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例五”的窗口。
本节的目的是设计一个下图所示的窗口，在 ListView 中用分组的形式显示各国：



设计步骤：

1、在窗口中分别插入一个 ListView 和 Button 控件。

2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Groups.Clear() '清除原来的分组
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.LargeIcon '用大图标模式
Dim gps() As  String  = {"欧洲","亚洲","非洲"} '定义分组名称
Dim nms() As String = {"中国","德国","英国","日本","俄罗斯","法国","埃及","利比亚","赞比亚"} '定义标题
Dim ics() As String = {"China","Germany","UK","Japan","Russia","France","Egypt","Libya","Zambia"} '定义图标键值
For Each gp As String In gps
    Dim vg As WinForm.ListViewGroup = lvw.Groups.Add() '增加分组
    vg.Name = gp '指定分组名称
    vg.Text = gp & "国家" '指定分组标题
Next
For i As Integer = 0 To nms.Length - 1
    lvw.Images.AddImage(ics(i),ics(i) & ".ico", ics(i) & "48.ico") '添加代表这个国家的小图标
    Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    r.Text = nms(i) '指定标题
    r.ImageKey = ics(i) '指定图标键值
    Select Case nms(i)  '指定各个国家所属的分组
      Case "德国","英国","俄罗斯","法国"
        r.Group = "欧洲"
      Case "中国","日本"
        r.Group = "亚洲"
      Case "埃及","利比亚","赞比亚"
        r.Group = "非洲"
```

```
    End Select
Next
```

提示：

必须先添加分组到Groups集合中，然后指定行的Group属性，将其分配到相关分组中。

4.12.1.7 分组显示表中数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例六”的窗口。
本节的目的是设计一个下图所示的窗口，ListView 从数据表中提取数据，并用分组的形式显示各国 的详细信息：



设计步骤：

1、在窗口中分别插入一个 ListView、Button 控件和 CheckBox 控件，以及三个 RadioButton 控件

2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '停止绘制
lvw.Groups.Clear() '清除原来的分组
lvw.Columns.Clear() '清除原来的列
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.Details '显示模式为详细信息
Dim cls() As String = {"国家","人口","面积","语言","宗教"} '定义列名
Dim wds() As Integer = {100,100,150,80,80} '定义列宽
```

```
For i As Integer = 0 To  cls.Length - 1  '增加列
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i) '指定标题,这里标题和列名相同
    c.Width = wds(i) '指定列宽
Next
For Each zh As String In DataTables("国家").GetValues("洲") '增加分组
    Dim grp As WinForm.ListViewGroup = lvw.Groups.Add()
    grp.Name = zh
    grp.Text = zh
Next
For Each dr As DataRow In DataTables("国家").DataRows
    Dim Key As String = dr("图标")
    Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的一对图标
    r.ImageKey = Key '指定图标键值
    r.Group = dr("洲") '指定所属分组
    For Each cl As String In cls '逐列取值
      Select Case cl
        Case "人口","面积" '如果是人口列或面积列
          r(cl) = format(dr(cl),"#,000") '则显示千位分割符号
        Case Else
          r(cl) = dr(cl)
      End Select
    Next
Next
lvw.ResumeRedraw() '恢复绘制
```

3、将 CheckBox 的 CheckedChanged 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.ShowGroups = e.Sender.Checked
```

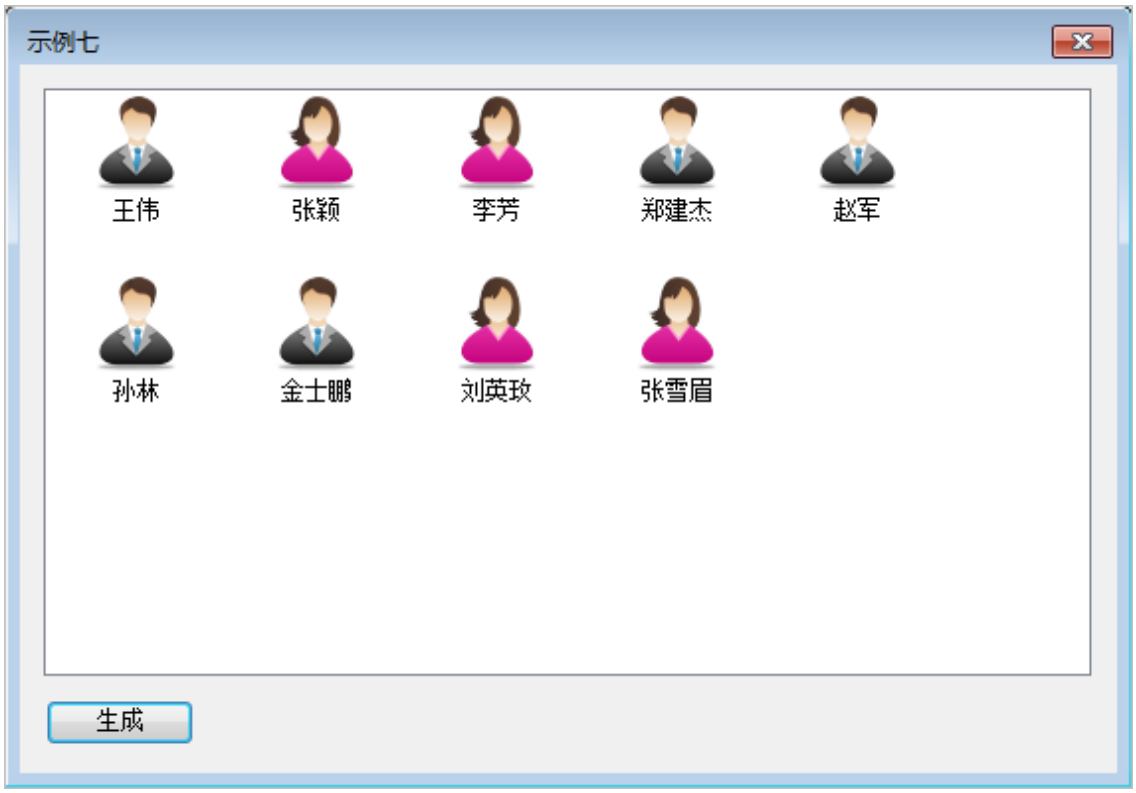
一些提示：

1、需要向ListView中大量添加行的时候，为提高效率，应先执行StopRedraw方法，暂停绘制ListView，添加行结束后，再执行ResumeRedraw方法，恢复绘制。
2、通过设置ListView的ShowGroups属性，可以在普通模式和分组模式中切换。

4.12.1.8 用图片分类数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例 七”的窗口。

前面的例子，每一行都有自己独有的图片。
还有很多时候，是相同性质的行，显示同样的图片，通过图片来显示表示此行所属的分类。
本节的任务是从员工表中提取数据，生成下图所示的 ListView，男员工和女员工分别用不同的图标表示：



设计步骤：

1、在窗口中分别插入一个 ListView 和 Button 控件。
2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.LargeIcon '显示模式为大图标
lvw.Images.AddLargeImage("Man", "Man48.ico") '添加代表男性的一对图标
lvw.Images.AddLargeImage("Woman", "Woman48.ico") '添加代表女性的一对图标
For Each dr As DataRow In DataTables("员工").DataRows '从数据表中提取数据
    Dim vr As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
    vr.Text = dr("姓名")   '设置标题
    If dr("性别") = "男" Then '设置分组
        vr.ImageKey = "Man"
    Else
        vr.ImageKey = "Woman"
    End If
Next
lvw.ResumeRedraw() '恢复绘制
```

4.12.1.9 直接显示员工照片

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例八”的窗口。

本节的任务是从员工表中提取数据，在 ListView 中显示每个员工的姓名和照片：



设计步骤：

1、在窗口中分别插入一个 ListView 和 Button 控件。

2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.LargeIcon '显示模式为大图标
lvw.Images.LargeSize = New Size(130,150) '定义大图标尺寸
For Each dr As DataRow In DataTables("员工").DataRows '从数据表中提取数据
    Dim Key As String = dr("照片") '获取此员工的照片文件
    lvw.Images.AddLargeImage(Key, Key) '添加照片,直接用文件名作为图片键值
    Dim vr As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    vr.Text = dr("姓名")  '设置标题
    vr.ImageKey = Key '设置图片键值
Next
```

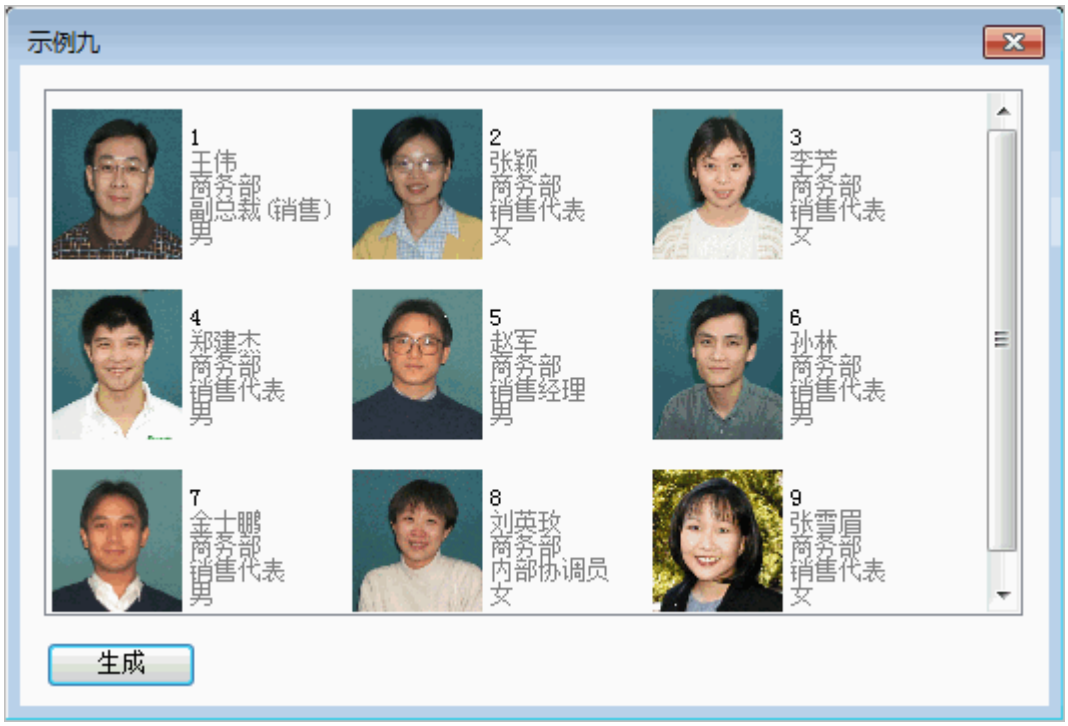
```
lvw.ResumeRedraw() '恢复绘制
```

4.12.1.10 用平铺方式显示数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例九”的窗口。

之前已经接触了大图标、小图标和详细信息三种显示模式，下面介绍如何以平铺方式显示数据。

本节的任务是设计一个下图所示的窗口，ListView 以平铺方式显示员工的照片以及详细信息：



设计步骤：

1、在窗口中分别插入一个 ListView 和 Button 控件。

2、将按钮的 Click 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.Tile '显示模式为平铺
lvw.Images.LargeSize = New Size(65,75) '定义大图标尺寸
lvw.TitleSize = New Size(150,90)  '设置平铺区域的大小
Dim cls() As String = {"编号","姓名","部门","职务","性别"} '定义列名
For i As Integer = 0 To  cls.Length - 1  '增加列
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
Next
For Each dr As DataRow In DataTables("员工").DataRows '从数据表中提取数据
    Dim Key As String = dr("照片") '获取此员工的照片文件
```

```
    lvw.Images.AddLargeImage(Key, Key) '添加照片,直接用文件名作为图片键值
    Dim vr As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
    vr.ToolTipText = dr("备注")   '设置此行的动态提示
    vr.ImageKey = Key '设置图片键值
    For Each cl As String In cls '逐列取值
      vr(cl) = dr(cl)
    Next
Next
lvw.ResumeRedraw() '恢复绘制
```

一些提示：

1、默认情况下，ListView 会自动根据显示内容计算平铺区域的大小，我们也可以通过 TitleSize 属性明确设置平铺区域的大小。
2、由于备注列内容太长，不适合直接显示在 ListView 中，所以上面的例子中，我们将行的 ToolTipText 属性设置为员工的备注，这样在 ListView 中将鼠标移到某个员工，就会自动显示这个员工的备注。

4.12.1.11 响应用户的双击动作

ListView 有个 RowActivate 事件，用户双击某行后会触发此事件。
此事件并没有返回双击行的 e 参数，但是通过 ListView 的 Current 属性，可以获取当前行，也就是用户双击的行。

如果 ListView 只显示一列内容，我们可以通过行的 Text 属性，判断用户双击的是哪一行，例如：

```
Dim lr As WinForm.ListViewRow = e.Sender.Current
MessageBox.Show("你双击的是:" & lr.Text)
```

如果 ListView 显示多列内容，我们可以通过获取当前行指定列的内容，判断用户双击的是哪一行，例如：

```
Dim lr As WinForm.ListViewRow = e.Sender.Current
MessageBox.Show("你双击的是:" & lr("国家"))
```

4.12.1.12 处理选定的多行

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十”的窗口。
在默认情况下，按住 Ctrl 键单击 ListView 中的行，可以依次选定所单击的行:



通过 ListView 的 SelectedRows 属性，可以返回所有选定的行，例如上图按钮的 Click 事件代码为：

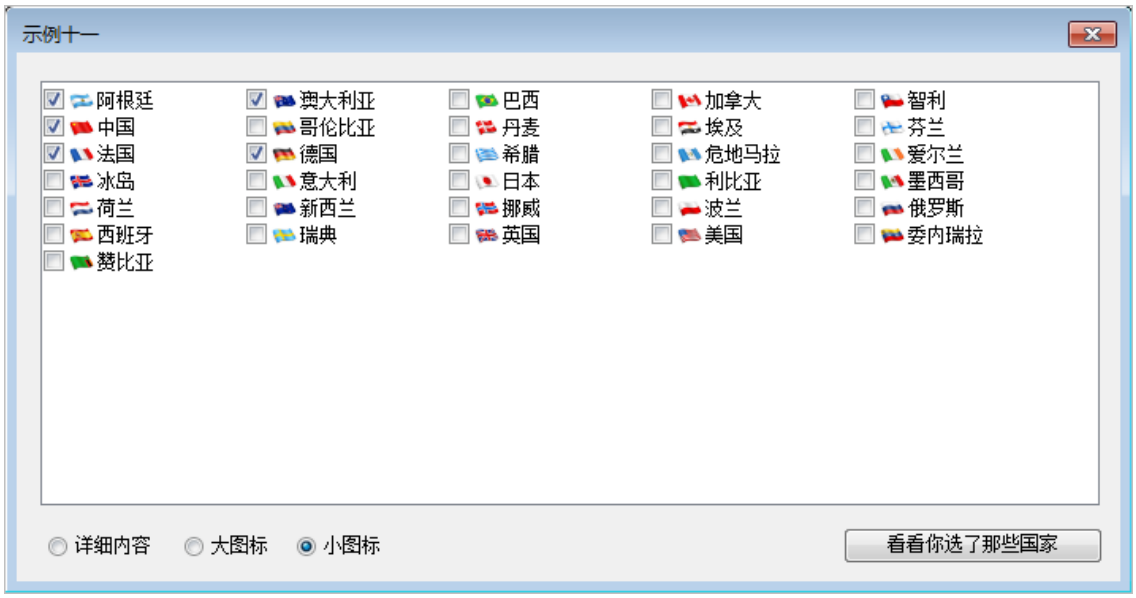
```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim vls As String = ""
For Each vr As WinForm.ListViewRow In lvw.SelectedRows
    vls = vls & "," & vr("国家")
Next
vls = vls.Trim(",")
If vls > "" Then
    MessageBox.show("你选择的国家有:" & vls)
Else
    MessageBox.show("你并没有选择任何国家!" )
End If
```

提示：
如果想禁止多选，将 ListView 的 MultiSelect 属性设置为 False 即可。

4.12.1.13 使用复选框选择行

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十一”的窗口。
ListView 有个 CheckBoxes 属性，将其设置为 True，可以显示复选框：





ListView 有个 CheckedRows 属性，用于返回通过复选框选择的所有行，例如上图按钮的 Click 事件代码为：

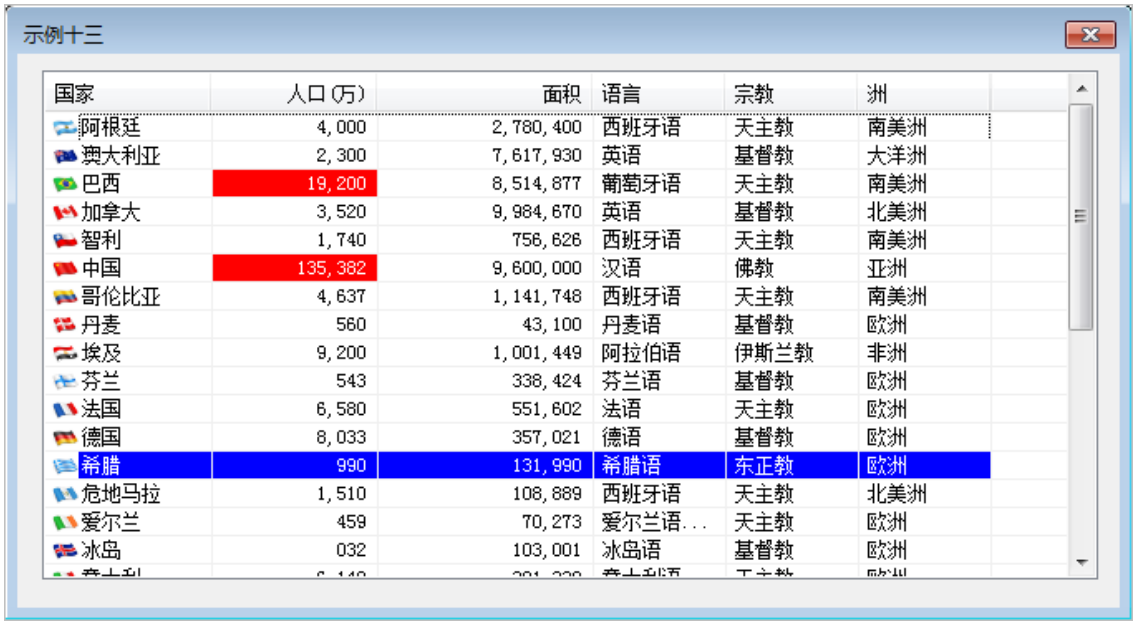
```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim vls As String = ""
For Each vr As WinForm.ListViewRow In lvw.CheckedRows
    vls = vls & "," & vr("国家")
Next
vls = vls.Trim(",")
If vls > "" Then
    MessageBox.show("你选择的国家有:" & vls)
Else
    MessageBox.show("你并没有选择任何国家!" )
End If
```

提示：

1、平铺模式不支持复选框。
2、最好将MultiSelect属性设置为False，因为既然用复选框了，就没有必要再提供按Ctrl多选的功能了。
3、在平铺模式下，不能显示复选框。

4.12.1.14 行与单元格的样式

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十三”的窗口。
ListView 的行有个 Cell 属性，可以获取此行指定列单元格。
ListView 的行和单元格都可以设置样式，包括字体、背景颜色和字体颜色：



以上面的图为例，假定要用蓝底白字整行显示东正教国家，以红底白字标出人口列中数值超过 10000 的单元格，代码为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.Columns.Clear() '清除原来的列
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.Details '显示模式为详细信息
lvw.GridLines = True '显示网格线
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {100,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
```

```
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
For Each dr As DataRow In DataTables("国家").DataRows '从数据表中提取数据
    Dim Key As String = dr("图标")
    Dim vr As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
    lvw.Images.AddSmallImage(Key, Key & ".ico") '添加代表这个国家的图标
    vr.ImageKey = Key '指定图标键值
    For Each cl As String In cls '逐列取值
        Select Case cl
            Case "人口","面积" '如果是人口列或面积列
                vr(cl) = format(dr(cl),"#,000") '则显示千位分割符号
            Case Else
                vr(cl) = dr(cl)
        End Select
    Next
    If dr("宗教") = "东正教" Then '用蓝底白字整行显示东正教国家
        vr.BackColor = Color.Blue
        Vr.ForeColor = Color.White
    End If
    If dr("人口") > 10000 Then '人口列数值大于10000的单元格,用红底白字显示
        vr.CellUseRowStyle = False '必须将行的CellUseRowStyle属性设置为False,才能单独设置此行单元格的格式
        With vr.Cells("人口")
            .BackColor = Color.Red
            .ForeColor = Color.White
        End With
    End If
Next
lvw.ResumeRedraw() '恢复绘制
```

一些提示:

1、在默认情况下，对于单元格格式的设置是不会生效的，只有将行的 CellUseRowStyle 属性设置为 False，才能单独设置此行单元格的样式。

2、列没有样式。

4.12.1.15 用虚拟模式显示大量数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十四”的窗口。

前面的例子，都是事先生成好 ListView 中的行，然后加入到 ListView。

如果你有大量数据要显示，例如有数万行，这种方式意味着长时间的等待，你的客户可能会向你

抱怨。
ListView 提供了一个虚拟模式，行并不需要事先生成，而是需要显示或调用的时候生成，这样我们可以瞬间“生成”一个数万行的 ListView，注意这个“生成”是带引号的，因为其实并没有生成。
ListView 提供了两个设置虚拟模式的属性：

- VirtualMode
  逻辑型，用于设置是否使用虚拟模式

- VirtualListSize
  整数型，用于设置虚拟模式下 ListView 的行数

ListView 提供了一个 RetrieveVirtualRow 事件，此事件在虚拟模式下需要生成行时触发，其 e 参数有：

- Row
  返回要生成的行

- Index
  返回此行的位置

示例
我们的任务是用虚拟模式显示一个表的数据。
设计步骤：
1、在窗口插入一个 ListView 控件
2、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.Details  '显示模式为详细内容
lvw.GridLines = True '显示网格线
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {120,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
   c.Text = cls(i)'指定列标题
   c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
lvw.VirtualMode = True  '以虚拟模式显示
lvw.VirtualListSize = DataTables("国家").DataRows.count '设置虚拟模式下要显示的行数
lvw.ResumeRedraw() '恢复绘制
```

上面的代码生成了 ListView 的列，但是没有添加行，只是通过以下两行代码使用虚拟模式，以及虚拟模式下的行数：

```
lvw.VirtualMode = True  '以虚拟模式显示
lvw.VirtualListSize = DataTables("国家").DataRows.count '设置虚拟模式下要显示的行数
```

3、将 ListView 的 RetrieveVirtualRow 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Sender
Dim dr As DataRow = DataTables("国家").DataRows(e.Index) '根据行位置从数据表中取得对应的DataRow
Dim Key As String = dr("图标")
lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的图标
e.Row.Imagekey = key '指定图标键值
For Each cl As WinForm.ListViewColumn In lvw.Columns '逐列取值
    Select Case cl.Name
        Case "人口","面积" '如果是人口列或面积列
            e.Row(cl.Name) = Format(dr(cl.Name),"#,000") '则显示千位分割符号
        Case Else
            e.Row(cl.Name) = dr(cl.Name)
    End Select
Next
```

提示：

1、虚拟模式不能显示分组，也无法用平铺模式显示数据，如果设置为平铺模式，将自动切换到大图标模式。

2、虚拟模式虽然能瞬间显示大量的数据，但是有一些情况也必须考虑，例如你有数万行要显示，每一行都有不同的图片，那么随着用户不断地滚动，真正显示完所有的行，那么在内存中维护这数万个图片也是一笔不小的开支，所以数据量大的时候，即使用虚拟模式，也应该考虑图片的开支，通常是一组类型的数据用一个图片(例如windows的资源管理其中，每种类型的文件用一个图标表示)，而不是每行数据一个图片，当然上面的描述并非绝对，一切以实际测试为准，用户感受好的设计，就是好的设计。

4.12.1.16 用分页方式显示大量数据

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十五”的窗口。

面对大量数据，除了虚拟模式，我们还可以自己编码，用分页的方式显示大量的数据。

下面我们就学习如何以分页的方式用 ListView 显示表中数据，假定每页显示 5 行。

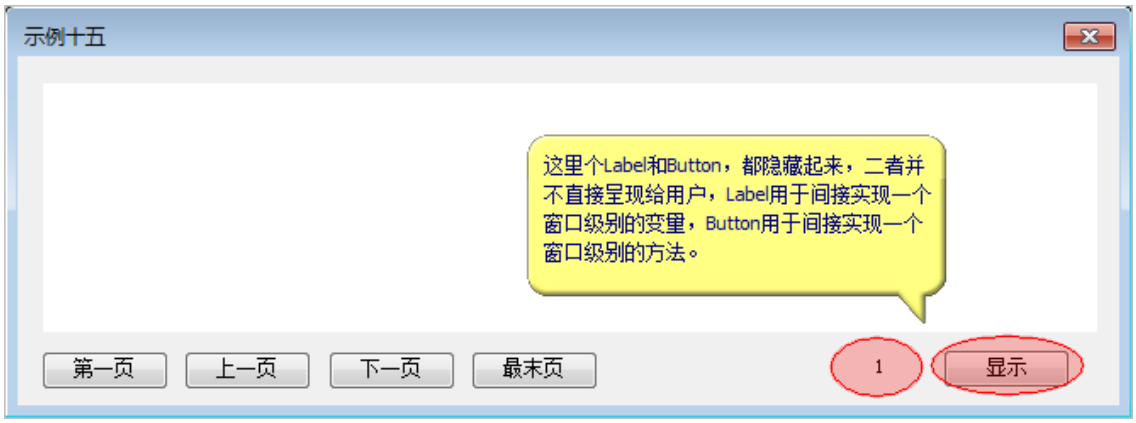
除了分页显示数据，本节还有一个要点，就是在 Foxtable 没有窗口级别变量和方法的情况下，如何在窗口的不同事件中共享变量和方法，希望大家留意。

设计步骤：

1、在窗口中插入一个 ListView、5 个 Button，1 个 Label，

2、将 Label 的文本设置为 1，可见属性设置为 False，由于 Foxtable 并没有提供窗口级别的变量，我们利用这个不可见的 Label 保存当面页号，方便在本窗口的各事件中调用。

3、将"显示"按钮的 Visible 属性设置为 False，由于 Foxtable 并没有提供窗口级别的方法，我们利用这个不可见的 Button 间接实现一个窗口界别的方法，方便在本窗口的各事件中调用。



4、窗口的 AfterLoad 事件设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.Details  '显示模式为详细内容
lvw.GridLines = True '显示网格线
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {120,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
lvw.ResumeRedraw() '恢复绘制
e.Form.Controls("btnShow").PerformClick '模拟单击显示按钮,显示第一页数据
```

5、设置各按钮的事件：

按钮	代码
显示	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1") Dim pg As Integer = e.Form.Controls("Label1").Text Dim idx1 As Integer = (pg - 1) * 5 '获取要显示的第一行 Dim idx2 As Integer = pg * 5 - 1 '获取要显示的最末行 idx2 = Math.Min(DataTables("国家").dataRows.Count -1,idx2) 'idx2不能超过表的最大行数 lvw.Images.Clear() '清除之前的图片 lvw.Rows.Clear() '清除之前的行 For idx As Integer = idx1 To idx2 Dim dr As DataRow = DataTables("国家").DataRows(idx) '根据行位置从

	数据表中取得对应的DataRow Dim vr As WinForm.ListViewRow = lvw.Rows.Add() '在ListView增加一行 Dim Key As String = dr("图标") lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的图标 vr.Imagekey = key '指定图标键值 For Each cl As WinForm.ListViewColumn In lvw.Columns '逐列取值 Select Case cl.Name Case "人口","面积" '如果是人口列或面积列 vr(cl.Name) = Format(dr(cl.Name),"#,000") '则显示千位分割符号 Case Else vr(cl.Name) = dr(cl.Name) End Select Next Next

第一页	Dim lbl As WinForm.Label = e.Form.Controls("Label1")
	Dim btn As WinForm.Button = e.Form.Controls("btnShow")
	Dim pg As Integer = lbl.Text '当前页号
	If pg <> 1 Then
	lbl.Text = 1
	End If
	btn.PerformClick() '模拟单击显示按钮

上一页	Dim lbl As WinForm.Label = e.Form.Controls("Label1")
	Dim btn As WinForm.Button = e.Form.Controls("btnShow")
	Dim pg As Integer = lbl.Text '当前页号
	If pg > 1 Then
	lbl.Text = pg - 1
	End If
	btn.PerformClick() '模拟单击显示按钮

下一页	Dim lbl As WinForm.Label = e.Form.Controls("Label1")
	Dim btn As WinForm.Button = e.Form.Controls("btnShow")
	Dim pg As Integer = lbl.Text '当前页号
	Dim pgs As Integer = math.Ceiling(DataTables("国家").DataRows.Count / 5) '计算出总页数
	If pg < pgs Then
	lbl.Text = pg + 1
	End If
	btn.PerformClick() '模拟单击显示按钮

最末页	Dim lbl As WinForm.Label = e.Form.Controls("Label1") Dim btn As WinForm.Button = e.Form.Controls("btnShow") Dim pg As Integer = lbl.Text '当前页号 Dim pgs As Integer = math.Ceiling(DataTables("国家").DataRows.Count / 5)

```
'计算出总页数
If pg < pgs  Then
    lbl.Text = pgs
End If
btn.PerformClick()  '模拟单击显示按钮
```

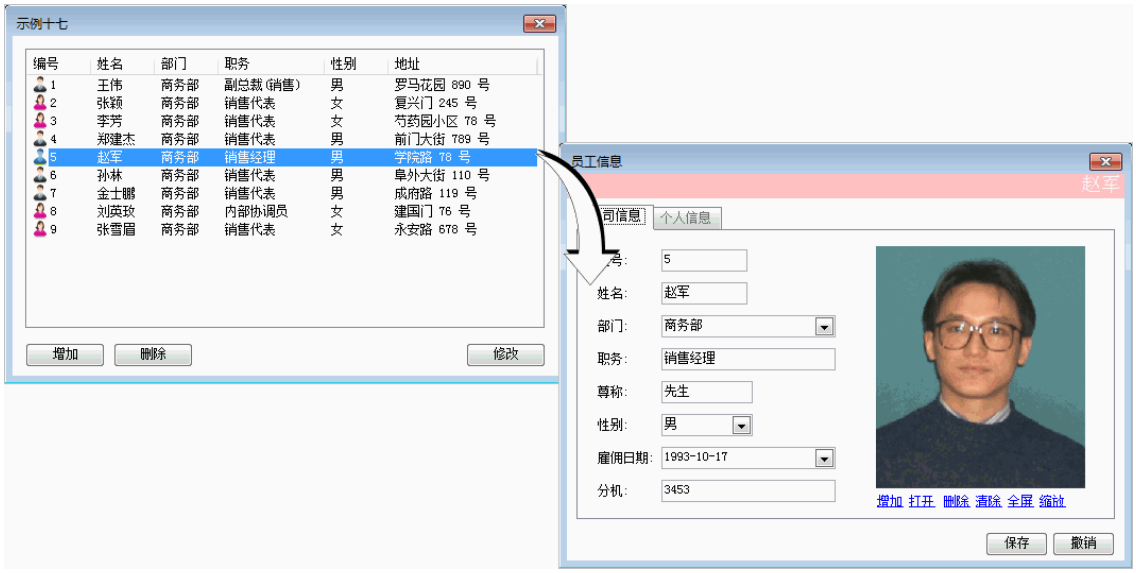
提示：

上面的例子用ListView来分页显示DataTable中的数据，有的时候，DataTable自己就是分页加载的，此时ListView无需考虑分页的问题，只需显示DataTable的所有数据即可。这种情况可以参考该 名为“示例十六”的窗口 ，由于分页加载数据，以及用ListView显示DataTable中所有数据，都已经多次讲述过，我们就不再单独讲述这个例子了。

4.12.1.17 ListView 和数据表的互动

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十七”的窗口。
ListView 本身的数据编辑功能很弱，只能编辑标题(显示多列内容时，第一列作为标题)。
多数时候，我们用 ListView 只是展示数据，而数据编辑和管理需要通过 DataTable 和 Table 来实现。

本节的任务：
1、在 ListView 中双击某员工，能打开一个编辑窗口来修改此员工的信息。
2、单击增加、删除按钮，可以增加、删除员工。



设计步骤:
1、新建一个窗口，插入一个 ListView 控件和三个按钮
2、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.View = ViewMode.Details
```

```
lvw.Images.AddSmallImage("Man", "Man.ico") '添加代表男性的图标
lvw.Images.AddSmallImage("Woman", "Woman.ico") '添加代表女性的图标
Dim cls() As String = {"编号","姓名","部门","职务","性别","地址"} '定义列名
Dim wds() As String = {60,60,60,100,60,140} '定义列宽
For i As Integer = 0 To  cls.Length - 1  '增加列
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Text = cls(i) '指定列标题
    c.Name = cls(i) '指定列名
    c.Width = wds(i) '指定列宽
Next
For Each dr As DataRow In DataTables("员工").DataRows '从数据表中提取数据
    Dim vr As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    For Each cl As String In cls '逐列取值
        vr(cl) = dr(cl)
    Next
    If dr("性别") = "男" Then '设置分组
        vr.ImageKey = "Man"
    Else
        vr.ImageKey = "Woman"
    End If
    vr.Tag= dr '将DataRow赋值给ListViewRow的Tag属性,将二者联系起来
Next
```

上述代码中最关键的一行是：

```
vr.Tag= dr '将 DataRow 赋值给 ListViewRow 的 Tag 属性,将二者联系起来
```

我们将 ListViewRow 对应的 DataRow 保存在 Tag 属性中，这样即可在其它事件中，通过 ListViewRow 的 Tag 属性获取对应的 DataRow。

Tag 属性可以存储任何类型的数据，当然多数时候存储的是 DataRow、Row 或对应的主键值。

3、将 ListView 的 RowActivate 事件代码设置为：

```
Dim btn As WinForm.Button = e.Form.Controls("btnEdit")
btn.PerformClick() '模拟单击修改按钮
```

4、三个按钮的代码分别设置为：

按钮	代码

增加	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
	Dim r As Row = Tables("员工").AddNew()
	Dim cnt As Integer = DataTables("员工").DataRows.Count '记录打开窗口前的行数
	Forms("员工信息").Open()
	If DataTables("员工").DataRows.Count = cnt '如果新增行已经保存
	  Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
	  For Each cl As WinForm.ListViewColumn In lvw.Columns '逐列取值
	  vr(cl.Name) = r(cl.Name)
	  Next

	``
	    If r("性别") = "男" Then '设置分组
	        vr.ImageKey = "Man"
	    Else
	        vr.ImageKey = "Woman"
	    End If
	    vr.Tag = r.DataRow '将DataRow赋值给ListViewRow的Tag属性,将二者联系起来
	End If
	``

删除	```
	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
	Dim vr As WinForm.ListViewRow = lvw.Current '获取ListView中选定的行
	If vr Is Nothing Then '如果不存在当前行,也就是内有选定任何一行
	    Return
	End If
	Dim dr As DataRow = vr.tag '获取此行对应的DataRow
	dr.Delete '从DataTable删除行
	vr.Delete '从ListView删除行
	``

修改	```
	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
	Dim vr As WinForm.ListViewRow = lvw.Current '获取ListView的当前行
	If vr Is Nothing Then '如果不存在当前行,也就是内有选定任何一行
	    Return
	End If
	Dim dr As DataRow = vr.tag '获取此行对应的DataRow
	Dim ps As Integer = Tables("员工").FindRow(dr)
	If ps >= 0 Then
	    Tables("员工").Position = ps
	    Forms("员工信息").Open
	    For Each cl As WinForm.ListViewColumn In lvw.Columns '逐列更新值
	        vr(cl.Name) = r(cl.Name)
	    Next
	End If
	``

4.12.1.18 虚拟模式与数据表的互动

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十 八”的窗口。

由于虚拟模式可以瞬间显示大量数据，所以实际使用 ListView 的时候，使用虚拟模式的几率非常大。

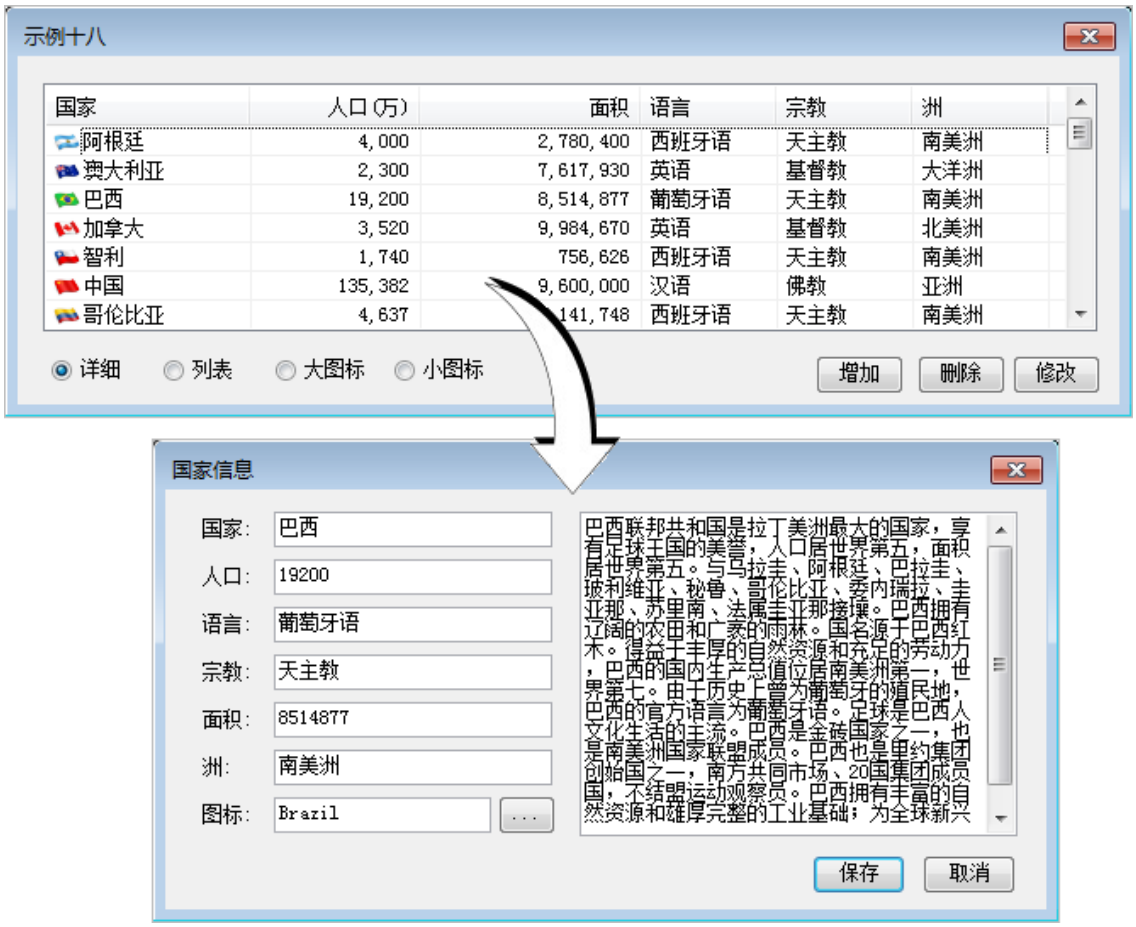
FoxTable 针对虚拟模式进行了大量的优化，与数据表的互动就是其中之一，你会看到，和常规模式相比，虚拟模式下和数据表的互动反倒变得简单了。

本节的任务是设计下图所示的系统，要求：

1、ListView 用虚拟模式生成

2、在 ListView 中双击某国家，能宽带一个编辑窗口来修改此国家的信息。

2、单击增加、删除按钮，可以增加、删除国家。



设计步骤：

1、新建一个窗口，插入一个 ListView 控件和三个按钮

2、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.Details  '显示模式为详细内容
lvw.GridLines = True '显示网格线
lvw.MultiSelect = False  '禁止选择多行
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {120,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
lvw.VirtualMode = True  '以虚拟模式显示
lvw.VirtualListSize = DataTables("国家").DataRows.count '设置虚拟模式下要显示的行数
```

```
lvw.ResumeRedraw() '恢复绘制
```

3、将ListView的RetrieveVirtualRow事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Sender
Dim dr As DataRow = DataTables("国家").DataRows(e.Index) '根据行位置从数据表中取得对应的DataRow
Dim Key As String = dr("图标")
lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的图标
e.Row.Imagekey = key '指定图标键值
e.Row.Tag = dr '将DataRow赋值给ListViewRow的Tag属性,将二者联系起来
For Each cl As WinForm.ListViewColumn In lvw.Columns '逐列取值
    Select Case cl.Name
        Case "人口","面积" '如果是人口列或面积列
            e.Row(cl.Name) = Format(dr(cl.Name),"#,000") '则显示千位分割符号
        Case Else
            e.Row(cl.Name) = dr(cl.Name)
    End Select
Next
```

4、将ListView的RowActivate事件代码设置为：

```
Dim btn As WinForm.Button = e.Form.Controls("btnModify")
btn.PerformClick() '模拟单击修改按钮
```

5、三个按钮的代码分别设置为：

按钮	代码
增加	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1") Dim r As Row = Tables("国家").AddNew() Dim cnt As Integer = DataTables("国家").DataRows.Count '记录打开窗口前的行数 Forms("国家信息").Open() If DataTables("国家").DataRows.Count = cnt '如果新增行已经保存 lvw.VirtualListSize = lvw.VirtualListSize + 1 '通知ListView,系统增加了一行 End If
删除	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1") Dim vr As WinForm.ListViewRow = lvw.Current '获取ListView中选定的行 If vr Is Nothing Then '如果不存在当前行,也就是内有选定任何一行 Return End If Dim dr As DataRow = vr.tag '获取此行对应的DataRow dr.Delete '从DataTable删除行 vr.Delete '从ListView删除行
修改	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1") Dim vr As WinForm.ListViewRow = lvw.Current '获取ListView的当前行

```
If vr Is Nothing Then '如果不存在当前行,也就是内有选定任何一行
    Return
End If
Dim dr As DataRow = vr.tag '获取此行对应的DataRow
Dim ps As Integer = Tables("国家").FindRow(dr)
If ps >= 0 Then
    Tables("国家").Position = ps
    Forms("国家信息").Open()
    vr.Retrieve() '通知ListView,重新生成此行
End If
```

可以看出，在虚拟模式下，增加和修改按钮的代码变得简单了，只需简单地通知ListView，已经增加了一行或某一行需要重新生成，当你需要对ListView的显示做更多复杂设置时，你会觉得这种特性尤为可贵。

4.12.1.19 同时使用虚拟模式和分页

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例十九”的窗口。

需要显示大量数据时，我们有虚拟模式和分页显示两种解决方案，不过如果你愿意，你可以同时使用这两种方案，毕竟二者各有优势。

本节的任务是设计一个下图所示的窗口，同时使用虚拟模式和分页：

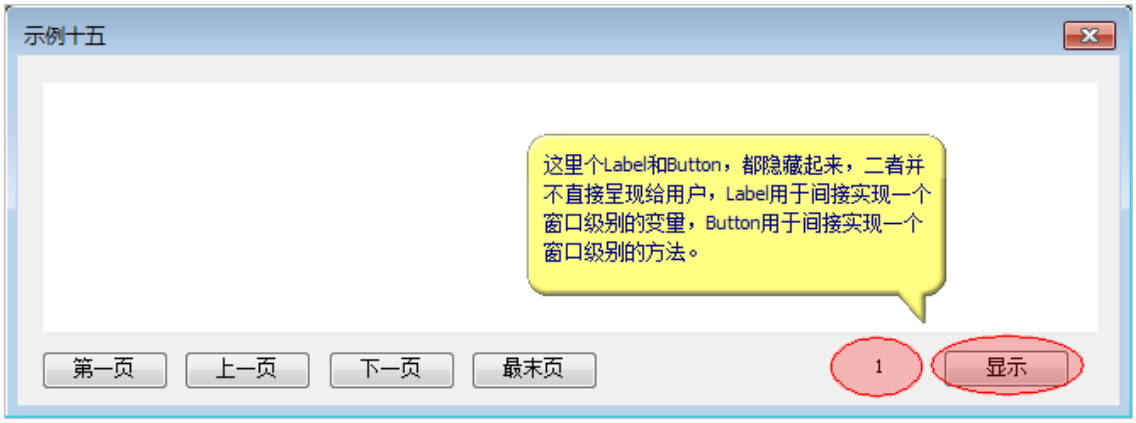


设计步骤：

1、在窗口中插入一个 ListView、5 个 Button，1 个 Label，

2、将 Label 的文本设置为 1，可见属性设置为 False，由于 Foxtable 并没有提供窗口级别的变量，我们利用这个不可见的 Label 保存当面页号，方便在本窗口的各事件中调用。

3、将"显示"按钮的 Visible 属性设置为 False，由于 Foxtable 并没有提供窗口级别的方法，我们利用这个不可见的 Button 间接实现一个窗口界别的方法，方便在本窗口的各事件中调用。



4、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.Details  '显示模式为详细内容
lvw.GridLines = True '显示网格线
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {120,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
lvw.VirtualMode = True '使用虚拟模式
lvw.ResumeRedraw() '恢复绘制
e.Form.Controls("btnShow").PerformClick '模拟单击显示按钮,显示第一页数据
```

5、设置各按钮的事件：

按钮	代码
显示	Dim lvw As WinForm.ListView = e.Form.Controls("ListView1") Dim pg As Integer = e.Form.Controls("Label1").Text Dim idx1 As Integer = (pg - 1) * 5 '获取要显示的第一行编号 Dim idx2 As Integer = pg * 5 - 1 '获取要显示的最末行编号 idx2 = math.Min(DataTables("国家").DataRows.Count - 1, idx2) '最末行的编号不能超过DataTable的实际行数 lvw.Images.Clear() '清除之前的图片 lvw.VirtualListSize = idx2 - idx1 + 1 '设置虚拟模式下的行数等于这一页的行数 lvw.RetrieveAll() '重新生成所有行.
第一页	Dim lbl As WinForm.Label = e.Form.Controls("Label1") Dim btn As WinForm.Button = e.Form.Controls("btnShow")

	Dim pg As Integer = lbl.Text '当前页号
	If pg <> 1 Then
	  lbl.Text = 1
	End If
	btn.PerformClick() '模拟单击显示按钮

上一页	Dim lbl As WinForm.Label = e.Form.Controls("Label1")
	Dim btn As WinForm.Button = e.Form.Controls("btnShow")
	Dim pg As Integer = lbl.Text '当前页号
	If pg > 1 Then
	  lbl.Text = pg - 1
	End If
	btn.PerformClick() '模拟单击显示按钮

下一页	Dim lbl As WinForm.Label = e.Form.Controls("Label1")
	Dim btn As WinForm.Button = e.Form.Controls("btnShow")
	Dim pg As Integer = lbl.Text '当前页号
	Dim pgs As Integer = math.Ceiling(DataTables("国家").DataRows.Count / 5) '计算出总页数
	If pg < pgs Then
	  lbl.Text = pg + 1
	End If
	btn.PerformClick() '模拟单击显示按钮

最末页	Dim lbl As WinForm.Label = e.Form.Controls("Label1")
	Dim btn As WinForm.Button = e.Form.Controls("btnShow")
	Dim pg As Integer = lbl.Text '当前页号
	Dim pgs As Integer = math.Ceiling(DataTables("国家").DataRows.Count / 5) '计算出总页数
	If pg < pgs Then
	  lbl.Text = pgs
	End If
	btn.PerformClick() '模拟单击显示按钮

6、将 ListView 的 RetrieveVirtualRow 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Sender
Dim pg As Integer = e.Form.Controls("Label1").Text
Dim idx As Integer = (pg - 1) * 5 '获取本页第一个DataRow的编号
idx = idx + e.Index '获取本次要调用的DataRow的编号
Dim dr As DataRow = DataTables("国家").DataRows(idx) '根据行位置从数据表中取得对应的DataRow
Dim Key As String = dr("图标")
lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的图标
e.Row.Imagekey = key '指定图标键值
For Each cl As WinForm.ListViewColumn In lvw.Columns '逐列取值
    Select Case cl.Name
        Case "人口","面积" '如果是人口列或面积列
```

```
            e.Row(cl.Name) = Format(dr(cl.Name),"#,000") '则显示千位分割符号
        Case Else
            e.Row(cl.Name) = dr(cl.Name)
    End Select
Next
```

提示：

上面的例子用 ListView 通过虚拟模式和分页来显示 DataTable 中的数据，有的时候，DataTable 自己就是分页加载的，此时 ListView 无需考虑分页的问题，只需用虚拟显示 DataTable 的所有数据即可，所以更加简单，如果你有疑惑，可以参考名为“示例二十”的窗口。

4.12.1.20 行也有 Name 属性的哦

多数时候，ListView 的数据来自 DataTable，前面我们已经看到，在生成 ListViewRow 的时候，可以将其 Tag 属性设置为对应的 DataRow，这样在 ListView 的事件中，我们可以轻松获得某个 ListViewRow 对应的 DataRow，但是反过来，要知道某个 DataRow 对应的 ListViewRow，则不太容易，往往遍历所有的 ListViewRow，逐行比较才行。
但是行有 Name 属性，我们可以在生成 ListViewRow 的时候，将 DataRow 的主键值复制给行的 Name 属性，例如：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.View = ViewMode.Details
Dim cls() As String = {"编号","姓名","部门","职务","性别","地址"} '定义列名
For i As Integer = 0 To  cls.Length - 1  '增加列
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Text = cls(i) '指定列标题
    c.Name = cls(i) '指定列名
Next
For Each dr As DataRow In DataTables("员工").DataRows '从数据表中提取数据
    Dim vr As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    For Each cl As String In cls '逐列取值
        vr(cl) = dr(cl)
    Next
    vr.Tag = dr '将DataRow赋值给ListViewRow的Tag属性
    vr.Name = dr("_Identify") ''将DataRow的主键赋值给ListViewRow的Name属性
Next
lvw.ResumeRedraw() '恢复绘制
```

这样我们随时获取 DataRow 对应的 ListViewRow，例如：

```
Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
Dim id As String =  Tables("员工").Current("_Identify")
Dim vr As WinForm.ListViewRow = lv.Rows(id)
```

```
vr.Delete()
```

一些用户可能会这样编写代码：

```
Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
Dim vr As WinForm.ListViewRow = lv.Rows(Tables("员工").Current("_Identify"))
```

这样是不行的，因为_Identify 等主键列通常是整数型，假定主键列的值是 10，那么上面的代码得到的是第 11 行（编号从 0 开始），而不是名称为“10”的行，这种错误不易发现，我自己就经常犯，大家务必要记住：需要根据行的名称获取行时，传递过去的参数必须是字符型，而不是整数型，如果主键是整数型，必须先将主键值转换为字符型再传递：

```
Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
Dim id As String =  Tables("员工").Current("_Identify")
Dim vr As WinForm.ListViewRow = lv.Rows(id)
```

4.12.1.21 虚拟模式下的行与遍历

上一节讲到，行也有 Name 属性，这样我们可以快速获取指定名称的行。

不过在虚拟模式下，要获取指定名称的行，可能付出一些代价。

以上一节的代码为例：

```
Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
Dim id As String =  Tables("员工").Current("_Identify")
Dim vr As WinForm.ListViewRow = lv.Rows(id)
vr.Delete()
```

假定由于虚拟模式下，行都是需要的时候生成，上面的代码首先会在已经生成的行中查找指定名称的行，如果找到则返回，如果没有找到，则执行 RetrieveVirtualRow 事件，往下生成一个新的行，如果新行的名称等于指定名称，则返回此新行，否则继续生成，继续比较，直到生成一个名称相符的行，或者所有行都生成完毕。

所以上面简单的一行代码，可能会导致 ListView 要生成所有的行，如果 ListView 有几万行，可能会消耗相当长的时间。

所以在虚拟模式下，最好不要通过行的名称来引用行，而是通过行的位置，例如：

```
Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
Dim vr As WinForm.ListViewRow = lv.Rows(20)
```

但是麻烦来了，如果要获取某个DataRow对应的ListViewRow，那么如何知道DataRow对应的ListViewRow的位置呢？ 这个只能用笨办法了，例如：

```
Dim dr As DataRow = Tables("国家").Current.DataRow
Dim idx As Integer = -1
For i As Integer = 0 To DataTables("国家").DataRows.Count - 1
    If DataTables("国家").DataRows(i)("_Identify") = dr("_Identify") Then
        idx = i
        Exit For
    End If
```

```
Next
If idx >= 0 Then
    Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
    Dim vr As WinForm.ListViewRow = lv.Rows(idx)
    vr.Delete()
End If
```

此外，在虚拟模式下，我们应该尽量避免遍历所有行，例如下面的代码：

```
Dim lv As WinForm.ListView = e.Form.Controls("ListView1")
For Each vr As WinForm.ListViewRow In lv.Rows
    '代码
Next
```

这段代码执行过程中，会生成所有的行，因为只有这样，才能完成遍历。

当然，如果虽然用了虚拟模式，但是一次显示的行数并不多，那么可以正常使用行名称来引用行，正常遍历行，无需为感受不到的性能差别花费太多精力。

上面就是虚拟模式的一些不足，但是和其带来的好处相比，这一点不足并不算什么，而且ListView和DataTable的互动，多数是从ListView到DataTale，从DataTable到ListView的互动极少，

4.12.1.22 轻松实现拖动行动能

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例二十一”的窗口。
ListView 中的行是拖动的，行可以在 ListView 内部拖动，以可视化的方式调整行的位置，也可以将行从一个 ListView 拖动到另一个 ListView，只要两个 ListView 的结构相同。

ListView 有两个和拖动相关的属性，分别是：

- AllowDrag
  是否允许拖动行，默认为 False

- AllowDrop
  是否 允许接收拖动过来的行，默认为 False

你如果要实现普通的拖动行功能，只需将这两个属性设置为 True 即可，是的，就是这么简单。
下面我们来做个练习，任务是实现下图所示的功能，ListView 中行可以通过拖动调整位置，且可以将调整后的顺序保存在 DataTable 中：



设计步骤：

1、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.Details '显示模式为详细信息
lvw.GridLines = True '显示网格线
lvw.AllowDrag = True '允许拖动行
lvw.AllowDrop = True '允许  接收拖动过来的行
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {100,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
For Each dr As DataRow In DataTables("国家").Select("", "_SortKey") '从数据表中提取数据
    Dim Key As String = dr("图标")
    Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    lvw.Images.AddImage(Key, Key & ".ico", Key & "48.ico") '添加代表这个国家的一对图标
    r.ImageKey = Key '指定图标键值
    r.tag = dr '将DataRow保存在行的
    For Each cl As String In cls '逐列取值
      Select Case cl
        Case "人口","面积" '如果是人口列或面积列
          r(cl) = format(dr(cl),"#,000") '则显示千位分割符号
```

```
28            Case Else
29                r(cl) = dr(cl)
30        End Select
31    Next
32 Next
33 lvw.ResumeRedraw() '恢复绘制
```

上面的代码中，和拖动相关的代码就是第5和第6行，另外注意第18行的，为了体现出“顺序”，我们根据_SortKey列的值的顺序来添加ListViewRow，并在第23行代码将对应的DataRow赋值给ListViewRow的Tag属性，方便之后调用。

2、将“保存顺序”按钮的代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
DataTables("国家").StopRedraw
For Each vr As WinForm.ListViewRow In lvw.Rows
    Dim dr As DataRow = vr.tag
    dr.BaseRow("_SortKey") = vr.Index
Next
DataTables("国家").ResumeRedraw
DataTables("国家").Save()
```

提示：

1、这个例子用_SortKey列作为顺序列，你也可以用其它列，只要这一列的值能代表行的顺序即可。
2、对于外部表，_SortKey列默认是没有的，需要启用插入行功能，才会增加这一列。
3、如果你足够细心，你也许已经发现我们是这样更新_SortKey这一列的值的：

```
dr.BaseRow("_SortKey") = vr.Index
```

这是因为_SortKey是系统列，不能直接更改它的值，必须通过BaseRow才能更改。

4.12.1.23 精确控制行的拖动行为

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例二十二”的窗口。
ListView 有个名为“BeforeDropRow”事件，在准备接收拖动过来的行之前触发，这个事件很重要，可以对拖动行为进行精确控制。
BeforeDropRow 的 e 参数有：
Form：　返回触发事件的窗口
Sender：　返回触发事件的控件，也就是准备接收行的 ListView
Source：　返回拖动过来的行原来所属的 ListView，如果在同一个 ListView 之间拖动，那么 Source 就等于 Sender
Add：　逻辑型，是否接收拖动过来的行，默认为 True，只有在不同的 ListView 之间拖动时，此

参数有意义。

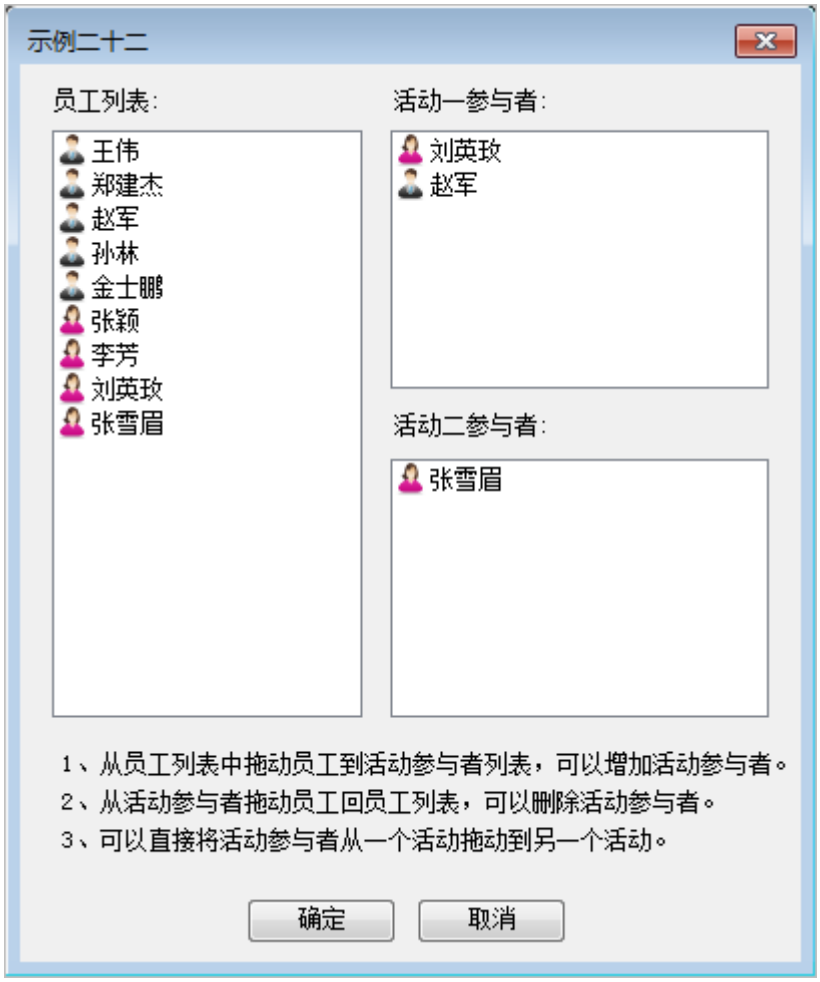
Delete： 逻辑型，是否从原来的 ListView 删除拖动过来的行，默认为 True，只有在不同的 ListView 之间拖动时，此参数有意义。

OldIndex： 拖动过来的行的原位置。

NewIndex： 新位置，将在此处接收拖动过来的行。

Cancel： 逻辑型，是否取消本次拖动，当然默认为 False。

下来用们用一个例子来演示这个事件的用法，任务是设计一个下图所示的窗口：



通过这个窗口，可以以可视化的形式选择或删除活动参与者，也可以活动参与者从一个活动拖到另一个活动，单击保存按钮后，将结果保存在下表中：

	活动	参与者
1	活动一	刘英玫, 赵军
2	活动二	张雪眉

设计步骤：

1、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw1 As WinForm.ListView = e.Form.Controls("ListView1") '员工列表
Dim lvw2 As WinForm.ListView = e.Form.Controls("ListView2") '活动一参与者列表
Dim lvw3 As WinForm.ListView = e.Form.Controls("ListView3") '活动二参与者列表
Dim lvws() As Winform.ListView = {lvw1,lvw2,lvw3} '看看我们是怎样批量设置某组控件属性的
For Each lvw As WinForm.ListView In lvws
    lvw.View = ViewMode.List
```

```
    lvw.AllowDrag = True
    lvw.AllowDrop = True
    lvw.Images.AddSmallImage("Man", "Man.ico") '添加代表男性的一对图标
    lvw.Images.AddSmallImage("Woman", "Woman.ico") '添加代表女性的一对图标
Next
For Each dr As DataRow In DataTables("员工").Select("","性别") '列出所有员工
    Dim vr As WinForm.ListViewRow = lvw1.Rows.Add()
    vr.Text = dr("姓名")
    vr.Name = dr("姓名") '将ListViewRow的Name属性设置为员工姓名
    If dr("性别") = "男" Then
        vr.ImageKey = "Man"
    Else
        vr.ImageKey = "Woman"
    End If
Next
Dim nms() As String = DataTables("活动").DataRows(0)("参与者").Split(",") '列出活动一参与者
For Each nm As String In nms
    If nm > "" Then
        Dim vr As WinForm.ListViewRow = lvw2.Rows.Add()
        vr.Text = nm
        vr.Name = nm
        vr.ImageKey = lvw1.Rows(nm).ImageKey  '从员工列表(ListView1)获取此员工的图标键值
    End If
Next
nms = DataTables("活动").DataRows(1)("参与者").Split(",") '列出活动二参与者
For Each nm As String In nms
    If nm > "" Then
        Dim vr As WinForm.ListViewRow = lvw3.Rows.Add()
        vr.Text = nm
        vr.Name = nm
        vr.ImageKey = lvw1.Rows(nm).ImageKey '从员工列表(ListView1)获取此员工的图标键值
    End If
Next
```

上面的代码中，我首先请你留意第 4 行到第 11 行的代码，当我们需要对一组控件进行相同的设置时，可以用这种方法减少代码量；其次留意第 15 行代码，此行代码将员工列表(ListView1)中的行的 Name 属性设置为对应员工的姓名，这样我们可以在 28、37 这两行代码通过员工姓名获取 ListView1 中这个员工对应的行，然后获取此行的 ImageKey 属性，并赋值给活动参与者(ListView2 和 ListView3)中对应行的 ImageKey 属性。

2、ListView2(活动一参与者)的 BeforeDropRow 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Sender
If lvw.Rows.Count >= 3  AndAlso e.Source.Name <> lvw.Name Then '如果已经有三人,而且是从其他ListView
拖动过来.
        Messagebox.show("每项活动最多允许三人参与!","提示
",MessageBoxButtons.OK,MessageBoxIcon.Information)
```

```
6     e.Cancel = True '取消本次拖动
7     Return
8  End If
9  If e.Source.Name =  "ListView1" '如果是从员工列表中拖过来的行
10    e.Delete = False '不要删除原来的项目
11 End If
12 Dim itm As WinForm.ListViewRow =  e.Source.Rows(e.OldIndex) '获取要拖动过来的行
13 If e.Source.Name <> e.Sender.Name Then '如果是从其他ListView中拖动过来的
14    If e.Sender.Rows.Contains(itm.Name) '判断活动一是否已经包括此员工
15       MessageBox.Show("此员工已经报名参加活动一，无需重复报名!","提示
16 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
         e.Cancel = True '取消本次拖动
      End If
   End If
```

首先留意第7行代码，意思是如果是从员工列表拖动过来的，那么就不需要删除原来的行；这里隐藏了另一个意思： 如果是从活动二参与者拖动过来的行，那么就原来的行就删除；也就是说，从员工列表拖动员工到活动参与者，是复制的动作，而从一个活动参与者列表拖动员工到另一个活动参与者，是移动的动作；这个看似复杂的判断，就这么三行代码搞定了。
其次请留意第11行的代码是如何判断某员工已经参与此项活动的，这就是行有Name属性的好处，当然你可以遍历判断，但是要多写几行代码。

3、ListView3(活动而参与者)的BeforeDropRow事件代码和ListView2完全相同，就不重复说明了。

4、最后将ListView1(员工列表)的BeforeDropRow事件设置为：

```
e.Add = False
```

这简单的一行代码的意思是：从活动参与者拖动员工回到员工列表时，会将此员工从活动参与者列表中删除，但不会在员工列表中再增加此员工，是一种直接删除的动作。

到这里我们总结一些，这一个例子用拖动实现了三种操作：

1、复制，从员工列表拖动员工到活动参与者列表
2、删除，从活动参与者列表拖动员工回员工列表
3、移动，从一个活动参与者列表拖动员工到另一个活动参与者列表

此外还进行了一些条件判断，例如每项活动最多允许3人参加，防止重复报名等。

4.12.1.24 简单的重命名功能

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例二十 三”的窗口。
ListView 提供了编辑功能，不过很简单，只能编辑标题，一般仅仅用于实现类似“重命名”这样的功

能。

ListView 默认是不允许编辑的，必须将 AllowEdit 属性设置为 True 后，才可以编辑。
如果如果 AllowEdit 属性为 True，有两种方法进入编辑状态：

1、先单击某行，选定此行，然后再单击此行
2、或者执行行的 BeginEdit 方法。
ListView 提供了两个和编辑事件，分别是 BeforeEdit 和 AfterEdit，顾名思义，前者在编辑前触发，后者在编辑后触发，二者的 e 参数相同，分别为：
Sender: 触发 Form 的控件
Form： 触发事件的窗口
Row： 正在编辑的行
Index： 正在编辑的行的位置
NewText：编辑后的标题内容
Cancel： 逻辑型，是否取消本次编辑

下面设计一个下图所示的窗口，列出所有员工，并提供更改员工姓名的功能。



设计步骤：
1、窗口的 AfterLoad 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.AllowEdit = True '允许编辑标题
lvw.View = ViewMode.LargeIcon '显示模式为大图标
lvw.Images.AddLargeImage("Man", "Man48.ico") '添加代表男性的一对图标
lvw.Images.AddLargeImage("Woman", "Woman48.ico") '添加代表女性的一对图标
For Each dr As DataRow In DataTables("员工").DataRows '从数据表中提取数据
    Dim vr As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
```

```
    vr.Text = dr("姓名")  '设置标题
    If dr("性别") = "男" Then '设置分组
        vr.ImageKey = "Man"
    Else
        vr.ImageKey = "Woman"
    End If
    vr.Tag = dr '将DataRow赋值给ListViewRow的Tag属性,方便以后调用.
Next
lvw.ResumeRedraw() '恢复绘制
```

2、ListView的BeforeEdit事件代码设置为：

```
Dim dr As DataRow = e.Row.tag
If dr.Locked Then '如果对应的DataRow已经锁定
    e.Cancel = True '取消本次重命名操作
      MessageBox.Show("此员工数据已经锁定，不能重命名!", "提示",
MessageBoxButtons.OK ,MessageBoxIcon.Information)
End If
```

3、ListView的AfterEdit事件代码设置为：

```
Dim dr As DataRow = e.Row.tag
If e.NewText  =  "" Then
        Messagebox.Show("员工姓名不能为空!","提示
",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True '取消重命名
Else
    dr("姓名") = e.NewText '修改对应的DataRow的姓名列的值
End If
```

4、重命名按钮的代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
If lvw.Current IsNot Nothing Then
    lvw.Current.BeginEdit()
End If
```

提示：

需要注意的时候，如果ListViewRow有多列内容，则只能对第一列进行编辑，因为此时第一列内容就是标题。

4.12.1.25 自行编码绘制单元格

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例二十四”的窗口。

ListView 有个 DrawViewCell 事件，在绘制某个单元格的时候触发，其 e 参数成员有：

Sender: 触发 Form 的控件

Form： 触发事件的窗口

Row： 正在绘制的行

Col： 正在绘制的列

X: 要绘制单元格的水平坐标位置，单位为像素

Y: 要绘制单元格的垂直坐标位置，单位为像素

Width： 要绘制单元格的宽度，单位为像素

Height： 要绘制单元格的高度，单位为像素

Graphics： 返回一个 Graphics 对象，通过这个对象可以在单元格中绘制图形

DrawDefault： 逻辑属性，是否取消系统默认的绘制单元格动作，此属性默认为 True，必须设置为 False 才能编码绘制单元格。

DrawText： 这是一个方法，执行此方法将用默认的方式绘制单元格的文本内容。

DrawBackground: 这是一个方法，执行此方法将用默认的当时绘制单元格背景。

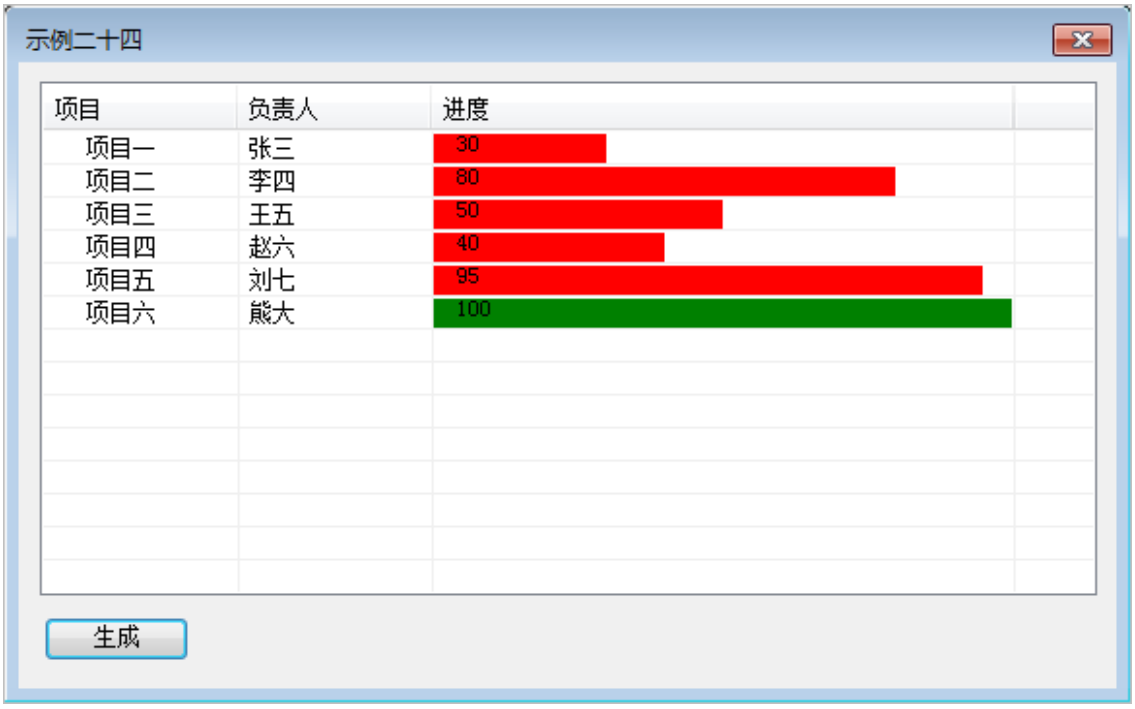
提示：

1、必须将 ListView 的 OwnerDraw 属性设置为 True，才能触发 DrawViewCell 事件。

2、必须将显示模式为 Details(详细信息)，否则 OwnerDraw 属性设置为 True 后，ListView 不会显示任何项目。

示例：

假定用 ListView 显示一个进度表中的数据，在进度单元格希望用图形来表示进度：



设计步骤：

1、首先设置生成按钮的代码，注意加粗的两行代码

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.Columns.Clear() '清除原来的列
lvw.Rows.Clear() '清除原来的行
lvw.Images.Clear() '清除原来的图片
lvw.View = ViewMode.Details '显示模式为详细信息时才能编码绘制单元格
lvw.OwnerDraw = True '必须将OwnerDraw事件设置为True,才会触发DrawViewCell事件
lvw.GridLines = True '显示网格线
Dim cls() As String = {"项目","负责人","进度"} '指定要显示的各列
Dim wds() As Integer = {100,100,300} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
For Each dr As DataRow In DataTables("进度").DataRows '从数据表中提取数据
    Dim r As  WinForm.ListViewRow =  lvw.Rows.Add() '增加一行
    For Each cl As String In cls '逐列取值
        r(cl) = dr(cl)
    Next
Next
lvw.ResumeRedraw() '恢复绘制
```

2、然后设置 DrawViewCell 事件的代码：

```
If e.Col.name = "进度" Then
    e.DrawDefault = False '取消默认的绘制
    Dim Width As Integer = (e.Width - 2 ) * e.Row("进度") \ 100
    If e.Row("进度") = 100 Then
        e.Graphics.FillRectangle(Brushes.Green,e.x + 1,e.y + 1, Width, e.Height - 2)
    Else
        e.Graphics.FillRectangle(Brushes.Red,e.x + 1,e.y + 1, Width, e.Height - 2)
    End If
    e.DrawText()
End If
```

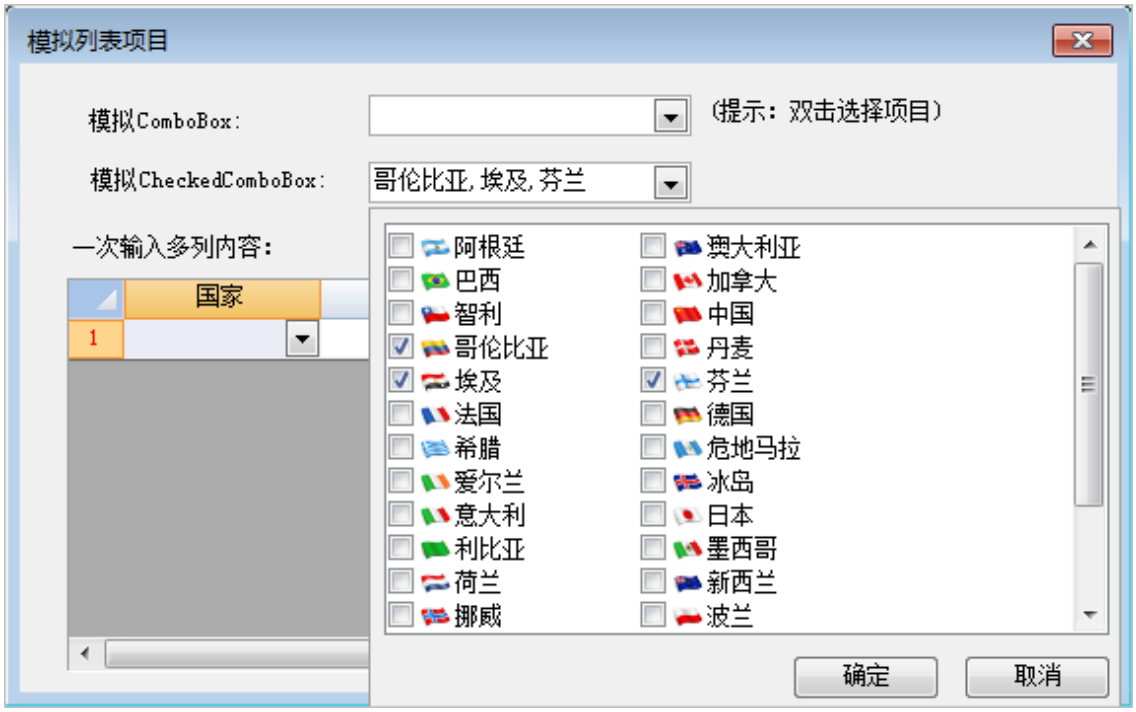
4.12.1.26 用 ListView 模拟下拉列表

本节学习如何用 ListView 模拟下拉列表。

为了便于学习本节的内容，可以先打开 CaseStudy 目录下的文件“ListView.Table”，并参考该文件名为“示例二十 五”的窗口。

这个窗口用 ListView 模拟了 ComboBox、CheckedComboBox，并展示了如何在表中利用 ListView 输入多列数据。

我们只介绍如何模拟 CheckedComboBox，其它内容请参考示例文件自行学习：



设计步骤：

1、新建一个名为“下拉窗口 2”的窗口，窗口类型设置为“DropDownForm”

2、下拉窗口的 AfterLoad 事件代码为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.SmallIcon '显示模式为小图标
lvw.CheckBoxes = True
For Each dr As DataRow In DataTables("国家").DataRows '从数据表中提取数据
    Dim Key As String = dr("图标")
    Dim r As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
    lvw.Images.AddSmallImage(Key, Key & ".ico") '添加代表这个国家的小图标
    r.ImageKey = Key '指定图标键值
    r.Text = dr("国家") '指定行的标题
    r.Name   = r.Text '将行的名称设置为标题,也就是国家
Next
lvw.ResumeRedraw() '恢复绘制
```

注意上面的代码， 不仅将国家作为行的标题，也将国家作为行的名称。

3、下拉窗口的 DropdownOpening 事件代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim ss As String = e.Form.DropDownBox.Value
For Each r As WinForm.ListViewRow In lvw.Rows
    r.Checked = False
Next
```

```
If ss > "" Then
   Dim vals() As String = ss.Split(",")
   For Each val As String In vals
      If lvw.Rows.Contains(val) Then
         lvw.Rows(val).Checked = True
      End If
    Next
End If
```

这样打开下拉窗口后，能自动勾选已经输入的国家，注意这里巧妙通过国家(也就是行的名称)来来快速检索行。

4、确定按钮的代码设置为：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim vls As String = ""
For Each vr As WinForm.ListViewRow In lvw.CheckedRows
   vls = vls & "," & vr.Text
Next
e.Form.DropDownBox.Value = vls.Trim(",")
e.Form.DropDownBox.CloseDropdown()
```

5、最后回到数据录入窗口，将对应的DropDownBox的下拉窗口属性设置为“下拉窗口2”

4.12.1.27 无需本末倒置

ListView 丰富了我们展示数据的方式，但是 Foxtable 的核心是 DataTable 和 Table，在开发系统的时候，展示数据主要方式应该是 Table，而不是 ListView。

4.12.2 类型参考

4.12.2.1 ListView

4.12.2.1.1 属性

4.12.2.1.1.1 View

设置 ListView 的显示模式，ViewMode 类型，可选值有：
Details： 详细内容
LargeIcon： 大图标
SmallIcon： 小图标

List：　列表
Tile：　平铺

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.View = ViewMode.LargeIcon
```

4.12.2.1.1.2 GridLines

设置 GridLines 否显示网格线，逻辑型，只有显示模式为详细内容的时候，此属性才有效。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.View = ViewMode.Details '设置显示模式为详细内容
lvw.GridLines = True '显示网格线
```

4.12.2.1.1.3 ShowGroups

逻辑型，是否显示分组，默认为 True。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.ShowGroups = False
```

4.12.2.1.1.4 MultiSelect

是否允许 ListView 选择多行，逻辑型，默认为 True。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.MultiSelect = False
```

4.12.2.1.1.5 CheckBoxes

逻辑型，是否允许 ListView 显示复选框。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.CheckBoxes = True
```

4.12.2.1.1.6 VirtualMode

逻辑型，设置 ListView 是否使用虚拟模式，默认为 False，需要和 VirtualListSize 属性配合使用。

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.VirtualMode = True '使用虚拟模式
lvw.VirtualListSize = 50 '虚拟模式下有 50 行数据要显示
```

4.12.2.1.1.7 VirtualListSize

设置 ListView 在虚拟模式下的行数。
示例：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.VirtualListSize = 50
```

4.12.2.1.1.8 Current

返回ListView的当前行。

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
If lvw.Current IsNot Nothing Then
        Dim vr As WinForm.ListViewRow = lvw.Current
        Messagebox.show("你选择的国家是" & vr.Text & ",面积为" & vr("面积"),"提示
",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

4.12.2.1.1.9 Position

返回 ListView 当前行的位置。
示例：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
If lvw.Position >= 0  Then
   MessageBox.show("你选择的是第" & lvw.Position & "行！合计有" & lvw.Rows.Count & "行!","提示
",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

4.12.2.1.1.10 AllowDrag

逻辑型，是否允许在 ListView 拖动行。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.AllowDrag = True '允许拖动行
lvw.AllowDrop = True '允许接收拖动过来的行
```

4.12.2.1.1.11 AllowDrop

逻辑型，是否允许在 ListView 接收拖动过来的行。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.AllowDrag = True '允许拖动行
lvw.AllowDrop = True '允许接收拖动过来的行
```

有一个需要特别注意的问题，当需要向 ListView 大量添加行之前，不能将 AllowDrop 属性设置为 True，可以在添加行之后，再将 AllowDrop 属性设置为 True，否则会大大影响添加行的效率；有兴趣的话，分别比较以下两段代码的效率。

1、AllowDrop 设置为 True 后，再添加行，效率低：

```
Dim dt As Date = Date.Now()
Dim lvw As WinForm.ListView = e.Form.Controls("ListView2")
lvw.View = ViewMode.SmallIcon
lvw.AllowDrop = True
lvw.AllowDrag = True
lvw.StopRedraw
For i As Integer = 1 To 1000
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Name = i
    vr.Text = "项目"  & i
Next
lvw.ResumeRedraw
```

2、先添加行，再将AllowDrop属性设置为True，效率高：

```
Dim dt As Date = Date.Now()
Dim lvw As WinForm.ListView = e.Form.Controls("ListView2")
lvw.View = ViewMode.SmallIcon
lvw.StopRedraw
For i As Integer = 1 To 1000
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
```

```
    vr.Name = i
    vr.Text = "项目"  & i
Next
lvw.AllowDrop = True
lvw.AllowDrag = True
lvw.ResumeRedraw
```

当然有大量数据需要显示的时候，还是建议用虚拟模式，比较快捷。

4.12.2.1.1.12 AllowEdit

逻辑型，是否允许 ListView 编辑标题。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.AllowEdit = True
```

提示： 如果是多列，此属性决定是否允许编辑第一列内容，因为此时第一列内容就是行的标题。

4.12.2.1.1.13 TitleSize

设置ListView平铺模式下，每行显示区域的大小。

示例：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.View = ViewMode.Tile
lvw.TitleSize = New Size(150,90)
```

4.12.2.1.1.14 Groups

返回一个集合，包括 ListView 的所有分组。
通过此集合，可以给 ListView 增加、删除分组，也可以获取指定名称的分组。
Groups 集合的成员有：

名称	说明
Count	属性，返回分组的数量
Add	方法，增加分组
Delete	方法，删除指定名称的分组
Contains	方法，判断是否存在指定名称的分组
Clear	方法，清除所有分组

示例一

增加分组：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Groups.Add("分组1")
lvw.Groups.Add("分组2")
For i As Integer = 0 To 9
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.text = "项目" & i
    vr.Group = iif( i <  5, "分组1" , "分组2")
Next
```

示例二

删除分组：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Groups.Delete("分组2")
```

示例三

清除所有分组：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Groups.Clear()
```

4.12.2.1.1.15 Columns

返回一个集合，包括 ListView 的所有列。

通过此集合，可以增加、删除列，也可以获取指定名称的列。

需要注意的是，一旦添加了行，就不可以再增加或删除列了。

Columns 集合的成员有：

名称	说明
Count	属性，返回列的数量
Add	方法，增加列
Delete	方法，删除指定名称的列
Contains	方法，判断是否存在指定名称的列
Clear	方法，清除所有列

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '定义列名
Dim wds() As Integer = {100,100,150,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
```

```
    c.Name = cls(i) '指定列名
    c.Text = cls(i) '指定标题,这里标题和列名相同
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("国家").TextAlign = HorizontalAlignment.Center
```

4.12.2.1.1.16 Rows

返回一个集合，包括 ListView 中的所有行。
通过此集合，我们可以增加、删除行，或获取指定位置或名称的行。
Rows 集合的成员有

名称	说明
Count	属性，返回行的数量
Add	方法，增加行
Delete	方法，删除指定位置或名称的行
Contains	方法，判断是否存在指定名称的行
Clear	方法，清除所有行

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张三","李四","王五","赵六"}
For Each nm As String In nms
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm
    vr.Name = nm
Next
If lvw.Rows.Contains("王五")  Then '判断是否存在名为王五的行
    lvw.Rows.Delete("王五") '删除名为"王五"的行
End If
lvw.Rows.Delete(0) '删除第一行
```

4.12.2.1.1.17 SelectedRows

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim vls As String = ""
For Each vr As WinForm.ListViewRow In lvw.SelectedRows
    vls = vls & "," & vr("国家")
Next
vls = vls.Trim(",")
If vls > "" Then
    MessageBox.show("你选择的国家有:" & vls)
```

```
Else
    MessageBox.show("你并没有选择任何国家!" )
End If
```

4.12.2.1.1.18 CheckedRows

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim vls As String = ""
For Each vr As WinForm.ListViewRow In lvw.CheckedRows
    vls = vls & "," & vr("国家")
Next
vls = vls.Trim(",")
If vls > "" Then
    MessageBox.show("你选择的国家有:" & vls)
Else
    MessageBox.show("你并没有选择任何国家!" )
End If
```

4.12.2.1.1.19 OwnerDraw

逻辑型，默认为 False，设置为 True 触发后会 DrawViewCell 事件。
参考：编码绘制单元格

4.12.2.1.2 方法

4.12.2.1.2.1 RetrieveAll

ListView 处于虚拟模式下时，执行此方法可以重新生成所有行。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.RetrieveAll()
```

4.12.2.1.2.2 Reset

重置 ListView，清除所有的分组、列、行和图片。
示例：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Reset()
```

4.12.2.1.2.3 StopRedraw

停止绘制 ListView，通常在大量添加行的时候先执行 StopRedraw，添加结束后再执行 ResueRedraw 恢复绘制，以提高效率。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.View = ViewMode.SmallIcon
lvw.StopRedraw
For i As Integer = 1 To 1000
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Name = i
    vr.Text = "项目"  & i
Next
lvw.ResumeRedraw
```

4.12.2.1.2.4 ResumeRedraw

恢复绘制 ListView，参考 StopRedraw

4.12.2.1.3 事件

4.12.2.1.3.1 RowActivate

选定某行后按回车键，或者双击某行后触发。
e 参数属性:
Sender：返回触发事件的控件
Form: 返回控件所在的窗口
提示：通过 ListView 的 Current 属性，可以获取双击的行。

4.12.2.1.3.2 RetrieveVirtualRow

在虚拟模式下生成某行时触发。
e 参数属性：

Sender：返回触发事件的控件
Form: 返回控件所在的窗口
Row： 返回要生成的行
Index： 返回此行的位置
参考： 用虚拟模式显示大量数据

4.12.2.1.3.3 BeforeDragRow

开始拖动某行前触发。
e 参数属性：
Sender：返回触发事件的控件
Form: 返回控件所在的窗口
Row： 返回要拖动的行
Index： 返回此行的位置
Cancel：逻辑型，设置为 True，取消此次拖动。

4.12.2.1.3.4 BeforeDropRow

Form： 返回触发事件的窗口
Sender： 返回触发事件的控件，也就是准备接收行的 ListView
Source： 返回拖动过来的行原来所属的 ListView，如果在同一个 ListView 之间拖动，那么 Source 就等于 Sender
Add： 逻辑型，是否接收拖动过来的行，默认为 True，只有在不同的 ListView 之间拖动时，此参数有意义。
Delete： 逻辑型，是否从原来的 ListView 删除拖动过来的行，默认为 True，只有在不同的 ListView 之间拖动时，此参数有意义。
OldIndex： 拖动过来的行的原位置。
NewIndex： 新位置，将在此处接收拖动过来的行。
Cancel： 逻辑型，是否取消本次拖动，当然默认为 False。

4.12.2.1.3.5 RowSelectionChanged

某行的选中状态发生变化后执行。
e 参数属性：

Sender：返回触发事件的控件
Form: 返回控件所在的窗口
Row： 返回选中状态发生变化的行
Index： 返回此行的位置

4.12.2.1.3.6 BeforeEdit

编辑行的标题前执行。
e 参数属性：
Sender: 触发 Form 的控件
Form： 触发事件的窗口
Row： 正在编辑的行
Index： 正在编辑的行的位置
NewText：编辑后的标题内容
Cancel： 逻辑型，是否取消本次编辑

参考：简单的重命名功能

4.12.2.1.3.7 AfterEdit

编辑行的标题后执行。
e 参数属性：
Sender: 触发 Form 的控件
Form： 触发事件的窗口
Row： 正在编辑的行
Index： 正在编辑的行的位置
NewText：编辑后的标题内容
Cancel： 逻辑型，是否取消本次编辑
参考：简单的重命名功能

4.12.2.1.3.8 RowChecking

某行的复选框勾选状态发生变化前执行。
e 参数属性：

Sender：返回触发事件的控件
Form: 返回控件所在的窗口
Row： 返回选中状态发生变化的行
Index： 返回此行的位置
Cancel：逻辑型，设为 True，取消本次操作。

4.12.2.1.3.9 RowChecked

某行的复选框勾选状态发生变化后执行。
e 参数属性：
Sender：返回触发事件的控件
Form: 返回控件所在的窗口
Row： 返回选中状态发生变化的行
Index： 返回此行的位置

4.12.2.1.3.10 DrawViewCell

在绘制某个单元格的时候触发。
e 参数成员：
Sender: 触发 Form 的控件
Form： 触发事件的窗口
Row： 正在绘制的行
Col： 正在绘制的列
X: 要绘制单元格的水平坐标位置，单位为像素
Y: 要绘制单元格的垂直坐标位置，单位为像素

Width：　　要绘制单元格的宽度，单位为像素
Height：　　要绘制单元格的高度，单位为像素
Graphics：　　返回一个 Graphics 对象，通过这个对象可以在单元格中绘制图形
DrawDefault：　　逻辑属性，是否取消系统默认的绘制单元格动作，此属性默认为 True，必须设置为 False 才能编码绘制单元格。
DrawText：　　这是一个方法，执行此方法将用默认的方式绘制单元格的文本内容。
DrawBackground:　这是一个方法，执行此方法将用默认的当时绘制单元格背景。
参考：编码绘制单元格

4.12.2.2 ListViewGroup

ListViewGroup 表示 ListView 的分组。
通过 ListView 的 Groups 属性，可以增加、删除分组，或者获取指定名称的分组。
ListViewGroup 的属性有：

- Name
  字符型，返回或设置分组的名称

- Text
  字符型，返回或设置分组的标题

- Rows
  返回一个集合，包括此分组下的所有行。

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.CheckBoxes = True '显示复选框
Dim nms() As String = {"分组1","分组2","分组3"}
For Each nm As String In nms '添加分组
    Dim grp As WinForm.ListViewGroup = lvw.Groups.Add()
    grp.Name = nm
    grp.Text = nm
Next
For i As Integer = 0 To 9 '添加行
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.text = "项目" & i
    vr.Group = nms( i Mod 3) '指定行所属的分组
Next
For Each vr As WinForm.ListViewRow In lvw.Groups("分组1").Rows '勾选分组1中的所有行
    vr.Checked = True
Next
```

4.12.2.3 ListViewColumn

ListViewColumn 表示 ListView 的列。
通过 ListView 的 Columns 属性，可以增加、删除列，或者获取指定名称的列。
ListViewColumn 的成员有：

名称	说明
Name	属性，返回或设置列的名称
Text	属性，返回或设置列的标题
Width	属性，返回或设置列的宽度
Index	属性，返回列的位置
TextAlign	属性，返回或设置列的文本对齐方式，HorizontalAlignment 枚举类型，包括三个可选值： Center： 居中 Left：靠左 Right：靠右
AutoResize	方法，自动调整此列的列宽

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.StopRedraw() '暂停绘制
lvw.View = ViewMode.Details '显示模式为详细信息
lvw.GridLines = True '显示网格线
Dim cls() As String = {"国家","人口","面积","语言","宗教","洲"} '指定要显示的各列
Dim wds() As Integer = {100,100,130,80,80,80} '定义列宽
For i As Integer = 0 To  cls.Length - 1
    Dim c As WinForm.ListViewColumn = lvw.Columns.Add()
    c.Name = cls(i) '指定列名
    c.Text = cls(i)'指定列标题
    c.Width = wds(i) '指定列宽
Next
lvw.Columns("人口").Text = "人口(万)" '修改人口列的标题
lvw.Columns("人口").TextAlign = HorizontalAlignment.Right '人口列靠右对齐
lvw.Columns("面积").TextAlign = HorizontalAlignment.Right '面积列靠右对齐
```

4.12.2.4 ListViewRow

4.12.2.4.1 ListViewRow

ListViewRow 表示 ListView 的行。
通过 ListView 的 Rows 属性，可以增加、删除行，或者获取指定位置或名称的行。
示例：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张三","李四","王五","赵六"}
For Each nm As String In nms
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm   '设置行标题
    vr.Name = nm   '设置行的名称
Next
If lvw.Rows.Contains("王五")  Then '判断是否存在名为王五的行
    lvw.Rows("王五").Text = "我是王五" '修改该王五的标题
End If
lvw.Rows.Delete(0) '删除第一行
```

4.12.2.4.2 属性

4.12.2.4.2.1 Name

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张三","李四","王五","赵六"}
For Each nm As String In nms
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm   '设置行标题
    vr.Name = nm   '设置行的名称
Next
If lvw.Rows.Contains("王五")  Then '判断是否存在名为王五的行
    Messagebox.Show("王五已经存在")
End If
```

4.12.2.4.2.2 Text

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张三","李四","王五","赵六"}
For Each nm As String In nms
```

```
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm   '设置行标题
    vr.Name = nm   '设置行的名称
Next
lvw.Rows("张三").Text = "张经理" '修改张三这行的标题
```

4.12.2.4.2.3 Tag

返回或设置 ListViewRow 的附加数据，此属性可以存储任何类型的数据，当 ListView 的行来自数据表时，一般将 ListViewRow 的 Tag 属性设置为对应的 DataRow 或 Row。
参考：ListView 和数据表的互动

4.12.2.4.2.4 Group

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms1() As String = {"张飞","关羽","张云","马超","黄忠"}
Dim nms2() As String = {"庞统","徐庶","孔明","程昱","贾栩"}
lvw.Groups.Add("武将") '添加分组
lvw.Groups.Add("谋士") '添加分组
For Each nm As String In nms1
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm   '设置行标题
    vr.Group = "武将"   '设置行的分组
Next
For Each nm As String In nms2
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm   '设置行标题
    vr.Group = "谋士"   '设置行的分组
Next
```

4.12.2.4.2.5 ImageKey

返回或设置行的图标键值。
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Images.AddImage("China","China.Ico","China48.Ico")  '增加代表中国的一对图标,图标键值为"China"
lvw.Images.AddImage("USA","USA.Ico","USA48.Ico") ''增加代表美国的一对图标,图标键值为"USA"

Dim r As   WinForm.ListViewRow =   lvw.Rows.Add() '增加一行
r.Text = "中国" '指定行的标题
r.ImageKey = "China" '指定行的图标键值
```

```
r = lvw.Rows.Add() '再增加一行
r.Text = "美国" '指定行的标题
r.ImageKey = "USA" '指定行的图标键值
```

提示：图标文件应该事先复制到项目文件所在目录的Images或Attachments子目录中 ，否则添加图标的时候，必须明确指定键值。

4.12.2.4.2.6 CellUseRowStyle

逻辑型，设置此 ListViewRow 的各单元格的样式是否统一使用行的样式，默认为 True。
如果需要单独设置此行中某些单元格的样式，必须将此属性设置为 False。
参考：行与单元格的样式

4.12.2.4.2.7 ToolTipText

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张飞","吕布","赵云"}
Dim tps() As String = {"张飞大嗓门","吕布爱貂蝉","赵云有点帅"}
For i As Integer = 0 To nms.Length - 1
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nms(i)
    vr.ToolTipText = tps(i)
Next
```

4.12.2.4.2.8 Selected

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张飞","吕布","赵云"}
For i As Integer = 0 To nms.Length - 1
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nms(i)
    vr.Name = nms(i)
Next
lvw.Rows(1).Selected = True '选中第二行
```

4.12.2.4.2.9 Checked

返回或设置 ListView 的复选框是否选中。
示例
选中所有行的复选框：

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
For Each r As WinForm.ListViewRow In lvw.Rows
```

```
    r.Checked = True
Next
```

4.12.2.4.2.10 Index

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张三","李四","王五","赵六"}
For Each nm As String In nms
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm   '设置行标题
    vr.Name = nm   '设置行的名称
Next
Dim idx As Integer = lvw.Rows("张三").Index
Messagebox.Show("张三位于" & (idx + 1) & "行","提示")
```

4.12.2.4.2.11 BackColor

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms1() As String = {"张飞","关羽","张云","马超","黄忠"}
Dim nms2() As String = {"庞统","徐庶","孔明","程昱","贾栩"}
For Each nm As String In nms1
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm
    vr.BackColor = Color.Red
    vr.ForeColor =  Color.White
Next
For Each nm As String In nms2
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm
    vr.BackColor = Color.Blue
    vr.ForeColor =  Color.White
Next
```

4.12.2.4.2.12 ForeColor

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms1() As String = {"张飞","关羽","张云","马超","黄忠"}
Dim nms2() As String = {"庞统","徐庶","孔明","程昱","贾栩"}
For Each nm As String In nms1
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
```

```
    vr.Text = nm
    vr.BackColor = Color.Red
    vr.ForeColor =  Color.White
Next
For Each nm As String In nms2
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm
    vr.BackColor = Color.Blue
    vr.ForeColor =  Color.White
Next
```

4.12.2.4.2.13 Font

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
Dim nms() As String = {"张飞","关羽","张云","马超","黄忠"}
For Each nm As String In nms
    Dim vr As WinForm.ListViewRow = lvw.Rows.Add()
    vr.Text = nm
    vr.Name = nm
Next
lvw.Rows("黄忠").Font = New Font("宋体",10, FontStyle.Strikeout)
```

4.12.2.4.3 方法

4.12.2.4.3.1 Delete

删除 ListViewRow
示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Rows(0).Delete() '删除第一行
If lvw.Rows.Contains("黄忠") Then '如果存在名为黄忠的行
    lvw.Rows("黄忠").Delete()
End If
```

4.12.2.4.3.2 EnsureVisible

确保ListViewRow在可见范围内。
如果ListViewRow在可见范围之外，执行此方法，会自动滚动ListView，使得执行此方法的ListViewRow回到可见范围内。

示例

```
Dim lvw As WinForm.ListView = e.Form.Controls("ListView1")
lvw.Rows(268).Selected = True '选中第269行
lvw.Rows(268).EnsureVisible()  '使得第 269 行可见
```

4.12.2.4.3.3 Retrieve

在虚拟模式下，针对此 ListViewRow 重新触发 RetrieveVirtualRow 事件，也就是重新生成此行。
参考：虚拟模式和数据表的互动

4.12.2.5 ListViewCell

ListViewCell 表示 ListView 中的单元格。
通过 ListViewRow 的 Cells 属性可以返回指定列名称或位置的单元格。
ListViewCell 的成员有：

名称	说明
Text	属性，返回或设置单元格内容。
BackColor	属性，返回或设置单元格的背景颜色。
ForeColor	属性，返回或设置单元格的字体颜色。
Font	属性，返回或设置单元格的字体。
ResetStyle	方法，清除单元格的样式，和所属行的样式保持一致。

通常只有在需要单独设置某单元格的样式时，才会用得着 ListViewCell，参考：行与单元格的样式

4.13 条形码

4.13.1 关于条形码

商品社会是离不开条形码的，Foxtable 提供了非常强大的条形码功能（包括控件和批量打印功能），可以生成 91 种条形码，几乎包括了所有的一维码和二维码：
AustraliaPost
AztecCode
Codabar
Code11
Code128
Code16k
Code32
Code39
Code93
CompactPdf417

DanishPostal39
DataMatrix
DeutschePostIdentcode
DeutschePostLeitcode
Ean13
Ean8
Ean99
EanUpcAddOn2
EanUpcAddOn5
EanVelocity
FedExGround96
FrenchPostal39AR
GS1128
GS1DataBar14
GS1DataBar14Stacked
GS1DataBar14StackedOmnidirectional
GS1DataBar14Truncated
GS1DataBarExpanded
GS1DataBarExpandedStacked
GS1DataBarLimited
GS1DataBarOmnidirectional
GS1DataBarStacked
GS1DataBarStackedOmnidirectional
GS1DataBarTruncated
HibcLic128
HibcLic39
HibcPas128
HibcPas39
IATA2of5
Industrial2of5
Interleaved2of5
Isbn
Isbt128
Ismn
Issn
ItalianPost25
Itf14
Jan13
Jan8
MacroPdf417
Matrix2of5
MaxiCode
MicroPdf417
MicroQRCode
Msi

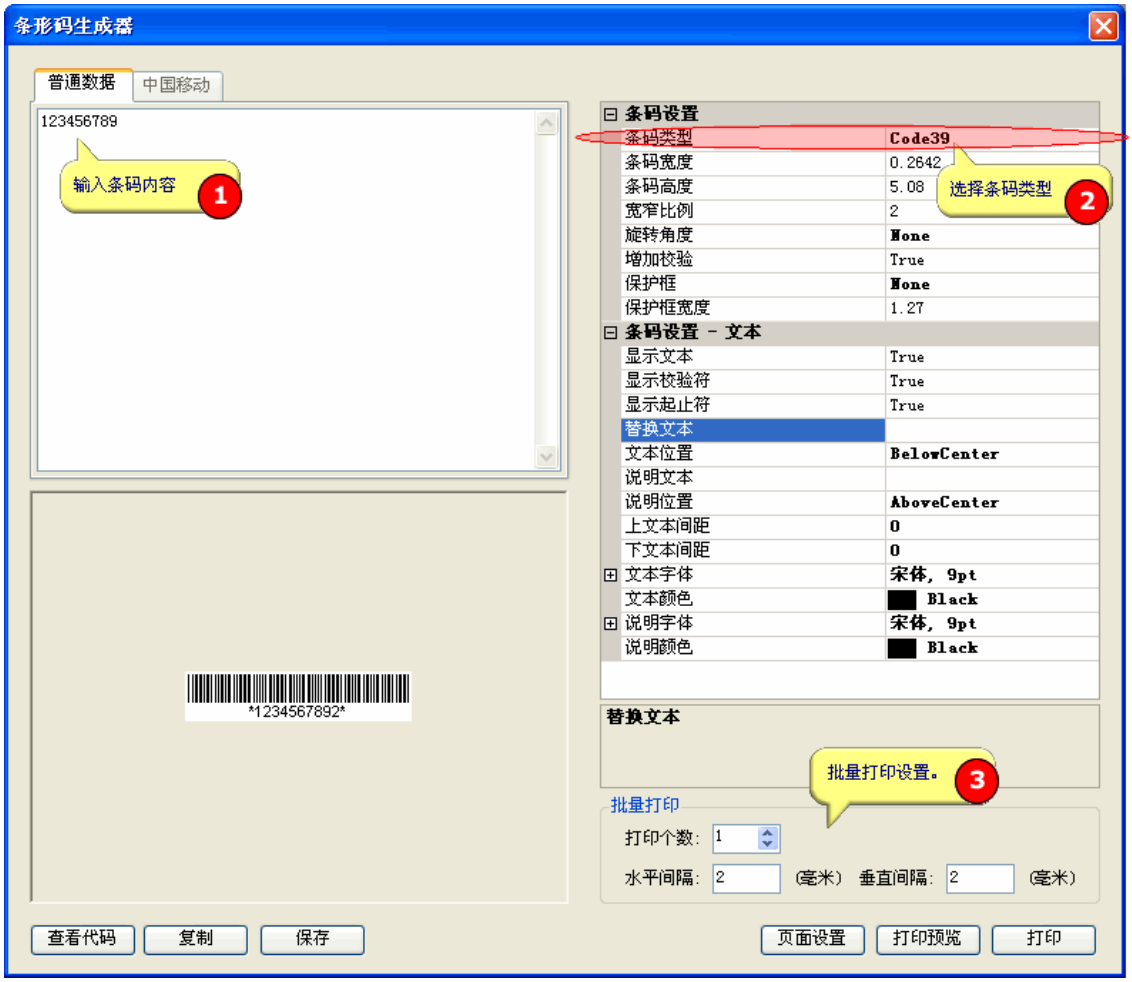
NumlyNumber
Opc
Pdf417
Pharmacode
Planet
Postnet
Pzn
QRCode
RoyalMail
RoyalTpgPostKix
Rss14
Rss14Stacked
Rss14StackedOmnidirectional
Rss14Truncated
RssExpanded
RssExpandedStacked
RssLimited
Scc14
Semacode
SingaporePost
Sscc18
SwissPostParcel
Telepen
UccEan128
UpcA
UpcE
UspsFim
UspsHorizontalBars
UspsIntelligentMail
UspsOneCode4CB
UspsPicCode128
UspsPicUccEan128
UspsSackLabel
UspsTrayLabel
VicsBol
VicsScacPro

4.13.2 批量打印条形码

如果需要临时打印一批条形码，可以使用 foxtable 提供的批量打印条形码功能：



此命令位于菜单的“杂项”功能区，下图是打印窗口，此窗口除了可以批量打印条形码外，还可以将生成的条形码复制到剪贴板，或者保存为一个图片文件 ：



提示：在上图的属性设置中，和尺寸相关的属性其单位均为毫米。

4.13.3 基本属性

4.13.3.1 条形码的尺寸

条形码图片的高度可以直接指定，不过宽度则无法直接指定，需要通过两个属性来间接确定：设置一维码尺寸的属性：

- **条码高度**
  指定生成的条形码图片的高度。

- **窄条宽度**
  条形码是由宽条和窄条及空白组成的，“窄条宽度”属性用于指定窄条的宽度。

- **宽窄比例**
  设置宽条和窄条的宽度比例，宽条的宽度等于：窄条宽度 * 宽窄比例。

- **静区宽度**
  设置条形码图片两侧空白区域的宽度。



设置二维码尺寸的属性：

- **模组尺寸**
  如果条码的类型是二维码，只能通过此属性设置二维码的大小，此属性越大，生成的二维码图片就越大。

4.13.3.2 条形码的校验

如果条码内容为“123456”，条码类型为 Code39 为例。
下图是有校验的时候生成的条形码图片:



下图是无校验的时候生成的条形码图片：



有两个属性和校验有关：

- **增加校验**
  是否在生成的条形图片中增加用于校验的条码。

- **显示校验符**
  是否在生成的条形码图片中显示用于校验的符号，例如上面第一个图中最后一个字符"L"，就是校验符。

4.13.3.3 文本的显示

在默认情况下，生成的条形码图片会显示条形码内容，例如下图底端的字符“123456”：



和文本显示相关的属性有：

- 显示文本
  是否在生成的条形码图片中显示文本。

- 显示校验符
  是否在显示的文本中包括校验符。

- 显示起止符
  是否在现实的文本中包括起止符号，例如上图中的“*”号。

- 文本位置
  用于设置文本在图片中的位置，默认为"BelowCenter"，也就是底部居中。

- 替换文本
  文本内容默认就是编码内容，你可以用此属性设置一个专门用于显示的文本，来替换编码内容。

- 说明文本
  可以在默认的文本外，再设置一个说明文本，两者可以同时显示，例如下图中的 foxtable 就是说明文本：



- 说明位置
  用于设置说明文本你的位置，默认为“AboveCenter”，也就是顶端居中。

- 上文本间距
  用于设置条形码图片上端的文本和图片本身之间的间距。

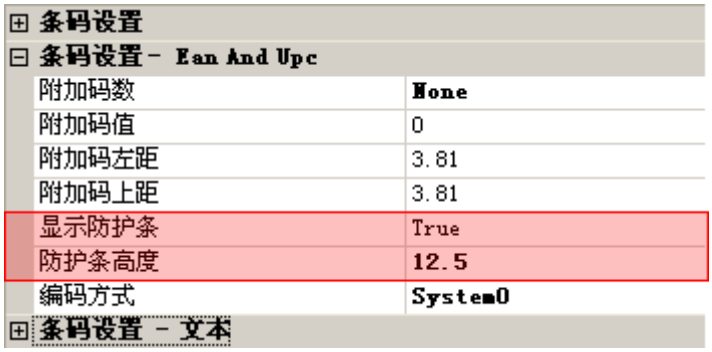
- 下文本间距
  用于设置条形码图片下端的文本和图片本身之间的间距。

4.13.3.4 防护条

有些条形码是有防护条的，例如：Ean8、Ean13、Jan8、Jan13、UpcA、UpcE 等，防护条有左、中、右三个，通常比数据条长：



通过属性设置，你可以决定是否显示防护条，以及防护条的高度：



4.13.3.5 旋转条形码图片

通过设置“旋转角度”属性，可以让生成的条形码图片旋转一定的角度，下图的图片旋转了 90 度：

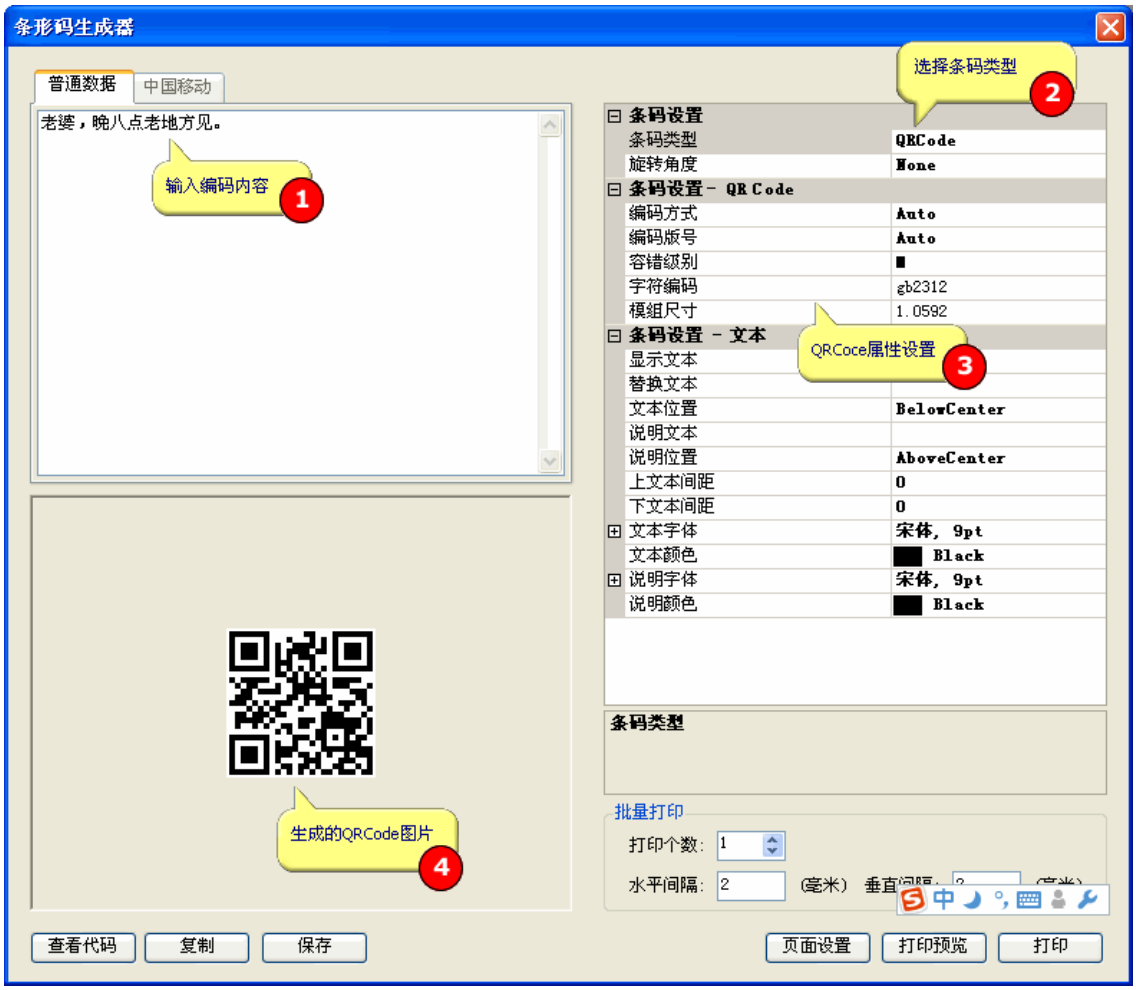


可供选择的角度有 0、90、180、270 四个角度。

4.13.4 常用二维码

4.13.4.1 QRCode

QRCode 码是由日本 Denso 公司研制的一种矩阵二维码符号，它具有信息容量大、可靠性高、支持汉字等特点，你甚至可以将一篇短文或者一个表单存储在 QRCode 格式的二维码中。
QRCode 也是中国移动所支持的两种二维码之一，目前大部分手机(特别是智能手机)都具备 QRCode 扫描功能。
生成 QRCode 的过程：



属性：

- 编码方式
  用于选择 QRCode 编码的方式，一般将其设置为“Auto”，由系统自己决定最佳的编码方式即可。

- 编码版号
  QRCode 的编码版号越大，生成的图片越大，图片容纳的信息越多，一般将此属性设置为“Auto”，由系统计算最佳版号即可。

- 容错级别
  QRCode 有四种容错级别:L、M、Q、H、分别表示：低、中、高、最高，默认为 M(中)；存储同样的编码信息，容错级别越高，生成的图片越大。

- 字符编码
  设置 QRCode 字符的编码格式，一般选择 gb2312 或 utf-8 即可，这两种编码均支持汉字。

- 模组大小
  QRCode 二维码图片由很多黑白的正方形小点组成，这些小点称之为模组。本属性指定单个某组的大小，此属性越大，生成的二维码图片越大，可分辨性越高。

4.13.4.2 DataMatrix

DataMatrix 是一种矩阵式二维条码，在同样的内容下，DataMatrix 生成的条形码尺寸是目前所有条码中最小的，特别适用于小零件的标识，以及直接印刷在实体上，Intel 出品的 CPU 上就有 DataMatrix 格式的二维码。



主要属性：

- 编码方式
  用于选择 DataMatrix 编码的方式，一般将其设置为“Auto”，由系统自己决定即可。

- 编码格式
  DataMatrix 二维码图片由很多黑白的正方形小点组成的一个矩阵，这些小点称之为模组；本属性决定生成的矩阵由多少个模组组成，C10X10，表示有 10*10 模组。
  通常将本属性设置为“Auto”,由系统自己决定最佳的矩阵大小。

- 模组尺寸
  DataMatrix 二维码图片由很多黑白的正方形小点组成，这些小点称之为模组。本属性指定单个某组的大小，此属性越大，生成的二维码图片越大，可分辨性越高。

4.13.4.3 PDF417

PDF417 二维条码是一种堆叠式二维条码，是由美国 SYMBOL 公司发明的，PDF（Portable Data File）意思是“便携数据文件”。组成条码的每一个条码字符由 4 个条和 4 个空共 17 个模块构成，故称为 PDF417 条码。



PDF417 生成的二维码图片，其高度由“窄条宽度 * 宽窄比例”计算得出。
主要属性：
编码方式

用于选择 PDF417 编码的方式，一般将其设置为“Auto”，由系统自己决定即可。

容错级别

用于选择 PDF417 编码的容错级别，共有 9 个容错级别，存储同样的编码信息，容错级别越高，生成的图片越大。

列数

用于设置 PDF417 编码图片的列数，值的范围在 1 到 30 之间，列数越大，生成的图片越宽，如果设为 0，则由系统自动计算得出合适的列数。

行数

用于设置 PDF417 编码图片的行数，值的范围在 3 到 30 之间，行数越大，生成的图片越高，如果设为 0，则由系统自动计算得出合适的行数。

高宽比例

用于设置 PDF417 编码图片高度和宽度的比例，值的范围在 1 到 1 之间，只有在行数和列数均设置为 0 的时候，本属性才会生效。

4.13.4.4 中国移动的二维码应用

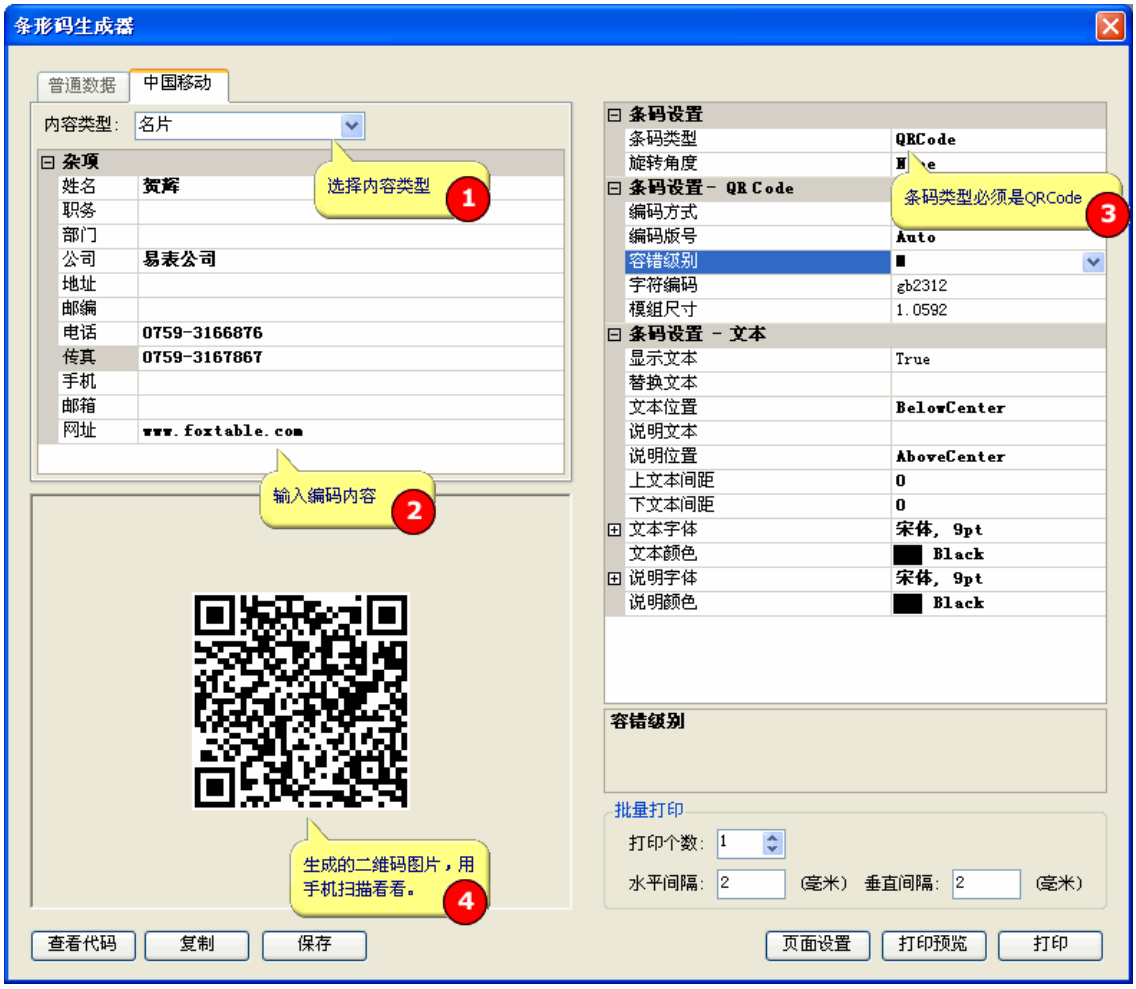
Foxtable 可以生成中国移动格式(采用 QR Code 编码)的二维码图片，供手机直接扫描使用。

例如你可以在你的名片上加上二维码图片，这样用户直接扫描一下，即可快速准确地将你的名片信息读取到手机的通讯录中，你还可以在宣传资料中加入二维码，用户只需扫描一下，即可将预先设置的短信内容发送到指定的号码，同样也可以将网址编码成一个二维码，让用户扫描一下即可访问你的网址，无需手工输入网址。

目前大部分手机(特别是智能手机)都具备 QRCode 扫描功能，如果你的手机还没有，可以到以下网址下载：

http://www.quickmark.com.tw/

下图是利用 Foxtable 生成中国移动格式二维码图片的过程：



4.13.5 BarCode(控件)

4.13.5.1 BarCode

除前面介绍条形码生成的批量打印功能外，Foxtable 提供了一个 BarCode 控件， 用于在窗口中生成各种条形码，包括各种一维码和二维码:



由于支持数据绑定，所以 Foxtable 内置的条形码功能，远比市面上一些专业的条形码生成软件来得更为实用。

和一般的控件不同，BarCode 的数据绑定是有一些特殊的，可以绑定三个字段，分别用于指定条码内容、替换文本、说明文本的来源字段：



有关条形码的基本属性设置，前面的章节已经介绍，这里就不在重复。

4.13.5.2 方法

4.13.5.2.1 SaveImage

将生成的图片保存为文件，语法：

SaveImage(FileName, DPI)

FileName：指定文件名。

DPI:　　可选参数，每英寸的像素点数，默认为 300，DPI 越大，图像越清晰。

系统会根据文件的扩展名以合适的格式保存文件，例如文件名是“c:\data\bar.bmp”，系统会以 BMP 格式保存文件，如果文件名是"c:\data\bar.wmf"，则以 WMF 格式保存文件。

示例：

```
With e.Form.Controls("BarCode1")
    .SaveImage("c:\data\001.bmp")
    .SaveImage("c:\data\002.bmp",600)
End With
```

4.13.5.2.2 CopyImage

将生成的图片复制到剪贴板中，语法：

CopyImage(DPI)
DPI: 可选参数，每英寸的像素点数，默认为 300，DPI 越大，图像越清晰。
示例

```
e.Form.Controls("BarCode1").CopyImage(600)
```

4.13.5.2.3 GetImage

获得一个 Image 对象，表示生成的图片：
GetImage(DPI)
DPI: 可选参数，每英寸的像素点数，默认为 300，DPI 越大，图像越清晰。
示例

```
e.Form.Controls("BarCode1").GetImage(600)
```

4.13.5.2.4 DrawOnCanvas

在指定的 Graphics 对象上绘制生成的条形码图片，有关 Graphics 对象，请参考 GDI+入门
语法：
DrawOnCanvas(graphics, x, y, scale)
graphics： 用于绘图的 Graphics
x: 水平起始位置
y: 垂直起始位置
scale 可选参数，指定缩放倍数

4.13.6 BarCodeBuilder

BarCodeBuilder 有 BarCode 的全部属性和成员。
BarCodeBuilder 不需要加入到窗口中，即可生成条形码图片。
示例一

```
Dim Bar As New BarCodeBuilder
Bar.Symbology = Barpro.Symbology.Code39
Bar.Code = "123456"
Bar.BarWidth = 0.3
Bar.BarHeight = 8
Bar.SaveImage("c:\data\MyBar.jpg")
```

示例二
假定表 A 有一个名为"条码"的列，希望批量打印出所有条码，代码为：

```
Dim doc As New PrintDoc
Dim img As prt.RenderImage
Dim Bar As New BarCodeBuilder
Bar.Symbology = Barpro.Symbology.Code39
Bar.BarRatio = 0.3
Bar.BarHeight = 8
```

```
Doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight
For Each r As Row In Tables("表 A").Rows
    bar.Code = r("条码")
    img = new prt.RenderImage
    img.Image = bar.GetImage
    Doc.Body.Children.Add(img)
Next
Doc.Preview()
```

如果你想每行打印四个条码，可以用下面的代码：

```
Dim doc As New PrintDoc
Dim img As prt.RenderImage
Dim Bar As New BarCodeBuilder
Bar.Symbology = Barpro.Symbology.Code39
Bar.BarRatio = 0.3
Bar.BarHeight = 8
Doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight
For Each r As Row In Tables("表 A").Rows
    For i As Integer = 1 To 4
        bar.Code = r("条码")
        img = new prt.RenderImage
        img.Image = bar.GetImage
        Doc.Body.Children.Add(img)
    Next
Next
Doc.Preview()
```

小提示：

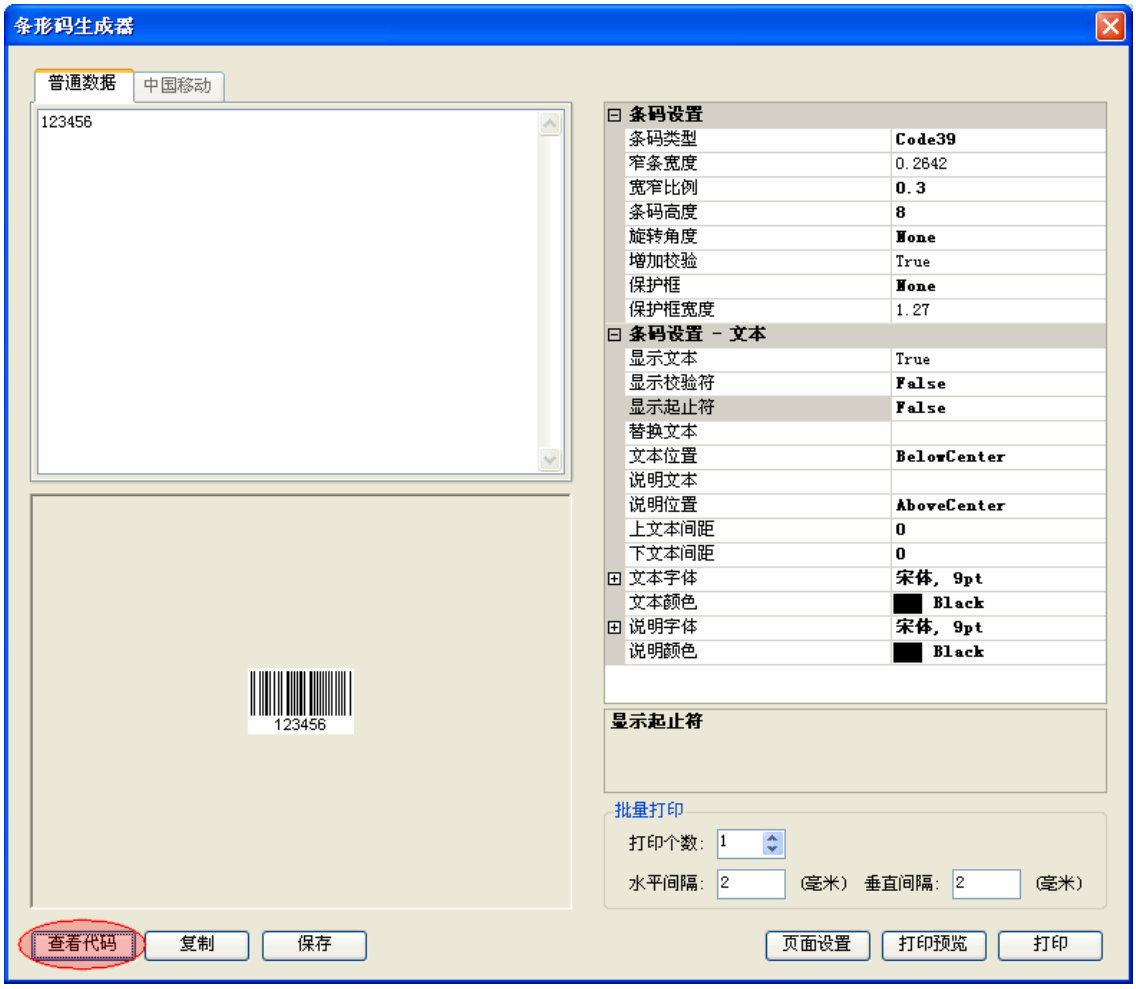
你也许会有困惑，帮助并没有介绍 BarCodeBuilder 的属性，实际编码的时候如何设置其属性呢？

不用担心，Foxtable 为您考虑得非常周到，请参考：自动生成条形码代码

4.13.7 自动生成代码

Foxtable 能够生成 91 种条形码，几乎包括了所有的一维码和二维码，每一种条形码都会有自己独特的属性，所以一一介绍这些属性几乎是不现实的，对于用户的学习也不利。

不过不用担心，虽然我们没有介绍所有的属性，但是却提供了一个更为便利和友好的方式，只需在批量打印条形码窗口，输入条码内容，合适的条码类型，并设置好其属性，然后单击“查看代码”按钮，即可自动生成一段代码：



例如按照上图所示设置好之后，单击查看代码按钮，即可生成代码：

```
Dim Bar As New BarCodeBuilder
Bar.Symbology = Barpro.Symbology.Code39
Bar.Code = "123456"
Bar.BarRatio = 0.3
Bar.BarHeight = 8
Bar.DisplayChecksum = False
Bar.DisplayStartStopChar = False
Bar.SaveImage("输入文件路径和名称")
```

4.13.8 为 Excel 报表增加条形码功能

Excel 报表并没有条形码功能，但是有引用图片文件的功能。

如果我们将 BarCode 控件或 BarCodeBuilder 类型生成的图表保存为一个约定的文件，并在 Excel 报表中引用此约定的文件，那么就等于间接地为 Excel 报表加上了条形码功能。

示例

1、设计模版的时候，引用约定的图片文件"myimg.wmf"。

2、在窗口中加入一个 Button 控件。

3、Button 控件的代码设置为：

```
Dim Bar As New BarCodeBuilder
```

```
Bar.Symbology = Barpro.Symbology.Code39
Bar.Code = "123456"
Bar.BarRatio = 0.3
Bar.BarHeight = 8
bar.SaveImage(ProjectPath & "Images\myimg.wmf",300)
'生成 Excel 报表：
Dim Book As New XLS.Book(ProjectPath & "Attachments\销售统计.xls")
Book.Build()
Book.Save(ProjectPath & "Reports\销售统计.xls")
```

提示：保存图片的时候 DPI 参数设置为 300 或者更高，因为默认的 DPI 参数为 96，打印精度不高。

4.14 动态加载

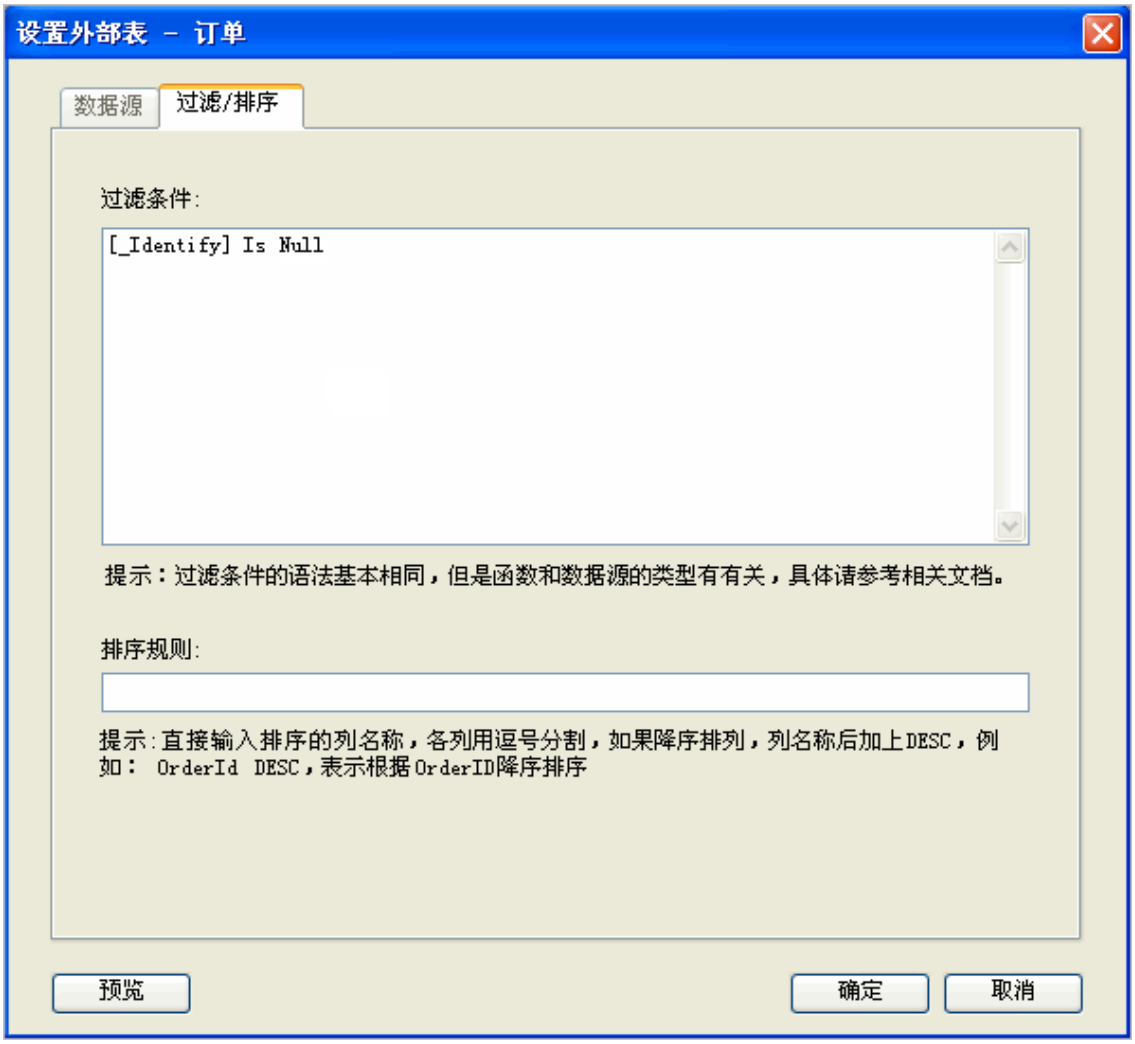
4.14.1 动态加载数据

4.14.1.1 限制初始加载量

在默认情况下，Foxtable 会加载所有数据，如果数据量较大，这种方式显然是不恰当的。
利用现有功能，我们可以实现在打开项目的时候，不加载或者只加载部分数据，在运行的过程中，根据需要加载数据。

打开项目时控制加载数据量

对于内部表，请参考事件：**BeforeLoadInnerTable**
对于外部表，请参考事件：**BeforeLoadOuterTable**
对于外部表，如果初始加载条件是固定的，应该在定义数据表时直接定义加载条件，而不是使用 BeforeLoadOuterTable 事件：

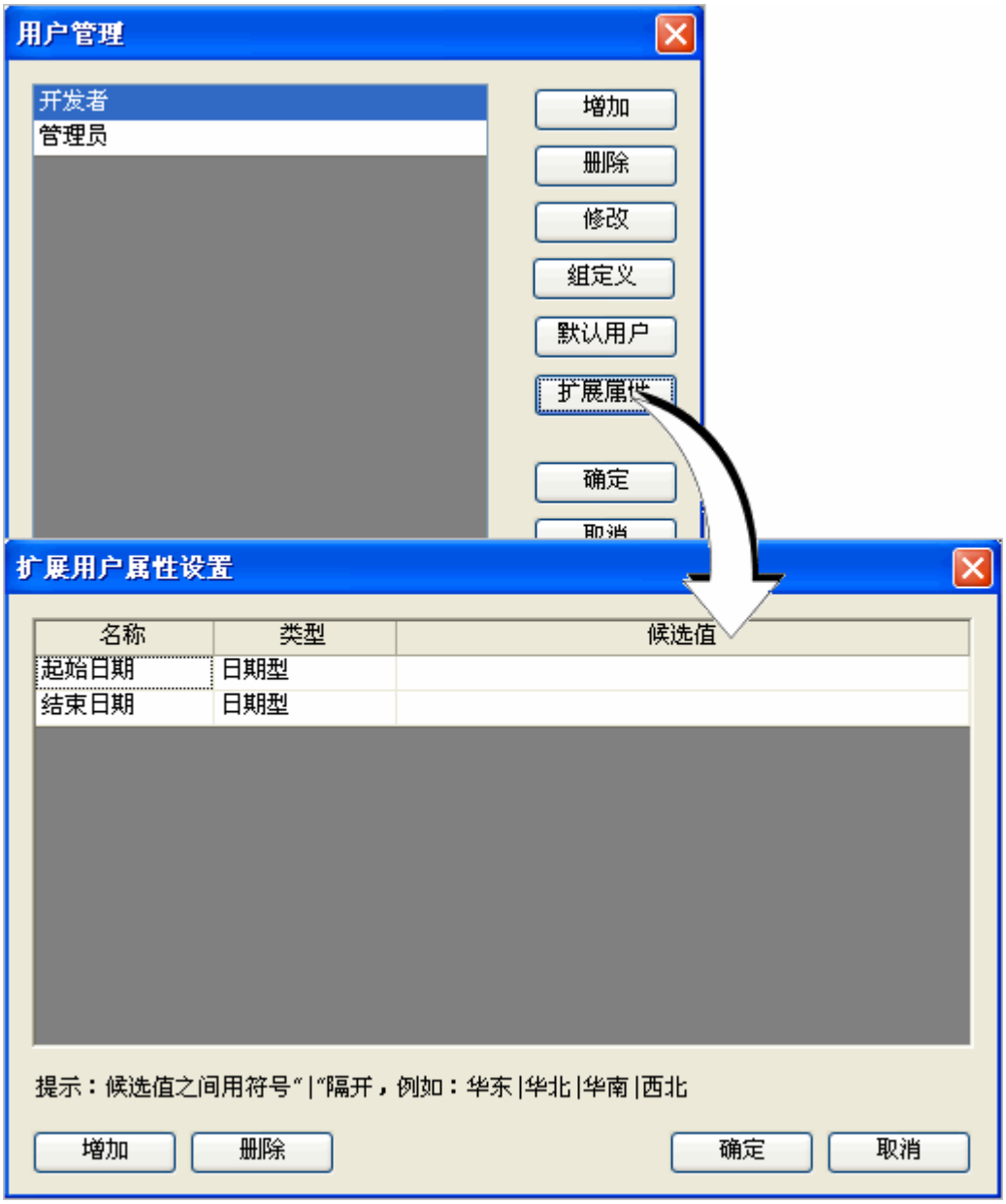


利用扩展用户属性设置初始加载条件

通常利用扩展用户属性，让用户在登录的时候，选择要加载的数据范围。

例如有一个订单管理系统，希望在用户登录的时候，能够指定一个日期范围，仅加载该日期范围内的订单。

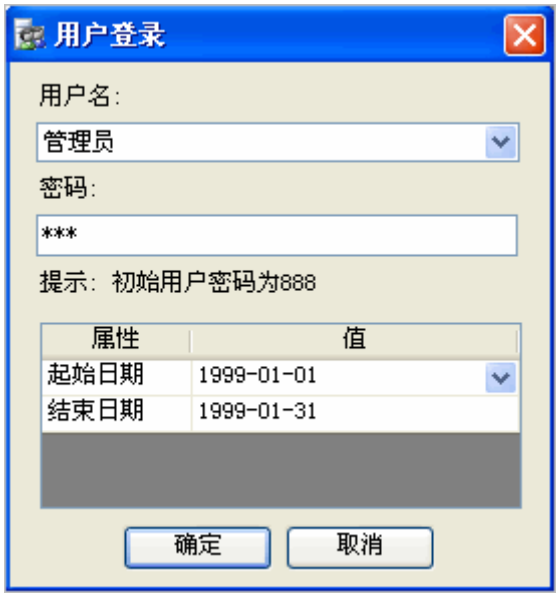
首先我们得在用户管理中增加起始日期、结束日期两个扩展属性：



假定使用的是内部表，然后在 BeforeLoadInnerTable 事件中加入如下代码：

```
If e.DataTableName = "订单" Then
    e.Filter = "[日期] >= #" & e.User.ExtendedValues("起始日期") & "# And [日期] <= #" & e.User.ExtendedValues("结束日期") & "#"
End If
```

这样即可在登录的时候，由用户自行输入要加载数据的日期范围：



4.14.1.2 运行过程中加载

4.14.1.2.1 相关属性和方法

DataTable 提供了以下属性和方法，用于在运行过程中动态加载数据。

名称	类型	说明
LoadFilter	属性	字符型，用于设置重新加载数据的条件表达式，语法和所使用的数据源有关。
LoadOrder	属性	字符型，指定加载数据的顺序。
LoadTop	属性	字符型，指定要加载的行数，如果是分页加载，则用于指定每页的行数。
LoadReverse	属性	逻辑型，是否倒序加载，即是否先加载新数据，此属性只有在分页加载的时候才有效。
LoadOver	属性	字符型，用于指定分页加载依据列。
LoadPage	属性	整数型，指定要加载的页号。
TotalPages	属性	整数型，返回分页加载时，总的可加载总页数。
TotalRows	属性	整数型，返回分页加载时，后台数据表符合当前加载条件的总行数。
Load	方法	加载数据。

示例一

首先利用 DataTable 的 LoadFiler 属性设置加载条件，然后执行 Load 方法重新加载数据。

这样我们就可以根据用户输入的条件，重新加载数据。

例如：

```
DataTables("订单").LoadFilter = "[日期] = #" & Date.Today & "#"
```

```
DataTables("订单").Load
```

上面的代码将从后台中加载今天的订单。

需要注意的是，LoadFilter 采用的是 SQL 语法，和常规表达式的语法有所不同，且不同的数据源，语法也会有细节差异。

例如对于 SQL SERVER 数据源，日期必须用单引号括起来：

```
DataTables("订单").LoadFilter = "[日期] = '" & Date.Today & "'"
DataTables("订单").Load
```

请参考：SQL 表达式的语法差异。

示例二

可以用 LoadTop 属性设置要加载的行数，通常和 LoadOrder 配合使用。

例如加载最近的 100 条订单：

```
DataTables("订单").LoadTop = "100"
DataTables("订单").LoadOrder = "日期  Desc"
DataTables("订单").Load
```

需要注意的是，上述代码加载进来的数据并不会按照日期排序，只是按日期排序加载最近的 100 条订单而已，至于加载进来后的数据的顺序，由 Table 的 Sort 属性决定。

我们可以加上关键词 Percent，表示按百分比加载。

例如按日期顺序，加载产品为 PD01 的最近的 20%的订单：

```
DataTables("订单").LoadTop = "20 Percent"
DataTables("订单").LoadOrder = "日期  Desc"
DataTables("订单").LoadFilter = "产品  = 'PD01'"
DataTables("订单").Load
```

提示：按百分比加载对于 Oracle 数据源无效，因为 Oracle 不支持 Percent 关键词。

示例三

如果一个表的行数非常多，可以考虑分页加载，此时用 LoadTop 指定每页的行数，用 LoadPage 要加载的页号。

下面的代码每页加载 50 个订单，加载第三页，注意页号从 0 开始，所以第三页的编号为 2：

```
With DataTables("订单")
    .LoadTop = 50
    .LoadPage = 2
    .Load
End With
```

需要注意的是分页加载不能按百分比，只能按行数，例如下面的代码，始终加载的都是前 10%的订单，而不是第三个 10%的订单：

```
With DataTables("订单")
    .LoadTop = "10 Percent"
    .LoadPage = 2
    .Load
End With
```

示例四

分页加载的时候，默认是根据主键分页的，可以用 LoadOver 属性改变分页列。

例如：

```
With DataTables("订单")
```

```
    .LoadTop = 20
    .LoadPage = 3
    .LoadOver = "日期"
    .Load()
End With
```

表示根据日期列分页，每页20行，加载第四页。

我们的建议是：尽量采用主键列或者其他值不会重复的列（如身份证号码），作为分页依据列，因为如果分页依据列的值有重复，将会影响分页的准确性，例如上面的代码根据日期分页，每页显示20行，如果某一天有40个订单，那么这40个订单会显示在同一个页面中，而不会按我们期望的那样分成两页显示。

示例五

分页加载默认是顺序，如果希望倒序加载，可以将LoadReverse属性设置为True。
例如按日期分页倒序加载，也就是最新的数据在前：

```
With DataTables("订单")
    .LoadTop = 20
    .LoadPage = 3
    .LoadOver = "日期"
    .LoadReverse = True
    .Load()
End With
```

需要注意的是，经常有用户用LoadOrder属性来设置分页加载的顺序，这是无效的。
要记住：分页加载的时候，用LoadOver和LoadReverse来设置分页列和加载顺序，非分页加载的时候，用LoadOrder设置加载顺序。

示例六

在分页加载时，可以用 TotalPages 属性返回总的可加载总页数，这样我们就可以设计出上一页、下一页、第一页、最后一页这样的加载按钮。
在使用此属性之前，必须先设置 LoadTop 属性，如果还有加载条件，也得先设置 LoadFilter 属性，才能返回正确的页数。
例如：

```
Dim p As Integer = 3
With DataTables("订单")
    .LoadTop = 20
    If p <= .TotalPages Then
        .LoadPage = p
        .Load
    End If
End With
```

示例七

如果你只想按照目前的加载设置，重新加载表，也就是同步表，只需：

```
DataTables("订单").Load
```

再例如重新加载所有表

```
For Each dt As DataTable In DataTables
    dt.Load()
Next
```

在默认的情况下，Load 在加载数据之前，会先保存现有数据。

Load 有一个可选逻辑参数，如果设置为 False，将不保存现有数据，例如：

```
DataTables("订单").Load(False)
```

4.14.1.2.2 分页加载示例

本节的例子可以参考 CaseStudy 目录下的文件：分页加载.Table。

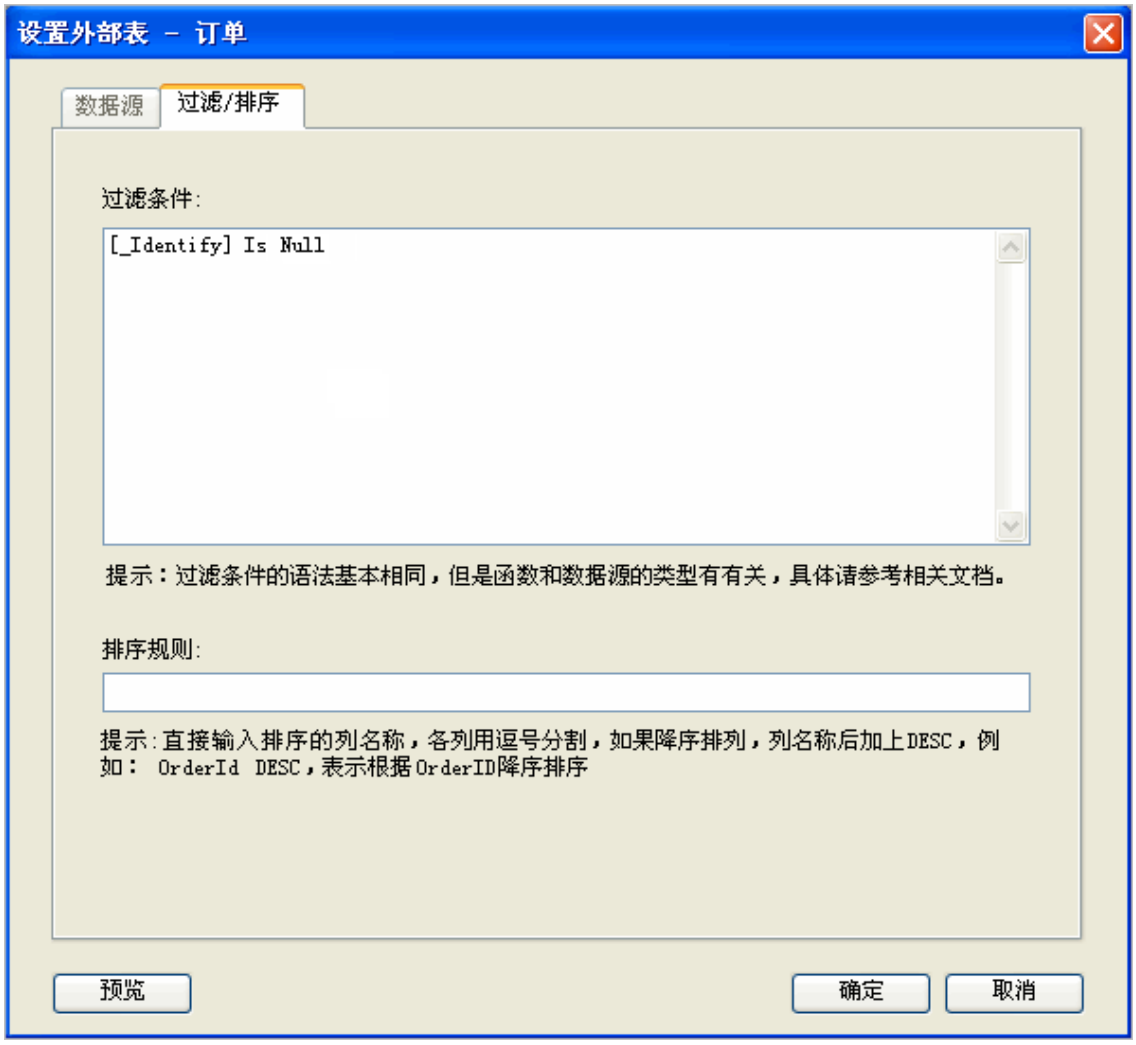
假定要求订单表分页加载，每页 20 行，提供第一页、上一页、下一页、最末页等加载按钮：

1、首先要确保默认不加载任何数据。

如果是内部表，设置项目事件 BeforeLoadInnerTable 的代码为：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

如果是外部表，可以在定义外部表的时候，直接定义加载条件：

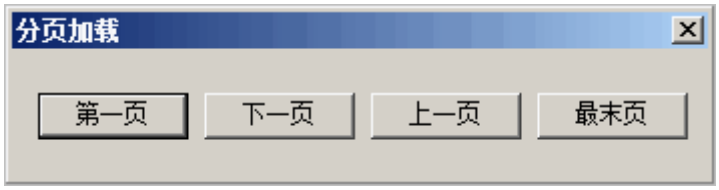


2、然后在项目事件 AfterOpenProject 中设置代码，使得打开项目后，自动加载第一页：

```
With DataTables("订单")
    .LoadFilter = "" '一定要清除加载条件
    .LoadTop = 20
    .LoadPage = 0
    .Load()
End With
```

提示：由于已经在第一步设置了加载条件"[_Identify] Is Null"，所以必须在 AfterOpenProject 事件 中清除此条件，否则不会加载任何数据 。

3、最后设计一个下图所示的加载窗口：



各按钮的代码如下:

按钮	代码
第一页	With DataTables("订单") If .LoadPage <> 0 Then

	.LoadTop = 20 .LoadPage = 0 .Load() End If End With

下一页	With DataTables("订单")
	If .LoadPage < .TotalPages - 1 Then
	.LoadPage = .LoadPage + 1
	.Load()
	End If
	End With

上一页	With DataTables("订单")
	If .LoadPage > 0 Then
	.LoadPage = .LoadPage - 1
	.Load()
	End If
	End With

最末页	With DataTables("订单")
	If .LoadPage < .TotalPages - 1 Then
	.LoadPage = .TotalPages - 1
	.Load()
	End If
	End With

提示：

上面的例子讲述了如何针对常规的表实现分页加载，本节的示例文件其实还有一个名为“窗口 2”的窗口，用于演示如何针对 SQLTable 实现分页加载，由于本质上和上面的 例子没有什么差别，就不再重复讲述了，大家有兴趣的话可以自己看看。

4.14.1.2.3 加载树示例

其实大多数时候，我们是不需要自己设计分页加载的，因为 Foxtable 内置的加载树兼具了分页加载和树形加载的功能：



假定对于订单表要采用上图所示的加载树，并开启分页加载功能，每次加载 30 行。

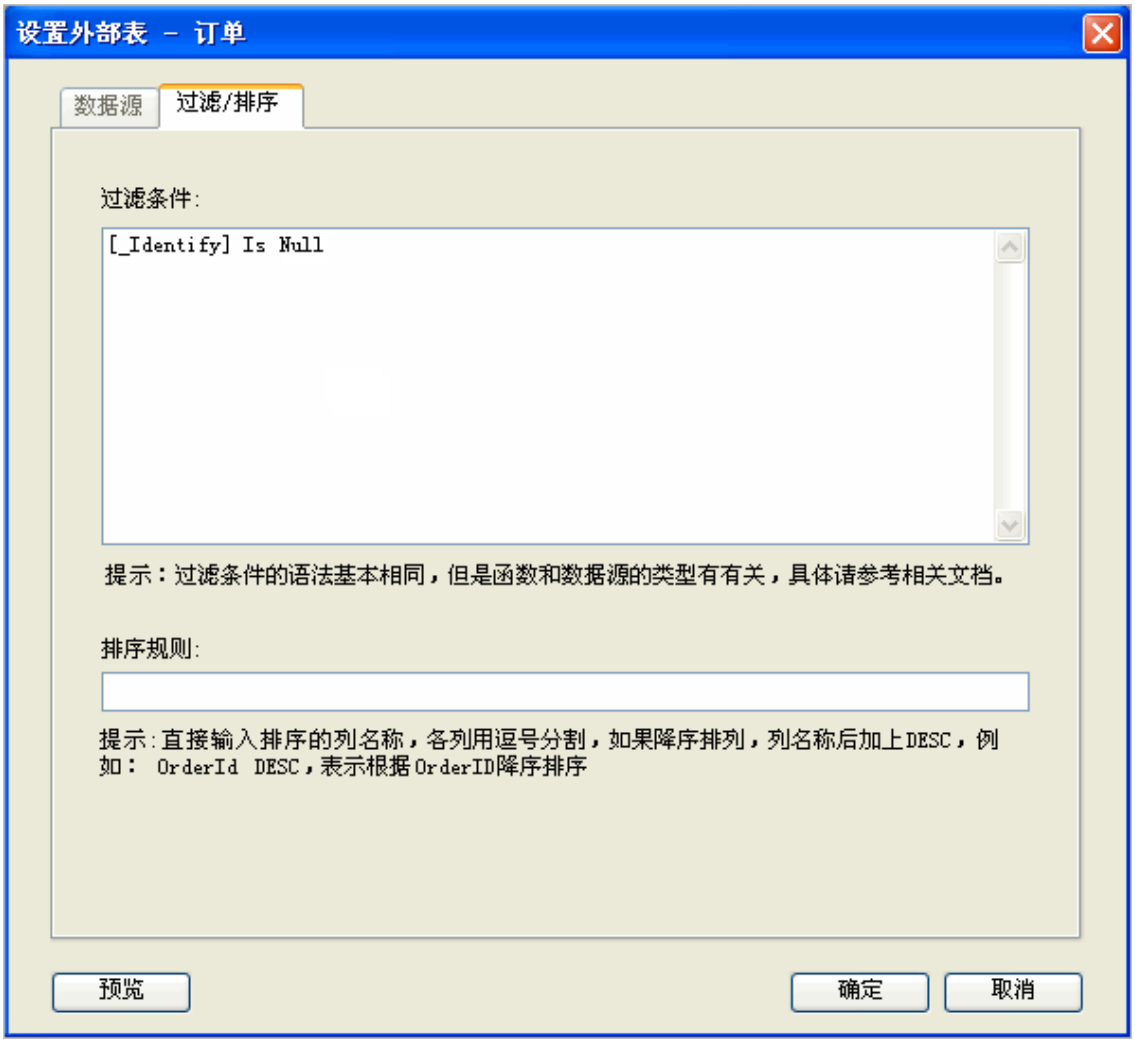
设计步骤：

1、首先要确保默认不加载任何数据。

如果是内部表，设置项目事件 BeforeLoadInnerTable 的代码为：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

如果是外部表，可以在定义外部表的时候，直接定义加载条件：



2、在 AfterOpenProject 事件中设置代码，打开加载树 ，每页加载 30 行，并默认加载第一页数据：

```
Tables("订单").OpenLoadTree("产品|客户",150,30,True)
```

关于打开加载树的更多知识，请参考：OpenLoadTree

4.14.1.2.4 加载树与权限管理

OpenLoadTree 方法的加最后一个参数 Filter 很容易被人忽视，这是一个条件表达式，一旦指定此参数，那么只会根据符合此条件的行生成加载树；单击加载树中的节点，也只会加载符合此条件

的行。

例如，假定希望雇员分组的用户在登录之后，只加载自己负责的数据，可以将打开加载树的代码改为：

```
Dim flt As String
If User.Group = "雇员" Then
	flt = "雇员 = '" & User.Name & "'"
End If
Tables("订单").OpenLoadTree("日期 YM|产品",150,100,True,flt)
```

4.14.1.2.5 追载新数据

Load 方法在执行的时候，会清除已经加载的行，然后重新加载行，加载条件由 LoadFilter 属性指定。

有的时候并不希望清除已经加载的行，只是希望从后台找出符合条件的行，追加到当前表中，我们称之为追载。

用于追载的方法是 AppendLoad。

语法：

AppendLoad(Filter,Save)

Filter：字符型，用于设置追载条件，采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。

Save: 可选参数，逻辑型，是否在追载数据前先保存现有数据，默认为 True，既先保存后追载。

AppendLoad 会返回一个集合，包含所有追载的行(DataRow)。

需要注意的是：

1、必须确保即将追载的行，并不存在于表中，也就是这些行之前并未加载，否则将报错。

2、AppendLoad 适合追载少量数据，大量数据一般用 Load 重新加载。

示例

假定甲、乙两个用户同时编辑表 A，甲负责增加行，并输入部分列的数据，乙负责输入其它列的数据。

我们需要为乙用户设计一个按钮，用于将打开项目以来，甲用户新增加的行追载到表中，按钮的代码为：

```
Dim id As Integer = DataTables("表 A").Compute("Max(_Identify)")
Dim Filter As String = "[_Identify] > " & id
DataTables("表 A").AppendLoad(Filter, False)
```

上述代码先计算出表 A 中"_Identify"列的最大值，然后从后台追载"_Identify"列的值大于此最大值的行，这些行就是甲用户新增加的行。

提示：

1、甲用户新增行并输入数据后，必须保存，乙用户才可追载此新增加的行，这是因为 AppendLoad 是从后台追载的，如果新增行没有保存，后台就不存在此行， 乙用户也就谈不上追载了。

2、AppendLoad 的作用远不止于追载新增数据，在设计一些数据自动流转的项目，例如工作流时，AppendLoad 方法是关键。

4.14.1.2.6 按条件刷新数据

AppendLoad 不仅可以用于追载数据，还可以用于按条件批量刷新数据，也就是重新加载符合某一条件的数据。

示例

例如刷新今天的订单，即重新加载今天的订单：

```
LockBaseMainForm() '锁定主界面
DataTables("订单").StopRedraw()
DataTables("订单").RemoveFor("日期 = #" & Date.Today & "#")
DataTables("订单").AppendLoad("日期 = #" & Date.Today & "#")
DataTables("订单").ResumeRedraw()
UnLockBaseMainForm() '解锁主界面
```

提示：

1、由于AppendLoad只能追载并不存在于DataTable中的行，所以我们首先要用RemoveFor移除要刷新的行，然后再用AppendLoad追载这些行。

2、我们只能用这种方法小批量地刷新数据，如果需要大量刷新数据，不如直接执行DataTable的Load方法来得简洁高效。

解决屏幕闪烁的终极方法

通过表的StopRedraw和ResumeRedraw方法，可以在执行某些操作之前停止绘制表，操作完成之后恢复绘制表，以避免屏幕闪烁。

但是某些特殊情况，这一招可能会失效，例如上面的代码中连续使用了RemoveFor和AppendLoad，此时单单依靠表的StopRedraw和ResumeRedraw，屏幕 依然会有明显的闪烁。

Foxtable提供了一对方法，分别为LockBaseMainForm和UnLockBaseMainForm方法，前者用于停止绘制主界面，后者用于恢复绘制主界面。

如果使用常规的方法无法解决屏幕闪烁问题，可以如本节的示例代码一样，将相关的代码放在这一对方法之间即可。

如果表位于窗口之中，需要将代码放在Form的StopRedraw和ReseumeRedraw方法之间，才能避免闪烁：

```
Forms("窗口名").StopRedraw() '停止绘制窗口
DataTables("订单").StopRedraw()
DataTables("订单").RemoveFor("日期 = #" & Date.Today & "#")
DataTables("订单").AppendLoad("日期 = #" & Date.Today & "#")
DataTables("订单").ResumeRedraw()
Forms("窗口名").ReseumeRedraw() '停止绘制窗口
```

并非所有的操作都需要如此处理，多数时候只需使用表的StopRedraw和ResumeRedraw方法，就可

以避免闪烁的发生，到目前为止，我也只发现连续使用RemoveFor和AppendLoad的时候，才需要做这样的特殊处理。

4.14.1.2.7 父行和子行的同步刷新

我们知道刷新（重新加载）指定行的数据非常简单，只需执行 DataRow 或者 Row 的 Load 方法即可，例如重新加载订单表的当前行数据：

```
Tables("订单").Current.Load()
```

如果两个表之间建立了关联，希望重新加载某一父行数据后，也需要重新加载该父行对应的全部子行。
例如订单表和订单明细表通过订单编号建立了关联，希望重新加载当前订单及其订单明细，代码为：

```
Dim r As Row = Tables("订单").Current
Dim fl As String = "订单编号  = '"  & r("订单编号") & "'"
r.Load() '加载父行
LockBaseMainForm() '锁定主窗口
DataTables("订单明细").StopRedraw '停止绘制订单明细
DataTables("订单明细").RemoveFor(fl) '移除当前订单的订单明细
DataTables("订单明细").Appendload(fl) '追载当前订单的订单明细
DataTables("订单明细").ResumeRedraw '恢复绘制订单明细
UnLockBaseMainForm() '解锁主窗口
```

代码的原理是 首先刷新当前订单，然后用RemoveFor移除该订单对应的全部订单明细，最后用AppendLoad追载该订单对应的全部订单明细。

很多人可能会习惯这样写代码：

```
Dim dr As DataRow = Tables("订单").Current.DataRow
dr.Load()
For Each cr As DataRow In dr.GetChildRows("订单明细")
    cr.Load()
Next
```

显然代码看上去更简洁，但是有缺陷，因为无法加载他人新增的订单明细，也无法移除他人已经删除的订单明细，仅仅是更新已经加载的订单明细而已。

4.14.1.2.8 按时段加载数据

我们经常需要加载某个时段的数据，下面列出了各种情况的加载代码给大家参考。
加载指定年份数据

例如加载 2011 年的订单：

```
Dim y As Integer = 2011 '指定年份
Dim dt1 As New Date(y, 1, 1)
Dim dt2 As New Date(y, 12, 31)
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

加载指定月份数据

例如加载 2011 年 6 月份的订单：

```
Dim y As Integer = 2011 '指定年份
Dim m As Integer = 6 '指定月份
Dim dt1 As New Date(y, m, 1)
Dim dt2 As New Date(y, m, Date.DaysInMonth(y,m)) '获取该月的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

加载指定季度的数据

例如加载 2011 年 2 季度的订单：

```
Dim y As Integer = 2011 '指定年份
Dim q As Integer = 2 '指定季度
Dim dt1 As New Date(y, 3 * (q - 1) + 1, 1) '获取该季度的第一天
Dim dt2 As New Date(y, 3 * q, Date.DaysInMonth(y,3 * q)) '获取该季度的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

加载本年数据

例如加载本年的订单：

```
Dim y As Integer = Date.Today.Year
Dim dt1 As New Date(y, 1, 1)
Dim dt2 As New Date(y, 12, 31)
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

加载本季度数据

例如加载本季度的订单：

```
Dim y As Integer = Date.Today.Year
Dim q As Integer = (Date.Today.Month - 1) \ 3 + 1 '计算现在是第几个季度
Dim dt1 As New Date(y, 3 * (q - 1) + 1, 1) '获取本季度的第一天
Dim dt2 As New Date(y, 3 * q, Date.DaysInMonth(y,3 * q)) '获取本季度的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

注意第二行代码，用的是整除运算符（\），而不是通常的除法运算符（/），整除运算符会直接对结果取整。

加载本月数据

例如加载本月的订单：

```
Dim y As Integer = Date.Today.Year
Dim m As Integer = Date.Today.Month
Dim dt1 As New Date(y, m, 1)
Dim dt2 As New Date(y, m, Date.DaysInMonth(y, m)) '获取本月的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

加载本周数据

例如加载本周的订单：

```
Dim y As Integer = Date.Today.Year
Dim w As Integer = Date.Today.DayOfWeek '算出今天是星期几
Dim dt1 As Date = Date.Today.AddDays(-w) '获取本周的第一天
Dim dt2 As Date = Date.Today.AddDays(6 - w) '获取本周的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

注意事项

1、前面的代码全部是以内部表或Access数据源作为数据源的，如果你使用SQL Server作为数据源，那么日期不是用符号#括起来，而是用单引号括起来，所以下面这样代码：

```
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
```

应该改为：

```
Filter = "日期 >= '" & dt1 & "' And 日期 <= '" & dt2 & "'"
```

2、这一节虽然以加载数据为例，但有关知识同样可以用于筛选，只需将最好两行代码：

```
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

改为：

```
Tables("订单").Filter = Filter
```

即可用于筛选数据。

例如筛选本周订单：

```
Dim y As Integer = Date.Today.Year
Dim w As Integer = Date.Today.DayOfWeek '算出今天是星期几
Dim dt1 As Date = Date.Today.AddDays(-w) '获取本周的第一天
Dim dt2 As Date = Date.Today.AddDays(6 - w) '获取本周的最后一天
Dim Filter As String
Filter = "日期 >= #" & dt1 & "# And 日期 <= #" & dt2 & "#"
Tables("订单").Filter = Filter
```

4.14.1.2.9 SQLLoad

本节内容适合熟练掌握 SQL 语句的用户。
Foxtable 提供的动态加载和分页加载，使用简单，能胜任绝大多数场合的需求。
但是也有局限，因为 Load 方法需要拆解现有 Select 语句，然后根据 LoadFilter 等属性的设置，重新合成 Select 语句，在面对复杂的 Select 语句的时候，这种拆解和合成可能会失败，导致数据加载失败。
为解决这个问题，Foxtable 专门提供了一个 SQLLoad，该方法允许用户自行编写 Select 语句加载数据。
使用 SQLLoad 方法有两个原则：
1、新的 Select 语句不能改变表的原有结构。
2、新的 Select 语句也必须从相同的后台表中提取数据。
如果表是数据表，那么必须遵循上述两个原则，如果表是查询表，只需遵循第一个原则即可。
示例
我们知道 SQL Server 从 2005 开始，提供了一个 ROW_NUMBER 函数， 用于返回行号，通过这个函数，可以非常方便地实现分页加载。

下面我们用一个具体的例子，讲述一下如何基于这个函数来实现分页加载，假定每页 50 行：
1、增加一个查询表，其 Select 语句为：

Select * From (Select Row_Number() Over(Order by [订单 ID]) As RowNum, * from 订单) As a Where RowNum >= 1 And RowNum <= 50
默认加载前 50 行数据，也就是第一页。
2、在全局代码中定义一个 Public 变量，用于记录当前页号：

```
Public Page As Integer = 1
```

3、增加一个自定义函数，函数名为 LoadData，用于加载当前页数据：

```
Dim r1 As Integer = (Page - 1) * 50 + 1
Dim r2 As Integer = Page * 50
Dim sql As String
sql = "Select * From (Select Row_Number() Over(Order by [订单 ID]) As RowNum, * from 订单) As a "
sql = sql & " Where RowNum >= " & r1 & " And RowNum <= " & r2
DataTables("订单").SQLLoad(sql)
```

4、再增加一个自定义函数，名为 Pages，用于计算总的页数：

```
Dim cmd As New SQLCommand
Dim cnt As Integer
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select Count(*) From {订单}"
cnt = cmd.ExecuteScalar
Return Math.Ceiling(cnt / 50) '计算总的页数,每页50行
```

3、在窗口或者菜单中，加上四个按钮，按钮的 Click 事件代码分别设置为：

按钮	代码
第一页	Page = 1 Functions.Execute("LoadData")

上一页	If Page > 1 Then
	  Page = Page - 1
	  Functions.Execute("LoadData")
	End If

下一页	If Page < Functions.Execute("Pages") Then
	  Page = Page + 1
	  Functions.Execute("LoadData")
	End If

最末页	Dim pg As Integer = Functions.Execute("Pages")
	If Page <> pg Then
	  Page = pg
	  Functions.Execute("LoadData")
	End If

4.14.1.2.10 使用后台数据库的视图

前面已经提到，对于复杂的 Selelct 语句，Foxtable 的分页加载和加载树往往无能为力，尽管提供了一个 SQLLoad 方法来实现分页加载，但是使用上还是不够便利，而且也无法解决加载树的问题。其实我们可以将复杂的 Select 语句定义为后台数据库的一个视图(Access 称之为查询)，这样我们就可以将这个 Select 语句当做一个普通的表使用，以常规的方式来实现分页加 载和显示加载树。

示例

1、以 Access 数据源为例，假定有一段如下所示用于销售分析的 Select 语句：

```
SELECT 订单.[订单 ID], 订单.订单日期, 员工扩展信息.员工姓名 AS 员工, 客户扩展信息.公司 AS 客户名称, 产品.产品名称, 扩展订单明细.总价 AS 销售, 客户扩展信息.[省/市/自治区], 客户扩展信息.[国家/地区], 员工扩展信息.ID AS [员工 ID], 产品.ID AS [产品 ID], 客户扩展信息.ID AS [客户 ID], Year([订单日期]) AS 年份, Month([订单日期]) AS 月份, DatePart("q",[订单日期]) AS 季度, Format([订单日期],"mmmm") AS 月份名, 产品.类别, 产品.产品名称 AS 产品, 客户扩展信息.公司 AS 客户, (Month([订单日期]) Mod 3) AS 季度月
FROM ((((订单 INNER JOIN 客户扩展信息 ON 订单.[客户 ID] = 客户扩展信息.ID) LEFT JOIN 员工扩展信息 ON 订单.[员工 ID] = 员工扩展信息.ID) INNER JOIN 扩展订单明细 ON 订单.[订单 ID] = 扩展订单明细.[订单 ID]) INNER JOIN 产品 ON 扩展订单明细.[产品 ID] = 产品.ID) INNER JOIN 发票 ON 订单.[订单 ID] = 发票.[订单 ID];
```

显然这段语句过于复杂了，如果在 Foxtable 中用这段语句生成一个查询表，将没有办法正常实现分页加载和加载树等动能。

2、用 Access 打开这个数据库文件，然后定义一个查询，查询的名称为"销售分析"，查询的 SQL 语句就设置为上面这段 Select 语句。

3、在 Foxtable 增加一个名为"销售分析"的查询表，用 Select 语句定义：

```
Select * From {销售分析} Where 产品名称 Is null
```

提示：加上一个不可能成立的条件“产品名称 Is Null”，使得这个查询表默认不会加载任何数据。

```
DataTables("销售分析").LoadOver = "订单ID"
Tables("销售分析").OpenLoadTree("订单日期 YM",150,30,True)
```

这样打开项目之后，就会自动打开加载树，每页30行，并自动加载第一页数据。

提示：由于本例的查询表没有_Identify列，所以设置分页加载的时候，要用LoadLover属性明确指定分页依据列。

5、我们可以在任何地方使用这个后台查询，例如：

```
Tables("窗口1_Table1").Fill("Select * From {销售分析} Where 产品名称 = '葡萄干'","nwnd",True)
```

总结：

通过在后台数据库定义查询(SQL Server称之为视图)，将复杂的SQL语句当做一个普通的表来使用，让Foxtable只处理自己所擅长的部分，可以大大提高动态加载的兼容性。

4.14.1.3 加载其它表的数据

DataTable 的 Load 方法是从同一个后台数据表中加载数据的。
如果需要从从其它表加载数据，则需要使用 Fill 方法。
Fill 方法能够根据指定的 SQL 语句，从指定的数据源中加载数据。
语法：
Fill(SelectString, IsQuery)
或
Fill(SelectString, ConnectionName，IsQuery)
SelectString： 字符型，用于设置 SQL 语句。
ConnectionName： 字符型，指定数据源名称，如果不指定，表示使用内部数据源。
IsQuery： 逻辑型，设为 True，表示查询表，设为 False，表示数据表 。
数据表默认是可以编辑和保存的。
查询表默认不能编辑，通过将其 AllowEdit 属性设置为 True，可以编辑查询结果，但不能保存编辑结果。
示例一

```
DataTables("表 A").Fill("Select * From {订单}", False)
```

执行上述代码后，表 A 将显示订单表的数据，尽管其名称还是表 A。
如果将 IsQuery 参数设置为 True，将生成查询表：

```
DataTables("表 A").Fill("Select * From {订单}", True)
```

查询表的数据可以来自多个表，而且可以不包括主键列，但是查询表默认不能编辑，且不能保存编辑结果。
示例二
生成数据表时，也就是 IsQuery 参数设置为 False 时，如果只加载部分列，则必须将主键列包括在内，例如：

```
DataTables("订单").Fill("Select [_Identify], 产品, 客户, 数量, 日期 From {订单}", False)
```

示例三
如果要从外部数据源加载，务必指定外部数据源的名称。
例如:

```
DataTables("表 A").Fill("Select * From {订单}", "nwind", False)
```

上述代码将从名为 nwind 的数据源中加载订单表的数据到表 A。
有了 Fill 方法，我们甚至可以在设计系统的时候，用临时表代替数据表，在启动项目后，根据需要从不同的数据源和表中加载数据。
示例四
Select 语句可以动态合成，例如加载今天过生日的员工：

```
DataTables("表 A").Fill("Select * From {员工} Where Month([出生日期]) = " & Date.Today.Month & " And Day([出生日期]) = " & Date.Today.Day, False)
```

4.14.1.4 窗口中的 Table 控件

窗口中的 Table 控件有以下动态加载数据的方法。
使用 DataTable 的 Load 方法
如果是从同一个表中动态加载数据，可以使用 DataTable 的 Load 方法。

例如希望在某 SQLTable 显示今天的订单，可以在设计窗口的时候，将其 SQL 语句设置为：

```
Select * From {订单} Where [_Identify] Is Null
```

然后在窗口的 AfterLoad 事件加上代码：

```
Tables("窗口 1_Table1").DataTable.LoadFilter = "[日期] = #" & Date.Today & "#"
Tables("窗口 1_Table1").DataTable.Load
```

使用 Table 的 Fill 方法

如果是不同的表中动态加载数据，可以使用 Table 的 Fill 方法。

对于窗口中的 SQLQuery、SQLTable 型 Table，以及非副本的 Normal 型 Table，都可以使用 Fill 方法动态加载数据。

Table 的 Fill 方法和 DataTable 的语法类似：

语法

Fill(SelectString, IsQuery)

或者

Fill(SelectString, ConnectionName, IsQuery)

SelectString： Select 语句

ConnectionName：可选参数，指定数据源名称

IsQuery： 逻辑型，设为 True，将生成 SQLQuery 型 Table，否则生成 SQLTable 型 Table。

SQLTable 是数据表，默认是可以编辑和保存的。

SQLQuery 是查询表，默认不能编辑，通过将其 AllowEdit 属性设置为 True，可以编辑查询结果，但不能保存编辑结果。

示例

```
Tables("窗口 1_Table1").Fill("Select * From {客户}","nwnd",True)
```

不管 Table 原来是什么类型，显示的是什么数据，执行上述代码后，Table 将成为 SQLQuery 类型，显示 nwnd 数据源中的客户表数据。

需要特别注意的是,如果只想取部分列生成 SQLTable，那么必须包括主键列在内，例如：

```
Tables("窗口 1_Table1").Fill("Select [_Identify],[客户],[数量],[单价] From {订单}",False)
```

这里假定订单表是内部数据表，内部数据表的主键列是_Identify，如果是外部表，请使用实际的主键列名称。

如果要生成 SQLQuery，则没有这个限制：

```
Tables("窗口 1_Table1").Fill("Select [客户],[数量],[单价] From {订单}",True)
```

一定要注意，如果要生成 SQLTable，第二个参数要设置为 False，如果要生成 SQLQuery，第二个参数则要设置为 True，切不可搞混。

DataSource 属性

Table 有一个 DataSource 属性，可以通过此属性动态设置其绑定的表。

例如：

```
Tables("窗口 1_Table1").DataSource = DataTables("订单")
```

执行上述代码后，不管原来的 Table 是何种类型(非副本的 Normal 类型 Table 除外)，都将成为订单表的一个副本。

我们知道 Foxtable 是通过以下类型生成统计表或临时表的：

GroupTableBuilder

CrossTableBuilder

SQLGroupTableBuilder

SQLCrossTableBuilder

SQLJoinTableBuilder

DataTableBuilder

这些类型都有一个 BuildDataSource 方法，可以生成一个对象，此对象可以直接绑定到 SQLTable 和 SQLQuery 类型的 Table，以及副本型的 Table。

例如将某按钮的 Click 事件代码设为：

```
Dim b As New GroupTableBuilder("统计表 1",DataTables("订单"))
b.Groups.AddDef("日期","月份")
b.Totals.AddDef("数量")
Tables("窗口 1_Table1").DataSource = b.BuildDataSource()
```

那么单击此按钮，窗口中的 Table1 将显示各月销售数量。

提示：上面的代码中生成的统计表的名称并非"统计表 1"，而是"窗口 1_Table1"，也就是说 GroupTableBuilder 指定的表名是无效的，切记切记。

只有 SQLTable、SQLQuery 和作为副本的 Normal 型 Table，才可以使用 Table 类型的 Fill 方法和 DataSource 属性，对于非副本的 Normal 型 Table，只能使用 DataTable 的 Fill 和 Load 方法。

判断 Table 的类型

如果需要判断 Table 的类型，以及是否是副本，请参考:

TableType

IsCopy

4.14.1.5 运行中增加 Table 控件

4.14.1.6 善用 SQLJoinTableBuilder

前面提到，DataTable 和 Table 都有 Fill 方法，可以从另一个表加载数据。

Fill 使用的是 SQL 语句，如果遇到较为复杂的需求，例如同时从多个表中提取列加载，普通用户可能无法应付。

不过不用担心，有了 SQLJoinTableBuilder，不管多复杂的需求，都可以自动生成 SQL 语句供你调用。

假定有三个表：

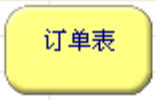
	产品ID	产品名称
1	P01	运动饮料
2	P02	温馨奶酪
3	P03	三合一麦片
4	P04	浓缩咖啡
5	P05	盐水鸭

产品表

	客户ID	客户名称
1	C01	红阳事业
2	C02	威航货运有限公司
3	C03	浩天旅行社
4	C04	立日股份有限公司
5	C05	福星制衣厂股份有限公司

客户表

	产品ID	客户ID	单价	折扣	数量	日期
1	P05	C03	17	0.1	690	2000-01-02
2	P02	C01	20	0.1	414	2000-01-03
3	P05	C04	17	0	332	2000-01-03
4	P01	C03	18	0.15	-9	2000-01-04
5	P03	C03	10	0	445	2000-01-04
6	P04	C04	17.6	0	246	2000-01-07



希望从提取产品表的产品名称列、客户表的客户名称列，订单表的所有列，并利用订单表的数量、单价、折扣计算出金额，加载到某窗口的 Table 控件中，
假定这三个表来自于一个名为"Sale"的外部数据源，代码为：

```
Dim jb As New SQLJoinTableBuilder("查询表1","订单")
jb.ConnectionName = "Sale"
jb.AddTable("订单","产品ID","产品","产品ID")
jb.AddTable("订单", "客户ID","客户","客户ID")
jb.AddCols("产品名称","客户名称","日期", "数量","单价")
jb.AddExp("金额","数量 * 单价 * (1-折扣)")
Tables("窗口1_Table1").Fill(jb.BuildSql,"Sale",True)
```

提示：由于数据来自于多个数据表，所以Fill的最后一个参数必须设置为True。

4.14.1.7 运行中创建数据源

外部数据源通常是事先定义好的，利用项目事件 BeforeConnectOuterDataSource，可以在打开项目的时候，动态地改变数据源。
但是很多时候，我们希望在运行过程中，动态地改变数据源，甚至让同一个表显示不同数据源中的数据。
关于如何动态创建数据源，请参考：
Connections

4.14.1.8 运行中切换数据源

DataTable 有一个方法 ChangeConnection，用于切换数据源 。
语法：
ChangeConnection(ConnName)
ConnName: 数据源名称
示例
例如我们希望订单表加载另一个数据源 test 中的数据，可以：

```
DataTables("订单").ChangeConnection("test")
```

一旦切换数据源，就会自动执行 DataTable 的 Load 方法，重新加载数据，你可以事先设置好加载条件：

```
DataTables("订单").LoadFilter = "[客户 ID] = 'TOMSP'"
DataTables("订单").ChangeConnection("test")
```

需要注意的是：如果要切换某个表的数据源，那么新的数据源必须和原来的数据源具有同样名称

的一个表，而且结构必须完全相同，否则会带来不可预料的错误。

4.14.1.9 父表子表同步加载

4.14.1.9.1 父表与子表同步加载之一

假定有两个表，分别是订单和订单明细，两者通过"订单 ID"列关联起来，订单是父表，订单明细是子表，因为每个订单对应有若干订单明细。
由于数据量比较大，决定采用动态加载，每次只加载一定数量的订单进行处理，显然作为一个合格的系统，应该同时加载这些订单的订单明细。
要完成这样的任务很简单，只需在订单表的 AfterLoad 事件中加入以下代码即可：

```
If DataTables(″订单″).DataRows.Count = 0 Then '如果订单表没有数据
        DataTables(″订单明细″).LoadFilter = ″订单 ID Is Null″ '不加载订单明细
Else
        Dim ids As String
        For Each dr As DataRow In DataTables(″订单″).DataRows
                ids = ids & ″,″ & dr(″订单 ID″)
        Next
        ids= ids.Trim(″,″)
        DataTables(″订单明细″).LoadFilter = ″订单 ID In (″ & ids & ″)″
End If
DataTables(″订单明细″).Load()
```

每次订单表加载数据后，都会触发 AfterLoad 事件，执行上述代码。
上述代码会自动扫描已加载订单的订单 ID，并合成订单明细表的加载条件，最后重新加载订单明细表。
假定订单表加载了 6 个订单，其订单 ID 分别为 1、2、6、7、8、9，显然上述代码为订单明细表合成的加载条件为：订单 ID In (1,2,6,7,8,9)，这样重新加载订单明细表时，将只加载这 6 个订单的订单明细。
可见代码的关键在于 In 运算符，在学习 SQL 语句的时候，我们会介绍此运算符。
上面的代码假定订单 ID 是整数型，如果订单 ID 是字符型，那么代码应该改为：

```
If DataTables("订单").DataRows.Count = 0 Then '如果订单表没有数据
   DataTables("订单明细").LoadFilter = "订单 ID Is Null" '不加载订单明细
Else
   Dim ids As String
   For Each dr As DataRow In DataTables("订单").DataRows
      ids = ids & ",'" & dr("订单 ID") & "'"
   Next
   ids= ids.Trim(",")
   DataTables("订单明细").LoadFilter = "订单 ID In (" & ids & ")"
End If
DataTables("订单明细").Load()
```

我们在编写表达式的时候，始终要记得字符用单引号括起来，而数值则可直接使用，帮助没有办法在每次编写此类代码的时候，分别为每种数据类型提供一个例子，希望大家自己多加留意。

示例

下面我们来一个具体的例子，订单表采用加载树加载，按年月每页加载 100 个订单，并同步加载这 100 个订单的订单明细：

设计步骤:

1、首先将 BeforeLoadInnerTable 事件代码设置为：

```
Select Case e.DataTableName
    Case "订单","订单明细"
        e.Filter = "[_Identify] Is Null"
End Select
```

确保订单表和订单明细表在初始状态不会加载任何数据。

2、然后在 AfterOpenProject 事件中加入代码：

```
Tables("订单").OpenLoadTree("订购日期 YM",120,100,True)
```

这样项目打开后，将自动显示订单表的加载树，每页加载 100 行，并自动加载第一页数据。

3、将订单表的 AfterLoad 事件设置为（假定订单 ID 是整数型）：

```
If DataTables("订单").DataRows.Count = 0 Then '如果订单表没有数据
   DataTables("订单明细").LoadFilter = "订单 ID Is Null" '不加载订单明细
Else
   Dim ids As String
   For Each dr As DataRow In DataTables("订单").DataRows
      ids = ids & "," & dr("订单 ID")
   Next
   ids= ids.Trim(",")
   DataTables("订单明细").LoadFilter = "订单 ID In (" & ids & ")"
End If
DataTables("订单明细").Load()
```

至此，我们这个关联加载系统已经完工，前后不过 16 行代码。

提示：

上面的例子是针对内部数据表的，如果使用的是外部数据源，那么第一步可以省略，只需在设置外部数据表的时候，将加载条件设置为：[_Identify] Is Null

4.14.1.9.2 父表与子表同步加载之二

上一节介绍的父表与子表同步加载的方法，对于多数用户来说，可能稍微复杂了一点，而且效率相对比较低，帮助之所以保留这种方法，主要还是为了让大家熟悉和演练一下 SQL 语句。

实际上，Foxtable 提供了非常简单高效的同步加载方法，你要做的只需在父表的 AfterLoad 事件中加上一行代码：

```
e.DataTable.LoadChildren("子表")
```

例如订单和订单明细通过订单编号建立关联，订单表动态加载，每次只加载一定数量的订单进行处理，为了每次加载订单后，能同步加载这些订单的订单明细，只需在订单表的 AfterLoad 事件加上代码：

```
e.DataTable.LoadChildren("订单明细")
```

是的，就是这么简单，因为你用的是 Foxtable。
LoadChildren 是 DataTable 的一个方法，用于加载子表数据，而且只加载那些父表已经存在对应行的子表数据。
语法
LoadChildren(子表名称)
LoadChildren(子表名称, 父表关联列名称, 子表关联列名称)
由 LoadChildren 的语法可以看出，如果父表和子表没有建立关联，我们可以通过指定关联列来实现同步加载，例如：

```
DataTables("订单").LoadChildren("订单明细", "订单编号", "订单编号")
```

并非只能通过父表加载子表数据，只要你愿意，也可以通过子表加载父表数据，语法：

```
DataTables("子表").LoadChildren("父表", "关联列", "关联列")
```

4.14.1.9.3 父表和子表同步加载之三

前面讲述的同步加载，都是在父表加载完成后，将父表已加载行的全部子行加载进来，这种方式很实用，符合大多数场合的需求。
如果父表中每一行，在子表中对应有很多子行，或父表一次加载的数据量比较大，也许此时你并不希望将子表对应的数据全部加载进来。
例如我们可以在选择某个订单进行处理的时候，才加载此订单对应的订单明细进来，避免一次加载子表过多的数据。
为此只需将订单表的 CurrentChanged 事件中加上代码(假定订单编号是整数型)：

```
If e.Table.Current IsNot Nothing Then
    DataTables("订单明细").LoadFilter = "订单编号 = " & e.Table.Current("订单编号")
Else
    DataTables("订单明细").LoadFilter = "[_Identify] is null"
End If
DataTables("订单明细").Load()
```

这样我们选择某行订单，就会自动加载对应的订单明细。
上面的代码也有不足，例如处理 A 订单的时候，会自动加载 A 订单的明细，当处理完 B 订单后再回到 A 订单，又得重新加载一次 A 订单的订单明细，显然这种重复加载是一种"浪费"。
我们可以再完善一下代码，避免数据重复加载：

```
If e.Table.Current IsNot Nothing Then
    Dim Filter As String = "订单编号 = " & e.Table.Current("订单编号")
    If  DataTables("订单明细").Find(Filter) Is Nothing Then '如果对应的订单明细没有加载过
      DataTables("订单明细").AppendLoad(Filter) '则追载此订单的订单明细
    End If
End If
```

这里采用 AppendLoad 来追载没有加载过的订单明细，避免了重复加载，效率远远好过第一个方案。

4.14.1.10 动态增加表

4.14.1.10.1 QueryBuilder

用于编码生成查询表。
属性：
TableName:　查询表名称
TableCaption:　查询表标题
SelectString:　SQL 查询语句
ConnectionName：指定数据源名称，如果不指定，则使用内部数据源。
需要特别注意的是：如果使用内部数据源，表名必须用大括号括起来，否则不能正常查询。
这是因为内部数据表的名称，并不是你看到的表名。
使用外部数据源则没有这个限制。
示例：

```
Dim q As new QueryBuilder
q.TableName = "销量统计"
q.SelectString = "Select  产品,Sum(数量) As  数量  From {订单} Group By  产品"
q.Build
MainTable = Tables("销量统计")
```

上面的代码将生成一个查询表，该查询表会统计出不同产品的销售数量。

4.14.1.10.2 OuterTableBuilder

用于编码生成外部数据表。
属性：
TableName:　外部数据表名称
TableCaption:　外部数据表标题
SelectString:　Select 语句
ConnectionName：指定数据源名称，如果不指定，则使用内部数据源。
OuterTableBuilder 的数据只能来自于一个表，因为它生成的表是可以编辑保存的。
示例
假定建立了一个外部数据源 Northwind，要生成一个名为订单的外部表，包括 1998 年全年的订单：

```
Dim q As new OuterTableBuilder
q.TableName = "订单"
q.TableCaption = "1998年订单"
q.ConnectionName = "Northwind"
q.SelectString = "Select * From {订单} Where Year(订购日期) = 1998 "
q.Build
```

4.14.1.10.3 SQLJoinTableBuilder

除 QueryBuilder 外，Foxtable 还提供了一个更加实用的 SQLJoinTableBuilder，无需 Select 语句，也能随心所欲地生成各种查询表。
请参考：查询表生成器

4.14.1.10.4 在窗口中动态创建 Table

利用 QueryBuilder 和 OutTableBuilder 动态生成的表，都会出现在主界面中，使用起来和常规的表无异。
如果要在窗口中动态增加 Table，则不能使用 QueryBuilder 和 OuterTableBuilder。
Form(窗口)提供了三个方法用来在窗口中动态创建 Table。
CreateSQLTable

用于创建 SQLTable 类型的 Table，语法：
CreateSQLTable(Name, SqlText , ConnectionName)
参数说明:
Name： 控件名称
SQLText： Select 语句
ConnectionName：数据源名称
CreateSQLQuery

用于创建 SQLQuery 类型的 Table，语法：
CreateSQLQuery(Name, SqlText , ConnectionName)
参数说明:
Name： 控件名称
SQLText： Select 语句
ConnectionName：数据源名称
CreateTable
用于创建 Normal 型 Table，语法:
CreateTable(Name, TableName, IsCopy)
Name： 控件名称
TableName： 数据表名称
IsCopy： 逻辑型，是否作为副本，设为 True 作为副本。
示例一
创建一个 SQLTable 型 Table 控件，从 nwind 数据源中加载产品表的数据：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateSQLTable("Table1", "Select * From {产品}","nwind")
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

示例二
创建一个 SQLQuery 型 Table 控件，加载内部数据源中的产品表：

```
Dim tbl As WinForm.Table
```

```
tbl = e.Form.CreateSQLQuery("Table1", "Select * From {产品}","")
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

示例三

创建一个 Normal 型 Table 控件，将订单表移到窗口中：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateTable("Table1", "订单", False)
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

示例四

创建订单表的副本：

```
Dim tbl As WinForm.Table
tbl = e.Form.CreateTable("Table1", "订单", True)
tbl.SetBounds(10, 10, 600, 400)
e.Form.AddControl(tbl)
```

4.14.1.11 动态加载数据总结

前面介绍了各种动态加载数据的方法，现在总结一下：

1、如果是从同一个表重新加载数据，那么使用 DataTable 的 Load 方法。

2、如果要从同一个表中追加新的数据到 DataTable，那么请使用 AppendLoad 方法。

3、如果要刷新 DataTable 中的部分数据，可以先用 RemoveFor 移除这些数据，然后用 AppendLoad 追载这些数据。

4、如果需要自行编写 Select 语句加载同一个表的数据，那么用 SQLLoad 方法。

5、如果需要从另一个表加载数据，那么用 Fill 方法。

6、DataTable 和 Table 都有 Fill 方法，对于标准的表，应该使用 DataTable 的 Fill 方法 ，对于窗口中的 SQLTable 和 SQLquery 型 Table，以及作为副本的 Normal 型 Table，应该使用 Table 的 Fill 方法。

7、如果需要从另一个数据源加载一个结构完全相同的表，那么用 DataTable 的 ChangeConnection 方法。

8、对于窗口中的 Table，通过设置 DataSource 属性，可以显示任何已经加载的表的数据。

9、如果要在主界面中增加查询表，可以使用 QueryBuilder，如果要在主界面中增加外部数据表，可以使用 OuterTableBuilder。

10、如果要在窗口中动态增加 Table，需要使用 Form 的 CreateSQLTable、CreateSQLQuery、CreateTable 方法。

11、如果希望同步加载子表数据，可以用 LoadChildren 方法。

4.14.2 处理后台数据

4.14.2.1 关于后台数据的处理

有了动态加载，可以根据需要只加载少量数据进行处理，并随时可以从后台重新加载另一批数据，这样我们面对海量数据的时候，才能应付自如。
但是很多时候后，我们需要处理所有数据，包括未加载的后台数据，例如统计和查询。
假定有一个订单管理系统，有 1000 万个订单，显然为了统计总销售数量，直接将 1000 万个订单全部加载，是很不现实的。
多数时候，后台的数据的统计、查询和处理，可以使用 SQL 语句来完成，后面会有专门的一章对此进行介绍。

但是熟练掌握 SQL 语句，对于多数用户来说可能有些困难，Foxtable 作为一个面向普通用户的高效开发工具，也不应该如此要求用户。
所以 Foxtable 针对动态加载做了大量的工作，不仅所有的统计都可以直接针对后台数据，还可以通过菜单直接对后台数据进行查询筛选，大大降低了动态加载对整体数据处理的影响。
不仅如此，DataTable 还新增了一系列的方法，专门用于对后台数据进行查询、计算和更新，接下来我们一一介绍。

4.14.2.2 SQLCompute

DataTable 的 SQLCompute 方法和 Compute 方法类似，不过 Compute 用于计算已经加载的数据，而 SQLCompute 用于计算后台所有数据。
语法：
SQLCompute(Expression, Filter)
Expression: 要计算的表达式，使用聚合函数进行统计。
Filter： 可选参数，用于设置计算条件，采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。
示例一

统计产品PD01的销售数量：

```
Dim Val As Integer
Val = DataTables("订单").SQLCompute("Sum(数量)","产品 = 'PD01'")
Output.Show(Val)
```

不管PD01的订单是否加载，上面的代码都能得出正确的结果。

示例二

统计雇员EP01成交的订购数量超过500的订单数：

```
Dim cnt As Integer
cnt = DataTables("订单").SQLCompute("Count(*)", "雇员 = 'EP01' And 数量 > 500")
Output.Show("订单数:" & cnt)
```

示例三

和Compute不同，SQLCompute不能统计表达式列，因为表达式列在后台数据库并不存在。
不过没关系，因为SQLCompute可以直接对表达式进行统计，例如统计产品PD01的销售金额：

```
Dim Val As Double
Val = DataTables("订单").SQLCompute("Sum(数量 * 单价 * (1 - 折扣))","产品 = 'PD01'")
Output.Show(Val)
```

4.14.2.3 SQLReplaceFor

DataTable 的 SQLReplaceFor 方法和 ReplaceFor 有些类似，不过 ReplaceFor 用于批量更新已经加载的数据，而 SQLReplaceFor 用于批量更新后台数据。
语法：
SQLReplaceFor(DataColName,Value,Filter,IsExpression)
DataColName： 要更新的列
Value： 更新值或表达式
Filter： 更新条件，采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。
IsExpression：逻辑型，设置为 True，表示 Value 参数不是一个值，而是一个表达式。
SQLReplaceFor 会返回一个整数，表示合计有多少行的值被替换了。
提示：
1、SQLReplaceFor 的操作是即时生效的，且无法撤销，所以在编写代码的时候，务必要先用测试数据进行验证，或者做好备份，以免对数据代码造成无法挽回的影响。
2、SQLReplaceFor 执行后，已经加载到 DataTable 的数据并不会受影响，还是保持原值不变，直到重新加载或重新打开项目。
示例一
将产品为 PD01 的订单的折扣全部改为 0.05：

```
Dim cnt As Integer
cnt = DataTables("订单").SQLReplaceFor("折扣", 0.05, "产品 = 'PD01'")
Output.Show("合计更新了" & cnt & "条记录的折扣!")
```

示例二

和ReplaceFor不同，SQLReplaceFor的更新值可以是一个表达式，例如将数量大于800的订单折扣，

在原基础上增加0.05：

```
Dim cnt As Integer
cnt = DataTables("订单").SQLReplaceFor("折扣", "折扣 + 0.05", "数量 > 800",True)
Output.Show("合计更新了" & cnt & "条记录的折扣!")
```

提示：如果更新值是一个表达式，请将第四个参数设置为True。

4.14.2.4 SQLDeleteFor

DataTable 的 SQLDeleteFor 方法和 DeleteFor 有些类似，不过 DeleteFor 用于批量删除已经加载的数据，而 SQLDeleteFor 用于批量删除后台数据。
语法：
SQLDeleteFor(Filter)
Filter： 删除条件，采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。

SQLDeleteFor 执行完毕后，会返回一个整数，表示合计删除了多少行。
SQLDeleteFor 的删除是即时生效的，且无法撤销，所以在编写代码的时候，务必要先用测试数据进行验证，或者做好备份，以免对数据代码造成无法挽回的影响。
示例一
删除产品 PD01 的全部订单：

```
Dim cnt As Integer
cnt = DataTables("订单").SQLDeleteFor("产品 = 'PD01'")
Output.Show("合计删除了" & cnt & "条记录!")
```

示例二

在DataTable中一次性删除大量的行，然后保存，效率是比较低的。
我们可以用SQLDeleteFor直接从后台删除符合条件的行，然后用RemoveFor从DataTable移除符合条件的行，效率会提高很多，例如：

```
DataTables("表A").SQLDeleteFor("产品编号 = 1")
DataTables("表A").RemoveFor("产品编号 = 1")
```

4.14.2.5 SQLFind

SQLFind 和 Find 类似，用与查找符合条件的行，如果找到的话，返回找到的行，否则返回 Nothing。不过 Find 是在 DataTable 查找符合条件的行，而 SQLFind 是在后台数据库中查找。

语法：

SQLFind(Filter,Sort,Index)

Filter: 条件表达式，采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。

Sort: 可选参数，指定排序方式。

Index: 可选参数，指定返回第几个符合条件的行，0 表示第一行。

示例一：

```
Dim dr As DataRow
dr = DataTables("产品").SQLFind("产品编号 = '03'") '找出编号为 03 的产品
With DataTables("订单")
    dr = .SQLFind("产品 = 'PD01'","日期") '找出第一次订购 PD01 产品的记录
    dr = .SQLFind("产品 = 'PD01'","日期",1) '找出第二次订购 PD01 产品的记录
End With
```

示例二：

有的时候，我们需要查找倒数第几行数据，例如最近一次订购某产品的记录。

可以参考下面的代码：

```
Dim dr As DataRow
With DataTables("订单")
    dr = .SQLFind("产品 = 'PD01'","日期 Desc") '找出最后一次订购 PD01 产品的记录
    dr = .SQLFind("产品 = 'PD01'","日期 Desc",1) '找出倒数第二次订购 PD01 产品的记录
End With
```

可以看到代码和原来几乎一样，唯一的变化是排序参数，被改为：日期 Desc

加上 DESC 使得日期按照降序排序，最后的日期排在最前面，我们所找出的第一条记录，就是最后一次订购产品 PD01 的记录。

示例三

通常应该在代码中判断是否找到了符合条件的行，然后再运行后续的代码。

例如要找出最近一次订购产品数量超过 1000 的订单，并显示订单的日期和客户：

```
Dim dr As DataRow
dr = DataTables("订单").SQLFind("数量 > 1000","日期 Desc")
If dr IsNot Nothing Then '如果找到的话
    Output.Show("日期:" & dr("日期"))
    Output.Show("客户:" & dr("客户"))
End If
```

如果我们不加上判断，直接：

```
Dim dr As DataRow
dr = DataTables("订单").SQLFind("数量 > 1000","日期 Desc")
Output.Show("日期:" & dr("日期"))
Output.Show("客户:" & dr("客户"))
```

一旦订单表并不存在订购数量超过 1000 的订单，那么 Find 方法返回 Nothing，导致后续代码运行出错。

示例四

可以修改并保存 SQLFind 查找出来的行，例如：

```
Dim dr As DataRow
dr = DataTables("订单").SQLFind("数量 > 1000","日期 Desc")
If dr IsNot Nothing Then '如果找到符合条件的行
    dr("折扣") = 0.14
    dr.Save()
End If
```

SQLFind 每次只能找出一条符合条件的行，如果要同时找出所有符合条件的行，可以使用 SQLSelect 方法。

4.14.2.6 SQLSelect

DataTable 的 SQLSelect 方法和 Select 方法类似，不过 Select 用于从已经加载的数据中查询数据，而 SQLSelect 用于从后台数据中查询数据。
语法：
SQLSelect(Filter,Top,Sort)
Filter： 查询条件，采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。
Top： 可选参数，指定返回的行数
Sort： 可选参数，指定查询顺序，和 Top 配合使用。
SQLSelect 会返回一个集合，包括所有符合条件的行。

示例一
从后台查询出产品 PD01 的订单，并累计出数量：

```
Dim drs As List(of DataRow)  = DataTables("订单").SQLSelect("产品 = 'PD01'")
Dim Sum As Integer
For Each dr As DataRow In drs
    Sum = Sum + dr("数量")
Next
Output.Show(Sum)
```

提示：这只是一个介绍SQLSelect用法的例子，你真要计算后台数据，请使用SQLCompute方法，会简洁高效很多。

示例二

如果是数据表，可以修改通过SQLSelect筛选出来的行，并保存之：

```
Dim drs As List(of DataRow) = DataTables("订单").SQLSelect("产品 = 'PD01'")
For Each dr As DataRow In drs
    dr("折扣") = 0.1
```

```
Next
DataTables("订单").SQLUpdate(drs)
```

SQLUpdate专门用于批量保存通过SQLSelect筛选出来的行。

提示：这只是一个介绍SQLSelect用法的例子，你真要更新后台数据，请使用SQLReplaceFor方法，会简洁高效很多。

示例三

通过Top和Sort参数配合使用，我们可以返回指定顺序的前n行。
例如按日期顺序，返回产品PD01的前10个订单：

```
Dim drs As List(of DataRow)
drs = DataTables("订单").SQLSelect("产品 = 'PD02'",10, "日期 Desc")
```

示例四

如果需要按顺序返回所有行，可以将Top参数设置为""，例如按日期顺序，返回所有产品为PD01的订单：

```
Dim drs As List(of DataRow)
drs = DataTables("订单").SQLSelect("产品 = 'PD02'","", "日期 Desc")
```

4.14.2.7 SQLUpdate

SQLUpdate 用于批量保存通过 SQLSelect 筛选出来的行。

示例

```
Dim drs As List(of DataRow) = DataTables("订单").SQLSelect("产品 = 'PD01'")
For Each dr As DataRow In drs
    dr("折扣") = 0.1
Next
DataTables("订单").SQLUpdate(drs)
```

提示：
1、上面这种简单的更新，直接使用 SQLReplaceFor 方法，会简洁高效很多。
2、只有更新逻辑比较复杂的时候，才可以考虑用 SQLSelect 和 SQLUpdate。

4.14.2.8 SQLGetValues

DataTable 的 SQLGetValues 方法和 GetValues 方法类似，都是从指定列中，获取不重复的值，以集合的形式返回。
不过 GetValues 方法是从已经加载的数据中提取，而 SQLGetValues 是从后台所有数据中提取。

SQLGetValues(ColumnName,Filter,Sort)
ColumnName：列名称，从此列中提取不重复的值。
Filter:　可选参数，指定一个条件表达式，只返回符合此条件的值
Sort:　　可选参数，指定排序列，如果省略，则根据取值列排序，通常无需设置。
提示，Filter 参数采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。

示例一

列出产品表中所有的产品名称：

```
Dim Products As List(Of String)
Products = DataTables("产品").SQLGetValues("产品名称")
For Each Product As String In Products
    Output.Show(Product)
Next
```

示例二

返回的值默认按照取值列排序，我们可以另外指定排序列；例如按产品编号顺序，列出产品表中所有的产品名称：

```
Dim Products As List(Of String)
Products = DataTables("产品").SQLGetValues("产品名称", "", "产品编号")
For Each Product As String In Products
    Output.Show(Product)
Next
```

示例三

给排序列加上 DESC 关键词，返回值可以降序排序，例如按最近一次订货的日期顺序，列出订单表中的客户名单：

```
Dim Customers As List(Of String)
Customers = DataTables("订单").SQLGetValues("客户", "", "日期 Desc")
For Each Customer As String In Customers
    Output.Show(Customer)
Next
```

示例四

可以设置取值条件，例如按单间顺序列出产品表中单价大于 100 的产品：

```
Dim Products As List(Of String)
Products = DataTables("产品").SQLGetValues("产品名称", "单价 > 100", "单价")
For Each Product As String In Products
    Output.Show(Product)
Next
```

4.14.2.9 SQLGetComboListString

SQLGetComboListString 和 GetComboListString 类似， 都是从指定的列中提取不重复的值，用符号"|"将这些值连接成一个字符串，并返回这个字符串。
不过 GetComboListString 方法是从已经加载的数据中提取，而 SQLGetComboListString 是从后台所有数据中提取。

语法：

SQLGetComboListString(ColumnName，Filter, Sort)
ColumnName：列名称，从此列中提取不重复的值。
Filter: 可选参数，指定一个条件表达式，只返回符合此条件的值。
Sort: 可选参数，指定排序列，如果省略，则根据取值列排序，通常无需设置。
提示，Filter 参数采用的是 SQL 语法，和常规表达式的语法有所不同，请参考：SQL 表达式的语法差异。

示例一
列出订单表所有的客户：

```
Dim s As String  = DataTables("订单").SQLGetComboListString("客户")
output.show(s)
```

示例二
返回的值默认按取值列排序，可以另外指定排序列，例如按产品编号顺序，列出产品表中所有的产品名称：

```
Dim s As String = DataTables("产品").SQLGetComboListString("产品名称","","产品编号")
output.show(s)
```

示例三
可以给排序列加上关键词，例如按最近一次订货的日期顺序，列出订单表中的客户名单：

```
Dim s As String  = DataTables("订单").SQLGetComboListString("客户","","日期 Desc")
Output.Show(s)
```

实例四
可以设置取值条件，例如从客户表中提取华北地区的客户名单：

```
Dim s As String
s = DataTables("客户").SQLGetComboListString("客户名称","[地区] = '华北'")
Output.Show(s)
```

示例五
可以同时设置取值条件和排序列，例如按日期顺序，列出 2012 年 6 月 1 日后订购过 PD01 产品的客户名单：

```
Dim s As String
```

```
s = DataTables("订单").SQLGetComboListString("客户","[产品]= 'PD01' And [日期] > #6/1/2012#","日期")
Output.Show(s)
```

4.14.2.10 SQLAddNew

SQLAddNew 和 AddNew 类似，用于增加行。

不同的是，SQLAddNew 增加的行，并不会出现在 DataTable 中。

示例

直接在后台的订单表增加一行:

```
Dim dr As DataRow = DataTables("订单").SQLAddNew()
dr("产品") = "PD01"
dr("客户") = "CS01"
dr("雇员") = "EP01"
dr("数量") = 100
dr("单价") = 12
dr("折扣") = 0.1
dr("日期") = Date.Today
dr.Save()
```

执行上述代码后，你会发现前端的订单表并没有出现新增加的行，重新打开项目或重新加载订单表后，可以看到新增的行。

需要注意的是，只有执行 Save 方法，SQLAddNew 增加的行才会真正写入后台数据库，在这之前，行只是一个临时的存在。

SQLAddNew 的缺点：

1、SQLAddNew 只能用于外部表，不能用于内部表。

2、用 SQlAddNew 新增的行，保存后将不再可用，如果一个表的主键是自增的，而且你想保存行后能获取行的主键，或者你希望保存行后还能继续使用此行，那么就不能用 SQLAddNew。

我们可以用 DAddNew 方法实现和 SQLAddNew 相似的功能：

```
Dim dr As DataRow = DataTables("表 A").AddNew()
'处理代码
dr.Save()
dr.Remove
'新增行可以继续使用，可以在这里获取新增行的自增主键。
```

4.14.2.11 一个综合实例

本节的内容，可以参考 CaseStudy 目录下的文件：自动输入后台数据.Table

在学习事件编程的时候，我们接触到了自动输入的例子。

这个例子有一个问题，就是必须将行政区域表的所有数据加载到 Foxtable，上述代码才能完成自动输入的任务，由于中国的县级行政区域不过 2000 多个，所以这不算大的问题。
如果基础数据表不是行政区域，而是一个有几万行的表，那么仅仅为了实现自动输入而将整个表加载到 Foxtable 中是得不偿失的，如果有几十万甚至过百万行的数据，那么加载整个表根本就是不现实的。
所以本节我们要学习在不加载基础数据的情况下，如何实现自动输入。
在后面的《SQL 相关》会讲述一个和本节基本相同的例子，那是个完全用 SQL 语句实现的例子，参考：用 SQL 语句实现自动输入
本节不用 SQL 语句，采用前面介绍的 DataTable 的后台数据处理方法来实现同样的功能，且同样不加载行政区域表的数据，你会看到 本节的代码要简洁很多。
设计步骤：
1、在 BeforeLoadInnerTable 中设置代码：

```
If e.DataTableName = "行政区域" Then
    e.Filter = "[_Identify] Is Null"
End If
```

这个项目中的行政区域表是内部数据表，上述的代码使得打开项目后，行政区域表不会加载任何数据。
如果是外部表，参考: BeforeLoadOuterTable
2、省市列的列表项目是固定的，在项目事件 AfterOpenProject 加入代码：

```
Tables("客户").Cols("省市").Combolist = DataTables("行政区域").SQLGetComboListString("省市")
' Tables("行政区域").Visible = True '实际使用的时候,可以启用此行代码,以隐藏行政区域表
```

这样打开项目后，会从后台的行政区域表提取不重复的省市值作为客户表省市列的列表项目，尽管行政区域表并没有加载数据。

3、在客户表的PrepareEdit事件中加入代码：

```
If e.IsFocusCell Then '如果是焦点单元格
    If e.Col.Name = "县市" Then '如果正在编辑的是县市列
        e.Col.Combolist = DataTables("行政区域").SQLGetComboListString("县市", "[省市] = '" & e.Row("省市") & "'")
    End If
End If
```

这样在编辑客户表的县市列之前，会从后台的行政区域表提取该省市的县市值，作为县市列的列表项目使用。

4、最后将客户表DataColChanged事件的代码设置为：

```
If e.DataCol.Name = "省市" OrElse e.DataCol.Name = "县市" Then
    Dim dr As DataRow = e.DataRow
    Dim sr As DataRow
```

```
    sr = DataTables("行政区域").SQLFind("[省市] = '" & dr("省市") & "' And [县市] = '" & dr("县市") & "'")
'在后台查找符合条件的行
    If sr IsNot Nothing Then '如果找到了符合条件的行
        dr("区号") = sr("区号")
        dr("邮编") = sr("邮编")
    Else
        dr("区号") = Nothing
        dr("邮编") = Nothing
    End If
End If
```

这样一旦在客户表的某行输入省市和县市两列内容后，会从后台的行政区域表查找对应的行，如果找到，则将该行的区号和邮编写入客户表中，否则清除区号和邮编。

从这个例子可以看出，采用 Foxtable 内置的处理后台数据的方法，和直接编写 SQL 语句相比，代码要简洁得多，也不需要改变原来的编程习惯。

另一种执行方式

DataTable 的所有后台处理方法，当然都要基于 DataTable 运行，所以上面的示例中，我们加载了一个空的行政区域表，然后基于此空表处理后台的行政区域表的数据。

虽然我们可以通过代码在 Foxtale 的主界面中隐藏这个空表，但还是有一定程度的不便，而且有较多数量只需后台处理不需要加载数据的表时，还会加大系统对资源的消耗。

如果愿意，我们可以考虑另一种方式，通过 SQL 语句生成一个临时表，基于这个临时表执行后台数据处理方法。

首先我们在全局代码中定义一个变量：

```
Public xzqy As DataTable
```

然后在 AfterOpenProject 事件中加上代码：

```
Dim cmd As New SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "SELECT * From {行政区域} Where [省市] Is Null"
xzqy = cmd.ExecuteReader
```

上面的代码生成一个空的临时表，保存在全局变量 xzqy 中，这个临时表对应后台的行政区域表。

现在我们就可以通过此临时表处理后台的行政区域表的数据了，例如：

```
Tables("客户").Cols("省市").Combolist = xzqy.SQLGetComboListString("省市")
```

需要注意，如果不单单是调用后台数据，还要向后台写入数据，生成临时表的时候，必须给 ExecuteReader 加上参数 True，例如：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
```

```
cmd.CommandText = "Select * From {表名} Where [_Identify] Is Null"
Dim dt As DataTable = cmd.ExecuteReader(True)
```

当然表必须有主键，才能更新其数据。

关于这种执行方式的更全面的例子，可以参考：自定义用户管理之二

4.14.2.12 分页加载与流水账

本节的内容，可以参考 CaseStudy 目录下的文件：分页流水账.Table
关于流水账的设计，请参考：高效率的流水账
由于分页加载往往意味着大数据量、网络环境、多用户等元素，所以上述的流水账设计方案将不再适用，需要重新 设计。
好消息是分页加载同时也意味着每次加载的数据量有限，我们可以充分利用这个优势，在每次加载数据后重算余额，其实这样的设计更加符合实际应用的需求。
示例一
同样假定有个简单的流水帐式的表格，希望在分页加载的情况下，输入收入和支出，能够自动计算出余额:

	日期	收入	支出	余额
1	1999-02-01	1		1
2	1999-02-03	7	5	3
3	1999-02-05	2		5
4	1999-02-07	4	6	3
5	1999-02-09	8		11
6	1999-02-11	4		15
7	1999-02-13	6	4	17
8	1999-02-15	3		20
9	1999-02-17	9	6	23
10	1999-02-19	4		27

设计步骤：
1、删除原来的余额列，然后在项目事件 Initialize 中设置代码:

```
DataTables("例子一").DataCols.Add("余额",Gettype(Double))  '添加余额列,用于动态显示余额
```

技巧：
A、由于余额是每次加载数据后动态计算得出的，所以这里应该采用表达式列，而不是数据列；这样不仅可以节省存储空间，更重要的是，不会因为重算余额导致系统误认为此行被修改了，需要保存。
B、当然这个表达式列和我们平时的表达式列不同，因为没有给它设置表达式；没有设置表达式的表达式列，可以用代码向该列赋值，希望大家能记住这一点。
C、也许有人会问，为什么要在 Initialize 事件，而不是 AfterOpenProject 事件添加列？这是因为在 Initialize 事件添加的列，可以保存列宽和列位置。

2、将 DataColChanged 事件设为:

```
Select Case e.DataCol.Name
    Case "收入","支出"
        Dim dr As DataRow
        Dim drs As List(of DataRow)
        dr = e.DataTable.Find("[_SortKey] < " & e.DataRow("_SortKey"), "[_SortKey] Desc") '找出上一行
        If dr Is Nothing Then '如果没有找到上一行,说明本行就是第一行
            '计算之前的结存
            Dim jk As Double
            Dim ck As Double
            jk = DataTables("例子一").SQLCompute("Sum(收入)","[_SortKey] < " & e.DataRow("_SortKey"))
            ck = DataTables("例子一").SQLCompute("Sum(支出)","[_SortKey] < " & e.DataRow("_SortKey"))
            e.DataRow("余额") = e.DataRow("收入") - e.DataRow("支出") + jk -ck
            dr = e.DataRow
        End If
        drs = e.DataTable.Select("[_SortKey] >= " & dr("_SortKey"), "[_SortKey]")
        For i As Integer = 1 To drs.Count - 1 '重算余下行的余额
            drs(i)("余额") = drs(i-1)("余额") + drs(i)("收入") - drs(i)("支出")
        Next
End Select
```

上述代码的原理：

A、如果修改的是第一行，那么就先用 SQLCompute 方法，从后台计算出此行之前的结存，第一行的收入减去支出，再加上之前的结存，就是第一行的余额。
B、如果不是第一行数据，那么该行的收入减去支出，再加上上一行的结存，就是此行的余额。
可以看出，由于有了 SQLCompute 等后台数据处理行数，设计这么一个分页加载的流水账，变得轻松起来了。
3、表事件 AfterMoveRow 的代码保持不变：

```
Dim Key As Decimal
Dim Index As Integer
Dim dc As DataCol
Index = Math.Min(e.OldIndex, e.NewIndex)
Key = e.Table.Rows(Index)("_SortKey")
dc = e.Table.DataTable.DataCols("入库")
dc.RaiseDataColChanged("[_SortKey] >= " & Key)
```

4、表事件 DataRowDeleting 的代码同样保持不变：

```
e.DataRow("收入") = 0
e.DataRow("支出") = 0
```

5、设置表事件 AfterLoad 的代码为：

```
If e.DataTable.DataRows.count > 0 Then
    Dim dr As DataRow = DataTables("例子一").Find("","[_SortKey]")
    e.DataTable.DataCols("支出").RaiseDataColChanged(dr)
```

```
End If
```

这段代码在每次加载数据后执行，模拟拟第一行的支出发生变化，刷新已加载行的余额，这是非常关键的一步。

6、最后 AfterOpenproject 事件代码设置为：

```
With DataTables("例子一")
    .LoadOver = "_SortKey" '设置分页加载依据列
    .LoadFilter = "" '清除加载条件
End With
Forms("窗口1").Open() '打开设计好的分页加载窗口
```

提示：由于我们的流水账是根据_SortKey的顺序计算的，所以分页加载的语句列必须设置为_SortKey列。

示例二

对于下面这中区分产品的分页加载流水账(参考示例文件中的例子二)：

	日期	产品	入库	出库	库存
1	1999-02-01	PD01	1		1
2	1999-02-03	PD02	7	5	2
3	1999-02-05	PD03	2	2	0
4	1999-02-07	PD01	4	3	2
5	1999-02-09	PD02	8		10
6	1999-02-11	PD03	4		4
7	1999-02-13	PD01	6	4	4
8	1999-02-15	PD02	3		13
9	1999-02-17	PD03	9	6	7
10	1999-02-19	PD01	4		8
11	1999-02-17	PD02	7	4	16

只需将 DataColChanged 事件代码改为：

```
Select Case e.DataCol.Name
    Case "产品","入库","出库"
        Dim dr As DataRow
        Dim mr As DataRow = e.DataRow
        Dim drs As List(of DataRow)
        Dim jk As Integer
        Dim ck As Integer
        dr = e.DataTable.Find("[_SortKey] < " & mr("_SortKey") & " And [产品] = '" & mr("产品") & "'", "[_SortKey] Desc")
        If dr Is Nothing Then '如果是第一行,计算结存
            jk = DataTables("例子二").SQLCompute("Sum(入库)","[_SortKey] < " & mr("_SortKey") & " And 产品 = '" & mr("产品") & "'")
            ck = DataTables("例子二").SQLCompute("Sum(出库)","[_SortKey] < " & mr("_SortKey")
```

```
& " And 产品 = '" & mr("产品") & "'")
        mr("库存") = e.DataRow("入库") - e.DataRow("出库") + jk - ck
        dr = mr
      End If
      drs = e.DataTable.Select("[_SortKey] >= " & dr("_SortKey") & " And [产品] = '" & dr("产品") & "'",
"[_SortKey]")
      For i As Integer = 1 To drs.Count - 1
        drs(i)("库存") = drs(i-1)("库存") + drs(i)("入库") - drs(i)("出库")
      Next
      If e.DataCol.Name = "产品" AndAlso e.OldValue IsNot Nothing AndAlso e.OldValue <> e.NewValue
Then '如果修改的是产品列
        dr = e.DataTable.Find("[_SortKey] < " & mr("_SortKey") & " And [产品] = '" & e.OldValue & "'",
"[_SortKey] Desc") '找出旧产品的上一行
        If dr Is Nothing Then '如果不存在上一行,那么本行就是旧产品的第一行,由于已经被修改为新
产品,需要找出旧产品现在的第一行
          dr = e.DataTable.Find("[产品] = '" & e.OldValue & "'", "[_SortKey]") '找出旧产品现在的第一行
          If dr IsNot Nothing Then '如果找到,计算库存
            jk = DataTables("例子二").SQLCompute("Sum(入库)","[_SortKey] < " & mr("_SortKey") & "
And 产品 = '" & e.OldValue & "'")
            ck = DataTables("例子二").SQLCompute("Sum(出库)","[_SortKey] < " &
mr("_SortKey") & " And 产品 = '" & e.OldValue & "'")
            dr("库存") = dr("入库") - dr("出库") + jk - ck
          End If
        End If
        If dr IsNot Nothing Then
          drs = e.DataTable.Select("[_SortKey] >= " & dr("_SortKey") & " And [产品] = '" & dr("产品") &
"'", "[_SortKey]")
          For i As Integer = 1 To drs.Count - 1
            drs(i)("库存") = drs(i-1)("库存") + drs(i)("入库") - drs(i)("出库")
          Next
        End If
      End If
End Select
```

表事件 AfterMoveRow 的代码保持不变：

```
Dim Key As Decimal
Dim Index As Integer
Dim Filter As String
Dim r As Row
Index = Math.Min(e.OldIndex, e.NewIndex)
Key = e.Table.Rows(Index)("_SortKey")
r = e.Table.Rows(e.NewIndex)
Filter = "[_SortKey] >= " & Key & " And [产品] = '" & r("产品") & "'"
e.Table.DataTable.DataCols("入库").RaiseDataColChanged(Filter)
```

表事件 DataRowDeleting 的代码保持不变：

```
e.DataRow("入库") = 0
e.DataRow("出库") = 0
```

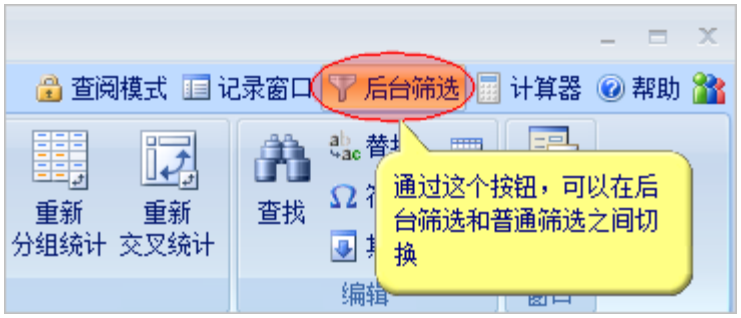
表事件AfterLoad的代码有变化：

```
Dim drs As New List(of DataRow)
For Each nm As String In e.DataTable.GetValues("产品") '找出每个产品的第一行数据，添加到集合drs中
    drs.Add(e.DataTable.Find("产品  = '" & nm & "'", "[_SortKey]"))
Next
For Each r As DataRow In drs
    e.DataTable.DataCols("入库").RaiseDataColChanged(r)
Next
```

这个代码在数据加载后，重置每个产品第一行的入库列，刷新每个产品的库存，重置某产品的第一行，等于该产品所有行的库存都会刷新，所以没有必要重置所有行的。

4.14.2.13 筛选后台数据

如果设置了分页加载数据，当我们通过菜单进行筛选的时候，默认只是对当前页的数据进行筛选，不会筛选出所有符合条件的行，显然这不够方便。
从 2014 版的 Foxtbale 开始，在分页加载的时候，我们依然可以对所有数据进行筛选，要做的只是按下配置栏中的“后台筛选”按钮，然后正常筛选数据就行：



提示：只有在分页加载的时候，才能按下这个按钮。
有了后台筛选，我们可以在只加载少量数据的时候，直接对所有数据进行筛选，感觉就像所有数据都已经加载了一样：



提示：

1、后台筛选默认是关闭的，需要通过按下配置栏中的“后台筛选”按钮来启动，如果希望某表默认开启后台筛选，可以在表属性设置中，将“分页后台筛选”属性设置为 True。

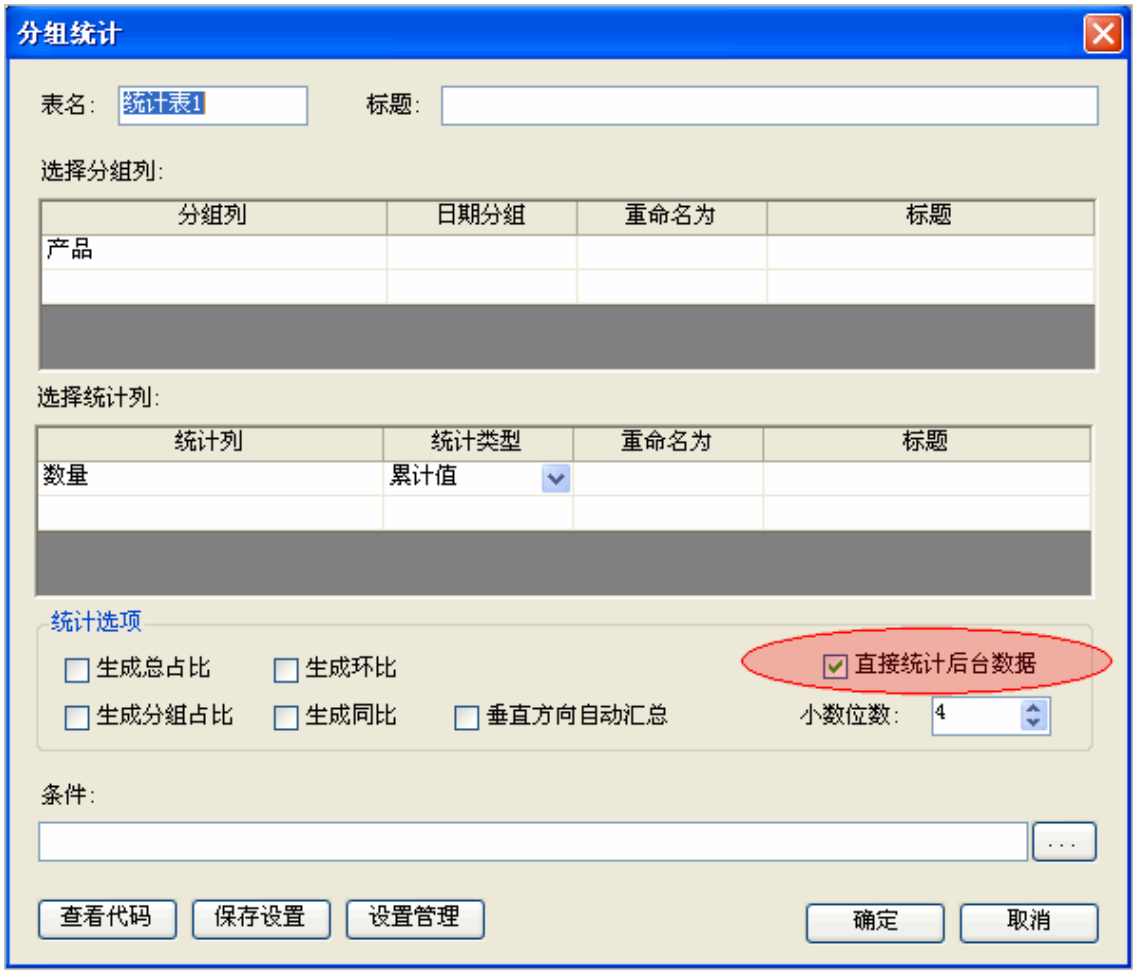
2、菜单中的所有筛选，都支持后台筛选，只有筛选树例外，因为如果用筛选树进行后台筛选，不如直接用加载树算了。

3、关联表不能进行后台筛选，只能进行普通筛选。

4.14.2.14 统计后台数据

Foxtable 的所有统计工具都可以对后台数据进行统计:

1、菜单中的交叉统计和分组统计，默认只统计已加载数据，如果选择下图所示“直接统计后台数据”的选项，即可统计后台所有数据。



2、如果使用代码进行分组统计或交叉统计，可以将 GroupTableBuilder 或 CrossTableBuilder 的 FromServer 属性设为 True，即可直接统计后台所有数据。

3、此外 Foxtable 还提供了 SQLGroupTableBuilder 和 SQLCrossTableBuilder 两个专业统计工具，专门用于后台数据统计和跨表数据统计。

4、如果打开了加载树并开启了分页加载，那么后台统计并不会统计所有数据，而是统计选定节点下的所有数据，不管这些数据是否已经加载；如果要统计所有数据，请先选择节点"加载所有行"。

4.14.2.15 分页加载与数据统计

在显示加载树并开启分页加载功能后，通过菜单进行分组统计或交叉统计时，需要区分三种情况：

1、统计选定节点下的所有数据

如果满足以下三个条件：

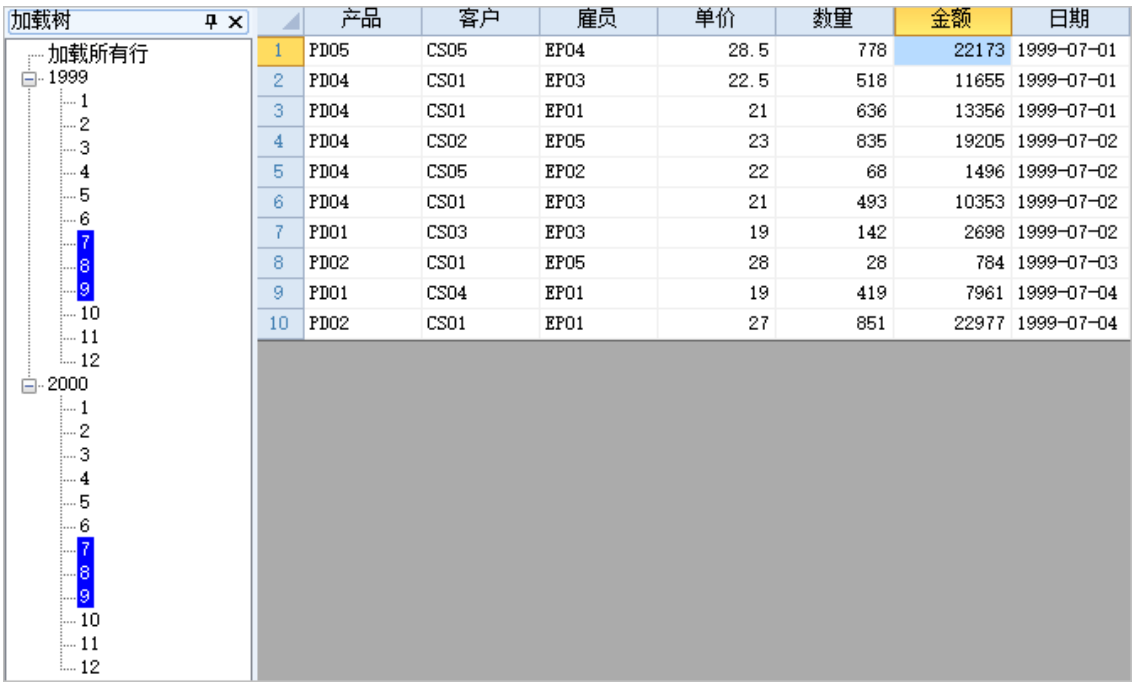
1、显示加载树

2、启用了分页加载功能

3、开启了后台筛选功能 ；或者虽然没有开启后台筛选，但统计的时候，勾选了"直接统计后台数据"。

那么通过菜单进行交叉统计或分组统计，统计的是所有选定节点（单个或多个）下的全部数据，不管这些数据是否已经加载。

例如我们需要统计比较相邻两年的 7、8、9 月份的销售数据，只需在加载树中选择这两年的 7、8、9 月份节点：



然后根据产品（水平分组）按年月（垂直分组）对数量进行交叉统计：



得到的统计结果为：

	产品	1999年			2000年		
		7月	8月	9月	7月	8月	9月
1	PD01	5204	8103	11014	6861	6830	9805
2	PD02	6807	9755	8657	12682	8774	8897
3	PD03	6088	8579	10107	12709	9137	5660
4	PD04	11864	13565	12863	3821	7926	6915
5	PD05	9332	11102	8595	7580	13786	6433

2、仅统计已经加载的数据

如果没有开启后台筛选功能，且没有勾选"直接统计后台数据"，那么通过菜单进行的分组统计或交叉统计，将仅统计已经加载的数据，也就是当前页面的数据。

3、统计后台所有数据

显然，如果要统计所有数据，必须先选择节点"加载所有行"，然后开启后台筛选功能，或者统计的时候勾选"直接统计后台数据"。

4.14.3 表的动态加载与卸载

4.14.3.1 关于动态加载表

动态加载数据，只是针对表的数据而言，表本身始终会加载的。

一个规模较大的系统，表的数量可能会很多，如果多达上百个甚至几百个表，那么即使不加载任何数据，全部加载空表也是相当费时的。

我见过很多用户，为减少初始加载的表的数量，不采用常规的设计方式，而是大量采用窗口，在窗口中插入 SQLTable 型或者 SQLQuery 型 Table 控件，这样在打开窗口的时候，才加载表的结构和数据，以提高系统的启动速度；但这种设计方式， 背离了 Foxtable“简单高效”的初衷，对于多数用户来说并不太容易控制，而且每次打开窗口都要加载一次数据，有效率的损失；此外由于这种设计方式，会导致表依附于窗口，关联和表达式这些常用好用的功能，以及跨表的数据计算与调用，也不容易实现。

如果表本身也可以动态加载，在需要的时候加载表的结构和数据，在不需要的时候还能卸载掉这个表，就能做到在打开项目的时候只加载极少的表，而大部分的表留在打开相关功能模块的时候加载；且这些功能模块暂时不使用的时候，还能卸载掉相关的表，那么以后不管有多少个表，我们都可以按常规设计 系统了。

4.14.3.2 一个小例子

为了要进一步了解动态加载和卸载表带来的便利，可以打开 CaseStudy 目录下的文件：表的加载与卸载.Table

打开此文件，是一个空白的系统，我们只需寥寥几行代码，即可动态加载和卸载整个系统。

可见表的动态加载，可以让我们在一个项目中轻松集成众多功能模块，随意切换，对于启动速度却几乎没有影响。

4.14.3.3 初始不加载表

BeforeLoadInnerTable 和 BeforeLoadOuterTable 事件都有一个 Cancel 属性，将此属性设置为 True，将不加载对应的表。

示例一

例如在 BeforeLoadInnerTable(内部表)或 BeforeLoadOuterTable(外部表)中设置代码：

```
Select Case e.DataTableName
    Case "表A","表B" '表A和表B无条件加载
    Case "表C"
        If e.User.Type <> UserTypeEnum.Developer Then '只有开发者才加载表C
            e.Cancel = True
        End If
    Case Else '其他所有表默认都不加载
        e.Cancel = True
End Select
```

这样在打开项目的时候，表A和表B始终都会加载，表C只有在开发者打开项目的时候才会加载，而其它表都不会加载。

示例二

例如在 BeforeLoadInnerTable(内部表)或 BeforeLoadOuterTable(外部表)中设置代码：

```
Select Case e.DataTableName
    Case "表 A"
        e.Cancel = (e.User.Name = "张三")
    Case "表 B"
        e.Cancel = (e.User.Name = "李四")
    Case "表 C"
        e.Cancel = (e.User.Name = "王五")
End Select
```

这样张三打开项目，将不会加载表 A，需要注意的是，如果打开项目后，有代码引用表 A，将会出错，因为此时表 A 是不存在的。

同样李四打开项目不会加载表 B，王五打开项目不会加载表 C。

4.14.3.4 加载单个表

DataTables 有个 Load 方法，用于加载表。

示例

```
If DataTables.Contains("表C") = False Then '如果表C没有加载
```

```
    DataTables.Load("表C") '加载表C
End If
```

需要注意的是，只有项目中本来已经存在，但目前并未加载的表，才能使用Load方法加载之，所以：

1、对于外部表，只有事先通过菜单将其添加到项目中，并初始不加载，或者加载后又被卸载掉，才能使用Load方法加载此表。
2、对于外部表，如果事先没有通过菜单将其添加到项目中，不能使用Load方法加载，但是可以使用OuterTableBuilder加载。

4.14.3.5 卸载单个表

DataTables 有个 Unload 方法，用于卸载表。
示例

```
If DataTables.Contains("表C")  Then '如果表C已经加载
    DataTables.Unload("表C") '卸载表C
End If
```

提示:

1、这个方法可以卸载除SQLTable和SQLQuery之外的任何表，不管这个表是以何种方式生成的；至于SQLTable和SQLQuery，关闭窗口就自动卸载了。
2、如果某个表被窗口中的Table控件绑定了，那么在卸载此表之前，必须先关闭窗口。

4.14.3.6 加载多个表

可以一次加载多个表，表名和表名之间用符号“|”隔开。
示例

```
If DataTables.Contains("产品") = False Then
    DataTables.Load("产品|订单|订单明细")
End If
```

重要提示：
如果两个或多个表之间建立了关联，那么这些表必须在同一个 Load 方法中同时加载，不可以分别加载。

4.14.3.7 卸载多个表

可以一次卸载多个表，表名和表名之间用符号“|”隔开。

示例

```
If DataTables.Contains("产品") Then
    DataTables.Unload("产品|订单|订单明细")
End If
```

重要提示：
如果两个或多个表之间建立了关联，那么这些表必须在同一个 Unload 方法中同时卸载，不可以分别卸载。

4.14.3.8 使用关闭表按钮

我们可以在主界面中显示一个关闭表按钮：



此按钮默认不显示，要显示这个按钮，只需项目属性设置中将“显示关闭按钮”设置为 True。
实际上，用户单击这个按钮，并不会真的关闭表，只是触发项目事件 BeforeCloseTable 而已。
BeforeCloseTable 事件并没有任何 e 参数，我们可以通过全局变量 MainTable 获得当前活动的表，也就是用户希望关闭的表。

示例一
如果希望用户单击关闭表按钮之后，能够隐藏此表，为此可以将 BeforeCloseTable 事件代码设置为：

```
MainTable.Visible = False
```

示例二
如果希望用户单击关闭表按钮之后，能够直接卸载此表，为此可以将 BeforeCloseTable 事件代码设置为：

```
DataTables.UnLoad(MainTable.Name)
```

示例三

再例如禁止卸载表 A 和表 B，其他表则允许卸载，为此可以将 BeforeCloseTable 事件代码设置为：

```
Select Case MainTable.Name
    Case "表A","表B"  '如果是表A和表B,不执行任何操作
    Case Else
        DataTables.UnLoad(MainTable.Name)
End Select
```

示例四

我们知道，如果两个或多个表之间建立了关联，那么这些表必须在同一个 Unload 方法中同时卸载，不可以分别卸载。
假定一个系统含有产品、客户、订单三个表，这三个表已经建立关联，如果允许用户通过单击关闭按钮来卸载这三个表，可以将 BeforeCloseTable 事件代码设置为：

```
Select Case MainTable.Name
    Case "产品","客户","订单"
        DataTables.Unload("产品|客户|订单")
End Select
```

这样用户选择这三个表中的任何一个单击关闭按钮的时候，都会同时卸载这三个表。

4.14.3.9 如何设置加载条件

通过 DataTables 的 Load 方法加载表的时候，同样会触发 BeforeLoadInnerTable(内部表)或 BeforeLoadOuterTable(表)事件，可以通过这两个事件设置数据加载条件。

你也许会奇怪，前面不是通过 BeforeLoadInnerTable 和 BeforeLoadOuterTable 的 Cancel 参数，来实现初始不加载表的目的吗？ 再次触发这两个事件，Cancel 参数还是为 True，怎么能正常加载表呢？不用担心，Cancel 参数只有在打开项目的时候才有效，所以我们依然可以通过这两个事件实现初始不加载表和设置加载条件的双重目的。

示例一

假定订单表是内部表，希望订单表只加载当日的订单，但是打开项目是，初始并不加载订单表，为此可将 BeforeLoadInnerTable 事件代码为：

```
If e.DataTableName = "订单" Then
    e.Cancel = True '打开项目的时候不加载此表
    e.Filter = "日期 = #" & Date.Today & "#" '设置加载条件
End If
```

当项目启动后，我们随时可以通过下面的代码将订单表加载进来，且只加载当日的订单:

```
If DataTables.Contains("订单") = False
```

```
    DataTables.Load("订单")
End If
```

示例二

假定订单表是外部表， 希望初始并不加载订单表，而是在运行过程打开加载树进行分页加载，为此可将BeforeLoadOuterTable事件代码为：

```
If e.DataTableName = "订单" Then
    e.Cancel = True '打开项目的时候不加载此表
    e.SelectString = "Select * From {订单} Where [_Identify] Is Null" '设置加载条件,注意如果主键不是_Identify,请改为实际的主键名称
End If
```

当项目启动后，我们随时可以通过下面的代码将订单表加载进来，并打开加载树进行分页加载：

```
If DataTables.Contains("订单") = False Then
    DataTables.Load("订单") '加载订单表，注意由于BeforeLoadOuterTable事件设置的代码，执行后只加载了结构，没有数据
End If
Tables("订单").OpenLoadTree("日期  YM",150,30,True) '打开加载树,加载树宽度150个像素,分页加载,每页30行
MainTable = Tables("订单") '切换到订单表
```

4.14.4 SQL 表达式的语法差异

Foxtable 中所有和动态加载的相关的属性和方法，包括那些直接统计或处理后台数据的方法，其条件表达式的语法均为 SQL 语法，且表达式中的表名和列名必须和后台数据库保持一致。
采用的数据源不同，SQL 语法会有一些细节产别。
关于 SQL，后面有专门一章介绍，现在先介绍一下 SQL 表达式和常规表达式的几个主要差异：
1、不能使用*作为通配符，只能使用%作为通配符，例如：

```
DataTables("客户").LoadFilter = "公司名称  Like '%贸易%'"
DataTables("客户").Load()
```

2、 如果数据源是SQL Server，日期用单引号括起来，而不是用#，例如：

```
DataTables("订单").LoadFilter = "日期  = '" & Date.Today & "'"
DataTables("订单").Load()
```

如果是内部表或Access数据源，日期还是用符号#括起来：

```
DataTables("订单").LoadFilter = "日期  = #" & Date.Today & "#"
DataTables("订单").Load()
```

3、SQL Server中的逻辑列，用0表示False，1表示True，例如加载已经结账的订单：

```
DataTables("订单").LoadFilter = "结账 = 1"
DataTables("订单").Load()
```

如果是如果是内部表或Access数据源，还是用True和False表示逻辑值，例如：

```
DataTables("订单").LoadFilter = "结账 = True"
DataTables("订单").Load()
```

4、常规表达式的函数非常有限，而SQL的函数更加丰富，但是不同的数据源，函数有些差别。

如果数据源是SQL Server，可以使用以下函数：
http://www.foxtable.com/help/topics/1827.htm

如果是内部表或Access数据源，可以使用以下函数：
http://www.foxtable.com/help/topics/0692.htm

例如，用SQLCompute统计2013年的销售数量：

```
Dim Val As Integer
Val = DataTables("订单").SQLCompute("Sum(数量)", "Year(日期) = 2013")
```

如果用Compute统计，表达式会长很多，而且必须已经加载2013年的全部数据：

```
Dim Val As Integer
Val = DataTables("订单").Compute("Sum(数量)", "日期 >= #1/1/2013# And 日期 <= #12/31/2013#")
```

4.15 SQL 相关

4.15.1 SQL 语言入门

4.15.1.1 关于 SQL

之前我们已经接触了 SQL 语言，例如在讲述动态数据加载的时候，使用的 Select 语句，就属于 SQL 语言。
本章我们将系统地介绍 SQL 语言。
SQL 语言直接在后台进行操作，先天具备速度上的优势，而且非常灵活。
SQL 语言的内容很多，我们只需掌握其中的 SELECT 语句即可，要成为一个真正的数据管理高手，掌握 Select 语句是很有必要的，我们的课程也以 Select 语句为主。
至于 Delete、Update、Insert Into 语句，虽然也有简单的讲述，但在 Foxtable 中使用的机会并不多。
不同类型的数据源，其 SQL 语言基本通用，细微的差别我们会在介绍 Select 语句的时候一一点明。
至于函数，不同的数据源会有较大的差别，我们会分开讲述，学习的时候根据需要选择。
内部数据表的 SQL 语句及函数，和 Access 完全一致。

4.15.1.2 一个特殊要求

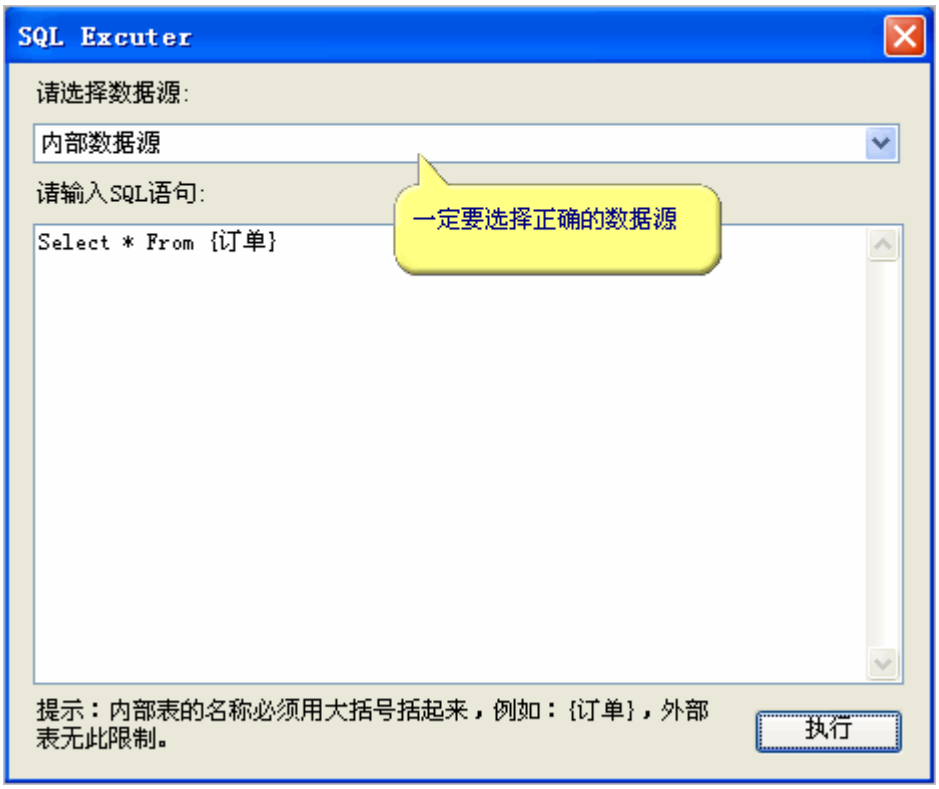
不管你使用何种数据源，Foxtable 中的 SQL 语句，表名都需要用大括号括起来，例如：
SELECT * FROM {订单}
这是有别于其他任何开发工具的要求，大家需要留意。
特殊的列名用方括号括起来，例如：
Select [_Identify], 日期, 客户, 产品, 数量 From {订单}
其中_Identify 因为是以符号开头，所以必须用方括号括起来，至于日期、客户、产品等列，是否使用方括号都无所谓，主要看个人习惯。
对于 Oracle 数据源，我们同样要求使用方括号将特殊列名括起来，Oracle 用户不用担心，虽然 Oracle 的语法要求用双引号而不是方括号，但是 Foxtable 在将 SQL 语句提交到 Oracle 执行之前，会自动将方括号转换为双引号的。
需要注意的是，对于 Oracle 来说，方括号中的列名是区分大小写的，没有使用方括号的列，则自动转换为大写。

4.15.1.3 测试 SQL 语句

在菜单的杂项功能区中，单击下面的"执行 SQL"按钮：



出现 SQL 执行窗口，你可以在这里选择数据源，输入 SQL 语句，然后单击“执行”按钮执行：



4.15.1.4 SELECT 语句

一、基本语法

SELECT 字段列表 FROM {表名} WHERE 条件 GROUP BY 分组字段 ORDER BY 排序字段

其中"WHERE"、"GROUP BY"和"ORDER BY"是可选的 ，这三者的顺序是不能调换的。

再次提示：SQL 语句中的表名必须用大括号括起来，这是 Foxtable 的一个非常特殊的要求。

示例一

SELECT 产品, 客户, 单价, 数量, 日期 FROM {订单}

字段列表不能包括表达式列，例如金额列是一个表达式列，是通过数量和单价计算得出的，所以金额列不能出现在字段列表中。

这是因为表达式列并没有存储在后台的数据源中。

示例二

如果你要包括所有字段，那么直接用*代替字段列表即可：

SELECT * FROM {订单}

示例三

如果需要排除重复的行，可以在字段列表前面加上关键词 DISTINCT，例如获得所有产品名称：

SELECT DISTINCT 产品 FROM {订单}

再例如获得每个客户订购过的产品：

SELECT DISTINCT 客户, 产品 FROM {订单}

示例四

如果不需要获得所有的数据，可以使用 TOP 关键词指定行数，假定只需加载 10 行数据：

SELECT TOP 10 * FROM {订单}

如果要按百分数加载数据，那么请加上 PERCENT 关键词，例如加载 10%的订单：

SELECT TOP 10 PERCENT * FROM {订单}

TOP 和 PERCENT 关键词对于 Oracle 数据源是无效的，Oracle 用 ROWNUM，这是一个默认的列，表示加载行的序号，例如：

Select * From {EMPLOYEES} Where ROWNUM <= 50

表示从 EMPLOYEES 表加载前 50 行数据。

示例五

可以用 ORDER BY 指定排序方式，例如根据日期顺序加载数据:

SELECT * FROM {订单} ORDER BY 日期

默认的排序方式是升序，如果希望降序排序，可以在排序列后面加上 DESC 关键词，例如：

SELECT * FROM {订单} ORDER BY 日期 DESC

排序列可以是多列，例如首先根据产品排序，相同产品的行则根据日期排序：

SELECT * FROM {订单} ORDER BY 产品,日期

示例六

TOP 关键词如果没有和 ORDER BY 配合使用，那么毫无意义，因为得到的只是随机提取的一些行。

如果你要获得最近的 10 个订单：

SELECT TOP 10 * FROM {订单} ORDER BY 日期 DESC

注意必须加上关键词 DESC，否则你得到的将是最早的 10 个订单，而不是最近的 10 个订单

由于 Oracle 没有 TOP 关键词，所以要实现上述目标，须使用下面的语句：

SELECT * FROM {订单} Where ROWNUM <= 10 ORDER BY 日期 DESC

示例七

可以使用 AS 关键词重新命名列，例如：

SELECT 产品, 客户, 单价, 数量 AS 订购数量,日期 AS 订购日期 FROM {订单}

上述查询语句得到的结果中，"数量"列将被改名为"订购数量"，"日期"列被改名为"订购日期"

示例八

你还可以直接在 SELECT 语句中使用表达式，例如：

SELECT 产品, 客户, 单价, 数量, 数量 * 单价 AS 金额, 日期 FROM {订单}

上述的语句新生成一个金额列，该列的值等于数量乘以单价(数量*单价)

二、条件表达式

可以用 WHERE 关键词设置查询条件，多个条件可以用 AND 或者 OR 运算符连接。

例如加载产品为 PD01 和 PD02 订单：

SELECT * FROM {订单} WHERE 产品 = 'PD01' OR 产品 = 'PD02'

再例如加载 1999 年 1 月份的订单：

SELECT * FROM {订单} WHERE 日期 >= #1/1/1999# AND 日期 <= #1/31/1999#

如果数据源是 SQl SERVER，日期常量用单引号而不是#号括起来，例如：

SELECT * FROM {订单} WHERE 日期 >= '1/1/1999' AND 日期 <= '1/31/1999'

如果是 Oracle 的数据源，必须用 TO_DATE 函数转换，例如加载 1999 年 10 月 21 日的订单：

SELECT * FROM {订单} WHERE 日期 = TO_DATE('10/21/1999','MM/DD/YYYY')

IN 运算符

IN 运算符是用来判断一个表达式的值是否属于一个指定列表中的值。

例如同样是加载产品 PD01 和 PD02 的订单，用 IN 运算符可以简化为：

SELECT * FROM {订单} WHERE 产品 IN ('PD01','PD02')

可以用 NOT 关键词来进行反向的操作，例如加载除产品 PD01 和 PD02 之外的订单：

SELECT * FROM {订单} WHERE 产品 NOT IN ('PD01','PD02')

显然，指定的值越多，IN 运算符的优势越明显。

BETWEEN 运算符

BETWEEN 运算符用于判断一个表达式的值是否介于一个特定的范围之间，起始值和终止值用运算符 AND 连接起来
例如加载 1999 年 1 月份的订单：
SELECT * FROM {订单} WHERE 日期 BETWEEN #1/1/1999# AND #1/31/1999#
再例如加载折扣在 0.1 到 0.2 之间的订单
SELECT * FROM {订单} WHERE 折扣 BETWEEN 0.1 AND 0.2
可以用 NOT 关键词来进行反向的操作，例如加载折扣在 0.1 到 0.2 之外的订单：
SELECT * FROM {订单} WHERE 折扣 NOT BETWEEN 0.1 AND 0.2

LIKE 运算符

LIKE 运算符用于将指定表达式的值和一个模式字符串进行比较，判断两者是否匹配。
LIKE 可以使用通配符，通配符包括：

_(下划线)
匹配单个字符
%（百分号）
匹配任意个数的字符

例如：
Like 'C%'，
表示以字符 C 开始的值。
在例如：
Like 'A_e%'
表示第一个字母是 A，第三个字母是 e。
找出姓名为三个字，且第二个字是“文”的客户：
Select * From {客户} Where 姓名 Like '_文_'
找出客户表中的贸易公司：
SELECT * FROM {客户} WHERE 公司名称 LIKE '%贸易%'
可以利用 NOT 关键词进行反向操作，例如：
SELECT * FROM {客户} WHERE 公司名称 NOT LIKE '%贸易%'
找出所有非贸易公司。

IS NULL 运算符

IS NULL 操作符被用于判断一个表达式的值是否为空，例如：
SELECT * FROM {订单} WHERE 折扣 IS NULL
表示加载没有输入折扣的订单，注意 0 不是空，所以上面的语句不会加载折扣为 0 的订单。
SELECT * FROM {订单} WHERE 折扣 IS NULL OR 折扣 = 0
表示加载出没有输入折扣或者折扣为 0 的订单。
可以用 NOT 进行反向操作，例如：
SELECT * FROM {订单} WHERE 折扣 IS NOT NULL
表示加载出已经输入折扣的订单。

三、 数据统计和聚合函数

聚合函数用于统计数据，通常和 GROUP BY 关键词配合使用。

SUM

返回指定表达式的总计值。

AVG

返回指定表达式的平均值。

MAX

返回指定表达式的最大值。

MIN

返回指定表达式的最小值。

COUNT

返回符合指定条件的记录数

Var

返回指定表达式的总体方差

VarP

返回指定表达式的总体方差

StDev

返回指定表达式的总体标准偏差

StDevP

返回指定表达式的总体样本标准偏差

示例一

统计订单表总的订购数量：

SELECT SUM(数量) AS 总计 FROM {订单}

再例如统计总的订单数:

SELECT Count(*) AS 总订单数 FROM {订单}

可以同时进行多个统计：

SELECT SUM(数量) AS 总计, AVG(数量) AS 平均 , MAX(数量) AS 最大, MIN(数量) AS 最小 FROM {订单}

可以得到订单表数量列的累计值、平均值、最大值、最小值。

示例二

更多的时候，聚合函数是 GROUP BY 关键词配合使用的，用于进行分组统计。

例如统计每个客户订购产品的数量：

SELECT 客户,SUM(数量) AS 数量 FROM {订单} GROUP BY 客户

分组列必须同时包括在 SELECT 子句和 GROUP BY 子句中，例如上面的客户列。

示例三

统计每个客户的订单数、订购数量、订购金额：

SELECT 客户,Count(*) AS 订单数, SUM(数量) AS 数量, SUM(数量 * 单价 * (1 - 折扣)) AS 金额 FROM {订单} GROUP BY 客户

注意金额列是通过一个表达式计算得出的。

示例四

可以有多个分组列，例如统计每个客户订购不同产品的数量：

SELECT 客户, 产品, SUM(数量) AS 数量 FROM {订单} GROUP BY 客户,产品

四、HAVING 字句

HAVING 和 WHERE 有点类似，不同的是 WHERE 用于过滤记录，而 HAVING 用于过滤分组。

例如我要统计出定购 PD01 产品的客户以及其订购数量，而且只列出总订购数量超过 500 的客户：

SELECT 客户,SUM(数量) FROM {订单} WHERE 产品 = 'PD01' GROUP BY 客户 HAVING SUM(数量) > 500

五、UNION 运算符

UNION 运算符用于组合两个查询的结果。

例如有一个客户表，一个供应商表，我需要得到所有在中国的客户和供应商的名称和地址。
因为数据位于不同的表中，显然，我们需要用两个查询才能完成任务：
SELECT 公司名称, 地址 FROM {客户} WHERE 国家 = '中国' UNION SELECT 公司名称, 地址 FROM {供应商} WHERE 国家 = '中国'
两个查询的字段名称、个数、类型必须完全一致才行。
默认情况下，UNION 会自动排除重复的行，然后你可以用 ALL 关键词来确保返回所有的行，例如：
SELECT 公司名称, 地址 FROM {客户} WHERE 国家 = '中国' UNION ALL SELECT 公司名称, 地址 FROM {供应商} WHERE 国家 = '中国'

六、 子查询

例如希望得到数量大于平均订购数量的订单：
Select * From {订单} Where 数量 > (Select Avg(数量) From {订单})
上述语句中的绿色部分，是一个独立的 SQL 语句，用于得到平均订购数量，这就是子查询，子查询必须用括号括起来。
子查询可以针对另一个表，例如希望得到单价大于平均销售单价的产品：
Select * From {产品} Where 单价 > (Select Avg(单价) From {订单明细})
再例如希望得到订单数超过 50 个的产品：
Select * From {产品} Where (Select Count(*) From {订单明细} Where {订单明细}.产品 ID = 产品.产品 ID) > 50
子查询只能包括一列，多于一列会报错。

IN 子查询

In 子查询用于判断查询结果的某列，在另一个查询的某列是否有对应的值。
例如查询在 1998 年 5 月 1 日之后订购过产品的客户的详细资料：
SELECT * FROM {客户} WHERE 客户 ID IN (SELECT 客户 ID FROM {订单} WHERE 订购日期 > #1998-5-1#)
你可以用 NOT 进行反向操作，例如我们希望查询在 1998 年 5 月 1 日之后没有订购过产品的客户的详细资料：
SELECT * FROM {客户} WHERE 客户 ID NOT IN (SELECT 客户 ID FROM {订单} WHERE 订购日期 > #1998-5-1#)

EXISTS 子查询

EXISTS 用于判断子查询是否有记录。
例如要列出所有订购过产品的客户资料：
SELECT * FROM {客户} WHERE EXISTS (SELECT * FROM {订单} WHERE {订单}.客户 ID = {客户}.客户 ID)
请注意子查询的比较语句：
{订单}.客户 ID = {客户}.客户 ID
该语句明确了比较字段的表名，因为两个字段来源于不同的表。
你可以用 NOT 进行反向操作，例如列出从来没有订购过产品的客户资料：
SELECT * FROM {客户} WHERE NOT EXISTS (SELECT * FROM {订单} WHERE {订单}.客户 ID = {客户}.客户 ID)

七、多表查询

一般用户可以忽略以下内容，因为 Foxtable 提供了 SQLJoinTableBuilder，可以轻松解决这种多表查询问题。

有的时候，你可能希望查询表的列来自于多个表，可以通过 JOIN 语句来实现这样的功能。JOIN 语句可以让你从已经定义了相互关系的工作表中检索记录，而不用管记录和工作表之间的关系是一对一、一对多还是多对多。

语法：

SELECT 字段列表 FROM {表 1} INNER JOIN {表 2} ON {表 1}.列名 = {表 2}.列名

ON 关键词指定两个表通过哪一列进行连接，指定的字段必须具备相同的数据类型和长度。

如果某个字段在不止一个表中出现，必须在 Select 语句中明确表的名称，格式为： {表名}.列名

示例：

SELECT {订单明细}.*, 产品名称 FROM {订单明细} INNER JOIN {产品} ON {订单明细}.产品 ID = {产品}.产品 ID ORDER BY 订单 ID

上述语句生成的查询表，包括订单明细的所有列，以及该订单明细对应的产品名称（来源于产品表）

INNER JOIN 要求两个表必须都有对应的记录，才会在查询表中生成记录。你还可以用 LEFT JOIN 或者 RIGHT JOIN，前者只要求左边的表有对应的记录即可，后者只要求右边的表有对应的记录即可。

数据不仅可以来自于两个表，还可以来自于更多的表，例如下面的语句：

SELECT 订购日期, {订单明细}.*, 产品名称 FROM ({订单明细} INNER JOIN {产品} ON {订单明细}.产品 ID = {产品}.产品 ID) INNER JOIN {订单} ON {订单明细}.订单 ID = {订单}.订单 ID ORDER BY 订购日期

上面的查询表的列来自于三个表，其中订购日期来自于订单表，产品名称来自于产品表，其余来自订单明细表。

下面的查询，列来自于四个表：

SELECT 订购日期,公司名称,产品名称, 数量, {订单明细}.单价, 折扣, (数量 * {订单明细}.单价 * (1-折扣)) AS 金额 FROM (({订单明细} INNER JOIN {产品} ON {订单明细}.产品 ID = {产品}.产品 ID) INNER JOIN {订单} ON {订单明细}.订单 ID = {订单}.订单 ID) INNER JOIN {客户} ON {订单}.客户 ID = {客户}.客户 ID

在上面的查询中，因为产品表和订单明细表都有单价列，所以字段列表中对于单价列的引用，要明确引用的是哪一个表的单价列，例如：订单明细.单价

此外，在有多个 JOIN 语句的情况下，靠前的 JOIN 语句要用圆括号括起来，如前面的例子所示。

4.15.1.5 DELETE 语句

用于删除指定条件的行。

语法：

DELETE FROM 表名 WHERE 条件语句

例如：

DELETE FROM {订单} WHERE Year([日期]) = 1998

表示删除 1998 年的订单。

4.15.1.6 UPDATE 语句

用于直接向记录中写入数据，语法：

UPDATE 表名 SET 列名 1 = 新值 1,列名 2 = 新值 2 .... WHERE 条件语句

例如：
UPDATE {订单} SET 折扣 = 0.1 WHERE 数量 > 5000
表示将订购数量大于 5000 的订单的折扣设为 0.1。
UPDATE 可以跨表更新数据，例如：
UPDATE {订单},{产品} SET {订单}.单价 = {产品}.单价 * 1.1 WHERE {订单}.[产品编号] = {产品}.[产品编号]
表示将产品表中的单价乘以 1.1 填入订单表中相同产品编号的行的单价列。

4.15.1.7 Insert Into 语句

用于向表中添加行。
本语句不能用于内部数据表，只能向外部数据表添加行。
语法
Insert Into 表名 (列名 1, 列名 2, 列名 3....) Values(值 1, 值 2, 值 3...)
例如：
Insert Into 订单 (编号,产品) Values(100,'PD01')
表示向订单表插入一行，编号列的内容设为 100，产品列的内容为 PD01

4.15.1.8 符号总结

1、SQL 语句中的表名必须用大括号括起来，这是 Foxtable 的一个非常特殊的要求，例如：
SELECT * FROM {订单}
2、字段名可以直接使用，也可以用方括号括起来，例如：
SELECT [客户],[产品],[日期],[数量] FROM {订单}
或
SELECT 客户,产品,日期,数量 FROM {订单}
3、一些特殊的列名，例如包括符号的列名，必须用方括号括起来，例如：
Select [_Identify],日期,数量 From {订单}
上面的语句中，列名"_Identify"因为包括符号"_"，所以必须用方括号括起来，否则会提示错误。
4、SQL 语句中的字符必须用单引号括起来，例如：
SELECT * FROM {订单} WHERE 产品 = 'PD01' OR 产品 = 'PD02'
5、数字可直接使用，例如：
Select * From {订单} Where 数量 > 100 And 折扣 = 0.1
6、对于内部表和 Access 数据源，日期用符号#括起来，例如：
SELECT * FROM {订单} WHERE 日期 >= #1/1/1999# AND 日期 <= #1/31/1999#
对于 SQL SERVER 数据源，日期用单引号括起来，例如：
SELECT * FROM {订单} WHERE 日期 >= '1/1/1999' AND 日期 <= '1/31/1999'
对于 Oracle 数据源，必须用 TO_DATE 函数将字符转换为日期，例如加载 1999 年 10 月 21 日的订单：
SELECT * FROM {订单} WHERE 日期 = TO_DATE('10/21/1999','MM/DD/YYYY')

4.15.1.9 SQL Server 的逻辑列

我曾经使用下面的 SQL 语句设置逻辑列的值：
UPDATE {表 A} SET 结账 = False
在使用内部表和 Access 数据源的时候，上述 SQL 语句均能正常执行，当转为 SQl Server 的时候，却出现了错误提示。
这是因为 SQL Server 用 0 表示 False，1 表示 True，所以上述语句应该改为：
UPDATE {表 A} SET 结账 = 0
同样在使用条件表达式的时候，也要做适当的调整，例如：
Select * From {表 A} Where 结账 = 1
表示获得已经结账的行。
需要注意的是，SQL Server 这种特性只有在使用 SQL 语句的时候才存在，数据一旦加载到 Foxtable，不管数据来自何种数据源，逻辑列的值都是 False 和 True，不再是 0 和 1。

4.15.2 SQL 函数参考

4.15.2.1 Access

本章介绍的是使用 Access 数据源时，SQL 语句支持的常用函数。
内部数据源支持的 SQL 函数，和 Access 完全相同。
日期函数
Year
返回给定日期的年份。
例如统计不同年份的销售额：
SELECT Year(日期) AS 年, SUM(数量) AS 数量 FROM {订单} GROUP BY Year(日期)
Month
返回给定日期的月份。
例如统计 1999 年各月的销售额：
SELECT Month(日期) AS 月, SUM(数量) AS 数量 FROM {订单} WHERE Year(日期) = 1999 GROUP BY Month(日期)
Day
返回给定日期的天
Date
返回系统日期
例如查询当天的订单：
SELECT * FROM {订单} WHERE 日期 = Date()
DatePart
返回给定日期时间的指定部分
语法
DatePart(Interval, date)
Interval: 字符型，表示希望返回的时间部分
Date: 给定的日期值

Interval 包含以下设置：
yyyy 年
q 季度
m 月
y 某年的某一天
d 天
w 工作日
ww 周
h 时
n 分
s 秒
例如统计 1999 年的各季度每种产品的销售数量：
SELECT 产品, DatePart('q', 日期) AS 季度, SUM(数量) AS 数量 FROM {订单} WHERE Year(日期) = 1999 GROUP BY 产品, DatePart('q', 日期)
DateDiff
计算两个给定日期的间隔。
语法

DateDiff(Interval, Date1, Date2)
Interval：表示时间间隔的类型。
Date1：起始日期
Date2：结束日期
Interval 包含以下设置：
yyyy 年
q 季度
m 月
y 某年的某一天
d 天
w 工作日
ww 周
h 时
n 分
s 秒
例如加载最近两个月内的订单：
SELECT * FROM {订单} WHERE DateDiff('m', 日期, Date()) < 2
DateAdd
为一给定的日期加上给定的时间间隔。
语法

DateAdd(Interval, Number, Date)
Interval：表示时间间隔的类型，可设置的值和 DateDiff 函数相同。
Number: 表示要加上的时间间隔数量。
Date: 日期值
例如加载未来两个月要发货的订单：

SELECT * FROM {订单} WHERE 发货日期 BETWEEN Date() AND DateAdd('m', 2, Date())

数学函数

Abs

返回数值的绝对值。

Int

返回数值的整数部分。

Round

将数值四舍五入到指定的小数位数。

例如:

SELECT 客户,Count(*) AS 订单数, SUM(数量) AS 数量, Round(SUM(数量 * 单价 * (1 - 折扣)), 2) AS 金额 FROM {订单} GROUP BY 客户

上述语句的金额由数量、单价、折扣计算得出，且保留两位小数。

字符函数

Len

返回字符串的长度。

Mid

返回字符串从指定位置开始指定个数的字符，例如：Mid('abcdef',2,3) = 'bcd'

InStr

返回一个字符串在另一个字符串中首次出现的位置，例如：Instr('abcd','bc') = 2

LCase

将字符串转换为小写。

UCase

将字符串转换为大写。

Left

返回字符串从左边开始指定个数的字符，例如：Left('abcd',2) = 'ab'

Right

返回字符串从右边开始指定个数的字符。

流程函数

Choose

根据一个整数表达式的结算结果，来决定返回那个位置的值

语法：

Choose(expr, value1, value2, value3...)

expr: 整数表达式，如果结果是 1，则返回 value1，如果结果是 2，则返回 value2，依次类推。

例如：choose(2,'a','b') = 'b'

Switch

按顺序计算一系列的表达式，如果某一表达式成立，则返回其随后的值。

语法：

Switch(expr1, value1, expr2, value2, expr3, value3...)

expr1、expre2、expr3：表示要计算的表达式，expr1 成立的话，返回值 value1，expr2 成立的话，返回值 value2，依次类推。

IIF

根据一个逻辑表达式的计算结果，返回两个候选值之一。

语法：

IIf(expr, truepart, falsepart)

expr: 逻辑表达式
truepart：逻辑表达式成立时返回的值
falsepart: 逻辑表达式不成立时返回的值
例如：
SELECT 客户, 产品, 日期, 数量, 数量 * 单价 * (1 - IIF(折扣 IS NULL, 0, 折扣)) As 金额 FROM {订单}
这样，即使某一行没有输入折扣，也能正常计算出金额。

4.15.2.2 SQL Server

本章介绍的是使用 SQL Server 的时候，SQL 语句支持的常用函数。
日期函数
Year
返回给定日期的年份。
例如统计不同年份的销售额：
SELECT Year(日期) AS 年, SUM(数量) AS 数量 FROM {订单} GROUP BY Year(日期)
Month
返回给定日期的月份
例如统计 1999 年各月的销售额：
SELECT Month(日期) AS 月, SUM(数量) AS 数量 FROM {订单} WHERE Year(日期) = 1999 GROUP BY Month(日期)
Day
返回指定日期的天。
DatePart
返回给定日期的指定部分。
语法
DatePart(Interval, date)
Interval: 字符型，表示希望返回的时间部分
Date: 给定的日期值
Interval 包含以下设置：
yyyy 年
q 季度
m 月
y 某年的某一天
d 天
dw 工作日
ww 周
hh 时
n 分
s 秒
例如统计每年的订单数：
SELECT DatePart(yyyy, 订购日期) AS 年, Count(*) AS 订单数 FROM {订单} GROUP BY DatePart(yyyy, 订购日期)
GetDate

返回系统日期，含时间。
例如加载今年的订单：
Select * From {订单} Where Year(订购日期) = Year(GetDate())
DateDiff
计算两个日期的间隔。
语法

DateDiff(Interval, Date1, Date2)
Interval：表示时间间隔的类型。
Date1：起始日期
Date2：结束日期
Interval 包含以下设置：
yyyy 年
q 季度
m 月
y 某年的某一天
d 天
w 工作日
ww 周
h 时
n 分
s 秒
例如加载最近两个月内的订单：
SELECT * FROM {订单} WHERE DateDiff(m, 订购日期, GetDate()) < 2
DateAdd
为一给定的日期加上指定的时间间隔。
语法

DateAdd(Interval, Number, Date)
Interval：表示时间间隔的类型，可设置值 DateDiff 函数相同。
Number: 表示要加上的时间间隔数量。
Date: 日期值
例如加载未来两个月要发货的订单：
SELECT * FROM {订单} WHERE 发货日期 BETWEEN GetDate() AND DateAdd(m, 2, GetDate())
字符函数
Len
返回字符串的长度
Lower
将字符串转换为小写。
Upper
将字符串转换为大写。
SubString
返回从指定位置开始指定长度的字符串，例如：SubString('abcdefg', 2, 3) = 'bcd'
CharIndex

返回给定子字符串首次出现的位置，例如：CharIndex('b','abcdefg') = 2

Left

取左侧指定长度的字符串。

Right

取右侧指定长度的字符串。

Replace

将指定的子字符串替换为其它的字符串，例如：Replace('abcdefg','bc','45') = 'a45defg'

数学函数

Abs

返回给定数值的绝对值。

Ceiling

返回大于或等于给定数值的最小整数。

Floor

返回小于或等于给定数值的最大整数。

Round

将数值四舍五入到指定的小数位数。

例如：

SELECT 客户,Count(*) AS 订单数, SUM(数量) AS 数量, Round(SUM(数量 * 单价 * (1 - 折扣)), 2) AS 金额 FROM {订单} GROUP BY 客户

上述语句的金额由数量、单价、折扣计算得出，且保留两位小数。

空值处理函数

ISNULL

判定一个表达式的值是否为空，如果不为空，则返回表达式的值，否则返回一个替代值。

语法

ISNULL(expression, value)

expression：一个表达式，如果此表达式的值不为空，则返回此值。

value： 一个替代值，如果表达式的值为空，则返回此替代值。

例如：

Select 日期, 产品, 数量, 数量, 单价, ISNULL(折扣,0) As 折扣 From {订单}

上述的语句中，如果已经输入折扣，则正常显示折扣，否则折扣列显示的是 0。

再次提示：0 和空值是两回事，空值表示没有输入数据，而 0 意味着用户已经输入了一个为 0 的值。

COALESCE

COALESC 函数可以指定任意多个参数，并返回这些参数中第一个不为空的值，如果全部为空，则

返回空值。

COALESCE 函数相当于 ISNULL 函数的增强版。

条件语句

Case When

Case When 语句用于进行条件判断。

语法 1

```
Case  表达式
When  比较值 1 Then  返回值 1
When  比较值 2 Then  返回值 2
When  比较值 3 Then  返回值 3
....
Else  默认返回值
End
```

当表达式等于比较值 1 时，则得到返回值 1，当表达式等于比较值 2 时，则得到返回值 2，其余类推；当比较结束，没有符合的比较值时，则得到默认返回值。

例如

```
Select CName As  客户名称，
(Case CLevel
When 1 Then 'VIP 客户'
When 2 Then '高级客户'
When 3 Then '普通客户'
Else '错误等级'
End) As  客户等级  From {客户}
```

语法 2

```
Case
When  表达式 1 Then  返回值 1
When  表达式 2 Then  返回值 2
When  表达式 3 Then  返回值 3
...
Else  默认返回值
End
```

当表达式 1 成立时，则得到返回值 1，当表达式 2 成立时，则得到返回值 2，其余类推；没有表达式成立时，则得到默认返回值。

例如：

```
Select  姓名,
(Case
When  体重  < 40 Then '偏瘦'
When  体重  > 50 Then '偏胖'
Else '适中'
End) As  体型  From {学生}
```

4.15.2.3 Oracle

本章介绍的是使用 Oracle 时候，SQL 语句支持的常用函数。

字符函数

INSTR

返回给定子字符串首次出现的位置。

LENGTH

返回给定字符串的长度。

LOWER

返回字符串的小写形式。

UPPER

返回字符串大写形式。

SUBSTR

返回从指定位置开始指定长度的字符串，例如：SUBSTR('abcdefg', 2, 3) = 'bcd'

REPLACE

将指定的子字符串替换为其它的字符串，例如：REPLACE('abcdefg','bc','45') = 'a45defg'

数学函数

Abs

返回给定数值的绝对值。

CEIL

返回大于或等于给定数值的最小整数。

Floor

返回小于或等于给定数值的最大整数。

Round

将数值四舍五入到指定的小数位数。

例如：

SELECT 客户,Count(*) AS 订单数, SUM(数量) AS 数量, Round(SUM(数量 * 单价 * (1 - 折扣)),

2) AS 金额 FROM {订单} GROUP BY 客户

上述语句的金额由数量、单价、折扣计算得出，且保留两位小数。

日期函数

SYSDATE

返回系统日期和时间。

例如加载当天的订单：

Select * From 订单 Where 订购日期 = SYSDATE

TO_CHAR

获得日期值的指定部分。

语法：

TO_CHAR(date,interval)

date: 日期值

interval：字符型，表示希望返回的时间部分。

interval 的可设置值有：

YYYY：年

Q： 季度

MM： 月

WW： 周

DD： 天

HH24：小时

MI： 分钟

SS： 秒

D： 星期几

DDD： 一年中的某一天

需要注意的是，TO_CHAR 返回的是一个数字，而不是数值，例如加载 1998 年入职的员工：

Select * From EMPLOYEES Where To_Char(Hire_Date,'YYYY') = '1998'

TO_DATE

将字符串转换为日期。

例如加载 1994 年 8 月 14 日入职的员工：

Select * From EMPLOYEES Where HIRE_DATE = TO_DATE('8/16/1994','MM/DD/YYYY')

ADD_MONTHS

给日期值加上指定的月数。

例如加载最近两个月入职的员工：

Select * From EMPLOYEES Where Hire_Date > ADD_MONTHS(SYSDATE,-2)

空值处理函数

NVL

判定一个表达式的值是否为空，如果不为空，则返回表达式的值，否则返回一个替代值。

语法

NVL(expression, value)

expression：一个表达式，如果此表达式的值不为空，则返回此值。

value： 一个替代值，如果表达式的值为空，则返回此替代值。

例如：

Select 日期, 产品, 数量, 数量, 单价, NVL(折扣,0) As 折扣 From {订单}

上述的语句中，如果已经输入折扣，则正常显示折扣，否则折扣列显示的是 0。

再次提示：0 和空值是两回事，空值表示没有输入数据，而 0 意味着用户已经输入了一个为 0 的值。

COALESCE

COALESC 函数可以指定任意多个参数，并返回这些参数中第一个不为空的值，如果全部为空，则返回空值。

COALESCE 函数相当于 NVL 函数的增强版。

条件语句

Case When

Case When 语句用于进行条件判断。

语法 1

```
Case  表达式
When  比较值 1 Then  返回值 1
When  比较值 2 Then  返回值 2
When  比较值 3 Then  返回值 3
....
Else  默认返回值
End
```

当表达式等于比较值 1 时，则得到返回值 1，当表达式等于比较值 2 时，则得到返回值 2，其余类推；当比较结束，没有符合的比较值时，则得到默认返回值。

例如

```
Select CName As  客户名称，
(Case CLevel
When 1 Then 'VIP 客户'
When 2 Then '高级客户'
When 3 Then '普通客户'
Else '错误等级'
End) As  客户等级  From {客户}
```

语法 2

```
Case
When  表达式 1 Then  返回值 1
When  表达式 2 Then  返回值 2
When  表达式 3 Then  返回值 3
...
Else  默认返回值
End
```

当表达式 1 成立时，则得到返回值 1，当表达式 2 成立时，则得到返回值 2，其余类推；没有表达式成立时，则得到默认返回值。

例如：

```
Select  姓名,
(Case
When  体重  < 40 Then '偏瘦'
When  体重  > 50 Then '偏胖'
Else '适中'
End) As  体型  From {学生}
```

4.15.3 SQLCommad

4.15.3.1 SQLCommand

这个类用于执行 SQL 命令。
属性：

- **CommandText**
  字符型，用于设置 SQL 命令。

- **ConnectionName**
  字符型，用于指定数据源名称，如果省略，则使用内部数据源。

- **CommandTimeOut**
  设置超时时限，默认为 30 秒，一般不需要设置此属性，除非需要进行非常耗时超过 30 秒的操作，以至于出现超时错误，此时可适当地加大 CommandTimeOut 属性的值。

本节的代码大都使用内部数据表作为示例，如果使用外部数据源，记得设置数据源名称
方法：
ExecuteScalar
用于执行返回单个值的命令，例如：

```
Dim val As Integer
Dim cmd As New SQLCommand
cmd.CommandText = "Select Sum(数量) From {订单} Where 日期 > #2012/2/21#"
val  = cmd.ExecuteScalar()
```

上面的代码统计 2012 年 2 月 21 日之后的销售总量，并保存在变量 val 中。
再例如：

```
Dim ld As Date
Dim cmd As New SQLCommand
cmd.CommandText = "Select Max(日期) From {订单} Where 产品 = 'PD01'"
ld = cmd.ExecuteScalar()
```

上面的代码得到最后一次有人订购 PD01 的日期，并保存在变量 ld 中。
再例如：

```
Dim ld As Date
Dim cmd As New SQLCommand
cmd.CommandText = "Select top 1 日期 From {订单} Where 产品 = 'PD01' Order by 日期 Desc"
ld = cmd.ExecuteScalar()
```

上面的代码得到最后一次有人订购 PD01 的日期，并保存在变量 ld 中 ，和前一段代码功能一样，

但是方法不同，希望大家体会。
ExecuteScalar 经常用来判断是否存在指定条件的数据，例如：

```
Dim cmd As New SQLCommand
Dim nm As String  = "贺辉"
cmd.ConnectionName = "User"
cmd.CommandText ="Select Count(*) From {Users} Where Name = '" & nm & "'"
If cmd.ExecuteScalar > 0 Then
        Messagebox.Show("已经存在同名用户","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

ExecuteValues

用于执行一次返回多个值的命令，例如：

```
Dim cmd As new SQLCommand
cmd.CommandText = "Select 部门,职务 from {员工} Where 姓名 = '王伟'"
cmd.ConnectionName = "数据源名称"
Dim Values = cmd.ExecuteValues
If Values.Count > 0 Then
    Output.show(Values("部门"))
    Output.show(Values("职务"))
End If
```

再例如：

```
Dim cmd As new SQLCommand
cmd.CommandText = "Select Sum(数量) As 数量, Sum(数量 * 单价) As 金额 from {订单} Where 产品 = 'PD01'"
cmd.ConnectionName = "数据源名称"
Dim Values = cmd.ExecuteValues
If Values.Count > 0 Then
    Output.show(Values("数量"))
    Output.show(Values("金额"))
End If
```

ExecuteNonQuery
用于执行不返回任何值的 SQL 命令，例如 DELETE 和 UPDATE 命令。
该方法会返回一个整数，表示受影响的行数。
示例一

```
Dim cmd As New SQLCommand
Dim Count As Integer
cmd.CommandText = "DELETE FROM {订单} WHERE 产品 = 'PD01'"
Count= cmd.ExecuteNonQuery()
Messagebox.Show( "总共删除" & Count & "行！ ")
```

以上代码会删除产品 PD01 的所有订单，执行速度很快。
但是执行完成后，你会发现订单表中产品 PD01 订单并没有被删除，这是因为 DELETE 是直接从后台删除的。
所以我们还需要用 Remove 方法移除已经删除的行：

```
Dim cmd As New SQLCommand
Dim Count As Integer
cmd.CommandText = "DELETE FROM {订单} WHERE  产品  = 'PD01'"
Count= cmd.ExecuteNonQuery()
DataTables("订单").RemoveFor("产品  = 'PD01'")
Messagebox.Show( "总共删除" & Count & "行！ ")
```

也可以用 Load 方法重新加载表：

```
Dim cmd As New SQLCommand
cmd.CommandText = "DELETE FROM {订单} WHERE  产品  = 'PD02'"
cmd.ExecuteNonQuery()
DataTables("订单").Load()
```

用 Remove 的效率更高，但用 Load 可以同时刷新数据，可根据需要选择。
当然如果删除的数据之前并没有被加载，那么就不需要在删除后执行 Remove 或 Load 方法。

示例二

```
Dim cmd As New SQLCommand
cmd.CommandText = "UPDATE {订单} SET  折扣  = 0.2 WHERE  数量  > 500"
cmd.ExecuteNonQuery()
DataTables ("订单").Load()
```

上述代码将订购数量超过 500 的订单的折扣设为 0.2，同样在代码的最后调用了 Load 方法来显示最新数据。
如果要更新的行之前并没有被加载，那么就不需要在更新后调用 Load 方法。
UPDATE 本身的执行速度很快，而且无需保存即刻生效，但是同样需要重新加载表，才能看到更新后的数据。
所以只有在需要更新大量的行，或者要更新没有加载的数据，才可以考虑用 UPDATE 命令

ExecuteReader

ExecuteReader 用于从后台提取数据生成一个临时的 DataTable，而这个 DataTable 只存在于代码的运行过程中，不会以表的形式呈现给用户。

示例一

同样，假定窗口中有一个组合框(ComboBox)，希望这个窗口能够列出订单表所有的客户名称，如果订单表采用动态加载，或者根本没有加载，那么显然无法从订单表中获得所有客户名称，只能利用 ExecuteReader 直接从后台提取：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim cmb As WinForm.ComboBox
cmd.CommandText = "SELECT DISTINCT  客户  From {订单}"
dt = cmd.ExecuteReader()
cmb = e.Form.Controls("ComboBox1")
cmb.ComboList= dt.GetComboListString("客户")
```

上述代码可以设置在窗口的 AfterLoad 事件中。

示例二

同样有了 ExecuteReader,就可以直接利用后台数据给列生成下拉目录树：

```
'生成数据表
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.ConnectionName = "region"
cmd.CommandText = "SELECT DISTINCT 省,市县,区号,邮编 From {行政区域}"
dt = cmd.ExecuteReader()
'生成目录树
Dim tb As New DropTreeBuilder
tb.SourceTable = dt
tb.TreeCols = "省|市县"
tb.SourceCols = "省|市县|区号|邮编"
tb.ReceiveCols = "省|市县|区号|邮编"
Tables("客户").Cols("省").DropTree = tb.Build()
```

示例三

同样利用 ExecuteReader，可以直接从后台提取数据合并到某个表中：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim Cols1() As String = {"来源列一","来源列二","来源列三"}
Dim Cols2() As String = {"接收列一","接收列二","接收列三"}
cmd.CommandText = "SELECT * From {表 A} Where 条件语句"
dt = cmd.ExecuteReader()
For Each dr1 As DataRow In dt.DataRows
    Dim dr2 As DataRow = DataTables("表 B").AddNew()
    For i As Integer = 0 To Cols1.Length -1
        dr2(Cols2(i)) = dr1(Cols1(i))
    Next
Next
```

示例四

前面提到 ExecuteReader 生成的 DataTable 会在代码运行结束后自动销毁。

如果希望 DataTable 在代码结束后能够继续存在，可以使用 Public 变量引用这个 DataTable。

例如首先在全局代码中定义一个 DataTable 类型的 Public 变量：

```
Public dt As DataTable
```

然后在项目事件 AfterOpenProject 中设置代码，将生成的 DataTable 赋值给此变量：

```
Dim cmd As New SQLCommand
cmd.CommandText = "SELECT DISTINCT 客户 From {订单}"
dt = cmd.ExecuteReader()
```

这样就可以在任何位置的代码引用此变量：

```
Dim cmb As WinForm.ComboBox
cmb = e.Form.Controls("ComboBox1")
cmb.ComboList= dt.GetComboListString("客户")
```

示例五

ExecuteReader 有一个可选参数，如果设置为 True，那么生成的 DataTable 不仅可以修改，还可以保存。
新建一个文件，在命令窗口测试下面的代码，会在 A 的第一行的第一列写入 123：

```
Dim cmd As new SQLCommand
Dim dt As DataTable
cmd.CommandText = "select * From {表A}"
dt = cmd.ExecuteReader(True) '记得将参数设置为True
dt.DataRows(0)("第一列") = 123
dt.Save()
DataTables("表A").Load() '重新加载表A,看看值是否已经变化
```

有了这个特性，很多时候，我们不再需要合成Update语句来更新没有加载的后台数据了。

例如要将订购数量超过100的订单的折扣，统一设置为0.05，代码为：

```
Dim cmd As new SQLCommand
Dim dt As DataTable
cmd.CommandText = "Select [_Identify],折扣  From {订单} Where  数量  >= 500"
dt = cmd.ExecuteReader(True) '记得将参数设置为True
For Each dr As DataRow In dt.DataRows
   dr("折扣") = 0.05
Next
dt.Save()
```

提示：

1、Select语句只加载需要的列，这样可以提高运行速度。
2、如果只选择部分列，那么必须包括主键列，否则保存的时候会出错！

4.15.3.2 参数化 SQLCommand

随着 Foxtable 开始提供 OpenQQ 和 HttpServer 等后台服务功能，直接使用 SQL 语句读写后台数据源的场合会越来越多。

提示：如果需要使用 SQL 直接读写后台数据，请务必使用外部数据源。
何为参数化
假定你用的是 SQL Server 数据源，我们在订单表增加一行的 SQL 语句为：
Insert Into 订单 (客户，日期，订单编号) Values('01', '2018-05-01','100')
在 Foxtable 中编写的代码为：

```
Dim cid As String = "01"
Dim rqi As Date = Date.Today
Dim oid As String = "100"
```

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Insert Into 订单 (客户, 日期, 订单编号) Values('" & cid & "', '" & Date.Today & "','" & oid & "')"
cmd.ExecuteNonQuery
```

是不是有点头晕？ 其实我和你一样头晕，这里仅用了 3 列作为示例，实际开发时可能有数十列，这样的写法就是职业成员也会崩溃的。
我们可以换一种写法，采用参数化方式，代码为：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Insert Into 订单 (客户, 日期, 订单编号) Values(?,?,?)"
cmd.Parameters.Add("@客户","01")
cmd.Parameters.Add("@日期",Date.Today)
cmd.Parameters.Add("@订单编号",100)
cmd.ExecuteNonQuery
```

显然这种方式要轻松很多，再多的列也不会让您头晕了。

参数化SQLCommand很简单，将SQL语句中需要动态取值的参数全部用问号(?)代替,然后通过SQLcommand的Parameters集合按顺序指定各参数的值。

提示：我在编写以上示例代码的时候，单独建立了一个外部表，一开始正常，但很快发现再也无法增加行了，我诧异了几分钟后，才想起来外部表默认只加载10行数据，所以上述代码都正常增加了行，只是没有加载而已，特此提醒，免你犯同样的错误。

重要的是顺序

参数的名称并无意义，重要的是顺序，如果你将代码改为：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Insert Into 订单 (客户, 日期, 订单编号) Values(?,?,?)"
cmd.Parameters.Add("@日期",Date.Today)
cmd.Parameters.Add("@客户","01")
cmd.Parameters.Add("@订单编号",100)
cmd.ExecuteNonQuery
```

这里只是将日期和客户两个参数的增加顺序调整了一下，看起来不应该有问题，但是运行却会出错，我们来分析一下原因。

下面的代码看起来是给日期列指定值:

```
cmd.Parameters.Add("@日期",Date.Today)
```

但其实不然，因为这是第一个增加的参数，所以这个参数会赋值给客户列（第一个需要指定值的列），同样下面的代码：

```
cmd.Parameters.Add("@客户","01")
```

会将"01"赋值给日期列（第二个需要指定值的列），因为这是第二个增加的参数。

所以编写代码的时候，必须严格按照SQL语句中的参数顺序赋值。

由于参数是根据顺序而不是名称赋值，所以参数名称 并不会影响代码的实际执行，以下代码完全可以正常运行：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Insert Into 订单 (客户, 日期, 订单编号) Values(?,?,?)"
cmd.Parameters.Add("@日期","01")
cmd.Parameters.Add("@客户",Date.Today)
cmd.Parameters.Add("@p",100)
cmd.ExecuteNonQuery
```

虽如此，但是给参数指定一个有描述性的名称是必须的，因为这有助于今后的维护。

更多例子

所有的语句都可以使用参数化，例如用 Delete 语句删除指定编号的订单：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Delete From 订单 Where 订单编号 = ?"
cmd.Parameters.Add("@订单编号",100)
cmd.ExecuteNonQuery
```

例如用Update语句将数量大于500的订单的折扣设置为0.15：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "UPDATE 订单 SET 折扣 = ? WHERE 数量 > ?"
cmd.Parameters.Add("@折扣",0.15)
cmd.Parameters.Add("@数量",500)
cmd.ExecuteNonQuery
```

用Select语句获取指定日期范围内的订单：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "SELECT * FROM {订单} WHERE 日期 >= ? AND 日期 <= ?"
cmd.Parameters.Add("@开始日期",#2/1/2018#)
```

```
cmd.Parameters.Add("@结束日期",#3/31/2018#)
Dim dt As DataTable = cmd.ExecuteReader()
```

4.15.3.3 执行存储过程

一般用户请忽略本节内容。
假定我们在 SQL Server 创建了一个存储过程：

```
CREATE PROCEDURE GetOrders
    @客户  nvarchar(8)
AS
BEGIN
    SELECT * FROM  订单  WHERE  客户=@客户
END
```

执行此存储过程有两种方法。
方法一
可以直接用存储过程名执行，此时需要将 StoredProcedure 属性设置为 True：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "GetOrders" '指定存储过程名
cmd.StoredProcedure = True '表示CommandText内容不是标准的SQL语句,而是存储过程名
cmd.Parameters.Add("@客户","联想")
Dim dt As DataTable = cmd.ExecuteReader
```

方法二
也可以用 EXEC 语句执行存储过程：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "EXEC GetOrders ?"
cmd.Parameters.Add("@客户","联想")
Dim dt As DataTable = cmd.ExecuteReader
```

下面的语句：

```
EXEC GetOrders ?
```

表示执行存储过程GetOrders，这个存储过程需要一个参数（SQL语用"?"表示参数，多个参数用逗号分隔）。

存储过程的三种参数

存储过程有两种参数，分别为输入参数和输出参数。
有的存储过程执行完毕后会返回一个值，这个值需要通过参数获取，我们称之为返回参数。

为区分这是这三种参数，首先需要了解一下 SQLCommand 的 Parameters 属性的 Add 方法的完整语法：

Add(Name, Value)

Add(Name, Value, Output)

Add(Name, Value, Output, Size)

Name	参数名称
Value	参数的值
Output	逻辑型，省略表示输入参数，True 表示输出参数，False 表示返回参数。
Size	整数型，用于指定输入参数或返回参数的大小，默认为 32，只有字符型的参数才需要设定。

增加输入参数最为简单，例如：

```
cmd.Parameters.Add("@客户","联想")
```

增加输出参数需要将Add方法的第三个参数设置为True，例如：

```
cmd.Parameters.Add("@日期",Date.today,True)
```

增加返回参数需要将Add方法的第三个参数设置为False，例如：

```
cmd.Parameters.Add("@Count", 0, False)
```

接下来我们开始介绍如何使用输出参数和返回参数。

获取输出参数的值

假定我们创建了一个存储过程，用于获取指定编号订单的日期：

```
CREATE PROCEDURE GetOrderDate
    @订单编号  nvarchar(8),
    @日期  datetime OUTPUT
AS
BEGIN
    SELECT @日期=日期  FROM  订单  WHERE  订单编号=@订单编号
END
```

这个存储过程有两个参数，第一个为输入参数，用于指定订单编号，第二个为输出参数，用于获取该编号订单的日期。

在Foxtable使用上述存储过程的代码为：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "GetOrderDate"
cmd.StoredProcedure = True
cmd.Parameters.Add("@订单编号","1002")
cmd.Parameters.Add("@日期",Date.today,True)
```

```
cmd.ExecuteNonQuery
Dim Val As Date   = cmd.Parameters("@日期") '获取存储过程返回的日期值
```

注意增加输出参数必须将Add方法的第三个参数设置为True，例如：

```
cmd.Parameters.Add("@日期",Date.today,True)
```

如果采用SQL语句执行，需要在语句中将有关参数标记为output，例如：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "exec GetOrderDate ?,? output" '注意第二个参数被标记为output了。
cmd.Parameters.Add("@订单编号","1002")
cmd.Parameters.Add("@日期",Date.today,True)
cmd.ExecuteNonQuery
Dim Val As Date   = cmd.Parameters("@日期")
```

获取存储过程的返回值

假定我们创建了一个存储过程，用于获取指定日期某客户的全部订单，并返回订单数：

```
CREATE PROCEDURE GetOrders3 (@CID nvarchar(10), @DT datetime)
AS
SELECT * FROM {订单} Where  客户  = @CID AND  日期=@DT
RETURN @@RowCount
```

在Foxtable执行以上存储过程的参考代码：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "GetOrders3"
cmd.StoredProcedure = True
cmd.Parameters.Add("@Count",0,False) '这是第一个参数,用于获取存储过程的返回值
cmd.Parameters.Add("@客户","Dell") '第二个参数指定客户
cmd.Parameters.Add("@日期",#4/13/2018#) '第三个参数指定日期
Dim dt As DataTable = cmd.ExecuteReader '获取订单生成DataTable
Dim count As Integer = Cmd.Parameters("@Count") '取得存储过程的返回值
```

注意增加返回参数必须将Add方法的第三个参数设置为False，例如：

```
cmd.Parameters.Add("@Count",0,False) '这是第一个参数,用于获取存储过程的返回值
```

如果采用SQL语句执行，参考代码为：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
```

```
cmd.CommandText = "exec ?=GetOrders3 ?,?"
cmd.Parameters.Add("@Count",0,True) '这是第一个参数,用于获取存储过程的返回值
cmd.Parameters.Add("@客户","Dell") '第二个参数指定客户
cmd.Parameters.Add("@日期",#4/13/2018#) '第三个参数指定日期
Dim dt As DataTable = cmd.ExecuteReader '获取订单生成DataTable
Dim count As Integer   = Cmd.Parameters("@Count") '取得存储过程的返回值
```

以上代码采用SQL语句执行存储过程：

```
exec ?=GetOrders3 ?,?
```

这里有三个参数，等号左边的参数用于获取存储过程的返回值（但此时不是返回参数，而是输出参数，只有直接执行存储过程时才有返回参数一说），等号右边的是两个输入参数，分别用于指定客户和日期。

三种参数混合使用示例

假定我们定义了如下的存储过程 ，用户获取指定编号订单的客户名称：

```
CREATE PROCEDURE GetOrderByID
  @订单编号  varchar(8),
  @客户  nvarchar(50) OUTPUT
AS
BEGIN
  SELECT @客户=客户  FROM  订单  WHERE  订单编号=@订单编号
  IF @@Error <> 0
   RETURN -1
  ELSE
   RETURN 100
END
```

在Foxtable执行上述存储过程的参考代码：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "GetOrderByID"
cmd.StoredProcedure = True
cmd.Parameters.Add("@Err", 12, False) '返回参数
cmd.Parameters.Add("@订单编号","1002") '输入参数
cmd.Parameters.Add("@客户", "", True) '输出参数
cmd.ExecuteNonQuery
Output.Show( cmd.Parameters("@客户"))
Output.Show( cmd.Parameters("@Err"))
```

需要特别注意的是：返回参数必须作为第一个参数添加，否则会出错。

采用SQL语句执行上述存储过程的参考代码：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Exec ?=GetOrderByID ?,? output"
cmd.Parameters.Add("@Err", 12, True) '用于获取存储过程的返回值,注意这是输出参数,不是输入参数
cmd.Parameters.Add("@订单编号","1002") '输入参数
cmd.Parameters.Add("@客户", "", True) '输出参数
cmd.ExecuteNonQuery
Output.Show( cmd.Parameters("@客户"))
Output.Show( cmd.Parameters("@Err"))
```

4.15.3.4 获取自动增量主键

本节内容仅适用于 SQL Server 数据源。
通过执行 Insert 语句可以直接在后台增加行，如果需要后续处理，或者在多用户环境下，通知其他用户追载此行，那么我们就需要知道新增行的主键。
对于非自动增量的主键，因为我们增加行的时候，就已经确定了主键，所以不存在获取主键的问题。
对于自动增量型主键，主键值是由后台数据库自动生成的，需要采用特殊手段来获取。
我们可以在 Insert 语句之后合并一条 Select 语句：Select SCOPE_IDENTITY()
两条语句之间用分号隔开，例如：
例如：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "Orders"
cmd.CommandText = "Insert Into 订单 (客户, 日期, 数量) Values(?,?,?);Select SCOPE_IDENTITY()"
cmd.Parameters.Add("@客户","联想")
cmd.Parameters.Add("@日期",Date.Today)
cmd.Parameters.Add("@数量",100)
Dim id As Integer = cmd.ExecuteScalar()
Output.Show("新增行主键:" & id)
```

4.15.4 SQL 应用实例

4.15.4.1 自行编码合并外部数据

我们知道，用 Merger 可以合并外部数据到表中，不过 Merger 没有排除重复内容的选项，有时不能满足我们的要求。

例如项目中有一个员工表，要从一个外部文件中合并数据到这个员工表中，希望在合并过程中检查身份证号码，如果员工表已经存在相同身份证号码的行，则跳过此行。
假定被合并的表和员工表的结构相同，代码如下：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "SELECT * From {员工}"
dt = cmd.ExecuteReader()
For Each dr As DataRow In dt.DataRows
    If DataTables("员工").Find("身份证号码 = '" & dr("身份证号码") & "'") Is Nothing Then
        Dim nr As DataRow = DataTables("员工").AddNew()
        For Each dc As DataCol In dt.DataCols
            nr(dc.name) = dr(dc.name)
        Next
    End If
Next
```

可见要自己设计一个功能更强的 Merger，其实是件很简单的事情。
如果两个表的结构不同，可以参考下面的代码：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim sCols() As String = {"列名 1", "列名 2", "列名 3"}
Dim dCols() As String = {"列名 A", "列名 B", "列名 C"}
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "SELECT * From {员工}"
dt = cmd.ExecuteReader()
For Each dr As DataRow In dt.DataRows
    Dim nr As DataRow = DataTables("员工").AddNew()
    For i As Integer =0 To sCols.Length -1
        nr(dCols(i)) = dr(sCols(i))
    Next
Next
```

4.15.4.2 获取 SQL Server 服务器时间

由于用户可以修改计算机的系统时间，所以根据系统时间进行一些条件判断，不是很可靠，存在“作弊”的可能。
如果你采用 SQL Server 作为数据源，可以用下面的代码获得 SQL Server 服务器的时间：

```
Dim cmd As New SQLCommand
Dim dt As Date
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select GetDate()"
dt = cmd.ExecuteScalar()
```

如果只想获得时间的日期部分，代码为：

```
Dim cmd As New SQLCommand
Dim dt As Date
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select GetDate()"
dt = cmd.ExecuteScalar()
dt = dt.Date
```

4.15.4.3 用 SQL 语句实现自动输入

在学习事件编程的时候，我们接触到了自动输入的例子。
这个例子有一个问题，就是必须将行政区域表的所有数据加载到 Foxtable，上述代码才能完成自动输入的任务，由于中国的县级行政区域不过 2000 多个，所以这不算大的问题。
如果基础数据表不是行政区域，而是一个有几万行的表，那么仅仅为了实现自动输入而将整个表加载到 Foxtable 中是得不偿失的，如果有几十万甚至过百万行的数据，那么加载整个表根本就是不现实的。
所以本节我们要学习在不加载基础数据表的情况下，如何实现自动输入。
本节的例子可以参考 CaseStudy 目录下的文件：SQL 与自动输入.Table。
该文件使用外部数据源作为基础数据表，数据源文件为：行政区域.mdb。
设计步骤
1、首先省市列的列表项目是固定的，所以我们在项目事件 AfterOpenProject 加入代码：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.ConnectionName = "行政区域"
cmd.CommandText = "SELECT DISTINCT 省市 From {行政区域}"
dt = cmd.ExecuteReader
Tables("客户").Cols("省市").Combolist = dt.GetComboListString("省市")
```

这样打开项目后，会从后台的行政区域表提取不重复的省市值作为客户表省市列的列表项目，尽管行政区域表并没有加载。
2、在客户表的 PrepareEdit 事件中加入代码：

```
If e.IsFocusCell Then
    If e.Col.Name = "县市" Then
        Dim cmd As New SQLCommand
        Dim dt As DataTable
        cmd.ConnectionName = "行政区域"
        cmd.CommandText = "SELECT DISTINCT 县市 From {行政区域} Where [省市] = '" & e.Row("省市") & "'"
        dt = cmd.ExecuteReader
        e.Col.Combolist = dt.GetComboListString("县市")
    End If
End If
```

这样在编辑客户表的县市列之前，会从后台的行政区域表提取该省市的县市值，作为县市列的列表项目使用。

3、最后将客户表 DataColChanged 事件的代码设置为：

```
If e.DataCol.Name = "省市" Orelse e.DataCol.Name = "县市" Then
    Dim cmd As New SQLCommand
    Dim dt As DataTable
    Dim dr As DataRow = e.DataRow
    cmd.ConnectionName = "行政区域"
    cmd.CommandText = "SELECT * From {行政区域} Where [省市] = '" & dr("省市") & "' And [县市] = '" & dr("县市") & "'"
    dt = cmd.ExecuteReader
    If dt.DataRows.Count > 0 Then
        dr("区号") = dt.DataRows(0)("区号")
        dr("邮编") = dt.DataRows(0)("邮编")
    Else
        dr("区号") = Nothing
        dr("邮编") = Nothing
    End If
End If
```

这样一旦在客户表的某行输入省市和县市两列内容后，会从后台的行政区域表查找对应的行，如果找到，则将该行的区号和邮编写入客户表中，否则清除区号和邮编。

4.15.4.4 用 SQL 语句设计加载树

Foxtable 内置了加载树功能，可以轻松地管理大容量数据。

本节我们将自行编写代码设计一个加载树，并非多此一举，毕竟有时我们会有一些特殊的要求，内置的加载树不一定能满足我们的需要。

本示例可以参考 CaseStudy 目录下的文件“动态加载演示.Table”

假定你有一个订单表，也许有几百个产品，上十万个订单，那么将其全部载入到 Foxtable 中是不现实的。

下面我们将设计一个目录树列出所有的产品和客户，当我们双击某个产品或客户时， 自动加载对应的订单。

实现这样的功能，其实很简单：

1、首先参考限制初始加载量，使得打开项目的时候，订单表不会加载任何数据。

本示例使用的是内部数据表，为了初始不加载任何定订单数据，我们在项目事件 BeforeLoadInnerTable 设置了如下的代码：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

2、然后新建一个窗口，窗口中插入一个目录树(TreeView),窗口的 AfterLod 事件代码设为：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
```

```
cmd.CommandText = "SELECT DISTINCT 产品,客户 From {订单}"
dt = cmd.ExecuteReader()
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt,"产品|客户")
trv.Nodes.Insert("加载所有数据",0)
```

这样一旦打开窗口，该目录树列出所有的产品和客户。
由于订单表只是加载了部分数据，所以我们不能直接根据订单表来生成目录树，因为订单表的数据不完整，只能用 ExecuteReader 方法直接从后台提取完整的产品和客户数据。
注意 Select 语句中，一定要有 DISTINCT 关键词，这样返回的才是不重复的产品和客户名单，虽然对目录树的生成没有影响，但是会大大降低数据加载量，对于网络环境，这是非常重要的。
3、最后将目录树的 NodeMouseDoubleClick 事件设为：

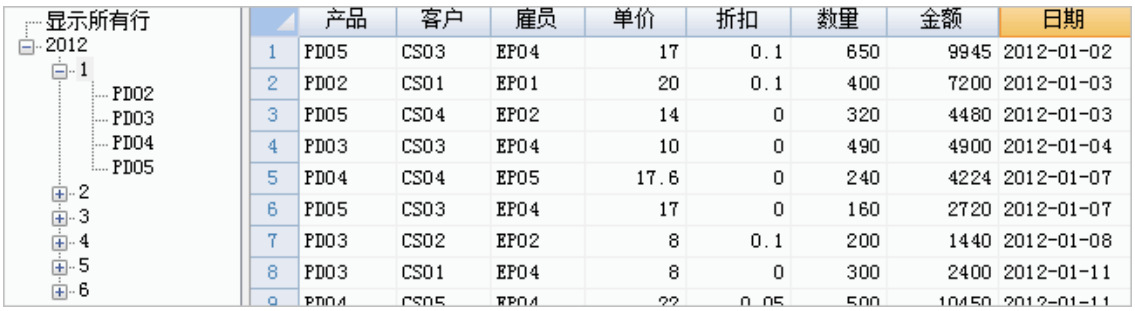
```
Dim Filter As String
If e.node.Text = "加载所有数据" Then
    Filter = ""
Else
    Dim dr As DataRow = e.Node.DataRow '获取生成此节点的DataRow
    Select Case e.Node.Level
      Case 0
        Filter ="[产品] = '" & dr("产品") & "'"
      Case 1
        Filter ="[产品] = '" & dr("产品") & "' And [客户] = '" & dr("客户") & "'"
    End Select
End If
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

这样我们只需双击某个节点，既可动态加载对应的订单。



日期列与加载树

如果需要根据年月来加载不同的产品：



	产品	客户	雇员	单价	折扣	数量	金额	日期
1	PD05	CS03	EP04	17	0.1	650	9945	2012-01-02
2	PD02	CS01	EP01	20	0.1	400	7200	2012-01-03
3	PD05	CS04	EP02	14	0	320	4480	2012-01-03
4	PD03	CS03	EP04	10	0	490	4900	2012-01-04
5	PD04	CS04	EP05	17.6	0	240	4224	2012-01-07
6	PD05	CS03	EP04	17	0	160	2720	2012-01-07
7	PD03	CS02	EP02	8	0.1	200	1440	2012-01-08
8	PD03	CS01	EP04	8	0	300	2400	2012-01-11
9	PD04	CS05	EP04	22	0.05	500	10450	2012-01-11

可以如下修改代码。

1、窗口的 AfterLoad 事件代码改为：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.CommandText = "SELECT DISTINCT Year(日期) As 年, Month(日期) As 月, 产品 From {订单}"
dt = cmd.ExecuteReader()
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt, "年|月|产品")
```

2、目录树的 NodeMouseDoubleClick 事件代码改为：

```
Dim dr As DataRow = e.Node.DataRow '获取生成此节点的行
Dim Filter As String
Dim Year As Integer = dr("年")
Dim Month As Integer = dr("月")
Dim Product As String = dr("产品")
Select Case e.Node.Level
    Case 0
        Filter ="Year(日期) = " & Year
    Case 1
        Filter ="Year(日期) = " & Year & " And Month(日期) = " & Month
    Case 2
        Filter ="Year(日期) = " & Year & " And Month(日期) = " & Month & " And 产品 = '" & Product & "'"
End Select
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

4.15.4.5 设计带分页的加载树

本节内容参考 CaseStudy 目录下的示例文件：分页加载树.table

上一节已经讲述如何自行编码设计加载树。

这一节更进一步，在加载树的基础上加上分页加载：



设计步骤

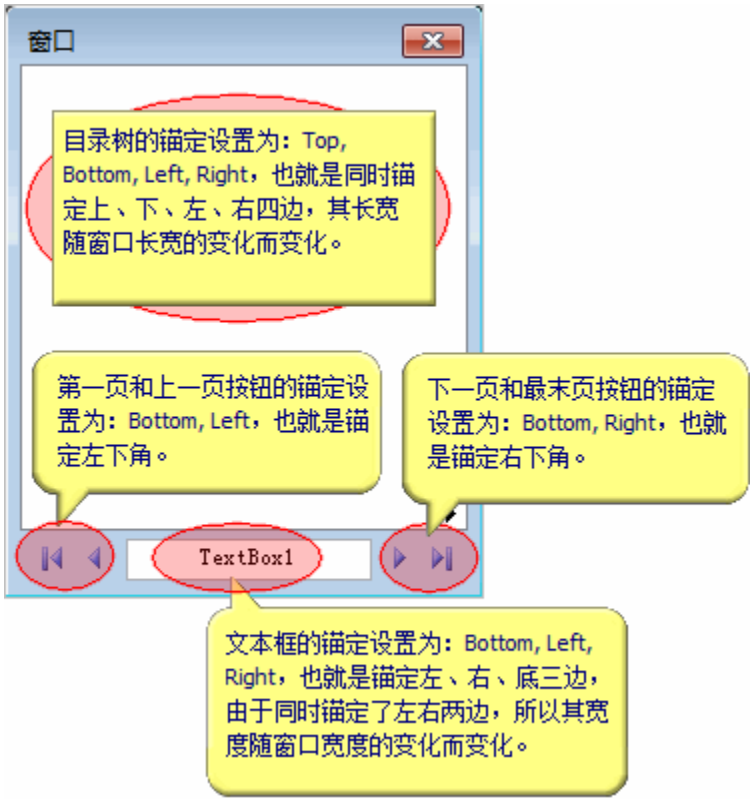
1、这个示例文件使用的是外部数据源，数据文件为 CaseStudy“订单.mdb”。

2、首先为了初始不加载任何数据，我们在 BeforeLoadOuterTable 事件中设置代码:

```
If e.DataTableName = "订单" Then
    e.SelectString = "Select * From  订单  Where [编号] Is Null"
End If
```

提示： 这个订单表的主键并非我们常用的"_Identify"，而是一个名为"编号"的自动增量列，因为主键不可能为空，所以上述代码使得打开项目后，订单表不会加载任何数据。

3、设计好一个下图所示的窗口，注意要合理设置各个控件的锚定属性，确保窗口大小发生变化的时候，各控件依然能显示在正确的位置上，各控件的锚定设置如下图：



你可能留意到各按钮的外观和常规的按钮不一样，设计这种平面按钮很简单，只需将按钮的类型设置为Flat，并将边框厚度设置为0。

4、设置窗口的AfterLoad事件代码为：

```
'生成目录树
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.ConnectionName = "test"
cmd.CommandText = "SELECT DISTINCT 产品,客户 From {订单}"
dt = cmd.ExecuteReader()
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt,"产品|客户")
trv.Nodes.Insert("加载所有数据",0)

'加载第一页数据
With DataTables("订单")
    .LoadFilter = "" '清除加载条件
    .LoadPage = 0 '加载第一页
    .LoadTop = 5 '每页5行
    .Load()
    e.Form.Controls("TextBox1").Value = 1 & "/" & .TotalPages
End With
```

5、设置目录树的NodeMouseDoubleClik事件代码为：

```
Dim Filter As String
If e.node.Text = "加载所有数据" Then
    Filter = ""
Else
    Dim dr As DataRow = e.Node.DataRow
    Select Case e.Node.Level
        Case 0
            Filter ="[产品] = '" & dr("产品") & "'"
        Case 1
            Filter ="[产品] = '" & dr("产品") & "' And [客户] = '" & dr("客户") & "'"
    End Select
End If
With DataTables("订单")
    .LoadFilter = Filter '设置加载条件
    .LoadPage = 0 '加载第一页
    .LoadTop = 5 '每页5行
    .Load()
    e.Form.Controls("TextBox1").Value = 1 & "/" & .TotalPages
End With
```

6、设置各分页按钮的代码：

按钮	代码
第一页	With DataTables("订单") .LoadPage = 0 '加载第一页 .Load() e.Form.Controls("TextBox1").Value = 1 & "/" & .TotalPages End With

上一页	With DataTables("订单")
	If .LoadPage > 0 Then
	.LoadPage = .LoadPage - 1
	.Load()
	e.Form.Controls("TextBox1").Value = (.LoadPage + 1) & "/" & .TotalPages
	End If
	End With

下一页	With DataTables("订单") If .LoadPage < .TotalPages - 1 .LoadPage = .LoadPage + 1 .Load() e.Form.Controls("TextBox1").Value = (.LoadPage + 1) & "/" & .TotalPages End If

	End With

最末页	With DataTables("订单")
	  If .LoadPage < .TotalPages - 1
	  .LoadPage = .TotalPages - 1
	  .Load()
	  e.Form.Controls("TextBox1").Value = .TotalPages & "/" & .TotalPages
	  End If
	End With

4.15.4.6 在加载树显示统计数据

本节的示例，可以参考 CaseStudy 目录下的文件：统计加载树.Table
假定我们要设计一个下图所示的加载树，和普通的加载树不同，这个加载树的年、月节点不是纯数字的，会显示"年"、"月"字样，而且每个节点会显示该节点下的订单总数：



	产品	客户	雇员	单价	数量	金额	日期
1	PD05	CS03	EP04	30	650	19500	2012-01-02
2	PD03	CS04	EP01	13.5	610	8235	2012-01-02
3	PD02	CS04	EP04	25.5	342	8721	2012-01-02
4	PD01	CS01	EP05	18	968	17424	2012-01-03
5	PD02	CS02	EP04	27.5	733	20157.5	2012-01-03
6	PD03	CS03	EP04	12.5	595	7437.5	2012-01-03
7	PD05	CS02	EP02	28.5	698	19893	2012-01-04
8	PD01	CS01	EP04	18	10	180	2012-01-04
9	PD05	CS01	EP02	28.5	554	15789	2012-01-04
10	PD03	CS05	EP03	12.5	247	3087.5	2012-01-04
11	PD05	CS01	EP05	31.5	243	7654.5	2012-01-04
12	PD03	CS04	EP02	13	227	2951	2012-01-04
13	PD02	CS04	EP04	27	832	22464	2012-01-04
14	PD03	CS03	EP05	13.5	285	3847.5	2012-01-04
15	PD03	CS02	EP04	12.5	706	8825	2012-01-05
16	PD01	CS05	EP02	17.5	472	8260	2012-01-06

设计步骤：

1、新建一个窗口，窗口类型设为主窗口，在窗口中插入一个分区面板。
2、在右分区插入一个 Table(表)，将其绑定到主表(订单表)，并将其停靠属性设置为 Fill。
3、在左分区插入一个 TreeView(目录树)，并将其停靠属性设置为 Fill。
4、将分区面板的停靠属性也设为 Fill。
5、将窗口的 AfterLoad 事件设为：

```
'首先生成统计表
Dim dt As DataTable
Dim b As New SQLGroupTableBuilder("统计表1","订单")
b.Groups.AddDef("日期",DateGroupEnum.Year,"年")
b.Groups.AddDef("日期",DateGroupEnum.Month,"月")
b.Groups.AddDef("产品")
b.Totals.AddDef("_Identify",AggregateEnum.Count,"订单数") '根据主键列来统计记录数，也就是
订单数
dt = b.Build(True) '参数设置为True,生成一个临时表,不在主界面显示
```

```
'然后根据统计表生成目录树
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt, "年|月|产品")
trv.StopRedraw
For Each nd As WinForm.TreeNode In trv.AllNodes
    Dim Year As Integer = nd.DataRow("年")
    Dim Month As Integer = nd.DataRow("月")
    Dim Product As String = nd.DataRow("产品")
    Select Case nd.Level
        Case 0
            nd.Text = nd.text & "年(" & dt.Compute("Sum(订单数)","年 = " & Year) & "条)"
        Case 1
            nd.Text = nd.text & "月(" & dt.Compute("Sum(订单数)","年 = " & Year & " And 月 = " & Month) & "条)"
        Case 2
            nd.Text = nd.text & "(" & nd.DataRow("订单数") & ")"
    End Select
Next
trv.Nodes.Insert("加载所有行","加载所有行(" & dt.Compute("Sum(订单数)") & "条)",0)
trv.ResumeRedraw
```

上面的代码中，请留意第 8 行代码：

dt = b.Build(True) '参数设置为 True,生成一个临时表,不在主界面显示

加上参数 True 后，Build 生成的表，不会在主界面显示，当不再有地方引用这个表或这个表中的行和列时，这个表就会被自动销毁，参考不显示生成的表。

以本节的例子来说，由于根据这个表生成了目录树，目录树节点的 DataRow 属性会引用这个表中的行，所以直到关闭窗口，这个表才会被自动销毁。

6、将目录树的 NodeMouseDoubleClick 事件设为：

```
Dim Filter As String = ""
If e.Node.Name <> "加载所有行" Then '要用Name属性,不能用Text属性
    Dim Year As Integer = e.Node.DataRow("年")
    Dim Month As Integer = e.Node.DataRow("月")
    Dim Product As String = e.Node.DataRow("产品")
    Select Case e.Node.Level
        Case 0
            Filter ="Year(日期) = " & Year
        Case 1
            Filter ="Year(日期) = " & Year & " And Month(日期) = " & Month
        Case 2
            Filter ="Year(日期) = " & Year & " And Month(日期) = " & Month & " And 产品 = '" & Product & "'"
    End Select
End If
DataTables("订单").LoadFilter = Filter
DataTables("订单").Load()
```

4.15.4.7 网络环境下的复杂编号

普通用户可以忽略本节内容。

本节内容可以参考 CaseStudy 目录下的文件：网络环境下的复杂编号.table。

提醒：由于 OpenQQ 的出现，本节的内容已经过时，请参考：用 OpenQQ 实现网络环境下的编号

使用自动增量主键，例如_Identify 列，能够有效解决编号的问题，即使是多人同时向一个表中增加行，也不会出现重复的编号。

但是自动增量主键只是一个数字，无法实现复杂的含有特殊信息的编号。

如果自己编码生成编号，如何避免在多人同时操作时产生重复的编号，是一个难题。

本节用一个简单的例子，来探讨如何解决这个问题。

本节的例子使用外部数据源，数据源文件为 CaseStudy 目录下的文件:编号.mdb

该文件只包括两个表，一个订单表，一个编号表。

希望订单表的编号能够自动生成，编号的格式为：0000-0000，左边四位为前缀，两位表示年，两位表示月，右边四位表示顺序号。

编号表并非常规的数据表，不加载到 Foxtable 中，而是直接通过 SQL 语句操作。

编号表包括两列，分别是：前缀和顺序号，前者为字符型，由两位年两位月组成，后者为整数型，用于保存此前缀的当前顺序号。

设计步骤：

1、因为订单表的编号列是主键列，所以在任何时候都不能为空，必须在增加行的时候，为新增行生成一个临时的编号，为此我们将订单表的 DataRowAdding 事件设置为：

```
Static Index As Integer = 99999
e.DataRow("编号") = Format(Date.Today(),"yyMM") & "-" & Index
Index = Index - 1
```

和正式的编号不同，临时编号的顺序号为 5 位，而且是从 99999 开始倒序编号，这样用户一看就知道这是临时的编号。

2、在保存的时候，为新增的行生成正式的编号，BeforeSaveDataRow 事件代码设置为：

```
Dim dr As DataRow = e.DataRow
Dim pf As String
If dr.RowState <> DataRowState.Added Then '如果不是新增行
    Return '那么返回
ElseIf dr.IsNull("日期") Then '如果没有输入日期
    e.Cancel = True '取消保存此行
    MessageBox.Show("必须输入日期!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
Else
    pf = Format(dr("日期"),"yyMM") '否则获得编号的前缀,两位年,两位月
End If
Dim cmd1 As New SQLCommand
Dim cmd2 As New SQLCommand
Dim Key As Integer
cmd1.ConnectionName = "编号" '设置数据源名称
cmd2.ConnectionName = "编号"
cmd1.commandText = "Select Count(*) From [编号] Where [前缀] = '" & pf & "'"
```

```
If cmd1.ExecuteScalar = 0 Then '如果编号表不存在前缀的行,那么增加一行
    cmd1.commandtext = "Insert Into  编号  (前缀, 顺序号) Values('" & pf & "',1)"
    cmd1.ExecuteNonQuery
End If
cmd1.commandText = "Select [顺序号] From [编号] Where [前缀] = '" & pf & "'"
Do
    Key = cmd1.ExecuteScalar() '从后台获得顺序号
    cmd2.commandText = "Update [编号] Set [顺序号] = " & (Key + 1) & " Where [顺序号] = " & Key & " And [前缀] = '" & pf & "'"
    If cmd2.ExecuteNonQuery() > 0 Then '更新顺序号
        Exit Do '更新成功则退出循环
    End If
Loop
e.DataRow("编号") = pf & "-" & Format(Key,"0000")
```

第 10 行代码获得编号的前缀（由两位年两位月组成），第 17、18 行代码判断编号表是否存在此前缀的记录，如果不存在，第 19、20 行代码则增加此前缀的记录，并将顺序号设置为 1。

最关键的是第 22 到第 30 行代码，第 24 行代码首先从后台的编号表获得此前缀的顺序号，第 26 行代码尝试将该前缀的顺序号加 1，如果操作成功则退出循环，并在第 30 行将前缀和顺序号组合成编号写入到要保存的行中。

那么在多人同时使用的时候，是如何避免重复的呢；假定 A 和 B 几乎同时执行了第 24 行代码，获得的顺序号相同，同样又几乎同时执行第 26 行代码，假定 A 用户抢先了一点，成功将该前缀的顺序号增加了 1，随后 B 用户执行的时候，由于顺序号已经加 1，所以 Update 语句的条件不符合，导致第 26 行的 ExecuteNonQuery 方法返回 0， 无法退出循环，于是 B 用户只能重复上述操作：获得顺序号，将顺序号加 1，直到成功为止。

多表编号

上述的代码只是对一个订单表进行自动编号，如果一个项目中有多个表需要进行类似的编号，可以修改编号表的结构，包括三列，分别是：表名、前缀和顺序号，表名和前缀是字符型，顺序号是整数型。

所有需要自动编号的表的 DataRowAdding 事件代码依然为：

```
Static Index As Integer = 99999
e.DataRow("编号") = Format(Date.Today(),"yyMM") & "-" & Index
Index = Index - 1
```

而 BeforeSaveDataRow 事件代码需要稍微调整一下，加上表名的条件：

```
Dim dr As DataRow = e.DataRow
Dim pf As String
If dr.RowState <> DataRowState.Added Then '如果不是新增行
    Return '那么返回
ElseIf dr.IsNull("日期") Then '如果没有输入日期
    e.Cancel = True '取消保存此行
    MessageBox.Show("必须输入日期!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
Else
    pf = Format(dr("日期"),"yyMM") '否则获得编号的前缀,两位年,两位月
End If
Dim cmd1 As New SQLCommand
Dim cmd2 As New SQLCommand
Dim Key As Integer
Dim nm As String = e.DataTable.name
cmd1.ConnectionName = "编号" '设置数据源名称
cmd2.ConnectionName = "编号"
cmd1.commandText = "Select Count(*) From [编号] Where [前缀] = '" & pf & "' And  表名  = '" & nm & "'"
If cmd1.ExecuteScalar = 0 Then '如果编号表不存在前缀的行,那么增加一行
    cmd1.commandtext = "Insert Into  编号  (表名,  前缀,  顺序号) Values('" & nm & "','" & pf & "',1)"
    cmd1.ExecuteNonQuery
End If
cmd1.commandText = "Select [顺序号] From [编号] Where [前缀] = '" & pf & "' And  表名  = '" & nm & "'"
Do
    Key = cmd1.ExecuteScalar() '从后台获得顺序号
    cmd2.commandText = "Update [编号] Set [顺序号] = " & (Key + 1) & " Where [顺序号] = " & Key & " And [前缀] = '" & pf & "' And  表名  = '" & nm & "'"
    If cmd2.ExecuteNonQuery() > 0 Then '更新顺序号
        Exit Do '更新成功则退出循环
    End If
Loop
e.DataRow("编号") = pf & "-" & Format(Key,"0000")
```

4.15.4.8 避免多人同时编辑同一行

本节讲述如何避免多人同时编辑同一行，既如何实现独占式编辑。

提醒：由于 OpenQQ 的出现，本节的内容已经过时，请参考：用 OpenQQ 实现独占式编辑

虽然本节的内容有些过时，但是依然有参考价值，所以我们依然保留了这一节的内容，有兴趣的话，可以看看。

示例一

本示例可以参考 CaseStudy 目录下的文件：独占式编辑之一.Table

请分别启动多个 Foxtable 打开此文件，并以不同的用户登录，同一行只要有人开始编辑，那么直到此人保存此行，其他人都不能编辑此行。

设计步骤：

1、首先给表增加一个字符列，名为"编辑者"，用于记录正在编辑某行数据的用户名。

2、在项目事件 AfterOpenProject 设置代码：

```
'解锁非正常退出时被锁定的记录
DataTables("员工").ReplaceFor("编辑者",Nothing,"编辑者 = '" & User.Name & "'")
DataTables("员工").Save()
Tables("员工").Cols.Remove("编辑者") '移除编辑者列,避免用户直接修改此列
```

3、将表事件 StartEdit 代码设置为：

```
If e.Row("编辑者") = User.Name Then '如果编辑者是当前用户
    Return
End If
Dim cmd As New SQLCommand
Dim exp As String  = "编辑者 Is Null And [_Identify] = " & e.Row("_Identify")
cmd.CommandText = "Update {员工} Set 编辑者 = '" & User.Name & "' Where " & exp
If cmd.ExecuteNonQuery = 1 Then
    e.Row("编辑者") = User.Name
Else
    cmd.CommandText = "Select 编辑者 From {员工} Where [_Identify] = " & e.Row("_Identify")
    Dim nm As String = cmd.ExecuteScalar
    If nm > "" Then
        MessageBox.show(nm & " 正在编辑此行!"," 提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
        MessageBox.show(" 此行可能已经被删除，无法编辑!"," 提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
    e.Cancel = True '取消编辑
End If
```

注意这里用的是 StartEdit 事件，而不是 PrepareEdit 事件，也就是只有用户真正开始编辑数据时，才会执行上述代码。

我们来分析一下上述代码的工作原理，假定某行没有人编辑，张三和李四同时开始编辑此行，同时开始执行上述代码，如果张三抢先了一点，那么执行了 Update 语句之后，编辑者列的内容成为“张三”，由于 Update 语句设置了条件，只有编辑者列为空时，Update 语句才会执行，所以随后李四在执行 Update 语句时，由于已经没有符合条件的行，导致 ExecuteNonQuery 方法返回 0， 于是取消编辑，并用 Select 语句从后台找出正在编辑此行的用户名，告知李四。

4、将表事件 DataRowAdding 代码设置为：

```
e.DataRow("编辑者") = User.name
```

5、将表事件 BeforSaveDatarow 代码设置为：

```
If e.DataRow.RowState = DataRowState.Added Then '如果是新增行
    e.DataRow("编辑者") = Nothing
Else
    Dim cmd As New SQLCommand
    cmd.CommandText = "Update {员工} Set  编辑者  = Null Where [_Identify] = " & e.DataRow("_Identify")
    If cmd.ExecuteNonQuery = 1 Then
       e.DataRow("编辑者") = Nothing
    End If
End If
```

上述代码在保存行之前，清除此行的编辑者列内容，也就是解锁此行。

你也许会奇怪，为什么新增行可以直接清空编辑者列的内容，但其它行要通过Update语句设置？
假定现有一行数据，用户张三双击此行某个单元格开始编辑，系统执行StartEdit事件，将编辑者列的内容设置为“张三”，后台此列的内容也通过Update语句设置为了“张三”，如果此用户在编辑过程中，没有做任何修改，就选择了其他单元格，然后保存，此时如果在BeforeSaveDataRow事件中直接清空编辑者列内容，那么此行所有列的内容和最初完全一样，Foxtable将不会保存此行，所以虽然你清空了编辑者列的内容，但是后台的编辑者列还是“张三”，导致记录被锁死。
这里涉及到Foxtable的保存机制：在保存某行之前，系统会进行一个判断，如果此行所有列内容和最初完全相同，则跳过此行，以提高保存效率。

6、最后设置BeforeCloseProject事件代码为：

```
Dim cmd As New SQLCommand
cmd.CommandText = "Update {员工} Set  编辑者  = Null Where  编辑者  = '" & User.Name & "'"
cmd.ExecuteNonQuery
```

这样在关闭项目前，将清除当前用户锁定的全部行。

示例二

上面讲述的是直接在表中录入数据的时候，如何实现独占式编辑，下面讲述如何在窗口中实现独占式编辑。

本示例可以参考CaseStudy目录下的文件：独占式编辑之二.Table
请分别启动多个Foxtable打开此文件，并以不同的用户登录，只要有人双击某行开始编辑，那么直到此人结束编辑，其他人都不能编辑此行。

1、首先给表增加一个字符列，名为"编辑者"，用于记录正在编辑某行数据的用户名。
2、在项目事件 AfterOpenProject 设置代码：

```
'解锁非正常退出时被锁定的记录
DataTables("员工").ReplaceFor("编辑者",Nothing,"编辑者 = '" & User.Name & "'")
DataTables("员工").Save()
Tables("员工").Cols.Remove("编辑者") '移除编辑者列,避免用户直接修改此列
```

3、将表事件PrepareEdit代码设置为：

```
e.Cancel = True
```

4、将表事件 DataRowAdding 代码设置为：

```
e.DataRow("编辑者") = User.name
```

5、将表事件DoubleClick代码设置为：

```
If e.Row("编辑者") = User.Name Then '如果编辑者是当前用户
    Return
End If
Dim cmd As New SQLCommand
Dim exp As String  = "编辑者 Is Null And [_Identify] = " & e.Row("_Identify")
cmd.CommandText = "Update {员工} Set 编辑者 = '" & User.Name & "' Where " & exp
If cmd.ExecuteNonQuery = 1 Then
    e.Row("编辑者") = User.Name
    e.Row.Save()
    Forms("编辑窗口").Open()
Else
    cmd.CommandText = "Select 编辑者 From {员工} Where [_Identify] = " & e.Row("_Identify")
    Dim nm As String = cmd.ExecuteScalar
    If nm > "" Then
        MessageBox.show(nm & " 正在编辑此行!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
        MessageBox.show("此行可能已经被删除，无法编辑!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

上述代码的前面已经讲述，这里就不重复了。

6、设计一个下图所示的编辑窗口，注意在设计窗口的时候，应该将“显示控制框”属性设置为False，这样窗口就没有了关闭按钮，避免用户直接关闭窗口：



7、窗口“保存修改”按钮的代码为：

```
Tables("员工").Current("编辑者") = Nothing
Tables("员工").Current.Save()
e.Form.Close()
```

8、窗口“撤销修改”按钮的代码为：

```
Tables("员工").Current.Reject() '撤销编辑
Tables("员工").Current("编辑者") = Nothing
Tables("员工").Current.Save()
e.Form.Close()
```

这样不管用户保存还是撤销修改，编辑者列的内容都会被清除，让后来者得以继续编辑此行。

9、最后设置BeforeCloseProject事件代码为：

```
Dim cmd As New SQLCommand
cmd.CommandText = "Update {员工} Set  编辑者  = Null Where  编辑者  = '" & User.Name & "'"
cmd.ExecuteNonQuery
```

这样在关闭项目前，将清除当前用户锁定的全部行。

4.15.4.9 SQL 语句的替代方案

本节的内容，可以参考 CaseStudy 目录下的文件：自动输入后台数据.Table
Foxtable 对于 SQL 语句的依赖，远远低于其它开发工具，因为 Foxtable 的 DataTable 提供了一系列处理后台数据的方法，包括：

SQLCompute	计算后台所有数据
SQLReplaceFor	批量更新后台数
SQLDeleteFor	批量删除后台数据
SQLSelect	从后台查询数据
SQLUpdate	保存通过 SQLSelect 查询得到的数据
SQLFind	从后台查找数据
SQLGetValues	从后台的指定列中提取不重复值，以集合的形式返回
SQLGetComboListString	从后台的指定的列中提取不重复的值，用符号"|"将这些值连接成一个字符串，并返回这个字符串

这些方法的使用非常简单，可以部分代替原来需要试用 SQL 语句的场合，唯一的局限是基于 DataTable 运行，所以表必须已经加载，当然可以只加载表结构，不加载任何数据。
我们之前讲述过一个用 SQL 语句实现自动输入的例子，参考：用 SQL 语句实现自动输入
接下来我们不用 SQL 语句，采用上面的方法来实现同样的功能，且同样不加载行政区域表的数据 ，你会看到新的方法代码要简洁很多。
设计步骤：
1、在 BeforeLoadInnerTable 中设置代码：

```
If e.DataTableName = "行政区域" Then
    e.Filter = "[_Identify] Is Null"
End If
```

这个项目中的行政区域表是内部数据表，上述的代码使得打开项目后，行政区域表不会加载任何数据。
如果是外部表，参考: BeforeLoadOuterTable
2、省市列的列表项目是固定的，在项目事件 AfterOpenProject 加入代码：

```
Tables("客户").Cols("省市").Combolist = DataTables("行政区域").SQLGetComboListString("省市")
' Tables("行政区域").Visible = True '实际使用的时候,可以启用此行代码,以隐藏行政区域表
```

这样打开项目后，会从后台的行政区域表提取不重复的省市值作为客户表省市列的列表项目，尽管行政区域表并没有加载数据。

3、在客户表的PrepareEdit事件中加入代码：

```
If e.IsFocusCell Then '如果是焦点单元格
```

```
    If e.Col.Name = "县市" Then '如果正在编辑的是县市列
        e.Col.Combolist = DataTables("行政区域").SQLGetComboListString("县市", "[省市] = '" & e.Row("省市") & "'")
    End If
End If
```

这样在编辑客户表的县市列之前，会从后台的行政区域表提取该省市的县市值，作为县市列的列表项目使用。

4、最后将客户表DataColChanged事件的代码设置为：

```
If e.DataCol.Name = "省市" OrElse e.DataCol.Name = "县市" Then
    Dim dr As DataRow = e.DataRow
    Dim sr As DataRow
    sr = DataTables("行政区域").SQLFind("[省市] = '" & dr("省市") & "' And [县市] = '" & dr("县市") & "'") '在后台查找符合条件的行
    If sr IsNot Nothing Then '如果找到了符合条件的行
        dr("区号") = sr("区号")
        dr("邮编") = sr("邮编")
    Else
        dr("区号") = Nothing
        dr("邮编") = Nothing
    End If
End If
```

这样一旦在客户表的某行输入省市和县市两列内容后，会从后台的行政区域表查找对应的行，如果找到，则将该行的区号和邮编写入客户表中，否则清除区号和邮编。

从这个例子可以看出，采用 Foxtable 内置的处理后台数据的方法，和直接编写 SQL 语句相比，代码要简洁得多，也不需要改变原来的编程习惯。

另一种执行方式

前面已经提到 Foxtable 所有后台数据处理函数，都要基于 DataTable 运行，所以上面的示例中，我们加载了一个空的行政区域表，然后基于此空表处理后台的行政区域表的数据。

虽然我们可以通过代码在 Foxtale 的主界面中隐藏这个空表，但还是有一定程度的不便，而且有较多数量只需后台处理不需要加载数据的表时，还会加大系统对资源的消耗。

如果愿意，我们可以考虑另一种方式，通过 SQL 语句生成一个临时表，基于这个临时表执行后台数据处理方法。

首先我们在全局代码中定义一个变量：

```
Public xzqy As DataTable
```

然后在 AfterOpenProject 事件中加上代码：

```
Dim cmd As New SQLCommand
cmd.ConnectionName = "数据源名称"
```

```
cmd.CommandText = "SELECT * From {行政区域} Where [省市] Is Null"
xzqy = cmd.ExecuteReader
```

上面的代码生成一个空的临时表，保存在全局变量 xzqy 中，这个临时表对应后台的行政区域表。

现在我们就可以通过此临时表处理后台的行政区域表的数据了，例如：

```
Tables("客户").Cols("省市").Combolist = xzqy.SQLGetComboListString("省市")
```

需要注意，如果不单单是调用后台数据，还要向后台写入数据，生成临时表的时候，必须给 ExecuteReader 加上参数 True，例如：

```
Dim cmd As new SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select * From {表名} Where [_Identify] Is Null"
Dim dt As DataTable = cmd.ExecuteReader(True)
```

当然表必须有主键，才能更新其数据。

关于这种执行方式的更全面的例子，可以参考：自定义用户管理之二

4.15.4.10 垂直表和水平表的转换

在一些场合，可能需要对表的结构进行横竖转换。

示例一

假定表 A 的结构如下：

	姓名	课程	分数
1	张三	语文	74
2	张三	数学	83
3	张三	物理	93
4	李四	语文	74
5	李四	数学	84
6	李四	物理	94

希望将其转换为表 B，结构为：

	姓名	数学	物理	语文
1	李四	84	94	74
2	张三	83	93	74

转换代码为：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("姓名", Gettype(String), 32)
For Each v As String In DataTables("表A").GetValues("课程")
    dtb.AddDef(v, Gettype(Double))
Next
```

```
dtb.Build()
For Each v As String In DataTables("表A").GetValues("姓名")
    Dim dr1 As DataRow = DataTables("表B").AddNew()
    dr1("姓名") = v
    For Each dr2 As DataRow In DataTables("表A").Select("姓名 = '" & v & "'")
        dr1(dr2("课程")) = dr2("分数")
    Next
Next
MainTable = Tables("表B")
```

我们也可以 SQL 语句来实现，更加简单：

```
Select 姓名,Sum(iif(课程 = '语文',分数,0)) As 语文,Sum(iif(课程 = '数学',分数,0)) As 数学,Sum(iif(课程 = '英语',分数,0)) As 英语,Sum(iif(课程 = '物理',分数,0)) As 物理,Sum(iif(课程 = '化学',分数,0)) As 化学 From {学生成绩} Group by 姓名
```

上述 SQL 语句巧妙地利用了分组统计，希望大家细细体会一下其中的奥妙。

需要注意的是，如果使用的数据源是 SQL Server，是不能使用 iif 函数的，需要用 Case When 语句：

```
Select 姓名,Sum(Case When 课程 = '语文' Then 分数 Else 0 End) As 语文,Sum(Case When 课程 = '数学' Then 分数 Else 0 End) As 数学,Sum(Case When 课程 = '英语' Then 分数 Else 0 End) As 英语,Sum(Case When 课程 = '物理' Then 分数 Else 0 End) As 物理,Sum(Case When 课程 = '化学' Then 分数 Else 0 End) As 化学 From {学生成绩} Group by 姓名
```

我们可以新建一个查询表，或者用 QueryBuilder，将其 Select 语句设置为 上述语句即可。

示例二

接下来更复杂点，假定有下图所示的学生成绩表，在上面的基础上多了一个班级列：

	班级	姓名	科目	分数
1	1班	陈成进	语文	133
2	1班	林景阳	语文	129
3	1班	苏　捷	语文	132
4	1班	谢晓霞	语文	129
5	2班	谢杰立	语文	129
6	2班	谢晓霞	语文	134
7	2班	杨景亮	语文	132
8	2班	张李华	语文	133
9	4班	谢杰立	语文	129
10	4班	谢晓霞	语文	134
11	4班	杨景亮	语文	132
12	4班	张李华	语文	133
13	1班	陈成进	数学	136
14	1班	苏　捷	数学	136
15	1班	谢杰立	数学	133
16	2班	林景阳	数学	133
17	2班	杨景亮	数学	136
18	4班	林景阳	数学	133

需要将其转换为水平结构的表，而且在原有科目的基础上，再增加一个总分列：

	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136	140	71	75	555
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529
11	1班	张李华	124	118	124	83	88	537
12	2班	陈成进	110	58	106	36	34	344
13	2班	陈诗淇	113	123	117	78	78	509
14	2班	陈思达	113	55	92	55	46	361

可以参考下面的代码：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("班级",Gettype(String),10)
dtb.AddDef("姓名",Gettype(String),10)
For Each v As String In DataTables("学生成绩").GetValues("科目")
    dtb.AddDef(v, Gettype(Double))
Next
dtb.Build()
For Each v As String() In DataTables("学生成绩").GetValues("班级|姓名")
    Dim dr1 As DataRow = DataTables("表B").AddNew()
    dr1("班级") = v(0)
    dr1("姓名") = v(1)
    For Each dr2 As DataRow In DataTables ("学生成绩").Select("班级 = '" & v(0) & "' And 姓名 = '" &
v(1) & "'")
        dr1(dr2("科目")) = dr2("分数")
    Next
Next
DataTables("表B").DataCols.Add("总分",Gettype(Double),"[语文] + [英语] + [数学] + [物理] + [化学]")
MainTable = Tables("表B")
```

当然，我们同样也可以直接用 SQL 语句来实现：

```
Select 班级,姓名,Sum(iif(科目 = '语文',分数,0)) As 语文,Sum(iif(科目 = '数学',分数,0)) As 数
学,Sum(iif(科目 = '英语',分数,0)) As 英语,Sum(iif(科目 = '物理',分数,0)) As 物理,Sum(iif(科目 = '
化学',分数,0)) As 化学,Sum(分数) As 总分 From {学生成绩} Group by 班级,姓名
```

上述 SQL 语句巧妙地利用了分组统计，希望大家细细体会一下其中的奥妙。

需要注意的是，如果使用的数据源是 SQL Server，是不能使用 iif 函数的，需要用 Case When 语句：

```
Select 班级,姓名,Sum(Case When 科目 = '语文' Then 分数 Else 0 End) As 语文,Sum(Case When
科目 = '数学' Then 分数 Else 0 End) As 数学,Sum(Case When 科目 = '英语' Then 分数 Else 0 End)
As 英语,Sum(Case When 科目 = '物理' Then 分数 Else 0 End) As 物理,Sum(Case When 科目 = '
化学' Then 分数 Else 0 End) As 化学,Sum(分数) As 总分 From {学生成绩} Group by 班级,姓名
```

我们可以新建一个查询表，或者用 QueryBuilder，将其 Select 语句设置为 上述语句即可。

示例三

现在反过来，假定有个水平结构的表：

	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136	140	71	75	555
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529
11	1班	张李华	124	118	124	83	88	537
12	2班	陈成进	110	58	106	36	34	344
13	2班	陈诗淇	113	123	117	78	78	509
14	2班	陈思达	113	55	92	55	46	361

需要转换为垂直结构的表：

	班级	姓名	科目	分数
1	1班	陈成进	语文	133
2	1班	林景阳	语文	129
3	1班	苏　捷	语文	132
4	1班	谢晓霞	语文	129
5	2班	谢杰立	语文	129
6	2班	谢晓霞	语文	134
7	2班	杨景亮	语文	132
8	2班	张李华	语文	133
9	4班	谢杰立	语文	129
10	4班	谢晓霞	语文	134
11	4班	杨景亮	语文	132
12	4班	张李华	语文	133
13	1班	陈成进	数学	136
14	1班	苏　捷	数学	136
15	1班	谢杰立	数学	133
16	2班	林景阳	数学	133
17	2班	杨景亮	数学	136
18	4班	林景阳	数学	133

转换代码为：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("班级", Gettype(String), 4)
dtb.AddDef("姓名", Gettype(String), 10)
dtb.AddDef("科目", Gettype(String), 10)
dtb.AddDef("分数", Gettype(Double))
dtb.Build()
```

```
Dim kms() As String = {"语文","数学","英语","物理","化学"}
For Each dr1 As DataRow In DataTables("成绩表").DataRows
    For Each km As String In kms
        Dim dr2 As DataRow = DataTables("表B").AddNew()
        dr2("班级") = dr1("班级")
        dr2("姓名") = dr1("姓名")
        dr2("科目") = km
        dr2("分数") = dr1(km)
    Next
Next
MainTable = Tables("表B")
```

我们也可以用 SQL 语句实现,新建一个查询表，或者用 QueryBuilder，将其 Select 语句设置为：

Select 班级,姓名,'语文' As 科目,语文 As 分数 From {学生成绩} Union All Select 班级,姓名,'数学' As 科目,数学 As 分数 From {学生成绩} Union All Select 班级,姓名,'英语' As 科目,英语 As 分数 From {学生成绩} Union All Select 班级,姓名,'化学' As 科目,化学 As 分数 From {学生成绩} Union All Select 班级,姓名,'物理' As 科目,物理 As 分数 From {学生成绩}

上面的语句用 Union All 连接了 5 个查询，每个查询针对一个科目，以第一个科目语文为例，其 Select 语句为：

Select 班级, 姓名, '语文' As 科目,语文 as 分数 From {学生成绩}

这个 Select 语句生成的表包括四列，分别是班级、姓名、科目、分数，其中班级、语文就是原始成绩表的班级列和语文列，而科目列的内容固定为"语文"，分数列就是原始成绩表的语文列，只不过被重命名为分数列。

其它科目的查询语句一样，用 Union All 组合每个科目的查询语句，就得到我们所需要的表了。

4.15.5 编写通用代码

我们知道不同的数据源，其 SQL 语句的语法会有所差异。

最常见的差异有两个，之前已经介绍过，为了巩固，下面再复述一遍。

日期值的差异

对于内部数据源和 Access 数据源，日期用符号#括起来，例如：

SELECT * FROM {订单} WHERE 日期 >= #1/1/1999# AND 日期 <= #1/31/1999#

如果数据源是 SQl SERVER，日期常量用单引号而不是#号括起来，例如：

SELECT * FROM {订单} WHERE 日期 >= '1/1/1999' AND 日期 <= '1/31/1999'

逻辑值的差异

此外对于逻辑值，内部数据源和 Access 数据源采用 True 和 False，例如：

SELECT * FROM {订单} Where 结账 = True

如果是 SQL Server 数据源，则用 0 表示 False，1 表示 True，例如

Select * From {订单} Where 结账 = 1

表示查询出已经结账的订单。

如果要查询未结账的订单，则应该为:

Select * From {订单} Where 结账 = 0

为数据源编写通用代码

不同数据源之间的语法差异，使得我们启用一个新的数据源后，往往需要重新修改相关代码。

不过 Foxtable 很好地解决了这个问题。
数据源(Connection)有一个 SourceType 属性，整数型，用于返回数据源的类型，1 表示 Access，2 表示 SQL Server，3 表示 Oracle。
示例：

```
If Connections("nwnd").SourceType = 1 Then 'Access数据源
    Tables("窗口1_Table1").Fill("Select * From 订单 Where 日期 = #" & Date.Today & "#","nwnd",False)
ElseIf Connections("nwnd").SourceType = 2 Then 'SQL Server数据源
    Tables("窗口1_Table1").Fill("Select * From 订单 Where 日期 = '" & Date.Today & "'","nwnd",True)
End If
```

上述代码，不管你是Access还是Sql Server数据源，都能正常将当天的订单加载到窗口的Table控件中。

为DataTable编写通用代码

DataTable同样有一个SourceType属性，整数型，用于返回数据源的类型，1表示内部表或Access，2表示SQL Server，3表示Oracle。

示例

```
Dim dt As DataTable = DataTables("订单")
If dt.SourceType = 1 Then 'Access数据源
    dt.LoadFilter = "日期 = #" & Date.Today & "#"
ElseIf dt.SourceType = 2 Then 'SQL Server数据源
    dt.LoadFilter = "日期 = '" & Date.Today & "'"
End If
dt.Load()
```

上述代码，不管使用的是内部表，还是Access数据源，或者Sql Server数据源，都能正常将当天的订单加载到订单表中。

4.15.6 关于表达式列

有次我使用了下面的 SQL 语句：
Select 产品, Sum(数量) As 数量, Sum(金额) As 金额 From {订单} Group By 产品
在执行的时候，提示了这样的错误：别名“金额”循环引用！
当时我觉得很奇怪，明明没有循环引用啊！
其实这是因为金额列是一个表达式列，并不存在于后台数据库，所以正确的 SQL 语句是：
Select 产品, Sum(数量) As 数量, Sum(数量 * 单价 * (1 - 折扣)) As 金额 From {订单} Group By 产品
同样下面的 SQL 语句也是错误的：
Select 产品, 数量, 金额 From {订单}

执行的时候会提示错误：至少有一个参数没有指定值！
正确的语句是：
Select 产品, 数量, (数量 * 单价 * (1 - 折扣)) As 金额 From {订单}
表达式列不能出现在 SQL 语句中，这一点大家一定要记得。

4.15.7 再谈 INNER JOIN

一般用户可以忽略本节内容。
我们已经知道，通过 INNER JOIN 语句可以查询多个表的数据，其中的表可以是另一个 Select 语句，利用这个功能，可以完成很多用常规编码很难解决的问题。
你可以打开 CaseStudy 目录下的示例文件“统计演示.table”来测试下面的 Select 语句。
示例一
例如我们要查询出每个产品最近一次订购的日期、数量以及客户，可以使用下面的 Select 语句：

```
Select a.产品,a.客户,a.日期,a.数量 From {订单} a INNER JOIN (Select 产品,Max(日期) As 日期 From {订单} Group By 产品) b on a.产品 = b.产品 And a.日期 = b.日期
```

上面的 Select 语句，其实有两个表，分别是 a 和 b，a 表就是订单表，红色部分的语句，给订单表指定了一个别名"a"，b 表就上面绿色部分的 Select 语句，注意这段 Select 语句要用括号括起来，在括号之后指定别名“b”。
示例二
再例如，我们要查询出每个产品订购数量最大的订单，Select 语句如下：

```
Select a.产品,a.客户,a.日期,a.数量 From {订单} a INNER JOIN (Select 产品,Max(数量) As 数量 From {订单} Group By 产品) b on a.产品 = b.产品 And a.数量 = b.数量 Order By a.产品
```

4.15.8 记得设置链接名称

如果 SQLCommand 操作的是外部数据源，那么一定要设置其 ConnectionName 属性，指定外部数据源名称：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
cmd.ConnectionName = "外部数据源名称"
cmd.CommandText = "SELECT DISTINCT 产品,客户 From {订单}"
dt = cmd.ExecuteReader()
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
trv.BuildTree(dt, "产品|客户")
```

很多用户，包括我自己，都会犯这个错误，明明代码正确，却没有结果，折腾半天，只是因为没有设置数据源名称而已。

4.15.9 后台数据的更新方式

用 Update 语句更新
如果要更新没有加载的后台据，多数时候，我们用 Update 语句，例如：

```
Dim cmd As New SQLCommand
```

```
cmd.CommandText = "UPDATE {订单} SET 折扣 = 0.2 WHERE 数量 > 500"
cmd.ExecuteNonQuery()
```

表示将订购数量超过 500 的订单的折扣设置为 0.2。

用自带的方法更新

如果某个表已经加载结构，但是没有加载数据，或者只加载了部分数据，可以用 DataTable 自带的方法查询或更新后台数据。

DataTable 和后台数据处理相关的方法有：

SQLCompute	计算后台所有数据
SQLReplaceFor	批量更新后台数
SQLDeleteFor	批量删除后台数据
SQLSelect	从后台查询数据
SQLUpdate	保存通过 SQLSelect 查询得到的数据
SQLFind	从后台查找数据
SQLGetValues	从后台的指定列中提取不重复值，以集合的形式返回
SQLGetComboListString	从后台的指定的列中提取不重复的值，用符号"|"将这些值连接成一个字符串，并返回这个字符串

例如：

```
DataTables("订单").SQLReplaceFor("折扣", 0.02, "数量 > 500")
```

表示将订购数量超过 500 的订单的折扣设置为 0.2，不管这些订单是否已经加载，都会更新。

用临时表更新

我们也可以直接用临时表来更新后台数据，例如：

```
Dim dt As DataTable
Dim cmd As New SQLCommand
cmd.CommandText = "Select [_Identify],折扣 From {订单} Where 数量 > 500" '注意要包括主键列
dt = cmd.ExecuteReader(True) '注意可选参数设置为True
For Each dr As DataRow In dt.DataRows
    dr("折扣") = 0.2
Next
dt.Save()
```

同样可以将订购数量超过500的订单的折扣设置为0.2，这个方法的好处是临时表中的数据已经是更新后的了，无需重新加载。

使用临时表更新数据的要点：

1、Select语句必须将主键列包括进来，例如上面的例子，虽然我们只需更新折扣列，但是必须将[Identify]列包括在Select语句中，否则临时表的任何更新都无法保存。

2、Select语句尽量尽量只包括主键和要更新的列，不要包含用不到的列，以减少数据加载量，特别是开发基于网络的管理系统的时候，更要注意。

3、必须将ExecuteReader的可选参数设置为True。

上述代码都使用内部数据表作为示例，如果使用外部数据源，记得设置数据源名称

临时表与全局变量

在实际开发的过程中，我们可以需要经常在代码中调用某个表的数据，但是又不需要在界面中显示这个表。
对于这种情况，我们可以使用临时表加载这个表的数据，同时将临时表保存在一个Public变量中，这样就可以在任何位置调用这个表，无需反复加载。

假定我们建立了一个外部数据源"Region",其中有个名为“行政区域”的基础数据表，这个表已经输入了全国所有县级行政区域的资料，包括省市、市县、区号、邮编四列。
我们不希望主界面中显示这个表，但是希望能随时通过代码调用这个表的数据。
当然我们可以正常加载这个表，然后将这个表的Visible属性设置为False即可，不过这次我们用临时表来解决：

1、首先在全局代码中定义一个Public变量：

```
Public rdt As DataTable
```

2、在项目事件AfterOpenProject中设置代码：

```
Dim cmd As New SQLCommand
cmd.ConnectionName = "region"
cmd.CommandText = "Select * From {行政区域}"
rdt = cmd.ExecuteReader()
```

3、现在我们可以在任何地方调用这个表的数据，例如窗口中有两个ComboBox控件，第一个ComboBox列出所有的省市，第二个ComboBox根据第一个ComboBox所选择的省市，列出该省市所有的县市。

首先将窗口的AfterLoad事件代码设置为：

```
e.Form.Controls("ComboBox1").ComboList = rdt.GetComboListString("省市")
```

然后将ComboBox2的Enter事件代码设置为：

```
Dim sh As String = e.Form.Controls("ComboBox1").value
e.sender.ComboList = rdt.GetComboListString("县市","省市 = '" & sh & "'")
```

4、再例如希望在客户表中输入省市、县市两列后，区号和邮编能自动输入，可以将客户表的DataColChanged事件代码设置为：

```
If e.DataCol.Name = "省市" Orelse e.DataCol.Name = "县市" Then
    Dim dr As DataRow
    Dim Filter As String
    Filter = "[省市] = '" & e.DataRow("省市") & "' And [县市] = '" & e.DataRow("县市") & "'"
```

```
    dr = rdt.Find(Filter)
    If dr IsNot Nothing Then
        e.DataRow("区号") = dr("区号")
        e.DataRow("邮编") = dr("邮编")
    Else
        e.DataRow("区号") = Nothing
        e.DataRow("邮编") = Nothing
    End If
End If
```

你也可以用纯粹的SQL语言来实现类似的功能（参考用SQL语句实现自动输入），是用临时表的好处是数据只需加载一次， 坏处是需要加载所有数据，数据量特别大的时候可能不太合适。
如果你不想用SQL语句实现，数据量又比较大，可以考虑临时表只加载结构不加载数据，同时代码中的GetComboListString改为SQLGetComboListString，Find改为SQLFind，具体参考：一个综合示例

4.15.10 容易混淆表达式语法

Foxtable 中有很多地方要使用条件表达式，表达式的使用有两种情形。

针对已经加载的数据

DataTable 的 Select 、 Find 、 Compute 、 RepalceFor 、 DeleteFor 、 RemoveFor 、 GetValues 、 GetComboListString 方法。
Table 的 Filter 属性和 FindRow 方法
DataCol 的 Expression 属性。
GroupTableBuilder 和 CrossTableBuilder 的 Filter 属性 。
这些方法或属性针对的是已经加载的数据，适用于使用指南中《表达式》这一章介绍的语法，请参考：运算符和函数 和 条件表达式。

针对后台所有数据

DataTable 的 Load 方法是从后台重新加载数据，所以 LoadFilter 属性的语法和前面的不同，它适用于 SQL 语法，和 Select 语句的 Where 条件表达式语法相同，但是具体语法和函数随不同的数据源而所有差异。
同样 AppendLoad 用于从后台追载新的数据，所以其 Filter 参数适用于 SQL 语法。
DataTable 的一系列处理后台数据的方法：SQLCompute、SQLFind、SQLSelect、SQLReplaceFor、SQLDeleteFor、SQLGetValues、SQLGetComboListString，其条件参数显然都适用于 SQL 语法。
DataTable 的 Fill 和 SQLLoad 方法，以及 Table 的 Fill 方法，都是直接使用 Select 语句从后台加载数据，当然都适用 SQL 语法。
此外 SQLGroupTableBuilder、SQLCrossTableBuilder 和 SQLJoinTableBuilder 由于统计(或查询)的是后台数据，所以其 Filter 属性同样适用于 SQL 语法。
如果将 GroupTableBuilder 和 CrossTableBuilder 的 FromServer 属性设置为 True，此时二者统计的是后台数据，所以其 Filter 属性 将适用于 SQL 语法。
需要注意的是，采用 SQL 语法的时候，表达式中的表名和列名必须和后台数据库一致。

在开发指南的《SQL 相关》这一章，有 SQL 语法的介详细绍，请参考：Select 语法 Access 函数 SQL Server 函数

4.16 消息推送

4.16.1 关于消息推送

为方便大家理解消息推送的意义，我们先来一个现实生活的例子：
假定你早上 6 点必须起床，那么最好设置一个闹钟，在早上 6 点主动叫醒你。
早上 6 点闹钟响起，等于主动给你发送了一个消息：6 点到了，该起床了。
这就是最简单的消息主动推送功能。
如果没有闹铃，躺在床上的你，需要不停地看时间，确认是否已经 6 点了，这就是被动查询。
我们在实际开发管理软件的时候，会遇到同样的问题，例如：
送货单由用户 A 开单，由用户 B 审核。
当 A 开送货单保存时，同时由系统自动发给 B 发一个消息：“用户 A 已新增送货单 20151028001，请审核”。
B 收到此消息后，点此消息，可以打开这张送货单，进行审核，审核后，反馈回给 A 一个消息：“送货单 20151028001 已由用户 B 审”

在没有消息推送功能的情况下，A 只能将通知消息写入后台数据库，B 要反复查询数据库，看看是否有新的通知消息，这就是被动查询。
在多用户环境下，如果用户查询数据库的时间间隔过短，会增加服务器负担，如果间隔时间过长，用户则无法及时收到通知。
而且反复查询数据库，系统可能会出现卡顿现象，导致用户体验不佳，在互联网环境下尤甚。

如果有了消息推送功能，A 可以直接发一个消息给 B，就像微信或手机短信一样，B 能即时收到消息，并将处理结果用消息形式返回给 A。
这样的设计，既不会增加服务器负担，也不会有任何的迟延，更不会出现卡顿现象。
所以一个开发工具，是否有消息推送功能，会给系统性能和用户体验带来非常大的影响。

Foxtable 从 2016 版开始，开始内置消息推送功能，虽然有点迟，但终于有了。
从此，数据可以客户端和客户端之间，以及客户端和服务器之间实时传递，而这之前，数据只能被动地躺在数据库中等候查询。
从此，以前一些实现起来较为复杂的问题，例如多用户环境下的数据即时同步、复杂编号、流程设计、权限管理等等，实现起来将变得非常的轻松高效。
从此，Foxtable 不再是一个单纯的客户端开发工具，现在我们可以用 Foxtable 开发服务端程序了。
两个消息推送工具
Foxtable 用于消息推送的工具有两个，分别为 OpenQQ 和 UDPClient。
OpenQQ 整合度更高，功能更强，是 Foxtable 主要的消息推送工具。
UDPClient 相对简单一些，主要用于和第三方软件进行通讯。
接下来我们会分别介绍这两个类。

4.16.2 OpenQQ

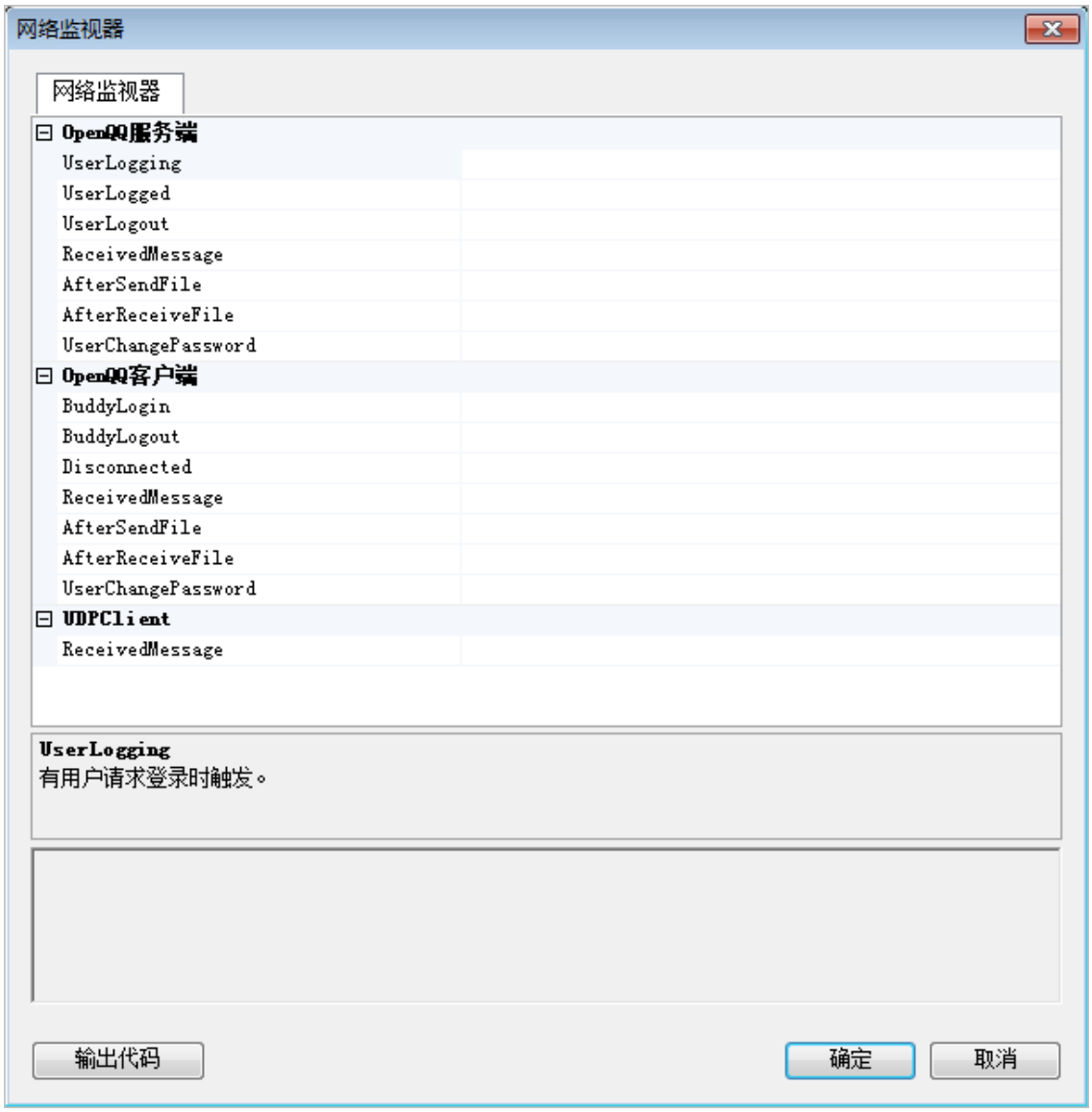
4.16.2.1 关于 OpenQQ

OpenQQ 最 Foxtable 最主要的消息推送工具，OpenQQ 不仅可以主动推送和接收文字信息，还可以实现图片和文件的自动收发。

OpenQQ 功能非常强大，却很容易掌握，任何人都可以在几分钟内建立起自己的消息推送系统，大家跟随帮助按顺序学习即可。

要使用 OpenQQ，必须有服务器，开发的时候，我们要设计两个项目，分别运行于服务器和客户端。

在菜单的“管理项目”功能区，单击“网络监视器”，可以设置 OpenQQ 的事件代码，事件分 OpenQQ 服务端和 OpenQQ 客户端两组:



可以看到两组事件中，有不少是同名的，编写代码的时候，注意不要混淆了。

重要的问题多说一遍：实际开发的时候，为便于调试分析，服务端项目和客户端项目可以在同一

台电脑运行，但必须是两个不同的项目，否则无法正常运行，切记。

4.16.2.2 快速入门

4.16.2.2.1 服务端的设计

4.16.2.2.1.1 准备用户表

本节的内容可以参考 CaseStudy 目录下的例子：QQServer.Table
首先我们应该设计一个表，这个表包括部门、姓名、密码、好友四列：

用户

	部门	姓名	密码	好友
1	生产部	李桂兰	888	技术部.董燕,行政部.罗迪
2	生产部	伍景珠	888	
3	生产部	邓玉婷	888	
4	生产部	李晓燕	888	技术部.陈倩,行政部.罗迪
5	生产部	林海	888	
6	生产部	朱新志	888	
7	生产部	杨刚	888	
8	技术部	董燕	888	
9	技术部	陈倩	888	
10	技术部	沈小燕	888	

对于比较大的企业，通常相同部门的工作人员自动互为好友，无需记录，非本部门的好友，记录在好友列中。
如果是小企业，那么所有员工自动互为好友即可。
我们在 QQServer.Table 这个例子中，还设计了一个名为"选择好友"窗口，用于选择其他 部门的同事作为好友并记录在好友列中.
选择好友列，单击右边的 按钮"..."，可以打开“选择好友”窗口，关于这个窗口的设计，我们就不介绍了，这不符合本章的主题。
用户名的组成
OpenQQ 的用户名包括两部分，格式为：分组名.用户名
在我们这个用户表中，用户名的格式就是：部门.姓名，例如：生产部.李桂兰
需要说明的是，这不是硬性的规定，你可以按照自己的需要随意命名用户。
但是如果你要使用 OpenQQ 内置的聊天窗口，那么最好按照这个规范来命名。

4.16.2.2.1.2 设置用户登录事件

本节的内容可以参考 CaseStudy 目录下的例子：QQServer.Table
有用户登录 OpenQQ 时，会触发 OpenQQ 服务端事件 UserLogging， 该事件用于验证登录用户身份，并在验证通过后，给用户设置好友。
此事件的 e 参数有：

UserName： 字符型，返回登录用户名
Password： 字符型，返回登录用户密码
Success： 逻辑型，默认为 False，设置为 True 允许此用户登录
Message: 字符型，通常用于设置返回给用户的欢迎信息或错误信息。
Buddies： 字符型集合，用于设置登录用户的好友名单
在菜单的“管理项目”功能区，单击“网络监视器”，设置 UserLogging 事件的代码如下：

```
Dim pts() As String = e.UserName.Split(".")
Dim dr As DataRow
dr = DataTables("用户").Find("部门 = '" & pts(0) & "' And 姓名 = '" & pts(1) & "'")
If dr Is Nothing Then '判断此用户不存在
    e.Message = "用户名错误!" '返回给登录者的提示信息
    Return
End If
If dr("密码") <> e.Password Then '如果密码错误
    e.Message = "登录密码错误!" '返回给登录者的提示信息
    Return
End If
e.Success = True '允许用户登录
'添加存储在好友列的非本部门的好友
If dr.IsNull("好友") = False Then
    Dim nms() As String = dr("好友").Split(",")
    e.Buddies.AddRange(nms)
End If
'最后添加相同部门的同事为好友
For Each dr In DataTables("用户").Select("部门 = '" & pts(0) & "'")
    Dim nm As String = dr("部门") & "." & dr("姓名")
    If nm <> e.UserName Then '如果不是登录者本人
        e.Buddies.Add(nm)
    End If
Next
```

以上代码判断用户名和密码是否正确，如果正确，将e参数Success设置为True，允许该用户登录，然后将该用户的好友添加到Buddies集合中。
需要注意的是，好友必须是双向的，也就是两个用户必须互为好友，相互之间才能收发信息。

如果企业员工不多，可以直接将所有员工相互加为好友，实际上示例文件QQServer.Table就是这样的：

```
Dim pts() As String = e.UserName.Split(".")
im dr1 As DataRow
dr1 = DataTables("用户").Find("部门 = '" & pts(0) & "' And 姓名 = '" & pts(1) & "'")
If dr1 Is Nothing Then '判断此用户不存在
    e.Message = "用户名错误!" '返回给登录者的提示信息
    Return
```

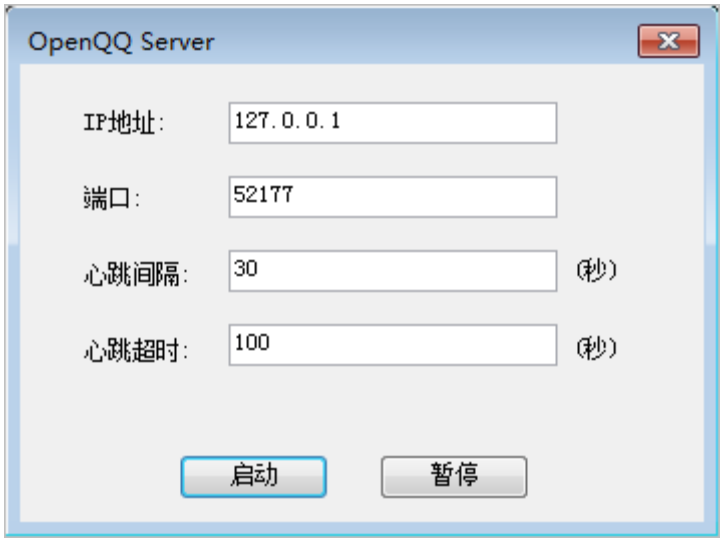
```
End If
If dr1("密码") <> e.Password Then '如果密码错误
    e.Message  = "登录密码错误!" '返回给登录者的提示信息
    Return
End If
e.Success = True '允许用户登录
'添加所有同事为好友
For Each dr2 As DataRow In DataTables("用户").DataRows
    Dim nm As String =  dr2("部门") & "." & dr2("姓名")
    If nm <> e.UserName Then '如果不是登录者本人
       e.Buddies.Add(nm)
    End If
Next
```

4.16.2.2.1.3 启动服务端

本节的内容可以参考 CaseStudy 目录下的例子：QQServer.Table
启动服务端的代码很简单，你可以在菜单或窗口增加一个按钮，按钮的代码设置为：

```
QQServer.ServerIp = "192.168.100.6" '指定IP地址
QQServer.ServerPort = 52177 '指定登录端口
QQServer.HeartbeatInterval = 30 '指定心跳时间
QQServer.HeartbeatTimeout = 100 '指定心跳超时时间
QQServer.Start() '启动QQServer
```

当然，实际使用的时候，我们应该设计一个窗口，让最终用户自己输入这些参数：



4.16.2.2.1.4 关于 IP、端口和防火墙

在本机调试

对于开发者而言，开发时服务端和客户端最好都位于本机，这样测试起来方便，此时可以将 QQServer 的 IP 地址设置为“127.0.0.1”，客户端通过"127.0.0.1"即可访问本机的 QQServer。

显然不同电脑之间，是不可能用 IP"127.0.0.1"相互访问的。

需要注意的是，虽然客户端和服务端可以在同一台电脑，但是必须分开不同的项目，否则无法正常运行。

关于端口

计算机中会有很多程序访问网络，但是计算机通常只有一个 IP 地址，那么不同的程序如何使用同一个 IP 收发数据呢？

这就涉及到端口，同一计算机中不同的程序，使用不同的端口访问网络。

QQServer 建议采用 52177 作为默认登录端口，除非和其它软件冲突，否则没有必要修改。

如果启动 QQServer 的时候提示"通常每个套接字地址(协议/网络地址/端口)只允许使用一次。"，就表示此端口已经被其他程序使用，这个时候你就只能换一个端口了，当然你得排除是否之前启动了 QQServer，还没有关闭呢？

端口最大的编号是 65536，如果要自定义端口，建议采用 20000 以上编号的端口。

多 IP 问题

如果服务器有多个 IP，不同的用户群通过不同的 IP 接入服务器，如果将 QQserver 设置为其中一个 IP，会导致某一部分的用户无法接入。

此时可以将 QQServer 的 IP 设置为"0.0.0.0"，例如：

```
QQServer.ServerIp = "0.0.0.0" '这个 IP 表示允许客户端通过本机的任何 IP 接入。
QQServer.ServerPort = 52177 '指定端口
QQServer.HeartbeatInterval = 30 '指定心跳时间
QQServer.HeartbeatTimeout = 100 '指定心跳超时时间
QQServer.Start() '启动 QQServer
```

关于防火墙

如果你开启了防火墙，记得将 Foxtable 加入白名单，以免被防火墙屏蔽。

4.16.2.2.1.5 关于心跳信号

普通用户可以忽略本节内容。

我们前面已经看到，在启动 QQServer 时，需要设置心跳间隔和心跳超时。

当 QQClient 连接到 QQServer 时，会根据 QQServer 的设置，定期向 QQServer 和好友发送心跳信号。

心跳信号的作用有两方面：

维持通道

多数时候，客户端是通过路由器连接外网的，这样局域网中的多个计算机，可以共享一个外网 IP 和外网进行通讯。

外网是不能直接访问内网计算机的，只能由内网计算机发起访问。

当我们在内网计算机使用某一端口经过路由器连接外网时，会将内网地址(包括 IP 和端口)转换为相应的外网地址。

例如局域网中计算机 A 在内网的 IP 地址是：192.168.100.8，路由器的外网 IP 是 87.68.151.180，计算机 A 使用端口 8088 通过路由器连接外网时,会进行如下的转换：
192.168.100.8088(计算机 A) → 87.68.151.180:62148(路由器) → 外网
也就是说，外网的计算机收到来自计算机 A 的信息时，会认为这个信息的来源地址是 87.68.151.180:62148。
提醒：62148 是随机的，可能是路由器任何一个未被占用的端口
当外网计算机向计算机 A 发送信息时，会将信息发送发送到 87.68.151.180:62148，局域网的路由器收到此信息后，将此消息转发到内网地址 192.168.100.8088，所以计算机 A 也就收到了外网发来的信息：
外网 → 87.68.151.180:62148(路由器) → 192.168.100.8088(计算机 A)
这等于在计算机 A 和外网之间开辟了一条双向通道，路由器会维持很多这样的通道，每个通道对应一个端口，路由器会记忆每个端口(通道)对应的是局域网中的那台计算机。
由于端口是有限的，如果某一个通道长期没有使用，此通道就会被关闭，从而腾出对应的端口给其他计算机或程序使用。
所以局域网中的 QQClient 连接外网的 QQserver，需要定期发送信号，以便持续占用所分配的外网端口，这就是心跳信号。
实际上在启动 QQClient 的时候,并不需要指定端口,Foxtable 会自动给 QQClient 分配一个端口的。
但是 QQServer 是需要明确指定端口的，这是为什么呢？
这是因为 QQClient 是主动连接 QQServer 的，当然需要事先知道 QQServer 的 IP 地址和端口，这就像打电话，拨号的一方需要先知道对方的号码。
检测意外断线
如果我们正常退出某个 QQClient，QQClient 会在退出之前，向 QQserver 和好友发送“我下线了”的通知信号。
如果某个 QQClient 的网络意外中断,那么服务器和其它客户端是不会收到该 QQClient 已经下线的通知的，以至于始终认为该 QQClient 在线。
如何解决这个问题呢？ 只能通过心跳信号了。
当服务器超过一定时间（心跳超时）没有收到某个 QQClient 的心跳信号时,就会认为这个 QQClient 已经意外断线，就会给该 QQClient 的好友发送其“超时断线”的通知。
显然 QQServer 的心跳间隔和心跳超时设置得越小，就越能快速检查出这种意外断线，但是如果在线用户量大，那么间隔太小的话， 就会加重服务器的负担。
默认的心跳间隔是 30 秒，心跳超时是 100 秒，具体时间大家可以自己调整 ，心跳超时一般设置为心跳间隔的 3 倍左右，也就是连续 3 次没有收到心跳信号，即认为客户端已经离线。
需要注意的是，必须重启 QQServer，心跳间隔和心跳超时才能生效。

4.16.2.2.1.6 指定端口范围

QQServer 的 Port 属性，只是指定了一个登录端口，实际运行的时候，QQServer 还会随机使用空闲端口用于收发信息和文件。
通过 PortRange 属性，我们可以限定 QQServer 使用的随机端口范围，例如：

```
QQServer.ServerIp = "192.168.100.6" '指定 IP 地址
QQServer.ServerPort = 52177 '指定登录端口
QQServer.HeartbeatInterval = 30 '指定心跳时间
QQServer.HeartbeatTimeout = 100 '指定心跳超时时间
QQServer.PortRange = "52000/53000" '指定随机端口范围
```

```
QQServer.Start() '启动 QQServer
```

随机端口范围的宽度一半不应该低于 200，用户数越多，范围宽度应该 相应增加，否则因为端口数量不够，会导致信息和文件发送失败，甚至无法登录。

重要提示： 新开通的阿里云服务器，必须指定端口范围才能正常使用 OpenQQ；同时需要在云服务器后台的安全组设置中，开放这个范围的端口接入，注意通讯协议为 UDP。

4.16.2.2.2 客户端的设计

4.16.2.2.2.1 启动客户端

本节的内容可以参考CaseStudy目录下的例子：QQClient.Table

启动客户端的代码很简单：

```
QQClient.ServerIP = "127.0.0.1" '指定服务器IP地址
QQClient.ServerPort = 52177   '指定服务器 登录端口
QQClient.UserName = "生产部.李桂兰"
QQClient.Password = "888"
If QQClient.Start() = True '如果登录成功
        Dim msg As String =   "恭喜,OpenQQ登录成功!"
        If QQClient.ServerMessage > "" Then '如果服务器返回了欢迎信息
                msg = msg & QQClient.ServerMessage
        End If
        PopMessage(msg,"提示",PopiconEnum.Infomation,5)
Else '如果登录失败,显示服务器返回错误信息
     PopMessage("QQClient登录失败,原因:" & vbcrlf & QQClient.ServerMessage,"提示
",PopiconEnum.Error,5)
End If
```

提示：QQClient的Start方法在登录成功后返回True，否则返回False，通过ServerMessage我们可以获得登录失败的原因 ，也可以在登录成功后通过这个属性获取服务器的欢迎信息。

实际使用的时候，通常我们会设计一个窗口，让用户输入服务器地址、端口以及登录用户名和密码：



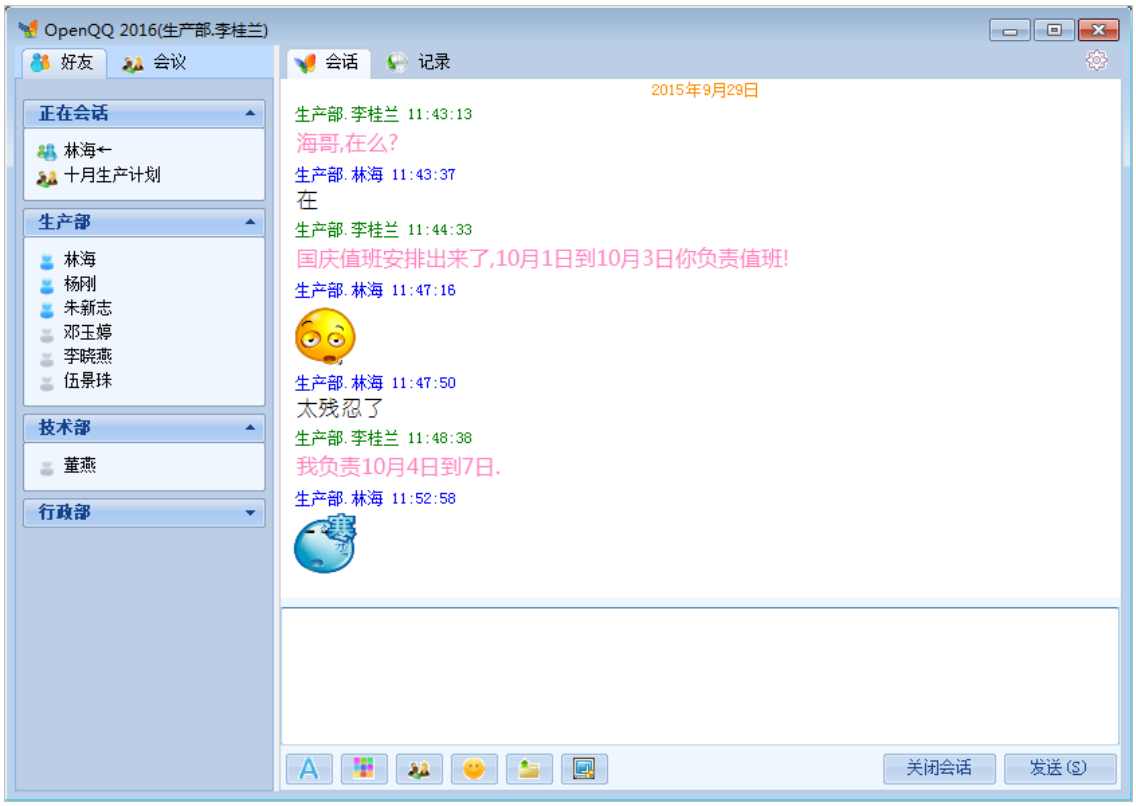
4.16.2.2.2.2 打开内置聊天窗口

OpenQQ 作为 Foxtable 主要的消息推送工具，其主要任务是进行自动化的信息传递和控制，而不是用来设计 QQ 这样聊天软件。

但是 OpenQQ 还是提供了一个内置的聊天窗口。
该窗口基本实现了腾讯 QQ 的功能，可以进行消息、图片和文件的收发，还提供了抓图功能，此外不管是消息、图片还是文件，都可以离线收发。
通过下面的代码可以打开内置聊天窗口：

```
If QQClient.Ready Then
    QQClient.OpenChatWindow()
End If
```

内置的聊天窗口很简洁，消息发送、截图、文件传输等常用功能都具备了：



这个窗口和腾讯QQ相比，有一个好处，就是可以通过代码自动发送消息、文件和图片，更适合整合在我们的管理系统中。

4.16.2.2.2.3 使用代码发送信息

QQServer 和 QQClient 有一个名为 Send 的方法，用于发送信息，语法为：

Send(BuddyName, Message, Anyway)

BuddyName: 用户(好友)名称

Message： 信息内容

AnyWay： 可选参数，逻辑型，如果对方不在线，是否发离线消息。

例如向生产部的林海发送消息：

```
QQClient.Send("生产部.林海","今天的任务完成了吗")
```

离线消息

在默认情况下，只有对方在线，Send 才会将消息发送出去，否则直接返回。

如果希望在对方离线的时候，依然发送离线消息，可以将第三个参数设置为 True，例如：

```
QQClient.Send("生产部.林海","今天的任务完成了吗",True)
```

实际上，此时 Send 将消息发送给了服务端，服务端收到此消息后，会将该消息保存下来，当好友上线后，服务端将此消息转发给好友。

需要注意的是，使用 Send 发送的消息，并不会出现在内置的聊天窗口中，我们必须使用代码来接收之。

4.16.2.2.2.4 使用代码接收信息

对方通过 Send 代码发来的消息，接收方只能通过代码接收。
QQserver 和 QQClient 都有个名为 ReceivedMessage 事件，当接收到对方通过 Send 发来的信息时，会触发此事件。
该事件的 e 参数有：
UserName：字符型，返回用户(好友)名称，如果是服务端(QQServer)发来的信息，则返回空。
Message: 字符型，返回信息内容
示例
例如需要在一个表中记录所有收到的信息，可以在 ReceivedMessage 事件设置代码：

```
Dim dr As DataRow = DataTables("信息").AddNew
dr("发送者") = iif(e.UserName > "",e.UserName,"服务器")
dr("时间") = Date.Now()
dr("信息") = e.Message
```

提醒：

在编写代码的时候，你会发现有两个ReceivedMessage事件，一个在OpenQQ服务端分组，一个在OpenQQ客户端分组，前者对应的是QQServer，后者对应的是QQClient， 注意不要搞混。

4.16.2.2.2.5 客户端和服务端之间的通讯

服务端(QQServer)和客户端(QQClient)之间也可以通过代码收发信息的。
客户端向服务端发发送信息
QQClient 用 Send 发送信息时，如果不指定接收者，就会直接发给服务器：
QQClient.Send("","明天天气不错")
或直接：
QQClient.Send("明天天气不错")
服务端向客户端发送信息
QQServer 也有 Send 方法，用于向客户端发送信息，例如：
QQServer.Send("生产部.林海","测试信息")

如果希望对方在离线的时候，依然发送离线消息，可以将第三个参数设置为 True，例如：

```
QQServer.Send("生产部.林海","测试信息",True)
```

这样 QQServer 会先将消息暂存起来，等到对方上线，再将消息发出。
服务端接收消息
QQServer 同样有 ReceivedMessage 事件，服务端收到来自客户端的消息时，会触发此事件。
该事件的 e 参数有：
UserName：字符型，返回客户端名称。
Message: 字符型，返回信息内容。

例如需要在一个表中记录所有收到的信息，可以在 ReceivedMessage 事件设置代码：

```
Dim dr As DataRow = DataTables("信息").AddNew
dr("发送者") = e.UserName
dr("时间") = Date.Now()
dr("信息") = e.Message
```

提醒：

在编写代码的时候，你会发现有两个ReceivedMessage事件，一个在OpenQQ服务端分组，一个在OpenQQ客户端分组，前者对应的是QQServer，后者对应的是QQClient， 注意不要搞混。

4.16.2.2.2.6 使用代码收发文件

QQServer 和 QQClient 都有 SendFile 方法，用于发送文件。
语法：

```
SendFile(UserName, FileName)
```

UserName：用户(好友)名。
FileName： 文件名，含路径。
例如客户端向好友发送文件：

```
QQClient.SendFile("生产部.林海","c:\data\150928.doc")
```

如果客户端要向服务器发送文件，不指定接收者即可，例如：

```
QQClient.SendFile("c:\data\150928.doc")
```

服务器也可以向客户端发送文件，例如：

```
QQServer.SendFile("生产部.林海","c:\data\150928.doc")
```

QQClient 通过 SendFile 向好友发送文件时，如果对方 不在线，并不会发送离线文件，而是会直接返回，所以通常需要判断一下：

```
Dim bd As QQBuddy = QQClient.Buddies("技术部.林欣华")
If bd.OnLine  '如果对方在线
   QQClient.SendFile("技术部.林欣华","c:\data\151022.doc")
Else
      MessageBox.Show(" 对 方 离 线 ， 无 法 发 送 文 件 !"," 提 示 ",
MessageBoxButtons.OK ,MessageBoxIcon.Information)
End If
```

SendFile 用异步方式，在后台静默发送文件，且发送方和接收方都不会显示进度，也不会在内置聊天窗口留下痕迹。
发送过程可以继续其他操作，文件发送完毕后，系统会通过事件来通知双方：
1、发送方触发事件 AfterSendFile。
2、接收方触发事件 AfterReceiveFile。

QQServer 和 QQClient 都有这两个事件，注意设置代码的时候不要混淆了。

两个事件的 e 参数是相同的：
UserName：字符型，返回对方名称，如果对方是服务器，则返回空。
FileName：字符型，返回发送或接收的文件名，含路径。
Success： 逻辑型，如果发送或接收成功，返回 True，否则返回 False。

4.16.2.2.2.7 内置聊天窗口相关

前面已经提到，用 Send 发送的消息，以及用 SendFile 发送的文件，都不会显示在内置聊天窗口 ，而是触发相关事件。
QQClient 另外提供了一组方法，用于向好友或会议发送消息、文件和图片。
这组方法同样是在后台静默发送，同样不会显示进度条，但是发送完成后，内容会显示在双方的内置聊天窗口中，而不是触发相关事件。
这组方法有：

- **SendChatMessage**

  用于向好友发送信息，语法：

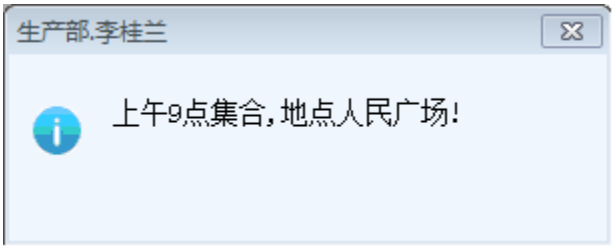
  SendChatMessage(BuddyName, Message)

  BuddyName: 好友名称或会议名称。
  Message： 消息内容

  例如：
  QQClient.SendChatMessage("生产部.林海","上午 9 点集合,地点人民广场!")
  如果希望对方以弹窗显示收到的信息，可以在消息前面加上"{!}"，例如：

  QQClient.SendChatMessage("生产部.林海","{!}上午 9 点集合,地点人民广场!")

  接收方在收到消息之后，屏幕右下角会自动弹出一个窗口，用于显示此消息：

  提示：弹窗消息同样会记录并显示在内置聊天窗口，只是在此之外多了一个弹窗提示。

- **SendChatFile**

  用于向好友发送文件，语法：

  SendChatFile(BuddyName, FileName)

BuddyName: 好友名称或会议名称。
FileName： 文件名，含路径。

- **SendChatImage**

  用于向好友发送图片文件，语法：

  SendChatImage(BuddyName, FileName)

  BuddyName: 好友名称或会议名称。
  FileName： 图片文件名，含路径。

  SendChatImage 和 SendChatFile 有些类似，不同是 SendChatImage 发送完成后，会在聊天窗口直接显示图片，而不是显示一个连接。

重要提示
如果好友没有上线，SendChatFile 和 SendChatImage 会自动以离线形式将图片或文件发送给对方。这一点和前面介绍的 SendFile 方法是不同的。

4.16.2.2.2.8 让用户自主设置密码

本节的内容可以参考CaseStudy目录下的例子：

QQClient.Table
QQServer.Table

在 OpenQQ 内置的聊天窗口中，单击右上角的设置按钮，出现的设置菜单，有“修改密码”的命令。默认情况下，单击"修改密码"命令，并不会有任何动作。
实际上，我们单击这个命令，会触发一个客户端事件 UserChangePassword，我们需要在这个事件中编写代码，向服务端提交更改密码的申请，服务端收到这个申请后，会触发服务端的 UserChangePassword 事件，我们 同样需要在这里编写代码，保存用户提交的新密码。
客户端的设计
1、打开文件 QQClient.Table，首先在客户端设计一个下图所示的窗口，用于修改密码：



2、该窗口的确定按钮的代码设置为：

```
Dim OldPassword As String = e.Form.Controls("txtOld").Text
Dim NewPassword1 As String = e.Form.Controls("txtNew1").Text
Dim NewPassword2 As String = e.Form.Controls("txtNew2").Text
If OldPassword  = "" Then
    Popmessage("请输入原密码!", "提示", 0, 5)
    Return
End If
If NewPassword1 <> NewPassword2 Then
    Popmessage("新密码输入不一致!", "提示", 0, 5)
    Return
End If
Dim msg As String = QQClient.ChangePassword(OldPassword, NewPassword1) '向服务端提交更改密码的申请
If msg = "OK" Then
    Popmessage("恭喜,密码修改成功!", "提示", 0, 5)
    e.Form.close()
ElseIf msg > "" Then
    Popmessage("密码修改失败,原因: " & vbcrlf & msg, "提示", 0, 5)
Else
    Popmessage("未获得服务器回应,无法判断密码是否修改成功!", "提示", 0, 5)
End If
```

提示：QQClient的ChangePassword方法用于向服务端提交更改密码的申请，第一个参数为原密码，第二个参数为新密码。

3、将 OpenQQ 客户端的 UserChangePassword 事件代码设置为：

```
Forms("修改密码").Show()
```

服务端的设计

打开文件 QQServer.Table，将 OpenQQ 服务端的 UserChangePassword 事件代码设置为：

```
Dim pts() As String = e.UserName.Split(".")
Dim dr As DataRow = DataTables("用户").Find("部门 = '" & pts(0) & "' And 姓名 = '" & pts(1) & "'")
If dr Is Nothing OrElse dr("密码") <> e.OldPassword Then
   e.ReturnMessage = "原密码不符!"
Else
    dr("密码") = e.NewPassword
    dr.Save()
    e.ReturnMessage = "OK" '返回约定的字符，通知客户端，密码修改成功
End If
```

建议大家实际开发的时候，密码的传递、保存和比较，都不要使用明码，而是使用加密后的字符串。

4.16.2.2.2.9 在菜单提示新消息

本节的内容可以参考 CaseStudy 目录下的文件：QQClient.Table。
本节的任务是在菜单的配置栏加一个 OpenQQ 按钮：



这个按钮可以实现以下功能：

1、当 OpenQQ 离线时，按钮图标为，当 OpenQQ 在线时，按钮图标为，这样我们就可以通过这个按钮判断 OpenQQ 的状态。
2、当 OpenQQ 离线时，单击这个按钮，可以打开自定义的登录窗口。
3、当 OpenQQ 在线时，单击这个按钮，可以打开内置聊天窗口。
4、当收到新消息时，能自动显示未读新消息的条数，并自动闪烁，以提示用户有新消息。
提示：这里的未读新消息统计，仅针对通过内置聊天窗口收发的消息。
设计步骤：
1、将上述两个图标复制到项目的 Images 目录下，的文件名为 offline.ico,的文件名为 online.ico
2、在配置栏增加这个按钮，按钮图标设置为 offline.ico，按钮的代码设置为：

```
If QQClient.Ready Then
    QQClient.OpenChatWindow()
Else
    Forms("QQClient").Open()
End If
```

3、假定这个新增按钮名称为"OpenQQ"，在菜单的SystemIdle事件增加以下代码：

```
'此段代码用于显示OpenQQ的未读消息
Static OpenQQ As RibbonMenu.Button
Static LastTime As Date
```

```
If OpenQQ Is Nothing Then
    OpenQQ = ConfigBar.Items("OpenQQ")
End If
Dim txt As String
If QQClient.Ready
    If QQClient.UnreadCount > 0 Then
        txt = "(" & QQClient.UnreadCount & "条)"
    End If
End If
If OpenQQ.Text <> txt Then
    OpenQQ.Text = txt
End If
If OpenQQ.Text > "" Then
    Dim ntp As timeSpan = Date.Now - LastTime
    If ntp.TotalSeconds >= 0.5 Then
        LastTime = Date.Now
        If OpenQQ.ForeColor = Color.Red Then
            OpenQQ.ForeColor = Color.Transparent
        Else
            OpenQQ.ForeColor = Color.Red
        End If
    End If
End If
```

上面代码利用新增加的按钮标题显示未读新消息数量，并每隔0.5秒切换按钮的字体颜色，形成闪烁效果，以提示用户有新消息。

通过这段代码，你还可以学到一招： 定期执行某项任务，并非一定要通过计划或定时器，望认真体会。

4、将自定义登录窗口的登录按钮的代码设置为：

```
If QQClient.Ready Then
    MessageBox.show("QQClient已经启动,请先关闭","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
QQClient.ServerIP = e.Form.Controls("txtIP").Value   '指定服务器IP地址
QQClient.ServerPort = e.Form.Controls("txtPort").Value   '指定服务器端口
QQClient.UserName = e.Form.Controls("cmbDepartment").Value & "." & e.Form.Controls("txtUserName").Value '指定登录用户名
QQClient.Password = e.Form.Controls("txtPassword").Value
If QQClient.Start() = True '如果登录成功
    e.Form.Controls("btnChat").Enabled = True
    Dim msg As String =   "恭喜,OpenQQ登录成功!"
    If QQClient.ServerMessage > "" Then '如果服务器返回了欢迎信息
```

```
        msg = msg & QQClient.ServerMessage
    End If
    popMessage(msg,"提示",PopiconEnum.Infomation,5)
Else '如果登录失败,显示服务器返回错误信息
    e.Form.Controls("btnChat").Enabled = False
    PopMessage("QQClient登录失败,原因:" & vbcrlf & QQClient.ServerMessage,"提示",PopiconEnum.Error,5)
End If
```

5、将OpenQQ客户端事件Connected事件代码设置为：

```
ConfigBar.Items("OpenQQ").SmallImage = GetImage("online.ico")
```

这行代码用于在OpenQQ登录成功后，将菜单按钮的图标改为online.ico。

6、最后将OpenQQ客户端事件Disconnected事件代码设置为：

```
ConfigBar.Items("OpenQQ").SmallImage = GetImage("offline.ico")
```

这样在OpenQQ离线后，菜单按钮的图标会自动改为offline.ico。

4.16.2.2.3 内置身份验证

4.16.2.2.3.1 关于内置身份验证

Foxtable 本身有用户管理功能，我们可以直接使用 Foxtable 的内置身份验证来完成 OpenQQ 的登录验证。

必要条件

显然，只有客户端项目和服务端项目都使用同样的用户数据，OpenQQ 才能使用内置身份验证。

在默认情况下，Foxtable 的用户名、用户密码等数据是保存在项目文件中，客户端和服务端是独立的，无法满足以上要求。

所以必须使用外部数据源存储用户数据，OpenQQ 才能使用内置身份验证，请参考：外部数据源与用户管理

4.16.2.2.3.2 服务端的设置

启动服务端的时候，只需将 QQServer 的 Buildin 属性设置为 True，就可以启用内置身份验证：

```
QQServer.ServerIp = "192.168.100.6" '指定 IP 地址
QQServer.ServerPort = 52177 '指定登录端口
QQServer.HeartbeatInterval = 30 '指定心跳时间
QQServer.HeartbeatTimeout = 100 '指定心跳超时时间
QQserver.Buildin = True '启用内置身份验证
```

```
QQServer.Start() '启动 QQServer
```

4.16.2.2.3.3 客户端的设置

如果客户端在登录 OpenQQ 时，不指定用户名和密码，系统就会自动调用内置用户的用户名和密码来登录 OpenQQ。
例如：

```
QQClient.ServerIP = "127.0.0.1"
QQClient.ServerPort = 52177
QQClient.UserName = ""
QQClient.Password = ""
If QQClient.Start() = True
    Dim msg As String =  "恭喜,OpenQQ 登录成功!"
    If QQClient.ServerMessage > "" Then
       msg = msg & QQClient.ServerMessage
    End If
    PopMessage(msg,"提示",PopiconEnum.Infomation,5)
Else
   PopMessage("QQClient 登录失败,原因:" & vbcrlf & QQClient.ServerMessage,"提示",PopiconEnum.Error,5)
End If
```

提示：

1、要使上述代码有效，必须事先将 QQServer 的 Buildin 属性设置为 True。
2、上述代码对于开发者和管理员无效，参考：开发者和管理员的特殊性

4.16.2.2.3.4 用户名的组成

使用内置身份验证时，OpenQQ的用户名包括分组和用户名，例如某个用户的分组是"主管"，用户名是"张三"，那么对应的OpenQQ的用户名就是"主管.张三"，向他发送信息的代码就是：

```
QQClient.Send("主管.张三","今天的任务完成了吗")
```

当然，如果没有给这个用户设置分组，他的OpenQQ用户名就是"张三":

```
QQClient.Send("张三","今天的任务完成了吗")
```

4.16.2.2.3.5 验证用户身份

启用内置身验证后，服务端的登录身份验证变得非常简单。

OpenQQ 服务端用于登录身份验证的事件是 UserLogging，该事件有个 e 参数 User，类型为 UserInfo，

如果内置身份验证通过，该参数返回登录用户，否则返回 Nothing。

所以判断是否通过身份验证的代码很简单：

```
If e.User IsNot Nothing Then
  '通过了内置身份验证
  e.Success = True '允许用户登录
Else
  '未通过内置身份验证
End If
```

例如通过验证后添加所有用户为好友：

```
If e.User IsNot Nothing Then
  e.Success = True '允许登录
  For Each u As UserInfo In Users
    If u.Name <> e.User.Name  Then
      If u.Group > ""  Then
        e.Buddies.Add(u.Group & "." & u.Name)
      Else
        e.Buddies.Add(u.Name)
      End If
    End If
  Next
End If
```

有条件添加好友

在添加好友的时候，我们可以进行一些判断，例如经理分组的用户，只添加经理和主管两个分组的用户为好友；主管分组的用户，添加所有用户为好友；员工分组的用户添加主管和员工两个分组的用户为好友：

```
If e.User IsNot Nothing Then '如果通过了内置身份验证, e.User表示登录的用户
  e.Success = True '允许登录
   Select Case e.User.Group
    Case "经理"
      For Each u As UserInfo In Users
        If u.Name <> e.User.Name  Then
          If u.Group = "经理" OrElse u.Group = "主管" Then
            e.Buddies.Add(u.Group & "." & u.Name)
          End If
        End If
      Next
    Case "主管"
      For Each u As UserInfo In Users
        If u.Name <> e.User.Name  Then
```

```
            If u.Group > ""  Then
              e.Buddies.Add(u.Group & "." & u.Name)
            Else
              e.Buddies.Add(u.Name)
            End If
          End If
        Next
      Case "员工"
        For Each u As UserInfo In Users
          If u.Name <> e.User.Name  Then
            If u.Group = "主管" OrElse u.Group = "员工" Then
              e.Buddies.Add(u.Group & "." & u.Name)
            End If
          End If
        Next
    End Select
End If
```

当然你也可以根据用户角色来选择好友，这里就不举例了。

虽然可以根据用户分组或角色选择好友，但显然局限性比较大，不够灵活 ，这是下一节要解决的问题。

4.16.2.2.3.6 自由设置好友

本节的内容可以参考 CaseStudy 目录下的文件：内置验证 QQServer.Table，内置验证 QQClient.Table

上一节讲到，使用内置身份验证时，虽然可以根据用户分组或角色选择好友， 但局限性比较大，不够灵活。

你可以自行扩展服务器项目，让服务端管理人员可以给登录者自由选择好友。

设计步骤

1、首先在服务端项目增加一个名为“好友”的表，这个表有两列，分别是“用户名”和“好友”，前者是字符型，后者是备注型：

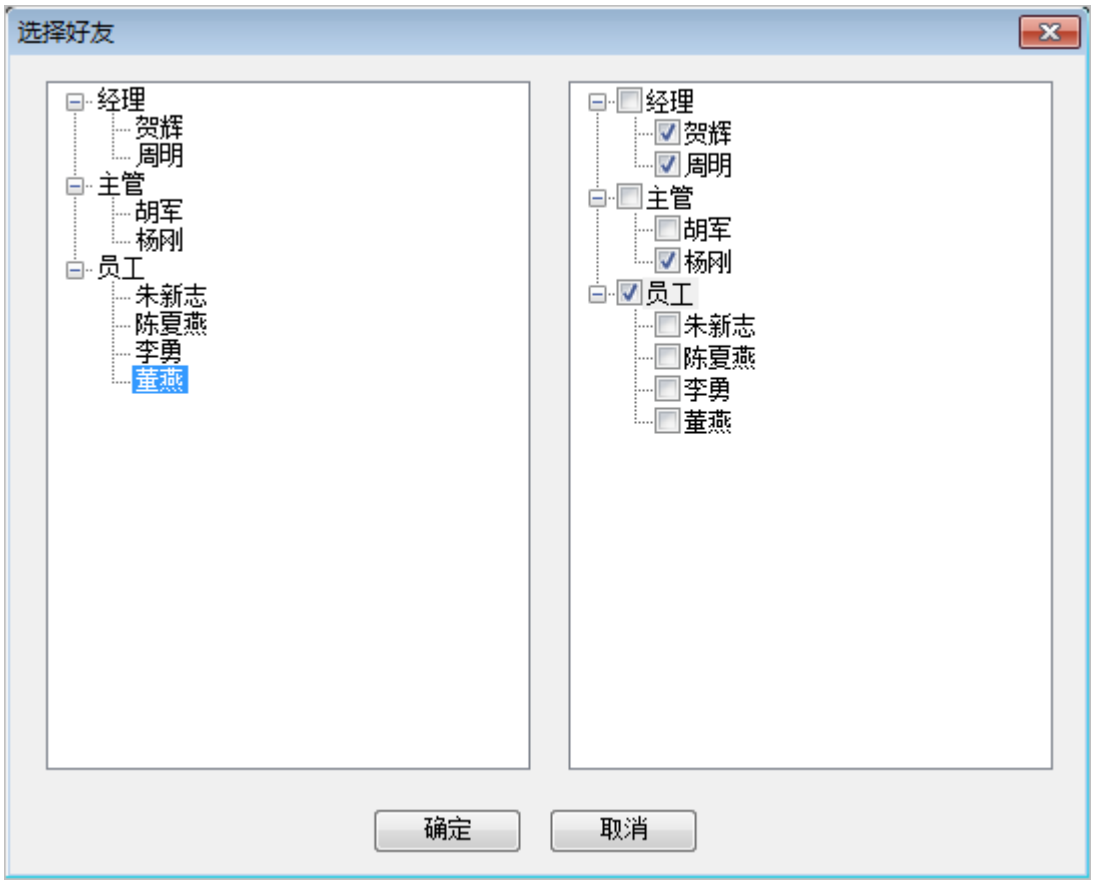
好友

	员工	好友
1	贺辉	[主管], 周明, 董燕
2	杨刚	贺辉, 周明, 胡军, 陈夏燕, 李勇, 董燕
3	董燕	[员工], 贺辉, 周明, 杨刚
4	林海	[员工]
5	朱新志	[员工]
6	陈夏燕	[员工]
7	李勇	[员工]
8	周明	[经理], [主管], 董燕

提示：

上表的好友列，可以包括单个的用户名，也可以包括分组名，其中分组名用方括号括起来。例如上表中董燕的好友包括贺辉、周明、杨刚，以及所有分组为员工的用户。

2、然后设计一个下图所示的窗口：



窗口的AfterLoad事件代码为：

```
Dim trv1 As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim trv2 As WinForm.TreeView = e.Form.Controls("TreeView2")
Dim trvs() As WinForm.TreeView = {trv1,trv2}
```

```
For Each trv As WinForm.TreeView In trvs
    For Each u As UserInfo In Users
        If u.Type = UserTypeEnum.User Then
            If trv.Nodes.Contains(u.Group) = False Then
                trv.Nodes.Add(u.Group)
            End If
            trv.Nodes(u.Group).Nodes.Add(u.Name)
        End If
    Next
Next
trv2.CheckBoxes = True '右边目录树显示复选框
```

左边目录树的AfterSelectNode事件代码设置为：

```
Dim trv1 As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim trv2 As WinForm.TreeView = e.Form.Controls("TreeView2")
Dim dr As DataRow
If e.node.Level = 1 Then
    dr = DataTables("好友").Find("员工 = '" & e.Node.Name & "'")
End If
Dim nms As New List(of String)
If dr IsNot Nothing AndAlso dr.IsNull("好友") = False
    nms.AddRange(dr("好友").Split(","))
End If
For Each nd As WinForm.TreeNode In trv2.AllNodes
    If nd.Level = 0 Then
        nd.Checked = nms.Contains("[" &nd.Name & "]")
    Else
        nd.Checked = nms.Contains(nd.Name)
    End If
Next
trv2.Enabled = (e.node.Level = 1)
```

右边目录树的AfterCheckNode事件代码设置为：

```
Dim trv1 As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim trv2 As WinForm.TreeView = e.Form.Controls("TreeView2")
Dim nd1 As WinForm.TreeNode = trv1.SelectedNode
If nd1.Level = 0 Then
    Return
End If
Dim dr As DataRow
If e.Node.ParentNode IsNot Nothing Then '取消父节点的选中标记,因为选中子节点,就不要选中父节点了
    e.Node.ParentNode.Checked = False
```

```
End If
For Each cnd As WinForm.TreeNode In e.Node.Nodes '取消子节点的选中标记,因为选中父节点,就不要选中子节点了
    cnd.Checked = False
Next
dr = DataTables("好友").Find("员工 = '" & nd1.Name & "'")
If dr Is Nothing  Then
    dr = DataTables("好友").AddNew()
    dr("员工") = nd1.Name
End If
Dim nms As String
For Each nd2 As WinForm.TreeNode In trv2.Nodes '先添加分组
    If nd2.Checked Then
        nms  = nms & ",[" &  nd2.Name & "]"  '分组名用方括号括起来,表示这是一个分组.
    End If
Next
For Each nd2 As WinForm.TreeNode In trv2.AllNodes '然后是普通用户
    If nd2.Level = 1 AndAlso nd2.Checked AndAlso nd1.Name <> nd2.Name Then
        nms = nms & "," &  nd2.Name
    End If
Next
If nms > "" Then
    dr("好友") = nms.Trim(",")
Else
    dr("好友") = Nothing
End If
```

4、最后将QQserver的UserLogging事件代码设置为：

```
If e.User Is Nothing Then
    Return
Else
    e.Success = True
End If
Dim dr As DataRow
dr = DataTables("好友").Find("员工 = '" & e.User.Name & "'")
If dr IsNot Nothing Then
    Dim nms As New List(of String)
    nms.AddRange(dr("好友").Split(","))
    For Each u As UserInfo In Users
        If u.Name <> e.User.Name Then
            If nms.Contains(u.Name) OrElse nms.Contains("[" & u.Group & "]")
                If u.Group > ""  Then
                    e.Buddies.Add(u.Group & "." & u.Name)
                Else
```

```
                e.Buddies.Add(u.Name)
            End If
        End If
    End If
  Next
Else
  For Each u As UserInfo In Users
    If u.Name <> e.User.Name  Then
      If u.Group > ""  Then
        e.Buddies.Add(u.Group & "." & u.Name)
      Else
        e.Buddies.Add(u.Name)
      End If
    End If
  Next
End If
```

4.16.2.2.3.7 更新用户属性

使用内置身份验证时，为提高身份验证和添加好友的效率，Open 服务端会一次将所有用户数据加载进来，但这会带来一些问题。
例如管理人员在客户端修改了某用户的分组和角色，但是这些修改并不会在服务端即时体现出来，直到重启 QQServer。
所以建议尽量在服务端进行用户管理。
如果用户管理的工作一定要在客户端进行，可以在完成之后给服务端发送一个自定义信息，通知服务器重新加载用户数据。
客户端管理用户的代码为：

```
Syscmd.Project.Users()
If User.Type <> UserTypeEnum.User Then '只有开发者和管理员才有必要通知服务端刷新
  QQClient.Send("约定格式的信号")
End If
```

服务端的 ReceivedMessage 事件代码为：

```
If e.Message = "约定格式的信息" Then
  Users.Load()
End If
```

提示： Load是Users新增的一个方法，用于重新加载用户数据。

4.16.2.2.3.8 开发者和管理员的特殊性

采用外部数据表存储用户信息，其实存储的只是普通用户的信息，开发者和管理员的信息始终是存储在本地项目的。

因此 OpenQQ 使用内置身份验证的时候，默认不包括开发者和管理员，只有普通用户才能使用内置身份验证登录 OpenQQ。
通常这对实际的使用并没有影响，因为实际使用的时候，客户端都是普通用户。
如果开发过程中，为便于调试分析，希望开发者和管理员也能使用内置身份登录 OpenQQ，可以将客户端的登录代码要调整为：

```
If QQClient.Ready Then
        MessageBox.show("QQClient 已 经 启 动 ， 请 先 关 闭 "," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
QQClient.ServerIP = e.Form.Controls("txtIP").Value  '指定服务器IP地址
QQClient.ServerPort = e.Form.Controls("txtPort").Value  '指定服务器端口
If user.Type = UserTypeEnum.User Then '如果是普通用户
    QQClient.UserName =""
    QQClient.Password = ""
Else '如果是开发者或管理员
    Dim pwd As String
    If InputPassWord(pwd,"提示","请输入" & User.Name & "的密码:") Then
      QQClient.UserName = User.Name
      QQClient.Password = pwd
    Else
      Return
    End If
End If
If QQClient.Start() = True '如果登录成功
    e.Form.Controls("btnChat").Enabled = True
    Dim msg As String =  "恭喜,OpenQQ登录成功!"
    If QQClient.ServerMessage > "" Then '如果服务器返回了欢迎信息
      msg = msg & QQClient.ServerMessage
    End If
    ConfigBar.Items("OpenQQ").SmallImage = GetImage("online.ico")
    popMessage(msg,"提示",PopiconEnum.Infomation,5)
Else '如果登录失败,显示服务器返回错误信息
    e.Form.Controls("btnChat").Enabled = False
     PopMessage("QQClient 登 录 失 败 , 原 因 :" & vbcrlf & QQClient.ServerMessage," 提 示 ",PopiconEnum.Error,5)
End If
```

意思是开发者和管理员登录 OpenQQ 时，需要再输入一次密码。
至于服务端代码，则不需要调整。

4.16.2.2.4 内置聊天窗口

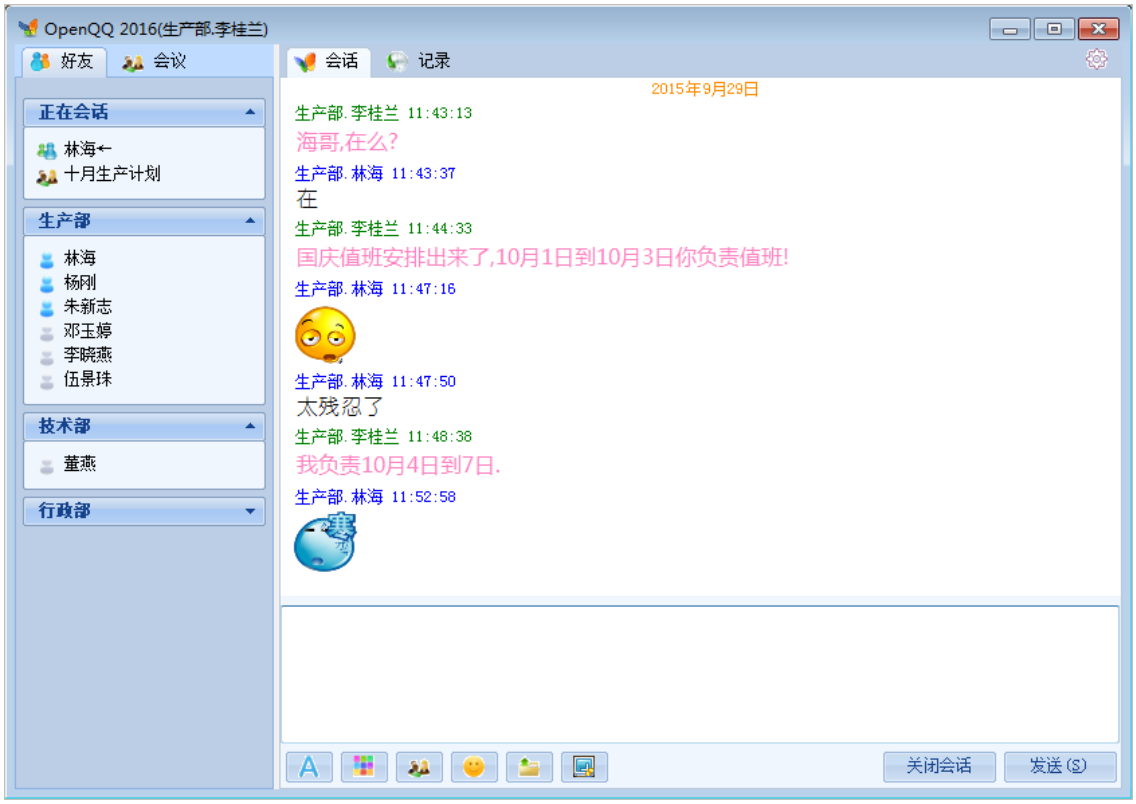
4.16.2.2.4.1 打开聊天窗口

OpenQQ 作为 Foxtable 主要的消息推送工具，其主要任务是进行自动化的信息传递和控制，而不是用来设计 QQ 这样聊天软件。

但是 OpenQQ 还是提供了一个内置的聊天窗口。
该窗口基本实现了腾讯 QQ 的功能，可以进行消息、图片和文件的收发，还提供了抓图功能，此外不管是消息、图片还是文件，都可以离线收发。
通过下面的代码可以打开内置聊天窗口：

```
If QQClient.Ready Then
    QQClient.OpenChatWindow()
End If
```

内置的聊天窗口很简洁，消息发送、截图、文件传输等常用功能都具备了：



这个窗口和腾讯QQ相比，有一个好处，就是可以通过代码自动发送消息、文件和图片，更适合整合在我们的管理系统中。

4.16.2.2.4.2 发送文字消息

本节内容介绍内置聊天窗口和文字消息相关的部分。

正常发送

输入文字内容，单击发送按钮，即可正常发送文字消息。

离线消息

如果对方不在线，消息会自动发给服务器，等对方上线后，服务器会自动将消息转发给对方。

消息回执

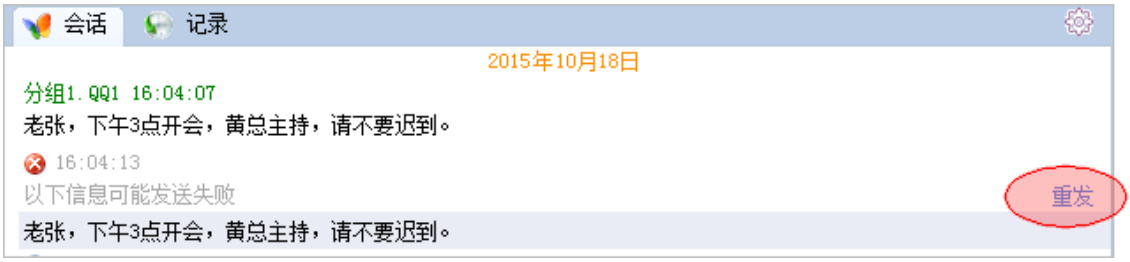
接收方收到消息后，会返回一个回执给发送方，表示我已经收到了这个消息。

如果发送方在一定时间内没有收到回执，会认为这个消息发送失败，会在聊天窗口提示用户：



重发消息

单击发送失败的消息，消息右上角会有“重发”命令，单击该命令，即可重发之前发送失败的消息：



中转发送

如果聊天双方网络连接状况不佳(跨运营上的宽带之间经常如此)，消息重复提示发送失败，我们可以改用中转发送。

所谓中转发送，就是消息不直接发给接收方，而是先发给服务器，服务器收到后即时转发给接收方。

中转发送有两个途径：

1、输入要发送的文字内容，按住 Ctrl 键的同时单击发送按钮。

2、选择发送失败的消息，按住 Ctrl 键的同时单击重发命令。

4.16.2.2.4.3 发送屏幕截图

本节介绍内置聊天窗口屏幕截图与发送相关的功能。

正常发送

在聊天窗口单击截图按钮，截取屏幕某一区域，确定后即可自动将截取的图片发送给对方。

回执信号

接收方收到图片后，会返回一个回执给发送方，表示我已经收到了这个图片。
如果发送方在一定时间内没有收到回执，会认为这个消图片发送失败，会在聊天窗口提示用户：



重发图片

选择发送失败的图片，图片右上角会出现重发命令，单击此命令，即可重发之前发送失败的图片。

中转发送

如果聊天双方网络连接状况不佳(跨运营上的宽带之间经常如此)，图片发送总是失败，我们可以改用中转发送。
所谓中转发送，就是图片不直接发给接收方，而是先发给服务器，服务器收到后即时转发给接收方。
中转发送有两个途径：
1、按住 Ctrl 键，单击截图按钮，然后松开 Ctrl 键，截图完成之后，会自动中转发送截取的图片。
2、选择发送失败的图片，按住 Ctrl 键的同时单击重发命令。

4.16.2.2.4.4 发送表情

在内置聊天窗口，单击“表情”按钮，会列出 81 个表情图片，单击某表情图片，即可发送表情。
如果先按住 Ctrl 键，再单击表情图片，会先将表情发送到服务器，服务器收到表情后，再转发给接收方，这就是中转发送。

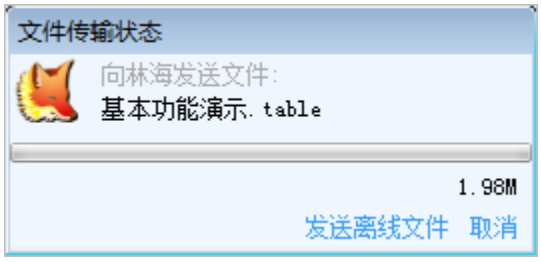
在双方网络连接情况不佳的时候，可以使用中转方式发送文字、图片和表情。
同样，对方在收到表情后，会返回一个回执信号，表示我收到表情了。
如果发送方在一定时间内，没有收到回执信号，会认为表情发送失败，并在窗口给出提示：



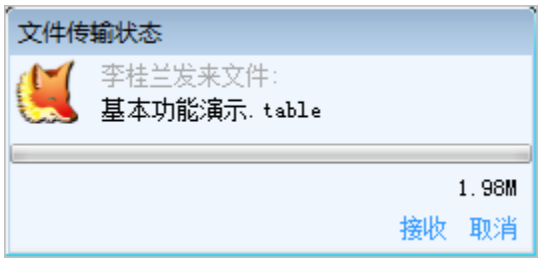
单击重发命令即可重发表情，或者按住 Ctrl 键的同时单击重发命令，即可以中转方式重发表情。
重要提示：
所有表情图片都位于 Foxtable 安装目录下的 faces 子目录，你可以增加删除表情图片。
需要特别注意的时，为提高效率，发送表情其实并不是发送表情图片，而是发送表情对应的图片文件名。
所以发送方和接收方的 faces 目录下的表情图片文件必须完全一致，否则发送方发送一个笑脸，接收方显示的可能是一个哭脸，甚至根本不会显示。

4.16.2.2.4.5 发送文件

发送方在内置聊天窗口，单击单击“发送文件”按钮，选择要发送的文件后， 屏幕右下角出现下图所示的窗口：

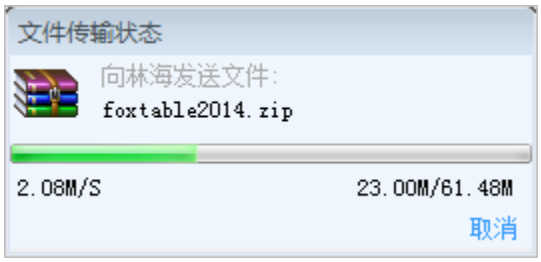


同时接收方的屏幕右下角会出现下图所示的窗口：



接收方可以选择接收文件，也可以单击取消拒绝接收文件
在等待接收方做出响应前，发送方可以选择发送离线文件，也就是先将文件发送到服务器，再由服务器转发给接收方。
我们可以一次选择多个文件发送。
如果在内置聊天窗口，按住 Ctrl 键的同时单击“发送文件”按钮，然后松开 Ctrl 键，选择要发送的

文件，系统会自动以离线形式发送文件，无需接收双方响应。
收发过程会显示进度，不管是发送方还是接收方，都可以随时取消：



和腾讯 QQ 不同，OpenQQ 的进度窗口显示在屏幕右下角，独立于聊天窗口，所以在文件收发过程中，我们可以关闭聊天窗口，方便其他操作的进行。
发送方的聊天记录有详细的文件发送记录，包括什么时候请求发送，什么时候开始发送，什么时候发送完成：



接收方的聊天记录亦有详细的文件接收记录：



如果是离线发送，服务器并不会立即将收到的文件转发给接收方，而是发一条信息通知接收方收到了离线文件：



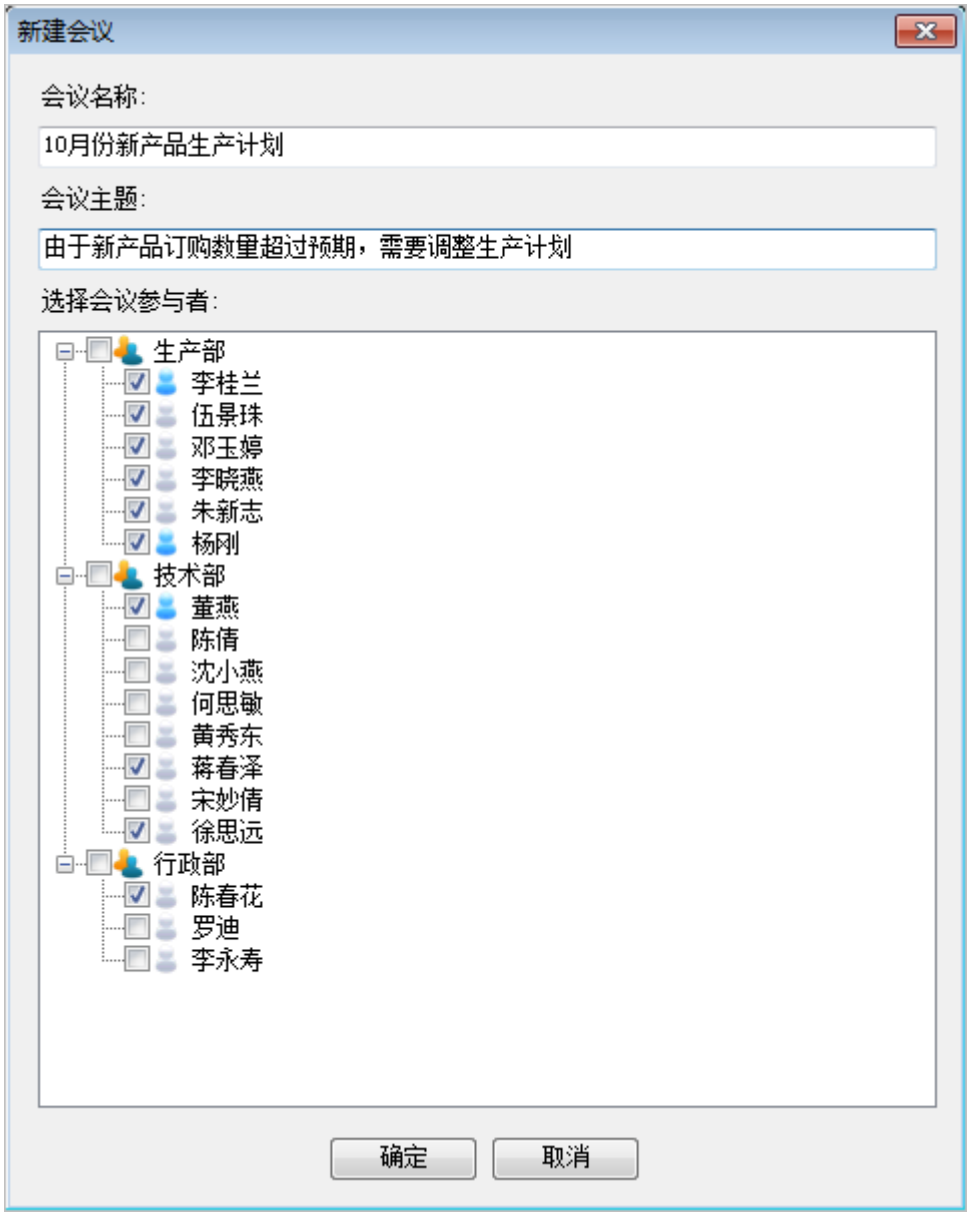
接收方可以单击下载命令，将文件下载到本地，接收方什么时候下载的，系统也会记录在聊天内

容中：

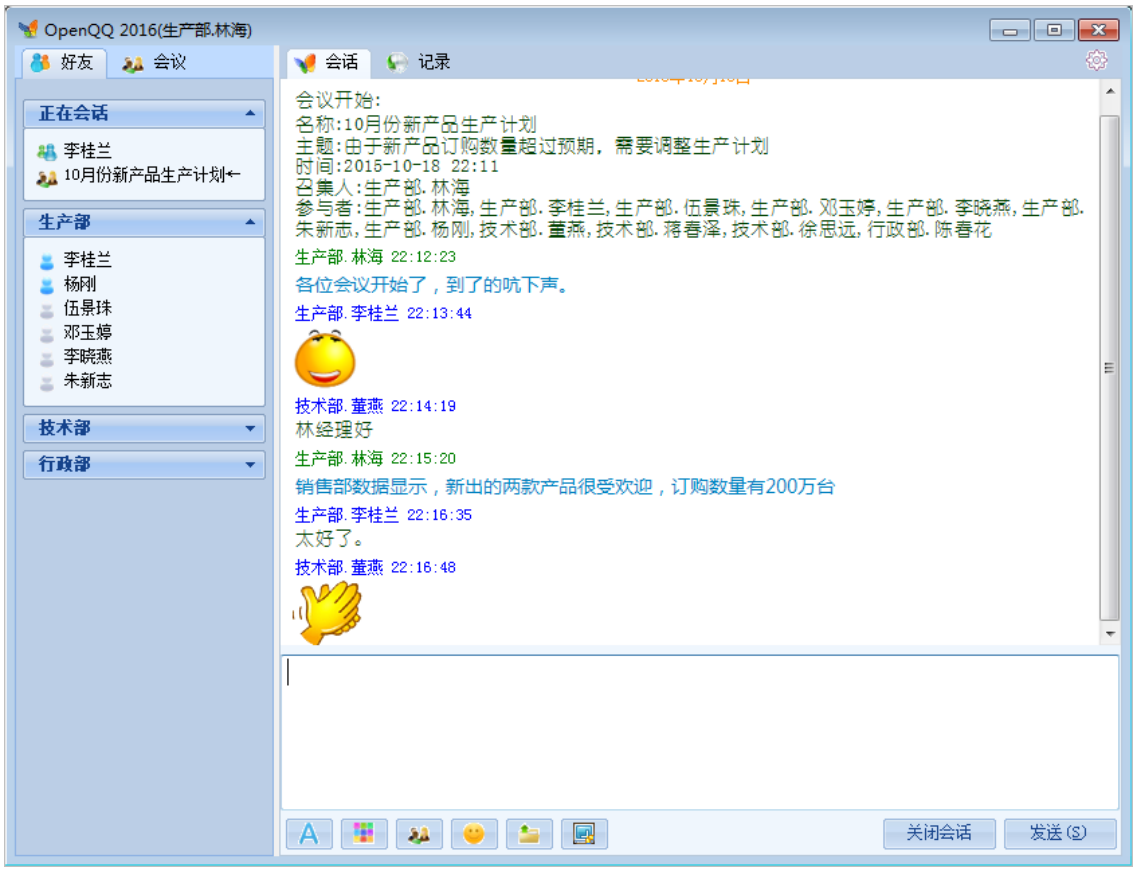


4.16.2.2.4.6 会议功能

内置聊天窗口没有群的功能，取而代之的是会议功能。
任何人都可以召集会议。
在内置聊天窗口，单击“召集会议”按钮，会出现新建会议窗口：



在这个窗口输入会议名称和主题，勾选参与会议的人员后，单击确定，既可开始会议：



参与会议的人相互之间可以不是好友，但必须都是会议召集人的好友。

所有会议参与人员都可以在会议上发言，分享图片和文件。

会议进行过程中，没有上线的会议参与人员，在上线之后能收到所有会议记录。

单击会议窗口左上角的设置按钮，会出现一个菜单，通过这个菜单可以重新选择会议参与人员，也可以结束会议。

只有会议召集人，才可以重选会议参与人员，或结束会议。

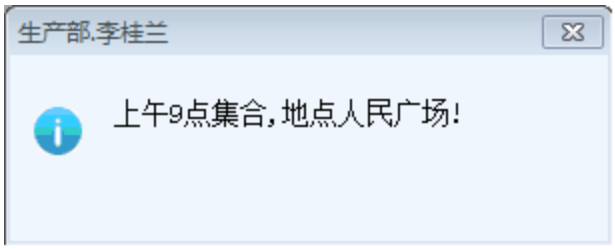
单击窗口左边的会议页签，会列出所有的会议，包括已经结束的历史会议。

4.16.2.2.4.7 发送弹窗消息

在内置聊天窗口，单击发送按钮右侧的“V”，会出现一个菜单：



如果单击“弹窗发送”，接收方在收到消息之后，屏幕右下角会自动弹出一个窗口，用于显示此消息：



提示：弹窗消息同样会记录并显示在内置聊天窗口，只是在此之外多了一个弹窗提示。

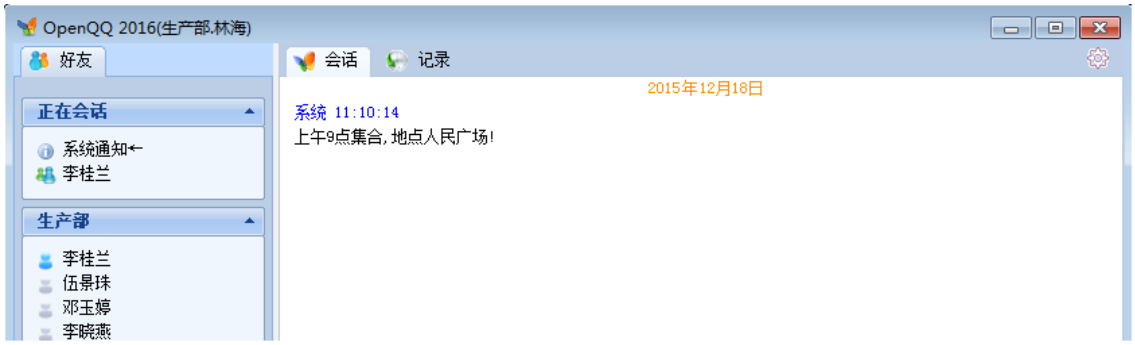
用代码发送弹窗消息

如果使用代码发送聊天消息，可以在消息前面加上"{!}"，例如：

```
QQClient.SendChatMessage("生产部.林海","{!}上午 9 点集合,地点人民广场!")
```

4.16.2.2.4.8 接收系统通知

在默认情况下，内置聊天窗口会显示一个系统通知对话，用于接收 QQServer 发来的系统通知：



如果不希望显示系统通知会话，可以在启动 QQClient 之前将其 ReceiveNotice 属性设置为 False。

QQServer 提供了三个方法，用于向客户端的内置聊天窗口发送系统通知，分别是：

- **SendNotice**

  用于发送文字通知，语法:

  SendNotice(Name,Message)

  Name: 用户名或者会议名。
  Message: 通知内容。

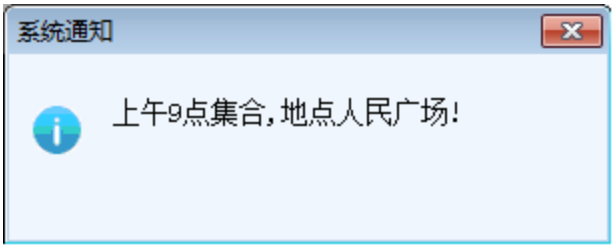
  例如：

  QQServer.SendNotice("生产部.林海","上午 9 点集合,地点人民广场!")

  如果希望客户端用弹窗显示收到的通知，可以在内容前加上"{!}"，例如：

  QQServer.SendNotice("生产部.林海","{!}上午 9 点集合,地点人民广场!")

客户端收到此通知后，同样会在内置聊天窗口显示此通知，但是会另外弹出一个窗口提示：



- **SendNoticeFile**

  用于发送文件通知，语法：

  SendNoticeFile(Name,File)

  Name: 用户名或会议名。
  File：要发送的文件名，含路径。

  例如：

  QQServer.SendNoticeFile("生产部.林海","c:\data\管理项目 1.rar")

- **SendNoticeImage**

  用于发送图片通知，语法：

  SendNoticeImage(Name,File)

  Name: 用户名或会议名。
  File：要发送的图片文件名，含路径。

  例如：

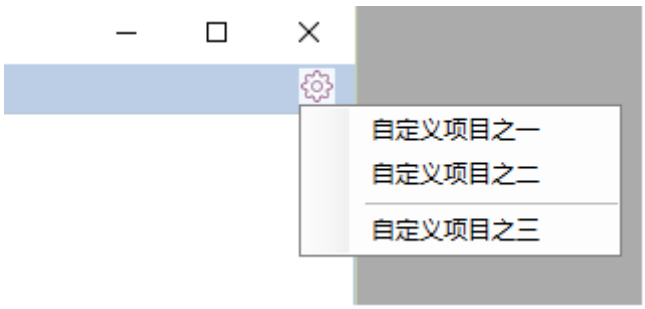
  QQServer.SendNoticeImage("生产部.林海","c:\data\123.jpg")

4.16.2.2.4.9 自定义聊天菜单

单击内置聊天窗口右上角的按钮，会弹出一个菜单，我们可以对这个菜单进行自定义。
自定义聊天菜单很简单，设置 QQClint 的 MenuItems 属性即可，例如：

QQClient.MenuItems = "自定义项目之一|自定义项目之二|-|自定义项目之三"

执行上述代码后，打开内置聊天窗口，菜单会变为：



提示：

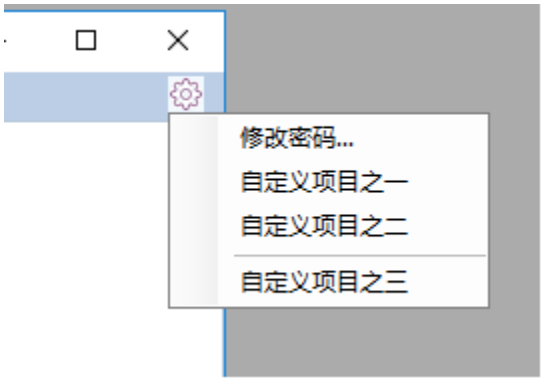
1、符号"-"在对应位置插入一个分割条。

2、需要在打开内置聊天窗口之前设置 MenuItems，否则不会生效。

如果在设置 MenuItems 属性时，以符号"|"开头，则会保留原有菜单项目，例如：

QQClient.MenuItems = "|自定义项目之一|自定义项目之二|-|自定义项目之三"

菜单将变为：



单击自定义菜单后，会触发事件 CustomMenuClick，通常将该事件的代码设置为：

```
Select e.MenuText
    Case "自定义项目之一"
        '要执行的代码
    Case "自定义项目之二"
        '要执行的代码
    Case "自定义项目之三"
        '要执行的代码
End Select
```

如果要根据当前选定用户和消息内容动态显示不同的菜单，可以在 BeforeShowMenu 事件设置代码，例如：

```
If e.BuddyName.StartsWith("生产部.") Then '如果当前用户的分组是生产部
    QQClient.MenuItems = "生产部菜单项之一|生产部菜单项之二"
ElseIf e.BuddyName.StartsWith("销售部.") Then '如果当前用户的分组是销售部
    QQClient.MenuItems = "销售部菜单项之一|销售部菜单项之二"
Else
    QQClient.MenuItems = "|"   '使用默认菜单
End If
```

4.16.2.2.4.10 其他杂项

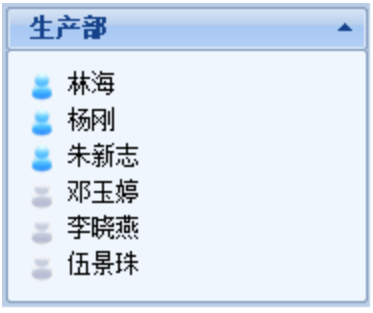
客户端的历史记录

客户端的聊天记录保存在项目文件夹的QQClient目录下，格式为Access。
内置聊天窗口只有历史记录查看功能，并没有提供历史记录管理功能，大家可以自行扩展。
接收到的文件存储在QQClient目录下的ReceivedFiles子目录中，按月份目录存储，以免单个目录下文件过多。

服务端的离线消息

离线消息存储在服务端项目文件夹QQServer子目录下，格式为Access，等接收方上线后，QQServer会将离线消息转发给接收方，然后删除离线消息。
如果接收方始终不上线，那么离线消息会始终存储在服务端，可能导致文件越来越大。
QQServer并没有提供离线消息管理的功能，大家可以自行扩展，例如做个计划，每天运行一次，用SQL语句自动删除两个星期之前的离线消息。
发送的离线文件，以及在会议中分享的文件，存储在QQServer目录下的UploadFiles子目录中，按月份目录存储，以免单个目录下文件过多。

字体和颜色设置
内置聊天窗口提供了字体和颜色设置按钮，我方和对方的字体以及颜色，可以分别设置，以示区分。
对模式窗口进行截图
由于模式窗口打开后，在关闭窗口前不能进行任何操作，所以无法调用内置聊天窗口截图和发送消息。
如果在模式窗口同时按 Ctrl+Shift+F6，可以打开内置聊天窗口，此时内置聊天窗口可以对此模式窗口进行截图。
在线状态提示
OpenQQ 并没有提供头像设置功能，所有用户的头像都是一样的，但是会用不同的颜色来区分离线用户和在线用户：



在对话过程中，对方上线或下线都会即时给出提示：



4.16.2.2.5 关于 PopMessage

你可能已经留意到，OpenQQ 提供的示例文件，从不用 MessageBox 显示提示信息，而是用 PopMessage。

PopMessage 和 MessageBox 类似，同样用于提示信息。

PopMessage 的好处在于：

1、信息显示在屏幕右下角。

2、不会中断程序的执行，可以继续进行任何操作。

3、可以设置信息显示的时长，达到指定时长后会自动关闭提示。

但是 PopMessage 没有提供和用户交互的按钮，所以 PopMessage 适合显示单纯的提示信息，MessageBox 适合显示需要和用户交互的提示信息。

需要特别说明一点：

在使用 OpenQQ 的时候，尽量用 PopMessage，不要用 MessageBox；因为 OpenQQ 异步接收信息，但是同步触发事件，而 MessageBox 会阻断事件的触发。

MessageBox 显示后，OpenQQ 可以继续收取信息，但是直到用户关闭 MessageBox 对话框，系统都不会触发任何事件。

所以如果用户长时间不关闭 MessageBox，会积压大量的信息待处理。

所以我们应该尽量用 PopMessage，而不是 MessageBox，特别是服务端项目，最好杜绝 MessageBox，因为服务端信息量通常远远大于客户端，而且服务端通常也不需要和用户交互。

即使需要和用户交互，也可以自己设计一个模式窗口来显示提示信息。

参考：PopMessage

4.16.2.2.6 路由器与 NAT 穿透

普通用户可以忽略本节内容。
为减轻服务器负担，OpenQQ 在收发文字、图片和文件的时候，通常无需经过服务器器中转，客户端和客户端之间是直接通讯的，即 p2p 方式。
我们知道，外网计算机是不能主动发送数据给内网计算机的，只有收到过内网计算机发来的数据之后，外网计算机才能向该内网计算机发送数据。

假设客户端 A 和客户端 B 分别位于不同的局域网中，那么对于 A、B 相互就是外网计算机，不管是 A 发给 B，还是 B 发给 A，对方都是收不到的。
那么 A 和 B 是怎么建立信道进行通讯的呢？这就涉及到 NAT 穿透的问题，穿透要通过服务器搭桥，所以任何 p2p 的软件，都需要有服务器的。

但是并非所有路由器的 NAT 都能成功穿透，如果 OpenQQ 检查到 A 和 B 之间没有穿透成功，那么 A 和 B 之间的通讯就会自动通过服务器中转。
所以不管 NAT 穿透是否成功，对于最终用户的使用基本上是没有影响的。

怎么判断 NAT 穿透是否成功呢？ 方法有两个：
1、在内置聊天窗口向对方发送文件，如果没等双方响应，系统自动开始用离线文件形式发送，说明 NAT 穿透不成功，和对方的 p2p 信道没有建立起来。
2、QQBuddy 提供了一个 p2p 属性，如果已经穿透对方的 NAT，数据无需经过服务器中转，可直达对方，此属性会返回 True，例如：

```
For Each bd As QQBuddy In QQClient.Buddies
    If bd.Online Then
        Output.show(bd.Name & iif(bd.p2p,"(已穿透)","(未穿透)"))
    End If
Next
```

路由器 NAT 的类型有以下四种：

完全锥型(Full Cone NAT)
限制锥型(Restricted Cone NAT)
端口限制锥型(Port Restricted Cone NAT)
对称型(Symmetric NAT)
前三种 NAT 类型都是可以穿透的，只有最后一种对称型无法穿透。
所以我们在采购路由器的时候，可以了解一下其 NAT 类型，另外企业用的话，最好用企业级的路由器。
好消息是，市面上绝大多数路由器都是前三种类型的，我到目前为止测试了不下 10 种路由器，还没有遇到完全不能穿透的。
而且一般企业的 OpenQQ 通讯数据量，就是完全通过服务器中转，也不至于让服务器服承担过重，除非持续发送大文件。
此外，局域网也不存在 NAT 穿透的问题。
所以对于多数用户来说，本节的问题其实不是问题，有所了解即可。

需要注意的是，穿透不成功并非总是 NAT 类型的问题，可能是被防火墙阻拦了；也可能可能是网络问题，伙伴们都知道，国内三大电信运营商之间的互联互通情况还是很不理想的。

4.16.2.3 类型参考

4.16.2.3.1 QQServer

4.16.2.3.1.1 属性

QQServer 包括以下属性：

属性名	说明
ServerIP	字符型，指定 QQserver 的 IP 地址。
ServerPort	整数型，指定 QQServer 的登录端口。
PortRange	字符型，指定 QQServer 的随机端口范围，参考：指定端口范围
HeartbeatInterval	整数型，指定心跳信号间隔，默认为 30，单位为秒
HeartbeatTimeout	整数型，指定心跳信号超时时长，默认为 100，单位为秒，也就是 100 秒内没有收到某客户端的心跳信号，就认为该客户端已经离线。
Ready	逻辑型，用于判断 QQserver 是否处于启动状态
Buildin	逻辑型，是否使用内置身份验证，默认为 False。
Users	QQUser 集合，包括所有已经登录的用户；QQUser 有两个属性，Name 返回用户名，Tag 用于设置用户的附加数据。

示例一

启动 QQServer:

```
QQServer.ServerIp = "192.168.100.6" '指定IP地址
QQServer.ServerPort = 52177 '指定端口
QQServer.HeartbeatInterval = 30 '指定心跳时间
QQServer.HeartbeatTimeout = 100 '指定心跳超时时间
QQServer.Start() '启动QQServer
```

示例二

列出所有已经登录的用户：

```
For Each User As QQUser In QQserver.Users
    Output.Show(User.Name)
Next
```

4.16.2.3.1.2 方法

4.16.2.3.1.2.1 Start

启动 QQServer。
例如：

```
QQServer.ServerIp = "192.168.100.6" '指定 IP 地址
QQServer.ServerPort = 52177 '指定端口
QQServer.HeartbeatInterval = 30 '指定心跳时间
QQServer.HeartbeatTimeout = 100 '指定心跳超时时间
QQServer.Start() '启动 QQServer
```

4.16.2.3.1.2.2 Stop

停止 QQServer。

4.16.2.3.1.2.3 Send

发送信息，语法：
Send(UserName,Message,AnyWay)
UserName: 客户端名称
Message： 信息内容
AnyWay： 可选参数，逻辑型，如果客户端不在线，是否发离线消息。
例如：

```
QQServer.Send("生产部.林海","测试信息")
```

在默认情况下，只有对方在线，QQserver 才会将消息发送出去，否则直接返回。

如果希望在对方离线的时候，依然发送离线消息，可以将第三个参数设置为 True，例如：

```
QQServer.Send("生产部.林海","测试信息",True)
```

这样 QQServer 会先将消息暂存起来，等到客户端上线，再将消息发出。

4.16.2.3.1.2.4 SendFile

用代码向客户端发送文件。
如果客户端没有上线，此方法会直接返回，不会发送离线文件。
语法：

```
SendFile(UserName, FileName)
```

UserName：用户名。
FileName： 文件名，含路径。
例如：

```
If QQServer.Users.Contains("生产部.李桂兰") Then '判断此用户是否在线
    QQServer.SendFile("生产部.李桂兰", "c:\data\abc.doc")
End If
```

SendFile 方法用异步方式，在后台静默发送文件，发送完毕后，会通过事件来通知双方：
1、发送方触发 OpenQQ 服务端事件 AfterSendFile。
2、接收方触发 OpenQQ 客户端事件 AfterReceiveFile。

4.16.2.3.1.2.5 SendNotice

用于向客户端发送文字通知，语法:

SendNotice(Name,Message)

Name: 用户名或者会议名。
Message: 通知内容。

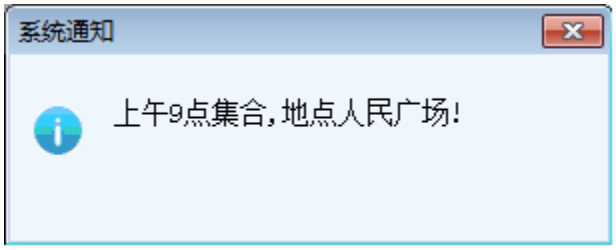
例如：

QQServer.SendNotice("生产部.林海","上午 9 点集合,地点人民广场!")

如果希望客户端用弹窗显示收到的通知，可以在内容前加上"{!}"，例如：

QQServer.SendNotice("生产部.林海","{!}上午 9 点集合,地点人民广场!")

客户端收到此通知后，同样会在内置聊天窗口显示此通知，但是会另外弹出一个窗口提示：



4.16.2.3.1.2.6 SendNoticeFile

用于发送文件通知，语法：

SendNoticeFile(Name,File)

Name: 用户名或会议名。
File：要发送的文件名，含路径。

例如：

```
QQServer.SendNoticeFile("生产部.林海","c:\data\管理项目 1.rar")
```

4.16.2.3.1.2.7 SendNoticeImage

用于发送图片通知，语法：

```
SendNoticeImage(Name,File)
```

Name: 用户名或会议名。
File：要发送的图片文件名，含路径。

例如：

```
QQServer.SendNoticeImage("生产部.林海","c:\data\123.jpg")
```

4.16.2.3.1.2.8 KickOff

踢出用户。
语法：
KickOff(UserName)
UserName：字符型，指定用户名，此用户将被踢出。
例如：

```
If QQServer.Users.Contains("生产部.李桂兰") Then '判断此用户是否登录
    QQServer.KickOff("生产部.李桂兰") '踢出此用户
End If
```

4.16.2.3.1.3 事件

4.16.2.3.1.3.1 UserLogging

有用户申请登录时触发。
e 参数方法：
UserName: 字符型，返回登录用户名
Password: 字符型，返回登录用户密码
Success： 逻辑型，默认为 False，设置为 True，则允许用户登录
Buddies： 字符型集合，用于设置登录者的好友名单。
Message： 字符型，通常在禁止用户登陆时，返回一个错误提示给用户；或者在登录成功时，给用户返回一个欢迎信息。

User： UserInfo 类型，当用户使用内置身份验证登录 OpenQQ， 通过验证后此属性返回登录用户，否则返回 Nothing。

4.16.2.3.1.3.2 UserLogged

有用户登录成功后触发。
e 参数属性：
UserName：字符型，返回登录用户名。

4.16.2.3.1.3.3 UserLogout

有用户退出登录后触发。
e 参数属性：
UserName：字符型，返回退出登录用户名。
Mode： 整数型，返回用户退出登录的原因，请参考下表：

Mode 值	对应原因
1	用户主动退出，例如关闭项目或执行 QQClient 的 Stop 方法。
2	用户被服务器踢出。
3	服务器在规定时间内容内没有收到用户的心跳信号。
4	同名用户在其他位置重新登录，导致当前登录被服务器中断。

4.16.2.3.1.3.4 ReceivedMessage

收到来自用户(客户端)发来的信息时触发。
e 参数属性：
UserName: 字符型，返回用户名。
Message： 字符型，返回消息内容。
ReturnValue： 字符型，用于设置需要给用户返回的信息。

4.16.2.3.1.3.5 AfterSendFile

向客户端发送文件完毕后触发。
只有使用 SendFile 发送文件，才会触发此事件。
e 参数属性：
UserName：字符型，返回用户名称。
FileName：字符型，返回发送的文件名，含路径。
Success： 逻辑型，如果发送成功，返回 True，否则返回 False。

4.16.2.3.1.3.6 AfterReceiveFile

服务器收到客户端发来的文件后触发。
只有客户端使用 SendFile 向服务器发送文件，服务端才会触发此事件。
e 参数属性：
UserName：字符型，返回客户端用户名称。
FileName：字符型，返回收到的文件名，含路径。
Success： 逻辑型，如果接收成功，返回 True，否则返回 False。

4.16.2.3.1.3.7 UserChangePassword

用户(客户端)申请更改密码时触发。
e 参数属性：
UserName： 字符型，返回用户名。
OldPassword: 字符型，返回用户提交的旧密码。
NewPassword： 字符型，返回用户提交的新密码。
ReturnMessage：字符型，返回给用户的信息。
参考：让用户自主设置密码

4.16.2.3.2 QQClient

4.16.2.3.2.1 属性

QQClient 包括以下属性：

属性名	说明
ServerIP	字符型，指定 QQserver 的 IP 地址。
ServerPort	整数型，指定 QQServer 的端口号。
UserName	字符型，指定登录用户名。
Password	字符型，指定登录用户密码。
ServerMessage	字符型，用户在登录成功或失败后，获取服务器返回的信息。
Ready	逻辑型，用于判断 QQClient 是否已经启动。
Buddies	QQBuddy 集合，包括所有的好友，QQBuddy 有四个属性： Name： 字符型，返回好友名称 Online：逻辑型，用于判断好友是否在线 p2p： 逻辑型，用于判断是否已经穿透对方 NAT Tag： 用于设置好友的一些附加数据，可以是任何数据类型。
ThroughServer	逻辑型，默认为 False，如果设置为 True，在发送消息时，不管对方是否在线，都会通过服务器转发。
UnreadCount	整数型，返回未读消息数量，仅统计通过内置聊天窗口收发的信息。

ReceiveNotice	逻辑型，是否在内置聊天窗口接收并显示 QQServer 发来的系统通知，默认为 True，通常无需设置。
MenuItems	字符型，用于设置自定义菜单，参考自定义聊天菜单
AllowSendFile	逻辑型，是否允许用户通过内置聊天窗口传输文件，默认为 True。

示例

```
QQClient.ServerIP = "127.0.0.1" '指定服务器IP地址
QQClient.ServerPort = 52177  '指定服务器端口
QQClient.UserName = "生产部.李桂兰"
QQClient.Password = "888"
If QQClient.Start() = True '如果登录成功
    For Each bd As QQBuddy In QQClient.Buddies '显示好友名和在线状态
        Output.Show(bd.Name & "(" & IIF(bd.Online,"在线","离线") & ")")
    Next
Else '如果登录失败,显示服务器返回错误信息
    PopMessage("QQClient 登 录 失 败 , 原 因 :" & vbcrlf & QQClient.ServerMessage," 提 示 ",PopiconEnum.Error,5)
End If
```

4.16.2.3.2.2 方法

4.16.2.3.2.2.1 Start

启动 QQClient
语法:
Start(TimeOut)
TimeOut： 可选参数，整数型，默认值为 5，单位为秒；用于指定超时时间，在此时间内没有收到服务器反馈，则认为登录失败。

4.16.2.3.2.2.2 Stop

QQClient 退出登录。

4.16.2.3.2.2.3 Send

用于发送信息，语法为：
Send(BuddyName, Message, Anyway)
BuddyName: 好友名称
Message： 信息内容
AnyWay： 可选参数，逻辑型，如果对方不在线，是否发离线消息。
例如向生产部的林海发送消息：

```
QQClient.Send("生产部.林海","今天的任务完成了吗")
```

离线消息

在默认情况下，只有对方在线，QQClient 才会将消息发送出去，否则直接返回。

如果希望在对方离线的时候，依然发送离线消息，可以将第三个参数设置为 True，例如：

```
QQClient.Send("生产部.林海","今天的任务完成了吗",True)
```

实际上，此时 QQClient 将消息发送给了服务端，服务端收到此消息后，会将该消息保存下来， 等到好友上线后，服务端将此消息转发给好友。

使用 Send 发送的消息，并不会出现在内置的聊天窗口中，我们必须使用代码来接收之。

向服务器发送消息

QQClient 用 Send 发送信息时，如果不指定接收者，就会直接发给服务器：
QQClient.Send("","明天天气不错")
或直接：
QQClient.Send("明天天气不错")

4.16.2.3.2.2.4 SendWait

SendWait 用于向服务器发送信息，并在收到服务器回复之前，停止执行后续代码。
语法：
SendWait(Message,TimeOut)
Message: 字符型，指定要发送的内容。
TimeOut: 整数型，可选参数，用于设置等待时间，默认值为 5，单位为秒，如果在此时间内没有收到服务器回复，则继续执行后续代码。

示例
假定客户端需要获取服务器时间。
1、首先在服务器端的 ReceivedMessage 事件设置代码：

```
If e.Message = "GetDate" Then
    e.ReturnValue = Date.Now
End If
```

2、客户端可以通过下面的代码获取服务器时间：

```
If QQClient.Ready Then
    Dim msg As String = QQClient.SendWait("GetDate",5)
    If msg = "" Then
        Popmessage("未获服务器响应!","提示",PopIconEnum.Error)
    Else
        Popmessage("当前时间为:" & msg,"提示",PopIconEnum.Infomation)
```

```
    End If
End If
```

提示：GetDate不是固定的，只是客户端可服务器约定的字符，你可以约定各种字符，用于客户端和 服务器之间的交互。

Send 和 SendWait 的差别

和 SendWait 一样，上一节介绍的 Send 也可以向服务器发送消息，但二者是有差别的。

Send 收发过程：

1、Send 不返回值，发送消息后直接执行后续代码，并不等待服务器返回消息。

2、服务器收到 Send 发送的消息后，触发 QQServer 的 ReceivedMessage 事件，服务器可以通过 e 参数 ReturnValue 返回消息。

3、客户端收到服务器返回的消息后，触发 QQClient 的 ReceivedMessage 事件，通过这个事件获取服务器返回的消息。

SendWait 收发过程：

1、SendWait 会返回值，发送消息后会暂停执行，直到收到服务器返回的消息，才会执行后续代码。

2、服务器收到 SendWait 发送的消息后，同样触发 QQServer 的 ReceivedMessage 事件，服务器可以通过 e 参数 ReturnValue 返回消息。

3、客户端收到服务器返回的消息后，并不触发 QQClient 的 ReceivedMessage 事件，而是作为 SendWait 的返回值。

4、如果在规定时间内，SendWait 没有收到服务器的返回值，那么 SendWait 返回空值，并继续执行后续代码。

显然，当需要和服务器即时交互的时候，使用 SendWait，否则使用 Send。

4.16.2.3.2.2.5 SendFile

用代码向好友或服务器发送文件。

向好友发送文件的语法：

SendFile(BuddyName, FileName,ShowProcess)

BuddyName： 好友名。

FileName： 文件名，含路径。

ShowProcess：可选参数，辑型，是否显示发送进度，默认为 False。

向服务器发送文件的语法：

SendFile(FileName,ShowProcess)

FileName： 文件名，含路径。

ShowProcess：可选参数，逻辑型，是否显示发送进度，默认为 False。

用 SendFile 向好友发送文件时，如果好友不在线，将直接返回，并不会以离线文件形式发送。

例如：

```
Dim bd As QQBuddy = QQClient.Buddies("技术部.林欣华")
If bd.OnLine Then '如果对方在线
    QQClient.SendFile("技术部.林欣华","c:\data\151022.doc")
End If
```

SendFile 采用异步方式，在后台静默发送文件，当文件发送完毕后，会通过事件来通知双方：
1、发送方会触发 OpenQQ 客户端事件 AfterSendFile。
2、如果是向好友发送，接收方会触发 OpenQQ 客户端事件 AfterReceiveFile。
3、如果是向服务器发送，接收方会触发 OpenQQ 服务端事件 AfterReceiveFile。

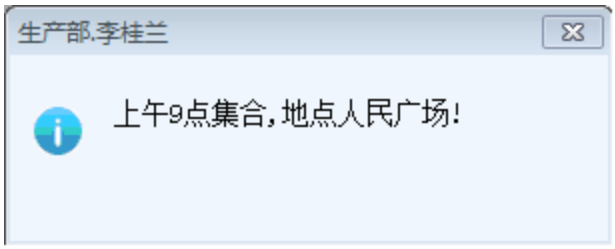
4.16.2.3.2.2.6 SendChatMessage

向好友或会议发送信息，不会触发事件，信息会显示在双方的内置聊天窗口中，如果对方不在线，自动转为离线消息发送。

本质上和直接在聊天窗口输入内容后单击发送按钮是一回事，通常用于自动发送一些文字性通知。
语法：
SendChatMessage(BuddyName, Message)
BuddyName: 好友名称或会议名称。
Message： 消息内容
例如：
QQClient.SendChatMessage("生产部.林海","上午 9 点集合,地点人民广场!")
如果希望对方以弹窗显示收到的信息，可以在消息前面加上"{!}"，例如：

```
QQClient.SendChatMessage("生产部.林海","{!}上午 9 点集合,地点人民广场!")
```

接收方在收到消息之后，屏幕右下角会自动弹出一个窗口，用于显示此消息：



提示：弹窗消息同样会记录并显示在内置聊天窗口，只是在此之外多了一个弹窗提示。

4.16.2.3.2.2.7 SendChatFile

用于向好友或者会议发送文件。
语法：
SendChatFile(BuddyName, FileName， ShowProcess)
BuddyName: 好友名称或会议名称。
FileName： 文件名，含路径。
ShowProcess：可选参数，辑型，是否显示发送进度，默认为 False。
例如：

QQClient.SendChatFile("技术部.林欣华","c:\data\151022.doc")
和 SendFile 一样，SendChatFile 同样用异步方式 在后台静默发送，默认也不会显示进度条。

和 SendFile 不同的是:

1、SendChatFile 发送完毕之后，不会触发双方的事件，而是将文件显示在双方的聊天窗口中。
2、如果对方不在线，自动以离线文件形式发送。

所以用 SendChatFile 发送文件，本质上和通过内置聊天窗口发送文件是一样的。

4.16.2.3.2.2.8 SendChatImage

用于向好友或者会议发送图片。
语法：
SendChatImage(BuddyName, FileName, ShowProcess)
BuddyName: 好友名称或会议名称。
FileName： 图片文件名，含路径。
ShowProcess：可选参数，辑型，是否显示发送进度，默认为 False。
SendChatImage 发送完毕后，不会触发双方的事件，而是在双向聊天窗口显示图片。
如果对方不在线，SendChatImage 会自动以离线图片形式发送。

所以用 SendChatImage 发送图片，本质上和通过内置聊天窗口发送图片是一样的。

4.16.2.3.2.2.9 RequestFile

客户端请求 QQServer 发送指定的文件到本机。
语法：
RequestFile(FileName,TimeOut,ShowProcess)
FileName：指定要下载的文件名，包括相对路径
TimeOut： 可选参数，指定等待服务器响应的时间，单位为秒，默认为 5 秒。
ShowProcess：可选参数，辑型，是否显示接收进度，默认为 False。
RequestFile 会返回一个整数值，0 表示提供的路径或文件名错误，1 表示请求成功，2 表示在规定时间内没有收到服务器没有反应。
说明：
RequestFile 只是简单地向 QQServer 发一个约定格式的型号，QQServer 收到这个消息后， 先返回一个消息，告知客户端是否接受其请求，再利用 SendFile 方法将指定的文件发送给客户端(如果接受客户端请求的话)。
大家可以利用这个思路，进行自己的功能扩展。
RequestFile 只能请求 QQServer 的 UploadFiles 子目录下的文件，用户上传的文件也存储在这里，按月分目录存储，以避免单个目录下文件过多。
文件名要包括月份路径，使用相对路径即可，例如要请求 QQServer 发送 2015 年 10 月份目录下的某个文件：

```
Dim ret As Integer = QQClient.RequestFile("201510\123.jpg",2)
If ret= 0 Then
```

```
    PopMessage("文件名或路径错误")
ElseIf ret= 2 Then
    PopMessage("没有收到服务器的回应!","提示",PopIconEnum.Error)
End If
```

上述代码如果请求成功，并不会出现出现任何提示，默认也不会显示文件接收进度（如需显示进度，可以将 ShowProcess 设置为 True），可以继续进行其他操作，好像什么都没有发生。
当文件下载完成后，会触发 AfterReceiveFile 事件。

需要注意的是，如果本地已经有 abc.jpg 这个文件，下载完成后，文件会被自动更名为 abc(1).jpg，如果 abc(1).jpg 也已经存在，就会被自动更名为 abc(2).jpg。

4.16.2.3.2.2.10 ChangePassWord

用于向服务端申请更改密码。
语法：
ChangePassWord(OldPassword,NewPassword)
OldPassword: 原密码
NewPassword：新密码
参考：让用户自主设置密码。

4.16.2.3.2.2.11 OpenChatWindow

打开内置的聊天窗口。
语法：
OpenChatWindow()
OpenChatWindow(Caption)
OpenChatWindow(Caption,IconFile)

参数说明：

Caption: 用于指定内置聊天窗口标题
IconFile: 用于指定内置聊天窗口的图标文件，必须为 Icon 格式，如果文件已经事先复制到管理项目的 Images 子目录下,则只需指定文件名即可,否则需要包括路径。

4.16.2.3.2.3 事件

4.16.2.3.2.3.1 Connected

QQClient 连接到 QQServer 后触发，也就是客户端上线后触发。

4.16.2.3.2.3.2 Disconnected

用户（QQClient）和服务器(QQServer)连接中断后触发。
e 参数属性：
Mode： 整数型，返回连接中断的原因，请参考下表：

Mode 值	对应原因
1	用户主动退出，例如关闭项目或执行 QQClient 的 Stop 方法。
2	用户被服务器踢出。
3	服务器在规定时间内容内没有收到用户的心跳信号。
4	同名用户在其他位置重新登录，导致当前登录被服务器中断。
5	服务器端关闭 QQServer。

4.16.2.3.2.3.3 BuddyLogin

有好友上线时触发。
e 参数属性：
BuddyName：上线好友的名称

4.16.2.3.2.3.4 BuddyLogout

有好友下线时触发。
e 参数属性：
BuddyName：下线好友的名称。
Mode： 整数型，返回好友下线原因，请参考下表：

Mode 值	对应原因
1	好友主动退出，例如好友关闭项目或执行 QQClient 的 Stop 方法。
2	好友被服务器踢出
3	服务器在规定时间内容内没有收到好友的心跳信号。
4	同名好友在其他位置重新登录，导致当前登录被服务器中断。

4.16.2.3.2.3.5 ReceivedMessage

收到来自好友或服务器的信息时触发。
e 参数属性有：
UserName：字符型，返回好友名称，如果是服务端(QQServer)发来的信息，则返回空。
Message: 字符型，返回信息内容。

4.16.2.3.2.3.6 AfterSendFile

向服务器或好友发送文件完毕后触发。
只有使用 SendFile 发送文件，才会触发此事件。
e 参数属性：
UserName：字符型，返回好友名称，如果是向服务器发送，则返回空
FileName：字符型，返回发送的文件名，含路径。
Success： 逻辑型，如果发送成功，返回 True，否则返回 False。
ServerFileName：如果是向服务器发送，可以通过此属性获取此文件在服务器对应的文件名，具体参考下面的解释。
关于 ServerFileName
假定我们向服务器或好友发送一个文件名，名称为：abc.doc
对方收到文件后，如果已经存在名为 abc.doc 的文件，会将新收到的文件改名为 abc(1).doc，依次类推。
多数时候，这不是问题，因为接收方的 AfterReceiveFile 事件可以获取改名后的文件。
但是如果我方向服务器发送一个文件，然后通知其他好友下载这个文件，那么需要直道这个文件在服务器对应的文件名，这就是 ServerFileName 的作用。

4.16.2.3.2.3.7 AfterReceiveFile

收到好友或者服务器发来的文件后触发。
只有服务器或好友使用 SendFile 发送文件，才会触发此事件。
e 参数属性：
UserName：字符型，返回好友名称，如果是服务器发来文件，则返回空。
FileName：字符型，返回收到的文件名，含路径。
Success： 逻辑型，如果接收成功，返回 True，否则返回 False。

4.16.2.3.2.3.8 UserChangePassword

在内置聊天窗口单击“修改密码”命令时触发。
参考: 让用户主动设置密码

4.16.2.3.2.3.9 BeforeShowMenu

在内置聊天窗口的菜单显示前触发，通常用于动态设置菜单。
e 参数：
BuddyName: 字符型，返回内置聊天窗口当前选定用户名
Content: 字符型，返回内置聊天窗口当前选定的消息内容
ContentType: 整数型，返回内置聊天窗口单签选定的消息类型
Sender: 字符型，返回内置聊天窗口当前选定消息的发送者
SendTime: 日期时间星，返回内置聊天窗口当前选定消息的发送时间
参考： 自定义聊天菜单
ContentType 类型表：

值	类型
0	文字
1	表情
2	图片
3	文件
4	已经下载的离线文件
5	还未下载的图片
6	还未下载的离线文件
7	常规系统消息
8	错误系统消息
9	成功系统消息
10	发送失败消息

4.16.2.3.2.3.10 CustomMenuClick

单击聊天窗口中的自定义菜单项目后触发。
e 参数：
MenuText: 所单击菜单项目的标题
BuddyName: 字符型，返回内置聊天窗口当前选定用户名
Content: 字符型，返回内置聊天窗口当前选定的消息内容
ContentType: 整数型，返回内置聊天窗口单签选定的消息类型
Sender: 字符型，返回内置聊天窗口当前选定消息的发送者
SendTime: 日期时间星，返回内置聊天窗口当前选定消息的发送时间
参考： 自定义聊天菜单
ContentType 类型表：

值	类型
0	文字
1	表情
2	图片
3	文件
4	已经下载的离线文件
5	还未下载的图片
6	还未下载的离线文件
7	常规系统消息
8	错误系统消息
9	成功系统消息

10	发送失败消息

4.16.2.4 应用实例

4.16.2.4.1 控制在线用户数

有了 OpenQQ,可以很方便地控制系统的在线用户数。
服务端的 UserLogging 事件代码设置为：

```
If QQServer.Users.Count >= 10 Then
    e.Success = False
    e.Message = "已经达到最大允许的在线用户数!"
Else
    '验证用户名和密码的代码
End If
```

客户端的AfterOpenProject事件:

```
QQClient.ServerIP = "192.168.100.8" '指定服务器IP地址
QQClient.ServerPort = 52177  '指定服务器端口
QQClient.UserName = "生产部.李桂兰"
QQClient.Password = "888"
If QQClient.Start() = False '如果登录失败
     MessageBox.show(" 登 录 失 败 , 原 因 :"  &  vbcrlf  &  QQClient.ServerMessage," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Syscmd.Project.Exit()
    Return
End If
```

4.16.2.4.2 即时同步数据

在多用户协同工作的情况下，如果某用户修改数据后保存，其他用户只有重新加载才能看到最新的数据，参考：数据同步
实际上大多数程序，都是这种被动刷新的方式，例如你在使用网银的时候，只有刷新页面，才能看到刚刚发生的交易；在论坛浏览帖子的时候，只有刷新页面，才能看到他人刚发表的帖子。
也有一些程序，是定时刷新，不管数据有没有变动，到了一定时间，都重新加载数据。
但是上述方式，都没有办法做到即时同步，而且这些方式并非只刷新已经变动的数据，而是重新加载整个页面、甚至整个表，大大增加了服务器的负担，既不即时，也不高效。
有了 OpenQQ，我们可以保存某行数据后，通知其他好友，这一行的数据有变化，请重新加载，这样就能做到即时、高效了。
设计步骤：

1、在编辑窗口加一个保存按钮，按钮代码设置为：

```
Dim r As Row = Tables("表A").Current
Dim p As String = IIF(r.DataRow.RowState = DataRowState.Added, "A#", "U#")
r.Save()
'一定要在保存后合成信息,因为新增行的主键在保存后才生成
Dim msg As String = p & "表A" & "#" & r("_Identify")
For Each bd As QQBuddy In QQClient.Buddies
    If bd.Online Then
        QQClient.Send(bd.name, msg)
    End If
Next
```

这段代码用于合成通知信息发送给好友，例如"U#表A#8"表A主键为8的行发生了修改，"A#表A#10"表示表A新增了1行，新增行的主键为10。
你可以任意定义这种格式的通知。

2、再增加一个删除按钮，按钮代码设置为：

```
Dim r As Row = Tables("表A").Current
Dim msg As String = "D#" & "表A" & "#" & r("_Identify")
r.Delete()
Tables("表A").Save()
For Each bd As QQBuddy In QQClient.Buddies
    If bd.Online Then
        QQClient.Send(bd.name, msg)
    End If
Next
```

这段代码用于删除行后合成通知信息发送给好友，例如"D#表A#8"表A主键为8的行被删除了。

3、在 QQClient 的 ReceivedMessage 事件中修改代码：

```
Dim msg As String = e.Message
If msg.StartsWith("U#") Then '表示修改了某行
    Dim pts() As String = msg.Split("#")
    If pts.Length = 3 Then
        Dim dr As DataRow = DataTables(pts(1)).Find("[_Identify] = " & pts(2))
        If dr IsNot Nothing  Then
            dr.Load() '重新加载此行
        End If
    End If
ElseIf msg.StartsWith("A#") Then '表示增加了行
    Dim pts() As String = msg.Split("#")
    If pts.Length = 3 Then
```

```
        DataTables(pts(1)).AppendLoad("[_Identify] = " &  pts(2)) '追载新增加的行.
    End If
ElseIf msg.StartsWith("D#") Then '表示删除了行
    Dim pts() As String = msg.Split("#")
    If pts.Length = 3 Then
        DataTables(pts(1)).RemoveFor("[_Identify] = "  & pts(2)) '移除行
    End If
End If
```

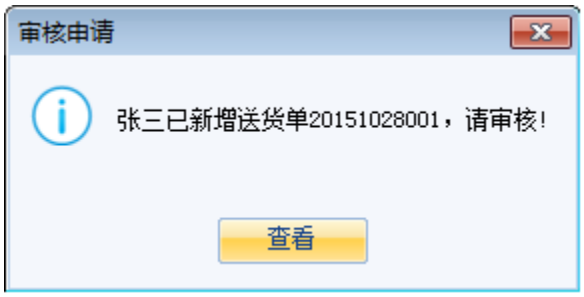
这样，多用户环境下，任何一个客户端做的修改、增加，删除，保存后都会立即体现在其他客户端上。

4.16.2.4.3 自定义信息提示窗口

多数时候，我们会用 PopMessage 进行信息提示。
但是 PopMessage 没有提供按钮，在一些需要交互操作的场合是无法满足要求的。

不过 Foxtable 2016 新增了一个窗口类型 PopupForm，PopupForm 同样以弹窗形式，在屏幕右下角显示。
我们可以设计一个 PopupForm 类型的窗口，代替 PopMessage 进行消息提示，这样就能满足上述要求了。
本节内容需要实现下述任务：

送货单由用户张三开单，由李四审核。
当张三开送货单保存时，系统自动发给李四发一个消息：“张三已新增送货单 20151028001，请审核”。
李四收到此消息后，点此消息，打开这张送货单，进行审核，审核后，反馈回张三消息：“送货单 20151028001 已由用户李四审”
下图就是我们要设计的提示窗口：



李四收到信息之后，单击提示窗口的查看按钮，即可加载(或刷新)并定位到此送货单，并自动打开送货单编辑窗口，查看该送货单信息并审核。
本节内容使用 OpenQQ 发送和接收信息，关于 OpenQQ 服务端和客户端的设计，前面的章节已经有详细介绍，这里就不重复。
以下的设计全部在客户端项目完成。
设计步骤
1、首先我们要定义一个集合，用于存储收到的审核申请，在全局代码中定义一个 Public 变量：

```
Public fhs As New List(of String)
```

因为存在一种可能，就是张三连续开了多个送货单，李四因故没有即时审核，所以我们应该将收到的审核申请存储在集合 fhs 中，逐个提醒李四。

2、现在我们要约定一个消息格式，审核申请的格式为"+@|主键|送货单号|制单人"，审核完成的格式为"-@|送货单号|审核人"。

3、在发货单编辑窗口的保存按钮中，写入如下代码：

```
Dim r As Row  = Tables("送货单").Current
r.Save()
QQClient.Send("李四","+@|" & r("_Identify") & "|" & r("单号") & "|" & User.Name)
```

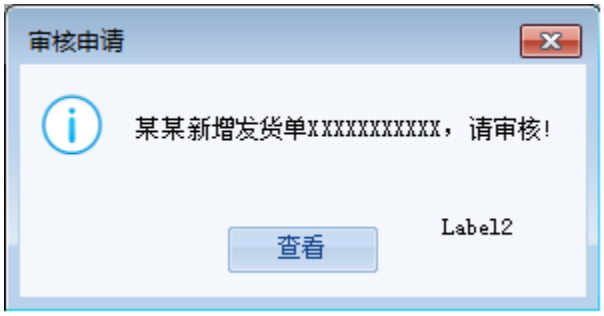
4、在发货单编辑窗口的审核按钮中，写入如下代码：

```
Dim r As Row  = Tables("送货单").Current
r("审核") = True
r.Save()
'发送审核通过的消息
QQClient.Send("张三","-@|" & r("单  号") & "|" & User.Name)
```

5、在 OpenQQ 客户端的 MessageReceived 事件中编写代码：

```
Dim pts() As String  = e.Message.Split("|")
If pts.Length = 4 AndAlso pts(0) = "+@" '如果是申请审核消息
    fhs.Add(pts(1) & "|" & pts(2) & "|" & pts(3)) '将主健、单号和制单人  合成为一个字符串存储在集合中
ElseIf pts.Length = 3 AndAlso pts(0) = "-@" '如果是审核通过消息
    PopMessage("送货单" & pts(1) & "已由" & pts(2) & "审核通过!")
    Dim dr As DataRow = DataTables("送货单").Find("单号  = '" & pts(1) & "'")
    If dr IsNot Nothing Then
        dr.Load
    End If
End If
```

6、在客户端项目新建一个下图所示的窗口，窗口名为“审核提示”，窗口类型设置为 PopupForm：



这个窗口有两个 Label 控件，Label1 用于显示提示信息，Label2 的可见属性设置为 False，用于存储发货单的主键(Identify)。

这是一个技巧，因为 Foxtable 并没有提供窗口级别的变量，所以通常用一些不可见的 Label 控件来存储临时信息。
将查看按钮的代码设置为：

```
DataTables("送货单").Save() '接下来要根据主键查找，必须先保存，因为只有保存后新增行的主键才是有效值。
Dim id As String = e.Form.Controls("Label2").Text
Dim dr As DataRow = DataTables("送货单").Find("[_Identify] = " & id)
If dr Is Nothing Then
    DataTables("送货单").AppendLoad("[_Identify] = " & id)
    dr =  DataTables("送货单").Find("[_Identify] = " & id)
Else
    dr.Load()
End If
If dr IsNot Nothing Then
    Dim wz As Integer = Tables("送货单").FindRow(dr)
    If wz < 0 Then '如果因为筛选导致此送货单不可见
        Tables("送货单").Filter = ""
        wz =Tables("送货单").FindRow(dr)
    End If
    If wz >= 0 Then
        Tables("送货单").Position = wz
        Forms("发货单").Open()
    End If
End If
e.Form.Close()
```

上面的代码不难理解，需要注意的是，为什么必须首先保存一下送货单表呢？
实际上如果李四只有审核送货单的权限，没有增加送货单的权限，那么第一行保存表的代码是可以省略的，这又是为什么呢？
假定送货单表原来有10行(主键值0-9)，张三和李四都可以增加送货单，两人分别增加1个订货单，主键值都是10。
张三保存后，李四收到通知，如果李四不保存，直接用AppendLoad追载主键为10的行，由于表中已经有一个主键为10的行，所以会出现错误提示。
如果李四先保存，其新增行的主键会在保存后变为11，此时就可以正常根据通知信息追载主键为10的行了。

7、在项目事件SystemIdle中加入代码：

```
If fhs.Count > 0 AndAlso Forms("审核申请").Opened = False Then
    Dim pts() As String = fhs(0).Split("|")
    fhs.RemoveAt(0)
    With Forms("审核申请")
        .Open()
        .Controls("Label1").Text = pts(2) & "新增订货单" & pts(1) & ",请审核!"
```

```
        .Controls("Label2").Text = pts(0) '将发货单主健设置为Label2的标题
    End With
End If
```

上述代码在系统空闲的时间检查fhs集合，判断是否有新的审核申请；如果有新的申请且审核申请提示窗口未显示，则显示集合中的第一个审核申请，并将其从集合中移除。
上述代码将待审核送货单的主键设置为Label2的标题，这样在审核申请窗口单击查看按钮时，可以根据Label2的标题找出待审核的订单。

4.16.2.4.4 独占式编辑

本节讲述如何避免多人同时编辑同一行，既如何实现独占式编辑。
利用 OpenQQ，可以很轻松实现此功能。
要实现此功能，服务器端和客户端都要进行相应的设计，我们学习的时候，也要结合客户端和服务端的代码来理解。
服务器端的设计
1、在服务器端项目的全局代码中，加入如下代码：

```
Public tbrk As new Dictionary(of String,String)
```

tbrk 是一个字典，用于登记每一行是谁在编辑。
我们约定接下来编码的时候：字典的键由表名和行的主键组合成，值则等于编辑者名。

2、在服务端项目的 OpenQQ 服务端的 ReceivedMessage 事件加上代码：

```
Dim msg As String = e.Message
If msg.StartsWith("?#") AndAlso msg.EndsWith("#?") Then '收到请求编辑信号
    Dim Key As String = msg.SubString(2,msg.Length - 4)
    If tbrk.Containskey(Key) = False Then '如果无人编辑此行
        tbrk.Add(Key,e.UserName) '登记申请者为此行的编辑者
        e.ReturnValue = "OK" '通知申请者可以编辑
    ElseIf tbrk(Key) = e.UserName Then '如果申请者就是之前登记的编辑者
        e.ReturnValue = "OK" '通知申请者可以编辑
    Else '如果之前登记的编辑者为其他人
        e.ReturnValue = tbrk(Key) & "正在编辑此行!" '告知申请者是谁在编辑此行
    End If
ElseIf msg.StartsWith("!#") AndAlso msg.EndsWith("#!") Then '收到结束编辑信号
    Dim Key As String = msg.SubString(2,msg.Length - 4)
    If tbrk.Containskey(Key)  Then
        tbrk.Remove(Key) '从集合中移除此行的编辑登记
    End If
End If
```

上述代码的注释详尽明了，就不做解释了。

3、在服务端项目的OpenQQ服务端的UserLogout事件加上代码：

```
Dim Keys As New List(of String)
For Each Key As String In tbrk.Keys
   If tbrk(Key) = e.UserName Then
     Keys.Add(Key)
    End If
Next
For Each Key As String In Keys
    tbrk.Remove(Key)
Next
```

这样当有用户退出登录时，不管他是正常退出还是异常退出，都可以将字典中该用户的编辑登记移除，避免死锁。
客户端用户异常退出时，服务端的编辑登记的移除会有一个延时，时长取决于QQServer的HeartbeatTimeout(心跳超时)属性。
上述编码有一个地方需要注意，我们不能在遍历字典或集合的过程中移除其成员(运行时会报错)，所以我们用一个临时的集合Keys，先将要移除的键值保存在这个集合中，最后遍历这个集合中的键值，从字典tbrk中移除这些键值。

客户端的设计

1、在客户端项目的全局代码中，加入如下代码：

```
Public tbrk As new List(of String)
```

tbrk 是一个集合，用于记录当前用户正在编辑的行，接下来编码的时候，我们用表名加上行的主键值来表示某行。

2、在客户端的对应表的StartEdit事件中编写代码：

```
Dim r As Row = e.Table.Current
If r.DataRow.RowState = DataRowState.Added Then '新增行正常编辑
    Return
End If
If QQClient.Ready = False Then '如果QQClient没有启动,则禁止编辑
    PopMessage("必须启动QQClient,才能编辑此表数据!","提示",PopIconEnum.Infomation,5)
    e.Cancel = True
    Return
End If
Dim key As String = e.Table.DataTable.Name   & ":" & r("_Identify")
If tbrk.Contains(key) Then '如果  本人之前已经编辑此行,则正常编辑
    Return
Else '如果  本人之前没有编辑此行
    Dim msg = QQClient.SendWait("?#" & Key & "#?",5) '向服务器发送请求编辑信息
```

```
    If msg = "OK" Then '如果服务器返回OK
        tbrk.Add(key) '在本地登记正在编辑此行
    ElseIf msg > "" Then '否则显示服务器返回的信息,并取消编辑
        PopMessage("无法编辑此行,因为:" & vbcrlf & msg ,"提示",PopIconEnum.Infomation,5)
        e.Cancel = True
    Else '如果服务器没有返回信息,则取消编辑
        PopMessage("因服务器无响应,无法编辑此行!" ,"提示",PopIconEnum.Infomation,5)
        e.Cancel = True
    End If
End If
```

3、在客户端的对应表的BeforeSaveDataRow事件中编写代码：

```
If e.DataRow.RowState=DataRowState.Modified  Then
    Dim Key As String = e.DataTable.Name & ":" & e.DataRow("_Identify")
    If tbrk.Contains(Key) Then
        tbrk.Remove(Key) '移除本地编辑登记
        QQClient.Send("!#" & Key & "#!") '通知服务器此行已经结束 编辑
    End If
End If
```

这样客户端在保存某行前，会从集合tbrk移除此行的在本地的编辑登记，并给服务端发送一个信息，通知服务端我已经结束编辑此行了，其他人可以开始编辑了。

代码似乎编写完毕，但是有一个漏洞：
假定你双击某个单元格进入编辑状态，服务器已经登记你为此行的编辑者，如果你不做任何修改退出编辑状态，那么之后保存的时候，此行并不会触发BeforeSaveDataRow事件，以至于服务器始终认为你还在编辑此行。
所以接下来还需要完善。

4、在客户端对应表的AfterEdit事件中编写代码：

```
If e.Row.DataRow.RowState =DataRowState.Unchanged Then
    Dim Key As String = e.Table.DataTable.Name & ":" & e.Row("_Identify")
    If tbrk.Contains(Key) Then
        tbrk.Remove(Key)
        QQClient.Send("!#" & Key & "#!")
    End If
End If
```

在用户结束编辑某单元格的时候，上述代码判断当前行是否已经修改过，如果没有修改，则移除此行在本地的编辑标记，并通知服务器此行的编辑已经结束，其他人可以编辑此行了。

至此，我们的独占式编辑功能已经完成。

各种可能的异常

我们看看，如果网络不稳定或客户端异常中断，会发生什么：

异常	后果	解决方法
服务器没有收到客户端发出的编辑请求	本次申请失败，对后续操作没有影响，任何用户都可以继续尝试编辑此行。	
客户端没有收到服务器发出的允许编辑信号	本次申请失败，但对本人的后续操作没有影响，不过其他用户对此行的编辑申请将被拒绝。	1、本人可以继续尝试编辑此行，编辑后保存，其他用户即可正常申请编辑此行了。 2、如果本人关闭项目或退出 QQClient，那么其他用户也可以正常申请编辑此行了。 3、如果本人没有进行上述操作，但是其他人需要编辑此行，由于其他人在尝试编辑的时候，会知道是谁正在编辑此行，他可以通知你采取措施解除锁定。
服务端没有收到客户端发出的结束编辑信号	对本人的后续操作没有影响，不过其他用户对此行的编辑申请将被拒绝。	同上
客户端在编辑过程中异常退出	其他用户对此行的编辑申请将被拒绝。	1、本人可以重新打开项目，继续编辑此行，编辑后保存，其他用户即可正常申请编辑此行了。 2、本人可以重新打开项目，然后直接关闭，其他用户也可以正常申请编辑此行了。 3、即使本人异常退出后，不再打开项目，一段时间后，服务器也会自动清除此行的编辑标记，其他用户可以继续申

		请编辑此行，等待的时间由 QQServer 的 HeartbeatTimeout(心跳超时)属性决定。

可以看到，即使偶尔出现网络不稳定，即使异常退出，本方案也不会带来死锁这样难以接受的后果。

如果是窗口编辑

上述方案针对的是直接在表中编辑数据，如果不是在表中编辑，而是在窗口编辑，编码方式也是基本相同的。

通常只需将打开编辑窗口的代码改为：

```
Dim r As Row = Tables("表B").Current
If QQClient.Ready = False Then '如果QQClient没有启动,则禁止编辑
    PopMessage("必须启动QQClient,才能编辑此表数据!","提示",PopIconEnum.Infomation,5)
    Return
End If
If r.DataRow.RowState <>  DataRowState.Added Then '如果不是新增行
    Dim key As String = "表B"  & ":" & r("_Identify")
    If tbrk.Contains(key) = False Then
      Dim msg = QQClient.SendWait("?#" & Key & "#?",5) '向服务器发送请求编辑信息
      If msg = "OK" Then '如果服务器返回OK
        tbrk.Add(key) '在本地登记正在编辑此行
      ElseIf msg > "" Then '否则显示服务器返回的信息,并取消编辑
        PopMessage("无法编辑此行,因为:" & vbcrlf & msg ,"提示",PopIconEnum.Infomation,5)
        Return
      Else '如果服务器没有返回信息,则取消编辑
        PopMessage("因服务器无响应,无法编辑此行!" ,"提示",PopIconEnum.Infomation,5)
        Return
      End If
    End If
End If
Forms("编辑窗口").Open()
```

4.16.2.4.5 网络环境下的复杂编号

我们之前介绍的编号方法，在多用户环境下会出现重复，参考：常用编号生成方法

有了 Open，我们可以轻松解决这个问题。

以下例子，在实际测试的时候，如果表中已经有不符合规范的编号，请先删除之，否则会影响编号的生成结果

一、按类别编号

假定有下图所示的一个表，编号根据类别生成，前两位为类别，后三位为顺序号：

	类别	编号
1	HA	HA001
2	HD	HD001
3	FX	FX001
4	FX	FX002
5	HA	HA002
6	HA	HA003
7	FX	FX003
8	HA	HA004

这里假定使用的是外部数据源。

服务端的设计

1、在服务端的全局代码中编写代码：

```
Public flbhs As New Dictionary(of String,Integer)
```

2、在服务端的 AfterOpenProjet 事件中编写代码：

```
Dim dt As DataTable
Dim cmd As New SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select  类别, Max(编号) As  编号  From {产品} Group By  类别"
dt = cmd.ExecuteReader
flbhs.Clear()
For Each dr As DataRow In dt.DataRows
    Dim qz As String = dr("类别") '编号前缀
    Dim bh As String =   dr("编号")
    Dim id As Integer
    If bh.Length = 5 Then
        bh = bh.SubString(2)
        If Integer.TryParse(bh,id) Then
            flbhs.Add(qz, id)
        End If
    End If
Next
```

上述代码在启动项目后，用 SQL 语句提取每个类别的最大编号，将其整数部分存储在字典 flbhs 中。

3、在服务端的 OpenQQ 服务端事件 ReceivedMessage 中编写代码：

```
Dim msg As String = e.Message
If msg.StartsWith(":f") AndAlso msg.EndsWith("f:") Then
    msg = msg.SubString(2, msg.Length - 4)
    If flbhs.ContainsKey(msg) Then '如果存在这个类别的编号
        flbhs(msg) = flbhs(msg) + 1 '将该类别最大编号加1
    Else
```

```
        flbhs.Add(msg,1) '如果是这个列表的首次编号,则编号等于1
    End If
    e.ReturnValue = flbhs(msg) '将编号返回给客户端
End If
```

客户端的设计

选择客户端项目对应的表，在其DataColChanged事件中加上代码：

```
If e.DataCol.Name = "类别" Then
    If e.DataRow.IsNull("类别") Then
        e.DataRow("编号") = Nothing
    Else
        If QQClient.Ready = False Then
            PopMessage("QQClient未启动,无法生成编号!","提示",PopIconEnum.Infomation,5)
        Else
            Dim bh As String = e.DataRow("类别")
            Dim rt As String =  QQClient.SendWait(":f" & bh & "f:")
            Dim id As Integer
            If rt > "" Then
                If  Integer.TryParse(rt,id)  Then
                    e.DataRow("编号") = bh & Format(id,"000")
                Else
                    PopMessage("服务器返回错误信息:" & rt,"提示",PopIconEnum.Infomation,5)
                End If
            Else
                PopMessage("服务器无响应,无法生成编号!","提示",PopIconEnum.Infomation,5)
            End If
        End If
    End If
End If
```

二、按月生成编号

假定有个订单表，需要按月自动生成编号，四位年，两位月，最后三位是顺序号，如下图所示：

	日期	编号
1	2011-05-02	201105-001
2	2011-05-02	201105-002
3	2011-05-20	201105-003
4	2011-06-16	201106-001
5	2011-06-25	201106-002
6	2011-06-29	201106-003

这里假定使用的是外部数据源。

服务端的设计

1、在服务端的全局代码中编写代码：

```
Public flbhs As new Dictionary(of String,Integer)
```

2、在服务端的 AfterOpenProjet 事件中编写代码：

```
Dim dt As DataTable
Dim cmd As New SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select Year(日期) As 年, Month(日期) As 月, Max(编号) as 编号 From {订单} Group By Year(日期), Month(日期)"
dt = cmd.ExecuteReader
flbhs.Clear()
For Each dr As DataRow In dt.DataRows
    Dim qz As String = dr("年") & Format(dr("月"),"00") '编号前缀,4位年,2位月
    Dim bh As String = dr("编号")
    Dim id As Integer
    If bh.Length = 10 Then
        bh = bh.SubString(7)
        If Integer.TryParse(bh,id) Then
            flbhs.Add(qz, id)
        End If
    End If
Next
```

上述代码在启动项目后，用 SQL 语句提取现有数据的每月最大编号，将其整数部分存储在字典 flbhs 中。

3、在服务端的 OpenQQ 服务端事件 ReceivedMessage 中编写代码：

```
Dim msg As String = e.Message
If msg.StartsWith(":p") AndAlso msg.EndsWith("p:") Then
    msg = msg.SubString(2, msg.Length - 4)
    If flbhs.ContainsKey(msg) Then '如果存在这个月的编号
        flbhs(msg) = flbhs(msg) + 1 '将该月最大编号加1
    Else
        flbhs.Add(msg,1) '如果是这个月的首次编号,则编号等于1
    End If
    e.ReturnValue = flbhs(msg) '将编号返回给客户端
End If
```

客户端的设计

选择客户端项目的订单表，在其DataColChanged事件中加上代码：

```
If e.DataCol.Name = "日期" Then
    If e.DataRow.IsNull("日期") Then
        e.DataRow("编号") = Nothing
```

```
    Else
        If QQClient.Ready = False Then
            PopMessage("QQClient未启动,无法生成编号!","提示",PopIconEnum.Infomation,5)
        Else
            Dim bh As String = Format(e.DataRow("日期"),"yyyyMM")
            Dim rt As String =  QQClient.SendWait(":p" & bh & "p:")
            Dim id As Integer
            If rt > "" Then
                If  Integer.TryParse(rt,id)  Then
                    e.DataRow("编号") = bh & "-" & Format(id,"000")
                Else
                    PopMessage("服务器返回错误信息:" & rt,"提示",PopIconEnum.Infomation,5)
                End If
            Else
                PopMessage("服务器无响应,无法生成编号!","提示",PopIconEnum.Infomation,5)
            End If
        End If
    End If
End If
```

三、按日期和类别编号

假定有个表，需要按月自动生成编号，根据工程代码按顺序编号，前 4 位是工程代码，然后是 4 位年，2 位月，最后 4 位是顺序号，如下图所示：

	工程代码	制单日期	单据编号
1	NGAY	2012-03-02	NGAY-201203-0001
2	NGAY	2012-03-03	NGAY-201203-0002
3	NGAY	2012-03-04	NGAY-201203-0003
4	NGAY	2012-03-05	NGAY-201203-0004
5	NGAY	2012-03-06	NGAY-201203-0005
6	NGAY	2012-03-07	NGAY-201203-0006
7	NGAY	2012-03-08	NGAY-201203-0007
8	NGAY	2012-03-09	NGAY-201203-0008
9	NGAY	2012-03-01	NGAY-201203-0009
10	QYGC	2012-03-02	QYGC-201203-0001
11	QYGC	2012-03-03	QYGC-201203-0002
12	QYGC	2012-03-04	QYGC-201203-0003
13	QYGC	2012-03-05	QYGC-201203-0004
14	QYGC	2012-03-06	QYGC-201203-0005
15	QYGC	2012-04-14	QYGC-201204-0001
16	QYGC	2012-04-14	QYGC-201204-0002
17	QYGC	2012-04-14	QYGC-201204-0003

其实如果你看懂了前面的例子，这个例子就是多余的，无非就是在服务器端记录每一个前缀的最大序号，遇到客户端申请编号的时候，将最大序号加 1，然后返回给客户端。

编程的是千变万化的，但万变不离其宗，基本原理明白了，就可以举一反三，以不变应万变。

接下来我们看看是不是和第一个例子一样。

服务端的设计

1、在服务端的全局代码中编写代码：

```
Public flbhs As new Dictionary(of String,Integer)
```

2、在服务端的 AfterOpenProjet 事件中编写代码：

```
Dim dt As DataTable
Dim cmd As New SQLCommand
cmd.ConnectionName = "数据源名称"
cmd.CommandText = "Select 工程代码,Year(制单日期) As 年, Month(制单日期) As 月, Max(单据编号) as 单据编号 From {工程} Group By 工程代码,Year(制单日期), Month(制单日期)"
dt = cmd.ExecuteReader
flbhs.Clear()
For Each dr As DataRow In dt.DataRows
    Dim qz As String = dr("工程代码") & "-" & dr("年") & Format(dr("月"),"00") '编号前缀,4位工程代码,4位年,2位月
    Dim bh As String = dr("单据编号")
    Dim id As Integer
    If bh.Length = 16 Then
        bh = bh.SubString(12)
        If Integer.TryParse(bh,id) Then
            flbhs.Add(qz, id)
        End If
    End If
Next
```

上述代码除了表名、列名的变化，以及给编号前缀加上工程代码外，真正发生变化的就是两个数字，你可以想想为什么一个由10改为16，一个由7改为12?

3、在服务端的 OpenQQ 服务端事件 ReceivedMessage 中编写代码：

```
Dim msg As String = e.Message
If msg.StartsWith(":g") AndAlso msg.EndsWith("g:") Then
    msg = msg.SubString(2, msg.Length - 4)
    If flbhs.ContainsKey(msg) Then '如果存在这个月的编号
        flbhs(msg) = flbhs(msg) + 1 '将该月最大编号加1
    Else
        flbhs.Add(msg,1) '如果这个月的首次编号,则编号等于1
    End If
    e.ReturnValue = flbhs(msg)
End If
```

上述代码没有任何改变。

客户端的设计

选择客户端项目的对应的表，在其DataColChanged事件中加上代码：

```
Select e.DataCol.Name
    Case "制单日期","工程代码"
      If e.DataRow.IsNull("制单日期") OrElse e.DataRow.IsNull("工程代码") Then
        e.DataRow("单据编号") = Nothing
      Else
        If QQClient.Ready = False Then
          PopMessage("QQClient未启动,无法生成编号!","提示",PopIconEnum.Infomation,5)
        Else
          Dim bh As String = e.DataRow("工程代码") & "-" & Format(e.DataRow("制单日期"),"yyyyMM")
          Dim rt As String = QQClient.SendWait(":g" & bh & "g:")
          Dim id As Integer
          If rt > "" Then
            If Integer.TryParse(rt,id) Then
              e.DataRow("单据编号") = bh & "-" & Format(id,"0000")
            Else
              PopMessage("服务器返回错误信息:" & rt,"提示",PopIconEnum.Infomation,5)
            End If
          Else
            PopMessage("服务器无响应,无法生成编号!","提示",PopIconEnum.Infomation,5)
          End If
        End If
      End If
End Select
```

上述代码除了表名、列名的变化，以及给编号前缀加上工程代码外，唯一发生变化的只是将：

```
Format(id,"000")
```

改成了

```
Format(id,"0000")
```

因为顺序号原来是3位，现在是4位。

4.16.3 UDPClient

4.16.3.1 关于 UDPClient

内置的 OpenQQ，让服务器和客户端，以及客户端之间的通讯变得很方便。
但是 OpenQQ 只能在 Foxtable 中使用，如果其他程序需要和 Foxtable 通讯，是无法通过 OpenQQ

来实现的。

要解决其他程序(包括微信和网页)和 Foxtable 之间的通讯问题，可以使用 UDPClient。
提示： 此功能仅高级开发版提供。

4.16.3.2 类型参考

4.16.3.2.1 属性

UDPClient 的属性有：

属性	说明
IP	IP 地址
Port	端口
Ready	逻辑型，用于判断 UDPClient 是否已经启动。
EncodingName	字符型，用于指定 UDPClient 使用的编码方式。

内网使用的时候，IP 属性设置为内网 IP，外网使用的时候，IP 属性设置为外网 IP.
如果同时要和内网和外网通讯，可以不设置 IP，或者将 IP 地址设置为“0.0.0.0”，这样不管对方是外网还是内网，都是可以访问的。
如果是本机不同程序之间的通讯，IP 可以使用"127.0.0.1"。
UDPClient 在收发字符串的时候，默认使用 Undicode(utf-16)编码方式，可以通过 EncodingName 设置为别的编码，常用的编码有：

gb2312
big5
GB18030
hz-gb-2312
utf-8
utf-16

4.16.3.2.2 方法

4.16.3.2.2.1 Start

用于启动 UDPClient。
示例：

```
UDPClients.Add("UDP1")
UDPClients("UDP1").IP = "127.0.0.1"
UDPClients("UDP1").Port = 52178
UDPClients("UDP1").Start()
```

可以看到同一个项目，可以启动多个 UDPClient。

4.16.3.2.2.2 Stop

停止 UDPClient。

4.16.3.2.2.3 Send

用于发送信息。
语法:
Send(IP, Port, Message)
Send(IP, Port, Bytes)
IP:　　对方 IP 地址
Port:　对方端口
Message：要发送的信息
bytes:　要发送的字节数组。

4.16.3.2.3 事件

4.16.3.2.3.1 ReceivedMessage

UDPClient 收到信息后触发。
e 参数属性：
Bytes：　　字节数组，以字节数组形式返回收到的信息。
Message：　字符型，以字符形式返回收到的信息，编码格式由 EncodingName 属性决定。
ReturnBytes：字节数组，指定要返回给发送者的字节数组。
ReturnValue：字符型，指定要返回给发送者的字符信息。
IP：　　字符型，返回对方 IP。
Port：　　整数型，返回对方端口。
IPEndpoint：　IPEndpoint 型，返回对方的 IPEndpoint

说明：

不管是 Bytes 属性，还是 Message 属性，都可以返回收到的信息，只是前者以字节数组形式返回，后者以字符串形式返回，编码格式由 EncodingName 属性决定。
发送端和接收端的编码格式必须一致，接收端才能正确还原字符串，否则会出现乱码。
如果希望给发送者返回信息，可以设置 ReturnValue 属性或 ReturnBytes 属性，前者指定返回的字符信息，后者指定返回的字节数组。
当两个同时指定时，只有 ReturnValue 会生效。

4.16.3.3 和其他程序的通讯

有了 UDPClient，就可以和其他程序通讯，包括微信、网页、手机 APP 等等。
其他程序不一定要位于本机，可以和网络上任何终端进行通讯。
如果其他程序位于本机，可以使用 IP 地址"127.0.0.1",否则请使用正确的 IP 地址。
例如你有个微信公众平台或网站，希望能和 Foxtable 对接起来，充分利用 Foxtable 在数据管理和统计分析方面的优势。
你可以在微信公众平台或网站所在的电脑运行一个 Foxtable 项目。
1、在项目的 AfterOpenProject 事件中设置代码：

```
UDPClients.Add("UDP1")
UDPClients("UDP1").IP = "127.0.0.1"
UDPClients("UDP1").Port = 52178
UDPClients("UDP1").Start()
```

2、在菜单的“管理项目”功能区，单击“网络监视器”，设置 UDPClient 的 ReceivedMessage 事件代码：

```
Dim msg As String = e.Message
Dim ret As String
If e.Message.StartsWith("#") AndAlso e.Message.EndsWith("#")  '假定收到的信息头尾都是"#",表示统计指定产品的销量
    msg   = msg.Trim("#")
    msg = msg.Replace("#","','")
    Dim b As New GroupTableBuilder("统计表1",DataTables("订单"))
    b.Groups.AddDef("产品")
    b.Totals.AddDef("数量")
    b.Filter = "产品  in ('" & msg & "')"
    Dim dt As DataTable   = b.Build(True)
    For Each dr As DataRow In dt.DataRows
      If ret > ""then
        ret = ret & vbcrlf
      End If
      ret = ret & dr("产品") & vbtab & dr("数量")
    Next
    e.ReturnValue = ret
End If
```

Foxtable 收到其他程序发来"#产品 1#产品 2#产品 3#"这样格式的信息后，会自动统计出各产品的数量，并返回之。
你可以自己定义各种格式的字符串，用于完成不同的任务。

3、基本上任何编程工具，都可以使用 UDP 进行数据收发的。假定你使用的是 vb.net 开发其他程序，你可以使用下面的代码和本机的 foxtable 通讯：

```
Dim udp As New System.Net.Sockets.UdpClient(0)
Dim ip As New System.Net.IPEndPoint(System.Net.IPAddress.Parse("127.0.0.1"), 52178)
Dim bytes() As Byte = System.Text.Encoding.Unicode.GetBytes("#PD01#PD02#PD03#")
udp.Send(bytes, bytes.Length, ip)
Dim sdt As Date = Date.Now
While Date.Now < sdt.AddSeconds(5) '等到5秒
    If udp.Available > 0 Then
        Dim remoteIp As New System.Net.IPEndPoint(System.Net.IPAddress.Any, 0)
        Dim rbytes() As Byte = udp.Receive(remoteIp)
        Dim rev As String = System.Text.Encoding.Unicode.GetString(rbytes)
        MessageBox.Show(rev)
    End If
End While
udp.Close
```

你也可以设计一个网页或APP，让用户通过网页或APP填报数据，收集之后通过UDP将数据发送给Foxtable，再将Foxtable返回的数据展示在网页和APP上。

4.16.3.4 与 ASPNET 网页进行交互示例

官方网站为本节内容提供了示例，下载地址：
http://www.foxtable.com/samples/UDPWeb.rar

本节的任务是设计一个简单的订单查看网页，并可以修改和删除订单明细产品。网站的数据不是直接操作数据库获取的，而是通过和 Foxtable 的 UDPClient 服务端进行通讯取得。

要实现网页和 Foxtable 的通讯，首先我们的先定义好通讯的格式。（虽然我们是可以把任何对象比如.Net 的 Datatable 序列化后进行 UDP 的传输，但不是本节讨论的话题，自定义传输格式的控制性更好，也减少了许多不必要的冗余信息。）

本示例定义传输的字符串的格式如下：
（1）发送和返回的信息均以“%”符号开始和结束
（2）查询的字符串以大写字母“G”开头
（3）更新数据以大写字母“U”开头
（4）删除数据以大写字母“D”开头
（5）查询的返回值里主表和明细表的内容以“$”符号分割
（6）明细表的多行内容以“@”符号分割
（7）表内容多个字段名称和值以“|”符号分割

基本的原理就是网站服务器的 ASP.NET 程序收到客户通过网页提交的数据或命令后，通过 UDP 将其转给在同一台服务器的 Foxtable 程序，Foxtable 处理完毕后将结果传给 ASP.NET 程序，ASP.NET 再将结果呈现在客户的网页上。
不管是网页、微信、手机 APP，还是任何第三方的软件，都可以通过这个方法和 foxtable 进行数

据交互。

提示：本机程序使用 UDP 通讯的时候，必须使用 IP“127.0.0.1”

1. 设计 ASPNET 网页

首先打开 VS，创建一个 ASP.NET 网站的项目，默认就会生成一个起始页面 Default.aspx。我们就在 Default.aspx 设计好页面的展示，如下图：



在订单编号框中输入订单编号，点击“查询”按钮，通过下面的代码获取订单内容：

```
result = UDPCommunication("%G|" & orderNum & "%")
```

然后进行内容解析：

```
If result IsNot Nothing Then
        If result.StartsWith("%") AndAlso result.EndsWith("%") Then
                Session("OrderNum") = orderNum '记录订单编号
                Dim orders() As String = result.Trim("%").Split("$") '
分离主表和明细
                Dim orderdetails() As String = orders(0).Split("|")
                txtName.Value = orderdetails(0)
                txtOrderDate.Value = orderdetails(1)
                txtDeliverDate.Value = orderdetails(2)
                If orders.Length = 2 Then
                        If orders(1).StartsWith("@") AndAlso
orders(1).EndsWith("@") Then
                                Dim detailArry() As String =
orders(1).Trim("@").Split("@") '分离不同明细

                                For Each de As String In detailArry
                                        Dim details() As String =
de.Split("|")

                                        Dim dr As DataRow
                                        dr = order.NewRow()
                                        For i As Integer = 0 To
details.Length - 1

                                                dr(i) = details(i) '定
义的字段必须和返回值的字段顺序一致

                                        Next
```

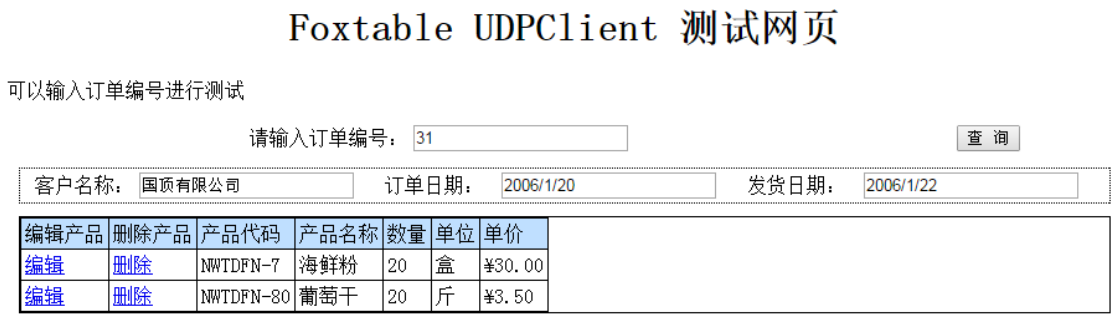
```
                                        order.Rows.Add(dr)
                                    Next
                                End If
                            End If
                        End If
                End If
```

这里定义一个函数来和 Foxtable 的 UDPClient 进行通讯，如

```
Function UDPCommunication(ByVal input As String) As String
        Dim udp As New System.Net.Sockets.UdpClient(0)
        Dim ip As New
System.Net.IPEndPoint(System.Net.IPAddress.Parse("127.0.0.1"), 52177) '定义通讯的网络端点，地址和端口号要和和UDPClient启动时的一致

        Dim bytes() As Byte = System.Text.Encoding.Unicode.GetBytes(input)
        udp.Send(bytes, bytes.Length, ip) '发送数据
        Dim result As String = "!false"
        Dim sdt As Date = Date.Now
        While Date.Now < sdt.AddSeconds(1) '等1秒
                If udp.Available > 0 Then
                        Dim remoteIp As New
System.Net.IPEndPoint(System.Net.IPAddress.Any, 0)
                        Dim rbytes() As Byte = udp.Receive(remoteIp)
                        result = System.Text.Encoding.Unicode.GetString(rbytes)
'获取返回值
                End If
        End While
        udp.Close()
        Return result
    End Function
```

比如输入订单编号 31，查询完成后的页面如下：



在明细中点击删除，就可以删除一条明细，代码如下

```
                Dim numCell As TableCell = e.Item.Cells(2)
                Dim num As String = numCell.Text
               Dim result As String = UDPCommunication("%D|订单编号|" &
Session("OrderNum").ToString() & "|产品代码|" & num & "%")
               If result = "!false" Then
                        p1.InnerText = "删除产品" & e.Item.Cells(3).Text & "出错！"
                Else
                        orderView.RowFilter = "产品代码='" & num + "'"
                        If orderView.Count > 0 Then
                                orderView.Delete(0)
                        End If
                End If
                orderView.RowFilter = ""
                BindGrid()
```

同样，编辑的情况也类似，具体参考示例的 VS 项目。

2. Foxtable 的 UDPClient 服务器端设计

2.1 设计一个下图所示的窗口：



窗口中启东按钮的代码为：

```
If UDPClients.Count = 0 Then
       UDPClients.Add("UDP1",new UDPClient("UDP1") )
End If
UDPClients("UDP1").IP = e.Form.Controls("txtIP").Value
UDPClients("UDP1").Port = e.Form.Controls("txtPort").Value
UDPClients("UDP1").Start()
```

2.2 在菜单的“管理项目”功能区，单击“网络监视器”，设置 UDPClient 的 ReceivedMessage 事件代码：

```
Dim drnew As Row = Tables("通讯记录").AddNew()
```

```
drnew("接收时间") = now
drnew("接收内容") = e.Message
Dim msg As String = e.Message
Dim ret As String = "!false"

If e.Message.StartsWith("%") AndAlso e.Message.EndsWith("%")    '假定收到的信息头尾都是"%"
        msg    = msg.Trim("%")
        Dim prefix As String = msg.SubString(0,1)
        Select Case prefix
                Case "G" '查询
                        msg = msg.TrimStart("G","|")
                        Dim dr As DataRow = DataTables("订单").Find("订单编号 ='" & msg & "'")
                        If dr IsNot Nothing Then
                                ret = "%" & dr("客户名称") & "|" & dr("订单日期") & "|"    & dr("发货日期")'拼接主表字段内容

                                Dim drs As List(Of DataRow)
                                drs = dr.GetChildRows("订单明细")
                                If drs.Count > 0 Then
                                        ret = ret & "$" '分割主表和明细
                                        For Each dr1 As DataRow In drs
                                                ret = ret & "@" & dr1("产品代码") & "|" & dr1("产品名称") & "|" & dr1("数量") & "|" & dr1("单位") & "|" & dr1("单价")'拼接每一行明细的字段内容,必须和网页中定义的列顺序一致
                                        Next
                                        ret = ret & "@"
                                End If

                                ret &= "%"
                        End If
                Case "U" '更新
                        msg = msg.TrimStart("U","|")
                        Dim dataArray() As String = msg.Split("|")
                        If dataArray.Length = 8 Then
                                Dim dr As DataRow = DataTables("订单明细").Find( dataArray(0) & "='" & dataArray(1) & "' and " & dataArray(2) & "='" & dataArray(3) & "'")
                                If dr IsNot Nothing Then '接收的内包含列名称和列值,这样就不用判断是更新那一列
                                        dr(dataArray(4)) = dataArray(5)
                                        dr(dataArray(6)) = dataArray(7)
                                        DataTables("订单明细").Save()
```

```
                                ret = "!true"
                            End If
                        End If
                Case "D" '删除
                        msg = msg.TrimStart("D","|")
                        Dim dataArray() As String = msg.Split("|")
                        If dataArray.Length = 4 Then
                            Dim dr As DataRow = DataTables("订单明细
").Find(dataArray(0) & "='" & dataArray(1) & "' and " & dataArray(2) & "='" & dataArray(3)
& "'")
                            If dr IsNot Nothing Then
                                dr.Delete()
                                ret = "!true"
                                DataTables("订单明细").Save()
                            End If
                        End If
                Case Else
        End Select
        drnew("发送内容") = ret
        e.ReturnValue = ret
        drnew("发送时间") = now
End If
drnew.Save()
```

代码比较简单，就不多做解释了。

4.17 权限管理

4.17.1 关于权限管理

实际上前面的章节，我们已经陆续接触到一些权限管理方面的知识，本章将进行进一步的总结。

Foxtable 采用事件驱动，例如编辑某单元格前，会触发 PrepareEdit 事件，你可以在这里设置代码，判断当前用户是否有编辑此单元格的权限，如果没有，则取消编辑操作。
不仅仅是编辑单元格，在 Foxtable 执行任何一项操作，都会触发相应的事件，例如增加行，会触发 BeforeAddDataRow 事件，删除行会触发 BeforeDeleteDataRow 事件，调整列宽会触发 BeforeResizeColumn 事件，甚至显示一个单元格，也会触发 DrawCell 事件，用以决定是否显示、如何显示此单元格的内容。
所以 Foxtable 的权限管理可以做得非常的精细，不仅可以控制某个表、某一行、某一列，还可以控制某个单元格的内容能被那些用户编辑或查看；你甚至可以伪装数据，使某单元格的数据在某

些用户前面可以真实显示，例如 0.8，而另一些用户看到的却是伪装后的数据，例如 0.5，或者空白；不仅如此，任何一个菜单命令，任何一个按钮，任何一项操作，小到移动行，调整列宽这些细微的操作，都可以设置为必须经过授权才能执行，可以说，在 Foxtable 中，没有一项操作是可以脱离授权控制的。

正因为 Foxtable 的权限管理非常灵活，所以其权限管理没有定式，可以说是千变万化，随心所欲，能满足客户对于权限管理方面任何复杂的需求。

4.17.2 数据管理权限

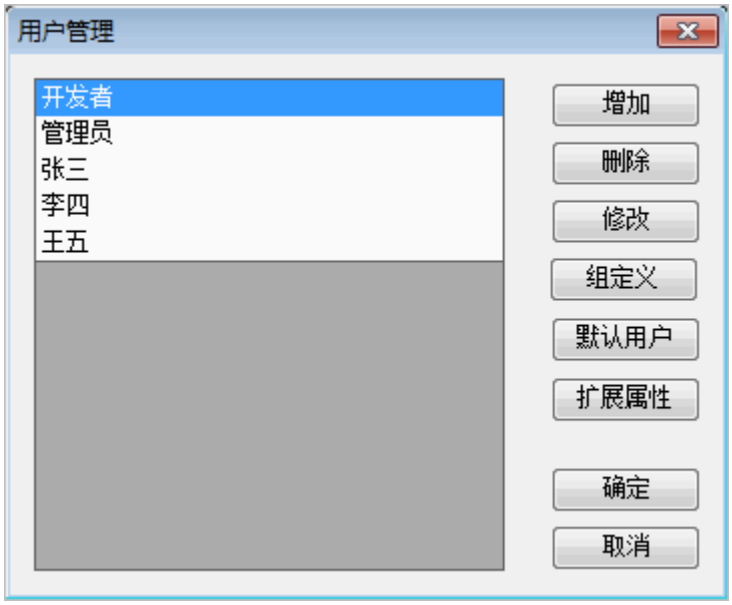
4.17.2.1 让不同用户编辑不同的行

假定订单表有一个业务员列，用于登记负责每个订单的业务员。

如果希望每个业务员只能编辑自己负责的订单，对于其他业务员负责的订单，则只能查看不能编辑。

设计步骤：

1、首先在用户管理中，加上每个业务员:



2、然后在订单表的 DataRowAdding 事件 中加入代码：

```
e.DataRow("业务员") = User.name
```

这样新增订单的时候，会自动在业务员列填入当前登录用户名。

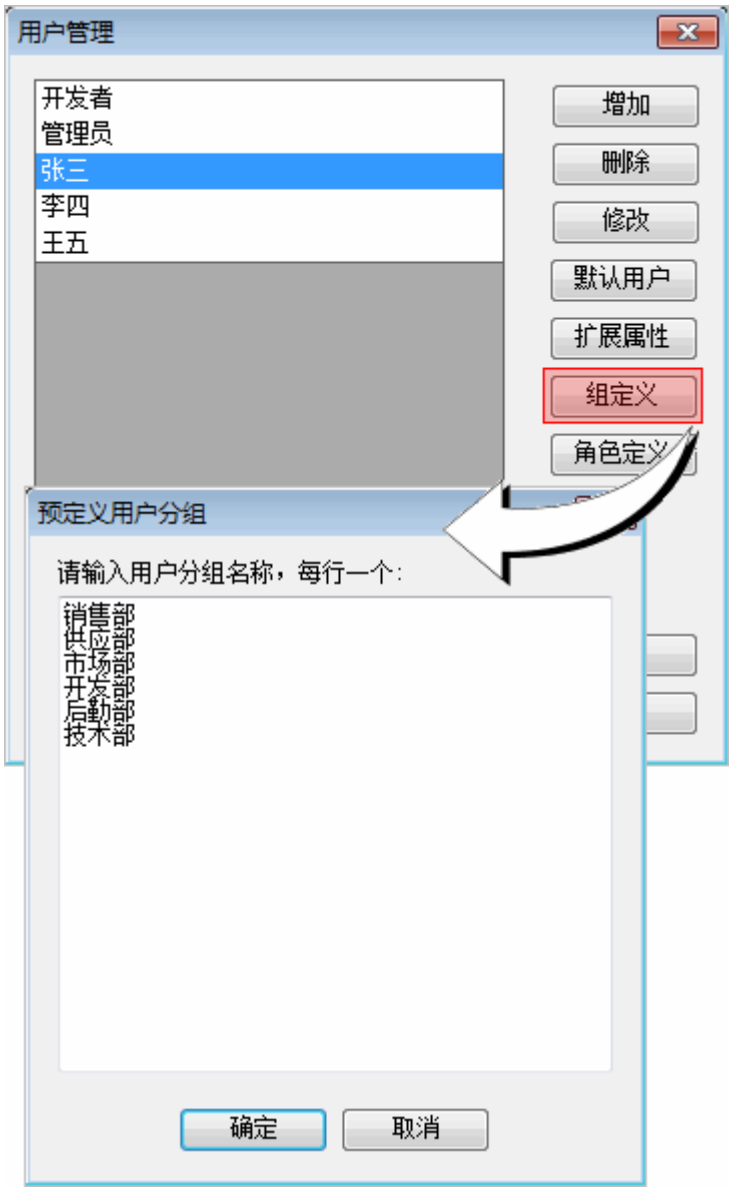
3、最后将订单表的 PrepareEdit 事件设置为：

```
If e.Row("业务员") <> User.Name Then
    e.Cancel = True
End If
```

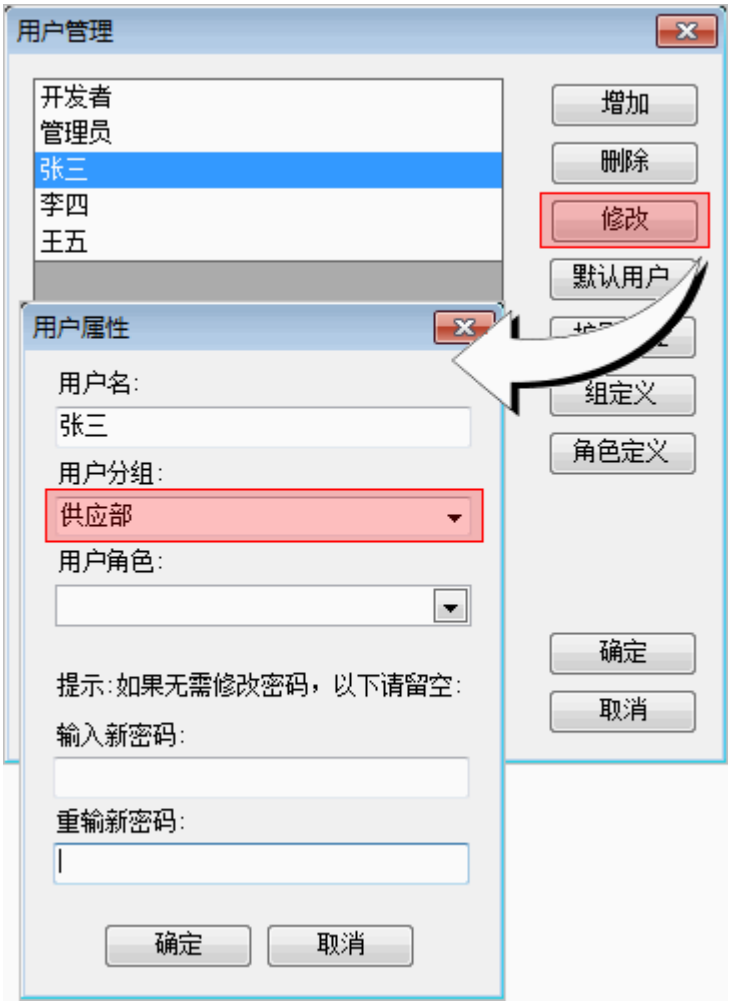
这样在编辑某个订单前，会自动判断负责此订单的业务员是否和登录用户名相同，如果不同，则取消编辑。

4.17.2.2 让不同部门编辑不同的行

假定某个表有一个部门列，用于登记负责此记录的部门。
如果 要求每个部门的用户只能编辑本部门的记录，对于其他部门负责的行，则只能查看不能编辑。
设计步骤
1、首先要在用户管理中定义用户分组，用部门名称作为用户分组名：



2、为每个用户指定分组，也就是每个用户所属的部门：



3、然后将表的 DataRowAdding 事件设置为：

```
e.DataRow("部门") = User.Group
```

这样新增行的时候，会自动在部门列填入当前登录用户所属的分组，也就是该用户所属的部门。

4、最后将表的 PrepareEdit 事件设置为：

```
If e.Row("部门") <> User.Group Then
    e.Cancel = True
End If
```

这样在编辑某行前，会自动判断负责此行的部门列是否和登录用户所属分组相同，如果不同，则取消编辑。

4.17.2.3 编辑前进行复杂条件判断

Foxtable 的权限管理不仅可以基于用户名或用户分组，还可以进行更复杂的条件判断。

假定订单表有一个业务员列，用于登记负责每个订单的业务员，另外有一个结账列，逻辑型，用于记录此订单是否已经结账。

现在要求：

1、每个业务员只能编辑自己负责的订单，对于其他业务员负责的订单，则只能查看不能编辑。

2、张经理可以编辑任何业务员的订单。

3、对于经结账的记录，普通业务员不能编辑。

4、对于未超过 10 天的订单，即使已经结账，张经理还是可以编辑。

设计步骤：

1、将订单表的 DataRowAdding 事件设置为：

```
e.DataRow("业务员") = User.name
```

这样新增订单的时候，会自动在业务员列填入当前登录用户名。

2、最后将订单表的 PrepareEdit 事件设置为：

```
If User.Name = "张经理" Then '如果是张经理
    If e.Row("结账") = True Then '此订单已经结账
        Dim tv As Date = e.Row("日期")
        Dim tp As TimeSpan = Date.Today - tv
        If tp.TotalDays >= 10 Then '如果订单已经超过 10 天
            e.Cancel = True '则取消编辑
        End If
    End If
ElseIf e.Row("业务员") <> User.Name Then '如果是其他用户,且业务员和登录用户名不同
    e.Cancel = True '则取消编辑
End If
```

4.17.2.4 让不同用户编辑不同的列

假定一个表，要求：

1、禁止张三和李四编辑第一列、第二列和第三列。

2、禁止王五和赵六编辑第四列、第五列和第六列。

3、只有张经理编辑才能编辑第十列。

为此可将此表的 PrepareEdit 事件代码设置为：

```
Select Case User.Name
    Case "张三","李四"
        Select Case e.Col.Name
            Case "第一列","第二列","第三列"
                e.Cancel = True
        End Select
    Case "王五", "赵六"
        Select Case e.Col.Name
            Case "第四列","第五列","第六列"
                e.Cancel = True
        End Select
End Select
If e.Col.Name = "第十列" Then
    If User.Name <> "张经理" Then
        e.Cancel = True
    End If
End If
```

实际应用的时候，将上述代码替换为具体的列名和用户名即可。

上面的代码是基于用户名的，同样可以改为基于用户分组的，只需将上述代码中的 User.Name 改为 User.Group，并将用户名替换为分组名即可。

4.17.2.5 让不同用户处理不同的行

我们已经学会了如何设置代码，使得用户只能编辑自己负责的行，而对于其他用户负责的行，则只能查看，不能编辑。

不过有的时候，对于其他用户负责的行，连查看都是不允许的。

对于这种情况，我们可以用动态加载的功能，在打开项目后，只加载登录用户负责的行即可实现目的。

以订单表为例，我们要求：

1、张经理可以处理所有订单。

2、其他用户只能处理自己负责的订单。

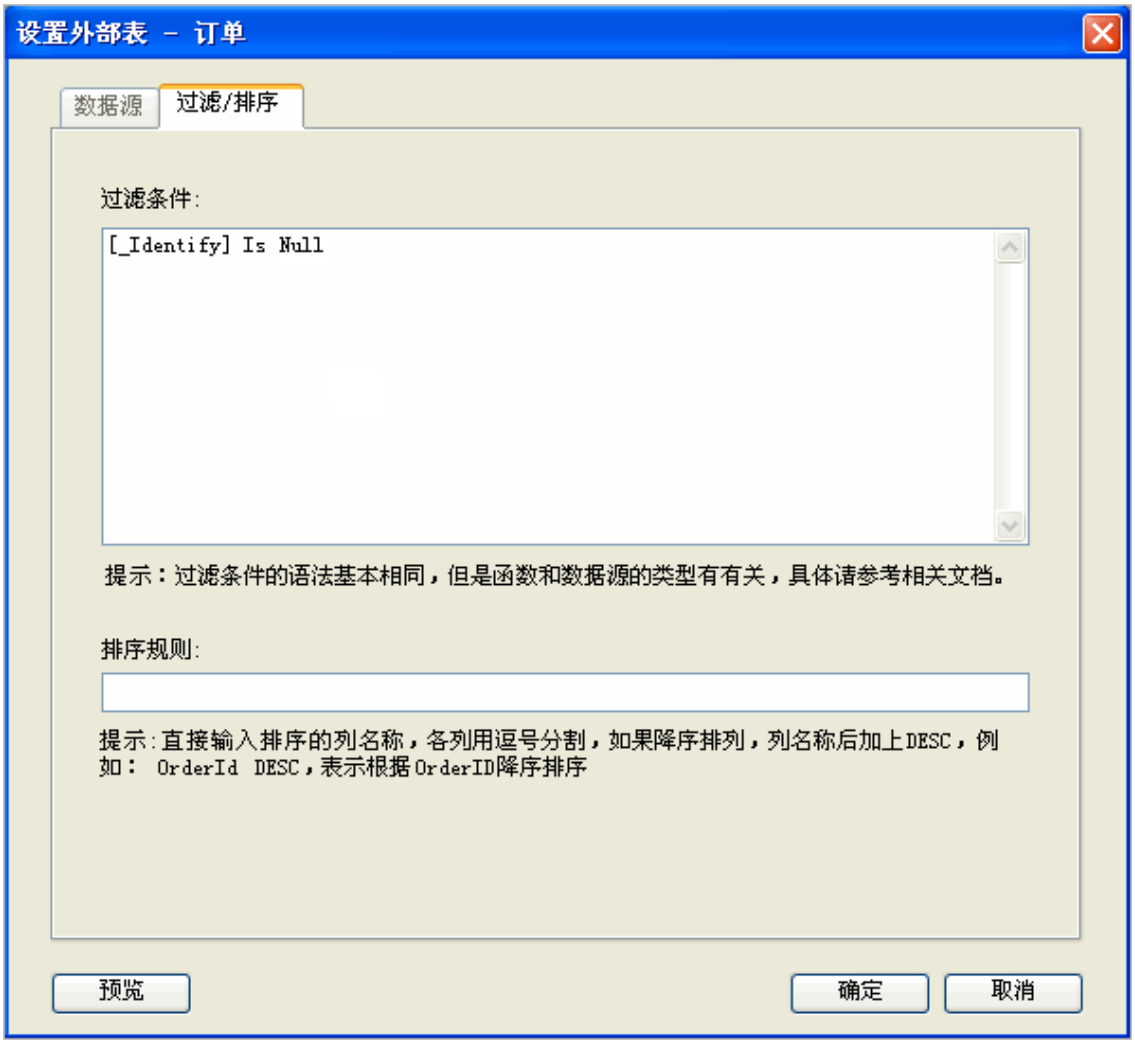
设计步骤：

1、首先我们要确保在初始状态下，不加载任何订单，对于内部表和外部表，有不同的处理方式。

如果是内部表，设置 BeforeLoadInnerTable 事件为：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

如果是外部表，可以在定义外部表的时候，直接定义加载条件：



由于任何行的_Identify 列的内容都不可能为空，所以上述代码和设置会使得初始状态不会加载任何订单。

2、在项目的 LoadUserSetting 事件中设置代码，加载用户负责的订单:

```
If User.Name = "张经理" Then
    DataTables("订单").LoadFilter = ""
Else
    DataTables("订单").LoadFilter = "业务员 = '" & User.Name & "'"
End If
DataTables("订单").Load()
```

这样用户登录后，系统判断登录用户是否是张经理，如果是的，则加载所有订单，否则只加载登录用户负责的订单。

上面的代码是基于用户名的，同样可以改为基于用户分组的，只需将上述代码中的 User.Name 改为 User.Group，并将用户名替换为分组名即可。

4.17.2.6 让不同用户处理不同的列

前面我们已经学会了如何设置代码，使得某些列只能被特定的用户编辑。

有的时候，可能希望某些列只能被特定的用户查看和编辑，而其他用户连查看都是不允许的。

假定一个表，要求：

1、张三和李四不能看到第一列、第二列和第三列。

2、王五和赵六不能看到第四列、第五列和第六列。

3、只有张经理编辑才能看到第十列。

为此可以在项目的 AfterOpenProject 事件中设置代码：

```
Select Case User.Name
    Case "张三","李四"
        Tables("订单").Cols.Remove("第一列")
        Tables("订单").Cols.Remove("第二列")
        Tables("订单").Cols.Remove("第三列")
    Case "王五", "赵六"
        Tables("订单").Cols.Remove("第四列")
        Tables("订单").Cols.Remove("第五列")
        Tables("订单").Cols.Remove("第六列")
End Select
If User.Name <> "张经理" Then
    Tables("订单").Cols.Remove("第 十列")
End If
```

提示：上述代码必须重新打开项目才会正常生效，切换用户是无效的。

上述代码在打开项目后执行，移除登录用户无权处理的列，例如王五登录后，第四列、第五列和第六列将被移除(注意不是删除)，即使执行取消隐藏列命令，也无法将这些列重新显示出来。

虽然这些列已经从 Table 中移除，用户无法编辑和查看其内容，但是仍然可以通过代码或表达式使用或设置这些列的内容。

上述代码是基于用户名的，同样可以改为基于用户分组，只需将上述代码中的 User.Name 改为 User.Group，并将用户名替换为分组名即可。

4.17.2.7 精确控制单元格编辑权限

Foxtable 的权限不仅可以针对表、行、列，还可以精确控制到单个单元格。
假定有一个名为“标注”的列， 当此列中某单元格内容为空的时候，任何用户都可以编辑，如果单元格已经有内容，则只有张经理才能编辑。
为此可将表的 PrepareEdit 事件代码设置为：

```
If e.Col.Name = "标注" Then '如果修改的是标注列
    If e.Row.IsNull("标注") = False Then '如果已经有内容
        If User.Name <> "张经理" Then '且用户不是张经理
            e.Cancel = True '那么禁止编辑
        End If
    End If
End If
```

4.17.2.8 给用户授予输入值许可范围

我们之前的例子，都是在编辑之前，执行 PrepareEdit 事件，判断用户是否有编辑此列、此行或此单元格的权限。
权限管理不会总是如此简单，有时不需要在编辑前进行判断，而是在编辑后，判断用户输入的值是否超出授权范围。
例如给订单打折，不同等级的用户，不同的订购数量，能打折的程度是不同的。
此时我们不能使用 PrepareEdit 事件，因为这个是在编辑前执行，要在编辑后判断输入值是否超出授权范围，通常用 ValidateEdit 或者 DataColChanging 事件。
假定订单表中有个折扣列，普通业务员给出的折扣不能超过 5%，张经理则不能超过 10%，对于订购数量达到 1000 的订单，张经理有权给出 15%的折扣。
为此可将此表的 ValidateEdit 代码设置为：

```
If e.Col.Name = "折扣" Then '如果修改的是折扣列
    Dim v As Double = e.Text '取得新输入的折扣
    If User.Name = "张经理" Then '如果用户是张经理
        If v > 0.15 Then '且折扣大于 0.15
            e.Cancel = True '取消此次输入
        ElseIf v> 0.1 AndAlso e.Row("数量") < 1000 Then '如果折扣大于 0.1,但数量少于 1000
            e.Cancel = True '取消此次输入
        End If
    ElseIf v > 0.05 Then '如果是普通用户,且折扣大于 0.05
        e.Cancel = True '取消此次输入
    End If
    If e.Cancel Then '如果本次输入被取消,则提示用户原因
        Messagebox.show("你输入的折扣超出授权范围!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
    End If
End If
```

我们在介绍 ValidateEdit 和 DataColChanging 事件的时候，也提供了很多验证输入值的例子，大家

可以复习一下。

4.17.2.9 让不同用户处理不同的表

通过一些简单的代码，我们可以让不同的人编辑或查看不同的表，也可以为不同的用户加载不同的表。

1、让不同用户编辑不同的表

例如在表 A 的 PrepareEdit 设置代码：

```
Select Case User.Name
    Case "张三","李四"
        e.Cancel = True
End Select
```

即可禁止张三和李四编辑表 A。

2、让不同用户查看不同的表

例如在项目事件 AfteOpenProject 事件中设置代码：

```
Select Case User.Name
    Case "张三","李四"
        Tables("表 A").Visible = False
    Case "王五"
        Tables("表 B").Visible = False
End Select
```

即可使得张三或李四打开项目后，将看不到表 A，王五打开项目后，将看不到表 B，不过这些表仍然可以通过代码使用。

3、为不同用户加载不同的表

BeforeLoadInnerTable 和 BeforeLoadOuterTable 事件都有一个 Cancel 属性，将此属性设置为 True，将不加载对应的表。

例如在 BeforeLoadInnerTable(内部表)或 BeforeLoadOuterTable(外部表)中设置代码：

```
Select Case e.DataTableName
    Case "表 A"
        e.Cancel = (e.User.Name = "张三")
    Case "表 B"
        e.Cancel = (e.User.Name = "李四")
    Case "表 C"
        e.Cancel = (e.User.Name = "王五")
End Select
```

这样张三打开项目，将不会加载表 A，需要注意的是，如果打开项目后，有代码引用表 A，将会出错，因为此时表 A 是不存在的。

同样李四打开项目不会加载表 B，王五打开项目不会加载表 C。

4.17.2.10 为某些操作设置权限

不仅是查看和编辑，Foxtable 中任何操作都可以进行授权设置，例如移动列，移动行、删除行、锁定行、调整列宽和列位置等等。
假定有个表，要求只有张经理才能锁定或取消锁定行，可以将表的 DataRowLockedChanging 事件设置为：

```
If User.Name <> "张经理" Then
    e.Cancel = True
     Messagebox.show("你无权  锁定和取消锁定行!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

4.17.2.11 记录重要数据的编辑人和编辑时间

假定产品表中有一个单价列，你认为这一列比较重要，所以希望增加一列，用于记录：谁在什么时候修改了此产品的单价。
假定新增列的名称为"Log"，你可以在产品的 AfterEdit 事件中设置如下代码：

```
If e.Col.Name = "单价" Then
    e.Row("Log") = User.Name & " " & Date.Now
End If
```

为了禁止人为修改 Log 列的内容，还必须在 PrepareEdit 事件中设置代码：

```
If e.Col.Name = "Log" Then
    e.Cancel = True
End If
```

4.17.2.12 隐藏或混淆重要数据

假定表中某列的数据非常重要，只能张经理和王总监二人才能查阅，希望其他人打开此表后，此列内容全部用一串星号(*)代替。
要实现此目的，可以将此表的 DrawCell 事件代码设置为：

```
If e.Col.Name = "列名" Then '请改为实际列名
    If User.name <> "张经理" AndAlso UserName <> "王总监" Then '如果不是指定用户
        e.Text = "****" '则修改显示内容
    End If
End If
```

你可以用空白和其他任何字符或符号来代替要显示的内容，
再例如在订单表中，要求每个业务员只可以查看自己所负责订单的折扣，而查看其他业务员负责的订单的时候，折扣一律显示为"0"。
要实现此目的，可以将订单表的 DrawCell 事件代码设置为：

```
If e.Col.Name = "折扣" Then
    If User.Name <> e.Row("业务员")
        e.Text = "0"
```

```
    End If
End If
```

4.17.2.13 领导审批功能的实现

假定有两个用户分组，分别为经理和雇员，经理有审批和取消审批的功能，某行一旦通过审批，将禁止修改。
为此可以增加一个逻辑型列，名为“审批”，然后将表事件 PrepareEdit 的代码设置为：

```
If e.Col.Name = "审批" Then '只有经理才能审批
    If User.Group <> "经理"  Then
       e.Cancel = True
    End If
ElseIf e.Row("审批") = True Then '如果此行已经审批了,则禁止修改
    e.Cancel = True
End If
```

经理级别的用户要审批某行时，只需单击勾选此行的审批列即可，通过审批的行，经理和雇员都不能编辑，除非经理取消审核。

有的时候，可能希望能记录审批人和审批时间，可以增加一个名为“审批人”的字符型，用于记录审批人，另外增加一个日期时间列，名为“审批日期”，用于记录审批日期。
然后在窗口或菜单中加一个审批按钮，将按钮的代码设置为：

```
If User.Group = "经理" Then
    Tables("表A").Current("审批  人") = User.Name
    Tables("表A").Current("审批日期") = Date.Today()
Else
    MessageBox.show("你无审批权限!")
End If
```

当然还得来个取消审批的按钮，代码为：

```
If User.Group = "经理" Then
    Tables("表A").Current("审批  人") = Nothing
    Tables("表A").Current("审批日期") = Nothing
Else
    MessageBox.show("你无取消审批权限!")
End If
```

最后将表事件PrepareEdit的代码设置为：

```
If e.Row.IsNull("审批人") = False Then '如果审批人列不为空
    e.Cancel = True '则禁止编辑
End If
```

这样经过审批的行，将不能编辑。

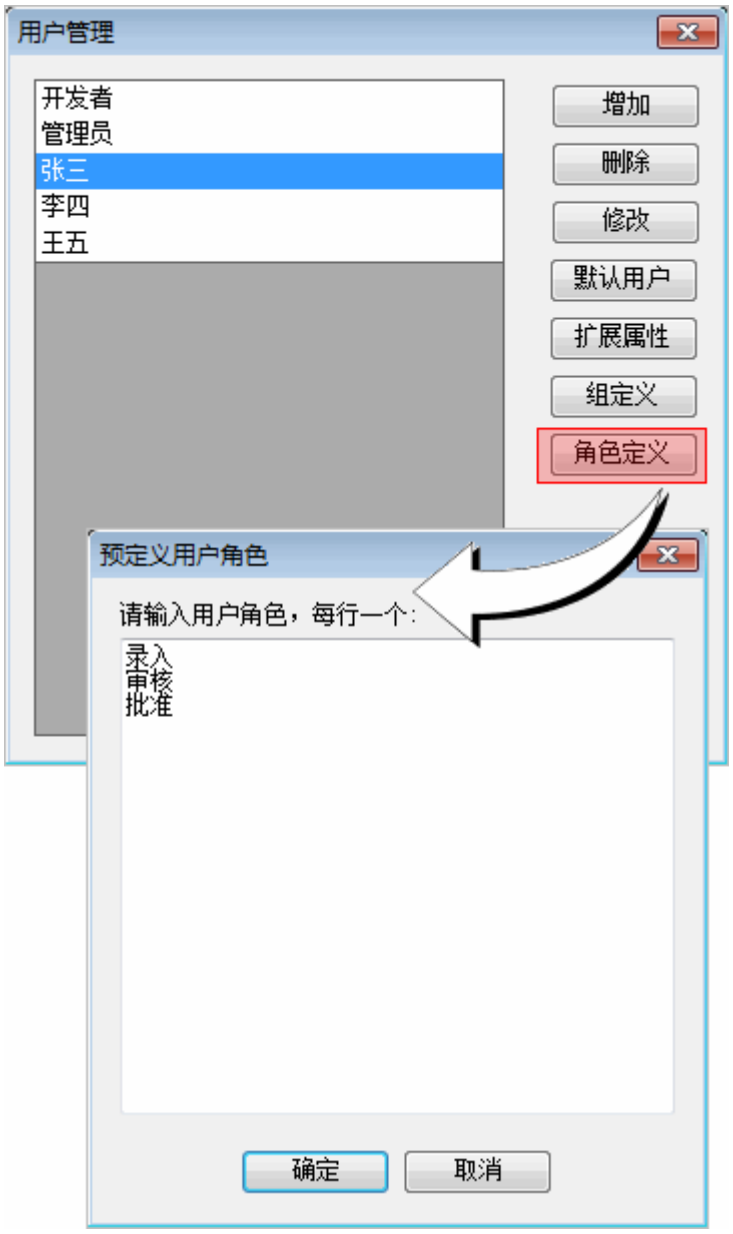
4.17.2.14 禁止非法复制数据

你可能碰到过这样的网站，只能查看网页内容，但不能复制网页内容。
Foxtable 同样可以实现类似的功能，防止重要数据被非法复制传播。
DataTable 有一个 AllowClipBoard 的逻辑属性，只需将其设置为 False，即可禁止复制该表的数据。
示例
假定有一个名为客户的用户分组，这个分组的用户登录项目后，禁止其复制所有表的数据，为此只需将 LoadUserSetting 事件代码设置为：

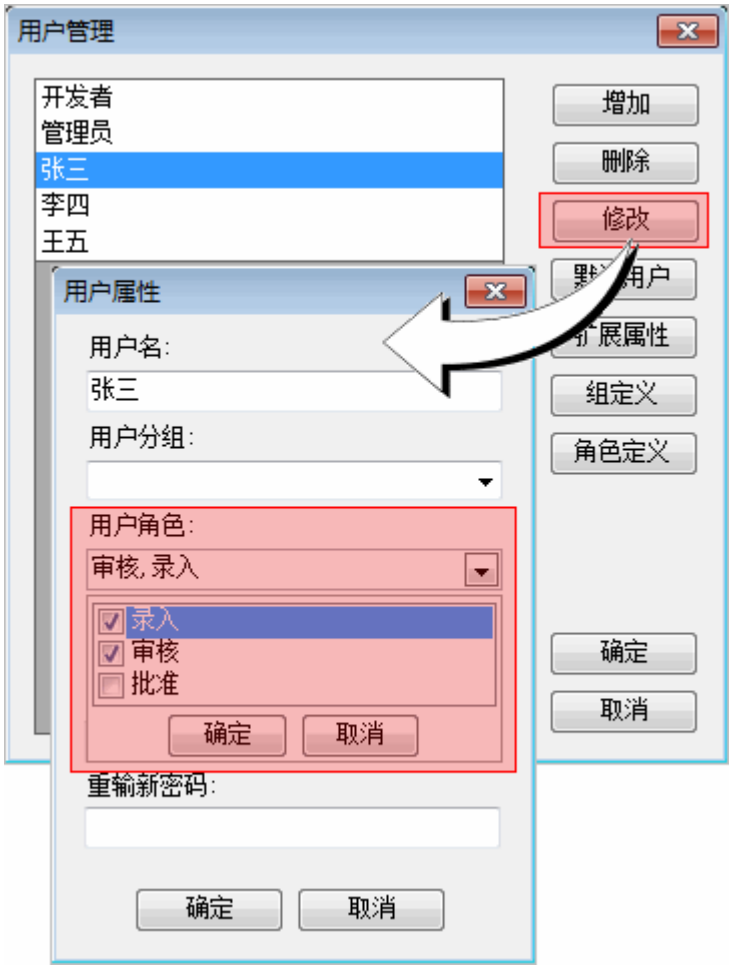
```
If User.Group = "客户" Then
    For Each dt As DataTable In DataTables
        dt.AllowClipBoard = False
    Next
End If
```

4.17.2.15 用户角色和权限管理

前面的权限管理都是基于用户名和用户分组的，这能满足绝大多数场合的需求。
有的时候一个用户需要承担多个角色，此时不能采用用户分组来区分用户，因为一个用户只能属于一个分组。
最新版本的 Foxtable 引入了用户角色的概念，一个用户可以具备多个角色。
定义用户角色
1、首先在用户管理中，定义好用户角色：



2、然后指定每个用户的角色，一个用户可以具备多个角色：



根据用户角色设置权限

假定在窗口或菜单中有个审核按钮，要求具备审核角色的用户才能审核，可以将此按钮的代码设置为：

```
If User.IsRole("审核") Then
    '审核代码
Else
    MessageBox.Show("你无审核权限!")
End If
```

4.17.2.16 录入窗口与权限管理

输入框的授权

前面所有有关编辑的权限设置，都是针对数据表的。

如果采用录入窗口编辑数据，那么这些权限设置是无效的，我们需要另外的方法。

假定设计了一个录入窗口用于录入表 A 的数据，要求张三负责输入第一列、第二列、第三列的内容，李四负责输入第五列、第六列、第七列的内容。

要实现此目的，只需在窗口的全局事件的 Enter 事件中设置代码：

```
Dim ctl As WinForm.Control = e.Sender
Select Case ctl.BindingField
    Case "表A.第一列", "表A.第二列", "表A.第三列"
        If User.Name = "张三" Then
            ctl.ReadOnly = BooleanEnum.False '允许编辑
        Else
            ctl.ReadOnly = BooleanEnum.True '禁止编辑
        End If
    Case "表A.第四列", "表A.第五列", "表A.第六列"
        If User.Name = "李四" Then
            ctl.ReadOnly = BooleanEnum.False '允许编辑
        Else
            ctl.ReadOnly = BooleanEnum.True '禁止编辑
        End If
End Select
```

参考： BindingField ReadOnly

如果只有个别的文本框需要进行权限控制，就没有必要使用全局事件，将代码直接设置在文本框的 Enter 事件即可。
例如窗口中有一文本框，只有张三才能在其中输入数据，可将此文本框的 Enter 事件代码设置为：

```
If User.Name = "张三" Then
    ctl.ReadOnly = BooleanEnum.False '允许编辑
Else
    ctl.ReadOnly = BooleanEnum.True '禁止编辑
End If
```

按钮的授权

除了输入框，有些按钮可能也需要授权的，只能特定的用户才能执行。
有两个方法可以对按钮进行授权，假定某个按钮执行经理级别的用户才能执行，可以将该按钮的Click事件代码改为：

```
If User.Group = "经理" Then
    '原来的代码
Else
    MessageBox.show("你无权执行此项操作!")
End If
```

另一个方法是在窗口的 AfterLoad 事件设置代码，隐藏或禁用当前用户无权使用的按钮，例如我们在 AfterLoad 事件设置代码：

```
If User.Group <> "经理" Then
    e.Form.Controls("Button1").Enabled = False
    e.Form.Controls("Button3").Enabled = False
End If
```

这样当非经理级别的用户打开窗口后，Button1 和 Button3 将处于禁用状态，如果你想隐藏这两个按钮，只需将上述代码中的 Enabled 替换为 Visible。

4.17.2.17 菜单与权限管理

菜单的功能区、功能组以及所有的菜单组件，都有 Visible 和 Enabled 属性，这样我们可以根据登录用户的不同，隐藏和禁用某些菜单项目，实现菜单的授权。

示例一

我们增加一个用户“录入员”，如果该用户登录，隐藏“数据表”和“管理项目”两个功能区，并禁用杂项功能区的“执行 SQL”和 “命令窗口”两个按钮。

第一步当然是增加一个名为“录入员”的用户，然后在菜单系统事件 LoadUserSetting 加入如下代码：

```
If User.Name = "录入员" Then
    RibbonTabs("Table").Visible = False
    RibbonTabs("Project").Visible = False
    RibbonTabs("Other").Groups("Execute").Items("SQLExecute").Enabled = False
    RibbonTabs("Other").Groups("Execute").Items("Command").Enabled = False
Else
    RibbonTabs("Table").Visible = True
    RibbonTabs("Project").Visible = True
    RibbonTabs("Other").Groups("Execute").Items("SQLExecute").Enabled = True
    RibbonTabs("Other").Groups("Execute").Items("Command").Enabled = True
End If
```

设计完成后，请切换到录入员用户，看看是否已经实现设计要求。

示例二

在默认情况下，锁定某个表，将不能修改此表的数据，也不能增加或删除行，但是可以增加、删除、更改列；也就是说，锁定表只是保护数据，却不保护结构。

如果希望表被锁定的情况下，能同时禁止修改数据和结构，可以在系统事件 CurrentTableChanged 中设置代码：

```
If CurrentTable.AllowEdit Then
    RibbonTabs("Table").Groups("Column").Items("InsertColumn").Enabled = True
    RibbonTabs("Table").Groups("Column").Items("AddColumn").Enabled = True
    RibbonTabs("Table").Groups("Column").Items("ModiColumn").Enabled = True
    RibbonTabs("Table").Groups("Column").Items("DeleteColumn").Enabled = True
Else
    RibbonTabs("Table").Groups("Column").Items("InsertColumn").Enabled = False
    RibbonTabs("Table").Groups("Column").Items("AddColumn").Enabled = False
```

```
    RibbonTabs("Table").Groups("Column").Items("ModiColumn").Enabled = False
    RibbonTabs("Table").Groups("Column").Items("DeleteColumn").Enabled = False
End If
```

原理很简单，在选择不同的表之后，根据当前表(CurrentTable)的 AllowEdit 属性，来禁止或启用与修改表结构相关的菜单按钮。
上面的代码也可以简化为：

```
RibbonTabs("Table").Groups("Column").Items("InsertColumn").Enabled = CurrentTable.AllowEdit
RibbonTabs("Table").Groups("Column").Items("AddColumn").Enabled = CurrentTable.AllowEdit
RibbonTabs("Table").Groups("Column").Items("ModiColumn").Enabled = CurrentTable.AllowEdit
RibbonTabs("Table").Groups("Column").Items("DeleteColumn").Enabled = CurrentTable.AllowEdit
```

示例三
举一反三，同样我们可以在 MainTableChanged 或 CurrentTableChanged 事件中设置代码，使得选择某些表后，自动禁用指定的菜单命令，而选择另一些表后，则自动启用指定的菜单命令。
例如我们在 MainTableChanged 设置如下的代码：

```
Select Case MainTable.name
    Case "表A","表C"
        RibbonTabs("Table").Groups("Column").Items("InsertColumn").Enabled = True
        RibbonTabs("Table").Groups("Column").Items("AddColumn").Enabled = True
        RibbonTabs("Table").Groups("Column").Items("ModiColumn").Enabled = True
        RibbonTabs("Table").Groups("Column").Items("DeleteColumn").Enabled = True
    Case "表B"
        RibbonTabs("Table").Groups("Column").Items("InsertColumn").Enabled = False
        RibbonTabs("Table").Groups("Column").Items("AddColumn").Enabled = False
        RibbonTabs("Table").Groups("Column").Items("ModiColumn").Enabled = False
        RibbonTabs("Table").Groups("Column").Items("DeleteColumn").Enabled = False
End Select
```

这样我们选择表 A 或者表 C 的时候，可以通过菜单命令修改其表结构，而选择表 B 的时候，将无法通过菜单命令修改其结构，因为相关菜单命令已经禁用。

4.17.2.18 可视化授权的实现

本节的内容，可以参考 CaseStudy 目录下的文件：可视化授权.Table
Foxtable 的权限管理非常完善，几乎可以控制每一个细节，大到一个表是否可见，小到一个单元格是否可编辑，都可以精确地控制。
但是很多易表用户，都留恋易表的可视化授权方式。
其实在 Foxtable 中，我们只需十几行的代码，就可以设计出一个可视化授权管理功能。
首先我们设计一个下图所示的表，名为授权表：

	用户名	表名	列名	不可见	不可编辑
1	张三	表A		☑	☐
2	张三	表B		☐	☑
3	李四	表B		☑	☐
4	李四	表C		☐	☑
5	王五	表A	第三列	☑	☐
6	王五	表A	第八列	☐	☑

然后项目事件 LoadUserSetting 中，加入如下代码：

```
For Each t As Table In Tables
    t.Visible = True
    t.AllowEdit = true
    For Each c As Col In t.Cols
        c.Visible = True
        c.AllowEdit = True
    Next
Next
Tables("授权表").Visible = (User.Type <> UserTypeEnum.User )
If User.Type = UserTypeEnum.User Then
    For Each dr As DataRow In DataTables("授权表").Select("用户名 = '" & User.Name & "'" )
        If dr.IsNull("列名") Then
            Tables(dr("表名")).Visible = Not dr("不可见")
            Tables(dr("表名")).AllowEdit = Not dr("不可编辑")
        Else
            Tables(dr("表名")).Cols(dr("列名")).Visible = Not dr("不可见")
            Tables(dr("表名")).Cols(dr("列名")).AllowEdit = Not dr("不可编辑")
        End If
    Next
End If
```

上面的代码，条理清晰，原理简单，相信不用我解释，大家也能看明白。

代码是通用的，如果你需要禁止某用户编辑某个表或列，或者禁止某用户查看某个表或列，只需在授权表中加入相应的设置即可。

也就说是，需要禁止某个权限的时候，才需要在授权表加入相应的条目。

例如按照上图的设置，张三看不到表 A，不能编辑表 B，李四则看不到表 B，不能编辑表 C，至于王五，比较特殊，他看不到表 A 的第三列，不能编辑表 A 的第八列。

实际应用的时候，我们更多的是根据用户分组进行授权，此时只需将授权表的用户名改为分组名，同时将代码改为：

```
For Each t As Table In Tables
    t.Visible = True
    t.AllowEdit = true
    For Each c As Col In t.Cols
        c.Visible = True
        c.AllowEdit = True
    Next
Next
Tables("授权表").Visible = (User.Type <> UserTypeEnum.User )
If User.Type = UserTypeEnum.User Then
```

```
    For Each dr As DataRow In DataTables("授权表").Select("分组名 = '" & User.Group & "'" )
        If dr.IsNull("列名") Then
            Tables(dr("表名")).Visible = Not dr("不可见")
            Tables(dr("表名")).AllowEdit = Not dr("不可编辑")
        Else
            Tables(dr("表名")).Cols(dr("列名")).Visible = Not dr("不可见")
            Tables(dr("表名")).Cols(dr("列名")).AllowEdit = Not dr("不可编辑")
        End If
    Next
End If
```

如果不考虑中途切换用户，那么代码可以挪到项目事件 AfterOpenProject 中，下面的这一段代码可以删除：

```
For Each t As Table In Tables
    t.Visible = True
    t.AllowEdit = true
    For Each c As Col In t.Cols
        c.Visible = True
        c.AllowEdit = True
    Next
Next
```

本节的内容只是提供了一个思路，其实授权是千变万化的，没有固定的模式，大家应该根据实际需要来确定自己的授权方式。

4.17.2.19 目录树形式的授权

普通用户可以忽略本节的内容。

本节内容可以参考 CaseStudy 目录下的文件: 目录树形式的授权.Table

设计步骤：

1、新建一个下图所示的表：

	分组	权限	用户
1	订单管理		经理, 杨刚
2	订单管理	审核订单	主管
3	订单管理	批准订单	
4	订单管理	新建订单	李勇寿, 陈夏燕
5	订单管理	修改订单	董燕, 林海, 朱新志, 李勇寿, 陈夏燕
6	生产管理		贺辉, 董燕
7	生产管理	下料	林海, 李勇寿
8	生产管理	出片	董燕, 陈夏燕
9	生产管理	印刷	陈夏燕
10	生产管理	外协	主管
11	生产管理	成品	杨刚, 李勇寿
12	生产管理	发货	林海, 朱新志
13	业务管理		经理, 主管
14	业务管理	修改报价	陈夏燕
15	业务管理	拼版工单	李勇寿

这里列出具备此权限的用户，以逗号分割。

2、新建一个下图所示的窗口，右边目录树的“显示复选框”属性设置为 True：



3、设置窗口和控件的事件代码.

窗口的 AfterLoad 事件代码为:

```
Dim trv1 As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim trv2 As WinForm.TreeView = e.Form.Controls("TreeView2")
trv1.BuildTree("授权","分组|权限")
For Each u As UserInfo In Users
```

```
    If u.Type = UserTypeEnum.User Then
        If trv2.Nodes.Contains(u.Group) = False Then
            trv2.Nodes.Add(u.Group)
        End If
        trv2.Nodes(u.Group).Nodes.Add(u.Name)
    End If
Next
Trv1.Nodes(0).Expand()
trv2.ExpandAll()
```

左边目录树的 AfterSelectNode 事件代码：

```
Dim trv1 As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim trv2 As WinForm.TreeView = e.Form.Controls("TreeView2")
Dim dr As DataRow
If e.node.Level = 0 Then
    dr = DataTables("授权").Find("分组 = '" & e.Node.Name & "' And 权限 Is Null")
Else
    dr = DataTables("授权").Find("分组 = '" & e.Node.ParentNode.Name & "' And 权限 = '" & e.Node.Name & "'")
End If
Dim nms As New List(of String)
If dr IsNot Nothing AndAlso dr.IsNull("用户") = False
    nms.AddRange(dr("用户").Split(","))
End If
For Each nd As WinForm.TreeNode In trv2.AllNodes
    nd.Checked = nms.Contains(nd.Name)
Next
```

左边目录树的 AfterEditNode 事件代码：

```
If e.NewText = "" Then
    e.Cancel = True
ElseIf e.Node.Level = 0 Then '重命名分组
    Dim dt As DataTable = DataTables("授权")
    Dim dr As DataRow
    dr = dt.Find("分组 = '" & e.NewText & "' And 权限 Is Null" ) '判断新输入的分组是否存在
    If dr IsNot Nothing '如果存在,取消输入.
        MessageBox.Show("此分组已经存在!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
        e.cancel = True
    Else
        For Each dr In dt.Select("分组 = '" & e.Node.Name & "'" ) '重命名分组
            dr("分组") = e.NewText
        Next
        e.Node.Name = e.NewText '使得节点的名称和标题保持一致.
    End If
ElseIf e.Node.Level = 1 '重命名权限
```

```
    Dim dt As DataTable = DataTables("授权")
    Dim dr As DataRow
    Dim fz As String = e.Node.ParentNode.Name '获得分组名
    dr = dt.Find("分组 = '" & fz & "' And 权限 = '" & e.NewText & "'" ) '判断新输入的权限是否存在
    If dr IsNot Nothing '如果存在,取消输入.
        MessageBox.Show(" 此 权 限 已 经 存 在 !", " 提 示 ", MessageBoxButtons.OK,
MessageBoxIcon.Information)
        e.Cancel = True
    Else
        dr = dt.Find("分组 = '" & fz & "' And 权限 = '" & e.Node.Name & "'" ) '找出对应的行
        dr("权限") = e.NewText '重命名权限
        e.Node.Name = e.NewText '使得节点的名称和标题保持一致.
    End If
End If
```

右边目录树的 AfterCheckNode 事件：

```
Dim trv1 As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim trv2 As WinForm.TreeView = e.Form.Controls("TreeView2")
Dim nd1 As WinForm.TreeNode = trv1.SelectedNode
Dim dr As DataRow
If e.Node.ParentNode IsNot Nothing Then '取消父节点的选中标记,因为选中子节点,就不要选中父节点了
    e.Node.ParentNode.Checked = False
End If
For Each cnd As WinForm.TreeNode In e.Node.Nodes '取消子节点的选中标记,因为选中父节点,就不要选中子节点了
    cnd.Checked = False
Next
If nd1.Level = 0 Then
    dr = DataTables("授权").Find("分组 = '" & nd1.Name & "' And 权限 Is Null")
Else
    dr = DataTables("授权").Find("分组 = '" & nd1.ParentNode.Name & "' And 权限 = '" & nd1.Name & "'")
End If
If dr IsNot Nothing Then
    Dim nms As String
    For Each nd2 As WinForm.TreeNode In trv2.AllNodes
      If nd2.Checked  Then
        nms  = nms & "," &  nd2.Name
      End If
    Next
    If nms > "" Then
      dr("用户") = nms.Trim(",")
    Else
      dr("用户") = Nothing
    End If
End If
```

增加分类按钮的代码：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim i As Integer
trv.Select()
Do
    i = i+ 1
    Dim nm As String = "分组" & i
    If trv.Nodes.Contains(nm) = False
        Dim dr As DataRow = DataTables("授权").AddNew
        Dim nd As WinForm.TreeNode = trv.Nodes.Add(nm)
        dr("分组") = nm
        trv.SelectedNode = nd
        trv.BeginEdit
        Exit Do
    End If
Loop
```

增加权限按钮的代码：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim pd As WinForm.TreeNode = trv.SelectedNode
trv.Select()
If pd.Level = 1 Then
    pd = pd.ParentNode
End If
Dim i As Integer
Do
    i = i+ 1
    Dim nm As String = "权限" & i
    If pd.Nodes.Contains(nm) = False
        Dim nd As WinForm.TreeNode = pd.Nodes.Add(nm)
        Dim dr As DataRow = DataTables("授权").AddNew()
        dr("分组") = pd.name
        dr("权限") = nm
        trv.SelectedNode = nd
        trv.BeginEdit
        Exit Do
    End If
Loop
```

删除按钮的代码：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim nd As WinForm.TreeNode = trv.SelectedNode
trv.Select()
If nd Is Nothing Then
    Return
End If
```

```
If nd.Level = 0 Then '删除分组
    Dim dt As DataTable = DataTables("授权")
        If MessageBox.Show("确定要删除此分组码?","提示",MessageBoxButtons.YesNo,MessageBoxIcon.Question) = DialogResult.Yes Then
        For Each dr As DataRow In dt.Select("分组 = '" & nd.Name & "'")
            dr.Delete
        Next
        nd.Delete
    End If
ElseIf nd.Level = 1 '删除权限
    Dim dt As DataTable = DataTables("授权")
    Dim dr As DataRow
    Dim fz As String = nd.ParentNode.Name '获得分组名
    dr = dt.Find("分组 = '" & fz & "' And 权限 = '" & nd.Name & "'") '找出对应的行
    dr.Delete
    nd.Delete
End If
```

4、增加一个自定义函数，函数名为 ShouQuan，代码为：

```
If User.Type <> UserTypeEnum.User Then ' 如果是开发者或者管理员
    Return True '则具备所有权限
End If
Dim dt As DataTable = DataTables("授权")
Dim dr As DataRow
'首先判断分组的授权用户是否包括此用户或此用户所属的分组
dr = dt.Find("分组 = '" & args(0) & "' And 权限 Is Null")
If dr Is Nothing Then
        MessageBox.show("不存在名为'" & args(0) & "'分组!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return False
End If
If dr.IsNull("用户") = False Then
    Dim nms() As String = dr("用户").Split(",")
    For Each nm As String In nms
        If nm = User.Group OrElse nm = User.Name Then '如果授权用户包括登录用户所属的分组或其用户名
            Return True '返回 True
        End If
    Next
End If
'然后判断权限的授权用户是否包括此用户或此用户所属的分组
If Args(1) = "" Then
    Return False
End If
dr = dt.Find("分组 = '" & args(0) & "' And 权限 = '" & args(1) & "'")
If dr Is Nothing Then
```

```
        MessageBox.show("不存在名为"""  &  args(1)  &  "权限!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return False
End If
If dr.IsNull("用户") = False Then
    Dim nms() As String = dr("用户").Split(",")
    For Each nm As String In nms
        If nm = User.Group OrElse nm = User.Name Then '如果授权用户包括登录用户所属的分组或其用户名
            Return True '返回 True
        End If
    Next
End If
Return False
```

5、使用说明

实际使用的时候，要判断登录用户是否具备特定的权限是很简单的，例如判断用户是否具备批准订单的权限：

```
If Functions.Execute("ShouQuan","订单管理","批准订单") Then
    '用于批准订单的代码
End If
```

再例如判断用户是否具备生产管理分组的全部权限：

```
If Functions.Execute("ShouQuan","生产管理","") Then
    '代码
End If
```

4.17.2.20 单元格的密码保护

假定某列内容如果为空，则允许输入，如果不为空，则需要先双击此列，输入一个密码后，才能修改此列的内容。

设计步骤：

1、设置表的 PrePareEdit 事件代码为：

```
If e.Col.Name = "某列" Then   '如果编辑的是某列
    If e.Row.IsNull("某列") = False Then '而且某列已经有内容
      e.Cancel = True '则取消编辑
    End If
End If
```

这样一旦此列已经有内容，则无法以常规方式直接修改此列的内容。

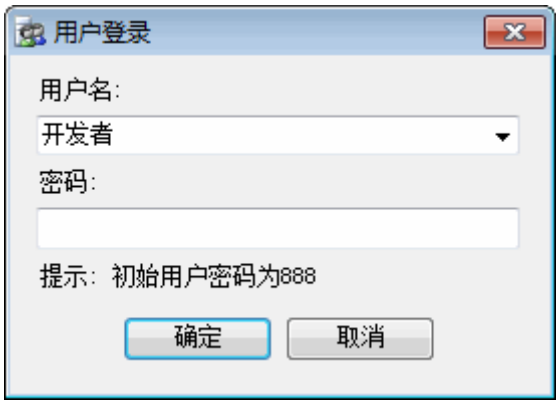
2、设置表的DoubleClick事件代码为：

```
If e.col.Name = "某列" Then '如果编辑的是某列
    Dim psd As String
```

```
    InputValue(psd,"验证","请输入密码") '输入密码
    If psd = "正确的密码" Then '如果密码正确
        e.Table.StartEditing '强行进入编辑状态
    Else
        Messagebox.Show("密码错误","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

4.17.2.21 用户登录窗口的自定义

Foxtable 内置了用户管理功能，默认的用户登录窗口很简洁：



但是你可以对这个窗口进行设置，设计出个性化的登录窗口。

我们先来测试一下，从 Foxtable 的安装目录下，复制文件 login.jpg 和 logform.txt 到你的项目文件夹中，然后重新打开项目，你会发现登录窗口为变为：



如何定义内置登录窗口

在 Foxtable 的安装目录下，有两个设置文件，分别是:

logform1.txt

logform2.txt

前者是无扩展用户属性的登录窗口默认设置，后者是有扩展用户属性的登录窗口默认设置。
当你需要对某个项目的登录窗口进行个性化设置的时候，根据需要选择上述文件中一个，复制到项目文件夹中，然后将将文件名改为:
logform.txt
然后打开这个文件，在这个文件的基础上调整相关设置即可。
当然如果你想偷懒，可以直接从 Foxtable 的安装目录下，复制文件 login.jpg 和 logform.txt 到你的项目文件夹中，然后换个个性化一点的图片即可。
下表是每项设置的详细说明，所有位置和尺寸都是以像素为单位：

格式	说明
Form@宽度@高度	设置登录窗口宽度和高度。 例如： Form@275@197 表示登录窗口的宽度为 275，高度为 197。
Label1@内容@水平位置@垂直位置	设置用户名标签的内容和位置。 例如： Label1@用户名:@13@11 表示标签内容为"用户名：",水平位置为 13，垂直位置为 11
TextBox1@水平位置@垂直位置@宽度	设置用户名输入框的位置和宽度。 例如： TextBox1@13@30@241 表示水平位置 13，垂直位置为 30，宽度为 241
Label2@内容@水平位置@垂直位置	设置密码标签的内容和位置。 例如： Label2@密码:@13@57 表示标签内容为"密码：",水平位置为 13，垂直位置为 57
TextBox2@水平位置@垂直位置@宽度	设置密码输入框的位置和宽度。 例如： TextBox2@13@76@241 表示水平位置 13，垂直位置为 76，宽度为 241
Label3@内容@水平位置@垂直位置	设置登录提示标签的内容和位置： 例如： Label3@提示:初始用户密码为 888@13@104 表示标签内容为"提示:初始用户密码为 888",水平位置为 13，垂直位置为 104
Button1@标题@水平位置@垂直位置@宽度@高度	设置确定按钮的标题、位置和尺寸。 例如： Button1@确定@56@138@75@23 表示标题为“确定”，水平位置为 56，垂直位置为 138，宽度为 75，高度为 23
Button2@标题@水平位置@垂直位置@宽度@高度	设置取消按钮的标题、位置和尺寸。 例如： Button2@取消@137@138@75@23

	表示标题为“取消”，水平位置为 131，垂直位置为 138，宽度为 75，高度为 23
FormBorder@True 或 FormBorder@False	设置登录窗口是否为无边框窗口。 FormBorder@True： 有边框 FormBorder@False：无边框
Image1@文件名	设置确定按钮的图片文件。 例如： Image1@确定.ico 表示确定按钮的图片来自于文件"确定.ico"，必须事先将此文件复制到项目目录中。
Image2@文件名	设置取消按钮的图片文件。 例如： Image2@取消.ico 表示取消按钮的图片来自于文件"取消.ico"，必须事先将此文件复制到项目目录中。
Image3@文件名	设置登陆窗口的背景图片。 例如： Image3@背景.jpg 表示窗口的背景图片自于文件"背景.jpg"，必须事先将此文件复制到项目目录中。
Grid@水平位置@垂直位置@宽度@高度	设置扩展用户属性表的位置和大小。 例如： Grid@15@130@240@93 表示水平位置为 15，垂直位置为 130，宽度为 240，高度为 93

如果要恢复默认的登录窗口，直接删除文件 logform.txt 即可。

4.17.2.22 外部数据源与用户管理

在默认情况下，Foxtable 的用户名、用户密码、用户分组、用户角色等数据保存在项目文件中。在多用户环境下，增加、删除、更改用户后，需要更新各终端的项目文件，这对于大型的应用系统，特别是远程应用的时候，是一件很麻烦的事情。

不过从 Foxtable 2013 开始，用户可以很方便地使用外部数据表存储用户信息，要做的只是单击几次鼠标。

步骤

1、首先要在外部数据库中新建一个数据表，用于保存用户信息，这个表必须包括三列，分别是：

列名	数据类型	说明
Name	字符型	必须将此列设置为主键！
Type	整数型	

Config	备注型	

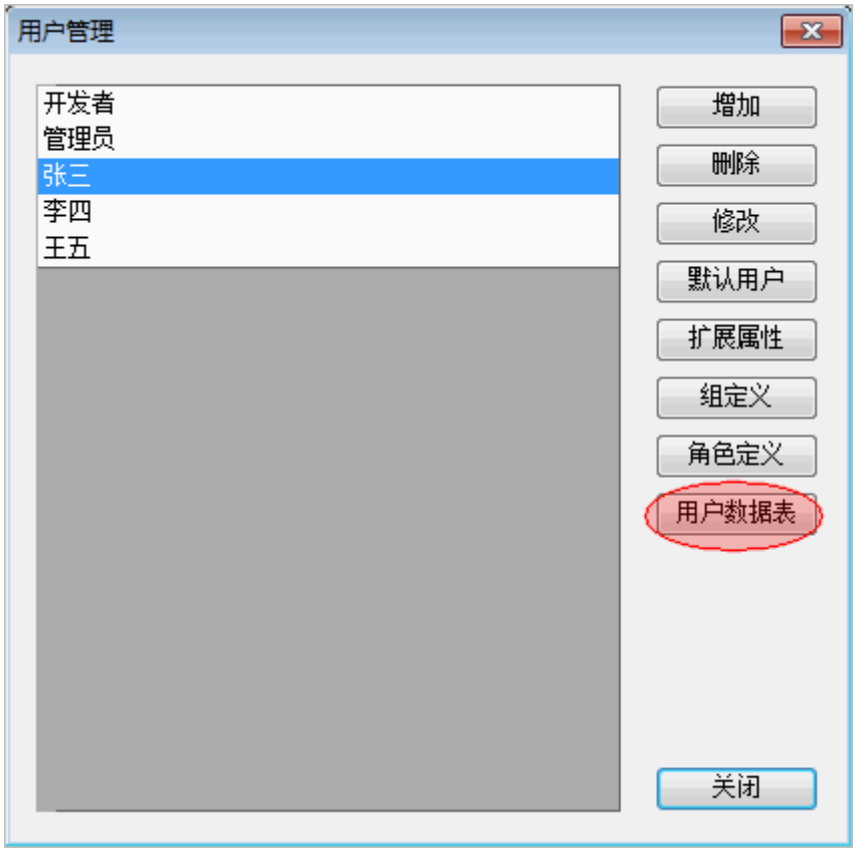
假定创建的这个表的名称为“Users”。

如果你直接在 SQL Server 中创建此表，注意要按下图所示设置列类型：

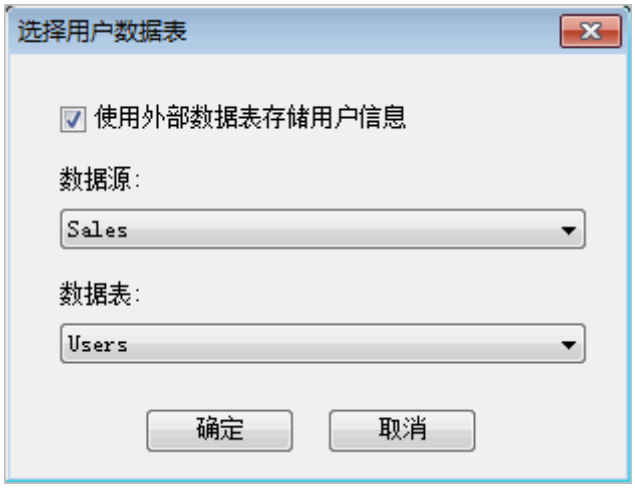


2、创建一个数据源，连接到这个数据库。

3、在用户管理中单击“用户数据表”按钮：



4、然后用于存储用户信息的外部数据源和外部数据表：



需要注意的是：

1、开发者和管理员的设置信息，始终保存在项目文件中，只有普通用户的设置信息，才会保存在外部表中，这样即使外部数据源无法连接，我们依然可以登录系统，重新调整设置。

2、项目文件中的用户设置信息，和外部数据表中的用户设置信息，是各自独立存在的，可以随时切换。

3、由于使用外部表存储用户信息时，是先连接数据源，再打开登录窗口，所以不能在BeforeConnectOuterDataSource 事件中使用扩展用户属性，否则会出错。

4.17.2.23 自定义用户与权限管理之一

一般用户请忽略本节内容。

上一节已经讲述如何将用户数据存储在外部数据表中。

如果你需要更大的灵活性(通常没有这个必要了)，也可以完全全抛开了 foxtable 自身的用户管理系统，自己重新设计一套用户管理系统。

本节内容完全采用 SQL 语言来实现，可以说是对 SQL 语言的一次大演练。

本节内容可以参考 CaseStudy 目录下的文件"自定义用户管理之一.Table"，用户信息保存在一个 Access 文件"用户.mdb"中。

设计步骤

一、在文件“用户.mdb”新建一个名为“Users”的表，包括 Name、Group、Password 三列，分别 用于保存用户名、用户分组和密码。

二、建立一个外部数据源，连接到文件“用户.mdb”，数据源的名称设置为“User”。

三、打开 Foxtable 的用户管理，增加一个用户，并将其设置为默认用户，这样打开项目的时候，将不会出现 foxtable 的用户登录窗口。

提示：示例文件“自定义用户管理.Table""菜单中的用户管理和用户登录，已经被自定义的用户管理和登录代替，必须先按 Ctrl+Shift+F12 回到系统菜单，才能打开 Foxtable 的用户管理。

四、在全局代码中定义两个变量：

```
Public _UserName As String
Public _UserGroup As String
```

_UserName 用户保存当前登录用户的名称，_UserGroup 用于保存该用户的分组。

五、新建一个窗口名为“用户管理”窗口：



窗口的 AfterLoad 事件代码为：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim lst As WinForm.ListBox = e.Form.Controls("ListBox1")
cmd.ConnectionName = "User"
cmd.CommandText = "SELECT DISTINCT Name From {Users}"
dt = cmd.ExecuteReader()
For Each dr As DataRow In dt.Datarows
    lst.Items.Add(dr("Name"))
Next
If Lst.Items.Count > 0 Then
    lst.SelectedIndex = 0
End If
```

"增加用户"按钮的代码为：

```
Forms("增加用户").Open()
```

“更改用户”按钮的代码为：

```
If e.Form.Controls("ListBox1").SelectedIndex >=0 Then
    Forms("更改用户").Open()
End If
```

"删除用户“按钮的代码为：

```
Dim lst As WinForm.ListBox = e.Form.Controls("ListBox1")
If lst.SelectedIndex >=0 Then
    Dim UserName As String = lst.SelectedItem
    Dim cmd As New SQLCommand
    cmd.ConnectionName = "User"
    cmd.CommandText = "Delete From {Users} Where [Name] = '" & UserName & "'"
```

```
    cmd.ExecuteNonQuery
    lst.Items.RemoveAt(lst.SelectedIndex)
End If
```

六、新建一个名为"增加用户"的窗口:



"确定"按钮的代码设置为:

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim UserGroup As String = e.Form.Controls("UserGroup").Value
Dim PassWord As String = e.Form.Controls("PassWord").Value
Dim cmd As New SQLCommand
cmd.ConnectionName = "User"
If UserName = "" OrElse UserGroup = "" Then
        Messagebox.show("请输入用户名和用户分组!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Select Count(*) From {Users} Where Name = '" & UserName & "'"
If cmd.ExecuteScalar > 0 Then
      Messagebox.show("增加用户失败,已经能存在同名用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Insert Into {Users} ([Name],[Group],[Password]) Values ('"
cmd.CommandText = cmd.CommandText & UserName & "','" & UserGroup & "','" & Password & "')"
If cmd.ExecuteNonQuery = 1 Then '返回 1 表示增加成功
    If Forms("用户管理").Opened Then
        With Forms("用户管理").Controls("ListBox1")
            .Items.Add(UserName)
            .SelectedIndex = .Items.Count -1
        End With
    End If
    e.Form.Close()
End If
```

七、新建一个名为“更改用户”的窗口：



窗口的 AfterLoad 事件代码为:

```
Dim UserName As String
Dim UserGroup As String
Dim PassWord As String
Dim cmd As New SQLCommand
Dim dt As DataTable
If Forms("用户管理").Opened Then
    UserName = Forms("用户管理").Controls("ListBox1").SelectedItem
Else
    UserName = _UserName '_UserName 为全局变量,用于保存当前用户名
    e.Form.Controls("UserName").Enabled = False
    e.Form.Controls("UserGroup").Enabled = False
End If
e.Form.Text = e.Form.Text & "-" & UserName
cmd.ConnectionName = "User"
cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
dt = cmd.ExecuteReader
If dt.DataRows.Count = 1 Then
    e.Form.Controls("UserName").Value = UserName
    e.Form.Controls("UserGroup").Value = dt.DataRows(0)("Group")
    e.Form.Controls("PassWord").Value = dt.DataRows(0)("PassWord")
Else
    MessageBox.Show("用户不存在或者已经被删除!","提示",MessageBoxButtons.OK)
    e.Form.Close()
End If
```

上面的代码在打开“更改用户”窗口之前，先判断“用户管理”窗口是否已经打开，如果已经打开，则更改“用户管理”窗口中选定的用户；否则更改当前登录的用户，且禁止更改用户名和用户分组，只能更改用户密码。

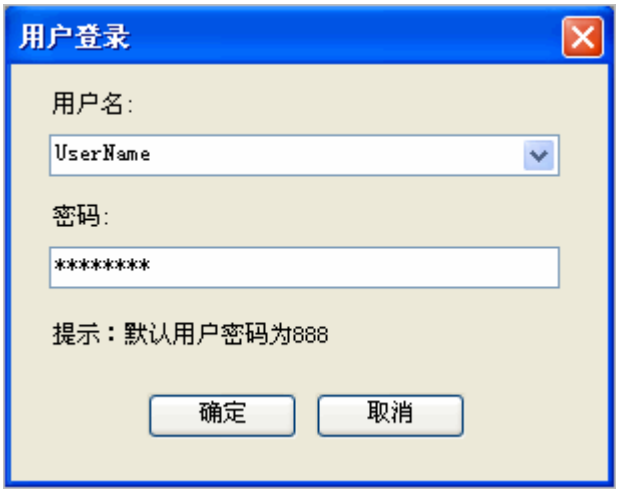
这样当我们需要给登录用户提供更改密码功能的时候，只需增加一个菜单按钮，按钮代码设置为：

```
Forms("更改用户").Open()
```

"确定"按钮的代码为：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim UserGroup As String = e.Form.Controls("UserGroup").Value
Dim PassWord As String = e.Form.Controls("PassWord").Value
Dim OldUserName As String
Dim cmd As New SQLCommand
Dim Parts() As String = e.Form.Text.Split("-")
OldUserName = Parts(Parts.Length -1)
cmd.ConnectionName = "User"
If UserName = "" OrElse UserGroup = "" Then
        Messagebox.show(" 请 输 入 用 户 名 和 用 户 分 组 !"," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Update {Users} Set [Name] = '" & UserName & "',[Group] = '" & UserGroup
cmd.CommandText = cmd.CommandText & "', [Password] = '" & Password & "' Where [Name]= '" & OldUserName & "'"
If cmd.ExecuteNonQuery = 1 Then '返回 1 表示更改成功
    If Forms("用户管理").Opened Then
        With Forms("用户管理").Controls("ListBox1")
            .Items(.SelectedIndex) = UserName
        End With
    End If
    e.Form.Close
Else
    Messagebox.show("更改用户失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

八、增加一个名为“用户登录”的窗口：



窗口的 AfterLoad 事件代码为：

```
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim cmb As WinForm.ComboBox = e.Form.Controls("UserName")
cmd.ConnectionName = "User"
```

```
cmd.CommandText = "SELECT DISTINCT Name From {Users}"
dt = cmd.ExecuteReader()
For Each dr As DataRow In dt.Datarows
    cmb.Items.Add(dr("Name"))
Next
cmb.SelectedIndex = 0
```

"确定"按钮的代码为：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim dr As DataRow
cmd.ConnectionName = "User"
If UserName = ""   Then
    Messagebox.show("请选择用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
dt = cmd.ExecuteReader
If dt.DataRows.Count = 0 Then
    Messagebox.show("此用户不存在!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
dr = dt.DataRows(0)
If e.Form.Controls("PassWord").Value = dr("Password") Then
    _UserName = UserName
    _UserGroup = dr("Group")
    e.Form.Close
Else
    Messagebox.show("密码错误!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

“取消”按钮的代码为：

```
e.Form.Close()
```

为了防止用户跳过确定和取消按钮，直接关闭窗口，请将窗口的“显示控制框”属性设置为 False。

九、在项目事件 AfterOpenProject 中加入代码：

```
Forms("用户登录").Open()
If _UserName = "" Then
    Syscmd.Project.Exit()
    Return '这个要加上，因为既然要退出，就不应该再执行后面的初始化代码了，否则会出错
End If
'后续初始化代码
```

提示：为确保用户登录窗口为第一个打开的窗口，登录窗口的窗口类型必须设置为“模式”，且应该将其他所有独立或模式窗口的“自动打开”属性设置为 False。

十、最后修改菜单中的"用户管理"和"切换用户"按钮的代码，启用"用户管理"按钮的代码为：

Forms("用户管理").Open()
"切换用户"按钮的代码为：
Forms("用户登录").Open()
你可以先应用系统菜单“system.ribbon”，然后再修改。
如果需要，还可以增加一个更改密码的按钮，代码设置为：
Forms("更改用户").Open()

总结

经过上述 10 个步骤的设计，我们已经建立了自己的用户管理和登录系统，用户登录后，用户名保存在全局变量_User 中，用户分组保存在全局变量_UserGroup 中。
我们只需对原来的权限代码稍作修改，即可适用自定义的用户管理系统，例如原来的代码为：

```
If User.Group = "经理" Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

现在只需改为：

```
If _UserGroup = "经理" Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

可以看出，自定义的用户管理系统，在应用上和内置的不会有任何差别。
不过使用自定义用户管理系统的时候，需要注意 事件触发顺序，由于自定义的用户登录窗口是在 AfterOpenProject 事件中打开，意味着在此事件之前触发执行的事件，是不能使用_UserName 和 _UserGroup 变量的。
例如有的用户希望使用动态加载功能，在用户登录后，订单表只加载业务员为登录用户的订单，所有在项目事件 BeforeLoadInnerTable 设置了代码：

```
If e.DataTableName = "订单" Then
    e.Filter = "业务员 = '" & _UserName & "'"
End If
```

结果重新打开项目后，发现订单表根本没有加载任何数据，如果在代码中用 MessageBox 显示一下合成的加载条件：

```
If e.DataTableName = "订单" Then
    e.Filter = "业务员 = '" & _UserName & "'"
    MessageBox.Show(e.Filter)
End If
```

会发现合成的加载条件为：

业务员 = ''

为什么会这样呢？
这是因为 BeforeLoadInnerTable 是在 AfterOpenProject 事件之前触发执行，此时_UserName 还是空值。

显然 要实现我们的目标，得使用新的设计方法：

1、首先在 BeforeLoadInnerTable 事件加入下面的代码，确保订单表初始状态不会加载任何数据：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

2、然后修改用户登录窗口的“确定”按钮的代码，在后面加上加载数据的代码：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim dr As DataRow
cmd.ConnectionName = "User"
If UserName = "" Then
    Messagebox.show("请选择用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
dt = cmd.ExecuteReader
dr = dt.DataRows(0)
If e.Form.Controls("PassWord").Value = dr("Password") Then
    _UserName = UserName
    _UserGroup = dr("Group")
    e.Form.Close
Else
    Messagebox.show("密码错误!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
DataTables("订单").LoadFilter = "业务员 = '" & _UserName & "'"
DataTables("订单").Load()
```

上面代码的最后两行，就是用于根据登录用户名动态加载订单表数据的：

```
DataTables("订单").LoadFilter = "业务员 = '" & _UserName & "'"
DataTables("订单").Load()
```

以下几个事件在 AfterOpenProject 事件之前触发执行：

BeforeOpenProject

BeforeConnectOuterDataSource

BeforeLoadInnerTable

BeforeLoadOuterTable

Initialize

另外 由于设置了默认用户，绕开了内置的登录窗口，而且菜单中的登录命令已经改为自定义的登录命令，所以用户将无法以开发者和管理员身份登录系统。

如果希望以开发者或管理员身份登录系统，可以在打开项目的过程中始终按住 Ctrl 键。

也可以考虑在菜单中另外加入一个系统登录命令，代码设置为：Syscmd.Project.Switchuser()

最后自定义用户登录显然不会触发 LoadUserSetting 事件，所以该事件中的代码应该移植到自定义用户登录窗口的"确定"按钮中。

4.17.2.24 自定义用户与权限管理之二

一般用户请忽略本节内容。
上一节已经 讲述了如何自己设计用户管理系统，由于完全使用 SQL 语句来实现，对于一般的用户来说，可能有些困难。
本节实现完全一模一样的功能，但是基本上不使用 SQL 语句，而是采用 Foxtable 自己的后台数据处理函数，所以更简单，更适合一般用户掌握。
还是要申明一下，因为内置的用户数据可以存储在外部数据表中，所以多数情况下并没有必要自定义用户管理，只是通过这两个例子，让大家可以演练一下 SQL 语句和后台数据处理函数的使用方法。
本节内容可以参考 CaseStudy 目录下的文件"自定义用户管理之二.Table"，用户信息保存在一个 Access 文件"用户.mdb"中。
设计步骤
一、在文件“用户.mdb”新建一个名为“Users”的表，包括 Name、Group、Password 三列，分别用于保存用户名、用户分组和密码 ，其中 Name 列设置为主键。
二、建立一个外部数据源，连接到文件“用户.mdb”，数据源的名称设置为“User”。
三、打开 Foxtable 的用户管理，增加一个用户，并将其设置为默认用户，这样打开项目的时候，将不会出现 foxtable 的用户登录窗口。
提示：示例文件“自定义用户管理.Table""菜单中的用户管理和用户登录，已经被自定义的用户管理和登录代替，必须先按 Ctrl+Shift+F12 回到系统菜单，才能打开 Foxtable 的用户管理。
四、在全局代码中定义两个变量：

```
Public _UserName As String '用户名
Public _UserGroup As String '用户分组
Public _UserTable As DataTable '用户表
```

_UserName 用户保存当前登录用户的名称，_UserGroup 用于保存该用户的分组，_UserTable 是一个临时表，后面将基于此表读写后台的用户表。
五、在项目事件 AfterOpenProject 加上代码：

```
Dim cmd As New SQLCommand
cmd.ConnectionName = "User"
cmd.CommandText = "SELECT * From {Users} Where [Name] Is Null"
_UserTable = cmd.ExecuteReader(True)
```

上面的代码基于后台的用户表生成一个临时表，这个临时表没有加载数据，后面将基于此表和后台的用户表进行交互。
注意这里要将 ExecuteReader 的参数设置为 True，因为我们需要增加、删除和修改用户 ，也就是说，不仅要读用户表，还要写用户表。
六、新建一个窗口名为“用户管理”窗口：



窗口的 AfterLoad 事件代码为：

```
Dim lst As WinForm.ListBox = e.Form.Controls("ListBox1")
lst.ComboList = _UserTable.SQLGetComboListString("Name")
If Lst.Items.Count > 0 Then
    lst.SelectedIndex = 0
End If
```

"增加用户"按钮的代码为：

```
Forms("增加用户").Open()
```

"更改用户"按钮的代码为：

```
If e.Form.Controls("ListBox1").SelectedIndex >=0 Then
    Forms("更改用户").Open()
End If
```

"删除用户“按钮的代码为：

```
Dim lst As WinForm.ListBox = e.Form.Controls("ListBox1")
If lst.SelectedIndex >=0 Then
    Dim UserName As String = lst.SelectedItem
    _UserTable.SQLDeleteFor("[Name] = '" & UserName & "'")
    lst.Items.RemoveAt(lst.SelectedIndex)
    lst.Select()
End If
```

七、新建一个名为"增加用户"的窗口：



"确定"按钮的代码设置为：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim UserGroup As String  = e.Form.Controls("UserGroup").Value
Dim PassWord As String = e.Form.Controls("PassWord").Value
Dim dr As DataRow
If UserName = "" OrElse UserGroup = "" Then
      Messagebox.show("请输入用户名和用户分组!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
   Return
End If
If _UserTable.SQLFind("[Name] = '" & UserName & "'") IsNot Nothing Then
     Messagebox.show("增加用户失败，已经能存在同名用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
   Return
End If
dr = _UserTable.AddNew()
dr("Name") = UserName
dr("Group") = UserGroup
dr("Password") = PassWord
_UserTable.Save()
e.Form.Close()
If Forms("用户管理").Opened Then
   With Forms("用户管理").Controls("ListBox1")
      .Items.Add(UserName)
      .SelectedIndex = .Items.Count -1
   End With
End If
```

八、新建一个名为“更改用户”的窗口：



窗口的 AfterLoad 事件代码为:

```
Dim UserName As String
Dim UserGroup As String
Dim PassWord As String
Dim dr As DataRow
If Forms("用户管理").Opened Then '正常打开
    UserName = Forms("用户管理").Controls("ListBox1").SelectedItem
Else
   UserName = _UserName '_UserName为全局变量,用于保存当前用户名
   e.Form.Controls("UserName").Enabled = False
   e.Form.Controls("UserGroup").Enabled = False
End If
e.Form.Text = e.Form.Text & "-" & UserName '将要更改的用户名保存在窗口标题中,方便其他事件调用
dr = _UserTable.SQLFind("[Name] = '" & UserName & "'")
If dr IsNot Nothing Then
    e.Form.Controls("UserName").Value = UserName
    e.Form.Controls("UserGroup").Value = dr("Group")
    e.Form.Controls("PassWord").Value = dr("PassWord")
Else
    MessageBox.Show("用户不存在或者已经被删除!","提示",MessageBoxButtons.OK)
    e.Form.Close()
End If
```

上面的代码在打开“更改用户”窗口之前，先判断“用户管理”窗口是否已经打开，如果已经打开，则更改“用户管理”窗口中选定的用户；否则更改当前登录的用户，且禁止更改用户名和用户分组，只能更改用户密码。

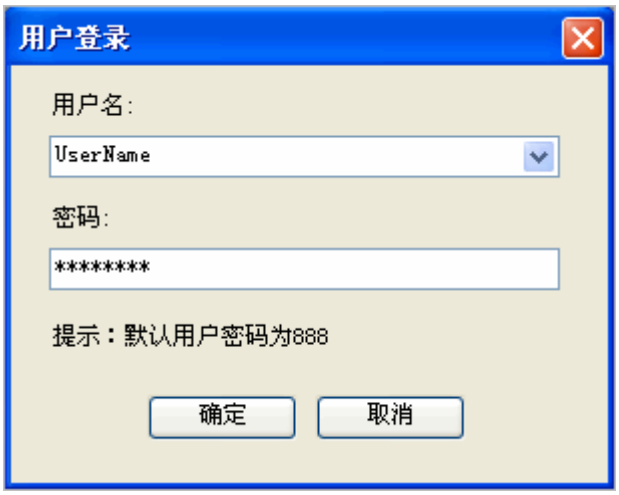
这样当我们需要给登录用户提供更改密码功能的时候，只需增加一个菜单按钮，按钮代码设置为：

```
Forms("更改用户").Open()
```

"确定"按钮的代码为：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim UserGroup As String  = e.Form.Controls("UserGroup").Value
Dim PassWord As String = e.Form.Controls("PassWord").Value
Dim OldUserName As String
Dim Parts() As String = e.Form.Text.Split("-"c)
OldUserName =  Parts(Parts.Length -1)
If UserName = "" OrElse UserGroup = "" Then
        Messagebox.show("请输入用户名和用户分组!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
Dim dr As DataRow = _UserTable.SQLFind("[Name]= '" & OldUserName & "'")
If dr IsNot Nothing Then
    dr("Name") = UserName
    dr("Group") = UserGroup
    dr("Password") = PassWord
    dr.Save()
    If Forms("用户管理").Opened Then
      With Forms("用户管理").Controls("ListBox1")
        .Items(.SelectedIndex) = UserName
      End With
    End If
    e.Form.Close
Else
    Messagebox.show("更改用户失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

九、增加一个名为“用户登录”的窗口：



窗口的 AfterLoad 事件代码为：

```
Dim cmb As WinForm.ComboBox = e.Form.Controls("UserName")
cmb.ComboList = _UserTable.SQLGetComboListString("Name")
```

"确定"按钮的代码为：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim dr As DataRow
If UserName = ""  Then
    Messagebox.show("请选择用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
dr = _UserTable.SQLFind("[Name] = '" & UserName & "'")
If dr Is Nothing Then
    Messagebox.show("此用户不存在!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
If e.Form.Controls("PassWord").Value = dr("Password") Then
    _UserName = UserName
    _UserGroup = dr("Group")
    e.Form.Close
Else
    Messagebox.show("密码错误!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

为了防止用户跳过确定和取消按钮，直接关闭窗口，请将窗口的“显示控制框”属性设置为 False。
十、将项目事件 AfterOpenProject 的代码改为：

```
Dim cmd As New SQLCommand
cmd.ConnectionName = "User"
cmd.CommandText = "SELECT * From {Users} Where [Name] Is Null"
_UserTable = cmd.ExecuteReader(True)

Forms("用户登录").Open()
If _UserName = "" Then
    Syscmd.Project.Exit()
    Return '这个要加上,因为既然要退出,就不应该再执行后面的初始化代码了,否则会出错
End If
'后续初始化代码
```

第一段是之前已经写好的代码，第二段是新加上去的，为的是打开项目后能出现登录窗口。
为确保用户登录窗口为第一个打开的窗口，登录窗口的窗口类型必须设置为“模式”，且应该将其他所有独立或模式窗口的“自动打开”属性设置为 False。
十一、最后修改菜单中的"用户管理"和"切换用户"按钮的代码，启用"用户管理"按钮的代码为：
Forms("用户管理").Open()
"切换用户"按钮的代码为：
Forms("用户登录").Open()
你可以先应用系统菜单“system.ribbon”，然后再修改。
如果需要，还可以增加一个更改密码的按钮，代码设置为：

```
Forms("更改用户").Open()
```

总结

经过上述 11 个步骤的设计，我们已经建立了自己的用户管理和登录系统，用户登录后，用户名保存在全局变量_User 中，用户分组保存在全局变量_UserGroup 中。

我们只需对原来的权限代码稍作修改，即可适用自定义的用户管理系统，例如原来的代码为：

```
If User.Group = "经理" Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

现在只需改为：

```
If _UserGroup = "经理" Then
    DataTables("表 A").AllowEdit = True
Else
    DataTables("表 A").AllowEdit = False
End If
```

可以看出，自定义的用户管理系统，在应用上和内置的不会有任何差别。

不过使用自定义用户管理系统的时候，需要注意 事件触发顺序，由于自定义的用户登录窗口是在 AfterOpenProject 事件中打开，意味着在此事件之前触发执行的事件，是不能使用_UserName 和 _UserGroup 变量的。

例如有的用户希望使用动态加载功能，在用户登录后，订单表只加载业务员为登录用户的订单，所有在项目事件 BeforeLoadInnerTable 设置了代码：

```
If e.DataTableName = "订单" Then
    e.Filter = "业务员 = '" & _UserName & "'"
End If
```

结果重新打开项目后，发现订单表根本没有加载任何数据，如果在代码中用 MessageBox 显示一下合成的加载条件：

```
If e.DataTableName = "订单" Then
    e.Filter = "业务员 = '" & _UserName & "'"
    MessageBox.Show(e.Filter)
End If
```

会发现合成的加载条件为：

```
业务员 = '
```

为什么会这样呢？

这是因为 BeforeLoadInnerTable 是在 AfterOpenProject 事件之前触发执行，此时_UserName 还是空值。

显然 要实现我们的目标，得使用新的设计方法：

1、首先在 BeforeLoadInnerTable 事件加入下面的代码，确保订单表初始状态不会加载任何数据：

```
If e.DataTableName = "订单" Then
    e.Filter = "[_Identify] Is Null"
End If
```

2、然后修改用户登录窗口的“确定”按钮的代码，在后面加上加载数据的代码：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim dr As DataRow
cmd.ConnectionName = "User"
If UserName = "" Then
    Messagebox.show("请选择用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
dt = cmd.ExecuteReader
dr = dt.DataRows(0)
If e.Form.Controls("PassWord").Value = dr("Password") Then
    _UserName = UserName
    _UserGroup = dr("Group")
    e.Form.Close
Else
    Messagebox.show("密码错误!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
DataTables("订单").LoadFilter = "业务员 = '" & _UserName & "'"
DataTables("订单").Load()
```

上面代码的最后两行，就是用于根据登录用户名动态加载订单表数据的：

```
DataTables("订单").LoadFilter = "业务员 = '" & _UserName & "'"
DataTables("订单").Load()
```

以下几个事件在 AfterOpenProject 事件之前触发执行：

BeforeOpenProject

BeforeConnectOuterDataSource

BeforeLoadInnerTable

BeforeLoadOuterTable

Initialize

另外 由于设置了默认用户，绕开了内置的登录窗口，而且菜单中的登录命令已经改为自定义的登录命令，所以用户将无法以开发者和管理员身份登录系统。

如果希望以开发者或管理员身份登录系统，可以在打开项目的过程中始终按住 Ctrl 键。

也可以考虑在菜单中另外加入一个系统登录命令，代码设置为：Syscmd.Project.Switchuser()

最后自定义用户登录显然不会触发 LoadUserSetting 事件，所以该事件中的代码应该移植到自定义用户登录窗口的"确定"按钮中。

4.17.3 文件管理权限

4.17.3.1 FTP 与权限管理

本章内容仅对高级开发版有效。

在网络环境下，Foxtable 通过 FTP 实现了文件的共享和管理，并提供了一个 FTP 客户端，用于完成日常的文件管理工作。
在 Foxtable 2018 之前，FTP 客户端是没有办法设置权限的，用户打开 FTP 客户端之后，可下载、上传和删除任何文件，甚至删除整个目录。

虽然为 FTPClient 提供了一些属性，用于禁用上传、下载、删除等按钮，但是这些属性是一刀切性质，无法满足动态权限管理的需求。

从 Foxtable 2018 开始，我们可以在某用户进入 FTP 客户端之后，精确控制其 能查看哪些文件，能下载哪些文件，能删除哪些文件，以及能访问哪些目录，能删除哪些目录，能否创建目录等等，总之 FTP 客户端中的所有操作都可以进行权限设置。
提示：如果你不是在互联网环境下，而是在局域网环境下使用 Foxtable，我们依然建议用 FTP 而不是共享文件夹来实现文件管理，因为这样可以精确控制用户权限。
Foxtable 提供了以下事件，用于对 FTP 客户端进行权限设置：

- **BeforeEnterFolder**
  在进入某个目录之前执行，通过此事件可以禁止用户访问某些目录。

- **BeforeShowDetail**
  在列出文件和目录之前执行，通过此事件可以隐藏某些目录和文件。

- **BeforeDelete**
  在删除某个文件或目录之前执行，通过此事件可以禁止用户删除特定的文件或目录。

- **AfterDelete**
  在删除某个文件或目录之后执行，通过此事件可以记录用户删除了哪些文件和目录。

- **BeforeRename**
  在重命名文件或目录之前执行，通过此事件可以可以禁止用户重命特定的文件或目录。

- **AfterRename**
  在重命名文件或目录之后执行，通过此事件可以可以记录用户重命名了哪些文件和目录。

- **BeforeUpload**
  在上传文件之前执行，通过此事件可以完全禁止用户上传文件，或禁止用户只能上传特定类型的文件，或只能将文件上传到指定的目录。

- **AfterUpload**
  在文件成功上传之后执行，通过此事件可以记录用户上传了哪些文件。

- **BeforeDownload**
  在下载文件之前执行，通过此事件可以完全禁止用户下载文件，或禁止用户只能下载特定类型的文件，或禁止用户下载特定目录中的文件。

- **AfterDownload**
  在下载文件之后执行，通过此事件可以记录用户下载了哪些文件。

- **BeforeCreateFolder**
  在创建目录之前执行，通过此事件可以禁止某些用户创建目录。

- **AfterCreateFolder**
  在创建目录之后执行，通过此事件可以记录用户创建了哪些目录。

在菜单的“管理项目”功能区，单击“网络监视器”按钮，可以为上述事件编写代码。

4.17.3.2 BeforeEnterFolder

在进入某个目录之前执行，通过此事件可以禁止用户访问某些目录。

e 参数属性：

FolderName	字符型，要进入的目录名称。
Cancel	逻辑型，如果设置为 True，将禁止用户进入此目录。
FTPName	字符型，返回 FTP 的 FTPName 属性

示例一

假定根目录下有个名为“报价单”的目录，希望禁止非销售分组的用户进入此目录，可以将 BeforeEnterFolder 事件代码设置为：

```
If e.FolderName = "/报价单" Then
    If User.Group <> "销售" Then
        e.Cancel = True
            MessageBox.Show("你无访问此目录的权限!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

示例二

假定每个用户只能访问自己的目录，同时有一个名为“公共”的目录，所有用户都能访问；此外如果用户分组为经理，则能访问所有用户的目录。

首先我们在 FTP 的根目录下创建一个名为“公共”的目录；之后为每个用户单独创建一个目录，目录名为用户名，如“张三”、“李四”。

最后将 BeforeEnterFolder 事件代码设置为：

```
If e.FolderName = "/" OrElse e.FolderName = "/公共" OrElse e.FolderName = "/" & User.Name Then
ElseIf User.Group = "经理" Then
Else
```

```
    e.Cancel = True
        MessageBox.Show("你无权访问他人目录!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

提示：

我们也可以利用BeforeListDetails事件，只显示用户有权访问的目录和文件，断了用户非法访问的念头。

4.17.3.3 BeforeListDetails

在列出文件和目录之前执行，通过此事件可以隐藏部分目录和文件。

e 参数属性：

CurFolder	字符型，FTP 的当前目录，文件将被上传到此目录。
Infos	集合，包括即将显示的文件和目录信息。 Infos 的元素是字符串数组，每个数组包括三个元素。 对于文件，三个元素分别问：文件名（不含路径），最近一次修改时间，文件大小。 对于目录，三个元素分别为：目录名（不含路径），最近一次修改时间、字符"D"，用于表示这是一个目录。
FTPName	字符型，返回 FTP 的 FTPName 属性

e 参数方法：

Remove 将某个文件或目录从显示内容中排除掉。

示例一

假定对于非经理分组的用户，要求：

1、隐藏“归档”目录。

2、隐藏后缀名为"rar"的文件。

3、隐藏"/VIP"目录下的"客户名单.xls"文件。

为此可以将 BeforeListDetails 事件代码设置为：

```
If User.Group <> "经理" Then
    For Each Info() As String In e.Infos
        If Info(0) = "归档" AndAlso Info(2) = "D" Then '隐藏归档目录
            e.Remove(Info)
        End If
        If Right(Info(0),4) = ".rar" Then '隐藏后缀名为rar的文件
            e.Remove(Info)
        End If
        If  Info(0) = "客户名单.xls" AndAlso e.CurFolder = "/VIP" Then '隐藏VIP目录下的客户名单.xls
            e.Remove(Info)
        End If
    Next
```

```
End If
```

示例二

假定希望隐藏"报价单"目录下修改日期为3个月之前的文件，可以将BeforeListDetails事件代码设置为：

```
If e.CurFolder = "报价单"  Then
    For Each Info() As String In e.Infos
        Dim md As Date = CDate(Info(1)) '第二个元素虽然存储的是日期,但类型是字符串,需要先转换为日期.
        If md.AddMonths(3) < Date.Today Then
            e.Remove(Info)
        End If
    Next
End If
```

4.17.3.4 BeforeDelete

在删除某个文件或目录之前执行，通过此事件可以禁止用户删除特定的文件或目录。

e 参数属性

IsFolder	逻辑型，如果被删除的是目录，则此属性返回 True。
FileName	字符型，要删除的文件名或目录名，含完整路径。
ModifyTime	日期型，返回此文件或目录最近一次修改的日期。
Cancel	逻辑型，如果设置为 True，将取消此次删除操作。
CancelAll	逻辑型，如果设置为 True，将取消本次及后续删除操作。
FTPName	字符型，返回 FTP 的 FTPName 属性

示例一

假定根目录下有个名为“归档”的子目录，要禁止删除此目录下的任何文件，只需将 BeforeDelete 事件代码设置为：

```
If e.IsFolder AndAlso e.FileName = "/归档" Then '首先禁止删除归档目录本身
    e.Cancel = True
ElseIf Left(e.FileName,4) =  "/归档/" Then '禁止删除归档目录下的文件或子目录
    e.Cancel = True
End If
If e.Cancel Then
     MessageBox.Show(" 禁 止 删 除 归 档 目 录 下 的 内 容 ",  " 提 示 ",
MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

示例二

如果用户在归档目录下选择了 100 个文件进行删除，那么上述代码会弹出 100 次提示，用户可能会感到愤怒崩溃.

要解决这个问题很简单，只需将 CancelAll 设置为 True：

```
If e.IsFolder AndAlso e.FileName = "/归档" Then '首先禁止删除归档目录
    e.CancelAll = True
ElseIf Left(e.FileName,4) =  "/归档/" Then '禁止删除归档目录下的文件或子目录
    e.CancelAll = True
End If
If e.CancelAll Then
      MessageBox.Show("禁止删除归档目录下的内容", "提示", MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

示例三

如果要禁止删除三个月之前的文件，可以将BeforeDelete代码设置为：

```
Dim tp As Date =  e.ModifyTime.AddMonths(3)
If tp < Date.Today Then
    e.Cancel = True
     MessageBox.Show("禁止删除三个月以前的文件!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

4.17.3.5 AfterDelete

在删除某个文件或目录之后执行，通过此事件可以记录用户删除了哪些文件和目录。

e 参数属性

FileName	字符型，要删除的文件名或目录名，含完整路径。
ModifyTime	日期型，返回文件最近一次修改的日期。
IsFolder	逻辑型，如果被删除的是目录，则此属性返回 True。
FTPName	字符型，返回 FTP 的 FTPName 属性

4.17.3.6 BeforeRename

在重命名文件或目录之前执行，通过此事件可以可以禁止用户重命特定的文件或目录。

e 属性属性：

IsFolder	逻辑型，如果要重命名的是目录，则此属性返回 True。
OldName	字符型，原文件名或目录名。

NewName	字符型，新文件名或目录名。
ModifyTime	日期型，返回此文件或目录最近一次修改的日期。
Cancel	逻辑型，如果设置为 True，将取消此次删除操作。
FTPName	字符型，返回 FTP 的 FTPName 属性

示例一

假定根目录下有个名为“归档”的子目录，要求禁止重命名此目录下的任何文件或目录，可以将 BeforeRename 事件代码设置为：

```
If e.IsFolder AndAlso e.OldName = "/归档" Then '首先禁止重命名归档目录本身
    e.Cancel = True
ElseIf Left(e.OldName,4) =  "/归档/" Then '禁止重命名归档目录下的文件和子目录
    e.Cancel = True
End If
If e.Cancel Then
      MessageBox.Show(" 禁 止  重 命 名 归 档 目 录 下 的 内 容 ",  " 提 示 ", MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

示例二

如果要禁止重命名三个月之前的文件，可以将BeforeRename事件代码设置为：

```
Dim tp As Date =  e.ModifyTime.AddMonths(3)
If tp < Date.Today Then
    e.Cancel = True
    MessageBox.Show(" 禁 止  重 命 名 三 个 月 以 前 的 文 件 !",  " 提 示 ",  MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

4.17.3.7 AfterRename

在重命名文件或目录之后执行，通过此事件可以可以记录用户重命名了哪些文件和目录。

e 属性属性：

IsFolder	逻辑型，如果要重命名的是目录，则此属性返回 True。
OldName	字符型，原文件名或目录名。
NewName	字符型，新文件名或目录名。
ModifyTime	日期型，返回此文件或目录最近一次修改的日期。
FTPName	字符型，返回 FTP 的 FTPName 属性

4.17.3.8 BeforeDownload

在下载文件之前执行，通过此事件可以禁止用户下载某些文件。
e 属性属性：

File	字符型，要下载的文件名，含路径。
Cancel	逻辑型，设置为 True 取消本次下载操作。
CancelAll	逻辑型，设置为 True 取消本次及后续下载操作。
ModifyTime	日期型，返回此文件或目录最近一次修改的日期(对列窗口无效)。
FTPName	字符型，返回 FTP 的 FTPName 属性

示例
假定根目录下有个名为“报价单”的目录，希望禁止非销售分组的用户下载此目录下的文件，可将 BeforeDownLoad 事件代码设置为：

```
If Left(e.File,5) = "/报价单/" Then
    If User.Group <> "销售" Then
        e.Cancel = True
            MessageBox.Show("你无权下载报价文件!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

示例二

如果用户在报价单目录下选择了 100 个文件下载，那么上述代码会弹出 100 次提示，用户可能会感到愤怒崩溃.
要解决这个问题很简单，只需将 CancelAll 设置为 True：

```
If Left(e.File,5) = "/报价单/" Then
    If User.Group <> "销售" Then
        e.CancelAll = True
            MessageBox.Show("你无权下载报价文件!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

提示：

我们也可以利用BeforeListDetails事件，只显示用户有权下载的文件，断了用户非法下载的念头。

4.17.3.9 AfterDownload

在下载文件之后执行，通过此事件可以记录用户下载了哪些文件。
e 属性属性：

File	字符型，要下载的文件名，含路径。
ModifyTime	日期型，返回此文件最近一次修改的日期(对列窗口无效)。
FTPName	字符型，返回 FTP 的 FTPName 属性

4.17.3.10 BeforeUpload

在上传文件之前执行，通过此事件可以完全禁止用户上传文件，或允许用户只能上传特定类型的文件，或只能将文件上传到指定的目录。

e 属性属性：

LocalFile	字符型，要上传的本地文件，含路径。
CurFolder	字符型，FTP 的当前目录，文件将被上传到此目录。
Cancel	逻辑型，设置为 True 将取消本次上传操作。
CancelAll	逻辑型，设置为 True 取消本次及后续上传操作。
FTPName	字符型，返回 FTP 的 FTPName 属性

示例

假定 FTP 根目录下有个名为"归档"的子目录，要求只有经理级别的用户才能上传文件到此目录，为此可以将 BeforeUpload 事件代码设置为：

```
If  e.CurFolder = "/归档" OrElse Left(e.CurFolder, 4) = "/归档/" Then
    If User.Group <> "经理" Then
        e.Cancelall= True
        MessageBox.Show(" 只 能 经 理 级 别 的 用 户 才 能 归 档 文 件 !"," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

4.17.3.11 AfterUpload

在文件成功上传之后执行，通过此事件可以记录用户上传了哪些文件。

e 属性属性：

LocalFile	字符型，要上传的本地文件，含路径。
CurFolder	字符型，FTP 的当前目录，文件将被上传到此目录。
FTPName	字符型，返回 FTP 的 FTPName 属性

4.17.3.12 BeforeCreateFolder

在创建目录之前执行，通过此事件可以禁止某些用户创建目录。

e 参数属性：

FolderName	字符型，要创建的目录名称，含完整路径。
Cancel	逻辑型，如果设置为 True，将取消创建目录的操作。
FTPName	字符型，返回 FTP 的 FTPName 属性

示例

例如只允许管理员创建目录，可以将 BeforeCreateFolder 事件代码设置为：

```
If User.Type <> UserTypeEnum.Administrator Then
    e.Cancel = True
    MessageBox.show("你无权创建目录!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

4.17.3.13 AfterCreateFolder

在创建目录之后执行，通过此事件可以记录用户创建了哪些目录。

e 参数属性：

FolderName	字符型，要创建的目录名称。
FTPName	字符型，返回 FTP 的 FTPName 属性。

4.17.3.14 区分多个 FTP 客户端

FTP 客户端事件是统一设置的，我们无法单独为某个 FTP 客户端设置事件。

Foxtable 2018 为 FTPClient 增加了一个 FTPName 属性，这是一个字符型属性，用于标识不同的 FTP 客户端。

你可能已经留意到，所有的 FTP 客户端事件的 e 参数，都有一个名为 FTPName 的属性，我们可以通过此属性获取触发事件的 FTPClient 的 FTPName 属性值。

需要注意的是，FTPClient 的属性必须在打开 FTP 客户端之前设置，否则无效。

例如如果某个图片列采用了 FTP，需要设置此图片列的 FTPName 属性，那么就必须静止此图片列的窗口自动打开（在列属性设置窗口中，将“自动打开”属性设为 False），而是在 AfterOpenProject 事件中用代码设置好 FTPName 属性后再打开列窗口：

```
DataTables("表A").DataCols("照片").FTPClient.FTPName= "myFTP"
Tables("表A").Cols("照片").OpenWindow()
```

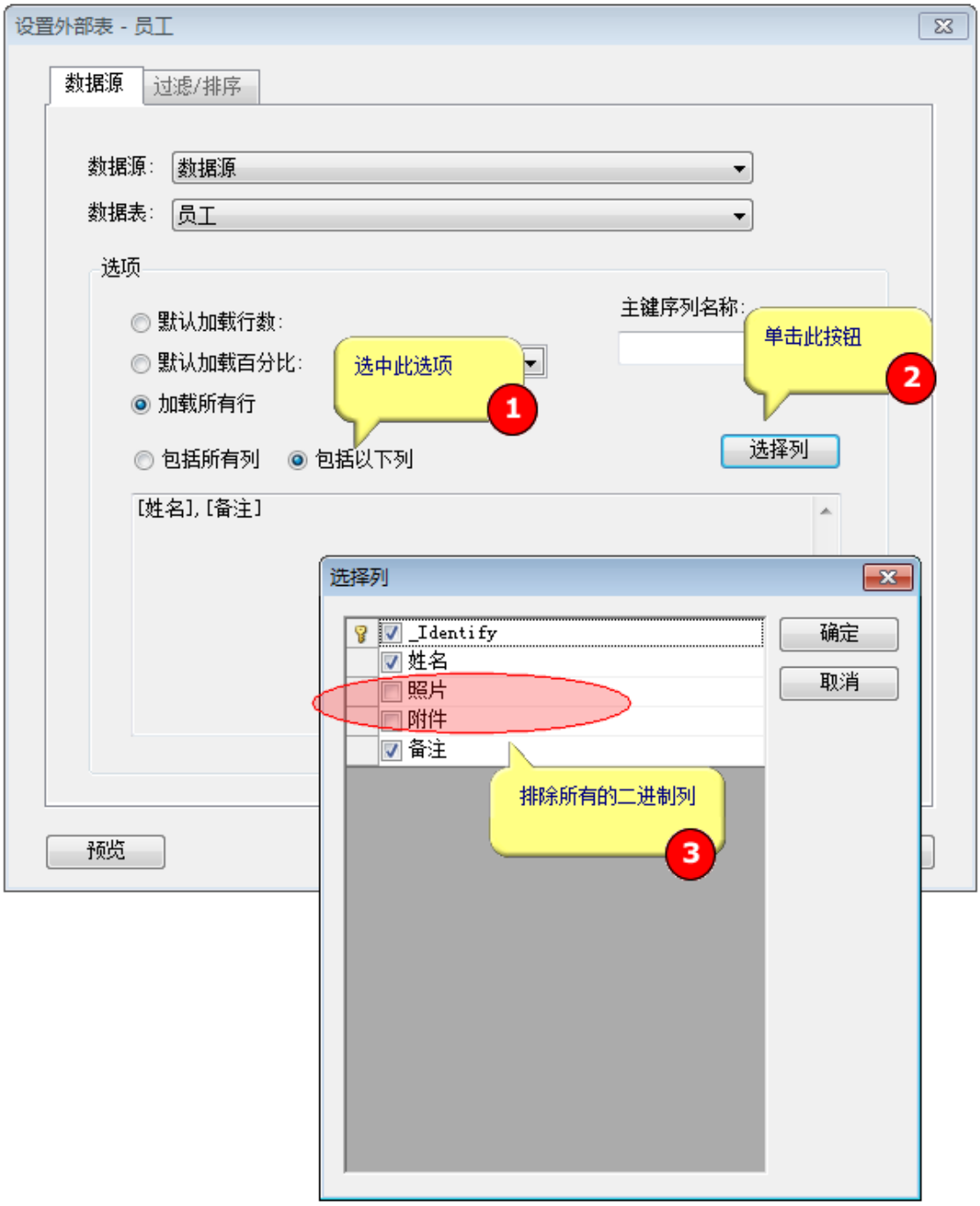
不仅是FTPName，所有需要通过代码进行设置的属性，都应该如上处理。

4.18 二进制列

4.18.1 关于二进制列

二进制列可以用于存储各种文件。

其实 Foxtable 并不推荐大家使用二进制列，而是建议大家使用内置的 FTP 来管理文件，内置的 FTP 已经和 Foxtable 的图片管理器及文件管理器完美地整合在一起，有更好的功能和性能。
但是确实有部分客户因各种原因，希望采用二进制列来管理文件，本章就介绍如何在 Foxtable 中使用二进制列。
由于内部表没有办法增加二进制列，所以只有使用外部数据源时，才能使用二进制列管理文件。
虽然 Access 也有二进制列，但是由于 Access 单个数据库文件不能超过 2G，所以使用二进制列时，如果文件比较大，建议大家采用 SQL Server。
你并没有办法直接在 Foxtable 中给外部表增加二进制列，所以你只能在原数据库中事先增加好二进制列，Access 对应的列类型为"OLE 对象",SQL Server 对应的列类型为"Image"。
Foxtable 的 DataRow 类型提供了多个直接操控二进制列的方法，这些方法直接在后台执行，所以在 Foxtable 中添加外部表的时候，没有必要 将二进制列加载进来。
准确地说，外部表不是没有必要将二进制列加载进来，而是万万不可将二进制列加载进来，因为二进制列会占据大量的内存，很容易导致系统运行迟缓甚至崩溃。
下图介绍了如何在增加外部表的时候，排除二进制列：



4.18.2 SQLSetValue

DataRow 的一个方法，直接向后台的某列写入数据，通常用来向未加载的列写入数据。

语法：

SQLSetValue(Filed,Value)

Filed：字符型，列名。
Value：要写入的值。
示例

假定员工表有个名为"CRC"的列，且此列并未加载。

向员工表当前行的"CRC"列写入某文件的CRC检验值，代码为：

```
Dim dr As DataRow = Tables("员工").Current.DataRow
Dim val As String   = CRCCheckFile("c:\data\1.doc")
dr.SQLSetValue("CRC",val)
```

如果要删除当前行CRC列的值，代码为：

```
Dim dr As DataRow = Tables("员工").Current.DataRow
dr.SQLSetValue("CRC值",Nothing)
```

当然我们可以自己编写SQL语句，通过SQLCommand执行之，一样可以实现向后台写数据的目的。和SQLCommand相比较，SQLSetValue的优势是使用快捷简单，劣势是一次只能写入一个数据，大家根据需要选择吧。

4.18.3 SQLGetValue

DataRow 的一个方法，直接从后台提取某列的数据，一般用于提取未加载列的数据。
语法：
SQLGetValue(Filed)
Filed: 字符型，列名。
示例
从后台提取当前行的名为 Info 列的值：

```
Dim val As String
Dim dr As DataRow = Tables("员工").Current.DataRow
val = dr.SQLGetValue("Info")
Output.Show(Val)
```

当然我们可以自己编写SQL语句，通过SQLCommand执行之，一样可以实现直接读取后台数据的目的。
不过，SQLGetValue使用起来显然要简洁方便很多，大家根据需要选择吧。

4.18.4 SQLInsertFile

DataRow 的一个方法，用于直接向二进制列中插入文件。
语法：

SQLInsertFile(Field,FileName)

Field:　字符型，二进制列的列名
FileName:　字符型，要插入文件的文件名
示例
选择一个 Word 文件，并插入到员工表的附件列中：

```
If Tables("员工").Current Is Nothing Then
    Return
End If
Dim dr As DataRow = Tables("员工").Current.DataRow
If dr.RowState = DataRowState.Added Then '如果是新增行,必须先保存才能插入文件
    dr.Save()
End If
Dim dlg As New OpenFileDialog
dlg.Filter = "Word文件|*.doc"
If dlg.ShowDialog =DialogResult.OK Then
    dr.SQLInsertFile("附件",dlg.FileName) '插入文件
End If
```

提示：请自行思考一下，为什么新增行必须先保存才能插入文件呢？

4.18.5 SQLLoadFile

DataRow的一个方法，用于从二进制列中提取文件。
语法：
SQLLoadFile(Field,FileName)

Field:　字符型，二进制列的列名
FileName:　字符型，文件名，提取的内容将保存在此文件中。
如果成功提取文件，返回 True，否则返回 False。
示例

假定员工表的附件列中存有Word文件，下面的代码可以提取这个Word文件并打开之：

```
If Tables(″员工″).Current Is Nothing Then
        Return
End If
Dim dr As DataRow = Tables(″员工″).Current.DataRow
Dim fl As String =   ProjectPath &   dr(″_Identify″) & ″.doc″
If dr.SQLLoadFile(″附件″,fl) Then ’如果提取文件成功
        Dim Proc As New Process ’打开文件
        Proc.File = fl
```

```
        Proc.Start()
Else
        Messagebox.Show("附件提取失败,可能并不存在附件!", "提示", MessageBoxButtons.OK,
MessageBoxIcon.Information)
End If
```

4.18.6 SQlLoadImage

DataRow的一个方法，用于从二进制列中提取图片。
语法：
SQLInsertFile(Field)

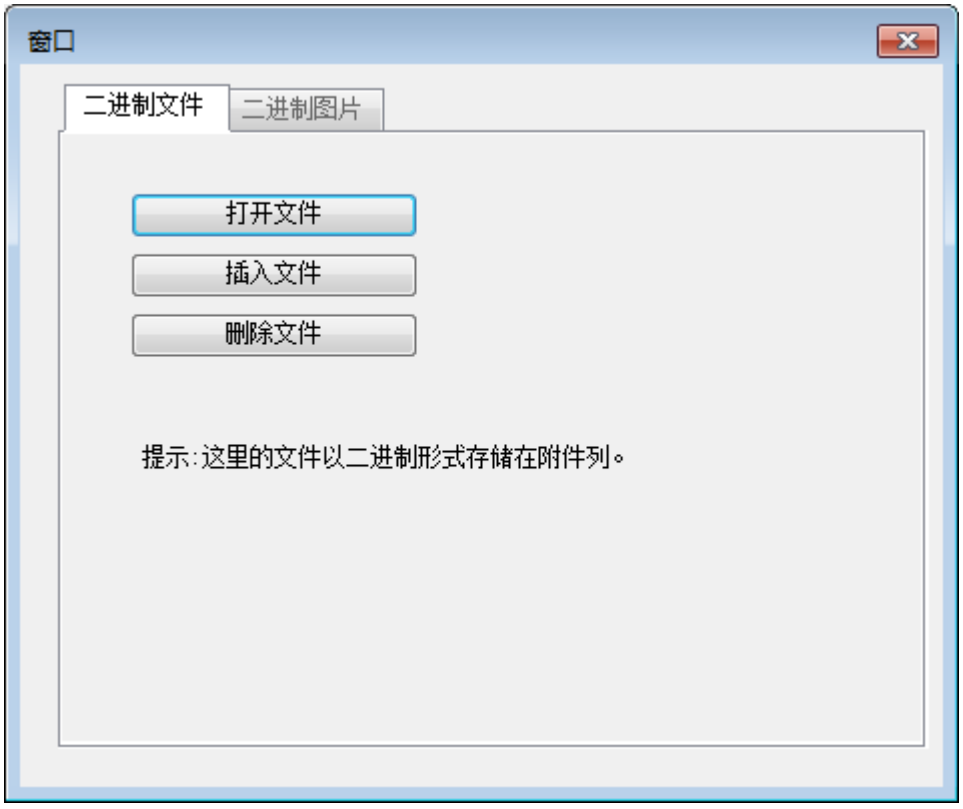
Field: 字符型，二进制列的列名
如果成功提取图片，返回一个 Image，否则返回 Nothing。
示例

下面的代码可以从员工表的照片列提取图片并显示在窗口的PictureBox控件中：

```
If Forms("窗口1").Opened Then '如果窗口已经打开
        Dim pbx As WinForm.PictureBox = Forms("窗口1").Controls("PictureBox1")
        If Tables("员工").Current Is Nothing Then
                pbx.Image = Nothing
        Else
                pbx.Image = Tables("员工").Current.DataRow.SQlLoadImage("照片") '从后
台提取照片并显示
        End If
End If
```

4.18.7 使用二进制列的简单示例

本节的目的是完整地介绍如何使用二进制列来管理文件和图片。
本节的内容可以参考 CaseStudy 目录下的文件：二进制列.Table
该示例文件使用了一个 Access 文件作为外部数据源，文件名为：二进制列.mdb
这个文件有一个名为员工的表，这个表包括两个二进制列，分别为附件和照片，前者用于存储 word 文件，后者用于存储照片：

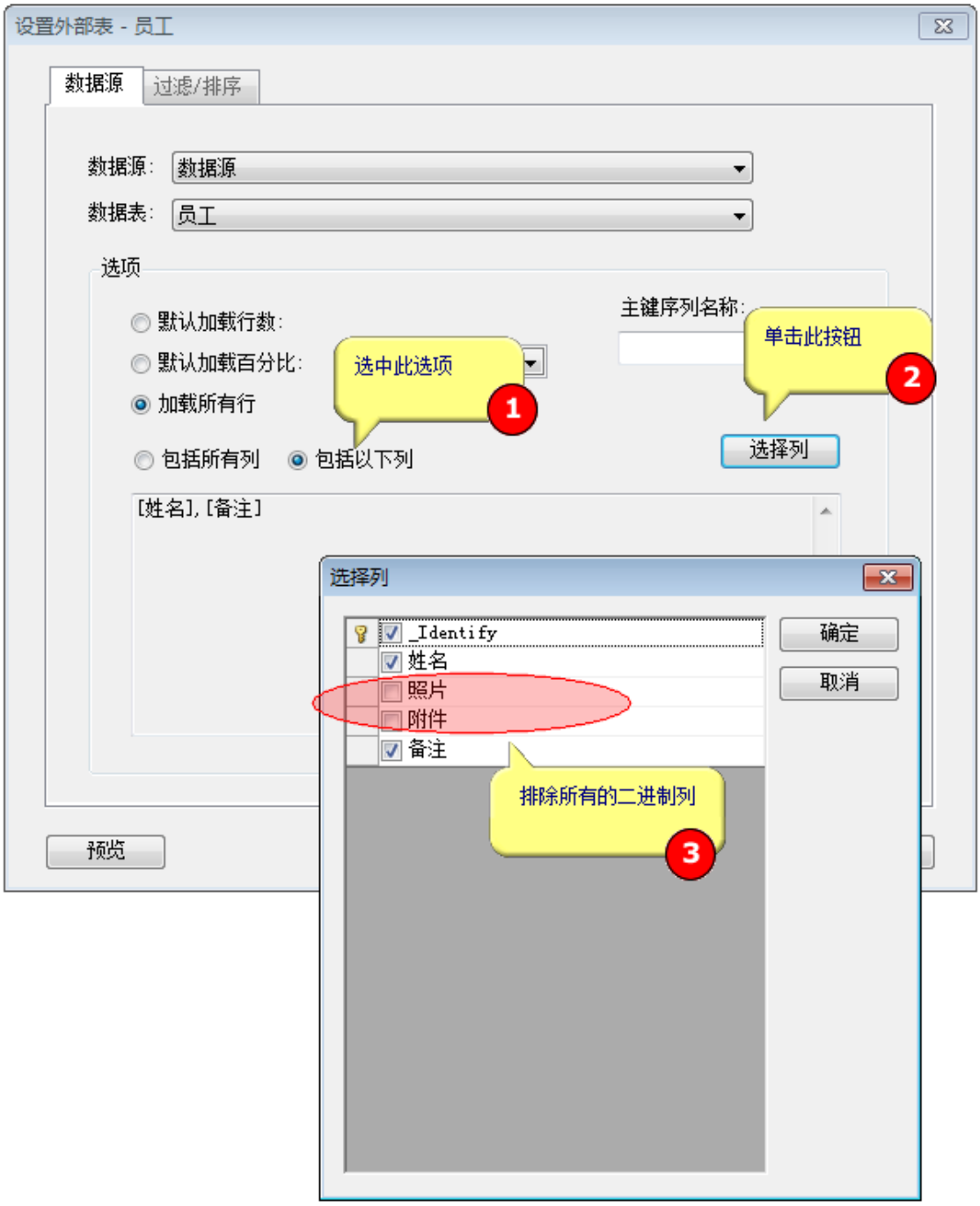




当然，二进制列可以存储任何格式的文件，本节只是一个示例而已。

设计步骤

1、首先设置好外部数据源，并记得在添加外部表的时候，排除所有的二进制列：



2、设计好窗口，窗口各个按钮的代码如下：

按钮名称	代码

打开文件	If Tables("员工").Current Is Nothing Then
	  Return
	End If
	Dim dr As DataRow = Tables("员工").Current.DataRow
	Dim fl As String = ProjectPath & dr("_Identify") & ".doc"
	If dr.SQLLoadFile("附件",fl) Then '如果提取文件成功
	  Dim Proc As New Process '打开文件
	  Proc.File = fl
	  Proc.Start()

	Else
	Messagebox.Show("附件提取失败,可能并不存在附件!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
	End If

插入文件	If Tables("员工").Current Is Nothing Then
	Return
	End If
	Dim dr As DataRow = Tables("员工").Current.DataRow
	If dr.RowState = DataRowState.Added Then '如果是新增行,必须先保存才能插入文件
	dr.Save()
	End If
	Dim dlg As New OpenFileDialog
	dlg.Filter = "Word文件|*.doc"
	If dlg.ShowDialog =DialogResult.OK Then
	dr.SQLInsertFile("附件",dlg.FileName) '插入文件
	End If
	
	提示：请自行思考一下，为什么新增行必须先保存才能插入文件呢？

删除文件	If Tables("员工").Current IsNot Nothing Then
	Tables("员工").Current.DataRow.SQLSetValue("附件",Nothing) '从后台删除附件列内容
	End If

插入照片	If Tables("员工").Current Is Nothing Then
	Return
	End If
	Dim dr As DataRow = Tables("员工").Current.DataRow
	Dim dlg As New OpenFileDialog
	dlg.Filter = "图形文件|*.bmp;*.jpg;*.gif;*.png"
	If dlg.ShowDialog = DialogResult.OK Then
	dr.SQLInsertFile("照片",dlg.FileName) '插入文件
	Dim pic As WinForm.PictureBox = e.Form.Controls("PictureBox1")
	pic.Image = GetImage(dlg.FileName)
	End If

删除照片	If Tables("员工").Current IsNot Nothing Then
	Tables("员工").Current.DataRow.SQLSetValue("照片",Nothing) '从后台删除照片列内容
	Dim pic As WinForm.PictureBox = e.Form.Controls("PictureBox1")
	pic.Image = Nothing
	End If

3、将订单表的CurrentChanged事件代码设置为：

```
If Forms("窗口1").Opened Then '如果窗口已经打开
    Dim pbx As WinForm.PictureBox = Forms("窗口1").Controls("PictureBox1")
    If Tables("员工").Current Is Nothing Then
        pbx.Image = Nothing
    Else
        pbx.Image = Tables("员工").Current.DataRow.SQlLoadImage("照片") '从后台提取照片并显示
    End If
End If
```

这样在员工表中选择不同的行时，会自动从后台的照片列提取此员工的照片，并显示在窗口的PictureBox中

4、最后将窗口的AfterLoad事件代码设置为：

```
Tables("员工").RaiseCurrentChanged() '强行触发 CurrentChanged 事件
```

这样打开窗口后，PictureBox会自动显示当前员工的照片，否则需要换一次行才会显示。
当然你可以直接在这里编写代码来提取照片，但既然相关代码已经在表的CurrentChanged写好了，就没有必要重复编写了。

4.18.8 多文件类型与本地缓存

如果要在二进制列要插入任意类型的文件，那么我们必须增加一列，用于存储所插入文件的后缀名。
同样为了提高运行效率，可以在插入文件的时候，同时在数据库中存储所插入文件的 crc 校验值；从数据库中提取文件之前，先判断本地是否存在此文件，如果存在，则将本地文件的 crc 校验值和数据库中的存储的 crc 校验值进行比较，如果相等，则直接使用本地文件，否则从数据库中提取文件。
后缀名和 crc 校验值可以存储在同一个列中。
设计过程：
1、数据库库中有两列，分别是附件列和信息列，附件列是二进制列，用于存储文件本身，信息列是字符列，用于存储文件的后缀名和 crc 校验值。
2、增加外部表的时候，将附件类和信息列排除，不要加载到 Foxtable 中。
3、插入文件的参考代码：

```
If Tables("员工").Current Is Nothing Then
    Return
End If
Dim dr As DataRow = Tables("员工").Current.DataRow
If dr.RowState = DataRowState.Added Then '如果是新增行,必须先保存才能插入文件
```

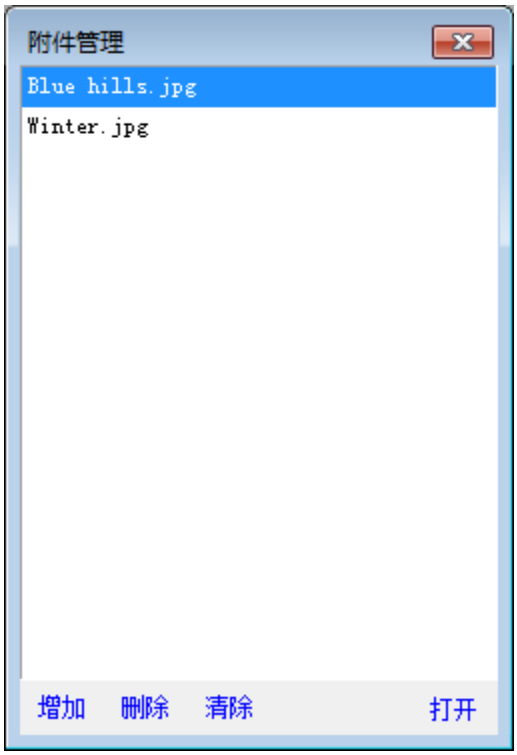
```
    dr.Save()
End If
Dim dlg As New OpenFileDialog
dlg.Filter = "图形文件|*.bmp;*.jpg;*.gif;*.png"
If dlg.ShowDialog =DialogResult.OK Then
    Dim fl As String = dlg.FileName
    Dim ex As String = fl.SubString(fl.LastIndexOf(".") + 1) '获取文件后缀名
    ex = ex & "|" & CRCCheckFile(fl) '将后缀名和文件的crc校验值组合成一个字符串
    dr.SQLInsertFile("附件",fl) '插入文件
    dr.SQLSetValue("信息",ex) '将文件的后缀名和crc校验值保存在信息列
End If
```

4、打开文件的参考代码：

```
If Tables("员工").Current Is Nothing Then
    Return
End If
Dim dr As DataRow = Tables("员工").Current.DataRow
Dim pts() As String = dr.SQLGetValue("信息").Split("|")
Dim fl As String =  ProjectPath &  dr("_Identify") & "." & pts(0)
If FileSys.FileExists(fl) AndAlso CRCCheckFile(fl) = pts(1) Then '如果本地存在同名文件且CRC校验值相同
    '则直接使用本地文件
Else '否则从数据库提取文件
    If dr.SQLLoadFile("附件",fl) = False Then '如果提取文件失败
        Messagebox.Show("附件提取失败,可能并不存在附件!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
        Return
    End If
End If
Dim Proc As New Process '打开文件
Proc.File = fl
Proc.Start()
```

4.18.9 二进制列的多文件管理

本节的内容可以参考 CaseStudy 目录下的示例文件：多文件二进制列.Table
该示例文件使用了一个 Access 文件作为外部数据源，文件名为：多文件二进制列.mdb
该示例演示了通过二进制列来设计一个多文件管理器:



这个文件管理器和 Foxtable 内置的文件管理器是相似的。

实际上二进制列只能插入一个文件，并没有办法插入多个文件。

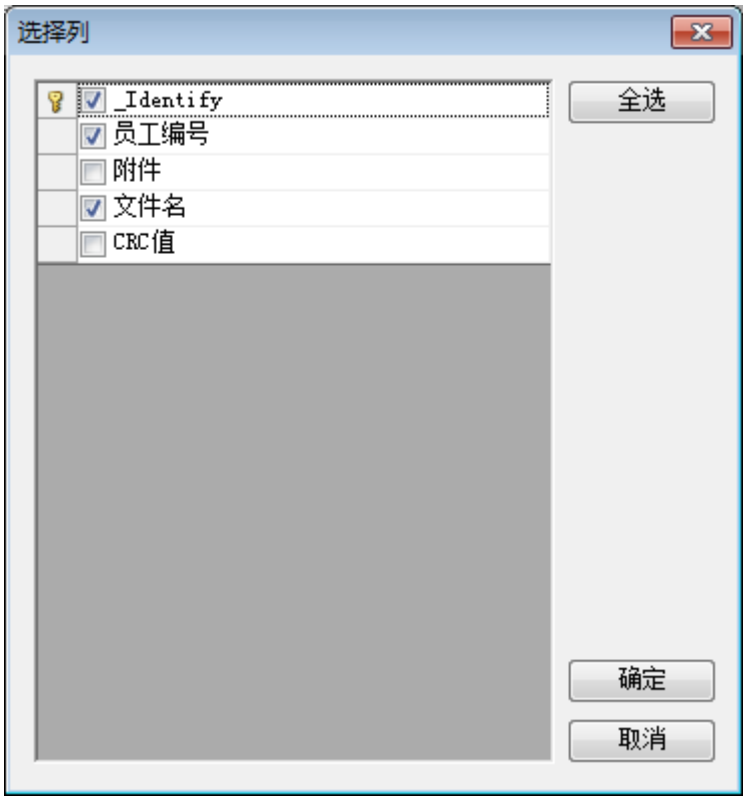
当需要管理多个文件时，我们只能使用关联表，新建一个专门用于插入文件的表，作为子表，和原来的表建立关联。

设计过程：

1、删除员工表中原来用于插入文件的附件列。

2、新建一个表，名为“附件”，包括编号(整数型)、附件(二进制型)、文件名(字符型)、CRC 值(字符型)等列，当然_Identify 列是必须有的：

3、将这个附件表作为外部表添加到 Foxtable 中，添加的时候，请排除附件和 CRC 值这两列：



4、员工表和附件表建立关联，员工表作为父表，关联列为"_Identify"，附件表作为子表，关联列为"编号"：



5、设计一个下图所示的窗口：



6、窗口中的 Table 控件绑定到关联表"员工.附件"，允许编辑设置为 False，显示行号设置为 False，网格线宽度设置为 0，整行突出设置为 True，显示列设置为"文件名|100"。

7、窗口的 AfterLoad 事件代码设置为：

```
Tables("员工.附件").ExtendLastCol = True '水平方向自动填满
Tables("员工.附件").SetHeaderRowHeight(0,-1) '隐藏列标题
DataTables("附件").SysStyles("EmptyArea").BackColor = Color.White '空白区的背景设置为白色
```

这样Table控件的外观就和ListBox(列表框)控件看起来是一样的。

8、窗口各按钮的代码：

按钮	代码

增加	Dim dlg As New OpenFileDialog
	dlg.Filter = "图形文件|*.bmp;*.jpg;*.gif;*.png"
	If dlg.ShowDialog =DialogResult.OK Then
	  Dim fl As String = dlg.FileName
	  Dim r As Row = Tables("员工.附件").AddNew()
	  r("文件名") = FileSys.GetName(fl) '写入文件名
	  r.Save '必须先保存
	  r.DataRow.SQLInsertFile("附件",fl) '插入文件
	  r.DataRow.SQLSetValue("CRC值", CRCCheckFile(fl)) '保存crc检验值
	End If

删除	Dim r As Row = Tables("员工.附件").Current
	If r IsNot Nothing Then
	  r.Delete
	End If

清除	For Each dr As DataRow In Tables("员工").Current.DataRow.GetChildRows("附件") dr.Delete() Next

打开	If Tables("员工.附件").Current Is Nothing Then
	  Return
	End If
	Dim dr As DataRow = Tables("员工.附件").Current.DataRow
	Dim fl As String = ProjectPath & dr("文件名")
	If FileSys.FileExists(fl) AndAlso CRCCheckFile(fl) = dr.SQLGetValue("CRC值") Then '如果本地存在同名文件且CRC校验值相同
	  '则直接使用本地文件
	Else '否则从数据库提取文件
	  If dr.SQLLoadFile("附件",fl) = False Then '如果提取文件失败
	  Messagebox.Show("附件提取失败,可能并不存在附件!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
	  Return
	  End If
	End If
	Dim Proc As New Process '打开文件
	Proc.File = fl
	Proc.Start()

9、最后将附件表的DoubleClick事件代码设置为：

```
If Forms("窗口1").Opened() Then '如果文件管理窗口已经打开
    If e.Table.name = "员工.附件" '如果双击的是关联表
        Dim btn As WinForm.Button =  Forms("窗口1").Controls("Button4")
        btn.PerformClick() '模拟单击打开文件按钮
    End If
End If
```

这样在Table控件中双击某文件，就会模拟单击打开按钮，从而打开所双击的文件。

4.19 数据上报

4.19.1 远程数据上报

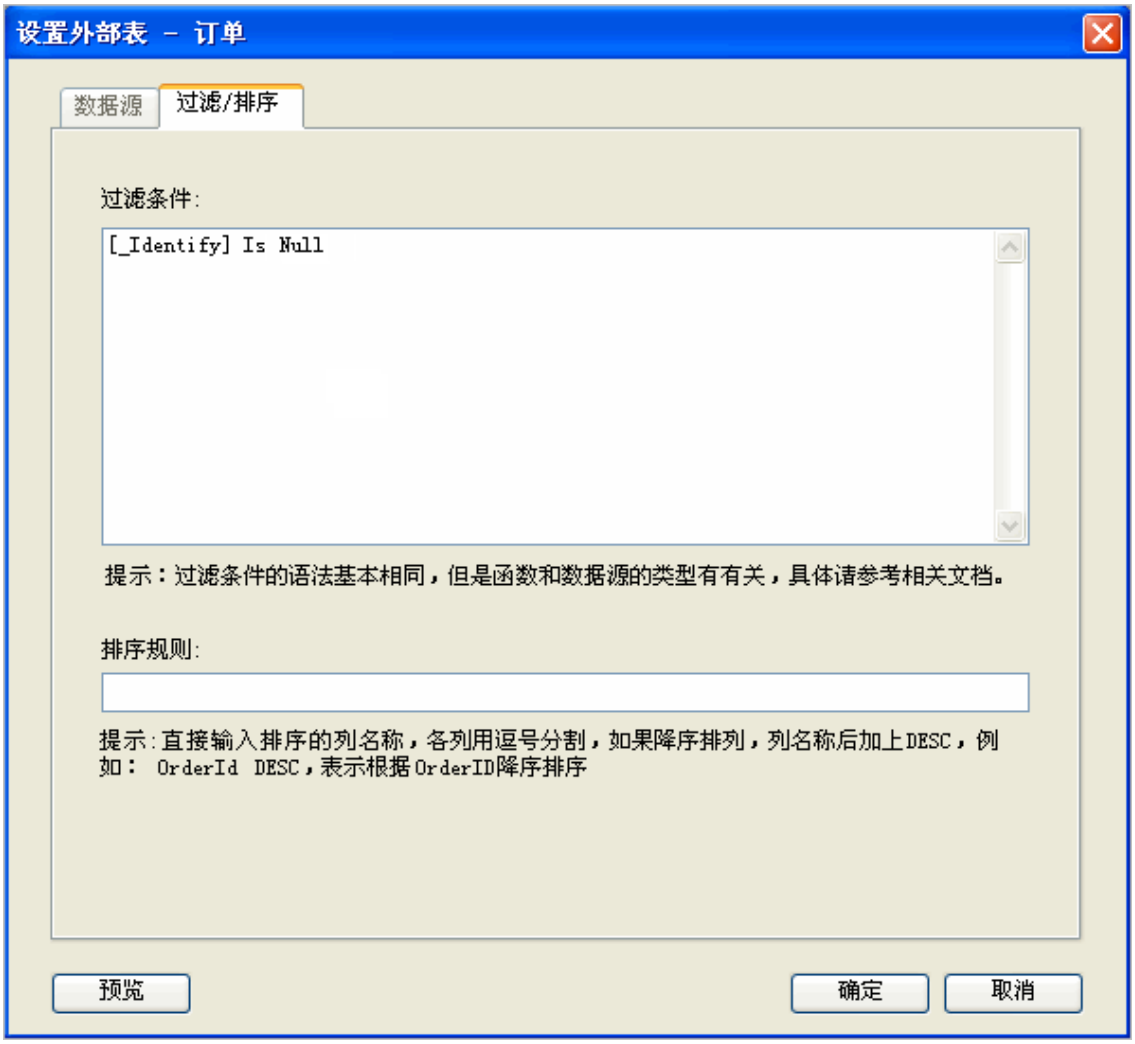
在所有的数据上报方案中，基于 Sql Server 的上报是最为简单的，而且非常灵活。
设计步骤
1、有一台能通过互联网访问的服务器，安装上 SQL Server。
2、你可以直接在 SQl Server 建立数据库和数据表，然后在 Foxtable 连接这个数据库，并添加外部

表。

3、你也可以直接在 FoxTable 建立内部表，完成之后将此表重定向到 Sql Server 中。

4、既然是远程上报，那么就不应该加载现有数据，至少不应该加载太多现有数据。

首先在定义外部表的时候，通过设置条件，使得初始不加载任何数据：



5、如果有必要的话，可以在 AfterOpenProject、LoadUserSetting 或 BeforeLoadOuterTable 事件中设置代码，为用户加载部分数据。

6、将你的项目发给客户而使用，至此你的远程上报系统已经设计完毕，前后不会超过 10 分钟。

7、本章的内容也适合局域网。

4.19.2 通过手机上报

从 Foxtable 2017 开始，高级开发版开始提供移动开发功能，使得远程上报数据变得更加方便。

Foxtable 的移动开发功能以 B/S 形式实现，内置手机网页自动生成功能，你不需要额外的学习，你可以完全不懂网页设计，也可以快速开发基于手机的企业管理系统，包括服务端的架设和网页的生成，全都可以在 Foxtable 中用最简单的方式完成，不仅简单到人人都能掌握，而且效率极高，相对于其它开发工具，代码节省了 85%左右。

此外高级开发版还提供了微信接入功能，用户可以直接通过微信输入、查询、统计和审核数据。

下面是 Foxtable 自动生成的一些手机网页：



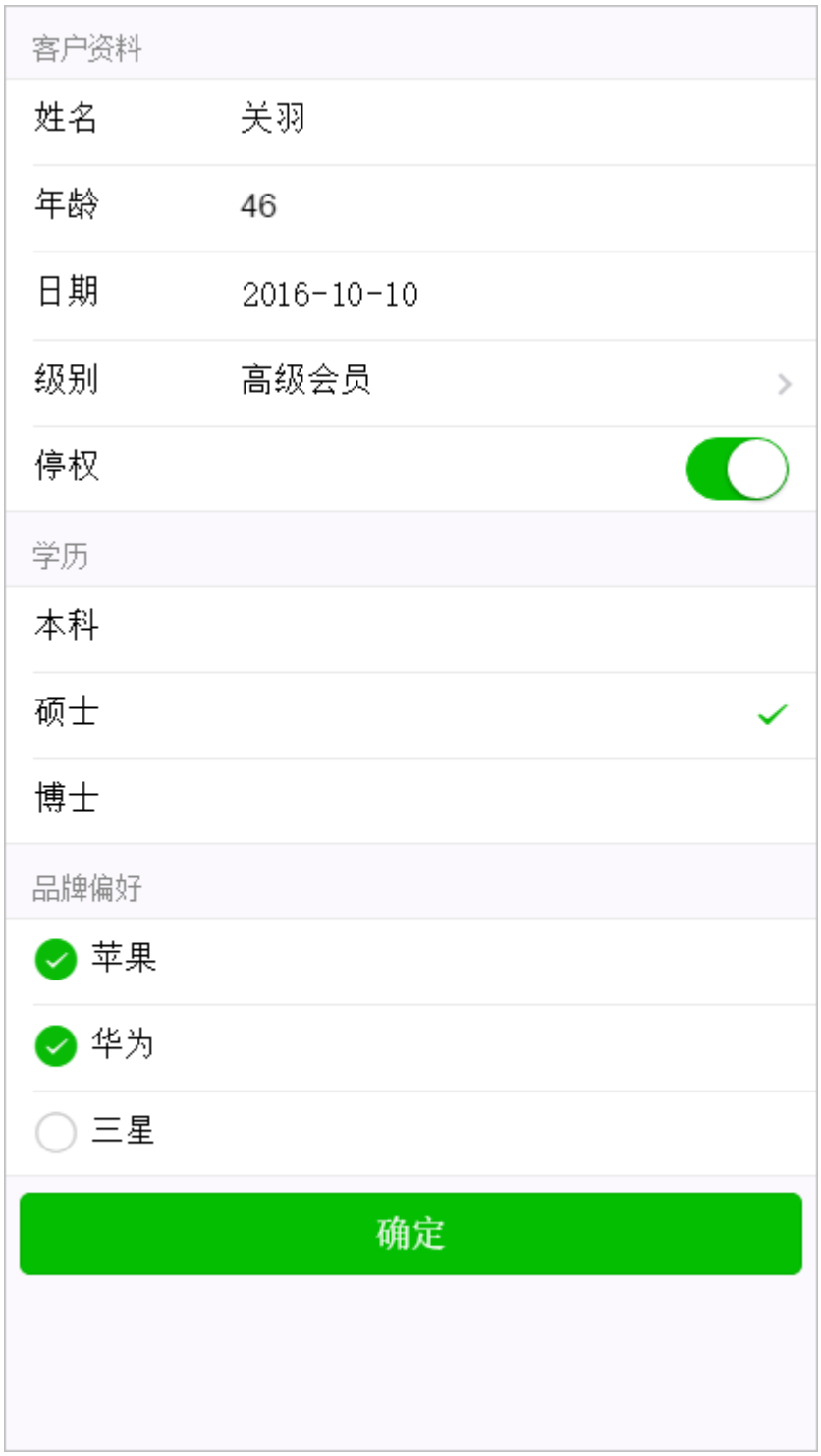
客户资料

姓名	舒淇
地点	蒙古草原
日期	2012-10-12




基本资料
姓名
年龄
卡号
请输入卡号
密码
请输入密码
手机
获取验证码
验证码
+86
请输入联系电话


编辑订单
编号
161206005
客户
柳传志
日期
2017-02-12
总数量
23
总金额
111024
产品
数量
单价
金额
Foxtable 开发版
21
4688
98448
iphone 6 plus
2
6288
12576
增加明细
保存
返回
生成Excel格式订单



如果需要了解更多关于 Foxtable 移动开发功能的信息，可以访问：

http://www.foxtable.com/mobilehelp/

4.19.3 用 FTP 上报数据

如果没有条件搭建 SQL Server 服务器，那么也可以考虑通过 FTP 实现远程数据上报。

1、将需要上报的数据生成一个 Access 文件，并上传到 FTP 服务，例如将当天订单上报：

```
'导出当天订单
Dim ex As New Exporter
ex.SourceTableName = "订单"
ex.FilePath = "c:\temp\gd.mdb"
ex.Filter = "日期 = #" & Date.Today & "#"
If FileSys.FileExists(ex.FilePath) Then '如果目标文件已经存在
    FileSys.DeleteFile(ex.FilePath) '则删除之
End If
ex.Export()
'上传到 FTP 服务器
Dim ftp1 As new ftpclient
ftp1.host="ftp.baidu.com"
ftp1.Account = "foxtable"
ftp1.password = "168168"
ftp1.upload(ex.FilePath,"/gd.mdb",True) '上传
```

2、总部接收数据的代码：

```
'下载上报数据
Dim ftp1 As new ftpclient
ftp1.host="ftp.baidu.com"
ftp1.Account = "foxtable"
ftp1.password = "168168"
If ftp1.FileExists("/gd.mdb") = False Then
    MessageBox.Show("广东分公司还未上报今天数据","提示")
    Return
End If
ftp1.Download("/gd.mdb","c:\temp\gd.mdb") '下载 ftp 上的上报文件
ftp1.Delete("/gd.mdb") '下载成功后删除 ftp 上的上报文件.
'合并上报数据
Dim mg As New Merger
mg.SourcePath = "c:\temp\gd.mdb"
mg.SourceTableName = "订单"
mg.DataTableName = "订单"
mg.Merge()
Filesys.DeleteFile("c:\temp\gd.mdb") '合并后删除下载文件
```

4.19.4 通过邮件上报数据

如果你既没有 SQL Server 服务器，也没有 FTP，也可以考虑通过邮件上报数据，只是没有那么自动。

假定各分公司每天上报当天订单。

1、将要上报的数据生成为一个 Access 文件，并通过邮件发送给总部：

```
'导出当天订单
Dim ex As New Exporter
ex.SourceTableName = "订单"
ex.FilePath = "c:\temp\gd.mdb"
ex.Filter = "日期 = #" & Date.Today & "#"
If FileSys.FileExists(ex.FilePath) Then '如果目标文件已经存在
    FileSys.DeleteFile(ex.FilePath) '则删除之
End If
ex.Export()
'通过邮件发送
Dim m As New MailSender
m.Host = "smtp.21cn.net"
m.Account = "xiaoliu"
m.Password = "xiaoliu1234567"
m.From = "feng@32cn.net"
m.To = "zong@21cn.ent"
m.Title = "订单上报"
m.Content = "订单上报"
m.AddAttachments("c:\temp\gd.mdb") '将上报文件作为附件发送
m.SendAsync()
```

2、目前 Foxtable 并未提供接收邮件的功能，用户只能手工将各分公司通过邮件发来的附件保存在指定目录，然后通过下面的代码批量合并：

```
For Each File As String In Filesys.GetFiles("c:\temp")
    If File.EndsWith(".mdb") '如果后缀名是 mdb,说明此文件是 access 格式
        Dim mg As New Merger
        mg.SourcePath = file
        mg.SourceTableName = "订单"
        mg.DataTableName = "订单"
        mg.Merge()
        Filesys.DeleteFile(File) '合并后删除文件
    End If
Next
```

有了上面的代码，即使有上百个分公司上报数据，也只需单击一个按钮即可完成合并。

4.20 工作流

4.20.1 关于工作流

所谓工作流，就是各司其职、按部就班，由指定的人按规定的顺序来完成各项工作。

大多数有一定编程能力的开发工具都能实现工作流，不过用 Foxtable 实现工作流更是信手拈来，

任何用户都可以轻松实现复杂的流程管理，远比那些所谓的工作流软件更为实用和灵活。
特别是 Foxtable 2016 内置消息推送功能 OpenQQ 和 UDPClient 后，工作流的开发更加高效快捷。
除工作流之外，本章会接触一些新的概念，例如传统编辑方式，内部通讯、并发冲突处理等等。
我们见到不少用户，以为权限管理、工作流、远程上报有多复杂，多专业，需要花大价钱购买专业软件才能实现，其实这些都是很基本的功能，有了 Foxtable，任何用户都可以轻松解决。

4.20.2 审批流程的实现

假定某个表的行需要经过一个录入、审核和批准的流程，要求：

1、主管负责审核，经理负责批准。
2、先审核后批准，禁止批准未通过审核的行。
3、禁止编辑已经通过审核的行。
4、对于已经通过批准的行，禁止取消审核。
3、能记录审核人和审核日期以及批准人和批准日期。

首先在表中加入审核、审核日期、批准、批准日期四列。
然后在菜单或者窗口中加入审核、批准、取消审核、取消批准四个按钮，代码分别为：

按钮	代码

审核	If User.Group = "主管" Then
	  Tables("表 A").Current("审核") = User.Name
	  Tables("表 A").Current("审核日期") = Date.Today()
	  Tables("表 A").Current.Save()
	Else
	  MessageBox.show("你无审核权限!")
	End If

批准	If User.Group = "经理" Then
	  If Tables("表 A").current.IsNull("审核") Then
	  MessageBox.Show("不能批准还未通过审核的行.")
	  Else
	  Tables("表 A").Current("批准") = User.Name
	  Tables("表 A").Current("批准日期") = Date.Today()
	  Tables("表 A").Current.Save()
	  End If
	Else
	  MessageBox.show("你无批准权限!")
	End If

取消审核	If User.Group = "主管" Then
	  If Tables("表 A").Current.IsNull("批准") Then
	  Tables("表 A").Current("审核") = Nothing
	  Tables("表 A").Current("审核日期") = Nothing

	Tables("表 A").Current.Save()
	Else
	MessageBox.Show("此行已被批准,不能取消审核!")
	End If
	Else
	MessageBox.show("你无取消审核权限!")
	End If

取消批准	If User.Group = "经理" Then
	Tables("表 A").Current("批准") = Nothing
	Tables("表 A").Current("批准日期") = Nothing
	Tables("表 A").Current.Save()
	Else
	MessageBox.show("你无取消批准权限!")
	End If

最后将表事件PrepareEdit的代码设置为：

```
If e.Row.IsNull("审核") = False Then '如果审批列不为空
    e.Cancel = True '则禁止编辑
End If
```

用于禁止修改已经通过审核的行。

4.20.3 主动通知下一个流程

企业的业务流程可能有很多层，如何在某一层完成之后，自动通知负责下一层的同事，即时处理此业务的后续流程呢？

以我们上一节介绍的审批流程为例:

如何在负责录入的员工录入数据之后，主动通知有审核权限的员工加载并审核此行数据呢？
如何在负责审核的员工在审核某行数据之后，主动通知有批准权限的员工加载并批准此行数据呢？
Foxtable 从 2016 版开始，内置了 OpenQQ，可以很方便地解决这个问题，参考：利用 OpenQQ 高效即时同步数据
我们可以在上述代码的基础上改进一下，例如希望有新的订单需要批准时，能弹出一个窗口提醒一下经理。
为此可以将 ReceivedMessage 时间改为：

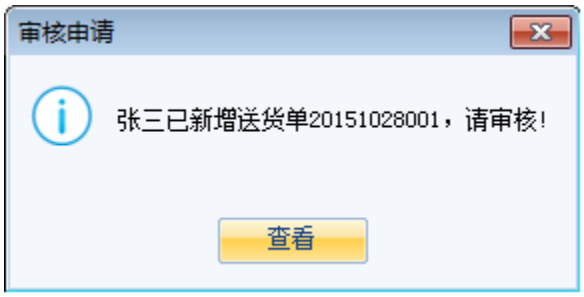
```
Dim msg As String = e.Message
Dim dr As DataRow
If msg.StartsWith("U#") Then '表示修改了某行
```

```
    Dim pts() As String = msg.Split("#")
    If pts.Length = 3 Then
        dr = DataTables(pts(1)).Find("[_Identify] = " & pts(2))
        If dr IsNot Nothing  Then
            dr.Load() '重新加载此行
        End If
    End If
ElseIf msg.StartsWith("A#") Then '表示增加了行
    Dim pts() As String = msg.Split("#")
    If pts.Length = 3 Then
        Dim drs As List(of DataRow) = DataTables(pts(1)).AppendLoad("[_Identify] = " &  pts(2)) '追载新增加的行.
        If drs.Count > 0 Then
            dr = drs(0)
        End If
    End If
ElseIf msg.StartsWith("D#") Then '表示删除了行
    Dim pts() As String = msg.Split("#")
    If pts.Length = 3 Then
        DataTables(pts(1)).RemoveFor("[_Identify] = "  & pts(2)) '移除行
    End If
End If
If User.Group = "经理" And dr.DataTable.Name = "订单"  Then
    If dr IsNot Nothing AndAlso dr.IsNull("审核") = False AndAlso dr.IsNull("批准") Then
        PopMessage("有新的订单需要审核!","提示",PopIconEnum.Infomation)
    End If
End If
```

4.20.4 自定义流程通知窗口

上一节介绍了如何在工作流中发通知信息。
本节我们更进一步，自定义一个流程消息提示窗口，单击窗口中的按钮可以自动完成一些操作。
本节内容需要实现下述任务：

送货单由用户张三开单，由李四审核。
当张三开送货单保存时，系统自动发给李四发一个消息：“张三已新增送货单 20151028001，请审核”。
李四收到此消息后，点此消息，打开这张送货单，进行审核，审核后，反馈回张三消息：“送货单 20151028001 已由用户李四审”
下图就是我们要设计的提示窗口：



李四收到信息之后，单击提示窗口的查看按钮，即可加载(或刷新)并定位到此送货单，并自动打开送货单编辑窗口，查看该送货单信息并审核。

本节内容使用 OpenQQ 发送和接收信息，关于 OpenQQ 服务端和客户端的设计，前面的章节已经有详细介绍，这里就不重复。

以下的设计全部在客户端项目完成。

设计步骤

1、首先我们要定义一个集合，用于存储收到的审核申请，在全局代码中定义一个 Public 变量：

```
Public fhs As New List(of String)
```

因为存在一种可能，就是张三连续开了多个送货单，李四因故没有即时审核，所以我们应该将收到的审核申请存储在集合 fhs 中，逐个提醒李四。

2、现在我们要约定一个消息格式，审核申请的格式为"+@|主键|送货单号|制单人"，审核完成的格式为"-@|送货单号|审核人"。

3、在发货单编辑窗口的保存按钮中，写入如下代码：

```
Dim r As Row  = Tables("送货单").Current
r.Save()
QQClient.Send("李四","+@|" & r("_Identify") & "|" & r("单号") & "|" & User.Name)
```

4、在发货单编辑窗口的审核按钮中，写入如下代码：

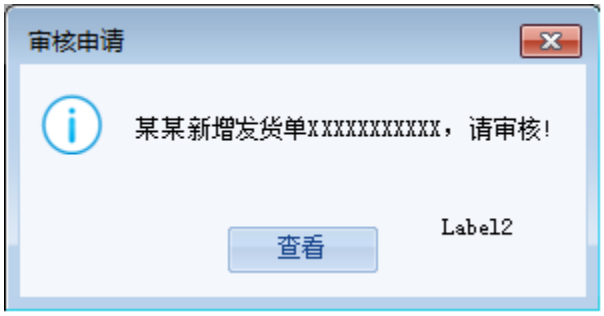
```
Dim r As Row  = Tables("送货单").Current
r("审核") = True
r.Save()
'发送审核通过的消息
QQClient.Send("张三","-@|" & r("单  号") & "|" & User.Name)
```

5、在 OpenQQ 客户端的 MessageReceived 事件中编写代码：

```
Dim pts() As String  = e.Message.Split("|")
If pts.Length = 4 AndAlso pts(0) = "+@" '如果是申请审核消息
    fhs.Add(pts(1) & "|" & pts(2) & "|" & pts(3)) '将主健、单号和制单人 合成为一个字符串存储在集合中
ElseIf pts.Length = 3 AndAlso pts(0) = "-@" '如果是审核通过消息
    PopMessage("送货单" & pts(1) & "已由" & pts(2) & "审核通过!")
    Dim dr As DataRow = DataTables("送货单").Find("单号 = '" & pts(1) & "'")
    If dr IsNot Nothing Then
```

```
        dr.Load
    End If
End If
```

6、在客户端项目新建一个下图所示的窗口，窗口名为“审核提示”，窗口类型设置为 PopupForm：



这个窗口有两个 Label 控件，Label1 用于显示提示信息，Label2 的可见属性设置为 False，用于存储发货单的主键(Identify)。
这是一个技巧，因为 Foxtable 并没有提供窗口级别的变量，所以通常用一些不可见的 Label 控件来存储临时信息。
将查看按钮的代码设置为：

```
DataTables("送货单").Save() '接下来要根据主键查找，必须先保存，因为只有保存后新增行的主键才是有效值。
Dim id As String = e.Form.Controls("Label2").Text
Dim dr As DataRow = DataTables("送货单").Find("[_Identify] = " & id)
If dr Is Nothing Then
    DataTables("送货单").AppendLoad("[_Identify] = " & id)
    dr =  DataTables("送货单").Find("[_Identify] = " & id)
Else
    dr.Load()
End If
If dr IsNot Nothing Then
    Dim wz As Integer = Tables("送货单").FindRow(dr)
    If wz < 0 Then '如果因为筛选导致此送货单不可见
        Tables("送货单").Filter = ""
        wz =Tables("送货单").FindRow(dr)
    End If
    If wz >= 0 Then
        Tables("送货单").Position = wz
        Forms("发货单").Open()
    End If
End If
e.Form.Close()
```

上面的代码不难理解，需要注意的是，为什么必须首先保存一下送货单表呢？
实际上如果李四只有审核送货单的权限，没有增加送货单的权限，那么第一行保存表的代码是可以省略的，这又是为什么呢？

假定送货单表原来有10行(主键值0-9)，张三和李四都可以增加送货单，两人分别增加1个订货单，主键值都是10。
张三保存后，李四收到通知，如果李四不保存，直接用AppendLoad追载主键为10的行，由于表中已经有一个主键为10的行，所以会出现错误提示。
如果李四先保存，其新增行的主键会在保存后变为11，此时就可以正常根据通知信息追载主键为10的行了。

7、在项目事件SystemIdle中加入代码：

```
If fhs.Count > 0 AndAlso Forms("审核申请").Opened = False Then
    Dim pts() As String = fhs(0).Split("|")
    fhs.RemoveAt(0)
    With Forms("审核申请")
        .Open()
        .Controls("Label1").Text = pts(2) & "新增订货单" & pts(1) & ",请审核!"
        .Controls("Label2").Text = pts(0) '将发货单主健设置为Label2的标题
    End With
End If
```

上述代码在系统空闲的时间检查fhs集合，判断是否有新的审核申请；如果有新的申请且审核申请提示窗口未显示，则显示集合中的第一个审核申请，并将其从集合中移除。
上述代码将待审核送货单的主键设置为Label2的标题，这样在审核申请窗口单击查看按钮时，可以根据Label2的标题找出待审核的订单。

4.20.5 有关知识点

工作流讲究的是数据在用户之间高效流转，如果设计的时候，多多针对单行处理，会比处理整个表更加高效。
例如某行编辑完毕，需要保存此行以流转到下一流程，显然单单保存此行会比保存整个表快很多。
FoxTable 的 DataRow 和 Row 均有以下方法，用于处理单行数据：

方法	说明
Save	保存此行
Load	重新加载此行
Reject	撤销对此行的修改
Remove	移除此行

通过 DataRow 或 Row 执行上述方法，效果是一样的，实际上通过 Row 执行上述方法的时候，调用的就是 DataRow 的同名方法。
同时为 Row 和 DataRow 提供这些方法，虽然有些冗余，但有利于简洁编码。
此外工作流还需要及时从后台加载待处理的数据，显然用追载的方法，将后台新增待处理数据追加到表中，比重新加载整个表要高效很多。
DataTable 的 AppendLoad 方法用于追载数据，语法：
AppendLoad(Filter,Save)

Filter：字符型，用于设置追载条件。
Save: 可选参数，逻辑型，是否在追载数据前先保存现有数据，默认为 True，既先保存后追载。
AppendLoad 会返回一个集合，包含所有追载的行(DataRow)。
需要注意的是:
1、必须确保即将追载的行，并不存在于表中，也就是这些行之前并未加载，否则将报错。
2、AppendLoad 适合追载少量数据，大量数据一般用 Load 重新加载。
示例：
假定甲、乙两个用户同时编辑表 A，甲负责增加行，并输入部分列的数据，乙负责输入其它列的数据。
我们需要为乙用户设计一个按钮，用于将打开项目以来，甲用户新增加的行追载到表中，按钮的代码为：

```
Dim id As Integer = DataTables("表 A").Compute("Max(_Identify)")
Dim Filter As String = "[_Identify] > " & id
DataTables("表 A").AppendLoad(Filter, False)
```

上述代码先计算出表 A 中"_Identify"列的最大值，然后从后台追载"_Identify"列的值大于此最大值的行，这些行就是甲用户新增加的行。
提示：甲用户新增行并输入数据后，必须保存，乙用户才可追载此新增加的行，这是因为 AppendLoad 是从后台追载的，如果新增行没有保存，后台就不存在此行， 乙用户也就谈不上追载了。

4.20.6 传统编辑方式

本节内容可以参考 CaseStudy 目录下的文件：传统编辑方式.Table
在默认情况下，Foxtable 允许用户修改表中任何一行数据，也可以随意增加行，在全部修改完毕后，单击保存按钮统一保存。
在工作流环境下，这种处理方式可能不符合要求，例如有的工作流会要求逐行处理数据，处理完毕后要及时保存并流转到下一流程。
所以在讲述流程之前，我们有必要学习一下如何实现传统编辑方式。
本节要做一个下图所示的录入界面：



要求：

1、禁止直接在表中编辑数据，改用录入窗口形式编辑。

2、双击表中的任何一行，即可打开录入窗口编辑此行。

3、每次只能修改一条记录(也就是一行)。

4、在保存或者撤销此记录的修改之前，不能移到其他行，也不能关闭窗口。

5、单击"保存修改"按钮保存此次修改，单击"撤销修改"按钮，撤销此次修改，所有列返回修改前的值。

设计步骤：

1、设计好上图所示的窗口，窗口类型为模式窗口，这样在用户关闭此窗口之前不能进行任何操作。

2、将窗口的 BeforeClose 事件代码设置为：

```
If Tables("员工").Current.DataRow.RowState <> DataRowState.Unchanged Then '如果当前行已经修改过
    e.Cancel = True
End If
```

上面的代码在关闭窗口前执行，如果用户已经修改当前记录(行)，但未保存，将禁止关闭窗口，直到保存或撤销修改。

提示：行状态在修改前是 Unchanged，修改后是 Modified，保存修改或撤销修改后，又回到 Unchanged。

3、窗口中各按钮的代码：

按钮	代码

上一条	With Tables("员工")
	    If .Current IsNot Nothing AndAlso .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	        .Position = .Position - 1
	    End If

	End With

下一条	With Tables("员工")
	  If .Current IsNot Nothing AndAlso .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = .Position + 1
	  End If
	End With

第一条	With Tables("员工")
	  If .Current IsNot Nothing AndAlso .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = 0
	  End If
	End With

最末条	With Tables("员工")
	  If .Current IsNot Nothing AndAlso .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = .Rows.Count - 1
	  End If
	End With

新增	With Tables("员工")
	  If .Current Is Nothing OrElse .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  Tables("员工").AddNew()
	  End If
	End With

删除	With Tables("员工")
	  If .Current IsNot Nothing Then
	  .Current.Delete
	  End If
	End With

保存修改	With Tables("员工")
	  If .Current IsNot Nothing Then
	  .Current.Save()
	  End If
	End With

撤销修改	With Tables("员工")
	  If .Current IsNot Nothing Then
	  .Current.Reject()
	  End If
	End With

从上面的代码可以看出，如果当前记录(行)已经修改但没有保存，窗口中的上一条、下一条、第一条、最末条、新增按钮将拒绝执行，直到保存或撤销修改。

4、将员工表的 PrepareEdit 事件代码设置为：

```
e.Cancel = True
```

这样就禁止了直接从表中修改数据。

5、将员工表的 DoubleClick 事件代码设置为：

```
Forms("编辑窗口").Open()
```

这样双击某行，将自动打开编辑窗口编辑此行数据。

4.20.7 独占式编辑

本节讲述如何避免多人同时编辑同一行，既如何实现独占式编辑。

提醒：由于 OpenQQ 的出现，本节的内容已经过时，请参考：用 OpenQQ 实现独占式编辑

虽然本节的内容有些过时，但是依然有参考价值，所以我们依然保留了这一节的内容，有兴趣的话，可以看看。

示例一

本示例可以参考 CaseStudy 目录下的文件：独占式编辑之一.Table

请分别启动多个 Foxtable 打开此文件，并以不同的用户登录，同一行只要有人开始编辑，那么直到此人保存此行，其他人都不能编辑此行。

设计步骤：

1、首先给表增加一个字符列，名为"编辑者"，用于记录正在编辑某行数据的用户名。

2、在项目事件 AfterOpenProject 设置代码：

```
'解锁非正常退出时被锁定的记录
DataTables("员工").ReplaceFor("编辑者",Nothing,"编辑者 = '" & User.Name & "'")
DataTables("员工").Save()
Tables("员工").Cols.Remove("编辑者") '移除编辑者列,避免用户直接修改此列
```

3、将表事件 StartEdit 代码设置为：

```
If e.Row("编辑者") = User.Name Then '如果编辑者是当前用户
    Return
End If
Dim cmd As New SQLCommand
Dim exp As String  = "编辑者 Is Null And [_Identify] = " & e.Row("_Identify")
cmd.CommandText = "Update {员工} Set 编辑者 = '" & User.Name & "' Where " & exp
If cmd.ExecuteNonQuery = 1 Then
    e.Row("编辑者") = User.Name
Else
    cmd.CommandText = "Select 编辑者 From {员工} Where [_Identify] = " & e.Row("_Identify")
    Dim nm As String = cmd.ExecuteScalar
    If nm > "" Then
        MessageBox.show(nm & "正在编辑此行!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
        MessageBox.show("此行可能已经被删除，无法编辑!","提示
```

```
",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
    e.Cancel = True '取消编辑
End If
```

注意这里用的是 StartEdit 事件，而不是 PrepareEdit 事件，也就是只有用户真正开始编辑数据时，才会执行上述代码。

我们来分析一下上述代码的工作原理，假定某行没有人编辑，张三和李四同时开始编辑此行，同时开始执行上述代码，如果张三抢先了一点，那么执行了 Update 语句之后，编辑者列的内容成为“张三”，由于 Update 语句设置了条件，只有编辑者列为空时，Update 语句才会执行，所以随后李四在执行 Update 语句时，由于已经没有符合条件的行，导致 ExecuteNonQuery 方法返回 0， 于是取消编辑，并用 Select 语句从后台找出正在编辑此行的用户名，告知李四。

4、将表事件 DataRowAdding 代码设置为：

```
e.DataRow("编辑者") = User.name
```

5、将表事件 BeforSaveDatarow 代码设置为：

```
If e.DataRow.RowState = DataRowState.Added Then '如果是新增行
    e.DataRow("编辑者") = Nothing
Else
    Dim cmd As New SQLCommand
    cmd.CommandText = "Update {员工} Set  编辑者  = Null Where [_Identify] = " & e.DataRow("_Identify")
    If cmd.ExecuteNonQuery = 1 Then
      e.DataRow("编辑者") = Nothing
    End If
End If
```

上述代码在保存行之前，清除此行的编辑者列内容，也就是解锁此行。

你也许会奇怪，为什么新增行可以直接清空编辑者列的内容，但其它行要通过Update语句设置？
假定现有一行数据，用户张三双击此行某个单元格开始编辑，系统执行StartEdit事件，将编辑者列的内容设置为“张三”，后台此列的内容也通过Update语句设置为了“张三”，如果此用户在编辑过程中，没有做任何修改，就选择了其他单元格，然后保存，此时如果在BeforeSaveDataRow事件中直接清空编辑者列内容，那么此行所有列的内容和最初完全一样，Foxtable将不会保存此行，所以虽然你清空了编辑者列的内容，但是后台的编辑者列还是“张三”，导致记录被锁死。
这里涉及到Foxtable的保存机制：在保存某行之前，系统会进行一个判断，如果此行所有列内容和最初完全相同，则跳过此行，以提高保存效率。

6、最后设置BeforeCloseProject事件代码为：

```
Dim cmd As New SQLCommand
cmd.CommandText = "Update {员工} Set  编辑者  = Null Where  编辑者  = '" & User.Name & "'"
```

```
cmd.ExecuteNonQuery
```

这样在关闭项目前，将清除当前用户锁定的全部行。

示例二

上面讲述的是直接在表中录入数据的时候，如何实现独占式编辑，下面讲述如何在窗口中实现独占式编辑。
本示例可以参考CaseStudy目录下的文件：独占式编辑之二.Table
请分别启动多个Foxtable打开此文件，并以不同的用户登录，只要有人双击某行开始编辑，那么直到此人结束编辑，其他人都不能编辑此行。

1、首先给表增加一个字符列，名为"编辑者"，用于记录正在编辑某行数据的用户名。
2、在项目事件 AfterOpenProject 设置代码：

```
'解锁非正常退出时被锁定的记录
DataTables("员工").ReplaceFor("编辑者",Nothing,"编辑者 = '" & User.Name & "'")
DataTables("员工").Save()
Tables("员工").Cols.Remove("编辑者") '移除编辑者列,避免用户直接修改此列
```

3、将表事件PrepareEdit代码设置为：

```
e.Cancel = True
```

4、将表事件 DataRowAdding 代码设置为：

```
e.DataRow("编辑者") = User.name
```

5、将表事件DoubleClick代码设置为：

```
If e.Row("编辑者") = User.Name Then '如果编辑者是当前用户
    Return
End If
Dim cmd As New SQLCommand
Dim exp As String  = "编辑者 Is Null And [_Identify] = " & e.Row("_Identify")
cmd.CommandText = "Update {员工} Set 编辑者 = '" & User.Name & "' Where " & exp
If cmd.ExecuteNonQuery = 1 Then
    e.Row("编辑者") = User.Name
    e.Row.Save()
    Forms("编辑窗口").Open()
Else
    cmd.CommandText = "Select 编辑者 From {员工} Where [_Identify] = " & e.Row("_Identify")
    Dim nm As String = cmd.ExecuteScalar
    If nm > "" Then
        MessageBox.show(nm & " 正在编辑此行!"," 提示
",MessageBoxButtons.OK,MessageBoxIcon.Information)
```

```
    Else
        MessageBox.show("此行可能已经被删除，无法编辑!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

上述代码的前面已经讲述，这里就不重复了。

6、设计一个下图所示的编辑窗口，注意在设计窗口的时候，应该将“显示控制框”属性设置为False，这样窗口就没有了关闭按钮，避免用户直接关闭窗口：



7、窗口“保存修改”按钮的代码为：

```
Tables("员工").Current("编辑者") = Nothing
Tables("员工").Current.Save()
e.Form.Close()
```

8、窗口“撤销修改”按钮的代码为：

```
Tables("员工").Current.Reject() '撤销编辑
Tables("员工").Current("编辑者") = Nothing
Tables("员工").Current.Save()
e.Form.Close()
```

这样不管用户保存还是撤销修改，编辑者列的内容都会被清除，让后来者得以继续编辑此行。

9、最后设置BeforeCloseProject事件代码为：

```
Dim cmd As New SQLCommand
cmd.CommandText = "Update {员工} Set  编辑者  = Null Where  编辑者  = '" & User.Name & "'"
cmd.ExecuteNonQuery
```

这样在关闭项目前，将清除当前用户锁定的全部行。

4.20.8 用户间通讯

提示：本节的内容已经过时，大家可以直接使用 Foxtable 内置的 OpenQQ 实现用户间的即时通讯功能。

在学习以下内容之前，可以先打开 CaseStudy 目录下的文件：用户通讯.Table

请启动两个 Foxtable 打开此文件，分别用张三和李四登录，互相发送信息，5 秒钟之后对方即可收到，具体时间可以自行调整，但不宜过短，因为时间间隔越短，服务器负载就越大，用户多的时候尤甚 。

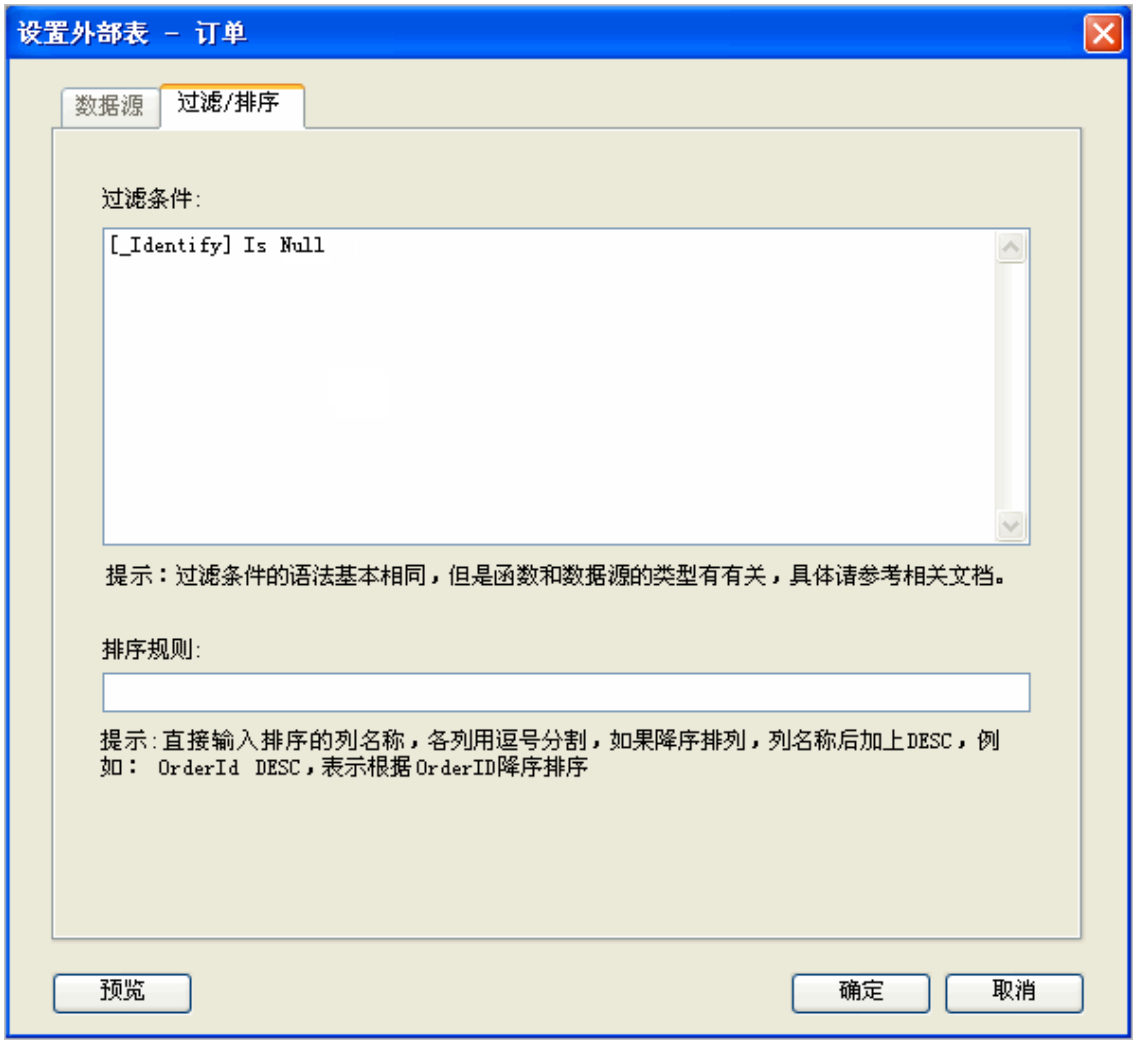
设计步骤：

1、首先通过代码或者设置，使得信息表在初始状态不加载任何数据。

如果是内部表，设置 BeforeLoadInnerTable 事件为：

```
If e.DataTableName = "信息表" Then
    e.Filter = "[_Identify] Is Null"
End If
```

如果是外部表，可以在定义外部表的时候，直接定义加载条件：



3、然后在项目事件 LoadUserSetting 事件中设置代码，只加载登录用户发送或接受的信息：

```
Dim Filter As String
Select Case User.Name
    Case "张三","李四", "王五"
        Filter = "接收者 = '" & User.Name & "' Or 发送者 = '" & User.Name & "'"
    Case Else
        Filter = "" '其他用户记载所有行
End Select
DataTables("信息表").LoadFilter = Filter
DataTables("信息表").Load()
Dim drs As List(of DataRow) = DataTables("信息表").Select("接收者 = '" & User.Name & "' And 接收时间 Is Null")
If drs.Count > 0 Then
    For Each dr As DataRow In drs
        dr("接收时间") = Date.Now
    Next
    DataTables("信息表").Save()
End If
```

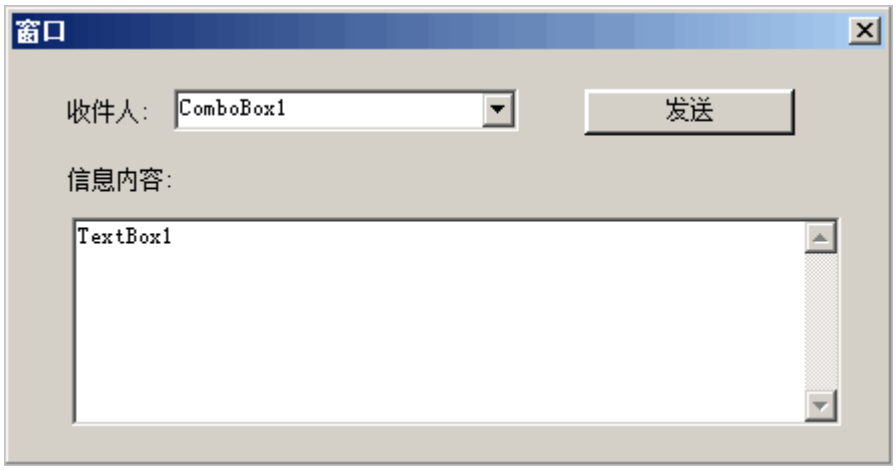
上面的代码只针对张三、李四、王五进行了设置，所以如果是其他用户登录，如开发者或管理员，

将加载所有行。

4、在 PrepareEdit 事件中设置代码，禁止用户直接编辑信息表：

```
e.Cancel = True
```

5、新建一个下图所示的窗口：



发送按钮的代码为：

```
Dim re As String = e.Form.Controls("ComboBox1").Value '获得接收用户名
Dim cn As String = e.Form.Controls("TextBox1").Text '获得发送内容
If re = "" OrElse cn = "" Then
    Return
End If
If re = User.Name Then
        MessageBox.Show("不能给自己发信息!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
Dim r As Row = Tables("信息表").AddNew()
r("发送者") = User.Name
r("发送时间") = Date.Now
r("接收者") = re
r("内容") = cn
r.Save()
```

6、新建一个计划，计划的执行间隔为 5 秒(即 5000 毫秒)，代码为：

```
Dim flt As String
Dim drs As List(Of DataRow)
flt = "接收者 = '" & User.Name & "' And 接收时间 Is Null"
drs = DataTables("信息表").AppendLoad(flt, False)
For Each dr As DataRow In drs
    dr("接收时间") = Date.Now()
    dr.Save()
Next
```

实际应用的时候，只要不影响工作时效，时间间隔应该尽可能长一些，以免用户较多的时候，服务器负载过重，毕竟我们设计的不是一个类似 QQ 的即时通讯系统，大多数时候只是用于内部通

知的发送而已 。

4.20.9 数据的自动流转之一

一般用户可以忽略本节内容。
本节的内容可以参考 CaseStudy 目录下的文件：自动流转一.Table
打开此文件的时候，请以张三、李四或王五登录。

我们用一个简单的例子来介绍如何在 Foxtable 实现数据的自动流转，所谓自动流转，就是本流程能自动接收上一流程处理完毕的数据，同时本流程处理完毕的数据，也能自动提交给下一流程，无需人工干预。
流程是无定式的，Foxtable 可以随心所欲地定义工作流程，本节只是介绍一些基本的方法和原理。
假定表 A 有九列，分三个流程处理，要求：
1、流程一由张三负责，输入第一列、第二列的数据。
2、流程二由李四负责，输入第三列、第四列的数据。
3、流程三由王五负责，输入第五列、第六列的数据。
4、上级流程处理完毕，才能处理本级流程，例如流程一处理完毕，才能处理流程二。
5、禁止李四和王五增加行，因为他们不是流程的初始发起者，只是后续流程的处理者。
6、只加载登录用户负责处理的行，例如李四负责流程二，所以李四登录后，只加载流程一处理完毕，但流程二未处理的行。
7、保存后移除(注意不是删除)处理完毕的行，并从后台追载新的待处理行。
8、如果所有行均处理完毕，那么定期从后台追载新的待处理行。
设计步骤：
1、给表增加一个名为“进度”的字符型列，用于记录流程的处理进度，此列的值和流程进度的对应关系为：

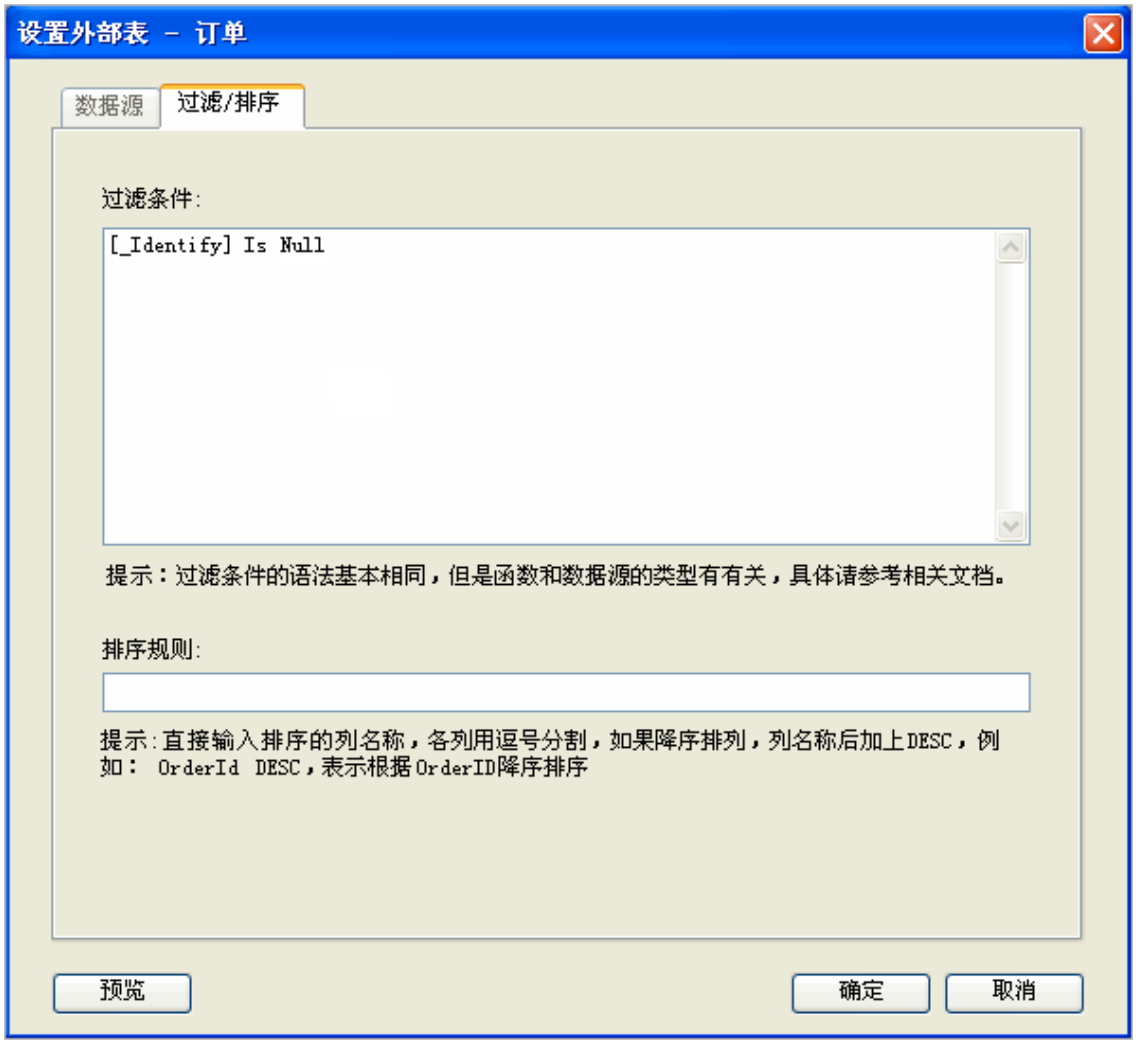
值	进度
1A	流程一开始
1B	流程一完毕
2A	流程二开始
2B	流程二完毕
3A	流程三开始
3B	流程三完毕

每个流程都有开始和结束标记，是行自动流转的需要。
例如新增行的初始流程状态为"1A"，张三处理完毕后，状态变为"1B"（表示流程一处理完毕）,李四检测到后台有状态为"1B"的行，将其追载到表中处理，并将其状态设置为"2A"(表示流程二已经开始)并保存。
假定没有开始标记，只有结束标记，当李四再次追载后台数据时，由于此行的状态还是"1B"，将无法判断此行是否已经追载，会影响行的自动流转效率。
2、为了只加载登录用户负责处理的行，首先必须确保默认不加载任何数据：
如果是内部表，设置项目事件 BeforeLoadInnerTable 的代码为：

```
If e.DataTableName = "表 A" Then
```

```
    e.Filter = "[_Identify] Is Null"
End If
```

如果是外部表，可以在定义外部表的时候，直接定义加载条件：



3、然后在项目事件 LoadUserSetting 中设置代码，加载由登录用户负责处理的行，并设置本流程开始标记：

```
'加载登录用户负责处理的行
Dim Filter As String
Dim bj As String
Select Case User.Name
    Case "张三"
        Filter = "进度 = '1A'" '进度一开始
    Case "李四"
        Filter = "进度 = '1B' Or 进度 = '2A'" '进度一结束或进度二开始
        bj = "2A"
    Case "王五"
        Filter = "进度 = '2B' Or 进度 = '3A'" '进度二结束或进度三开始
        bj = "3A"
    Case Else
        Filter = "" '其他用户加载全部记录
End Select
```

```
DataTables("表 A").LoadFilter = Filter
DataTables("表 A").Load()
'设置本流程开始标记
If bj >"" Then
    For Each dr As DataRow In DataTables("表 A").DataRows
        dr("进度") = bj
    Next
End If
DataTables("表 A").Save() '一定要保存，以更新后台的流程开始标记
```

4、每个用户只能编辑本流程范围内的列，且进度列是不允许任何人编辑的，为此将表的 PrepareEdit 事件代码设置为：

```
Select Case e.Col.Name
    Case "进度"
        e.Cancel = True
    Case "第一列","第二列"
        e.Cancel = (User.Name <> "张三")
    Case "第三列","第四列"
        e.Cancel = (User.Name <> "李四")
    Case "第五列","第六列"
        e.Cancel = (User.Name <> "王五")
End Select
```

5、为了禁止除张三之外的人增加行，将表的 BeforeAddDataRow 事件代码设置为：

```
If User.Name <> "张三" Then
    e.Cancel= True
End If
```

6、将表的 DataRowAdding 事件代码设置为：

```
e.DataRow("进度") = "1A"
```

这样新增行的时候，进度列的值默认为"1A"。

7、进度列的值需要根据其他列的输入进度来计算得出，为此将表的 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.Name
    Case "第一列","第二列","第三列","第四列","第五列","第六列"
        Dim dr As DataRow = e.DataRow
        Dim jd As String = "1A"
        If dr.IsNull("第一列") = False AndAlso dr.IsNull("第二列") = False Then
            jd = "1B"
            If dr.IsNull("第三列") = False AndAlso dr.IsNull("第四列") = False Then
                jd = "2B"
                If dr.IsNull("第五列") = False AndAlso dr.IsNull("第六列") = False Then
                    jd = "3B"
                End If
            End If
        End If
        If jd = "1B" AndAlso dr("进度") = "2A" Then '注意这里的处理技巧，意思是 2A 不能回到
```

```
1B，3A 不能回到 2B
        ElseIf jd = "2B" AndAlso dr("进度") = "3A" Then
        Else
            dr("进度") = jd
        End If
End Select
```

8、为了在保存后自动移除已经处理完毕的行，并加载新的待处理行，将项目事件 AfterSaveProject 事件代码设置为：

```
Dim Filter As String
Dim bj As String
Dim drs As List(Of DataRow)
'移除处理完毕的行
Select Case User.Name
    Case "张三"
        Filter = "进度  = '1B'"
    Case "李四"
        Filter = "进度  = '2B'"
    Case "王五"
        Filter = "进度  = '3B'"
    Case Else
        Return '其他用户正常返回
End Select
DataTables("表 A").RemoveFor(Filter)
'追载待处理的行
Select Case User.Name
    Case "李四"
        Filter = "进度  = '1B'"
        bj= "2A"
    Case "王五"
        Filter = "进度  = '2B'"
        bj = "3A"
    Case Else
        Return '其他用户正常返回
End Select
drs = DataTables("表 A").AppendLoad(Filter,False)
'为新追载的行设置流程开始标记
If drs.Count > 0 Then
    For Each dr As DataRow In drs
        dr("进度") = bj
        dr.Save() '一定要保存，以更新后台的流程进度标记
    Next
End If
```

提示：

1、没有数据要保存的时候，你也可以单击菜单中的保存按钮来追载新的待处理行，因为上面的代

码使得保存按钮具备了保存和追载的双重功能。

2、如果需要的话，也可以在菜单或者窗口增加一个按钮，单单用于追载待处理行，按钮代码为：

```
Dim Filter As String
Dim bj As String
Dim drs As List(Of DataRow)
Select Case User.Name
    Case "李四"
        Filter = "进度  = '1B'"
        bj= "2A"
    Case "王五"
        Filter = "进度  = '2B'"
        bj = "3A"
    Case Else
Return '其他用户不追载
End Select
drs = DataTables("表 A").AppendLoad(Filter,False) '追载待处理行
'为新追载的行设置流程开始标记
If drs.Count > 0 Then
    For Each dr As DataRow In drs
        dr("进度") = bj
        dr.Save() '一定要保存，以更新后台的流程进度标记
    Next
End If
```

9、在计划管理中新增一个计划，计划的执行间隔为 10 秒（即 10000 毫秒），代码为：

```
Dim Filter As String
Dim bj As String
Dim drs As List(Of DataRow)
If DataTables("表 A").DataRows.Count > 0 Then
    Return
End If
Select Case User.Name
    Case "李四"
        Filter = "进度  = '1B'"
        bj= "2A"
    Case "王五"
        Filter = "进度  = '2B'"
        bj = "3A"
    Case Else
Return '其他用户不追载
End Select
drs = DataTables("表 A").AppendLoad(Filter,False) '追载待处理行
If drs.Count > 0 Then '设置新流程的开始标记
    For Each dr As DataRow In drs
        dr("进度") = bj
```

```
        dr.Save() '一定要保存 ，以更新后台的流程进度标记
    Next
End If
```

上面的代码就会每隔 10 秒执行一次，如果当前表已经没有数据，就从后台追载新的待处理行，并为新追载行设置本流程开始标记。
具体间隔时间可以根据需要调整，但不宜过短，以免服务器的负载过重。

利用消息推送实现即时刷新

前述代码利用计划，每隔 10 秒追载一下待处理的数据，用户量大的时候，这种定时轮询的方式会导致服务器负载过重。
Foxtable 2016 增加了消息推送工具 OpenQQ，可以通过代码进行信息的自动收发。
这样负责上一流程的用户处理完成之后，可以给负责下一流程的用户给发一个约定格式的信号，该用户收到信号后，立即自动追载新的数据。
这种主动追载数据的方式，不仅即时，而且高效，具体实现可以参考帮助文件《消息推送》这一章。

4.20.10 数据的自动流转之二

一般用户可以忽略本节内容。
本节的内容可以参考 CaseStudy 目录下的文件：自动流转二.Table
打开此文件的时候，请以张三、李四、王五或赵六登录。

上一节介绍的自动流转，是批量处理、批量保存、批量流转的。
本节介绍的自动流转是逐行处理、逐行保存，逐行流转的。
本节对于流程定义与上节基本相同，只是增加了一个流程：
1、流程一由张三负责，输入第一列、第二列的数据。
2、流程二由李四负责，输入第三列、第四列的数据。
3、流程三由王五负责，输入第五列、第六列的数据。
4、流程四由赵六负责，输入第七列、第八列的数据。
4、上级流程处理完毕，才能处理本级流程，例如流程一处理完毕，才能处理流程二。
5、禁止李四、王五、赵六增加行，因为他们不是流程的初始发起者，只是后续流程的处理者。
6、只加载登录用户负责处理的行，例如李四负责流程二，所以李四登录后，只加载流程一处理完毕，但流程二未处理的行。
但要求：
1、禁止直接在表中编辑数据，改用录入窗口形式编辑。
2、双击表中的任何一行，即可打开录入窗口编辑此行(即处理本流程)。
3、每次只能处理一条记录（行）
4、在保存或取消此记录的修改前，不能移到其他记录，也不能关闭窗口。
5、单击窗口中的"保存"按钮，保存对当前行的修改，如果此行已经处理完毕，则移除此行（注意不是删除）。
6、单击窗口中的"撤销"按钮，撤销对当前行的修改，所有列返回修改前的值。
7、单击窗口中的"追载"按钮，从后台追载新的待处理记录。
8、单击窗口中的"打回"按钮，将当前行返回上一流程重新处理。

9、如果所有行均处理完毕，那么定期从后台追载新的待处理行。

设计步骤：

1、给表增加一个名为“进度”的字符型列，用于记录流程的处理进度，此列的值和流程进度的对应关系为：

值	进度
1A	流程一开始
1B	流程一完毕
2A	流程二开始
2B	流程二完毕
3A	流程三开始
3B	流程三完毕
4A	流程四开始
4B	流程四完毕

每个流程都有开始和结束标记，是行自动流转的需要。

例如新增行的初始流程状态为"1A"，张三处理完毕后，状态变为"1B"（表示流程一处理完毕）,李四检测到后台有状态为"1B"的行，将其追载到表中处理，并将其状态设置为"2A"(表示流程二开始)并保存。

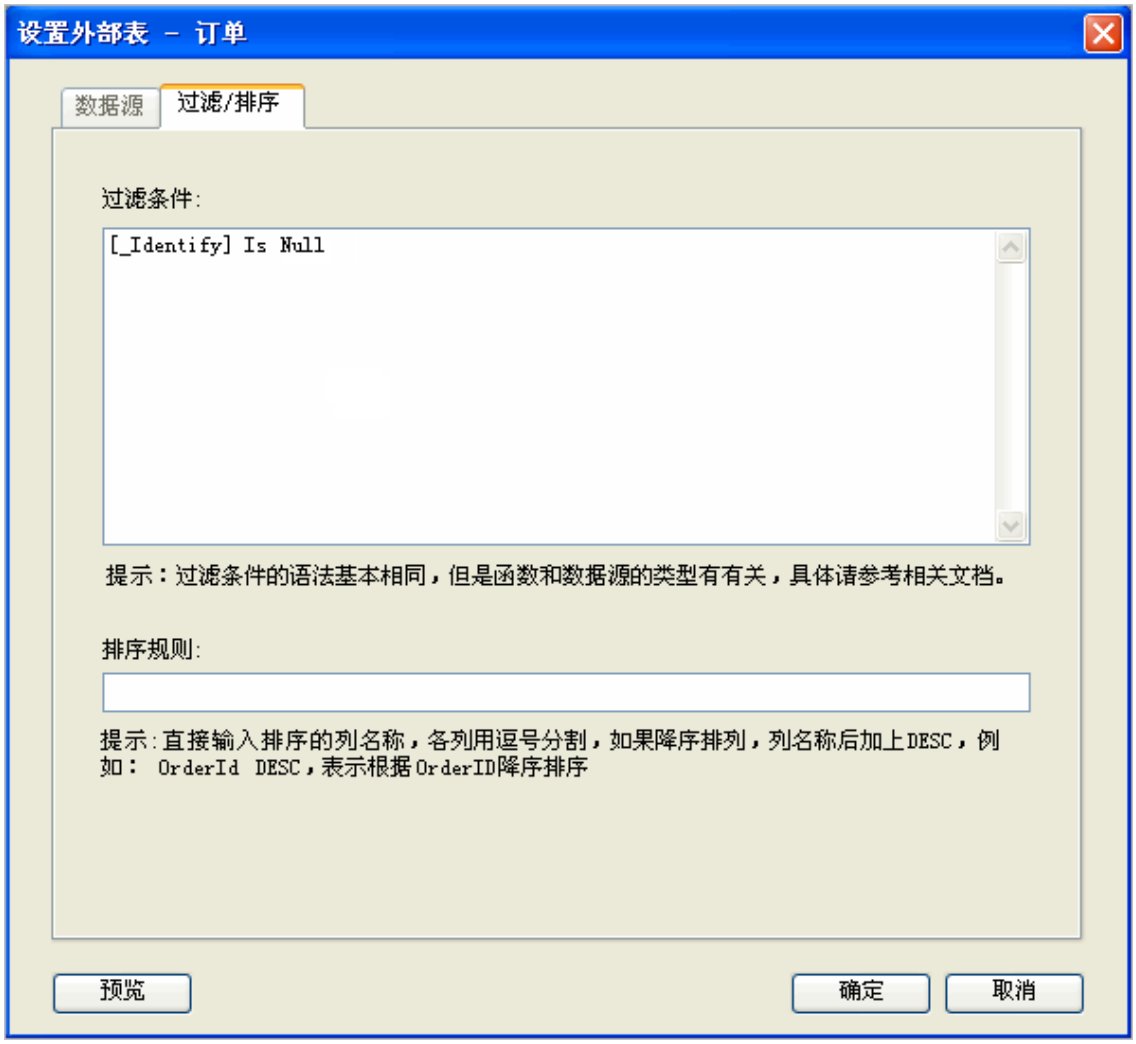
假定没有开始标记，只有结束标记，当李四再次追载后台数据时，由于此行的状态还是"1B"，将无法判断此行是否已经追载，影响行的自动流转效率。

2、为了只加载登录用户负责处理的行，首先必须确保默认不加载任何数据：

如果是内部表，设置项目事件 BeforeLoadInnerTable 的代码为：

```
If e.DataTableName = "表 A" Then
    e.Filter = "[_Identify] Is Null"
End If
```

如果是外部表，可以在定义外部表的时候，直接定义加载条件：



3、然后在项目事件 LoadUserSetting 中设置代码，加载由登录用户负责处理的行，并设置本流程开始标记：

```
'加载登录用户负责处理的行
Dim Filter As String
Dim bj As String
Select Case User.Name
    Case "张三"
        Filter = "进度  Is Null Or  进度  = '1A'" '进度为空或进度一开始
        bj = "1A"
    Case "李四"
        Filter = "进度  = '1B' Or  进度  = '2A'" '进度一结束或进度二开始
        bj = "2A"
    Case "王五"
        Filter = "进  = '2B' Or  进度  = '3A'" '进度二结束或进度三开始
        bj = "3A"
    Case "赵六"
        Filter = "进度  = '3B' Or  进度  = '4A'" '进度三结束或进度四开始
        bj = "4A"
    Case Else
```

```
        Filter = "" '其他用户加载全部记录
End Select
DataTables("表 A").LoadFilter = Filter
DataTables("表 A").Load()
'设置本流程开始标记
If bj >"" Then
    For Each dr As DataRow In DataTables("表 A").DataRows
        dr("进度") = bj
    Next
End If
DataTables("表 A").Save() '一定要保存,以更新后台的流程开始标记
```

4、禁止直接在表中输入数据，为此将表的 PrepareEdit 事件代码设置为：

```
e.Cancel = True
```

5、为了禁止除张三之外的人增加行，将表的 BeforeAddDataRow 事件代码设置为：

```
If User.Name <> "张三" Then
    e.Cancel= True
End If
```

6、将表的 DataRowAdding 事件代码设置为：

```
e.DataRow("进度") = "1A"
```

这样新增行的时候，进度列的值默认为"1A"。

7、在计划管理中新增一个计划，计划的执行间隔为 10 秒（即 10000 毫秒），代码为：

```
Dim Filter As String
Dim bj As String
Dim drs As List(Of DataRow)
If DataTables("表 A").DataRows.Count > 0 Then
    Return
End If
Select Case User.Name
    Case "张三"
        Filter = "进度  Is Null" '进度为空
        bj = "1A"
    Case "李四"
        Filter = "进度  = '1B'" '进度一结束
        bj = "2A"
    Case "王五"
        Filter = "进度  = '2B'" '进度二结束
        bj = "3A"
    Case "赵六"
        Filter = "进度  = '3B'" '进度三结束
        bj = "4A"
    Case Else
        Return '其他用户不追载
End Select
drs = DataTables("表 A").AppendLoad(Filter,False) '追载待处理行
```

```
If drs.Count > 0 Then '设置新流程的开始标记
    For Each dr As DataRow In drs
        dr("进度") = bj
        dr.Save() '一定要保存,以更新后台的流程开始标记
    Next
End If
```

上面的代码就会每隔 10 秒执行一次，如果当前表已经没有数据，就从后台追载新的待处理行，并为新追载行设置本流程开始标记。

具体间隔时间可以根据需要调整，但不宜过短，以免服务器的负载过重。

利用消息推送实现即时刷新

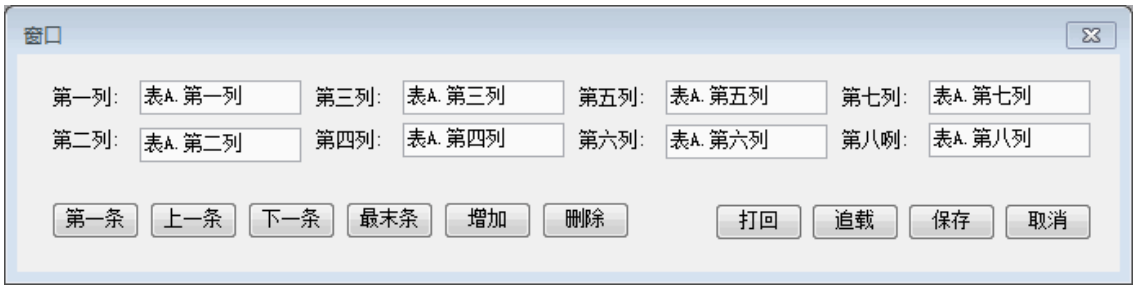
前述代码利用计划，每隔 10 秒追载一下待处理的数据，用户量大的时候，这种定时轮询的方式会导致服务器负载过重。

Foxtable 2016 内置了消息推送工具 OpenQQ，可以通过代码进行信息的自动收发。

这样负责上一流程的用户处理完成之后，可以给负责下一流程的用户给发一个约定格式的信号，该用户收到信号后，立即自动追载新的数据。

这种主动追载数据的方式，不仅即时，而且高效，具体实现可以参考帮助文件《消息推送》这一章。

8、新建一个下图所示的窗口：



为了编码方便，八个文本框的名称和其绑定的列名保持一致，例如第一个文本框绑定到第一列，其名称也应该设置为第一列：



将窗口的 AfterLoad 事件代码设置为：

```
e.Form.Controls("第一列").Enabled = (User.Name = "张三")
e.Form.Controls("第二列").Enabled = (User.Name = "张三")
e.Form.Controls("第三列").Enabled = (User.Name = "李四")
e.Form.Controls("第四列").Enabled = (User.Name = "李四")
e.Form.Controls("第五列").Enabled = (User.Name = "王五")
e.Form.Controls("第六列").Enabled = (User.Name = "王五")
```

```
e.Form.Controls("第七列").Enabled = (User.Name = "赵六")
e.Form.Controls("第八列").Enabled = (User.Name = "赵六")
```

上述代码确保每个用户只能编辑自己负责的列。

窗口的 BeforeClose 事件设置为：

```
If Tables("表 A").Current.DataRow.RowState <> DataRowState.Unchanged Then '如果当前行已经修改过
    e.Cancel = True
End If
```

上面的代码在关闭窗口前执行，如果当前记录(行)是已经修改但没有保存，将禁止关闭窗口，直到保存或撤销修改。

窗口中各按钮的代码：

按钮	代码

上一条	With Tables("表 A")
	  If .Current IsNot Nothing Then
	  If .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = .Position - 1
	  End If
	  End If
	End With

下一条	With Tables("表 A")
	  If .Current IsNot Nothing Then
	  If .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = .Position + 1
	  End If
	  End If
	End With

第一条	With Tables("表 A")
	  If .Current IsNot Nothing Then
	  If .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = 0
	  End If
	  End If
	End With

最末条	With Tables("表 A")
	  If .Current IsNot Nothing Then
	  If .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .Position = .Rows.Count - 1
	  End If
	  End If

	End With

新增	With Tables("表 A")
	  If .Current Is Nothing OrElse .Current.DataRow.RowState = DataRowState.Unchanged Then '如果当前行未曾修改
	  .AddNew()
	  End If
	End With

删除	With Tables("表 A")
	  If .Current IsNot Nothing Then
	  .Current.Delete()
	  End If
	End With

追载	Dim Filter As String
	Dim bj As String
	Dim drs As List(Of DataRow)
	Select Case User.Name
	  Case "张三"
	  Filter = "进度 Is Null" '进度为空
	  bj = "1A"
	  Case "李四"
	  Filter = "进度 = '1B'" '进度一结束
	  bj = "2A"
	  Case "王五"
	  Filter = "进度 = '2B'" '进度二结束
	  bj = "3A"
	  Case "赵六"
	  Filter = "进度 = '3B'" '进度三结束
	  bj = "4A"
	  Case Else
	  Return '其他用户不追载
	End Select
	drs = DataTables("表 A").AppendLoad(Filter,False) '追载待处理行
	If drs.Count > 0 Then '设置新流程的开始标记
	  For Each dr As DataRow In drs
	  dr("进度") = bj
	  dr.Save()
	  Next
	End If

打回	Dim r As Row = Tables("表 A").Current
	If r IsNot Nothing
	  Select Case User.Name
	  Case "李四"
	  If r("进度") = "2A" Then '进度 3 打回的时候,将进度标记设置为 Nothing,

	意思是进度 1 处理无效,需要重新处理. r("进度") = Nothing r("第三列") = Nothing r("第四列") = Nothing r.Save() r.Remove() End If Case "王五" If r("进度") = "3A" Then '进度 3 打回的时候,将进度标记设置为 1B,意思是进度 1 已经完成,进度 2 重新处理. r("进度") = "1B" r("第五列") = Nothing r("第六列") = Nothing r.Save() r.Remove() End If Case "赵六" If r("进度") = "4A" Then '进度 4 打回的时候,将进度标记设置为 2B,意思是进度 2 已经完成,进度 3 重新处理. r("进度") = "2B" r("第七列") = Nothing r("第八列") = Nothing r.Save() r.Remove() End If End Select End If 说明： 以进度 3 打回给进度 2 为例，需要将进度调整为“1B”，也许你会感到奇怪，为什么不直接将进度调整为“2A”？ 这是为了方便负责进度 2 的用户及时追载打回的行，因为负责进度 2 的用户会定期追载进度为“1B”的行，也就是说那些进度 1 已经完成但是进度 2 还未开始的行；如果进度 3 打回给进度 2 的时候，直接将进度调整为“2A”，那么负责负责进度 2 的用户只有重新打开项目，才能看到这些被打回的行。

保存	Dim r As Row = Tables("表 A").Current
	If r IsNot Nothing Then
	Dim jd As String = "1A"
	If r.IsNull("第一列") = False AndAlso r.IsNull("第二列") = False Then
	jd = "1B"
	If r.IsNull("第三列") = False AndAlso r.IsNull("第四列") = False Then
	jd = "2B"
	If r.IsNull("第五列") = False AndAlso r.IsNull("第六列") = False Then
	jd = "3B"
	If r.IsNull("第七列") = False AndAlso r.IsNull("第八列") = False Then
	jd = "4B"
	End If

	End If
	End If
	End If
	If jd = "1B" AndAlso r("进度") = "2A" Then
	ElseIf jd = "2B" AndAlso r("进度") = "3A" Then
	ElseIf jd = "3B" AndAlso r("进度") = "4A" Then
	Else
	r("进度") = jd
	End If
	r.Save()
	Select Case User.Name
	Case "张三"
	If r("进度") = "1B" Then
	r.Remove
	End If
	Case "李四"
	If r("进度") = "2B" Then
	r.Remove
	End If
	Case "王五"
	If r("进度") = "3B" Then
	r.Remove
	End If
	Case "赵六"
	If r("进度") = "4B" Then
	r.Remove
	End If
	Case Else
	Return '其他用户正常返回
	End Select
	End If

取消	With Tables("表 A")
	If.Current IsNot Nothing Then
	.Current.Reject()
	End If
	End With

从上面的代码可以看出，如果当前记录(行)已经修改但没有保存，窗口中的上一条、下一条、第一条、最末条、新增按钮将拒绝执行，直到保存或撤销修改。

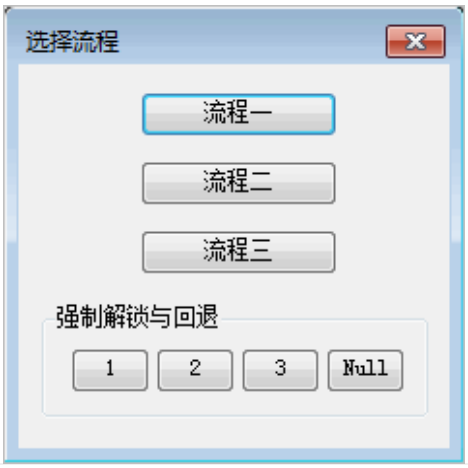
10、将表的 DoubleClick 事件代码设置为：

Forms("窗口 1").Open()

4.20.11 工作流和并发冲突

普通用户可以忽略本节内容。
提示：
本节内容在编写的时候，Foxtable 还没有内置 OpenQQ，无法实现消息的主动推送和接收。所以本节的内容其实是有些过时的。
从 Foxtable 2016 开始，我们可以 OpenQQ 来实现工作流，并解决并发冲突的问题，且更加方便可靠，请参考：OpenQQ、工作流与并发冲突
尽管过时，但是本节内容依然有参考价值，有兴趣的用户可以学习一下。

本节的例子可以参考 CaseStudy 目录下的文件：工作流与并发冲突.Table
在浏览本节内容的时候，建议将帮助窗口最大化。
之前介绍的工作流，都是一人负责个流程，有时工作流会更复杂一些，例如一项工作分为多个流程，每一个流程会有多人负责，此时就需要考虑“抢先”的问题。例如 A、B、C 三人都有权限处理某一流程，但同一时候只能有一人处理，假定 A 用户已经“抢先”处理此流程，当 B 用户和 C 用户尝试处理同一流程的时候，系统必须能够通知两位后来者：已经有人在处理此流程了 ，这就是所谓的并发冲突。
本节我们用一个简单例子来说明如何将工作流和并发冲突相结合，其中并发冲突的处理技巧，和数据源无关，正因为不需要借助特定数据源的功能，所以通用性很强。
假定一个表有九列，分成三个工作流程来处理：
流程 1 处理一、二、三列
流程 2 处理四、五、六列
流程 3 处理七、八、九列
只有在上一流程已经处理完毕的情况下，才可以处理下一流程，例如只有输入一、二、三列的值后，才能输入四、五、六列的值。
本次设计要完成的任务为：
1、在针对某行处理一个流程前，首先自动检索该行的最新数据和流程处理进度，以决定能否处理此流程。
2、如果能够处理此流程，那么在开始处理之前先锁定此行，使得其他用户不能再处理此行，避免出现多个用户同时处理某一行的情况。
3、用户在处理完一个流程后，在保存处理结果的同时，还应该解锁行，使得其他用户可以处理此行。
4、提供强制解锁和流程回退功能，当某行出现死锁的时候，可以将某行解锁并强行回退到任意流程。
提示：这里的锁定行，和我们之间介绍的锁定行是不同的，这里是在后台锁定某个行，使得其他用户不能编辑此行。
设计步骤：
1、首先我们需要加一个整数型的标记列， 标记列的值等于 1、2、3 的时候，分别表示已经处理完成流程 1、流程 2、流程 3，当标志列的值等于-1 的时候，表示有人正在处理此行，也就是被锁定了。
2、将表锁定，避免用户直接在表中输入数据
3、增加一个选择流程的窗口，窗口的设计如下：



按钮名称	代码

流程一	Dim r As Row = Tables("表 A").Current
	r.DataRow.Load() '重新加载此行的数据
	If r.IsNull("标记") Then
	Dim cmd as New SQLCommand
	cmd.CommandText = "Update {表 A} Set 标记 = -1 Where 标记 Is Null And [_Identify] = " & r("_Identify")
	If cmd.ExecuteNonQuery = 1 Then '防止并发冲突,上面的条件是很关键的
	Forms("流程 1").Open()
	Else
	MessageBox.Show("其他用户已经抢先处理此行!")
	End If
	ElseIf r("标记") = -1 Then
	Messagebox.Show("其他用户正在处理此行!")
	Else
	MessageBox.Show("流程 1 已经完成!")
	End If

流程二	Dim r As Row = Tables("表 A").Current
	r.DataRow.Load() '重新加载此行的数据
	If r.IsNull("标记") Then
	MessageBox.Show("流程 1 没有完成!")
	ElseIf r("标记") = 1 Then
	Dim cmd as New SQLCommand
	cmd.CommandText = "Update {表 A} Set 标记 = -1 Where 标记 = 1 And [_Identify] = " & r("_Identify")
	If cmd.ExecuteNonQuery = 1 Then '防止并发冲突,上面的条件是很关键的
	Forms("流程 2").Open()
	Else
	MessageBox.Show("其他用户已经抢先处理此行!")
	End If
	ElseIf r("标记") = -1 Then

	Messagebox.Show("其他用户正在处理此行!")
	Else
	MessageBox.Show("流程 2 已经完成!")
	End If

流程三	Dim r As Row = Tables("表 A").Current
	r.DataRow.Load() '重新加载此行的数据
	If r.IsNull("标记") OrElse r("标记") = 1 Then
	MessageBox.Show("流程 2 没有完成!")
	ElseIf r("标记") = 2 Then
	Dim cmd as New SQLCommand
	cmd.CommandText = "Update {表 A} Set 标记 = -1 Where 标记 = 2 And [_Identify] = " & r("_Identify")
	If cmd.ExecuteNonQuery = 1 Then '防止并发冲突,上面的条件是很关键的
	Forms("流程 3").Open()
	Else
	MessageBox.Show("其他用户已经抢先处理此行!")
	End If
	ElseIf r("标记") = -1 Then
	Messagebox.Show("其他用户正在处理此行!")
	Else
	MessageBox.Show("流程 3 已经完成!")
	End If

强制解锁与流程回退(1)	'强制解锁,并回退到流程 1 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = 1 Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()
强制解锁与流程回退(2)	'强制解锁,并回退到流程 2 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = 2 Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()
强制解锁与流程回退(3)	'强制解锁,并回退到流程 3 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = 3 Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()
强制解锁	'强制解锁,并回退到初始状态 Dim dr As DataRow = Tables("表 A").Current.DataRow

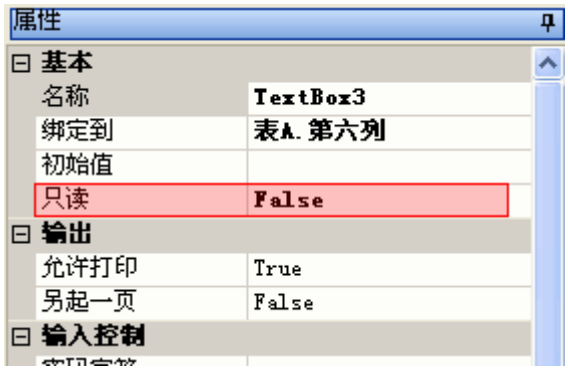
与流程回退(Null)	Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = Null Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()

提示：SQLCommand 的 ExecuteNonQuery 命令会返回一个整数值，表示受此命令影响的行数。

4、增加三个模式窗口，分别为流程 1、流程 2、流程 3，用于处理各个流程，每个流程窗口的设计都是类似的，用于输入该流程 应该处理的数据，例如流程 2 窗口的设计为：



由于表是锁定的，所以流程窗口中文本框的只读属性应该设置为 False，否则不能输入数据：



给流程窗口的按钮设置代码，以流程 2 窗口为例：

按钮名称	代码

确定	With Tables("表 A").Current
	  If .Isnull("第四列") OrElse .IsNull("第五列") OrElse .IsNull("第六列") Then
	  MessageBox.Show("流程二没有处理完!")
	  Return
	  End If
	End With
	Tables("表 A").Current("标记") = 2
	DataTables("表 A").Save()
	e.Form.Close()

取消	Tables("表 A").Current.Reject() Tables("表 A").Current("标记") = 1 DataTables("表 A").Save()

	e.Form.Close()

提示：为了防止用户跳过确定和取消按钮，直接关闭窗口，流程 1、流程 2、流程 3 三个窗口的“显示控制框”属性设置为 False，这样窗口标题栏就不会出现关闭按钮：





防止并发冲突的原理

至此我们这个简单的工作流系统已经设计完成，本系统不仅能够实现工作流，而且可以防止并发冲突，从某用户开始编辑某行数据开始，任何人都不能编辑此行数据，直到该用户结束或取消编辑。

其实现的原理很简单，以某个用户准备对某行数据进行流程 2 处理为例，用户单击流程选择窗口的流程 2 按钮后，执行的代码为：

```
Dim r As Row = Tables("表 A").Current
r.DataRow.Load() '重新加载此行的数据
If r.IsNull("标记") Then
    MessageBox.Show("流程 1 没有完成!")
ElseIf r("标记") = 1 Then
    Dim cmd as New SQLCommand
    cmd.CommandText = "Update {表 A} Set  标记  = -1 Where  标记  = 1 And [_Identify] = " &
r("_Identify")
    If cmd.ExecuteNonQuery = 1 Then '防止并发冲突,上面的条件是很关键的
      Forms("流程 2").Open()
    Else
      MessageBox.Show("其他用户已经抢先处理此行!")
    End If
ElseIf r("标记") = -1 Then
    Messagebox.Show("其他用户正在处理此行!")
```

```
Else
   MessageBox.Show("流程 2 已经完成!")
End If
```

为便于讲解，我们对上面的代码进行了编号：

1、第 2 行代码重新加载此行，获得最新的数据和流程处理进度。

2、第 3 行代码判断流程进度是否为空，如果为空表示流程 1 没有处理完成，所以不能处理流程 2；第 5 行代码判断当前进度是否等于 1，如果等于 1，表示可以处理流程 2。

3、第 6、7、8 行是处理并发冲突的关键代码，用 Update 语句将当前行的标记列设置为-1，等于通知其他用户，我开始处理此行了；Update 语句的条件设置是至关重要的，主键值条件的加入是当然的，也是必须的，大家都能理解，最关键的是“标记 = 1”的条件；假定 A 和 B 用户几乎在同一时间对同一行进行流程 2 处理，两个人在重新加载此行后，都得到了此行目前流程处理进度为 1 的信息，可以进行流程 2 的处理，于是两个人都试图用 Update 语句将当前行的标记设置为-1，但是只有一个人的 ExecuteNonQuery 方法(第 8 行)返回值会等于-1；例如假定 A 用户抢先执行了第 8 行的代码，ExecuteNonQuery 的返回值等于 1，于是正常开始流程 2 的处理，B 用户随后同样执行第 8 行代码，但由于此时标记列的内容 已经等于-1，不符合第 7 行设置的条件之一“标记 = 1”，所以不会有任何符合条件的行，于是第 8 行 ExecuteNonQuery 方法的返回值不等于 1，不能正常进入流程 2 的处理。

上面是进入流程的处理， 下面讲讲退出流程的处理，以流程 2 窗口为例，该窗口的确定按钮代码为：

```
With Tables("表 A").Current
   If .Isnull("第四列") OrElse .IsNull("第五列") OrElse .IsNull("第六列") Then
      MessageBox.Show("流程二没有处理完!")
      Return
   End If
End With
Tables("表 A").Current("标记") = 2
DataTables("表 A").Save()
e.Form.Close()
```

上面的代码首先判断流程 2 是否正常处理完毕，然后在第 7 行代码将当前行标记列的值设置为 2，并保存数据，等于告诉其他用户：我已经处理完流程 2 了，你们可以进行后续处理工作了。

流程 2 窗口取消按钮的代码为：

```
Tables("表 A").Current.Reject()
Tables("表 A").Current("标记") = 1
DataTables("表 A").Save()
e.Form.Close()
```

第 1 行代码撤销对当前行的修改，第 2 行代码将标记列的值设置为 1，并保存数据，等于告诉其他

用户：我放弃处理流程 2 了，流程进度重新回到 1，你们可以继续处理此行了。

强制解锁

如果某个用户正在针对某行处理一个流程，此时因为断电或者其它原因导致非正常退出，那么此行的标记列内容将一直等于-1，任何人将不能再处理此行了，所以我们还应该提供“解锁”的功能，强行将标志列的内容设置为某个值。

例如下面的代码，将标志列的内容强行设置为 2：

```
'强制解锁,并将流程设置为 2
Dim dr As DataRow = Tables("表 A").Current.DataRow
Dim cmd as New SQLCommand
cmd.CommandText = "Update {表 A} Set  标记  = 2 Where [_Identify] = " & dr("_Identify")
cmd.ExecuteNonQuery
dr.Load()
```

提示：实际应用的时候，应该设置为只有特定用户才能执行强制解锁功能。

4.20.12 OpenQQ、工作流与并发冲突

普通用户可以忽略本节的内容。

本节的任务和上一节是一样的，但是本节将结合 OpenQQ 来实现。

之前介绍的工作流，都是一人负责个流程，有时工作流会更复杂一些，例如一项工作分为多个流程，每一个流程会有多人负责，此时就需要考虑“抢先”的问题。例如 A、B、C 三人都有权限处理某一流程，但同一时候只能有一人处理，假定 A 用户已经“抢先”处理此流程，当 B 用户和 C 用户尝试处理同一流程的时候，系统必须能够通知两位后来者：已经有人在处理此流程了 ，这就是所谓的并发冲突。

本节我们用一个简单例子来说明如何将工作流和并发冲突相结合，其中并发冲突的处理技巧，和数据源无关，正因为不需要借助特定数据源的功能，所以通用性很强。

假定一个表有九列，分成三个工作流程来处理：

流程 1 处理一、二、三列

流程 2 处理四、五、六列

流程 3 处理七、八、九列

只有在上一流程已经处理完毕的情况下，才可以处理下一流程，例如只有输入一、二、三列的值后，才能输入四、五、六列的值。

本次设计要完成的任务为：

1、在针对某行处理一个流程前，首先自动检索该行的最新数据和流程处理进度，以决定能否处理此流程。

2、如果能够处理此流程，那么在开始处理之前先锁定此行，使得其他用户不能再处理此行，避免出现多个用户同时处理某一行的情况。

3、用户在处理完一个流程后，在保存处理结果的同时，还应该解锁行，使得其他用户可以处理此行。

4、提供流程回退功能，可以将某行回退到任意流程。

提示：

1、这里的锁定行，和我们之间介绍的锁定行是不同的，这里是在后台锁定某个行，使得其他用户不能编辑此行。

2、本节采用 OpenQQ 解决并发问题，不会出现死锁情况。

服务端的设计

1、在服务端项目的全局代码中加上代码：

```
Public pdids As New Dictionary(Of Integer, String)
```

这里定义了一个字典，键值是行的主键，值是正在处理这行的用户名。

2、在服务端的 OpenQQ 服务端事件 ReceivedMessage 中编写代码：

```
Dim msg As String = e.Message
If msg.StartsWith("%i") AndAlso msg.EndsWith("i%") Then '收到申请信号
    msg = msg.SubString(2, msg.Length - 4)
    Dim id As Integer
    If Integer.TryParse(msg,id) Then
      If pdids.ContainsKey(id) Then '如果有人正在处理此行
        e.ReturnValue = pdids(id) '通知申请人是谁在处理此行
      Else
        e.ReturnValue = "OK" '通知申请人可以处理此行
        pdids.Add(id, e.UserName) '登记此行  的处理者为申请者
      End If
    End If
ElseIf msg.StartsWith("@i") AndAlso msg.EndsWith("i@") Then '收到处理完成信号
    msg = msg.SubString(2, msg.Length - 4)
    Dim id As Integer
    If Integer.TryParse(msg,id) Then
      If pdids.ContainsKey(id) Then
        pdids.Remove(id) '从字典中移除此行的编辑登记
      End If
    End If
End If
```

3、在服务端的OpenQQ服务端事件UserLogout中编写代码：

```
Dim keys As New List(of Integer)
For Each key As Integer In pdids.Keys
    If pdids(key) = e.UserName Then
      keys.Add(key)
    End If
Next
For Each key As Integer In keys
    pdids.Remove(key)
Next
```

这样当有用户退出登录时，不管他是正常退出还是异常退出，都可以将字典中该用户的编辑登记移除，避免死锁。

客户端用户异常退出时，服务端的编辑登记的清除会有一个延时，时长取决于QQServer的HeartbeatTimeout(心跳超时)属性。

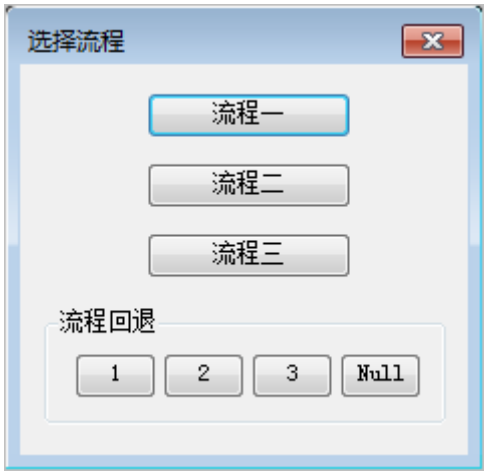
上述编码有一个地方需要注意，我们不能在遍历字典和集合的过程中移除其成员(运行时会报错)，所以我们用一个临时的集合Keys，先将要移除的键值保存在这个集合中，最后遍历这个集合中的键值，从字典pdids中移除这些键值。

客户端的设计

1、首先我们需要加一个整数型的标记列， 标记列的值等于 1、2、3 的时候，分别表示已经处理完成流程 1、流程 2、流程 3。

2、将表锁定，避免用户直接在表中输入数据

3、增加一个选择流程的窗口，窗口的设计如下：



按钮名称	代码

流程一	If QQClient.Ready = False Then
	PopMessage("请先启东QQClient","提示",PopIconEnum.Infomation,5)
	Return
	End If
	Dim r As Row = Tables("表A").Current
	r.DataRow.Load() '重新加载此行的数据
	If r.IsNull("标记") Then
	Dim msg As String = QQClient.SendWait("%i" & r("_Identify") & "i%")
	If msg = "OK" Then
	Forms("流程1").Open() '开始处理流程1
	ElseIf msg > "" Then
	PopMessage(msg & "正在处理此行","提示",PopIconEnum.Infomation,5)
	Else
	PopMessage("服务器无响应","提示",PopIconEnum.Infomation,5)
	End If
	Else
	PopMessage("流程1已经处理完成","提示",PopIconEnum.Infomation,5)
	End If

流程二	If QQClient.Ready = False Then PopMessage("请先启东QQClient","提示",PopIconEnum.Infomation,5) Return

	End If
	Dim r As Row = Tables("表A").Current
	r.DataRow.Load() '重新加载此行的数据
	If r.IsNull("标记") Then
	MessageBox.Show("流程1没有完成!")
	ElseIf r("标记") = 1 Then
	Dim msg As String = QQClient.SendWait("%i" & r("_Identify") & "i%")
	If msg = "OK" Then
	Forms("流程2").Open() '开始处理流程2
	ElseIf msg > "" Then
	PopMessage(msg & "正在处理此行","提示",PopIconEnum.Infomation,5)
	Else
	PopMessage("服务器无响应","提示",PopIconEnum.Infomation,5)
	End If
	Else
	MessageBox.Show("流程2已经完成!")
	End If

流程三	If QQClient.Ready = False Then
	PopMessage("请先启东QQClient","提示",PopIconEnum.Infomation,5)
	Return
	End If
	Dim r As Row = Tables("表A").Current
	r.DataRow.Load() '重新加载此行的数据
	If r.IsNull("标记") OrElse r("标记") = 1 Then
	MessageBox.Show("流程2没有完成!")
	ElseIf r("标记") = 2 Then
	Dim msg As String = QQClient.SendWait("%i" & r("_Identify") & "i%")
	If msg = "OK" Then
	Forms("流程3").Open() '开始处理流程3
	ElseIf msg > "" Then
	PopMessage(msg & "正在处理此行","提示",PopIconEnum.Infomation,5)
	Else
	PopMessage("服务器无响应","提示",PopIconEnum.Infomation,5)
	End If
	Else
	MessageBox.Show("流程3已经完成!")
	End If

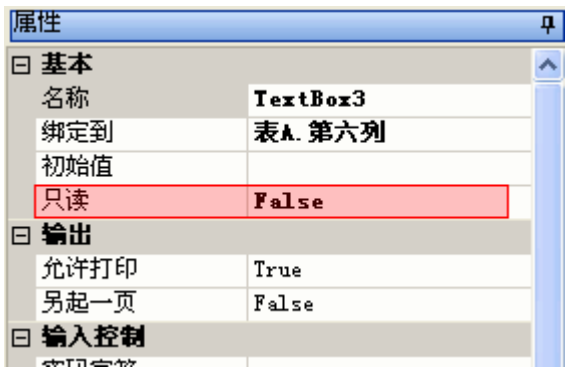
流程回退(1)	'回退到流程 1 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = 1 Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()

流程回退(2)	'回退到流程 2 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = 2 Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()
流程回退(3)	'回退到流程 3 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = 3 Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()
流程回退(Null)	'回退到初始状态 Dim dr As DataRow = Tables("表 A").Current.DataRow Dim cmd as New SQLCommand cmd.CommandText = "Update {表 A} Set 标记 = Null Where [_Identify] = " & dr("_Identify") cmd.ExecuteNonQuery dr.Load()

4、增加三个模式窗口，分别为流程 1、流程 2、流程 3，用于处理各个流程，每个流程窗口的设计都是类似的，用于输入该流程 应该处理的数据，例如流程 2 窗口的设计为：



由于表是锁定的，所以流程窗口中文本框的只读属性应该设置为 False，否则不能输入数据：



窗口的 BeforeClose 事件代码设置代码为：

```
QQClient.Send("@i" & Tables("表 A").Current("_Identify") & "i@") '通知服务器处理完成
```

这样关闭窗口前，会通知服务器移除此行的编辑登记，这样其他用户就可以处理这行了。

最后给流程窗口的按钮设置代码，以流程 2 窗口为例：

按钮名称	代码

确定	With Tables("表 A").Current
	  If .Isnull("第四列") OrElse .IsNull("第五列") OrElse .IsNull("第六列") Then
	  MessageBox.Show("流程二没有处理完!")
	  Return
	  End If
	End With
	Tables("表 A").Current("标记") = 2
	DataTables("表 A").Save()
	e.Form.Close()

取消	Tables("表 A").Current.Reject() Tables("表 A").Current("标记") = 1 DataTables("表 A").Save() e.Form.Close()

4.21 单元格绘图

4.21.1 在单元格中直接绘图

在介绍表事件 DrawCell 的时候，我们了解了通过更改文本和样式，让单元格的显示变得丰富多彩。其实这只是 DrawCell 事件的最基础用法，在此之外，该事件还提供了通过 GDI+绘图的功能。

DrawCell 事件的完整 e 参数属性：

Table: 准备绘制的表

Row: 准备绘制的行

Col: 准备绘制的列

Style: 指定自定义样式的名称，如果用默认的样式绘制单元格，无须设置 Style 属性。

Text: 字符型，获得或者设置要绘制的文本内容

X: 要绘制单元格的水平坐标位置，单位为像素

Y: 要绘制单元格的垂直坐标位置，单位为像素

Width: 要绘制单元格的宽度，单位为像素

Height: 要绘制单元格的高度，单位为像素

Graphics: 返回一个 Graphics 对象，通过这个对象可以在单元格中绘制图形。

MouseHover: 逻辑型，如果鼠标停在此单元格，则返回 True，否则返回 False。

IsPrint: 逻辑型，是否是因为打印表而触发了 DrawCell 事件。

此外该事件的 e 参数还提供了两个方法：

StartDraw：开始绘图

EndDraw: 结束绘图

所有的绘图代码，必须在这两个方法之间。

有关绘图的基本知识，可以参考 GDI+快速入门 。

示例

此示例可以参考 CaseStudy 目录下的文件“单元格绘图.Table”。

假定表中有两列，分别是项目和进度，希望能够直接在项目列的单元格中用条形图表示进度，DrawCell 事件的代码为：

```
If e.Col.Name = "进度" AndAlso e.Row.IsNull("进度") = False Then
    e.StartDraw()
    Dim Width As Integer = (e.Width - 2 )* e.Row("进度") \ 100
    If e.Row("进度") = 100 Then
        e.Graphics.FillRectangle(Brushes.Green,e.x + 1,e.y + 1, Width, e.Height - 2)
    Else
        e.Graphics.FillRectangle(Brushes.Red,e.x + 1,e.y + 1, Width, e.Height - 2)
    End If
    e.EndDraw()
End If
```

绘制出的效果为：

	项目	进度
1	项目一	40
2	项目二	70
3	项目三	100
4	项目四	20
5	项目五	10

提示：一些复杂的单元格绘图，可能需要修改行高，DrawCell 事件并没有设置行高的参数，必须借助 BeforeDrawRow 事件。

有了 DrawCell 事件，我们可以在单元格绘制任意复杂的图片，例如下图是一个建筑用的钢筋计算表：

	排序号	层次	构件名称	配筋	钢筋规格	件数	根数	计算简图	单根钢筋计算式
1	138	二层	KZ1	4C18	C18	2	4	4000	4000+1.2*40*18*1
2	139	二层	KZ1	8C16	C16	2	8	4000	4000+1.2*40*16*1
3	140	二层	KZ1	A8@100/200	A8	2	29		450*4+550*4+550/3*2+450/3*2
4	44	基础层	KZ1	4C18	C18	2	4	216 1910	1480-70+12*18+500+1.2*4
5	45	基础层	KZ1	8C16	C16	2	8	192 1910	1480-70+12*16+500+1.2*4
6	46	基础层	KZ1	A8@100/200	A8	2	11		450*4+550*4+550/3*2+450/3*2
7	181	三层	KZ1	4C18	C18	2	4	4000	4000+1.2*40*18*1
8	182	三层	KZ1	8C16	C16	2	8	4000	4000+1.2*40*16*1
9	183	三层	KZ1	A8@100/200	A8	2	29		450*4+550*4+550/3*2+450/3*2

4.21.2 在单元格绘制按钮

本节内容可以参考 CaseStudy 目录下的文件：绘制单元格按钮.Table
本节的任务是在表的左侧显示一列，该列的每个单元格显示一个删除按钮，双击此按钮即可删除对应的行。

	删	第二列	第三列	第四列	第五列
1	✕	1			
2	✕	2			
3	✕	3			
4	✕	4			
5	✕	5			
6	✕	6			
7	✕	7			
8	✕	8			

设计步骤：
1、将设计好的图标文件 delete1.ico 和 delete2.ico 复制到项目文件所在目录的 Images 子目录下。

delete1.ico	✕	默认图标
delete2.ico	✕	鼠标经过时的图标

2、在全局代码中加入以下代码：

```
Public delimg1 As Image = GetImage("delete1.ico")
Public delimg2 As Image = GetImage("delete2.ico")
```

上述代码获得两个图标的图片，并赋值给 两个全局变量 delimg1 和 delimg2。

3、在项目事件 AfterOpenProject 加入代码：

```
If DataTables("表A").DataCols.Contains("删") = False Then
    DataTables("表A").DataCols.Add("删",Gettype(String),1)
    Tables("表A").Cols("删").Width = 24
    Tables("表A").Cols("删").Move(0)
End If
```

上述代码在表 A 中增加一列，名称"删"，宽度设置为 24，并将其移至第一列的位置。
4、将表的 PrepareEdit 事件加入以下代码

```
If e.Col.Name = "删" Then
   e.Cancel = True
End If
```

5、将表的 DrawCell 事件加入以下代码：

```
If e.Col.name= "删" Then
    e.StartDraw
    If e.MouseHover = False '如果鼠标不在此单元格
        e.Graphics.DrawImage(delimg1, e.x + 3,e.y + 3) '绘制第一个图标
    Else '如果鼠标在此单元格
        e.Graphics.DrawImage(delimg2, e.x + 2,e.y + 2) '绘制第二个图标
    End If
    e.EndDraw
End If
```

6、在表的DoubleClick事件加入以下代码：

```
If e.Col.name = "删" Then
    e.Row.Delete()
End If
```

提示：这个例子绘制的按钮有两个状态，分别是正常和鼠标经过时，如果你要绘制有三个状态的按钮，也就是正常、鼠标经过时和鼠标按下时，可以用系统变量 MouseButton 来判断鼠标是否按下。

效率探讨

我们也可以删除全局代码，直接将 DrawCell 事件改为：

```
If e.Col.name= "删" Then
    e.StartDraw
    If e.MouseHover = False '如果鼠标不在此单元格
        e.Graphics.DrawImage(GetImage("delete1.ico"), e.x + 3,e.y + 3) '绘制第一个图标
    Else '如果鼠标在此单元格
        e.Graphics.DrawImage(GetImage("delete2.ico"), e.x + 2,e.y + 2) '绘制第二个图标
    End If
    e.EndDraw
End If
```

但是这样每次绘制单元格的时候，都要读取图标文件来生成图片，效率会低一些。

所以实际应用的时候，我们在全局代码中读取图标文件生成图片，并赋值给全局变量：

```
Public delimg1 As Image = GetImage("delete1.ico")
Public delimg2 As Image = GetImage("delete2.ico")
```

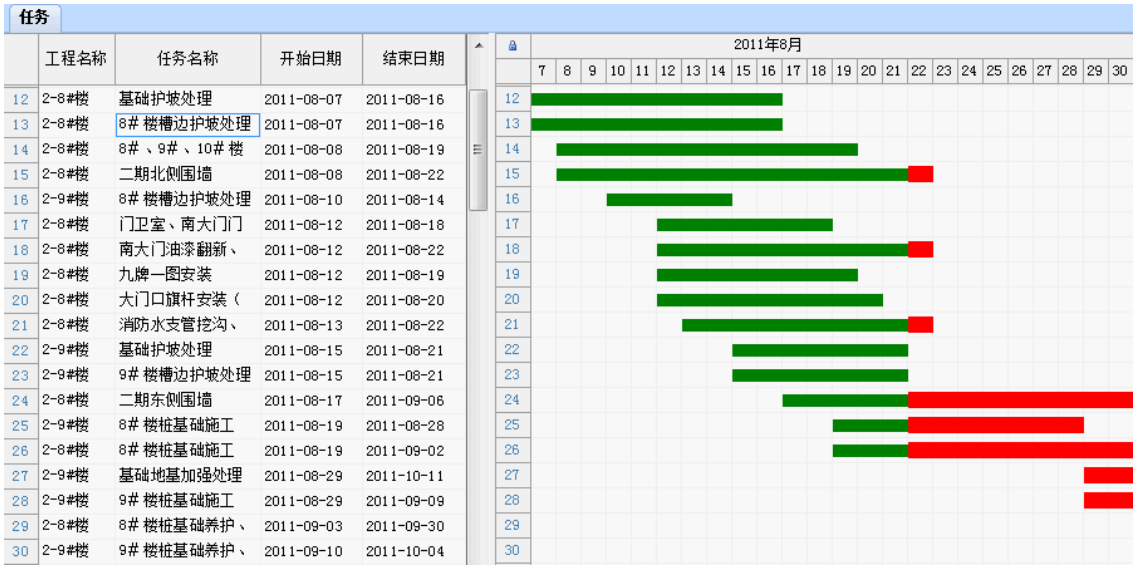
这样每次绘制单元格的时候，可以直接使用全局变量来绘制图片，无需反复读取图标文件，所以效率会高一些：

```
If e.Col.name= "删" Then
    e.StartDraw
    If e.MouseHover = False '如果鼠标不在此单元格
        e.Graphics.DrawImage(delimg1, e.x + 3,e.y + 3) '绘制第一个图标
    Else '如果鼠标在此单元格
        e.Graphics.DrawImage(delimg2, e.x + 2,e.y + 2) '绘制第二个图标
    End If
    e.EndDraw
```

End If

4.21.3 在单元格绘制甘特图

上一节已经讲到，DrawCell 事件可以直接使用 GDI+绘图。
在此基础上，本节内容讲述如何用 DrawCell 事件绘制甘特图。
由于是自己编码绘制，所以会比普通的甘特图控件更加灵活，可以附加很多其他的功能。
本节内容可以参考 CaseStudy 目录下的文件：编码绘制甘特图.Table
该文件实现了以下功能：
1、左表显示任务，右表显示甘特图。
2、甘特图表的结构，是根据任务表的开始日期和结束日期范围自动生成的。
3、上下滚动任务表，甘特图表会同步滚动，反之亦然。
4、选择某个任务，会同步选择此任务的甘特图，反之亦然。
5、在任务表进筛选后，能够根据筛选后的日期范围，重新生成甘特图表的结构。
6、甘特图以绿色表示过去的日期，以红色表示未来的日期。



相关知识要点：AfterScroll ScrollPosition 可见区域 GDI+快速入门

设计步骤

1、新建一个窗口，窗口类型设置为主窗口。
2、再插入一个 SplitContainer(分区面板)，停靠属性设置为 Fill。
3、在分区面板的左分区插入一个 Table 控件，绑定到任务表，停靠属性设置为 Fill，名称为 Table1。
4、在分区面板的右分区插入一个 Table 控件，类型设置为 SQLTable，停靠属性设置为 Fill，名称为 Table2。
5、在全局代码中定义一个 Public 变量：

```
Public DrawGannt As Boolean = True
```

DrawGannt 相当于一个开关，将其设置为 True，正常绘制甘特图，将其设置为 False，暂停绘制甘特图。
6、自定义内部函数，分别为:

函数名	说明	代码
BuildGanttTable	根据任务	`Dim tbl As Table = Tables("任务")`

	表生成甘特图表的结构。	Tables("窗口 1_Table2").StopRedraw()
		Dim StartDate As Date = tbl.Compute("Min(开始日期)")
		Dim EndDate As Date = tbl.Compute("Max(结束日期)")
		Dim dt As Date = StartDate
		Dim Builder As New DataTableBuilder("统计")
		Do
		Dim nm As String = dt.Year & "年
		" & dt.Month & "月_" & dt.Day
		Builder.Adddef(nm,Gettype(String),1)
		dt = dt.Adddays(1)
		If dt > Enddate Then
		Exit Do
		End If
		Loop
		Tables("窗口
		1_Table2").DataSource = Builder.BuildDataSource
		For Each cl As Col In Tables("窗口 1_Table2").Cols
		cl.width = 20
		Next
		Functions.Execute("AddGanttRows")
		Tables("窗口 1_Table2").ResumeRedraw()

AddGanttRows	根据任务表的行数在甘特图表添加对应的行数。	Dim tbl As DataTable = DataTables("窗口 1_Table2") tbl.StopRedraw tbl.DataRows.Clear() For i As Integer = 1 To Tables("任务").Rows.Count tbl.AddNew() Next tbl.Save() tbl.ResumeRedraw

SetGanntLeftCol	设置甘特图表左边第一个可见列，使得甘特图表的自动左右滚动，确保任务表第一个可见任务的甘特图可见。	If Forms("窗口 1").Opened()
		Dim t1 As Table = Tables("任务")
		Dim t2 As Table = Tables("窗口 1_Table2")
		If t1.TopVisibleRow >=0 Then
		Dim r As Row = t1.Rows(t1.TopVisibleRow)
		Dim dt As Date = r("开始日期")
		Dim nm As String = dt.Year & "年
		" & dt.Month &"月_" & dt.Day
		Dim idx As Integer= t2.cols(nm).Index
		t2.LeftVisibleCol = idx
		End If
		End If

7、重新回到窗口设计器，将窗口的 AfterLoad 事件代码设置为：

```
Functions.Execute("BuildGanttTable")
```

8、选择窗口中的 Table2 控件，设置有关事件代码：

事件	代码

AfterScroll	Dim tbl As Table = Tables("任务")
	If e.Table.TopVisibleRow <> tbl.TopVisibleRow Then
	  tbl.TopVisibleRow = e.Table.TopVisibleRow
	  e.Table.TopVisibleRow = e.Table.TopVisibleRow '此行不可少,确保第一个可见行完整显示
	End If

DrawCell	If DrawGannt Then
	  Dim r As Row = Tables("任务").Rows(e.Row.Index)
	  Dim dt1 As Date = r("开始日期")
	  Dim dt2 As Date = r("结束日期")
	  Dim dt As Date = e.Col.Name.Replace("年","-").Replace("月_","-")
	  If dt>=dt1 AndAlso dt<=dt2 Then
	  e.StartDraw()
	  If dt < Date.Today Then
	  e.Graphics.FillRectangle(Brushes.Green,e.x ,e.y + 5, e.Width, e.Height - 10)
	  Else
	  e.Graphics.FillRectangle(Brushes.Red,e.x ,e.y + 3, e.Width, e.Height - 7)
	  End If
	  e.EndDraw()
	  End If
	End If

PositionChanged	Dim tbl As Table = Tables("任务")
	If tbl.Position <> e.Table.Position Then
	  tbl.Position = e.Table.Position
	End If

9、回到主界面，设置任务表的相关事件代码：

事件	代码

AfterScroll	If Forms("窗口 1").Opened Then
	  Dim tbl As Table = Tables("窗口 1_Table2")
	  If e.Table.TopVisibleRow <> tbl.TopVisibleRow Then
	  tbl.TopVisibleRow = e.Table.TopVisibleRow
	  e.Table.TopVisibleRow = e.Table.TopVisibleRow '此行不可少,确保第一个可见行完整显示
	  End If
	  Functions.Execute("SetGanntLeftCol")
	End If

AfterSort	If Forms("窗口 1").Opened()

	Tables("窗口 1_Table2").Refresh() '刷新甘特图表 Functions.Execute("SetGanntLeftCol") End If
BeforeFilter	DrawGannt = False '暂停绘制甘特图

AfterFilter	If Forms("窗口 1").Opened Then
	Functions.Execute("BuildGanttTable")
	Functions.Execute("AddGanttRows")
	Functions.Execute("SetGanntLeftCol")
	End If
	DrawGannt = True '回复绘制甘特图
	If Forms("窗口 1").Opened Then
	Tables("窗口 1_Table2").Refresh() '刷新甘特图表
	End If

PositionChanged	If Forms("窗口 1").Opened() Then
	Dim tbl As Table = Tables("窗口 1_Table2")
	If tbl.Position <> e.Table.Position Then
	tbl.Position = e.Table.Position
	End If
	End If

4.22 网络相关

4.22.1 MailSender

MailSender 用于通过 SMTP 服务器发送邮件。

属性：

- Host
  用于指定主机地址，可以是域名，也可以是 IP 地址。
- Port
  用于指定 SMTP 服务的端口，默认为 25，此属性通常无需设定。
- Account
  用于指定登录邮件服务器的帐号
- Password
  用于指定登录邮件服务器的密码

- From
  用于指定发件人的邮件地址

- To
  用于指定收件人的邮件地址

- Title
  用于指定邮件标题

- Content
  用于指定邮件的正文内容

- EnableSsl
  逻辑型，是否使用安全套接字层（SSL）加密连接。

- Encoding
  用于指定邮件编码，Encoding 类型，可选的编码有：

  Default
  ASCII
  BigEndianUnicode
  Unicode
  UTF32
  UTF7
  UTF8

  默认为 Default，表示系采用统编码，对于中文操作系统，通常是 GB2312

方法：

- Send
  发送邮件，直到邮件发送结束，才能进行其他操作。

- SendAsync
  以异步方式发送邮件，发送邮件的同时，可以进行其他操作。

- AddReceiver
  如果有多个收件人，可以用此方法添加任意多个收件人的邮件地址。

- AddAttachments
  用于添加附件。

示例一

```
Dim m As New MailSender
m.Host = "smtp.21cn.net"
m.Account = "xiaoliu"
```

```
m.Password = "xiaoliu1234567"
m.From = "xiaoliu@Sina.com.cn"
m.To = "meili@Sina.com.cn"
m.Title = "李总明天来北京"
m.Content = "航班号是 gz1234,上午 11 点到达,请注意接机"
m.SendAsync()
```

示例一

如果接收方出现邮件内容乱码，可以明确指定邮件内容编码，例如用 UTF8 编码邮件内容：

```
Dim m As New MailSender
m.Host = "smtp.21cn.net"
m.Account = "xiaoliu"
m.Password = "xiaoliu1234567"
m.From = "xiaoliu@Sina.com.cn"
m.To = "meili@Sina.com.cn"
m.Encoding = Encoding.UTF8
m.Title = "李总明天来北京"
m.Content = "航班号是 gz1234,上午 11 点到达,请注意接机"
m.SendAsync()
```

示例三

可以同时添加多个收件人：

```
Dim m As New MailSender
m.Host = "smtp.21cn.net"
m.Account = "xiaoliu"
m.Password = "xiaoliu1234567"
m.From = "xiaoliu@Sina.com.cn"
m.AddReceiver("meili@Sina.com.cn")
m.AddReceiver("yangzi@Microsoft.com")
m.Title = "李总明天来北京"
m.Content = "航班号是 gz1234,上午 11 点到达,请注意接机"
m.SendAsync()
```

示例四

还可以加入附件：

```
Dim m As New MailSender
m.Host = "smtp.21cn.net"
m.Account = "xiaoliu"
m.Password = "xiaoliu1234567"
m.From = "xiaoliu@Sina.com.cn"
m.To = "meili@Sina.com.cn"
m.Title = "初稿已经完成"
m.Content = "附件为初稿，请您审阅!"
m.AddAttachments("c:\data\table.txt")
m.SendAsync()
```

示例五

有了 MailSender，我们可以根据数据表内容动态合成邮件发送，例如下面的代码从借款表中筛选出 10 天内到期的客户，并发送催款邮件：

```
Dim m As New MailSender
Dim ln As String = vbcrlf & vbcrlf '定义两个换行
m.Host = "smtp.21cn.net"
m.Account = "xiaoliu"
m.Password = "xiaoliu1234567"
m.From = "xiaoliu@Sina.com.cn"
For Each dr As DataRow In DataTables("借款").Select("到期日期<= #" & Date.Today.AddDays(10) & "#")
    m.To = dr("邮件地址")
    m.Title = "催款通知"
    m.Content = dr("姓名") & dr("尊称") & ":" & ln & "您的贷款将于" & dr("到期日期") & "到期, "
    m.Content = m.Content & "金额为" & dr("金额") & "元, 请准时还款!" & ln & "某某公司" & vbcrlf & Date.today
    m.SendAsync()
Next
```

4.22.2 使用 QQ 邮箱发送邮件

如果你使用的是 QQ 邮箱，需要按以下步骤设置才能使用 MainSender 发送邮件：

1、QQ 邮箱的其 POP3 和 SMTP 服务默认是关闭的，你需要开启：

http://service.mail.qq.com/cgi-bin/help?subtype=1&id=28&no=166

2、QQ 邮箱的密码和 QQ 密码不同，需要获取授权码作为邮箱密码：

http://service.mail.qq.com/cgi-bin/help?subtype=1&&id=28&&no=1001256

3、MailSender 的 EnableSsl 属性设置为 True。

示例代码

```
Dim m As New MailSender
m.Host = "smtp.qq.com"
m.EnableSsl = True '必须将EnableSsl属性设置为True
m.Account = "apple@qq.com"
m.Password = "c409ddc47aa74638" '注意这不是QQ密码,是授权码
m.From = "apple@qq.com"
m.To = "1234567@qq.com"
m.Title = "李总明天来北京"
m.Content = "航班号是gz1234,上午11点到达,请注意接机"
m.Send()
```

4.22.3 FTPClient

FTPClient 类提供了一套完整的方法，用于管理 FTP 服务器上的文件和目录。

属性

- Host
  返回或设置 FTP 服务器地址，可以是域名或 IP 地址。

- Account
  返回或设置登录用户名

- Password
  返回或设置登录密码。

- TimeOut
  整数型，用于设置尝试操作的毫秒数，默认为 10000，也就是 10 秒。

- RootDir
  字符型，设置根目录。

- UTF8
  逻辑型，是否使用 UTF8 编码。

- CurrentDir
  字符型，返回 FTP 的当前目录。

- Connected
  逻辑型，判断 FTP 是否已经连接成功。

- Port
  整数型，默认为 21，用于指定 FTP 端口号，一般无需设置，除非你改变了默认的端口号。

- UploadEnabled
  逻辑型，用于决定 Ftp 管理器中的上传按钮是否可用。

- DownloadEnabled
  逻辑型，用于决定 Ftp 管理器中的下载按钮是否可用。

- RenameEnabled
  逻辑型，用于决定 Ftp 管理器中的重命名按钮是否可用。

- DeleteEnabled
  逻辑型，用于决定 Ftp 管理器中的删除按钮是否可用。

- MakerDirEnabled
  逻辑型，用于决定 Ftp 管理器中的新建文件夹按钮是否可用。

- OpenFileEnabled
  逻辑型，用于决定 Ftp 管理器中的打开文件按钮是否可用。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
ftp1.DeleteEnabled =False '禁用删除文件按钮
ftp1.RenameEnabled = False '禁用重命名文件按钮
ftp1.RootDir = "\photo" '设置根目录
ftp1.OpenManager()
```

方法

OpenManager

打开 FTP 管理器。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
ftp1.Changedir("\photo")
ftp1.OpenManager()
```

Download

从 FTP 上下载文件，语法：

Download(sourceFileName, destinationFileName, ShowUI)

sourceFileName：要下载的文件，含路径

destinationFileName：保存下载文件的文件名和路径。

ShowUI：可选参数，逻辑型，是否显示下载进度，默认为 False。

如果下载成功，返回 True，否则返回 False:

示例

```
Dim ftp1 As new ftpclient
ftp1.host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.password = "138238110"
If ftp1.Download("\photo\Desert.jpg","c:\data\Desert.jpg") = True Then
    Messagebox.show("下载完成!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
Else
    Messagebox.show("下载失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

Upload

将文件上传到 FTP，语法：

Upload(sourceFileName, destinationFileName, ShowUI)

sourceFileName：要上传的本地文件，含路径

destinationFileName：目标文件，含路径

ShowUI：逻辑型，是否显示上传进度，默认为 False。

如果上传成功，返回 True，否则返回 False

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.Upload("c:\data\Desert.jpg","\update\Desert.jpg") = True Then
    Messagebox.show("上传完成!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
Else
    Messagebox.show("上传失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

DeleteFile

删除 FTP 上的文件，语法：

DeleteFle(FileName)

FileName：要删除的文件，含路径

如果删除成功，返回 True，否则返回 False

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.DeleteFile("\update\Desert.jpg") = True Then
    Messagebox.show("删除完成!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
Else
```

```
    Messagebox.show("删除失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

DeleteDir

删除 FTP 上的目录，语法：

DeleteDir(DirName,AnyWay)

DirName：要删除的目录名称。

AnyWay：可选参数，如果设为 True，将删除目录下的文件及其子目录，如果省略此参数，则只有此目录为空的时候，才能删除。

如果删除成功，返回 True，否则返回 False

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.DeleteDir("\data", True) = True Then
    Messagebox.show("删除完成!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
Else
    Messagebox.show("删除失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

GetFileList

用于返回 FTP 服务器指定目录下的全部文件名，以字符串集合的形式返回，语法：

GetFileList(Path)

Path：可选参数，用于指定目录，如果省略，则返回当前目录下的全部文件名。

示例

例如列出 photo 目录下的所有文件：

```
Dim ftp1 As New FtpClient
Dim fls As List(of String)
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
fls = ftp1.GetFileList("\photo")
For Each fl As String In fls
    Output.Show(fl)
Next
```

还可以使用通配符，例如列出photo目录下的所有jpg文件：

```
Dim ftp1 As New FtpClient
Dim fls As List(of String)
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
```

```
ftp1.Password = "138238110"
fls = ftp1.GetFileList("\photo\*.jpg")
For Each fl As String In fls
    Output.Show(fl)
Next
```

GetDirList

用于返回 FTP 服务器指定目录下的全部子目录，以字符串集合的形式返回，语法：

GetDirList(Path)

Path：可选参数，用于指定目录，如果省略，则返回当前目录下的全部子目录名。

示例

```
Dim ftp1 As New FtpClient
Dim dls As List(of String)
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
dls = ftp1.GetDirList("\photo")
For Each dl As String In dls
    Output.Show(dl)
Next
```

GetDetailList

用于返回 FTP 服务器指定目录下的全部文件和子目录，包括文件的名称、修改时间和大小，各项之间用符号*分隔。

GetDetaiList(Path)

Path：可选参数，用于指定目录，如果省略，则返回当前目录下的全部子目录名。

```
Dim ftp1 As New FtpClient
Dim dls As List(of String)
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
dls = ftp1.GetDetailList("\")
For Each dl As String In dls
    Output.Show(dl)
Next
```

GetFileSize

返回 ftp 上指定文件的大小，以字节为单位。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
```

```
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
Dim sz As Integer
sz = ftp1.GetFileSize("\photo\jz.png")
Output.Show("文件大小:" & sz)
```

GetFileTime

返回 ftp 上指定文件最后一次修改的日期时间。

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
Dim sz As Date
sz = ftp1.GetFileTime("\photo\jz.png")
Output.Show("修改时间:" & sz)
```

FileExists

判断指定的文件是否存在，语法：

FileExists(FileName)

FileName：要判断的文件名，含路径。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.FileExists("\photo\jz.png") Then '如果存在则下载
    If ftp1.Download("\photo\jz.png","c:\data\jz.png") Then
        messagebox.Show("下载完成!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
        messagebox.Show("下载失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
Else
    messagebox.Show("此文件不存在!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

DirExists

判断指定的目录是否存在，语法：

DirExists(Path)

Path：要判断的目录。

示例

```
Dim ftp1 As New FtpClient
```

```
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.DirExists("\photo") Then
    MessageBox.show("目录已经存在","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

Rename

用于重命名文件或目录，语法：

Rename(currentName, newName)

currentName：要重命名的文件或目录，含路径。

newName：新的名称，含路径。

如果重命名成功，则返回 True，否则返回 False.

示例

重命名文件：

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.Rename("\Photo\Desert.jpg","\Photo\Desert1.jpg") Then
    Messagebox.show("重命名文件成功!")
End If
```

重命名目录：

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.Rename("\Key\","\NewKey\") Then
    Messagebox.show("重命名目录成功!")
End If
```

写这段帮助的时候，我意外发现Rename还可以移动文件。

例如将文件"jz.png"从photo目录移到到newkey目录：

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.Rename("\photo\jz.png","\newkey\jz.png") Then
    Messagebox.show("移动文件成功!")
End If
```

MakeDir

在 ftp 上创建目录，语法：

MakeDir(Path)

Path：要创建的目录。

如果创建目录成功，则返回 True，否则返回 False.

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.MakeDir("\photo\hdx") Then
    Messagebox.Show("创建目录成功!")
Else
    Messagebox.Show("创建目录失败!")
End If
```

DeleteDir

删除 ftp 上的指定目录，语法：

DeleteDir(Path)

Path：要删除的目录

如果删除成功，则返回 True，否则返回 False。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
If ftp1.DeleteDir("\photo\hdx") Then
    Messagebox.Show("删除目录成功!")
Else
    Messagebox.Show("删除目录失败!")
End If
```

ChangeDir

将指定目录设置为当前目录，语法：

ChangeDir(path)

path：要设置为当前目录的目录。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
Messagebox.Show("当前目录: " & ftp1.Currentdir)
```

```
ftp1.Changedir("\photo")
Messagebox.Show("当前目录: " & ftp1.Currentdir)
```

SelectFile

打开文件对话框，从 ftp 上选择文件，返回一个字符串集合，返回选择的一个或多个文件。

示例

```
Dim ftp1 As New FtpClient
Dim files As New List(of String)
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
ftp1.ChangeDir("\photo")
files = ftp1.SelectFile()
For each file As String In files
    Output.Show(file)
Next
```

SelectFolder

打开目录选择对话框，从ftp上选择一个目录。

示例

```
Dim ftp1 As New FtpClient
Dim folder As String
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
ftp1.Password = "138238110"
folder = ftp1.SelectFolder()
If folder > "" Then
    Messagebox.show("你选择的目录是:" & folder)
End If
```

Connect

连接FTP，如果连接成功，返回True，否则返回False。

FTP默认是自动判断并连接的，正如前面的示例代码，并不需要先连接FTP再进行操作。

如果愿意，也可以Connected属性判断FTP是否已经连通，用Connect方法显式连接FTP。

示例

```
Dim ftp1 As New FtpClient
ftp1.Host="196.128.143.28"
ftp1.Account = "foxuser"
```

```
ftp1.Password = "138238110"
ftp1.Changedir("\photo")
If ftp1.Connected = False '如果FTP没有连接
    If ftp1.Connect Then '连接FTP
        ftp1.OpenManager() '如果连接成功,打开FTP客户端
    Else
        Messagebox.show("连接FTP失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

Close

关闭FTP连接

4.22.4 Network

4.22.4.1 Network

Network 类提供了几个方法，用于 Ping 远程服务器以及上载和下载文件。

Ping

DownloadFile

UploadFile

4.22.4.2 方法

4.22.4.2.1 Ping

对指定服务器执行 Ping 操作。

语法：

Ping(address)

Ping(address ,timeout)

address：要 ping 的域名或者 IP 地址，域名前面不能有"http://"

timeout：可选参数，在 Ping 通之前持续尝试的时间，单位为毫秒，默认值为 500。

如果 Ping 通，则返回 True，否则返回 False

示例:

```
If Network.Ping("www.egridsoft.com") Then
    Messagebox.Show("可以访问易表主页")
End If
If Network.Ping("198.01.01.01") Then
    Messagebox.Show("该服务器可用")
End If
```

需要注意的是，如果服务器设置成禁止 Ping 入，那么就不能用 Ping 来判断服务器是否可用了。

4.22.4.2.2 DownloadFile

下载指定的远程文件，并将其保存在指定的位置。
语法：
Network.DownloadFile(address ,destinationFileName)
Network.DownloadFile(address ,destinationFileName ,userName ,password)
Network.DownloadFile(address ,destinationFileName ,userName ,password ,showUI ,connectionTimeout ,overwrite)
参数说明：

- address
  要下载的文件的路径，其中包括文件名和主机地址

- destinationFileName
  保存下载文件的文件名和路径

- userName
  要进行身份验证的用户名

- password
  要进行身份验证的密码

- showUI
  逻辑型，是否显示下载进度

- connectionTimeout
  整数型，设置超时间隔(以毫秒为单位)，默认值为 100000(100 秒)

- overwrite
  逻辑型，是否覆盖现有文件

示例
Network.DownloadFile("http://www.foxtable.com/download/release.rar", "c:\data\Release.rar")
下面的代码会显示下载进度：
Network.DownloadFile("http://www.foxtable.com/download/release.rar",
"c:\data\Release.rar" ,"" ,"" ,True, 100000, True)
对于带有密码的 ftp，可以直接在 address 参数中加入户名和密码信息，格式为：
ftp://户名:密码@地址/
例如：

Network.DownloadFile("ftp://zhanshang:138238@egridsoft.com/public/isam.zip","c:\data\isam.zip")

上面假定帐户为zhanshang，密码为138238，ftp服务器的址为ftp://egridsoft.com。
从该服务器的地public目录下载文件isam.zip，保存到本地的c:\data目录下

上面的代码等效于：

Network.DownloadFile("ftp://egridsoft.com/public/isam.zip","c:\data\isam.zip","zhanshang","138238")

4.22.4.2.3 UploadFile

将指定文件上传到指定的远程服务器上。
语法：
UploadFile(sourceFileName ,address)
UploadFile(sourceFileName ,address ,userName ,password)
UploadFile(sourceFileName ,address ,userName ,password ,showUI ,connectionTimeout)
参数说明：

- sourceFileName
  要上传的文件，含路径。

- address
  目标文件，含主机地址、路径和文件名

- userName
  要进行身份验证的用户名

- password
  要进行身份验证的密码

- showUI
  逻辑型，是否显示上传进度

- connectionTimeout
  整数型，设置超时间隔(以毫秒为单位)，默认值为 100000(100 秒)

对于带有密码的 ftp，可以直接在参数 address 中加入户名和密码信息，格式为：
ftp://户名:密码@地址/
示例：
Network.UploadFile("c:\data\table.txt", "ftp://zhanshang:138238@egridsoft.com/public/table.txt")

上面假定帐户为zhanshang，密码为138238，ftp服务器的地址为ftp://egridsoft.com。
将本地c:\data目录下的文件table.txt，上传到服务器的public目录下。

4.22.5 TryConnectHost

用于判断服务器的地址是否可以解析。
前面提到用 Network 的 Ping 方法判断指定的地址服务器是否 可以连通，但是有缺陷，在网络不通的情况下，会出现错误提示。
为了提高判断速度，并避免出现错误提示，我们可以 用 TryConnectHost 先判断服务器地址是否可以解析，然后再用 Ping 测试服务器是否可以连接。
示例一

```
If TryConnectHost("www.egridsoft.com") Then
    If Network.Ping("www.egridsoft.com") Then
        Messagebox.Show("可以访问易表主页")
    End If
End If
```

示例二

对于一般的管理系统，网络问题出在服务器端的比较少，一般问题都出在客户端。
在服务器端稳定的情况下，为了提高判断速度，可以省去用ping判断的这一步，例如：

```
If TryConnectHost("www.egridsoft.com") Then
    Messagebox.Show("可以访问易表主页")
End If
```

4.22.6 网络状态监测事件

4.22.7 串口通讯

4.22.7.1 串口基础

如果你没有使用串口进行通讯的必要，可以忽略有关串口的所有内容。
Foxtable 提供了一个 Ports 集合，用于管理串口。
Ports 的有一个 PortNames 属性，该属性返回一个字符串集合，包括当前计算机所有可用串口的名称。
例如需要列出所有的串口名称：

```
For Each sp As String In Ports.PortNames
    Output.Show(sp)
Next
```

如果要使用某个串口，必须首先将其加入 Ports 集合中，例如：

```
Ports.Add("COM2")
```

Ports 集合中的串口，必须先打开才能使用：

```
Ports("COM2").Open()
```

例如某个项目中需要使用 COM2 口，可以在该项目的 AfterOpenProject 事件中加入代码：

```
Ports.Add("COM2")
Ports("COM2").Open()
```

串口一旦打开，就可以发送和接收数据。

例如向 COM2 写入字符"abcd"：

```
Ports("COM2").Write("abcd")
```

从 COM2 中接收数据：

```
Dim s As String = Ports("COM2").ReadExisting
```

如果串口不再使用，则应该关闭，例如：

```
Ports("COM2").Close()
```

4.22.7.2 监视串口

如果需要实时接收串口数据，可以管理项目功能区，单击下面的按钮：



然后设置 DataReceived 事件代码。

DataReceived 事件在串口接收到数据的时候触发。

e 参数属性：

Port：接收到数据的串口。

注意必须先打开串口，才能接收数据，关于如何打开串口，请参考上一节“串口基础”。

示例

例如窗口中有个标签(Label)，希望用这个标签即时显示从串口中接收到的数据，只需将 DataReceived 事件代码设置为：

```
Forms("窗口 1").Controls("Label1").Text = e.Port.ReadExisting()
```

如果有多个串口接收数据，可以根据串口的名称来判断：

```
Select Case e.Port.PortName
    Case "COM1"
        Forms("窗口 1").Controls("Label1").Text = e.Port.ReadExisting()
    Case "COM2"
        Forms("窗口 1").Controls("Label2").Text = e.Port.ReadExisting()
End Select
```

4.22.7.3 类型参考

4.22.7.3.1 Ports

名称	类型	说明
PortNames	属性	返回一个字符串集合，包括当前计算机所有可用串口的名称。 例如需要列出所有的串口名称： For Each sp As String In Ports.PortNames Output.Show(sp) Next
Add	方法	将指定名称的串口加入到 Ports 集合，只有加入到 Ports 集合的串口，才可以在 Foxtable 中使用。 例如： Ports.Add("COM2")
Remove	方法	将指定名称的串口从 Ports 集合中删除。 例如： Ports.Remove("COM2")

Contains	方法	判断 Ports 集合是否包括指定名称的串口。
		例如：
		If Ports.Contains("COM2") = False Then
		Ports.Add("COM2")
		End If

Clear	方法	清除 Ports 集合中的所有串口。

4.22.7.3.2 Port

4.22.7.3.2.1 属性

名称	说明
PortName	字符型，返回端口的名称。
BaudRate	整数型，获取或设置串行波特率。
DataBits	整数型，获取或设置每个字节的标准数据位长度。
StopBits	获取或设置每个字节的标准停止位数，StopBits 型枚举，可选值有： None: 必使用停止位。 One: 使用一个停止位。 OnePointFive: 使用 1.5 个停止位。

	Two: 使用两个停止位。
Parity	获取或设置奇偶校验检查协议，Parity 型枚举，可选值有： None: 不发生奇偶校验检查。 Even: 设置奇偶校验位，使位数等于偶数。 Mark: 将奇偶校验位保留为 1。 Odd: 设置奇偶校验位，使位数等于奇数。 Space: 将奇偶校验位保留为 0。
Handshake	获取或设置串行端口数据传输的握手协议，Handshake 型枚举，可选值有： • None 不使用握手协议。 • RequestToSend 使用请求发送 (RTS) 硬件流控制。RTS 发出信号，指出数据可用于传输。如果输入缓冲区已满，RTS 行将被设置为 false。当输入缓冲区中有更多可用空间时，RTS 行将被设置为 true。 • RequestToSendXOnXOff 同时使用请求发送 (RTS) 硬件控制和 XON/XOFF 软件控制。 • XOnXOff 使用 XON/XOFF 软件控制协议。发送 XOFF 控制以停止数据传输。发送 XON 控制以继续传输。使用这些软件控制，而不是使用请求发送 (RTS) 和清除发送 (CTS) 硬件控制。
Encoding	指定传输文本的编码方式，Encoding 类型，可选的编码有： Default ASCII BigEndianUnicode Unicode UTF32 UTF8
ReceivedBytesThreshold	整数型，获取或设置触发 DataReceived 事件前内部输入缓冲区中至少应该有的字节数。
DtrEnable	逻辑型，是否在串行通信过程中启用数据终端就绪 (DTR) 信号。

RtsEnable	逻辑型，是否在串行通信中是否启用请求发送 (RTS) 信号
BytesToRead	整数型，获取接收缓冲区中数据的字节数。
BytesToWrite	整数型，获取发送缓冲区中数据的字节数。
ReadTimeout	整数型，获取或设置读取操作未完成时发生超时之前的毫秒数。
WriteTimeout	整数型，获取或设置写入操作未完成时发生超时之前的毫秒数。
ReadBufferSize	整数型，获取或设置输入缓冲区的大小
WriteBufferSize	整数型，获取或设置串行端口输出缓冲区的大小。
IsOpen	逻辑型，判断端口是否已经打开
BreakState	逻辑型，获取或设置中断信号状态。
CDHolding	逻辑型，获取端口的载波检测行状态。
DsrHolding	逻辑型，获取数据设置就绪 (DSR) 信号的状态。
CtsHolding	逻辑型，获取“可以发送”行的状态。
DiscardNull	逻辑型，是否在端口和接收缓冲区之间传输时是忽略 Null 字节。
NewLine	字符型，获取或设置用于解释 ReadLine 和 WriteLine 方法调用结束的值，也就是换行符，默认为换行符 vbCrLf
ParityReplace	Byte 型，获取或设置一个字节，该字节在发生奇偶校验错误时替换数据流中的无效字节。

提示：

1、如果串口已经打开，则不能设置其属性，必须先关闭。

2、串口属性一般不需要设置，默认的属性设置符合大多数场合的需要。

示例

```
Ports.Add("COM2")
With Ports("COM2")
    .Close()
    .Parity = Parity.None
    .Handshake = Handshake.None
    .StopBits = StopBits.One
    .BaudRate = 9600
    .DataBits = 8
    .ReadBufferSize() = 4096
    .WriteBufferSize() = 2048
    .Open
End With
```

4.22.7.3.2.2 方法

名称	说明
Open	打开串口。 串口必须加入到 Ports 集合并打开，才能使用。 示例 Ports.Add("COM2") Ports("COM2").Open()
Close	关闭串口
ReadExisting	读取输入缓冲区中的所有字符。 示例 Dim s As String s = Ports("COM2").ReadExisting()
ReadLine	读取输入缓冲区的字符，直到遇到换行符， 返回的内容不会包括换行符。 如果读取过程没有遇到换行符，将引发超时错误。 示例 Dim s As String s = Ports("COM2").ReadLine()
ReadTo	读取输入缓冲区的字符，直到遇到指定的结束符，返回的内容不会包括此结束符。 如果读取过程没有遇到指定的结束符，将引发超时错误。 语法： ReadTo(Value) Value：指定的结束符 示例: Dim s As String s = Ports("COM2").ReadTo("|")
Read	从输入缓冲区读取一些字节，并将这些字节写入字节数组中指定的偏移量处。 语法： Read(buffer, offset, count) buffer: 字节数组，从输入缓冲区中读取的字节将写入此数组。 offset: 开始写入的偏移量。 count: 要读取的字节数。 示例 Dim cnt As Integer = Ports("COM2").BytesToRead If cnt > 0 Then '缓冲区是否有数据 Dim Val(cnt - 1) As Byte Ports("COM2").Read(val,0,cnt)

	End If
ReadByte	读取输入缓冲区的第一个字节。 Dim v As Byte = Ports("COM2").ReadByte
Write	语法一 Write(Value) 将字符串写入输出缓冲区。 Value: 要写入的字符串 示例 Dim s As String = "foxtable" Ports("COM2").Write(s) 语法二 Write(buffer, offset, count) 将指定数量的字节写入输出缓冲区中的指定偏移量处。 buffer: 将写入到输出缓冲区的字节数组。 offset: 开始写入的偏移量。 count: 要写入的字节数 示例 Dim Val() As Byte = New Byte() {49,50,51,52} Ports("COM2").Write(Val, 0, Val.Length)
WriteLine	将指定的字符串和换行符写入输出缓冲区。 语法： WriteLine(Value) Value: 要写入的字符串 示例 Dim s As String = "foxtable" Ports("COM2").WriteLine(s)
GetBytes	将字符串转换为字节数组，编码格式为 Unicode。 语法 GetBytes(Value) Value：要转换为字节的字符串。
GetString	将字节数组转换为字符串，编码格式为 Unicode。 语法: GetString(bytes) bytes: 要转换为字符串的数组。

4.22.7.4 实例

4.22.7.4.1 字符串收发

启用串口

首先在项目事件 AfterOpenProject 中打开需要使用的串口：

```
Ports.Add("COM2")
Ports("COM2").Open()
```

发送字符串

```
Ports("COM2").Write("foxtable")
```

接收字符串

将 DataReceived 事件代码设置为：

```
Dim s As String
s = e.Port.ReadExisting()
```

你也可以在任何地方使用下面的代码接收串口数据：

```
Dim s As String
s = Ports("COM2").ReadExisting()
```

4.22.7.4.2 字符传输和编码

在默认情况下，串口使用 ASCII 编码发送字符，所以不支持中文字符的收发。

如果要收发中文字符，必须使用 Encoding 属性指定串口的编码方式，可选的编码有：

Default

ASCII

BigEndianUnicode

Unicode

UTF32

UTF8

后四种编码都支持中文字符，不过通常使用的是 Unicode 或者 UTF8。

示例

为了让串口支持中文字符的收发，决定采用 Unicode 编码方式，可以在项目事件 AfterOpenProject 中加入代码：

```
Ports.Add("COM2")
Ports("COM2").Encoding = Encoding.Unicode
Ports("COM2").Open()
```

提示：

1、接收端和发送端必须使用相同的编码。

2、外接串口设备，一般使用的都是 ASCII 码；但是如果收发的字符串出现乱码，那么应该向设备提供商了解其所采用的编码，然后在代码中做出正确的设置。

4.22.7.4.3 字节收发

启用串口

首先在项目事件 AfterOpenProject 中打开需要使用的串口：

```
Ports.Add("COM2")
Ports("COM2").Open()
```

发送字节

```
Dim Val() As Byte = New Byte() {49,50,51,52}
```

```
Ports("COM2").Write(Val, 0, Val.Length)
```

接收字节

将 DataReceived 事件代码设置为：

```
Dim cnt As Integer = e.Port.BytesToRead
If cnt > 0 Then '缓冲区是否有数据
    Dim Val(cnt - 1) As Byte
    e.Port.Read(val,0,cnt)
End If
```

你也可以在任何地方使用下面的代码接收串口数据：

```
Dim cnt As Integer = Ports("COM2").BytesToRead
If cnt > 0 Then '缓冲区是否有数据
    Dim Val(cnt - 1) As Byte
    Ports("COM2").Read(val,0,cnt)
End If
```

4.23 软件加密

4.23.1 关于配套的 UKey

Foxtable 内置 UKey(俗称加密狗)功能，UKey 主要用于以下两个方面：
1、可以对自己开发的程序进行加密，防止盗版，保护自己的权益。
2、可以在权限管理和流程管理中，使用 UKey 进行身份验证。
用户需要从 Foxtable 官方购买配套的 UKey，才能使用内置的 UKey 功能。
当然你可以采购第三方的 UKey 进行开发，但是使用官方配套提供的 UKey 有以下好处：
1、因为相关功能已经内置，不涉及专业知识，所以任何用户都可以快速掌握 UKey 的用法。
2、软件和硬件均由 Foxtable 官方提供，兼容性更好，而且可以获得官方更好的技术支持。
3、对于非专业人士而言，采用官方提供的 UKey 更加安全，因为官方为每个客户提供的 UKey 都是独一无二的，且难以复制的，任何人都无法采购到和你相同的 UKey，增加了破解的难度。

4.23.2 UKey 参考

Foxtable 有一个名为 UKey 的系统变量，表示连接到计算机的 UKey。
UKey 的成员有：

- Start
  启动 UKey

- Encrypt1
  固化加密函数

- Encrypt2
  用户算法函数

- WriteStr
  向 UKey 的存储器写入字符

- ReadStr
  从 UKey 的存储器读取字符

- UID
  返回 UKey 的唯一编号

接下来我们开始介绍如何使用 UKey。

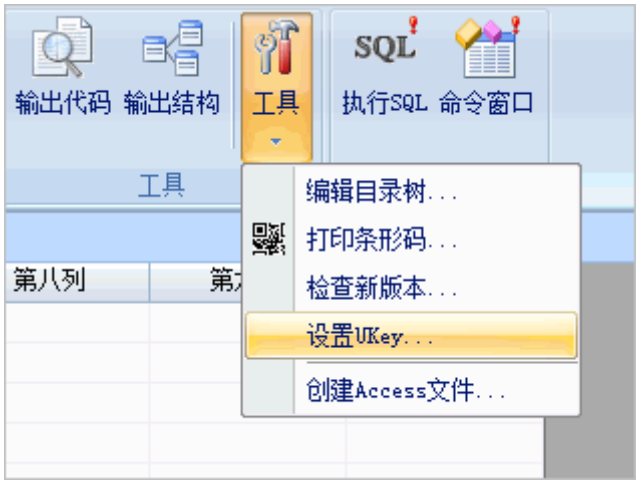
4.23.3 启动 UKey

在使用 UKey 之前，必须执行 Start 方法来启动 UKey。
Start 方法返回一个逻辑值，如果成功启动 UKey，则返回 True，否则返回 False.
例如：

```
If UKey.Start() Then
    MessageBox.Show("已经启动UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
Else
    MessageBox.Show("启动UKey失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

4.23.4 固化加密函数

UKey 有一个固化加密函数 Encrypt1，不同用户的 UKey，此函数的加密算法是不同的，也就是不同用户用 Encrypt1 加密同样的字符，得到的结果是不同的。
当你第一次向官方采购 UKey 的时候，官方会分配一个订货编号给你，此订货编号决定了 Encrypt1 的加密算法。
之后只需向官方提供这个订货编号，官方即可再次提供相同算法的 UKey 供您采购。
除非你将订货编号泄露给第三方，否则任何人采购的 UKey，其 Encrypt1 的加密算法都不会和你相同。
Encrypt1 函数使用步骤:
1、插上你从 Foxtable 官方采购的 UKey。
2、在菜单中执行“设置 UKey”命令：



3、在设置窗口中选择“算法测试”页，输入要加密的字符，然后单击“固化算法”按钮，得到加密结果：



4、现在就可以在代码中使用上图所示的加密结果，例如我们可以在 BeforeOpenProject 事件中 设置下面的代码：

```
If UKey.Start() Then
    If UKey.Encrypt1("abc") <> "2F7EFE3A0632AE5C" Then
        MessageBox.Show("请插上XX公司提供的UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
        e.Cancel = True
    End If
Else
    MessageBox.Show("启动UKey失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True
End If
```

这样用户只有插上你提供的 UKey，才能打开你开发的程序。

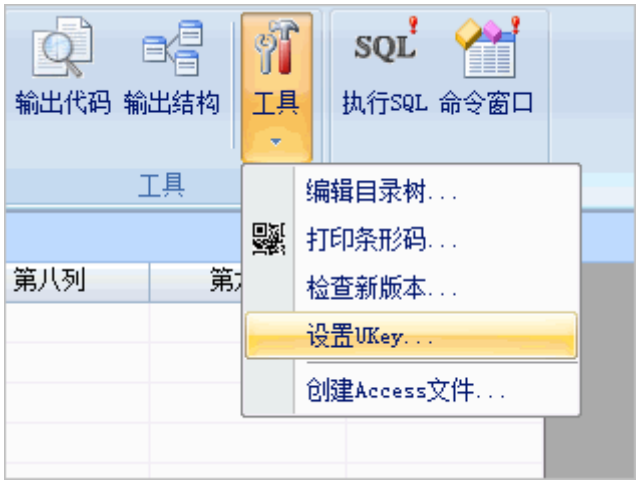
4.23.5 用户算法函数

UKey 有一个用户加密函数 Encrypt2，此函数的加密算法由用户决定。

Encrypt2 函数使用步骤：

1、插上你从 Foxtable 官方采购的 UKey

2、在菜单中执行“设置 UKey”命令：



3、在窗口中输入用户算法密匙(可以单击右侧的按钮随机生成)，然后单击“设置”按钮设置：



重要提示：

A、同样的 UKey，不同的用户算法密匙，用 Encrypt2 函数加密同样的字符，会得到不同的加密结果，

B、所有的 UKey 都必须采用相同的用户算法密匙设置，以保证 Encrypt2 函数的算法相同。

C、必须妥善保存自己的用户算法密匙，一旦丢失，Encrypt2 函数的算法将没有办法和以前的 UKey

保持一致。

D、即使有人从 FoxTable 官方采购到固化算法你完全一样的 UKey(实际上不可能，除非你泄露了订货编号)，他也只能做到的 Encrypt1 算法函数和你相同，而 Encrypt2 函数算法是没有可能和你相同的，除非你将用户算法密匙 也泄露给他了。

所以必须妥善保管好 UKey 的订货编号和用户算法密匙，而且最好分别保存在不同的位置，不得向任何人泄露，以确保安全。

4、设置好用户算法密匙后，选择“算法测试”页，输入要加密的字符，单击“用户算法”按钮，得到加密结果：



5、现在就可以在代码中使用上图所示的加密结果，例如我们可以在 BeforeOpenProject 事件中使用下面的代码：

```
If UKey.Start() Then
    If UKey.Encrypt2("abc") <> "483DBF9FDD0574C0" Then
            MessageBox.Show("请插上XX公司提供的UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
        e.Cancel = True
    End If
Else
    MessageBox.Show("启动UKey失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True
End If
```

这样用户只有插上你提供的 UKey，才能打开你开发的程序。

4.23.6 软件加密示例

下面是一个简单实用的软件加密示例。

1、在 BeforeOpenProject 事件中设置下面的代码：

```
If UKey.Start() Then
    If UKey.Encrypt2("abc") <> "483DBF9FDD0574C0" Then
        MessageBox.Show("请插上XX公司提供的UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
        e.Cancel = True
    End If
Else
    MessageBox.Show("启动UKey失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    e.Cancel = True
End If
```

上面的代码采用用户加密函数Encrypt2判断用户已经插上你公司提供的UKey，如果不是，则拒绝打开项目。

2、增加一个计划，执行间隔为10000毫秒，也就是10秒，计划代码为：

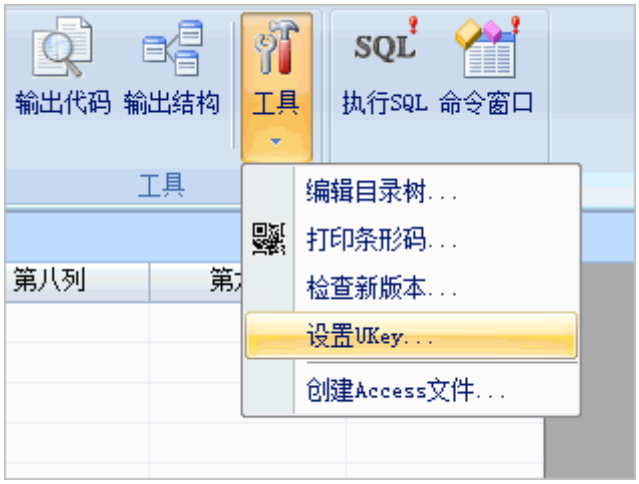
```
If UKey.Encrypt1("abc") <> "E5FC1B19625C5C4A" Then
    MessageBox.Show("请插上XX公司提供的UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    DataTables.Save()
    Syscmd.Project.Exit
End If
```

上面的代码，每10秒钟执行一次，采用固化加密函数Encrypt1判断你提供的UKey是否依然插在计算机上，如果没有，则保存项目后退出。

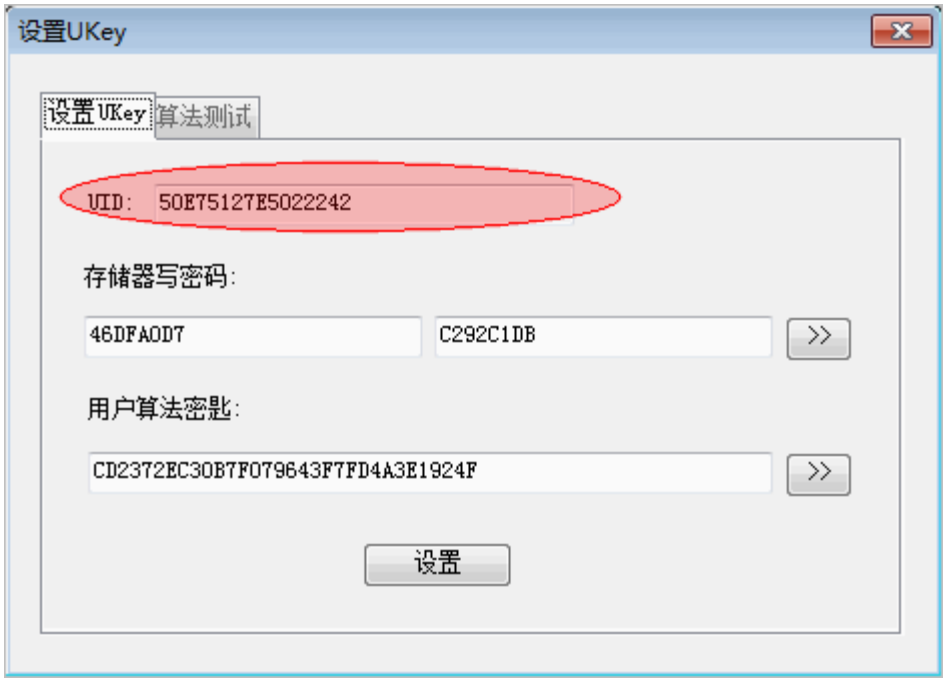
4.23.7 获取 UKey 编号

UKey 有一个 UID 属性，返回 UKey 的编号，此编号是唯一的，因为没有任何两个 UKey 的 UID 会相同。
我们可以很方便地利用 UID 进行身份验证，增加企业管理系统的安全性。

例如一个管理系统，肯定有用户登录界面，输入用户名和密码才能进入系统，但只要是密码，都有被猜出和破解的可能性，如果我们再利用 UKey 进行身份验证，不仅需要输入正确的用户名和密码，而且需要插上正确的 UKey，才能进入系统，那么将大大增加系统的安全性；这就是为什么几乎所有网银都要使用 UKey 的原因。
要使用 UKey 进行身份识别，首先要获取 UKey 的 UID：
1、在菜单中执行“设置 UKey”命令：



2、每插上一个 UKey，下图就会自动显示其 UID:

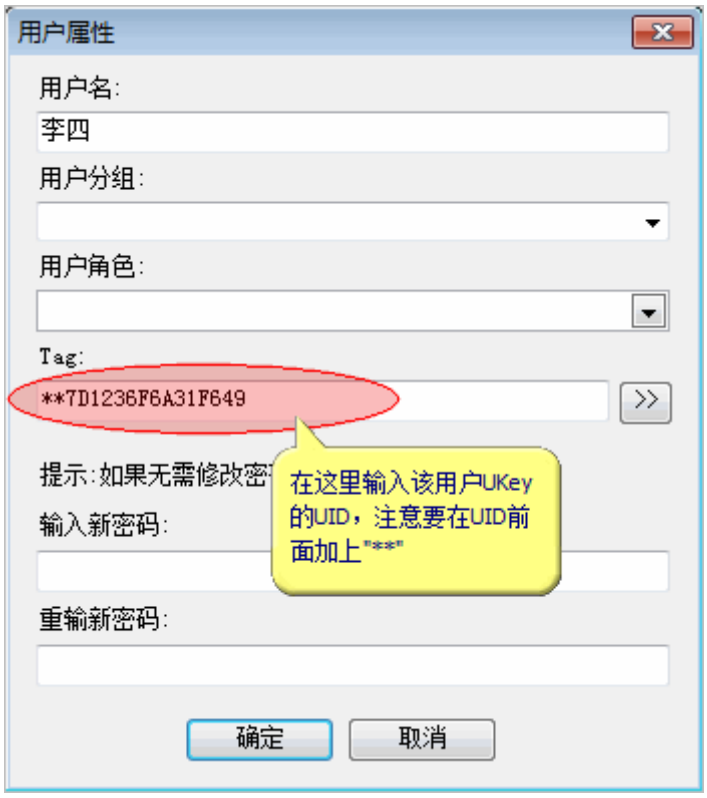


4.23.8 身份验证示例

下面介绍如何在用户登录的时候，使用 UKey 进行身份验证。

使用系统登录时的身份验证

如果使用 Foxtable 自带的用户管理和登录窗口，可以非常简单地加上 UKey 身份验证功能。只需在用户属性窗口的 Tag 输入框，输入分配给该用户的 UKey 的 UID 即可，注意 UID 前面必须加上两个“*”号：



那么此用户登录的时候，必须插上分配给他的 UKey，否则即使输入正确的用户名和密码，也会提示：



自定义用户管理时身份验证

如果使用自定义用户管理，加上 UKey 身份验证功能也是比较方便的。
关于自定义用户管理，可以先参考：自定义用户与权限管理，现在要在这个基础上加上 UKey 身份验证功能。

1、在原来的“Users”表增加一名为 UID 的字符列，现在此表合计有 Name、Group、Password、UID 四列，分别用于保存用户名、用户分组、密码和 UID。

2、修改“更改用户”窗口，加上输入 UID 的输入框，并将此输入框的名称改为 UID：



2、将“更改用户”窗口的 AfterLoad 事件代码改为(加粗的一行为新增加的代码)：

```
Dim UserName As String
Dim UserGroup As String
Dim PassWord As String
Dim cmd As New SQLCommand
Dim dt As DataTable
If Forms("用户管理").Opened Then '正常打开
    UserName = Forms("用户管理").Controls("ListBox1").SelectedItem
Else
    UserName = _UserName '_UserName 为全局变量,用于保存当前用户名
    e.Form.Controls("UserName").Enabled = False
    e.Form.Controls("UserGroup").Enabled = False
End If
e.Form.Text = e.Form.Text & "-" & UserName
cmd.ConnectionName = "User"
cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
dt = cmd.ExecuteReader
If dt.DataRows.Count = 1 Then
    e.Form.Controls("UserName").Value = UserName
    e.Form.Controls("UserGroup").Value = dt.DataRows(0)("Group")
    e.Form.Controls("PassWord").Value = dt.DataRows(0)("PassWord")
    e.Form.Controls("UID").Value = dt.DataRows(0)("UID")
Else
    MessageBox.Show("用户不存在或者已经被删除!","提示",MessageBoxButtons.OK)
    e.Form.Close()
End If
```

3、将“更改用户”窗口的“确定”按钮的代码改为(加粗的两处为新增加或有改动的代码)：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim UserGroup As String   = e.Form.Controls("UserGroup").Value
```

```
Dim PassWord As String = e.Form.Controls("PassWord").Value
Dim UID As String = e.Form.Controls("UID").Value
Dim OldUserName As String
Dim cmd As New SQLCommand
Dim Parts() As String = e.Form.Text.Split("-"c)
OldUserName =Parts(Parts.Length -1)
cmd.ConnectionName = "User"
If UserName = "" OrElse UserGroup = "" Then
    Messagebox.show("请输入用户名和用户分组!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
  Return
End If
cmd.CommandText = "Update {Users} Set [Name] = '" & UserName & "',[Group] = '" & UserGroup
cmd.CommandText = cmd.CommandText & "', [Password] = '" & Password & "', [UID] = '" & UID & "' Where [Name]= '" & OldUserName & "'"
If cmd.ExecuteNonQuery = 1 Then '返回1表示更改成功
  If Forms("用户管理").Opened Then
    With Forms("用户管理").Controls("ListBox1")
      .Items(.SelectedIndex) = UserName
    End With
  End If
  e.Form.Close
Else
  Messagebox.show("更改用户失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

4、同样在“增加用户”窗口，加上 UID 输入框，并将此输入框的名称改为 UID：



5、然后将“增加用户”窗口的“确定”按钮代码改为(加粗处为新增加或有变动的代码)：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim UserGroup As String  = e.Form.Controls("UserGroup").Value
```

```
Dim PassWord As String = e.Form.Controls("PassWord").Value
Dim UID As String = e.Form.Controls("UID").Value
Dim cmd As New SQLCommand
cmd.ConnectionName = "User"
If UserName = "" OrElse UserGroup = "" Then
    Messagebox.show("请输入用户名和用户分组!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Select Count(*) From {Users} Where Name = '" & UserName & "'"
If cmd.ExecuteScalar > 0 Then
      Messagebox.show("增加用户失败，已经能存在同名用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Insert Into {Users} ([Name],[Group],[Password],[UID]) Values ('"
cmd.CommandText = cmd.CommandText & UserName & "','" & UserGroup & "','" & Password & "','" & UID & "')"
If cmd.ExecuteNonQuery = 1 Then '返回1表示增加成功
    If Forms("用户管理").Opened Then
      With Forms("用户管理").Controls("ListBox1")
        .Items.Add(UserName)
        .SelectedIndex = .Items.Count -1
      End With
    End If
    e.Form.Close()
End If
```

6、将“用户登录”窗口的确定按钮代码改为(加粗部分为新增加的代码)：

```
Dim UserName As String = e.Form.Controls("UserName").Value
Dim cmd As New SQLCommand
Dim dt As DataTable
Dim dr As DataRow
cmd.ConnectionName = "User"
If UserName = ""  Then
    Messagebox.show("请选择用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Return
End If
cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
dt = cmd.ExecuteReader
dr = dt.DataRows(0)
Dim UID As String = dr("UID")
If UID > "" Then
    If UKey.Start = False OrElse UID <> UKey.UID  Then
          MessageBox.show("请插上登录用户的UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
```

```
        Return
    End If
End If
If e.Form.Controls("PassWord").Value = dr("Password") Then
    _UserName = UserName
    _UserGroup = dr("Group")
    e.Form.Close
Else
    Messagebox.show("密码错误!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

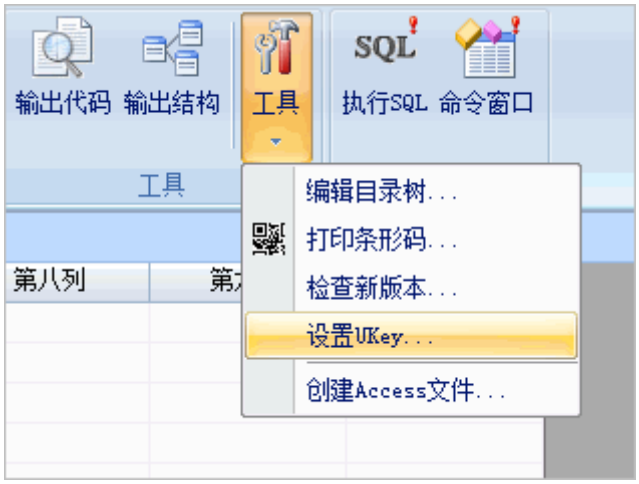
4.23.9 设置存储器写密码

官方提供的配套 UKey 内置 400 个字节的存储器。

要使用存储器，首先必须设置写密码，这是显然的，如果任何人都能向存储器写入内容，就没有安全性可言了：

1、插上你从 Foxtable 官方采购的 UKey

2、在菜单中执行“设置 UKey”命令：



3、在设置窗口输入存储器写密码（也可以单击右侧的按钮随机生成），然后单击“设置”按钮：



4、存储器写密码由两组 8 位 16 进制数组成，必须妥善保管好写密码，因为如果新旧 UKey 的写密码不同，你将无法用同样的代码向存储器中写入数据。

4.23.10 存储器的数据读写

向存储器写数据

UKey 有一个 WriteStr 函数，用于向存取器写入数据，语法为：

WriteStr(Index,Content,Password1,Password2)

Index: 写入位置,因为存储器只有 400 个字节，所以 Index 的范围是 0 到 399

Content： 写入内容

Password1： 第一组写入密码

Password1： 第二组写入密码

如果写入成功，则返回 True，否则返回 False。

示例：

```
If UKey.Start() Then
    If UKey.WriteStr(0,"联想ThinkPad","46DFA0D7","C292C1DB") = True
        MessageBox.Show("成功写入数据!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

上述代码中的"46DFA0D7"和"C292C1DB"是我们在上一节中设置好的写密码。

我们知道，存储器合计只有400个字节，通常我们会将存储器分成若干区域，例如0到49保存授权用户名称，50到59保存启动次数，60到100保存权限信息等等。

所以再向存储器写入数据的时候，通常需要计算写入内容的长度，在这里每个汉字(包括全角的字母和符号)等于两个字节，普通字符为1个字节，例如"联想ThinkPad"的长度为12个字节。

UKey有一个函数LenStrA，用计算字符串的字节长度。

示例：

```
If UKey.Start() Then
    Dim Val As String = "联想Lenovo"
    If UKey.LenStrA(Val) > 50 Then
        MessageBox.Show("用户名长度不能超过50个字节!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
      If UKey.WriteStr(0,val,"46DFA0D7","C292C1DB") = True
            MessageBox.Show("成功写入数据!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
      End If
    End If
End If
```

需要注意的是，由于存储器只有400个字节，所以起始位置和写入内容的长度之和，不能超过400，否则将无法写入。

从存储器读数据

UKey 有一个 ReadStr 函数，用于从存取器读取数据，语法为：

ReadStr(Index,Length)

Index：开始读取的位置

Length：读取长度

如果读取成功，则返回读取的内容，否则返回空值。

需要注意的是，由于存储器只有 400 个字节，所以 Index 和 Length 两个参数之和，不能超过 400，否则将读不到数据。

示例：

```
If UKey.Start() Then
    Dim val As String
    val = UKey.ReadStr(0,50).Trim()
    If val > "" Then
        MessageBox.Show("授权用户:" & val ,"提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
End If
```

4.23.11 软件加密与授权示例

假定你的软件按使用时间或启动次数销售，需要插上 UKey 才能使用，而且超过指定的日期或启动次数后，将不能再启动软件。

1、首先要分配存储器空间，这里假定：0-49 存储客户名称，50-59 存储截止日期，60-64 存储允许

启动次数，65-69 存储实际已经启动 过的次数。

2、然后你单独建立一个项目，在这个项目中设计一个 UKey 授权窗口，用于输入授权用户、截止日期和允许启动次数：



3、授权按钮的代码设置为：

```
Dim kh As String = e.Form.Controls("TextBox1").Text
Dim rq As String =  Format(e.Form.Controls("DateTimePicker1").Value,"yyyy-MM-dd")
Dim cs As String = e.Form.Controls("NumericComboBox1").Value
Dim qd As String = "     " '5个半角空格,用于初始化保存实际启动次数的存储器空间.
Dim pw1 As String = "46DFA0D7"
Dim pw2 As String = "C292C1DB"
kh = kh.PadRight(50," ").SubString(0,50) '客户名称最多50个字符,不够50用空格补
cs = cs.PadRight(5," ").Substring(0,5) '启动次数最多允许5位数,不够的用空格补
If UKey.Start() Then
    If UKey.WriteStr(0,kh,pw1,pw2) AndAlso UKey.WriteStr(50,rq,pw1,pw2) AndAlso UKey.WriteStr(60,cs,pw1,pw2) AndAlso UKey.WriteStr(65,qd,pw1,pw2)
        Messagebox.Show("授权成功!","提示", MessageBoxButtons.OK,MessageBoxIcon.Information)
    Else
        Messagebox.Show("写入授权信息失败!","提示", MessageBoxButtons.OK,MessageBoxIcon.Information)
    End If
Else
    Messagebox.Show("请插上UKey!","提示", MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

你也许会奇怪，为什么客户名称和次数，都要固定长度，不够的补空格？

你想一下，如果你之前已经对某个UKey进行过授权，授权客户是“北京市公安局”，现在想重新授权，将客户改为“联想电脑”，如果不采用固定长度且补空格的方式，那么重新授权后，客户名称将是"联想电脑安局"。

5、现在回到原来的项目中，将项目事件BeforeOpenProject事件代码设置为：

```
If UKey.Start() Then
    If UKey.Encrypt2("abc") <> "483DBF9FDD0574C0" Then
```

```
            MessageBox.Show("请插上XX公司提供的UKey!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
            e.Cancel = True
        Else
            Dim khs As String = UKey.ReadStr(0,50) '从存储器读取0到49这50个字节的内容,也就是授权客户名称
            Dim rqs As String = UKey.ReadStr(50,10) '从存储器读取50到59这10个字节的内容,也就是截止日期
            Dim css As String = UKey.ReadStr(60,5) '从存储器读取60到64这5个字节的内容,也就是允许启动次数
            Dim yqs As String = UKey.ReadStr(65,5) '从存储器读取65到69这5个字节的内容,也就是已经启动次数
            Dim rq As Date
            Dim cs As Integer
            Dim qs As Integer
            '从存储器读取出来值的是字符,所以还需要转换一下,因为截止日期是Date型,次数是Integer型
            Date.TryParse(rqs,rq)
            Integer.TryParse(css,cs)
            Integer.TryParse(yqs,qs)
            qs = qs + 1 '已经启动次数加1
            UKey.WriteStr(65,qs,"46DFA0D7","C292C1DB") '将已经启动次数写入存储器
            If qs > cs '如果已经启动次数大于允许启动次数
                MessageBox.Show("你的UKey已经超过允许启动次数!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
                e.Cancel = True
            ElseIf Date.Today > rq '如果今天的日期大于截止日期
                MessageBox.Show("你的UKey已经过期!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
                e.Cancel = True
            End If
            If e.Cancel = False Then
                MessageBox.Show("尊敬的用户" & khs.Trim() & ",欢迎使用本软件!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
            End If
        End If
    Else
        MessageBox.Show("启动UKey失败!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
        e.Cancel = True
    End If
```

上面的代码采用用户加密函数Encrypt2判断用户已经插上你公司提供的UKey，并判断此UKey是否还在授权范围之内，所有条件符合才允许打开你的项目。

7、最后增加一个计划，执行间隔为10000毫秒，也就是10秒，计划代码为：

```
If UKey.Encrypt1("abc") <> "E5FC1B19625C5C4A" Then
        MessageBox.Show(" 请 插 上 XX 公 司 提 供 的 UKey!"," 提 示 ",MessageBoxButtons.OK,MessageBoxIcon.Information)
    DataTables.Save()
    Syscmd.Project.Exit
End If
```

上面的代码，每 10 秒钟执行一次，采用固化加密函数 Encrypt1 判断你提供的 UKey 是否依然插在计算机上，如果没有，则保存项目后退出。

4.23.12 另一种软件加密方式

采用 UKey 进行软件加密和授权，是目前最为可靠和安全的方式。
如果不想用 UKey，也可以用下面介绍的方法，即获取计算机的硬件配置信息，然后据此生成机器码和注册码，输入正确的注册码才能启动系统。

Foxtable 提供了一个全局变量 ComputerId，该变量返回 12 个字符，前 4 位表示硬盘信息，中间 4 位表示主板信息，最后 4 位表示 CPU 信息。
多数情况下，不同的电脑会有不同的 ComputerId，而且即使格式化硬盘重新安装系统，ComputerId 的返回值也不会改变。

这种方式可以省去采购 UKey 的费用，但是和使用 UKey 加密相比，存在以下不足：
1、安全性差，对于有稍微有点破解常识的人来说，破解这样的加密系统可以说是手到擒来。
2、虽然理论上不同的电脑有不同的 ComputerId，但是实际上 ComputerId 相同的电脑是经常存在的，而且格式化或重新安装系统后，ComputerId 还是 会有一定的概率发生变化的。
由于 UKey 的费用并不高，而安全性和可靠性却强得多，所以官方推荐大家采用 UKey 加密，只有并不重要的系统才可以考虑下面的加密方式。
在学习下面的示例之前，请大家先回顾一下：保存设置信息

示例

本示例可以参考 CaseStudy 目录下的文件“注册系统设计演示.table”。
假定你提供了一个项目给客户试用，希望前 30 次正常启动，30 次后出现注册窗口，60 次后如果没有注册，则拒绝进入。
设计步骤：
1、设计一个注册码输入窗口，窗口类型为模式，窗口名称为“注册”，窗口中的控件如下图：



2、将窗口的 AfterLoad 事件设为：

```
e.Form.Controls("TextBox1").Value = ComputerId
e.Form.Controls("TextBox2").Value = GetConfigValue("Register" & ComputerId, "")
```

因为可能有不同的电脑使用项目，所以用于保存注册码的设置名称必须包括机器码，以避免注册码被覆盖。

3、将窗口中“确定”按钮的代码设为：

```
SaveConfigValue("Register" & ComputerId, e.Form.Controls("TextBox2").Value)
e.Form.Close()
```

4、最后将项目的 AfterOpenProject 事件设为：

```
Dim n As Integer = GetConfigValue("Count",1)
Dim Code As String = GetConfigValue("Register" & ComputerId,"")
Dim Ok As Boolean
If Code > "" AndAlso DecryptText(Code,"abc","abc") = ComputerId Then '如果注册码正确
    OK = True
Else
    If n > 30 Then
        Forms("注册").Open()
        Code = GetConfigValue("Register" & ComputerId,"")
        If Code > "" AndAlso DecryptText(Code,"abc","abc") = ComputerId Then '如果注册码正确
            OK = True
        End If
    End If
    If n > 60 AndAlso Ok = False Then
        Messagebox.Show("您正在使用的产品已经超出试用次数!")
        Syscmd.Project.Exit()
    End If
End If
n = n + 1
SaveConfigValue("Count",n)
```

至此我们为自己的项目文件加上了试用和注册功能，用户提供机器码之后，可以在命令窗口执行下面的代码计算得出注册码：

```
Output.Show(EncryptText("用户提供的机器码","abc","abc"))
```

提示:

上面的代码，使用次数是保存在设置信息中的，而设置信息是存储在项目文件自身中的，所以覆盖项目文件，意味着使用次数会归零，用户可以继续使用。

所以实际应用的时候，应该同时在注册表中保存使用次数，启动的时候同时从设置信息和注册表读取使用次数，并取最大值。

关于注册表的读写，参考：**Registry**

4.24 协同开发

4.24.1 多人协同开发

尽管 Foxtable 的开发效率非常高，以至于有的时候，一个基于 Foxtable 的个人开发者，足足抵上一个小小的开发团队，成就了不少“个人英雄”。
但还是有一些管理项目，可能会需要多人协同开发，以便更快更好地完成开发任务。
从 2014 年 9 月开始，Foxtable 开始内在支持多人协同开发，例如多人同时编辑同一个表的不同事件， 不仅可以分别保存，互不影响，而且可以即时在对方电脑生效，无需重新打开项目。
不过既然是多人协同开发，分工合作是必须的，所以必然涉及到设计权限的指定问题，例如让不同的人负责不同的表或窗口的设计，各司其职，各负其责。
本章的内容就是介绍如何指定设计权限，需要注意的是，和设计权限指定相关的功能，试用版是不能使用的。

4.24.2 指定责任设计者

此功能对于试用版无效。
多人协同开发的时候，责任明确，各司其职是很重要的，Foxtable 可以将每项设计工作指定到具体的开发人员。
项目属性、表属性、菜单属性、窗口属性的设置窗口，都有一个名为“责任设计者”属性：



对于指定了责任设计者的表、窗口、菜单，只有指定的责任设计者才能对其进行修改设计。
如果在项目属性指定了责任设计者，那么项目属性、全局变量、全局代码、外部引用、计划管理、文件监视器、设备监视器等和项目整体相关的设计任务，均由此责任设计者负责。

4.24.3 谁创建谁设计

此功能对于试用版无效。

在设置项目属性的时候，有个名为“创建者设计”的属性，如果将其设置为 True，那么:
1、谁创建的表，谁就是责任设计者。
2、谁创建的窗口，谁就是责任设计者。
这个属性默认为 False，如果一个项目计划多人同时开发，那么请将此属性设置为 True，这样就无须后来再指定责任设计者。

4.24.4 让开发者退居幕后

开发者拥有最高的权限，即使指定了责任设计者，开发者本人还是可以对任何表和窗口进行设计，项目属性和菜单亦如此。
为了避免冲突，建议让开发者退居幕后，平时不要使用开发者登录，除非需要修改某任务的责任设计者，因为只有开发者才能更改责任设计者。
还有一件事情需要注意，我们知道自定义菜单生效的时候，只有开发者登录，才能同时"按 Ctrl + Shift + F12"回到系统菜单，所以负责菜单的设计者要做到下列两点之一：
1、自定义菜单加一个按钮，按纽代码为：

```
Syscmd.Project.OpenSystemMenu()
```

这样其它设计者单击这个按钮，可以回到系统菜单。
2、或者在设计菜单的时候，将相关设计功能，例如项目属性、表属性、窗口管理、菜单设计等等，全部加入到菜单的某个功能区（假定这个功能区名称为"设计"）中，方便其他设计者调用；此外将所有的设计者划分到某用户分组(假定这个用户分组的名称为"设计者")中，在 AfterOpenProject 事件中加上代码：

```
If User.Group <> "设计者" Then
    RibbonTabs.Remove("设计")
End If
```

4.24.5 只加载自己负责的表

如果每个设计者负责一个相对独立的模块，那么可以在打开项目的时候，只加载和此模块相关的表。
前面的《表的动态加载和卸载》，对此功能有详细讲述，这里就不重复了。

4.24.6 拆分成多个项目设计

前面的介绍的方法，是将所有功能模块集成在一个项目中，多个设计人员对同一个项目进行开发。我们也可以将一个大型系统拆分成多个项目开发，例如销售系统、采购系统、财务系统、工资系统、人事系统、仓管系统、生产系统等等，但所有的系统都共用一个数据库，然后在每个项目加上切换到其它项目的按钮，采用下面介绍的方法打开其他项目。
我们可以通过下面的代码打开项目文件：

```
Syscmd.Project.Open(File, Args1, Args2, Args3....)
```

File: 要打开的项目文件名称，含路径。
Args1, Args2, Args3：可选，传递给要打开项目的的参数，个数不限。

系统变量 OpenFileArgs 是一个集合，目标项目被打开后，可以通过此集合获得传递过来的参数。

示例

以 CaseStudy 目录下的项目文件“自定义用户管理.Table”为例，参考：自定义用户管理。

该项目文件原 AfterOpenProject 事件代码为：

```
Forms("用户登录").Open()
```

如果我们将 AfterOpenProject 事件代码改为：

```
If OpenFileargs.Count = 2 Then '如果传递了两个参数
    Dim UserName As String = openFileargs(0) '获得第一个参数,也就是用户名
    Dim cmd As New SQLCommand
    Dim dt As DataTable
    Dim dr As DataRow
    cmd.ConnectionName = "User"
    If UserName = "" Then
        Messagebox.show("请选择用户!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
        Return
    End If
    cmd.CommandText = "Select * From {Users} Where [Name] = '" & UserName & "'"
    dt = cmd.ExecuteReader
    dr = dt.DataRows(0)
    If dr("Password") = OpenFileArgs(1) Then '如果密码等于第二个参数
        _UserName = OpenFileargs(0)
        _UserGroup = dr("Group")
        Return '直接返回,不出现登录窗口
    End If
End If
Forms("用户登录").Open()
```

那么我们就可以在其他项目中通过下面的下面的代码直接打开此项目：

```
Dim fl As String = "C:\foxtable\商业版\CaseStudy\自定义用户管理.Table"
Syscmd.Project.Open(fl,"吴晓","888")
```

传递过去的用户名为“吴晓”，密码为“888”，如果用户名和密码正确，将直接进入系统，否则出现登陆窗口。

这样每个项目都是独立的，开发的时候容易驾驭，而实际使用的时候，这些项目看起来又是一个整体(共用数据库和用户信息)，能相互无缝切换。

但是采用这种方式不能采用系统内置的用户管理，必须自定义用户管理，代码的重用也不如集成在一个项目开发好。

重要提示：

已经发布的项目，也可以采用上述方法打开。

发布后的项目文件位于 Publish 目录下的 Project 子目录下，文件名和原项目文件相同，后缀名为：FoxEx

4.25 源码保护

4.25.1 保护整个项目

此功能仅向高级开发版用户提供。

大型项目的开发，通常需要多个程序员组成一个开发团队，进行协同开发，最新版本的 Foxtable 对于协同开发的支持非常完善，具体可以参考前面的介绍。
在多人协同开发的时候，如何有效防止参与开发的程序员盗用最终的开发成果，一直是一个难题。

如果使用 Foxtable 高级开发版，则不用担心这个问题，我们可以通过高级开发版限制开发人员的权限，让其只能参与开发，无法带走开发成果。
高级开发版通常配合普通开发版使用，团队主管使用高级开发版，团队成员使用普通开发版。
当然普通开发者也可以使用高级开发版，只需主管首先插上自己专属的高级开发版加密锁，对项目设置好权限即可，之后其他人即使使用高级开发版开发项目，也只能在设定权限下进行操作。
即使是一个人开发，也可以使用高级开发版授权，将授权 ID 设置为高级开发版自身的 ID 即可，这样其他人即使拿到你的项目，即使知道项目的开发者密码，也无法打开你的项目。
高级开发版可以实现：

1、禁止团队成员非法使用项目

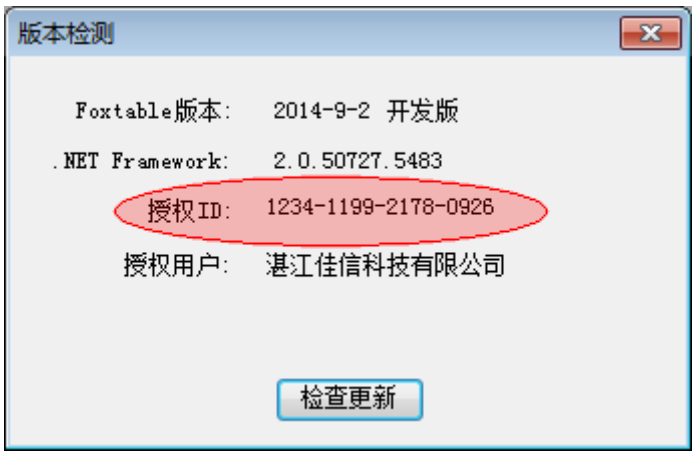
通过高级开发版，可以对项目进行授权设置，使得只有插上经授权的加密锁，才能打开该项目，否则即使是开发者本人，离开了授权的加密锁，也无法打开项目。

2、禁止团队成员一些关键权限

开发团队的主管通过高级开发版，还可以禁止参与团队成员在开发过程中的以下权限：
1、输出结构
2、输出代码
3、发布项目
4、将窗口另存为模版
5、将菜单另存为模版
显然，将上述权限全部禁止后，参与开发的程序员如果要盗用最终的开发成果，除了从零开始重写，别无他法。
所以团队开发版的推出，大大增强了团队主管的控制能力，能更好地第保护企业权益。

使用高级开发版进行授权的步骤：

1、首先逐个插上要授权的其他开发版的加密锁，启动 Foxtable 开发版，执行菜单命令“杂项 - 工具 - 检查新版本”，记录好窗口中显示的授权 ID：



2、然后插上高级开发版的加密锁，打开要进行授权设置的项目，在菜单的“管理项目”功能区单击“项目信息”：

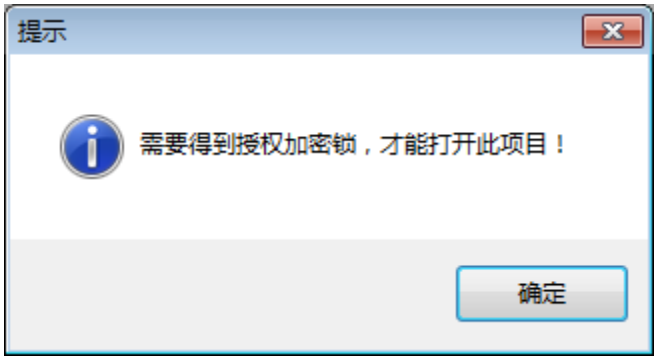


3、在出现的窗口中，选择“授权”页，输入要授权的加密锁的授权 ID，每行一个：

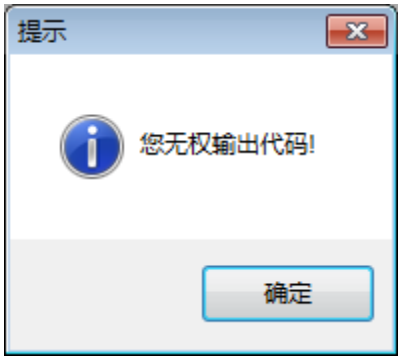


提示：只有高级开发版，才会出现“授权”页。

4、之后没有经过授权的加密锁，即使知道开发者密码，打开项目的时候也会提示：



5、团队成员使用经过授权的加密锁打开项目进行开发过程中，如果要执行未经授权的操作，例如输出代码的时候，会出现提示：

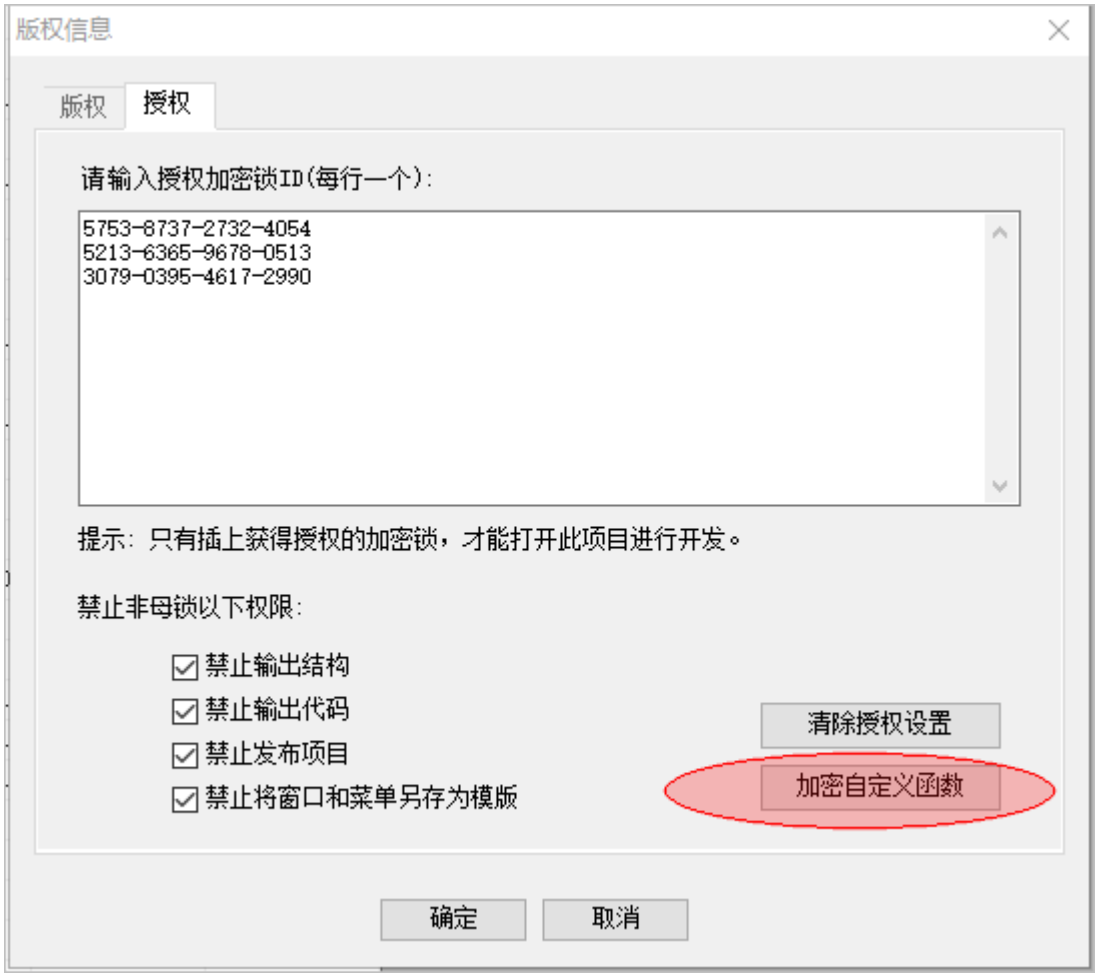


4.25.2 保护关键代码

此功能仅向高级开发版用户提供。

如果你是第三方开发者，基于 Foxtable 开发了一些拓展功能，希望销售给客户，同时又不想不泄露关键代码，这在 Foxtable 2017 之前，是一个难以解决的问题。

从 Foxtable 2017 开始，高级开发版的授权页面多了一个“加密自定义函数”的按钮，你可以选择加密部分或全部自定义函数：



加密自定义函数对于现有项目没有任何影响，而是同一个文件夹中生成一个新的项目文件，格式

为：项目名称_Encrypt.foxdb
被加密的自定义函数，可以正常被调用执行，但将不再出现在函数管理器中，任何人都将无法查看其代码。
如果你希望基于 Foxtable 进行一些源代码级别的商业交易，但希望有选择性地交付部分源代码，同时不影响整体功能的演示和运行，那么“加密自定义函数”将为你解决一个大难题。
自定义函数一旦加密，将无法再还原，所以务必要保管好原项目。
购买者可以基于加密后的项目进行拓展开发，对于销售者而言，加密后的项目仅限于交易，所有后期开发应继续基于未加密的原始项目。
除了整个项目的交易，你可以直接交易某些自定义函数，例如让客户将其开发中的项目发给你，你给他的项目加上一些自定义函数，并将这些函数加密，然后将项目发回给客户，这样客户可以使用你设计的函数，但是看不到函数源代码。
重要技巧：
Foxtable 在编译的时候，会将函数名称进行混淆，如果你的函数越多，这种混淆保护的力度就越大，为了更好地保护你的代码，可以将一些代码较多的函数拆分成多个小函数，原则很简单：函数越多越好。

4.25.3 更改授权母锁

如果某个项目没有进行过授权，那么插上任何一个高级开发版的加密锁，打开项目后，在菜单的“管理项目”功能区单击“项目信息”，都会出现授权页面。

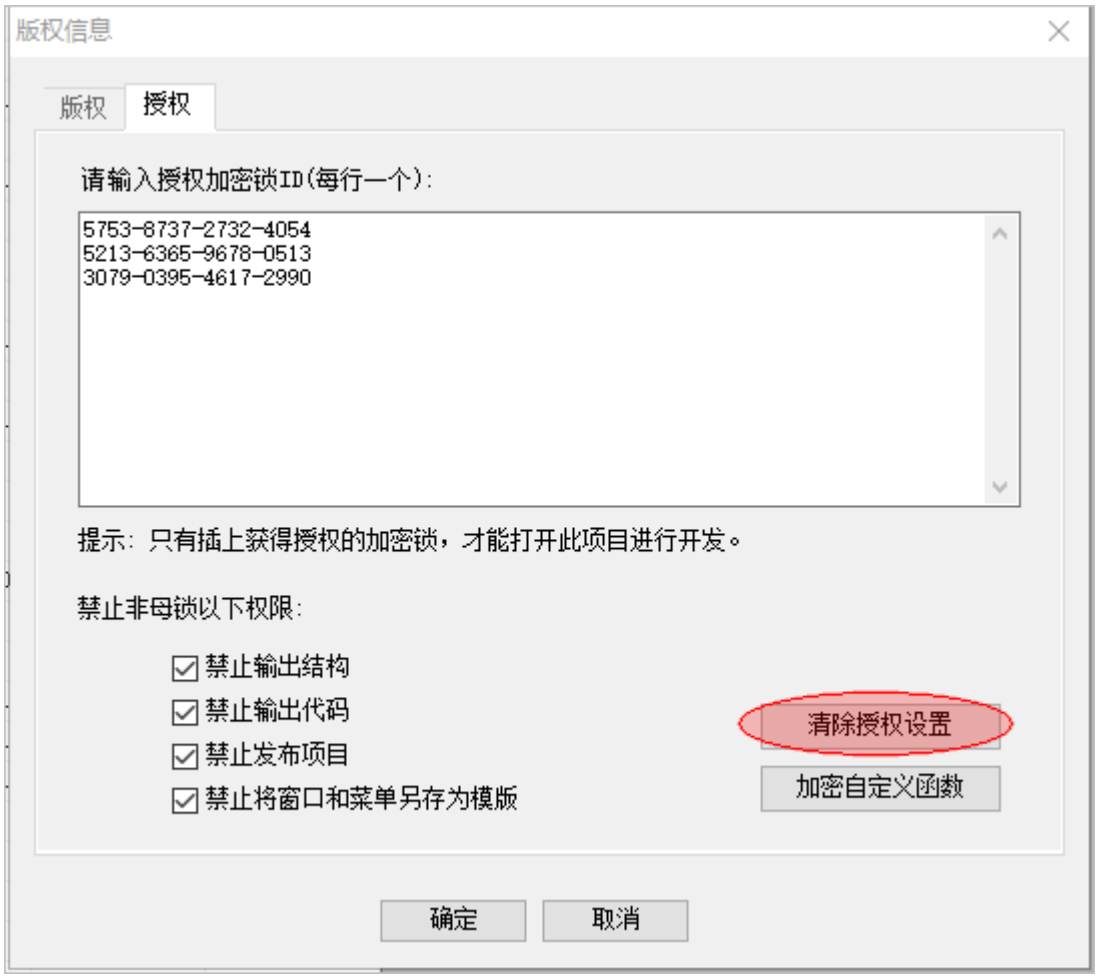
一旦进行过授权，那么只有插上第一次授权的高级开发版的加密锁，才能修改授权，用其他任何加密锁（包括其他高级开发版的加密锁）打开项目后，都不会再出现授权页面。

负责对项目进行授权的高级开发版加密锁，称之为项目的“授权母锁”。

必须保管好授权母锁，一旦丢失，你将再也无法进入授权页面。

授权母锁是无法修改的，如果要更改授权母锁，必须在清除项目的授权设置后，用新的授权母锁进行授权。

如果要清除授权设置，单击下图所示按钮即可：



4.25.4 代码授权模式

当你作为第三方开发者向自己的用户销售代码模块时，可选择的授权方式有：

基于电脑授权

Foxtable 提供全局变量 ComputerID 用于识别当前电脑，返回值和电脑的主板、硬盘以及 CPU 关联。

基于加密锁授权

Foxtable 提供全局变量 DongleID 用于识别用户所采用的 Foxtable 加密锁，这是推荐的方式。

基于项目授权

Foxtable 提供了全局变量 ProjectID 用于识别当前项目，必须注意复制项目的 ProjectID 是相同的。

基于 UKey 授权

你可以向我们购买 UKey 进行加密授权，但你的用户在进行二次开发时，需要同时插上你的 UKey 和 Foxtable 的加密锁，稍显不便。

4.25.5 发布后的授权

你的用户在二次开发完成后，是需要编译发布的，显然对于编译发布后的项目，你应该解除或调整授权方式。

Foxtable 提供全局变量 IssueName，对于编译发布后的项目，此变量的返回值是“运行环境”，你可以据此解除或调整授权方式。

4.26 杂项

4.26.1 语音朗读

FoxTable 可以自动将一段文本朗读出来。
在默认的情况下，Windows Vista/7 可以正常朗读中英文，而 Windows XP 只能朗读英文，不过你可以另外安装语音库来解决这个问题。
实际上，使你用的是 Windows Vista/7，也可以另外安装语音库来获得更好的朗读效果。
提示：无需去刻意理解下面的代码，实际工作的时候，只需套用即可。
示例一
如果你要用默认的设置朗读语音，那么很简单：

```
Dim sp As New DotNetSpeech.SpVoice()
sp.Speak("I am from china.", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFlagsAsync)
```

执行上面的代码，计算机或自动朗读出“I am from china.”
SpVoice 有三个方法，分别为：
Speak： 朗读文本
Pause： 暂停朗读
Resume：继续朗读
示例二
上面的朗读代码是异步的，在朗读的同时，你可以继续进行下一步的操作。
如果是同步朗读，那么在朗读过程中，将挂起程序，不能进行任何操作，直到朗读结束。
同步朗读的代码：

```
Dim sp As New DotNetSpeech.SpVoice()
sp.Speak("I am from china.", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFDefault)
```

示例三
通过设置 Rate 属性，可以调整朗读语速，Rate 的默认值是 0，可以为负数，值越大，语速越快。
你可以在命令窗口分别执行下面两段代码，比较一下朗读速度。
快速朗读：

```
Dim sp As New DotNetSpeech.SpVoice()
sp.rate = 3
sp.Speak("I am from china.", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFlagsAsync)
```

低速朗读：

```
Dim sp As New DotNetSpeech.SpVoice()
sp.rate = -3
sp.Speak("I am from china.", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFlagsAsync)
```

示例四

你可以通过下面的代码来获得所有已经安装的语音库的名称：

```
Dim sp As new DotNetSpeech.SpVoice()
For Each st As Dotnetspeech.SpObjectToken In sp.GetVoices
    Output.Show(st.GetDescription)
Next
```

示例五

朗读的时候，可以选择语音库，例如选择 1 号语音库朗读：

```
Dim sp As New DotNetSpeech.SpVoice()
sp.Voice = sp.GetVoices(String.Empty,String.Empty).Item(0)
sp.Speak("I am from china.", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFlagsAsync)
```

示例六

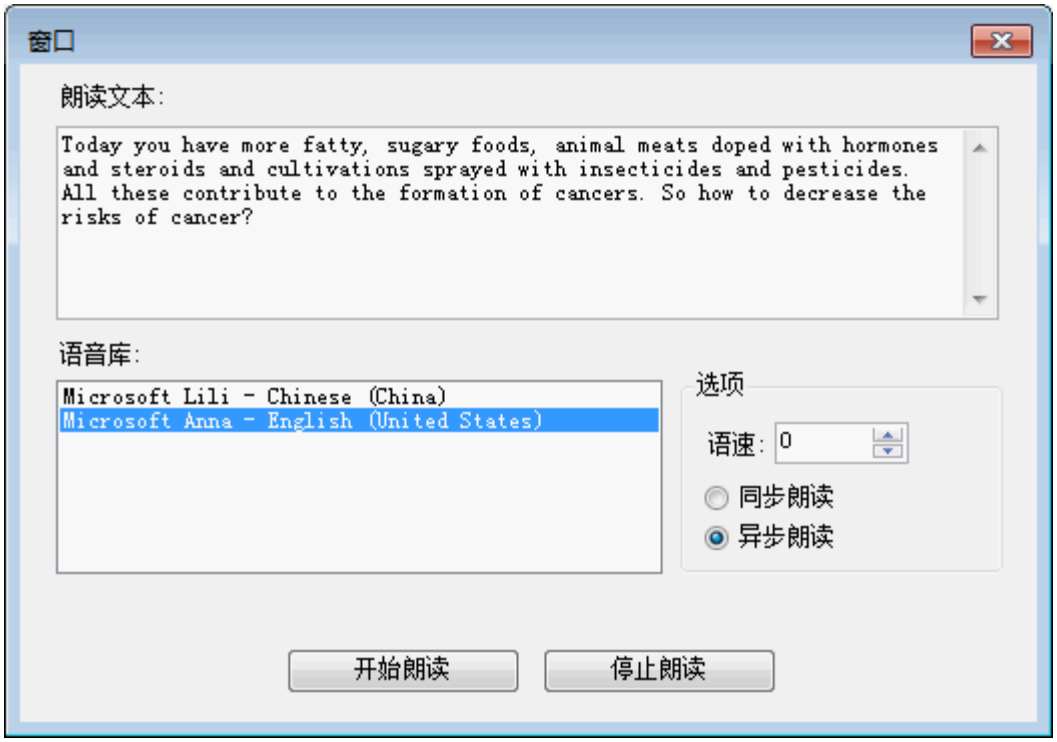
可以将朗读结果录制为一个 wav 文件，例如：

```
Dim Voice As new DotNetSpeech.SpVoice()
Dim Stream As new DotNetSpeech.SpFileStreamClass()
Stream.Open("c:\temp\test.wav", DotNetSpeech.SpeechStreamFileMode.SSFMCreateForWrite, False)
Voice.AudioOutputStream = Stream
Voice.Speak("I love you.",DotNetSpeech.SpeechVoiceSpeakFlags.SVSFDefault)
Stream.Close()
```

示例七

这是一个综合示例，可以打开 CaseStudy 目录下的示例文件：语音朗读.Table

本示例的界面如下：



该示例可以列出已经安装的语音库，客户用已选择语音库，设置语速，并可选择是异步还是同步朗读 ，还可以在阅读过程中停止阅读。
因为开始朗读和停止朗读两个按钮需要使用同一个 SpVoice 才有效，所以 SpVoice 的定义为 Public 变量：

```
Public Speaker As New DotNetSpeech.SpVoice()
```

窗口的 AfterLoad 事件代码为：

```
Dim ls As WinForm.ListBox = e.Form.Controls("ListBox1")
For Each st As Dotnetspeech.SpObjectToken In Speaker.GetVoices
    ls.Items.Add(st.GetDescription)
Next
If ls.items.count > 0 Then
   ls.SelectedIndex = 0
End If
```

上面的代码会在窗口的 ListBox1 中列出已经安装的语音库的名称，并默认选择第一个语音库。
开始朗读按钮的代码为：

```
Dim yy As WinForm.ListBox = e.Form.Controls("ListBox1")
Dim vl As String = e.Form.Controls("txtContent").Value '获得要朗读的文本
Speaker.Rate = e.Form.Controls("boxSpeed").Value '设置语速
Speaker.Voice = Speaker.GetVoices(String.Empty,String.Empty).Item(yy.SelectedIndex) '选择语音库
Speaker.Speak("", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFPurgeBeforeSpeak) '停止之前的朗读
If   e.Form.Controls("rdoAsync").Checked Then
   Speaker.Speak(vl, DotNetSpeech.SpeechVoiceSpeakFlags.SVSFlagsAsync) '异步朗读
Else
   Speaker.Speak(vl, DotNetSpeech.SpeechVoiceSpeakFlags.SVSFDefault) '同步朗读
End If
```

停止朗读按钮的代码为：

```
Speaker.Speak("", DotNetSpeech.SpeechVoiceSpeakFlags.SVSFPurgeBeforeSpeak) '停止朗读
```

4.26.2 语音校对

有了语音朗读功能，我们可以轻松给 Foxtable 加上语音校对功能。
实现步骤：
1、在全局代码中定义一个 Public 变量：

```
Public sp As New DotNetSpeech.SpVoice()
```

2、在表的 ValidateEdit 设置代码：

```
sp.Speak(e.text, DotNetSpeech.SpeechVoiceSpeakFlags.SVSFlagsAsync)
```

这样在表中每次输入一个单元格的数据，都会用语音自动朗读刚刚输入的数据。

4.26.3 自定义帮助文件

管理项目开发完成后，一般会附上 CHM 格式的帮助文件。
为了将帮助文件和系统更好地集成，要注意以下两点：
1、帮助文件必须和项目文件同名，假定你的项目文件为"工资管理.Table"，那么帮助文件就必须命名为"工资管理.chm"
2、帮助文件必须和项目文件处于同一文件夹中。
如果满足上述两点，用户按 F1 键，打开的将是管理项目的帮助文件，而不是 Foxtable 自身的帮助文件。
如果希望单击某按钮能够打开帮助，可以将按钮的 Click 事件代码设置为：

```
ShowHelpWindow()
```

我们知道 CHM 格式的帮助文件实际上是由多个 HTML 文件组合而成的，如果希望打开帮助的时候，能够自动显示某个特定的 HTML 文件，只需直接给 ShowHelpWindow 函数指定要显示的文件名即可，例如：

```
ShowHelpWindow("0008.htm")
```

4.26.4 容易混淆表达式语法

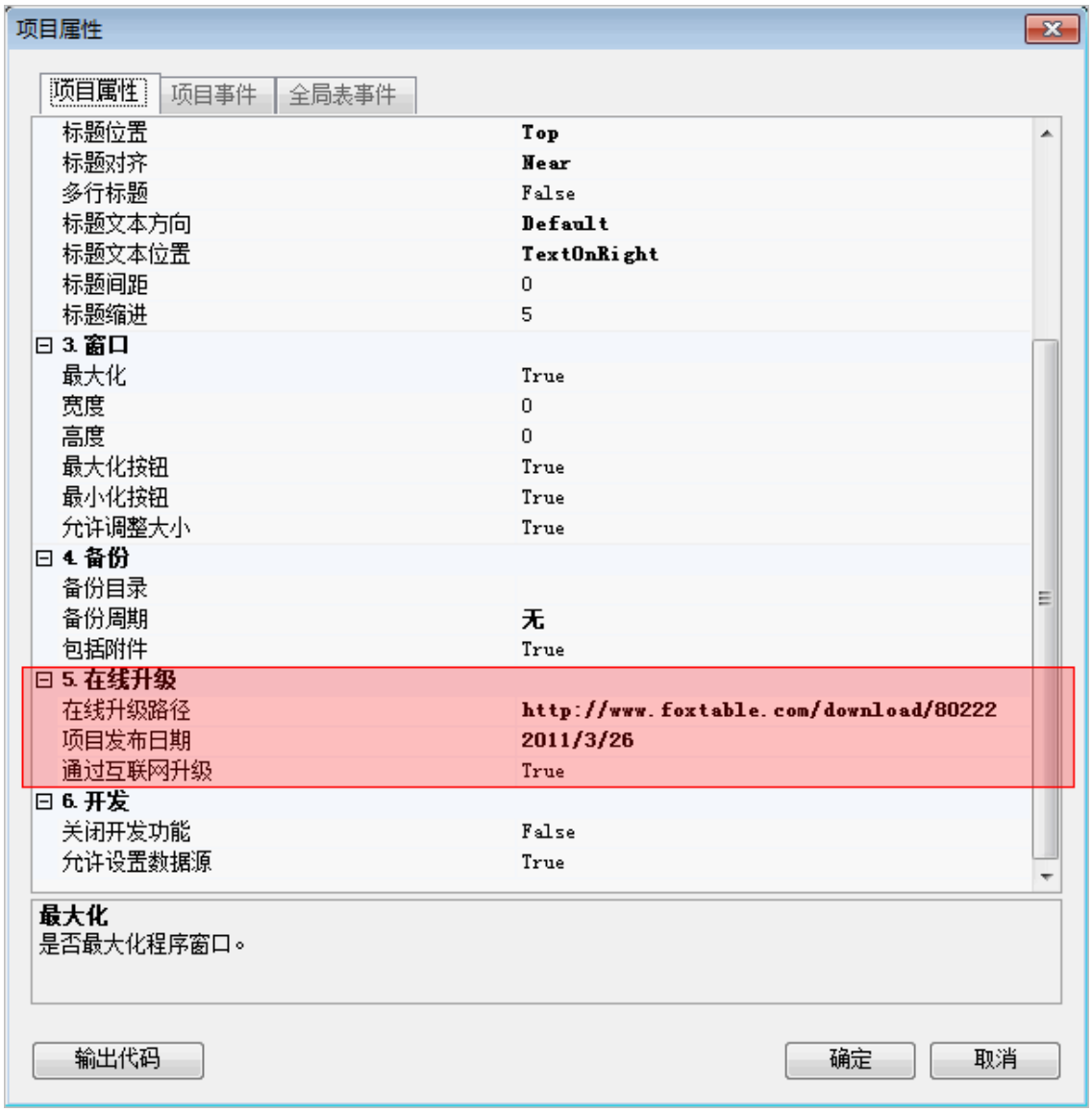
Foxtable 中有很多地方要使用条件表达式，表达式的使用有两种情形。
针对已经加载的数据
DataTable 的 Select、Find、Compute、RepalceFor、DeleteFor、RemoveFor、GetValues、GetComboListString 方法。
Table 的 Filter 属性和 FindRow 方法
DataCol 的 Expression 属性。
GroupTableBuilder 和 CrossTableBuilder 的 Filter 属性 。
这些方法或属性针对的是已经加载的数据，适用于使用指南中《表达式》这一章介绍的语法，请参考：运算符和函数 和 条件表达式。
针对后台所有数据
DataTable 的 Load 方法是从后台重新加载数据，所以 LoadFilter 属性的语法和前面的不同，它适用于 SQL 语法，和 Select 语句的 Where 条件表达式语法相同，但是具体语法和函数随不同的数据源而所有差异。
同样 AppendLoad 用于从后台追载新的数据，所以其 Filter 参数适用于 SQL 语法。
DataTable 的一系列处理后台数据的方法：SQLCompute、SQLFind、SQLSelect、SQLReplaceFor、SQLDeleteFor、SQLGetValues、SQLGetComboListString，其条件参数显然都适用于 SQL 语法。
DataTable 的 Fill 和 SQLLoad 方法，以及 Table 的 Fill 方法，都是直接使用 Select 语句从后台加载数据，当然都适用 SQL 语法。

此外 SQLGroupTableBuilder、SQLCrossTableBuilder 和 SQLJoinTableBuilder 由于统计(或查询)的是后台数据，所以其 Filter 属性同样适用于 SQL 语法。
如果将 GroupTableBuilder 和 CrossTableBuilder 的 FromServer 属性设置为 True，此时二者统计的是后台数据，所以其 Filter 属性 将适用于 SQL 语法。
需要注意的是，采用 SQL 语法的时候，表达式中的表名和列名必须和后台数据库一致。

在开发指南的《SQL 相关》这一章，有 SQL 语法的介详细绍，请参考：Select 语法 Access 函数 SQL Server 函数

4.26.5 远程自动升级的实现

本节的内容可以参考 CaseStudy 目录下的文件：远程更新.Table
我们知道，B/S 结构的软件易于部署，而 C/S 结构的软件则能提供更好的交互性。
虽然鱼和熊掌不可兼得，不过 Foxtable 却较好地融合了二者的优势。
作为 C/S 结构的软件，FoxTable 有着良好的交互性，能充分利用客户端的计算能力，提供更好的性能和功能。
同时 FoxTable 内置远程升级功能，当项目有新的版本时，你只需简单的一行代码，即可同步更新所有客户端的项目。
需要注意的是： 试用版不具备本功能，只有商业版和开发版才能实现远程自动升级。
设计步骤：
1、项目属性中有两项用于设置远程升级：



- 在线升级路径
  指定远程升级包的下载路径。

- 项目发布日期
  指定项目的发布日期，此日期将用于和远程升级包的发布日期进行比较，以判断当前项目是否需要升级。

2、制作升级包

首先需要注意的是，升级包必须是 zip 格式的压缩文件。

当开发者准备发布项目的新版本时，首先应该将项目属性中的项目发布日期设置为当天日期，然后将需要更新的文件压缩成一个 zip 文件作为升级包使用，升级包的文件名为：update.zip。

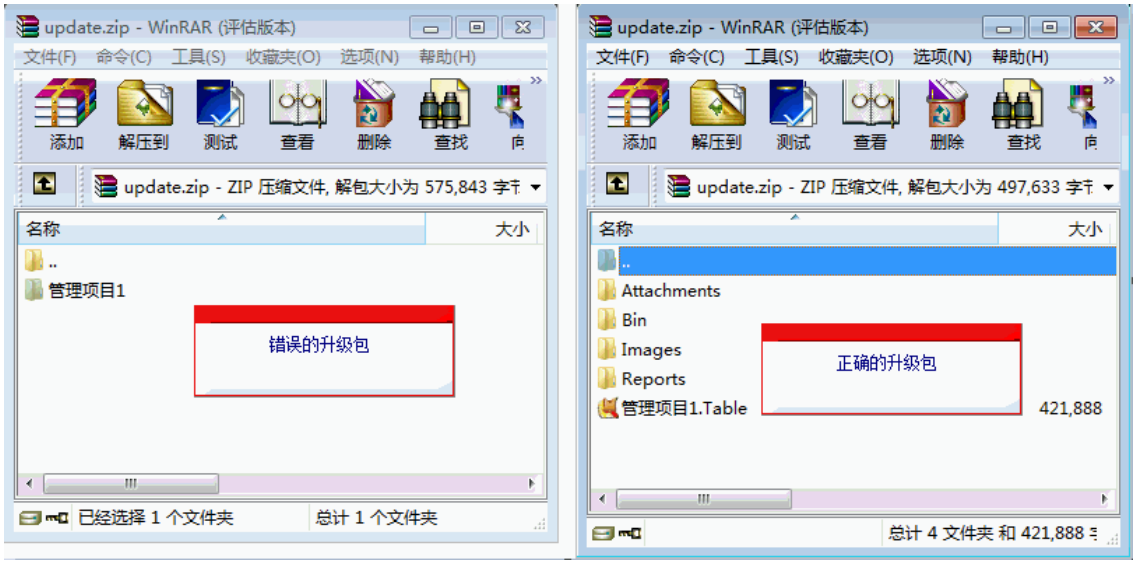
升级包中的各文件必须保持原有的相对路径，项目文件位于压缩文件的根目录，而 Images、Attachments 和 Bin 则作为子目录。

如果需要的话，可以只将更改过的文件包括在升级包中。

然后新建一个名为 update.txt 的文本文件，该文本文件的内容为压缩包的发布日期，此日期必须和项目属性中设置的项目发布日期相同，通常就是当天日期。

最后将 update.zip 和 update.txt 文件上传到项目属性中指定的在线升级路径中。
即使你没有发布新的项目，也应该在在线升级路径放置一个包含项目发布日期的 update.txt 文件，以免用户在线升级的时候出现错误提示。
如果你对于制作升级包还有疑惑，可以参考我们为示例文件“远程更新.Table”制作的远程升级包，地址为：

http://www.foxtable.com/download/80211/update.zip
http://www.foxtable.com/download/80211/update.txt
需要注意的是，经常有用户制作升级包的时候，将整个项目文件夹压缩进去，这样是不对的，会导致升级失败。
我们需要将项目文件夹下的文件及其子目录压缩到升级包中，而不是直接将整个项目文件夹压缩到升级包中，这一点要特别注意：



3、在线升级
你可以在你的项目中加上一个在线升级的菜单命令或者按钮，代码很简单：
Syscmd.Project.Update()
执行上面的代码，系统首先检索远程升级路径中的 update.txt 文件，获得升级包的发布日期，然后用这个日期和本地项目的发布日期进行比较，如果升级包的日期较新，则下载 update.zip 文件，并自动解压更新有关文件。
至此我们的在线升级功能已经完成，即使你是一个初次接触编程的人，应该也能即时掌握。
Update 的语法：
Syscmd.Project.Update(Prom1, Prom2)

Prom1: 可选参数，逻辑型，没有发现新版本时，是否提示用户。
Prom2: 可选参数，逻辑型，发现新版本时，是否提示用户升级。
例如希望自动升级能够完全"静默"，发现新版本时不提示，自动开始升级，没有发现新版本时也不提示：
Syscmd.Project.Update(False,False)
如果我们在 AfterOpenProject 下面的代码：

```
Syscmd.Project.Update(False,False)
Forms("窗口1").Open() '可能会出错
```

```
'后续其它代码
```

目的是希望打开项目的时候自动升级，然后打开窗口 1。
但是上述代码肯定会在升级成功之后提示错误，因为一旦升级成功，Foxtable 就会关闭，后面的打开窗口代码就会出错，后续的其它代码一样可能出错。
我们可以用下面的代码避免错误：

```
If Syscmd.Project.Update(False,False) = False Then '如果没有升级
    Forms("窗口1").open()
    '后续其它代码
End If
```

也就是说Update执行后会返回一个逻辑值，如果升级成功，返回True，否则返回False。

总之，如果需要在 AfterOpenPoject 事件中实现自动升级，那么原有的 AfterOpenProject 事件代码必须包括在这个判断中，只有没有升级的情况下，才执行原来的代码：

```
If Syscmd.Project.Update(False,False) = False Then '如果没有升级
    '原 AfterOpenProject 事件代码
End If
```

4、基于局域网的在线升级
在线升级默认是基于互联网的，在升级之前，会自动检测是否连接互联网。
如果要基于局域网进行在线升级，除了将升级包路径设置为局域网路径外，还必须将属性“通过互联网升级”设置为 False：



5、使用 FTP 地址升级

升级地址也可以使用 FTP 地址，例如：

ftp://zhanshang:138238@egridsoft.com/update

上面假定帐户为“zhanshang”，密码为“138238”，ftp 服务器的址为“ftp://egridsoft.com”，升级包所在目录为“update”。

6、和远程升级相关的系统变量

- PublishDate
  日期型变量，返回在项目属性中设置的项目发布日期。

- UpdatePath
  字符型，用于设置或返回在线升级路径，有了这个变量，我们可以动态设置升级路径。

- RemoteUpdate
  逻辑型，是否通过互联网升级。

4.26.6 定向数据表

同样一个系统，可能需要使用不同的数据源。

Foxtable 提供了数据表定向功能，即使系统已经设计完成，你也可以将数据表重新定向到不同的数据源。

此功能也为我们的设计带来了极大的便利，我们可以使用内部数据表进行系统开发，在开发完成后，只需单击几次鼠标，即可转换到一个 SQL Server 或者 Access 的数据源，并自动在新的数据源生成相同结构的数据表；反之，如果你之前的系统是基于 SQL Server 或者 Access 的数据源开发的，也只需单击几次鼠标即可转换为内部数据表，这样你随时可以为自己的项目生成一个“低端”版本交给他人试用；定向不仅可以在内外数据源之间进行，也可以在外部数据源之间进行，彻底摆脱因数据源的变动而需要重新调整和设计系统的烦恼。

注意事项

不是任何数据表都可以重新定向的，对于外部数据表，只有满足下述条件的表才能重定向：

1、主键列的名称为"_Identify"，且为自动增量型。

2、必须有一个名称为"_Locked"的逻辑列，用于保存行的锁定标记。

至于外部查询表，必须满足以下条件：

1、查询语句的表名必须也用大括号括起来，以便和内部数据表保持一致，例如：

```
SELECT 客户,SUM(数量) AS 数量 FROM {订单} GROUP BY 客户
```

2、必须同时定向查询表和查询表的数据来源表，例如上面查询表的数据来自于订单表，如果要重定向这个查询，那么必须将客户表重定向到相同的数据源。

数据表重定向规则：

1、如果目标数据源存在相同结构的同名表，则直接定向到该表。

2、如果存在同名表，但结构不同，则拒绝定向。

3、如果目标数据源不存在同名表，将自动创建一个相同结构的空白同名表，并定向到此表。

重定向之前必须备份你的项目，理由：

1、一旦重定向中途失败，可能导致你的项目再也无法回到正常状态。

2、内部数据表重定向到外部数据源之后，会被自动删除。

3、Foxtable 在重定向之前，也会自动备份项目，但是这不能代替你自己的备份，多一个备份，多一个保险。

重定向之后必须检查涉及到数据源的代码，例如你本来使用的内部数据表，下面的代码能够正常执行：

```
Dim q As new QueryBuilder
q.TableName = "销量统计"
q.SelectString = "Select 产品,Sum(数量) As 数量 From {订单} Group By 产品"
q.Build
MainTable = Tables("销量统计")
```

如果将订单表重新定向到一个 SQL Server 数据源，假定数据源的名称为"SaleData"，那么上面的代码必须改为：

```
Dim q As new QueryBuilder
q.ConnectionName = "SaleData"
```

```
q.TableName = "销量统计"
q.SelectString = "Select 产品,Sum(数量) As 数量 From {订单} Group By 产品"
q.Build
MainTable = Tables("销量统计")
```

如果你是从一个外部数据源定向到另一个外部数据源，也要记得修改代码中有关数据源的名称；或者你定向结束后，删除原数据源，将新数据源命名为原数据源同样的名称，这样就无须检查和修改代码了。

重定向表

如果你要将数据表重定向到一个外部数据源，你必须首先创建一个数据库，这个数据库可以是空白的，然后在你的项目中增加一个新的数据源，连接此数据库。

重定向的操作很简单，在杂项功能区，单击"Redirect"按钮即可，例如下图将内部表全部定向到一个名为 FishMIS 的外部数据源：



带数据的定向

如果目标数据源不存在同名同结构的表，那么重定向只是创建一个相同结构的同名空白表，并不会包括任何数据。

如果你需要连同数据一起重定向，那么很简单：

1、外部表定向到内部表
定向结束后，利用菜单中的高速合并功能，将原数据合并过来。
2、外部表定向到新的外部数据源
将原数据源中的相关数据表导入到新的数据源，然后再开始定向。
3、内部表定向到外部数据源
在菜单中单击下面的按钮，会生成一个 Access 文件，默认包括所有的内部数据表:



你可以直接定向到这个 Access 文件；如果要定向到一个 SQL SERVER 数据源，那么只能在定向结束后，利用高速合并功能，将此 Access 文件中的数据合并到对应的表中。
提示：
ToAccess 功能生成的 Access 文件，_Identify 列的值是重新从 1 开始排列的，所以_Identify 列的值和原始数据可能不同。
这有一个隐患：如果基于_Identify 设置了关联，那么可能会导致父子关系的丢失和紊乱。

4.26.7 动态修改表结构

ADOXBuilder 用于动态创建表和列。
ADOXBuilder 对内部数据源、以及 Access 和 SQL Server 格式的外部数据源有效，对于 Oracle 格式的外部数据源无效。
增加列
在命令窗口执行下面的代码：

```
Dim Builder As New ADOXBuilder
Builder.Open() '打开 ADOXBuilder
With Builder.Tables("表 A")
    .AddColumn("日期" ,ADOXType.DateTime) '增加日期型列
    .AddColumn("产品" ,ADOXType.String, 12) '增加字符型列,长度指定为 12
    .AddColumn("客户" ,ADOXType.String, 20) '增加字符列,长度为 20
    .AddColumn("数量" ,ADOXType.Integer) '增加整数型列
    .AddColumn("备注" ,ADOXType.Text) '增加备注列
End With
Builder.Close() '关闭 ADOXBuilder
```

执行后重新打开项目，会发现表 A 已经增加了上述列。
ADOXType 是一个枚举，用于指定列的类型，其可选值包括：
String： 字符型
DateTime：日期时间型
Boolean： 逻辑型
Text： 备注型
Integer： 整数型
Long： 长整数型，注意内部表和 Access 数据源不支持此类型

Short： 短整数型
Byte： 微整数型
Double： 双精度小数型
Single： 单精度小数型
Decimal 高精度小数型

删除列

在命令窗口执行下面的代码：

```
Dim Builder As New ADOXBuilder
Builder.Open() '打开 ADOXBuilder
With Builder.Tables("表 A")
    .DeleteColumn("日期")
    .DeleteColumn("产品")
End With
Builder.Close() '关闭 ADOXBuilder
```

执行后重新打开项目，会发现表 A 的日期和产品列已经被删除。

重命名列

在命令窗口执行下面的代码：

```
Dim Builder As New ADOXBuilder
Builder.Open()
With Builder.Tables("表A")
    .RenameColumn("第一列","姓名")
End With
Builder.Close()
```

然后重新打开项目，表A的"第一列"已经被重命名为"姓名"。

增加表

在命令窗口执行下面的代码：

```
Dim Builder As New ADOXBuilder
Dim tbl As ADOXTable
Builder.Open()
tbl = Builder.NewTable("订单") '创建表
With tbl
    .AddColumn("日期" ,ADOXType.DateTime)
    .AddColumn("产品" ,ADOXType.String, 12)
    .AddColumn("客户" ,ADOXType.String, 20)
    .AddColumn("数量" ,ADOXType.Integer)
    .AddColumn("备注" ,ADOXType.Text)
End With
Builder.AddTable(tbl) '增加表
Builder.Close()
```

执行后重新打开项目，会发现多了一个订单表。

增加表是分步的，首先用 NewTable 方法创建表，最后用 AddTable 将创建的表增加到 Builder 中。

删除表

执行下面的代码，可以删除表 A 和表 B:

```
Dim Builder As New ADOXBuilder
Builder.Open()
With Builder
    .DeleteTable("表 A")
    .DeleteTable("表 B")
End With
Builder.Close()
```

执行上述代码后，同样应该重新打开项目。

重命名表

在命令窗口执行下面的代码，可以将"表 A"重命名为"订单"：

```
Dim Builder As New ADOXBuilder
Builder.Open()
With Builder.Tables("表A")
    .Rename("订单")
End With
Builder.Close()
```

执行上述代码后，同样应该重新打开项目。

使用外部数据源

前面的代码都是针对内部数据源的，如果要为外部数据源动态创建表和列，只需在定义 ADOXBuilder 的时候，指定数据源名称即可：

```
Dim Builder As New ADOXBuilder("数据源名称")
```

例如在名为 nwnd 的外部数据源中创建一个订单表：

```
Dim Builder As New ADOXBuilder("nwnd") '要指定数据源名称
Dim tbl As ADOXTable
Builder.Open()
tbl = Builder.NewTable("订单") '创建表
With tbl
    .AddColumn("日期" ,ADOXType.DateTime)
    .AddColumn("产品" ,ADOXType.String, 12)
    .AddColumn("客户" ,ADOXType.String, 20)
    .AddColumn("数量" ,ADOXType.Integer)
    .AddColumn("备注" ,ADOXType.Text)
End With
Builder.AddTable(tbl) '增加表
Builder.Close()
```

为外部表增加系统列

使用前述代码添加的表，默认会添加主键列“_Identify”。

如果是内部数据源，默认还会添加“_Locked”和"_SortKey"两列，前者用于保存行的锁定状态，后者用于开启插入行功能。

如果是外部数据表，默认不会添加“_Locked”和"_SortKey"两列，你可以使用 AddTable 方法的另一

个语法，来决定是否为外部表增加这两列：
AddTable(Table, AddLockColumn, AddSortColumn)
Table:　　要增加的表
AddLockColumn：　逻辑型，是否增加_Locked 列
AddSortColumn：　逻辑型，是否增加_SortKey 列
例如在名为 nwnd 的外部数据源中创建一个订单表，且为订单表添加“_Locked”和"_SortKey"两列:

```
Dim Builder As New ADOXBuilder("nwnd") '要指定数据源名称
Dim tbl As ADOXTable
Builder.Open()
tbl = Builder.NewTable("订单") '创建表
With tbl
    .AddColumn("日期" ,ADOXType.DateTime)
    .AddColumn("产品" ,ADOXType.String, 12)
    .AddColumn("客户" ,ADOXType.String, 20)
    .AddColumn("数量" ,ADOXType.Integer)
    .AddColumn("备注" ,ADOXType.Text)
End With
Builder.AddTable(tbl, True, True) '增加表
Builder.Close()
```

重新打开项目的代码
前面已经提到，动态创建的表和列之后，都必须重新打开项目。
可以使用下面的代码重新打开当前项目：

```
Syscmd.Project.Open(ProjectFile)
```

4.26.8 创建操作日志

OutPut 类有一个共享属性 Logs，返回一个 Log 集合，可以用来创建操作日志。
和我们之前接触的集合不同，通过 Logs 可以访问指定名称的 Log，例如：

```
Output.Logs("登录日志").Add(User.Name & "于" & Date.Now & "进入")
```

你不需要显示地创建一个 Log，然后添加到 Logs 中，首次执行上面的代码，系统会自动创建一个名为“登录日志”的 Log。

方法

Add

增加日志信息，语法：

Add(Messge)

Message：字符型,要增加的日志信息

增加的日志信息不会显示，但是可以用 Save 方法保存到一个文件中，这样我们就可以创建自己的操作日志了。
例如我们要在名为 ABC 的 Log 中增加今天的日期：

```
Output.Logs("ABC").Add(Date.Today)
```

Clear

清除指定 log 中现有日志信息，例如要清除名称为 ABC 的 Log 的日志信息：

```
Output.Logs("ABC").Clear()
```

如果要清除所有 Log 的信息，直接调用 Logs 集合的 Clear 方法即可：

```
Output.Logs.Clear()
```

Save

将调试信息保存到一个文本文件中，语法：

```
Save(FieName, Append)
```

FileName：字符型，目标文件名，要包括路径。
Append：　逻辑型，如果目标文件已经存在，此属性决定是将调试信息会加到源文件，还是直接覆盖原文件。

保存某一个 Log 后，会自动清除该 Log 的日志信息，这样可以避免信息的重复保存。

示例一，创建删除数据的日志

例如我们要在日志文件中记录什么人在什么时候在订单表中删除了一条记录，被删除订单的编号是什么，可以在订单表的DataRowDeleting事件中设置如下代码：

```
Output.Logs("删除日志").Add(User.Name & "在" & Date.Now & "删除了" & e.DataRow("订单编号"))
```

然后打开项目属性窗口，将 BeforeCloseProject 事件的代码设置如下：

```
Output.Logs("删除日志").Save("C:\Delete.txt",True)
```

这样就会在 C 盘根目录下创建一个 Delete.txt 文件，记录订单删除操作的日志。

日志内容是不会自动清除的，即使打开一个新的项目，上一个项目的日志内容还是会保存，所以我们需要在项目的 AfterOpenProject 设置如下代码，用于清除上一个项目的日志内容：

```
Output.Logs.Clear()
```

示例二，创建一个登录日志

在项目的 AfterOpenProject 事件中设置如下代码：

```
Output.Logs("登录日志").Add(User.Name & "： " & Date.Now )
Output.Logs("登录日志").Save("c:\log.txt",True)
Output.Logs("登录日志").Clear
```

简单的三行代码，即可记录什么人在什么时候进入了本系统。

4.26.9 创建系统列

如果某一列用字符“sys_”开头，例如“sys_标记”，则该列将会被自动隐藏，用户也无法通过取消隐藏 来重新显示该列。
需要注意的是，这种隐藏是针对非开发者的，也就是说对于开发者来说，以字符“sys_”开头的系统列，和普通的列无异。

4.26.10 农历日期

Foxtable 提供了一个 Lunar 类型，专门用于农历和公历之间的转换。
Lunar 的属性包括：

属性	说明
Year	返回农历年
Month	返回农历月，由于可能存在闰月，所以值在 1 到 13 之间，例如某年闰 7 月，那么该年闰 7 月的 Month 属性为 8，8 月的 Month 属性为 9
Day	返回农历日
LeapMonth	如果给定的日期在闰年，则返回第几个月是闰月，例如 2007 年是闰 7 月，等于是第 8 个月是闰月，所以 LeapMonth 属性为 8
IsLeapMonth	逻辑型，给定的日期如果在闰月，则返回 True
IsLeapYear	逻辑型，给定的日期如果在闰年，则返回 True
SolarDate	日期型，返回公历日期
LunarDate	字符型，返回农历日期的字符表示

定义一个 Lunar 的语法有两种。
语法一
New Lunar(Date)
Date: 公历日期
示例

```
'输出 2002 年 3 月 17 日对应的农历日期
Dim ln As New Lunar(#3/17/2002#)
output.show(ln.LunarDate)
'输出 2006 年 8 月 25 日对应的农历日期
ln = New Lunar(#8/25/2006#)
Output.show(ln.LunarDate)
If ln.IsLeapMonth Then
    Output.Show("这是闰月!")
End If
```

语法二

New Lunar(Year, Month, Day, Leap)

Year： 农历年
Month: 农历月
Day: 农历日
Leap: 逻辑型，是否闰月

示例

```
'获得农历 2008 年 8 月 15 日(中秋节)对应的阳历日期
Dim ln As New Lunar(2008,8,15,False)
output.show(ln.SolarDate)
'获得 2006 年闰 7 月 15 日对应的阳历日期
ln  = New Lunar(2006,7,15,true)
output.show(ln.SolarDate)
```

4.26.11 系统信息

SysInfo 类型通过其共享属性，提供一些常用的系统信息。

属性名	说明
ComputerName	计算机名称
OSVersion	操作系统版本号
OSFullName	操作系统名称
UserName	登陆操作系统的用户名
AvailablePhysicalMemory	可用的物理内存
TotalPhysicalMemory	总的物理内存
ScreenWidth	屏幕宽度
ScreenHeight	屏幕高度
WorkingAreaWidth	屏幕工作区宽度，排除任务栏、停靠窗口、

	停靠工具栏所占据的区域。
WorkingAreaHeight	屏幕工作区高度，排除任务栏、停靠窗口、停靠工具栏所占据的区域。

例如下面的代码，用于显示屏幕分辨率以及工作区大小：

```
Output.Show("显示器分辨率为：" & SysInfo.ScreenWidth & "*" & SysInfo.ScreenHeight)
Output.Show("可用工作区大小：" & SysInfo.WorkingAreaWidth & "*" & SysInfo.WorkingAreaHeight)
```

4.26.12 表达式求值

普通用户请忽略本节内容。

Foxtable 提供了一个 Eval 函数，用于自定义表达式求值。

需要注意的是，这个表达式是一个独立的系统，和我们之前用于列计算和筛选的表达式完全不同，这里的表达式采用 VBScript 语法。

语法

Eval(Expr)

Eval(Expr,Row)

Eval(Expr,DataRow)

Expr: 要求值的表达式

Row: 用于取值的 Row

DataRow: 用于取值的 DataRow

注意表达式中的列名称必须用方括号括起来，例如：[单价]

表达式中可以引用 Var 变量，除了变量名称需要用方括号括起来外，还需要在前面加上符号“！”，例如：[!起始日期]

除 Eval 函数外，Foxtable 还提供了两个和表达式求值相关的函数：

Eval2

Eval2 同样用于对表达式求值，语法和 Eval 完全一样，差别在计算表达式的时候遇到错误，Eval 弹出一个窗口报错，而 Eval2 则不会。

TryEval

TryEval 的语法和 Eval 完全一样，但此函数不是对表达式求值，而是用于检测表达式，如果此表达式正确则返回 True，否则返回 False。

下面的例子，都没有实际的意义，是为了说明 Eval 的用法。

示例一

```
Dim s As String = "1 + 2 + 3"
Output.Show(Eval(s))
```

输出结果为：6

示例二

基于订单表的第二行，计算表达式“[数量] * [单价]”：

```
Dim s As String = "[数量] * [单价]"
Dim r As Row = Tables("订单").Rows(1)
Output.Show(Eval(s,r))
```

示例三

从订单表中选出数量超过 500 的行，将金额设为“[数量] * [单价] * 0.8”。
注意运行下面的代码之前，必须保证金额列是一个数据列，而不是一个表达式列。

```
Dim exp As String = "[数量] * [单价] * 0.8"
Dim drs As List(of DataRow) = DataTables("订单").Select("数量 > 500")
For Each dr As DataRow In drs
    dr("金额") = Eval(exp,dr)
Next
```

示例四

假定希望根据第一列输入的计算公式，在第二列得出计算结果，可以将 DataColChanged 事件代码设为：

```
If e.DataCol.Name = "第一列" Then
    If e.DataRow.IsNull("第一列") Then
        e.DataRow("第二列") = Nothing
    Else
        e.DataRow("第二列") = Eval(e.DataRow("第一列"),e.DataRow)
    End If
End If
```

Eval 采用 VBScript 语法，没有方法和属性的概念，所有的计算都是通过函数完成的。
请参考：
VBScript 参考

4.26.13 模拟按键动作

普通用户可以忽略本节内容。
SendKeys 类用于模拟按键动作。
该类型有两个共享方法：

Send
SendWait

使用方式一样，差别在于 SendWait 在送出按 键后，必须等按键被系统处理完成才继续下一步的操作。

语法

Send(Keys)
SendWait(Keys)

参数 Keys 为要发送的击键字符串。

每个键都由一个或多个字符表示。若要指定单个键盘字符，请使用该字符本身。例如，若要表示字母 A，请将字符串“A”传递给方法。若要表示多个字符，例如表示字母 A、B 和 C，请将参数指定为“ABC”。

例如：

```
Dim Proc As New Process
Proc.File = "C:\table.txt"
Proc.Start
Proc.WaitForInputIdle() '等程序自动结束并进入空闲状态
Sendkeys.Send("I Like Foxtable") '发送按键信息
```

加号 (+)、插入符号 (^)、百分号 (%)、波浪号 (~) 以及圆括号 ( ) 对 SendKeys 具有特殊含义。若要指定这些字符中的某个字符，请将其放在大括号 ({}) 内。例如，若要指定加号，请使用“{+}”。若要指定大括号字符，请使用“{{}”和“{}}”。中括号 ([ ]) 对 SendKeys 没有特殊含义，但必须将它们放在大括号内。

若要指定在按键（如 Enter 或 Tab）时不显示的字符，以及表示操作而不表示字符的键，请使用下表中的代码：

键	代码
Backspace	{BACKSPACE}、{BS} 或 {BKSP}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Del 或 Delete	{DELETE} 或 {DEL}
向下键	{DOWN}
End	{END}
Enter	{ENTER} 或 ~
Esc	{ESC}
Help	{HELP}
Home	{HOME}
Ins 或 Insert	{INSERT} 或 {INS}
向左键	{LEFT}

Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Print Screen	{PRTSC}（保留供将来使用）
向右键	{RIGHT}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}
向上键	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}
F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
F13	{F13}
F14	{F14}
F15	{F15}
F16	{F16}
数字键盘加号	{ADD}
数字键盘减号	{SUBTRACT}
数字键盘乘号	{MULTIPLY}
数字键盘除号	{DIVIDE}

若要指定与 Shift、Ctrl 和 Alt 键的任意组合一起使用的键，请在这些键代码之前加上以下一段或多段代码。

键	代码
Shift	+
Ctrl	^
Alt	%

若要指定在按一些键时应同时按住 Shift、Ctrl 和 Alt 键的任意组合，请将这些键的代码放在括号内。例如，若要指定按 E 和 C 时同时按住 Shift 键，请使用“+(EC)”。若要指定按 E 时应按住 Shift，之后按 C 时不按住 Shift，请使用“+EC”。

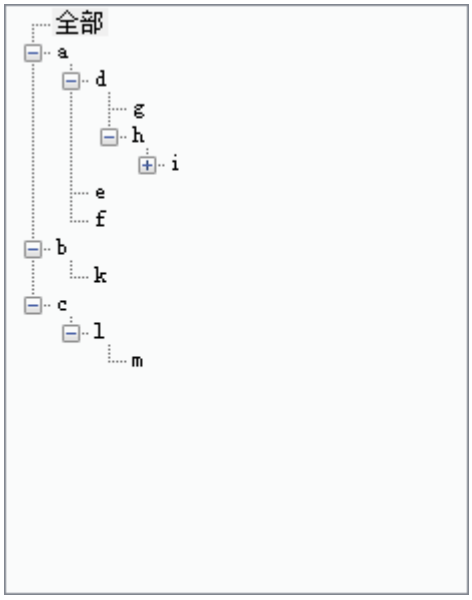
若要指定重复键，请使用格式 {key number}，Number 是一个数字，表示按键的次数，键和数字之间必须放一空格。例如，{LEFT 42} 表示按向左键 42 次；{h 10} 表示按 H 10 次。

4.26.14 递归函数使用实例

一般用户可以忽略本节内容。
系统学习过编程的人，对于递归函数都很熟悉，所谓递归函数，就是函数的自我调用。
Foxtable 内部函数也支持递归调用。
如果代码设计不当，出现无限递归，会导致堆栈溢出，使得 Foxtbale 直接崩溃退出。
本节的内容可以参考 CaseStudy 目录下的文件：递归函数.Table
假定有一个表：

	父键	键	标题
1		1	a
2		2	b
3		3	c
4	1	4	d
5	1	5	e
6	1	6	f
7	4	7	g
8	4	8	h
9	8	9	i
10	9	10	j
11	2	11	k
12	3	12	l
13	12	13	m

要生成下图所示的目录树，各节点的关系由“父键”和“键”两列的值决定：



而且要求双击某个节点，能将此节点及其子节点对应的行全部筛选出来。

设计步骤：

1、在内部函数中新建一个函数，名为 AddChildren，代码为：

```
Dim nd As WinForm.TreeNode = args(0)
Dim dt As DataTable = args(1)
Dim drs As List(of DataRow)
drs = dt.Select("[父键] = '" & nd.name & "'")
For Each dr As DataRow In drs
    Dim cd As Winform.TreeNode = nd.Nodes.Add(dr("键"),dr("标题"))
    Functions.Execute("AddChildren",cd,dt)
Next
```

上述代码中，出现了调用 AddChildren 函数自身的代码(第 8 行)：

```
Functions.Execute("AddChildren",cd,dt)
```

这种在函数自己调用自己的情况，就是常说的递归调用。

2、在窗口中加入一个目录树控件和一个按钮，按钮的代码为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim nd As WinForm.TreeNode
Dim dt As DataTable = DataTables("表A")
tr.StopRedraw()
tr.Nodes.Clear
tr.Nodes.Add("全部")
For Each dr As DataRow In dt.datarows
   If dr.IsNull("父键") Then
      nd = tr.Nodes.Add(dr("键"),dr("标题"))
      Functions.Execute("AddChildren",nd,dt)
   End If
Next
tr.ResumeRedraw()
```

单击此按钮，即可根据“父键”和“键”两列的值，生成目录树。
3、最后将目录树的 NodeMouseDoubleClick 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim key As String = e.Node.Name
Dim ids As String
If key = "全部" Then
    Tables("表A").Filter = ""
Else
    For Each nd As WinForm.TreeNode In trv.AllNodes
        Dim pd As WinForm.TreeNode = nd
        Do
            If pd.Name = key Then
                If ids > "" Then
                    ids= ids &   ","
                End If
                ids = ids & "'" & nd.name & "'"
                Exit Do
            Else
                pd = pd.ParentNode
                If pd Is Nothing Then
                    Exit Do
                End If
            End If
        Loop
    Next
    Tables("表A").Filter = "键  In (" & ids & ")"
End If
```

这样要求双击某个节点，即可将此节点及其子节点对应的行全部筛选出来。

4.26.15 用递归函数生成会计科目目录树

一般用户可以忽略本节内容。

本节内容可以参考 CaseStudy 目录下的文件：会计科目.Table
假定有下图所示会计科目表：

	科目代码	科目名称
1	1001	库存现金
2	1002	银行存款
3	100201	招商银行华强北支行
4	100202	深圳发展银行振华支行
5	1004	备用金
6	1012	其他货币资金
7	101201	银行汇票
8	101202	银行本票
9	101203	信用卡
10	101204	信用证保证金
11	101205	外埠存款

科目代码为四位数的是一级科目，六位数的是二级科目，八位数的是三级科目，要求根据科目代码生成下图所示的目录树:



设计步骤：

1、在内部函数中新建一个函数，名为 AddChildren，代码为：

```
Dim nd As WinForm.TreeNode = args(0)
Dim dt As DataTable = args(1)
For Each dr As DataRow In dt.Select("","科目代码")
    Dim km As String = dr("科目代码")
    If km.StartsWith(nd.name) AndAlso km.Length = nd.Name.Length + 2 Then
        Dim cd As Winform.TreeNode = nd.Nodes.Add(km,km & " " & dr("科目名称"))
        Functions.Execute("AddChildren",cd,dt)
    End If
Next
```

上述代码中，出现了调用 AddChildren 函数自身的代码(第 8 行)：

Functions.Execute("AddChildren",cd,dt)

这种在函数自己调用自己的情况，就是常说的递归调用。

2、新建一窗口，在窗口中加入一个目录树控件。
3、将窗口的 AfterLoad 事件代码设置为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim nd As WinForm.TreeNode
Dim dt As DataTable = DataTables("会计科目")
tr.StopRedraw()
tr.Nodes.Clear
tr.Nodes.Add("全部")
For Each dr As DataRow In dt.Select("","科目代码")
    If dr.IsNull("科目代码") = False AndAlso dr("科目代码").Length = 4 Then
        nd = tr.Nodes.Add(dr("科目代码"),dr("科目代码") & " " & dr("科目名称"))
        Functions.Execute("AddChildren",nd,dt)
    End If
Next
tr.ResumeRedraw()
```

这样打开此窗口，即可根据科目代码自动生成目录树。
4、最后将目录树的 NodeMouseDoubleClick 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
If e.Node.Name = "全部" Then
    Tables("会计科目").Filter = ""
Else
    Tables("会计科目").Filter = "科目代码  Like '" & e.Node.Name & "%'"
End If
```

这样要求双击某个节点，即可将此节点及其子节点对应的行全部筛选出来。

再进一步

下面的内容，可以参考项目文件“会计科目.Table”的“窗口 2”。
同样以上面的表为例，科目代码首字符 1 表示资产，2 表示负债,3 表示权益,4 表示成本,5 表示损益。
要求生成下图所示的目录树：



设计步骤：

1、在内部函数中新建一个函数，名为 AddChildren，代码为：

```
Dim nd As WinForm.TreeNode = args(0)
Dim dt As DataTable = args(1)
For Each dr As DataRow In dt.Select("","科目代码")
    Dim km As String = dr("科目代码")
    If km.StartsWith(nd.name) AndAlso km.Length = nd.Name.Length + 2 Then
        Dim cd As Winform.TreeNode = nd.Nodes.Add(km,km & " " & dr("科目名称"))
        Functions.Execute("AddChildren",cd,dt)
    End If
Next
```

2、新建一窗口，在窗口中加入一个目录树控件。

3、将窗口的 AfterLoad 事件代码设置为：

```
Dim tr As WinForm.TreeView = e.Form.Controls("TreeView1")
Dim rts() As String = {"全部","资产","负债","权益","成本","损益"}
Dim nd As WinForm.TreeNode
Dim dt As DataTable = DataTables("会计科目")
tr.StopRedraw()
tr.Nodes.Clear
For Each rt As String In rts
    tr.Nodes.Add(rt)
```

```
Next
For Each dr As DataRow In dt.Select("","科目代码")
    If dr.IsNull("科目代码") = False AndAlso dr("科目代码").Length = 4 Then
        Dim i As Integer = CInt(dr("科目代码").Substring(0,1))
        nd = tr.Nodes(i).Nodes.Add(dr("科目代码"),dr("科目代码") & " " & dr("科目名称"))
        Functions.Execute("AddChildren",nd,dt)
    End If
Next
tr.ResumeRedraw()
```

需要特别注意一点，定义数组的时候，顺序不能错乱：

```
Dim rts() As String = {"全部","资产","负债","权益","成本","损益"}
```

这样才能保证目录树编号为 1 的根节点是资产，编号为 2 的根节点是负债...，以简化后续代码的设计。

4、最后将目录树的 NodeMouseDoubleClick 事件代码设置为：

```
Dim trv As WinForm.TreeView = e.Form.Controls("TreeView1")
If e.Node.Level = 0
    If   e.Node.Name = "全部" Then
        Tables("会计科目").Filter = ""
    Else
        Tables("会计科目").Filter =  "科目代码  Like '" & e.Node.Index & "%'"
    End If
Else
    Tables("会计科目").Filter = "科目代码  Like '" & e.Node.Name & "%'"
End If
```

4.26.16 文件监视器

文件监视器用于监控指定目录下的文件，当创建、修改或删除文件时，会触发事件，通过设置对应的事件代码，可以间接和第三方的程序实现及时的数据交流。

在管理项目功能区，单击下面的按钮，可以设置文件监视器的属性和代码：



FileWatcher 表示文件监视器，其属性有：

- Enabled
  逻辑型，是否启用文件监视器。

- Path
  字符型，设置监视目录的路径，不能将监控目录设置为根目录。

- Filter
  字符型，设置要监控的文件类型，例如要监控文本文件，可以设置为"*.txt"；要监控指定的某个文件，可以直接设置为文件名，例如"data.mdb"；如果要监控所有文件，则设置为"*.*"

- Delay
  设置事件延迟触发时间，默认为 500 毫秒，也就是文件发生变动后 0.5 秒才触发对应的事件。
  在延迟触发时间内，如果同一个文件发生了多次变动，将只触发一次事件。
  最小可设置为 100 毫秒。
  保存一个大文件，可能要经过多次的写入，持续的时间也会比较长，所以合理设置此属性是相当重要的。

示例

```
With FileWatcher
    .Path = "c:\Data"
    .Filter = "Sale.mdb"
    .Delay = 1000
    .Enabled = True
End With
```

执行上面的代码后，将开始监控文件“c:\data\Sale.mdb”。
其实一般是通过菜单来设置 FileWatcher 的，很少需要利用代码来设置。
FileWatcher 有三个事件：

- FileCreated
  创建文件后执行。

- FileDeleted
  删除文件后执行。

- FileChanged
  修改文件后执行。

e 参数属性：
Name:　触发事件的文件名，不含路径。
FullPath：触发事件的文件名，含路径。

示例一

假如订单表是外部表，来自文件 c:\data\Sale.mdb，我们可以监控此文件，一旦此文件发生变动，就重新加载订单表数据。
首先我们要监控此文件，在 AfterOpenProject 事件中设置代码：

```
With FileWatcher
    .Path = "c:\Data"
```

```
    .Filter = "Sale.mdb"
    .Delay = 1000
    .Enabled = True
End With
```

然后将 FileChanged 事件代码设置为：

```
DataTables("表 A").Load()
```

看上去很简单，不过还有一个毛病，就是自己保存数据后，也会触发 FileChanged，导致重新加载数据。为避免这种情况，我们可以自己做一个保存按钮：

```
FileWatcher.Enabled = False
DataTables.Save()
FileWatcher.Enabled = True
```

上面的代码在保存前停用文件监视器，保存结束后，重新开启文件监视器。

示例二

利用文件监视器，还可以和第三方软件协同工作。

例如有个人事管理系统，希望第三方的拍照软件给员工拍照，希望拍好的照片以员工的工号作为文件名，并自动存入员工表的照片列。

假定拍照软件存放照片的目录是"c:\photo"，我们首先要监控此目录，在 AfterOpenProject 事件设置代码：

```
With FileWatcher
    .Path = "c:\Photo"
    .Delay = 1000
    .Enabled = True
End With
```

然后将 FileCreated 事件代码设置为：

```
Dim r As Row = Tables("员工").Current
r("照片") = r("工号") & ".jpg"
FileSys.CopyFile(e.FullPath, ProjectPath & "Attachments\" & r("照片"),True)
```

这里假定照片是 jpg 格式。

最后在菜单或者窗口中加一个按钮，用于启动第三方拍照软件：

```
Dim Proc As New Process
Proc.File = "拍照程序的路径和名称"
Proc.Start()
Proc.WaitForExit() '等带拍照程序退出。
```

4.26.17 使用剪贴板

名称	说明
SetText	将字符串添加到剪贴板中。 示例 Dim s As String = "我喜欢 Foxtable" ClipBoard.SetText(s)

GetText	获得剪贴板中的字符串。 示例 Dim s As String = ClipBoard.GetText() Output.Show(s)

ContainsText	判断剪贴板中是否有字符。
	示例
	If ClipBoard.ContainsText Then
	Dim s As String = ClipBoard.GetText()
	Output.Show(s)
	End If

SetImage	将图片添加到剪贴板中。 示例 Dim img As Image = GetImage("c:\logo.jpg") ClipBoard.SetImage(img)
GetImage	获得剪贴板中的图片。 Dim img As Image img = ClipBoard.GetImage

ContainsImage	判断剪贴板中是否有图片。
	
	If ClipBoard.ContainsImage Then
	Dim img As Image
	img = ClipBoard.GetImage()
	End If

Clear	清除剪贴板中的内容。

4.26.18 压缩和解压文件

ZipFile 类型用于压缩和解压文件，采用的压缩格式为 ZIP。

方法

名称	说明
Create	创建压缩文件。 示例 Dim zip As New zipFile zip.Create("c:\data\foxtable.zip")
Open	打开压缩文件。 示例 Dim zip As New zipFile zip.Open("c:\data\foxtable.zip")
Close	关闭压缩文件，ZipFile 使用结束后，应该执行此方法。

AddFile	添加文件。
	语法：
	AddFile(FileName)
	AddFile(FileName, NewFileName)
	FileName:　　要压缩的文件名，含路径。
	NewFileName:　　可选参数，设置压缩后的文件名，可以包括路径。
	示例一：
	
	Dim zip As New zipFile
	zip.Create("c:\data\foxtable.zip")
	zip.AddFile("c:\publish\foxtable.exe")
	zip.Close()
	示例二:
	
	备份项目文件，并按照“月.日”的格式重命名压缩后的文件名，这样我们可以同一个压缩文件中保留项目的多个备份:
	Dim zip As New zipFile
	Dim zFile As String = "c:\data\backup.zip"
	Dim NewName As String
	NewName = Date.Today.Month & "." & Date.Today.Day & ".Table"
	If FileSys.FileExists(zfile) Then '如果压缩文件已经存在
	　zip.Open(zFile) '则打开
	Else
	　zip.Create(zFile) '否则创建
	End If
	zip.AddFile(ProjectFile, NewName)
	zip.Close()
	示例三：

	可以给要压缩的文件指定路径，例如备份项目文件以及项目文件子目录 Images 下的所有文件: Dim zip As New zipFile zip.Create("c:\data\backup.zip") zip.AddFile(ProjectFile) For Each File As String In FileSys.GetFiles(ProjectPath & "Images") zip.AddFile(File, "Images\" & FileSys.GetName(File)) Next zip.Close() 上面的代码，压缩时如果不给 Images 子目录下的文件指定路径“Images”，那么这些文件将和项目文件一样，处于压缩文件的根目录，导致解压的时候无法准确还原路径。
AddFolder	添加目录，此目录下的所有文件将被压缩。 语法： AddFolder(Path) AddFolder(Path, Pattern) AddFolder(Path, Pattern, IncludeSubFolders) Path: 要压缩的目录 Patter: 可选参数，要匹配的模式 IncludeSubFolders: 可选参数，逻辑型，是否包括子目录，默认为 True，也就是包括。 示例一： 压缩 c:\publish 目录下的所有文件： Dim zip As New zipFile zip.Create("c:\data\backup.zip") zip.AddFolder("c:\publish") zip.Close() 示例二： 压缩 c:\publish 目录下的 gif 文件： Dim zip As New zipFile zip.Create("c:\data\backup.zip") zip.AddFolder("c:\publish", "*.gif") zip.Close() 示例三： 压缩 c:\publish 目录下的所有文件，但不包括子目录下的文件： Dim zip As New zipFile zip.Create("c:\data\backup.zip") zip.AddFolder("c:\publish", "*.*", False) zip.Close()
Extract	解压指定的文件。

	语法： Extract(FileName) Extract(FileName, NewFileName) FileName:　要解压的文件 NewFileName：　可选参数，指定新的文件名，必须包括路径 如果不指定 NewFileName 参数，则解压到压缩文件所在目录。 示例： Dim zip As New zipFile zip.Open("c:\data\backup.zip") zip.Extract("Foxtable.exe") zip.Extract("Images/Chart.ico") zip.Extract("Images/Bold.ico", "c:\data\Images\Bold.ico") zip.Close() 上面代码，将 foxtable.exe 和 chart.ico 两个文件解压到压缩文件所在的目录 c:\data，尽管在压缩文件中,Chart.Ico 位于子目录 Images 之下，而 Bold.ico 将解压到 c:\data\Images 目录下。 提示： 1、压缩文件内部的目录分割符号为"/"，而不是"\"。 2、如果要解压到指定的目录，必须先创建此目录，例如上例中的 c:\data\Images。
ExtractAll	解压所有文件到指定的目录。 示例: 将所有文件解压到目录 c:\test: Dim zip As New zipFile zip.Open("c:\data\backup.zip") zip.ExtractAll("c:\test") zip.Close()

Contains	判断压缩文件中是否存在指定的文件。
	示例：
	Dim zip As New zipFile
	Dim zFile As String = "c:\data\backup.zip"
	Dim NewName As String
	NewName = Date.Today.Month & "." & Date.Today.Day & ".Table"
	If FileSys.FileExists(zfile) Then
	　zip.Open(zFile)
	Else
	　zip.Create(zFile)
	End If
	If zip.Contains(NewName) = False Then
	　zip.AddFile(ProjectFile, NewName)

	End If zip.Close()
Remove	从压缩文件中删除指定的文件。 示例： Dim zip As New zipFile zip.Open("c:\data\backup.zip") zip.Remove("foxtable.exe") zip.Close()

属性

名称	说明
Count	整数型，返回压缩文件中文件的个数。
UseUtf8Encoding	逻辑型，压缩包中的文件名和注释是否采用 UTF8 格式编码。
Files	返回一个字符集合，包括压缩文件中所有文件名。 示例 Dim zip As New zipFile zip.Open("c:\data\backup.zip") For Each File As String In zip.Files Output.Show(File) Next zip.Close()
Password	字符型，返回或设置压缩文件的目录。 示例一 压缩的时候设置密码： Dim zip As New zipFile zip.Create("c:\data\backup.zip") zip.Password = "mypassword" zip.AddFolder("c:\publish") zip.Close() 解压的时候设置密码： Dim zip As New zipFile zip.Open("c:\data\backup.zip") zip.Password = "mypassword" zip.ExtractAll("c:\data\backup") zip.Close() 实例二 不同的文件可以设置不同的密码： Dim zip As New zipFile zip.Create("c:\data\backup.zip") zip.Password = "mypassword" zip.AddFile("c:\publish\foxtable.exe") zip.Password = "yourpassword"

```
zip.AddFile("c:\publish\config.dat")
zip.Close()
解压的时候必须为各文件设置对应的密码：
Dim zip As New zipFile
zip.Open("c:\data\backup.zip")
zip.Password = "mypassword"
zip.Extract("foxtable.exe")
zip.Password = "yourpassword"
zip.Extract("config.dat")
zip.Close()
```

4.26.19 判断文件是否已经打开

FileSys 虽然提供了很多处理文件和目录的函数，但是欠缺一个判断文件是否已经打开的函数，而在开发管理软件的时候，这样的判断需求偶尔会遇到的。

为此 Foxtable 在 FileSys 之外提供了一个 FileIsOpened 函数，用于判断指定的文件是否已经打开。

示例：

```
If FileIsOpened("c:\data\employee.xls")= True Then
    MessageBox.Show("文件已经打开")
End If
```

4.26.20 垂直表和水平表的转换

在一些场合，可能需要对表的结构进行横竖转换。

示例一

假定表 A 的结构如下：

	姓名	课程	分数
1	张三	语文	74
2	张三	数学	83
3	张三	物理	93
4	李四	语文	74
5	李四	数学	84
6	李四	物理	94

希望将其转换为表 B，结构为：

	姓名	数学	物理	语文
1	李四	84	94	74
2	张三	83	93	74

转换代码为：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("姓名", Gettype(String), 32)
For Each v As String In DataTables("表A").GetValues("课程")
```

```
    dtb.AddDef(v, Gettype(Double))
Next
dtb.Build()
For Each v As String In DataTables("表A").GetValues("姓名")
    Dim dr1 As DataRow = DataTables("表B").AddNew()
    dr1("姓名") = v
    For Each dr2 As DataRow In DataTables("表A").Select("姓名 = '" & v & "'")
        dr1(dr2("课程")) = dr2("分数")
    Next
Next
MainTable = Tables("表B")
```

我们也可以 SQL 语句来实现，更加简单：

Select 姓名,Sum(iif(课程 = '语文',分数,0)) As 语文,Sum(iif(课程 = '数学',分数,0)) As 数学,Sum(iif(课程 = '英语',分数,0)) As 英语,Sum(iif(课程 = '物理',分数,0)) As 物理,Sum(iif(课程 = '化学',分数,0)) As 化学 From {学生成绩} Group by 姓名

上述 SQL 语句巧妙地利用了分组统计，希望大家细细体会一下其中的奥妙。

需要注意的是，如果使用的数据源是 SQL Server，是不能使用 iif 函数的，需要用 Case When 语句：

Select 姓名,Sum(Case When 课程 = '语文' Then 分数 Else 0 End) As 语文,Sum(Case When 课程 = '数学' Then 分数 Else 0 End) As 数学,Sum(Case When 课程 = '英语' Then 分数 Else 0 End) As 英语,Sum(Case When 课程 = '物理' Then 分数 Else 0 End) As 物理,Sum(Case When 课程 = '化学' Then 分数 Else 0 End) As 化学 From {学生成绩} Group by 姓名

我们可以新建一个查询表，或者用 QueryBuilder，将其 Select 语句设置为 上述语句即可。

示例二

接下来更复杂点，假定有下图所示的学生成绩表，在上面的基础上多了一个班级列：

	班级	姓名	科目	分数
1	1班	陈成进	语文	133
2	1班	林景阳	语文	129
3	1班	苏　捷	语文	132
4	1班	谢晓霞	语文	129
5	2班	谢杰立	语文	129
6	2班	谢晓霞	语文	134
7	2班	杨景亮	语文	132
8	2班	张李华	语文	133
9	4班	谢杰立	语文	129
10	4班	谢晓霞	语文	134
11	4班	杨景亮	语文	132
12	4班	张李华	语文	133
13	1班	陈成进	数学	136
14	1班	苏　捷	数学	136
15	1班	谢杰立	数学	133
16	2班	林景阳	数学	133
17	2班	杨景亮	数学	136
18	4班	林景阳	数学	133

需要将其转换为水平结构的表，而且在原有科目的基础上，再增加一个总分列：

	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136	140	71	75	555
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529
11	1班	张李华	124	118	124	83	88	537
12	2班	陈成进	110	58	106	36	34	344
13	2班	陈诗淇	113	123	117	78	78	509
14	2班	陈思达	113	55	92	55	46	361

可以参考下面的代码：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("班级",Gettype(String),10)
dtb.AddDef("姓名",Gettype(String),10)
For Each v As String In DataTables("学生成绩").GetValues("科目")
    dtb.AddDef(v, Gettype(Double))
Next
dtb.Build()
For Each v As String() In DataTables("学生成绩").GetValues("班级|姓名")
    Dim dr1 As DataRow = DataTables("表B").AddNew()
    dr1("班级") = v(0)
    dr1("姓名") = v(1)
    For Each dr2 As DataRow In DataTables ("学生成绩").Select("班级 = '" & v(0) & "' And 姓名 = '" & v(1) & "'")
        dr1(dr2("科目")) = dr2("分数")
    Next
Next
DataTables("表B").DataCols.Add("总分",Gettype(Double),"[语文] + [英语] + [数学] + [物理] + [化学]")
MainTable = Tables("表B")
```

当然，我们同样也可以直接用 SQL 语句来实现：

```
Select 班级,姓名,Sum(iif(科目 = '语文',分数,0)) As 语文,Sum(iif(科目 = '数学',分数,0)) As 数学,Sum(iif(科目 = '英语',分数,0)) As 英语,Sum(iif(科目 = '物理',分数,0)) As 物理,Sum(iif(科目 = '化学',分数,0)) As 化学,Sum(分数) As 总分 From {学生成绩} Group by 班级,姓名
```

上述 SQL 语句巧妙地利用了分组统计，希望大家细细体会一下其中的奥妙。

需要注意的是，如果使用的数据源是 SQL Server，是不能使用 iif 函数的，需要用 Case When 语句：

```
Select 班级,姓名,Sum(Case When 科目 = '语文' Then 分数 Else 0 End) As 语文,Sum(Case When 科目 = '数学' Then 分数 Else 0 End) As 数学,Sum(Case When 科目 = '英语' Then 分数 Else 0 End) As 英语,Sum(Case When 科目 = '物理' Then 分数 Else 0 End) As 物理,Sum(Case When 科目 = '
```

化学' Then 分数 Else 0 End) As 化学,Sum(分数) As 总分 From {学生成绩} Group by 班级,姓名

我们可以新建一个查询表，或者用 QueryBuilder，将其 Select 语句设置为 上述语句即可。

示例三

现在反过来，假定有个水平结构的表：

	班级	姓名	语文	数学	英语	物理	化学	总分
1	1班	陈成进	133	136	140	71	75	555
2	1班	陈诗淇	110	58	106	36	34	344
3	1班	陈思达	123	75	98	79	66	441
4	1班	程康文	110	123	117	78	78	506
5	1班	邓玉婷	118	38	74	52	18	300
6	1班	林景阳	129	118	139	88	80	554
7	1班	苏　捷	132	136	146	93	94	601
8	1班	谢杰立	119	133	95	57	61	465
9	1班	谢晓霞	129	109	126	66	80	510
10	1班	杨景亮	116	120	132	78	83	529
11	1班	张李华	124	118	124	83	88	537
12	2班	陈成进	110	58	106	36	34	344
13	2班	陈诗淇	113	123	117	78	78	509
14	2班	陈思达	113	55	92	55	46	361

需要转换为垂直结构的表：

	班级	姓名	科目	分数
1	1班	陈成进	语文	133
2	1班	林景阳	语文	129
3	1班	苏　捷	语文	132
4	1班	谢晓霞	语文	129
5	2班	谢杰立	语文	129
6	2班	谢晓霞	语文	134
7	2班	杨景亮	语文	132
8	2班	张李华	语文	133
9	4班	谢杰立	语文	129
10	4班	谢晓霞	语文	134
11	4班	杨景亮	语文	132
12	4班	张李华	语文	133
13	1班	陈成进	数学	136
14	1班	苏　捷	数学	136
15	1班	谢杰立	数学	133
16	2班	林景阳	数学	133
17	2班	杨景亮	数学	136
18	4班	林景阳	数学	133

转换代码为：

```
Dim dtb As New DataTableBuilder("表B")
dtb.AddDef("班级", Gettype(String), 4)
dtb.AddDef("姓名", Gettype(String), 10)
dtb.AddDef("科目", Gettype(String), 10)
dtb.AddDef("分数", Gettype(Double))
```

```
dtb.Build()
Dim kms() As String = {"语文","数学","英语","物理","化学"}
For Each dr1 As DataRow In DataTables("成绩表").DataRows
    For Each km As String In kms
        Dim dr2 As DataRow = DataTables("表B").AddNew()
        dr2("班级") = dr1("班级")
        dr2("姓名") = dr1("姓名")
        dr2("科目") = km
        dr2("分数") = dr1(km)
    Next
Next
MainTable = Tables("表B")
```

我们也可以用 SQL 语句实现,新建一个查询表，或者用 QueryBuilder，将其 Select 语句设置为：

```
Select 班级,姓名,'语文' As 科目,语文 As 分数 From {学生成绩} Union All Select 班级,姓名,'数学' As 科目,数学 As 分数 From {学生成绩} Union All Select 班级,姓名,'英语' As 科目,英语 As 分数 From {学生成绩} Union All Select 班级,姓名,'化学' As 科目,化学 As 分数 From {学生成绩} Union All Select 班级,姓名,'物理' As 科目,物理 As 分数 From {学生成绩}
```

上面的语句用 Union All 连接了 5 个查询，每个查询针对一个科目，以第一个科目语文为例，其 Select 语句为：

```
Select 班级, 姓名, '语文' As 科目,语文 as 分数 From {学生成绩}
```

这个 Select 语句生成的表包括四列，分别是班级、姓名、科目、分数，其中班级、语文就是原始成绩表的班级列和语文列，而科目列的内容固定为"语文"，分数列就是原始成绩表的语文列，只不过被重命名为分数列。

其它科目的查询语句一样，用 Union All 组合每个科目的查询语句，就得到我们所需要的表了。

4.26.21 获取特殊目录的路径

Foxtable 提供了一个 SpecialFolder 类型，用于获得一些特殊目录的路径。

该类型的属性有：

DesktopDirectory：返回桌面目录

MyDocuments：返回我的文档目录

MyMusic：返回我的音乐目录

MyPictures：返回我的图片目录

ProgramFiles：返回程序目录

System：返回系统目录

示例：

下面的代码在我的文档目录创建了一个名为 test.txt 的文本文件：

```
Dim s As String = "I Like Foxtable"
Dim f As String = SpecialFolder.MyDocuments & "\test.txt"
FileSys.WriteAllText(f, s, False)
```

4.26.22 ShowAppWindow 方法

查找指定标题的外部应用程序窗口，并显示、隐藏或关闭之。
注意不是 Foxtable 中的窗口，而是外部应用程序窗口。
语法：
ShowAppWindow(Caption, State)

- Caption
  字符型，查找窗口标题包括的字符。

- State
  整数型，指定窗口状态。
  1：正常大小显示
  2：最大化显示
  3：最小化
  4：隐藏
  5、关闭

如果找到指定标题的窗口，在完成指定操作后，会返回 True，否则返回 False。
示例一
假定用 Excel 打开了一个名为"订单.xls"的文件，希望最大化显示此窗口：
ShowAppWindow("订单.xls",2)
如果你要关闭这个窗口，只需：
ShowAppWindow("订单.xls",5)
示例二
如果已经打开了“订单.xls”这个文件，则显示这个窗口，否则打开这个文件：

```
If ShowAppWindow("订单.XLS",1) = False Then
    Dim Proc As New Process
    Proc.File = "C:\data\订单.xls"
    Proc.Start
End If
```

4.26.23 Image 类

Image 表示图片。
Image 的属性有：
Width：以像素为单位的宽度
Height：以像素为单位的高度
HorizontalResolution：水平分辨率，也就是水平方向每英寸的像素个数。
VerticalResolution：垂直分辨率，也就是垂直方向每英寸的像素个数。

示例

计算指定图片的面积，以平方米为单位：

```
Dim img As Image = GetImage("c:\data\girl.jpg")
Dim mj As Double
If img IsNot Nothing Then
    mj = mj + img.width * img.height / img.HorizontalResolution / img.HorizontalResolution / 1550.0031
End If
OutPut.Show("面积:" & mj & "(平方米)")
```

提示： 1平方米等于1550.0031平方英寸。

4.26.24 StringBuilder 类

StringBuilder 通常用于大规模地合并字符串。

StringBuilder 的方法有：

名称	说明
Append	将指定的字符串合并到 StringBuilder 中
AppendLine	另起一行，然后将指定的字符串合并到 StringBuilder 中。
ToString	将 StringBuilder 中的内容转换为字符串。

示例

在命令窗口执行下面的代码：

```
Dim dt As Date = Date.Now
Dim v As String
Dim s As new StringBuilder
For i As Integer = 1 To 100000
    s.AppendLine("abc")
Next
v = s.Tostring
Output.Show((Date.now - dt).Totalseconds)
```

在我的电脑测试，上述代码耗时 0.015 秒。

如果不采用 StringBuilder，直接用字符变量和合并：

```
Dim dt As Date = Date.Now
Dim v As String
For i As Integer = 1 To 100000
    v = v & "abc" & vbcrlf
Next
Output.Show((Date.now - dt).Totalseconds)
```

在我的电脑运行上述代码，耗时 100.5 秒。

可以看出在大规模合并字符串的时候，使用 StringBuider 比直接使用字符变量的性能高出数千倍。

并非任何时候都要使用 StringBuilder，例如我们将上面的代码修改一下，循环次数改为 1000：

```
Dim dt As Date = Date.Now
Dim v As String
For i As Integer = 1 To 1000
    v = v & "abc" & vbcrlf
Next
Output.Show((Date.now - dt).TotalSeconds)
```

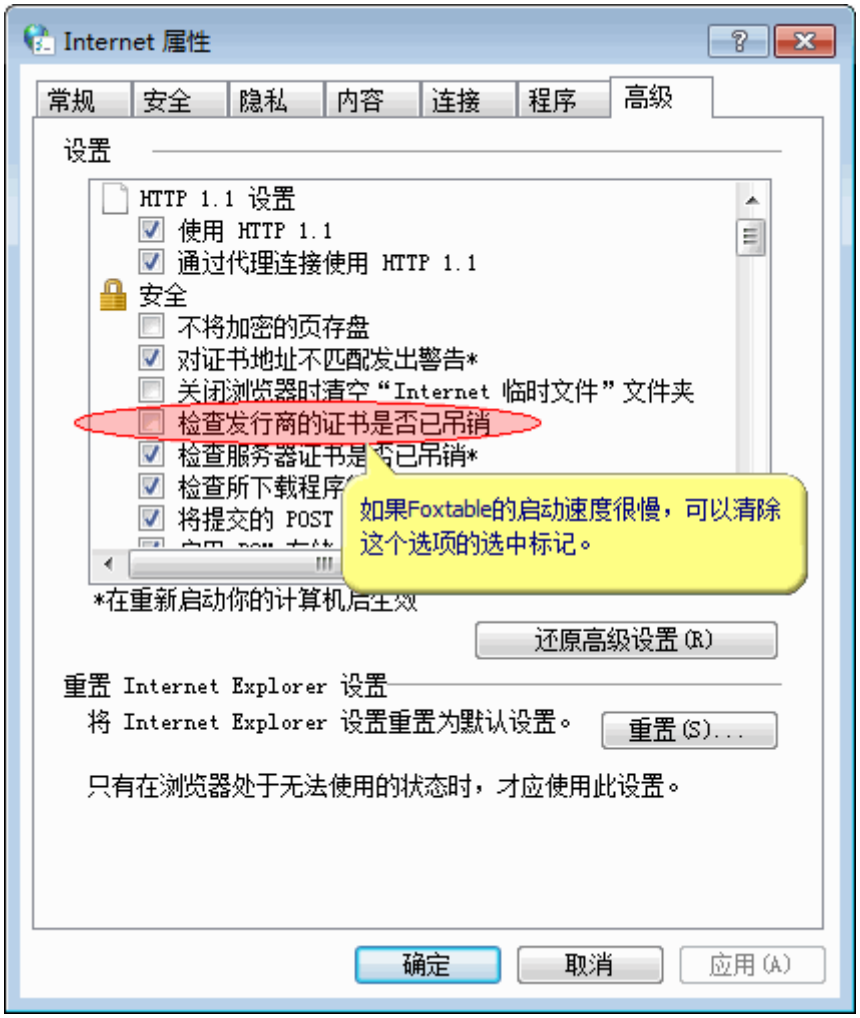
耗时只有 0.15 秒，如果循环次数改为 100，耗时为 0 秒，显然此时是否使用 StringBuilder，对于性能的影响都是轻微的。

4.26.25 网络与启动速度

有少数用户反映，个别电脑在连接局域网但不能连接互联网的情况下，Foxtable 的启动速度会很慢，有的甚至需要两分钟左右才能启动完毕。

其实不单单是 Foxtable，AutoCad 等软件一样存在这个问题。

解决这个问题的方法很简单，可以在控制面板中，打开 Internet 选项，在高级设置中，清除“检查发行商的证书是否吊销”的勾选状态：



4.26.26 修复 MDAC 2.8

有些 Windows XP 用户，无法更改表名或列名，也不能增加或删除列，这通常是 MDAC 2.8 已经损坏的原因。

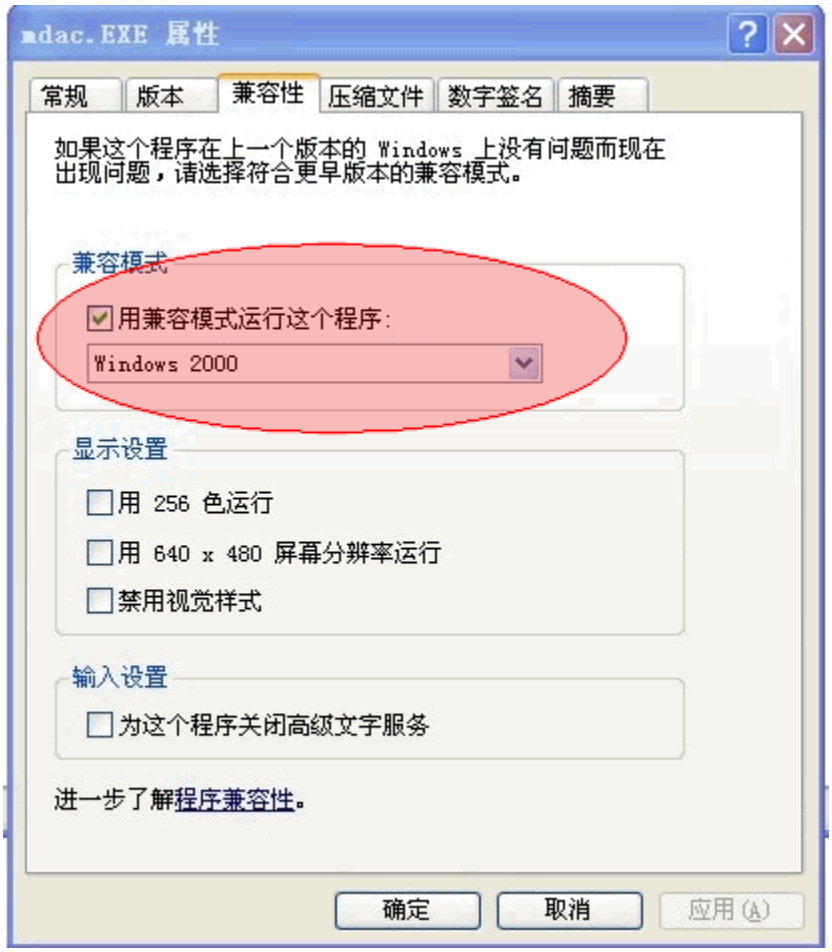
修复 MDAC 2.8 的步骤：

1、下载 MDAC 2.8 安装程序

http://www.foxtable.com/download/mdac.rar

2、解压后得到可执行文件 mdac.exe，但是这个文件还不能直接执行。

3、右击文件 mdac.exe，从弹出的快捷菜单中，单击“属性”，按下图所示设置其兼容模式:



4，设置好兼容模式后，双击 mdac.exe 文件，安装 MDAC 2.8。

5、安装完成后，重新启动电脑。

4.26.27 在代码中拆分和合并语句

将单个语句拆分为多行

编写代码时，有时会创建一些很长的语句，使得您必须在代码编辑器中水平滚动。虽然这并不影响代码的运行，但会影响代码的阅读。

在这种情况下，应该考虑将单个长语句拆分为几行。

在要将该行断开的位置处使用行继续符，它是一个下划线 (_)，该下划线必须紧跟在空格后，并且在它后面紧跟行终止符（回车）。

例如：

```
If e.name = "成绩" Then
    e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & _
    e.ProjectPath & "成绩数据.mdb;Persist Security Info=False"
End If
```

上面的代码完全等效于：

```
If e.name = "成绩" Then
    e.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & e.ProjectPath & "成绩数据.mdb;Persist Security Info=False"
End if
```

再例如：

```
Dim s As String
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午,在四川抗震救灾的" & _
    "危急时刻,中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川" & _
    "省地震灾区,慰问灾区干部群众,看望奋战在抗震救灾第一线的部队官兵，公" & _
    "安民警和医护人员,指导抗震救灾工作。"
```

注意:

行继续符必须是一行上最后的字符，在同一行上，行继续符后不能跟有任何其他内容。
单个参数名必须保持完整，不能在参数名中间使用行继续符
不能使用行继续符来拆分注释，注释开始后，编译器将不检查字符是否具有特殊含义。要将注释拆分为多行，请在每行的前面重复使用注释符号 (')。

将多个语句置于同一行

虽然我们的建议是让每条语句各占一行，但是 Visual Basic 同样允许在同一行上放置多条语句。用冒号 (:) 将各语句分开，如下面的示例所示：

```
text1.Text = "Hello" : text1.BackColor = System.Drawing.Color.Red
```

上面的代码等效于：

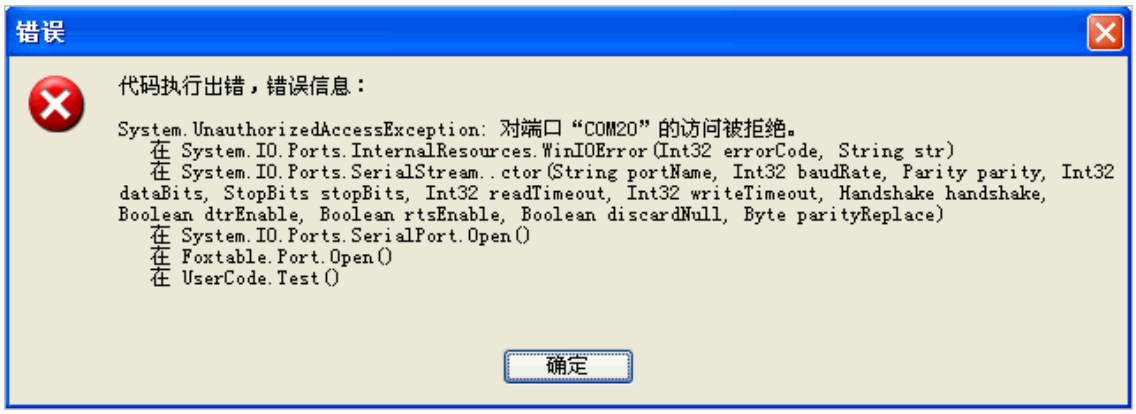
```
text1.Text = "Hello"
text1.BackColor = System.Drawing.Color.Red
```

4.26.28 捕获异常错误

有些错误很难事先判断，例如对于下面的代码：

```
Ports.Add("COM20")
Ports("COM20").Open()
```

如果 COM20 已经被其他程序所使用，那么上面的代码会出现错误提示：



显然这样的提示对于最终用户来说，是不太友好的，但是 Foxtable 并没有办法事先判断串口是否被其他程序占用，所以我们也无法给用户提供更友好的提示。

不过我们可以采用下面的语句捕获无法预判的错误：

```
Try
   代码一
Catch ex As Exception
   代码二
End Try
代码三
```

代码一为正常的代码，如果执行过程中没有发现错误，则跳过代码二，执行代码三，如果在执行代码一的过程中发现错误，则不提示错误信息，立即执行代码二，之后再执行代码三，这样我们就可以捕获代码一运行过程中的错误，在代码二中给用户提供更友好的提示。

例如：

```
Try
    Ports.Add("COM20")
    Ports("COM20").Open()
Catch ex As Exception
    MessageBox.Show("COM20 已经被其它程序占用,无法打开!","提示",MessageBoxButtons.OK)
End Try
```

不要轻易使用 Try 语句

不要轻易使用 Try 语句来捕获错误，除非遇到事先不能预判的错误，随便使用 Try 语句会降低代码质量和运行效率。

例如下面两段代码：

```
Dim dr As DataRow
dr = DataTables("订单").Find("[产品] = '酱油'")
If dr IsNot Nothing
    dr("折扣") = 0.05
Else
    MessageBox.Show("没有找到符合条件的行!")
End If
```

和：

```
Try
   Dim dr As DataRow
   dr = DataTables("订单").Find("[产品] = '酱油'")
```

```
    dr("折扣") = 0.05
Catch ex As Exception
    MessageBox.Show("没有找到符合条件的行!")
End Try
```

无论从哪个方面来说，第一段的质量和效率都好过第二段。

4.26.29 数据库事务

一般用户可以忽略本节内容。

Connection 有三个和事务相关的方法，分别是：

- BeginTransaction
  开始数据库事务

- Commit
  提交数据库事务

- Rollback
  回滚数据库事务

只有使用外部数据源，才能支持事务。

只有通过 SQLCommand 执行 SQL 语句的时候，才可以使用事务。

事务一旦开启，在提交或回滚之前，都只能执行通过 SQLCommand 执行 SQL 语句，不能有其他读写后台数据库的操作。

示例

下面的代码用于删除订单编号为 32 的订单及其订单明细：

```
Try
    Connections("数据源").BeginTransaction() '开始事务
    Dim cmd As new SQLCommand
    cmd.ConnectionName = "数据源"
    cmd.CommandText = "Delete From {订单} Where [订单编号] = 32"
    cmd.ExecuteNonQuery
    cmd.CommandText = "Delete From {订单明细} Where [订单编号] = 32"
    cmd.ExecuteNonQuery
    Connections("数据源").Commit '提交事务,所有操作生效
Catch ex As Exception '如果出错
    Connections("数据源").Rollback() '回滚事务,撤销所有操作
End Try
```

4.26.30 暂停执行事件代码

暂停所有事件

Foxtable 有一个逻辑型的系统变量，将其设置为 False，将停止 执行所有事件代码，将其设置为 True，将恢复执行所有事件代码，就像一个事件“开关”。

例如希望向一个表中追加大量数据，并要求在追加数据的过程中，不要执行任何事件代码，可以如下编写代码:

```
SystemReady = False
'追加数据代码
SystemReady = True
```

如果在执行上述代码的过程中发生错误退出，那么 SystemReady 将始终为 False，系统进入锁死状态，不会执行任何事件代码，单击菜单按钮也不会有反应。

所以比较安全的编码方式为：

```
SystemReady = False
Try
'追加数据代码
Catch ex As Exception
    MessageBox.Show("追加数据失败")
End Try
SystemReady = True
```

暂停单个事件

如果要暂停单个事件的执行，那么 SystemReady 是无能为力的。

因为 Foxtable 并没有针对单个事件的“开关”，不过我们可以自行设计一个这样的“开关”，而且设计过程很简单，只需寥寥几行代码即可：

1、首先我们在全局代码中定义一个 Public 变量，假定变量名为 pause：

```
Public pause As Boolean
```

2、然后在某事件的开始位置添加一段代码：

```
'判断是否执行事件:
If pause = True Then
    Return
End If
'原来事件代码
```

3、现在我们就可以为此事件设计开关了：

关闭此事件的代码为：

```
pause = True
```

开启此事件的代码为：

```
pause = False
```

4.26.31 常见效率问题

4.26.31.1 计算代码执行耗时

在编程的过程中，为了优化代码，提高效率，有时需要知道执行某段代码所耗费的时间。

可以参考下面的结构，自动显示某段代码所耗费的时间：

```
Dim st As Date = Date.Now
'要测试耗时的代码
MessageBox.Show("耗时: " & (Date.Now - st).TotalSeconds & "秒")
```

例如：

```
Dim st As Date = Date.Now '将开始时间保存在变量 st 中
Dim nms As List(of String) = DataTables("表 A").GetValues("客户")
DataTables("表 B").StopRedraw()
For Each nm As String In nms
    Dim dr As DataRow = DataTables("表 B").AddNew()
    dr("客户") = nm
    dr("付款合计") = DataTables("表 A").Compute("Sum(金额)", "客户  = '" & nm & "' And tk = 1")
    dr("欠款合计") = DataTables("表 A").Compute("Sum(金额)", "客户  = '" & nm & "' And tk = 0")
Next
DataTables("表 B").ResumeRedraw()
MessageBox.Show("耗时: " & (Date.Now - st).TotalSeconds & "秒") '计算并显示执行代码所花费的秒数
```

4.26.31.2 查询与赋值并存

以下内容需要使用 List(集合)和 Dictionary(字典)， 其中 Dictionary 大家较少接触，如果对其不熟悉，请先行参考：Dictionary

在技术支持的过程中，我遇到一个用户，他希望在第三列中标出第一列有而第二列无的数据 ，其中第一列和第二列为整数型：

	第一列	第二列	第三列
1	1744	6305	☑
2	13202	10772	☐
3	18135	28770	☑
4	22347	10263	☐
5	23214	15358	☐
6	7078	10155	☑
7	10737	17048	☑

我给出的代码为：

```
For Each dr As DataRow In DataTables("表 A").DataRows
    If DataTables("表 A").Find("第二列  = " & dr("第一列") ) Is Nothing Then
        dr("第三列") = True
```

```
    Else
        dr("第三列") = False
    End If
Next
```

随后我进行了测试，在数据为 1 万行，第三列无数据的时候，执行上述代码花费了整整 234 秒,显然这样的效率是让人难以接受的。

上面的代码并没有什么特别的地方，在遍历过程中找出符合条件的行，并给此行的某列赋值而已。

那么到底是查找出了问题，还是赋值出了问题，以至于运行效率如此之低呢？

为了找出原因，我分别测试了查询和赋值：

测试查询的代码：

```
Dim v As Boolean
For Each dr As DataRow In DataTables("表 A").DataRows
    If DataTables("表 A").Find("第二列  = " & dr("第一列")) Is Nothing Then
        v = dr("第三列")
    End If
Next
```

测试赋值的代码：

```
For Each dr As DataRow In DataTables("表 A").DataRows
    dr("第三列") = True
Next
```

出人意料的是，两段代码运行得都非常快，都在瞬间完成了。

那为什么第一段代码需要花费 234 秒？难道查询和赋值不能出现在同一个遍历语句中吗？

为此我改写了第一段代码，让查询和赋值分开进行，新的代码为：

```
Dim lst1 As New List(of DataRow)
Dim lst2 AS New List(of DataRow)
For Each dr As DataRow In DataTables("表 A").DataRows
    If DataTables("表 A").Find("第二列  = " & dr("第一列")) Is Nothing Then
        lst1.Add(dr)
    Else
        lst2.Add(dr)
    End If
Next
For Each dr As DataRow In lst1
    dr("第三列") = True
Next
For Each dr As DataRow In lst2
    dr("第三列") = False
Next
```

同样在数据为 1 万行，第三列无数据的时候，执行上述代码只花了 0.8 秒，比原来的 234 秒快了整整 300 倍，效率差距之大，令人瞠目。

原因应该找到了，为了验证我的想法，我打算对 Compute 方法进行测试，因为 Compute 在计算过程同样要进行查询。

于是我另外写了两段代码，这次不使用 Find 进行查询，而是使用 Compute 方法进行计算：

常规的代码，计算和赋值在同一个遍历语句：

```
Dim v As Double
```

```
For Each dr As DataRow In DataTables("表 A").DataRows
    v = DataTables("表 A").Compute("Count(第二列 )", "第二列 = " & dr("第一列"))
    dr("第四列") = v
Next
```

估计效率更高的代码，计算和赋值分开进行:

```
Dim Dic As new Dictionary(of DataRow, Integer)
Dim v As Double
For Each dr As DataRow In DataTables("表 A").DataRows
    v = DataTables("表 A").Compute("Count(第二列 )", "第二列 = " & dr("第一列"))
    dic.Add(dr, v)
Next
For Each dr As DataRow In dic.Keys
    dr("第四列") = dic(dr)
Next
```

果不其然，经过测试，第二段比第一段同样快了整整 300 倍。

现在我总结一下，符合以下三个条件，会出现效率低下的情况：

1、用 For 语句遍历某个表。

2、遍历过程中会用 Find 或 Select 查询此表，或者用 Compute 方法统计此表。

3、遍历过程中会大量修改此表中某些行的值，被修改的行数越多，对于性能影响越大，如果被修改的行数很少，则几乎没有影响。

强调一下，上述三项中提到的表必须都是同一个表。

我们改写的代码之所以高效， 是因为改写后的代码使得上述 2、3 项不再出现在同一个遍历语句中，而是在不同的遍历语句中出现。

4.26.31.3 非等于比较

在学习本节内容之前，请先打开 CaseStudy 目录下的文件：代码优化.Table

这次问题的表结构如下：

	项目	日期	支出	累计支出
1	A68	2000-01-03	3032	
2	A57	2000-01-03	989	
3	A9	2000-01-04	1265	
4	A69	2000-01-08	1391	
5	A9	2000-01-10	650	
6	A20	2000-01-13	2032	
7	A6	2000-01-13	2974	
8	A55	2000-01-20	1507	
9	A59	2000-01-26	2944	
10	A67	2000-01-27	1950	
11	A37	2000-01-29	1384	

要求在累计支出列计算出每个项目截止到当日为止的累计支出，标准的代码是：

```
Dim st As Date = Date.Now
Dim dic As New Dictionary(of DataRow, String)
Dim drs As List(of DataRow)
For Each dr1 As DataRow In DataTables("表 A").DataRows
```

```
    Dim Filter As String
    Filter = "项目 = '" & dr1("项目") & "' And 日期 <= #" & dr1("日期") & "#"
    dr1("累计支出") = DataTables("表 A").Compute("Sum(支出)",Filter)
Next
MessageBox.Show("耗时: " & (Date.Now - st).TotalSeconds & "秒")
```

在一万行数据的情况下，执行上述代码花了 390 秒，这完全在意料之中。

有了示例一的经验，我信心十足地给出了“优化”后的代码：

```
Dim dic As New Dictionary(of DataRow, String)
Dim drs As List(of DataRow)
For Each dr1 As DataRow In DataTables("表 A").DataRows
    Dim Filter As String
    Filter = "项目 = '" & dr1("项目") & "' And 日期 <= #" & dr1("日期") & "#"
    dic.Add(dr1, DataTables("表 A").Compute("Sum(支出)",Filter))
Next
For Each dr As DataRow In dic.Keys
    dr("累计支出") = dic(dr)
Next
```

可出乎意料的是，在一万行数据的情况下，执行上述代码同样花了 390 秒，效率竟然没有丝毫的提升。

为什么之前的招数失效了呢？

仔细分析一下可以看出，和之前的例子不同，上面的查询条件不全是“等于”比较，其中日期的比较是“小于等于”比较。

实际上在查询中，“等于”比较是最高效的，其他任何形式的比较效率都要低很多，也许真的就是这个“小于等于”比较影响了代码的执行效率。

但是“小于等于”比较是必须，不可能不用，我们唯一能着手的就是减少“小于等于”比较的次数，改良后的代码为：

```
Dim dic As New Dictionary(of DataRow, String)
Dim drs As List(of DataRow)
For Each dr1 As DataRow In DataTables("表 A").DataRows
    Dim dv As Date = dr1("日期")
    Dim sm As Double = 0
    drs = DataTables("表 A").Select("项目 = '" & dr1("项目") & "'", "日期") '注意要根据日期排序
    For Each dr2 As DataRow In drs
        If dr2("日期") <= dv Then
            sm = sm + dr2("支出")
        Else
            Exit For
        End If
    Next
    dic.Add(dr1, sm)
Next
For Each dr As DataRow In dic.Keys
    dr("累计支出") = dic(dr)
Next
```

同样在一万行数据的情况下，执行上述代码花了 2 秒，看来之前的判断完全正确。
代码的原理是，先筛选出相同项目的行，再比较这些行的日期，从而大大较少了“小于等于”比较的次数，如果你的项目是 100 个，那么比较次数就减少了 100 倍，运行效率也就提高了 100 倍。
既然查询中的非等于比较是影响效率的关键，于是我索性彻底改写代码，连查询也不用了，最新的代码为：

```
Dim drs As List(of DataRow) = DataTables("表 A").Select("", "项目,日期") '注意排序参数
drs(0)("累计支出") = drs(0)("支出") '设置第一行的累计支出
For i As Integer = 1 To drs.Count - 1  '从第二行开始逐行计算累计支出
    If drs(i)("项目") = drs(i - 1)("项目") Then
        drs(i)("累计支出") = drs(i-1)("累计支出") + drs(i)("支出")
    Else
        drs(i)("累计支出") = drs(i)("支出")
    End If
Next
```

经过测试，处理 1 万行数据只用了 0.9 秒，效率又提升了一倍有余，而且代码更为简洁易懂。

从 390 秒到 2 秒，再从 2 秒到 0.9 秒，可见代码的优化对于效率的提升是何其重要。

代码效率的提升是没有定式的，实际编码的过程中只有运用排除法，找出可能影响效率的因素，然后有针对性地优化改进。

4.26.31.4 逻辑列的查询

在学习本节内容之前，请先打开 CaseStudy 目录下的文件：逻辑列的查询.Table
本节内容有两个表，表 A 的结构为：

	客户	金额	已付款
1	PD670	26009	☑
2	PD125	12289	☑
3	PD067	20008	☑
4	PD044	46615	☐
5	PD823	21431	☑
6	PD967	36551	☑

表 B 的结构为：

	客户	付款合计	欠款合计
1			
2			
3			
4			

希望在表 B 统计出客户的付款和欠款总额，通常大家会写出这样的代码：

```
Dim nms As List(of String) = DataTables("表 A").GetValues("客户")
DataTables("表 B").StopRedraw()
For Each nm As String In nms
    Dim dr As DataRow = DataTables("表 B").AddNew()
    dr("客户") = nm
```

```
    dr("付款合计") = DataTables("表 A").Compute("Sum(金额)", "客户 = '" & nm & "' And 已付款 = True")
    dr("欠款合计") = DataTables("表 A").Compute("Sum(金额)", "客户 = '" & nm & "' And 已付款 = False")
Next
DataTables("表 B").ResumeRedraw()
```

在表 A 为 5000 行，客户数为 1000 的时候，执行上述代码所花费的时间为 26 秒，效率不太理想。经过分析，我怀疑是因为逻辑值的比较导致运行效率偏低，为了验证这个想法，我将已付款列改为字符型，用"Y"表示已经付款，"N"表示未付款，将代码改为：

```
Dim nms As List(of String) = DataTables("表 A").GetValues("客户")
DataTables("表 B").StopRedraw()
For Each nm As String In nms
    Dim dr As DataRow = DataTables("表 B").AddNew()
    dr("客户") = nm
    dr("付款合计") = DataTables("表 A").Compute("Sum(金额)", "客户 = '" & nm & "' And 已付款 = 'Y'")
    dr("欠款合计") = DataTables("表 A").Compute("Sum(金额)", "客户 = '" & nm & "' And 已付款 = 'N'")
Next
DataTables("表 B").ResumeRedraw()
```

在表 A 为 5000 行，客户数为 1000 的时候，执行上述代码所花费的时间为 0.25 秒，效率比之前提高了 100 倍。

但显然逻辑列是不可能不用的，是什么原因导致逻辑列的查询效率如此之低？ 又有什么解决办法呢？

实际上 在 DataTable 中，逻辑列的值并不是以 True 和 False 存储的，而是以 1 和 0 存储的，当我们从 DataTable 中取值时，会自动将 1 转换为 True，0 转换为 False 后返回。

既然如此，我重新将已付款列改为逻辑型，然后修改代码，直接在表达式中用 1 和 0 代替 True 和 False 进行比较：

```
Dim nms As List(of String) = DataTables("表 A").GetValues("客户")
DataTables("表 B").StopRedraw()
For Each nm As String In nms
    Dim dr As DataRow = DataTables("表 B").AddNew()
    dr("客户") = nm
    dr("付款合计") = DataTables("表 A").Compute("Sum(金额)", "客户 = '" & nm & "' And 已付款 = 1")
    dr("欠款合计") = DataTables("表 A").Compute("Sum(金额)", "客户 = '" & nm & "' And 已付款 = 0")
Next
DataTables("表 B").ResumeRedraw()
```

在表 A 为 5000 行，客户数为 1000 的时候，执行上述代码所花费的时间为 0.23 秒，效果非常好。

在查询表达式中用 1 代替 True，用 0 代替 False，效率提高超过 100 倍，切记切记。

4.26.31.5 表达式列的影响

如果表达式列引用父表数据，或者统计子表数据，那么对于批量修改数据的效率影响很大，不过我们有很好的方法避免这种情况的发生。

在编写此节内容之前，我建立了一个文件，该文件包括两个表，分别是产品和订单，两者建立了关联。

产品表的“总金额”列是一个表达式列，其表达式为：Sum(Child.金额)。

订单表则有两个表达式列，其中“单价”列的表达式为：Parent.单价，“金额”列的表达式为：数量 * 单价 * (1 - 折扣)

产品表有 1 千行数据，订单表有 1 万行数据。

示例一

假定希望所有产品都提价 5%，代码为：

```
For Each dr As DataRow In DataTables("产品").DataRows
    dr("单价") = dr("单价") * 1.05
Next
```

在产品表只有区区 1000 行数据的情况下，执行上述代码居然花费了 31 秒。

这是因为订单表的单价列引用了产品表的单价，每修改一个产品的单价，都必须同步去更新该产品所有订单的单价。

为了避开这种影响，我将代码改为：

```
DataTables("订单").DataCols("单价").Expression = ""
For Each dr As DataRow In DataTables("产品").DataRows
    dr("单价") = dr("单价") * 1.05
Next
DataTables("订单").DataCols("单价").Expression = "Parent.单价"
```

经过测试，执行上述代码花费 0.16 秒，比之前提高了足足 200 倍。

原理很简单，先清除订单表单价列的表达式，待代码执行结束后，再重新设置此列的表达式。

实际上只要子表引用了父表的数据，批量修改父表中的任何一列都会受到影响，当然批量修改被引用的列受影响的程度最为明显。

示例二

假定希望给所有订单增加 5%的折扣，代码为：

```
For Each dr As DataRow In DataTables("订单").DataRows
    dr("折扣") = dr("折扣") + 0.05
Next
```

经过测试，执行上述代码耗时 78 秒，效率很低。

究其原因，是因为修改订单的折扣，会重算订单的金额，而订单金额的变化又会重算产品的总金额。

知道原因就好办，我将代码改为：

```
DataTables("订单").DataCols("金额").Expression = ""
For Each dr As DataRow In DataTables("订单").DataRows
    dr("折扣") = dr("折扣") + 0.05
Next
DataTables("订单").DataCols("金额").Expression = "数量 * 单价 * (1 - 折扣)"
```

经过测试，执行上述代码仅耗时 0.95 秒，效率大幅度提高。

如果将所有的表达式清除：

```
DataTables("订单").DataCols("金额").Expression = ""
DataTables("订单").DataCols("单价").Expression = ""
DataTables("产品").DataCols("总金额").Expression = ""
For Each dr As DataRow In DataTables("订单").DataRows
    dr("折扣") = dr("折扣") + 0.05
Next
DataTables("订单").DataCols("单价").Expression = "Parent.单价"
DataTables("订单").DataCols("金额").Expression = "数量 * 单价 * (1 - 折扣)"
DataTables("产品").DataCols("总金额").Expression = "Sum(Child.金额)"
```

效率得到进一步的提高，现在耗时只需 0.54 秒，当然一般没有必要这么做，清除影响最大的表达式即可，毕竟不到半秒的差距在实际使用的过程中是很难感觉到的。

4.26.31.6 流水账与重置列

假定下图的流水账有 10000 行数据：

	日期	收入	支出	余额
1	1999-02-01	1		1
2	1999-02-03	7	5	3
3	1999-02-05	2		5
4	1999-02-07	4	6	3
5	1999-02-09	8		11
6	1999-02-11	4		15
7	1999-02-13	6	4	17
8	1999-02-15	3		20
9	1999-02-17	9	6	23
10	1999-02-19	4		27

如果重置收入列，当我们重置第一行的时候，会重算所有 10000 行的余额，当重置第二行的时候，会重算剩下的 9999 行的余额，同样重置第三行的时候，回重算余下的 9998 行的余额，这样算下来，重置一次收入列，等于重算了 50005000 行的余额，也许一个昼夜也无法完成这样一次重置，而我们实际只有 10000 行数据，也只需重算 1 万行的余额。

显然我们只需重置第一行的收入列，即可重算所有 10000 行的余额，所以我们应该针对上面的流水账加上一个自定义的重置命令：

```
Dim dr As DataRow
dr = DataTables("表名").Find("","[_SortKey]") '找出第一行
If dr IsNot Nothing Then
    '模拟第一行的支出发生变化，刷新已加载行的余额.
    DataTables("表名").DataCols("支出").RaiseDataColChanged(dr)
End If
```

上述代码的效率比直接重置收入列快 5000 倍。

对于下图这种区分产品的流水账:

	日期	产品	入库	出库	库存
1	1999-02-01	PD01	1		1
2	1999-02-03	PD02	7	5	2
3	1999-02-05	PD03	2	2	0
4	1999-02-07	PD01	4	3	2
5	1999-02-09	PD02	8		10
6	1999-02-11	PD03	4		4
7	1999-02-13	PD01	6	4	4
8	1999-02-15	PD02	3		13
9	1999-02-17	PD03	9	6	7
10	1999-02-19	PD01	4		8
11	1999-02-17	PD02	7	4	16

显然也只需重置每个产品的第一行的入库列，即可重算所有行的库存，所以我们也应该加上一个自定义的重置按钮：

```
Dim nms As New List(Of String)
Dim drs As New List(of DataRow)
With DataTables("表名")
    nms = .GetValues("产品")
    For Each nm As String In nms '找出每个产品的第一行数据，添加到集合 drs 中
        drs.Add(.Find("产品  = '" & nm & "'", "[_SortKey]"))
    Next
    For Each dr As DataRow In drs
        .DataCols("入库").RaiseDataColChanged(dr)
    Next
End With
```

4.26.31.7 用 Table 控件模拟关联表

其实不管是双向生成还是单向生成，都会产生关联表，对于性能有一定程度的影响，只是双向生成的时候尤甚。

所以要追求极致的性能，最好是在设置关联的时候，将关联表生成模式选择为“无”。

但是关联表确实给我们带来了很大的便利，我们岂能因噎废食？

可否鱼和熊掌兼得，既可以得到关联表的好处，又不影响性能呢？

完全可以，这就是本节要介绍的内容。

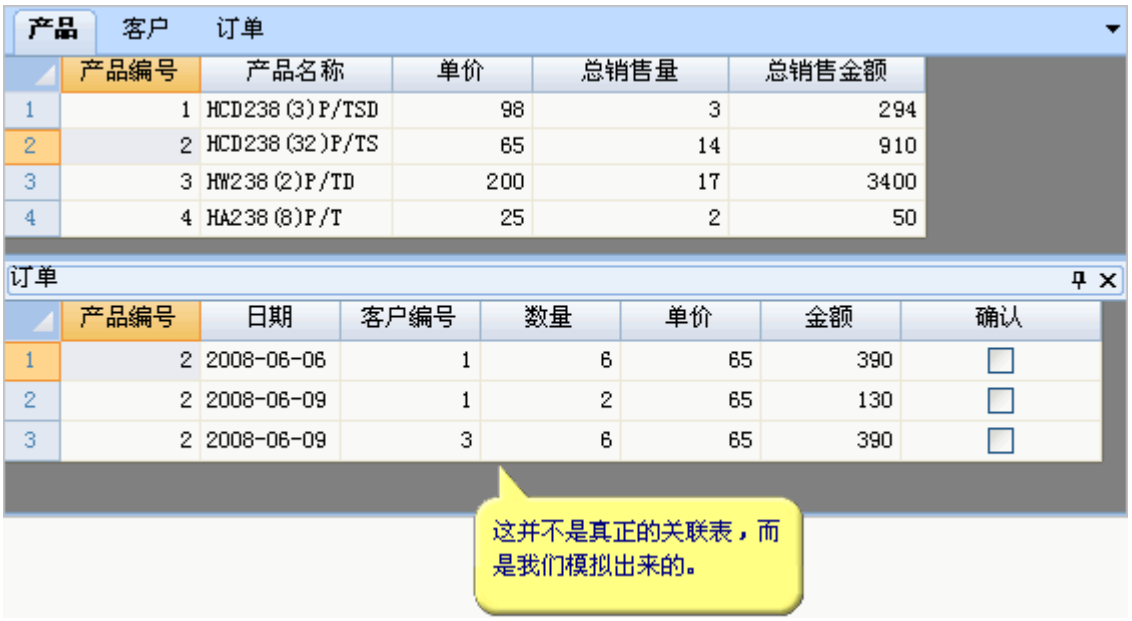
请首先请打开 CaseStudy 目录下的文件：模拟关联表.Table

该文件的产品表和订单表之间正常建立了关联， 所以可以充分享受关联带来的好处，例如使用表达式来统计和引用数据，在代码中使用 GetParentRow 和 GetChildRows 方法快捷定位或筛选出关联行。

由于在建立这个关联的时候，关联表的生成模式选择了“无”，不会像常规那样自动创建和显示关联表，所以此关联的存在，对于性能几乎没有影响。

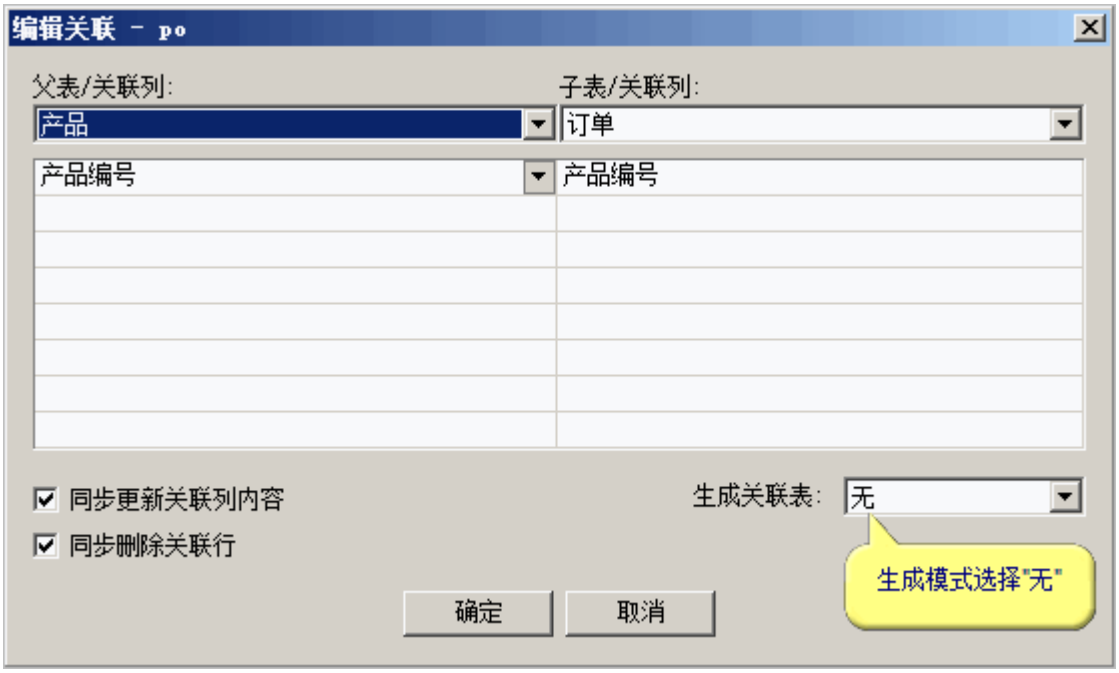
但是我们用窗口和 Table 控件模拟了一个关联表，打开此窗口后，如果在产品表选择某产品，窗口中的 Table 控件可以自动显示出该产品的所有订单，表现得就像真的关联表一样。

由于这个“关联表”是模拟的，平时并不存在，打开窗口的时候出现，关闭窗口的时候自动消失，所以这个“关联表”对于性能的影响极微。



设计步骤：

1、在产品表和订单表之间建立关联，关联名为"po"，注意关联表生成模式要选择为“无”：



2、在产品表新建一个窗口，窗口类型为停靠，停靠位置为底端。

3、插入一个 Table 控件，绑定到订单表，将其“作为副本”属性设置为 True，“允许编辑”属性设置为 True，“停靠”属性设为 Fill。

4、窗口的 AfterLoad 事件设为：

```
Dim t As Table = Tables("窗口 1_Table1")
With Tables("产品")
    If .Current Is Nothing Then
        t.Filter = "False"
    Else
        t.Filter = "产品编号 = " & .Current("产品编号")
    End IF
End With
```

此段代码确保打开窗口后，Tables("窗口 1_Table1")只显示当前选定产品的订单。

5、关闭窗口设计器，回到产品表，打开表属性设置窗口，将其 CurrentChanged 事件代码设为：

```
If Forms("窗口 1").Opened() '一定要判断用于模拟关联表的窗口是否已经打开
    Dim t As Table = Tables("窗口 1_Table1")
    With Tables("产品")
        If .Current Is Nothing Then
            t.Filter = "False"
        Else
            t.Filter = "产品编号 = " & .Current("产品编号")
        End IF
    End With
End If
```

当我们在产品表选择不同的产品时，上述代码会判断窗口是否已经打开， 如果已经打开，则更新Tables("窗口 1_Table1")的 Filter 属性，使其显示新选定产品的订单。

6、将订单表的 DataRowAdding 事件代码设置为：

```
If CurrentTable.Name = "窗口1_Table1" Then '如果当前表是模拟的关联表
    Dim r As Row = Tables("产品").Current
    If r IsNot Nothing Then
        e.DataRow("产品编号") = r("产品编号")
    End If
End If
```

当我们选择这个模拟的关联表，然后单击菜单中的增加行按钮时，上述代码会检查产品表中是否有选定行，如果有则新增订单自动继承此选定产品的产品编号。

如果没有这段代码，在模拟的关联表增加行时，新增行是看不见的，因为新增行的产品编号为空，而模拟的关联表只显示产品编号为选定产品的订单。

至此，我们用窗口的Table控件模拟出来的关联表，和真正的关联表一样，能够动态显示选定产品的订单，新增行也能自动继承选定产品的产品编号，不同的只是这个“关联表”的名称是Tables("窗口1_Table1")，而不是Tables("产品.订单")。

7、在产品表增加两个表达式列，分别为总销售量和总销售金额，总销售量的表达式：Sum(Child(po).数量)，总销售金额的表达式为：Sum(Child(po).金额)

8、将订单表的 DataColChanged 事件代码设置为：

```
Select Case e.DataCol.Name
    Case "产品编号"
        Dim pr As DataRow = e.DataRow.GetParentRow("产品")
        If pr IsNot Nothing Then
            e.DataRow("单价") = pr("单价")
        End If
End Select
```

这样在订单表输入产品编号，会自动从产品表中找出对应的产品，然后取其单价填入到订单表的

单价列中。

9、将产品表的 DataColChanged 事件代码设置为：

```
If e.DataCol.Name = "单价" Then '更新未确认订单的单价
    Dim drs As List(of DataRow)
    drs = e.DataRow.GetChildRows("订单")
    For Each dr As DataRow In drs
        If dr("确认") = False Then
            dr("单价") = e.DataRow("单价")
        End If
    Next
End If
```

这样在产品表修改某产品的单价，系统会找出订购此产品的全部订单，然后并将未确认订单的单价更新为新的单价。

一点提示：

我们没有必要将关联表视为洪水猛兽，只有确实因为关联表过多，对性能有严重影响时，才可以考虑编码模拟关联表。

4.26.32 高级话题

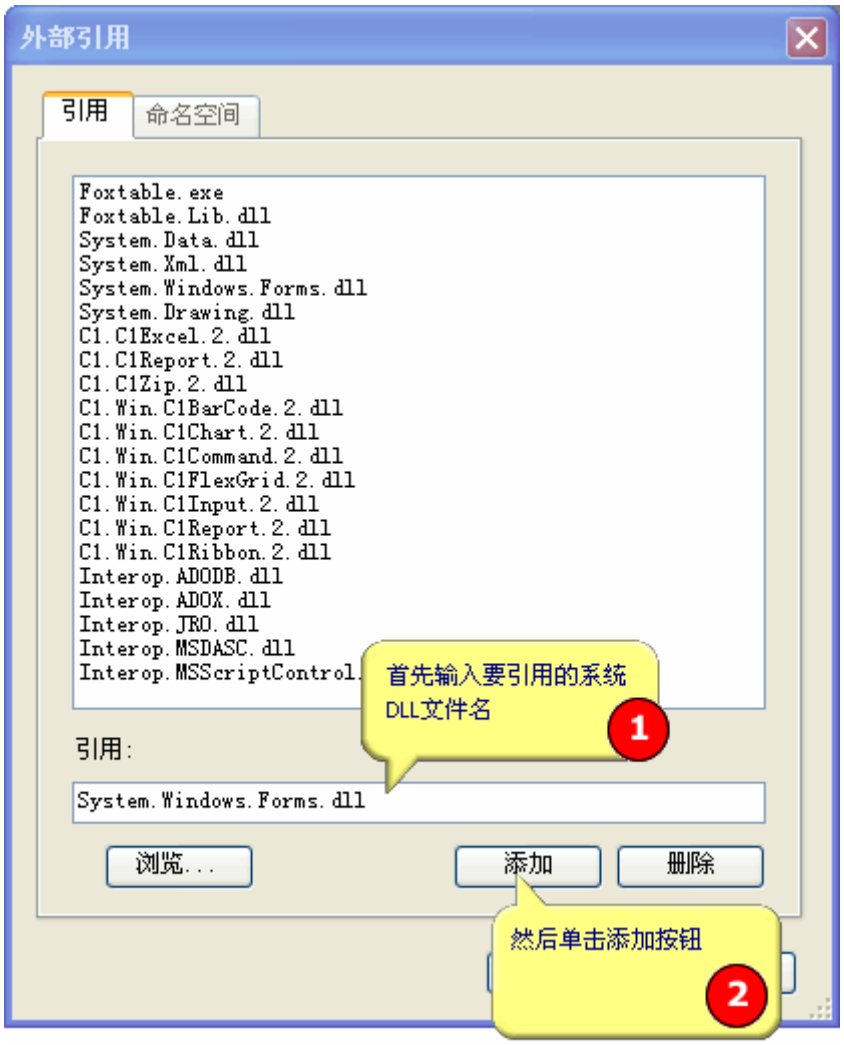
4.26.32.1 引用托管 DLL 文件

本节内容是针对专业程序员的，普通用户可以忽略。

1、在菜单的“管理项目”功能区，单击下面的按钮：

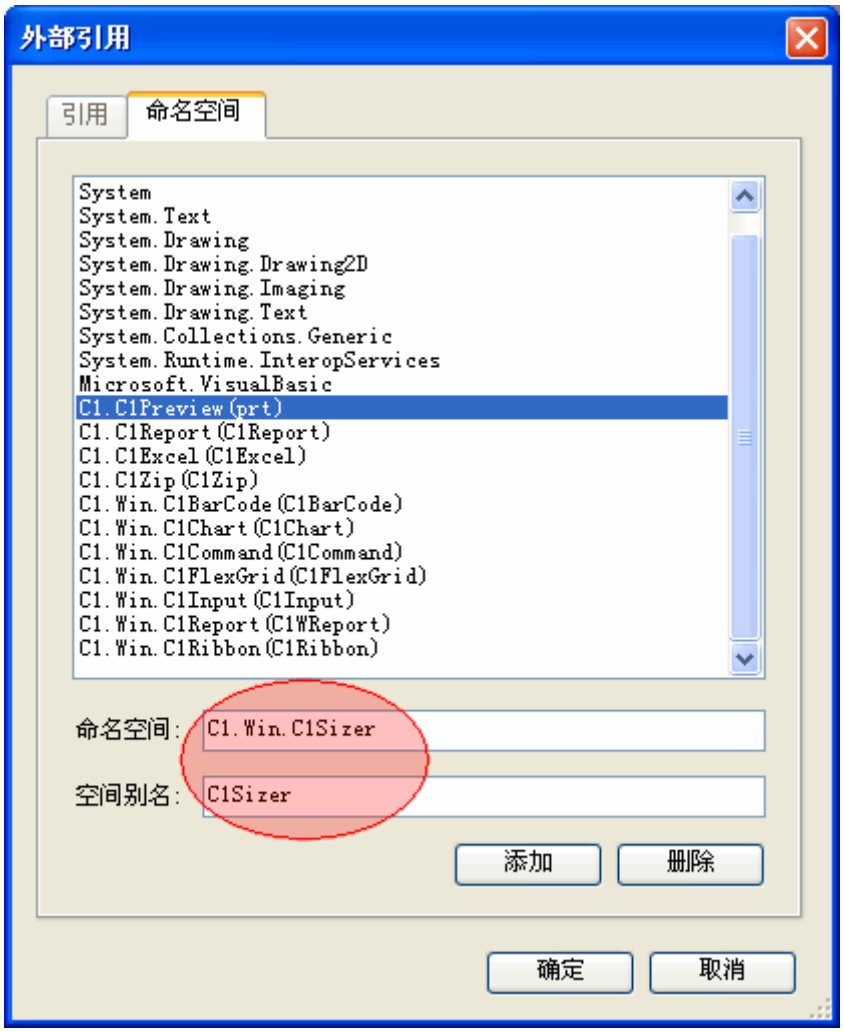


2、如果是.net 2.0 系统自带的 DLL 文件，无需复制到 foxtable 的安装目录，直接手工输入 DLL 文件名，然后再单击“添加”按钮：



3、如果是第三方的 DLL 文件，需要先复制到 Foxtable 目录下，然后单击“浏览”按钮，选择要引用的 DLL 文件，最后单击“添加”按钮。

4、可以添加一个命名空间，并为其指定一个别名，这样代码可以简洁一些：



有别于其它开发工具，在 Foxtable 添加命名空间的时候，必须指定别名。

4.26.32.2 定义 API 函数

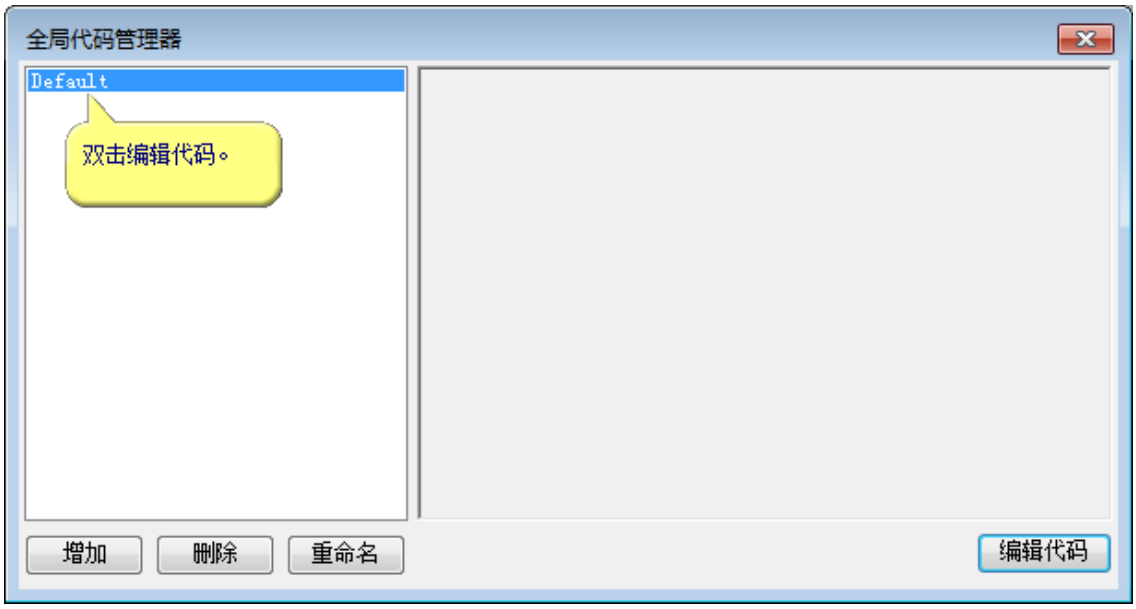
本节内容针对专业程序员，一般用户可以忽略。

可以在全局代码中定义 API 函数

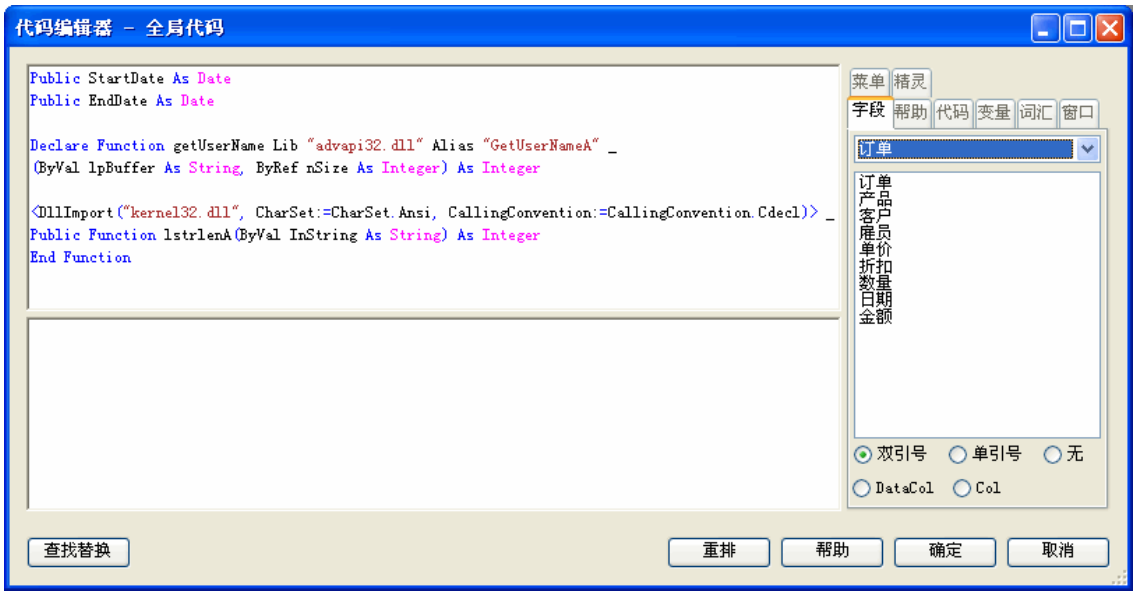
1、在“管理项目”功能区单击“全局代码”按钮。



2、全局代码可以有多个模块，默认已经有一个名为"Default"的模块，双击之即可编写代码。



3、下图的全局代码定义两个 Public 变量，两个 API 函数：



4、如果不是系统 DLL 文件，而是第三方的 DLL 文件，请事先将其复制到 foxtable 目录下。

5、可以在任何地方使用定义好的 API 函数，例如：

```
Dim n As Integer = lstrlenA("ABC")
```

4.26.32.3 定义过程

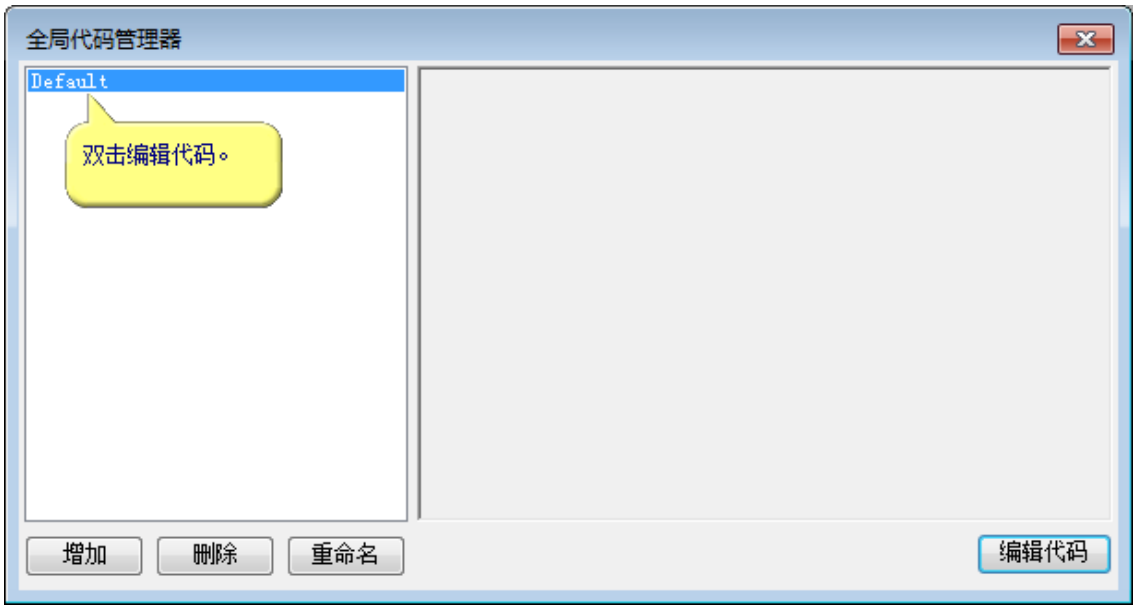
本节内容针对专业程序员，一般用户可以忽略。

如果你系统学习过 Visual Basic.net，可以在全局代码中直接用 Visual Basic 的语法规则定义 Sub 或者 Function 过程。

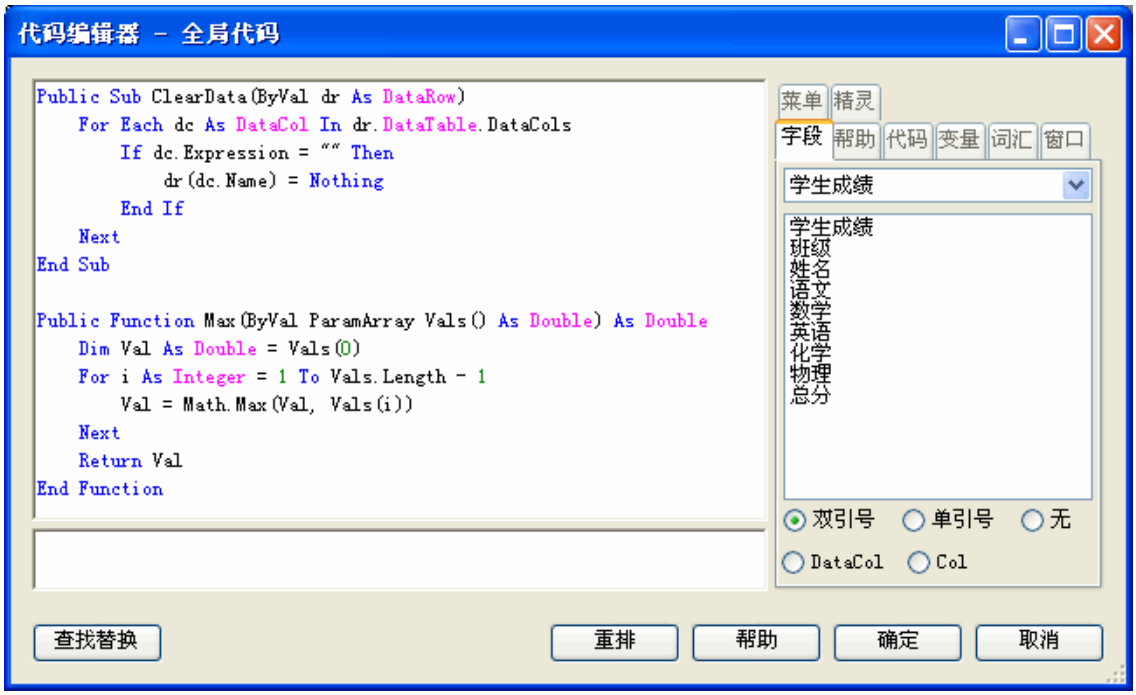
1、在“管理项目”功能区单击“全局代码”按钮。



2、全局代码可以有多个模块，默认已经有一个名为"Default"的模块，双击之即可编写代码。



3、例如下图定义了两个过程：



4、你可以在任何位置的代码中，使用上述定义好的过程：

```
Dim v As Double
v = Max(1.1, 1.2, 1.3)
ClearData(CurrentTable.Current.Datarow)
```

4.26.32.4 第三方开发接口

本节内容针对专业程序员，一般用户可以忽略。
DataTable 有一个 BaseTable 属性，返回底层的 System.Data.DataTable 对象 。
DataRow 有一个 BaseRow 属性，返回底层的 System.Data.Row 对象。
我们知道，通过常规的代码， 我们是没有办法修改_Identify、_SortKey 等系统列的内容。
但是通过上面两个属性，我们直接修改系统列的内容。
例如某个表启用了插入行的功能，如果在某位置插入了太多的行，最后该位置可能再也无法正常插入行，我们可以用下面的代码，重置_SortKey 列的内容：

```
Dim drs As List(of DataRow) = DataTables("表A").Select("","[_SortKey]")
For i As Integer = 0 To drs.Count - 1
    drs(i).BaseRow("_SortKey") = i
Next
```

执行上述代码后，插入行的功能即可恢复正常。

Table 有一个 Grid 属性，返回底层的 C1.Win.C1FlexGrid.C1FlexGrid 对象。
所有的控件都有一个 BaseControl 属性，返回其对应的 System.Windows.Forms.Control 对象
窗口有一个 Panel 对象，返回一个 Foxtable.FormPanel 容器，窗口所有的控件都在这个容器中
对于模式窗口和独立窗口，可以通过窗口的 BaseForm 属性，返回对应的 Windows.Forms.Form 对象

例如希望消除某个独立或模式窗口的边框，可以在窗口的 AfterLoad 事件中加入代码：

```
e.Form.BaseForm.FormBorderStyle = Windows.Forms.FormBorderStyle.None
```

在窗口添加第三方控件的时候，不能直接添加到 BaseForm 中，必须添加到 Panel 中。
例如需要在窗口中加入一个 RichTextBox 控件，可以在窗口的 AfterLoad 事件中加入代码：

```
Dim ct As New Windows.Forms.RichTextBox
ct.Dock = Windows.Forms.DockStyle.Fill
e.Form.Panel.Controls.Add(ct)
```

如果已经添加了第三方控件，建议在窗口的 BeforeClose 事件中，将这些控件移除并销毁。

如果要使用第三方的开发工具，请引用 foxtable.exe 和 foxtable.lib.dll，这两个文件包括 foxtable 所有公开的类型。
窗口中的 Table 可以直接绑定到 System.Data.DataTable 类型，例如：

```
Dim dt As New System.Data.DataTable
dt.Columns.Add("姓名",GetType(String))
dt.Columns.Add("出生日期",GetType(Date))
dt.Columns.Add("薪水",GetType(Double))
Tables("窗口 1_Table1").DataSource = dt
```

4.26.32.5 为第三方控件定义事件过程

本节内容针对专业程序员，一般用户可以忽略。
下面用一个简单的例子，说明如何添加第三方控件，并为其定义事件过程。
以.Net Framework 2.0 中的 Button 控件为例。
1、在全局代码中加入用于处理 Button 控件的 Click 事件的过程：

```
Public Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
   MessageBox.Show("这是自定义事件!")
End Sub
```

提示：过程必须是Public的，否则无法调用。

2、新建一个窗口，在窗口的AfterLoad事件加入代码：

```
Dim ct As New Windows.Forms.Button
ct.Text = "我的按钮"
e.Form.Panel.Controls.Add(ct)
AddHandler ct.Click, AddressOf Button1_Click
```

3、现在打开窗口，单击按钮，可以发现执行了我们在全局代码中为此按钮定义的Click事件代码。

提示：

为第三方控件指定事件过程的语句是AddHandler，此语句的语法为：

```
AddHandler 控件.事件, AddressOf 过程名
```

4.26.32.6 选择 B/S 还是 C/S

一些用户对于 B/S 架构有所偏爱，因为 B/S 易于部署和维护，这种优势是 C/S 所不能企及的。
但是 B/S 也有劣势：
1、功能弱，B/S 很难实现个性化的需求，即使最顶级的 B/S 软件，这方面也无法和普通的 C/S 软件相提并论。
2、性能差，这是因为 B/S 的大部分工作是由服务器承担的，无法充分利用客户端的处理能力。
3、响应慢，因为 B/S 网络传送的有效数据少，很大一部分是 HTML 标志，网络带宽利用率低。
所以一般面向公众的数据采集和发布，用 B/S 是有优势的，如果你要设计一套性能优越，用户体验好的管理软件，那么 C/S 是更好的选择，要知道客户对于管理软件的要求，通常是五花八门、出其不意的。

需要特别提及的是，面对 Foxtable，B/S 架构易于维护的优势也荡然无存，这是因为 Foxtable 开发的管理软件具备远程在线升级功能，开发者发布新版本后，客户端可以自动升级，甚至可以将整个管理项目自动替换为另一个。
此外 Foxtable 没有服务端，服务器只是作为数据存取之用，所有工作都在客户端完成，同时除了

存取数据，Foxtable 在其余时间是和服务器断开的，而且传输的都是纯粹数据，无任何冗余；所以同样配置的服务器，同样的带宽，用 FoxTable 开发的管理软件，其负载能力将是普通 B/S 管理软件的 10 倍以上。
而且 FoxTable 内置短信收发功能，可以通过短信收集、查询和审核数据。例如用户按照固定格式发个请求短信，系统受到这个短信后，将查询结果转换成一个网页，并自动将这个网页的地址发送到用户手机上，用户直接在手机上打开该网页地址就能查看查询结果了。

如果你还在 B/S 和 C/S 之间犹豫，相信上述内容已经给了你明确的答案。

5. 专业报表

5.1 关于专业报表

到目前为止 Foxtable 提供了三种报表输出方法：
1、Excel 报表
2、Word 报表
3、窗口打印
Excel 报表功能非常强大，几乎可以实现传统报表的所有功能，甚至一些传统报表无法解决的问题，Excel 报表都很可以轻松实现，更重要的是，Excel 报表的使用很简单，人人都可以掌握，设计效率也远高于传统报表；可以说 Excel 报表是 Foxtable 的主力报表输出工具。
Word 报表在复杂版面的设计方面具有无可比拟的优势和灵活性，但是数据调用不如 Excel 报表灵活多样，所以 Word 报表适合快速设计单记录且格式较为复杂的报表。
窗口打印可以媲美专业的票据打印工具，其直观的可视化设计、精确的对象定位，及特有的数据绑定，使得其特别适合于票据的输出，例如支票套打。
即将介绍的是专业报表，既然是专业报表，肯定不是给普通用户使用的，普通用户请跳过专业报表的全部内容。
Excel 报表和窗口打印已经足够满足我们日常的需要，请您好好掌握，除非你有特别的需要，否则没必要去研究专业报表。
专业报表是一个功能非常强大的报表开发工具，如果能够掌握的话，我们设计出的报表，可以达到很多专业报表开发软件的水准。
正因为“专业”，所以内容繁多，而且完全依靠用户编码完成，初学者需要长时间的学习和大量的练习，才能最终做到灵活应用。
如果你决心要掌握专业报表，那么我告诉你：别怕，除了数据绑定,专业报表大部分内容的逻辑都不复杂，学习难度不大，只是繁琐而以，只要你有时间和耐心，就一定能够掌握，而且对于专业报表来说，数据绑定也并非一个必须掌握的内容。
请先打开 CaseStudy 目录下的示例文件“专业报表.table”，才能正常编译执行本课程的所有示例代码。

5.2 报表基础

5.2.1 一行文本

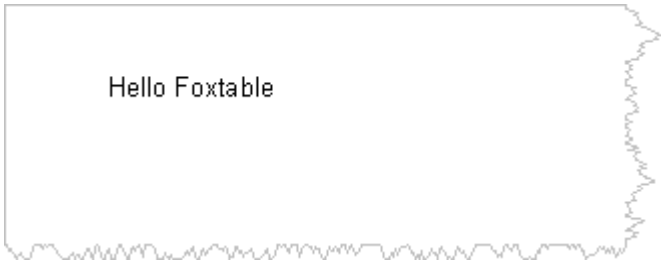
将以下代码复制到命令窗口执行：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

报表的类型为 PrintDoc，报表设计就是定义一个 PrintDoc 对象，然后向其中加入各种打印对象。Foxtable 提供了几十种打印对象，例如上图的 RenderText 是用于打印文本的，所有的打印对象都位于 prt 空间中，所以定义的时候需要加上前缀 prt，例如：

```
Dim rt As New prt.RenderText '定义一个文本对象
```

示例代码的打印结果：



5.2.2 样式设置

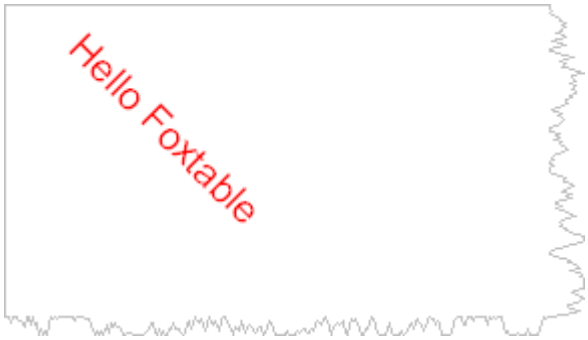
5.2.2.1 样式设置

样式用来设置打印对象的外观，通过对象的 Style 属性，即可对样式进行设置。

在命令窗口执行下面的代码，注意加粗的两行：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Style.TextColor = Color.Red '字符颜色设为红色
rt.Style.TextAngle = -45 '顺时钟旋转 45 度
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

这是打印结果：



Style 属性包括很多子属性，下表是其中最常用的，接下来我们会结合实际，逐个介绍这些属性的应用。

属性名	说明
BackColor	背景颜色
Borders	设置边框
CharSpacing	设置字符间距
Font	字体
TextAlignHorz	设置文本的水平对齐方式
TextAlignVert	设置文本的垂直对齐方式
TextAngle	设置文本旋转角度，负值为顺时钟，正值为逆时针。
TextColor	设置文本颜色
TextIndent	设置首行缩进幅度
WordWrap	文本长度超出对象宽度时，是否自动换行。
LineSpacing	设置行距，以百分比表示，例如 100 为单倍行距，150 为 1.5 倍行距
Padding	设置对象内容距离对象边界的距离
Spacing	设置对象和相邻对象的间隔距离

5.2.2.2 对象边框

Style 有一个子属性 Borders，用于设置边框类型，Borders 本身也包括四个子属性：
All:　统一设置边框
Top：　上边框
Bottom: 下边框
Left: ：左边框
Right: 右边框
边框是一个 LineDef 类型，创建一个 Linedef 的语法是：
New prt.Linedef()
New prt.Linedef(Color)

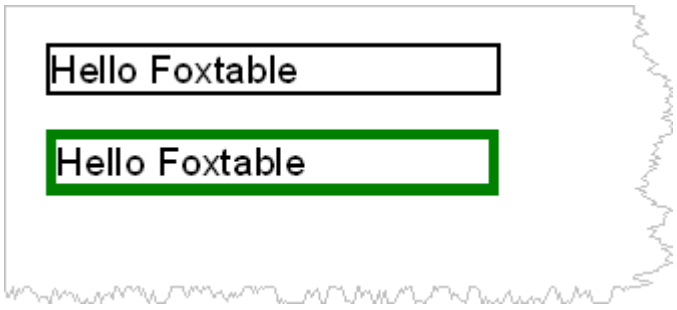
New prt.Linedef(Width, Color)
New prt.Linedef(Width, Color, DashStyle)
Color:　边框颜色。
Width:　边框宽度，单位为毫米。
DashStyle：DashStyle 型枚举，包括以下可选值：
Dash　由线段构成的直线。
DashDot　由线段和点间隔构成的直线。
DashDotDot　由线段和连续两个点间隔构成的直线
Dot　由点构成的直线。
Solid　实线，这是默认值。

示例一

在命令窗口执行下面的代码，注意加粗的两行：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = 40 '对象宽度为 40 毫米
rt.Style.Borders.All = New prt.Linedef
rt.Style.Spacing.Bottom = 3 '和下一个对象保持 3 毫米的距离
doc.Body.Children.Add(rt) '将文本对象加入到报表
'插入另一个文本框
rt = New prt.RenderText '再次新建一个文本框
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = 40 '对象宽度为 40 毫米
rt.Style.Borders.All = New prt.Linedef(1, Color.Green)
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：

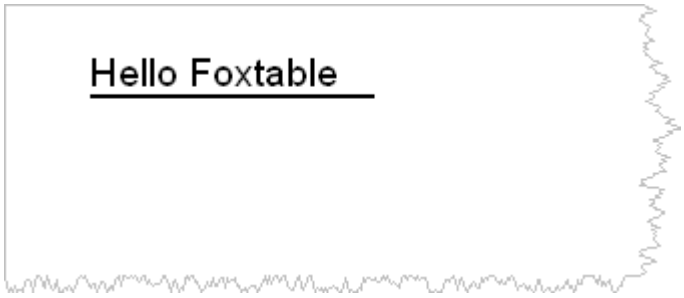


示例二

可以单独设置某一个方向的边框，例如：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = 25 '宽度为 25 毫米
rt.Style.Borders.Bottom = New prt.Linedef '设置底边框
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：

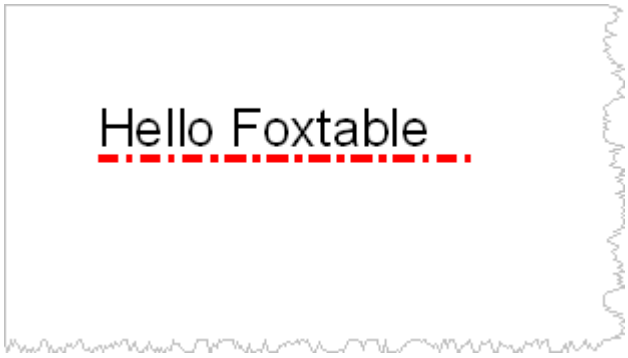


示例三

可以设置边框的类型，例如：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = 25 '宽度为 25 毫米
rt.Style.Borders.Bottom = New prt.Linedef(0.5, Color.Red, DashStyle.Dot) '设置底边框
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：



5.2.2.3 文本对齐

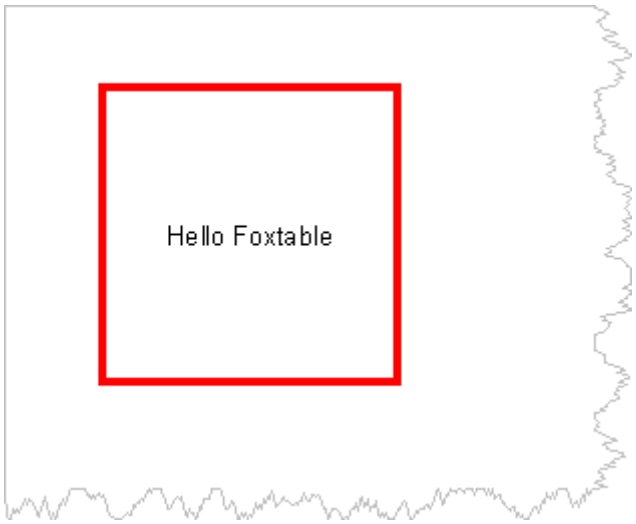
Style 有一个 TextAlignHorz 子属性，用于设置文本的水平对齐方式，
该属性是 AlignHorzEnum 型枚举，包括以下可选值：
Center: 居中
Left: 靠左
Right: 靠右
Justify：两端对齐
Style 有一个 TextAlignVert 子属性，用于设置文本的垂直对齐方式。
该属性是 AlignVertEnum 型枚举，包括以下可选值：
Bottom：靠下
Center：居中
Top： 靠上

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
```

```
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = 40 '宽度为 40 毫米
rt.Height = 40 '宽度为 40 毫米
rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：



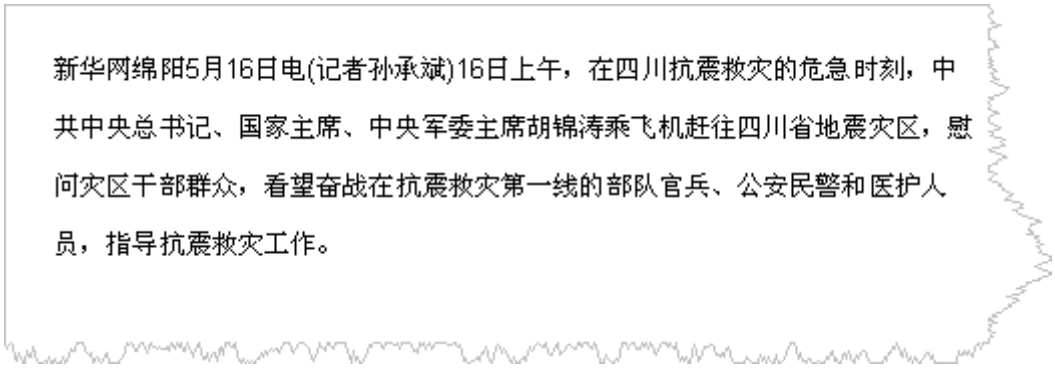
5.2.2.4 设置行距

Style 的 LineSpacing 子属性用于设置行距，以百分比表示，例如 100 为单倍行距，150 为 1.5 倍行距。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
Dim s As String
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
rt.Text = s '设置文本对象的内容
rt.Style.LineSpacing = 200 '设置双倍行距
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：



5.2.2.5 字符间距

Style 的 CharSpacing 子属性用于设置字符间距，单位为毫米。
例如：

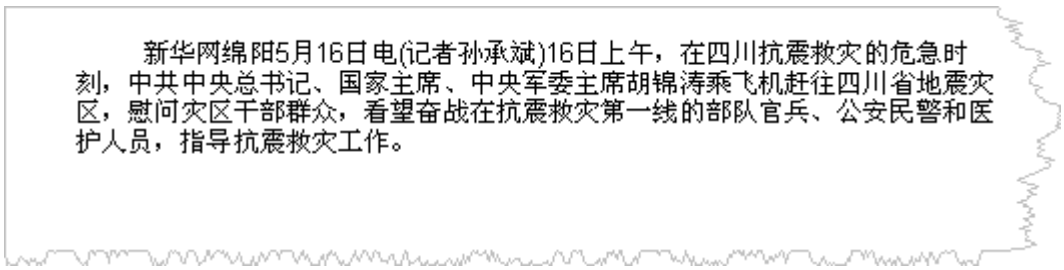
```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Style.CharSpacing = 2 '字符间距为 2 毫米
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

5.2.2.6 首行缩进

Style 有一个子属性 TextIndent，用于设置首行缩进幅度，单位为毫米。
示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
Dim s As String
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
rt.Text = s '设置文本对象的内容
rt.Style.TextIndent = 9 '首行缩进 9 毫米
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

下图是执行效果，首行已经缩进 9 毫米：



5.2.2.7 对象间隔

Spacing 是 Style 的子属性，用于设置和相邻对象的间隔，单位为毫米，Spacing 自己也包括四个子属性：

All:　全部方向的间隔

Top:　上间隔

Bottom:　下间隔

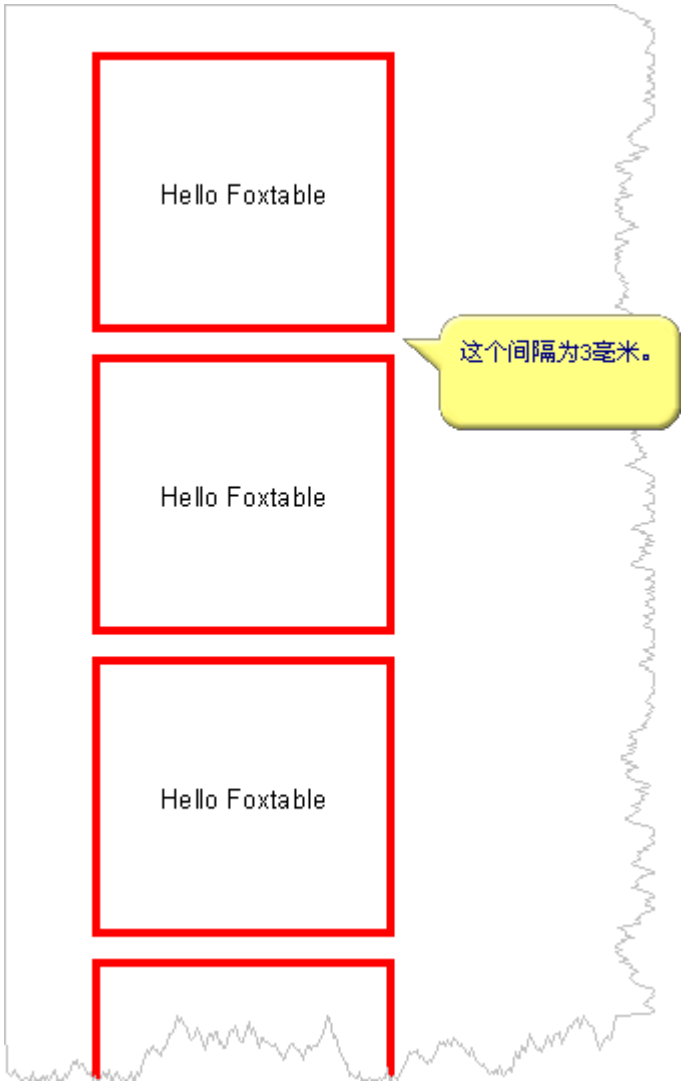
Left:　：左间隔

Right:　右间隔

示例：

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
For i AS integer = 0 To 10
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable" '设置文本对象的内容
    rt.Width = 40 '宽度为 40 毫米
    rt.Height = 40 '宽度为 40 毫米
    rt.Style.Spacing.Bottom = 3 '设置下间隔为 3 毫米
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

执行结果：



请删除下面这一行：

```
rt.Style.Spacing.Bottom = 3 '设置下间隔为 3 毫米
```

然后重新执行，你会发现所有对象紧挨在一起。

5.2.2.8 边距设置

Padding 属性和 Spacing 有点类似，Spacing 用于设置和其他对象之间的间隔，而 Padding 用于设置对象内容和对象边界之间的距离，通常称之为边距。

Padding 包括四个子属性：

All: 全部方向的边距

Top: 上边距

Bottom: 下边距

Left: ： 左边距

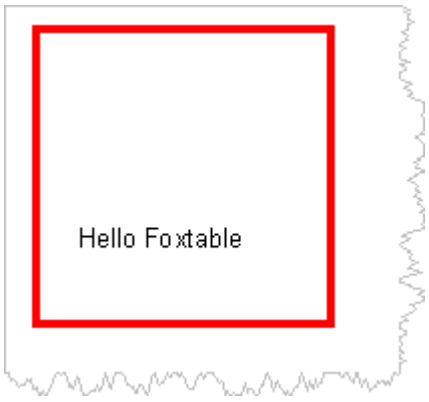
Right: 右边距

示例一

```
Dim doc As New PrintDoc '定义一个报表
```

```
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = 40 '宽度为 40 毫米
rt.Height = 40 '宽度为 40 毫米
rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
rt.Style.Padding.Top = 25 '上边距 25 毫米
rt.Style.Padding.Left = 5 '左边距 5 毫米
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：



实例二

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
For i AS integer = 0 To 10
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable " & i '设置文本对象的内容
    rt.Width = "Auto" '自动设置宽度
    rt.Style.Padding.All = 2 '所有方向的边距都设为 2
    rt.Style.Spacing.All = 2
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

执行效果：



你可以删除这一行，看看执行效果有什么不同：

```
rt.Style.Padding.All = 2 '所有方向的边距都设为 2
```

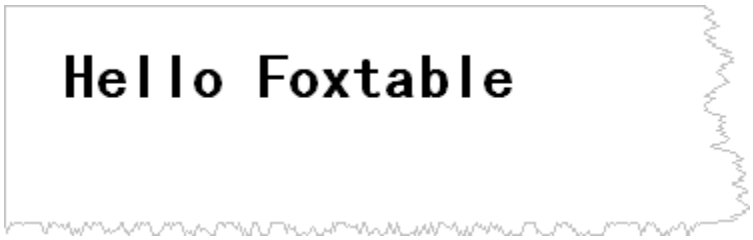
5.2.2.9 字体设置

Style 有一个子属性 Font，用于设置对象字体。

示例一

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
rt.Style.Font = New Font("宋体", 16, FontStyle.Bold) '设置字体
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：



字体设置是报表设计中一个经常性的工作，为了减轻用户的编码工作量，我们特地为 Style 属性提供了以下子属性，用于快捷地设置字体

属性名称	属性说明
FontName	字体名称

FontSize	字体大小
FontBold	逻辑型，字体是否加粗
FontItalic	逻辑型，字体是否倾斜
FontStrikeout	逻辑型，是否有删除线
FontUnderline	逻辑型，是否有下划线

示例二

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Style.FontBold = True '字体加粗
rt.Style.FontSize = 16 '字体大小为 16 磅
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

5.2.3 排列方式

你也许已经留意到，在之前的例子中，对象都是从上到下排列的，能不能从左到右，或者先从左到右，到达右边界之后，再另起一行？
答案是肯定的，否则这个报表工具也太“笨”了。
所有的容器(包括报表)都有一个 Stacking 属性，可以用来改变这种默认的排列方式，该属性为StackingRulesEnum 型枚举，共有三个可选值：

- BlockTopToBottom
  从上到下排列，当到达下边界的时候，在垂直方向增加一页，这是默认值。

- BlockLeftToRight
  从左到右排列，当到达右边界的时候，在水平方向增加一页。

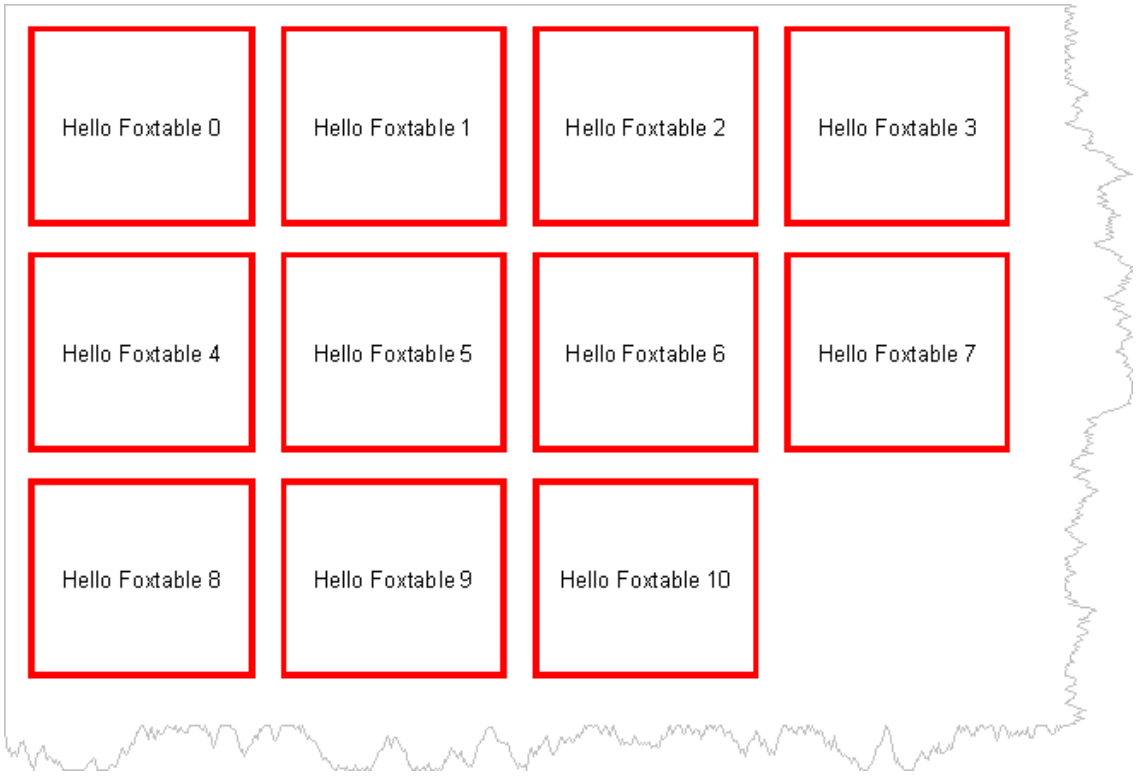
- InlineLeftToRight
  从左到右排列，当到达右边界的时候，另起一行继续排列，如果到达下边界，在垂直方向增加一页。

示例：

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight '设置排列方式
For i AS integer = 0 To 10
    rt = New prt.RenderText() '创建文本对象
```

```
    rt.Text = "Hello Foxtable " & i '设置文本对象的内容
    rt.Width = 39 '宽度为 39 毫米
    rt.Height = 35 '宽度为 35 毫米
    rt.Style.Spacing.All = 2 '设置各个方向的间隔
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

执行结果：



5.2.4 默认尺寸

在之前的代码中，我们都明确指定了对象的尺寸，如果我们不指定对象的尺寸，那么会出现什么样的情况呢？

请在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Style.Borders.All = New prt.Linedef(0.3, Color.Red) '设置边框
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

这是执行效果：



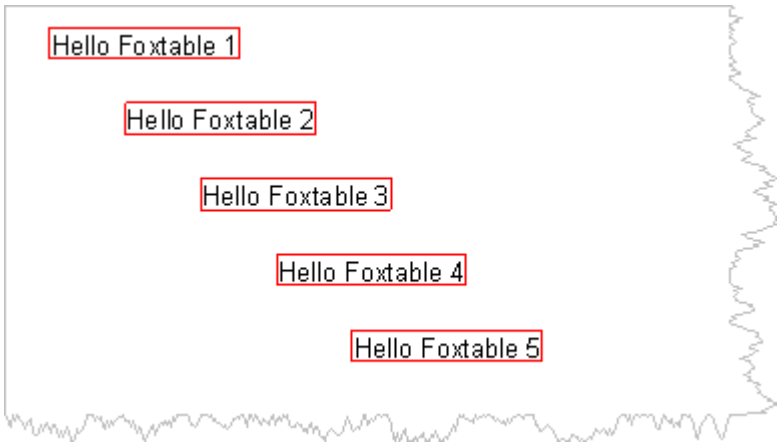
可以看到如果没有明确指定对象的尺寸，那么对象的高度会自动根据内容来确定，而对象的宽度等于容器的宽度。
这种默认的特性，会给我们带来极大的便利，以后你会慢慢体会到。

5.2.5 位置与尺寸

在之前所有的例子中，对象都是按顺序加入到报表中的，只能通过调整排列方式以及相互间隔，来间接影响对象的位置。
有的时候，我们会需要精确定位，例如单据套打。
精确定位的实现很简单，先指定对象的坐标位置，然后再加入到报表中即可。
示例

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
For i AS integer = 1 To 5
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable " & i '设置文本对象的内容
    rt.Width = "Auto" '自动设置宽度
    rt.X = i * 10 + 20 '指定水平位置
    rt.y = i * 10 + 20 '指定垂直位置
    rt.Style.Borders.All = New prt.Linedef(0.3, Color.Red) '设置边框
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

执行结果：



坐标的单位是毫米，原点为左上角。
需要注意的是左边位置必须大于页边距，否则无法显示和打印。
关于页面设置，后面会有专门的章节讲述。

详解位置与尺寸

前面已经提到过，长度、宽度的单位不是数值型，其实坐标位置也不是数值型。

如果使用数值型的数据，那么单位是毫米，例如：

```
rt.Width = 25 '宽度为 25 毫米
```

你也可以使用表达式来设置，例如要根据内容自动调整宽度，可以这样设置：

```
rt.Width = "Auto" '自动设置宽度
```

你可以在表达式中使用以下对象：

对象	说明
Parent	表示对象所在的容器，通常指的就是页面的可打印区域，如果对象位于表格中，那么容器就是对象所处的单元格，此外对象 RenderArea 也是容器，我们会专门介绍它。 Parent 的属性： Parent.Width: 容器宽度 Parent.Height:容器高度
Prev	表示上一个对象。 Prev 的属性： Prev.Left: 上一个对象的左边位置 Prev.Right: 上一个对象的右边位置 Prev.Top: 上一个对象顶端位置 Prev.Bottom: 上一个对象的底边位置 Prev.Width: 上一对象的宽度 Prev.Height: 上一个对象的高度

你可以使用百分比来设置尺寸：

```
rt.Width = "20%Parent.Width" '对象宽度为页面的 20%
```

你还可以进行加减:

```
rt.Width ="40%Parent.Width + 5" '宽度为页面宽度的 40%加上 5 毫米。
```

再例如:

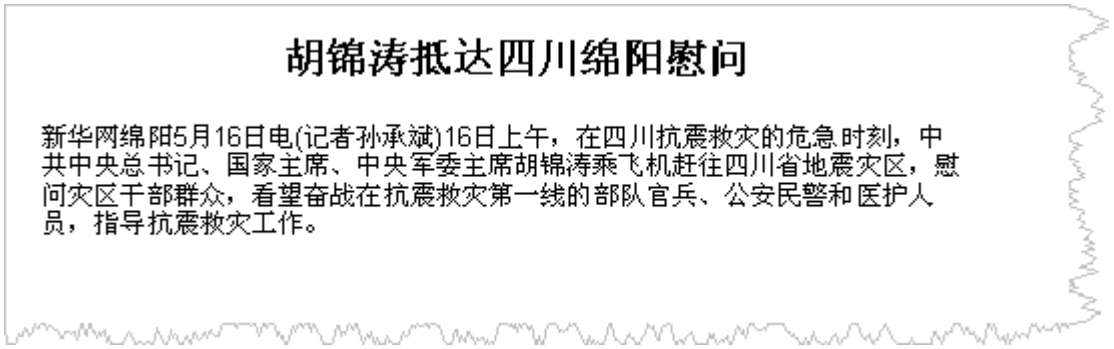
```
rt.y = "Prev.Bottom + 5" '垂直位置为上一对象的底边之下 5 毫米
```

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
Dim s As String
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
rt.Text = "胡锦涛抵达四川绵阳慰问" '设置文本对象的内容
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
rt.Style.FontBold = True '字体加粗
rt.Style.FontSize = 16 '字体大小为 16 磅
doc.Body.Children.Add(rt) '将文本对象加入到报表
rt = New prt.RenderText '创建另一个文本对象
rt.Text = s '设置文本内容
rt.y = "Prev.Bottom + 5" '设置垂直位置
doc.Body.Children.Add(rt) '将文本对象加入到报表
```

```
doc.Preview() '预览
```

执行结果：



下面的代码：

```
rt.y = "Prev.Bottom + 5" '设置垂直位置
```

等同于：

```
rt.Style.Spacing.Top = 5
```

虽然用 Spacing 属性一样可以实现，但是直接指定坐标位置，对于精确定位来说更灵活。

5.2.6 打印图片

5.2.6.1 打印图片

RenderImage 类型用于打印图片，它有一个 Image 属性，用于设置要打印的图片。

示例

```
Dim doc As New PrintDoc '定义一个报表对象
Dim rm As New prt.RenderImage '定义一个图片对象
rm.Image = GetImage("EP1.BMP") '设置图片
rm.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
doc.Body.Children.Add(rm) '将图片对象加入报表
Doc.Preview() '预览
```

执行结果：



5.2.6.2 设置图片

Style 有个 ImageAlign 子属性，用于设置图片的位置和拉伸方式。
ImageAlign 本身包括以下子属性：

属性	说明
AlignHorz	水平对齐位置，ImageAlignHorzEnum 型枚举，包括以下可选值： Center：居中 Left：　靠左 Right：　靠右
AlignVert	垂直对齐位置，ImageAlignVertEnum 型枚举，包括以下可选值： Bottom：靠底 Center：居中 Top：　靠上
StretchHorz	逻辑型，是否在水平方向拉伸，默认为 True
StretchVert	逻辑型，是否在垂直方向拉伸，默认为 True
KeepAspectRatio	逻辑型，拉伸图片时，是否保持原始纵横比例，默认为 True
TileHorz	逻辑型，图片是否在水平方向平铺，默认为 False
TileVert	逻辑型，图片是否在垂直方向平铺，默认为 False

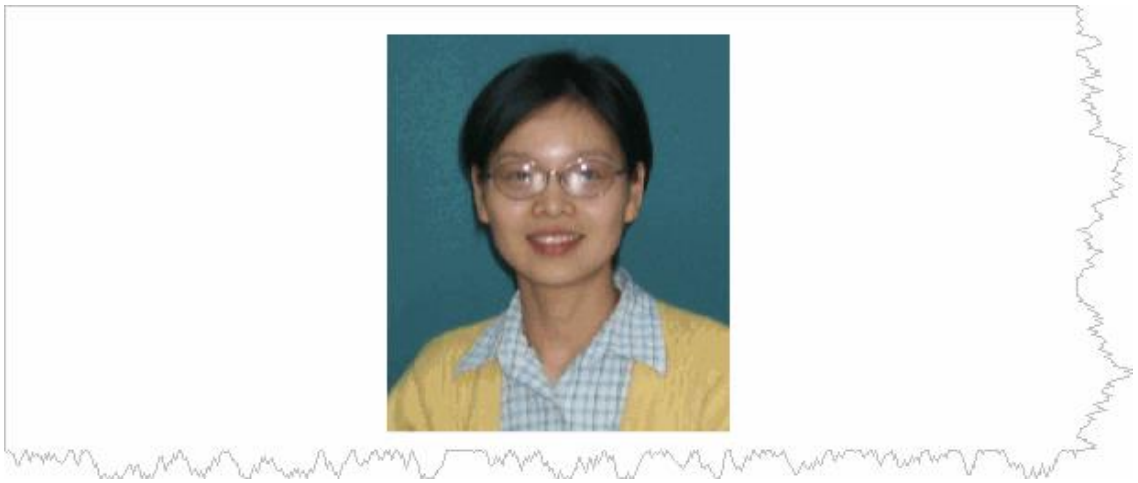
示例一

假定我们希望图片居中：

```
Dim doc As New PrintDoc '定义一个报表对象
Dim rm As New prt.RenderImage '定义一个图片对象
```

```
rm.Image = GetImage("EP1.BMP") '设置图片
rm.Width = "Parent.Width" '宽度等于页面宽度
rm.Height = "Auto" '高度由图片大小自动决定
rm.Style.ImageAlign.StretchHorz = False '禁止图片水平拉伸
rm.Style.ImageAlign.AlignHorz = prt.ImageAlignHorzEnum.Center '居中显示
doc.Body.Children.Add(rm) '将图片对象加入报表
Doc.Preview() '预览
```

执行结果：



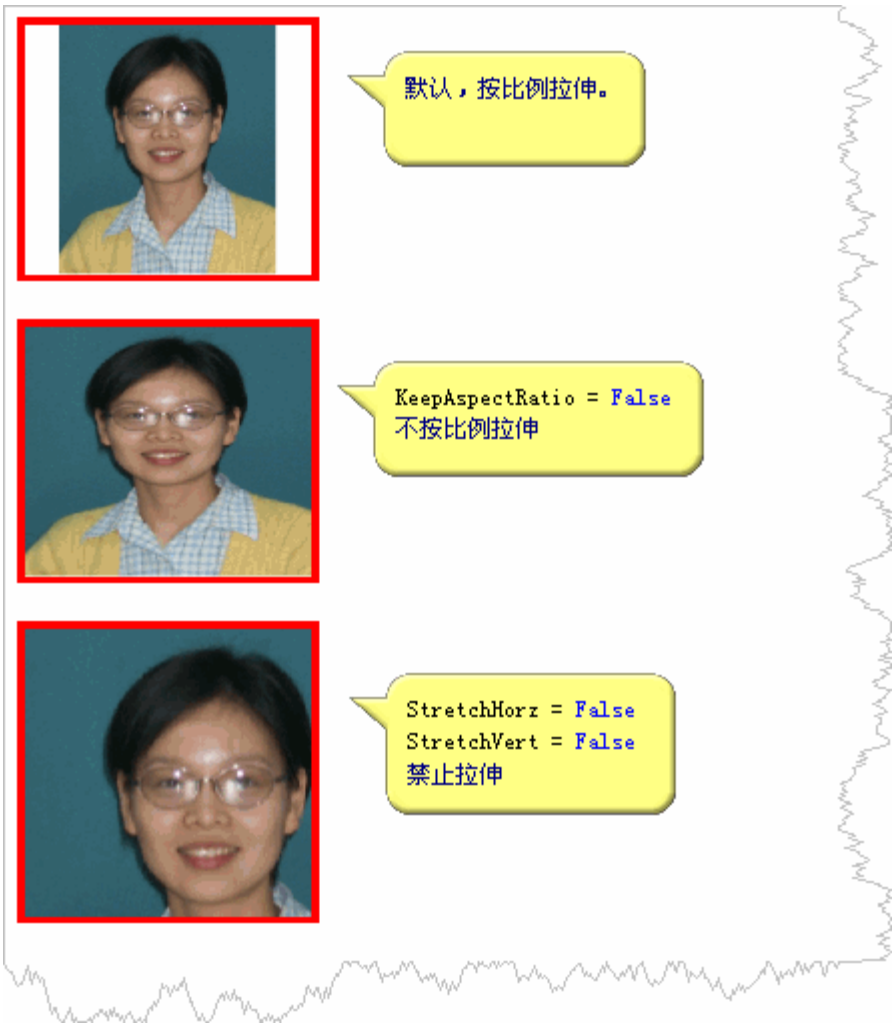
示例二

图片拉伸控制演示：

```
Dim doc As New PrintDoc '定义一个报表对象
Dim rm As New prt.RenderImage '定义一个图片对象
'默认，按比例拉伸
rm.Width = 40 '设置尺寸
rm.Height = 40
rm.Image = GetImage("EP1.BMP") '设置图片
rm.Style.ImageAlign.AlignHorz = prt.ImageAlignHorzEnum.Center '居中显示
rm.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
rm.Style.Spacing.Bottom = 5
doc.Body.Children.Add(rm) '将图片对象加入报表
'不按比例拉伸
rm = New prt.RenderImage '定义一个图片对象
rm.Width = 40 '设置尺寸
rm.Height = 40
rm.Image = GetImage("EP1.BMP") '设置图片
rm.Style.ImageAlign.KeepAspectRatio = False '取消保持原有比例
rm.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
rm.Style.Spacing.Bottom = 5
doc.Body.Children.Add(rm) '将图片对象加入报表
'禁止拉伸
rm = New prt.RenderImage '定义一个图片对象
rm.Width = 40 '设置尺寸
rm.Height = 40
```

```
rm.Image = GetImage("EP1.BMP") '设置图片
rm.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
rm.Style.ImageAlign.StretchHorz = False '禁止拉伸
rm.Style.ImageAlign.StretchVert = False
doc.Body.Children.Add(rm) '将图片对象加入报表
Doc.Preview() '预览
```

执行结果：



示例三

将图片拉伸至页面大小打印：

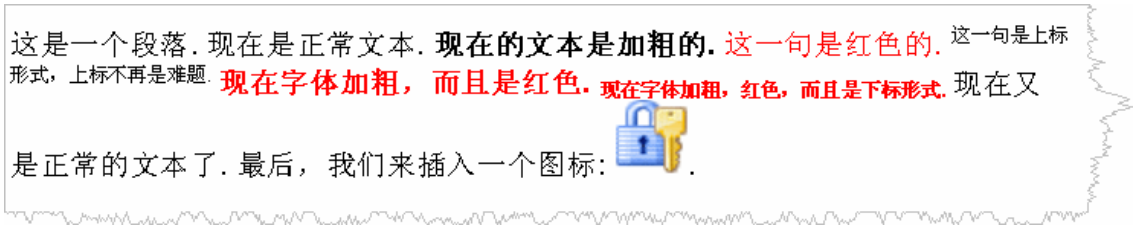
```
Dim doc As New PrintDoc '定义一个报表对象
Dim rm As New prt.RenderImage '定义一个图片对象
rm.Width = "100%Parent" '设置尺寸为容器(页面)尺寸
rm.Height = "100%Parent"
rm.Image = GetImage("EP1.BMP") '设置图片
rm.Style.ImageAlign.KeepAspectRatio = False '不保持比例
doc.Body.Children.Add(rm) '将图片对象加入报表
Doc.Preview() '预览
```

5.2.7 复杂图文

RenderText 对象用于打印文本，该对象中的文本格式是统一的，不能分别设置。
RenderParagraph 类型专门用于打印成段文本，而且段落中的文本格式可以不同，甚至可以图文混排。
我们先在命令窗口执行下面的代码，看看 RenderParagraph 的不同，然后我们再具体说明 RenderParagraph。

```
Dim doc As New PrintDoc '定义一个报表
Dim rpar As New prt.RenderParagraph() '定义一个段落对象
Dim f As New Font(rpar.Style.Font, FontStyle.Bold) '定义一个同样但加粗的字体
rpar.Content.AddText("这是一个段落. 现在是正常文本. ")
rpar.Content.AddText("现在的文本是加粗的. ", f)
rpar.Content.AddText("这一句是红色的. ", Color.Red)
rpar.Content.AddText("这一句是上标形式，上标不再是难题. ", prt.TextPositionEnum.Superscript)
rpar.Content.AddText("现在字体加粗，而且是红色. ", f, Color.Red)
rpar.Content.AddText("现在字体加粗，红色，而且是下标形式. ", f, Color.Red, prt.TextPositionEnum.Subscript)
rpar.Content.AddText("现在又是正常的文本了. ")
rpar.Content.AddText("最后，我们来插入一个图标: ")
rpar.Content.AddImage(getimage("logon.ico"))
rpar.Content.AddText(".")
doc.body.Children.add(rpar)
Doc.Preview() '预览报表
```

执行结果：



RenderParagraph 有一个属性 Content，表示 RenderParagraph 的内容集合，该属性有两个方法，分别是：

AddText: 向 RenderParagraph 中添加文本
AddImage：向 RenderParagraph 添加图片。

AddText 的语法:

```
AddText(String)
AddText(String, Color)
AddText(String, Font)
AddText(String, TextPosition)
```

AddText(String, Font, Color)
AddText(String, Color, TextPosition)
AddText(String, Font, TextPosition)
AddText(String, Font, Color, TextPosition)

参数说明：

- String
  文本内容

- Color
  文本颜色

- Font
  字体

- TextPosition
  文本位置，TextPositionEnum 型枚举，包括三个可选值：

  Normal:　正常位置
  Subscript:　下标
  Superscript:　上标

AddImage 的语法：

AddImage(Image)

参数说明：

Image：要插入的图片。

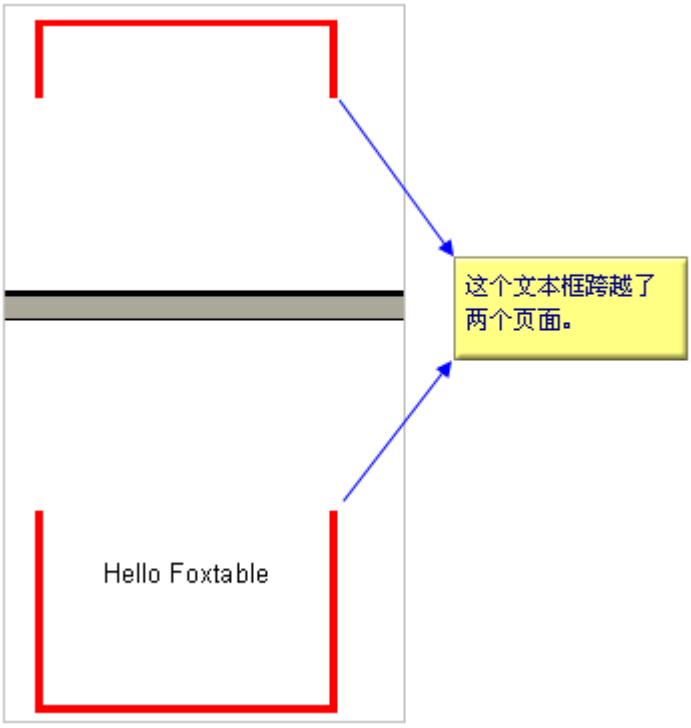
5.2.8 换页控制

首先在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
For i AS integer = 0 To 10
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable" '设置文本对象的内容
    rt.Width = 40 '宽度为 40 毫米
    rt.Height = 40 '宽度为 40 毫米
    rt.Style.Spacing.Bottom = 3 '设置下间隔为 3 毫米
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
```

```
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

下图是执行结果，我们看到有一个 RenderText 被分成了两部分，分别处在不同的页面：



为了避免这种情况发生，所有的打印对象都提供了两个属性：

SplitHorzBehavior：设置对象的水平分割方式

SplitVertBehavior：设置对象的垂直分割方式

这两个属性都是 SplitBehaviorEnum 型枚举，可选值有：

Never：　　不分割

SplitIfNeeded：　分割

现在我们只需如下修改代码，即可避免上图所出现的情况，注意加粗的一行：

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
For i AS integer = 0 To 10
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable" '设置文本对象的内容
    rt.Width = 40 '宽度为 40 毫米
    rt.Height = 40 '宽度为 40 毫米
    rt.Style.Spacing.Bottom = 3 '设置下间隔为 3 毫米
    rt.SplitVertBehavior = prt.SplitBehaviorEnum.Never '禁止被垂直分割
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
```

```
doc.Preview() '预览
```

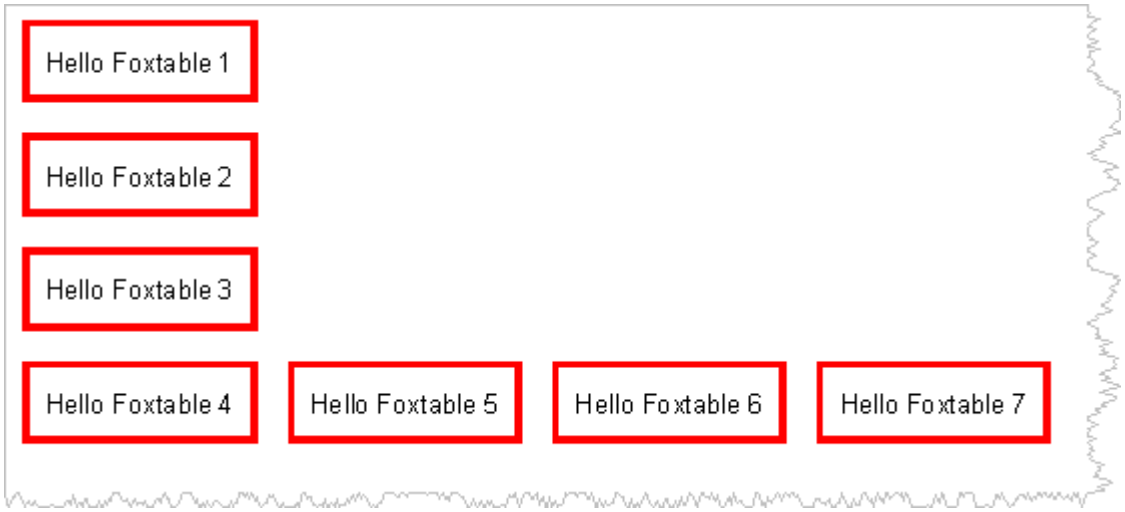
5.2.9 容器(RenderArea)

5.2.9.1 关于 RenderArea

RenderArea 是一个容器，可以用来容纳任何对象，包括 RenderArea。
RenderArea 也有 Stacking 属性，可以用来指定其子对象的排列方式。
如果一些对象需要用默认的方式排列，而另一些对象需要用一种特殊的方式排列，RenderArea 就可以派上用场了。
下面的代码，前 3 个文本对象(RenderText)按正常的排列方式（从上到下）加入到报表中，后四个被加入到一个 RenderArea 中，排列方式为从左到右：

```
Dim doc As New PrintDoc() '定义一个报表
Dim ra As New prt.RenderArea '定义一个容器
Dim rt As prt.RenderText '定义一个文本对象
For i AS integer = 1 To 3
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable " & i '设置文本对象的内容
    rt.Width = 35 '宽度为 35 毫米
    rt.Height = 15 '高度为 15 毫米
    rt.Style.Spacing.All = 2 '设置各个方向的间隔
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
ra.Stacking = prt.StackingRulesEnum.InlineLeftToRight '设置容器中对象的排列方式
For i AS integer = 4 To 7
    rt = New prt.RenderText() '创建文本对象
    rt.Text = "Hello Foxtable " & i '设置文本对象的内容
    rt.Width = 35 '宽度为 35 毫米
    rt.Height = 15 '高度为 15 毫米
    rt.Style.Spacing.All = 2 '设置各个方向的间隔
    rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
    ra.Children.Add(rt) '将文本对象加入到容器 ra 中
Next
doc.Body.Children.Add(ra) '将容器 ra 加入到报表
doc.Preview() '预览
```

执行结果：



5.2.9.2 容器与标签

一个页面中会有很多个标签，标签本身按顺序排列，但是标签中的内容，是需要精确定位的。
也就是说，页面内容的排列方式，和标签内容的排列方式是不同的。
用 RenderArea 可以轻松实现这样的要求。

下面是演示代码,每一个 RenderArea 就是一个标签，请留意 RenderArea 的子对象是如何定位的：

```
Dim doc As New PrintDoc '定义一个报表
doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight '设置排列方式
For i As integer = 1 to 100
    Dim ra As New prt.RenderArea
    Dim rt As prt.RenderText
    ra.Width = 53 '宽度 53 毫米
    ra.Height = 24 '高度 24 毫米
    ra.Style.Spacing.All = 1
    ra.Style.Borders.All = New prt.Linedef(0.3, Color.Red) '设置边框
    ra.SplitHorzBehavior = prt.SplitBehaviorEnum.Never '禁止水平分割
    ra.SplitVertBehavior = prt.SplitBehaviorEnum.Never '禁止垂直分割
    Doc.Body.ChildRen.Add(ra) '将容器加入到报表中

    rt = New prt.RenderText
    rt.Text = "姓名："
    rt.X = 2
    rt.Y = 2
    ra.Children.Add(rt) '添加到容器中

    rt = New prt.RenderText
    rt.Text = "黄财德"
    rt.X = 15
```

```
    rt.Y = 2
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "职务："
    rt.X = 2
    rt.Y = "Prev.Bottom + 2" '垂直坐标位于迁移对象之下的 2 毫米
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "总经理"
    rt.X = 15
    rt.Y = "Prev.Top" '垂直坐标和前一对象相同
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "编号："
    rt.X = 2
    rt.Y = "Prev.Bottom + 2"
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "44081919710917243"
    rt.X = 15
    rt.Y = "Prev.Top"
    ra.Children.Add(rt)
Next
Doc.Preview() '预览报表
```

执行结果：



注意一定要有下面两行代码：

```
ra.SplitHorzBehavior = prt.SplitBehaviorEnum.Never '禁止水平分割
ra.SplitVertBehavior = prt.SplitBehaviorEnum.Never '禁止垂直分割
```

否则可能会出现一个标签被分割在两页的情况。

5.2.9.3 容器的大小

如果你没有明确指定 RenderArea 的尺寸，那么 RenderArea 会自动根据子对象的位置和尺寸调整到最佳大小。
但是这时候，子对象的尺寸不能引用 RenderArea 的尺寸，否则不能正常显示。
理由很简单，因为在这种情况下，RenderArea 的尺寸由子对象决定，而子对象的尺寸又由 RenderArea 的尺寸决定，这是一个死循环。

不信可以在命令窗口执行下面的代码，你会发现，不会有任何内容显示：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
Dim ra As New prt.RenderArea
rt.Text = "Hello Foxtable" '设置文本对象的内容
rt.Width = "Parent.Width" '宽度等于容器宽度
rt.Height = "Parent.Height" '高度等于容器高度
rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
ra.Children.Add(rt) '加入到容器中
doc.Body.Children.Add(ra) '将容器加入到报表
doc.Preview() '预览
```

类似的原理适合于任何容器，包括表格。

5.2.10 页面设置

5.2.10.1 页面设置

PaperKind: 纸张类型代码
Landscape: 逻辑型，是否横向打印
Height: 设置页面高度，单位为毫米
Width: 页面宽度，单位为毫米
LeftMargin: 左边距，单位为毫米
RightMargin: 右边距，单位为毫米
TopMargin: 上边距，单位为毫米

BottomMargin: 底边距，单位为毫米
Color: 逻辑型，是否彩色打印

5.2.10.2 纸张类型

PaperKind 是一个整数，用于设置纸张类型，所有的纸张类型都有一个整数表示。
示例
例如将纸张类型改为 B5:

```
Dim doc As New PrintDoc() '定义一个报表
doc.PageSetting.PaperKind = 13 '纸张类型改为 B5
doc.Preview() '预览
```

下表列出了常用的纸张类型：

值	纸张类型
8	A3
9	A4
11	A5
12	B4
13	B5
66	A2
70	A6

纸张类型很多，全部纸张类型，请参考：纸张大全

5.2.10.3 横向打印

只需将 Landscape 属性设为 True，即可横向打印。
示例

```
Dim doc As New PrintDoc() '定义一个报表
doc.PageSetting.Landscape = True '横向打印
doc.Preview() '预览
```

5.2.10.4 自定义纸张

要使用自定义纸张，只需直接设置纸张的宽度和高度即可。
示例
例如使用一个 10 厘米宽，12 厘米高的纸张：

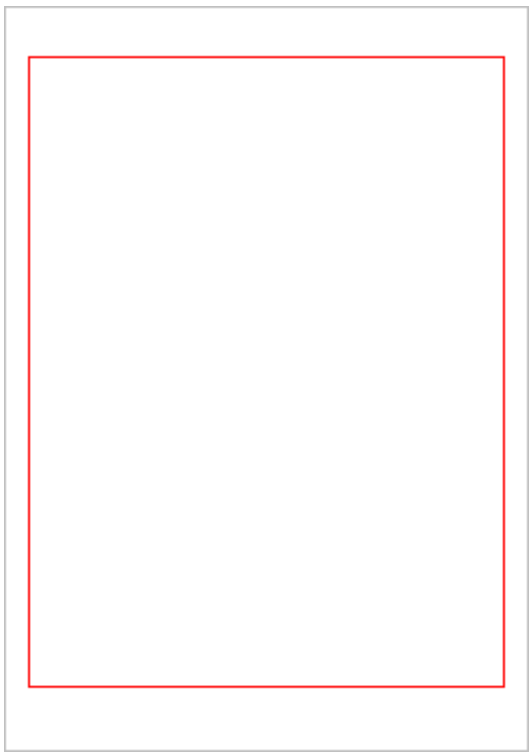
```
Dim doc As New PrintDoc() '定义一个报表
doc.PageSetting.Width = 100 '纸张宽度为 100 毫米
doc.PageSetting.Height = 120 '纸张高度为 120 毫米
doc.Preview() '预览
```

5.2.10.5 设置页边距

下面的代码，左右边距设为 1 厘米，上下边距设为 2 厘米，并在打印区域的四周绘制一个红框：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderText '定义一个文本对象
Doc.PageSetting.LeftMargin = 10 '设置左边距
Doc.PageSetting.RightMargin = 10 '设置右边距
Doc.PageSetting.TopMargin = 20 '设置上边距
Doc.PageSetting.BottomMargin = 20 '设置下边距
rt.Width = "Parent.Width" '对象宽度等于页面宽度
rt.Height = "Parent.Height" '对象高度等于页面高度
rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
doc.Body.Children.Add(rt) '将文本对象加入到报表
doc.Preview() '预览
```

执行结果：



5.2.10.6 连续进纸与自定义纸张

如果使用连续进纸的针式打印机，那么自定义纸张的时候，可能不会按照预期的尺寸进纸，还需要进行一些设置。

设置步骤：

1、在控制面板的设备与打印机窗口中，将此针式打印机设置为默认打印机。

2、单击“打印服务属性”：

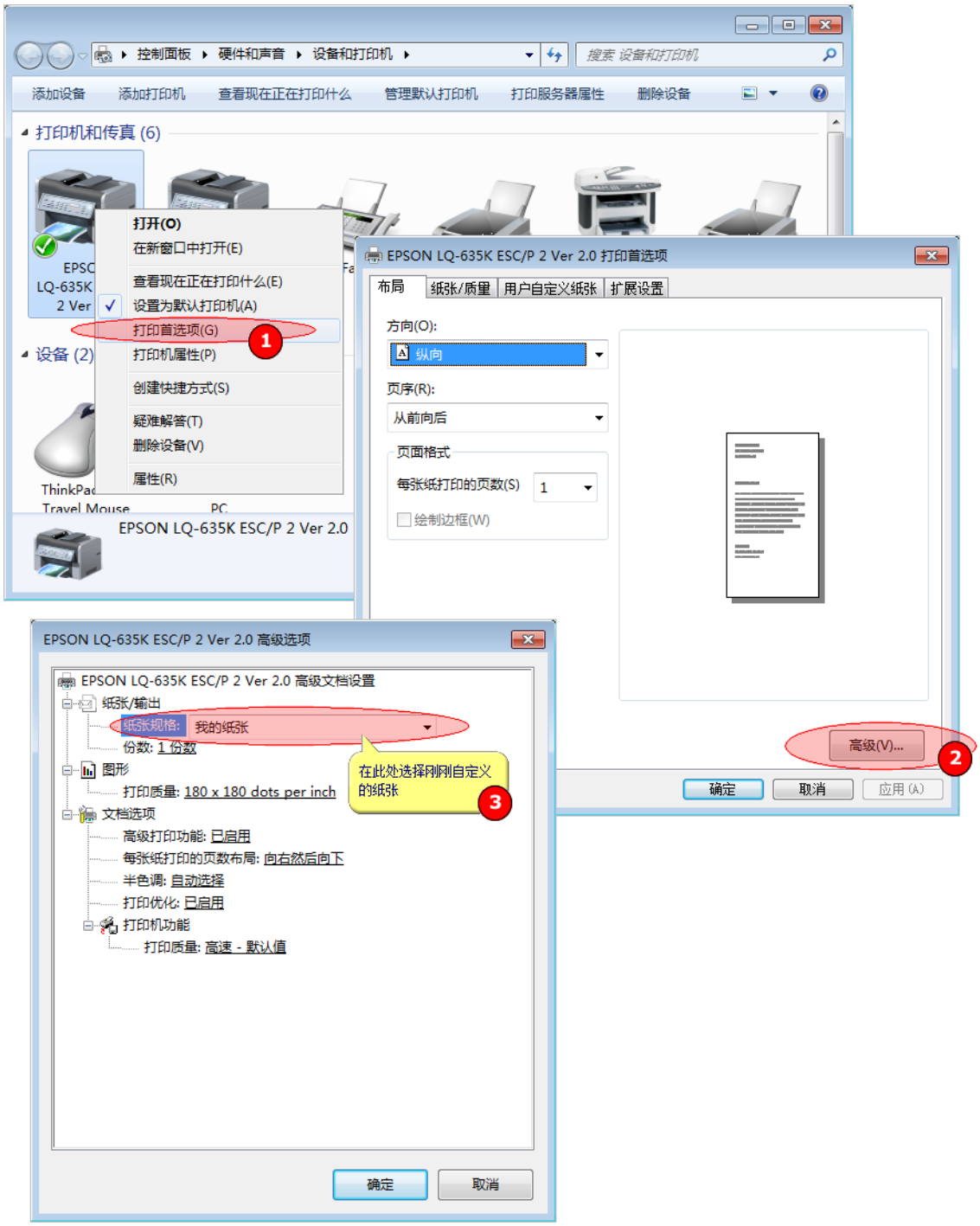


提示：如果是 Windows XP，请单击文件"菜单下的“服务器属性”命令。

3、在打印服务器属性窗口按下图所示新增一个自定义纸张：



4、右击要使用的针式打印机，从快捷菜单中选择“打印首选项”，按下图所示，将纸张规格设置为刚刚定义的纸张：



5、完成上述设置之后，我们就可以常规编写打印代码了：

```
Dim doc As New PrintDoc()
doc.PageSetting.Width = 200   '纸张宽度为200毫米
doc.PageSetting.Height = 170   '纸张高度为170毫米
doc.AutoRotate = False   '如果纸张的宽度超过高度,要加上这行代码
'添加打印内容
doc.Preview
```

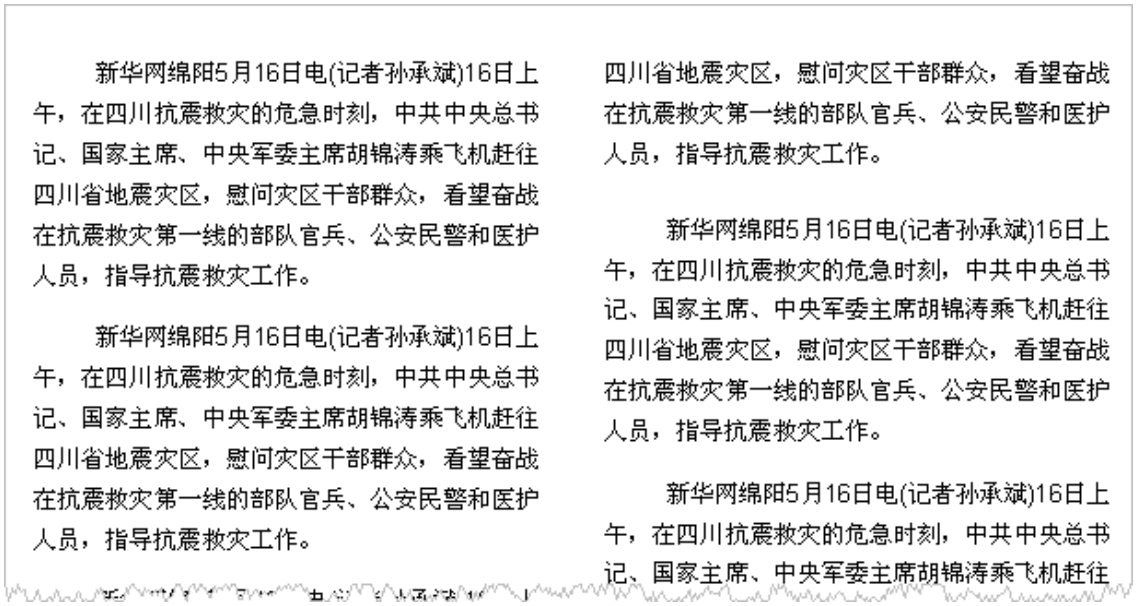
5.2.11 分栏打印

5.2.11.1 分栏打印

报表(PrintDoc)有一个 Columns 属性，表示页面分栏集合。
通过 Columns 的 Add 方法，可以添加分栏：
下面的代码将页面分为两栏，注意加粗的两行代码。

```
Dim doc As New PrintDoc '定义一个报表
Dim s As String '定义字符变量
Doc.Columns.Add() '增加第一栏
Doc.Columns.Add() '增加第二栏
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
For i As Integer = 1 to 20
    Dim rt As New prt.RenderText '定义一个文本对象
    rt.Text = s '设置文本对象的内容
    rt.Style.TextIndent = 9 '首行缩进 9 毫米
    rt.Style.Spacing.Bottom = 5 '每段间隔 5 毫米
    rt.Style.LineSpacing = 150 '1.5 倍行距
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

执行结果：

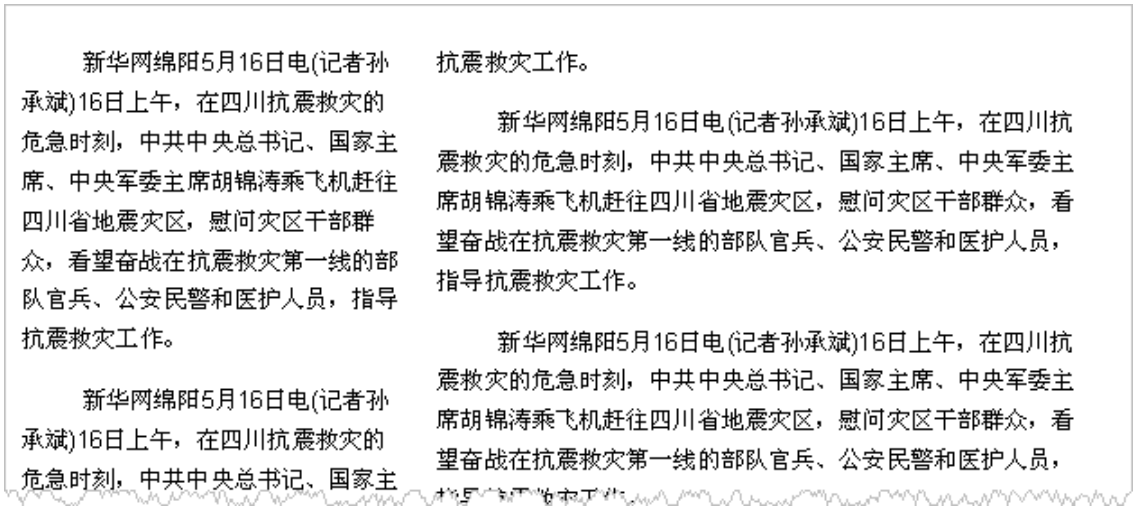


5.2.11.2 分栏设置

分栏有两个属性，Spacing 指定该分栏和下一分栏之间的距离，Width 指定该分栏的宽度。
很少需要去设置分栏的，因为默认设置通常会符合我们的要求。
示例：

```
Dim doc As New PrintDoc '定义一个报表
Dim s As String '定义字符变量
Doc.Columns.Add() '增加第一栏
Doc.Columns.Add() '增加第二栏
Doc.Columns(0).Width = 60 '第一栏的宽度为 60 毫米
Doc.Columns(0).Spacing = 1 '栏间距 1 毫米
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
For i As Integer = 1 to 20
    Dim rt As New prt.RenderText '定义一个文本对象
    rt.Text = s '设置文本对象的内容
    rt.Style.TextIndent = 9 '首行缩进 9 毫米
    rt.Style.Spacing.Bottom = 5 '每段间隔 5 毫米
    rt.Style.LineSpacing = 150 '1.5 倍行距
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

执行结果：



5.2.11.3 打印标签

之前我们通过调整排列方式，实现了一行打印多个标签的功能。

现在我们来看看如何通过分栏实现同样的功能。
示例:

```
Dim doc As New PrintDoc '定义一个报表
Doc.Columns.Add() '增加三栏
Doc.Columns.Add()
Doc.Columns.Add()
Doc.Columns(0).Spacing = 2 '栏间距 2 毫米
Doc.Columns(1).Spacing = 2 '栏间距 2 毫米
For i As integer = 1 to 100
    Dim ra As New prt.RenderArea
    Dim rt As prt.RenderText
    ra.Width = "Parent.Width" '宽度等于分栏宽度
    ra.Height = 24 '高度 24 毫米
    ra.SplitHorzBehavior = prt.SplitBehaviorEnum.Never '禁止水平分割
    ra.SplitVertBehavior = prt.SplitBehaviorEnum.Never '禁止垂直分割
    ra.Style.Spacing.Bottom = 2
    ra.Style.Borders.All = New prt.Linedef(0.3, Color.Red) '设置边框
    Doc.Body.ChildRen.Add(ra) '将容器加入到报表中
    rt = New prt.RenderText
    rt.Text = "姓名："
    rt.X = 2
    rt.Y = 2
    ra.Children.Add(rt) '添加到容器中

    rt = New prt.RenderText
    rt.Text = "黄财德"
    rt.X = 15
    rt.Y = 2
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "职务："
    rt.X = 2
    rt.Y = "Prev.Bottom + 2" '垂直坐标位于迁移对象之下的 2 毫米
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "总经理"
    rt.X = 15
    rt.Y = "Prev.Top" '垂直坐标和前一对象相同
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "编号："
    rt.X = 2
```

```
    rt.Y = "Prev.Bottom + 2"
    ra.Children.Add(rt)

    rt = New prt.RenderText
    rt.Text = "44081919710917243"
    rt.X = 15
    rt.Y = "Prev.Top"
    ra.Children.Add(rt)
Next
Doc.Preview() '预览报表
```

执行结果：



5.2.12 强制换页

先看看下面两个属性：

BreakAfter： 用于决定在打印此对象之后，是否换行、换栏、换页

BreakBefore：用于决定在打印此对象之前，是否换行、换栏、换页。

这两个属性都是 BreakEnum 型枚举，有以下可选值：

Column: 另起一栏

Line： 另起一行

Page： 另起一页

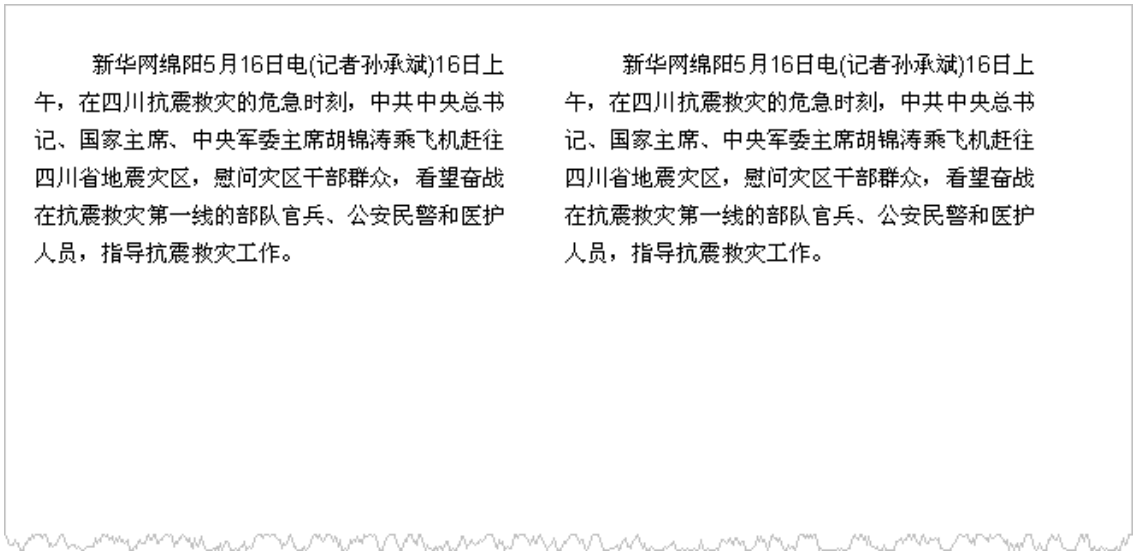
None： 无动作

示例一

```
Dim doc As New PrintDoc '定义一个报表
Dim s As String '定义字符变量
Doc.Columns.Add() '增加第一栏
Doc.Columns.Add() '增加第二栏
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
```

```
For i As Integer = 1 to 2
    Dim rt As New prt.RenderText '定义一个文本对象
    rt.Text = s '设置文本对象的内容
    rt.Style.TextIndent = 9 '首行缩进 9 毫米
    rt.Style.LineSpacing = 150 '1.5 倍行距
    rt.BreakAfter = prt.BreakEnum.Column '打印之后另起一栏
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
doc.Preview() '预览
```

可以看到，两个文本对象被打印在不同的分栏：



需要注意的是，如果给打印对象指定了坐标位置，那么其 BreakAfter 和 BreakBefore 属性是失效的；此时为了强制换页，我们可以加入一个 RenderEmpty 对象，RenderEmpty 是一个空对象，不会打印任何内容；设置 RenderEmpty 对象的 BreakAfter 或 BreakBefore 属性，不指定其坐标，加入到报表中，即可实现强制换页、换栏，换行。

示例二

例如下面的代码，由于所有的打印对象都需要指定坐标位置，所以只有通过增加一个 RenderEmpty 对象来控制换页：

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
Dim rm As prt.RenderEmpty '定一个空对象
For n As integer =1 To 2
    rm = new prt.RenderEmpty '定义一个新的空对象
    rm.BreakBefore = prt.BreakEnum.Page '打印前换页
    doc.Body.Children.Add(rm) '加入到报表中
    For i AS integer = 1 To 5
        rt = New prt.RenderText() '创建文本对象
        rt.Text = "Hello Foxtable " & i '设置文本对象的内容
        rt.Width = "Auto" '自动设置宽度
        rt.X = i * 10 + 20 '指定水平位置
```

```
        rt.y = i * 10 + 20 '指定垂直位置
        rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
        doc.Body.Children.Add(rt) '将文本对象加入到报表
    Next
Next
doc.Preview() '预览
```

5.2.13 页眉页脚

之前的示例代码，打印对象都是加入到主体(Body)区：

```
doc.Body.Children.Add(rt) '将文本对象加入到报表
```

那么如何将对象加入到页眉和页脚区呢？

报表有两个属性：

PageHeader: 指定页眉区的打印对象。

PageFooter: 指定页脚区的打印对象。

任何可打印对象都可以作为页眉、页脚使用。

页眉、页脚可以使用以下编码：

[PageNo]: 页号

[PageCount]: 总页数

[PageX]: 水平方向页号

[PageXCount]: 水平方向总页数

[PageY]: 垂直方向页号

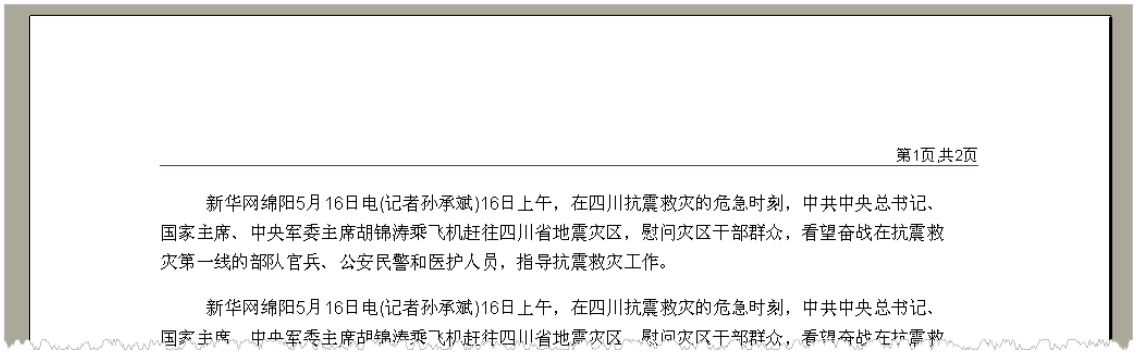
[PageYCount]: 垂直方向总页数。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
Dim s As String '定义字符变量
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
For i As Integer = 1 to 20
    rt = New prt.RenderText
    rt.Text = s '设置文本对象的内容
    rt.Style.TextIndent = 9 '首行缩进 9 毫米
    rt.Style.Spacing.Top = 5 '每段间隔 5 毫米
    rt.Style.LineSpacing = 150 '1.5 倍行距
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
'设置页眉
rt = New prt.RenderText '设置文本对象的内容
rt.Text = "第[PageNo]页,共[PageCount]页" '设置文本内容
```

```
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Right '靠右对齐
rt.Style.Borders.Bottom = New prt.LineDef(0.3, Color.Green) '设置底边框
rt.Style.Padding.Bottom = 0.5 '底端内容缩进 0.5 毫米
rt.Style.FontSize = 8 '字体大小为 8 磅
Doc.PageHeader = rt '作为页眉使用
doc.Preview() '预览
```

执行结果：



5.2.14 水印

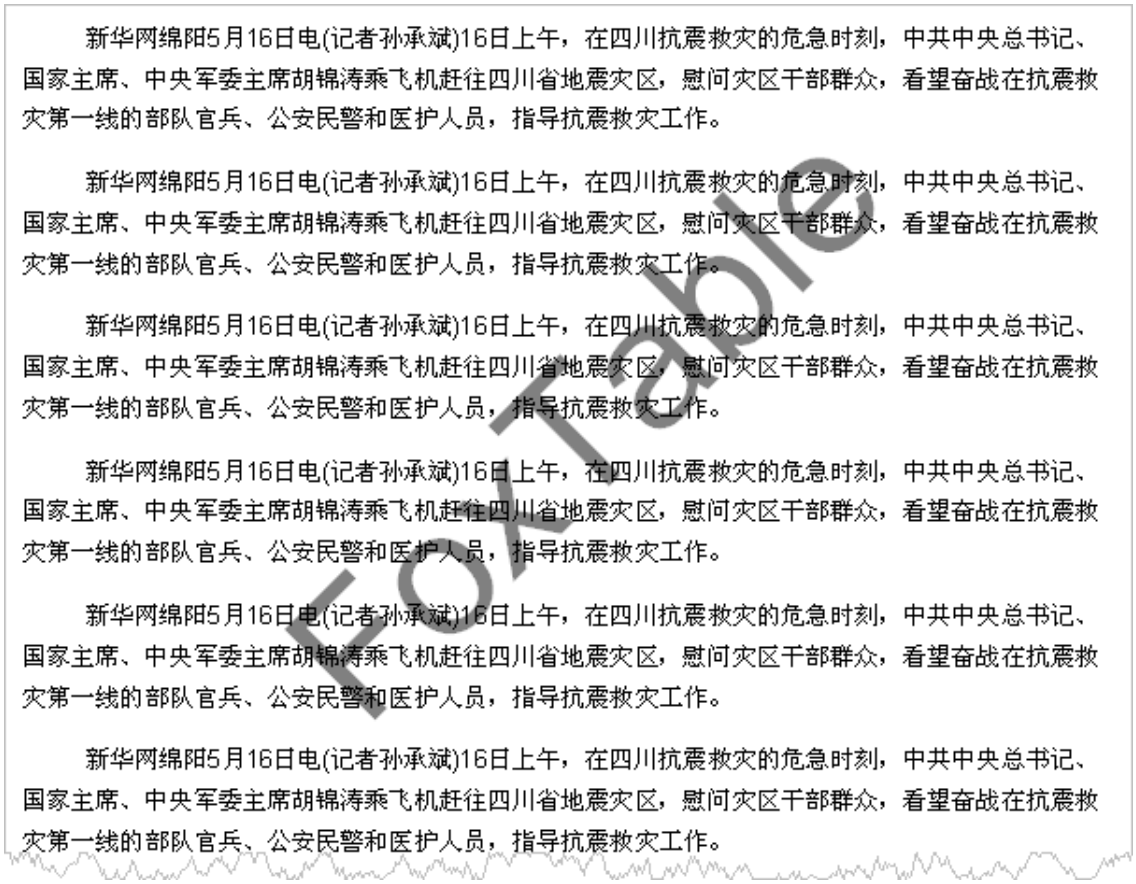
报表(PrintDoc)有一个名为 WaterMark 的属性，用于设置水印。
任何可打印对象都可以作为水印使用，当然通常使用图片和文字。
示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
Dim s As String '定义字符变量
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
For i As Integer = 1 to 20
    rt = New prt.RenderText
    rt.Text = s '设置文本对象的内容
    rt.Style.TextIndent = 9 '首行缩进 9 毫米
    rt.Style.Spacing.Top = 5 '每段间隔 5 毫米
    rt.Style.LineSpacing = 150 '1.5 倍行距
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next

'设置水印
rt = New prt.RenderText '设置文本对象的内容
rt.Text = "Foxtable" '设置文本内容
rt.Width = "Parent.Width" '宽度等于页面宽度
rt.Height = "Parent.Height" '高度等于页面高度
```

```
rt.Style.TextAngle = 45 '旋转 45 度
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中对齐
rt.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中对齐
rt.Style.FontSize = 72 '字体大小为 8 磅
rt.Style.TextColor = Color.Gray '文本颜色为灰色
Doc.WaterMark = rt '作为水印使用
doc.Preview() '预览
```

执行结果：



5.2.15 自动生成目录

如果你生成的报表特别长，例如长达 200 页，也许你喜欢给报表加上目录，就像书籍一样。

RenderToc 类型专门用于生成报表目录，该类型有一个 AddItem 方法，语法为：

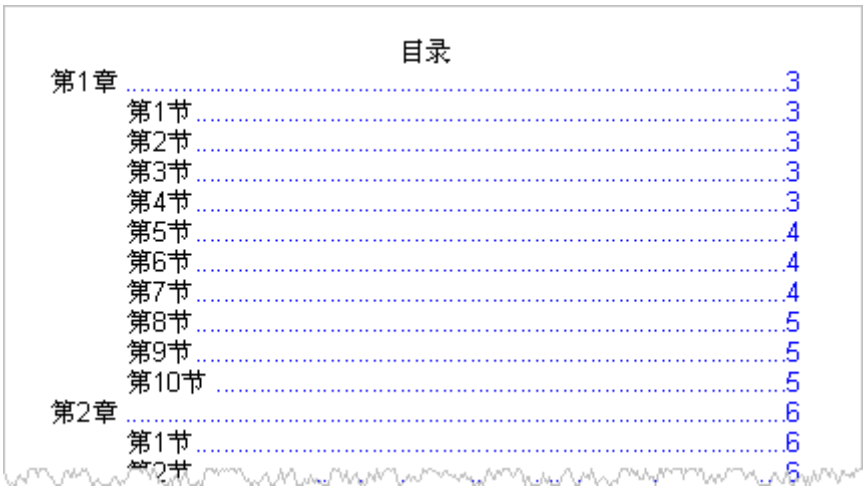
AddItem(目录名称,目录指向的对象,目录层级)

下面是一段示例代码，请留意加粗的几行：

```
Dim doc As New PrintDoc
Dim toc As New prt.RenderToc
Dim s As String
Dim rt As New prt.RenderText
toc.BreakAfter = Prt.BreakEnum.Page
rt.Text = "目录"
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center
```

```
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午,在四川抗震救灾的"
s = s & "危急时刻,中共中央总书记?国家主席?中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区,慰问灾区干部群众,看望奋战在抗震救灾第一线的部队官兵?公"
s = s & "安民警和医护人员,指导抗震救灾工作."
doc.Body.Children.Add(rt)
For n As Integer = 1 to 30
    rt = New prt.RenderText
    rt.Text = "第" & n & "章"
    doc.Body.Children.Add(rt)
    toc.AddItem(rt.Text, rt, 1)
    For m As integer = 1 to 10
        rt = New prt.RenderText
        rt.Text = s
        doc.Body.Children.Add(rt)
        toc.AddItem("第" & m & "节", rt, 2)
    Next
Next
doc.body.Children.Insert(1,toc)
doc.Preview()
```

这是生成的目录：



5.2.16 指定打印机

如果安装了多个打印机，且需要在非默认打印机上打印，可以设置 PrintDoc 的 PrinterName 属性，在指定的打印机上打印。

例如

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.Text = "Hello Foxtable"
doc.Body.Children.Add(rt)
Doc.PrinterName = "HP LaserJet 6L"
doc.Print()
```

5.2.17 禁止自动旋转

打印报表的时候，系统默认会自动根据实际纸张的大小，判断是否需要旋转打印内容，以便更好地配合纸张。
如果不需要自动旋转打印内容，可以将 AutoRotate 属性设为 False。
例如：

```
Dim doc As New PrintDoc
Dim ra As New prt.RenderArea
doc.PageSetting.Width = 150
doc.PageSetting.Height = 80
doc.AutoRotate = False '禁止自动旋转打印内容
ra.Width = "Parent.Width"
ra.Height = "Parent.Height"
ra.Style.Borders.All = New prt.Linedef(5, Color.Red)
doc.Body.Children.Add(ra)
doc.Print()
```

你可以删除下面这一行代码，重新打印一张，看看结果有什么不同:

```
doc.AutoRotate = False '禁止自动旋转打印内容
```

5.2.18 禁止打印

你可能希望给客户提供一个试用版的系统，在客户购买正版之前，只能打印预览，而不能打印。
要实现这个功能很简单，只需给 Preivew 加上一个参数 True 即可，例如：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.Text = "Hello Foxtable"
doc.Body.Children.Add(rt)
doc.Preview(True)
```

加上 True 参数之后，打印预览窗口的保存和打印按钮将不可用。
窗口打印是通过专业报表实现的，所以同样可以实现只能预览不能打印的效果，例如：

```
Dim doc As PrintDoc = e.Form.GernatePrintDoc()
doc.Preview(True)
```

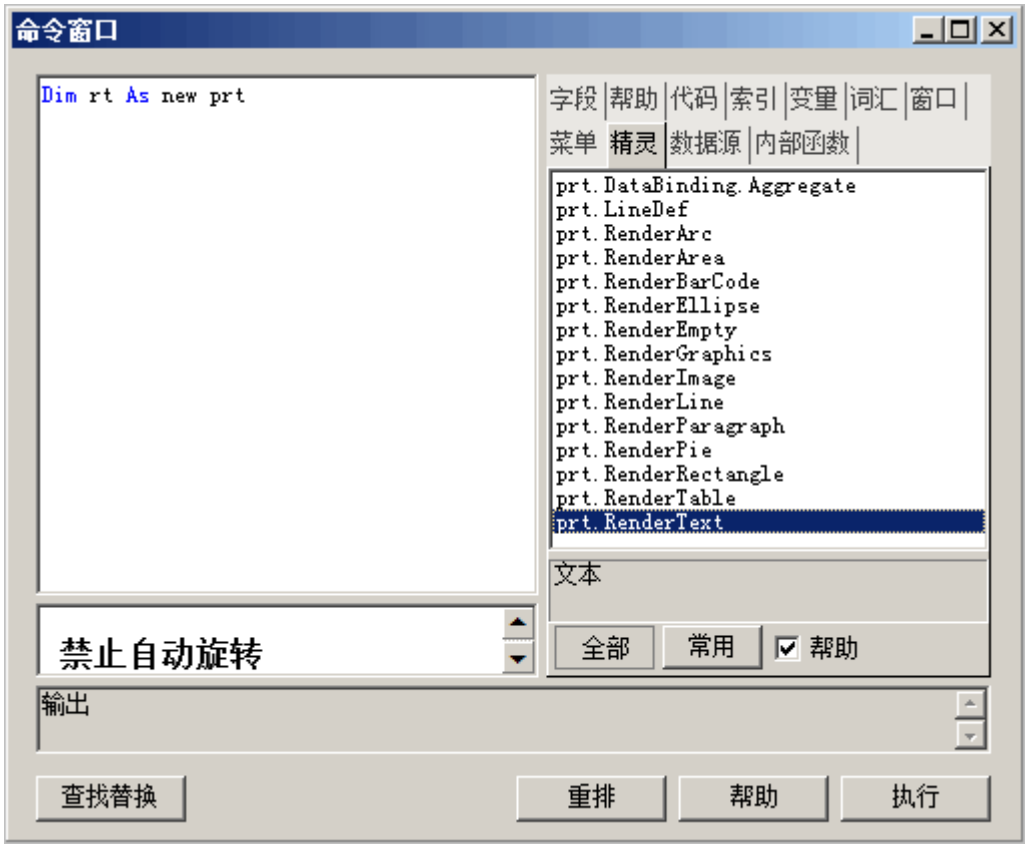
5.2.19 无需记忆任何东西

专业报表有大量的对象和属性，还提供了丰富的样式设置，任何人都不可能准确记住所有内容的。

但是 FoxTable 提供的智能输入功能，使得而我们无需记忆任何东西，照样可以轻松使用专业报表。例如我们输入代码：

Dim rt As new prt

然后按 Tab 键，即可自动列出所有报表对象，按上下箭头键选择对象，然后按回车键即可自动输入对象名称：

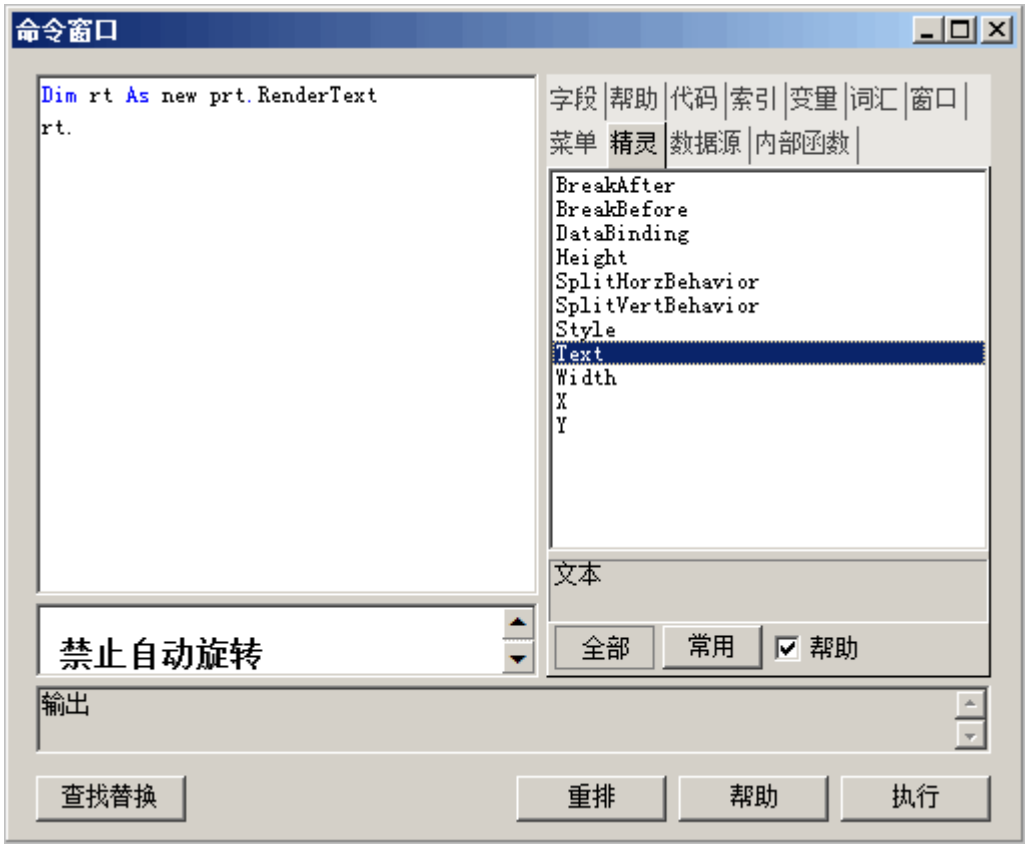


当输入代码：

Dim rt As new prt.RenderText

rt.

然后按 Tab 键，即可已自动列出 RenderText 的所有属性供选择，同样按上下箭头键选择属性，然后按回车键即可自动输入属性名称：



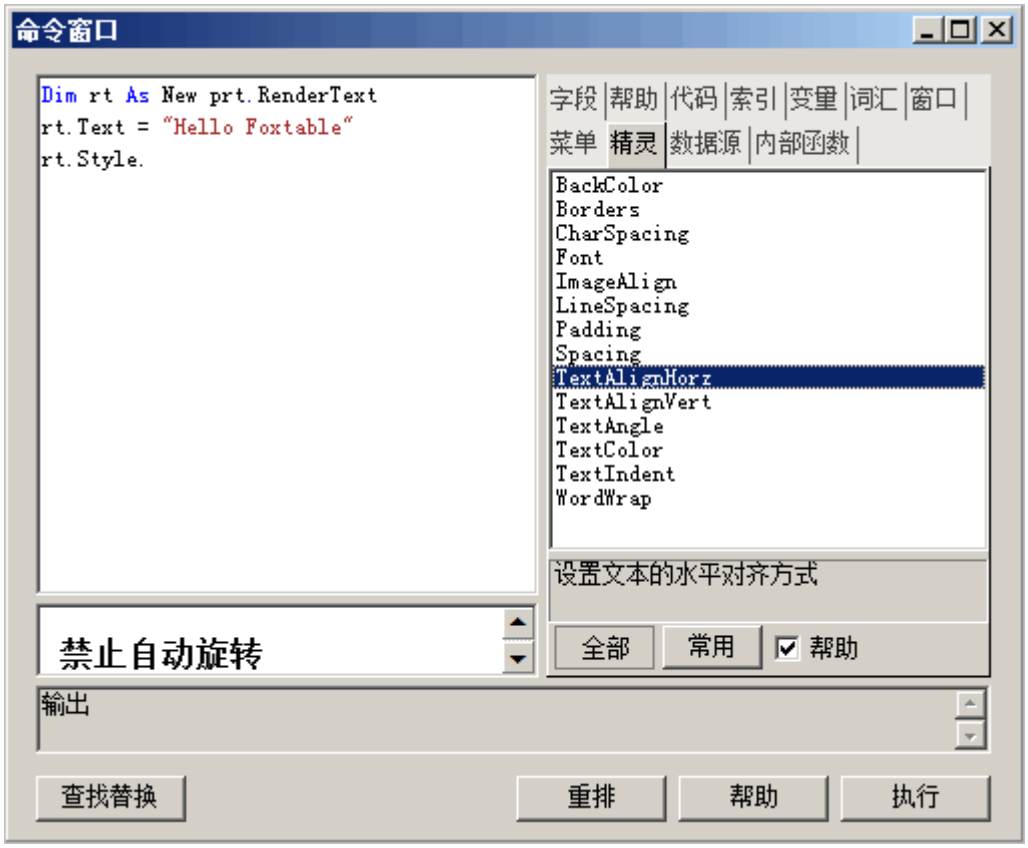
当输入代码：

Dim rt As New prt.RenderText

rt.Text = "Hello Foxtable"

rt.Style.

然后按 Tab，即可自动列出所有的样式名称供选择，同样按上下箭头键选择样式，然后按回车键即可自动输入样式名称：



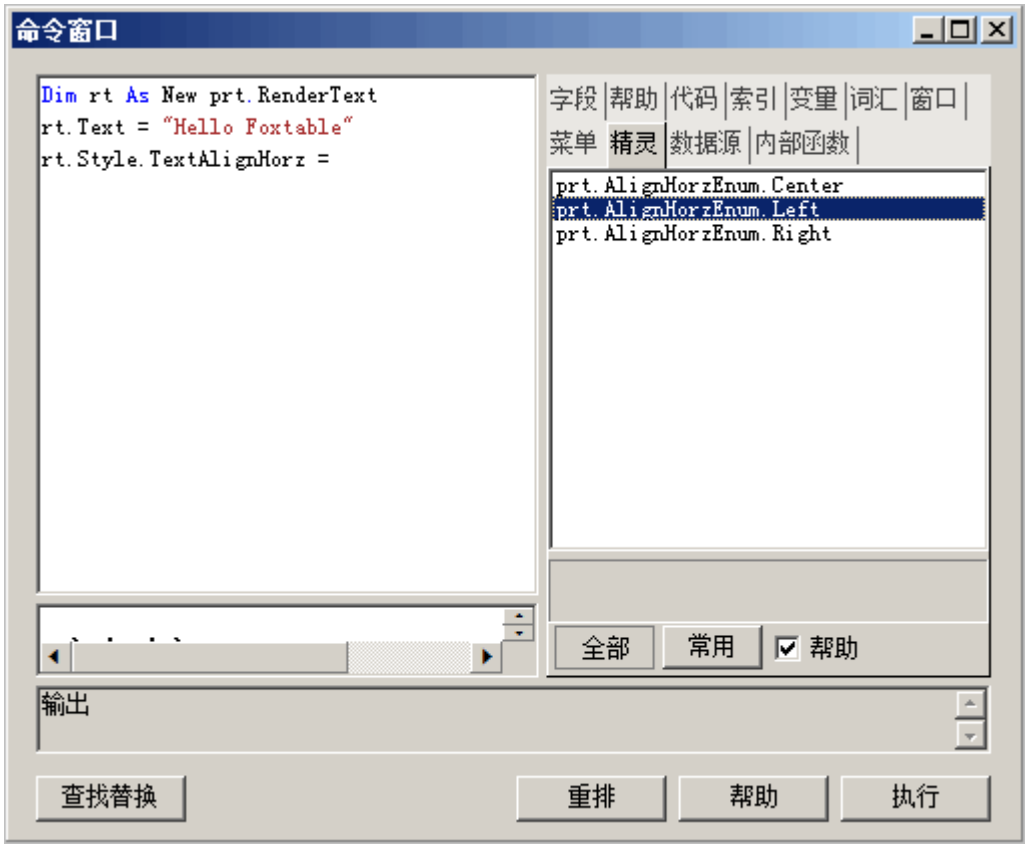
当输入代码：

Dim rt As New prt.RenderText

rt.Text = "Hello Foxtable"

rt.Style.TextAlignHorz =

然后按 Tab,即可自动列出 TextAlignHorz 样式的全部候选值，同样按上下箭头键选择候选值，然后按回车键即可自动输入候选值：



可以看出，不管是对象名称，还是属性名称，还是样式名称，还是属性或样式的候选值，都可以自动输入，不需要准确记忆任何东西。

5.3 表格基础

5.3.1 创建表格

表格的类型为 RenderTable，每一个单元格就是一个容器，通过单元格的 RenderObject 属性，可以在单元格中放置任何可打印的对象。

示例：

请在命令窗口执行下面的代码，该段代码将创建一个 6 行 6 列的表格：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
'下面的代码向表格中填入值
For r As integer = 0 to 5
    For c As integer = 0 to 5
        dim rx As New prt.RenderText '定义一个文本对象
        rx.Text = r & "," & c '为文本对象设置内容
        rt.Cells(r, c).RenderObject = rx '将文本对象放置在单元格中
    Next
```

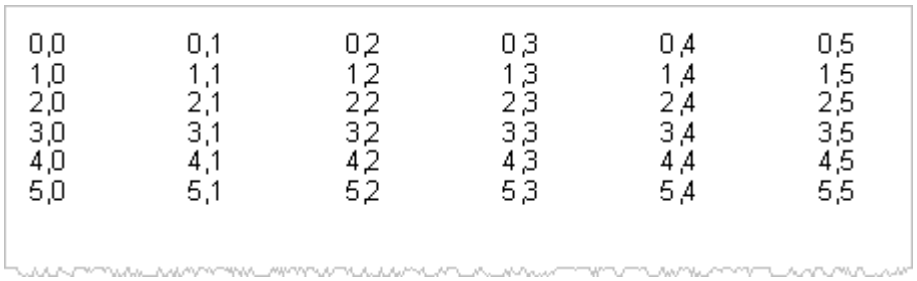
```
Next
doc.Preview() '预览报表
```

可以看到，RenderTable 的行和列是自动创建的，通过 Cell 属性引用指定的单元格，例如:

```
rt.Cell(0,0)
```

表示第一行第一个单元格。

下面是执行结果：



5.3.2 网格线

表格的样式(Style)有一个名为 GridLines 的子属性，用于设置网格线。

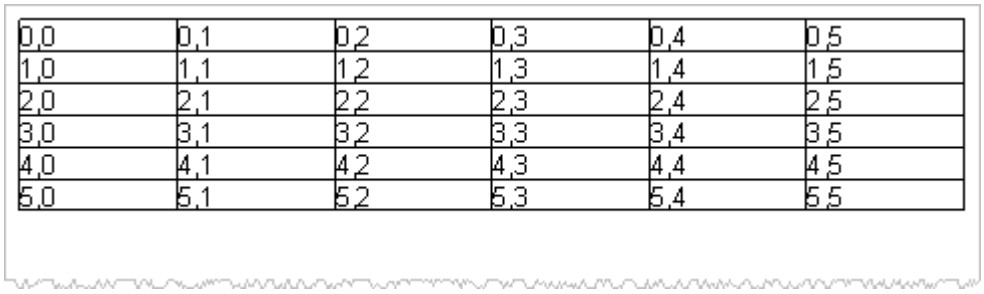
GridLines 本身又包括以下子属性：

All: 所有网格线
Bottom：底边框
Left: 左边框
Right： 右边框
Top： 上边框
Horz： 水平网格线
Vert： 垂直网格线

示例一

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
'下面的代码向表格中填入值
For r As integer = 0 to 5
    For c As integer = 0 to 5
        Dim rx As New prt.RenderText '定义一个文本对象
        rx.Text = r & "," & c '为文本对象设置内容
        rt.Cells(r, c).RenderObject = rx '将文本对象放置在单元格中
    Next
Next
doc.Preview() '预览报表
```

执行结果：

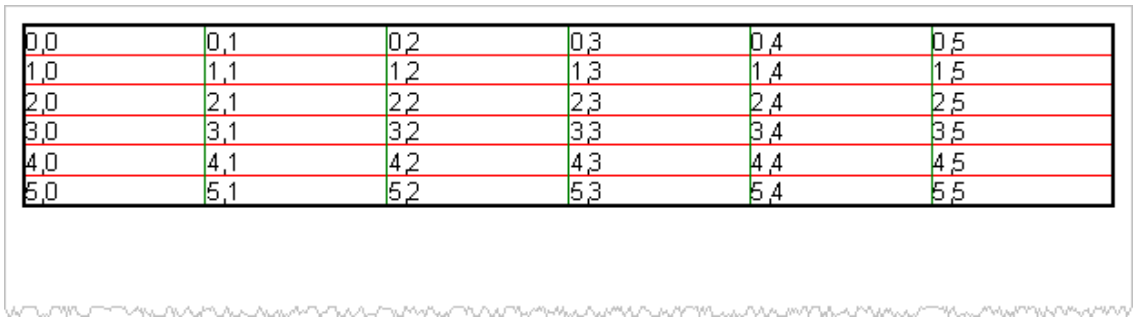


0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

示例二

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
'设置各位置的网格线
rt.Style.GridLines.All = New Prt.LineDef(0.5, Color.Black)
rt.Style.GridLines.Horz = new Prt.LineDef(Color.Red)
rt.Style.GridLines.Vert = new Prt.LineDef(Color.Green)
'下面的代码向表格中填入值
For r As integer = 0 to 5
    For c As integer = 0 to 5
        Dim rx As New prt.RenderText '定义一个文本对象
        rx.Text = r & "," & c '为文本对象设置内容
        rt.Cells(r, c).RenderObject = rx '将文本对象放置在单元格中
    Next
Next
doc.Preview() '预览报表
```

执行结果：



0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

5.3.3 默认尺寸

如果没有明确指定表宽的情况下，表宽等于容器宽度，通常就是页面可打印区域的宽度。

行高由该行内容的高度决定，如果该行没有内容，行高将等于 0。

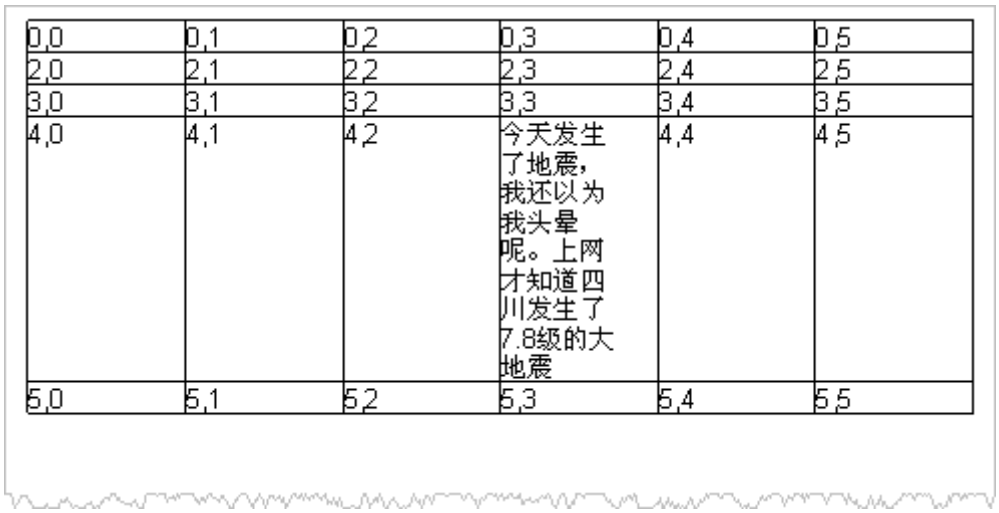
列宽等于表格宽度除以总列数。

示例

```
Dim doc As New PrintDoc '创建一个报表
Dim rt As New prt.RenderTable '创建一个表格对象
Dim rx As prt.RenderText '定义一个文本对象
```

```
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '设置网格线
'下面的代码向表格中填入值
For r As Integer = 0 to 5
    If r <> 1 Then
        For c As Integer = 0 to 5
            rx = New prt.RenderText '创建一个新的文本对象
            rx.text = r & "," & c '设置文本对象的内容
            rt.Cells(r, c).RenderObject = rx '放置在单元格中
        Next
    End If
Next
'创建一个新的文本对象，内容较长。
rx = New prt.RenderText
rx.text = "今天发生了地震，我还以为我头晕呢。上网才知道四川发生了 7.8 级的大地震"
rt.Cells(4,3).RenderObject = rx
doc.Preview() '预览报表
```

下图是执行结果，可以看到编号为 1 的行不见了，这是因为我们向单元格填入内容的时候，跳过了编号为 1 的行：



0,0	0,1	0,2	0,3	0,4	0,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	今天发生了地震，我还以为我头晕呢。上网才知道四川发生了7.8级的大地震	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

5.3.4 设定尺寸

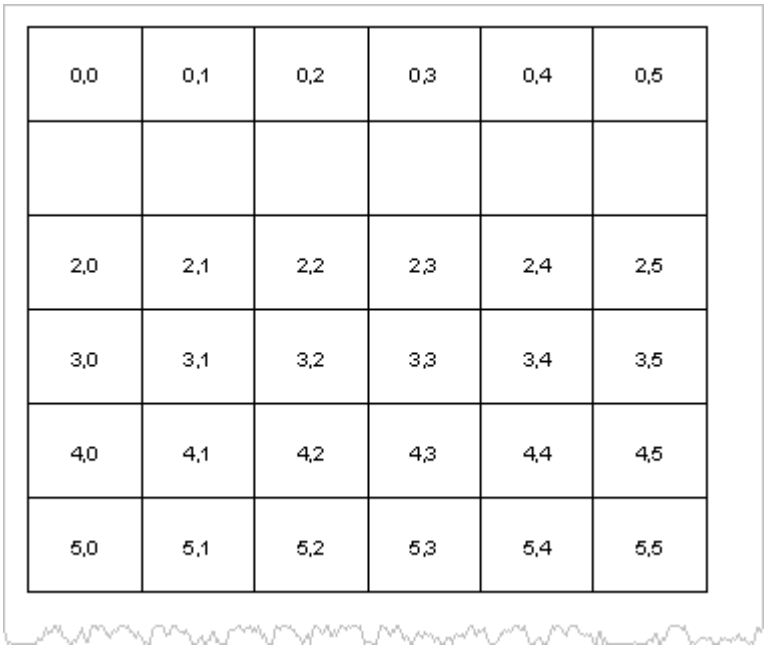
如果明确指定了表格的高度和宽度，那么列宽等于表宽除以总列数，行高等于表高除以总行数。

示例

```
Dim doc As New PrintDoc '创建一个报表
Dim rt As New prt.RenderTable '创建一个表格对象
Dim rx As prt.RenderText '定义一个文本对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Width = 120 '设置表格的宽度为 120 毫米
rt.Height = 100 '设置表格的高度为 100 毫米
rt.Style.GridLines.All = New prt.LineDef '设置网格线类型
```

```
'下面的代码向表格中填入值
For r As integer = 0 to 5
    If r = 1 Then
        Continue for
    End If
    For c As integer = 0 to 5
        rx = New prt.RenderText '创建一个新的文本对象
        rx.text = r & "," & c '设置文本对象的内容
        rx.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中
        rx.Style.TextAlignVert = prt.AlignVertEnum.Center '垂直居中
        rt.Cells(r, c).RenderObject = rx '放置在单元格中
    Next
Next
doc.Preview() '预览报表
```

下图是执行结果，可以看到第二行虽然没有设置内容，但是和其它行的高度是一样的：



0,0	0,1	0,2	0,3	0,4	0,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

5.3.5 指定行数和列数

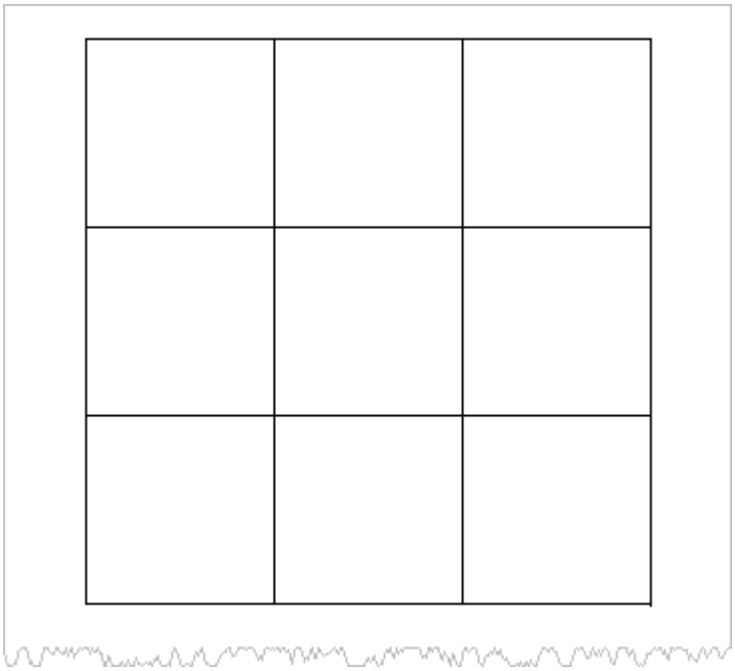
虽然行和列可以自动创建，但是也可以明确指定行数和列数：

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Width = 150 '表宽为 150 毫米
rt.Height = 150 '表高为 150 毫米
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
```

```
Doc.Preview() '预览报表
```

执行结果：

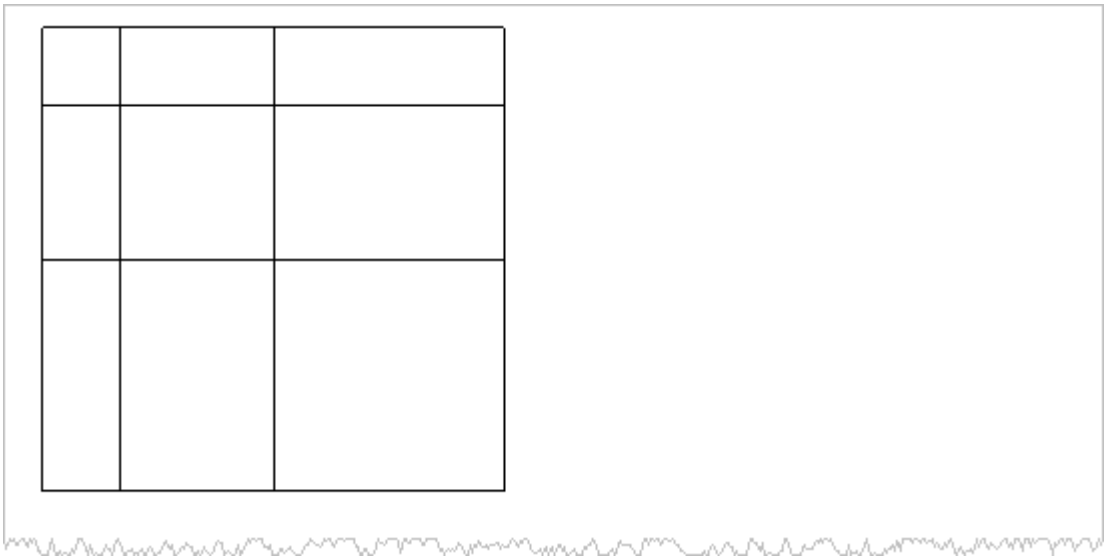


5.3.6 指定行高和列宽

示例一

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Width = "Auto"
For i As integer = 0 To 3
    rt.Rows(i).Height = 10 * i '设置行高
    rt.Cols(i).Width = 10 * i '设置列宽
next
Doc.Preview() '预览报表
```

这是执行结果：



需要注意的是，如果要分别指定了每一列的绝对宽度，那么下面这一行是必须的：

```
rt.Width = "Auto"
```

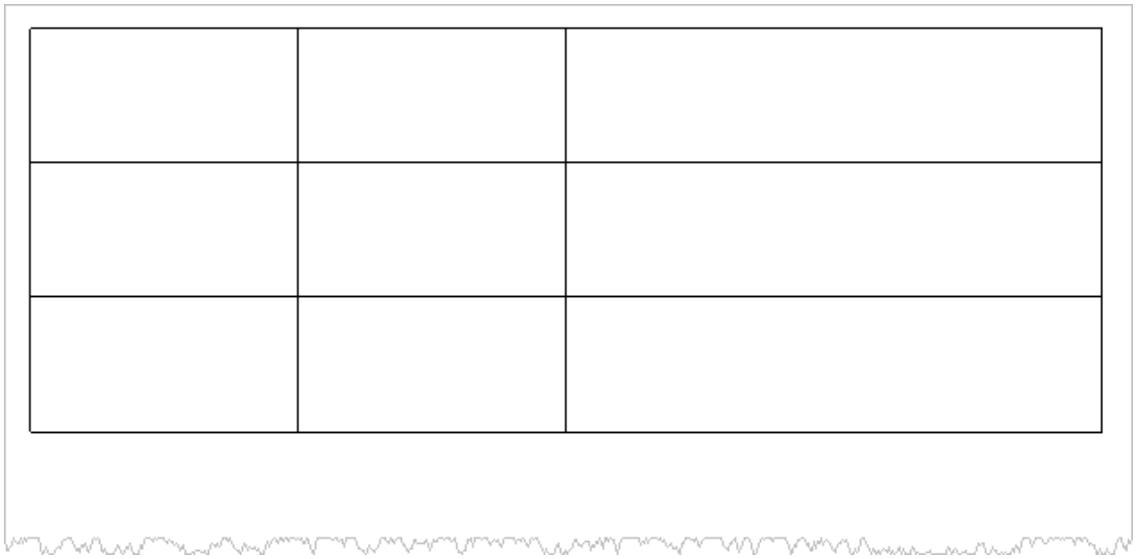
只有这样，表格的宽度才会自动等于各列所设置宽度之和，否则表格的宽度将始终等于页面可打印区域的宽度，而列宽的设置值将成为该列宽度占表宽的比例。

示例二

例如下面的代码，因为没有将表格的宽度设为自动，所以表格的宽度等于页宽，第一列和第二列的宽度分别占表宽的 25%，第三列占 50%：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Height = 60 '
rt.Rows.Count = 3
rt.Cols(0).Width = 25
rt.Cols(1).Width = 25
rt.Cols(2).Width = 50
Doc.Preview() '预览报表
```

执行结果：



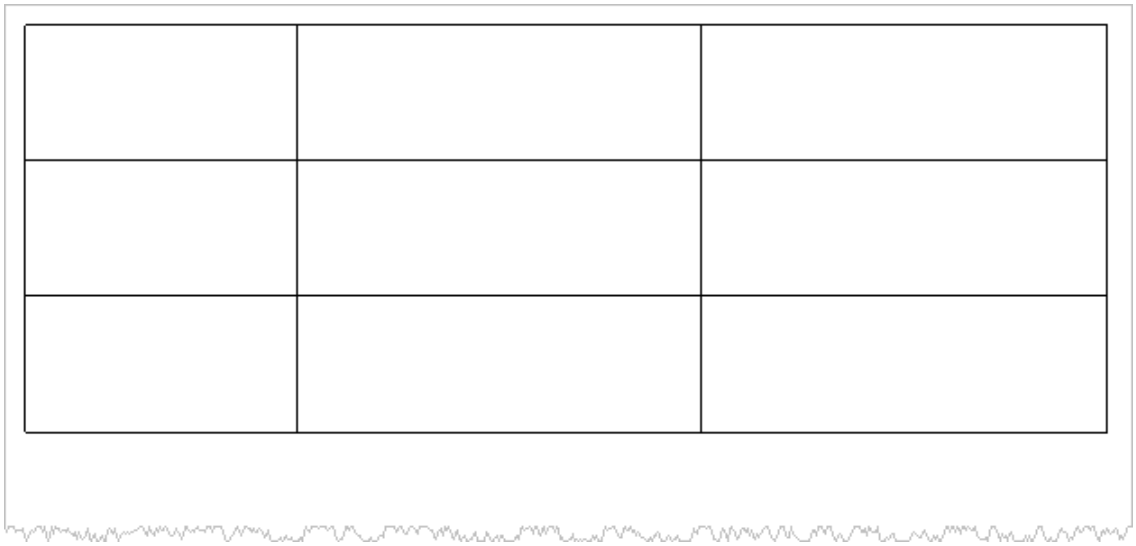
示例三

如果只是指定其中部分列的宽度，那么指定的列宽就是实际列宽，表宽减去这些列的宽度后，剩余的宽度将被平均分配到其余各列。

例如下面的代码，表格的宽度等于页宽，第一列的宽度为 40 毫米，剩余宽度被平均分配到另外两列：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Height = 60 '
rt.Rows.Count = 3
rt.Cols.Count = 3
rt.Cols(0).Width = 40
Doc.Preview() '预览报表
```

执行结果：

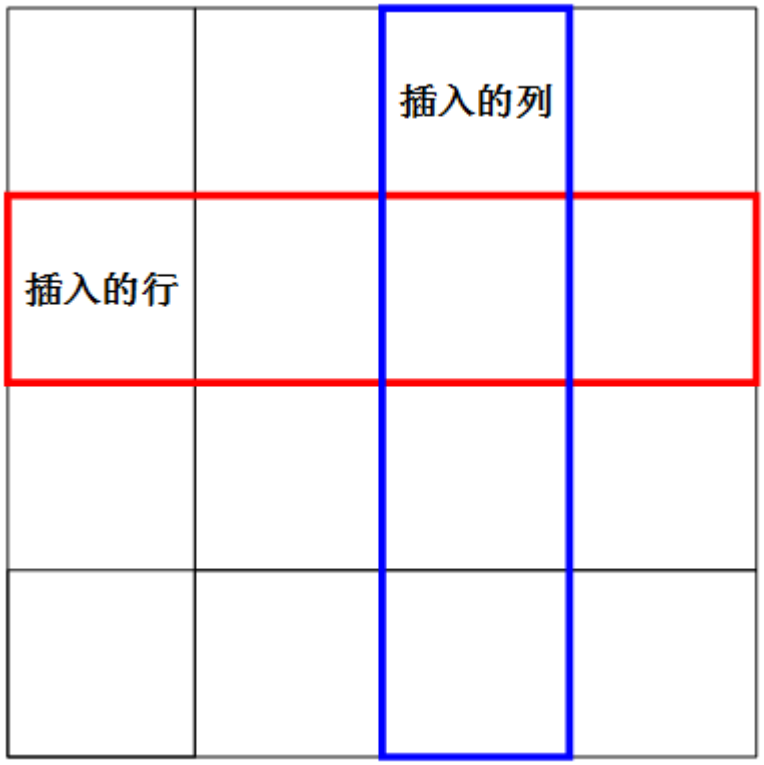


5.3.7 插入行和列

Rows 和 Cols 都有一个 Insert 方法，用于在指定位置插入行或列。
示例：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable()
doc.Body.Children.Add(rt)
rt.Style.GridLines.All = New prt.LineDef
rt.Width = 100
rt.Height = 100
rt.Rows.Count = 3 '设置行数为3行
rt.Cols.Count = 3 '设置列数为3列
rt.Rows.Insert(1) '在第2行位置插入一行
rt.Cols.Insert(2) '在第3列位置插入一列
rt.Cells(1,0).Text = "插入的行"
rt.Cells(0,2).Text = "插入的列"
rt.Style.Font = New Font("宋体", 14, FontStyle.Bold)
rt.Style.TextAlignVert = prt.AlignHorzEnum.Center
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center
rt.Rows(1).Style.Borders.All = New prt.LineDef("1mm", Color.Red)
rt.Cols(2).Style.Borders.All = New prt.LineDef("1mm", Color.Blue)
Doc.Preview()
```

执行结果：



5.3.8 单元格内距

在默认的情况下，单元格内容紧贴网格线：

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

我们知道 Spacing 属性可以用来设置对象之间的间隔，如果对象位于单元格中，则可用来设置对象和网格线的距离。

示例

```
Dim doc As New PrintDoc '创建一个报表
Dim rt As New prt.RenderTable '创建一个表格对象
Dim rx As prt.RenderText '定义一个文本对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
'下面的代码向表格中填入值
For r As integer = 0 to 5
    For c As integer = 0 to 5
        rx = New prt.RenderText '创建一个新的文本对象
```

```
        rx.text = r & "," & c '设置文本对象的内容
        rx.Style.Spacing.All = 1 '内容和网格线的距离为 1
        rt.Cells(r, c).RenderObject = rx '放置在单元格中
    Next
Next
doc.Preview() '预览报表
```

执行结果：

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

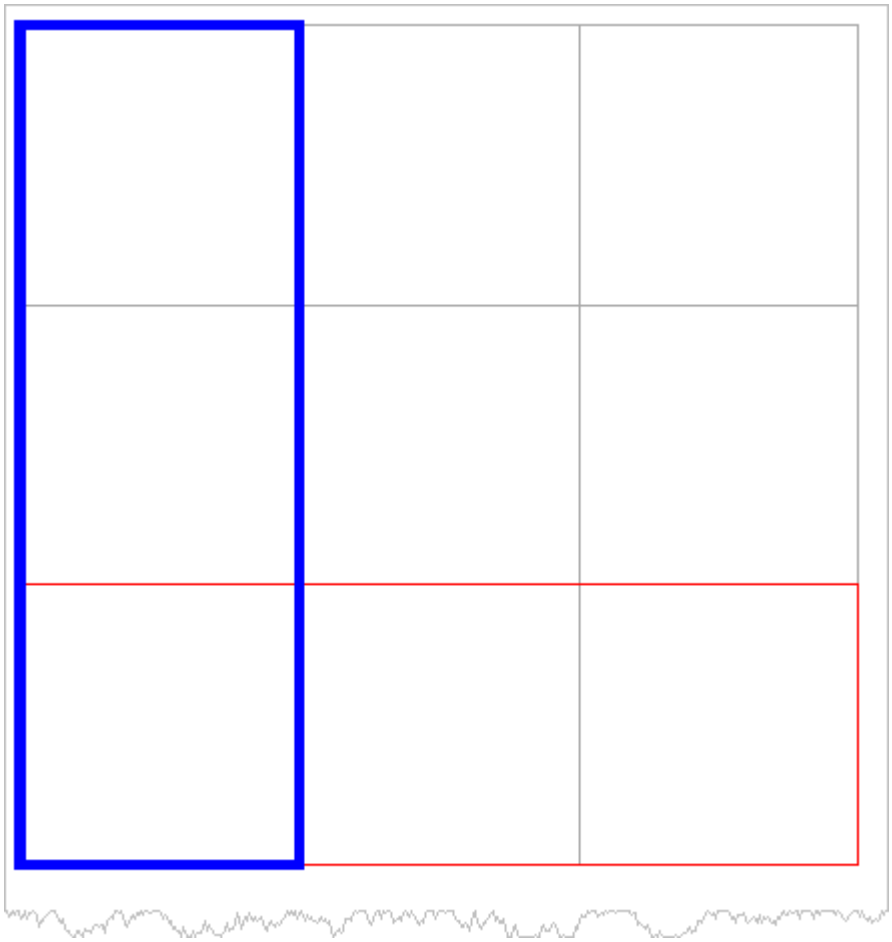
5.3.9 行列样式

行、列都有自己的 Style 属性，用于设置其外观。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef(Color.DarkGray) '将表格的颜色设为深灰色
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
rt.Width = 150 '设置表格的宽度
rt.Height = 150 '设置表格的高度
rt.Rows(2).Style.Borders.All = New prt.LineDef("0.5mm", Color.Red)
rt.Cols(0).Style.Borders.All = New prt.LineDef("2mm", Color.Blue)
Doc.Preview() '预览报表
```

执行结果：

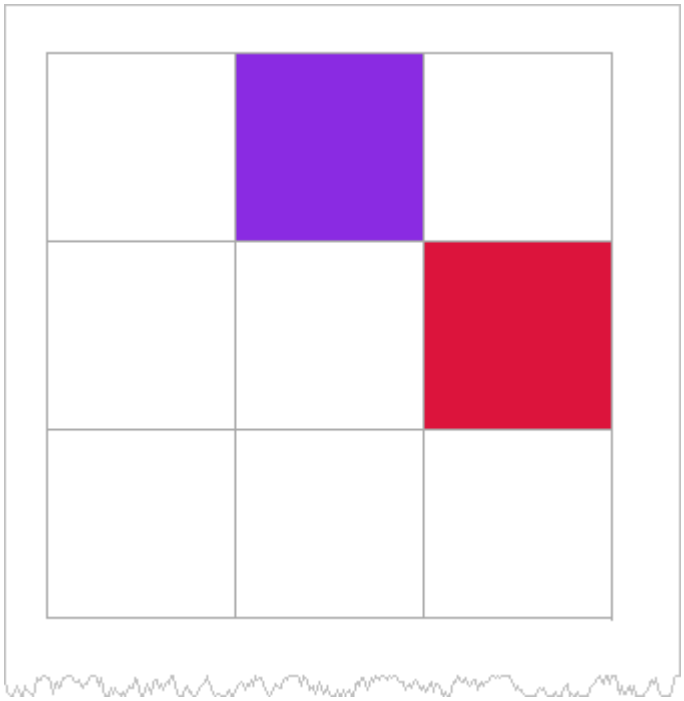


5.3.10 单元格样式

不仅行和列，其实单元格也有自己的 Style 属性。
示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef(Color.DarkGray) '将表格的颜色设为深灰色
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
rt.Width = 150 '设置表格的宽度
rt.Height = 150 '设置表格的高度
rt.Cells(1, 2).Style.BackColor = Color.Crimson
rt.Cells(0, 1).Style.BackColor = Color.BlueViolet
Doc.Preview() '预览报表
```

执行结果：



5.3.11 打印图片

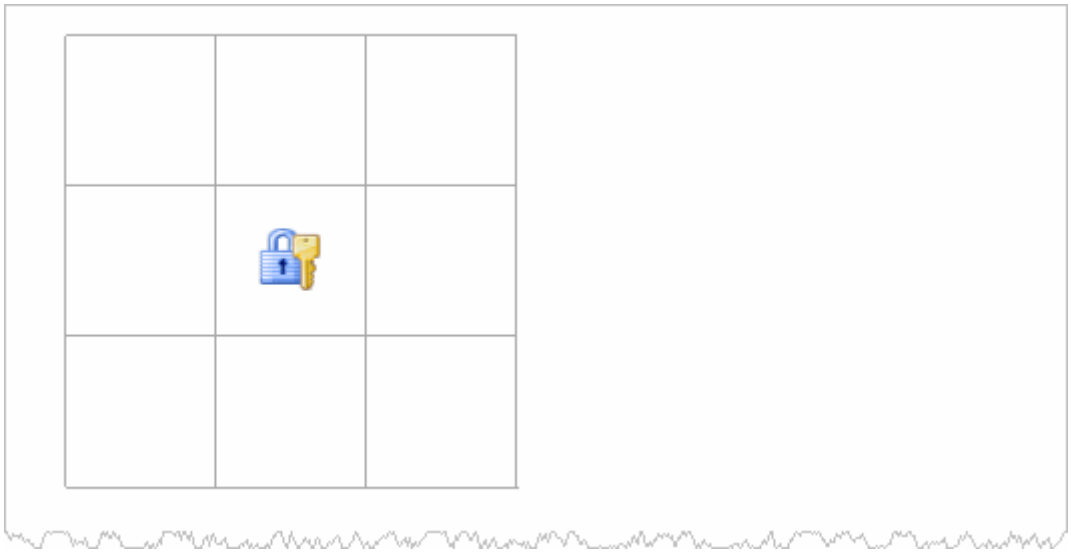
单元格中可以包含任何可打印的对象，包括图片，甚至是另一个表格。

示例

下面的代码向单元格中插入一个图片

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
Dim rm As New prt.RenderImage() '定义一个图片对象
rm.Image = GetImage("logon.ico") '请改为实际的图标名称和路径
rm.Style.ImageAlign.AlignHorz = prt.ImageAlignHorzEnum.Center '图片水平居中
rm.Style.ImageAlign.AlignVert = prt.ImageAlignVertEnum.Center '图片垂直居中
rm.Style.ImageAlign.StretchHorz = False '禁止水平方向扩展图片
rm.Style.ImageAlign.StretchVert = False '禁止垂直方向扩展图片
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef(Color.DarkGray) '将表格的颜色设为深灰色
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
rt.Width = 60 '设置表格的宽度
rt.Height = 60 '设置表格的高度
rt.Cells(1,1).RenderObject = rm '将单元格内容设置为图片对象 rm
Doc.Preview() '预览报表
```

执行结果：

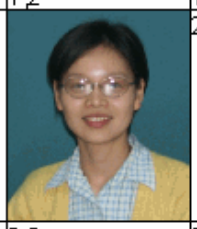


5.3.12 快速设置打印对象

单元格中的打印内容，绝大多数时候是文本或图片。
单元格提供了两个属性，用于快速设置要打印的文本或图片
Text: 指定单元格的文本内容
Image: 指定单元格的图片。
示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '将网格线设为默认类型
'下面的代码向表格中填入值
For r As integer = 0 to 5
    For c As integer = 0 to 5
        If r = 2 AndAlso c =2 Then
            rt.Cells(r, c).Image = GetImage("ep1.bmp")
        Else
            rt.Cells(r, c).Text = r & "," & c '
        End If
    Next
Next
doc.Preview() '预览报表
```

执行结果：

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1		2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

这一行代码：

```
rt.Cells(r, c).Text = r & "," & c '
```

等同于：

```
Dim rx As New prt.RenderText
rx.Text = r & "," & c
rt.Cells(r, c).RenderObject = rx
```

前一种方法简洁，后一种方法可以让你进行更多的设置。

5.3.13 统一设置属性

表格、行、列都有 Style 属性，这样我们就无需针对每一个单元格去设置其打印对象的属性。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
'下面的代码向表格中填入值
For r As Integer = 0 to 5
    For c As integer = 0 to 5
        rt.Cells(r, c).Text = r & "," & c '
    Next
Next
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Right '所有文本内容靠右对齐
rt.Cols(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '唯独第一列内容居中
doc.Preview() '预览报表
```

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

5.3.14 一个例外

单元格内边距，也就是内容和网格线的距离，是唯一不能通过表、行、列的 Style 属性设置的。对于表格来说，Style 的 Spacing 子属性设置的是表格和相邻对象的距离，而不是单元格内边距。表格、行、列有一个 CellStyle 属性，专门用于解决这些问题，顾名思义，CellStyle 当然完全是针对单元格的。。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.CellStyle.Spacing.All = 1 '单元格内边距设为 1 毫米
'下面的代码向表格中填入值
For r As Integer = 0 to 5
    For c As integer = 0 to 5
        rt.Cells(r, c).Text = r & "," & c '
    Next
Next
doc.Preview() '预览报表
```

执行结果：

0,0	0,1	0,2	0,3	0,4	0,5
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5

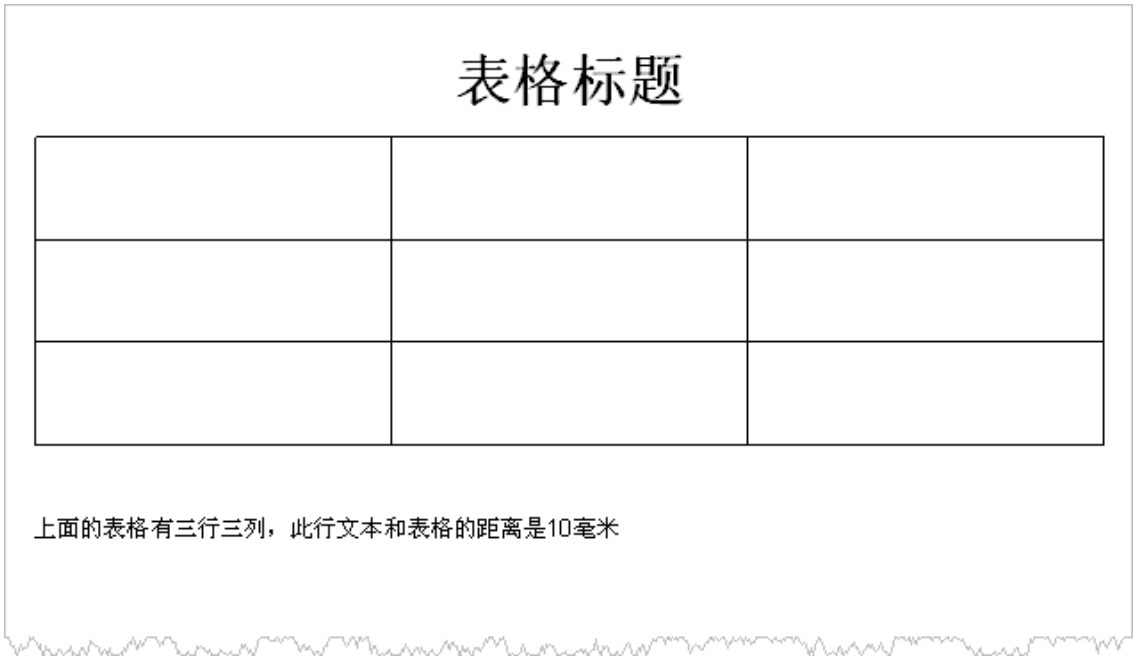
一般来说，如果你用 Style 无效，那么就改用 CellStyle 来试试，总有一个可以的。

5.3.15 给表格加上标题

示例代码：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
Dim rs As New prt.RenderText() '定义一个文本对象
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
rt.Height = "6cm" '设置表格的高度
rt.Style.Spacing.Top = 4 '表格和前面对象的垂直间隔为 4 毫米
rt.Style.Spacing.Bottom = 10 '表和和后续对象的垂直间隔为 10 毫米
rs.Text = "表格标题" '设置文本对象的内容
rs.Style.Font = New Font("宋体", 24 , FontStyle.Bold) '设置文本对象的字体
rs.Style.TextAlignHorz = prt.AlignHorzEnum.Center '文本内容水平居中
doc.Body.Children.Add(rs) '将文本对象加入到表格中
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rs = New prt.RenderText() '再次定义一个文本对象
rs.Text = "上面的表格有三行三列，此行文本和表格的距离是 10 毫米"
doc.Body.Children.Add(rs) '将文本对象加入到表格中
Doc.Preview() '预览报表
```

执行结果：

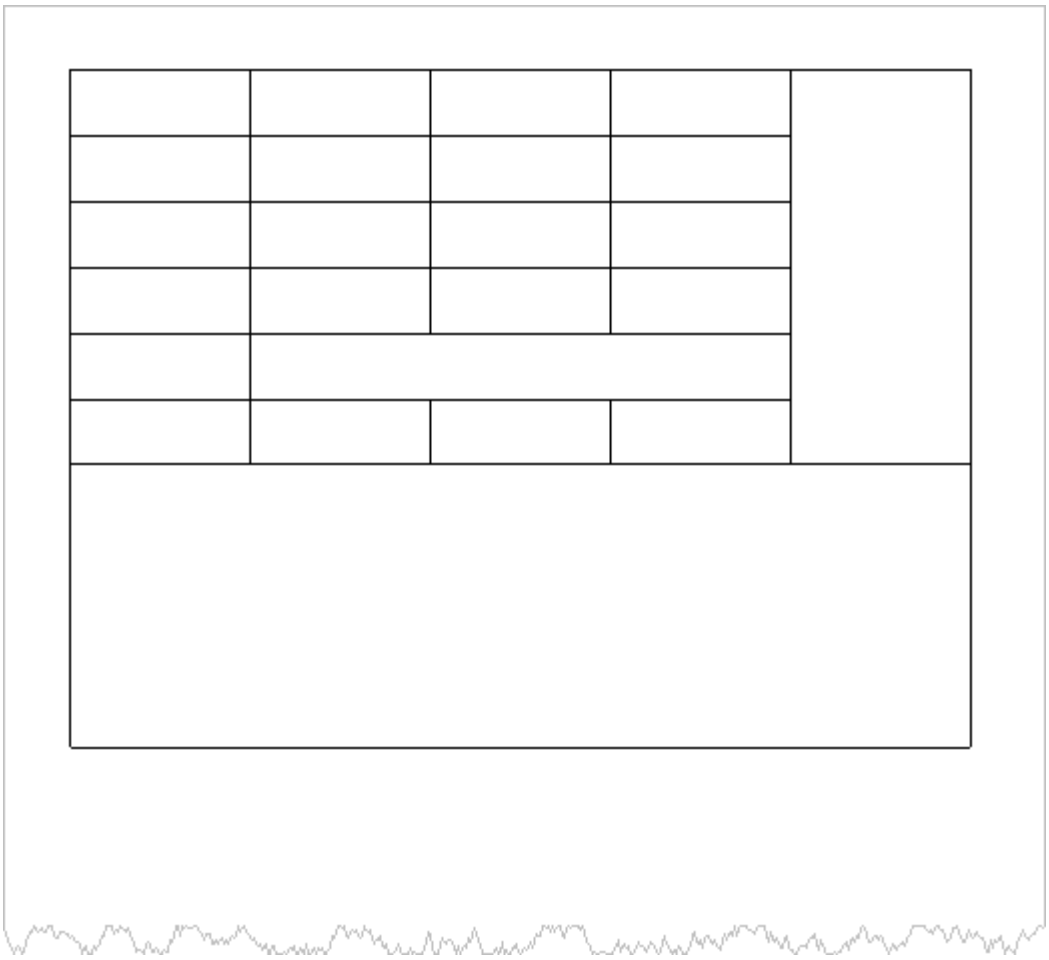


5.3.16 合并单元格

单元格有一个 SpanRows 属性，用于指定单元格向下合并的行数，另外有一个 SpanCols 属性，用于指定向右合并的列数。
示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.Rows.Count = 7 '设置总行数
rt.Cols.Count = 5 '设置总列数
rt.Height = 120 '设置表格的高度为 120 毫米
rt.Rows(6).Height = 50 '第 7 行的高度为 50 毫米
rt.Cells(0,4).SpanRows = 6 '第 1 行第 5 个单元格向下合并 6 行
rt.Cells(4,1).SpanCols = 3 '第 5 行第 2 个单元格向右合并 3 列
rt.Cells(6,0).SpanCols = 5 '第 7 行第 1 个单元格向右合并 5 列
Doc.Preview() '预览报表
```

执行结果：

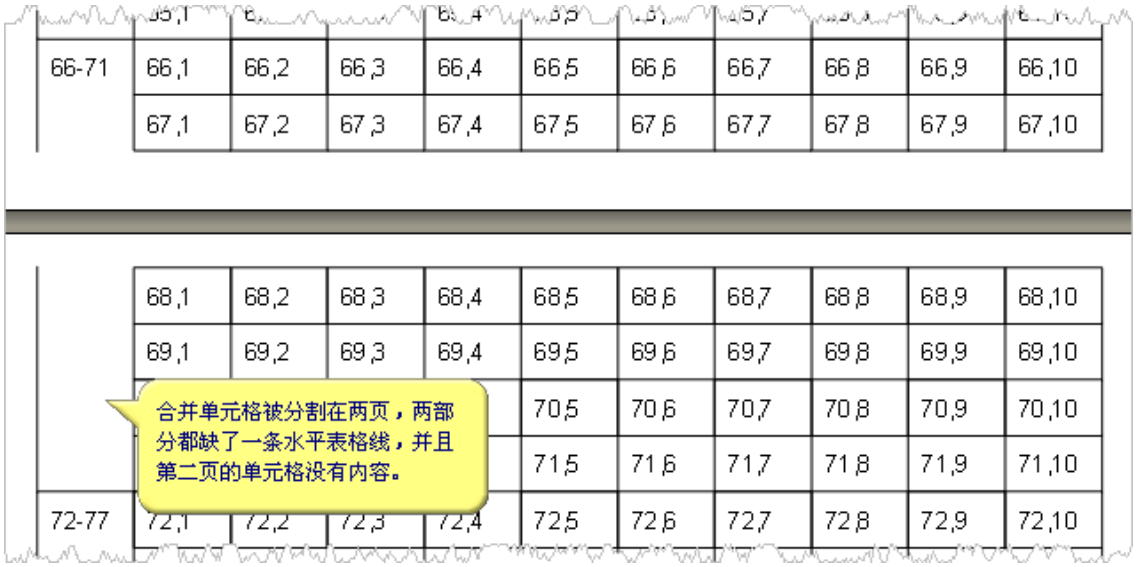


5.3.17 再谈合并单元格

在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable()
doc.Body.Children.Add(rt)
rt.Style.GridLines.All = New prt.Linedef
For r As integer = 0 to 100
    If r Mod 6 = 0 Then
        rt.Cells(r,0).Text = r & "-" & (r + 5)
        rt.Cells(r,0).SpanRows = 6
    End If
    For c As integer = 1 to 10
        rt.Cells(r, c).Text = r & "," & c
    Next
Next
doc.Preview()
```

在换页的位置会出现下图所示的情况：



- RenderTable 有一个属性 RepeatGridLinesVert，这是一个逻辑属性，用于设置单元格因为 垂直换页被分割为多个单元格时，是否重复打印水平表格线。

- 单元格有一个属性 VertSplitBehavior 属性，该属性是 CellSplitBehaviorEnum 性枚举，用于设置单元格因为 垂直换页被分割为多个单元格时，如何打印单元格内容。
  此枚举有三个可选值：

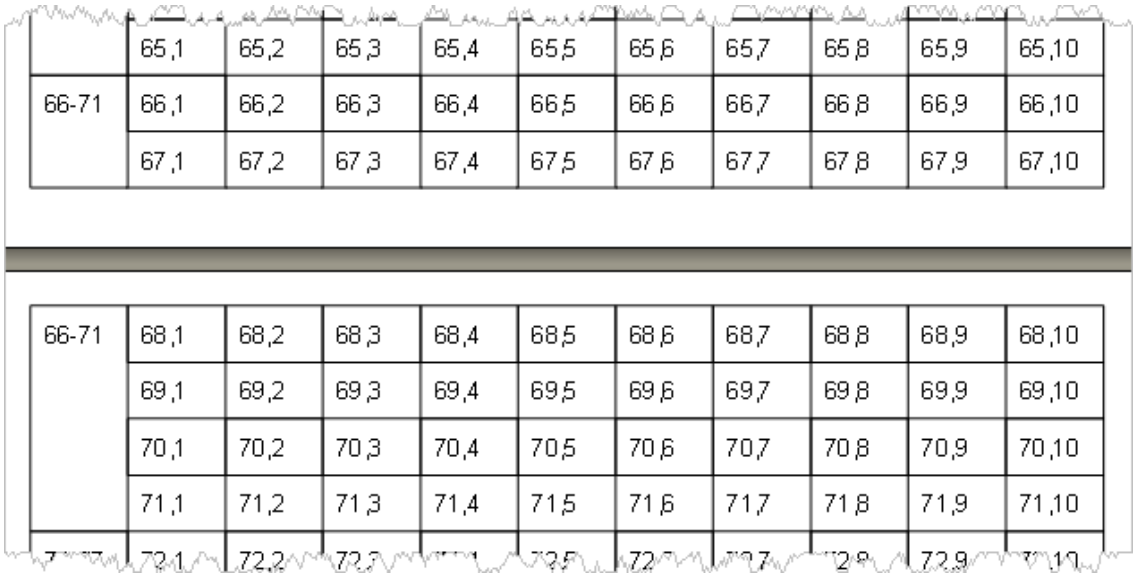
  Cut: 只打印一次
  Copy： 每次换页都打印
  Split：内容很长的时候，自动延伸到下一页。

综上所述，我们应该将代码改为:

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable()
doc.Body.Children.Add(rt)
rt.Style.GridLines.All = New prt.Linedef
rt.RepeatGridLinesVert = True
For r As integer = 0 to 100
    If r Mod 6 = 0 Then
        rt.Cells(r,0).Text = r & "-" & (r + 5)
        rt.Cells(r,0).SpanRows = 6
        rt.Cells(r,0).VertSplitBehavior = prt.CellSplitBehaviorEnum.Copy
    End If
    For c As integer = 1 to 10
        rt.Cells(r, c).Text = r & "," & c
    Next
Next
doc.Preview()
```

注意加粗的两行代码。

打印效果：



	65,1	65,2	65,3	65,4	65,5	65,6	65,7	65,8	65,9	65,10
66-71	66,1	66,2	66,3	66,4	66,5	66,6	66,7	66,8	66,9	66,10
	67,1	67,2	67,3	67,4	67,5	67,6	67,7	67,8	67,9	67,10

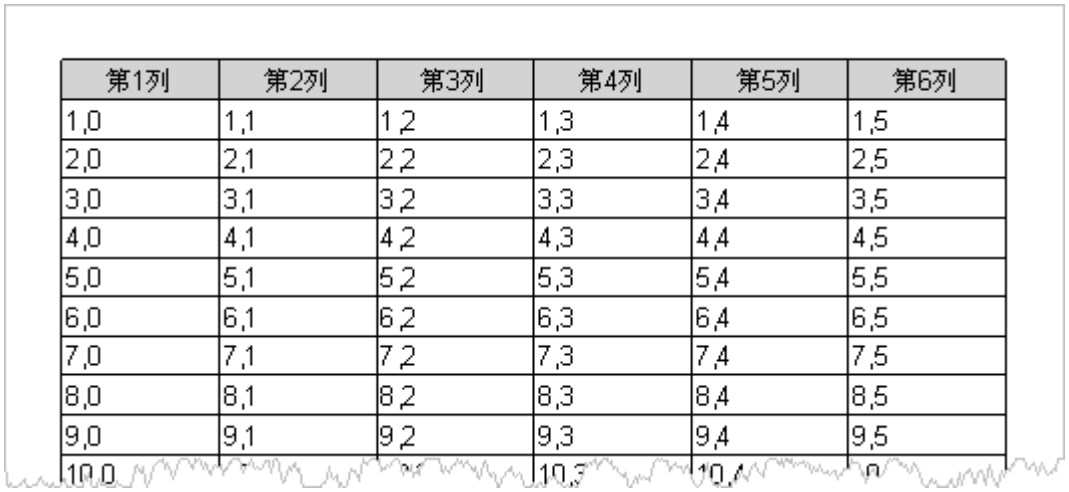
66-71	68,1	68,2	68,3	68,4	68,5	68,6	68,7	68,8	68,9	68,10
	69,1	69,2	69,3	69,4	69,5	69,6	69,7	69,8	69,9	69,10
	70,1	70,2	70,3	70,4	70,5	70,6	70,7	70,8	70,9	70,10
	71,1	71,2	71,3	71,4	71,5	71,6	71,7	71,8	71,9	71,10
	72,1	72,2	72,3						72,9	

5.3.18 丢失的表头

在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '将网格线设为默认类型
rt.CellStyle.Spacing.All = 0.5 '内容距离网格线 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.Rows(0).Style.TextAlignVert = prt.AlignVertEnum.Center '第一行内容垂直居中
rt.Rows(0).Style.BackColor = Color.LightGray '第一行背景颜色设为灰色。
'下面的代码向表格中填入值
For c As integer = 0 to 5
    rt.Cells(0, c).Text = "第" & (c + 1) & "列"
Next
For r As Integer = 1 To 200
    For c As integer = 0 to 5
        rt.Cells(r,c).Text = r & "," & c
    Next
Next
doc.Preview() '预览报表
```

执行结果：



第1列	第2列	第3列	第4列	第5列	第6列
1,0	1,1	1,2	1,3	1,4	1,5
2,0	2,1	2,2	2,3	2,4	2,5
3,0	3,1	3,2	3,3	3,4	3,5
4,0	4,1	4,2	4,3	4,4	4,5
5,0	5,1	5,2	5,3	5,4	5,5
6,0	6,1	6,2	6,3	6,4	6,5
7,0	7,1	7,2	7,3	7,4	7,5
8,0	8,1	8,2	8,3	8,4	8,5
9,0	9,1	9,2	9,3	9,4	9,5
10,0	[illegible]	[illegible]	10,3	10,4	[illegible]

显然，灰色的第一行是作为表头使用的，但是现在的问题是，这一行只在第一页打印，导致从第二页开始，就没有表头了。
如何才能让表头在每一页都打印呢，这是我们接下来要解决的问题。

5.3.19 行组与表头

所谓行组，就是将连续的几行组合在一起进行设置。
定义一个行组的语法是：

```
RenderTable.RowGroups(StartRow, Count)
```

StartRow: 起始行位置
Count: 行数

行组有 Style 属性，可以集中对其中的行进行设置。

例如下面的代码，将前两行的颜色设为灰色：

```
rt.RowGroups(0, 2).Style.BackColor = Color.LightGray '将前两行的背景颜色设为灰色。
```

行组还提供了一个 Header 属性，该属性为 TableHeaderEnum 型枚举，包括以下可选值：

- All
  在每一页的开始处重复行组。
  如果是分栏报表，则在每一栏的开始处重复。

- Column
  在每一栏的开始处重复行组。

- Page
  在每一页的开始处重复行组。

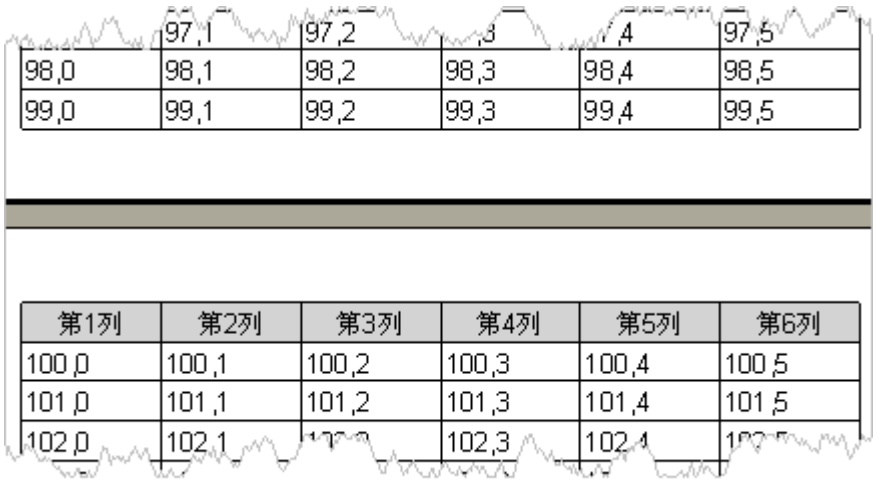
- None
  不重复，作为普通的行使用。

如果要将前几行作为表头使用，只需定义一个包括这几行的行组，然后将行组的 Header 属性设为 All 即可。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.CellStyle.Spacing.All = 0.5 '内容距离网格线 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.Rows(0).Style.TextAlignVert = prt.AlignVertEnum.Center '第一行内容垂直居中
rt.Rows(0).Style.BackColor = Color.LightGray '第一行背景颜色设为灰色。
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '将第一行作为表头。
'下面的代码向表格中填入值
For c As integer = 0 to 5
    rt.Cells(0, c).Text = "第" & (c + 1) & "列"
Next
For r As Integer = 1 To 200
    For c As integer = 0 to 5
        rt.Cells(r,c).Text = r & "," & c
    Next
Next
doc.Preview() '预览报表
```

执行上述代码后，你会发现每一页都会有表头了：



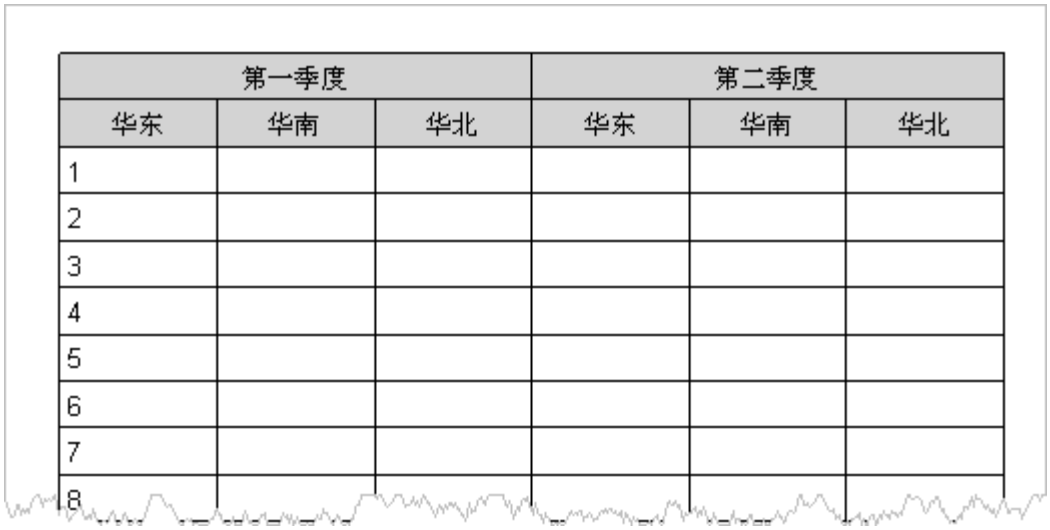
5.3.20 复杂的表头

现在，设计一个多层的复杂表头，是很容易的事。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.CellStyle.Spacing.All = 1 '内容距离网格线 1 毫米
rt.cells(0,0).Text = "第一季度" '设置第一行第一个单元格的内容
rt.Cells(0,3).Text = "第二季度" '设置第一行第四个单元格的内容
rt.cells(0,0).SpanCols = 3 '合并第一行前三个单元格
rt.Cells(0,3).SpanCols = 3 '合并第一行最后三个单元格
rt.Cells(1,0).Text= "华东"
rt.Cells(1,1).Text = "华南"
rt.Cells(1,2).Text = "华北"
rt.Cells(1,3).Text= "华东"
rt.Cells(1,4).Text = "华南"
rt.Cells(1,5).Text = "华北"
rt.RowGroups(0,2).Header = prt.TableHeaderEnum.All '前两行作为表头
rt.RowGroups(0,2).Style.BackColor = Color.LightGray '前两行的颜色设为灰色
rt.RowGroups(0,2).Style.TextAlignHorz = prt.AlignHorzEnum.Center '前两行的文本水平居中
rt.RowGroups(0,2).Style.TextAlignVert = prt.AlignVertEnum.Center '前两行的文本垂直居中
For i As integer = 1 To 100 '增加 100 行
    rt.Cells(i + 1, 0).Text = i
Next
Doc.Preview() '预览报表
```

下面是执行结果，是不是比划线的时代强多了：



第一季度			第二季度		
华东	华南	华北	华东	华南	华北
1					
2					
3					
4					
5					
6					
7					
8					

需要注意的是，用于表头的行必须位于表格的最前边。

5.3.21 每页都显示标题和表头

假定打印表的时候，希望每页都显示主标题、副标题和表头，如下图所示：



主标题内容

副标题内容

第一季度			第二季度		
华东	华南	华北	华东	华南	华北
1					
2					
3					
4					
5					

看起来有点麻烦，不过如果我们将主标题和副标题作为表头的一部分，那么问题迎刃而解，你可以在命令窗口测试下面的代码：

```
Dim doc As New PrintDoc   '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef   '设置网格线
rt.CellStyle.Spacing.All = 1 '内容距离网格线1毫米
'设置主标题
rt.Cells(0,0).text = "主标题内容"
rt.Cells(0,0).SpanCols = 6 '合并第一行全部单元格,用于显示主标题
rt.Cells(0,0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '主标题居中
rt.Cells(0,0).Style.Font = New Font("宋体", 16, FontStyle.Bold) '设置主标题字体
rt.Rows(0).Style.Borders.All = New prt.LineDef("0mm", Color.white) '去掉第一行的网格线
'设置副标题
```

```
rt.Cells(1,0).text = "                    副标题内容" '通过左边空格数量来调整副标题位置
rt.Cells(1,0).SpanCols = 6 '合并地二行全部单元格,用于显示副标题
rt.Cells(1,0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '副标题内容居中
rt.Rows(1).Style.Borders.All = New prt.LineDef("0mm", Color.white) '去掉第二行的网格线
rt.Rows(1).Style.Borders.Bottom = New prt.Linedef  '恢复第二行底端的网格线
rt.Rows(1).Height = 8 '设置第二行的高度,拉开和表格主体的距离.
'设置列标题
rt.cells(2,0).SpanCols = 3  '合并第一行前三个单元格
rt.Cells(2,3).SpanCols = 3  '合并第一行最后三个单元格
rt.cells(2,0).Text = "第一季度"  '设置第一行第一个单元格的内容
rt.Cells(2,3).Text = "第二季度"  '设置第一行第四个单元格的内容
rt.Cells(3,0).Text= "华东"
rt.Cells(3,1).Text = "华南"
rt.Cells(3,2).Text = "华北"
rt.Cells(3,3).Text= "华东"
rt.Cells(3,4).Text = "华南"
rt.Cells(3,5).Text = "华北"
rt.RowGroups(2,2).Style.BackColor = Color.LightGray '第三第四行的颜色设为灰色
rt.RowGroups(2,2).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第三第四行的文本水平居中
rt.RowGroups(2,2).Style.TextAlignVert = prt.AlignVertEnum.Center '第三第四行的文本垂直居中
rt.RowGroups(0,4).Header = prt.TableHeaderEnum.All  '前四行作为表头
'给表格增加行测试效果
For i As Integer =  2  To  100 '增加100行
    rt.Cells(i + 2,  0).Text = i
Next
Doc.Preview() '预览报表
```

5.3.22 列组

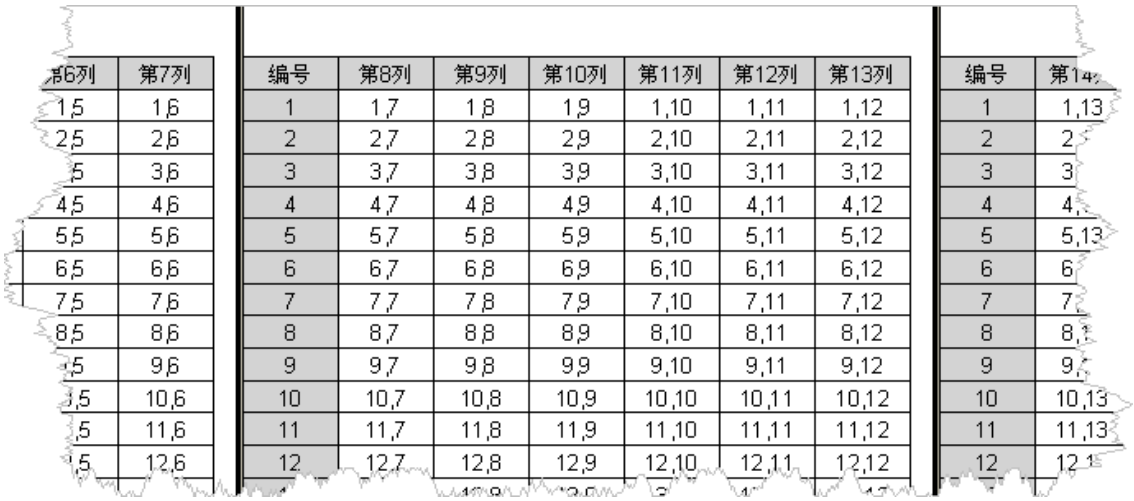
我们也可以将多列组合成一组进行设置，列组和行组一样，也有 Style 和 Header 属性。
假定一个表格很宽，以致水平方向要分页才行，我们希望最关键的一列，例如编号列，能够出现在每一页的左边。

示例：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.CellStyle.Spacing.All = 0.5 '内容距离网格线 0.5 毫米
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center '内容水平居中
rt.Style.TextAlignVert = prt.AlignVertEnum.Center '内容垂直居中
rt.Rows(0).Style.BackColor = Color.LightGray '第一行背景颜色设为灰色。
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '将第一行作为表头。
```

```
rt.Cols(0).Style.BackColor = Color.LightGray '第一列背景颜色设为灰色。
rt.ColGroups(0,1).Header = prt.TableHeaderEnum.All '将第一列作为表头。
rt.Width = "Auto" '表宽等于各列设置值之和
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时，可以水平换页
'下面的代码向表格中填入值
For c As Integer = 0 to 100
    rt.Cols(c).Width = 15 '明确指定每一列的宽度
    If c = 0 then
        rt.Cells(0,0). Text = "编号"
    Else
        rt.Cells(0, c).Text = "第" & (c + 1) & "列"
    End If
Next
For r As Integer = 1 To 50
    rt.Cells(r,0).Text = r
    For c As integer = 1 to 100
        rt.Cells(r,c).Text = r & "," & c
    Next
Next
doc.Preview() '预览报表
```

执行后，你会发现编号列显示在每一页的左边：



第6列	第7列	编号	第8列	第9列	第10列	第11列	第12列	第13列	编号	第14
1,5	1,6	1	1,7	1,8	1,9	1,10	1,11	1,12	1	1,13
2,5	2,6	2	2,7	2,8	2,9	2,10	2,11	2,12	2	2
3,5	3,6	3	3,7	3,8	3,9	3,10	3,11	3,12	3	3
4,5	4,6	4	4,7	4,8	4,9	4,10	4,11	4,12	4	4
5,5	5,6	5	5,7	5,8	5,9	5,10	5,11	5,12	5	5,13
6,5	6,6	6	6,7	6,8	6,9	6,10	6,11	6,12	6	6
7,5	7,6	7	7,7	7,8	7,9	7,10	7,11	7,12	7	7
8,5	8,6	8	8,7	8,8	8,9	8,10	8,11	8,12	8	8
9,5	9,6	9	9,7	9,8	9,9	9,10	9,11	9,12	9	9
10,5	10,6	10	10,7	10,8	10,9	10,10	10,11	10,12	10	10,13
11,5	11,6	11	11,7	11,8	11,9	11,10	11,11	11,12	11	11,13
12,5	12,6	12	12,7	12,8	12,9	12,10	12,11	12,12	12	12,1

再次提醒一下，如果需要表格的宽度等于各列的列宽设置值之和，必须满足三个条件：

1、表格的宽度设为“Auto”：

```
rt.Width = "Auto" '表宽等于各列设置值之和
```

2、如果宽度超过页宽，那么必须允许表格水平换页：

```
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded
```

3、每一列都必须设置列宽，不能漏掉任何一列，否则表宽始终等于页宽。

5.3.23 表尾

和表头(Header)对应的是表尾(Footer)

行组和列组都有一个 Footer 属性，该属性为 TableFooterEnum 型枚举，包括以下可选值：

- All
  在每一页的结尾处重复行组。
  如果是分栏报表，则在每一栏的结尾处重复。

- Column
  在每一栏的结尾处重复行组。

- Page
  在每一页的结尾处重复行组。

- None
  不重复，作为普通的行使用。

5.3.24 用表格设计标签

如果不需要很精确的定位，用表格设计标签更为方便。

示例：

```
Dim doc As New PrintDoc '定义一个报表
doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight '设置排列方式
For i As integer = 1 to 100
    Dim rt As New prt.RenderTable() '定义一个表格对象
    rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
    rt.Width = 50 '表格宽度为 50mm
    rt.Style.Spacing.All = 2 '和其他对象之间的间隔为 2mm
    rt.SplitVertBehavior = prt.SplitBehaviorEnum.Never '避免垂直换页的时候，表格被分割成两部分。
    rt.Cols(0).Width = 20
    rt.cells(0,0).Text = "姓名"
    rt.Cells(0,1).Text = "黄财德"
    rt.Cells(1,0).Text = "职务"
    rt.Cells(1,1).Text = "总经理"
    rt.Cells(2,0).Text= "工号"
    rt.cells(2,1).Text = "44081996008"
    doc.Body.Children.Add(rt) '将表格对象加入到报表中
```

```
Next
Doc.Preview() '预览报表
```

下图是执行结果，每一个表格就是一个标签：



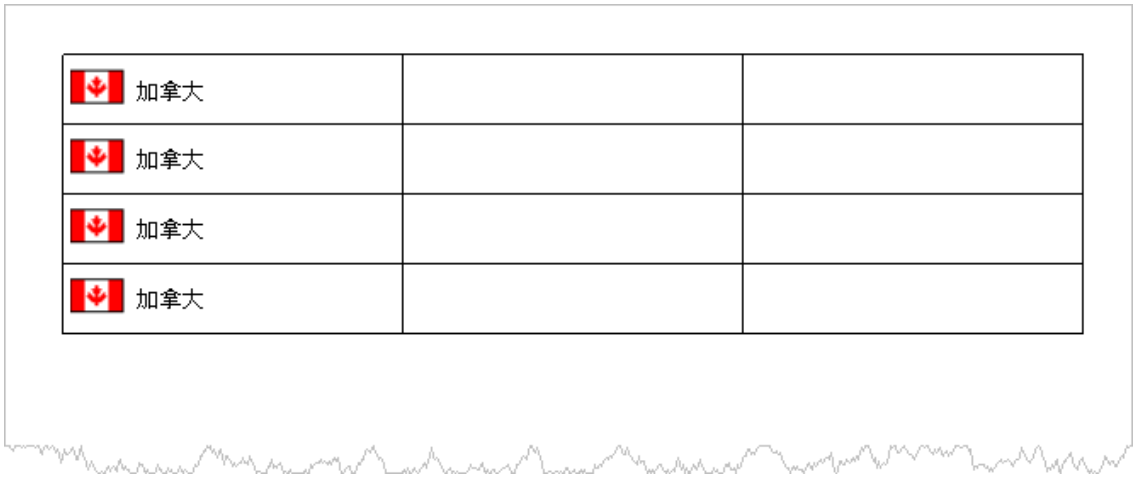
5.3.25 单元格的容器

我们已经提到过，一个单元格就是一个容器，单元格有一个 Area 属性，可以返回单元格的容器(RenderArea)，既然是容器，肯定可以容纳多个对象。

示例

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim ra As prt.RenderArea
Dim rm As prt.RenderImage
Dim rx As prt.RenderText
rt.Style.GridLines.All = New prt.LineDef
rt.CellStyle.Spacing.All = 1 '单元格内容和网格线距离 1 毫米
rt.Cols.Count = 3 '总列数为 3
For i As Integer = 0 To 3
    ra = rt.Cells(i,0).Area '引用单元格的容器
    rm = New prt.RenderImage '创建一个图片
    rx = New prt.RenderText '创建一个文本
    ra.Stacking = prt.StackingRulesEnum.BlockLeftToRight '排列方式改为从左到右
    '设置图片对象
    rm.Image = GetImage("Can.ico")
    rm.Style.Spacing.Right = 2 '距离右边的对象 2 毫米
    '设置文本对象
    rx.Text = "加拿大"
    rx.Height = "Prev.Height" 'rx 的高度等于图片对象 rm 的高度
    rx.Style.TextAlignVert = prt.AlignVertEnum.Center '文本垂直居中
    '设置 RenderArea，并将文本和图片加入到 RenderArea
    ra.Children.Add(rm) '加入图片到 RenderArea 中
    ra.Children.Add(rx) '加入文本到 RenderArea 中
Next
```

doc.Body.Children.Add(rt)
Doc.Preview()
执行结果



需要特别留意的的是，容器的高度等于最高子对象的高度，也就是图片的高度，为了让文本对象的内容能够在单元格中垂直居中，必须将文本对象的高度设置为图片对象的高度：

```
rx.Height = "Prev.Height" 'rx 的高度等于图片对象 rm 的高度
```

你可以删除这一行，体会一下该行代码所带来的不同。

5.3.26 表格与页眉页脚

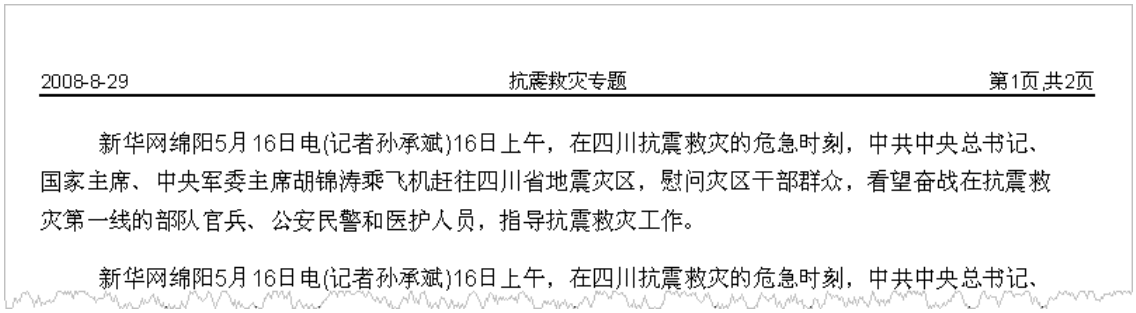
有的时候，页眉页脚可能要分为几个部分，例如左、中、右，对于这种情况，用表格来处理很方便。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
Dim s As String '定义字符变量
s = "新华网绵阳 5 月 16 日电(记者孙承斌)16 日上午，在四川抗震救灾的"
s = s & "危急时刻，中共中央总书记、国家主席、中央军委主席胡锦涛乘飞机赶往四川"
s = s & "省地震灾区，慰问灾区干部群众，看望奋战在抗震救灾第一线的部队官兵、公"
s = s & "安民警和医护人员，指导抗震救灾工作。"
For i As Integer = 1 to 20
    rt = New prt.RenderText
    rt.Text = s '设置文本对象的内容
    rt.Style.TextIndent = 9 '首行缩进 9 毫米
    rt.Style.Spacing.Top = 5 '每段间隔 5 毫米
```

```
    rt.Style.LineSpacing = 150 '1.5 倍行距
    doc.Body.Children.Add(rt) '将文本对象加入到报表
Next
'设置页眉
Dim rx As New prt.RenderTable
rx.Cells(0,0).Text = Date.Today
rx.Cells(0,1).Text = "抗震救灾专题"
rx.Cells(0,2).Text = "第[PageNo]页,共[PageCount]页"
rx.Cols(0).Style.TextAlignHorz = prt.AlignHorzEnum.Left
rx.Cols(1).Style.TextAlignHorz = prt.AlignHorzEnum.Center
rx.Cols(2).Style.TextAlignHorz = prt.AlignHorzEnum.right
rx.Style.Borders.Bottom = New prt.LineDef '设置底边框
rx.CellStyle.Spacing.Bottom = 0.5 '底端内容缩进 0.5 毫米
rx.Style.FontSize = 8 '字体大小为 8 磅
Doc.PageHeader = rx '作为页眉使用
doc.Preview() '预览
```

执行结果：



5.4 其他对象

5.4.1 线条

5.4.1.1 绘制线条

定义一个线条的语法是：

New prt.RenderLine(X1, Y1, X2, Y2, LineDef)

X1、Y1： 整数型，线条起始坐标位置

X2、Y2： 整数型，线条结束坐标位置

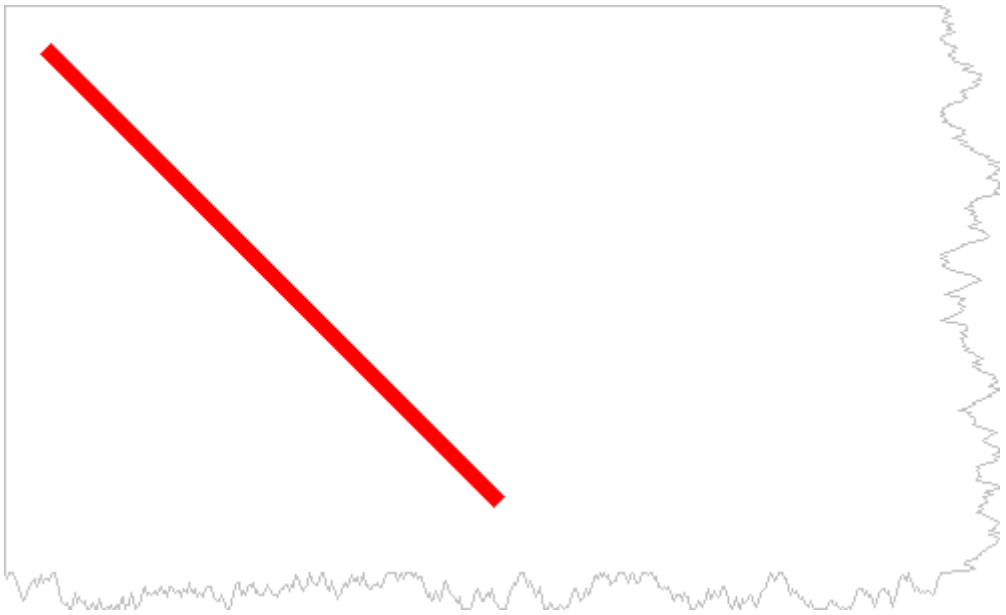
LineDef: 线条样式

示例

```
Dim doc As New PrintDoc '定义一个报表
```

```
Dim Line As New prt.LineDef(2, Color.Red)
Dim ln As New prt.RenderLine(20,20, 80, 80, Line)
doc.body.Children.Add(ln) '将线条加入到报表中
Doc.Preview() '预览报表
```

执行结果



创建一个 Linedef 的语法是：

```
New prt.Linedef()
New prt.Linedef(Color)
New prt.Linedef(Width, Color)
New prt.Linedef(Width, Color, DashStyle)
```

Color:　边框颜色。

Width:　边框宽度，单位为毫米。

DashStyle：DashStyle 型枚举，包括以下可选值：

Dash　由线段构成的直线。

DashDot　由线段和点间隔构成的直线。

DashDotDot　由线段和连续两个点间隔构成的直线

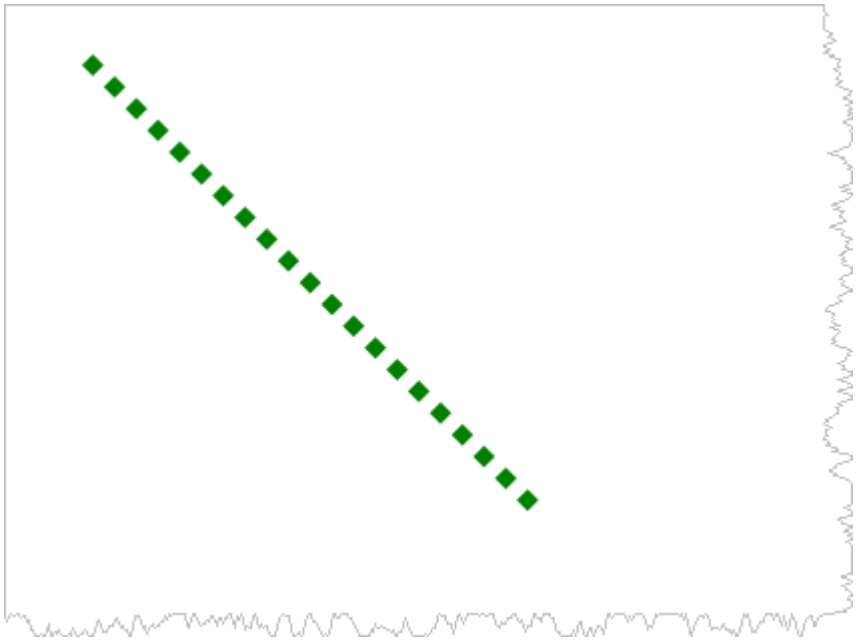
Dot　由点构成的直线。

Solid　实线，这是默认值。

示例：

```
Dim doc As New PrintDoc '定义一个报表
Dim Line As New prt.LineDef(2, Color.Green, DashStyle.Dot)
Dim ln As New prt.RenderLine(20,20, 80, 80, Line)
doc.body.Children.Add(ln) '将线条加入到报表中
Doc.Preview() '预览报表
```

执行结果



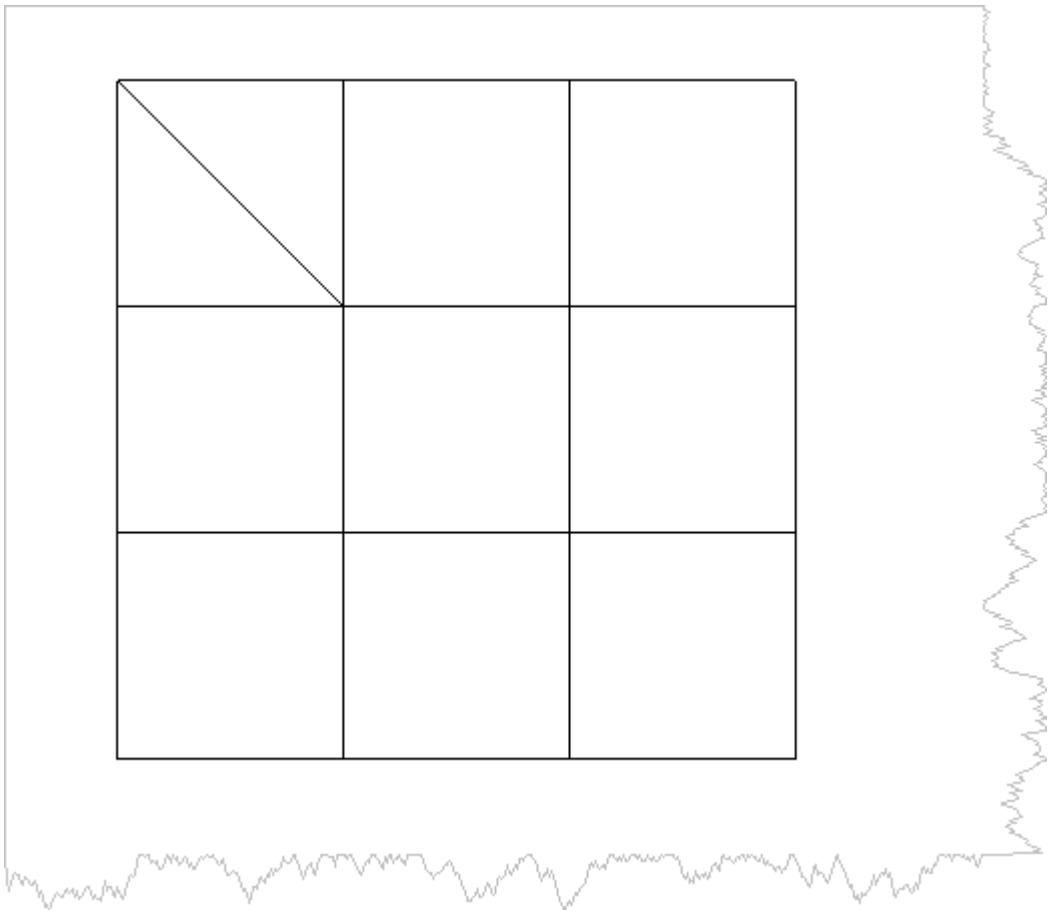
5.4.1.2 在单元格中绘制对角线

要在单元格格中绘制一个线条，可以先定义一个线条，其起始位置设为(0,0)，终止位置随便指定，然后将线条加入到单元格中，随后指定线条的 Width 和 Height 属性。

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
doc.Body.Children.Add(rt) '将表格对象加入到报表中
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Width = 90 '表宽为 90 毫米
rt.Height = 90 '表高为 90 毫米
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
Dim ln As New prt.RenderLine(0,0,1,1,New prt.Linedef) '定义一个线条
rt.Cells(0,0).RenderObject = ln '将线条加入到单元格中。
ln.Width = "Parent.Width"
ln.Height = "Parent.Height"
Doc.Preview() '预览报表
```

执行结果：



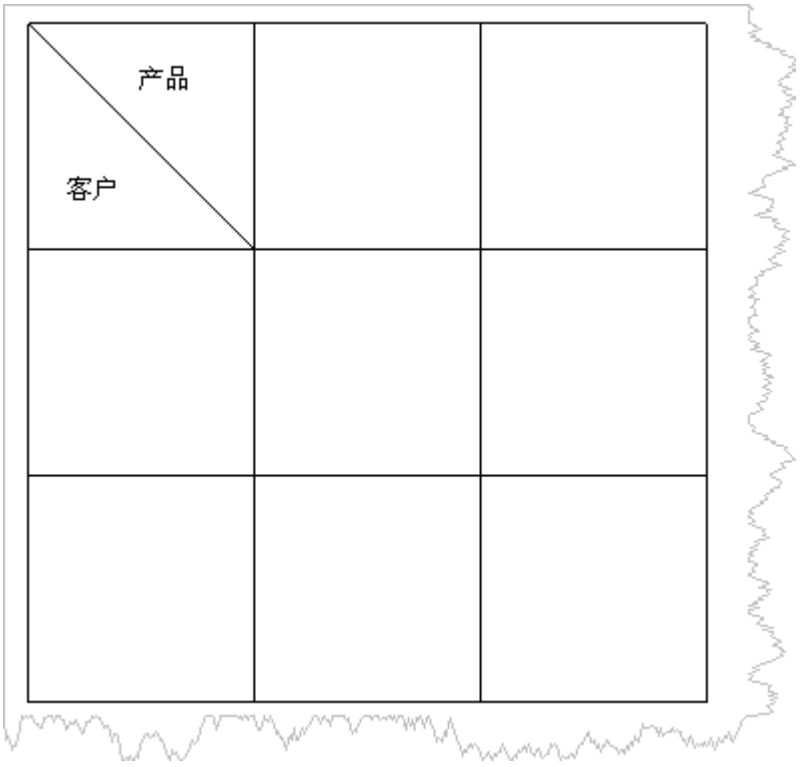
5.4.1.3 在单元格中同时显示文本和线条

示例

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
Dim ra As prt.RenderArea
doc.Body.Children.Add(rt) '将表格对象加入到报表中
ra = rt.Cells(0,0).Area '引用第一个单元格的容器。
rt.Style.GridLines.All = New prt.LineDef '将网格线类型设为默认类型
rt.Width = 90 '表宽为 90 毫米
rt.Height = 90 '表高为 90 毫米
rt.Rows.Count = 3 '设置行数
rt.Cols.Count = 3 '设置列数
'加入线条
Dim ln As New prt.RenderLine(0,0,1,1,New prt.Linedef)
ln.Width = "Parent.Width"
ln.Height = "Parent.Width"
ra.Children.Add(ln)
'加入第一个文本
Dim rx As New prt.RenderText
```

```
rx.Text = "产品"
rx.X = "Parent.Width - 15"
rx.y = "5"
ra.Children.Add(rx)
'加入第二个文本
rx = New prt.RenderText
rx.Text = "客户"
rx.X = "5"
rx.y = "Parent.Height - 10"
ra.Children.Add(rx)
Doc.Preview() '预览报表
```

执行结果：



实际使用的时候，请根据需要调整文本的坐标位置。

5.4.2 条形码

RenderBarCode 专门用于打印条形码，包括四个属性：

属性名	说明
BarCodeType	BarCodeEnum 型枚举，用于指定条形码类型，包括以下可选值： Codabar Code128 Code39 Code93

	CodeI2of5 Ean13 Ean8 PostNet UpcA None
BarDirection	BarDirectionEnum 型枚举，用于指定条形码方向，包括以下可选值： Normal：正常，也就是水平 Down：向下 Up：向上
ShowText	逻辑型，是否显示编码
Text	编码

示例

```
Dim doc As New PrintDoc
Dim rbc As New prt.RenderBarCode()
rbc.Height = 25
rbc.BarCodeType = BarCodeEnum.Code39
rbc.BarDirection = BarDirectionEnum.Normal
rbc.Text = "123456"
rbc.ShowText = True
doc.Body.Children.Add(rbc)
Doc.Preview()
```

执行结果：



提示：一般只需设置条形码的高度，宽度无需设置，会自动根据比例缩放；打印的条形码不能太大或太小，否则都会影响识别。

5.4.3 弧形

RenderArc 用于打印弧形，定义一个弧形的语法是：

New prt.RenderArc(Width, Height, startAngle, sweepAngle, Line, FillColor)

Width：　宽度 ，弧形属于椭圆的一部分，而此属性指定的就是此椭圆的宽度

Height:　高度，弧形属于椭圆的一部分，而此属性指定的就是此椭圆的高度

startAngle: 起始角度

sweepAngle: 扫过的角度值
Line: 可选参数，边框线条类型，LineDef 型枚举
FillColor: 可选参数，扇形填充颜色
示例

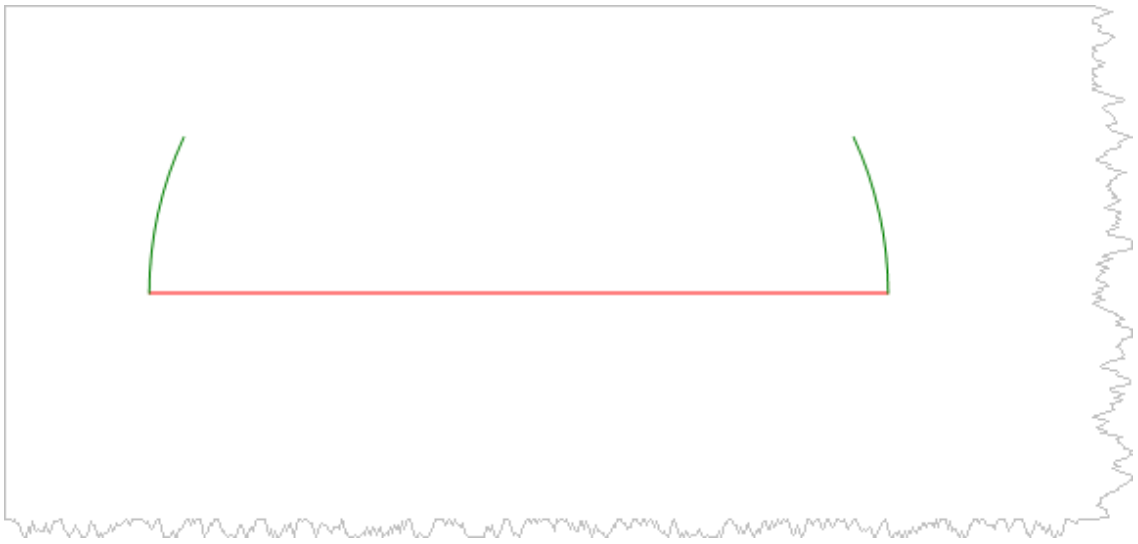
```
Dim doc As New PrintDoc
Dim Line As New prt.LineDef(Color.Red)
Dim ln As New prt.RenderLine(50,100,150, 100, Line)
doc.body.Children.Add(ln)

Dim Arc1 As New prt.RenderArc(100, 100, 0, -25, new prt.Linedef(Color.Green))
arc1.x = 50
arc1.y = 50
doc.Body.Children.Add(Arc1)

Dim Arc2 As New prt.RenderArc(100, 100, -155, -25, new prt.Linedef(Color.Green))
arc2.x = 50
arc2.y = 50

doc.Body.Children.Add(Arc2)
Doc.Preview()
```

执行结果：



5.4.4 椭圆形

RenderEllipse 用于打印椭圆形，定义一个椭圆的语法是：

New prt.RenderEllipse(Width, Height, Line, FillColor)
Width： 宽度
Height: 高度
Line: 边框线条类型，LineDef 型枚举，可选参数
FillColor: 填充颜色，可选参数
示例

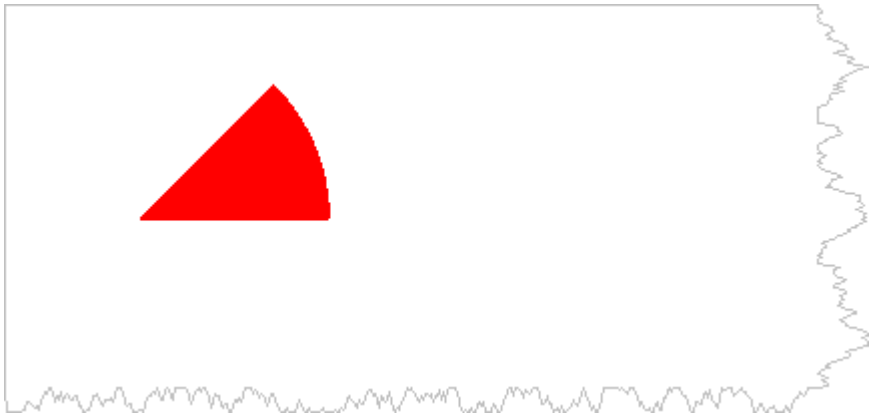
```
Dim doc As New PrintDoc
Dim esp As New prt.RenderEllipse(80,60)
esp.x = 25
esp.y = 25
doc.Body.Children.Add(esp)
Doc.Preview()
```

5.4.5 饼形

RenderPie 用于打印饼形，定义一个饼形的语法是：
New prt.RenderPie (Width, Height, startAngle, sweepAngle, Line, FillColor)
Width： 宽度
Height: 高度
startAngle: 起始角度
sweepAngle: 扫过的角度值
Line: 边框线条类型，LineDef 型枚举，可选参数
FillColor: 填充颜色，可选参数
示例

```
Dim doc As New PrintDoc
Dim pie As New prt.RenderPie(20,20, 50, 50, 0, -45, new prt.Linedef(Color.Red), Color.Red)
pie.x = 30
pie.y = 40
doc.Body.Children.Add(pie)
Doc.Preview()
```

执行结果：

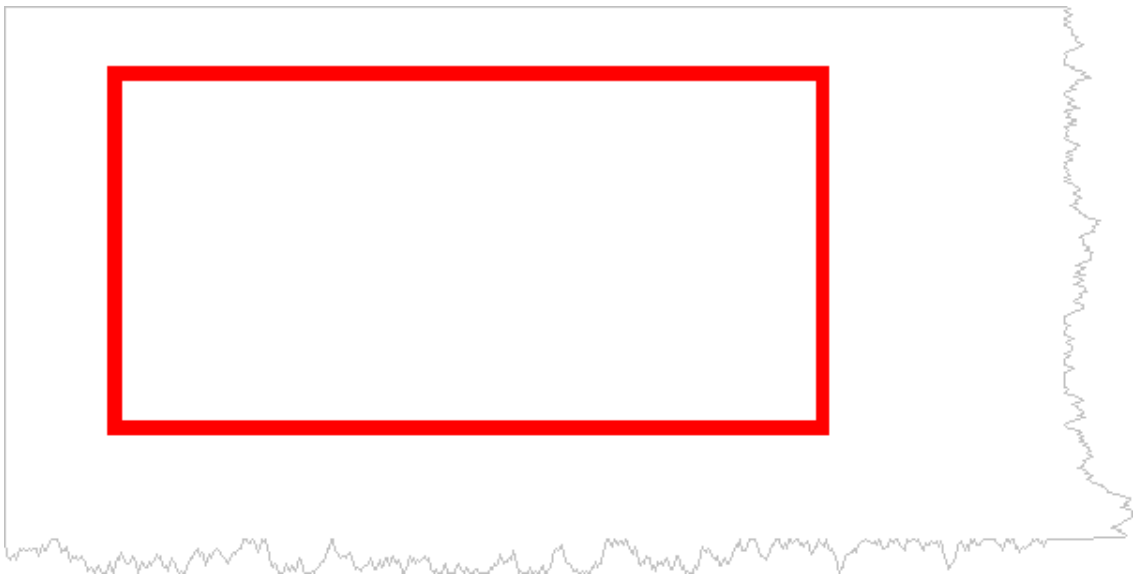


5.4.6 四边形

RenderEllipse 用于打印四边形，定义一个四边形的语法是：
New prt.RenderRectangle (Width, Height, Line, FillColor)
Width： 宽度
Height: 高度
Line: 边框线条类型，LineDef 型枚举，可选参数
FillColor: 填充颜色，可选参数
示例

```
Dim doc As New PrintDoc
Dim rg As New prt.RenderRectangle (100,50,New Prt.Linedef(2,Color.Red))
rg.x = 50
rg.y = 50
doc.Body.Children.Add(rg)
Doc.Preview()
```

执行结果：



5.4.7 空对象

如果给打印对象指定了坐标位置，那么其 BreakAfter 和 BreakBefore 属性是失效的；此时为了强制换页，我们可以加入一个 RenderEmpty 对象，RenderEmpty 是一个空对象，不会打印任何内容；设置 RenderEmpty 对象的 BreakAfter 或 BreakBefore 属性，不指定其坐标，加入到报表中，即可实现强制换页、换栏，换行。
示例
例如下面的代码，由于所有的打印对象都需要指定坐标位置，所以只有通过增加一个 RenderEmpty 对象来控制换页：

```
Dim doc As New PrintDoc() '定义一个报表
Dim rt As prt.RenderText '定义一个文本对象
Dim rm As prt.RenderEmpty '定一个空对象
```

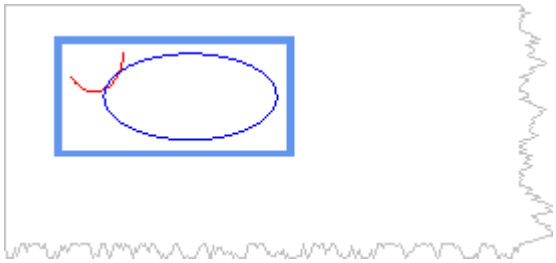
```
For n As integer =1 To 2
    rm = new prt.RenderEmpty '定义一个新的空对象
    rm.BreakBefore = prt.BreakEnum.Page '打印前换页
    doc.Body.Children.Add(rm) '加入到报表中
    For i AS integer = 1 To 5
        rt = New prt.RenderText() '创建文本对象
        rt.Text = "Hello Foxtable " & i '设置文本对象的内容
        rt.Width = "Auto" '自动设置宽度
        rt.X = i * 10 + 20 '指定水平位置
        rt.y = i * 10 + 20 '指定垂直位置
        rt.Style.Borders.All = New prt.Linedef(1, Color.Red) '设置边框
        doc.Body.Children.Add(rt) '将文本对象加入到报表
    Next
Next
doc.Preview() '预览
```

5.4.8 RenderGraphics

这个打印对象给高级用户使用，可以用 GDI+类库在其中随意绘制图形、文本。
大家可以忽略此对象，要知道 GDI+编程本身也可以写一本中等厚度的书。
示例

```
Dim doc As New PrintDoc
Dim rg1 As New prt.RenderGraphics()
rg1.Graphics.DrawArc(Pens.Red, 0, 0, 30, 40, 0, 135)
rg1.Graphics.DrawEllipse(Pens.Blue, 20, 20, 90, 45)
rg1.Style.Borders.All = New prt.LineDef("1mm", Color.CornflowerBlue)
rg1.x = 30
rg1.y = 30
Doc.Body.Children.Add(rg1)
Doc.Preview()
```

执行结果：



Foxtable 提供了一个 GDI+快速入门，虽然是针对窗口的，但是其中 Graphics 对象的属性和方法是完全通用的，大家可以参考一下。
GDI+快速入门

5.5 实战演练

5.5.1 打印表

5.5.1.1 通用代码

数据表通用打印代码：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("订单")
rt.Width = "Auto" '表格宽度为自动，也就是等于各列设置宽度之和
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时，可以水平换页
rt.Style.Font = tb.Font
For c As Integer = 0 To tb.Cols.Count -1 '逐列设置和填入内容
    rt.Cells(0,c).Text = tb.Cols(c).Name '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    rt.Cols(c).Width = tb.Cols(c).PrintWidth '列宽等于实际列宽
    If tb.Cols(c).IsNumeric Orelse tb.Cols(c).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = tb(r,c)
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组，将第一行设为表头
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

上述代码会打印所有行，如果只是希望打印选定行，代码如下：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb As Table = Tables("订单")
rt.Width = "Auto" '表格宽度为自动,也就是等于各列设置宽度之和
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时,可以水平换页
rt.Style.Font = tb.Font
For c As Integer = 0 To tb.Cols.Count -1 '逐列设置和填入内容
    rt.Cells(0,c).Text = tb.Cols(c).Name '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    rt.Cols(c).Width = tb.Cols(c).PrintWidth '列宽等于实际列宽
```

```
    If tb.Cols(c).IsNumeric OrElse tb.Cols(c).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As Integer = tb.TopPosition To tb.BottomPosition '开始填入该列内容
        rt.Cells(r + 1 - tb.TopPosition , c).Text = tb(r,c)
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组,将第一行设为表头
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

上述的代码，会打印所有列，包括隐藏列，如果要排除隐藏列，可以采用下面的代码：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("订单")
Dim ColNames As New List(Of String)
For Each cl As Col In tb.Cols '排除隐藏列
    If cl.Visible Then
        ColNames.Add(cl.Name)
    End If
Next
rt.Width = "Auto" '表格宽度为自动,也就是等于各列设置宽度之和
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时，可以水平换页
rt.Style.Font = tb.Font
For c As Integer = 0 To ColNames.Count - 1 '逐列设置和填入内容
    rt.Cells(0,c).Text = ColNames(c) '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth '列宽等于实际列宽
    If tb.Cols(ColNames(c)).IsNumeric Orelse tb.Cols(ColNames(c)).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = tb.Rows(r)(ColNames(c))
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组,将第一行设为表头.
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

如果希望缩放表格的宽度，使得恰好等于页宽，可以删除这一行：

```
rt.Width = "Auto" '表格宽度为自动，也就是等于各列设置宽度之和
```

5.5.1.2 标出感兴趣的数据

用红色加粗字体标出金额超出一万的订单：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("订单")
rt.Width = "Auto" '表格宽度为自动，也就是等于各列设置宽度之和
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时，可以水平换页
rt.Style.Font = tb.Font
For c As Integer = 0 To tb.Cols.Count -1 '逐列设置和填入内容
    rt.Cells(0,c).Text = tb.Cols(c).Name '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    rt.Cols(c).Width = tb.Cols(c).PrintWidth '列宽等于实际列宽
    If tb.Cols(c).IsNumeric Orelse tb.Cols(c).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = tb(r,c)
        If tb.Cols(c).Name = "金额" Then
            If tb.Rows(r)("金额") > 10000 Then
                rt.Cells(r + 1, c).Style.TextColor = Color.Red
                rt.Cells(r + 1, c).Style.FontBold = True
            End If
        End If
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组，将第一行设为表头
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

5.5.1.3 打印部分列

打印部分列的通用代码为：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("订单")
Dim ColNames As String() = New String(){"产品", "日期", "数量"}
rt.Width = "Auto" '表格宽度为自动，也就是等于各列设置宽度之和
```

```
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时，可以水平换页
rt.Style.Font = tb.Font
For c As Integer = 0 To ColNames.Length - 1 '逐列设置和填入内容
    rt.Cells(0,c).Text = ColNames(c) '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth '列宽等于实际列宽
    If tb.Cols(ColNames(c)).IsNumeric Orelse tb.Cols(ColNames(c)).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = tb.Rows(r)(ColNames(c))
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组，将第一行设为表头。
doc.Body.Children.Add(rt) '将表格加入到报表
'如果需要分栏打印，请加上下面两行
'doc.Columns.Add()
'doc.Columns.Add()
doc.Preview()
```

说明：

所有的代码我们都接触过，唯独这一句：

```
New String(){"产品", "日期", "数量"}
```

表示新建一个字符型数组，大括号中内容为数组内容，所以：

```
Dim ColNames As String() = New String(){"产品", "日期", "数量"}
```

等同于：

```
Dim ColNames(2) As String
ColNames(0) = "产品"
ColNames(1) = "数量"
ColNames(2) = "日期"
```

5.5.1.4 打印行号列

如果要打印行号列，可以参考下面的代码：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb As Table = Tables("订单")
Dim ColNames As New List(Of String)
For Each cl As Col In tb.Cols '排除隐藏列
    If cl.Visible Then
        ColNames.Add(cl.Name)
    End If
```

```
Next
rt.Width = "Auto" '表格宽度为自动,也就是等于各列设置宽度之和
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded '表格宽度超出页宽时,可以水平换页
rt.Style.Font = tb.Font
For c As Integer = 0 To ColNames.Count - 1 '逐列设置和填入内容
    rt.Cells(0,c).Text = ColNames(c) '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth '列宽等于实际列宽
    If tb.Cols(ColNames(c)).IsNumeric OrElse tb.Cols(ColNames(c)).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As Integer = 0 To tb.Rows.Count - 1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = tb.Rows(r)(ColNames(c))
    Next
Next
rt.Cols.Insert(0) '在左边插入一列,用于打印行号
rt.Cols(0).Width = 10 '设置行号列的宽度
For i As Integer = 1 To rt.Rows.Count - 1
    rt.Cells(i,0).text = i '逐行写入行号
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组,将第一行设为表头.
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

上述代码中，加粗的几行用于打印行号，其原理很简单： 按常规生成要打印的表，然后在左边插入一列，按顺序自动填入行号。

5.5.1.5 自动生成多层表头

利用专业报表自己编码打印表的时候，如果遇到多层表头，这对于大多数人来说，都是一个难题。Table 有一个 CreateReportHeader 方法，可以为 RenderTable 自动生成多层表头。

语法：

CreateReportHeader(RenderTable, VisibleOnly)

RenderTable：要生成表头的 RenderTable

VisibleOnly：可选参数，是否只包括可见列，默认为 True。

示例

如果要打印具有多层表头的 Table，可以参考下面的代码。

包括隐藏列：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("产品")
Dim hd As Integer = tb.HeaderRows '获得表头的层数
```

```
rt.Width = "Auto" '表格宽度为自动,也就是等于各列设置宽度之和
rt.Style.Font = tb.Font
tb.CreateReportHeader(rt,False) '生成表头,包括所有列
For c As Integer = 0 To tb.Cols.Count -1 '逐列设置和填入内容
    rt.Cols(c).Width = tb.Cols(c).PrintWidth '列宽等于实际列宽
    If tb.Cols(c).IsNumeric Orelse tb.Cols(c).IsDate Then '如果是数值或日期列
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
    End If
    For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
        rt.Cells(r + hd, c).Text = tb(r,c)
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.RowGroups(0, tb.HeaderRows).Header = prt.TableHeaderEnum.All '利用行组功能设置表头
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

排除隐藏列：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("产品")
Dim hd As Integer = tb.HeaderRows '获得表头的层数
Dim cnt As Integer '用于记录列位置
rt.Width = "Auto" '表格宽度为自动,也就是等于各列设置宽度之和
rt.Style.Font = tb.Font
tb.CreateReportHeader(rt,true) '生成表头,排除隐藏列
For c As Integer = 0 To tb.Cols.Count -1 '逐列设置和填入内容
    If tb.Cols(c).Visible Then
        rt.Cols(cnt).Width = tb.Cols(c).PrintWidth '列宽等于实际列宽
        If tb.Cols(c).IsNumeric Orelse tb.Cols(c).IsDate Then '如果是数值或日期列
            rt.Cols(cnt).Style.TextAlignHorz = prt.AlignHorzEnum.Right '数据水平靠右
        End If
        For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
            rt.Cells(r + hd, cnt).Text = tb(r,c)
        Next
        cnt = cnt + 1
    End If
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.RowGroups(0, tb.HeaderRows).Header = prt.TableHeaderEnum.All '利用行组功能设置表头
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

5.5.1.6 合并单元格

下面的代码打印订单表，并合并前两列，也就是产品和客户两列：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim tb As Table = Tables("订单")
Dim ColNames As String() = New String(){"产品", "客户","单价","折扣","数量","金额","日期"}
Dim MergeCols As Integer = 2 '指定要合并的列数
tb.Sort = "产品,客户" '根据合并列排序
rt.Width = "Auto"
rt.SplitHorzBehavior = prt.SplitBehaviorEnum.SplitIfNeeded
rt.Style.Font = tb.Font
rt.Style.TextAlignVert = prt.AlignVertEnum.Center
rt.RepeatGridLinesVert = True '换页后重复表格线
For c As Integer = 0 To ColNames.Length - 1
    Dim lr As Integer ' 用于保存合并区域的起始行
    rt.Cells(0,c).Text = ColNames(c)
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth
    If tb.Cols(ColNames(c)).IsNumeric Orelse tb.Cols(ColNames(c)).IsDate Then
        rt.Cols(c).Style.TextAlignHorz = prt.AlignHorzEnum.Right
    End If
    For r As integer = 0 To tb.Rows.Count - 1
        If c <= MergeCols - 1 then '如果是要合并的列
            Dim Merge As Boolean = True
            If r = 0 Then
                Merge = False
            Else
                For n As Integer = 0 To c
                    if tb.Rows(r)(ColNames(n)) <> tb.Rows(r - 1)(ColNames(n))
                        Merge = False
                        Exit For
                    End If
                Next
            End If
            If Merge Then
                rt.Cells(lr,c).SpanRows = rt.Cells(lr,c).SpanRows + 1
            Else
                rt.Cells(r + 1, c).Text = tb.Rows(r)(ColNames(c))
                rt.Cells(r + 1, c).VertSplitBehavior = prt.CellSplitBehaviorEnum.Copy '换页后重复
单元格
                lr = r + 1
            End If
        Else
```

```
            rt.Cells(r + 1, c).Text = tb.Rows(r)(ColNames(c))
        End If
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
rt.CellStyle.Spacing.All = 0.5
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All
doc.Body.Children.Add(rt)
doc.Preview()
```

上面的代码是通用的，要打印其它表格或者合并不同的列时，只需修改下面四行代码：

```
Dim tb As Table = Tables("订单")
Dim ColNames As String() = New String(){"产品", "客户","单价","折扣","数量","金额","日期"}
Dim MergeCols As Integer = 2 '指定要合并的列数
tb.Sort = "产品,客户" '根据合并列排序
```

注意指定要打印列的时候，要合并的列必须位于最前面。

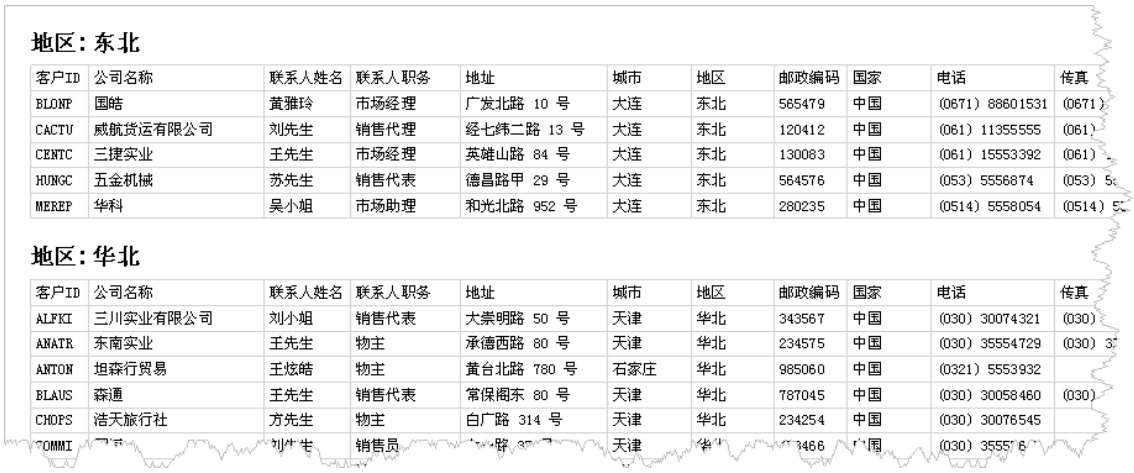
5.5.1.7 分组打印

打印客户表，按地区分组打印，代码如下：

```
Dim doc As New PrintDoc
Dim rt As Prt.RenderTable
Dim rx As prt.RenderText
Dim tbl As Table = Tables("客户")
Dim Rows As List(Of DataRow)
Dim Regions As List(Of String) = tbl.DataTable.GetValues("地区")
doc.Pagesetting.LandScape = True
For Each Region As String In Regions
    rx = New prt.RenderText
    rx.Style.FontSize = 14
    rx.Style.FontBold = True
    rx.Style.Spacing.Bottom = 2
    rx.Text = "地区: " & Region
    doc.Body.Children.Add(rx)
    rt = New prt.RenderTable
    rt.Style.Font = Tables("客户").Font
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center
    rt.Style.GridLines.All = New prt.LineDef(0.3,Color.LightGray)
    rt.Style.Spacing.Bottom = 5
    rt.CellStyle.Spacing.All = 1
    rt.Style.Font = tbl.Font
    Rows = tbl.Datatable.Select("[地区] = '" & Region & "'")
    For c As integer = 0 To tbl.Cols.Count - 1
        rt.Cells(0,c).Text = tbl.Cols(c).Name
```

```
        rt.Cols(c).Width = tbl.Cols(c).PrintWidth
        For r As integer = 0 To Rows.Count -1
            rt.Cells(r + 1, c).Text = rows(r)(tbl.Cols(c).Name)
        Next
    Next
    rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All
    doc.Body.Children.Add(rt)
Next
doc.preview()
```

打印效果：



地区：东北

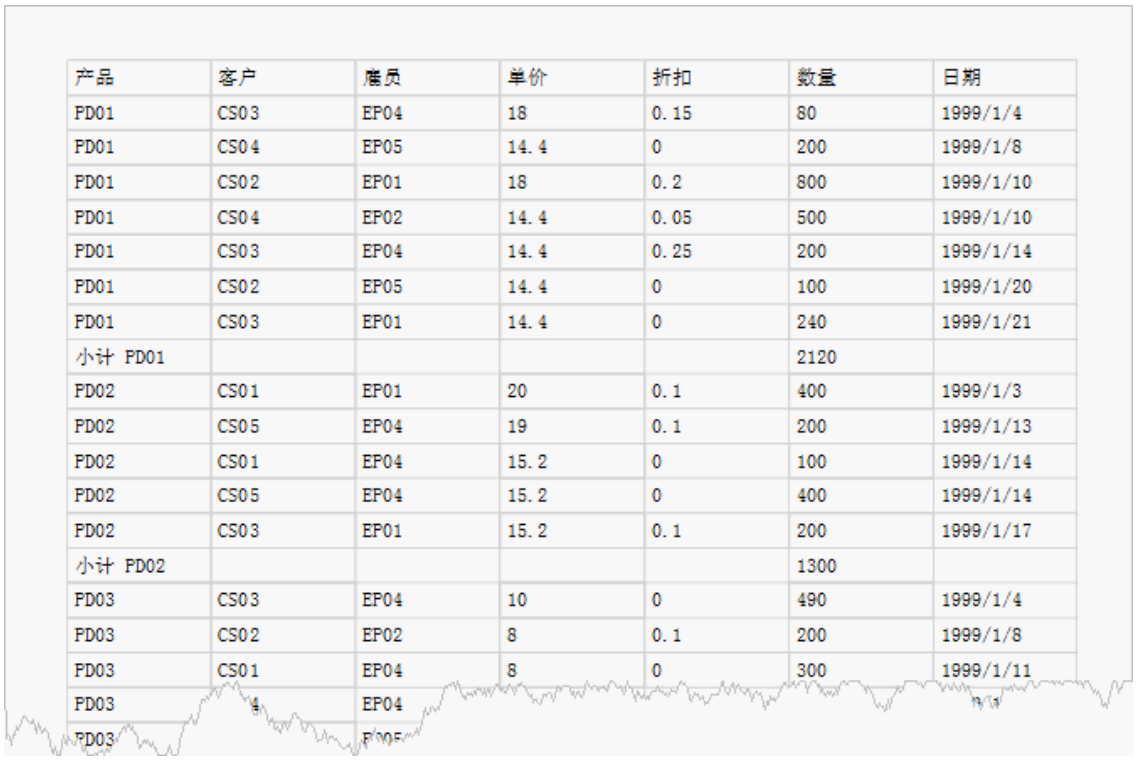
客户ID	公司名称	联系人姓名	联系人职务	地址	城市	地区	邮政编码	国家	电话	传真
BLONP	国皓	黄雅玲	市场经理	广发北路 10 号	大连	东北	565479	中国	(0671) 88601531	(0671)
CACTU	威航货运有限公司	刘先生	销售代理	经七纬二路 13 号	大连	东北	120412	中国	(061) 11355555	(061)
CENTC	三捷实业	王先生	市场经理	英雄山路 84 号	大连	东北	130083	中国	(061) 15553392	(061)
HUNGC	五金机械	苏先生	销售代表	德昌路甲 29 号	大连	东北	564576	中国	(053) 5556874	(053) 5
MEREP	华科	吴小姐	市场助理	和光北路 952 号	大连	东北	280235	中国	(0514) 5558054	(0514) 5

地区：华北

客户ID	公司名称	联系人姓名	联系人职务	地址	城市	地区	邮政编码	国家	电话	传真
ALFKI	三川实业有限公司	刘小姐	销售代表	大崇明路 50 号	天津	华北	343567	中国	(030) 30074321	(030)
ANATR	东南实业	王先生	物主	承德西路 80 号	天津	华北	234575	中国	(030) 35554729	(030) 3
ANTON	坦森行贸易	王炫皓	物主	黄台北路 780 号	石家庄	华北	985060	中国	(0321) 5553932	
BLAUS	森通	王先生	销售代表	常保阁东 80 号	天津	华北	787045	中国	(030) 30058460	(030)
CHOPS	浩天旅行社	方先生	物主	白广路 314 号	天津	华北	234254	中国	(030) 30076545	
COMMI	[illegible]	[illegible]	销售员	[illegible]	天津	华北	[illegible]466	[illegible]国	(030) 3555[illegible]6	

5.5.1.8 生成汇总模式

假定用专业报表打印订单表，希望按产品分组汇总：



产品	客户	雇员	单价	折扣	数量	日期
PD01	CS03	EP04	18	0.15	80	1999/1/4
PD01	CS04	EP05	14.4	0	200	1999/1/8
PD01	CS02	EP01	18	0.2	800	1999/1/10
PD01	CS04	EP02	14.4	0.05	500	1999/1/10
PD01	CS03	EP04	14.4	0.25	200	1999/1/14
PD01	CS02	EP05	14.4	0	100	1999/1/20
PD01	CS03	EP01	14.4	0	240	1999/1/21
小计 PD01					2120	
PD02	CS01	EP01	20	0.1	400	1999/1/3
PD02	CS05	EP04	19	0.1	200	1999/1/13
PD02	CS01	EP04	15.2	0	100	1999/1/14
PD02	CS05	EP04	15.2	0	400	1999/1/14
PD02	CS03	EP01	15.2	0.1	200	1999/1/17
小计 PD02					1300	
PD03	CS03	EP04	10	0	490	1999/1/4
PD03	CS02	EP02	8	0.1	200	1999/1/8
PD03	CS01	EP04	8	0	300	1999/1/11
PD03	4	EP04				
PD03		EP05				

可以参考下面的代码：

```
Dim doc As New PrintDoc
Dim rt As New Prt.RenderTable
Dim rx As prt.RenderText
Dim cnt As Integer
Dim tbl As Table = Tables("订单")
Dim drs As List(Of DataRow)
Dim pds As List(Of String) = tbl.DataTable.GetValues("产品",tbl.Filter)
rt.Style.GridLines.All = New prt.LineDef(0.3,Color.LightGray)
rt.CellStyle.Spacing.All = 1
rt.Style.Font = tbl.Font
For c As Integer = 0 To tbl.Cols.Count - 1 '生成列标题
    rt.Cells(0,c).Text = tbl.Cols(c).Name
    rt.Cols(c).Width = tbl.Cols(c).PrintWidth
Next
For Each pd As String In pds '分产品打印
    If tbl.Filter > "" Then
        drs = tbl.DataTable.Select("产品  = '" & pd & "' And " & tbl.Filter)
    Else
        drs = tbl.DataTable.Select("产品  = '" & pd & "' " )
    End If
    For Each dr As DataRow In drs '逐行打印此产品的订单
        cnt = rt.Rows.Count
        For c As Integer = 0 To tbl.Cols.Count - 1
            rt.Cells(cnt, c).Text = dr(tbl.Cols(c).Name)
        Next
```

```
    Next
    cnt = rt.Rows.Count
    rt.Cells(cnt, 0).Text = "小计 " & pd '打印产品的小计
    rt.Cells(cnt,5).Text = tbl.compute("Sum(数量)", "产品 = '" & pd & "'")
Next
cnt = rt.Rows.Count
rt.Cells(cnt, 0).Text = "合计 " '打印合计
rt.Cells(cnt, 5).Text = tbl.compute("Sum(数量)")
doc.Body.Children.Add(rt)
doc.preview()
```

代码说明：

上面的代码中，每次要打印一个新行，都先通过下面的代码获得已经打印的行数，并保存在变量 cnt 中：

```
cnt = rt.Rows.Count
```

假定已经打印了 9 行，编号为 0 到 8，执行上面的代码，cnt 的值就等于 9。

显然下一个订单或小计，应该打印在编号为 9 的行，也就是编号为 cnt 的行。

5.5.1.9 控制每页打印行数

打开 CaseStudy 目录下的文件：统计演示.Table

假定希望打印订单表，每页打印 20 行，可以参考下面的代码:

```
Dim doc As New PrintDoc
Dim tb As Table = Tables("订单")
Dim prs As Integer = 20 '每页 20 行
For p As Integer = 0 To math.Ceiling(tb.Rows.Count / prs) - 1
    Dim rt As New prt.RenderTable
    rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
    rt.CellStyle.Spacing.All = 0.5
    For c As Integer = 0 To tb.Cols.Count - 1
        rt.Cells(0,c).Text = tb.Cols(c).Name
        For r As Integer = p * prs To math.min(tb.Rows.Count - 1,( p + 1) * prs - 1)
            rt.Cells(r - p * prs + 1, c).Text = tb.rows(r)(c)
        Next
    Next
    If p < math.Ceiling(tb.Rows.Count / prs) - 1
        rt.BreakAfter = prt.BreakEnum.Page
    End If
    doc.Body.Children.Add(rt)
Next
doc.Preview()
```

如果是多层表头，可以采用下面的代码：

```
Dim doc As New PrintDoc
Dim tb As Table = Tables("订单")
Dim prs As Integer = 20 '每页 20 行
```

```
For p As Integer = 0 To math.Ceiling(tb.Rows.Count / prs) - 1
    Dim rt As New prt.RenderTable
    rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
    rt.CellStyle.Spacing.All = 0.5
    tb.CreateReportHeader(rt,False) '生成多层表头
    For c As Integer = 0 To tb.Cols.Count - 1
        For r As Integer = p * prs To math.min(tb.Rows.Count - 1,( p + 1) * prs - 1)
            rt.Cells(r - p * prs + tb.HeaderRows, c).Text = tb.rows(r)(c)
        Next
    Next
    If p < math.Ceiling(tb.Rows.Count / prs) - 1
        rt.BreakAfter = prt.BreakEnum.Page
    End If
    doc.Body.Children.Add(rt)
Next
doc.Preview()
```

有了上面的基础，我们可以很方便地给每页加上标题，例如下面的代码会在每页显示标题“2010 年订单”：

```
Dim doc As New PrintDoc
Dim tb As Table = Tables("订单")
Dim prs As Integer = 20 '每页 20 行
For p As Integer = 0 To math.Ceiling(tb.Rows.Count / prs) - 1
    Dim rx As New prt.RenderText
    rx.Text = "2010 年订单"
    rx.Style.Font = New Font("宋体", 16, FontStyle.Bold)
    rx.Style.TextAlignHorz = prt.AlignHorzEnum.Center
    rx.Style.Spacing.Bottom = 3
    Doc.body.Children.Add(rx)
    Dim rt As New prt.RenderTable
    rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
    rt.CellStyle.Spacing.All = 0.5
    For c As Integer = 0 To tb.Cols.Count - 1
        rt.Cells(0,c).Text = tb.Cols(c).Name
        For r As Integer = p * prs To math.min(tb.Rows.Count - 1,( p + 1) * prs - 1)
            rt.Cells(r - p * prs + 1, c).Text = tb.rows(r)(c)
        Next
    Next
    If p < math.Ceiling(tb.Rows.Count / prs) - 1
        rt.BreakAfter = prt.BreakEnum.Page
    End If
    doc.Body.Children.Add(rt)
Next
doc.Preview()
```

5.5.1.10 自动补空行

打开 CaseStudy 目录下的文件：统计演示.Table
现在希望打印订单表，每页 20 行，如果某页不够 20 行，则自动补空行。
代码如下：

```
Dim doc As New PrintDoc
Dim tb As Table = Tables("订单")
Dim prs As Integer = 20 '每页20行
For p As Integer = 0 To math.Ceiling(tb.Rows.Count / prs) - 1
    Dim rt As New prt.RenderTable
    rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
    rt.CellStyle.Spacing.All = 0.5
    For c As Integer = 0 To tb.Cols.Count - 1
        rt.Cells(0,c).Text = tb.Cols(c).Name
    Next
    For r As Integer = p * prs To math.min(tb.Rows.Count - 1,( p + 1) * prs - 1)
        For c As Integer = 0 To tb.Cols.Count - 1
            rt.Cells(r - p * prs + 1, c).Text = tb.rows(r)(c)
        Next
    Next
    If p = math.Ceiling(tb.Rows.Count / prs) - 1 '如果是最后一页
        For r As Integer = tb.Rows.Count To ( p + 1) * prs - 1 '补空行
            rt.Rows.Count = rt.Rows.Count + 1
            rt.Rows(rt.Rows.count -1)(0).text = " "
        Next
    Else
        rt.BreakAfter = prt.BreakEnum.Page '否则换页
    End If
    doc.Body.Children.Add(rt)
Next
doc.Preview()
```

5.5.1.11 分页汇总

打开 CaseStudy 目录下的文件：统计演示.Table
现在希望打印订单表，每页 20 行，每页最后一行显示数量和金额的小计，最后一页还要添加一行用于显示数量和金额的小计。
代码如下：

```
Dim doc As New PrintDoc
Dim tb As Table = Tables("订单")
Dim prs As Integer = 20 '每页20行
```

```
Dim sum1 As Integer = 0 '数量小计
Dim sum2 As Double = 0 '金额小计
Dim tsum1 As Integer = 0 '数量总计
Dim tsum2 As Double = 0 '金额总计
Dim idx1 As Integer= tb.cols("数量").Index
Dim idx2 As Integer= tb.cols("金额").Index
For p As Integer = 0 To math.Ceiling(tb.Rows.Count / prs) - 1
    Dim rt As New prt.RenderTable
    rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
    rt.CellStyle.Spacing.All = 0.5
    sum1 = 0
    sum2 = 0
    For c As Integer = 0 To tb.Cols.Count - 1
        rt.Cells(0,c).Text = tb.Cols(c).Name
    Next
    For r As Integer = p * prs To math.min(tb.Rows.Count - 1,( p + 1) * prs - 1)
        sum1 =sum1 + tb.rows(r)("数量")
        sum2 =sum2 + tb.rows(r)("金额")
        For c As Integer = 0 To tb.Cols.Count - 1
            rt.Cells(r - p * prs + 1, c).Text = tb.rows(r)(c)
        Next
    Next
    tsum1 = tsum1 + sum1
    tsum2 = tsum2 + sum2
    rt.Rows.Count = rt.Rows.Count + 1 '增加本页小计行
    rt.Rows(rt.Rows.count -1)(0).text = "本页小计"
    rt.Rows(rt.Rows.count -1)(idx1).text = sum1
    rt.Rows(rt.Rows.count -1)(idx2).text = sum2
    If p = math.Ceiling(tb.Rows.Count / prs) - 1 '如果是最后一页
        rt.Rows.Count = rt.Rows.Count + 1 '增加总计行
        rt.Rows(rt.Rows.count -1)(0).text = "总计"
        rt.Rows(rt.Rows.count -1)(3).text = tsum1
        rt.Rows(rt.Rows.count -1)(6).text = tsum2
    Else
        rt.BreakAfter = prt.BreakEnum.Page '否则换页
    End If
    doc.Body.Children.Add(rt)
Next
doc.Preview()
```

5.5.1.12 自动扩展某一列

打印员工表中的三列，分别是姓名、性别、部门、备注。

我们可以分别指定姓名、性别、部门三列的宽度，但不指定备注列的宽度，这样剩余的宽度将全

部为备注列所有：

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim tb as Table = Tables("员工")
Dim ColNames As String() = New String(){"姓名","性别","部门","备注"}
rt.Style.Font = tb.Font
For c As Integer = 0 To ColNames.Length - 1 '逐列设置和填入内容
    rt.Cells(0,c).Text = ColNames(c) '列名作为标题
    rt.Cells(0,c).Style.TextAlignHorz = prt.AlignHorzEnum.Center '标题内容水平居中
    If ColNames(c) <> "备注" Then '
        rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth
    End If
    For r As integer = 0 To tb.Rows.Count -1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = tb.Rows(r)(ColNames(c))
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 0.5 '单元格内距设为 0.5 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '利用行组，将第一行设为表头。
doc.Body.Children.Add(rt) '将表格加入到报表
doc.Preview()
```

打印效果：

姓名	性别	部门	备注
王伟	男	商务部	王伟获南京大学商业学士学位，获该校国际营销博士学位。他能说流利的法语和意大利语并能阅读德语。他加入公司时是销售代表，被提拔为销售经理并升任销售副总裁。王伟是销售管理圆桌协会，北京商业总会和太平洋周边进口协会的成员。
张颖	女	商务部	获北京大学心理学学士学位。她同时完成了“冷食的艺术”。张颖是国际美食协会的会员。
李芳	女	商务部	李芳获北京学院化学学士学位。她同时完成了食品零售管理资格程序。李芳被雇用为销售员并提升为销售代表。
郑建杰	男	商务部	郑建杰持有外国语学院英国文学学士学位和中国烹调艺术学院硕士学位。在他返回到北京的永久职位之前被临时派往上海办公室工作。
赵军	男	商务部	赵军毕业于复旦大学，获学士学位。自从加入公司成为一名销售代表，他花了6个月时间在北京办公室进行适应性工作，然后回到他在上海的固定职位。他被提拔为销售经理。赵军先生完成了课程“成功的电话销售”和“国际销售管理”。他的法语非常流利。

5.5.1.13 打印统计数据

CrossTableBuilder 和 CrossTableBuilder 提供了强大的统计功能。
在默认情况下，这两个类都会生成一个 Table，用于显示统计结果。
有的时候，我们仅仅希望打印统计结果，而不希望统计结果以 Table 的形式显示在界面中。

这两个类的 Build 方法都有一个可选参数，如果设为 True，将只生成一个临时的 DataTable，而不生成 Table。
所以当我们只想打印统计结果的时候，可以参考下面的代码：

```
Dim dt As DataTable
```

```
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
g.Totals.AddDef("数量")
dt = g.Build(True)
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim Count As Integer = 0
For Each Col AS DataCol In Dt.DataCols
    rt.Cells(0,Count).Text = Col.Name
    For r As integer = 0 To dt.DataRows.Count - 1
        rt.Cells(r +1,Count).Text = dt.DataRows(r)(Col.Name)
    Next
    Count = Count + 1
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
rt.CellStyle.Spacing.All = 1
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center
doc.Body.Children.Add(rt)
doc.Preview()
```

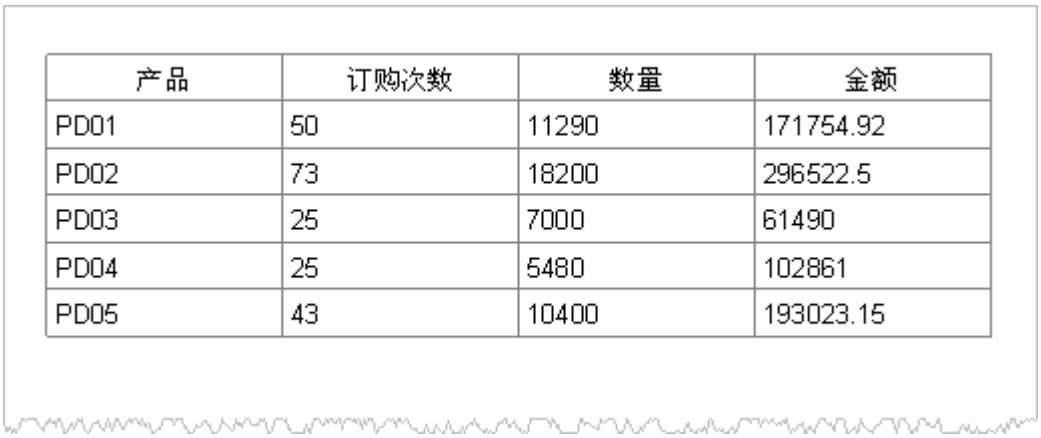
5.5.1.14 直接进行统计

要打印统计数据，并不一定要借助 CrossTableBuilder 或 CrossTableBuilder 来生成一个统计表，我们完全可以在报表生成的过程中进行统计。

例如下面的代码，直接打印出不同产品的销售数量和金额，没有借助任何统计表：

```
Dim Names As List(Of String) = Tables("订单").DataTable.GetValues("产品")
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
rt.Cells(0,0).Text = "产品"
rt.Cells(0,1).Text = "订购次数"
rt.Cells(0,2).Text = "数量"
rt.Cells(0,3).Text = "金额"
For i As integer = 0 To Names.Count -1
    rt.Cells(i+1,0).Text= Names(i)
    rt.Cells(i+1,1).Text = Tables("订单").DataTable.Compute("Count(日期)","产品='" & Names(i) & "'")
    rt.Cells(i+1,2).Text = Tables("订单").DataTable.Compute("Sum(数量)","产品='" & Names(i) & "'")
    rt.Cells(i+1,3).Text = Tables("订单").DataTable.Compute("Sum(金额)","产品='" & Names(i) & "'")
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 1 '内容距离网格线 1 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
doc.Body.Children.Add(rt)
doc.Preview()
```

打印结果：



产品	订购次数	数量	金额
PD01	50	11290	171754.92
PD02	73	18200	296522.5
PD03	25	7000	61490
PD04	25	5480	102861
PD05	43	10400	193023.15

5.5.1.15 打印后台数据

还记得 SQLCommand 吗？还记得 SQLCommand 的 ExecuteReader 方法吗？
有了 ExecuteReader，我们可以直接利用 SQL 查询语言取得后台数据，得到一个临时的 DataTable，使得打印后台数据成为可能：
本节同样要直接打印出 2009 年 6 月的订单：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim cmd As New SQLCommand '定义一个 SQL 命令
Dim dt As DataTable '定义一个数据表变量
Dim Count As Integer = 0
cmd.CommandText = "Select * From {订单} Where Year(日期) = 2009 And Month(日期) = 6"
dt = cmd.ExecuteReader() '生成一个临时表
For Each Col AS DataCol In Dt.DataCols
    rt.Cells(0,Count).Text = Col.Name
    For r As integer = 0 To dt.DataRows.Count - 1
        rt.Cells(r +1,Count).Text = dt.DataRows(r)(Col.Name)
    Next
    Count = Count + 1
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
rt.CellStyle.Spacing.All = 1 '内容距离网格线 1 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
doc.Body.Children.Add(rt)
doc.Preview()
```

其实上面的代码，是一个通用 SQL 查询结果打印程序，你要做的只是修改 SQL 语句来用于不同的统计。
CrossTableBuilder 和 CrossTableBuilder 提供了强大的统计功能，这两个类都有一个 FromServer 参数，只要设为 True，将对后台数据进行统计。

这两个类的 Build 方法有一个可选参数，如果设为 True，将只生成一个临时的 DataTable，而不生成 Table。

这样我们就可以直接统计并打印后台数据，而不需要首先在界面中显示统计结果：

```
Dim dt As DataTable
Dim g As New GroupTableBuilder("统计表 1", DataTables("订单"))
g.Groups.AddDef("产品")
g.Totals.AddDef("数量")
g.FromServer = True
dt = g.Build(True)
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim Count As Integer = 0
For Each Col AS DataCol In Dt.DataCols
    rt.Cells(0,Count).Text = Col.Name
    For r As integer = 0 To dt.DataRows.Count - 1
        rt.Cells(r +1,Count).Text = dt.DataRows(r)(Col.Name)
    Next
    Count = Count + 1
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
rt.CellStyle.Spacing.All = 1
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center
doc.Body.Children.Add(rt)
doc.Preview()
```

5.5.2 打印单据

5.5.2.1 员工资料卡

本节要打印一个下图所示的资料卡：



员工资料卡

姓名	王伟	出生日期	1962-2-19	
部门	商务部	雇佣日期	1992-8-14	
性别	男	职务	副总裁(销售)	
城市	北京	邮政编码	109801	
地址	罗马花园 890 号			
家庭电话	(010) 65559482	办公电话	3457	

王伟获南京大学商业学士学位，获该校国际营销博士学位。他能说流利的法语和意大利语并能阅读德语。他加入公司时是销售代表，被提拔为销售经理并升任销售副总裁。王伟是销售管理圆桌协会，北京商业总会和太平洋周边进口协会的成员。

代码看上去有点长，但是很好理解，而且大部分篇幅不过是向单元格填入内容而已：

```
Dim doc As New PrintDoc '定义一个报表
Dim rt As New prt.RenderTable() '定义一个表格对象
Dim rx As New prt.RenderText '定义一个文本对象
Dim CurRow As Row = Tables("员工").Current
'加入标题
rx.text = "员工资料卡"
rx.Style.FontBold = True '字体加粗
rx.Style.FontSize = 16 '大体大小为 16 磅
rx.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中排列
rx.Style.Spacing.Bottom = 3 '和下面的对象(表格)距离 3 毫米
doc.Body.Children.Add(rx) '加入到报表中
'指定行数、列数、列宽、行高
rt.Rows.Count = 7 '设置总行数
rt.Cols.Count = 5 '设置总列数
rt.Height = 80 '设置表格的高度为 80 毫米
rt.Rows(6).Height = 40 '设置第 7 行(显示备注的行)的高度为 40 毫米，剩余高度被平均分排到其他行
rt.Cols(0).Width = 24 '设置前四列的宽度，剩余的宽度被分配给 5 列(显示图片的那列)
rt.Cols(1).Width = 35
rt.Cols(2).Width = 24
rt.Cols(3).Width = 40
'设置合并单元格
rt.Cells(0,4).SpanRows = 6 '第 1 行第 5 个单元格向下合并 6 行(用于显示照片)
rt.Cells(4,1).SpanCols = 3 '第 5 行第 2 个单元格向右合并 3 列(用于显示地址)
rt.Cells(6,0).SpanCols = 5 '第 7 行第 1 个单元格向右合并 5 列(用于显示备注)
'设置表格样式
rt.CellStyle.Spacing.All = 1 '单元格内容缩进 1 毫米
```

```
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.Style.TextAlignVert = prt.AlignVertEnum.Center '内容垂直居中
rt.Rows(6).Style.TextAlignVert = prt.AlignVertEnum.Top '唯独第 7 行是备注，内容靠上对齐
'下面很简单，指定每一个单元格的内容
rt.Cells(0,0).Text= "姓名"
rt.Cells(0,1).Text = CurRow("姓名")
rt.Cells(0,2).Text= "出生日期"
rt.Cells(0,3).Text = CurRow("出生日期")
rt.Cells(1,0).Text= "部门"
rt.Cells(1,1).Text = CurRow("部门")
rt.Cells(1,2).Text= "雇佣日期"
rt.Cells(1,3).Text = CurRow("雇佣日期")
rt.Cells(2,0).Text= "性别"
rt.Cells(2,1).Text = CurRow("性别")
rt.Cells(2,2).Text= "职务"
rt.Cells(2,3).Text = CurRow("职务")
rt.Cells(3,0).Text= "城市"
rt.Cells(3,1).Text = CurRow("城市")
rt.Cells(3,2).Text= "邮政编码"
rt.Cells(3,3).Text = CurRow("邮政编码")
rt.Cells(4,0).Text= "地址"
rt.Cells(4,1).Text = CurRow("地址")
rt.Cells(5,0).Text= "家庭电话"
rt.Cells(5,1).Text = CurRow("家庭电话")
rt.Cells(5,2).Text= "办公电话"
rt.Cells(5,3).Text = CurRow("办公电话")
rt.Cells(6,0).Text = CurRow("备注")
rt.Cells(0,4).Image = GetImage(CurRow("照片"))
doc.Body.Children.Add(rt) '将表格对象加入到报表中
Doc.Preview() '预览报表
```

5.5.2.2 连续打印资料卡

本节还是打印员工资料卡，不过这次不是打印当前行，而是打印所有选定行。

每一个员工资料卡都是一个整体，不能因为换页而被分割，但是问题来了，因为一个资料卡包括两个对象：标题和表格，而我们只能禁止一个对象被分割，这样就可能导致标题和表格处在不同的页面中。

为解决这个问题，我们将标题和表格放在一个容器(RenderArea)中，一个容器就是一个资料卡，这样只需禁止容器被垂直分割即可。

请在员工表中选定多行，然后执行下面的代码：

```
Dim doc As New PrintDoc '定义一个报表
Dim tbl As Table = Tables("员工")
For i as Integer = tbl.TopRow To tbl.BottomRow
    Dim rw As Row = tbl.Rows(i)
```

```
Dim rt As New prt.RenderTable() '定义一个表格对象
Dim rx As New prt.RenderText '定义一个文本对象
Dim ra As New prt.RenderArea '定义一个容器
ra.SplitVertBehavior = prt.SplitBehaviorEnum.Never  '禁止容器因为分页而被垂直分割
'加入标题
rx.text = "员工资料卡"
rx.Style.FontBold = True '字体加粗
rx.Style.FontSize = 16 '大体大小为 16 磅
rx.Style.TextAlignHorz = prt.AlignHorzEnum.Center '水平居中排列
rx.Style.Spacing.Bottom = 3 '和下面的对象(表格)距离 3 毫米
ra.Children.Add(rx) '加入到容器中
'指定行数、列数、列宽、行高
rt.Rows.Count = 7 '设置总行数
rt.Cols.Count = 5 '设置总列数
rt.Height = 75 '设置表格的高度
rt.Rows(6).Height = 30 '设置第 7 行(显示备注的行)的高度，剩余高度被平均分排到其他行
rt.Cols(0).Width = 26 '设置前四列的宽度，剩余的宽度被分配给 5 列(显示图片的那列)
rt.Cols(1).Width = 35
rt.Cols(2).Width = 26
rt.Cols(3).Width = 40
'设置合并单元格
rt.Cells(0,4).SpanRows = 6 '第 1 行第 5 个单元格向下合并 6 行(用于显示照片)
rt.Cells(4,1).SpanCols = 3 '第 5 行第 2 个单元格向右合并 3 列(用于显示地址)
rt.Cells(6,0).SpanCols = 5 '第 7 行第 1 个单元格向右合并 5 列(用于显示备注)
'设置表格样式
rt.CellStyle.Spacing.All = 1 '单元格内容缩进 1 毫米
rt.Style.Spacing.Bottom = 5 '和下一个资料卡的距离是 5 毫米
rt.Style.GridLines.All = New prt.Linedef '设置网格线
rt.Style.TextAlignVert = prt.AlignVertEnum.Center '内容垂直居中
rt.Rows(6).Style.TextAlignVert = prt.AlignVertEnum.Top '唯独第 7 行是备注，内容靠上对齐
'下面很简单，指定每一个单元格的内容
rt.Cells(0,0).Text= "姓名"
rt.Cells(0,1).Text = rw("姓名")
rt.Cells(0,2).Text= "出生日期"
rt.Cells(0,3).Text = rw("出生日期")
rt.Cells(1,0).Text= "部门"
rt.Cells(1,1).Text = rw("部门")
rt.Cells(1,2).Text= "雇佣日期"
rt.Cells(1,3).Text = rw("雇佣日期")
rt.Cells(2,0).Text= "性别"
rt.Cells(2,1).Text = rw("性别")
rt.Cells(2,2).Text= "职务"
rt.Cells(2,3).Text = rw("职务")
rt.Cells(3,0).Text= "城市"
```

```
    rt.Cells(3,1).Text = rw("城市")
    rt.Cells(3,2).Text= "邮政编码"
    rt.Cells(3,3).Text = rw("邮政编码")
    rt.Cells(4,0).Text= "地址"
    rt.Cells(4,1).Text = rw("地址")
    rt.Cells(5,0).Text= "家庭电话"
    rt.Cells(5,1).Text = rw("家庭电话")
    rt.Cells(5,2).Text= "办公电话"
    rt.Cells(5,3).Text = rw("办公电话")
    rt.Cells(6,0).Text = rw("备注")
    rt.Cells(0,4).Image = GetImage(rw("照片"))
    ra.Children.Add(rt) '加入到容器中
    Doc.Body.ChildRen.Add(ra) '将容器加入到报表中
Next
Doc.Preview() '预览报表
```

执行结果:



员工资料卡

姓名	李芳	出生日期	1973-8-30	
部门	商务部	雇佣日期	1992-4-1	
性别	女	职务	销售代表	
城市	北京	邮政编码	198033	
地址	芍药园小区 78 号			
家庭电话	(010) 65553412	办公电话	3355	
李芳获北京学院化学学士学位。她同时完成了食品零售管理资格程序。李芳被雇用为销售员并提升为销售代表。				

员工资料卡

姓名	郑建杰	出生日期	1968-9-19	
部门	商务部	雇佣日期	1993-5-3	
性别	男	职务	销售代表	
城市	北京	邮政编码	198052	
地址	前门大街 789 号			
家庭电话	(010) 65558122	办公电话	5176	
郑建杰持有外国语学院英国文学学士学位和中国烹调艺术学院硕士学位。在他返回到北京的永久职位之前被临时派往上海办公室工作。				

5.5.2.3 插入图表

利用 ChartBuilder，我们可以很轻松地在专业报表中插入图表：

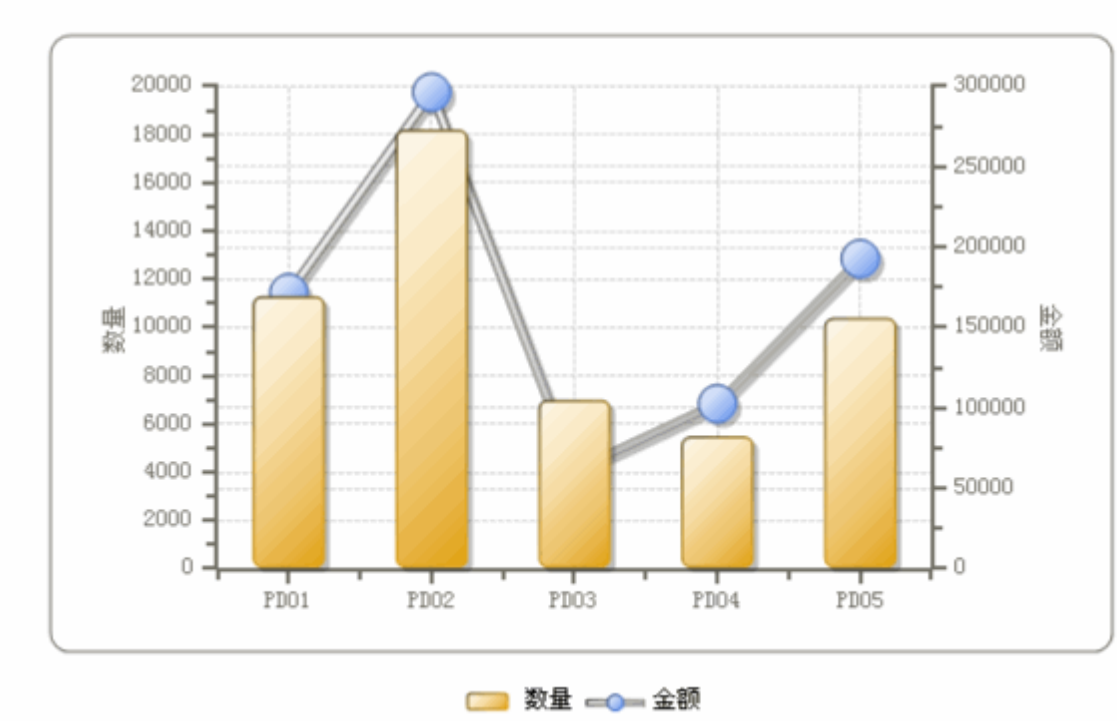
示例

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim cmd As New SQLCommand '定义一个 SQL 命令
Dim dt As DataTable '定义一个数据表变量
Dim Count As Integer = 0
'生成临时表
cmd.CommandText = "Select 产品,Sum(数量) As 数量,Sum(数量 * 单价 * (1 - 折扣)) As 金额 From {订单} Group By 产品"
dt = cmd.ExecuteReader() '生成一个临时表
'生成图表
Dim Chart As New ChartBuilder '定义一个图表变量
Dim Series As WinForm.ChartSeries '定义一个图系变量
Chart.VisualEffect = True '加上这一行,让你的图表更漂亮
Chart.ChartType = ChartTypeEnum.Bar '图表 1 类型该为 Bar(条形)
Chart.ChartType2 = ChartTypeEnum.XYPlot '图表 2 类型该为线形
Chart.DataTable = dt '设置绑定表
Chart.SeriesList.Clear() '清除图表原来的图系
Series = Chart.SeriesList.Add() '增加一个图系
Series.Text = "数量"
Series.X.DataField = "产品" 'X 轴绑定到月份列
Series.Y.DataField = "数量" 'Y 轴绑定到数量列
Series = Chart.SeriesList2.Add() '给图表 2 增加一个图系
Series.Text = "金额"
Series.X.DataField = "产品" 'X 轴绑定到月份列
Series.Y.DataField = "金额" 'Y 轴绑定到数量列
Chart.LegendVisible = True '显示图列
Chart.LegendCompass= CompassEnum.South '图列显示在南方(底端)
Chart.AxisY.Text = "数量"
Chart.AxisY2.Text = "金额"
Chart.PrintWidth = 150 '图表宽度为 150 毫米
Chart.PrintHeight = 100 '图标高度为 100 毫米
'打印表
For Each Col As DataCol In Dt.DataCols
    rt.Cells(0,Count).Text = Col.Name
    For r As integer = 0 To dt.DataRows.Count - 1
        rt.Cells(r +1,Count).Text = dt.DataRows(r)(Col.Name)
    Next
    Count = Count + 1
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray) '灰色网格线
```

```
rt.CellStyle.Spacing.All = 1 '内容距离网格线 1 毫米
rt.Rows(0).Style.TextAlignHorz = prt.AlignHorzEnum.Center '第一行内容水平居中
doc.Body.Children.Add(rt)
'打印图表
Dim rm As New prt.RenderImage
rm.Width = 150 '设置尺寸，这个尺寸最好和图表尺寸一致，以免打印出来的图表变形。
rm.Height = 100
rm.Image = Chart.Image()
rm.Style.ImageAlign.StretchHorz = False '禁止拉伸
rm.Style.ImageAlign.StretchVert = False
rm.Style.Spacing.Top = 3
doc.Body.Children.Add(rm) '将图片对象加入报表
doc.Preview()
```

执行结果：

产品	数量	金额
PD01	11290	171754.92
PD02	18200	296522.5
PD03	7000	61490
PD04	5480	102861
PD05	10400	193023.15



5.5.2.4 单据套打

有了专业报表，单据套打将是一件非常轻松的任务，您只需量量好每一个对象的尺寸，设置好对

象的坐标，然后将其加入到报表中即可。
例如有一个印刷好的员工资料卡：



员工资料卡

姓名		出生日期		
部门		雇佣日期		
性别		职务		
城市		邮政编码		
地址				
家庭电话		办公电话		

下面是套打员工资料卡的代码，由很多几乎相同的代码段组成：

```
Dim doc As New PrintDoc '定义一个报表
Dim rx As prt.RenderText '定义一个文本对象
Dim rm As prt.RenderImage '定义一个图形队形
Dim CurRow As Row = Tables("员工").Current

rx = new prt.RenderText
rx.Text = CurRow("姓名")
rx.x = 51
rx.y = 36
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("出生日期")
rx.x = 109
rx.y = 36
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("部门")
rx.x = 51
rx.y = 43
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("雇佣日期")
rx.x = 109
```

```
rx.y = 43
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("性别")
rx.x = 51
rx.y = 50
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("职务")
rx.x = 109
rx.y = 50
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("城市")
rx.x = 51
rx.y = 57
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("邮政编码")
rx.x = 109
rx.y = 57
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("地址")
rx.x = 51
rx.y = 63
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("家庭电话")
rx.x = 51
rx.y = 69.5
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("办公电话")
rx.x = 109
rx.y = 69.5
doc.body.Children.Add(rx)
```

```
rx = new prt.RenderText
rx.Text = CurRow("备注")
rx.x = 26.5
rx.y = 76
rx.Width= 163
doc.body.Children.Add(rx)

rm= New prt.RenderImage
rm.Height = 38
rm.Width = 34
rm.Image = GetImage(CurRow("照片"))
rm.x= 149.5
rm.y = 36
doc.body.Children.Add(rm)

Doc.Preview() '预览报表
```

5.5.2.5 连续套打

如果要连续套打，必须在打印每一个单据后，能够自动换页，准备套打下一个单据。
因为套打中的对象都是直接指定坐标后加入到到表中的，所以我们只能通过插入 RenderEmpty 来实现换页。
下面的代码，套打选定行的员工资料卡，请在员工表中选定多行进行测试：

```
Dim doc As New PrintDoc '定义一个报表
Dim rx As prt.RenderText '定义一个文本对象
Dim rm As prt.RenderImage '定义一个图形队形
Dim Currow As Row
Dim tbl As Table = Tables("员工")
For i as Integer = tbl.TopRow To tbl.BottomRow
    CurRow = tbl.rows(i)

    Dim rep As New prt.RenderEmpty '定义一个新的空对象
    rep.BreakBefore = prt.BreakEnum.Page '打印前换页
    doc.Body.Children.Add(rep ) '加入到报表中

    rx = new prt.RenderText
    rx.Text = CurRow("姓名")
    rx.x = 51
    rx.y = 36
    doc.body.Children.Add(rx)

    rx = new prt.RenderText
    rx.Text = CurRow("出生日期")
    rx.x = 109
```

```
rx.y = 36
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("部门")
rx.x = 51
rx.y = 43
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("雇佣日期")
rx.x = 109
rx.y = 43
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("性别")
rx.x = 51
rx.y = 50
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("职务")
rx.x = 109
rx.y = 50
doc.body.Children.Add(rx)
rx = new prt.RenderText
rx.Text = CurRow("城市")
rx.x = 51
rx.y = 57

doc.body.Children.Add(rx)
rx = new prt.RenderText
rx.Text = CurRow("邮政编码")
rx.x = 109
rx.y = 57
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("地址")
rx.x = 51
rx.y = 63
doc.body.Children.Add(rx)

rx = new prt.RenderText
```

```
    rx.Text = CurRow("家庭电话")
    rx.x = 51
    rx.y = 69.5
    doc.body.Children.Add(rx)

    rx = new prt.RenderText
    rx.Text = CurRow("办公电话")
    rx.x = 109
    rx.y = 69.5
    doc.body.Children.Add(rx)

    rx = new prt.RenderText
    rx.Text = CurRow("备注")
    rx.x = 26.5
    rx.y = 76
    rx.Width= 163
    doc.body.Children.Add(rx)

    rm= New prt.RenderImage
    rm.Height = 38
    rm.Width = 34
    rm.Image = GetImage(CurRow("照片"))
    rm.x= 149.5
    rm.y = 36
    doc.body.Children.Add(rm)
Next
Doc.Preview() '预览报表
```

5.5.2.6 套打技巧

有的时候，由于更换了打印机或纸张，所有打印对象都需要统一调整位置，以适用新的打印机或纸张。

一个一个去修改对象位置是很繁琐的，为解决这个问题，直接在代码中加入偏移参数，通过调整偏移参数来实现整体移动。

例如下面的代码，如果需要在水平方向统一移动对象，调整变量 X 的初始值即可，而修改变量 Y 的初始值，即可在垂直方向统一移动对象：

```
Dim doc As New PrintDoc '定义一个报表
Dim rx As prt.RenderText '定义一个文本对象
Dim x As Integer = 0 '定义水平偏移参数
Dim y As Integer = 0 '定义垂直偏移参数
Dim CurRow As Row = Tables("员工").Current

rx = new prt.RenderText
rx.Text = CurRow("姓名")
```

```
rx.x = 51 + x '设置坐标的时候，注意加上偏移参数
rx.y = 36 + y
doc.body.Children.Add(rx)

rx = new prt.RenderText
rx.Text = CurRow("出生日期")
rx.x = 109 + x
rx.y = 36 + y
doc.body.Children.Add(rx)
'加入更多的打印对象....

Doc.Preview() '预览报表
```

也许你会问，通过调整页边距不是一样可以的吗？当然不可以的，页边距的调整，只能影响没有指定坐标的打印对象的位置，而对于已经指定坐标的打印对象，其位置不随页边距的调整而变化，因为坐标的原点始终是页面左上角。

5.5.2.7 关联表和报表

在示例文件“专业报表.Table”中，类别表和产品表通过类别 ID 建立了关联，每一个类别下有多个产品。

现在希望按类别打印所有产品，代码如下：

```
Dim doc As New Printdoc
Dim rx As prt.RenderText
Dim rt As prt.RenderTable

rx = New prt.RenderText
rx.Style.FontSize = 14
rx.Style.FontBold = True
rx.Style.Spacing.Bottom = 5
rx.Text = "类别: " & Tables("类别").Current("类别名称")
doc.Body.Children.Add(rx)

rt = New prt.RenderTable
rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center
rt.Style.TextAlignVert = prt.AlignVertEnum.Center
rt.Style.Borders.Bottom = New prt.LineDef(0.3,Color.LightGray)
rt.CellStyle.Spacing.All = 1
rt.Cols.Count = 4
rt.Cells(0,0).Text = "产品名称"
rt.Cells(0,1).Text = "单价"
rt.Cells(0,2).Text = "库存量"
rt.Cells(0,3).Text = "订购量"
rt.Cells(0,4).Text = "再订购量"
rt.Cells(0,5).Text = "中止"
```

```
rt.rows(0).Style.Borders.Top = New prt.LineDef(1,Color.LightGray)
rt.rows(0).Style.Borders.Bottom = New prt.LineDef(1,Color.LightGray)
With Tables("类别.产品")
    For r As integer = 0 To .Rows.Count - 1 '遍历关联表每一行
        rt.Cells(r+1,0).Text = .rows(r)("产品名称")
        rt.Cells(r+1,1).Text = .rows(r)("单价")
        rt.Cells(r+1,2).Text = .rows(r)("库存量")
        rt.Cells(r+1,3).Text = .rows(r)("订购量")
        rt.Cells(r+1,4).Text = .rows(r)("再订购量")
        If .rows(r)("中止") = True Then
            Dim rm As New prt.RenderImage
            rm.Image = getImage("Check.Ico")
            rm.Style.ImageAlign.AlignHorz = prt.ImageAlignHorzEnum.Center
            rm.Style.ImageAlign.StretchHorz = False
            rm.Style.ImageAlign.StretchVert = False
            rt.Cells(r+1,5).RenderObject = rm
        End If
    Next
End With
doc.Body.Children.Add(rt)

rx = New prt.RenderText
rx.Style.FontBold = True
rx.Style.Spacing.Top = 3
rx.Text = "产品数目: " & Tables("类别.产品").Rows.Count
rx.Style.TextAlignHorz = prt.AlignHorzEnum.Right
doc.Body.Children.Add(rx)

doc.Preview
```

下图是执行结果：

类别: 谷类/麦片

产品名称	单价	库存量	订购量	再订购量	中止
糯米	21	104	0	25	
燕麦	9	61	0	25	
糙米	14	26	0	0	✓
三合一麦片	7	38	0	25	
白米	38	21	10	30	
小米	19.5	36	0	20	
黄豆	33.25	22	80	30	

产品数目: 7

请留意上面的代码中，我们是如何打印逻辑列的。

5.5.2.8 连续打印关联表

```
doc As New Printdoc
Dim rx As prt.RenderText
Dim rt As prt.RenderTable
Dim Rows As List(Of DataRow)
Dim tbl As Table = Tables("类别")

For i as Integer = tbl.TopRow To tbl.BottomRow
    rx = New prt.RenderText
    rx.BreakBefore = prt.BreakEnum.Page '另起一页再打印
    rx.Style.FontSize = 14
    rx.Style.FontBold = True
    rx.Style.Spacing.Bottom = 5
    rx.Text = "类别: " & Tables("类别").Rows(i)("类别名称")
    doc.Body.Children.Add(rx)

    rt = New prt.RenderTable
    rt.Style.TextAlignHorz = prt.AlignHorzEnum.Center
    rt.Style.TextAlignVert = prt.AlignVertEnum.Center
    rt.Style.Borders.Bottom = New prt.LineDef(0.3,Color.LightGray)
    rt.CellStyle.Spacing.All = 1
    rt.Cols.Count = 4
    rt.Cells(0,0).Text = "产品名称"
    rt.Cells(0,1).Text = "单价"
    rt.Cells(0,2).Text = "库存量"
    rt.Cells(0,3).Text = "订购量"
    rt.Cells(0,4).Text = "再订购量"
    rt.Cells(0,5).Text = "中止"
    rt.rows(0).Style.Borders.Top = New prt.LineDef(1,Color.LightGray)
    rt.rows(0).Style.Borders.Bottom = New prt.LineDef(1,Color.LightGray)

    Rows = Tables("类别").Rows(i).DataRow.GetChildRows("产品")
    For r As integer = 0 To Rows.Count - 1
        rt.Cells(r+1,0).Text = rows(r)("产品名称")
        rt.Cells(r+1,1).Text = rows(r)("单价")
        rt.Cells(r+1,2).Text = rows(r)("库存量")
        rt.Cells(r+1,3).Text = rows(r)("订购量")
        rt.Cells(r+1,4).Text = rows(r)("再订购量")
        If rows(r)("中止") = True Then
            Dim rm As New prt.RenderImage
            rm.Image = getImage("Check.Ico")
```

```
            rm.Style.ImageAlign.AlignHorz = prt.ImageAlignHorzEnum.Center
            rm.Style.ImageAlign.StretchHorz = False
            rm.Style.ImageAlign.StretchVert = False
            rt.Cells(r+1,5).RenderObject = rm
        End If
    Next

    doc.Body.Children.Add(rt)
    rx = New prt.RenderText
    rx.Style.FontBold = True
    rx.Style.Spacing.Top = 3
    rx.Text = "产品数目:" & Rows.Count
    rx.Style.TextAlignHorz = prt.AlignHorzEnum.Right
    doc.Body.Children.Add(rx)
Next
```

5.6 更专业的条形码功能

专业报表虽然提供了一个 RenderBarCode 类型用于打印条形码，但是功能有限，且只能打印几种常见的一维码，对于二维码束手无策。

我们可以用 BarCodeBuilder 类型来为专业报表实现更专业的条形码打印功能。

示例一

生成一个简单的条码：

```
Dim doc As New PrintDoc
Dim rg As New prt.RenderGraphics
Dim Bar As New BarCodeBuilder
Bar.Symbology = Barpro.Symbology.Code39
bar.Code = "12345678"
rg = new prt.RenderGraphics
bar.DrawOnCanvas(rg.Graphics,0,0,1)
Doc.Body.Children.Add(rg)
Doc.Preview()
```

示例二

根据某表的内容，批量打印二维码：

```
Dim doc As New PrintDoc
Dim rg As prt.RenderGraphics
Dim Bar As New BarCodeBuilder
Bar.Symbology = Barpro.Symbology.QRCode
Doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight
```

```
For Each r As Row In Tables("表A").Rows
    bar.Code = r("第一列")
    rg = new prt.RenderGraphics
    bar.DrawOnCanvas(rg.Graphics,0,0,1)
    rg.Style.Spacing.All = 3
    Doc.Body.Children.Add(rg)
Next
Doc.Preview()
```

5.7 增加图表功能

专业报表并没有图表功能，不过利用 ChartBuilder 我们方便地在专业报表中实现图表生成和打印功能。

示例：

```
'生成图表
Dim Chart As New ChartBuilder
Dim Series As WinForm.ChartSeries
Chart.DataSource = "统计表 1"
Chart.SeriesList.Clear()
Series = Chart.SeriesList.Add()
Series.X.DataField = "产品"
Series.Y.DataField = "数量"
Chart.PrintWidth =140
Chart.PrintHeight = 100
'打印图表
Dim doc As New PrintDoc
Dim rm As New prt.RenderImage
rm.Image = Chart.Image
doc.Body.Children.Add(rm)
Doc.Preview()
```

5.8 保存报表

报表除打印之外，通过下面三个方法，还可以保存为 htm 或 pdf 格式的文件，或保存为图片文件。

- SaveHTM
  将报表保存为 HTM 格式的文件。

- SavePDF
  将报表保存为 PDF 格式的文件。

- SaveImage
  将报表保存为图片文件。

示例一

```
Dim doc As New PrintDoc '定义一个新报表
Dim rt As New prt.RenderTable '定义一个新表格
Dim ColNames As String() = New String(){"客户","产品", "数量","单价","金额"}
Dim drs As List(of DataRow) = DataTables("订单").Select("日期 = #" & Date.Today & "#")
For c As Integer = 0 To ColNames.Length - 1 '逐列设置和填入内容
    rt.Cells(0,c).Text = ColNames(c) '列名作为标题
    For r As Integer = 0 To drs.Count -1 '开始填入该列内容
        rt.Cells(r + 1, c).Text = drs(r)(ColNames(c))
    Next
Next
rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
doc.Body.Children.Add(rt) '将表格加入到报表
doc.SaveHtm("c:\data\test.htm")
```

示例二

一些用户反馈在使用定义纸张，特别是纸张的宽度大过高度时，打印方向经常会遇到问题。

我们可以将报表保存为 PDF 文件，然后再打印，例如：

```
Dim fl As String = "c:\data\test.pdf"
Dim doc   As New PrintDoc
Dim ra As New prt.RenderArea
doc.PageSetting.Width = 150
doc.PageSetting.Height = 80
ra.Width = "Parent.Width"
ra.Height = "Parent.Height"
ra.Style.Borders.All = New prt.Linedef(5, Color.Red)
doc.Body.Children.Add(ra)
doc.SavePDF(fl) '保存为PDF文件
Dim Proc As New Process '打开PDF文件
Proc.File = fl
Proc.Start()
```

实例三

可以将生成的报表保存为图片文件，例如：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.Text = "Hello Foxtable"
rt.Style.TextColor = Color.Red
rt.Style.TextAngle = -45
doc.Body.Children.Add(rt)
doc.SaveImage("c:\data\abc.jpg")
```

需要注意的是，上面的代码执行后，你得到的图片文件名并不是"abc.jpg"，而是：

abc_page1.jpg
abc_page2.jpg
abc_page3.jpg
...

报表的每一页都会生成一个独立的图片文件。

SaveImage可以设置图片的分辨率，其语法为：

SaveImage(Filename，Resolution)

FileName: 要保存的文件名
Resolution: 可选参数，用于设置图片分辨率，默认为150，数值越大，图片越清晰，当然文件尺寸也会越大

5.9 数据绑定

5.9.1 绑定基础

5.9.1.1 最简单的绑定

首先还是打开我们的示例文件“专业报表.Table”，然后在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单") '将 rt 绑定到订单表
rt.Text= "[Fields!产品.Value]" '设置打印字段
doc.body.Children.Add(rt)
doc.Preview()
```

上述代码会打印出订单表中每一行的产品名称。
DataBinding 用于设置打印对象的绑定属性，该属性包括一些子属性，DataSource 表示数据来源，数据来源可以是 Table，还可以是数组或者集合。

需要注意的是，不能直接绑定到 Table，如果要绑定到 Table，必须用 BindTables 来设置，例如：

```
rt.DataBinding.DataSource = BindTables("订单")
```

如果对象的内容来自于某一列，其 Text 属性的设置格式如下：

[Fields!列名称.Value]

5.9.1.2 使用表达式

在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单") '将 rt 绑定到订单表
rt.Text= "[Math.Round(Fields!金额.Value,1)]" '设置打印表达式
doc.body.Children.Add(rt)
doc.Preview()
```

留意方括号中的内容：

Math.Round(Fields!金额.Value,1)

意思是打印的时候将金额列的内容，保留一位小数，你可以在 Text 属性中使用 Foxtable 支持的所有函数。

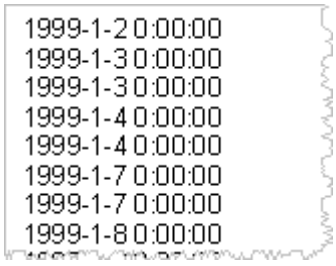
注意表达式必须用方括号括起来。

5.9.1.3 日期的打印

在命令窗口执行下面的代码：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单") '将 rt 绑定到订单表
rt.Text= "[Fields!日期.Value]"
doc.body.Children.Add(rt)
doc.Preview()
```

你会发现，打印出来的日期带有时间部分，并不符合我们的要求：



为了得到常规的日期格式，可以利用 Format 格式化日期数据：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单") '将 rt 绑定到订单表
rt.Text= "[Format(Fields!日期.Value,""yyyy-MM-dd"")]"
doc.body.Children.Add(rt)
```

doc.Preview()
要记得对于字符串中的每一个双引号，都必须用两个双引号来表示。

5.9.1.4 绑定多个字段

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单")
rt.Text= "[Fields!日期.Value]"
doc.body.Children.Add(rt)
rt = New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单")
rt.Text= "[Fields!数量.Value]"
doc.body.Children.Add(rt)
doc.Preview()
```

你会发现执行结果完全出乎我们的意料，先是打印所有的日期，然后打印所有的数量，而不是一个日期接一个数量。
这是因为每加入一个绑定对象，都会自动根据每一个数据行生成一个副本加入到报表中，然后再处理下一个对象。
为了解决这个问题，需要使用容器(RenderArea)，将原来的打印对象加入到容器中，这些打印对象不再需要设置绑定，只需为容器设置绑定即可。
例如：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra As New prt.RenderArea
ra.Style.Spacing.Bottom = 2
ra.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
rt.Text = "[Fields!日期.Value]"
ra.Children.Add(rt) '将打印对象添加到容器中
rt = New prt.RenderText
rt.Text= "[Fields!数量.Value]"
ra.Children.Add(rt)
doc.body.Children.Add(ra)
doc.Preview()
```

上述的代码将依次打印每一个订单的日期和数量。

5.9.1.5 打印标签

有了绑定，打印标签比以前更简单。
示例

```
Dim doc As New PrintDoc '定义一个报表
Doc.Stacking = prt.StackingRulesEnum.InlineLeftToRight
Dim ra As New prt.RenderArea
```

```
Dim rt As prt.RenderText
ra.Width = 40 '设置标签宽度
ra.SplitHorzBehavior = prt.SplitBehaviorEnum.Never '禁止水平分割
ra.SplitVertBehavior = prt.SplitBehaviorEnum.Never '禁止垂直分割
ra.Style.Spacing.Right = 2 '设置标签之间的所有和上下间隔为 2 毫米
ra.Style.Spacing.Bottom = 2
ra.Style.Padding.All = 1 ' 标签内容距离边框为 1 毫米
ra.Style.Borders.All = New prt.Linedef(0.3, Color.Red) '设置边框
ra.DataBinding.DataSource = BindTables("员工") '将容器绑定到员工表
Doc.Body.ChildRen.Add(ra) '将容器加入到报表中
rt = New prt.RenderText
rt.Text = "姓名：[Fields!姓名.Value]"
ra.Children.Add(rt) '添加到容器中
rt = New prt.RenderText
rt.Text = "部门：[Fields!部门.Value]"
ra.Children.Add(rt) '添加到容器中
rt = New prt.RenderText
rt.Text = "职务：[Fields!职务.Value]"
ra.Children.Add(rt) '添加到容器中
Doc.Preview() '预览报表
```

执行结果：



5.9.1.6 表格与绑定

行组也可以设置绑定，这样我们可以很方便地将一个表格绑定到数据表。

例如，打印订单表中的某些列，代码非常简单：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim tb as Table = Tables("订单")
Dim ColNames As String() = New String(){"产品", "客户","单价", "数量", "金额"}
```

```
For c As integer = 0 To ColNames.Length -1
    rt.Cells(0,c).Text = ColNames(c)
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth
    rt.Cells(1, c).Text = "[Fields!" & ColNames(c) & ".Value]" '设置绑定表达式
Next
rt.RowGroups(0,1).Style.TextAlignVert = prt.AlignVertEnum.Center '第一行内容垂直居中
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All '将第一行作为表头。
rt.RowGroups(1,1).DataBinding.DataSource = BindTables("订单") '将第二行绑定到订单表
doc.Body.Children.Add(rt)
doc.Preview()
```

注意这一行代码：

```
rt.RowGroups(1,1).DataBinding.DataSource = BindTables("订单") '将第二行绑定到订单表
```

该行代码将表格的第二行绑定到订单表，这样第二行将根据订单表中的每一行数据，自动生成副本，完成整个表格的打印。

5.9.2 分组报表

5.9.2.1 分组报表

DataBinding 有一个 Grouping 子属性，用于设置分组。

例如我们希望按产品分组，分别打印每一个产品的销售明细，代码如下：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra As New prt.RenderArea

ra.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
ra.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '根据产品进行分组
ra.Style.Spacing.Bottom = 2

'这一个 RenderText 无需设置 DataBinding，这样每一个分组只打印一次
rt.Text= "[Fields!产品.Value]明细:"
rt.Style.FontSize = 14
rt.Style.FontBold = True
ra.Children.Add(rt)

'这一个 RenderText 需要单独设置 DataBinding，这样才会针对每一个数据行打印副本
rt = New prt.RenderText
rt.Text = "日期:[Format(Fields!日期.Value,""yyyy-MM-dd"")] 客户:[Fields!客户.Value] 数量:[Fields!数量.Value]"
rt.DataBinding.DataSource = ra.DataBinding.DataSource '不能漏掉这一行
ra.Children.Add(rt)
```

doc.body.Children.Add(ra)
doc.Preview()
执行结果：



5.9.2.2 多个分组

可以根据多列进行分组。
例如下面的代码同时根据产品和雇员分组：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra As New prt.RenderArea
ra.Style.Spacing.Bottom = 2
ra.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
ra.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '增加两个分组，分别是产品和雇员
ra.DataBinding.Grouping.Expressions.Add("Fields!雇员.Value") '
ra.Style.Spacing.Bottom = 2

'这一个 RenderText 无需设置 DataBinding，这样每一个分组只打印一次
rt.Text= "产品:[Fields!产品.Value]  雇员:[Fields!雇员.Value]"
rt.Style.FontSize = 14
```

```
rt.Style.FontBold = True
ra.Children.Add(rt)

'这一个 RenderText 需要单独设置 DataBinding，这样才会针对每一个数据行打印副本
rt = New prt.RenderText
rt.Text = "日期:[Format(Fields!日期.Value,""yyyy-MM-dd"")] 客户:[Fields!客户.Value] 数量:[Fields!数量.Value]"
rt.DataBinding.DataSource = ra.DataBinding.DataSource '不能漏掉这一行
ra.Children.Add(rt)
doc.body.Children.Add(ra)
doc.Preview()
```

执行结果：



5.9.2.3 多层分组

下面的代码同样是根据产品和雇员分组，但是分两层：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra1 As New prt.RenderArea
Dim ra2 As New prt.RenderArea

'设置一级分组
ra1.Style.Spacing.Bottom = 2
ra1.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
ra1.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '增加产品分组
```

```
'打印一级分组的组名
rt.Text= "产品:[Fields!产品.Value]"
rt.Style.FontSize = 14
rt.Style.FontBold = True
rt.Style.Spacing.Bottom = 2
ra1.Children.Add(rt)
'设置二级分组
ra2 = New Prt.RenderArea
ra2.Style.Spacing.Bottom = 2
ra2.DataBinding.DataSource = ra1.DataBinding.DataSource '将容器绑定到订单表
ra2.DataBinding.Grouping.Expressions.Add("Fields!雇员.Value") '增加雇员分组
ra1.Children.Add(ra2) '二级分组的容器要加到一级分组的容器中
'打印二级分组的组名
rt = New prt.RenderText
rt.Text= "雇员:[Fields!雇员.Value]"
rt.Style.FontSize = 12
rt.Style.FontBold = True
rt.Style.Spacing.Bottom = 2
ra2.Children.Add(rt)
'打印明细
rt = New prt.RenderText
rt.Text = "日期:[Format(Fields!日期.Value,""yyyy-MM-dd"")] 客户:[Fields!客户.Value] 数量:[Fields!数量.Value]"
rt.DataBinding.DataSource = ra2.DataBinding.DataSource '不能漏掉这一行
ra2.Children.Add(rt) '添加到二级分组中

doc.body.Children.Add(ra1)
doc.Preview()
```

执行结果：



5.9.2.4 分组统计

5.9.2.4.1 关于分组统计

DataBinding.Aggregate 类型用于在报表中进行分组统计，该类型包括以下属性：

名称	说明
Name	名称，打印对象通过这个名称来引用这个 Aggregate 的统计结果。
DataBinding	指定绑定对象。
ExpressionText	统计表达式
Running	指定统计范围，包括两个值： 0:总计 1: 分组
Func	指定统计类型,包括以下可选值： 0：累积值 1：平均值 2：纪录数 3：最小值

	4：最大值 8：采样方差 9：总体方差 10：标准偏差 11：总体标准差 默认为 0，所以如果要进行求和统计，那么可以不设置 Func 属性。
IgnoreNullValues	逻辑型,统计的时候,是否忽略空值

定义一个 Aggregate 的语法是：

New prt.Aggregate(Name)

例如要打印出不同产品的销售额：

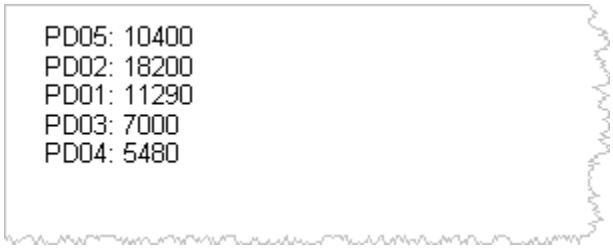
```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim Agg As New prt.DataBinding.Aggregate("GroupSum") '新定义统计的名称为 GroupSum，名称随意即可。

rt.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
rt.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '根据产品进行分组
rt.Text= "[Fields!产品.Value]: [Aggregates!GroupSum.Value]" '设置打印内容
doc.body.Children.Add(rt)

agg.DataBinding = rt.DataBinding 'Agg 的 DataBinding 属性，应该设为打印对象的 DataBinding
agg.Running = 1 '分组统计
agg.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(agg)'定义好的统计必须添加到报表的 DataSchema 中

doc.Preview()
```

执行结果：



引用一个定义好的 Aggregate 的语法是:

[Aggregates!统计名称.Value]

例如：

rt.Text= "[Fields!产品.Value]: [Aggregates!GroupSum.Value]"

5.9.2.4.2 按月分组统计

下面的代码，统计不同月份的销售额：

```
Dim doc As New PrintDoc
```

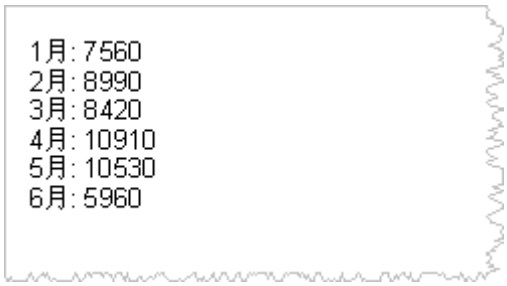
```
Dim rt As New prt.RenderText
Dim Agg As New prt.DataBinding.Aggregate("GroupSum") '新定义统计的名称为 GroupSum，名称随意即可。

rt.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
rt.DataBinding.Grouping.Expressions.Add("Fields!日期.Value.Month") '根据月份进行分组
rt.Text= "[Fields!日期.Value.Month]月: [Aggregates!GroupSum.Value]" '设置打印内容
doc.body.Children.Add(rt)

agg.DataBinding = rt.DataBinding 'Agg 的 DataBinding 属性，应该设为打印对象的 DataBinding
agg.Running = 1 '分组统计
agg.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(agg)'定义好的统计必须添加到报表的 DataSchema 中

doc.Preview()
```

执行结果：



5.9.2.4.3 按产品统计每月销售额

代码：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra As New prt.RenderArea
Dim Agg1 As New prt.DataBinding.Aggregate("GroupSum") '这个统计用于按产品进行分组统计
Dim Agg2 As New prt.DataBinding.Aggregate("SubGroupSum") '这个统计准备按月分组统计

ra.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
ra.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '根据产品进行分组
ra.Style.Spacing.Bottom = 2
doc.body.Children.Add(ra)

rt.Text= "[Fields!产品.Value]: [Aggregates!GroupSum.Value]" '设置打印内容
rt.Style.FontBold= True
rt.Style.FontSize=12
ra.Children.Add(rt)

'下面的 RenderText 用于统计某产品不同月份的销售数量，所以需要指定数据源和分组表达式
```

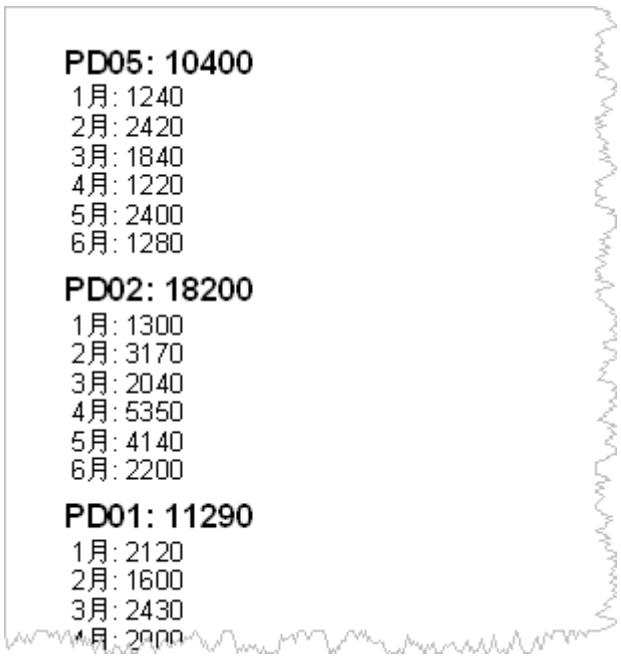
```
'也就是说，还需要在产品分组里头再按月分组。
rt = New prt.RenderText
rt.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
rt.DataBinding.Grouping.Expressions.Add("Fields!日期.Value.Month") '根据月份进行分组
rt.Text= " [Fields!日期.Value.Month]月: [Aggregates!SubGroupSum.Value]" '设置打印内容
ra.Children.Add(rt)

agg1.DataBinding = ra.DataBinding '这个一定要设置为 ra 的 DataBinding，因为 agg1 统计的分组来自于 ra 定义的分组
agg1.Running = 1 '分组统计
agg1.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(agg1)'定义好的统计必须添加到报表的 DataSchema 中

agg2.DataBinding = rt.DataBinding '这个一定要设置为 rt 的 DataBinding，因为 agg1 统计的分组来自于 rt 定义的分组
agg2.Running = 1 '分组统计
agg2.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(agg2)'定义好的统计必须添加到报表的 DataSchema 中

doc.Preview()
```

执行结果：



5.9.2.4.4 加上总计

打印不同月份的销售额，并且加上总计

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim gg As New prt.DataBinding.Aggregate("GroupSum") '用于分组统计。
```

```
Dim tg As New prt.DataBinding.Aggregate("TotalSum") '用于总计。

rt.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
rt.DataBinding.Grouping.Expressions.Add("Fields!日期.Value.Month") '根据月份进行分组
rt.Text= "[Fields!日期.Value.Month]月: [Aggregates!GroupSum.Value]" '设置打印内容
doc.body.Children.Add(rt)

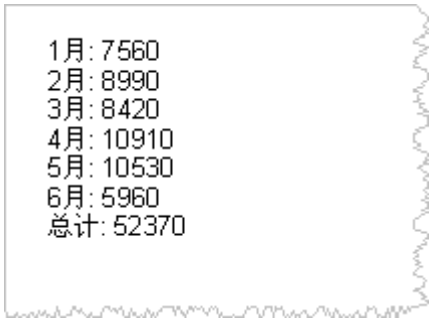
gg.DataBinding = rt.DataBinding 'Agg 的 DataBinding 属性，应该设为打印对象的 DataBinding
gg.Running = 1 '分组统计
gg.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(gg)'定义好的统计必须添加到报表的 DataSchema 中

tg.DataBinding = rt.DataBinding 'Agg 的 DataBinding 属性，应该设为打印对象的 DataBinding
tg.Running = 0 '总计
tg.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(tg)'定义好的统计必须添加到报表的 DataSchema 中

rt = New prt.RenderText
rt.Text= "总计: [Aggregates!TotalSum.Value]" '设置打印内容
doc.body.Children.Add(rt)

doc.Preview()
```

执行结果：



5.9.2.4.5 多个统计

统计不同产品的订单数、订购数量：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra As New prt.RenderArea
Dim Agg1 As New prt.DataBinding.Aggregate("Sum")
Dim Agg2 As New prt.DataBinding.Aggregate("Count")

ra.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
ra.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '根据产品进行分组
ra.Style.Spacing.Bottom = 2
```

```
doc.body.Children.Add(ra)

rt.Text= "[Fields!产品.Value]"
ra.Children.Add(rt)

rt = New prt.RenderText
rt.Text= "  订单数: [Aggregates!Count.Value]"
ra.Children.Add(rt)

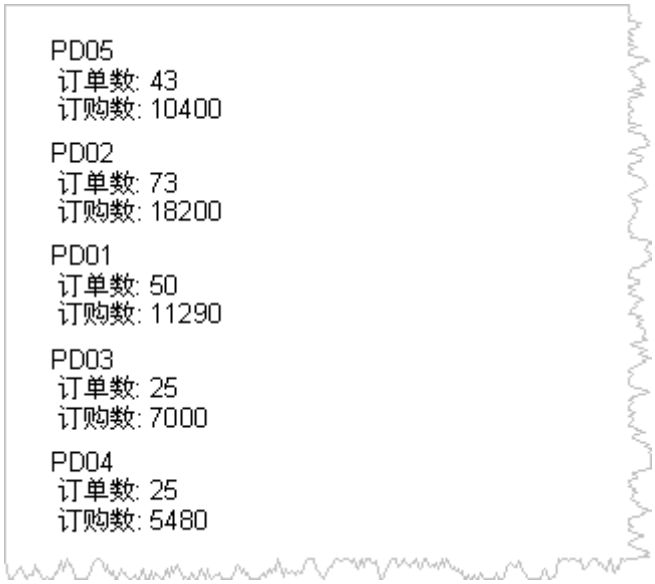
rt = New prt.RenderText
rt.Text= "  订购数: [Aggregates!Sum.Value]"
ra.Children.Add(rt)

agg1.DataBinding = ra.DataBinding '这个一定要设置为 ra 的 DataBinding，因为 agg1 统计的分组来自于 ra 定义的分组
agg1.Running = 1 '分组统计
agg1.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(agg1)'定义好的统计必须添加到报表的 DataSchema 中

agg2.DataBinding = ra.DataBinding '这个一定要设置为 rt 的 DataBinding，因为 agg1 统计的分组来自于 rt 定义的分组
agg2.Running = 1 '分组统计
agg2.ExpressionText = "Fields!产品.Value" '统计字段
agg2.Func = 2 '统计订单数
doc.DataSchema.Aggregates.Add(agg2)'定义好的统计必须添加到报表的 DataSchema 中

doc.Preview()
```

执行结果：



5.9.2.4.6 分组和明细

统计不同产品的销售额，并列出每一个订单：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderText
Dim ra As New prt.RenderArea
Dim Agg As New prt.DataBinding.Aggregate("GroupSum")

ra.Style.Spacing.Bottom = 4
ra.DataBinding.DataSource = BindTables("订单") '将容器绑定到订单表
ra.DataBinding.Grouping.Expressions.Add("Fields!产品.Value") '增加产品分组

agg.DataBinding = ra.DataBinding 'Agg 的 DataBinding 属性，应该设为打印对象的 DataBinding
agg.Running = 1 '分组统计
agg.ExpressionText = "Fields!数量.Value" '统计字段
doc.DataSchema.Aggregates.Add(agg)'定义好的统计必须添加到报表的 DataSchema 中

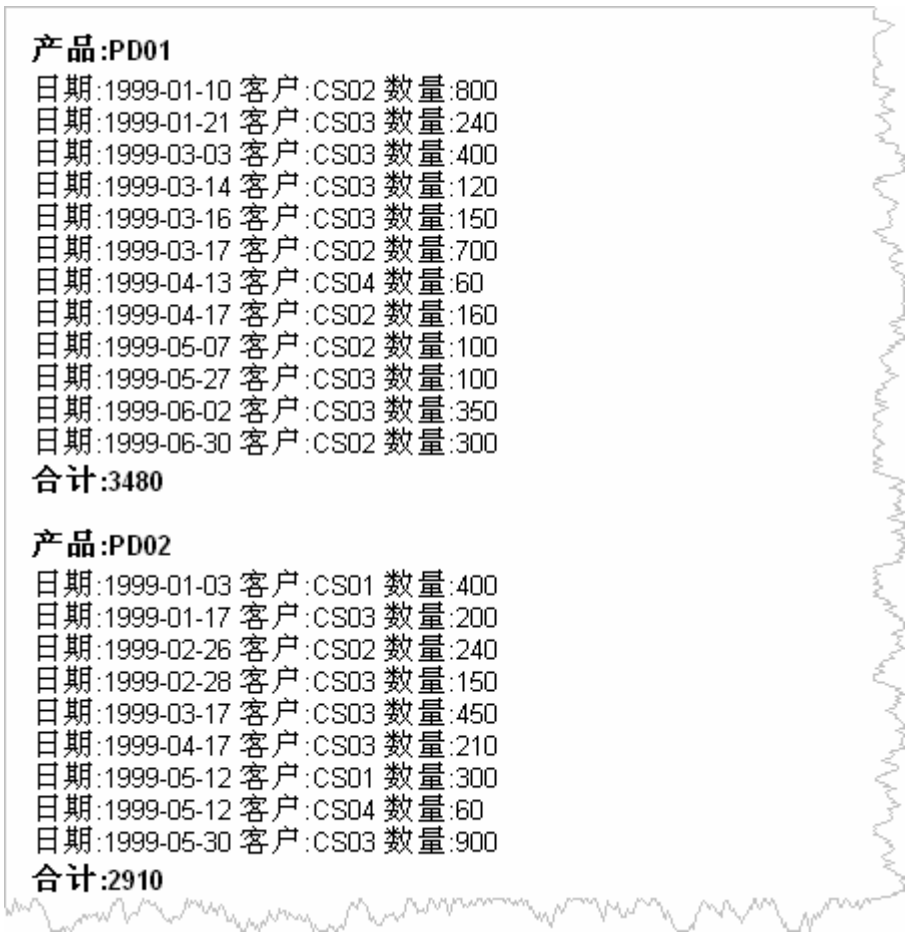
rt.Text= "产品:[Fields!产品.Value]"
rt.Style.FontBold = True
rt.Style.Spacing.Bottom = 1
ra.Children.Add(rt)

'这一个 RenderText 需要单独设置 DataBinding，这样才会针对每一个数据行打印副本
rt = New prt.RenderText
rt.Text = "日期:[Format(Fields!日期.Value,""yyyy-MM-dd"")] 客户:[Fields!客户.Value] 数量:[Fields!数量.Value]"
rt.DataBinding.DataSource = ra.DataBinding.DataSource '不能漏掉这一行
ra.Children.Add(rt)

rt = New prt.RenderText
rt.Text= "合计:[Aggregates!GroupSum.Value]"
rt.Style.FontBold = True
rt.Style.Spacing.Top = 1
ra.Children.Add(rt)

doc.body.Children.Add(ra)
doc.Preview()
```

执行结果：



5.9.2.4.7 表格与统计

用表格形式统计不同产品的销售额。

代码如下：

```
Dim doc As New PrintDoc
Dim rx As New prt.RenderText
Dim rt As New prt.RenderTable
Dim Agg1 As New prt.DataBinding.Aggregate("数量小计")
Dim Agg2 As New prt.DataBinding.Aggregate("金额小计")

rt.Style.GridLines.All = New prt.Linedef(Color.Gray)
rt.CellStyle.Spacing.All = 1
rt.Cells(0,0).Text = "产品"
rt.Cells(0,1).Text = "数量"
rt.Cells(0,2).Text = "金额"
rt.Cells(1,0).Text = "[Fields!产品.Value]"
rt.Cells(1,1).Text = "[Aggregates!数量小计.Value]"
rt.Cells(1,2).Text = "[Aggregates!金额小计.Value]"
```

```
With rt.RowGroups(1,1)
    .DataBinding.DataSource = BindTables("订单")
    .DataBinding.Grouping.Expressions.Add("Fields!产品.Value")

    agg1.DataBinding = .DataBinding
    agg1.Running = 1
    agg1.ExpressionText = "Fields!数量.Value"
    doc.DataSchema.Aggregates.Add(agg1)

    agg2.DataBinding = .DataBinding
    agg2.Running = 1
    agg2.ExpressionText = "Fields!金额.Value"
    doc.DataSchema.Aggregates.Add(agg2)
End With

doc.body.Children.Add(rt)
doc.Preview()
```

执行结果：

产品	数量	金额
PD05	10400	193023.15
PD02	18200	296522.5
PD01	11290	171754.92
PD03	7000	61490
PD04	5480	102861

5.9.2.4.8 汇总模式

我们可以完全用绑定来生成一个汇总模式：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim tb as Table = Tables("订单")
Dim ColNames As String() = New String(){"产品", "客户","单价", "数量", "金额"}
rt.Style.GridLines.All = New prt.Linedef(Color.Gray)
rt.CellStyle.Spacing.All = 1
For c As integer = 0 To ColNames.Length -1
    rt.Cells(0,c).Text = ColNames(c)
```

```
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth
    rt.Cells(1, c).Text = "[Fields!" & ColNames(c) & ".Value]"
Next
rt.Cells(2,0).Text ="小计"
rt.Cells(2,0).SpanCols = 3
rt.Cells(2,3).Text = "[Aggregates!数量小计.Value]"
rt.Cells(2,4).Text = "[Aggregates!金额小计.Value]"
rt.Cells(3,0).Text ="总计"
rt.Cells(3,0).SpanCols = 3
rt.Cells(3,3).Text = "[Aggregates!数量总计.Value]"
rt.Cells(3,4).Text = "[Aggregates!金额总计.Value]"
rt.Rows(0).Style.TextAlignVert = prt.AlignVertEnum.Center
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All
With rt.RowGroups(1,2)
    .DataBinding.DataSource = BindTables("订单")
    .DataBinding.Grouping.Expressions.Add("Fields!产品.Value")
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("数量小计", "Fields!数量.Value", .DataBinding, 1, 0))
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("金额小计", "Fields!金额.Value", .DataBinding, 1, 0))
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("数量总计", "Fields!数量.Value", .DataBinding, 0, 0))
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("金额总计", "Fields!金额.Value", .DataBinding, 0, 0))
End With
rt.RowGroups(1,1).DataBinding.DataSource = BindTables("订单")
doc.Body.Children.Add(rt)
doc.Preview()
```

执行结果：

产品	客户	单价	数量	金额
PD03	CS02	8	700	4200
PD03	CS02	10	200	1800
PD03	CS02	10	250	2500
PD03	CS01	10	240	2400
PD03	CS05	10	200	2000
PD03	CS05	10	300	3000
小计			2920	25040
PD04	CS04	17.6	240	4224
PD04	CS05	22	500	10450
PD04	CS03	22	30	660
PD04	CS03	22	200	3520
PD04	CS04	22	500	9900
小计			1470	28754
PD05	CS03	17.01	650	9729.72

下面的代码根据产品和客户分组，对数量和金额进行统计，代码比较繁琐，得有点钻研精神，才可以看明白的：

```
Dim doc As New PrintDoc
Dim rt As New prt.RenderTable
Dim tb as Table = Tables("订单")
Dim ColNames As String() = New String(){"产品", "客户","单价", "数量", "金额"}
rt.Style.GridLines.All = New prt.Linedef(Color.Gray)
rt.CellStyle.Spacing.All = 1
For c As integer = 0 To ColNames.Length -1
    rt.Cells(0,c).Text = ColNames(c)
    rt.Cols(c).Width = tb.Cols(ColNames(c)).PrintWidth
    rt.Cells(1, c).Text = "[Fields!" & ColNames(c) & ".Value]"
Next
rt.Cells(2,0).Text ="[Fields!客户.Value]小计"
rt.Cells(2,0).SpanCols = 3
rt.Cells(2,3).Text = "[Aggregates!数量小计 2.Value]"
rt.Cells(2,4).Text = "[Aggregates!金额小计 2.Value]"
rt.Cells(3,0).Text ="[Fields!产品.Value]小计"
rt.Cells(3,0).SpanCols = 3
rt.Cells(3,3).Text = "[Aggregates!数量小计 1.Value]"
rt.Cells(3,4).Text = "[Aggregates!金额小计 1.Value]"
rt.Cells(4,0).Text ="总计"
rt.Cells(4,0).SpanCols = 3
rt.Cells(4,3).Text = "[Aggregates!数量总计.Value]"
rt.Cells(4,4).Text = "[Aggregates!金额总计.Value]"
rt.Rows(0).Style.TextAlignVert = prt.AlignVertEnum.Center
rt.RowGroups(0,1).Header = prt.TableHeaderEnum.All
With rt.RowGroups(1,3)
    .DataBinding.DataSource = BindTables("订单")
    .DataBinding.Grouping.Expressions.Add("Fields!产品.Value")
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("数量小计 1", "Fields!数量.Value", .DataBinding, 1, 0))
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("金额小计 1", "Fields!金额.Value", .DataBinding, 1, 0))
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("数量总计", "Fields!数量.Value", .DataBinding, 0, 0))
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("金额总计", "Fields!金额.Value", .DataBinding, 0, 0))
End With
With rt.RowGroups(1,2)
    .DataBinding.DataSource = BindTables("订单")
    .DataBinding.Grouping.Expressions.Add("Fields!客户.Value")
     doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("数量小计 2", "Fields!数量.Value", .DataBinding, 1, 0))
```

```
    doc.DataSchema.Aggregates.Add(New prt.DataBinding.Aggregate("金额小计2", "Fields!金额.Value", .DataBinding, 1, 0))
End With
rt.RowGroups(1,1).DataBinding.DataSource = BindTables("订单")
doc.Body.Children.Add(rt)
doc.Preview()
```

6. 发布项目

6.1 关于发布

Foxtable 分为商业版和开发版，大家从主页下载的试用版也是商业版，试用期为半年左右。
商业版和开发版的差别在于有无发布功能，其余功能完全一致。
商业版开发的程序，可以复制到任何电脑运行，但是目标电脑也必须安装有 Foxtable。
开发版在商业版的基础上，增加了发布功能，项目经过发布后可以脱离 Foxtable，直接复制到任何电脑运行，不再需要安装 Foxtable。
此外经过发布后的程序，程序更加精简，运行速度更快。
所以本节的内容是针对开发版的，如果你是非开发版用户，可以跳过。

6.2 发布项目

开发版项目文件的后缀名为 FoxDB，格式和商业版不兼容。
如果您之前已经基于商业版开发了项目，之后又购买了开发版，那么自购买开发版起的一个月之内，我们可以免费提供两次将商业版项目转换为开发版项目的服务。
开发版可以发布设计好的项目，经过发布的项目，可以生成可执行文件，直接在任何电脑运行，不再需要安装和注册 Foxtable，所以不会再有额外费用的产生。
发布步骤
1、在管理项目功能区，单击“项目发布”按钮，会出现如下窗口，设置好相关栏目：



2、上述的栏目，只有可执行文件名是必须设置的；设置好之后，单击发布按钮，即可在项目文件所在目录下生成一个 Publish 子目录，该目录包括通过发布生成的所有文件。

3、将此 Publish 目录复制到其它电脑，即可通过执行自定义的可执行文件来启动自己的管理项目，例如上图生成的可执行文件为 DataMis.Exe。

4、项目发布功能会自动将需要的文件复制到 Publish 子目录，绝大多数时候，是不再需要人工干预的；如果还有必要的文件没有成功复制到 Publish 目录，可以单击“添加附件”，手工添加需要发布的文件和目录。

5、发布设置可以自动保存，下次发布，只需直接单击发布按钮即可，无需重新设置。

提示:

1、目标电脑必须安装有.Net Framework 2.0 或者更新的版本，才能运行发布后的程序 ，由于 Windows 7/Vista 系统已经默认有.Net Framework 2.0，所以可直接运行，而部分 Windows XP 的系统需要安装.Net Framework 2.0。

2、当我们运行发布后的程序时，Publish 目录相当于原来的 Foxtable 安装目录，而 Publish 目录下的子目录 Project，则相当于原来的项目文件所在目录，项目文件就位于这个目录中，只是后缀名已经改为".FoxEx"。

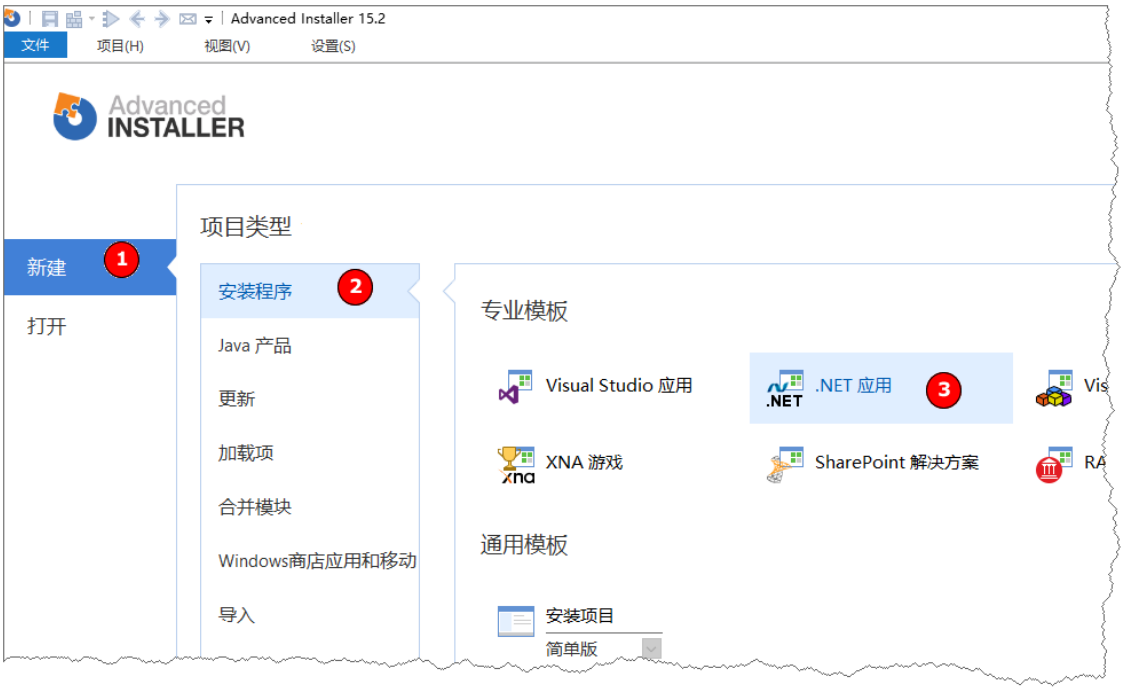
路径对应关系

项目发布之后，生成的 Publish 目录对应发布前的 Foxtable 安装目录，Publish 下的子目录 Project 对应发布前的项目目录，Project 子目录中有一个和原项目同名，但是后缀名为 FoxEx 的文件，此

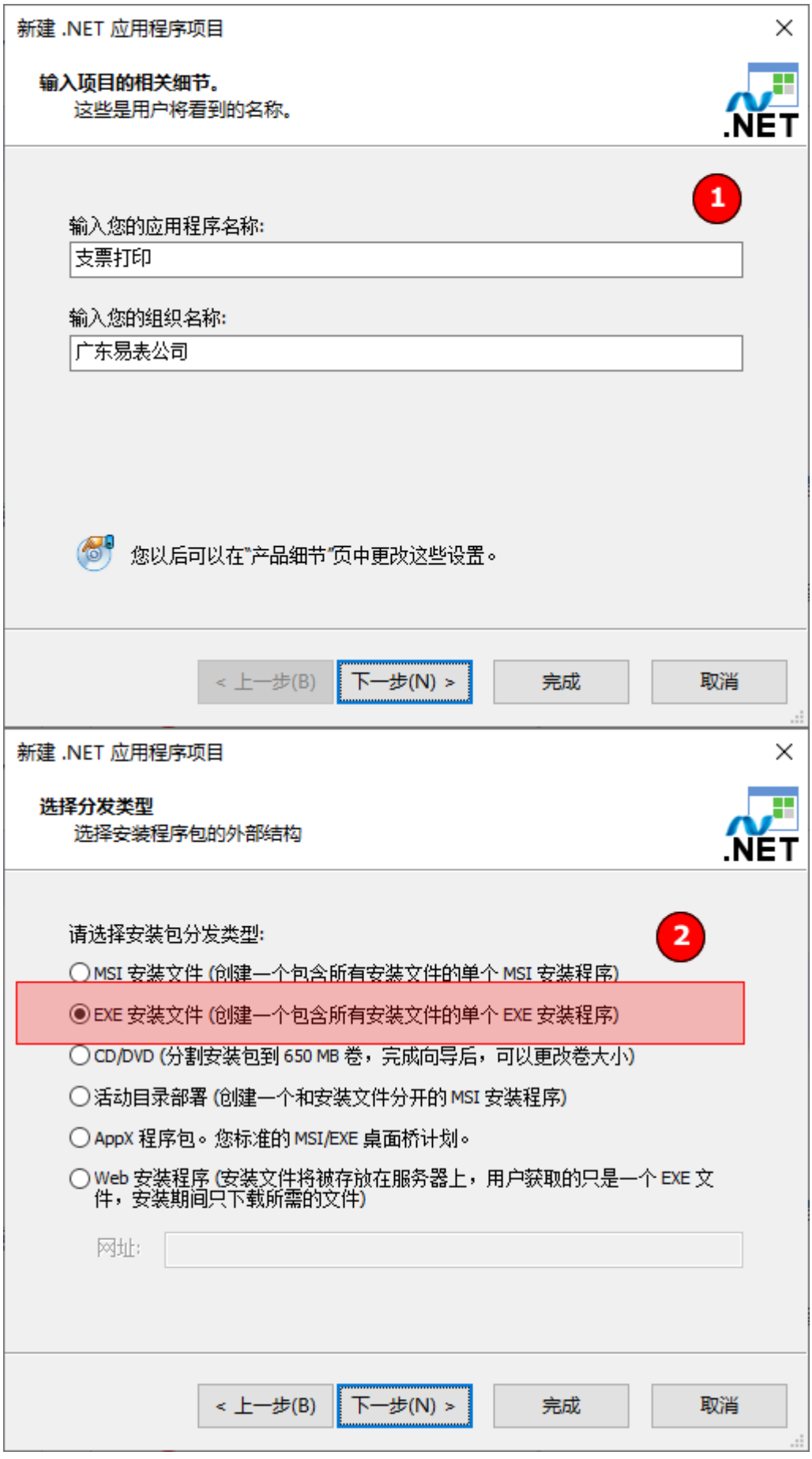
文件就是发布后的项目文件。
例如你有一些第三方的 Dll 文件，发布前的话，需要复制到 Foxtable 的安装目录才能引用，发布后，需要复制到 Publish 目录才能正常引用。
如果你原来的项目文件所在目录，有一些附属文件用于辅助项目的执行，那么发布后，应该将这些文件复制到 Project 子目录中；如果还有附属目录，那么这些附属目录也应该复制到 Project 中，其中 Attachments 和 Images 两个子目录会自动复制到 Project 目录中的，无需手工复制。

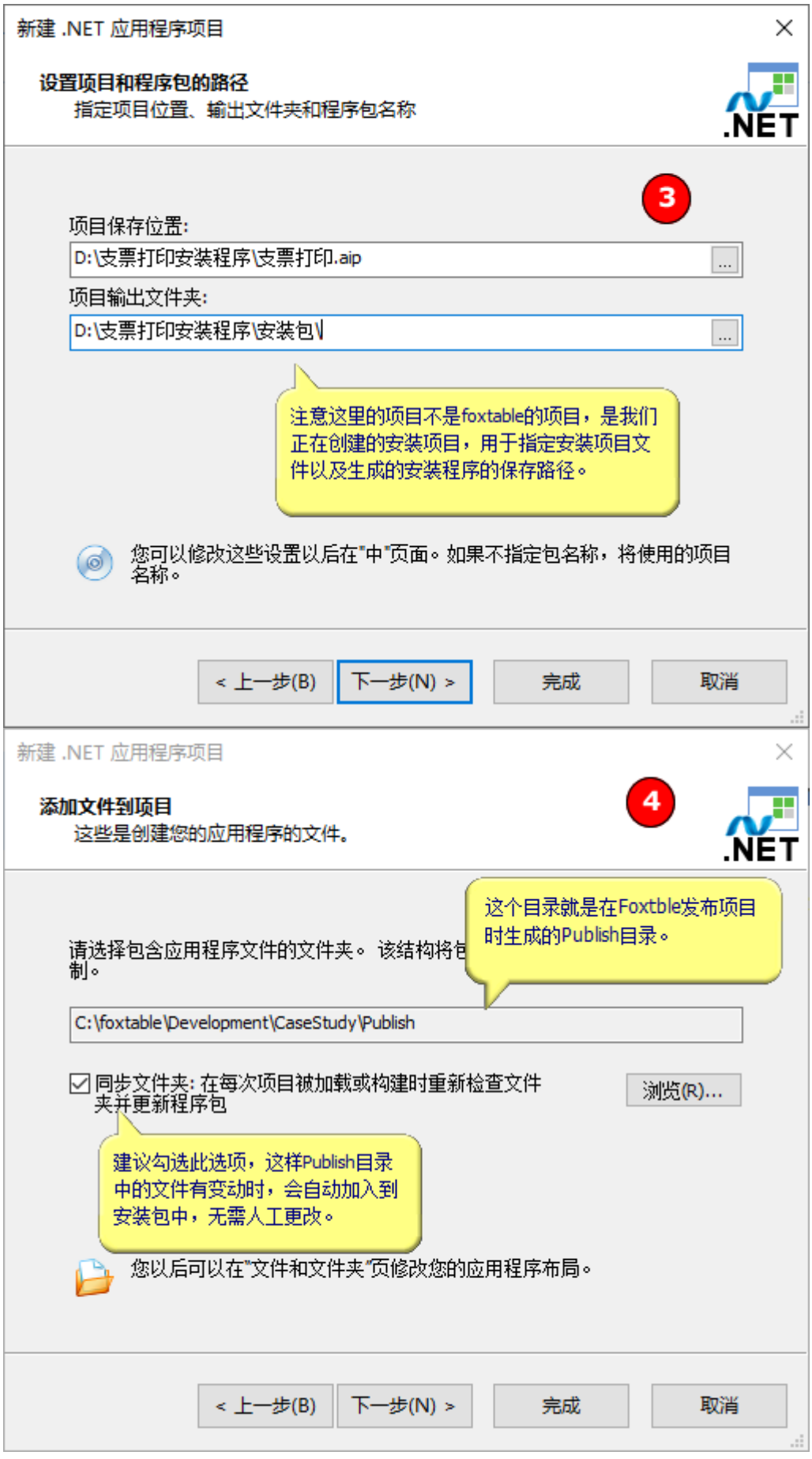
6.3 制作安装程序

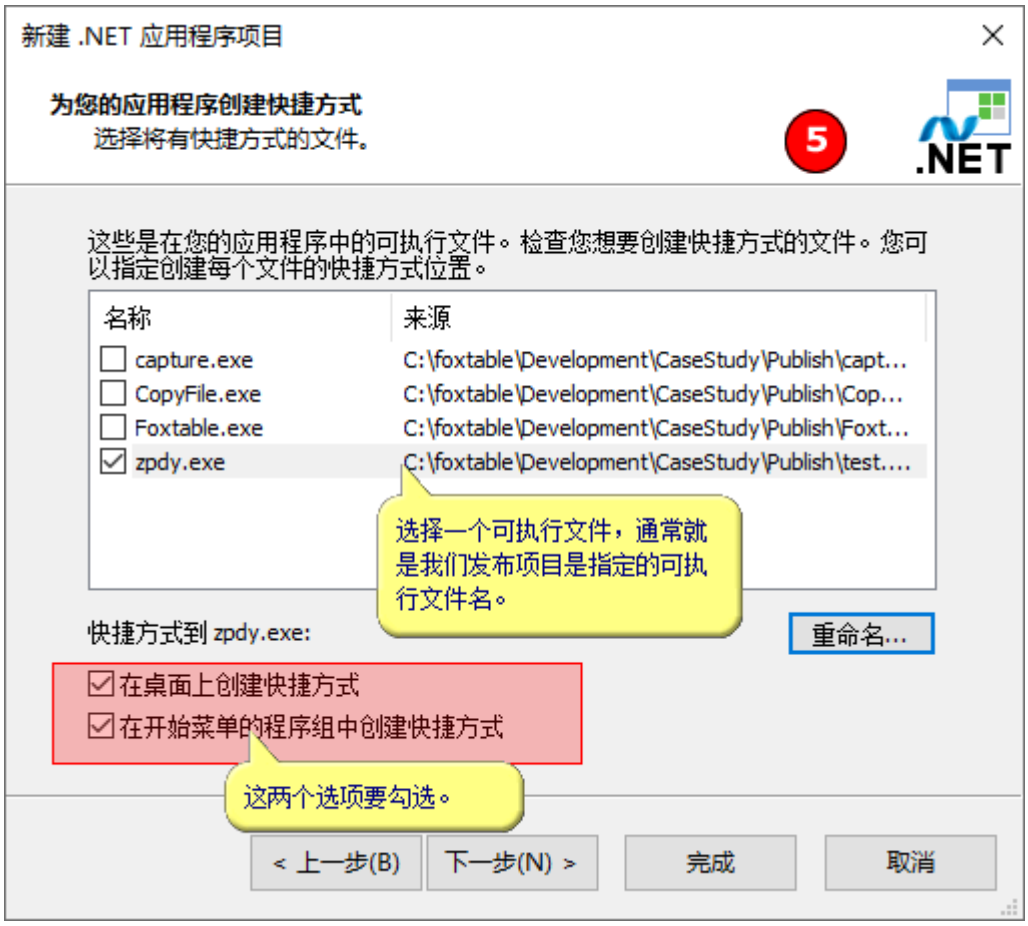
前面介绍了通过 Foxtable 的发布功能，能够生成一个 Publish 子目录，只需将此目录复制到任何电脑即可运行，不再需要 Foxtable。
我们可以用第三方的打包软件，将 Publish 目录的内容，做成一个安装程序，这样对于一些初级用户，会显得更友好一些，也便于公开发行自己设计的产品。
第三方的打包软件很多，这里介绍的是 Advanced Installer，该软件使用简单，功能也很强大。
下面我们以 Advanced Installer 15.2 为例，开始介绍安装程序的具体制作过程：
一、启动 Advanced Installer 15.2，选择新建页面，然后双击"安装程序"子页面中的".NET 应用"，如下图：



二、双击".NET 应用"之后，会连续出现多个向导对话框，按以下图示操作即可：

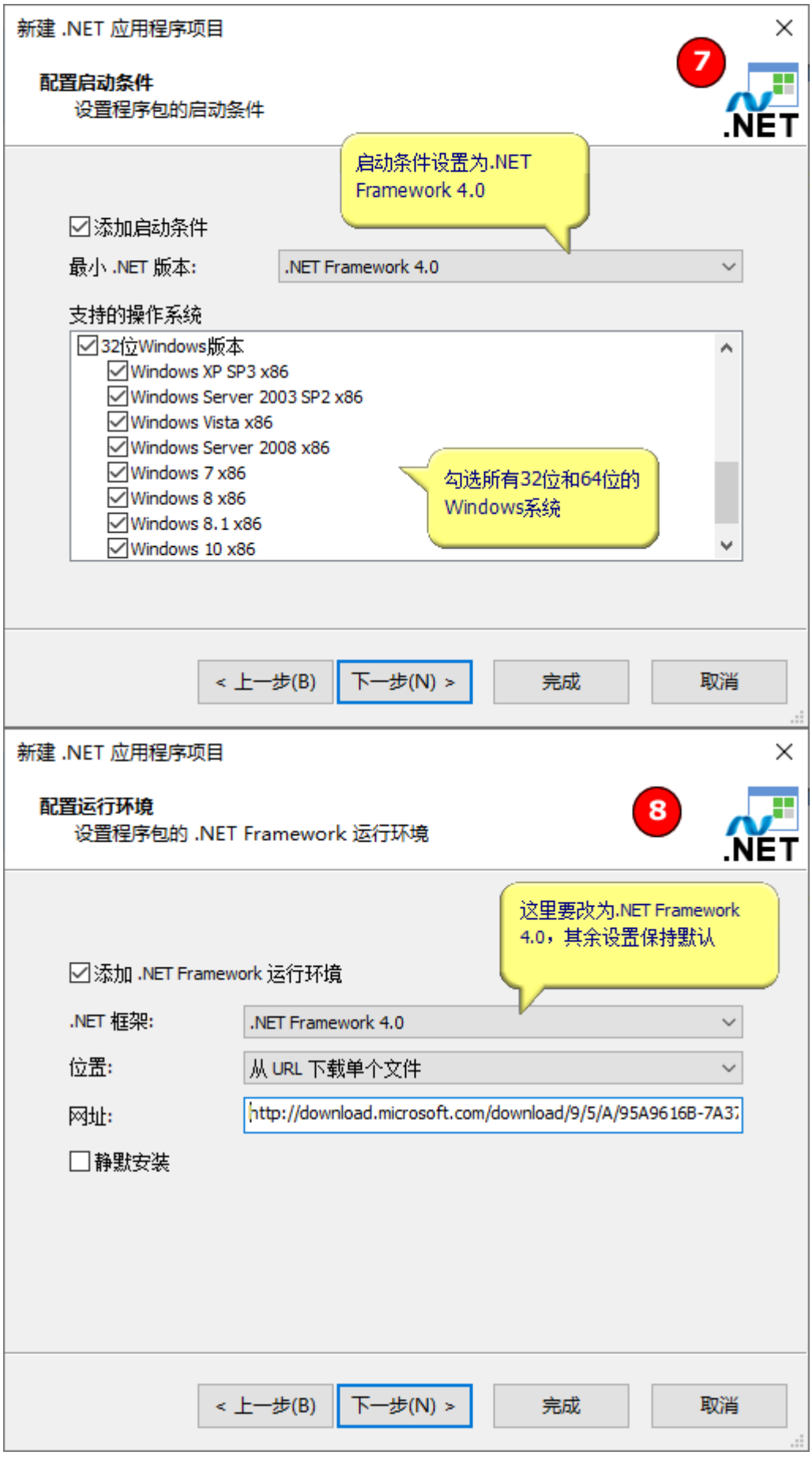

新建 .NET 应用程序项目
输入项目的相关细节。
这些是用户将看到的名称。
.NET
1
输入您的应用程序名称:
支票打印
输入您的组织名称:
广东易表公司
您以后可以在"产品细节"页中更改这些设置。
< 上一步(B)
下一步(N) >
完成
取消
新建 .NET 应用程序项目
选择分发类型
选择安装程序包的外部结构
.NET
2
请选择安装包分发类型:
MSI 安装文件 (创建一个包含所有安装文件的单个 MSI 安装程序)
EXE 安装文件 (创建一个包含所有安装文件的单个 EXE 安装程序)
CD/DVD (分割安装包到 650 MB 卷，完成向导后，可以更改卷大小)
活动目录部署 (创建一个和安装文件分开的 MSI 安装程序)
AppX 程序包。您标准的 MSI/EXE 桌面桥计划。
Web 安装程序 (安装文件将被存放在服务器上，用户获取的只是一个 EXE 文件，安装期间只下载所需的文件)
网址:
< 上一步(B)
下一步(N) >
完成
取消


新建 .NET 应用程序项目
设置项目和程序包的路径
指定项目位置、输出文件夹和程序包名称
.NET
3
项目保存位置:
D:\支票打印安装程序\支票打印.aip
项目输出文件夹:
D:\支票打印安装程序\安装包\
注意这里的项目不是foxtable的项目，是我们正在创建的安装项目，用于指定安装项目文件以及生成的安装程序的保存路径。
您可以修改这些设置以后在"中"页面。如果不指定包名称，将使用的项目名称。
< 上一步(B)
下一步(N) >
完成
取消
新建 .NET 应用程序项目
添加文件到项目
这些是创建您的应用程序的文件。
4
.NET
这个目录就是在Foxtble发布项目时生成的Publish目录。
请选择包含应用程序文件的文件夹。 该结构将包
制。
C:\foxtable\Development\CaseStudy\Publish
同步文件夹: 在每次项目被加载或构建时重新检查文件夹并更新程序包
浏览(R)...
建议勾选此选项，这样Publish目录中的文件有变动时，会自动加入到安装包中，无需人工更改。
您以后可以在"文件和文件夹"页修改您的应用程序布局。
< 上一步(B)
下一步(N) >
完成
取消



在上图中单击重命名按钮，为可执行文件制定快捷方式名称：




新建 .NET 应用程序项目
配置启动条件
设置程序包的启动条件
7
.NET
启动条件设置为.NET Framework 4.0
添加启动条件
最小 .NET 版本:
.NET Framework 4.0
支持的操作系统
32位Windows版本
Windows XP SP3 x86
Windows Server 2003 SP2 x86
Windows Vista x86
Windows Server 2008 x86
Windows 7 x86
Windows 8 x86
Windows 8.1 x86
Windows 10 x86
勾选所有32位和64位的Windows系统
< 上一步(B)
下一步(N) >
完成
取消
新建 .NET 应用程序项目
配置运行环境
设置程序包的 .NET Framework 运行环境
8
.NET
这里要改为.NET Framework 4.0，其余设置保持默认
添加 .NET Framework 运行环境
.NET 框架:
.NET Framework 4.0
位置:
从 URL 下载单个文件
网址:
http://download.microsoft.com/download/9/5/A/95A9616B-7A37
静默安装
< 上一步(B)
下一步(N) >
完成
取消



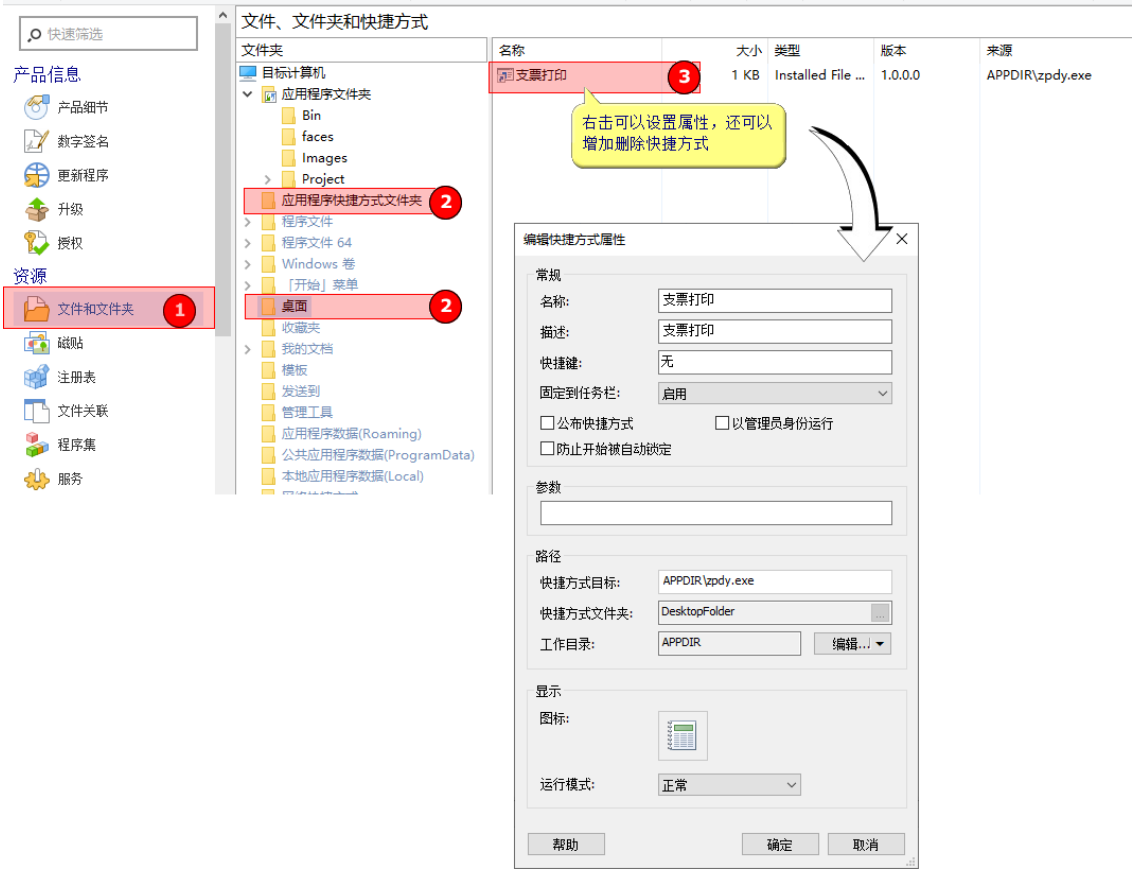
现在可以单击完成按钮，完成向导设置了。

三、默认的安装目录是 Program Files，不过这个目录并不适合安装基于 Foxtable 开发的程序，因为 Foxtable 在运行的过程中要编译用户代码，因此会动态生成和删除一些 DLL 文件，而在 Program Files 下动态生成和删除 DLL 文件，有时会被操作系统和一些安全软件禁止，从而导致 Foxtable 无法运行。

所以大家在制作安装程序的时候，一定要另外指定一个自己的目录，例如“c:\datamis”，"d:\jytsoft" 等等：



四、为桌面和开始菜单修改或创建快捷方式：

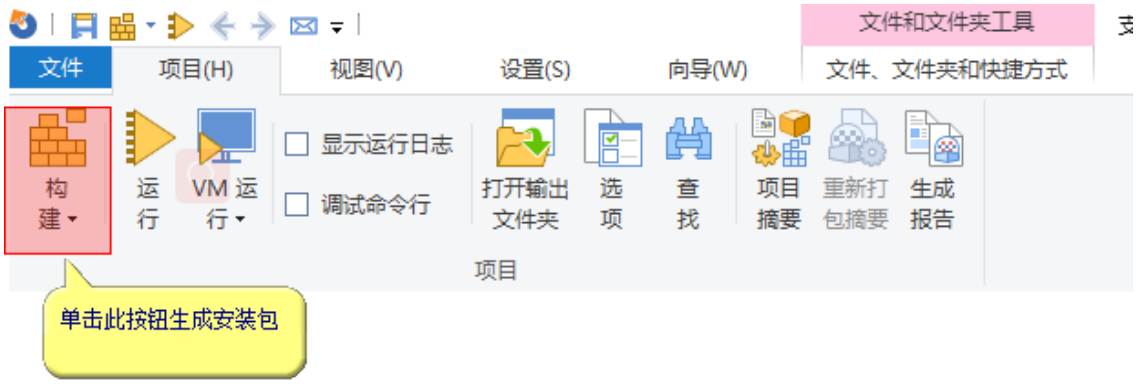


重要 提示：

本来选择上图中的"应用程序文件夹"，可以增加或删除要发布的文件，但由于我们之前已经在向导

窗口中选择了“同步文件文件夹”，所以会自动增加删除文件，无需再人工干预了 ，如果要增加或删除发布文件，应该事先在 Publish 目录增加或删除文件。

五、设置完成后，可以单击下图所示的按钮生成安装包：



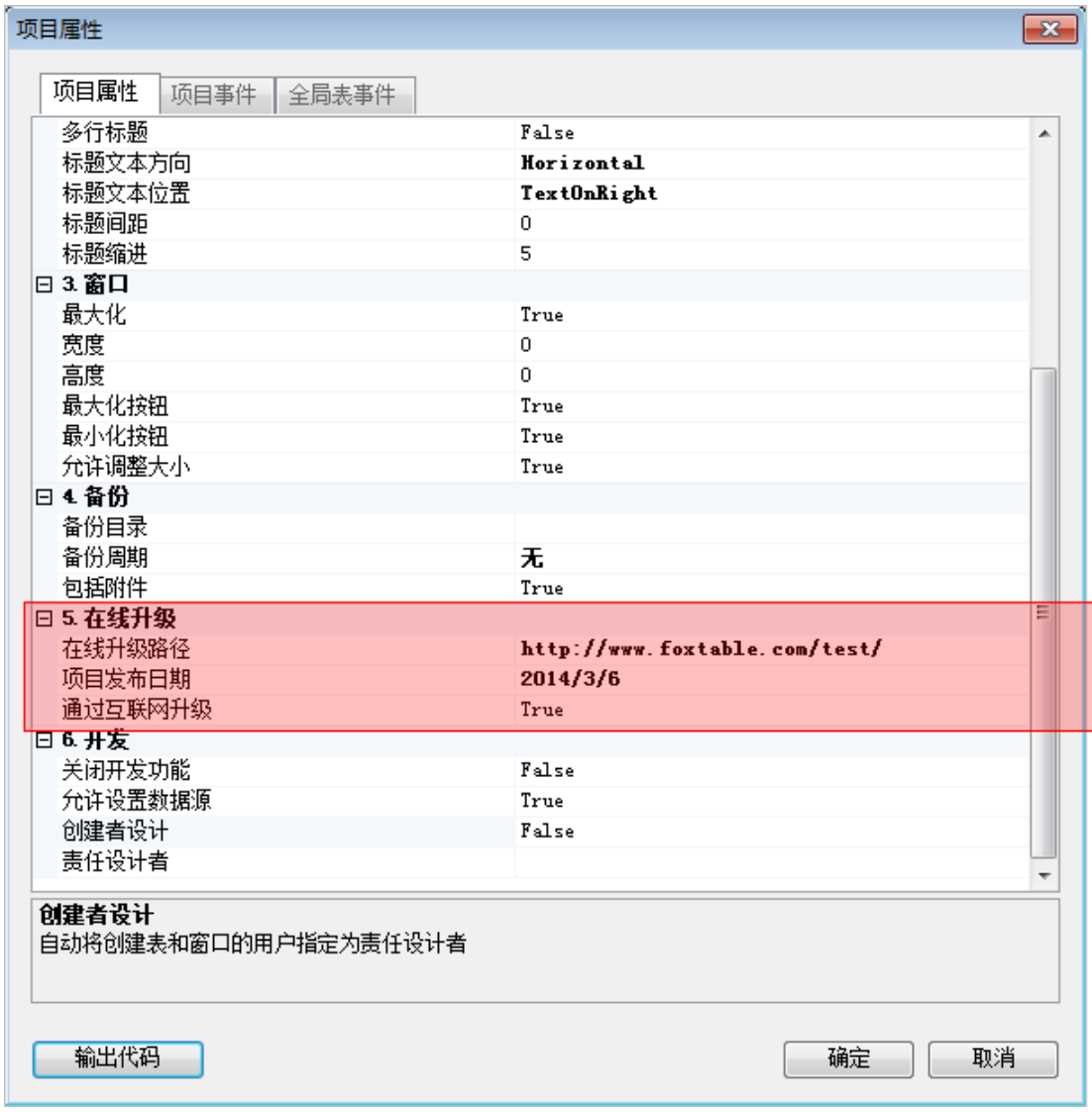
6.4 发布后程序的远程更新

发布后程序的远程更新，包括项目本身的更新，以及 Foxtable 运行环境的更新。

提示：由于本节的内容是针对发布后的程序，所以无法在发布前进行测试。

设计步骤：

1、项目属性中有两项用于设置远程升级：



- 在线升级路径
  指定远程升级包的下载路径。

- 项目发布日期
  指定项目的发布日期，此日期将用于和远程升级包的发布日期进行比较，以判断当前项目是否需要升级。

2、制作升级文件
升级文件有三个：

- **update.txt**

  update.txt 的内容是两个用符号"|"隔开的日期，前一个日期表示项目的版本，后一个日期表示运行此项目需要的 Foxtable 版本。

  例如：

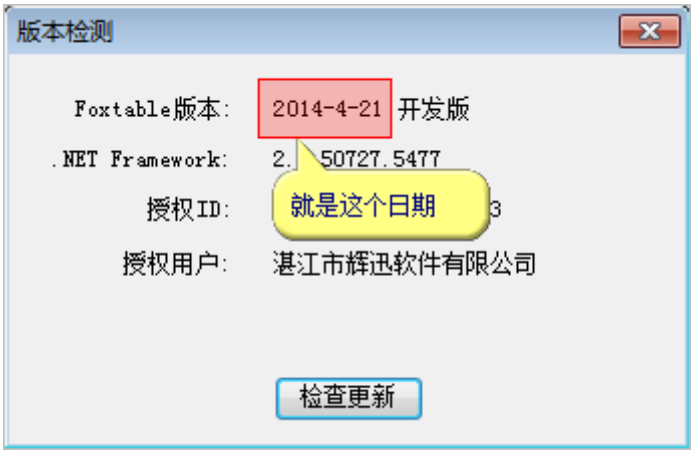
2014-04-28|2014-03-08

表示项目的版本是 2014-04-28，运行此项目的 Foxtable 版本是 2014-03-08。

提示：

A、第一个日期必须和项目属性中设置的版本日期一致，否则会反复提示升级。

B、第二个日期就是你用于开发的 Foxtable 的版本日期，通过“菜单 - 杂项 - 工具 - 检查新版本”，可以查出此日期：

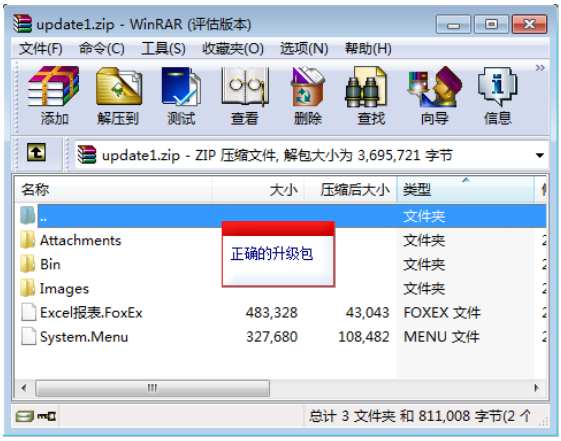


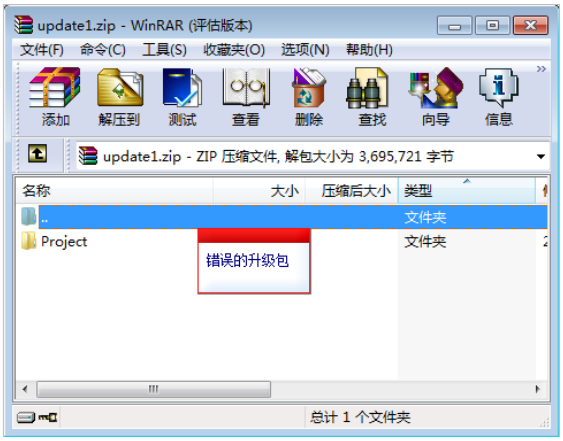
- **update1.zip**

update1.zip 是简化升级包，只包括项目，不包括运行环境。

打开项目文件夹，进入 Publish 目录下的 Project 子目录，将其中所有的文件和子目录压缩到一个名为 update1.zip 的文件中，压缩的时候必须选择 zip 格式。

请注意一个细节，是将 Project 文件夹下的文件及其子目录压缩到升级包中，而不是直接将整个 Project 文件夹压缩到升级包中：



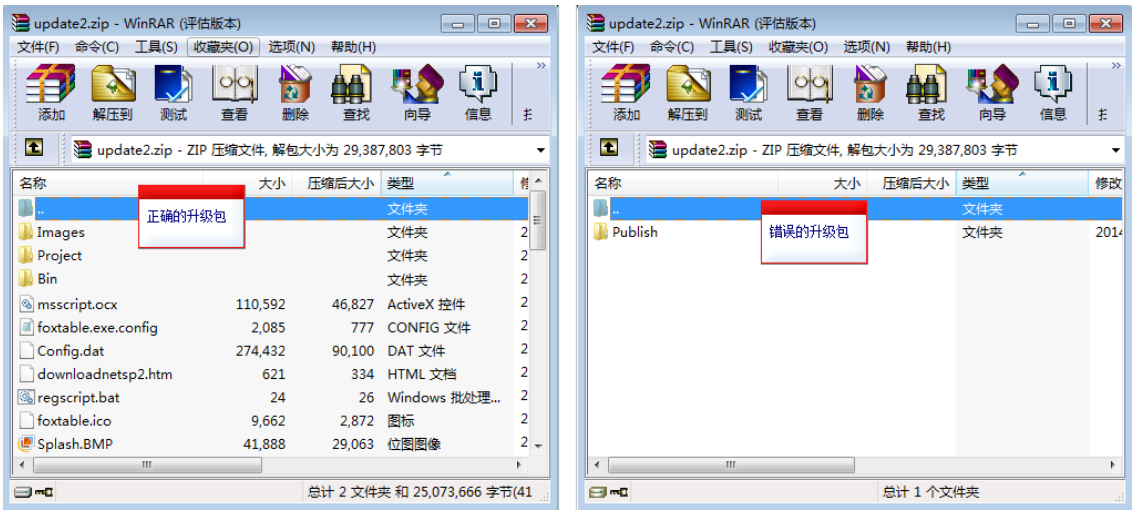


- **update2.zip**

  update2.zip 是完整升级包，包括运行环境和项目。

  打开项目文件夹，进入 Publish 目录，将其中所有的文件和子目录压缩到一个名为 update2.zip 的文件中，压缩的时候必须选择 zip 格式。

  请注意一个细节，是将 Publish 文件夹下的文件及其子目录压缩到升级包中，而不是直接将整个 Publish 文件夹压缩到升级包中：



最后将制作好的 update.txt、update1.zip、update2.zip 三个文件上传到项目属性中指定的在线升级路径中。

3、在线升级

需要提醒一下，本节介绍的内容是针对发布后程序的远程自动升级，所以要测试的话，必须运行发布后的程序进行测试。

你可以在你的项目中加上一个在线升级的菜单命令或者按钮，代码很简单：

```
Syscmd.Project.Update()
```

执行上面的代码，系统首先检索远程升级路径中的 update.txt 文件，获得项目版本日期和 Foxtable 版本日期，和本机的项目版本日期和 Foxtable 版本日期进行比较，然后根据比较结果进行动作：

A、如果本机的项目版本日期大于等于 update.txt 中的项目版本日期，则不进行任何动作。

B、如果本机的项目版本日期小于 update.txt 中的项目版本日期，且本机的 Foxtable 版本日期等于 update.txt 中的 Foxtable 版本日期，则开始下载简化升级包 update1.zip。

C、如果本机的项目版本日期小于 update.txt 中的项目版本日期，且本机的 Foxtable 版本日期不等于 update.txt 中的 Foxtable 版本日期，则开始下载完整升级包 update2.zip。

升级包下载完成后，会自动解压并覆盖本机的文件，完成升级。

因为单单升级项目本身的频率会远远超过同时升级项目和运行环境的频率，所以提供两个升级包，

可以让客户端的日常升级更加快捷。
至此我们的在线升级功能已经完成，即使你是一个初次接触编程的人，应该也能即时掌握。
Update 的语法：
Syscmd.Project.Update(Prom1, Prom2)

Prom1: 可选参数，逻辑型，没有发现新版本时，是否提示用户。
Prom2: 可选参数，逻辑型，发现新版本时，是否提示用户升级。
例如希望自动升级能够完全"静默"，发现新版本时不提示，自动开始升级，没有发现新版本时也不提示：
Syscmd.Project.Update(False,False)
如果我们在 AfterOpenProject 下面的代码：

```
Syscmd.Project.Update(False,False)
Forms("窗口 1").Open() '可能会出错
'后续其它代码
```

目的是希望打开项目的时候自动升级，然后打开窗口 1。
但是上述代码肯定会在升级成功之后提示错误，因为一旦升级成功，Foxtable 就会关闭，后面的打开窗口代码就会出错，当然后续其它的代码也一样会出错。
我们可以用下面的代码避免错误：

```
If Syscmd.Project.Update(False,False) = False Then '如果没有升级
    Forms("窗口 1").open()
    '后续其它代码
End If
```

也就是说 Update 执行后会返回一个逻辑值，如果升级成功，返回 True，否则返回 False。

总之，如果需要在 AfterOpenPoject 事件中实现自动升级，那么原有的 AfterOpenProject 事件代码必须包括在这个判断中，只有没有升级的情况下，才执行原来的代码：

```
If Syscmd.Project.Update(False,False) = False Then '如果没有升级
    '原 AfterOpenProject 事件代码
End If
```

4、基于局域网的在线升级
在线升级默认是基于互联网的，在升级之前，会自动检测是否连接互联网。
如果要基于局域网进行在线升级，除了将升级包路径设置为局域网路径外，还必须将属性“通过互联网升级”设置为 False：



5、使用 FTP 地址升级

升级地址也可以使用 FTP 地址，例如：
ftp://zhanshang:138238@egridsoft.com/update
上面假定帐户为“zhanshang”，密码为“138238”，ftp 服务器的址为“ftp://egridsoft.com”，升级包所在目录为“update”。

重要提示：

三个符号“:/@”在升级地址定义中有特殊意义，用于分割协议、账户、密码、服务器地址以及路径，所以不可以在账户名和密码中出现上述符号。
6、和远程升级相关的系统变量

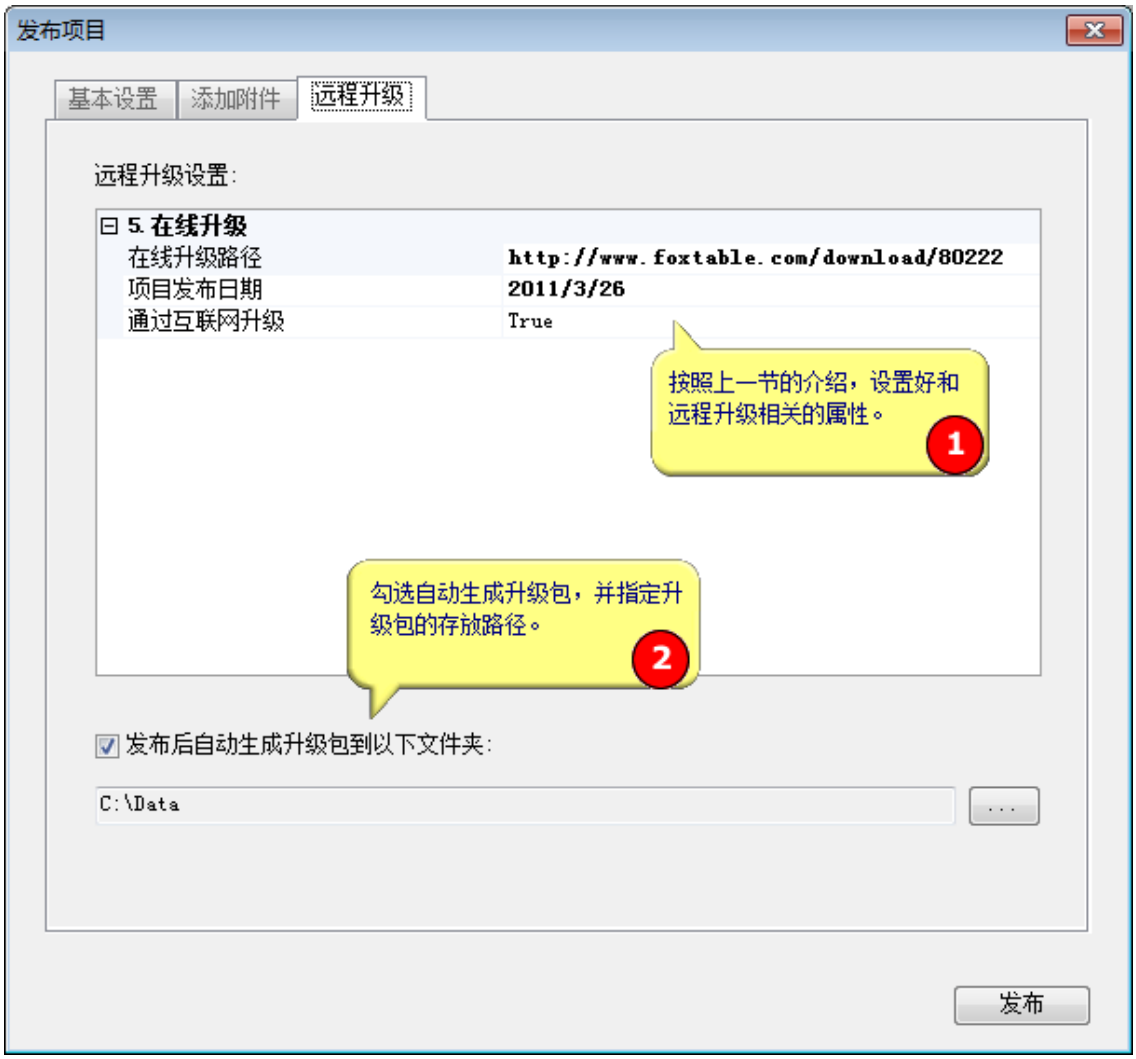
- PublishDate
  日期型变量，返回在项目属性中设置的项目发布日期。

- UpdatePath
  字符型，用于设置或返回在线升级路径，有了这个变量，我们可以动态设置升级路径。

- RemoteUpdate
  逻辑型，是否通过互联网升级。

6.5 自动生成远程升级文件

上一节介绍了如何针对发布后程序实现跨版本升级，我们是手工制作有关远程升级文件的。
如果没有特殊的要求，可以在发布项目时，让 Foxtable 自动为您生成相关远程升级文件：

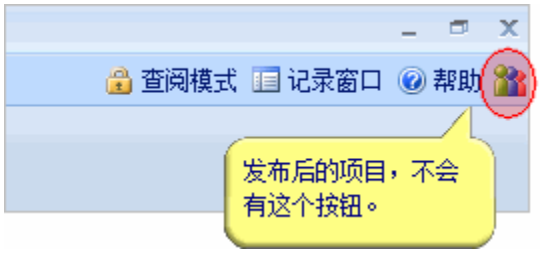


每次发布完成后，会自动在目标文件夹生成三个升级文件：
update.txt
update1.zip
update2.zip
你要做的只是将这三个文件上传到指定的远程升级路径。

6.6 自动消失的 Foxtable 痕迹

不管你使用的是商业版还是开发版，菜单中会有两处 Foxtable 的痕迹，而且是无法消除的。但是基于开发版开发项目，经过发布后，这两处痕迹就会自动消失：





例如下图是一个基于 Foxtable 开发的进销存软件，经过发布后，已经没有任何 Foxtable 的痕迹了，而且无需安装 Foxtable，可以直接复制到任何电脑运行，成为 了一个真正独立的软件：



6.7 自定义主程序和描述

实际上我们生成的 exe 文件，只是一个引导程序，最终运行的是 Foxtable 的主程序。
主程序默认的名称是 foxtable.exe。
我们可以在高级设置中，重新定义主程序的名称和描述。



防止杀毒软件误报

现在很多杀毒软件，只要是没有见过的可执行文件，都会报警，严重的会误报为病毒。

这导致用户发布后生成的可执行文件会被杀毒软件拦截
为解决这个问题，大家可以在发布后，删除自己的可执行文件，直接用主程序启动。

6.8 如何保护开发成果

此功能仅限高级开发版。
大型项目的开发，通常需要多个程序员组成一个开发团队，进行协同开发，最新版本的 Foxtable 对于协同开发的支持非常完善，具体可以参考《协同开发》这一章。
在多人协同开发的时候，如何有效防止参与开发的程序员盗用最终的开发成果，一直是一个难题。
如果使用 Foxtable 高级开发版，则不用担心这个问题，我们可以通过高级开发版限制开发人员的权限，让其只能参与开发，无法带走开发成果。
高级开发版通常配合普通开发版使用，团队主管使用高级开发版，团队成员使用普通开发版。
当然普通开发者也可以使用高级开发版，只需主管首先插上自己专属的高级开发版加密锁，对项目设置好权限即可，之后其他人即使使用高级开发版开发项目，也只能在设定权限下进行操作。
即使是一个人开发，也可以使用高级开发版授权，将授权 ID 设置为高级开发版自身的 ID 即可，这样其他人即使拿到你的项目，即使知道项目的开发者密码，也无法打开你的项目。
高级开发版可以实现：

1、禁止团队成员非法使用项目

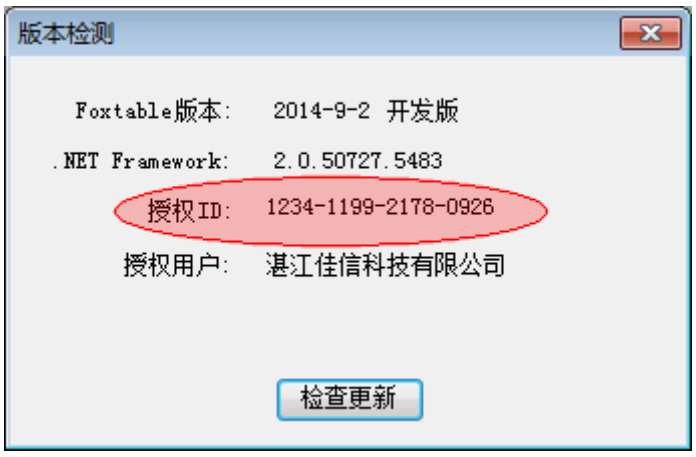
通过高级开发版，可以对项目进行授权设置，使得只有插上经授权的加密锁，才能打开该项目，否则即使是开发者本人，离开了授权的加密锁，也无法打开项目。

2、禁止团队成员一些关键权限

开发团队的主管通过高级开发版，还可以禁止参与团队成员在开发过程中的以下权限：
1、输出结构
2、输出代码
3、发布项目
4、将窗口另存为模版
5、将菜单另存为模版
显然，将上述权限全部禁止后，参与开发的程序员如果要盗用最终的开发成果，除了从零开始重写，别无他法。
所以团队开发版的推出，大大增强了团队主管的控制能力，能更好地第保护企业权益。

使用高级开发版进行授权的步骤：

1、首先逐个插上要授权的其他开发版的加密锁，启动 Foxtable 开发版，执行菜单命令“杂项 - 工具 - 检查新版本”，记录好窗口中显示的授权 ID：



2、然后插上高级开发版的加密锁，打开要进行授权设置的项目，在菜单的“管理项目”功能区单击“项目信息”：

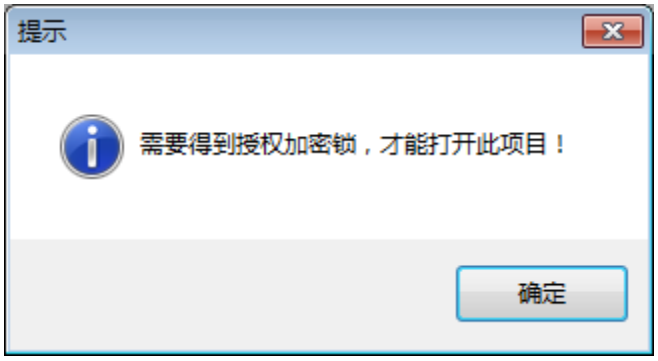


3、在出现的窗口中，选择“授权”页，输入要授权的加密锁的授权 ID，每行一个：

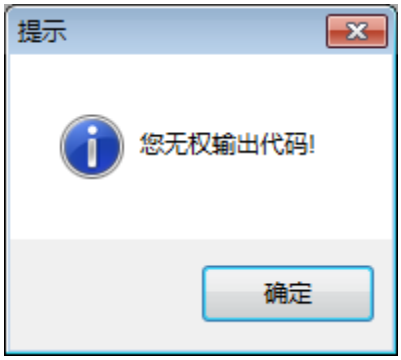


提示：只有高级开发版，才会出现“授权”页。

4、之后没有经过授权的加密锁，即使知道开发者密码，打开项目的时候也会提示：



5、团队成员使用经过授权的加密锁打开项目进行开发过程中，如果要执行未经授权的操作，例如输出代码的时候，会出现提示：

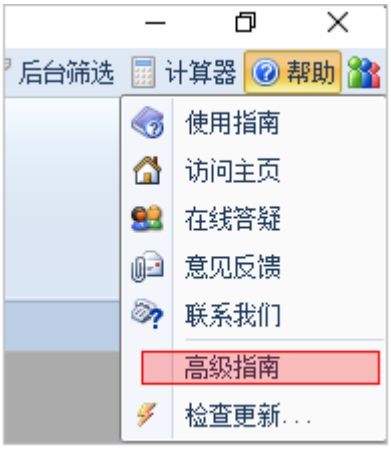


7. 高级开发指南

7.1 关于《高级开发指南》

除了本文档之外，Foxtable 还有一个《高级开发指南》的文档，用于介绍部分只有高级开发版才具备的功能，例如网页设计入门、手机管理系统的开发，微信接入、三层架构设计和多线程编程等等。

我们必须在已经掌握《使用指南》和《开发指南》的基础上，才能学习《高级开发指南》。
启动 Foxtable，单击右上角的帮助按钮，再单击"高级指南"命令，可以打开《高级开发指南》：



该文档目前由七个部分组成

- **HTML 入门**

  这一章介绍了如何开启一个 HTTP 服务，如果和用 HTML 生成网页，如何接收用户通过网页输入的数据，如何保存用户上传的文件等等。一般用户对于 HTML 稍作了解即可，因为实际开发的时候，我们一般会用内置的 WeUI 框架生成网页。

- **WeUI 框架**

  这一张是重点，通过这一章，你会发现在 Foxtable 生成网页是多么的简单。
  WeUI 框架来自于腾讯，我们对其进行了一些扩展，增加了表格和 AJAX 功能，通过框架，我们可以用非常简单的代码生成专业效果的网页，本框架主要针对移动设备，虽然也能在 PC 上使用，但在手机上的效果更好。

- **客户端类**

  这一章介绍了客户端类 HttpClient，Foxtable 通过 HttpClient 和 HTTP 服务进行交互，包括第三方的 HTTP 服务。我们在进行微信开发的时候，实际上就是通过 HttpClient 调用腾讯提供的 HTTP 服务。

- **JSON 解析**

  这一章介绍了 JSON 和 XML 数据的解析生成，互联网中的数据传递格式主要是 JSON 和 XML，例如进行微信开发的时候，你会发现除了回调部分的数据是 XML 格式，其余都是 JSON 格式。
  通过这一章，你会发现在 Foxtable 进行 JSON 和 XML 的解析与生成，实在是太轻松了。

- **微信接口**

  这一章介绍了如何在 Foxtable 进行微信开发，分别针对公众号和企业号进行讲述，基本上涵盖了所有微信常用接口，由于我们进行了适度的封装，微信开发变得前所未有的轻松。
  实际上不仅是微信开发，你可以用 Foxtable 完成任何第三方接口的开发，帮助就提供了一个阿里云快递查询接口的开发示例，只有短短的三行代码而已 。

- **Web 数据源**

  Foxtable 默认是两层结构，客户端直接连接数据库，这一章介绍如何使用 Web 数据源设计三层架构的系统，使得客户端不再直连数据库。

- **异步编程**
  这一节章介绍如何如何用异步编程同时处理多个任务，以提高服务器的负载能力。

《高级开发指南》在线地址：
http://www.foxtable.com/mobilehelp/

7.2 关于移动开发

Foxtable 的高级开发版提供了移动开发功能。

Foxtable 的移动开发功能以 B/S 形式实现，内置手机网页自动生成功能，你不需要额外的学习，你

可以完全不懂网页设计，也可以快速开发基于手机的企业管理系统，包括服务端的架设和网页的生成，全都可以在 Foxtable 中用最简单的方式完成，不仅简单到人人都能掌握，而且效率极高，相对于其它开发工具，代码节省了 85%左右。

此外高级开发版还提供了微信接入功能，用户可以直接通过微信输入、查询、统计和审核数据。

下面是 Foxtable 自动生成的一些手机网页：

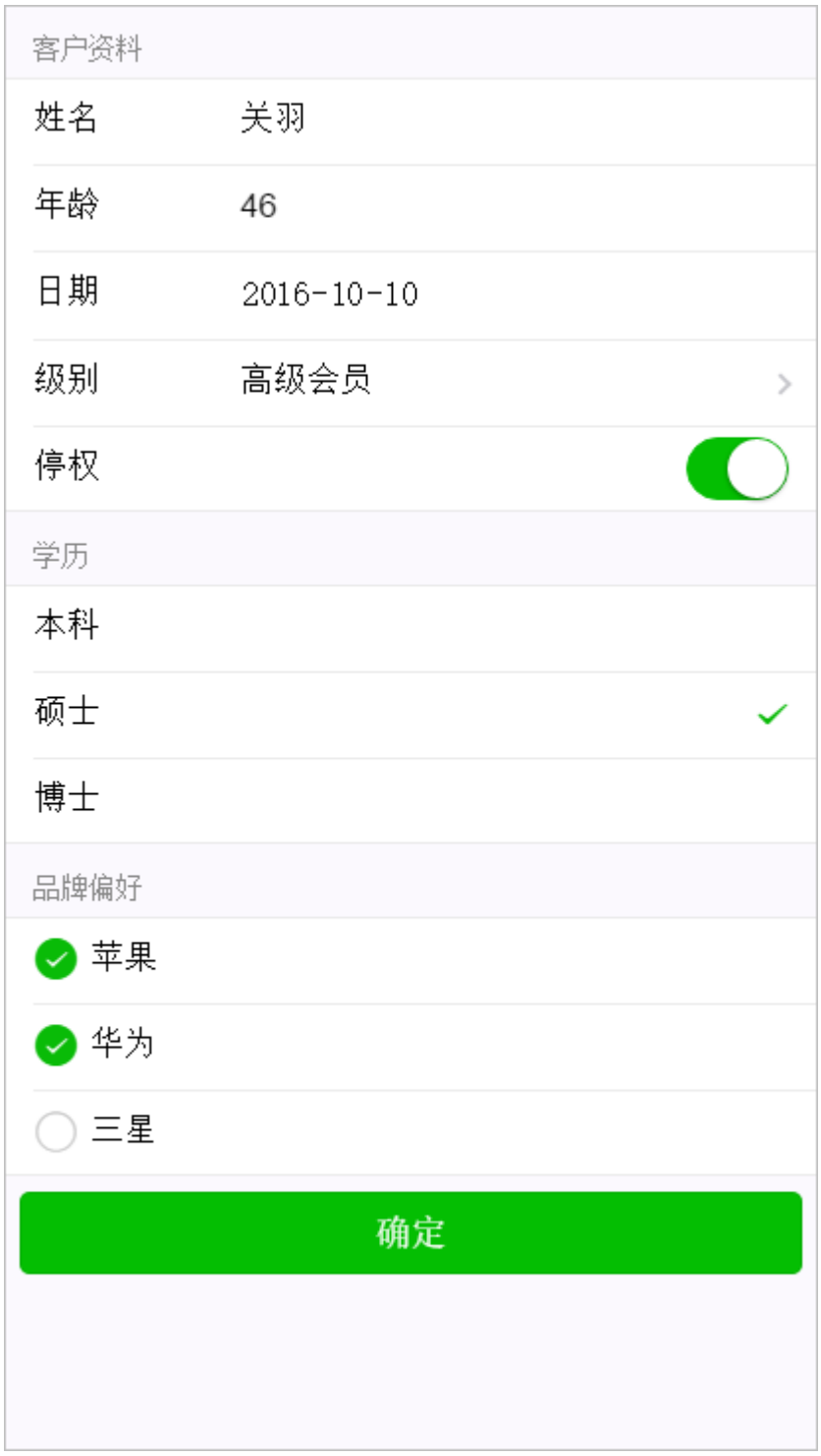


客户资料

姓名	舒淇
地点	蒙古草原
日期	2012-10-12




基本资料
姓名
年龄
卡号
请输入卡号
密码
请输入密码
手机
获取验证码
验证码
+86
请输入联系电话

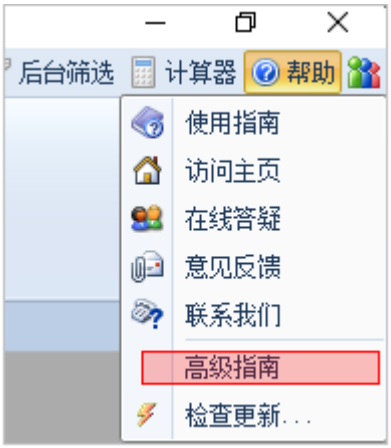

客户资料
姓名
关羽
年龄
46
日期
2016-10-10
级别
高级会员
停权
学历
本科
硕士
博士
品牌偏好
苹果
华为
三星
确定


编辑订单
编号
161206005
客户
柳传志
日期
2017-02-12
总数量
23
总金额
111024
产品
数量
单价
金额
Foxtable 开发版
21
4688
98448
iphone 6 plus
2
6288
12576
增加明细
保存
返回
生成Excel格式订单



Foxtable 的《使用指南》和《开发指南》并没有涉及到移动开发的内容。

移动开发的说明在《高级开发指南》中，启动 Foxtable，单击右上角的帮助按钮，再单击"高级指南"命令，可以打开《高级开发指南》：



8. 外接设备

8.1 短信收发

8.1.1 关于配套的短信猫

最新版本的 Foxtable，已经可以和微信、网页、APP 通讯，所以使用短信收发数据其实有些过时了。
Foxtable 集成了短信收发和来电显示的功能，有了短信收发功能，可以很方便地通过手机收集和查询数据，也可以通过手机来审批数据。
短信功能由金笛公司提供，需要从金迪公司购买配套的短信猫才能正常使用，必须使用 G 网的 SIM 卡，移动和联通均可，不支持 3G 制式的 SIM 卡，也不支持电信的 SIM 卡。
金笛公司网址:
http://www.sendsms.cn/
需要说明的是：

1、金笛公司和我们没有任何关系，我们只是从众多短信方案提供商中选择了一个较为有实力的。
2、试用版不具备此功能。

8.1.2 注册短信猫组件

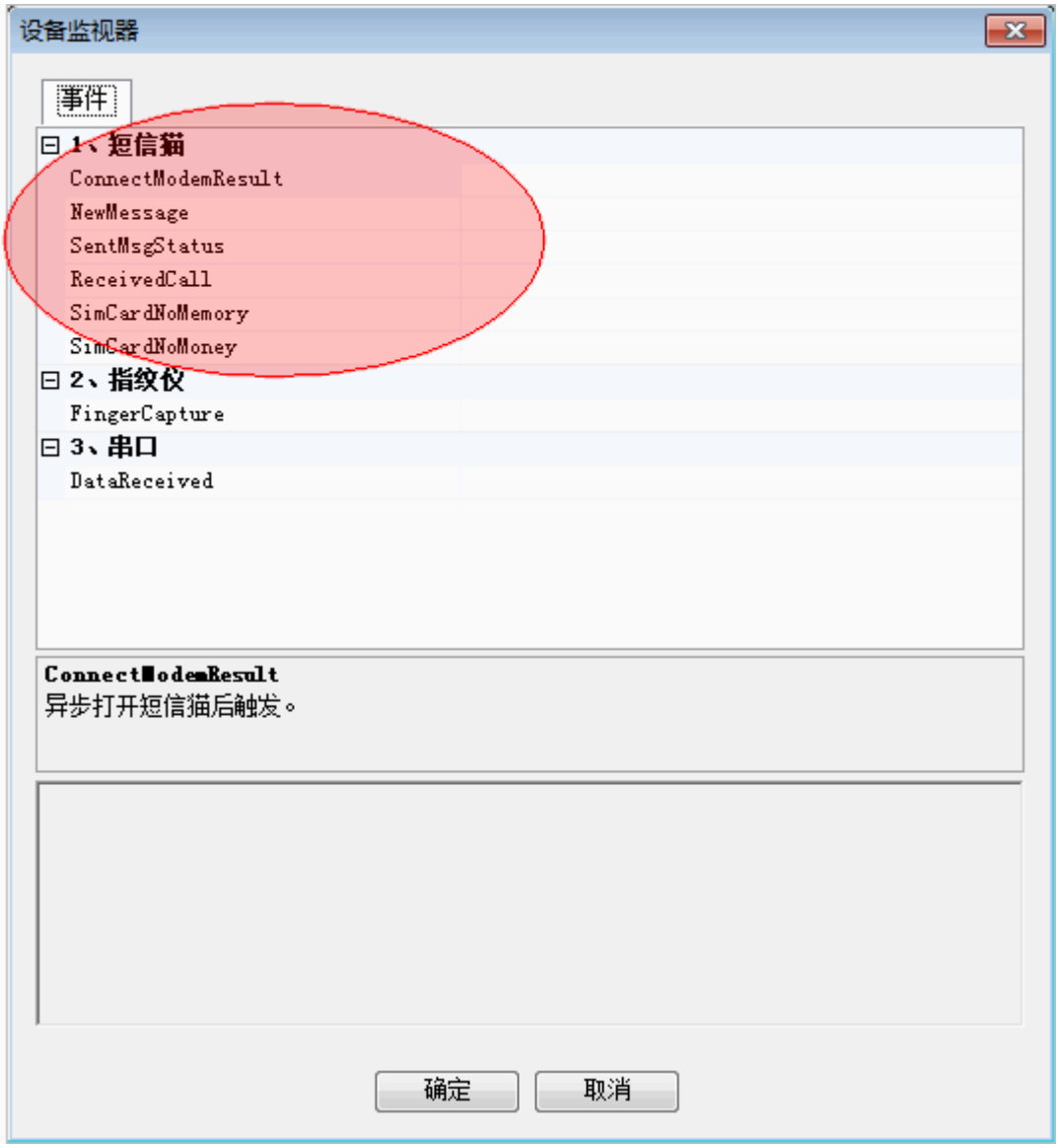
在使用短信猫收发短信之前，首先要注册短信猫组件，Foxtable 安装目录下有个 SMS 子目录，运行其中的文件 regsms.bat，即可注册短信猫组件。
再次提示：最新版本的 Foxtable，已经可以和微信、网页、APP 通讯，所以使用短信收发数据其实有些过时了。

8.1.3 设置短信猫事件

Foxtable 提供了一系列事件，用于监视短信猫。
在管理项目功能区，单击“设备监视器”按钮：



可以设置短信监视事件：



后面的章节，会有各个事件的具体介绍。

8.1.4 确定端口号

短信猫有串口和 USB 口两种接口类型，大家可以根据需要采购。

如果使用的是 USB 口的短信猫，需要安装驱动程序，安装驱动后，在设备管理器中可以看到下图所示的设备：



请记住上图中的 USB 串口的端口号，不同的电脑会有所不同，上图是 COM3，也就是端口号为 3，我们连接短信猫的时候，需要指定正确的端口号，否则会出错甚至导致程序崩溃。
同样，如果使用的是串口短信猫，也必须明确短信猫所接入串口的编号。

8.1.5 打开和关闭短信猫

Foxtable 提供了一个名为 FoxSMS 的系统变量，表示连接到计算机的短信猫。

异步方式打开短信猫

短信猫默认的工作方式是异步，也就是打开短信猫的过程中，你可以继续其他操作，短信猫连接结束后，会触发 ConnectModemResult 事件。
示例：
首先在 ConnectModemResult 事件设置代码：

```
If e.Succeed = True Then
    MessageBox.Show("短信猫连接成功!","提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("短信猫连接失败!","提示", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
```

然后将某个按钮的 Click 事件代码设置为：

```
FoxSms.SyncWorkMode = False '设置为异步工作模式
FoxSMS.CommPort = 3 '注意端口号千万不能搞错
FoxSMS.OpenCom()
```

这样单击这个按钮，即可打开短信猫，打开结束后，触发 ConnectModemResult 事件，通知用户短信猫是否成功打开。
提示：端口号千万不能搞错，否则可能导致程序崩溃，参考确定端口号

同步方式打开短信猫

你也可以用同步方式打开短信猫，打开短信猫的过程中，不能进行任何操作，直到打开操作结束。

示例：

```
FoxSms.SyncWorkMode = True '设置为同步工作模式
FoxSMS.CommPort = 3 '注意端口号千万不能搞错
If FoxSMS.OpenCom() = 0 Then
    MessageBox.Show("短信猫连接成功!","提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("短信猫连接失败!","提示", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
```

由于打开短信猫需要一定的时间，所以我们建议你使用异步方式打开。

关闭短信猫

关闭短信猫很简单，只需执行：

```
FoxSMS.CloseCom()
```

8.1.6 判断短信猫状态

FoxSMS 有一个 Ready 属性，短信猫打开的时候返回 True，关闭的时候返回 False，例如：

```
If FoxSMS.Ready = False Then
    MessageBox.Show("请先连接短信猫!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

8.1.7 设置短信中心号码

大多数情况下，我们并不需要设置短信中心号码，打开短信猫之后，FoxSMS 会自动从 SIM 卡获取短信中心号码。

在实际测试过程中，我们发现自动获得的短信中心号码，偶尔会有不正确的情况，导致无法正常发送短信，需要人为设置短信中心号码。

FoxSMS 有一个名为“CenterNo”的属性，用于返回或设置短信中心号码，例如：

```
FoxSMs.CenterNo = "13800100500"
```

如果要设置短信中心号码，应该在打开短信猫之后设置。

不同的运营商，不同的地区，短信中心号码都是不同的，请向当地运营商咨询。

8.1.8 发送短信

异步发送短信

异步发送短信首先将信息提交到短信猫，发送结束后触发 SentMsgStatus，整个发送过程不会影响系统的正常运行。

示例

```
If FoxSMS.Ready = False Then
        MessageBox.Show("请先连接短信猫!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
FoxSms.SyncWorkMode = False '设置为异步工作模式
FoxSMS.SendMsg("13800138000","异步发送短信测试")
```

如果你要跟踪短信是否发送成功，并将发送成功的信息记录在收件箱表， 可以先新建一个名为收件箱的表，包括三列，分别为号码、内容、时间。

然后在 SentMsgStatus 事件设置代码：

```
If e.Succeed Then '如果发送成功
    Dim r As Row = Tables("发件箱").AddNew()
    r("号码") = e.DestNumber
    r("内容") = e.Content
    r("时间") = Date.Now()
    r.Save()
Else
    MessageBox.Show("向" & e.DestNumber & "发送信息失败!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

同步发送短信

以同步方式发送短信，在发送短信的过程中不能进行任何操作，直到短信发送结束。

示例

```
If FoxSMS.Ready = False Then
        MessageBox.Show("请先连接短信猫!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
FoxSms.SyncWorkMode = True '设置为同步工作模式
If FoxSMS.SendMsg("13800138000","异步发送短信测试") = 0 Then
      MessageBox.Show("发送信息成功!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
      MessageBox.Show("发送信息失败!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

由于发送一条短信大概需要 4 秒左右，所以建议采用异步发送方式，特别是需要发送多条短信的时候。

同时向多个号码发送短信

不管是异步方式还是同步方式，都可以同时向多个号码发送短信，不同的号码之前用分号隔开:

示例

```
If FoxSMS.Ready = False Then
        MessageBox.Show("请先连接短信猫!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

```
FoxSms.SyncWorkMode = True '设置为同步工作模式
If FoxSMS.SendMsg("13800138000;15876377122;13232670670","测试短信内容") = 0 Then
    MessageBox.Show("发送信息成功!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
Else
    MessageBox.Show("发送信息失败!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
End If
```

8.1.9 读取短信

自动读取新短信

首先新建一个收件箱表，包括四列，分别是：发送时间、接收时间、号码、内容

然后将 NewMessage 事件代码设置为：

```
Dim r As Row = Tables("收件箱").AddNew()
r("号码") = e.FromNumber
r("内容") = e.Content
r("接收时间") = Date.Now
r("发送时间") = e.SentTime
r.Save()
FoxSms.DelMsgForIndex(e.Index) '从 SIM 卡删除此短信
```

现在打开短信猫，短信猫即可自动将收到的新短信登记到收件箱表 ，并从 SIM 卡删除此短信。

读取现有短信

如果要读取 SIM 卡中的现有短信，可以使用下面的代码：

```
If FoxSMS.Ready = False Then
    MessageBox.show("请先连接短信猫!", "提示", MessageBoxButtons.OK, MessageBoxIcon.Information)
    Return
End If
Dim cnt As Integer = FoxSMS.SimMsgCount
If cnt > 0 Then '如果 SIM 卡有短信
    Dim sendtime As String
    Dim strFrom As String
    Dim strContent As String
    Dim result As Integer
    FoxSMS.AutoDelMsg = True '自动从 SIM 卡删除已经读取的短信
    Tables("收件箱").StopRedraw()
    For Index As Integer = 1 To FoxSMS.SimMsgCapacity
        result = FoxSMS.ReadMsgForIndex(Index, strFrom, strContent, sendtime)
        If result = 0 Then
            Dim dr As Row = Tables("收件箱").AddNew()
            dr("号码") = strFrom
            dr("内容") = strContent
```

```
            dr("发送时间") = SendTime
            dr("接收时间") = Date.Now
        End If
    Next
    Tables("收件箱").DataTable.Save()
    Tables("收件箱").ResumeRedraw()
End If
```

8.1.10 记录来电号码

如果要记录来电号码，可以在 ReceivedCall 事件中设置代码：

```
Dim r As Row = Tables("来电记录").AddNew()
r("号码") = e.FromNumber
r("时间") = Date.Now
r.Save()
```

一般的短信猫并不具备接听电话的功能，大家可以设置呼叫转移，这样短信猫负责收发短信，其他电话负责接听来电。

提示：

1、如果要记录来电号码，不能使用无条件呼叫转移，应该使用无应答呼叫转移，否则短信猫无法触发 ReceivedCall 事件。

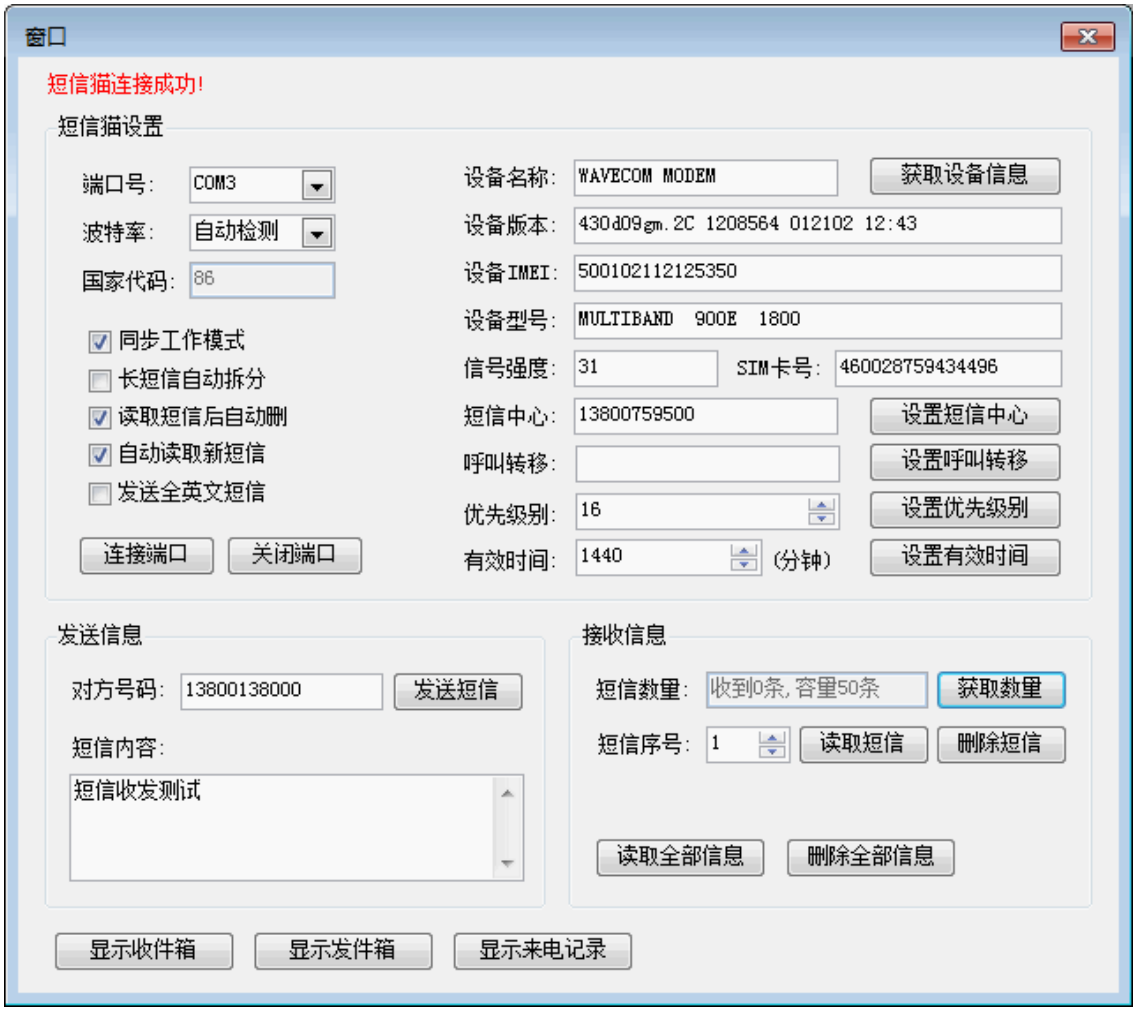
2、设置无应答呼叫转移的时候，延迟时间最好设置为 5 秒甚至更短，以免来电者等待时间过长。

2、FoxSMS 提供了设置呼叫转移号码的功能，但是设置的是无条件转移，建议大家用手机设置好呼叫转移，然后再将 SIM 卡插入短信猫。

3、如果不需要记录来电号码，请设置为无条件呼叫转移。

8.1.11 一个完整的例子

为了让大家更好地掌握短信猫的使用，Foxtable 提供了一个完整的例子，位于 CaseStudy 目录，文件名是"短信收发.Table"。



参考这个例子的时候，注意除了窗口有事件代码，请记得打开短信监视器，参考相关短信监视事件的代码。



8.1.12 应用实例

8.1.12.1 用短信收集数据

假定有一个订单表，允许业务员通过手机短信来提交订单。

设计步骤：

1、首先我们要约定提交信息的格式：@@客户@产品@数量@单价。例如收到短信“@@联想公司@PD01@800@128.5”，表示联想公司订购产品 PD01，数量是 800 台，单价是 128.5。

2、在项目事件 AfterOpenProject 设置代码：

```
FoxSms.SyncWorkMode = False '设置为异步工作模式
```

```
FoxSMS.CommPort = 3 '注意端口号千万不能搞错
FoxSMS.OpenCom()
```

3、在项目事件 BeforeCloseProject 事件中设置代码：

```
FoxSMS.CloseCom()
```

4、然后在短信监视器中，设置 ConnectModemResult 事件的代码为：

```
If e.Succeed = False Then
    MessageBox.Show("短信猫连接失败!","提示", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
```

5、然后在短信监视器中，设置 NewMessage 事件的代码为：

```
If e.Content.StartsWith("@@") Then '如果信息是以约定的@@开始
    Dim vals() As String = e.Content.Trim("@").Split("@")
    Dim ret As String
    If vals.Length = 4 Then '如果数组长度等于 4
        If vals(0) > "" AndAlso vals(1) > "" Then '且提交了客户名称和产品名称
            Dim Quantity As Integer
            Dim Price As Double
            If Integer.TryParse(Vals(2), Quantity) '如果数量格式正确
                If Double.TryParse(Vals(3), Price) Then '如果单价格式正确
                    Dim dr As DataRow = DataTables("订单").AddNew()
                    dr("客户") = vals(0)
                    dr("产品") = vals(1)
                    dr("数量") = vals(2)
                    dr("单价") = vals(3)
                    ret = "提交成功:" & e.Content
                End If
            End If
        End If
    End If
    If ret = "" Then
        ret = "格式错误:" & e.Content
    End If
    FoxSms.DelMsgForIndex(e.Index) '从 SIM 卡删除此短信
    FoxSMS.SendMsg(e.FromNumber,ret) '回复短信,告诉业务员,此订单是否提交成功
End If
```

8.1.12.2 用短信查询数据

假定有个仓管系统，允许用户通过手机短信查询某产品的库存的单价。

设计步骤

1、首先我们得规定提交信息的格式，假定我们规定的格式如下：

格式	说明
**产品 1*产品 2...	指定要查询的产品，不同的产品之间，用符号*隔开，示例： **PD01　表示查询 PD01。

	**PD01*PD02　表示查询 PD01 和 PD02。

2、在项目事件 AfterOpenProject 设置代码：

```
FoxSms.SyncWorkMode = False '设置为异步工作模式
FoxSMS.CommPort = 3 '注意端口号千万不能搞错
FoxSMS.OpenCom()
```

3、在项目事件 BeforeCloseProject 事件中设置代码：

```
FoxSMS.CloseCom()
```

4、然后在短信监视器中，设置 ConnectModemResult 事件的代码为：

```
If e.Succeed = False Then
    MessageBox.Show("短信猫连接失败!","提示", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
```

5、然后在短信监视器中，设置 NewMessage 事件的代码为：

```
If e.Content.StartsWith("**") Then '如果信息是以约定的**开始
    Dim vals() As String = e.Content.Trim("*").Split("*")
    Dim ret As String
    Dim flt As String
    For Each val As String In vals
        If flt > "" Then
            flt = flt & ","
        End If
        flt = flt & "'" & val & "'"
    Next
    If flt > "" Then
        Dim drs As List(Of DataRow) = DataTables("库存").Select("产品  In (" & flt & ")")
        For Each dr As DataRow In drs
            If ret > "" Then
                ret = ret & vbcrlf
            End If
            ret = ret & "产品:" & dr("产品") & "  库存:" & dr("数量") & "  单价:" & dr("单价")
        Next
        If ret = "" Then
            ret = "无此产品:" & e.Content
        End If
    Else
        ret = "格式错误:" & e.Content
    End If
    FoxSms.DelMsgForIndex(e.Index) '从 SIM 卡删除此短信
    FoxSMS.SendMsg(e.FromNumber,ret)
End If
```

8.1.12.3 用短信审批数据

假定有一个订单表，允许领导通过短信审批订单。

设计步骤：

1、首先我们要约定提交待审批订单、订单通过审核，订单未通过审核的格式：

格式	说明
##订单编号#订单内容	待审批订单，例如： ##0124#联想公司,产品 PD01,单价 125.8,数量 800，表示待审核的订单编号是 0124，后面是订单内容。
##订单编号#审批内容	表示订单通过审核，例如： ##0124，表示编号为 0124 的订单通过审核。 ##0124#注意跟进，表示编号为 0124 的订单通过审核，领导批示“注意跟进”。
??订单编号?审批内容	表示订单未通过审核，例如： ??0124，表示编号为 0124 的订单未通过审核， ??0124?单价过低，表示编号为 0124 的订单未通过审核，领导批示“单价过低”。

2、在项目事件 AfterOpenProject 设置代码：

```
FoxSms.SyncWorkMode = False '设置为异步工作模式
FoxSMS.CommPort = 3 '注意端口号千万不能搞错
FoxSMS.OpenCom()
```

3、在项目事件 BeforeCloseProject 事件中设置代码：

```
FoxSMS.CloseCom()
```

4、然后在短信监视器中，设置 ConnectModemResult 事件的代码为：

```
If e.Succeed = False Then
    MessageBox.Show("短信猫连接失败!","提示", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
```

5、在菜单或者窗口中增加一个“提交审核”按钮，将其代码设置为：

```
Dim msg As String
Dim r As Row = Tables("订单").Current
msg = "##" & r("编号") & "#" & r("客户") & ",产品:" & r("产品") & ",数量:" & r("数量") & ",单价:"
& r("单价")
FoxSMS.SendMsg("13800138000",msg) '请改为领导的手机号码
```

6、然后在短信监视器中，设置 NewMessage 事件的代码为：

```
Dim msg As String = e.Content
If msg.StartsWith("##") OrElse msg.StartsWith("??") Then
    msg = msg.Replace("?","#")
    Dim vals() As String = msg.Trim("#").Split("#")
    Dim dr As DataRow = DataTables("订单").Find("编号 = '" & vals(0) & "'")
    If dr IsNot Nothing Then
        dr("审核") = e.Content.StartsWith("##") '判断是否通过审批
        If vals.Length > 1 AndAlso vals(1)> "" Then '如果回复的短信包括审批内容
            If dr.IsNull("备注") = False Then
                dr("备注") = dr("备注") & vbcrlf
            End If
```

```
            dr("备注") = dr("备注") & vals(1) '在备注中记录审批内容
        End If
    Else
        FoxSMS.SendMsg(e.FromNumber,"订单编号错误:" & vals(0))
    End If
    FoxSms.DelMsgForIndex(e.Index) '从 SIM 卡删除此短信
End If
```

8.1.12.4 通过网页显示查询结果

通过短信查询数据，如果查询结果内容很多，例如要查询所有产品的库存，而产品的数量数十个甚至数百个，显然这时直接用短信回复是不合适的。

我们可以将结果保存为网页，然后将链接地址通过短信发给查询者，前提是你已经搭建好 web 服务器，而且 web 服务器已经安装了 Foxtable，并接上了短信猫；如果条件不许可，你也可以在其他电脑接收短信，生成查询结果网页文件，并 自动上传到 web 服务器，最后将连接地址发给查询者。

提示： 下面的示例用到了专业报表的知识。

示例

下载我们重新设计之前的库存查询示例，如果用户指定产品，以短信的形式返回这些产品的库存和单价，如果用户没有指定产品，则以网页的形式返回所有产品的库存和单价。

1、假定 web 服务器已经安装了 foxtable，并接上了短信猫，web 目录的是：c:\myweb。

2、在 c:\myweb 目录下建立一个子目录"query",用于存放以网页形式生成的查询结果。

3、规定提交查询信息的格式，假定我们规定的格式如下：

格式	说明
**产品 1*产品 2...	指定要查询的产品，不同的产品之间，用符号*隔开，如果省略产品，表示查询全部产品： 示例： ** 表示查询所有产品 **PD01 表示查询 PD01。 **PD01*PD02 表示查询 PD01 和 PD02。

2、在项目事件 AfterOpenProject 设置代码：

```
FoxSms.SyncWorkMode = False '设置为异步工作模式
FoxSMS.CommPort = 3 '注意端口号千万不能搞错
FoxSMS.OpenCom()
```

3、在项目事件 BeforeCloseProject 事件中设置代码：

```
FoxSMS.CloseCom()
```

4、然后在短信监视器中，设置 ConnectModemResult 事件的代码为：

```
If e.Succeed = False Then
    MessageBox.Show("短信猫连接失败!","提示", MessageBoxButtons.OK, MessageBoxIcon.Warning)
End If
```

5、然后在短信监视器中，设置 NewMessage 事件的代码为：

```
Dim ret As String
If e.Content = "**" '如果查询所有产品的库存
```

```
    Dim doc As New PrintDoc '定义一个新报表
    Dim rt As New prt.RenderTable '定义一个新表格
    Dim fl As String = "c:\myweb\query\" & rand.NextString(8) & ".htm" '生成一个随机的网页文件名
    Dim ColNames As String() = New String(){"产品", "数量","单价"}
    For c As Integer = 0 To ColNames.Length - 1 '逐列设置和填入内容
        rt.Cells(0,c).Text = ColNames(c) '列名作为标题
        For r As Integer = 0 To DataTables("库存").DataRows.Count -1 '开始填入该列内容
            rt.Cells(r + 1, c).Text = DataTables("库存").DataRows(r)(ColNames(c))
        Next
    Next
    rt.Style.Gridlines.All = New prt.Linedef(Color.Gray)
    doc.Body.Children.Add(rt) '将表格加入到报表
    doc.SaveHtm(fl) '保存为网页文件
    ret = "http://www.你的域名.com/query/" & FileSys.GetName(fl)
ElseIf e.Content.StartsWith("**") Then '如果查询指定产品的库存
    Dim vals() As String = e.Content.Trim("*").Split("*")
    Dim flt As String
    For Each val As String In vals
        If flt > "" Then
            flt = flt & ","
        End If
        flt = flt & "'" & val & "'"
    Next
    If flt > "" Then
        Dim drs As List(Of DataRow) = DataTables("库存").Select("产品  In (" & flt & ")")
        For Each dr As DataRow In drs
            If ret > "" Then
                ret = ret & vbcrlf
            End If
            ret = ret & "产品:" & dr("产品") & "  库存:" & dr("数量") & "  单价:" & dr("单价")
        Next
        If ret = "" Then
            ret = "无此产品:" & e.Content
        End If
    End If
End If
FoxSms.DelMsgForIndex(e.Index) '从 SIM 卡删除此短信
If ret > "" Then
    FoxSMS.SendMsg(e.FromNumber,ret)
End If
```

8.1.13 编程参考

8.1.13.1 属性

名称	数据类型	说明
AutoDelMsg	Boolean	设置调用读取短信方法后是否要自动从 SIM 卡中删除对应的短信。true 自动删除 false 不自动删除，默认为 false，运行时可读写。
CenterNo	String	SIM 入网地的短信中心号码(SMCA)，自动从 SIM 卡中读取，运行时可读写。
CommPort	Integer	设备与计算机连接的串口号，默认值为 1，运行时可读写。
Ready	Boolean	设备是否成功打开，运行时只读，True 已打开，false 未打开。
BaudRate	Long	设备与计算机的串口通讯的波特率，运行时可读写，默认值为 0，自动判断波特率。
MsgID	Long	提交短信后生成的编号，运行时可读写，此 ID 主要用于群发和异步发送时状态通知标识，在一定时间段内需保证唯一。
SyncWorkMode	Boolean	是否同步工作方式，打开设备前设置对打开设备有效，发送短信前设置对发送短信有效，运行时可读写，同步方式时，直到打开设备成功或发送成功才返回。
SignName	String	发送短信自动添加的签名，运行时可读写，发送时自动在发送内容前面添加，格式：[签名]:+内容。设置后对后续的所有发送请求有效直到被改变。
Timeouts	Integer	AT 指令执行超时时间，缺省 15 秒，运行时可读写，设置后对后续操作生效。
AutoSplite	Boolean	是否强制拆分长短信发送，运行时可读写，GSM 短信猫可支持长短信发送，True 表示强制。设置后对后续的所有发送请求有效直到被改变。
CountryCode	String	国际代码，缺省 86 表示中国，如果设为空，发送短信目标号码都需要添加国际代码，短信中心号码设置也需要添加国际代码，运行时可读写。
SendPriority	Integer	发送优先级，1-32，缺省 16，数越大，优先级越高。如果在异步工作模式同时提交很多短信，优先级高的优先发送，设置后对后续的所有发送请求有效直到被改变，运行时可读写。
EnglishMsg	Boolean	是否请求发送全英文短信，缺省 False，如果设为 True，包括签名和内容，都需要是英文内容，否则对方接收后是乱码，设置后对后续的所有发送请求有效直到被改变，运行时可读写。
MsgValidMinute	Integer	发送短信的有效期，分钟为单位，缺省一天 1440，主要用于对方如果没有开机，在有效期内开机，对方都可以收到短信。设置后对后

		续的所有发送请求有效直到被改变，运行时可读写。
Signal	Integer	设备信号强度，运行时只读，连接设备后可获取，一般 15-25 算正常，信号过弱可能导致发送短信速度慢，失败率也高。
ModemName	String	设备名称，运行时只读，连接设备后可获取。
ModemModel	String	设备型号，运行时只读，连接设备后可获取。
ModemIMEI	String	设备 IMEI 唯一标识，个别设备是硬盘 ID 或 MAC 地址，运行时只读，连接设备后可获取。
SimCardID	String	设备上 SIM 卡的卡号，主意不是手机号，运行时只读，连接设备后可获取。
CallTransfer	String	呼叫转移号码，连接设备后可读写，只能写，设为空表示取消呼叫转移。
WaitSend	Integer	当前设备等待发送的短信数量，运行时只读，连接设备后可获取。
SimMsgCount	Integer	返回 SIM 卡中已经接收的短信数量。
SimMsgcapacity	Integer	返回 SIM 卡的短信容量。
FirewareVerion	String	Fireware Verion，运行时只读，连接设备后可获取。
ErrInfo	String	最近一次操作失败或错误时的描述。

8.1.13.2 方法

- ATCommand

  向端口发送 AT 指令，语法：

  ATCommand(ATCmd)

  ATCmd: 字符串,要发送的 AT 指令

  返回一个字符表，表示 AT 指令的执行结果。

- CloseCom

  关闭连接设备的通讯端口，无参数。
  返回 0 表示关闭成功。

- DelMsgForIndex

  删除 SIM 卡中指定位置的短信，语法：

  DelMsgForIndex(Index)

Index: 整数，要删除的短信的位置

返回 0 表示删除成功。

- OpenCom

  打开连接设备的通讯端口，打开前请先设置端口，无参数。
  返回 0 表示成功。

- ReadMsgForIndex

  读取 SIM 卡中指定位置的短消息，语法:

  ReadMsgForIndex(Index, strFrom, strContent, sendTime)

  Index: 整数型，表示要读取的短信的位置。
  strFrom: 字符串，用于存储发送短信的号码。
  strContent：字符串，用于存储短信内容。
  SendTime：日期型，用于存储短信发送时间。

  返回 0 表示读取成功。

- SendMsg

  发送一条短信息，语法：

  SendMsg(PhoneNo,Content)

  PhoneNo: 字符串，对方的手机号码
  Content: 字符串，短消息内容.

  返回 0 表示发送成功。

- ClearSimMsg

  清除 SIM 卡上所有短信，无参数。

8.1.13.3 事件

8.1.13.3.1 ConnectModemResult

异步连接短信猫后触发。

e 参数：
Succeed：逻辑型，返回 True 表示连接成功。

8.1.13.3.2 SentMsgStatus

异步发送短信结束后执行。
e 参数属性：
DestNumber：字符型，目标电话号码。
Content： 字符型，短信内容。
SplitIndex：整数型，长短信拆封发送后，通过此参数返回拆分序号。
Succeed： 逻辑型，返回 True 表示发送成功。

8.1.13.3.3 NewMessage

收到新短信后触发。
e 参数：
Index： 整数性，此短信在 SIM 卡中的位置，之后要通过这个位置，才能读取和删除此短信。
Content： 字符型，短信内容。
FromNumber：字符型：发送短信的号码。
SentTime： 日期型，发送短信的时间。

8.1.13.3.4 ReceivedCall

收到新来电后触发。
e 参数属性：
FromNumber：字符型，返回来电号码。

8.1.13.3.5 SimCardNoMemory

SIM 卡短信已满后触发，无 e 参数。

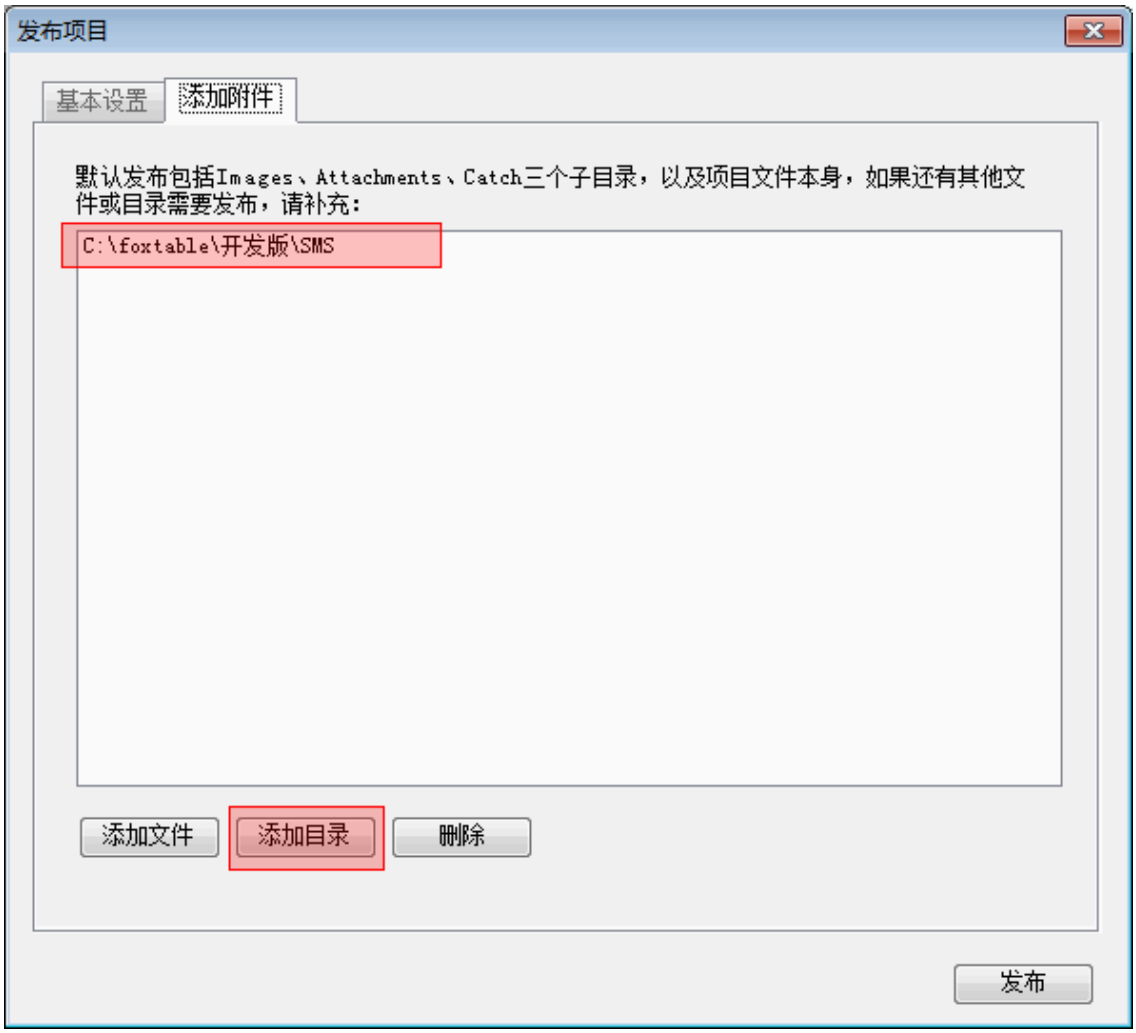
8.1.13.3.6 SimCardNoMoney

SIM 余额不足时触发，无 e 参数。

8.1.14 短信与发布

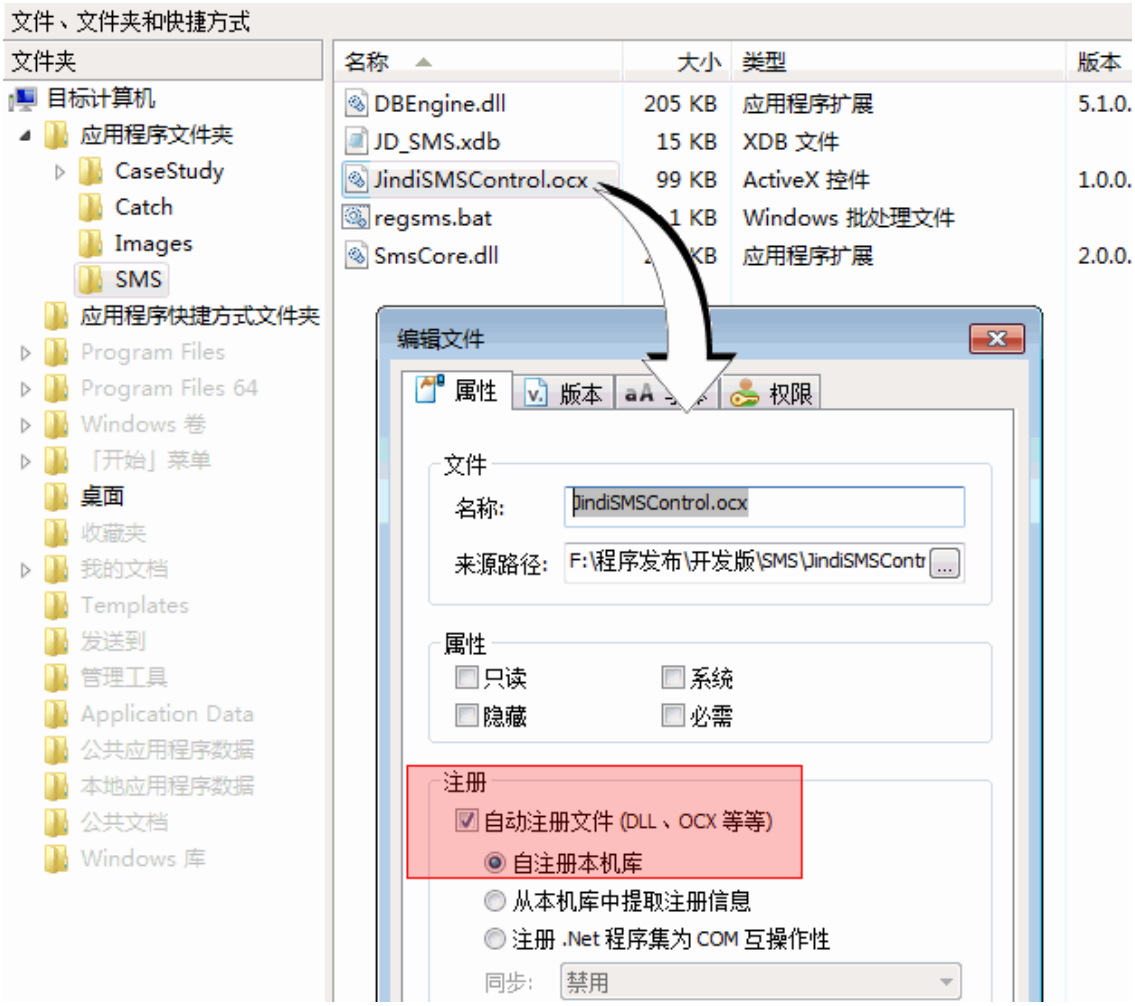
如果你使用的不是开发版，或者还未接触过程序发布，可以忽略本节的内容。
由于大部分管理系统，并不需要短信收发功能，所以默认情况下，发布的项目，并不会包括短信收发模块的文件。

短信收发模块的文件，位于 Foxtable 安装目录的 SMS 目录下，发布项目的时候，可以在"添加附件"页，单击"添加目录"按钮，将 SMS 目录添加进来。



如果你不想制作安装程序，想直接复制到其它电脑运行，那么复制到其他电脑后，要执行一次 SMS 文件夹下批处理文件 regsms.bat，才能正常收发短信。

如果要制作安装程序，在应用程序文件夹中，选择 SMS 文件夹，右击文件 JindiSMSControl.ocx，从快捷菜单中执行“属性”，在属性设置中选中选项“自动注册文件”：



这样在其他电脑安装后，才能正常收发短信。

此外，Publish 下新增加了一个名为“Interop.JinDI_SmsComLib.dll”的文件，对于以前的安装项目，记得将此文件添加到应用程序文件夹。

8.1.15 另一种发短信的方式

本短信示例使用吉信通公司提供的短信平台，你需要在该公司主页注册账号并充值，才可通过下面的代码联网发送短信。

吉信通公司的网址是：http://www.winic.org/

需要注意的是，吉信通公司和我们没有任何关系，提供类似短信平台的企业有很多，我们只是从中选择了一家作为示例而已。

示例代码：

```
Dim url As String
Dim http As Object
url = "http://service.winic.org:8009/sys_port/gateway/?id=您的账号&pwd=您的密码&to=接收短信手机号码&content=短信内容&time="
http = CreateObject("MSXML2.XMLHTTP")
http.Open("get", url, False)
http.send()
MessageBox.Show("返回信息:" & http.responseText)
```

可以设置多个接收短息手机号码，不同的号码之间用逗号（,）隔开。
返回信息的格式为： 状态码 / 发送条数 / 当次消费金额 / 总体余额 / 短信编号
返回状态码说明：

000 成送成功！
-01 当前账号余额不足！
-02 当前用户 ID 错误！
-03 当前密码错误！
-04 参数不够或参数内容的类型错误！
-05 手机号码格式不对！
-06 短信内容编码不对！
-07 短信内容含有敏感字符！
-8 无接收数据
-09 系统维护中..
-10 手机号码数量超长！（100 个/次 超 100 个请自行做循环）
-11 短信内容超长！（70 个字符）
-12 其它错误！
假定返回信息为:
000/Send:1/Consumption:.1/Tmoney:4.2/sid:0326121430883747
表示发送成功，本次发送量 1 条短信，消费了 0.1 元，还剩余额 4.2 元。
这种方式的好处是不需要短信猫，但是只能发，不能收。

8.2 指纹识别

8.2.1 关于指纹识别

Foxtable 内置指纹识别功能。
指纹识别可以更安全、更快捷地识别身份、保护数据。
例如登录系统的时候，只需按一下指纹，就能自动识别此用户的身份并进入系统，无需输入用户名和密码。
例如领导审批的时候，只需按一下指纹，就能自动识别按此指纹是否为领导的指纹，如果是的，则审批通过。
如果采用传统的方式审批，首先需要以领导名义登录系统，然后单击“审批”按钮审批，不仅繁琐，也不安全，例如领导离开之后，如果忘记关闭系统，其他人就可以冒名审批。
用户需要自行采购指纹仪，支持的型号有三种：
U.are.U 4000B
U.are.U 4500
U.are.U 5000
在将指纹仪连接到电脑之前，请先安装指纹仪驱动程序，下载地址：
http://www.foxtable.com/download/foxfinger.rar
用户采购的指纹仪需要得到官方的授权许可，输入官方提供的许可码，才能接入 Foxtable，这个目前是收费的。

提示：试用版不具备指纹识别功能。

8.2.2 FOXFinger 参考

Foxtable 提供了一个名为 FoxFinger 的系统变量，表示连接到计算机的指纹采集器。

方法

- Start
  启动指纹采集器

- Stop
  关闭指纹采集器

- Enroll
  登记指纹，如果登记成功，返回指纹生成的指纹模板。

- Verify
  比对指纹，语法：
  Verify(Template,Features)

  Template: 字符型，用于比对的指纹模板，事先通过 Enroll 方法采集。
  Features: 字符型，用于比对的指纹特征，在 FingerCapture 事件捕获。
  如果指纹特征和指纹模板匹配，则返回 True。

属性

- Ready
  逻辑型，判断指纹采集器是否已经连接并启动。

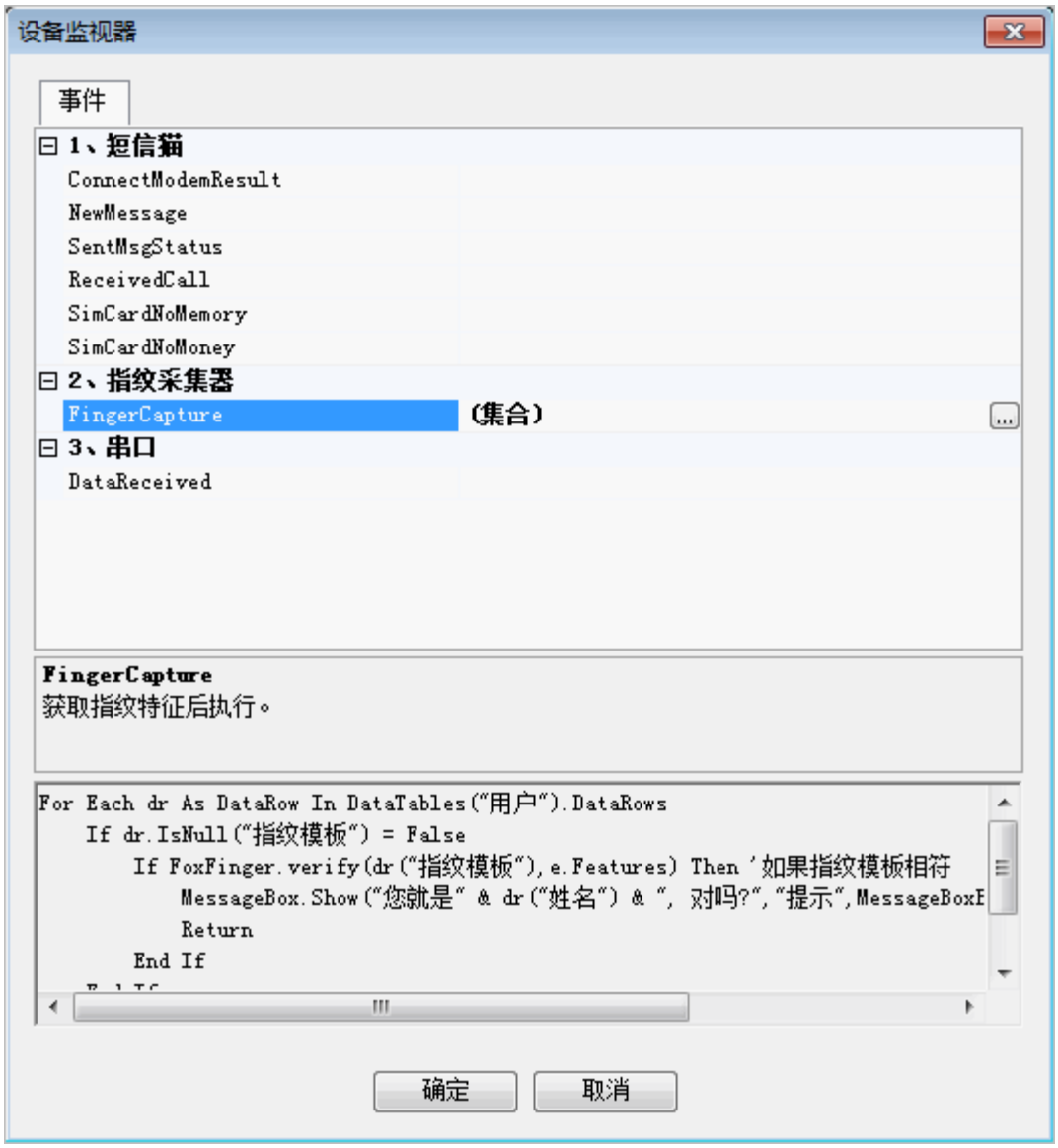
- SerialNumber
  字符型，返回指纹采集器的物理序列号，这个序列号是唯一的，每个指纹采集器均不相同，且不可改变。

8.2.3 设置指纹事件代码

在管理项目功能区，单击“设备监视器”按钮：



可以设置指纹采集器的代码：



指纹采集器就一个事件：

FingerCapture

在捕获用户指纹特征之后触发。

必须先启动指纹仪，才会触发此事件。

e 参数属性：

Features：字符型，返回捕获的指纹特征，用于和指纹模板进行比对。

Image： Image 类型，返回捕获的指纹图片。

e 参数方法：

SaveToFile

将捕获的指纹图片保存为一个文件，例如：

e.SaveToFile("c\data\finger.jpg")

8.2.4 基本概念与步骤

指纹模板与登记

要进行指纹识别，首先必须登记指纹。

所谓登记指纹，就是连续多次采集同一手指的指纹特征，然后对这些指纹特征进行综合分析，生成一个指纹模板，保存在数据库中。
假定有个用户表，有姓名和指纹模板两列，希望单击某个按钮，能够启动指纹登记窗口，指纹登记成功后，将生成的指纹模板保存到用户表当前行的指纹模板列。
为此，只需将此按钮的代码设置为：

```
Dim fg As String = FoxFinger.EnRoll() '登记指纹
If fg > ""  Then '如果登记成功
    Tables("用户").Current("指纹模板") = fg '保存获得的指纹模板
End If
```

下图是指纹登记窗口，需要连续取得四次合格的指纹特征后，才能生成一个指纹模板，登记成功。



提示： 用于保存指纹模板的列必须是备注型、

指纹特征与比对

所谓指纹比对，就是将在FingerCapture事件中捕获的指纹特征，和之前登记的指纹模板进行比对，判断是否是同一个指纹。

指纹比对通过Verify方法实现，语法：

Verify(Template,Features)

Template：字符型，用于比对的指纹模板。
Features：字符型，用于比对的指纹特征。

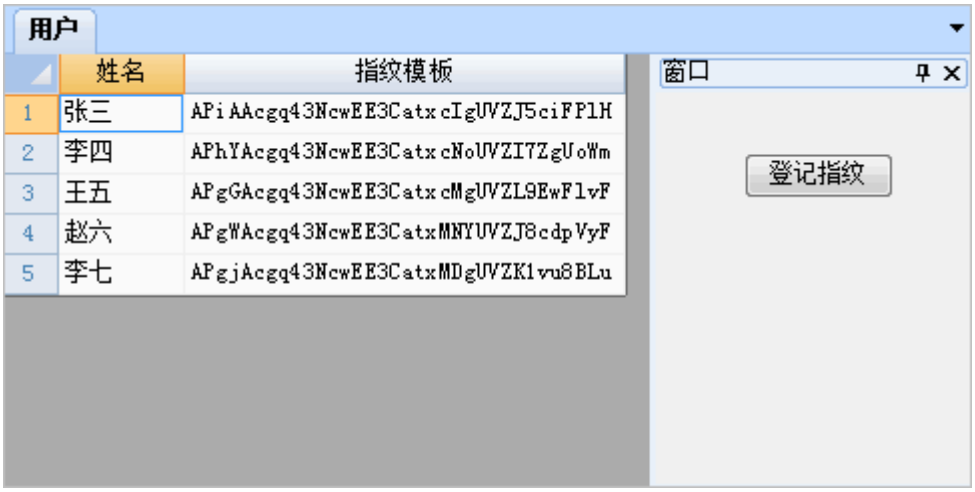
如果指纹特征和指纹模板匹配，则返回True。

例如有个用户表，包括用户姓名、指纹模板、审批三列，其中审批列是个逻辑列，用于标明此用户是否具备审批订单的权利。
现在希望有审批权限的用户，无需登录系统，只需按压指纹，就可以直接审批订单，为此可以将FingerCapture事件代码设置为：

```
If MainTable.Name = "订单" Then
    For Each dr As DataRow In DataTables("用户").Select("审批  = True") '
        If dr.IsNull("指纹模板") = False Then
            If FoxFinger.Verify(dr("指纹模板"),e.Features) Then '如果指纹匹配
                Tables("订单").Current("审批人") = dr("姓名")
                Tables("订单").Current("审批时间") = Date.Today()
                Return
            End If
        End If
    Next
    MessageBox.Show("你无权审核订单","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

8.2.5 一个简单的例子

本节的例子可以参考 CaseStudy 目录下的文件：指纹识别.Table



提示：保存指纹模板的列，必须是备注型。
该示例完整地演示了登记、修改和验证指纹模板的过程：
1、在表中增加一行，输入姓名，单击“登记指纹”按钮，出现“登记指纹”窗口，连续按压四次指纹后，系统会结合这四次指纹的信息，生成一个指纹模板，保存在“指纹模板”列：



2、如果要修改某个用户的指纹模板，先选择此用户，然后单击“登记指纹”。

3、任何时候按压指纹，都会将获得的指纹模板，和用户表中的指纹模板逐个进行比对，如果和某用户的指纹模板相符，就显示此用户的姓名。

设计步骤非常简单：

1、在项目事件 AfterOpenProject 中加上下面的代码，用于启动指纹仪：

```
FOXFinger.Start()
```

2、将窗口中“登记指纹”按钮的代码设置为：

```
Dim fg As String = FoxFinger.EnRoll() '登记指纹
If fg > ""  Then '如果登记成功
    Tables("用户").Current("指纹模板") = fg '保存获得的指纹模板
    Tables("用户").Current.Save()
End If
```

3、将FingerCapture事件的代码设置为：

```
For Each dr As DataRow In DataTables("用户").DataRows
    If dr.IsNull("指纹模板") = False Then
        If FoxFinger.verify(dr("指纹模板"),e.Features) Then '如果指纹模板相符
            MessageBox.Show("您就是" & dr("姓名") & ",对吗?","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
            Return
        End If
    End If
Next
MessageBox.Show("没有找到指纹相符的用户","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
```

8.2.6 一些 UI 图片

为方便大家设计和指纹识别相关的用户界面，特提供以下 UI 图片：









8.2.7 设计指纹登录系统

在学习本节内容之前，请先打开 CaseStudy 目录下的示例文件“指纹登录系统.Table”。
本节的任务是设计一个下图所示的登录窗口：



该登录窗口能自动识别用户指纹，指纹匹配和自动进入系统，显然和传统的密码登录方式相比，这种方式更加快捷安全，更有“科技”感。
设计步骤
1、首先准备两个 gif 图片，放在项目所在目录的 Images 子目录下：



文件名为 finger1.gif，等待识别指纹的时候显示。



文件名为 finger2.gif，识别指纹失败的时候显示。

2、在全局代码中定义一个 Public 变量:

```
Public _UserName As String
```

3、设计一个上图所示的窗口，这个窗口没有任何代码，就两个控件，一个 PictureBox 和一个 Label，PictureBox 的显示图片设置为 finger1.gif

4、在项目事件 AfterOpenProject 中加上代码：

```
FOXFinger.Start()
Forms("登录窗口").Open()
If _UserName = "" Then
    Syscmd.Project.Exit()
    Return '这个要加上,因为既然要退出,就不应该再执行后面的初始化代码了,否则会出错
End If
'后续初始化代码
```

5、将FingerCapture事件的代码设置为：

```
If Forms("登录窗口").Opened Then '如果登录窗口打开
    For Each dr As DataRow In DataTables("用户").DataRows
      If dr.IsNull("指纹模板") = False Then
        If FoxFinger.verify(dr("指纹模板"),e.Features) Then '如果指纹模板相符
          _UserName = dr("姓名")
          Forms("登录窗口").Close()
          Exit For '退出循环
        Else
          Dim pb As WinForm.PictureBox = Forms("登录窗口").Controls("PictureBox1")
          pb.Image = GetImage("Finger2.gif") '如果指纹失败，显示图片finger2.gif，用以提示用户
        End If
      End If
    Next
End If
```

8.2.8 设计指纹审核系统

大多数管理系统的审核功能，只要有审核权限的用户登录系统，即可随意审核通过所有符合条件的记录。

这种方式符合一般场合的需求，但是也有不足：

1、用户在登录系统后，有可能忘记退出，带来安全隐患。
2、而且有审核权限的用户，必须先登录系统，才能审核，不适合需要快速审核的场合。
例如你在银行柜台经常会看到这样的场景，A 工作人员处理完成某笔业务的时候，B 工作人员按压指纹进行审核，并不要求 A 先退出，由 B 登录系统后再审核。
设计一个这样的指纹审核系统其实是非常简单的，步骤：
1、首先设计一个用户表，包括三列，分别是姓名、指纹模版、审核，其中审核列是个逻辑列，用于标记某个用户是否有审核权限：

	姓名	指纹模板	审核
1	张三	APiAAcgq43NcwEE3CatxcIgUV	☐
2	李四	APhYAcgq43NcwEE3CatxcNoUV	☑
3	王五	APgGAcgq43NcwEE3CatxcMgUV	☐
4	赵六	APgWAcgq43NcwEE3CatxMNYUV	☑
5	李七	APgjAcgq43NcwEE3CatxMDgUV	☐

2、将 FingerCapture 事件的代码设置为：

```
If Forms("订单编辑").Opened AndAlso Forms.ActiveForm.Name = "订单编辑"  Then '如果当前窗口是订单编辑
    Dim r As Row = Tables("订单").Current '取得当前订单
    '在此处进行条件判断,符合条件的订单才允许审核.
    'If  条件不成立  Then
      'Return
    'End If
    For Each dr As DataRow In DataTables("用户").DataRows
      If dr.IsNull("指纹模板") = False Then
        If FoxFinger.verify(dr("指纹模板"),e.Features) Then '如果指纹模板相符
          If dr("审核") = True
            r("审核人") = dr("姓名")
            r("审核时间") = Date.Now()
            Return '完成审核, 退出系统
          End If
        End If
      End If
    Next
    MessageBox.Show("你无权审核订单!","提示",MessageBoxButtons.OK,MessageBoxIcon.Information)
End If
```

8.3 使用身份证阅读器

官方网站为本节内容提供了示例，下载地址：
http://www.foxtable.com/samples/Identify.rar
本节的例子以深证华视电子读写设备有限公司的 CVR-100U 为例，该公司的网址：

http://www.chinaidcard.com/

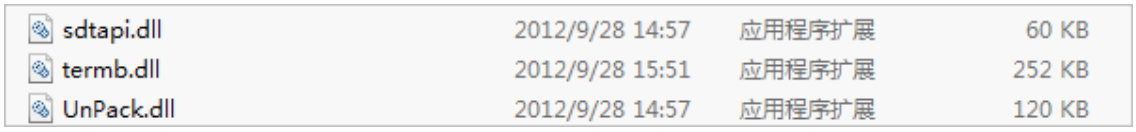
需要说明的是，该公司和我们没有任何关系，类似的阅读仪器有很多，大家可以根据需要选择。
本节的任务是设计一个下图所示的窗口，当有新的身份证接触阅读器的时候，能都自动读取身份证中的所有信息(包括照片)，并在 Foxtable 中新增一行数据，将读取的信息写入到新增行中。



设计步骤：
1、首先根据说明书，连接好阅读仪，并安装好设备驱动程序，正常情况下，在 Windows 的设备管理中，会出现下面一项：



2、根据该设备的开发文档，将其 SDK 文件复制到 Foxtable 的安装目录下，该设备的 SDK 文件包括以下几个文件:



我们提供的示例文件也包括了上图所示的 sdk 文件，下载地址：

http://www.foxtable.com/samples/Identify.rar

在 Foxtable 的全局代码中定义好 API 函数：

```
Public Declare Function CVR_InitComm Lib "termb.dll" Alias "CVR_InitComm" (ByVal Port As Integer) As Integer '连接设备
Public Declare Function CVR_Authenticate Lib "termb.dll" Alias "CVR_Authenticate" () As Integer '验证身份证
Public Declare Function CVR_Read_Content Lib "termb.dll" Alias "CVR_Read_Content" (ByVal Active As Integer) As Integer '读取身份证信息
Public Declare Function CVR_CloseComm Lib "termb.dll" Alias "CVR_CloseComm" () As Integer '关闭设备
```

在编写本节内容时，其 SDK 的最新版本是 4.0。

提示：由于他们的开发包是传统的非托管 DLL，所以只需复制到 Foxtable 安装目录即可使用，千万不要再去引用这些 DLL 文件。

3、根据开发文档，CVR_Read_Content 函数有多种读取模式，本示例使用的是模式 4，在此模式下，会生成相片文件"zp.bmp"和身份证信息文件"wz.txt"，后者的格式如下：

张三
男
汉
1951 年 12 月 25 日
广东省湛江市麻章人民大道 16 号
412111019511225148X
湛江市公安局赤坎分局
2006.11.28-2026.11.28

我们增加一个自定义函数，用于解析"wz.txt"文件的内容，并写入 FoxTable，函数名为“Parse”，代码为：

```
Dim Info As String
Info = FileSys.ReadAllText(ApplicationPath & "wz.txt", Encoding.Default) '读取生成的文本文件
info = info.Replace(vbcrlf,vblf) '将回车换行替换为换行
Dim pars() As String = Info.Split(vblf) '将读取的信息拆分为数组
Dim r As Row = Tables("表 A").AddNew
r("姓名") = pars(0).Trim()
r("性别") = pars(1).Trim()
r("名族") = pars(2).Trim()
r("出生日期") = pars(3).replace("年","-").Replace("月","-").Replace("日","")
r("住址") = pars(4)
r("身份证号码") = pars(5)
r("发证机关") = pars(6)
r("签发日期") = pars(7).SubString(0,10).Replace(".","-")
r("有效日期") = pars(7).SubString(11).Replace(",","-")
FileSys.CopyFile(ApplicationPath & "zp.bmp", ProjectPath & "\Attachments\" & r("身份证号码") & ".bmp",True) '照片复制到 Attachments 目录
r("照片") = r("身份证号码") & ".bmp"
```

4、将窗口的 AfterLoad 事件代码设置为:

```
Try
    Dim lb1 As WinForm.Label = e.Form.Controls("Label1")
    Dim iRetUSB As Integer
    For iPort As Integer = 1001 To 1016 '遍历端口号
        iRetUSB = CVR_InitComm(iPort)
        If iRetUSB = 1 Then
            Exit For
        End If
    Next
    If iRetUSB = 1 Then
        lb1.Text = "设备连接成功"
    Else
        lb1.Text = "设备连接失败"
        e.Form.TimerEnabled = False '停止计时器,使得自动读取功能失效
        e.Form.Controls("Button1").Enabled = False '禁用读卡按钮
        e.Form.Controls("CheckBox1").Enabled = False '禁用自动读取复选框
    End If
Catch ex As Exception
    MessageBox.Show(ex.ToString)
End Try
```

这样打开窗口的时候，将自动连接阅读器。

5、将窗口的 BeForeClose 事件代码设置为：

```
Try
    CVR_CloseComm() '断开设备
Catch ex As Exception
```

```
    MessageBox.Show(ex.ToString)
End Try
```

确保窗口关闭的时候，能自动断开阅读器。

6、启用窗口计时器，间隔设置为 500，TimerClick 事件代码设置为:

```
Dim lb1 As WinForm.Label = e.Form.Controls("Label1")
If CVR_Authenticate= 1 Then '如果身份证验证通过
    Dim readContent As Integer = CVR_Read_Content(4) '读取身份证信息,注意读取模式设置为 4.
    If readContent = 1 Then
        lb1.Text = "读身份证信息成功!"
        Functions.Execute("parse")
    Else
        lb1.Text = "读身份证信息失败!"
    End If
End If
```

提示：间隔不能设置得小于 500，否则会影响其他软件包括 foxtable 的运行，对于配置较差的电脑，最好是 1000 左右。

7、手工读取信息按钮的代码设置为：

```
Dim lb1 As WinForm.Label = e.Form.Controls("Label1")
If CVR_Authenticate= 1 Then '如果身份证验证通过
    Dim readContent As Integer = CVR_Read_Content(4) '读取身份证信息,注意读取模式设置为 4.
    If readContent = 1 Then
        lb1.Text = "读身份证信息成功!"
        Functions.Execute("parse")
    Else
        lb1.Text = "读身份证信息失败!"
    End If
Else
    lb1.Text = "没有检测到有效身份证!"
End If
```

8.4 来电显示与录音

官方网站为本节内容提供了示例，下载地址：

http://www.foxtable.com/samples/caller.rar

本节的任务是设计一个窗口，以软件方式实现电话终端的管理。本例实现来电弹屏、通话录音存储、通话详单记录、自动答录、快速拨号等多项功能

一. 录音盒

本节的例子以南京子佩软件科技有限公司的子佩录音盒为例，该公司的网址：

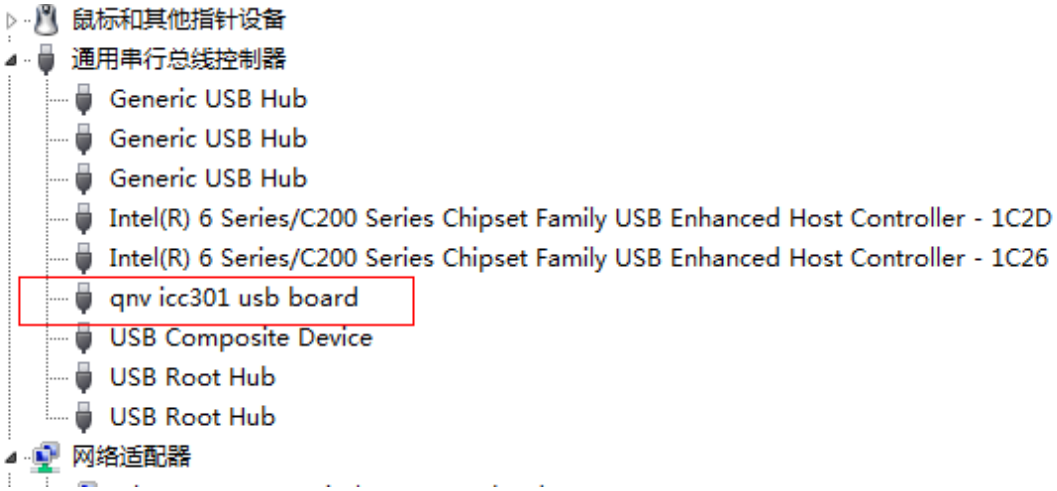
http://www.luyin888.com/

需要说明的是，该公司和我们没有任何关系，类似的电话录音盒有很多，大家可以根据需要选择。

二. 录音盒连接

根据说明书，连接好电话录音盒设备，并安装好设备驱动程序，正常情况下，在 Windows 的设备

管理中，会出现下面一项：



这就是录音盒设备。

三. Foxtable 示例

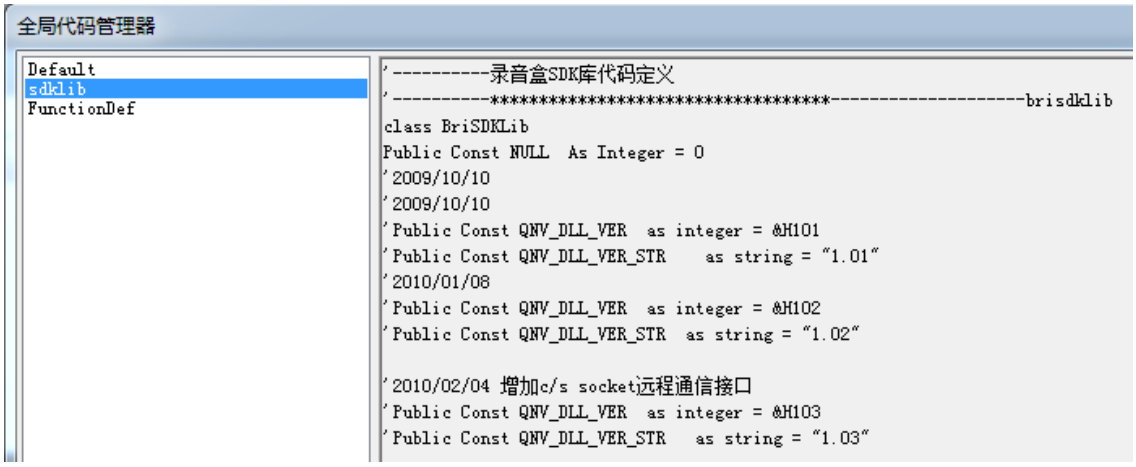
3.1 使用 SDK

3.1.1 根据该设备的开发文档，将其 SDK 文件复制到 Foxtable 的安装目录下，该设备的 SDK 文件包括 qnviccub.dll 和 bridge.dll 两个必须的 DLL 文件。

在编写本节内容时，其 SDK 的最新版本是 6.6。

提示：由于他们的开发包是传统的非托管 DLL，所以只需复制到 Foxtable 安装目录即可使用，千万不要再去引用这些 DLL 文件。

3.2.2. 参考开发文档和例子，在全局代码中定义好 API 函数和必须的全局变量，由于内容较多，具体请看示例。



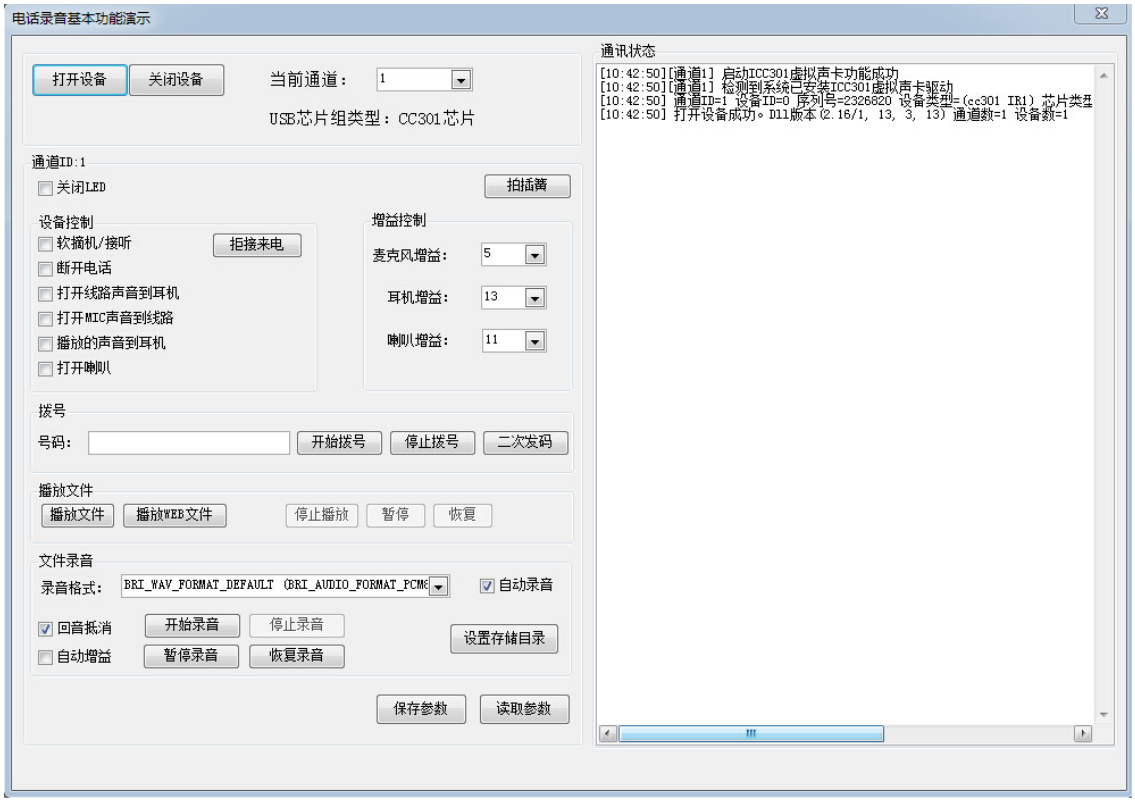
3.2 设计数据表

示例设计了三个表进行电话录音管理示例，具体应用需要根据实际情况自行设计表格进行管理：

表"电话记录"主要记录一次呼叫从摘机到挂机的通讯状态；表"用户管理"用来管理联系人和电话号码；表"设置"用来记录示例的一些配置。

3.3 设计设备管理窗口

示例设计了一个窗口（电话录音基本功能演示）来进行设备的控制操作。



窗口分为左右 2 个部分，左边是设备控制，右边是设备的通讯状态。通讯状态在窗口关闭后会追加到日志文件“通讯状态.txt”中，日志文件默认保存在项目所在目录中。

3.3.1 打开设备

要使用设备必须先调用 SDK 打开设备，根据开发文档，打开设备的调用为：

```
Dim lret As Int32 = BriSDKLib.QNV_OpenDevice(BriSDKLib.ODT_LBRIDGE, 0, "")
```

如果返回值大于 0，则设备打开成功，设备 LED 灯亮。

然后就可以获取设备可用的通道数，一个设备至少有一个通道，获取通道的函数调用为：

```
BriSDKLib.QNV_DevInfo(0, BriSDKLib.QNV_DEVINFO_GETCHANNELS)
```

全部代码参考示例打开设备按钮的 Click 事件代码。如果想一打开窗口就自动打开设备，可以把打开设备的调用放到窗口的 **AfterLoad** 事件中。甚至，可以把打开设备的调用放到项目的 **AfterOpenProject** 事件中，那么一打开项目就可以打开设备进行监控，不用再手动操作。

3.3.2 关闭设备

设备不用的时候，关闭设备，释放资源，关闭设备的函数调用为：

```
BriSDKLib.QNV_CloseDevice(BriSDKLib.ODT_ALL, 0)
```

示例同样在关闭窗口的事件 BeforeClose 中调用执行了关闭操作。如果打开设备是在项目的 AfterOpenProject 事件中，那么建议在项目的 BeforeCloseProject 中关闭设备，避免占用设备的通道，而导致无法再次被使用

3.4 设备消息处理

该设备主要使用了消息发布的方式来触发相应的事件，所以我们在全局代码定义一个自定义类进行通讯事件处理，类名称 FormEventReport，类定义为：

```
Public Class FormEventReport
Inherits System.Windows.Forms.Form
.....
End Class
```

FormEventReport 类使用之前必须先打开设备。

FormEventReport 类继承自 System.Windows.Forms.Form，所以实际上就是一个自定义的窗体，这样我们就可以重载基类中的 DefWndProc 过程来接收设备的事件消息。

首先在 FormEventReport 的 Load 过程中给设备的所有通道注册事件信息：

```
Private Sub FormCall_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
For i As Int16 = 0 To BriSDKLib.QNV_DevInfo(0, BriSDKLib.QNV_DEVINFO_GETCHANNELS) - 1
    '在windowproc处理所有通道接收到的消息
    BriSDKLib.QNV_Event(i, BriSDKLib.QNV_EVENT_REGWND, CType(Me.Handle, Int32), "", New StringBuilder(0), 0)
Next
vars("IsEventFormOpened") = True
End Sub
```

然后重载DefWndProc过程，判断接收到的消息参数，进行相应的操作：

```
Protected Overrides Sub DefWndProc(ByRef m As System.Windows.Forms.Message)
Select Case m.Msg
    Case BriSDKLib.BRI_EVENT_MESSAGE
        If True Then
            Dim EventData As BriSDKLib.TBriEvent_Data = DirectCast(Marshal.PtrToStructure(m.LParam, Gettype(BriSDKLib.TBriEvent_Data)), BriSDKLib.TBriEvent_Data)
            Select Case EventData.lEventType
                Case BriSDKLib.BriEvent_PhoneHook '电话机摘机
                    AppendStatus("电话机摘机",EventData.uChannelID)
                    drCurrentRow = DataTables("电话记录").AddNew()
                    Tables("电话记录").Position = Tables("电话记录").Rows.Count-1
                    AppendCallRecord("电话机摘机",EventData.uChannelID)
                    Exit Select
                Case BriSDKLib.BriEvent_PhoneHang '电话机挂机
        ......
        ......
End Sub
```

这里只是部分代码，完整代码请查看示例。

做法无非就是根据系统的消息，判断是否是本录音设备的消息，如果是，则取得消息中返回的设备信息，判断是设备的那个消息类型，来进行对应的操作。比如判断到是设备的摘机事件（BriEvent_PhoneHook），就增加一条电话记录。

示例定义了一个全局的FormEventReport类型变量EventForm，在打开设备后创建EventForm的实例，打开窗体后隐藏起来即可。因为这个窗体设计只是用来接收系统消息的，所以不需要显示。

最后在关闭设备的时候记得注销设备的消息处理，如本示例写在关闭设备按钮中：

```
'删除事件接收
For i As Integer = 0 To cboCurChn.Items.Count - 1
    BriSDKLib.QNV_Event(i,BriSDKLib.QNV_EVENT_UNREGWND,CType(basemainform.Handle, Int32),Nothing,Nothing,0)
Next
```

3.5 用户管理

在“电话记录”单元格中右键点击，出现有“添加联系人”菜单，点击后会转到“用户管理”表，如果已经有同一个电话号码的记录，则定位到相应的记录；如果没有，则会增加一条记录，默认添加上电话号码。

3.6 其它

3.6.1 关于设备返回值

SDK某些函数的调用，返回字符串的，实际是返回了字符串的内存起始地址，即Intptr类型。所以示例使用Marshal.PtrToStructure将数据从非托管内存块封送到新分配的指定类型的托管对象中。示例定义了TBriEvent_Data结构（位于SDK库类BriSDKLib中）和CallLog_Data结构（全局代码）来取得非非托管内存块数据。例如CallLog_Data用法如下：

1. 定义结构：

```
'呼叫信息结构，用于号码，录音文件路径等
<StructLayout(LayoutKind.Sequential)> _
Public Structure CallLog_Data
<MarshalAs(UnmanagedType.ByValArray, SizeConst: = BriSDKLib.MAX_BRIEVENT_DATA)> _
Public szData As [Byte]()
End Structure
```

由于未知返回值的长度是多少，所以示例统一使用SDK定义的MAX_BRIEVENT_DATA值，即600字节

2. 获取返回的地址

如获取呼叫号码：

```
Dim ipt As IntPtr = BriSDKLib.QNV_CallLog(EventData.uChannelID,BriSDKLib.QNV_CALLLOG_CALLID,Nothing,BriSDKLib.MAX_BRIEVENT_DATA)
```

3. 将数据从非托管内存块封送到新分配的CallLog_Data结构中

```
Dim calllogData As CallLog_Data = DirectCast(Marshal.PtrToStructure(ipt , Gettype(CallLog_Data)), CallLog_Data)
```

4. 使用Encoding.Default.GetString转换结构中的字节数组，目前封装在示例的函数ByteArrayToString中，如：

```
ByteArrayToString(calllogData.szData)
```

需要注意的是，转换后的字节数组长度是600字节，内容是大于实际返回值的，其中多余的字节中的数据必须处理掉，不然就会返回一些乱码，还好正常返回值后是以Null字符（chr(0)）结束的，所以ByteArrayToString函数中判断取得第一个Null字符，截断后面的内容。

8.5 智能 IC 卡读写示例

官方网站为本节内容提供了示例，下载地址：

http://www.foxtable.com/samples/iccard.rar

一. 关于智能卡读写器

本节的例子以深圳市明华澳汉科技股份有限公司的 USB 非接触式读写器 EYE 系列 eye-U010 为例，该公司的网址：

http://device.mwcard.com/

需要说明的是，该公司和我们没有任何关系，类似的智能卡读写器有很多，大家可以根据需要选择。

本节的例子可以参考 CaseStudy 目录下的文件：智能卡读写器连接示例.Table。

eye-U010 为无驱类型，插上 USB 口即可自动识别使用。

本示例智能卡以市场上常用的 M1（**Mifare 1**）智能卡读写为示例。其它类型的卡使用函数可能有不同，详情请参考 SDK 开发文档。

二. 读写器智能卡开发参考

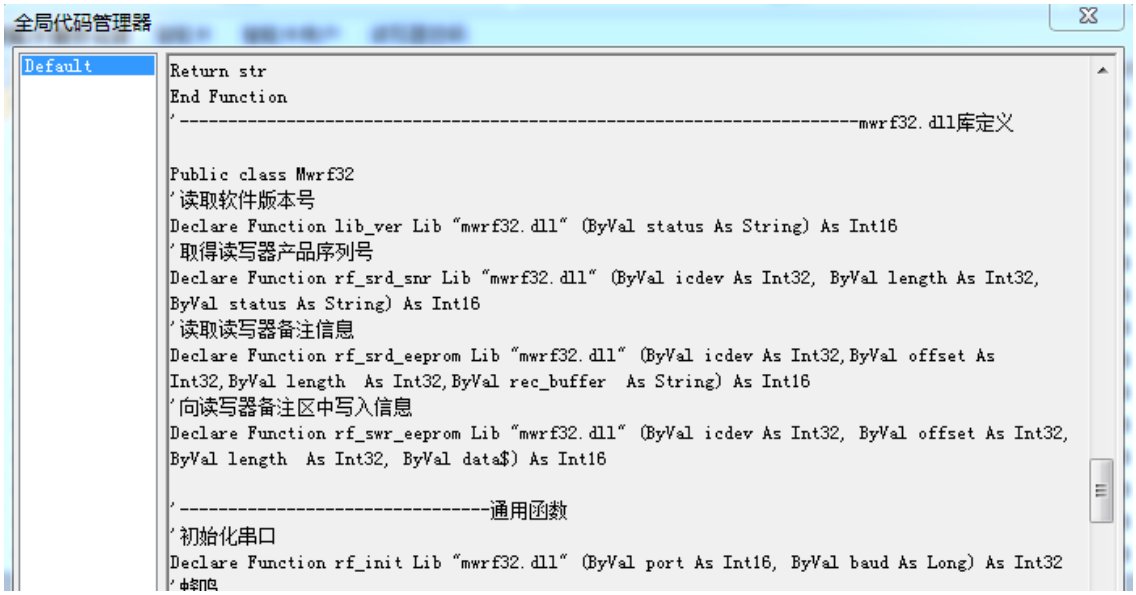
2.1 使用 SDK

2.1.1 根据该设备的开发文档，将其 SDK 文件复制到 Foxtable 的安装目录下，该设备的 SDK 文件仅有一个 DLL 文件：mwrf32.dll。

在编写本节内容时，其 SDK 的最新版本是 2.02。

提示：由于他们的开发包是传统的非托管 DLL，所以只需复制到 Foxtable 安装目录即可使用，千万不要再去引用这些 DLL 文件。

2.2.2. 参考开发文档和例子，在全局代码中定义好 API 函数和必须的全局变量，由于内容较多，具体请看示例。



2.2 开发要点说明

2.2.1 关于 M1 卡

M1 卡为 8K 位的非接触式 IC 卡，详细的特性介绍请参考 SDK 的帮助文档。这里主要针对一些需要注意的地方进行说明。

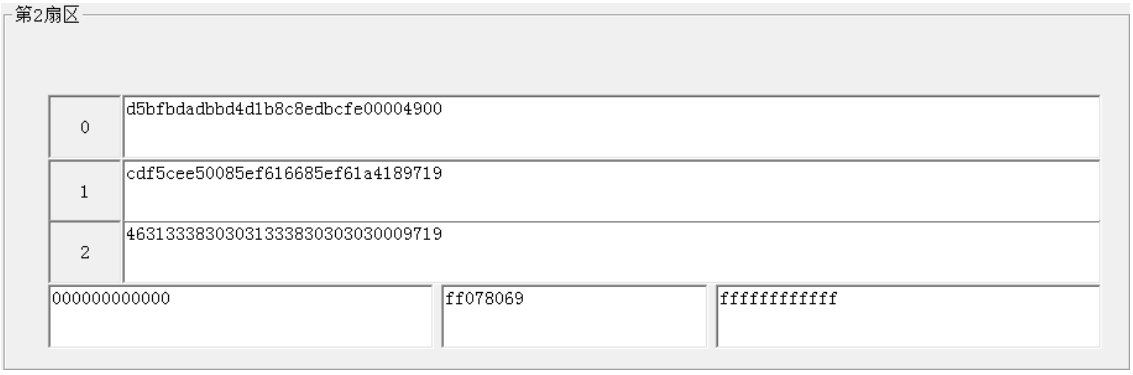
2.2.1.1 关于存储

M1 卡分为 16 个扇区，每个扇区 4 块（块 0～3），共 64 块，按块号编址为 0～63。第 0 扇区的块 0（即绝对地址 0 块）用于存放厂商代码（前 4 个字节为卡的序列号，其它为厂商代码），已经固化，不可更改。其他各扇区的块 0、块 1、块 2 为**数据块**，用于存贮数据；块 3 为**控制块**，存放密码 A、存取控制、密码 B，其结构如下：

A0A1A2A3A4A5	FF 07 80 69	B0B1B2B3B4B5
密码 A(6 字节)	存取控制(4 字节)	密码 B(6 字节)

M1 卡每一个块都可以存储最多 16 个字节的内容（16 个字符或 8 个汉字），以十六进制编码存储，如需存储更多的内容只能是分割存储到多个块中。

SDK 向智能卡存储汉字采用的是 GBK 内码，存储普通字符采用的是 ASCII 编码，值方式存储的是 16 进制值，均为十六进制形式，如下面厂家 Demo 截图所示，块 0 存储的是“湛 d5bf 江 bdad 辉 bbd4 迅 d1b8 软 c8ed 件 bcfe ”,块 2 存储的是字符 F13800138000,ASC 码为 463133383131313338303030。不满 16 个字节后面加 00（Chr(0)）作为结束，图中 00 后面的字符为 SDK 的控制符，不用理会，实际使用时，取 00 为结束符截断后面的内容即可，否则 00 后面出现的是乱码。



2.2.1.2 关于密码

M1 卡块每个扇区的第三块为**控制块**，左边 6 个字节是密码 A，按 SDK 和官方的说明是屏蔽不可读的，所以无论向密码 A 中写入任何密码都显示不了，通过 SDK 返回的都是 000000000000。

控制块中间 4 个字节（第 6 至第 9 字节）是存取控制，具体的控制位设置参考 SDK 文档，这里不做详述。需要说明的是存取控制一般采用智能卡出厂的默认值（ff078069）即可,如需自行控制扇区的读写权限请先参考 SDK 理解透存取控制位的变化，乱写的话可能导致扇区被锁无法操作扇区里的数据。

控制块最后 6 个字节是密码 B，写入的密码可以获取显示，不需要密码 B 的话也可以用来存储其它自定义的内容。

密码使用必须是 12 个十六进制字符，即是 0~9 或 a~f 这十六个字符。

建议实际应用的时候存储的内容都进行加密处理。

更改密码有 2 种方式：

一是专用块 3 修改函数 **rf_changeb3**，可以根据参数分别控制密码 A、密码 B 和控制位的修改。可以参考示例“改写密码”按钮代码。

二是使用写数据的方式更改块 3 的数据，比如函数 **rf_write_hex**，和 rf_changeb3 不同的是，rf_write_hex 需要把块 3 的内容拼成一起作为参数传递。可以参考示例“写入数据”按钮代码。

2.2.2 关于密码验证

智能卡的密码验证，有装载密码验证和直接密码验证 2 种方式：

装载密码即是把需要校验的密码装载到读写器的存储单元（RAM）中，可以单独装载单个扇区的密码 A 或者密码 B，也可以一次性装载所有扇区的密码。这种验证方式安全性高，装载密码可以和卡片交易可以分离操作；缺点是读写器密码存储单元有使用寿命，一般为十万次，如果装载过于频繁超过使用寿命，它将会坏掉不能再装载密码。

直接密码验证指 PC 机向读写器发送卡片认证指令时将认证密码作为命令参数一起发送，直接使用参数中的密码对卡片进行认证。这种验证方式没有装载密码验证的使用寿命限制；缺点是安全性不够。

厂商建议使用装载密码验证的方式，毕竟实际应用中频繁断电和更改密码还是很少的，十万次的使用寿命一般都够用了。

示例设计了一个表“读写器密码”记录装载的密码。为了测试方便，用的是明文，实际应用可以

进行加密。

2.2.3 SDK 函数使用规则

对已经连接设备并打开设备的情况下，对智能卡操作的函数调用建议如下：

（1）读卡：调用 rf_card()

（2）装载密码，调用 rf_load_key()

如果在打开设备的时候已经一次性装载所有扇区的密码则可以省略此步骤

（3）验证密码，调用 rf_authentication()

（4）进行操作，包括对卡的读写及值操作

（5）结束本次操作，调用 rf_halt()

每次操作都需要重复调用以上步骤，不然大部分低级函数会报错，返回无卡，或者验证密码失败的操作结果。

比如要读出扇区 2 的第 0 块的数据，调用方式如下，其中 icdev 为定义的全局变量，连接设备时存储设备句柄：

```
Dim snr(4) As Byte
Dim snr1 As New String(snr1, 8)
Dim databuff As New String(databuff, 16)
Dim sector = 2
Dim st = mwrf32.rf_card(icdev, 1, snr(0))'读卡
If st = 0 Then
    Mwrf32.hex_a(snr(0), snr1, 4)'转换为可识别的字符串
    e.Form.Controls("txtCardNumber").Text = snr1
    st = mwrf32.rf_authentication(icdev, 0, sector )'验证该扇区密码
    If st = 0 Then
        st = mwrf32.rf_read(icdev, sector * 4, databuff)
        If st = 0 Then
            databuff   = ClearNullString(databuff)
            e.Form.Controls("txtData0").Text = databuff
        Else
            '读块数据出错提示
        End If
    Else
        '密码验证出错提示
    End If
Else
    '读卡出错提示
End If
mwrf32.rf_halt(icdev)
```

为了重用方便，示例用一个函数 CheckCard，包装了 rf_card 和 rf_authentication，这样上面的例子及可以像下面这样，减少代码的重复：

```
Dim sector = 2
Dim databuff As New String(databuff, 16)
If CheckCard(sector ) Then
    Dim st = mwrf32.rf_read(icdev, sector * 4, databuff)
    If st = 0 Then
        databuff   = ClearNullString(databuff)
```

```
        e.Form.Controls("txtData0").Text = databuff
    Else
        '读块数据出错提示
    End If
End If
```

SDK 函数分低级函数和高级函数 2 种，具体区分详见 SDK 文档函数的功能描述。

其中高级函数包含几个低级函数的调用，都已经包含寻卡和验证密码的操作，并且函数结束会调用 rf_halt()，无需按照上面的 5 个步骤来操作，如示例刷新数据中调用的 M1 开专用高级读函数 rf_HL_read。

低级函数则需要按照上面的 5 个步骤按序操作。

2.2.4 寻卡模式

看 SDK 的寻卡模式说明可能会被误导：

> 寻卡模式分三种情况：IDLE 模式、ALL 模式及指定卡模式。
> 0——表示 IDLE 模式，一次只对一张卡操作；
> 1——表示 ALL 模式，一次可对多张卡操作；
> 2——表示指定卡模式，只对序列号等于 snr 的卡操作（高级函数才有）

模式 0，一次只对一张卡操作，是指对这张卡操作完成（调用高级函数，或者低级函数调用（如果是 ALL 模式函数操作后调用 rf_halt()也一样）结束）后必须拿离感应区再置回感应区内才能使函数调用再次生效。

模式 1，一次可对多张卡操作，并非可以同时操作多张卡，而是指对这张卡操作完成后，智能卡无需离开感应区即可进行其它函数的调用。模式 1 是可对一张卡进行多次操作的意思。

模式 2，指定序列号，必须是在先寻卡知道序列号的情况下才能使用，应用场景不多。少数几个高机函数有，在这些高级函数中，如果不传入序列号，则默认取当前在感应区中可用的卡，如果传入序列号，则只操作对应序列号的卡。

示例为了测试方便，大多采用了模式 1 的方式操作智能卡。

为了避免智能卡的误操作，厂方建议还是使用模式 0 的方式，然后使用 rf_halt()结束一次操作，这样操作完后，必须重新拿离卡才能再次操作，避免了类似忘记换卡又操作了一次的情况出现。当然也可以通过良好的设计来避免这种情况，比如写操作前进行序列号验证，多次写操作提示等等，这些具体的用法就不在此详述了。

示例批量添加卡就使用了模式 0 的特点。比如，需要增加一些新卡进行管理，可以一次放 20 张智能卡在读写器的感应区内，然后使用模式 0 循环寻卡，由于一张卡读过后只要不拿离感应区就不会被再次寻卡，这样直到所有的卡都被读过后，最后返回无卡的操作结果，结束循环。主要部分代码如下：

```
.....
Dim st = 0
Dim dr As DataRow
Do While st = 0 '所有卡都读过后最后会返回1,表示无卡,然后离开循环
    st = Mwrf32.rf_card(icdev,0,snr(0))
    AppendStatus(GetErrorDesc(st))
    If st = 0 Then
        Mwrf32.hex_a(snr(0), snr1, 4)
        dr = DataTables("智能卡").Find("序列号 = '" & snr1 & "'")'查找是否已经添加过这张卡,避免重复添加
        If dr IsNot Nothing Then
```

```
            AppendStatus("序列号 = " & snr1 & "(卡号:" & dr("卡号") & ")的智能卡已存在,无法重复添加")
        Else
            dr = DataTables("智能卡").AddNew()
            dr("序列号") = snr1
            dr("卡号") = i.ToString().PadLeft(5,"0")
            i = i+1
            f = f+1
        End If
    End If
Loop
.....
```

2.2.5 关于值操作

我们经常见到例如公交卡、饭卡、购物卡等等智能卡里面都是有金额的，可以刷卡消费。可见智能卡的值操作是很常见的，SDK 同样提供了方便的值操作函数，不过有一定的局限，就是只能操作正整数的值，当然这是可以理解的。

SDK 的值操作是把数字转换为 16 进制值存储的，而不是存储数值的十六进制编码。

其中 rf_initval、rf_increment、rf_decrement 分别实现初始化值、加值和减值操作，直接传入整形的值即可。

rf_readval 读值函数返回的是值的内存地址，返回的只是前 4 个字节，必须做处理才能获取正确的值。

从函数可以看出，SDK 的值操作只支持整形，至于可以存储多大范围的值，下面会讲到。

比如我们往一个块里用初始化值函数存入 1000，查看块里存入的 16 进制串，是这样的：

e803000017fcffffe803000004fb04fb

嘿，我的 1000 去那里了呢。这串字符其实要这样理解的，先把字符分成 4 份，4 个字节一份，就是这样：

1	2	3	4
e8030000	17fcffff	e8030000	04fb04fb

可以看出第 1 份和第 3 份是一样的。我们把第 1 份的 4 个字节倒过来，成了这样 000003e8，实际上就是&H3e8，转换成 10 进制就是 1000。也就是说 SDK 是把十进制值转换成 16 进制值后，不够 4 个字节（32 位）的前面用 0 凑齐 4 个字节，然后把这 4 个字节按低位开始倒序存储。比如 10 进制值 50000 转换为 16 进制值就是 C350，存入块中就是 50C30000。

基于这个特性，取值的时候需要做个转换，示例写了个全局函数

```
'根据值字节(4个字节长度)返回正确的值,适用于值操作,因为值操作块中存储的是值的16进制数值,而不是值的16进制编码
Public Function HexByteToLongValue(ByVal hex() As Byte) As Long
  Dim str As String = "&H"
  For i As Integer = hex.Length-1 To 0 Step -1
    str &= hex(i).ToString("X2")
  Next
  Return val(str)
End Function
```

那么第 2 份又是什么东西？同样把 4 个字节倒排为 fffffc17，然后取反就知道了，大家在命令窗口用 Output.Show(not &Hfffffc17)看看结果。

最后 4 个字节是厂家的控制信息，每个块固定都一样的，不用理会就是。
如果块存储的不是值形式，而是用非值函数存入的数据，那么用值函数操作就会返回“该块数据不是值格式”的错误代码。
可以看出，通过值方式操作的值范围是从 0 到&HFFFFFFFF 之间，当然一般也用不了那么大。

2.3 常用功能函数归纳

2.3.1 连接设备（初始化串口）

icdev = Mwrf32.rf_init(0, 9600)

成功返回设备句柄，由于 eye-U010 为无驱类型，所以参数不用设置，默认即可。
设备句柄 icdev 定义为全局变量，为其它功能操作必备。

2.3.2 关闭设备（释放串口）

Dim st = Mwrf32.rf_exit(icdev)

程序退出后必须释放设备的占用，否则不能被其它程序使用。

2.3.3 读取智能卡内容

主要有 5 个函数，分别是 **rf_HL_read、rf_HL_readhex、rf_read、rf_read_hex 和 rf_readval**，前 2 个是高级函数，最后一个是读值操作，其它是低级函数。
这里注意的是读值函数 **rf_readval**。SDK 的值操作是把数字转换为 16 进制值存储的，而不是存储数字的十六进制编码。

2.3.4 往智能卡写内容

通用函数（低级函数），分别是 **rf_write、rf_write_hex**
高级函数，分别是 **rf_HL_write、rf_HL_writehex**（示例测试时发现高级函数写块 3 会出错，厂家建议改回使用低级函数）
值操作函数，分别是 **rf_HL_initval、rf_increment、rf_decrement**

9. 附录

9.1 系统命令大全

行相关

列相关

编辑相关

表相关

统计相关

排序与筛选

导入导出

项目相关

报表相关

行相关

命令	说明
Syscmd.Row.AddNew()	增加一行
Syscmd.Row.AddNews()	增加多行
Syscmd.Row.Clone()	克隆行
Syscmd.Row.Insert()	插入行
Syscmd.Row.Delete()	删除行
Syscmd.Row.Lock()	锁定行
Syscmd.Row.UnLock()	取消锁定行
Syscmd.Row.MoveUp()	上移行
Syscmd.Row.MoveDown()	下移行
Syscmd.Row.Height()	设置行高
Syscmd.Row.HideSelectedRows()	隐藏选定行
Syscmd.Row.HideUnSelectedRows()	取消未选定行
Syscmd.Row.Load()	重新加载行

列相关

命令	说明
Syscmd.Column.AddDataColumn()	增加数据列
Syscmd.Column.InsertDataColumn()	插入数据列
Syscmd.Column.AddExpressionColumn()	增加表达式列
Syscmd.Column.InsertExpressionColumn()	插入表达式列
Syscmd.Column.EditExpression()	修改计算表达式
Syscmd.Column.Delete()	删除列
Syscmd.Column.Modify()	更改列
Syscmd.Column.Lock()	锁定列
Syscmd.Column.UnLock()	取消锁定列
Syscmd.Column.MoveLeft()	左移列
Syscmd.Column.MoveRight()	右移列
Syscmd.Column.Hide()	隐藏列
Syscmd.Column.UnHide()	取消隐藏列
Syscmd.Column.Freeze()	冻结列
Syscmd.Column.UnFreeze()	取消冻结列
Syscmd.Column.ModifyProperty()	设置列属性
Syscmd.Column.Width()	设置列宽

编辑相关

命令	说明
Syscmd.Edit.SerialFill()	系列填充
Syscmd.Edit.RepeatFill()	重复填充
Syscmd.Edit.Find()	查找
Syscmd.Edit.Replace()	替换
Syscmd.Edit.Undo()	撤销操作
Syscmd.Edit.Redo()	恢复操作
Syscmd.Edit.Symbol()	插入符号
Syscmd.Edit.Copy()	复制
Syscmd.Edit.Cut()	剪切
Syscmd.Edit.Paste()	粘贴
Syscmd.Edit.Delete()	删除
Syscmd.Edit.Reject()	还原数据
Syscmd.Edit.ClearZero()	清零

表相关

命令	说明
Syscmd.Table.AddDataTable()	增加内部数据表
Syscmd.Table.AddTemporaryTable()	增加临时表
Syscmd.Table.OuterTableManager()	打开外部数据表管理器
Syscmd.Table.QueryManager()	打开查询表管理器
Syscmd.Table.AddOuterQuery()	增加外部查询表
Syscmd.Table.ModiStructure()	查看和修改表结构
Syscmd.Table.Copy()	复制表
Syscmd.Table.Delete()	删除表
Syscmd.Table.Rename()	重命名表
Syscmd.Table.ModifyDataRelation()	设置表间关联
Syscmd.Table.ModifyProperty()	设置表属性
Syscmd.Table.Lock()	锁定表
Syscmd.Table.Unlock()	取消锁定表
Syscmd.Table.DockSetting()	停靠设置，对关联表有效。
Syscmd.Table.Load()	重新加载表(同步表)
Syscmd.Table.ModifyUserCellStyle()	设置自定义样式
Syscmd.Table.ModifyLookAndFeel()	设置表样式
Syscmd.Table.ChangeFont()	更改表的字体。

Syscmd.Table.ExcelReportManager()	打开 Excel 报表管理器
Syscmd.Project.ExcelVBCode()	设置 Excel 报表事件
Syscmd.Table.CodeReportManager()	打开专业报表管理器

统计相关

命令	说明
Syscmd.Table.SetMergeMode()	设置合并模式
Syscmd.Table.ToggleMergeMode()	切换合并模式
Syscmd.Table.SetSubtotalMode()	设置汇总模式
Syscmd.Table.SetGrandTotal()	设置合计模式
Syscmd.Table.ToggleSubtotalMode()	切换汇总模式
Syscmd.Table.ReSubtotal()	重复上次汇总模式
Syscmd.Table.ToggleGrandTotal()	切换合计模式
Syscmd.Table.NewSubtotalMode()	新的汇总模式
Syscmd.Table.BuildGroupTable()	分组统计
Syscmd.Table.NewGroupTable()	新的分组统计
Syscmd.Table.RebuildGroupTable()	重复上次分组统计
Syscmd.Table.BuildCrossTable()	交叉统计
Syscmd.Table.NewCrossTable()	新的交叉统计
Syscmd.Table.RebuildCrossTable()	重复上次交叉统计

排序与筛选

命令	说明
Syscmd.Table.ToggleSortAndFilter()	切换排序与筛选
Syscmd.Table.SaveTableView()	保存表视图
Syscmd.Table.TableViewManager()	打开表视图管理器
Syscmd.Filter.Advanced()	高级筛选
Syscmd.Filter.ShowSameValues()	显示重复值
Syscmd.Filter.HideSameValues()	隐藏重复值
Syscmd.Filter.ShowRedundantValues()	显示冗余值
Syscmd.Filter.Equal(True)	等于选定单元格的值
Syscmd.Filter.Equal(False)	等于指定值
Syscmd.Filter.UnEqual(True)	不等于选定单元格的值
Syscmd.Filter.UnEqual(False)	不等于指定值
Syscmd.Filter.MoreThan(True)	大于选定单元格的值
Syscmd.Filter.MoreThan(False)	大于指定值

Syscmd.Filter.NotMoreThan(True)	不大于选定单元格的值
Syscmd.Filter.NotMoreThan(False)	不大于指定值
Syscmd.Filter.LessThan(True)	小于选定单元格的值
Syscmd.Filter.LessThan(False)	小于指定值
Syscmd.Filter.NotLessThan(True)	不小于选定单元格的值
Syscmd.Filter.NotLessThan(False)	不小于指定值
Syscmd.Filter.FilterByQuarter(Quarter)	筛选某季度的数据，参数 Quarter 指定季度。
Syscmd.Filter.FilterByMonth(Month)	筛选某月的数据，参数 Month 指定月份。
Syscmd.Filter.FilterBoolean(Value)	逻辑列筛选，Value 设为 True，筛选已经选中的行，设为 False，筛选未选中的行。
Syscmd.Filter.LockStatusFilter(Value)	锁定状态筛选，参数 Value 设为 True，筛选已经锁定的行，设为 False，筛选未选中的行。
Syscmd.Filter.SameYear()	同年
Syscmd.Filter.SameQuarter()	同季度
Syscmd.Filter.SameMonth()	同月
Syscmd.Filter.SameDay()	同天
Syscmd.Filter.SameWeek()	同周
Syscmd.Filter.SameHour()	同时
Syscmd.Filter.SameMinute()	同分
Syscmd.Filter.SameSecond()	同秒
Syscmd.Filter.Values()	按值筛选
Syscmd.Filter.Expression()	表达式筛选
Syscmd.Filter.UnFilter()	撤销筛选
Syscmd.Filter.StartsWith()	字符开头是
Syscmd.Filter.NotStartsWith()	字符开头不是
Syscmd.Filter.EndsWith()	字符结尾是
Syscmd.Filter.NotEndsWith()	字符结尾不是
Syscmd.Filter.Contains()	包含
Syscmd.Filter.NotContains()	不包含
Syscmd.Filter.Like()	类似
Syscmd.Filter.NotLike	不类似
Syscmd.Filter.Between()	范围之内
Syscmd.Filter.NotBetween()	范围之外
Syscmd.Sort.Ascending()	升序排序
Syscmd.Sort.Descending()	降序排序

Syscmd.Sort.UnSort()	取消排序

导入导出

命令	说明
Syscmd.Table.ExportToText()	导出为文本文件
Syscmd.Table.ExportToXML()	导出为 XML 文件
Syscmd.Table.Merge()	合并表
Syscmd.Table.QuickMerge()	高速合并
Syscmd.Table.MergeFromXML()	合并 XML 文件
Syscmd.Table.MergeFromText()	合并文本文件
Syscmd.Project.Import()	导入数据
Syscmd.Project.QuickImport()	高速导入
Syscmd.Project.QuickExport()	高速导出
Syscmd.Project.ImportFromXML()	导入 XML 文件
Syscmd.Project.ImportFromText()	导入文本文件
Syscmd.Project.ExportToAccess()	导出为 Access 文件
Syscmd.Project.ExportToExcel()	导出为 Excel 文件

报表相关

命令	说明
Syscmd.Table.ExcelReportManager()	Excel 报表模版管理
Syscmd.Project.NewExcelTemplate()	新建 Excel 报表模板
Syscmd.Project.OpenExcelTemplate()	打开 Excel 报表模板
Syscmd.Project.ExcelVBCode()	设置 Excel 报表事件代码
Syscmd.Table.WordReportManager()	Word 报表模板管理
Syscmd.Project.NewWordTemplate()	新建 Word 报表模板
Syscmd.Project.OpenWordTemplate()	打开 Word 报表模板
Syscmd.Table.CodeReportManager()	专业报表管理

项目相关

命令	说明
Syscmd.Project.Open()	打开项目
Syscmd.Project.Open(FileName)	打开指定的项目
Syscmd.Project.Create()	创建新的项目
Syscmd.Project.ModifyProperty()	设置项目属性
Syscmd.Project.Compact()	压缩项目
Syscmd.Project.Compact(False)	压缩项目，压缩完成后不出现提示。

Syscmd.Project.Backup(Path,Auto)	备份项目 Path: 可选参数，指定备份文件存储路径 Auto: 可选参数，设为 True 不出现备份窗口，自动完成备份。
Syscmd.Project.Save()	保存项目
Syscmd.Project.SaveAs()	另存项目
Syscmd.Project.Exit()	正常退出 Foxtable
Syscmd.Project.Exit(True)	先保存数据，然后退出 Foxtable
Syscmd.Project.Exit(False)	不保存数据，强行退出 Foxtable
Syscmd.Project.DesignRibbonMenu()	设计菜单
Syscmd.Project.ModifyOuterDataSource()	外部数据源管理
Syscmd.Project.Users()	用户管理
Syscmd.Project.SwitchUser()	切换用户
Syscmd.Project.CommandWindow()	打开命令窗口
Syscmd.Project.Load()	重新加载所有表(同步所有表)
Syscmd.Project.EditMyTimers()	计划管理
Syscmd.Project.FormDesigner()	窗口管理
Syscmd.Project.Initialize()	初始化项目
Syscmd.Project.ExecuteSQL()	执行 SQL 语句
Syscmd.Project.FunctionManager()	内部函数定义
Syscmd.Project.Variable()	定义 Var 变量
Syscmd.Project.VBCode()	定义全局代码
Syscmd.Project.Refrence()	定义外部引用
Syscmd.Project.SetFileWatcher	设置文件监视器
Syscmd.Project.SetPortWatcher()	设置串口监视器
Syscmd.Project.ViewCode()	输出代码
Syscmd.Project.ViewStructure()	输出结构
Syscmd.Project.ToAccess()	ToAccess(生成一个包含所有内部表的 Access 文件)
Syscmd.Project.Redirect()	Redirect()
Syscmd.Project.EditTree()	编辑目录树
Syscmd.Project.OpenSystemMenu()	打开系统菜单
Syscmd.Project.OpenUserMenu()	打开用户菜单

9.2 中英文颜色对照表

颜色	名称	中文说明	RGB 格式
	LightPink	浅粉红	255,182,193
	Pink	粉红	255,192,203
	Crimson	猩红	220,20,60
	LavenderBlush	脸红的淡紫色	255,240,245
	PaleVioletRed	苍白的紫罗兰红色	219,112,147
	HotPink	热情的粉红	255,105,180
	DeepPink	深粉色	255,20,147
	MediumVioletRed	适中的紫罗兰红色	199,21,133
	Orchid	兰花的紫色	218,112,214
	Thistle	蓟	216,191,216
	plum	李子	221,160,221
	Violet	紫罗兰	238,130,238
	Magenta	洋红	255,0,255
	Fuchsia	灯笼海棠(紫红色)	255,0,255
	DarkMagenta	深洋红色	139,0,139
	Purple	紫色	128,0,128
	MediumOrchid	适中的兰花紫	186,85,211
	DarkViolet	深紫罗兰色	148,0,211
	DarkOrchid	深兰花紫	153,50,204
	Indigo	靛青	75,0,130
	BlueViolet	深紫罗兰的蓝色	138,43,226
	MediumPurple	适中的紫色	147,112,219
	MediumSlateBlue	适中的板岩暗蓝灰色	123,104,238
	SlateBlue	板岩暗蓝灰色	106,90,205
	DarkSlateBlue	深岩暗蓝灰色	72,61,139
	Lavender	熏衣草花的淡紫色	230,230,250
	GhostWhite	幽灵的白色	248,248,255
	Blue	纯蓝	0,0,255
	MediumBlue	适中的蓝色	0,0,205
	MidnightBlue	午夜的蓝色	25,25,112
	DarkBlue	深蓝色	0,0,139
	Navy	海军蓝	0,0,128
	RoyalBlue	皇军蓝	65,105,225
	CornflowerBlue	矢车菊的蓝色	100,149,237
	LightSteelBlue	淡钢蓝	176,196,222
	LightSlateGray	浅石板灰	119,136,153
	SlateGray	石板灰	112,128,144

	DodgerBlue	道奇蓝	30,144,255
	AliceBlue	爱丽丝蓝	240,248,255
	SteelBlue	钢蓝	70,130,180
	LightSkyBlue	淡蓝色	135,206,250
	SkyBlue	天蓝色	135,206,235
	DeepSkyBlue	深天蓝	0,191,255
	LightBLue	淡蓝	173,216,230
	PowDerBlue	火药蓝	176,224,230
	CadetBlue	军校蓝	95,158,160
	Azure	蔚蓝色	240,255,255
	LightCyan	淡青色	225,255,255
	PaleTurquoise	苍白的绿宝石	175,238,238
	Cyan	青色	0,255,255
	Aqua	水绿色	0,255,255
	DarkTurquoise	深绿宝石	0,206,209
	DarkSlateGray	深石板灰	47,79,79
	DarkCyan	深青色	0,139,139
	Teal	水鸭色	0,128,128
	MediumTurquoise	适中的绿宝石	72,209,204
	LightSeaGreen	浅海洋绿	32,178,170
	Turquoise	绿宝石	64,224,208
	Aquamarine	绿玉\碧绿色	127,255,170
	MediumAquamarine	适中的碧绿色	0,250,154
	MediumSpringGreen	适中的春天的绿色	245,255,250
	MintCream	薄荷奶油	0,255,127
	SpringGreen	春天的绿色	60,179,113
	SeaGreen	海洋绿	46,139,87
	Honeydew	蜂蜜	240,255,240
	LightGreen	淡绿色	144,238,144
	PaleGreen	苍白的绿色	152,251,152
	DarkSeaGreen	深海洋绿	143,188,143
	LimeGreen	酸橙绿	50,205,50
	Lime	酸橙色	0,255,0
	ForestGreen	森林绿	34,139,34
	Green	纯绿	0,128,0
	DarkGreen	深绿色	0,100,0
	Chartreuse	查特酒绿	127,255,0
	LawnGreen	草坪绿	124,252,0
	GreenYellow	绿黄色	173,255,47
	OliveDrab	橄榄土褐色	85,107,47
	Beige	米色(浅褐色)	107,142,35
	LightGoldenrodYellow	浅秋麒麟黄	250,250,210

	Ivory	象牙	255,255,240
	LightYellow	浅黄色	255,255,224
	Yellow	纯黄	255,255,0
	Olive	橄榄	128,128,0
	DarkKhaki	深卡其布	189,183,107
	LemonChiffon	柠檬薄纱	255,250,205
	PaleGoldenrod	灰秋麒麟	238,232,170
	Khaki	卡其布	240,230,140
	Gold	金	255,215,0
	Cornsilk	玉米色	255,248,220
	GoldEnrod	秋麒麟	218,165,32
	FloralWhite	花的白色	255,250,240
	OldLace	老饰带	253,245,230
	Wheat	小麦色	245,222,179
	Moccasin	鹿皮鞋	255,228,181
	Orange	橙色	255,165,0
	PapayaWhip	番木瓜	255,239,213
	BlanchedAlmond	漂白的杏仁	255,235,205
	NavajoWhite	Navajo 白	255,222,173
	AntiqueWhite	古代的白色	250,235,215
	Tan	晒黑	210,180,140
	BurlyWood	结实的树	222,184,135
	Bisque	(浓汤)乳脂,番茄等	255,228,196
	DarkOrange	深橙色	255,140,0
	Linen	亚麻布	250,240,230
	Peru	秘鲁	205,133,63
	PeachPuff	桃色	255,218,185
	SandyBrown	沙棕色	244,164,96
	Chocolate	巧克力	210,105,30
	SaddleBrown	马鞍棕色	139,69,19
	SeaShell	海贝壳	255,245,238
	Sienna	黄土赭色	160,82,45
	LightSalmon	浅鲜肉(鲑鱼)色	255,160,122
	Coral	珊瑚	255,127,80
	OrangeRed	橙红色	255,69,0
	DarkSalmon	深鲜肉(鲑鱼)色	233,150,122
	Tomato	番茄	255,99,71
	MistyRose	薄雾玫瑰	255,228,225
	Salmon	鲜肉(鲑鱼)色	250,128,114
	Snow	雪	255,250,250
	LightCoral	淡珊瑚色	240,128,128
	RosyBrown	玫瑰棕色	188,143,143

	IndianRed	印度红	205,92,92
	Red	纯红	255,0,0
	Brown	棕色	165,42,42
	FireBrick	耐火砖	178,34,34
	DarkRed	深红色	139,0,0
	Maroon	栗色	128,0,0
	White	纯白	255,255,255
	WhiteSmoke	白烟	245,245,245
	Gainsboro	Gainsboro	220,220,220
	LightGray	浅灰色	211,211,211
	Silver	银白色	192,192,192
	DarkGray	深灰色	169,169,169
	Gray	灰色	128,128,128
	DimGray	暗淡的灰色	105,105,105
	Black	纯黑	0,0,0

9.3 纸张编号与类型表

纸张类型	
1	Letter
2	LetterSmall
3	Tabloid
4	Ledger
5	Legal
6	Statement
7	Executive
8	A3
9	A4
10	A4Small
11	A5
12	B4
13	B5
14	Folio
15	Quarto

16	Standard10x14
17	Standard11x17
18	Note
19	Number9Envelope
20	Number10Envelope
21	Number11Envelope
22	Number12Envelope
23	Number14Envelope
24	CSheet
25	DSheet
26	ESheet
27	DLEnvelope
28	C5Envelope
29	C3Envelope
30	C4Envelope
31	C6Envelope
32	C65Envelope
33	B4Envelope
34	B5Envelope
35	B6Envelope
36	ItalyEnvelope
37	MonarchEnvelope
38	PersonalEnvelope
39	USStandardFanfold
40	GermanStandardFanfold
41	GermanLegalFanfold
42	ISOB4
43	JapanesePostcard
44	Standard9x11
45	Standard10x11
46	Standard15x11
47	InviteEnvelope

50	LetterExtra
51	LegalExtra
52	TabloidExtra
53	A4Extra
54	LetterTransverse
55	A4Transverse
56	LetterExtraTransverse
57	APlus
58	BPlus
59	LetterPlus
60	A4Plus
61	A5Transverse
62	B5Transverse
63	A3Extra
64	A5Extra
65	B5Extra
66	A2
67	A3Transverse
68	A3ExtraTransverse
69	**JapanesePostcard**
70	A6
71	JapaneseEnvelopeKakuNumber2
72	JapaneseEnvelopeKakuNumber3
73	JapaneseEnvelopeChouNumber3
74	JapaneseEnvelopeChouNumber4
75	LetterRotated
76	A3Rotated
77	A4Rotated
78	A5Rotated
79	B4JisRotated
80	B5JisRotated
81	JapanesePostcardRotated

82	JapaneseDoublePostcardRotated
83	A6Rotated
84	JapaneseEnvelopeKakuNumber2Rotated
85	JapaneseEnvelopeChouNumber3Rotated
86	JapaneseEnvelopeChouNumber4Rotated
87	**JapaneseEnvelopeChouNumber4Rotated**
88	B6Jis
89	B6JisRotated
90	Standard12x11
91	JapaneseEnvelopeYouNumber4
92	JapaneseEnvelopeYouNumber4Rotated
93	Prc16K
94	Prc32K
95	Prc32KBig
96	PrcEnvelopeNumber1
97	PrcEnvelopeNumber2
98	PrcEnvelopeNumber3
99	PrcEnvelopeNumber4
100	PrcEnvelopeNumber5
101	PrcEnvelopeNumber6
102	PrcEnvelopeNumber7
103	PrcEnvelopeNumber8
104	PrcEnvelopeNumber9
105	PrcEnvelopeNumber10
106	Prc16KRotated
107	Prc32KRotated
108	Prc32KBigRotated
109	PrcEnvelopeNumber1Rotated
110	PrcEnvelopeNumber2Rotated
111	PrcEnvelopeNumber3Rotated
112	PrcEnvelopeNumber4Rotated
113	PrcEnvelopeNumber5Rotated

114	PrcEnvelopeNumber6Rotated
115	PrcEnvelopeNumber7Rotated
116	PrcEnvelopeNumber8Rotated
117	PrcEnvelopeNumber9Rotated
118	PrcEnvelopeNumber10Rotated

9.4 Keys 枚举

成员名称	说明
A	A 键。
Add	加号键。
Alt	Alt 修改键。
Apps	应用程序键（Microsoft Natural Keyboard，人体工程学键盘）。
Attn	Attn 键。
B	B 键。
Back	Backspace 键。
BrowserBack	浏览器后退键（Windows 2000 或更高版本）。
BrowserFavorites	浏览器收藏夹键（Windows 2000 或更高版本）。
BrowserForward	浏览器前进键（Windows 2000 或更高版本）。
BrowserHome	浏览器主页键（Windows 2000 或更高版本）。
BrowserRefresh	浏览器刷新键（Windows 2000 或更高版本）。
BrowserSearch	浏览器搜索键（Windows 2000 或更高版本）。
BrowserStop	浏览器停止键（Windows 2000 或更高版本）。
C	C 键。
Cancel	Cancel 键。
Capital	Caps Lock 键。
CapsLock	Caps Lock 键。
Clear	Clear 键。
Control	Ctrl 修改键。
ControlKey	Ctrl 键。
Crsel	Crsel 键。

D	D 键。
D0	0 键。
D1	1 键。
D2	2 键。
D3	3 键。
D4	4 键。
D5	5 键。
D6	6 键。
D7	7 键。
D8	8 键。
D9	9 键。
Decimal	句点键。
Delete	DeL 键。
Divide	除号键。
Down	向下键。
E	E 键。
End	End 键。
Enter	Enter 键。
EraseEof	ERASE EOF 键。
Escape	Esc 键。
Execute	EXECUTE 键。
Exsel	Exsel 键。
F	F 键。
F1	F1 键。
F10	F10 键。
F11	F11 键。
F12	F12 键。
F13	F13 键。
F14	F14 键。
F15	F15 键。
F16	F16 键。
F17	F17 键。

F18	F18 键。
F19	F19 键。
F2	F2 键。
F20	F20 键。
F21	F21 键。
F22	F22 键。
F23	F23 键。
F24	F24 键。
F3	F3 键。
F4	F4 键。
F5	F5 键。
F6	F6 键。
F7	F7 键。
F8	F8 键。
F9	F9 键。
FinalMode	IME 最终模式键。
G	G 键。
H	H 键。
HanguelMode	IME Hanguel 模式键。（为了保持兼容性而设置；使用 **HangulMode**）
HangulMode	IME Hangul 模式键。
HanjaMode	IME Hanja 模式键。
Help	Help 键。
Home	Home 键。
I	I 键。
IMEAccept	IME 接受键，替换 **IMEAceept**。
IMEAceept	IME 接受键。已过时，请改用 **IMEAccept**。
IMEConvert	IME 转换键。
IMEModeChange	IME 模式更改键。
IMENonconvert	IME 非转换键。
Insert	Ins 键。
J	J 键。

JunjaMode	IME Junja 模式键。
K	K 键。
KanaMode	IME Kana 模式键。
KanjiMode	IME Kanji 模式键。
KeyCode	从键值提取键代码的位屏蔽。
L	L 键。
LaunchApplication1	启动应用程序一键（Windows 2000 或更高版本）。
LaunchApplication2	启动应用程序二键（Windows 2000 或更高版本）。
LaunchMail	启动邮件键（Windows 2000 或更高版本）。
LButton	鼠标左按钮。
LControlKey	左 Ctrl 键。
Left	向左键。
LineFeed	LINEFEED 键。
LMenu	左 Alt 键。
LShiftKey	左 Shift 键。
LWin	左 Windows 徽标键（Microsoft Natural Keyboard，人体工程学键盘）。
M	M 键。
MButton	鼠标中按钮（三个按钮的鼠标）。
MediaNextTrack	媒体下一曲目键（Windows 2000 或更高版本）。
MediaPlayPause	媒体播放暂停键（Windows 2000 或更高版本）。
MediaPreviousTrack	媒体上一曲目键（Windows 2000 或更高版本）。
MediaStop	媒体停止键（Windows 2000 或更高版本）。
Menu	Alt 键。
Modifiers	从键值提取修饰符的位屏蔽。
Multiply	乘号键。
N	N 键。
Next	Page Down 键。
NoName	保留以备将来使用的常数。
None	没有按任何键。
NumLock	Num Lock 键。
NumPad0	数字键盘上的 0 键。

NumPad1	数字键盘上的 1 键。
NumPad2	数字键盘上的 2 键。
NumPad3	数字键盘上的 3 键。
NumPad4	数字键盘上的 4 键。
NumPad5	数字键盘上的 5 键。
NumPad6	数字键盘上的 6 键。
NumPad7	数字键盘上的 7 键。
NumPad8	数字键盘上的 8 键。
NumPad9	数字键盘上的 9 键。
O	O 键。
Oem1	OEM 1 键。
Oem102	OEM 102 键。
Oem2	OEM 2 键。
Oem3	OEM 3 键。
Oem4	OEM 4 键。
Oem5	OEM 5 键。
Oem6	OEM 6 键。
Oem7	OEM 7 键。
Oem8	OEM 8 键。
OemBackslash	RT 102 键的键盘上的 OEM 尖括号或反斜杠键（Windows 2000 或更高版本）。
OemClear	Clear 键。
OemCloseBrackets	美式标准键盘上的 OEM 右括号键（Windows 2000 或更高版本）。
Oemcomma	任何国家/地区键盘上的 OEM 逗号键（Windows 2000 或更高版本）。
OemMinus	任何国家/地区键盘上的 OEM 减号键（Windows 2000 或更高版本）。
OemOpenBrackets	美式标准键盘上的 OEM 左括号键（Windows 2000 或更高版本）。
OemPeriod	任何国家/地区键盘上的 OEM 句点键（Windows 2000 或更高版本）。
OemPipe	美式标准键盘上的 OEM 管道键（Windows 2000 或更高版本）。
Oemplus	任何国家/地区键盘上的 OEM 加号键（Windows 2000 或更高版

	本）。
OemQuestion	美式标准键盘上的 OEM 问号键（Windows 2000 或更高版本）。
OemQuotes	美式标准键盘上的 OEM 单/双引号键（Windows 2000 或更高版本）。
OemSemicolon	美式标准键盘上的 OEM 分号键（Windows 2000 或更高版本）。
Oemtilde	美式标准键盘上的 OEM 波形符键（Windows 2000 或更高版本）。
P	P 键。
Pa1	PA1 键。
Packet	用于将 Unicode 字符当作键击传递。Packet 键值是用于非键盘输入法的 32 位虚拟键值的低位字。
PageDown	Page Down 键。
PageUp	Page Up 键。
Pause	Pause 键。
Play	Play 键。
Print	Print 键。
PrintScreen	Print Screen 键。
Prior	Page Up 键。
ProcessKey	Process Key 键。
Q	Q 键。
R	R 键。
RButton	鼠标右按钮。
RControlKey	右 Ctrl 键。
Return	Return 键。
Right	向右键。
RMenu	右 Alt 键。
RShiftKey	右 Shift 键。
RWin	右 Windows 徽标键（Microsoft Natural Keyboard，人体工程学键盘）。
S	S 键。
Scroll	Scroll Lock 键。
Select	Select 键。
SelectMedia	选择媒体键（Windows 2000 或更高版本）。

Separator	分隔符键。
Shift	Shift 修改键。
ShiftKey	Shift 键。
Sleep	计算机睡眠键。
Snapshot	Print Screen 键。
Space	空格键。
Subtract	减号键。
T	T 键。
Tab	Tab 键。
U	U 键。
Up	向上键。
V	V 键。
VolumeDown	减小音量键（Windows 2000 或更高版本）。
VolumeMute	静音键（Windows 2000 或更高版本）。
VolumeUp	增大音量键（Windows 2000 或更高版本）。
W	W 键。
X	X 键。
XButton1	第一个 X 鼠标按钮（五个按钮的鼠标）。
XButton2	第二个 X 鼠标按钮（五个按钮的鼠标）。
Y	Y 键。
Z	Z 键。
Zoom	Zoom 键。

9.5 VBScript 参考

运算符

运算符	说明
+	加
-	减
*	乘
/	除

\	整除，只返回结果的整数部分
^	指数运算，例如：2^3，表示 2 的 3 次方，等于 8
Mod	取模，例如：10 Mod 3，表示 10 除以 3 的余数，等于 1
&	字符串连接，例如："A" & "B"，等于"AB"
And	逻辑与
Or	逻辑或
Xor	逻辑异或
Not	逻辑非

函数

Abs(number)
返回绝对值。

Asc(string)
返回字符串第一个字符的 ANSI 码。

Atn(number)
返回反正弦值。
Cos(number)
返回数 number 的余弦值。

CBool(expression)
转换成 Boolean 数据类型。

CByte(expression)
转换成 Byte 数据类型。

CCur(expression)
转换成 Currency 数据类型。

CDate(expression)
转换成 Date 数据类型。

CDbl(expression)
转换成 Double 数据类型。

Chr(charcode)
把 ANSI 码转换成相应的字符。

CInt(expression)
转换成 Integer 数据类型。

CLng(expression)
转换成 Long 数据类型。

CSng(expression)
转换成 Single 数据类型。

CStr(expression)
转换成 String 数据类型。

Date
返回系统的当天日期。

DateAdd(interval,number,date)
向 date 值添加指定的时间间隔。

DateDiff(interval,date1,date2)
返回两个日期之间的间隔。
Interval 包含以下设置：
yyyy 年
q 季度
m 月
y 某年的某一天
d 天
w 工作日
ww 周
h 时
n 分
s 秒
DatePart(interval , date)
返回 Date 对象指定部分。
interval 的设置和 DateDiff 相同。
DateSerial(year,month,day)
转换成 Date 数据子类型变体值。

DateValue(date)
去掉 Date 值的时间部分。

Day(date)
根据参数 date 返回该天在一个月中的号数。

Exp(number)

返回 e 的 number 次方的值。

Fix(number)
转换成一个整数（对于负数，只入不舍）。

Hex(number)
返回数 number 的十六进制的值。

Hour(time)
根据参数 time 的值返回小时。
IIF(expression, truevalue, falsevalue)
如果逻辑表达式 expression 的计算结果为 True，将返回 truevalue，否则返回 falsevalue。
InStr(string1,string2)
返回字符串 string2 在字符串 string1 中第一次出现的位置。

InStrRev(string1,string2)
返回字符串 string2 在字符串 string1 中最后出现的位置。

Int(number)
返回一个整数(负数只舍不入)。

IsDate(expression)
确定表达式 expression 能否转换成日期格式。

IsEmpty(varname)
确定变量是否已经初始化了。

IsNull(expression)
确定表达式 expression 是否为空。

IsNumeric(expression)
确定表达式 expression 的结果是否是一个数。

Lcase(string)
以小写字符格式化。

Left(string,length)
返回指定长度的字符串 string 左部分。

Len(string)

返回字符串 string 的长度。

Log(number)

返回数 number 的自然对数值。

Ltrim(string)
删除字符串 string 左边多余的空格。

Mid(strin,start[,length])
返回指定长度的字符串 string 的子串。

Minute(time)
根据参数 time 的值返回分钟的值。

Month(date)
返回以数字表示的月份。

MonthName(month)
返回以名称表示的月份。

Now
根据系统，返回当前的日期和时间。

Oct(number)
返回数 number 的八进制值。

Replace(expression,find,replacewith)
用指定的子字符串 replacewith 替换指定的子字符串 find。
Right(string,length)
返回指定长度的字符串 string 的右部分子串。

Rnd([number])
产生一个随机数。

Round(number,decimals)
对 number 进行四舍五入,decimals 为要保留的小数位数。

RTrim(string)
删除字符串 string 右边多余的空格。

Second(time)
根据参数 time 的值返回该分的第几秒。

Sgn(number)
返回数 number 的正负号。

Sin(number)

返回数 number 的正弦值。

Space(number)
用指定数量的空格创建字符串。

Sqr(number)
返回数 number 的平方根值。

StrReverse(string1)
对字符串 string1 进行倒序。

String(number,character)
用 number 个字符 character 创建字符串。

Tan(number)
返回数 number 的正切。

Time
根据系统返回当前时间。

TimeSerial(hour,minute,second)
返回给定时分秒的时间值。
TimeValue(Date)
去掉 Date 值的日期部分。

Trim(string)
删除字符串 string 两边的多余空格。

Ucase(string)
用大写字母格式格式化字符串 string。

Weekday(date)
用数字形式返回星期几。
WeekDayName(weekday)
用名称形式返回星期几。
Year(date)
根据参数 date 返回年份。
CUDate(Date)
将日期 Date 转换为大写中文格式
例如：CUDate(#12/31/2008#)
等于：贰零零捌年拾贰月叁拾壹日
CLDate(Date)
将日期 Date 转换为小写中文格式
例如：CLDate(#12/31/2008#)

等于：二○○八年十二月三十一日
CCDate(Date)
将日期 Date 转换为大写中文格式，符合支票格式
例如：CUDate(#2/10/2008#)
等于：贰零零捌年零贰月壹拾日
GetDigit(Value,Position,Symbol)
返回数值 Value 中指定位置的数字，参数 Symbol 指定货币符号。
参数 Position 指定位置，0 表示个位,1 表示十位,2 表示百位,-1 表示小数点后第一位,-2 表示小数点后第二位,其余类推。
例如：
GetDigit(123,0,"￥") '等于"3"
GetDigit(123,1,"￥") '等于"2"
GetDigit(123,2,"￥") '等于"1"
GetDigit(123,3,"￥") '等于"￥"
GetDigit(123,3,"$") '等于"$"
GetDigit(1972.63,-1,"￥") '等于"6"
GetDigit(1972.63,-2,"￥") '等于"3"
CUMoney(Value)
将数值转换为中文货币格式。
例如：CUMoney(12345.12)
等于：壹万贰千叁佰肆拾伍圆壹角贰分
CUNumber(Value)
将数值转换为中文大写格式。
例如：CUNumber(123.45)
等于：壹佰贰拾叁点肆伍
CUNum(Value)
将阿拉伯数字转换为中文大写格式，只能处理整数。
例如：CUNum(123)
等于：壹贰叁
CLNum(Value)
将阿拉伯数字转换为中文小写格式，只能处理整数。
例如：CLNum(123)
等于：一二三
CCNumber(Value)
将阿拉伯数字转换为中文格式，只能处理整数，符合支票格式
例如：CUNumber(1)
等于：零壹
EUNumber(Value)
将整数转换为英文大写。
例如：EUNumber(12345)
等于：Twelve Thousand Three Hundred Forty Five

EUMoney(Value)
将数值转换为英文货币格式。

例如：EUMoney(12345.45)

等于：Twelve Thousand Three Hundred Forty Five Dollars And Forty Five Cents

Format

格式化数字和日期，VBScript 中的 Format 函数和 Foxtable 自身的 Format 行数相比，功能相对较弱，只能使用下面所示的格式：

字符	说明
0	数字占位符。显示一个数字或 0。如果表达式在格式字符串中出现 0 的位置上有数字，则显示该数字；否则在该位置显示 0。 如果数字的位数少于格式表达式中 0 的个数（小数点任一侧），则显示前导零或尾随零。如果数字的小数点分隔符右侧的位数多于格式表达式中小数点分隔符右侧零的个数，则将数字舍入到与零的个数相同的小数位置。如果数字的小数点分隔符左侧的位数多于格式表达式中小数点分隔符左侧零的个数，则不做任何修改地显示额外的数字。 下面是一些例子，每行代码之后的注释是格式化后的结果。 Format(12,"000") '012 Format(12,"0") '12 Format(12,"0.00") '12.00 Format(12,"00.00") '12.00 Format(0.5,"0") '1 Format(0.5,"0.0") '0.5 Format(10.56,"0.0") '10.6 Format(10.56,"00.00") '10.56
.	小数点占位符。 下面是一些例子，每行代码之后的注释是格式化后的结果。 Format(0.568,"0.00") '0.57 Format(0.56,"0.00") '0.56 Format(0.5,"0.00") '0.50
%	百分比占位符。用 100 乘以表达式。在格式字符串中出现百分比占位符的位置插入百分比字符 (%)。 例如： Format(0.56578,"0.00%") '56.68% Format(0.5657,"0.00%") '56.57% Format(0.565,"0.00%") '56.50% Format(0.56,"0.00%") ' 56.00%
yy	以带前导零的两位数字格式显示年份。
yyyy	以四位数字格式显示年份. 例如: Format(#12/31/2008#, "yy-MM-dd") '08-12-31

	Format(#12/31/2013#, "yy年MM月dd日") '13年29月31日 Format(#12/31/2008#, "yyyy-MM-dd") '2008-12-31 Format(#12/31/2008#, "yyyy年MM月dd日") '2008年12月31日
M	将月份显示为不带前导零的数字。
MM	将月份显示为带前导零的数字。 例如： Format(#2/28/2008#, "yyyy-MM-dd") '2008-02-28 Format(#2/28/2008#, "yyyy-M-dd") '2008-2-28
d	将日显示为不带前导零的数字（例如 1）。
dd	将日显示为带前导零的数字（例如 01）。 例如: Format(#2/8/2008#, "yyyy-MM-dd") '2008-02-08 Format(#2/8/2008#, "yyyy-M-d") '2008-2-8
h	使用 12 小时制将小时显示为不带前导零的数字（例如 1:15:15 PM）。
hh	使用 12 小时制将小时显示为带前导零的数字（例如 01:15:15 PM）。 例如： Format(#2:3:9#, "h:m:s") '2:3:9 Format(#2:3:9#, "hh:mm:ss") '02:03:09 Format(#14:3:9#, "h:m:s") '2:3:9 Format(#14:3:9#, "hh:mm:ss") '02:03:09
H	使用 24 小时制将小时显示为不带前导零的数字（例如 1:15:15）。
HH	使用 24 小时制将小时显示为带前导零的数字（例如 01:15:15）。 例如： Format(#2:3:9#, "HH:m:s") '2:3:9 Format(#2:3:9#, "HH:mm:ss") '02:03:09 Format(#14:3:9#, "HH:m:s") '14:3:9 Format(#14:3:9#, "HH:mm:ss") '14:03:09
m	将分钟显示为不带前导零的数字（例如 12:1:15）。
mm	将分钟显示为带前导零的数字（例如 12:01:15）。
s	将秒显示为不带前导零的数字（例如 12:15:5）。
ss	将秒显示为带前导零的数字（例如 12:15:05）。

9.6 Excel 与 VBA

如果你是 Excel 的资深用户，一定会留恋 VBA，实际上你的 VBA 知识在 Foxtable 一样有用，原来的 VBA 代码只需稍作修改，就可以在 Foxtable 中使用。
Foxtable 本身已经提供了很强大的 Excel 报表功能，可以利用报表模板来批量生成各种复杂的报表，所以本节的内容对于一般用户来说并无意义。
提示：
1、Excel 库在 Foxtable 中的命名空间为 MSExcel，所以代码中 Excel 的类型和函数都必须加上前缀 MSExcel，例如：MSExcel.Application、MSExcel.WorkBook。
2、Excel 与 VBA 的内容非常多，可以用一本厚厚的书讲述，我们不可能一一详述，本节的主要目的在于介绍其最基本的类型，以及如何直接在 Foxtable 中操控 Excel。

Application

Application 对象代表整个 Microsoft Excel 应用程序,它处于 Excel 对象的最高层次。
常用属性

- Visible

  隐藏 Excel，就是将 Application 对象的 Visible 属性设置为 False，从而使 Excel 本身不显示出来。如果要显示 Excel，可将其显示为 True。

  例如在命令窗口执行下面的代码，就将自动打开并可见 Excel 程序：

  ```
  Dim App As New MSExcel.Application
  App.Visible = True
  ```

- Caption

  默认情况下，Excel 窗口标题就是所打开文件的文件名。
  利用 Caption 属性可修改标题显示格式，例如：

  ```
  Dim App As New MSExcel.Application
  App.Visible = True
  App.Caption = "标题修改试验"
  ```

- DisplayAlerts
  是否显示警告信息对话框。比如，在合并单元格时，将默认弹出一个确认窗口；将 DisplayAlerts 设为 False 后，将直接合并而不再弹出对话框。

常用方法

- **Quit**

  关闭 Excel 应用程序，如：App.Quit

重要说明：只要定义了 Application，Excel 程序就会被打开。如果没有将其 Visible 属性设为 True，这个 Excel 程序就会一直停留在进程中。为避免这种现象，一定要在代码的最后一行加上 App.Quit，如果已经将 Visible 设为 True，就不用加 App.Quit。因为 Excel 程序在可见状态下，可直接手工关闭退出。

WorkBook

WorkBook 对象代表 Microsoft Excel 工作簿，它在 Application 对象的下一层。WorkBook 是 WorkBooks 集合的成员。

常用方法

- Add

  创建新的工作簿。如下面的代码，就将新建一个工作簿，该工作簿的第一个工作表的名称改为 hhh，同时在 A1 单元格写入“hhh123”，然后保存为 D:\123.xls。

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.Workbook = App.WorkBooks.Add
Wb.WorkSheets(1).name = "hhh"
Wb.WorkSheets(1).range("A1").Value = "hhh123"
Wb.SaveAs("d:\123.xls")
App.Quit
```

  需要说明的是，在创建新工作簿后，该工作簿就成为活动工作簿。

- Open

  打开现有工作簿。如：

```
Dim App As New MSExcel.Application
App.Visible = True
Dim Wb As MSExcel.Workbook = App.WorkBooks.Open("D:\report.xls")
```

- Save

  保存工作簿，如：Wb.Save

- SaveAs

  另存工作簿，如：Wb.SaveAs("D:\123.xls")

- Close

  关闭工作簿，如：Wb.Close
  注意，这种方法只是关闭了工作簿，并没有关闭 Microsoft Excel 应用程序。如果要在关闭工作簿的同时关闭 Excel 程序，可以使用 Application 对象的 Quit 方法。

WorkSheet

WorkSheet 代表一张工作表，它是 WorkBook 的下一级对象。WorkSheet 是 WorkSheets 集合的成员。

可以通过位置或者工作表名称来引用工作表，例如引用第一个工作表：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
```

如要引用最后一个表，可以这样：

```
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(Wb.WorkSheets.Count)
```

通过 WorkSheets 的名称引用。如：

```
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets("同期对比")
```

需要注意的是，和 Foxtable 不同，这里的编号是从 1 开始的，而 Foxtable 的所有集合，包括数组，编号都是从 0 开始的。
主要属性

- Name

  返回或设置工作表名称。
  例如显示所有工作表名称：

  ```
  Dim App As New MSExcel.Application
  Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
  For Each Ws As MSExcel.WorkSheet In Wb.WorkSheets
      MessageBox.Show(Ws.Name)
  Next
  App.Quit
  ```

  再例如，将第二个工作表的名称修改为“hhh”：

  ```
  Dim App As New MSExcel.Application
  Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
  Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
  Ws.Name = "hhh"
  Wb.Save
  ```

```
App.Quit
```

- Visible

  返回或设置工作表的显示状态。如：Ws.Visible = False

主要方法

- Add

  新建工作表。

  新建工作表有两个可选参数：Before 和 After，分别指定要增加的位置。如，下面的代码就是在“同期对比”表的后面，增加一个工作表，然后将表名称命名为“kkk”：

```
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets.Add(After:=Wb.WorkSheets("同期对比"))
Ws.Name = "kkk"
```

  如果要在“同期对比”表的前面增加,只需将 After 改为 Before。

- Copy

  复制工作表。

  复制工作表也可用 Before 和 After 指定复制位置。如不指定，将新建一个只包含指定工作表的工作簿。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets("同期对比") '指定要复制的工作表
With Wb.WorkSheets
   Ws.Copy(After:=.Item(.Count))'复制到最后面
   Ws.Copy(Before:=.Item(1))'复制到最前面
   Ws.Copy  '复制到一个新工作簿
End With
Wb.Save
App.Quit
```

- Move

  移动工作表。
  移动工作表可用 Before 和 After 指定移动位置。如不指定，将新建一个只包含指定工作表的工作簿。需要注意的是，当工作簿中只有一张工作表时是不能移动的。

- Delete

  删除工作表。
  在删除工作表时会弹出提示信息，因此可以将 Application 的 DisplayAlerts 属性设置为 False 来阻止信息框的显示。需要注意的是，当工作簿中只有一张工作表时是不能删除的。

- Protect

  保护工作表。
  Protect 方法可以指定保护密码参数，该密码为一个区分大小写的字符串。如果省略本参数，不用密码就可以取消对该工作表的保护。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets("同期对比")
Ws.Protect(Password:="hhh")  '以密码保护
Wb.Save
App.Quit
```

  如果直接写为：Ws.Protect，则无需密码就能取消保护。

  除此以外，Protect 中还可以使用以下参数：

  DrawingObjects：是否保护图表。如：Ws.Protect(DrawingObjects:=True)
  Contents：是否保护单元格内容。
  Scenarios：是否保护方案。
  UserInterfaceonly：是否保护用户界面，但不保护宏。如果省略本参数，则保护既应用于宏也应用于用户界面。
  AllowFormattingCells：是否允许用户为受保护的工作表上的任意单元格设置格式。
  AllowFormattingColumns：是否允许用户为受保护的工作表上的任意列设置格式。
  AllowFormattingRows：是否允许用户为受保护的工作表上的任意行设置格式。
  AllowInsertingColumns：是否允许用户在受保护的工作表上插入列。
  AllowInsertingRows：是否允许用户在受保护的工作表上插入行。
  AllowInsertingHyperlinks：是否允许用户在受保护的工作表中插入超链接。
  AllowDeletingColumns：是否允许用户在受保护的工作表上删除列，要删除的列中的每个单元格都是解除锁定的。
  AllowDeletingRows：是否允许用户在受保护的工作表上删除行，要删除的行中的每个单元格都是解除锁定的。
  AllowSorting：是否允许用户在受保护的工作表上进行排序。排序区域中的每个单元格必须是解除锁定的或取消保护的。
  AllowFiltering：是否允许用户在受保护的工作表上设置筛选。用户可以更改筛选条件，但是不能启用或禁用自动筛选功能。用户也可以在已有的自动筛选功能上设置筛选。
  AllowUsingPivotTables：是否允许用户在受保护的工作表上使用数据透视表。

- UnProtect

  撤销保护工作表。
  如果在保护工作表时没有设置密码，则直接使用 UnProtect 即可；如果设置了密码，则需要在撤销保护工作表时指定保护密码，如：
  Ws.UnProtect(Password:="hhh")

- Select
  选定并激活指定的工作表，如：Ws.Select

Range

WorkSheet 的下一级对象是 Range，它可以是某个单元格、某一行、某一列或者多个相邻或不相邻的单元格区域对象。Range 是 Excel 应用程序中使用最多的对象。在操作 Excel 的任何单元格区域之前，都要将其表示为一个 Range 对象，然后再使用该 Range 对象的属性和方法。

引用单元格

- 通过 Range 属性引用

  这也是引用单元格最常用的方式，如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range
Rg = Ws.Range("A5")  '引用单个单元格
Rg = Ws.Range("A3:B5")  '引用连续的单元格区域
Rg = Ws.Range("A1:D8,A20:C25,F6:G10")'引用不连续的单元格区域
Rg = Ws.Range("B:B") '引用单列，也可以 Columns 引用。如：Rg=Ws.Columns("B")，或者 Rg=Ws.Columns(2)
Rg = Ws.Range("B:D") '引用连续的多列
Rg = Ws.Range("A:A,C:C,H:H") '引用不连续的多列
Rg = Ws.Range("5:5") '引用单行
Rg = Ws.Range("5:20")'引用连续的多行
Rg = Ws.Range("1:1,3:3,5:5") '引用不连续的多行
App.Quit
```

- 通过 Cells 集合引用

  例如 Cells(i,j)就表示第 i 行、第 j 列处的单元格，i、j 都是整数。
  如果没有指定具体的行号和列号，那么就代表引用整个工作表。如，Rg = Ws.Cells。
  如果需要 按位置循环引用单元格区域的各个单元格，那么使用 Cells 属性是最方便的，例

如下面的代码就是将第 5-10 行、6-20 列的所有单元格内容替换为"abc"：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range
For i As Integer = 5 To 10
    For j As Integer = 6 To 20
        Rg = Ws.Cells(i,j)
        Rg.Value = "abc"
    Next
Next
App.Visible = True
```

- 通过 Range 和 Cells 联合引用单元格。

  这种方法特别适合于单元格区域变化的场合。
  例如下面的代码同样可以将第 5-10 行、6-20 列的所有单元格内容替换为"abc"：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("c:\abc.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(2)
Dim Rg As MSExcel.Range
Rg = Ws.Range(Ws.Cells(5,6), Ws.Cells(10,20))
Rg.Value = "abc"
App.Visible = True
```

- 引用已经使用的单元格区域。

  利用 UsedRange 属性，可以取得在指定工作表中已经使用的单元格区域。如下面的代码就将得到已经使用的行数、列数和单元格个数：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
MessageBox.Show(Rg.Rows.Count)
MessageBox.Show(Rg.Columns.Count)
MessageBox.Show(Rg.Count)
App.Quit
```

- 动态引用单元格

  方法一

利用 Offset 方法，以初始值所设定的单元格为起点，来相对移动任意的行或列，从而定位到新的单元格。下面的例子就是从目前的单元格 A2 开始，向下移动 5 行、向右移动 3 列，也就是定位到单元格 D7，并显示新的单元格地址：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A2")
Rg = Rg.Offset(5,3)
MessageBox.Show(Rg.Address)
App.Quit
```

方法二

利用 Resize 方法，可以将单元格范围改变为指定的大小，并引用变更后的单元格区域。需要注意的是，在使用 Resize 属性更改区域范围时，行参数和列参数是区域的总行数和总列数。下面的例子就是以单元格区域 A2:D4 的左上角单元格 A2 为基准，将单元格区域变更为有 6 行、6 列的单元格区域，即新引用的单元格区域为 A2:F7，并显示新的单元格范围：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A2:D4")
Rg = Rg.Resize(6,6)
MessageBox.Show(Rg.Address)
App.Quit
```

- 引用整行或整行区域。

```
Rg = Ws.Rows("5")  '引用第 5 行
Rg = Ws.Rows("5:6")  '引用第 5-6 行
```

- 引用整列或整列区域。

```
Rg = Ws.Columns("B")  '引用 B 列
Rg = Ws.Columns("B:F")'引用 B-F 列
```

- 引用某个单元格所在的整个行

利用 EntireRow 属性，可以获取某个单元格所在的整个行。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
```

```
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("C5").EntireRow
App.Quit
```

- 引用某个单元格所在的整个列

  利用 EntireColumn 属性，可以获取某个单元格所在的整个列。

主要属性

- Address

  获取单元格或单元格区域的地址。

- Count

  获取指定单元格区域内的单元格个数。

- Row

  获取指定单元格的行号。以下代码可获取单元格区域的行号范围：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
Dim RowBegin As Integer = Rg.Cells(1).Row
Dim RowEnd As Integer = Rg.Cells(Rg.Count).Row
MessageBox.Show(RowBegin)
MessageBox.Show(RowEnd)
App.Quit
```

- Rows.Count
  获取单元格区域内的行数，如：MessageBox.Show(Rg.Rows.Count)

- Column

  获取指定单元格的列号。以下代码可获取单元格区域的列号范围：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
Dim ColumnBegin As Integer = Rg.Cells(1).Column
Dim ColumnEnd As Integer = Rg.Cells(Rg.Count).Column
MessageBox.Show(ColumnBegin)
```

```
MessageBox.Show(ColumnEnd)
App.Quit
```

- Columns.Count
  获取单元格区域内的列数。如：MessageBox.Show(Rg.Columns.Count)

- Value

  返回或设置指定单元格的值。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Cells(1,1)
Rg.Value = 12345.88  '输入数字
Rg.Value = "abcde"'输入字符
Rg.Value = "'012345"  '如要输入数字字符串,要在前面加上前缀字符"'"
Rg.Value = "2006/5/20"'输入日期
Rg.Value = "2006-5-20"'输入日期
Rg.Value = Cdate("2006 年 5 月 20 日") '输入日期
Rg.Value = "10:30:20" '输入时间
Rg.Value = Cdate("10:30PM")  '输入时间
App.Visible = True
```

  通过 Value 和数组向单元格区域一次性输入多个数据。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Array As String() = {"a","b","c","d"}  '定义数组
Dim Rg As MSExcel.Range = Ws.Range("A1:D1")
Rg.Value = Array '写入指定的单元格
App.Visible = True
```

  通过 Value 向单元格区域输入连续值(自增值)。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("G1:G20")  '指定单元格区域
Rg.Cells(1).Value = 10 '设定初始值
Rg.Cells(1).AutoFill(Rg,MSExcel.XlAutoFillType.xlFillSeries)  '对指定区域生成自增值
App.Visible = True
```

同样的方式，可以实现其它内容的自增，如：

```
Rg.Cells(1).Value = "2009-10-10"  '也可以设为"星期一"、"1 月"等，则自动按星期和月份自增
Rg.Cells(1).AutoFill(Rg,MSExcel.XlAutoFillType.xlFillDefault)
```

如果在两个初始的单元格输入间隔某个数字的数据，还可以实现输入具有相同间隔的字符或数值，如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("G1:G20")
Dim Jg As MSExcel.Range = Ws.Range("G1:G2")
Jg.Cells(1).Value = 1
Jg.Cells(2).Value = 3
Jg.AutoFill(Rg,MSExcel.XlAutoFillType.xlFillDefault)
App.Visible = True
```

- Font

用于设置单元格的字体格式，例如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1")
Rg.Value = "Foxtable 操控 Excel 全攻略"
With Rg.Font
    .Name = "黑体" '字体
    .Size = 20 '字号
    .Bold = True   '加粗
    .Italic = True '斜体
    .ColorIndex = 3'颜色
End With
App.Visible = True
```

其它字体设置：

```
上标：Rg.Font.Superscript = True
下标：Rg.Font.Subscript = True
单下划线：Rg.Font.Underline = MSExcel.XlUnderlineStyle.xlUnderlineStyleSingle
双下划线：Rg.Font.Underline = MSExcel.XlUnderlineStyle.xlUnderlineStyleDouble
会计用单下划线：Rg.Font.Underline =
```

MSExcel.XlUnderlineStyle.xlUnderlineStyleSingleAccounting
会计用双下划线：Rg.Font.Underline =
MSExcel.XlUnderlineStyle.xlUnderlineStyleDoubleAccounting
取消下划线：Rg.Font.Underline = MSExcel.XlUnderlineStyle.xlUnderlineStyleNone

设置单元格字符串中一部分字符的字体格式:

利用 Range 对象的 Characters 属性，定位到要设置格式的字符，然后利用 Font 对其进行设置。Characters 中的第一个参数为起始位置，第二个参数为字符长度。如，下面的代码就是对“Foxtable 操控 Excel 全攻略”中的“操控”两字进行格式设置：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1")
Rg.Value = "Foxtable 操控 Excel 全攻略"
Dim Chr As MSExcel.Characters = Rg.Characters(9,2)
With Chr.Font
    .Name = "黑体" '字体
    .Size = 20 '字号
    .Bold = True    '加粗
    .Italic = True '斜体
    .ColorIndex = 3'颜色
End With
App.Visible = True
```

需要说明的是，部分字符串同样可以设置上标、下标和下划线。

- BorderAround

  设置单元格区域的外部边框。此属性有 3 个参数：

  - LineStyle

    边框的线型，可以为以下 XlLineStyle 值之一：
    xlContinuous
    xlDash
    xlDashDot
    xlDashDotDot
    xlDot
    xlDouble
    xlSlantDashDot
    xlLineStyleNone

xlNone

- Weight
  边框粗细。可以为以下 XlBorderWeight 常数之一：
  xlHairline
  xlMedium
  xlThick
  xlThin

- ColorIndex
  边框颜色编号。

示例代码：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
Rg.BorderAround(MSExcel.XlLineStyle.xlContinuous,MSExcel.XlBorderWeight.xlThick,4)
App.Visible = True
```

- Borders

用于置单元格区域的内部边框。

如下面的代码就是为已经使用的单元格全部加上边框：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
Rg.Borders.Linestyle = MSExcel.XlLineStyle.xlContinuous  '边框线型
Rg.Borders.Weight = MSExcel.XlBorderWeight.xlThin'边框粗细
Rg.Borders.ColorIndex = 3'边框颜色
App.Visible = True
```

除此之外，我们还可以为单元格有针对性的设置边框，它必须指定为 XlBordersIndex 枚举中的其中之一：

xlDiagonalDown：　从区域中每个单元格的左上角至右下角的边框(斜下)。
xlDiagonalUp：　从区域中每个单元格的左下角至右上角的边框(斜上)。
xlEdgeBottom：　区域底部的边框(底边框)。
xlEdgeLeft：　区域左边的边框(左边框)。
xlEdgeRight：　区域右边的边框(右边框)。

xlEdgeTop： 区域顶部的边框(上边框)。
xlInsideHorizontal：区域中所有单元格的水平边框，区域以外的边框除外(内部水平边框线)。
xlInsideVertical： 区域中所有单元格的垂直边框，区域以外的边框除外(内部垂直边框线)。

例如下面的代码就仅为已经使用的单元格区域设置了内部水平边框线：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
With Rg.Borders(MSExcel.XlBordersIndex.xlInsideHorizontal)
   .LineStyle = MSExcel.XlLineStyle.xlContinuous
   .Weight = MSExcel.XlBorderWeight.xlThin
   .ColorIndex = 3
End With
App.Visible = True
```

如要取消全部边框，则只需：Rg.Borders.Linestyle = 0

- Interior

利用 Range 对象的 Interior 属性，可以设置单元格的填充颜色、背景图案和背景颜色等。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.UsedRange
With Rg.Interior
   .ColorIndex = 3'单元格填充颜色为红色
   .Pattern = MSExcel.XlPattern.xlPatternCrissCross   '单元格背景图案为十字图案
   .PatternColorIndex = 6 '单元格背景颜色为黄色
End With
App.Visible = True
```

- NumberFormat

单元格格式是通过 NumberFormat 或 NumberFormatLocal 属性来完成的。例如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Ws.Range("A1").NumberFormat = "G/通用格式" '通用格式
Ws.Range("B1").NumberFormat = "0_ "   '数值
```

```
Ws.Range("C1").NumberFormat = "#,##0.00_);[红色](#,##0.00)" '货币
Ws.Range("D1").NumberFormat = "_ * #,##0.00_;_ * -#,##0.00_ ;_ * ""-""??_ ;_ @_ " '会计专用
Ws.Range("E1").NumberFormat = "yyyy-m-d" '日期
Ws.Range("F1").NumberFormat = "h:mm:ss"  '时间
Ws.Range("G1").NumberFormat = "0.00%"    '百分比
Ws.Range("H1").NumberFormat = "# ?/?"    '分数
Ws.Range("I1").NumberFormat = "0.00E+00" '科学记数
Ws.Range("J1").NumberFormat = "@"    '文本
App.Visible = True
```

可在上述代码执行完毕后，在相应的单元格中输入数据以查看格式效果。

- HorizontalAlignment
  VerticalAlignment

  这两个属性用于设置单元格的对齐方式。
  通常情况下，当单元格内的数据是文本时，数据自动左对齐；当单元格内的数据是数字时，数据自动右对齐。
  通过 HorizontalAlignment 属性可进行水平对齐设置，使用 VerticalAlignment 属性可进行垂直对齐设置。例如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1")
Rg.Value = "Foxtable 操控 Excel 全攻略"
Rg.HorizontalAlignment = MSExcel.Constants.xlRight '水平右对齐
Rg.HorizontalAlignment = MSExcel.Constants.xlLeft '水平左对齐'
Rg.HorizontalAlignment = MSExcel.Constants.xlCenter '水平居中
Rg.HorizontalAlignment = MSExcel.Constants.xlDistributed   '水平分散对齐
Rg.HorizontalAlignment = MSExcel.Constants.xlGeneral   '水平对齐恢复默认
Rg.VerticalAlignment = MSExcel.Constants.xlTop '垂直靠上
Rg.VerticalAlignment = MSExcel.Constants.xlBottom '垂直靠下
Rg.VerticalAlignment = MSExcel.Constants.xlCenter '垂直居中
Rg.VerticalAlignment = MSExcel.Constants.xlGeneral '垂直对齐恢复默认
App.Visible = True
```

- RowHeight
  ColumnWidth

  这两个属性用于设置单元格大小，RowHeight 用于设置行高，ColumnWidth 用于设置列宽。
  例如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1:C1")  '可指定任意单元格或单元格区域
Rg.RowHeight = 40 '行高 40 磅
Rg.ColumnWidth = 20  '列宽 20 磅
App.Visible = True
```

如果要恢复标准行高和标准列宽，可以这样：

```
Rg.RowHeight = Ws.StandardHeight
Rg.ColumnWidth = Ws.StandardWidth
```

- ShrinkToFit

是否缩小单元格内容以全部显示。
当单元格内的数据长度超过单元格宽度时，就会只显示一部分数据。
使用 ShrinkToFit 属性，可将单元格内的数据缩小，以适应单元格，使内容全部显示出来。
如：Rg.ShrinkToFit = True

如果反过来，希望自动调整单元格大小，使单元格能够显示全部数据。这需要用到 AutoFit 方法。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1")
Rg.Value = "Foxtable 操控 Excel 全攻略"
Rg.Font.Size = 50 '把字体放大
Rg.EntireColumn.AutoFit  '自动调整列宽
Rg.EntireRow.AutoFit  '自动调整行高
App.Visible = True
```

如果要调整工作表全部单元格的大小，可以使用 Cells 属性。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Cells
Rg.EntireColumn.AutoFit  '自动调整列宽
Rg.EntireRow.AutoFit  '自动调整行高
App.Visible = True
```

- Locked
  利用 Locked 属性可锁定指定的单元格，也可解除锁定。
  需要注意的是，如果工作表没有被保护，即使将单元格设置为锁定状态，该单元格仍可以操作。只有在工作表被保护的情况下，锁定单元格才能起作用。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1:A10")'指定任意单元格
Ws.Unprotect'撤销对工作表的保护
Ws.Cells.Locked = False '解除整个工作表所有单元格的锁定
Rg.Locked = True'锁定指定的单元格
Ws.Protect  '保护工作表
App.Visible = True
```

常用方法

- Select

  选定单元格或单元格区域。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Ws.Select '在选定单元格之前,必须先选定工作表!!
Ws.Range("A1:A10").Select'选定单元格
App.Visible = True
```

- Goto

  选定单元格，并滚动窗口直至目标区域的左上角单元格出现在窗口的左上角处。
  注意，这是 Application 的方法，不是 Range 的方法。

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("P100:P105")
Ws.Select
App.Goto(Rg,True)  '第 2 个参数用于决定是否滚动
App.Visible = True
```

- Activate

  激活单元格。
  需要注意的是，Activate 是用来激活对象的方法，而 Select 是用来选取对象的方法。使用

Select 方法可以一次选取多个单元格区域，而 Activate 一次只能激活一个单元格。
如下面的代码中，即使将激活单元格区域修改为"B5:B8"，但激活的单元格还是只有一个"B5"：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Ws.Select
Ws.Range("A1:D10").Select'选定单元格区域
Ws.Range("B5").Activate   '激活单元格
App.Visible = True
```

- Merge

合并单元格。
利用 Merge 方法，或将 MergeCells 属性设置为 True，都可以合并单元格。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1:D10")
App.DisplayAlerts = False '加上此行可禁止弹出合并前的提示
Rg.Merge   '合并指定区域的单元格
'Rg.MergeCells = True '用这种方式也可以合并
App.Visible = True
```

- UnMerge

取消合并单元格。
利用 UnMerge 方法，或将 MergeCells 属性设置为 False，可将已经合并的单元格重新分解为独立的单元格。
如下面的代码就是先利用 MergeArea 属性判断某个单元格是否为合并单元格的一部分，然后再进行取消：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A1")
If Rg.MergeArea.Address = Rg.Address Then
    MessageBox.Show("该单元格区域并不是合并后的单元格！")
Else
    App.DisplayAlerts = False    '加上此行可禁止弹出合并前的提示
    Rg.UnMerge   '取消合并
    'Rg.MergeCells = False   '用这种方式也可以取消
```

```
End If
App.Visible = True
```

- Clear
  清除单元格的全部信息。
  本方法将清除单元格的全部信息(包括数据、格式、批注等等)。

- ClearContents
  清除单元格的值。
  本方法仅清除单元格的值，但保留其格式设置和批注等信息。

- ClearComments
  清除单元格的批注。

- ClearFormats
  清除单元格的格式。

- ClearNotes
  清除单元格的批注和语音批注。

- Insert

  插入单元格。在插入单元格时，可以设置单元格的移动方向。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A10") '以这个指定的单元格为基准
Rg.Insert(MSExcel.XlDirection.xlToRight)  '将相应的单元格右移
App.Visible = True
```

  其中，插入单元格的移动方向为 XlDirection 枚举，可为以下值之一：

  xlUp：向上
  xlToRight：向右
  xlToLeft：向左
  xlDown：向下

  利用 Insert 方法，同时结合 EntireRow 和 EntireColumn 属性，可插入整行和整列。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A10")  '以这个指定的单元格为基准
```

```
Rg.EntireRow.Insert(MSExcel.XlInsertShiftDirection.xlShiftDown)'在基准单元格上面插入一行
Rg.EntireColumn.Insert(MSExcel.XlInsertShiftDirection.xlShiftToRight)  '在基准单元格左边插入一列
App.Visible = True
```

其中，插入整行或整列的移动方向为 XlInsertShiftDirection 枚举，且只能为以下值之一：

```
xlShiftToRight
xlShiftDown
```

- Delete

删除单元格。在删除单元格时，可以设置单元格的移动方向(在删除某个单元格后，旁边的单元格会根据设置的单元格移动方向填补空缺)。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A10")'以这个指定的单元格为基准
Rg.Delete(MSExcel.XlDirection.xlUp)  '下面的单元格上移
App.Visible = True
```

利用 Delete 方法，同时结合 EntireRow 和 EntireColumn 属性，可删除整行和整列。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("A10")'以这个指定的单元格为基准
Rg.EntireColumn.Delete(MSExcel.XlDirection.xlToLeft)  '右面的单元格左移
App.Visible = True
```

在删除多行或多列时，一样可以设置单元格的移动方向。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Rows("1:3")'选定多行
Rg.Delete(MSExcel.XlDirection.xlUp) '下面的单元格上移
Rg = Ws.Columns("D:F")  '选定多列
Rg.Delete(MSExcel.XlDirection.xlToLeft) '右面的单元格左移
App.Visible = True
```

可以删除工作表的全部单元格。如：

```
Rg = Ws.Cells
Rg.Delete
```

- Cut

  移动单元格或单元格区域。
  该方法有一个参数，用于指定要剪切到的目标区域。如果省略该参数，将把该区域复制到剪贴板。如下面的例子就是将单元格区域 B1:C4 的数据移动到以单元格 B3 为左上角的单元格区域：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
Dim Rg As MSExcel.Range = Ws.Range("B1:C4")
Rg.Cut(Ws.Range("B3"))
App.Visible = True
```

- Copy

  复制单元格或单元格区域。
  该方法有一个参数，用于指定要复制到的目标区域。如果省略该参数，将把该区域复制到剪贴板。

Chart

Chart 是 Excel 提供的对数据进行形象化和直观化处理的图表分析工具。
提示：CaseStudy 目录下有一个演示文件：Excel 图表.Table，此文件演示了如何利用本节的内容直接生成 Excel 的图表。
必须掌握的两个基本概念：

- ChartObject

  ChartObject 对象代表一个嵌入式图表，它的作用就是作为 Chart 对象的容器。每个 ChartObject 对象的属性和方法控制工作表上嵌入式图表的外观和尺寸，而 ChartObjects 则是指工作表中所有的嵌入式图表集合。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)'指定工作表
MessageBox.Show(Ws.ChartObjects.Count)  '返回所有嵌入式图表的数量
MessageBox.Show(Ws.ChartObjects(1).Name)'返回指定图表的名称
```

  通过 Add 方法可新增一个空白的嵌入式图表。其句法为：

```
Add(Left, Top, Width, Height)
```

Left 和 Top 是以磅为单位所指定的初始坐标，该坐标是相对于工作表上单元格“A1”左上角的坐标。
Width 和 Height 是以磅为单位指定新对象的初始大小 ，如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("c:\abc.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(2)'指定工作表
Dim Co As MSExcel.ChartObject = Ws.ChartObjects.Add(100,30,400,250)'新增
app.Visible = True
```

通过 Delete 方法可删除指定工作表中的所有嵌入式图表。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)'指定工作表
Dim Co As MSExcel.ChartObject = Ws.ChartObjects.Add(100,30,400,250)'新增
Ws.ChartObjects.Delete '删除
App.Quit
```

- Chart

Chart 为包含于 ChartObject 对象中的图表。如，引用 Chart 对象：

```
Dim Cht As MSExcel.Chart = Co.Chart
```

需要特别说明的是，以下介绍到的所有属性和方法中，除单独指出的以外，都是针对 Chart 的。

主要属性

- Visible

是否允许图表可见。

- ChartType

返回或设置图表类型，适用于 Chart 和 Series 对象。以下为 XlChartType 常量：

柱形图：xlColumnClustered
三维簇状柱形图：xl3DColumnClustered
堆积柱形图：xlColumnStacked
三维堆积柱形图：xl3DColumnStacked
百分比堆积柱形图：xlColumnStacked100
三维百分比堆积柱形图：xl3DColumnStacked100
三维柱形图：xl3DColumn
簇状条形图：xlBarClustered

三维簇状条形图：xl3DBarClustered
堆积条形图：xlBarStacked
三维堆积条形图：xl3DBarStacked
百分比堆积条形图：xlBarStacked100
三维百分比堆积条形图：xl3DBarStacked100
折线图：xlLine
数据点折线图：xlLineMarkers
堆积折线图：xlLineStacked
堆积数据点折线图：xlLineMarkersStacked
百分比堆积折线图：xlLineStacked100
百分比堆积数据点折线图：xlLIneMarkersStacked100
折线图三维折线图：xl3DLine
饼图：xlPie
分离型饼图：xlPieExploded
三维饼图：xl3Dpie
三维分离型饼图：xl3DPieExploded
复合饼图：xlPieOfPie
饼图复合柱饼图：xlBarOfPie
散点图：xlXYScatter
平滑线散点图：xlXYScatterSmooth
无数据点折线散点图：xlXYScatterLinesNoMarkers
折线散点图：xlXYScatterLines
无数据点折线散点图：xlXYScatterLinesNoMarkers
气泡图：xlBubble
三维气泡图 xlBubble3DEffect
面积图：xlArea
三维面积图：xl3DArea
堆积面积图：xlAreaStacked
三维堆积面积图：xl3DAreaStacked
百分比堆积面积图：xlAreaStacked100
三维百分比堆积面积图：xl3DAreaStacked100
圆环图：xlDoughnut
分离型圆环图：xlDoughnutExploded
雷达图：xlRadar
数据点雷达图：xlRadarMarkers
填充雷达图：xlRadarFilled
三维曲面图：xlSurface
曲面图（俯视图）：xlSurfaceTopView
三维曲面图（框架图）：xlSurfaceWireframe
曲面图（俯视框架图）：xlSurfaceTopViewWireframe
盘高-盘低-收盘图：xlStockHLC
成交量-盘高-盘低-收盘图：xlStockVHLC
开盘-盘高-盘低-收盘图：xlStockOHLC
成交量-开盘-盘高-盘低-收盘图：xlStockVOHLC

簇状柱形圆柱图：xlCylinderColClustered
簇状条形圆柱图：xlCylinderBarClustered
堆积柱形圆柱图：xlCylinderColStacked
堆积条形圆柱图：xlCylinderBarStacked
百分比堆积柱形圆柱图：xlCylinderColStacked100
百分比堆积条形圆柱图：xlCylinderBarStacked100
三维柱形圆柱图：xlCylinderCol
簇状柱形圆锥图：xlConeColClustered
簇状条形圆锥图：xlConeBarClustered
堆积柱形圆锥图：xlConeColStacked
堆积条形圆锥图：xlConeBarStacked
百分比堆积柱形圆锥图：xlConeColStacked100
百分比堆积条形圆锥图：xlConeBarStacked100
三维柱形圆锥图：xlConeCol
簇状柱形棱锥图：xlPyramidColClustered
簇状条形棱锥图：xlPyramidBarClustered
堆积柱形棱锥图：xlPyramidColStacked
堆积条形棱锥图：xlPyramidBarStacked
百分比堆积柱形棱锥图：xlPyramidColStacked100
百分比堆积条形棱锥图：xlPyramidBarStacked100
三维柱形棱锥图：xlPyramidCol

- HasTitle
  如果设置坐标轴或图表的标题可见，则为 True。
  本属性适用于 Chart 和 Axis 对象。其中，ChartTitle 代表图表标题，AxisTitle 代表坐标轴标题。

- ChartTitle
  图表标题。
  注意，只有将 HasTitle 属性设置为 True 后，才存在 ChartTitle 对象，从而才能使用该对象。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)'指定工作表
Dim Rg As MSExcel.Range = Ws.Range("A2:B18")  '图表数据源
ws.ChartObjects.Delete'删除工作表上已经存在的图表
Dim Co As MSExcel.ChartObject = Ws.ChartObjects.Add(100,30,400,250)  '创建图表对象
Dim Cht As MSExcel.Chart = Co.Chart
Cht.ChartType = MSExcel.XlChartType.xlLineMarkers'图表类型
Cht.SetSourceData(Rg,MSExcel.XlRowCol.xlColumns) '指定数据源和绘图方式
Cht.HasTitle = True
Cht.ChartTitle.Text = "测试图表标题"
With Cht.ChartTitle.Font  '设置标题字体
```

```
    .Size = 16
    .ColorIndex = 3
    .Name = "隶书"
End With
App.Visible = True
```

- HasAxis
  返回或者设置指定图表中出现的坐标轴， 句法：

  HasAxis(Index1, Index2)

  Index1，可选，表示坐标轴类型。可为下列 XlAxisType 常量之一：xlCategory、xlValue 或 xlSeriesAxis(xlSeriesAxis 系列坐标轴仅应用于三维图表)。
  Index2，可选，表示坐标轴组。可为下列两个 XlAxisGroup 常量之一：xlPrimary 或 xlSecondary(三维图表仅包含有关坐标轴集合)。

  如果对图表类型或 AxisGroup 属性进行更改，可能会对坐标轴进行创建或删除。

  如，显示图表中的主要值坐标轴：

  ```
  Cht.HasAxis(MSExcel.XlAxisType.xlValue, MSExcel.XlAxisGroup.xlPrimary) = True
  ```

- HasLengend
  是否显示图例。

- Legend
  返回一个 Legend 对象，该对象代表指定图表中的图例。
  如，下例将显示图例，并将图例的字体颜色设为蓝色：

  ```
  With Cht
      .HasLegend = True
      .Legend.Font.ColorIndex = 5
  End With
  ```

- HasDataTable
  如果图表有数据表，本属性为 True。

- DataTable
  返回一个 DataTable 对象，该对象代表图表数据表。
  例如，向内嵌图表添加带有分级显示边框的数据表，但无内部网格线：

  ```
  With Cht
      .HasDataTable = True
      With .DataTable
  ```

```
        .HasBorderOutline = True
        .HasBorderHorizontal = False
        .HasBorderVertical = False
    End With
End With
```

- ChartArea
  返回图表区域。
  如将图表区域的内部颜色设为红色，并将其边框颜色设为蓝色：

```
With Cht.ChartArea
    .Interior.ColorIndex = 3
    .Border.ColorIndex = 5
End With
```

- PlotArea
  返回一个 PlotArea 对象，该对象代表图表的绘图区域。
  如将绘图区域的内部颜色设为淡青色：
  Cht.PlotArea.Interior.ColorIndex = 8

- PlotBy
  返回或设置行和列在图表中作为数据系列使用的方式。
  可为下列 XlRowCol 常量之一：xlColumns 或 xlRows。如使图表按列分布数据：

```
Cht.PlotBy = xlColumns
```

- PlotVisibleOnly
  是否仅绘制可见单元格。当本属性为 False 时，所有可见单元格和隐藏单元格都绘制。
  如仅绘制可见单元格：

```
Cht.PlotVisibleOnly = True
```

- DisplayBlanksAs
  返回或者设置图表中空白单元格的处理方式。
  可为下列 XlDisplayBlanksAs 常量之一：xlNotPlotted、xlInterpolated 或 xlZero。
  如设置图表不绘制空白单元格：

```
Cht.DisplayBlanksAs = MSExcel.XlDisplayBlanksAs.xlNotPlotted
```

- ProtectContents
  本属性为 True 时，将保护整个图表。

- ProtectData
  本属性为 True 时，将保护内嵌图表中的数据。如：

Cht.ProtectData = True

- ProtectFormatting
  是否允许对图表元素进行添加、移动、调整大小或删除等操作。

- ProtectGoalSeek
  是否允许用鼠标操作修改图表数据点。

- ProtectSelection
  是否允许向图表添加形状。

- PageSetup
  返回一个 PageSetup 对象，该对象包含指定对象的所有页面设置。
  如设置图表中央标题的文字：
  Cht.PageSetup.CenterHeader = "December Sales"

主要方法：

- SetSourceData

  设置图表的源数据列,句法：

  SetSourceData(Source,PlotBy)

  Source 为 Range 类型，必需。该列包含源数据。
  PoltBy，可选，指定划分数据的方法。可以是下列 XlRowCol 常量之一：xlColumns 或 xlRows。

  如：Cht.SetSourceDataSource(Rg,MSExcel.XlRowCol.xlColumns)

- SetBackgroundPicture

  为图表设置背景图形,如：

  Cht.SetBackgroundPicture("d:\123.jpg")

- Axes

  获取一个对象，该对象表示图表上的单个坐标轴。该对象是 Axes 集合的成员，语法：

  Axes(Type, Group)

  Type: 指定要返回的轴，可为下列 XlAxisType 常量之一：xlValue、xlCategory 或 xlSeriesAxis(xlSeriesAxis 仅对三维图表有效)；
  Group: 为坐标轴组，可为下列 XlAxisGroup 常量之一： xlPrimary 或 xlSecondary。如果省略该参数，将使用主轴组(注意，三维图表只有一个坐标轴组)。

如：

```
With Cht.Axes(MSExcel.XlAxisType.xlCategory)  '获取 X 轴
    .HasTitle = True
    .AxisTitle.Text = "时间序列"  '添加标题
    .HasMajorGridlines = True  '显示主要网格线
    .HasMinorGridlines = True  '显示次要网格线
End With
With Cht.Axes(MSExcel.XlAxisType.xlValue) '获取 Y 轴
    .HasTitle = True
    .AxisTitle.Text = "销售量"  '添加标题
    .HasMajorGridlines = True'显示只要网格线
    .HasMinorGridlines = True'显示次要网格线
End With
```

- SeriesCollection

返回一个对象，该对象既可代表单个系列(Series 对象)，也可代表图表中所有系列的集合(SeriesCollection 对象)， 句法：
SeriesCollection(Index)

其中，Index 为可选项。Index 设置为具体的数值时，表示单个系列；Index 省略时，代表图表中所有系列的集合。

例如显示图表中第 1 个系列的数据标志：Cht.SeriesCollection(1).HasDataLabels = True
创建新系列并将其添加到图表中：Cht.SeriesCollection.Add(Ws.Range("B1:B19"))
得到数据系列数量：MessageBox.Show(Cht.SeriesCollection.Count)

示例代码：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1) '指定工作表
ws.ChartObjects.Delete '删除工作表上已经存在的图表
Dim Co As MSExcel.ChartObject = Ws.ChartObjects.Add(100,30,400,250)  '创建图表对象
Dim Cht As MSExcel.Chart = Co.Chart
Dim Rg As MSExcel.Range = Ws.Range("A2:B18") '图表数据源
Cht.ChartType = MSExcel.XlChartType.xlLineMarkers'图表类型
Cht.SetSourceData(Rg,MSExcel.XlRowCol.xlColumns) '指定数据源和绘图方式
Cht.SeriesCollection.Add(Ws.Range("C2:C18")) '新增一个数据系列
Dim S As MSExcel.Series = Cht.SeriesCollection(Cht.SeriesCollection.Count)  '引用新增的数据系列
S.ApplyCustomType(MSExcel.XlChartType.xlColumnClustered)'指定新增数据系列的图表类型
```

```
S.Interior.Color = RGB(255,0,0) '指定新增数据系列的颜色
App.Visible = True
```

- Paste

  将剪贴板中的数据粘贴到指定的图表中， 如：

```
Ws.Range("D2:D18").Copy
Cht.Paste '粘贴完成后，图表中将自动增加一个数据系列
```

- Points

  该方法适用于 Series 对象，返回该数据系列中的单个数据点或所有数据点的集合， 句法：

  Points(index)

  Index 用于指定数据点位置，系列中的数据点按从左至右的顺序编号，Points(1)为最左边的数据点，而 Points(S.Points.Count)为最右边的数据点。

  如对图表上第一个数据系列的最后一个数据点应用数据标志：

```
Dim i As Integer = S.Points.Count
S.Points(i).ApplyDataLabels
```

- ApplyCustomType

  将一个图表或系列设置为标准图表类型或自定义图表类型。本方法适用于 Chart 和 Series 对象。

  语法：

  ApplyCustomType(ChartType,TypeName)

  ChartType：标准的图表类型(详见 ChartType 属性)。对于 Chart 对象，该参数也可为下列 XlChartGallery 常量之一：xlBuiltIn、xlUserDefined 或 xlAnyGallery。
  TypeName： 可选值，仅对 Chart 对象使用。如果 ChartType 参数指定的是一个自定义图表库，则为自定义图表类型的名称。

  例如将“数据标记折线图”图表类型应用于图表：

```
Cht.ApplyCustomType(MSExcel.XlChartType.xlLineMarkers)
```

- ApplyDataLabels

将数据标志应用于图表中的某一数据点、某一数据系列或所有数据系列。本方法适用于 Chart、Point 或 Series 对象。

语法:

ApplyDataLabels(Type,LegendKey,AutoText,HasLeaderLines)

Type：可选，表示数据标志类型。可为下列 XlDataLabelsType 常量之一：

- xlDataLabelsShowNone：无数据标志。
- xlDataLabelsShowValue：数据点的值(若未指定本参数，即使用此设置)。
- xlDataLabelsShowPercent：总数的百分比，仅用于饼图或圆环图。
- xlDataLabelsShowLabel：数据点所属的分类，这是默认值。
- xlDataLabelsShowLabelAndPercent：占总数的百分比及数据点所属的分类，仅用于饼图或圆环图。

LegendKey：若指定为 True，则在数据点旁边显示图例标示。默认值为 False。
Autotext：可选。若指定为 True，则对象基于上下文自动产生合适的文本。
HasleadersLines：可选，适用于 Chart 和 Series 对象。若指定为 True，则序列有前导字符线。

例如对图表上的第一个数据系列应用分类标志：

Cht.SeriesCollection(1).ApplyDataLabels(MSExcel.TXlDataLabelsType.xlDataLabelsShowLabel)

- Delete

  删除图表。注意，这里的图表对象必须是 ChartObject。

  如：Co.Delete

- Export

  将图表以某种图形格式导出。注意，这里的图表对象必须是 Chart。
  如：Cht.Export("D:\123.JPG")

三维图表中的特有属性:

- Line3DGroup
  返回一个 ChartGroup 对象，该对象代表三维图表中的折线图组。
  如，设置图表的三维折线组，使其中的每一数据标记使用不同的颜色：

```
Cht.Line3DGroup.VaryByCategories = True
```

- Bar3Dgroup
  返回一个 ChartGroup 对象，该对象代表三维图表中的条形图组。
  如：设置三维条形图组的条形簇之间的距离等于条形宽度的一半：
  ```
  Cht.Bar3DGroup.GapWidth = 50
  ```

- Column3Dgroup
  返回一个 ChartGroup 对象，该对象代表三维图表中的柱形图表组。
  如：设置三维柱形图组的柱形簇之间的距离等于柱形宽度的一半：
  ```
  Cht.Column3DGroup.GapWidth = 50
  ```

- Area3Dgroup
  返回一个 ChartGroup 对象，该对象代表三维图表上的面积图组。
  如：为三维面积图表组添加垂直线：
  ```
  Cht.Area3DGroup.HasDropLines = True
  ```

- Pie3Dgroup
  返回一个 ChartGroup 对象，该对象代表三维图表中的饼图组。
  如，设置图表中的三维饼图组，使其中的每一数据标记使用不同的颜色：
  ```
  Cht.Pie3DGroup.VaryByCategories = True
  ```

- SurfaceGroup
  返回一个 ChartGroup 对象，该对象代表三维图表中的曲面图组。
  如，设置图表中的三维曲面图组，使其中的每一数据标志都使用不同的颜色：
  ```
  Cht.SurfaceGroup.VaryByCategories = True
  ```

- RightAngleAxes
  如果指定图表的坐标轴为直角，并与图表的转角或仰角无关，则为 True。
  本属性仅应用于三维折线图、柱形图和条形图。如果本属性设为 True，则自动忽略 Perspective 属性。如：
  ```
  Cht.RightAngleAxes = True
  ```

- Perspective
  返回或者设置三维图表视图的透视系数。必须为 0-100 之间的整数。
  如果 RightAngleAxes 属性为 True，可省略此属性。如，将 Chart 的透视系数设为 70：
  ```
  Cht.RightAngleAxes = False
  Cht.Perspective = 70
  ```

- AutoScaling
  如果要对三维图表进行缩放使其大小接近于等价的二维图表则设为 True。
  注意，RightAngleAxes 属性必须设为 True，此属性设置才有效。如：
  ```
  With Cht
     .RightAngleAxes = True
  ```

```
    .AutoScaling = True
End With
```

- BarShape
  返回或设置用于三维条形图或柱形图的形状。
  可为下列 XlBarShape 常量之一：xlBox，xlConeToMax，xlConeToPoint，xlCylinder，xlPyramidToMax 或 xlPyramidToPoint。
  如设置用于第一个数据系列的形状：
  Cht.SeriesCollection(1).BarShape = MSExcel.XlBarShape.xlConeToPoint

- Corners
  返回一个 Corners 对象，该对象代表三维图表的角。
  如选定图表的角：Cht.Corners.Select

- DepthPercent
  以图表宽度百分数的形式返回或者设置三维图表的深度(在 20 和 2000 之间)。
  如将图表的深度设置为其宽度的 50%：
  Cht.DepthPercent = 50

- HeightPercent
  以图表宽度比例(5 到 500 之间)的形式返回或者设置三维图表的高度。
  如将图表的高度设为其宽度的 80%：
  Cht.HeightPercent = 80

- GapDepth
  以数据标记宽度的百分比形式返回或者设置三维图表中数据系列之间的距离，本属性的值必须在 0 和 500 之间。
  如将图表数据系列之间的距离设为数据标记宽度的 200%：
  Cht.GapDepth = 200

- Elevation
  以角度为单位返回或者设置三维图表视图的仰角。
  图表仰角是指查看图表时视点所处的高度。对于大多数图表类型的默认值为 15，该属性的值必须在-90 和 90 之间，但三维条形图的仰角值必须在 0 和 44 之间。
  如将图表的仰角设为 34 度 ：
  Cht.Elevation = 34

- Floor
  返回一个 Floor 对象，代表三维图表的基底。
  如将图表的基底颜色设置为蓝色：
  Cht.Floor.Interior.ColorIndex = 5

- Rotation
  以度为单位返回或设置三维图表视图的转角(图形区绕 Z 轴的转角)，默认值为 20。

本属性的取值必须从 0 至 360(三维条形图从 0 至 44)。如：Cht.Rotation = 30

- Walls
  代表三维图表的背景墙。
  如：Cht.Walls.Interior.Pattern = MSExcel.Constants.xlGray75

- WallsAndGridlines2D
  如果在三维图表中绘制二维网格线，本属性为 True。

打印输出

利用 PageSetup 属性，可进行各种打印设置。如：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
With Ws.PageSetup
    '设置打印区域
    .PrintArea = "A1:H10"    '打印工作表的指定区域
    .PrintArea = Ws.UsedRange.Address    '打印工作表的使用区域
    .PrintTitleColumns = Ws.Columns("A:H").Address    '打印列标题(在每一页的左边重复出现)
    .PrintTitleRows = Ws.Rows(1).Address '打印行标题(在每一页的顶部重复出现)
    '设置页面
    .PaperSize = MSExcel.XlPaperSize.xlPaperA4    '纸张大小
    .LeftMargin = 30 '页面左边距
    .RightMargin = 30'页面右边距
    .TopMargin = 50   '页面顶部边距
    .BottomMargin = 50    '页面底部边距
    .HeaderMargin = 40    '页面顶端到页眉的距离
    .FooterMargin = 40    '页脚到页面底端的距离
    .CenterHorizontally = True    '页面水平居中
    .CenterVertically = True '页面垂直居中
    '设置页眉
    .LeftHeader = "打印日期: &D" '左页眉,&D 表示日期
    .CenterHeader  =  "&""隶书,常规""&20  数据分析表"    '中页眉,并将字体设置为隶书和 20 号字大小
    .RightHeader = "打印者: " & App.UserName '右页眉
    '设置页脚
    .LeftFooter = "文件: &F   &A" '左页脚,&F 表示文件名,&A 表示工作表名
    .CenterFooter = ""    '中页脚为空
    .RightFooter = "第  &P  页    共  &N  页"   '右页脚
    '打印模式
    .Orientation = MSExcel.xlPageOrientation.xlPortrait   '纵向打印
    .Orientation = MSExcel.xlPageOrientation.xlLandscape '横向打印
    .PrintHeadings = True'打印行号和列标
```

```
    .PrintGridlines = True    '打印网格线
    '缩放打印
    .Zoom = False'以下设置将缩印在一页内
    .FitToPagesWide = 1   '按照 1 页的宽度打印
    .FitToPagesTall = 1   '按照 1 页的高度打印
End With
App.Visible = True
Ws.PrintPreview
App.Quit
```

补充说明，Zoom 是用于设置打印工作表时的缩放比例，数值在 10 和 400 之间。如果本属性设为 False，则由 FitToPagesWide 属性和 FitToPagesTall 属性的设定值对工作表的缩放进行控制。
预览和打印

- PrintPreview

  打印预览。执行该方法时，如果将参数设为 False，则不允许在预览时进行打印设置。

  如：Ws.PrintPreview(False)

  注意，在执行打印预览时，必须先将 Application 的 Visible 设置为 True！

- PrintOut

  直接打印。如，Ws.PrintOut

  该方法还可通过设置有关参数来控制打印，如下面的代码就是打印第 1-10 页，打印份数为 3，打印前先预览，指定的打印机为"pdfFactory Pro"：

  Ws.PrintOut(From:=1,To:=10,Copies:=3,Preview:=True,ActivePrinter:="pdfFactory Pro")

换页打印
利用 PageBreak 属性可手工控制换页。

如 在工作表的第 24 行后设置一个手动分页符：
Ws.Rows(25).PageBreak = MSExcel.XlPageBreak.xlPageBreakManual

如 在工作表的第 3 列左侧设置一个手动分页符：
Ws.Columns(3).PageBreak = MSExcel.XlPageBreak.xlPageBreakManual

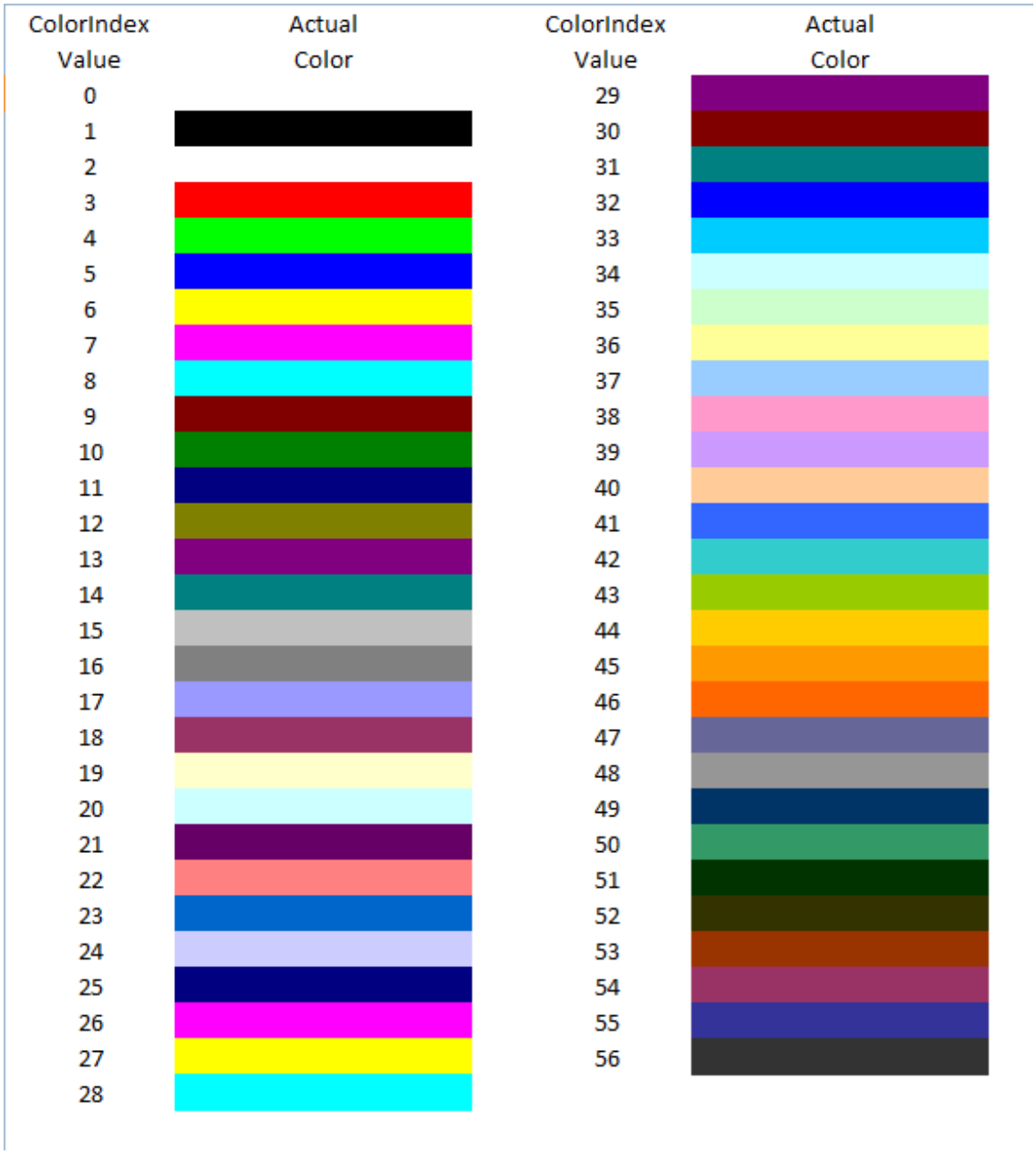
清除所有的手动分页符：
Ws.Cells.PageBreak = MSExcel.XlPageBreak.xlPageBreakNone

下面的例子就是对有内容的区域，每打印 5 行就换页：

```
Dim App As New MSExcel.Application
Dim Wb As MSExcel.WorkBook = App.WorkBooks.Open("D:\Report.xls")
Dim Ws As MSExcel.WorkSheet = Wb.WorkSheets(1)
'对 A 列从 65536 行开始向上查找,直到找到最后一个非空单元格为止,并得到其行号.也就是有内容的结束行
Dim EndRow As Integer = Ws.Range("A65536").End(MSExcel.XlDirection.xlUp).Row
'对 A 列从第 1 行开始向下查找,直到找到最后一个非空单元格为止,并得到其行号.也就是有内容的开始行
Dim FirstRow As Integer = Ws.Range("A1").End(MSExcel.XlDirection.xlDown).Row
'对 A 列从第 1 行开始向下查找,直到找到最后一个非空单元格为止,并得到其行号.也就是有内容的开始行
Ws.Cells.PageBreak = MSExcel.XlPageBreak.xlPageBreakNone'清除所有分页符
For i As Integer = FirstRow+6 To EndRow Step 5   '每 5 行就分页
    Ws.Rows(i).PageBreak = MSExcel.XlPageBreak.xlPageBreakManual
Next
App.Visible = True
Ws.PrintPreview
App.Quit
```

附件

Excel 的 ColorIndex(颜色编号)对照表：



9.7 Word 和 VBA

Foxtable 内置了 Word 的 VBA 支持，命名空间为 MSWord。
例如：

```
Dim app As New MSWord.Application
app.Documents.Open("c:\data\blank.doc")
app.Visible = True
```

关于Word和VBA，这里就不讲述了，有兴趣的话，可以自行在网上找些文档参考学习即可。