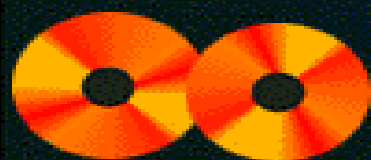




北京希望电脑公司出品

Visual Basic 6.0

中文版程序员指南



CD-ROM

“九五”国家重点电子出版物规划项目
希望计算机知识普及系列

Visual Basic 6.0 中文版程序员指南

Microsoft Corporation 著

微软（中国）有限公司 译

希望图书创作室 校

1 9 9 8

内 容 简 介

本书是美国微软出版社授权的 Visual Basic 6.0 系列手册之一，该书是这套系列书中的第一本。全书分两部分共十七章，还有四个附录，其主要内容包括：窗体、控件和菜单，工程管理、编程基础，创建用户界面，使用 Visual Basic 的标准控件，用对象和 ActiveX 部件编程，响应鼠标和键盘事件，使用文本和图形，调试代码和处理错误，处理驱动器、文件夹和文件、发布应用程序，Visual Basic 的规范、限制和文件格式，Visual Basic 编码规定及代码编译器开关。

本书既是 Visual Basic 6.0 软件的配套书，又是 Visual Basic 6.0 应用和开发人员必备的手册，同时也是大专院校相关专业师生的自学、教学参考书。

为方便读者，随书附配套电子书，本电子书的用法，请参见 CD 盘中的 Readme 文件。

欲购本书或需要技术支持的用户可直接与北京海淀 8721 信箱书刊部（邮编 100080）联系，电话 010-62562329，010-62541992，010-62531267 或传真至 010-62579874。

本书封四贴有微软出版社（Microsoft Press）防伪标签，无标签者不得出售，违者必究。

版 权 声 明

本书英文版名为《Visual Basic 6.0 Programmer's Guide》，由 Microsoft 出版社出版，版权归 Microsoft 出版社所有。本书中文版由 Microsoft 出版社授权出版。未经出版者书面许可，本书的任何部分不得以任何手段复制或传播。

Visual Basic 6.0 中文版程序员指南

Microsoft Corporation 著

微软（中国）有限公司 译

希望图书创作室 校

责任编辑 陆卫民

北京希望电脑公司 出品

北京海淀路 82 号（100080）

印刷

新华书店 新华书店音像发行所 各地书店及软件专卖店经销

* * * * *

1998 年 9 月第一版 开本：787×1092 1/16

1998 年 9 月第一次印刷 印张：54.375

字数：898 千字 印数：1~5000

新出音管[1998] 164 号

ISBN 7-980021-22-3/TP·15

定价：98 元（1CD，含配套书）

目 录

文 档 约 定	29
前 言	33
第一部分 Visual Basic 基础	37
第一章 Visual Basic 概述	39
欢迎使用 Visual Basic	39
安装 Visual Basic	42
运行安装之前	42
安装 Visual Basic	44
实际工作中的助手	45
上下文相关帮助	47
运行“帮助”中的代码示例	47
Visual Basic 联机链接	49
Microsoft 产品支持服务	50
第二章 用 Visual Basic 开发应用程序	51
Visual Basic 概念	52
集成开发环境的元素	55
启动 Visual Basic IDE	55
集成开发环境的元素	57

环境选项	62
SDI 或 MDI 界面	62
第一个 Visual Basic 应用程序	64
编写代码	72
Firstapp 示例应用程序	75
第三章 窗体、控件和菜单	81
属性、方法和事件概述	82
设计窗体	85
单击按钮执行操作	89
用于显示和输入文本的控件	91
用 Label 显示文本	92
为用户提供选择的控件	101
用 CheckBox 选择单个选项	101
用选项按钮进行选项分组	105
显示图片和图形的控件	115
使用 PictureBox 控件	115
轻量图形控件	118
Images 应用程序	121
附加控件	124
焦点概述	127
设置 Tab 键顺序	129
菜单基础	131

用对话框提示用户	136
第四章 工 程 管 理	139
使用工程	140
Visual Basic 工程的结构	143
创建、打开和保存工程	147
添加、删除和保存文件	149
在工程中添加控件	152
制作和运行可执行文件	158
设置工程选项	159
使用向导和外接程序	160
使用外接程序管理器	161
第五章 编 程 基 础	163
Visual Basic 应用程序的结构	164
事件驱动应用程序的工作方式	166
开始编码之前	167
代码编写机制	171
代码模块	172
使用“代码编辑器”	174
编码基础	179
变量、常数和数据类型概述	183
变量	184
理解变量的范围	189

高级变量主题	191
静态变量	196
常数	198
数据类型	201
高级变量主题	213
数组	221
动态数组	226
过程概述	229
Sub 过程	229
Function 过程	232
使用过程	234
向过程传递参数	238
控制结构概述	246
判定结构	246
循环结构	252
使用控制结构	257
使用对象	261
对象是什么	261
用对象能做什么	264
使用对象初步	265
对象之间的关系	270
创建对象	277

第二部分 Visual Basic 能做什么	289
第六章 创建用户界面	292
界面样式	293
多文档界面 (MDI) 应用程序	299
设计时使用 MDI 子窗体	301
MDI 窗体运行时的特性	302
MDI NotePad 应用程序	304
再论窗体	312
在应用程序中使用菜单	317
用菜单编辑器创建菜单	317
菜单标题与命名准则	324
创建子菜单	326
创建菜单控件数组	327
运行时创建和修改菜单	328
编写菜单控件的代码	333
显示弹出式菜单	333
MDI 应用程序中的菜单	336
工具栏	342
创建工具栏	342
协调菜单与工具栏的外观	346
对话框	348
模式与无模式的对话框	349

预定义对话框的使用	350
用窗体作为自定义对话框	353
各种显示类型的设计	361
设计时心里想着用户	364
界面设计的基础	365
可用性设计	372
当用户或系统出错时与用户交互	375
设计用户辅助模式	379

第七章 使用 **Visual Basic** 的标准控件 **385**

Visual Basic 控件简介	387
内部控件	389
标准 ActiveX 控件	393
更新旧版 Visual Basic 控件	395
控件命名约定	396
使用控件值	397
通过限制焦点验证控件数据	399
使用控件数组	400
设计时创建控件数组	403
运行时添加控件数组	405
控件方案：在控件数组中添加和删除控件	406
使用 ADO Data 控件	410
设置 ADO Data 控件的与数据库相关的属性	415

绑定到 ADO Data 控件的控件	418
使用 CheckBox 控件	419
使用 ComboBox 控件	422
使用 CommandButton 控件	432
使用 CommonDialog 控件	436
显示“打开”和“另存为”对话框	437
使用“颜色”对话框	440
使用“字体”对话框	442
使用“打印”对话框	446
使用 ShowHelp 方法显示帮助文件	449
使用 Data 控件	450
使用 DataCombo 和 DataList 控件	453
使用 DataCombo 和 DataList 控件连接两个表	455
创建一个简单的 DataCombo 应用程序	458
使用 DataGrid 控件	461
创建 Northwind 的 OLE DB 数据链接	466
使用 DataGrid 和 ADO Data 控件创建一个简单的数据库应用程序	468
使用列	473
操作 DataGrid 视图	475
使用 Bookmarks 和 SelBookmarks 跟踪记录	480
显示计算结果字段	481
与类模块一起使用 DataGrid 控件	482

使用 文件系统控件 (Directory List Box、Drive List Box 和 File List Box)	489
检查文件系统	489
驱动器列表框	491
目录列表框	492
标识单个目录	492
文件列表框	495
使用文件系统控件的组合	497
文件系统控件方案：文件搜索器应用程序	498
使用 Frame 控件	504
使用 Microsoft Hierarchical FlexGrid 控件	507
将数据绑定到 Hierarchical FlexGrid	510
Hierarchical FlexGrid 与分层结构的 Recordset	514
带区的格式设置	515
使用带区的扩展和收缩功能	523
获取结构信息	526
定制 Hierarchical FlexGrid 中的各个区域	528
将 Hierarchical FlexGrid 绑定到一个 Data Environment 上	534
对 Hierarchical FlexGrid 中的数据进行合并与排序	535
在 Hierarchical FlexGrid 电子表格中编辑单元	542
使用 Hierarchical FlexGrid 显示一个带有标头的大纲	551
使用 HScrollBar 和 VScrollBar 控件	559
Scroll Bar 控件方案：创建可滚动的图形视口	562

使用 Image 控件	568
使用 Label 控件	572
使用 Line 控件	575
使用 ListBox 控件	578
列表框控件方案 1：添加和删除项目	587
列表框控件方案 2：创建多列列表框	591
使用 OLE 容器控件	595
使用 OptionButton 控件	597
使用 PictureBox 控件	600
使用 RemoteData 控件	604
使用 Shape 控件	608
使用 TextBox 控件	612
使用 Timer 控件	616
Timer 控件方案：响应 Timer 事件	619
第八章 再论编程	622
利用多个工程进行工作	623
管理应用程序设置值	627
创建或保存应用程序设置值	628
检索应用程序设置值	629
删除应用程序设置值	631
使用条件编译	632
利用资源文件进行工作	635

利用模板进行工作	637
利用命令行开关进行工作	641
将工程编译成本机代码	642
创建自己的数据类型	646
通过枚举使用常数集进行工作	652
数组的高级功能	656
用集合替代数组	660
第九章 用对象编程	662
Visual Basic 中的对象	663
与对象有关的术语	663
找出对象所属的类	665
使用字符串名调用属性和方法	666
对一个对象执行多个动作	668
使用缺省属性	669
创建对象数组	672
创建对象集合	675
Visual Basic 集合对象	677
Visual Basic 中的集合	685
查找对象信息	691
对象浏览器的内容	693
查找和浏览对象	696
添加对象的描述	699

在过程之间移动	700
浏览其它应用程序的对象	701
创建自己的类	701
类：将用户定义类型和过程组织在一起	702
定制窗体类	705
类模块的循序渐进	710
调试类模块	718
Visual Basic 窗体的存活期	720
类模块和标准模块的比较	728
向类中添加属性和方法	733
向类中添加属性	734
属性过程与公共变量的比较	737
运用属性过程	738
向类中添加方法	748
是属性还是方法	750
使属性或方法成为缺省的	753
友元属性和方法	755
向类中添加事件	758
声明和引发事件	759
处理对象的事件	762
WithEvents 与窗体上的控件事件的比较	769
向窗体添加事件	770

声明、引发以及处理事件的总结	772
创建数据识别类	773
创建数据源	776
创建数据使用者	782
命名属性、方法和事件	785
多态	787
Visual Basic 是如何提供多态的	787
创建和实现接口	789
实现属性	796
关于对象和接口的简要补充讨论	799
代码重用的许多（内部）方面	804
用自己的对象编程	808
对象引用和引用计数	809
对象模型	810
创建自己的集合类	819
公有集合示例：稻草盖的房子	820
私有集合示例：树枝盖的房子	827
创建自己的集合类：砖块盖的房子	833
好的面向对象程序设计的优点	839
ActiveX 设计器	842
什么是 ActiveX 设计器	842
向“工程”菜单中添加 ActiveX 设计器	848

插入 ActiveX 设计器的新实例	850

第十章 用 部 件 编 程	852

ActiveX 部件的类型	853

进程内的服务器和进程外的服务器	855

使用 ActiveX 部件	856

创建对对象的引用	859

浏览 ActiveX 部件类型库	865

声明对象变量	867

将对象引用赋予变量	869

加速对象引用	873

使用对象的属性、方法和事件	876

释放 ActiveX 部件	880

定位对象模型	881

使用外部可创建对象和从属对象	886

处理 ActiveX 部件中的运行时错误	890

处理挂起 ActiveX 部件的请求	893

改变部件忙或请求挂起消息	896

控制超时间隔	900

产生部件忙超时错误	902

使用部件的可视界面	903

比较链接对象与嵌入对象	906

设计时用 OLE 容器控件插入对象	910

运行时用 OLE 容器控件创建对象	916
决定在 OLE 容器控件中如何显示对象	921
激活 OLE 容器控件中对象	922
响应移动容器或调整容器大小	923
保存和检索嵌入的数据	924

第十一章 响应鼠标和键盘事件 927

响应鼠标事件	928
AMouseDown 事件	929
MouseMove 事件	932
MouseUp 事件	935
检测鼠标按钮	937
结合 MouseDown 和 MouseUp 事件使用 Button 参数	940
结合 MouseMove 使用 Button 参数	941
使用 Button 参数增强图形鼠标应用程序	944
检测 SHIFT、CTRL 及 ALT 键的状态	946
拖放	949
启动自动拖动模式	950
改变拖动图标	950
放下对象时的响应	951
启动拖动或停止拖动时的控制	953
改变控件的位置	955
OLE 拖放	958

启动自动 OLE 拖放	961
OLE 拖放 DataObject 对象	963
OLE 拖放如何工作	966
启动 OLE 拖动操作	969
将 OLE 拖动源拖到 OLE 拖放目标的上方	973
将 OLE 拖动源放到 OLE 拖放目标上	980
用鼠标和键盘修改放下效果和用户反馈	984
创建自定义数据格式	986
从 Windows“资源管理器”中拖动文件	987
自定义鼠标指针	990
响应键盘事件	993
编写低级键盘处理程序	994
KeyPress 事件	996
KeyDown 和 KeyUp 事件	998
编写窗体级键盘处理程序	1002
中断后台处理	1005
允许用户中断任务	1006
使用 DoEvents	1007
使用文本和图形	1012
使用字体	1013
设置字体特征	1016
在窗体和图片框上显示文本	1020

在单行上显示不同的项	1022
在指定位置显示 Print 输出	1023
格式化数字、日期和时间	1026
使用选定的文本	1031
使用剪贴板对象传送文本和图形	1032
使用剪贴板剪切、复制和粘贴文本	1033
在剪贴板上使用多种格式	1036
检查剪贴板上的数据格式	1039
坐标系统概述	1041
改变对象的坐标系统	1043
刻度的转换	1049
使用图形控件	1051
给应用程序添加图片	1052
窗体和控件的图形属性概述	1057
用 AutoRedraw 创建持久的图形	1058
用 ClipControls 裁剪区域	1063
用 AutoRedraw 和 ClipControls 使图形分层	1065
动态地移动控件	1070
动态地改变控件大小	1076
创建简单动画	1077
使用图形方法	1079
图形方法画图的原理	1081

画点	1082
画各种直线和形状	1085
绘制方框	1088
画圆	1090
画椭圆	1094
在任意位置绘制图形	1096
指定线宽	1097
指定实线或虚线	1099
DrawMode 的显示控制作用	1101
窗体加载时创建图形	1104
使用颜色	1105
使用颜色属性	1106
使用 256 种颜色	1109
管理多个调色板	1112
使用图片对象	1119
打印	1121
从应用程序中打印	1122
使用 Printer 对象打印	1125
捕获打印机错误	1132
第十三章 调试代码和处理错误	1133
如何处理错误	1135
设计错误处理程序	1139

错误处理的分层结构	1147
通过生成错误测试错误处理	1154
联机错误处理	1156
集中化的错误处理	1162
关闭错误处理	1170
含 ActiveX 部件的错误处理	1172
调试方法	1178
避免错误	1183
设计时、运行时以及中断模式	1184
使用调试窗口	1187
使用中断模式	1191
在有问题的语句处进入中断模式	1191
用监视表达式监视数据	1195
编辑或删除监视表达式	1199
用断点有选择地中断执行	1202
使用 Stop 语句	1205
运行应用程序的选定部分	1206
使用跟踪	1207
监视调用堆栈	1211
用“立即”窗口测试数据和过程	1215
在“立即”窗口中打印信息	1215
为变量和属性赋值	1220

用“立即”窗口测试过程	1221
检查错误号	1224
使用“立即”窗口的提示	1225
特别的调试考虑	1227
在编译之前删除调试信息	1230
用断言检验代码	1231
使用请求时编译	1232
调试的提示	1233
第十四章 处理驱动器、文件夹和文件	1237
File System Object 模型介绍	1237
文件系统对象(File System Object)	1238
FSO 对象模型编程	1240
使用驱动器和文件夹	1243
使用文件	1249
用传统的文件 I/O 语句和函数处理文件	1256
使用顺序文件访问	1258
使用随机文件访问	1262
使用二进制文件访问	1267
第十五章 性能和兼容性设计	1270
优化	1271
优化速度	1274
优化代码	1275

性能评测	1283
优化显示速度	1284
优化感觉速度	1288
优化大小	1293
减小代码大小	1293
修剪图形	1300
分段应用程序	1303
优化对象	1308
编译型和解释型应用程序的比较	1312
与其它 Microsoft 应用程序的兼容性	1314
第十六章 国 际 化	1317
国际化软件的定义	1318
设计国际化软件	1320
使用本地化资源文件	1323
自动取款机示例应用程序	1325
设计区域识别的用户界面	1328
编写国际化代码时的一般考虑	1333
用 Visual Basic 编写国际化代码	1336
国际排序顺序和字符串比较	1341
国际文件的输入 / 输出	1346
基于日期的区域识别的 SQL 查询	1348
双字节字符集 (DBCS) 的特殊问题	1350

ANSI、DBCS 和 Unicode 的定义	1352
DBCS 排序顺序和字符串比较	1354
DBCS 字符串操作函数	1355
DBCS 环境下字体、显示和打印的考虑因素	1359
处理使用双字节字符的文件	1362
DBCS 环境的标识符	1363
允许 DBCS 的 KeyPress 事件	1364
调用 Windows API 函数	1364

Visual Basic 的双向特征	1365

第十七章 发布应用程序 1367

打包和部署向导	1368
启动打包和部署向导	1370
使用向导为应用程序打包	1374
打包功能	1377
标准软件包	1379
Internet 软件包	1383
从属文件	1386
使用向导部署应用程序	1389
部署功能	1391
部署应用程序	1391
管理向导脚本	1393
安装工具包 (Setup Toolkit).....	1394

修改安装工程	1397
需要发布的文件	1398
将文件安装到用户机器的什么位置	1401
手工编辑 Setup.lst 文件	1403
Bootstrap 和 Setup1 Files 段的格式	1408
Setup.lst 文件中的路径参数宏	1410
Setup.lst 文件中的注册键	1411
指定 Setup.lst 文件中的远程服务器部件	1412
手工创建发布媒体	1413
创建发布媒体	1414
与安装工具包一起使用打包和部署向导	1416
测试您的安装程序	1418
允许用户删除应用程序	1419
部署本地化的 ActiveX 控件	1422
附录 A Visual Basic 的规范、限制和文件格式	1424
Visual Basic 应用程序的系统需求	1424
工程限制	1425
控件限制	1426
代码限制	1427
数据限制	1429
系统资源限制	1431
工程文件格式	1432

窗体结构	1435
版本号	1437
窗体描述	1438
窗体描述属性	1442
ASCII 窗体示例	1445
加载窗体文件时的错误	1451
窗体加载错误日志消息	1452
工程文件 (.vbp) 格式	1456

附录 B Visual Basic 编码约定 1459

为什么要进行编码约定	1460
对象命名约定	1461
常量和变量命名约定	1467
结构化编码约定	1472

附录 C 本地代码编译器开关 1477

优化快速代码	1478
优化小代码	1478
不优化	1479
支持 Pentium Pro	1479
创建符号调试信息	1480
假设没有别名	1480
取消数组边界检查	1481
取消整数溢出检查	1481

取消浮点数错误检查	1482
取消对 Pentium FDIV 的安全检查	1482
允许无四舍五入的浮点操作	1483
附录 D 为应用程序添加帮助	1485
添加对帮助的支持	1486
添加对“这是什么”帮助的支持	1489
与应用程序一同发布帮助	1490

文档约定

本书使用了以下印刷约定。

约定示例	描述
Sub, If, Case Else, Print, True, BackColor, Click, Debug, Long <i>[expressionlist]</i> { W hile U ntil }	首字母大写的单词表示语言特定的关键字。 在语法说明中，方括号 ([]) 内部的项是可选的。 在语法说明中，花括号 ({ }) 和竖线 () 表示可在两个或者多个选项中选择一个。除非所有的选项又包含在方括号 ([]) 中，否则必须选择一个选项。
ENTER	小号大写字母表示键的名字和键的组合，例如 ENTER 和 CTRL+R。
ALT+F1	键名之间的加号 (+) 表示键组合。例如，ALT+F1 表示在按 ALT 同时按 F1 键。
DOWN ARROW	按照键上的箭头方向使用单个方向键 (LEFT、RIGHT、UP 或 DOWN)。“箭头键”是这些键的统称。
BACKSPACE, HOME	按照各自的名字使用其它定位键。
c:\Vb\Samples\Calldlls.vbp	大小写混合的路径和文件名。

本书中的编程样式

在本书中，所有的程序都遵循下面的规则。详细信息请参阅本书第五章“编程基础”。

- 关键字的首字母全部大写：
`'Sub, If, ChDir, Print` 和 `True` 是关键字。
`Print "Title Page"`
- 使用行标号，不使用行号。仅在错误处理例程中使用行标号：
`ErrorHandler:`
`Power = conFailure`
`End Function`
- 使用撇号（'）引导注释：
`'这是注释部分；在程序运行时，这两行`
`'将被忽略。`
- 在子过程、函数过程和属性过程中的控制流块和语句需要缩进：
`Private Sub cmdRemove_Click ()`
`Dim Ind As Integer`

`Ind = lstClient.ListIndex` `'取得索引。`
`'确定有选定的项。`
`If Ind >= 0 Then`

```

        l
End SubstClient.RemoveItem Ind '将其从列表框中删除。
    '显示编号。
    lblDisplay.Caption = lstClient.ListCount
Else
    Beep '如果没有项被选中，就发出鸣响。
End If

```

- 如果一句代码太长，无法在一行内写下（不包括注释），则可以在下一行继续，在行的末尾用续行字符表示一行尚未结束，续行字符是一个空格加一个下划字符（_）：

```

Sub Form_MouseDown (Button As Integer, _
Shift As Integer, X As Single, Y As Single)

```

- 内部的常数名是大小写混合的，它的前缀是两个字符，表示定义该常数的对象库。Visual Basic (VB) 和 Visual Basic for application (VBA) 对象库中的常数的前缀为 vb；数据访问对象 (DAO) 库中的常数的前缀为 db；Microsoft Excel 对象库中的常数的前缀为 xl。下面是几个例子：

```

vbTileHorizontal
dbAppendOnly
xlDialogBorder

```

在本书中，用户自定义的常数的前缀通常为 con，而且是大小写混合的。例如：

```

conYourOwnConstant

```

然而，在使用 Windows API 常数时，代码仍遵循 Visual Basic 的以前版本中使用的约定。例如：

EM_LINESCROLL

前 言

欢迎阅读 Visual Basic® 程序员指南，它是有关用 Visual Basic 编程的综合性手册。为了介绍 Visual Basic 丰富的特性和功能，本程序员指南将分为两部分。

第一部分包括了一些基本概念，如果您是对 Visual Basic 陌生的程序员，那么这部分对您会很有帮助，因为它提供了基础的知识。第二部分包括更高级的编程概念和技术。附录中提供了对使用本产品有帮助的附加信息。

第一部分 Visual Basic 基础

在 Visual Basic 中编程的介绍。

第二部分 用 Visual Basic 能做什么

有关 Visual Basic 编程的高级主题。

附录 A Visual Basic 的规范、限制和文件格式

Visual Basic 的技术细节。

附录 B Visual Basic 编码约定

编写一致、可读性强的代码建议指南。

附录 C 本机代码编译器开关

有关编译本机代码的命令行开关的细节。

附录 D 为应用程序添加帮助

为 Visual Basic 应用程序添加帮助的指南。

序

为了满足广大电脑爱好者学习 Visual Basic 6.0 中文版软件的需要，及配合微软（中国）有限公司发布 Visual Basic 6.0 中文版，美国微软出版社授权翻译、出版以下 Visual Basic 6.0 系列手册：

编号	书名	估计定价（元）
CX-2463	Visual Basic 6.0 中文版程序员指南	98
CX-2464	Visual Basic 6.0 中文版语言参考手册	120
CX-2542	Visual Basic 6.0 中文版控件参考手册	70
CX-2453	Visual Basic 6.0 中文版组件工具指南	70
CX-2477	Visual Basic 6.0 中文版编程基础	62
CX-2483	Visual Basic 6.0 中文版技巧	70
CX-2484	Visual Basic 6.0 中文版与 SQL Sever 编程	78
CX-2485	Visual Basic 6.0 中文版高级程序设计	78
CX-2486	Visual Basic 6.0 中文版组件开发指南	62

该系列书具备权威性、准确性和启发性，内容全面，是 VB 编程用户的必备工具书，欢迎各界新老朋友订购。

欲购书或需技术支持的读者，请直接与北京希望电脑公司联系，联系电话：
(010)- 62562329, 62541992 或传真至 (010)- 62579874，或写信至北京海
淀 8721 信箱书刊部 (邮编 100080)。

希望图书创作室

1998 年 9 月

第一部分 Visual Basic 基础

Visual Basic 所做的很多事情一点也不简单。它是一种强大的语言，您所能想到的编程任务，它基本都能完成。当然要想成为大师还需要学很多东西，但只要学了 Visual Basic 的基础知识，你的创造力就能迅速得到充分的发挥。

本书的前五章介绍基础知识，为使用 Visual Basic 编程打下基础。

第一章 Visual Basic 概述

说明如何安装 Visual Basic 及在工作时如何取得帮助。

第二章 用 Visual Basic 开发应用程序

介绍集成开发环境及创建第一个应用程序的过程。

第三章 窗体、控件和菜单

介绍创建应用程序所拼装的对象。

第四章 工程管理

介绍在 Visual Basic 中用于组织工作的工具。

第五章 编程基础

介绍 Visual Basic 语言要点。

第一章 Visual Basic 概述

本章的内容包括如何安装 Visual Basic，添加或删除 Visual Basic 部件，Visual Basic 的学习资料及如何得到更多的帮助。

目录

- 欢迎使用 Visual Basic
- 安装 Visual Basic
- 实际工作中的助手

欢迎使用 Visual Basic

欢迎使用 Microsoft Visual Basic，它提供了开发 Microsoft Windows(R) 应用程序最迅速、最简捷的方法。不论是 Microsoft Windows 应用程序的资深专业开发人员还是初学者，Visual Basic 都为他们提供了整套工具，以方便开发应用程序。

何谓 Visual Basic？“Visual”指的是开发图形用户界面 (GUI) 的方法。不

需编写大量代码去描述界面元素的外观和位置，而只要把预先建立的对象 add 到屏幕上的一点即可。如果已使用过诸如 Paint 之类的绘图程序，则实际上已掌握了创建用户界面的必要技巧。

“Basic”指的是 BASIC (Beginners All-Purpose Symbolit Instruction Code) 语言，一种在计算技术发展史上应用得最为广泛的语言。Visual Basic 在原有 BASIC 语言的基础上进一步发展，至今包含了数百条语句、函数及关键词，其中很多和 Windows GUI 有直接关系。专业人员可以用 Visual Basic 实现其它任何 Windows 编程语言的功能，而初学者只要掌握几个关键词就可以建立实用的应用程序。

Visual Basic 不仅是 Visual Basic 编程语言。Visual Basic 编程系统,Microsoft Excel 的 Applications Edition,Microsoft Access 和 Windows 的许多其他应用程序都使用这一语言。Visual Basic Scripting Edition (VBScript) 是广泛使用的脚本语言，它是 Visual Basic 语言的子集。这样，在学习 Visual Basic 中得到的经验可应用到所有这些领域中。

从开发个人使用或小组使用的小工具，到大型企业应用系统，甚至通过 Internet 的遍及全球分布式应用程序，都可在 Visual Basic 提供的工具中各取所需。

- 数据访问特性允许对包括 Microsoft SQL Server 和其它企业数据库在内的大部分数据库格式建立数据库和前端应用程序，以及可调整的服务器端部件。
- 有了 ActiveX 技术就可使用其它应用程序提供的功能，例如 Microsoft Word 字处理器，Microsoft Excel 电子数据表及其它 Windows 应用程

序。甚至可直接使用 VBP 或 VBE 创建的应用程序和对象。

- Internet 功能强大，使得您很容易在应用程序内通过 Internet 或 Intranet 访问文档和应用程序，或者创建 Internet 服务器应用程序。
- 已完成的应用程序是使用 Visual Basic 虚拟机的真正 .exe 文件，您可以自由发布。

Visual Basic 版本简介

Visual Basic 有三种版本，可以满足不同的开发需要。

- Visual Basic 学习版使编程人员轻松开发 Windows 和 Windows NT(r) 的应用程序。该版本包括所有的内部控件以及网格、选项卡和数据绑定控件。学习版提供的文档有 Learn VB Now CD 和包含全部联机文档的 Microsoft Developer Network CD。
- 专业版为专业编程人员提供了一整套功能完备的开发工具。该版本包括学习版的全部功能以及 ActiveX 控件、Internet Information Server Application Designer、集成的 Visual Database Tools 和 Data Environment、Active Data Objects 和 Dynamic HTML Page Designer。专业版提供的文档有 Visual Studio Professional Features 手册和包含全部联机文档的 Microsoft Developer Network CD。
- 企业版使得专业编程人员能够开发功能强大的组内分布式应用程序。该版本包括专业版的全部功能以及 Back Office 工具，例如 SQL Server、

Microsoft Transaction Server、Internet Information Server、Visual SourceSafe、SNA Server 等。企业版包括的印刷文档包括 Visual Studio Enterprise Features 手册以及包含全部联机文档的 Microsoft Developer Network CD。

安装 Visual Basic

使用安装 (Setup) 程序来安装 Visual Basic。安装程序将 Visual Basic 以及其它产品部件从 CD-ROM 安装到硬盘上。它还安装必要的文件以查看 Microsoft Developer Network CD 中的文档。您可以选择只安装 Visual Basic 文档和示例程序到您的计算机上。

重点 不能直接将 CD-ROM 上的文件复制到硬盘，然后从硬盘运行 Visual Basic。必须使用安装程序将文件解压缩并安装到合适的目录中。

- 在运行安装之前 (vbconBeforeYouRunSetup) 安装之前的检查。
- 安装 Visual (BasicvbconSettingUpVisualBasic) 安装 Visual Basic。

运行安装之前

在安装 Visual Basic 之前，必须确认计算机满足最低的安装要求，并阅读安装盘根目录下的 Readme 文件。

检查硬件和系统需求

为运行 Visual Basic，必须在计算机上安装相应的硬件和软件系统。这些系统要求包括：

- Microsoft Windows NT 95 或更高版本，或 Microsoft Windows NT Workstation 4.0（推荐 Service Pack 3）或更高版本。
- 486DX/66 MHz 或更高的处理器（推荐 Pentium 或更高的处理器），或任何运行于 Microsoft Windows NT Workstation 的 Alpha 处理器。
- 一个 CD-ROM 驱动器。
- Microsoft Windows 支持的 VGA 或分辨率更高的监视器。
- 16 MB RAM for Windows 95、32 MB of RAM for Windows NT Workstation。
- 鼠标或其它定点设备。

详细信息 关于系统安装要求的详细信息，请参阅附录 A 的“Visual Basic 的规范、限制和文件格式”中的“Visual Basic 应用程序的系统需求”。

阅读 Readme 文件

Readme 文件列举了自 Visual Basic 文档出版发行以来的更改。可在最初的安装屏幕上选取“Read Me First”来读取该文件，也可在 CD-ROM 的根目录下查找到这个文件。也可以从文档中的 Visual Basic Start Page 访问。文件的第一部分是有关安装 Visual Basic 的细节和新的信息，请检查一下。

安装 Visual Basic

运行安装程序时将会为 Visual Basic 创建目录，从而可选择要安装的 Visual Basic 部件。

除了 \Os 目录下的操作系统文件，CD 上的其它文件是没有被压缩的，所以可从盘上直接使用。例如，在 \Tools 目录下有许多的工具和部件，可以直接从 CD-ROM 运行或安装。

要从 CD 盘上安装，请按照以下步骤执行：

1. 在 CD-ROM 驱动器中插入 CD 盘。
2. 安装程序在 Disk 1 的根目录下，可用操作系统中的适当命令来运行。如果您的计算机能够在系统中运行 AutoPlay，则在插入 CD 盘时，安装程序将被自动加载。
3. 选取“安装 Visual Basic 6.0”。
4. 依照屏幕上的安装说明行事。

详细信息 有关安装 Visual Basic 的详细信息，请阅读 Readme 文件。

添加或删除 Visual Basic 部件

必要时可随意多次运行安装程序。例如，可在另一个目录下重新安装 Visual Basic，也可安装 Visual Basic 的其它部分。

要添加或删除 Visual Basic 部件，请按照以下步骤执行：

1. 在 CD-ROM 驱动器插入 CD 盘。

2. 安装程序在 CD 盘的根目录下，可用操作系统中的适当命令来运行。如果您的计算机能够在系统中运行 AutoPlay，则在插入 CD 盘时，安装程序将被自动加载。
3. 在“Microsoft Visual Basic 6.0 安装”对话框中选取“自定义”按钮。
4. 在“自定义”对话框的“选项”列表框中选定要安装的部件（或撤消选定要删除的部件）。
5. 依照屏幕上的安装说明行事。

启动 Visual Basic

一旦完成安装过程，就可单击 Windows 任务栏中的开始按钮，启动 Visual Basic。如果在系统中可启动 AutoPlay，则也可将 Visual Basic CD 插入驱动器中来启动 Visual Basic。

详细信息 请参阅第二章“用 Visual Basic 开发应用程序”。

实际工作中的助手

联机文档系统对 Visual Basic 提供的帮助可算得上是面面俱到。其内容包括：

- Visual Basic 的所有手册，提供了有关使用 Visual Basic 强大功能的概念性的信息。

- 语言参考，包括 Visual Basic 编程环境和广泛的语言内容的信息。
- Visual Basic 联机链接，提供指向 World Wide Web 中 Visual Basic 信息资源的指针。
- Microsoft 产品支持服务，提供技术支持信息。

注意 您可以从 MSDN CD 中查看文档（必须经过 MSDN 安装过程），或者在 MSDN 安装过程中，选择自定义 Visual Basic 安装，只安装文档和示例到您计算机上。

阅读帮助的内容

帮助的内容包括许多为阅读方便而设计的特性。

- Visual Basic 6 的新增内容
利用这一部分可以很快得知 Visual Basic 新增和增强了哪些功能。按功能分类，约有 200 个描述链接来提供详细信息。
- 快速查找
利用这一部分，可将覆盖整个文档的主题进行排序。例如，根据您的操作的工程种类的不同，可以找到各种各样的调试信息。这一部分的描述链接，使得检索更加容易。
- 主题摘要
在阅读各种主题之前，可以先了解手册中或各章中所含主题的摘要。由于提供了每个主题的内容提要，在书的开头、每篇或每章开头的摘要部分的描述链接，节省了用户的大量时间。

- 请参阅的链接
单击主题下面的“请参阅”链接，查看有关该主题的更详细的或相关的信息。

上下文相关帮助

Visual Basic 的许多部分是上下文相关的。上下文相关意味着不必搜寻“帮助”菜单就可直接获得有关这些部分的帮助。例如，为了获得有关 Visual Basic 语言中任何关键词的 Help，只须将插入点置于“代码”窗口中的关键词上并按 F1 键。

在 Visual Basic 界面的任何上下文相关部分上按 F1 键，就可显示有关该部分的信息。上下文相关部分是：

- Visual Basic 中的每个窗口（“属性”窗口、“代码”窗口等）
- 工具箱中的控件
- 窗体或文档对象内的对象
- “属性”窗口中的属性
- Visual Basic 关键词（语句、函数、属性、方法、事件和特殊对象）
- 错误信息

一旦打开“帮助”，按 F1 键就可获得怎样使用帮助的信息。

运行“帮助”中的代码示例

“帮助”中的许多程序语言主题包含代码示例，在 Visual Basic 中可运行它们。以下过程表明怎样从“帮助”中复制并运行代码示例。

注意 以下过程用于不含公开声明的代码示例。

要从“帮助”中复制一个代码示例，请按照以下步骤执行：

1. 从“工程”菜单中选择“添加窗体”，或使用现有窗体来创建新窗体（有关创建和使用窗体的详细信息，请参阅第二章，“用 Visual Basic 开发应用程序”。）
2. 从“帮助”菜单中选择“索引”。
3. 在“帮助”中搜索“图形”并转到“FillColor 属性”主题。
4. 在“FillColor 属性”主题中单击“示例”跳转按钮，它位于窗口顶部非滚动边缘（跳转按钮是一个词，单击它就可转到另一主题。跳转文本带下划线，并且而跳转文本有颜色）。

选择示例的子例程部分。注意，第一个 "Sub" 对子例程的开始作标记，最后一个 "End Sub" 对子例程的结束作标记。

5. 右键选择文本并从上下文菜单中选择“复制”。将文本复制到剪贴板中。
6. 返回所创建的窗体并双击该窗体以显示“代码”窗口。
7. 在“代码”窗口中的任何现有代码下设置插入点。
8. 在“编辑”菜单中选择“粘贴”。示例就出现在“代码”窗口中。
9. 在“运行”菜单中选择“启动”或按 F5 键。
10. 单击窗体，运行示例代码。

注意 有些代码示例需要在窗体中绘制控件。有关绘制控件的详细信息，请参阅第三章“窗体、控件和菜单”。

Visual Basic 联机链接

如果有调制解调器或其它访问手段，就可从 World Wide Web 中获得 Visual Basic 的更多信息。

Microsoft Web 站点

Microsoft World Wide Web 站点含有 Visual Basic 程序员感兴趣的几个区。Visual Basic 主页的地址为 <http://www.microsoft.com/vbasic/>。该站点所包含的信息如下：

- 新功能的更新、产品发布、相关产品、会议和特殊的事件。
- 有关 Visual Basic 功能的附加信息，包含白皮书、技巧和教程以及资源。
- New product downloads 包括产品支持服务中最常见问题的答案。

提示 Visual Basic web 站点还包含一个特定“Owner's Area”，对于注册的所有者(owners)来说，该站点包含许多免费的示例、部件、工具等等。希望您立刻到 <http://www.microsoft.com/vbasic/owners/> 这里注册您的 Visual Basic 版本。

要访问 Microsoft Visual Basic Web 站点，请按照以下步骤执行：

1. 从“帮助”菜单中选择“Web 上的 Microsoft”。
2. 从子菜单中选择合适的选项。

注意 在做以上工作之前必须安装 Web 浏览器并与 Internet 连接起来。

Microsoft Web 站点的某些内容可对 Microsoft Internet Explorer 最优

化，并可在 other 的浏览器上完全看到。可从 Web 站点卸载最新版的 Internet Explorer。

Microsoft 产品支持服务

Microsoft 提供了各种支持来帮助用户从 Visual Basic 获得更多东西。如果对产品感到有问题，则首先应查看联机文档。如果不能找到答案，则请与 Microsoft 产品支持服务部门联系。微软公司提供的支持服务遍及美国和全世界。有关详情，请参阅“在 Visual Basic ‘帮助’ 菜单下获得技术支持”。

把您的想法告诉我们

Microsoft 致力于尽可能向客户提供最优秀的产品。Visual Basic 的每个新版本都尽力满足 Windows 编程者不断变化的需求。我们期待听到客户的意见。如果您希望未来版本的 Visual Basic 在某些功能上有所改进，或希望它增加新的特性，请告诉我们。您可以使用 e-mail，我们的网址是 vbwish@microsoft.com，或者打电话到美国 (425) 936-WISH。

第二章 用 Visual Basic 开发应用程序

创建第一个 Visual Basic 应用程序只需要几分钟的时间。可以通过在窗体上“绘制”诸如文本框和命令按钮等控件来创建用户界面。然后，为窗体和控件设置属性以规定诸如标题、颜色和大小等的值。最后，编写代码将生命真正赋予应用程序。创建第一个应用程序所采取的基本步骤将是开发其它的应用程序所遵循的原则。

本章是应用程序开发过程的概述，描述使用 Visual Basic 所需要的术语和技巧，并且一步一步地指导创建几个简单的应用程序。

目录

- Visual Basic 概念
- 集成开发环境的元素
- 第一个 Visual Basic 应用程序

示例应用程序:Firstapp.vbp

本章的一些代码示例取自示例应用程序 Firstapp.vbp，该示例应用程序列在 Samples!Alink("vbsamples") 目录中。

Visual Basic 概念

为了理解应用程序开发过程，先要理解 Visual Basic 赖以创建的一些关键概念。因为 Visual Basic 是 Windows 开发语言，所以有必要与 Windows 环境保持一定的相似性。如果不熟悉 Windows 编程，就需要明白在 Windows 环境下编程和在其它环境下编程的一些根本性的差别。

Windows 的工作方式：窗口、事件和消息

全面地讨论 Windows 的内部工作机制将需要整整一本书的容量。没有必要深入了解所有的技术细节。Windows 的工作机制，简单地说就是三个关键的概念：窗口、事件和消息。

不妨简单地将窗口看作带有边界的矩形区域。也许已经了解几种不同类型的窗口：如，Windows 95 的“资源管理器”窗口、文字处理程序中的文档窗口或者弹出提示有约会信息的对话框。除了这些最普通的窗口外，实际上还有许多其它类型的窗口。命令按钮是一个窗口。图标、文本框、选项按钮和菜单条也都是窗口。

Microsoft Windows 操作系统通过给每一个窗口指定一个唯一的标识号（窗口句柄或 `hWnd`）来管理所有的窗口。操作系统连续地监视每一个窗口的活动或事件的信号。事件可以通过诸如单击鼠标或按下按键的操作而产生，也可以通过程序的控制而产生，甚至可以由另一个窗口的操作而产生。

每发生一次事件，将引发一条消息发送至操作系统。操作系统处理该消息并广播给其它窗口。然后，每一个窗口才能根据自身处理该条消息的指令而采取适当的操作（例如，当窗口解除了其它窗口的覆盖时，重显自身窗口）。

可以想像，处理各种窗口、事件和消息的所有可能的组合将有惊人的工作量。幸运的是，Visual Basic 使您摆脱了所有的低层消息处理。许多消息由 Visual Basic 自动处理了，其它的作为事件过程由编程者自行处理。这样可以快速创建强大的应用程序而毋需处理不必要的细节。

事件驱动模型

在传统的或“过程化”的应用程序中，应用程序自身控制了执行哪一部分代码和按何种顺序执行代码。从第一行代码执行程序并按应用程序中预定的路径执行，必要时调用过程。

在事件驱动的应用程序中，代码不是按照预定的路径执行，而是在响应不同的事件时执行不同的代码片段。事件可以由用户操作触发，也可以由来自操作系统或其它应用程序的消息触发，甚至由应用程序本身的消息触发。这些事件的顺序决定了代码执行的顺序，因此应用程序每次运行时所经过的代码的路径都是不同的。

因为事件的顺序是无法预测的，所以在代码中必须对执行时的“各种状态”作一定的假设。当作出某些假设时（例如，假设在运行来处理某一输入字段的过程之前，该输入字段必须包含确定的值），应该组织好应用程序的结

构，以确保该假设始终有效（例如，在输入字段中有值之前禁止使用启动该处理过程的命令按钮）。

在执行中代码也可以触发事件。例如，在程序中改变文本框中的文本将引发文本框的 `Change` 事件。如果 `Change` 事件中包含有代码，则将导致该代码的执行。如果原来假设该事件仅能由用户的交互操作所触发，则可能会产生意料之外的结果。正因为这一原因，所以在设计应用程序时理解事件驱动模型并牢记在心是非常重要的。

交互式开发

传统的应用程序开发过程可以分为三个明显的步骤：编码、编译和测试代码。但是 `Visual Basic` 与传统的语言不同，它使用交互式方法开发应用程序，使三个步骤之间不再有明显的界限。

在大多数语言里，如果编写代码时发生了错误，则在开始编译应用程序时该错误就会被编译器捕获。此时必须查找并改正该错误，然后再次进行编译，对每一个发现的错误都要重复这样的过程。`Visual Basic` 在编程者输入代码时便进行解释，即时捕获并突出显示大多数语法或拼写错误。看起来就像一位专家在监视代码的输入。

除即时捕获错误以外，`Visual Basic` 也在输入代码时部分地编译该代码。当准备运行和测试应用程序时，只需极短时间即可完成编译。如果编译器发现了错误，则将错误突出显示于代码中。这时可以更正错误并继续编译，而不需从头开始。

由于 Visual Basic 的交互特性，因此可以发现在开发应用程序时，您自己正频繁地运行着您的应用程序。通过这种方式，代码运行的效果可以在开发时进行测试，而不必等到编译完成以后。

集成开发环境的元素

Basic 的工作环境常常是指集成开发环境或 IDE，这是因为它在一个公共环境里集成了许多不同的功能，例如，设计、编辑、编译和调试。在大多传统开发工具中，每个功能都是以一个独立的程序运行，并都有自己的界面。本节，将讨论以下主题：

- 启动（Visual Basic IDEvbconStartingVisualBasicIDE）启动和运行。
- 集成开发环境元素（vbconIntegratedDevelopmentEnvironmentElements）介绍 IDE 的不同组成部分。
- 环境选项（vbconEnvironmentOptions）配置 Visual Basic 以满足个人的需要。

启动 Visual Basic IDE

当运行 Visual Basic 安装程序时，允许将程序项置于已存在的程序组中，或在 Windows 中为 Visual Basic 创建一个新的程序组和程序项。这时可准

备从 Windows 启动 Visual Basic。

要从 Windows 启动 Visual Basic，请按照以下步骤执行：

1. 单击任务条上的“启动”。
2. 选择“程序”，接着选取“Microsoft Visual Basic 6.0”。

- 或 -

在任务条上单击“启动”。

选定“程序”。

使用“Windows 资源管理器”寻找 Visual Basic 可执行文件。

3. 双击 Visual Basic 图标。

也可以创建一个 Visual Basic 快捷键，并双击该快捷键。

当第一次启动 Visual Basic 时，可以见到集成开发环境的界面，如图 2.1 所示。

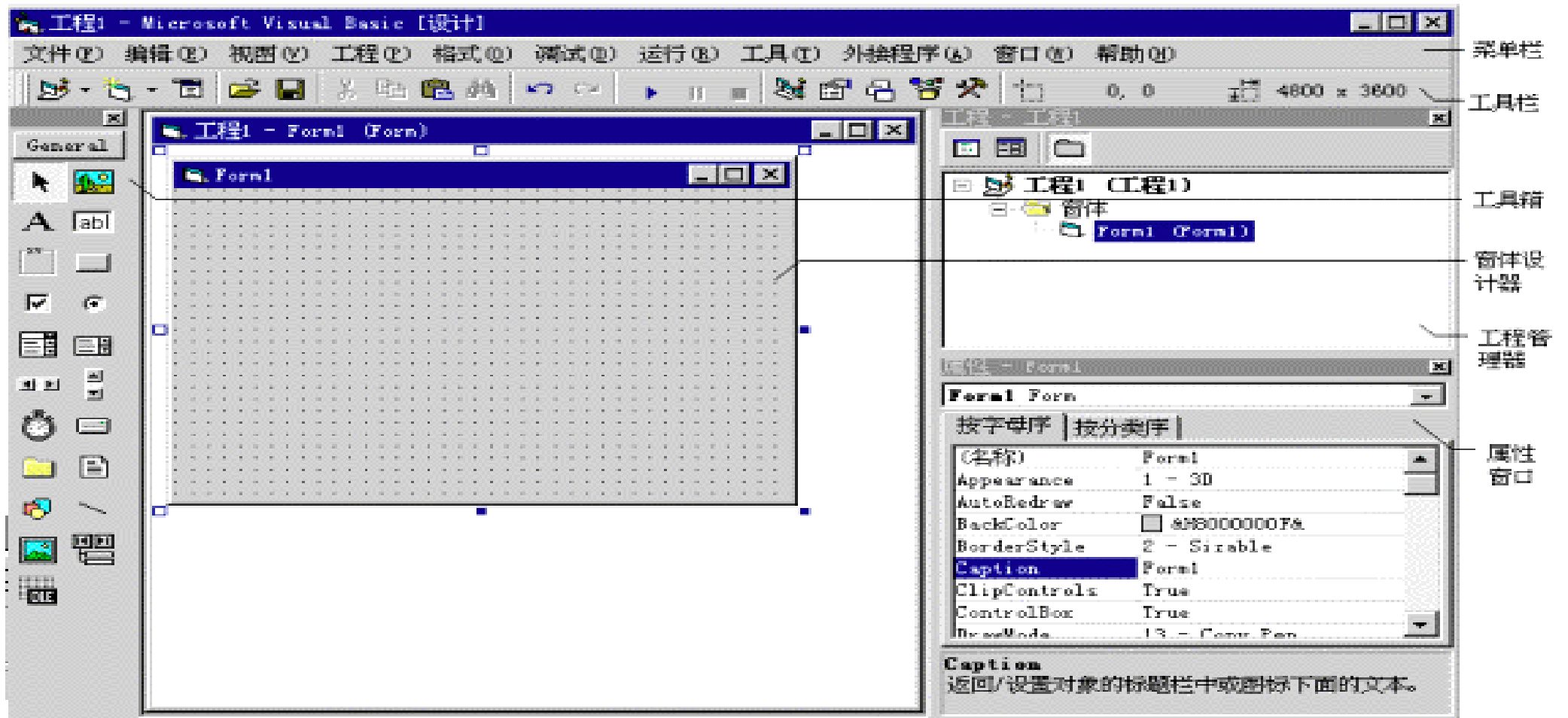


图 2.1 Visual Basic 集成开发环境

集成开发环境的元素

Visual Basic 集成开发环境 (IDE) 由以下元素组成：

菜单条

显示所使用的 Visual Basic 命令。除了提供标准“文件”、“编辑”、“视图”、“窗口”和“帮助”菜单之外，还提供编程专用的功能菜单，例如“工程”、“格式”或“调试”。

上下文菜单

包括经常执行的操作的快捷键。在要使用的对象上单击鼠标按钮即可打开上下文菜单。在上下文菜单中有效的专用快捷键清单取决于单击鼠标键所在环境。例如，在“工具箱”上单击鼠标右键时显示的上下文菜单，可以在上面选择显示“部件”对话框，隐含“工具箱”，停放或挂断“工具箱”，或在“工具箱”中添加自定义选项卡。

工具栏

在编程环境下提供对于常用命令的快速访问。单击工具栏上的按钮，则执行该按钮所代表的操作。按照缺省规定，启动 Visual Basic 之后显示“标准”工具栏。附加的编辑、窗体设计和调试的工具栏可以从“视图”菜单上的“工具栏”命令中移进或移出。

工具栏能紧贴在菜单条之下，或以垂直条状紧贴在左边框上。如果将它从

菜单下面拖开，则它能“悬”在窗口中。

工具箱

提供一组工具，用于设计时在窗体中放置控件。除了缺省的工具箱布局之外，还可以通过从上下文菜单中选定“添加选项卡”并在结果选项卡中添加控件来创建自定义布局。

详细信息 关于特殊控件的详细信息，请参阅第三章“窗体、控件和菜单”和第七章“使用 Visual Basic 标准控件”。关于怎样在“工具箱”添加控件的详细信息，请参阅四章“工程的管理”中“在工程中添加控件”。

工程管理器窗口

列出当前工程中的窗体和模块。工程是指用于创建一个应用程序的文件的集合。

详细信息 关于工程的信息，请参阅第四章“工程的管理”。

属性窗口

列出对选定窗体和控件的属性设置值。属性是指对象的特征，如大小、标题或颜色。

详细信息 关于属性的详细信息，请参阅第三章“窗体、控件和菜单”中“属性、方法和事件概述”

对象浏览器

列出工程中有效的对象，并提供在编码中漫游的快速方法。可以使用“对象浏览器”浏览在 Visual Basic 中的对象和其它应用程序，查看对那些对象有效的方法和属性，并将代码过程粘贴进自己的应用程序。

详细信息 关于使用“对象浏览器”查看过程的详细信息，请参阅第九章“用对象编程”中“查找对象信息”。关于使用外接程序扩展 Visual Basic 编程环境的详细信息，请参阅第四章“工程的管理”中“使用向导和外接程序”。

窗体设计器

作为自定义窗口用来设计应用程序的界面。在窗体中添加控件、图形和图片来创建所希望的外观。应用程序中每一个窗体都有自己的窗体设计器窗口。

详细信息 关于怎样在应用程序添加控件的信息，请参阅本章后面的“第一个 Visual Basic 应用程序”。关于界面设计的详细信息，请参阅第六章“创建用户界面”。

代码编辑器窗口

是输入应用程序代码的编辑器。应用程序的每个窗体或代码模块都有一个

单独的代码编辑器窗口。

详细信息 关于输入代码和使用代码编辑器的详细信息，请参阅第五章“编程基础”。

窗体布局窗口

Form Layout window（图 2.2）允许使用表示屏幕的小图像来布置应用程序中各窗体的位置。



图 2.2 窗体布局窗口

立即、本地和监视窗口

这些附加窗口是为调试应用程序提供的。它们只在 IDE 之中运行应用程序时才有效。

详细信息 关于调试和使用调试窗口的详细信息，请参阅第十三章“调试代码和处理错误”。

注意 使用调用外接程序的程序也可添加 Visual Basic 界面的功能。由微软和第三方开发者提供的外接程序能提供像源代码控件之类的特性，这些特性可支持组开发工程。

环境选项

Visual Basic 具有很大的灵活性，可以通过配置工作环境满足个人风格的最佳需要。可以在单个或多文档界面中间进行选择，并能调节各种集成开发环境 (IDE) 元素的尺寸和位置。所选择的布局将保留在 Visual Basic 的会话期之间。

SDI 或 MDI 界面

Visual Basic IDE 有两种不同的类型：单文档界面 (SDI) 或多文档界面 (MDI)。对 SDI 选项，所有 IDE 窗口可在屏幕上任何地方自由移动；只要 Visual Basic 是当前应用程序，它们将位于其它应用程序之上。对 MDI 选项，所有 IDE 窗口包含在一个大小可调的父窗口内。

要在 SDI 和 MDI 模式间切换，请按照以下步骤执行：

1. 从“工具”菜单中选定“选项”。
显示“选项”对话框。
2. 选定“高级”选项卡。

3. 选择或不选择“SDI 开发环境”复选框。

下次启动 Visual Basic 时，IDE 将以选定模式的模式启动。

- 或 -

用带 /sdi 或 /mdi 的参数的命令行运行 Visual Basic。

停放窗口

集成开发环境中的许多窗口能相互连接，或停放在屏幕边缘。包括：工具箱、窗体布局窗口、工程管理器、属性窗口、调色板、立即窗口、本地窗口和监视窗口。

对 MDI 选项，窗口可停放在父窗口的任意侧，而对于 SDI，窗口只能停放在菜单条下面。一给定窗口的“可连接的”功能，可以通过在“选项”对话框的“可连接的”选项卡上选定合适的复选框来打开或关闭，“选项”对话框可以从“工具”菜单上的“选项”命令选取。

要停放或移动窗口，请按照以下步骤执行：

1. 选定要停放或移动的窗口。
2. 按住鼠标左键拖动窗口到希望到达的位置。
3. 拖动时会显示窗口轮廓。
4. 释放鼠标按钮。

第一个 Visual Basic 应用程序

创建 Visual Basic 应用程序是很简单的事。有多简单？为回答这个问题，试着创建下面的 Hello, Visual Basic 和 Firstapp 应用程序。

你好， Visual Basic

创建 Visual Basic 应用程序有三个主要步骤：

1. 创建应用程序界面
2. 设置属性
3. 编写代码

为了说明这一实现过程，按照以下步骤创建一个简单应用程序，该应用程序由一个文本框和一个命令按钮组成。单击命令按钮，文本框中会出现 "Hello, world!" 消息。

创建应用程序界面

窗体是创建应用程序的基础。通过使用窗体可将窗口和对话框添加到应用程序中。也可把窗体作为项的容器，这些项是应用程序界面中的不可视部分。例如，应用程序中可能有一个作为图形容器的窗体，而这些图形是打算在其它窗体中显示的。

建造 Visual Basic 应用程序的第一步是创建窗体，这些窗体将是应用程序

界面的基础。然后在创建的窗体上绘制构成界面的对象。对于目前的第一个应用程序，你可使用工具箱中的两个控件。

按钮

控件



文本框

命令按钮

用工具箱绘制控件

1. 单击要绘制的控件的工具——此时是“文本框”。
2. 将指针移到窗体上。该指针变成十字线，如图 2.3 所示。

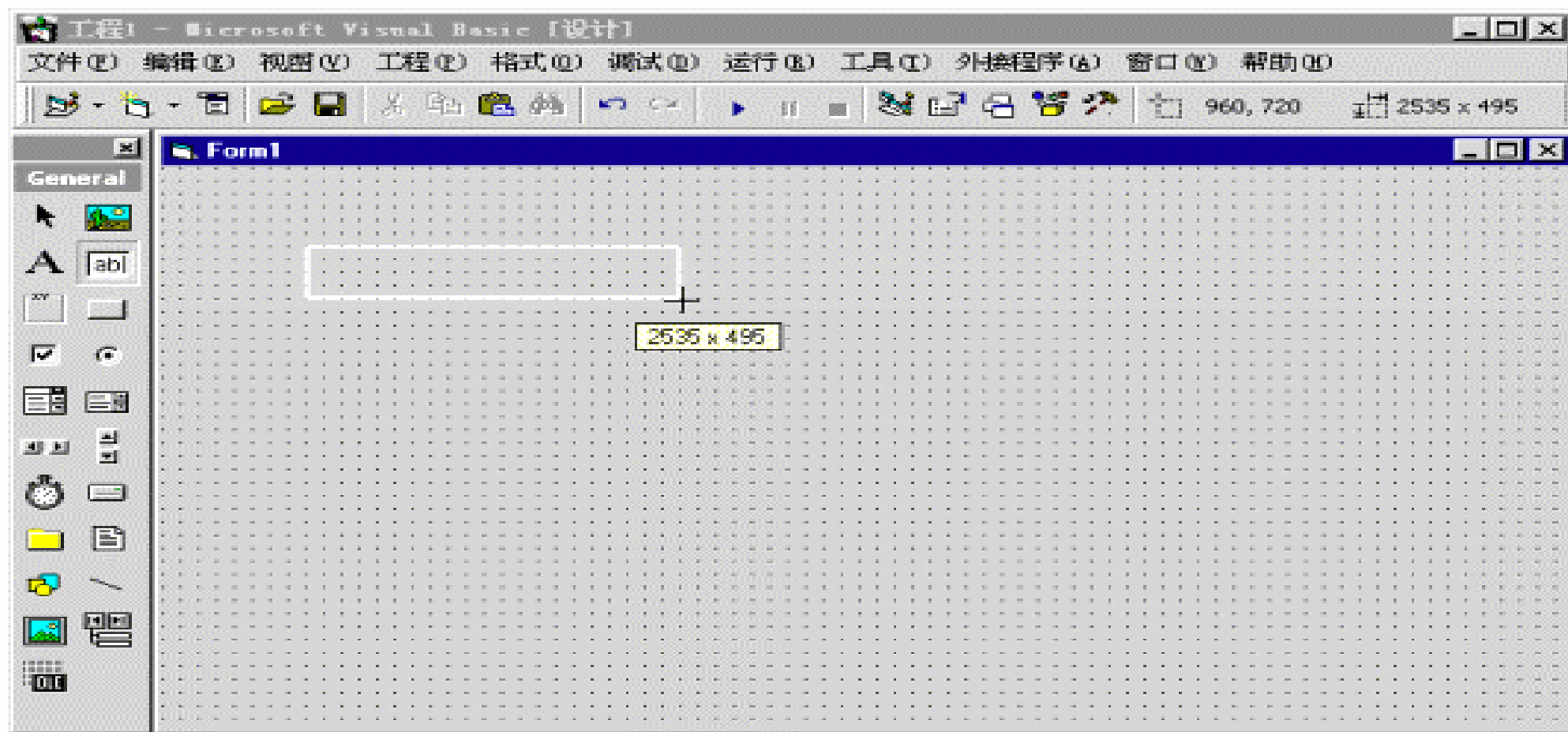


图 2.3 用工具箱绘制文本框

3. 将十字线放在控件的左上角所在处。
4. 拖动十字线画出适合您要的控件大小的方框（拖动的意思是按住鼠标左键用鼠标指针移动对象）。
5. 释放鼠标按钮。

控件出现在窗体上。

在窗体上添加控件的另一个简单方法是双击工具箱中的控件按钮。这样会在窗体中央创建一个尺寸为缺省值的控件；然后再将该控件移到窗体中的其它位置。

调整大小、移动和锁定控件

注意出现在控件四周的小矩形框称作尺寸句柄；下一步可用这些尺寸句柄调节控件尺寸。也可用鼠标、键盘和菜单命令移动控件、锁定和解锁控件位置以及调节控件位置。

调整控件的尺寸，请按照以下步骤执行：

1. 用鼠标单击要调整尺寸的控件。

选定的控件上出现尺寸句柄。

2. 将鼠标指针定位到尺寸柄上，拖动该尺寸柄直到控件达到所希望的大小为止。

角上的尺寸柄可以调整控件水平和垂直方向的大小，而边上的尺寸柄调整控件一个方向的大小。

3. 释放鼠标按钮。

- 或 -

用 SHIFT 键加上和箭头键调整选定控件的尺寸。

要移动控件，请：

- 用鼠标把窗体上的控件拖动到一新位置。

- 或 -

用“属性”窗口改变 Top 和 Left 属性。

选定控件后，可用 CTRL 键加箭头键每次移动控件一个网格单元。如果该网格关闭，控件每次移动一个像素。

要锁定所有控件位置，请：

- 从“格式”菜单，选取“锁定控件”。

- 或 -

在“窗体编辑器”工具栏上单击“锁定控件切换”按钮。

这个操作将把窗体上所有的控件锁定在当前位置，以防止已处于理想位置的控件因不小心而移动。本操作只锁住选定窗体上的全部控件，不影响其它窗体上的控件。这是一个切换命令，因此也可用来解锁控件位置。

要调节锁定控件的位置，请：

- 按住 CTRL 键，再用合适的箭头键可“微调”已获焦点的控件的位置。

- 或 -

也可在“属性”窗口中改变控件的 Top 和 Left 属性。

现在已生成了 "Hello, world!" 应用程序的界面，如图 2.4 所示。

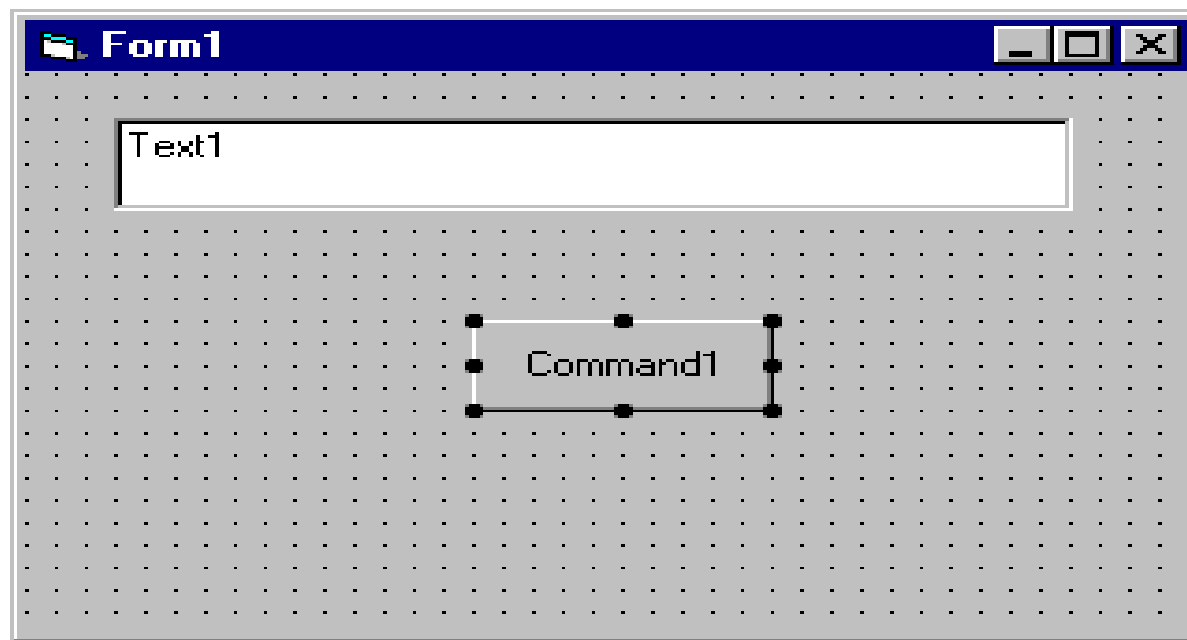


图 2.4 "Hello, world!" 应用程序的界面

设置属性

下一步是给创建的对象设置属性。属性窗口（图 2.5）给出了设置所有的窗体对象属性的简便方法。在“视图”菜单中选择“属性窗口”命令、单击工具栏上的“属性窗口”按钮或使用控件的上下文菜单，都可以打开属性窗口。



图 2.5 属性窗口

属性窗口包含如下的元素：

- 对象框——显示可设置属性的对象的名字。单击对象框右边的的箭头，显示当前窗体的对象列表。
- 排序——从按字母顺序排列的属性列表中进行选取，或从按逻辑（诸如与外观、字体或位置有关的）分类页的层次结构视图中进行选取。

- 属性列表——左列显示所选对象的全部属性，右列可以编辑和查看设置值。

要在“属性窗口”中设置属性，请按照以下步骤执行：

1. 从“视图”菜单中，选取“属性”，或在工具栏中单击“属性”按钮。
属性”窗口显示所选窗体或控件的属性设置值。
2. 从属性列表中，选定属性名。
3. 在右列中输入或选定新的属性设置值。

列举的属性有预定义的设置值清单。单击设置框右边的向下的箭头，可以显示这个清单，或者双击列表项，可以循环显示这清单。

以 "Hello, world!" 为例，现在要改变三种属性的设置值，而其它属性则采用缺省值。

对象	属性	设置值
窗体	Caption	Hello, world!
文本框	Text	空
命令按钮	Caption	OK

设置 Icon 属性

在 Visual Basic 中，所有窗体都有一个普通的缺省图标，它们在窗体最小化时出现。也许您想换一个能说明窗体或应用程序的具体用途的图标。设置窗体的 Icon 属性，可给窗体指定新的图标。32 x 32 像素的图标 Microsoft Windows 的 16-bit 版本的标准也可应用在 Winndows 95 和 Windows NT

中，另一种 16 x 16 像素的图标是用在 Windows 95 中的。

编写代码

代码编辑器窗口是编写应用程序的 Visual Basic 代码的地方。代码由语句、常数和声明部分组成。使用代码编辑器窗口，可以快速查看和编辑应用程序代码的任何部分。

要打开代码窗口，请：

- 双击要编写代码的窗体或控件。

- 或 -

从“工程管理器”窗口，选定窗体或模块的名称，然后选取“查看代码”按钮。

图 2.6 显示了在双击命令按钮控件后出现的代码编辑器窗口以及该命令的事件。

在同一个代码窗口中可以显示全部过程，也可只显示一个过程。

要同一代码窗口中显示全部过程，请按照以下步骤执行：

1. 在“工具”菜单下，选定“选项”对话框。
2. 在“选项”对话框的“编辑器”选项卡中，选取“缺省为全模式查看”左边的复选框。在“过程分隔符”左边的复选框，可在各过程间添加或去掉分隔线。

- 或 -

在代码编辑器窗口的左下角单击“全模块查看”按钮。

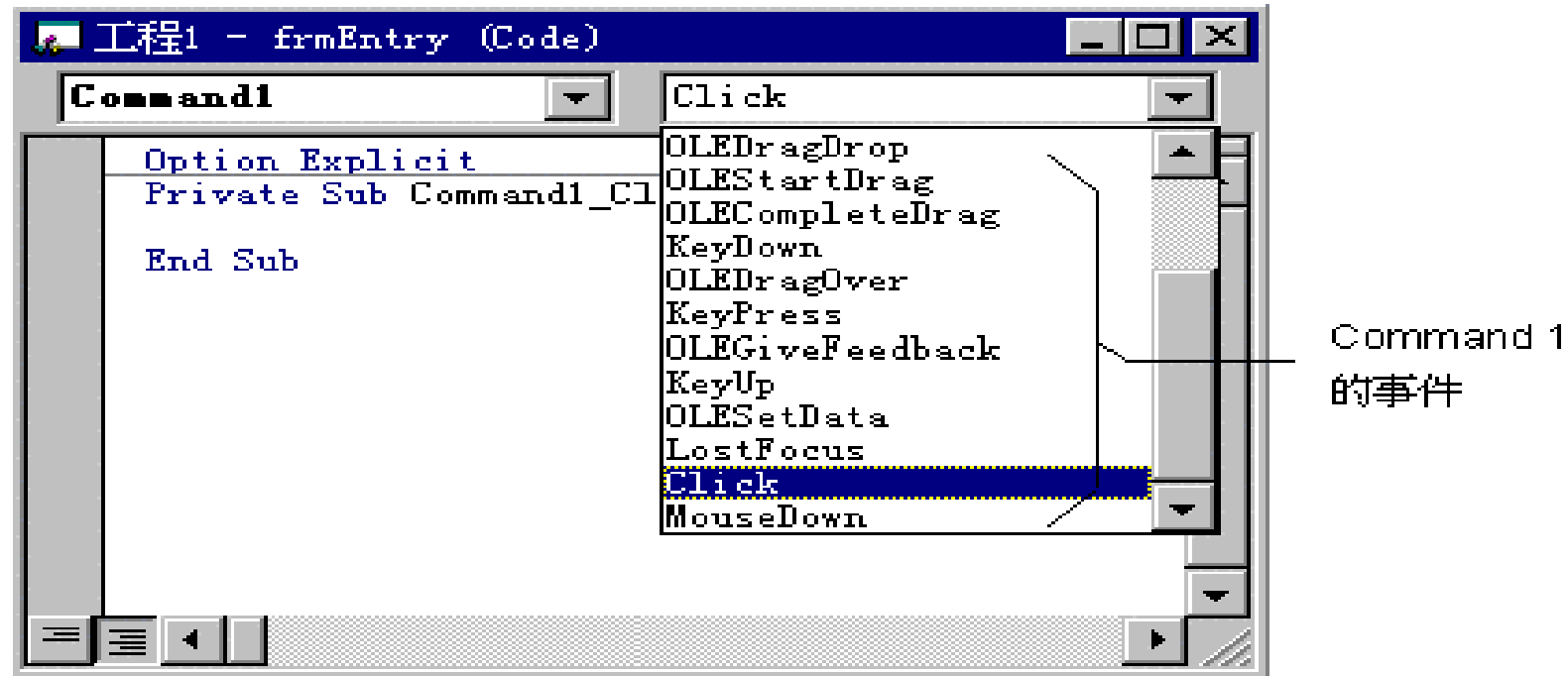


图 2.6 代码编辑器窗口

要代码窗口每次只显示一个过程，请按照以下步骤执行：

1. 在“工具”菜单下选定“选项”对话框。
2. 在“选项”对话框的“编辑器”选项卡中，清除“缺省为全模式查看”左边的复选框。

- 或 -

在代码编辑器窗口的左下角单击“过程查看”按钮。

代码窗口包含如下的元素：

- 对象列表框——显示所选对象的名称。单击列表框右边的箭头，显示和

该窗体关的所有对象的清单。

- 过程列表框——列出对象的过程或事件。该框显示选定过程的名称。在目前情况下是 Click 事件。选取该框右边的箭头可以显示这个对象的全部事件。

创建事件过程

Visual Basic 应用程序的代码被分为称为过程的小的代码块。事件过程，正如此处正要创建的一样，包含了事件发生（例如单击按钮）时要执行的代码。控件的事件过程由控件的实际名称（Name 属性中所指定的）、下划线（_）和事件名组合而成。例如，在单击一个名为 Command1 的命令按钮时调用的 Name 事件过程，可称为 Command1_Click 事件过程。

- 要创建事件过程，请按照以下步骤执行：
 1. 在“对象”列表框中，选定活动窗体中的一个对象名（活动窗体是指当前有焦点的窗体）。
在这一例子中，选定命令按钮 Command1。
 2. 在“过程”列表框中，选择指定对象的事件名。
此时 Click 过程已经被选定，因为它是命令按钮的缺省过程。注意，这时事件过程的模板已经显示在“代码”窗口中。
 3. 在 Sub 和 End Sub 语句之间输入下面的代码：
`Text1.Text = "Hello, world!"`

这个事件过程应该是：

```
Private Sub Command1_Click ()  
    Text1.Text = "Hello, world!"  
End Sub
```

值得注意的是，此处代码只简单改变了控件 Text1 的 Text 属性，读入 "Hello, world!"。这个例子的语法采用 object.property 的格式，其中 Text1 是对象，Text 是属性。响应应用程序运行中发生的事件时，可以用这种语法来改变任何窗体或控件的属性设置值。

详细信息 关于创建其它类型过程的详细信息，请参阅第五章“编程基础”中的“过程概述”一节。

运行应用程序

为了运行应用程序，可以从“运行”菜单中选择“启动”，或者单击工具栏中的“启动”按钮，或按 F5 键。单击刚才在窗体上创建的命令按钮，文本框中就会显示 "Hello, world!"。

Firstapp 示例应用程序

由于 Visual Basic 提供了远比本例应用程序丰富得多的工具，因此马上可以使用许多其的特性来管理和配制应用程序。复习示例应用程序是学习 Visual Basic 的极好方法。下述示例展示了用 Visual Basic 编制实用的应用

程序是多么的轻而易举。

应用程序 Firstapp 演示了如何使用 data 控件和 grid 控件显示取自数据库的信息表。Visual Basic 使得从应用程序内部到数据库的访问非常地容易。数据控件提供了在数据库的记录集中漫游的功能，以及使得在 grid 控件中显示的记录与数据集中的位置同步的能力。

本例包含一个数据控件、一个 MSFlexGrid 控件、一个列表框控件和两个命令按钮。网格显示了从 Northwind 数据库检索到的产品的信息表。当使用数据控件上的定位按钮选定网格的一项时，被选中的产品名便显示在数据控件上。还可以双击网格中当前选择位置，在列表框控件的“购书单”中添加当前选择的书名。

用 AddItem 方法可以添加列表框的信息。方法是作用于特定对象（本例中为 ListBox 对象）的 Visual Basic 函数。指定方法的语法 (object.method) 与设置属性的语法 (object.property) 类似。AddItem 方法可以在程序运行时动态地添加列表框的内容。相反，Clear 方法用于清除列表框的全部信息。

详细信息 关于方法的详细内容，请参阅第三章“窗体、控件和菜单”中的“属性、方法和事件概述”一节。

创建工程

创建工程首先要从“文件”菜单中选择“新建工程”，然后从“新建工程”对话框中选定“标准 EXE”（首次启动 Visual Basic 时将会显示“新建工程”对话框）。Visual Basic 创建一个新的工程并显示一个新的窗体。用一

个数据控件、一个 MSFlexGrid 控件、一个列表框控件以及两个命令按钮画出了应用程序的界面。MSFlexGrid 控件不在缺省的工具箱内，因此必须添加：

要添加控件到工具箱，请按照以下步骤执行：

1. 选定工具箱的上下文菜单中的“部件”（可以在工具箱窗口内部单击右键显示上下文菜单。）。
“部件”对话框被显示。
2. 找到“控件”列表框中的“MSFlexGrid (Microsoft Flex Grid 6.0)”项，然后选中其左边的复选框。
3. 单击“确定”按钮。

在工具箱中有了 MSFlexGrid 控件的图标。

使用工具箱在窗体上画出一个数据控件、一个 MSFlexGrid 控件、一个列表框控件和两个命令按钮。如果您不记得如何绘制，请参阅本章前面的 "Creating the Interface"。

设置属性

根据下表在属性窗口中设置对象的属性，其余的属性均使用缺省值。

对象	属性	设置
Form	Caption	Products
Data1	DatabaseName	path \biblio.mdb

续表

	RecordSource	All Titles
DataGrid1	DataSource	Data1
Command1	Caption	Clear
Command2	Caption	Exit

数据控件的 `DatabaseName` 属性必须包含数据库的实际路径。按照缺省规定，`Biblio.mdb` 数据库的按钮与 `Visual Basic` 安装在同一个目录下。在属性窗口中选择 `DatabaseName` 属性时，可以单击该属性右侧的按钮显示标准的“打开文件”对话框浏览文件。设置完 `DatabaseName` 属性后，属性窗口中的 `RecordSource` 属性将包含选中的数据库中的表或记录集的清单。将 `DataGrid` 控件的 `DataSource` 属性设置为 `Data1`，将自动地把网格和数据控制链接起来。

编写事件代码

程序所有的代码包含在 `Command1_Click`、`Command2_Click`、`Data1_Reposition` 和 `DataGrid1_DblClick` 事件过程中。双击窗体或控件显示代码窗口，然后输入各事件过程的代码。

将以下代码添加到 `Command1_Click` 事件过程中，用于单击按钮时清除列表框：

```
Private Sub Command1_Click ()  
    List1.Clear                '清除列表框。
```

End Sub

以上语句调用了列表框 List1 的 Clear 方法，Clear 方法删除该列表框的内容。

将以下代码添加到 Command2_Click 事件过程中，用于从内存中卸载窗体并终止应用程序：

```
Private Sub Command2_Click ()
```

```
    Unload Form1
```

```
End Sub ' 终止应用程序
```

End Sub

以上过程的第一条语句调用窗体的 Unload 事件。如果需要在终止程序时执行一个功能（例如保存文件），可以把代码放到窗体的 Unload 事件过程中。第二条语句调用 End 函数终止程序。

将以下代码添加到 Data1_Reposition 事件过程中，用于每当选择一个记录时更新标题：

```
Private Sub Data1_Reposition ()
```

```
    Data1.Caption = Data1.Recordset("Title")
```

```
End Sub
```

以上语句将右边的值（数据控件的记录集的 Title 字段）赋给左边的属性（数据控件对象的 Caption 属性）。

将以下代码添加到 DataGrid1_DblClick 事件过程中，用于当双击一个选定的行时向列表框添加信息。

```
Private Sub DataGrid1_DblClick ()
```

```
List1.AddItem Data1.Recordset("Title")  
End Sub
```

以上语句调用列表框 `List1` 的 `AddItem` 方法。添加到列表框中的文本被包含在该方法的参数中（此时为数据控件的记录集的 `title` 字段）。将一个值传递给参数类似于给属性赋值，与赋值语句不同的是不需要用等号。

保存工程

从“文件”菜单中选取“保存工程”命令来结束本次创建应用程序的工作。`Visual Basic` 将分别提示保存窗体和保存工程。可以将工程命名为“书店销售清单” (`"Bookstore Shopping List."`)。Windows 95 和 Windows NT 都允许使用长达 255 个字符的文件名而且可以包含空格。旧版的 `Microsoft Windows` 只允许八个字符的文件名外加三个字符的扩展名。

增强应用程序

这样第一个 `Visual Basic` 应用程序就完成了：这是一个功能简单却很有用的程序。可以以此为基础在自己的应用程序中添加新的功能，用自己的数据替换 `Biblio.mdb`。当然，要使本程序真正实用，还要添加基本的保存或打印列表框内容的功能、添加更多的信息,如价格、是否有货，甚至可以收集信用卡的信息和通过 `Internet` 传送订单。通过继续阅读《程序员指南》的其余部分将找到所有有关这些功能的例子和其它更多的知识。

第三章 窗体、控件和菜单

用 Visual Basic 创建一个应用程序，第一步是创建界面，它是用户与应用程序进行交互操作的可视部分。窗体和控件是创建界面的基本构造模块，也是创建应用程序所使用的对象。

窗体是一种对象，由属性定义其外观，由方法定义其行为，由事件定义其与用户的交互。通过设置窗体属性并编写响应事件的 Visual Basic 代码，就能定义出满足应用程序需要的对象。

控件是包括在窗体对象内的对象。每种类型的控件都有自己的一套属性、方法和事件，以适用于特定的目的。一些控件最适合在应用程序中输入或显示文本。另一些控件能够访问其它的应用程序和处理数据，就像这些远程应用程序是用户自己的代码一样。

本章介绍一些基本概念，使用户了解如何使用窗体、控件及与之相关的属性、方法和事件。本章将讨论许多标准控件及窗体专用项，例如菜单和对话框。

目录

- 属性、方法和事件概述
- 设计窗体
- 单击按钮执行操作

- 显示和输入文本的控件
- 为用户提供选择的控件
- 显示图片和图形的控件
- 附加控件
- 焦点概述
- 设置 Tab 键顺序
- 菜单基础
- 用对话框提示用户

应用程序示例 Controls.vbp

本章中的代码示例取自应用程序示例 `Controls.vbp`，该示例应用程序列在 `Samples!Alink("vbsamples")` 目录中。

属性、方法和事件概述

Visual Basic 的窗体和控件是具有自己的属性、方法和事件的对象。可以把属性看作一个对象的性质，把方法看作对象的动作，把事件看作对象的响应。

日常生活中的对象，如小孩玩的气球同样具有属性、方法和事件。气球的属性包括可以看到的一些性质，如它的直径和颜色。其它一些属性描述气

球的状态（充气的或未充气的）或不可见的性质，如它的寿命。通过定义，所有气球都具有这些属性；这些属性也会因气球的不同而不同。

气球还具有本身所固有的方法和动作。如：充气方法（用氦气充满气球的动作），放气方法（排出气球中的气体）和上升方法（放手让气球飞走）。所有的气球都具备这些能力。

气球还有预定义的对某些外部事件的响应。例如，气球对刺破它的事件响应是放气，对放手事件的响应是升空。

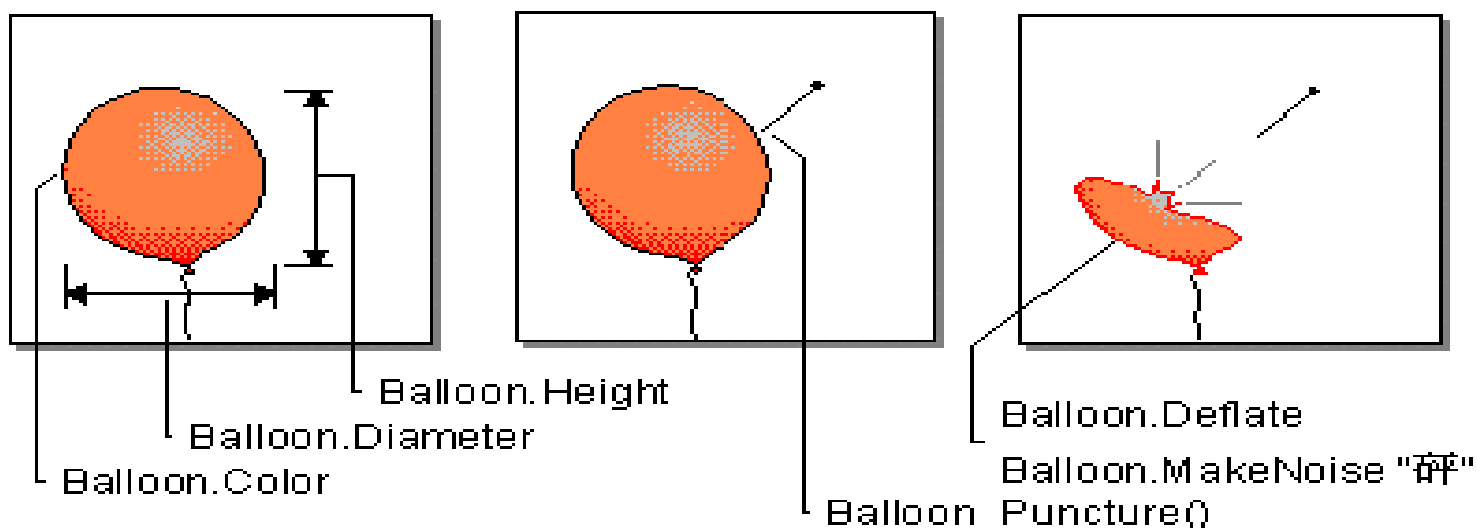


图 3.1 对象具有属性，对事件的响应和执行方法

如果能对气球编程的话，Visual Basic 代码将如下所示。其作用是设置气球的属性：

```
Balloon.Color = Red
```

```
Balloon.Diameter = 10
```

```
Balloon.Inflated = True
```

注意代码的语法，首先是对象 (Balloon)，接着是属性 (Color)，然后是赋值 (Red)。重复这条语句，但是，换一个不同的值，就可以改变气球的颜色。属性也可以在程序设计阶段在“属性”窗口中设置。

调用气球的方法是这样的：

```
Balloon.Inflate
```

```
Balloon.Deflate
```

```
Balloon.Rise 5
```

它的语法与属性的语法相似，对象（一个名词）后面紧跟着方法（一个动词）。在第三个例子中有一个附加项，称为一个参数，表示气球上升的高度。一些方法有一个或多个参数，它们对执行的动作做进一步的描述。

气球对一个事件的响应如下所示：

```
Sub Balloon_Puncture()
```

```
    Balloon.Deflate
```

```
    Balloon.MakeNoise "Bang"
```

```
    Balloon.Inflated = False
```

```
    Balloon.Diameter = 1
```

```
End Sub
```

本例中代码描述了当刺穿事件发生时气球的行为：调用 Deflate 方法，然后以 "Bang"（发出的声响）为参数调用 MakeNoise 方法。因为气球已不

再是充气状态，所以把 `Inflated` 属性置为 `False`，并给 `Diameter` 属性设置一个新值。

实际上对气球不能编程，但是，可以对 `Visual Basic` 的窗体或控件进行编程。作为程序员，由您作主。您决定应更改哪些属性，调用哪些方法，对哪些事件作出响应，从而得到希望的外观和行为。

设计窗体

窗体对象是 `Visual Basic` 应用程序的基本构造模块，是运行应用程序时与用户交互操作的实际窗口。窗体有自己的属性、事件和方法用于控制其外观和行为。

设计窗体的第一步是设置它的属性。这可以在设计时在“属性”窗口中完成，或者运行时由代码来实现。

注意 设计时，即在 `Visual Basic` 环境中创建应用程序的任何时刻，此时你可以操作窗体和控件，设置它们的属性，对它们的事件编程。运行时是指实际运行应用程序并与应用程序进行交互的时间。

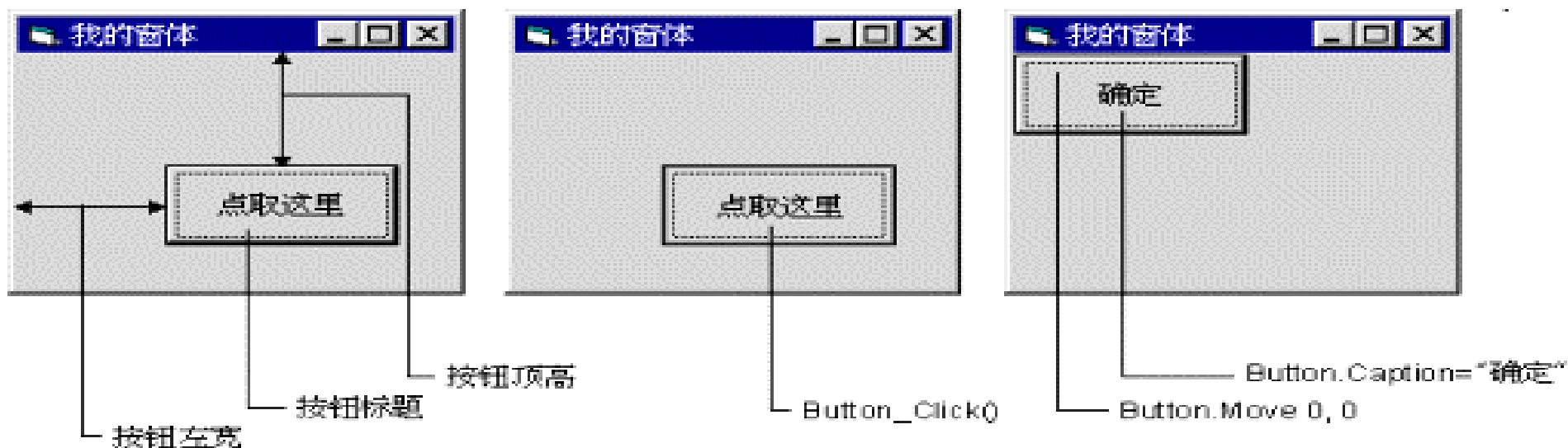


图 3.2 窗体和控件有自己的属性、事件和方法

设置窗体属性

窗体的许多属性会影响窗体的外观。Caption 属性决定窗体标题栏中显示的文本；图标属性设置在窗体最小化时显示的图标。MaxButton 和 MinButton 属性决定窗体是否能最大化或最小化。通过改变 BorderStyle 属性，可以控制窗体如何调整大小。

Height 和 Width 属性决定的窗体初始大小；Left 和 Top 属性根据屏幕的左上角确定窗体的位置。WindowState 属性可以把窗体设成在启动时最大化、最小化或正常大小。

Name 属性设置窗体的名称，在代码中用这个名称引用该窗体。首次在工

程中添加窗体时，该窗体的名称被缺省为 `Form1`；添加第二个窗体，其名称被缺省为 `Form2`，以此类推。最好给 `Name` 属性设置一个有实际意义的名称，如给一个条目窗体命名为 `"frmEntry"`。

熟悉这些窗体属性，最好的办法是实践。在“属性”窗口（图 3.3）中更改窗体的一些属性，然后运行该应用程序并观察修改的效果。如果想详细了解每个属性，可以选择该属性并按 `F1` 键查看联机帮助。

窗体事件和方法

窗体作为对象，能够执行方法并对事件作出响应。

无论是因为用户交互，还是通过代码调整窗体的大小，都会触发一个 `Resize` 事件。当窗体尺寸变化时，允许您在窗体上进行移动控件或调整控件大小等操作。

每当一个窗体变成活动窗体时，就会产生一个 `Activate` 事件；当另一个窗体或应用程序被激活时，就会产生 `Deactivate` 事件。这些事件对初始化或结束窗体行为十分方便。例如，在 `Activate` 事件中，可以编写代码突出显示一个特定文本框中的文本；在 `Deactivate` 事件中，可以把更改保存到一个文件或数据库中。

要使一个窗体可见，可调用 `Show` 方法：

`Form2.Show`

调用 `Show` 方法与设置窗体 `Visible` 属性为 `True` 具有相同的效果。

窗体的许多方法都调用文本或图形。`Print`、`Line`、`Circle` 和 `Refresh` 方法可

用于直接在窗体表面上写和画。这些方法和更多的方法将在第十二章“使用文本和图形”中详细讨论。

详细信息 关于窗体的详细信息，请参阅第六章“创建用户界面”中的“再论窗体”。



图 3.3 “属性”窗口

单击按钮执行操作

使用户与应用程序交互的最简便的方法是给用户提供一个按钮。可以用 Visual Basic 提供的 `CommandButton` 控件，或利用包含图形的 `Image` 控件，如一个图标，来创建自己的“按钮”。

使用 `CommandButton`

大多数 Visual Basic 应用程序都有 `CommandButton`，使用户通过简单的敲击按钮来执行操作。当用户选中按钮时，不仅会执行相应操作，还会使该按钮看上去象被按下并释放一样。无论何时，只要用户单击按钮，就会调用 `Click` 事件过程。将代码写入 `Click` 事件过程，执行想要执行的动作。在程序执行时，有多种方法可以选中 `CommandButton`：

- 用鼠标单击按钮。
- 按 `TAB` 键把焦点移到相应按钮上，然后按 `SPACE` 键或 `ENTER` 键选中该按钮。参阅本章后面的“焦点概述”。
- 按 `CommandButton` 的快捷键（`ALT` + 下划线的字符）。
- 在代码中设 `CommandButton` 的 `Value` 属性为 `True`。
- 在代码中调用 `CommandButton` 的 `Click` 事件。

```
cmdClose_Click
```

- 如果该 `CommandButton` 是窗体的缺省命令按钮，即使将焦点移到其它不是 `CommandButton` 的控件上，按下回车键也会选中该按钮。在设计时，通过设置某按钮的 `Default` 属性为 `True`，指定它为窗体的缺省 `CommandButton`。
 - 如果该 `CommandButton` 是窗体的缺省的取消按钮，即使把焦点移到其它控件上，也能通过按 `ESC` 键选中该按钮。在设计时，通过设置某按钮的 `Cancel` 属性为 `True`，指定它为窗体的缺省的取消按钮。
- 所有这些操作都使 `Visual Basic` 调用 `Click` 事件过程。

“测试按钮”应用程序

利用 `Caption` 属性可以在按钮上显示文字，告诉用户该按钮的功能。在图 3.4 中，在控件示例应用程序的测试按钮范例中包含一个 `Caption` 属性为 "Change Signal" 的 `CommandButton`（此例的工作为 `Controls.vbp` 示例应用程序中的 `Buttons.frm`）。

注意，`S` 是这个按钮的快捷键，用一条下划线表示。在 `Caption` 属性的正文内插入一个 `(&)` 符号，将使得此符号后面的字符成为该按钮的快捷键（例如，`Change &Signal`）。

用户单击 `CommandButton`，将执行该 `CommandButton` 的 `Click` 事件过程中的代码。在这个范例中，每次单击按钮，将变换一个不同的交通灯图标。

详细信息 关于 `CommandButton` 的属性的详细信息，请参阅第七章“使用 `Visual Basic` 的标准控件”。



图 3.4 带标题的 `CommandButton`

用于显示和输入文本的控件

`Label` 和 `TextBox` 控件是用于显示和输入文本的。让应用程序在窗体中显示文本时使用 `Label`，允许用户输入文本时用 `TextBox`。`Labels` 中的文本为只读文本，而 `TextBox` 中的文本为可编辑文本。

要提供这种功能	使用以下控件
可由用户编辑的文本，例如一个顺序条目 字段或一个密码框	TextBox
只能显示的文本，例如在窗体中标识一个 字段或给用户显示指令	Label

Labels 和 TextBoxes 将在以下部分讨论：

- 用 Label 显示文本（vbconUsingLabels）使用 Label 控件的基本知识。
- 使用 TextBox（vbconWorkingWithTextBox）使用 TextBox 的基本知识。

用 Label 显示文本

Label 控件显示的文本用户不能直接修改。如 TextBox 和 ScrollBar 这些没有自己的 Caption 属性的控件，可以用 Label 标识。在 Label 中实际显示的文本是由 Caption 属性控制的，该属性可以在设计时在“属性”窗口中设置或在运行时用代码赋值。

在缺省情况下，标题是 Label 控件中唯一的可见部分。但是，如果把 BorderStyle 属性设成 1（可以在设计时进行），那么 Label 就有了一个边框，看起来像一个 TextBox。还可以通过设置 Label 的 BackColor、BackStyle、ForeColor 和 Font 属性来，改变 Label 的外观。

改变 Label 尺寸以适应其内容

设计时可在“属性”窗口中指定单行 Label 的标题。但是如果要输入一个较长的或在运行时可能变化的标题，该怎么办呢？Label 提供了两种属性：AutoSize 和 WordWrap，用于帮助您改变控件尺寸以适合较长或较短的标题。

AutoSize 属性决定控件是否自动改变尺寸以适应其内容。如该属性设为 True，Label 就会根据其内容进行水平方向变化，如图 3.5 所示。

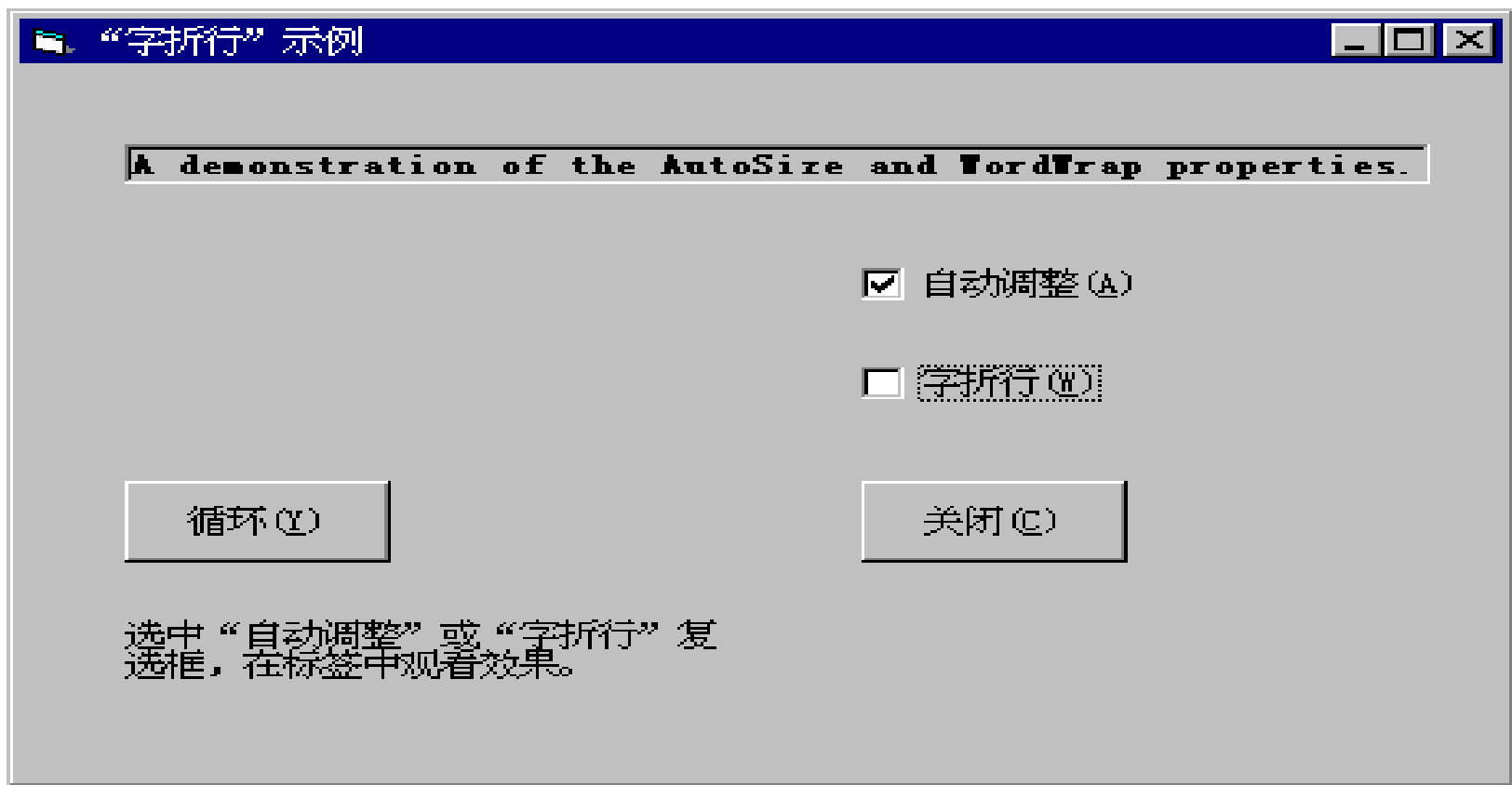


图 3.5 AutoSize 示例

WordWrap 属性使 Label 根据其内容进行垂直方向变化，而保持其宽度不变，如图 3.6 所示。本例的工作版，请参阅应用程序示例 Controls.vbp 中的 WordWrap.frm。

注意 如果在 Controls.vbp 中运行 AotoSize 示例，就会发现为了同时使用 WordWrap，必须两个复选框都选中。这是因为，要想使 Label 的

WordWrap 属性起作用，就必须设置 AutoSize 为 True。只有在一个单字的宽度超过控件的当前宽度时，Label 的宽度才会增加。

详细信息 关于 Label 控件属性的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

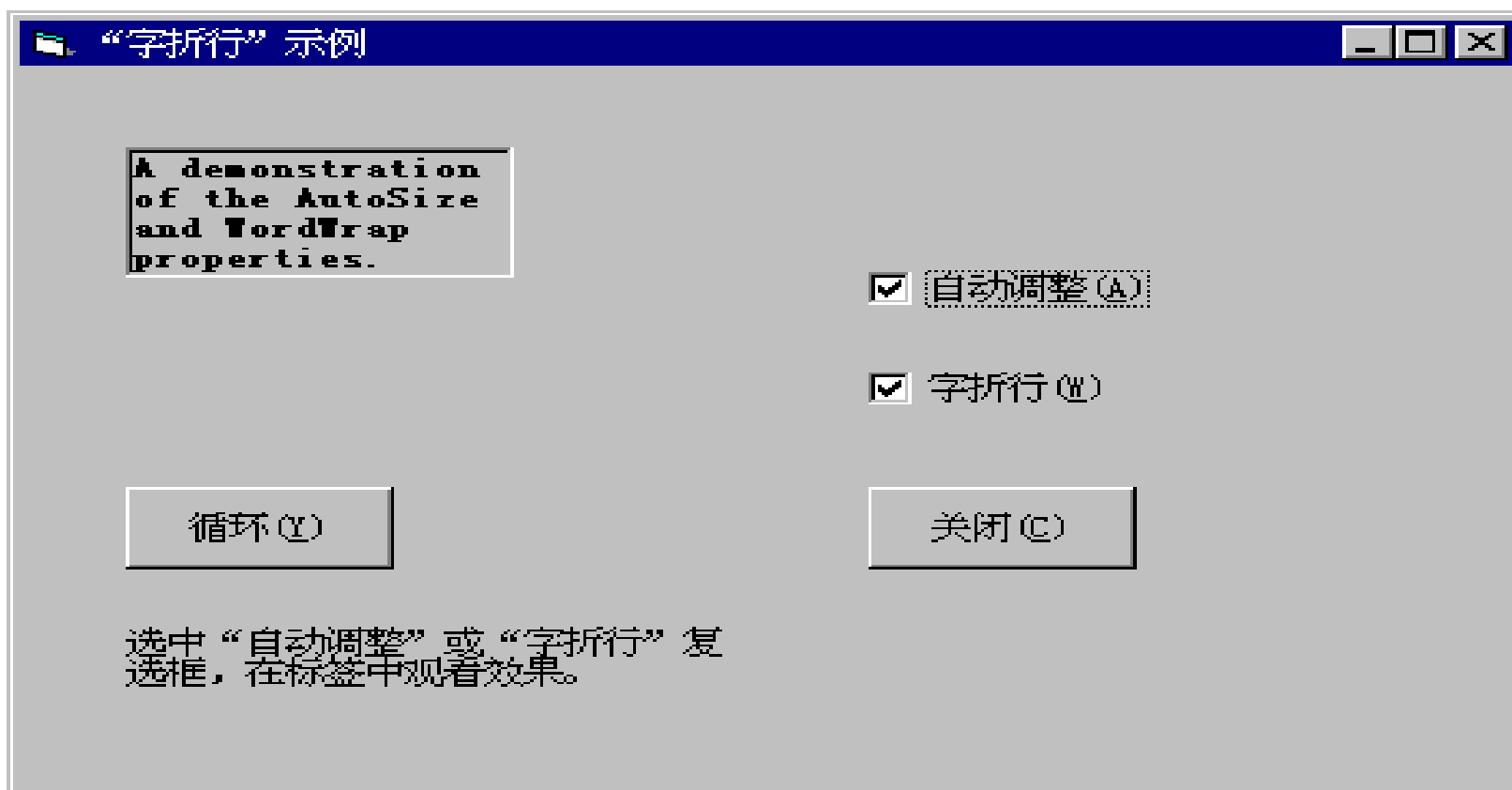


图 3.6 WordWrap 示例

使用 Text Boxes

TextBox 是一种通用控件，可以由用户输入文本或显示文本。除非把 TextBox 的 Locked 属性设为 True，否则不能用 TextBox 显示不希望用户更改的文本。

TextBox 中显示的实际文本是受 Text 属性控制的。Text 属性可以用三种方式设置：设计时在“属性”窗口进行、运行时通过代码置或在运行时由用户输入。通过读 Text 属性能在运行时检索 TextBox 的当前内容。

多行 Text Box 和 Word Wrap

TextBox 在缺省情况下只显示单行文本，且不显示 ScrollBar。如果文本长度超过可用空间，则只能显示部分文本。通过设置 MultiLine 和 ScrollBars 两种属性（只能在设计程序时设置），可以改变 TextBox 的外观和行为。

注意 不要把 ScrollBars 属性与 ScrollBar 控件混淆，ScrollBar 控件并不属于 TextBox，它具有自己的属性集。

把 MultiLine 属性设为 True，可以使 TextBox 在运行时接受或显示多行文本。只要没有水平方向 ScrollBar，多行 TextBox 中的文本会自动按字换行。ScrollBars 属性的缺省值被设置为 0 (None)。自动按字换行省去用户在行尾插入换行符的麻烦。当一行文本已超过所能显示的长度时，TextBox 自动将文本折回到下一行显示。

在设计时，不能在“属性”窗口输入换行符。在过程中，可以通过插入一

个回车加上换行符（ANSI 字符 13 和 0）来产生一个行断点。也可以用常量 `vbCrLf` 插入一个回车与换行符组合。例如，下面的事件过程是在加载窗体时，把两行文本放入一个多行 `TextBox (Text1)` 中的示例。

```
Sub Form_Load ()  
    Text1.Text = "Here are two lines" _  
    & vbCrLf & "in a text box"  
End Sub
```

使用 `TextBox` 中的文本

利用 `TextBox` 的 `SelStart`、`SelLength` 和 `SelText` 属性，可以控制 `TextBox` 的插入点和选择行为。这些属性仅能在运行时使用。

当一个 `TextBox` 首次得到焦点时，`TextBox` 缺省的插入点和光标位置在文本的最左边。用户可以用键盘和鼠标移动它们。当 `TextBox` 失去焦点而后再得到时，插入点位置与用户最后设置的位置一样。

在有些情况下，它可能与用户设置不一致。如：在字处理应用程序中，用户会希望新字符出现在已有文本后面；在数据项应用程序中，用户会希望他的输入替换原有条目。使用 `SelStart` 和 `SelLength` 属性，用户可以根据需要改变 `TextBox` 的行为。

`SelStart` 属性是一个数字，指示文本串内的插入点，其中 0 表示最左边的位置。如果 `SelStart` 属性值大于或等于文本中的字符数，那么插入点将被放在最后一个字符之后，如图 3.7 所示。该例子的工作版，请参阅应用程

序示例 Controls.vbp 中的 Text.frm。

SelLength 属性是一个设置插入点宽度的数值。把 **SelLength** 设为大于 0 的值，会选中并突出显示从当前插入点开始的 **SelLength** 个字符。图 3.8 显示选择的性能。

如果有一段文本被选中，此时用户键入的文字将替换被选中的文本。有些情况下，也可以用粘贴命令用新文本替换原有的文本。**SelText** 属性是一串文本，可以在运行时给它赋值以替换当前选中的文本。如果没有选中的文本，**SelText** 将在当前插入点插入文本。

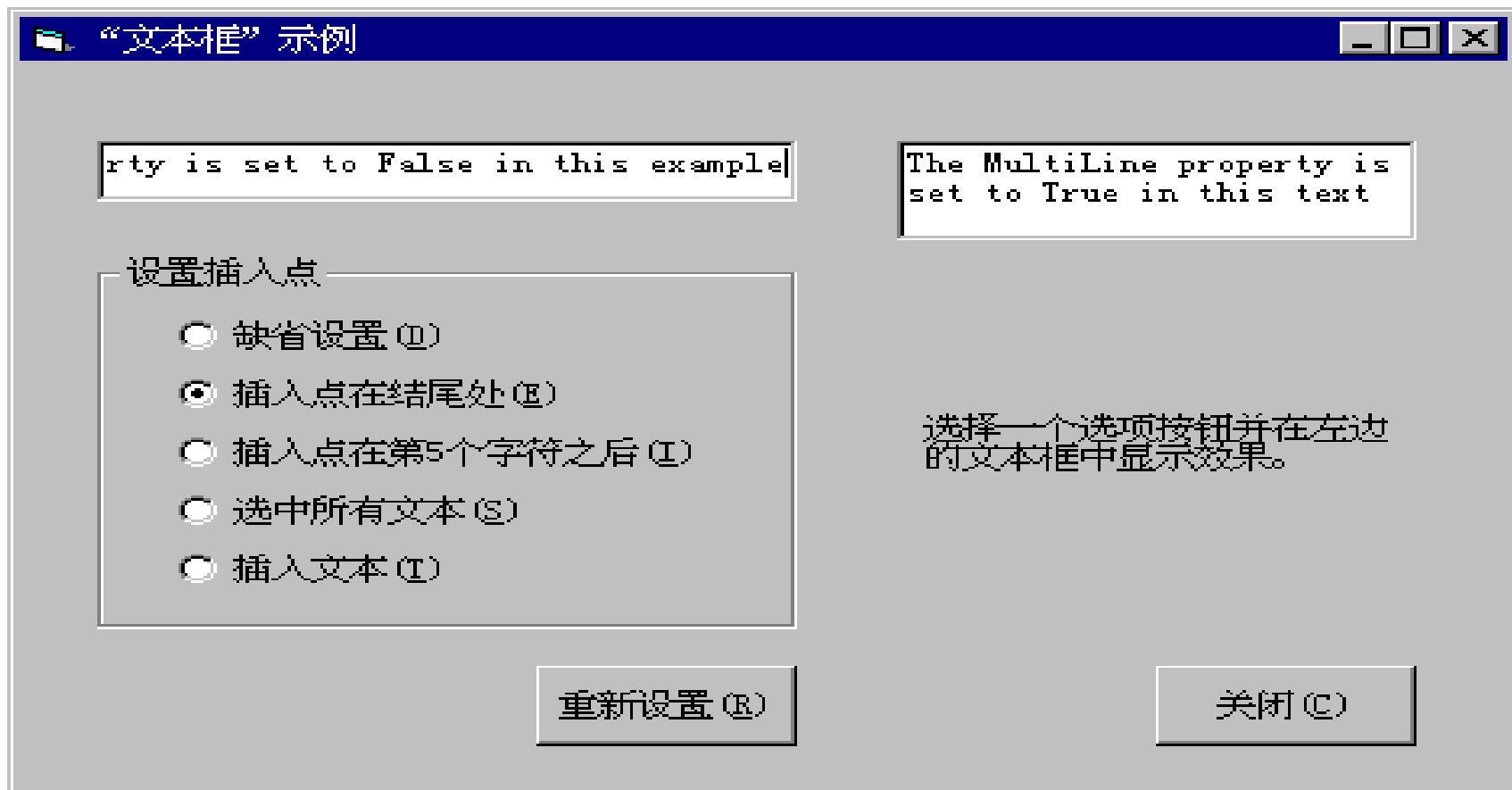


图 3.7 插入点示例

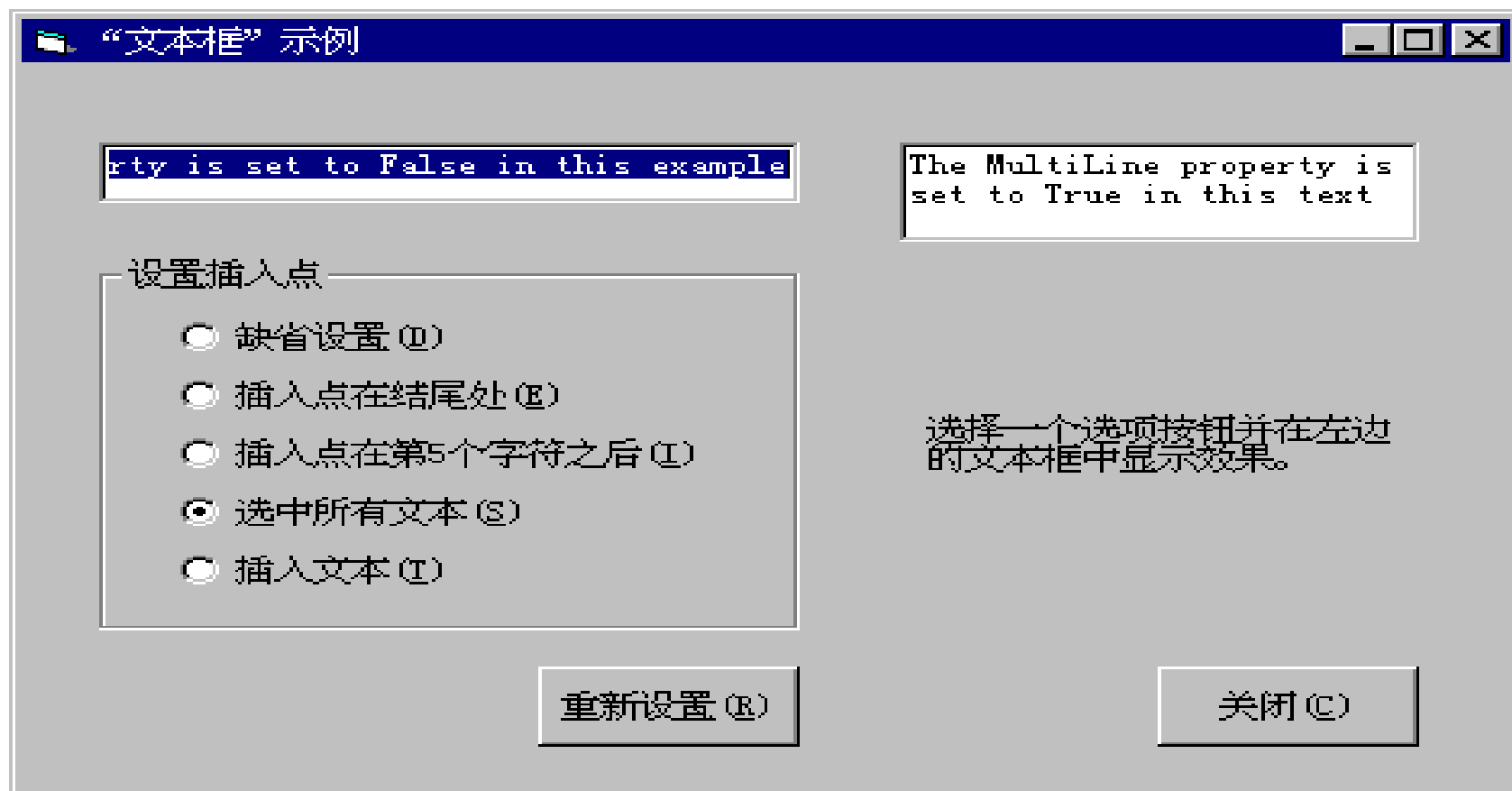


图 3.8 选择示例

详细信息 关于 TextBox 控件的属性的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

为用户提供选择的控件

大多数应用程序需要向用户提供选择，如简单的“ Yes/No”选项、或者从包含成百个可能性的列表中进行选择。 Visual Basic 包含几种用于提供选择的标准控件。下列表格总结了这些控件及其用途。

提供这种特征	用这种控件
用户可选择一个或多个选项的小选择集	CheckBox
用户只能选择一项的小选择集	OptionButton（如果需要附加的组,则使用 Frame）
用户可从中进行选择的滚动列表	Listbox
一个可滚动选择列表附加一个文本编辑域。用户可以从表中选择或在编辑域中键入选择	ComboBox

用 CheckBox 选择单个选项

CheckBox 表明一个特定的状态是选定 (on) 还是清除 (off)。在应用程序中使用 CheckBox 为用户提供了“ True/False”或“ yes/no”的选择。因为 CheckBox 彼此独立工作，所以用户可以同时选择任意多个 CheckBox。例如，在图 3.9 中，可同时选定粗体和斜体。

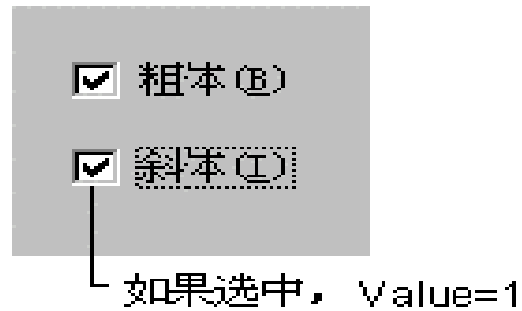


图 3.9 Check Boxes

CheckBox 应用程序

这个例子使用一个 `CheckBox` 来决定文本用普通字体还是用斜体字显示。关于此例的工作版本,请参阅应用程序示例 `Controls.vbp` 中的 `Check.frm`。如图 3.10 所示,此应用程序有一个 `TextBox`, 一个 `Label`, 一个 `CommandButton` 和两个 `CheckBox`。



图 3.10 CheckBox 示例

下表列出了应用程序中的对象的属性设置。

对象	属性	设置
Form	Name	frmCheck
	Caption	Check Box Example
Text box	Name	txtDisplay
	Text	Some sample text
First Check box	Name	chkBold

续表

	Caption	& Bold
Second Check box	Name	chkItalic
	Caption	& Italic
CommandButton	Name	cmdClose
	Caption	& Close

选定粗体或斜体时，CheckBox 的 Value 属性值设置为 1，不选定时为 0。缺省 Value 值为 0，所以除非改变 Value 属性值，否则第一次显示时不会选定 CheckBox。可以用常量 vbChecked 和 vbUnchecked 表示数值 1 和 0。

check box 应用程序中的事件

当单击 CheckBox 时，会发生 CheckBox 的 Click 事件。此事件过程测试是否选定 CheckBox(即是否 Value = vbChecked)。如果选定，通过设置 Font 对象的 Bold 或 Italic 属性，将文本转换为粗体或斜体；该 Font 对象是由 TextBox 的 Font 属性返回的。

```
Private Sub chkBold_Click ()
    If ChkBold.Value = vbChecked Then      '如果选定。
        txtDisplay.Font.Bold = True
    Else                                    '如果没有选定。
        txtDisplay.Font.Bold = False
    End If
End Sub
```



```
        End If
    End Sub
    Private Sub chkItalic_Click ()
        If ChkItalic.Value = vbChecked Then    '如果选定。
            txtDisplay.Font.Italic = True
        Else                                    '如果没有选定。
            txtDisplay.Font.Italic = False
        End If
    End Sub
```

用选项按钮进行选项分组

选项按钮表示给用户一组两个或更多的选择。但是，不同于 **CheckBox**，选项按钮总是作为一组的组成部分工作的；因此，选择一个选项按钮就会立即清除该组中的其它按钮。定义一个选项按钮组，告诉用户“这是一组选择，从中选一个，而且只选一个”。

例如，在图 3.11 所示的选项按钮组中，用户只能在三个选项按钮中选择一个。

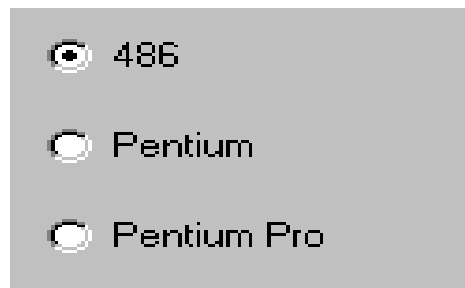


图 3.11 选择一个选项按钮

创建选项按钮组

直接放在一个窗体中（也就是不在 `Frame` 或 `PictureBox` 中）的所有的选项按钮构成一组。如果想创建别的选项按钮组，必须将其中一些选项按钮组放到 `Frame` 或 `PictureBox` 中去。

所有在给定的 `Frame` 中的选项按钮，组成一个独立的组，在 `PictureBox` 中的所有选项按钮也是如此。用这种方法生成一个独立的组时，总是要先画好一个 `Frame` 或 `PictureBox`，然后，将选项按钮画到里面。图 3.12 显示了一个具有两个选项按钮组的窗体。

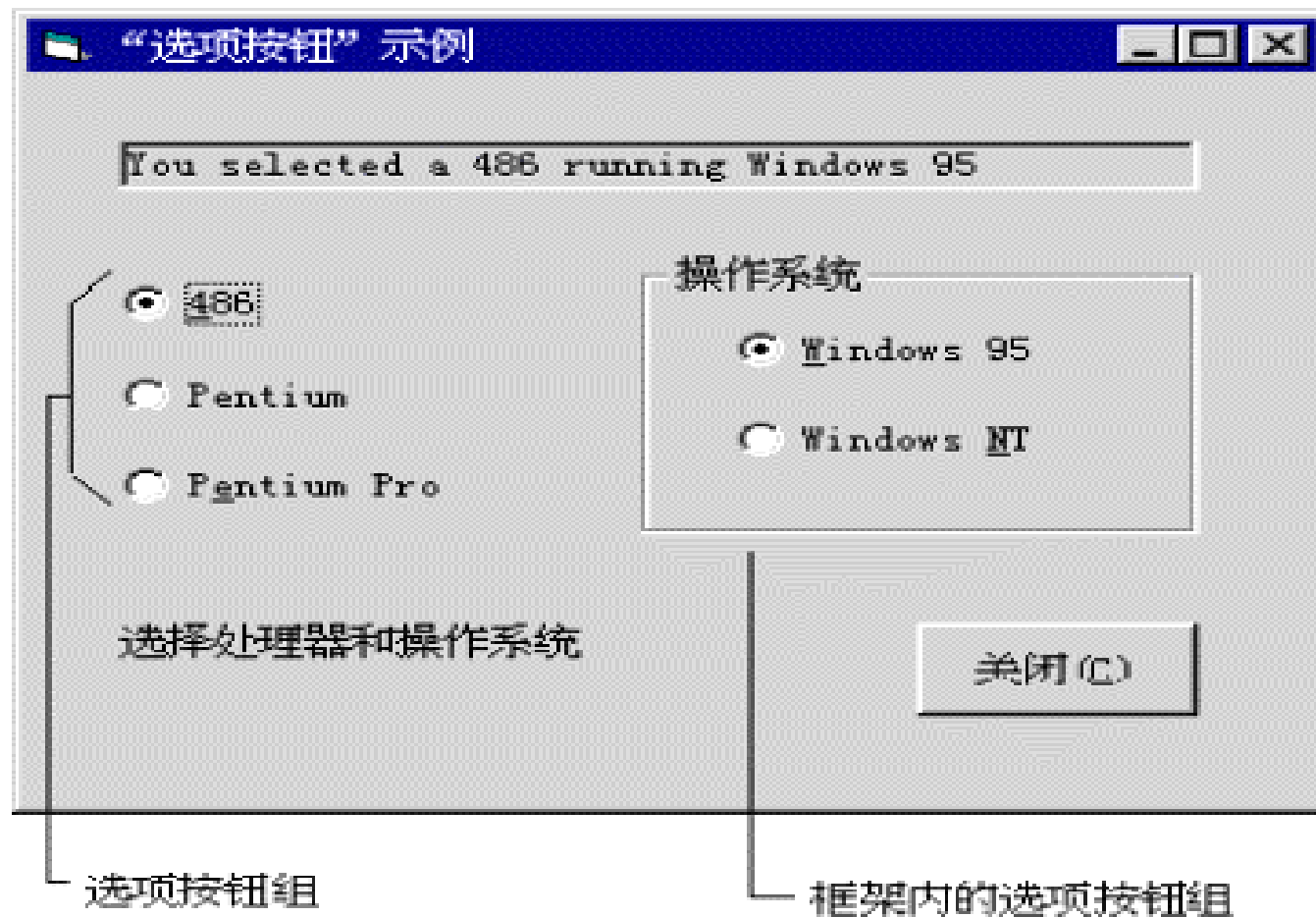


图 3.12 选项按钮组

在 Frame 中画了一组选项按钮后，用户只能在组内选择一个选项按钮。要在 Frame 中为控件分组，请按照以下步骤执行：

1. 在“工具箱”中选择 Frame 控件，并在窗体上画出 Frame。

2. 在“工具箱”中选择 `OptionButton` 控件，并在 `Frame` 内画出控件。

3. 如果还想在 `Frame` 中增加选项按钮，重复步骤 2。

先画 `Frame`，再在 `Frame` 上画每个控件，这样移动 `Frame` 时控件会一起移动。如果将已经存在的控件移到 `Frame` 上，此控件不会和 `Frame` 一起移动。

注意 如果将已存在的控件分组到一个 `Frame` 中去，可以选择这些控件，然后剪贴到 `Frame` 或 `Picture` 控件上去。

控件的容器

当控件是独立的对象时，窗体和控件之间存在着某种父子关系。图 3.12 表明选项按钮怎样才能包含在一个窗体或一个 `Frame` 控件中。

要理解容器的概念，首先要把所有的控件都想象为所在窗体的孩子。实际上，大多数控件都支持只读 `Parent` 属性，它的值就是控件所在的窗体。作为孩子，控件决定其在父窗体上的位置。控件的 `Left` 属性和 `Top` 属性是相对父窗体而言，且控件不能移出父窗体的边界。移动容器的同时也移动了控件，因此控件与容器的相对位置不变（即控件的 `Left` 属性和 `Top` 属性不变）。

选定或禁止选项按钮

一个选项按钮可以用以下方法选择：

- 在运行期间用鼠标单击选项按钮。

- 用 **Tab** 键定位到选项按钮组，然后在组内使用方向键（箭头键）定位选项按钮。
- 用代码将它的 **Value** 属性设置为 **True**：
`optChoice.Value = True`
- 使用在 **Label** 的标题中指定的快捷键。
- 要使某个按钮成为选项按钮组中的缺省按钮，只要在设计时将其 **Value** 属性设置成 **True**。它保持被选中状态，直到用户选择另一个不同的选项按钮或用代码改变它。

要禁用选项按钮，将其 **Enabled** 属性设置为 **False**。程序运行时，若此选项按钮变灰，表示无法选取此选项按钮。

Options 应用程序

图 3.13 所示的窗体使用选项按钮为一台计算机选择处理器类型和操作系统。当用户在组内选择了选项按钮后，**Label** 的标题就改变，反映当前的选择。关于此例的工作版本，请参阅应用程序示例 **Controls.vbp** 中的 **Options.frm**。



图 3.13 选项按钮举例

下表列出了应用程序中对象属性的设置。

对象	属性	设置
Label	Name	lblDisplay
	Caption	(空)
Command Button	Name	cmdClose
	Caption	& Close

续表

First option button	Name	opt486
	Caption	&486
Second option button	Name	opt586
	Caption	&Pentium
	Value	true
Third option button	Name	opt686
	Caption	&Pentium Pro
Frame	Name	fraSystem
	Caption	&Operating System
Fourth option button	Name	optWin95
	Caption	Windows 95
Fifth option button	Name	optWinNT
	Caption	Windows NT
	Value	True

Options 应用程序中的事件

选项应用程序对事件的响应如下：

- 前三个选项按钮的 Click 事件为窗体级的 string 变量 strComputer 分配一个相应的描述。
- 后两个选项按钮的 Click 事件为第二个窗体级变量 strSystem 分配一个相应的描述。

这个方法的关键就在于使用这两个窗体级变量，strComputer 和 strSystem。这两个变量具有不同的 string 值，选项按钮的最后选定就依赖于这些值。

每次选择新的选项按钮，其 Click 事件的代码就更新相应变量的值。

```
Private Sub opt586_Click()  
    strComputer = "Pentium"  
    Call DisplayCaption  
End Sub
```

然后它调用一个名为 DisplayCaption 的过程，此过程把两个变量连在一起并改变 Label 的 Caption 属性。

```
Sub DisplayCaption()  
    lblDisplay.Caption = "You selected a " & _  
        strComputer & " running " & strSystem  
End Sub
```

使用子过程是因为改变 Caption 属性的过程对所有五个选项按钮来说基本是一样的，只不过变量的值因情况而异。这就可以避免在每个 Click 事件中重复同样的代码。

详细信息 变量和子过程将在第五章“编程基础”中讨论。

使用 ListBox 和 ComboBox

ListBox 和 ComboBox 为用户提供了选择。按照缺省规定，选项以垂直单

列方式显示，也可以设置成多列方式。如果项目数量超过 `ComboBox` 或 `ListBox` 所能显示的数目，`ScrollBar` 自动出现在控件上。于是用户可以在列表中上下左右滚动。图 3.14 为一个单列 `ListBox`。



图 3.14 单列 `ListBox`

`ComboBox` 兼有 `TextBox` 和 `ListBox` 两者的功能。该控件允许用户通过键入文本或选择列表中的项目来进行选择。图 3.15 为一个 `ComboBox`。



图 3.15 `ComboBox`

与其它含有单值的控件相反，例如 `Label` 的 `Caption` 属性或 `TextBox` 的

text 属性，ListBox 和 ComboBox 含有多个值或值的集合。在运行时用内置的方法增加、删除及保持集合中的值。要将几条项目添加到名为 List1 的 ListBox 中，其代码如下：

```
List1.AddItem "Paris"
```

```
List1.AddItem "New York"
```

```
List1.AddItem "San Francisco"
```

ListBox 和 ComboBox 是在有限空间为用户提供大量选项的有效方法。

详细信息 关于 ListBox 和 ComboBox 控件的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

使用 ScrollBar 作为输入设备

滚动条通常与 TextBox 或窗口一起使用，但有时也可作输入设备。因为 ScrollBar 控件可按比例指示当前位置，所以可以单独使用以控制程序输入，如控制声音音量或调整图片颜色。HScrollBar（水平）和 VScrollBar（垂直）控件是独立的，它有自己的事件、属性和方法集。ScrollBar 控件与隶属于 TextBox、ListBox、ComboBox 或 MDI 窗体的内部 ScrollBar 不同。（TextBox 和 MDI 窗体具有 ScrollBars 属性，可增加或删除隶属于控件的 ScrollBar。）

现在 Windows 界面提示行支持 Slider 控件作为输入设备而不用 ScrollBar。关于 Slider 控件的例子，请参阅 Windows 95 的控件面板。Visual Basic 的专业版和企业版中包含一个 Windows 95 风格的 Slider 控件。

详细信息 关于 ScrollBar 控件的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

显示图片和图形的控件

因为 Windows 是图形用户界面，所以在应用程序上界面显示图形图像的方法十分重要。Visual Basic 包含四个控件以简化与图形有关的操作，它们是 PictureBox 控件、Image 控件、Shape 控件和 Line 控件。Image、Shape 和 Line 控件有时被称作轻量图形控件。它们需要较少的系统资源，因此比 PictureBox 控件显示得快并且包含 PictureBox 中可用的属性、方法和事件的子集。每一个控件都适用于一个特定的目的。

提供的功能	使用的控件
其它控件的容器	PictureBox
Printing 或 graphics 方法	PictureBox
显示图片	Image 或 PictureBox 控件
显示简单图形元素	Shape 或 Line 控件

使用 PictureBox 控件

PictureBox 控件的主要作用是为用户显示图片。实际显示图片由 Picture 属

性决定。Picture 属性包括被显示的图片的文件名（及可选的路径名）。

注意 窗体对象也具有 Picture 属性，通过设置该属性可直接在窗体背景上显示图片。

要在运行时显示或替换图片，可利用函数 LoadPicture 来设置 Picture 属性。提供图片文件名和可选路径名，由 LoadPicture 函数处理加载和显示图片的细节。

```
picMain.Picture = LoadPicture("VANGOGH.BMP")
```

PictureBox 控件具有 AutoSize 属性，当该属性设置为 True 时，PictureBox 能自动调整大小与显示的图片匹配。如果要用 AutoSize 属性设置为 True 的 PictureBox，设计窗体时就需要特别小心。图片将不考虑窗体上的其它控件而调整大小，这可能导致意想不到的后果，如覆盖其它控件。设计时应通过加载每一幅图片来检查是否有这种现象发生。

使用 PictureBox 作为容器

Picture box 控件也可以用作其它控件的容器。像 Frame 控件一样，可以在 PictureBox 控件上面加上其它控件。这些控件随 PictureBox 移动而移动，其 Top 和 Left 属性是相对 PictureBox 而言，而与窗体无关。

PictureBox 容器的一种普通用法是 ToolBar 或 StatusBar。可将 Image 控件置于这些容器中作为按钮或添加 Label 显示状态信息。将 Align 属性设置为 Top、Bottom、Left 或 Right，PictureBox 将粘贴在窗体的边缘。图 3.16 为一个 Align 属性设置为 Bottom 的 PictureBox。它包含两个可用于显示状态

信息的 Label 控件。

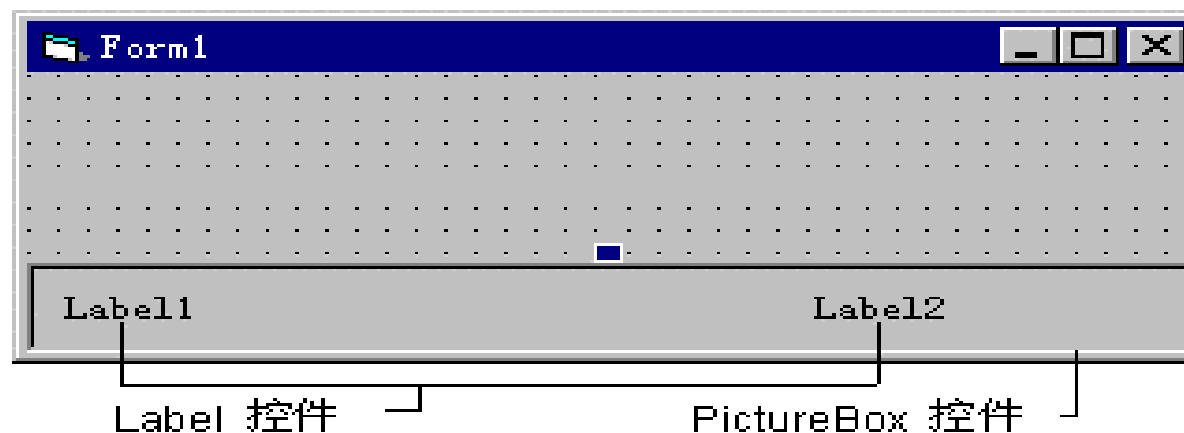


图 3.16 用作 StatusBar 的 PictureBox

PictureBox 的其它用法

PictureBox 控件有几种方法使 PictureBox 可用于其它目的。例如：把 PictureBox 看作是一块空画布，可以在它上面画画或打印，或者显示文本、图形，甚至是简单的动画。

Print 方法允许向 PictureBox 控件输出文本，如同向打印机输出一样。有几种字体属性可以控制由 Print 方法输入的文本的特性；Cls 方法可以清除输出。

Circle、Line、Point 和 Pset 方法可以用于在 PictureBox 中画图形。一些属性如 DrawWidth、FillColor 和 FillStyle 允许自定义图形的外观。

用 PaintPicture 方法在图片控件内移动 Image、并在几幅不同的 Image 间

快速变化就能产生动画。

详细信息 关于 PictureBox 控件的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

轻量图形控件

Image、Shape 和 Line 控件都被认为是轻量图形控件，也就是说，它们只支持 PictureBox 的属性、方法和事件的一个子集。因此，它们需要较少的系统资源而且加载也比 PictureBox 控件更快。

用 Image 控件代替 PictureBox

Image 控件与 PictureBox 控件相似，但它只用于显示图片。它不能作为其它控件的容器，也不支持 PictureBox 的高级方法。

图片加载于 Image 控件的方法和它们加载于 PictureBox 中的方法一样。设计时，将 Picture 属性设置为文件名和路径，运行时，利用 Loadpicture 函数。

Image 控件调整大小的行为与 PictureBox 不同。它具有 Stretch 属性，而 PictureBox 具有 AutoSize 属性。将 AutoSize 属性设为 True 可使 PictureBox 根据图片调整大小，设为 False 则图片将被剪切（只有一部分图片可见）。Stretch 属性设为 False（缺省值）时，Image 控件可根据图片调整大小。将 Stretch 属性设为 True 将根据 Image 控件的大小来调整图

片的大小，这可能使图片变形。

详细信息 关于 Image 控件的详细信息，请参阅第七章“使用 Visual Basic's 的标准控件”。

使用 Image 控件创建自己的按钮

Image 控件也可以识别 Click 事件，因此可在要用 CommandButton 的任何地方使用该控件。这种方法可以方便地建立有图片的按钮来取代标题。将几个 Image 控件组成一组，水平置于屏幕顶端（通常在一个 PictureBox 中），可以在应用程序中建立一个 ToolBar。

例如，对 Test Buttons 一例给出的 Image 控件，用户可以像选择 CommandButton 一样来选择它。窗体第一次被显示时，控件显示 Visual Basic 的图标库中三个交通图标中的一个。每单击 Image 控件一次，就显示一个不同的图标（该例的工作版本，请参阅 Controls.vbp 示例应用程序中的 Button.frm）。

如果设计时观察一下窗体，就会发现它实际包含所有三个图标，它们堆集在一起。改变顶部 Image 控件的 Visible 属性为 False，下一幅图像（如果其 Visible 属性为真）将显示于顶部。

图 3.17 为一个交通图标的 Image 控件 (Trffc10a.ico)。



图 3.17 有一个交通图标的 Image 控件

要在 Image 控件的周围建立一个边框，将 BorderStyle 属性设为 1——固定的单边框。

注意 与 CommandButton 不同，单击 Image 控件时，并不显示它被按下的动作。也就是说除非在 MouseDown 事件中改变位图，否则“按钮”正在被按下对用户来说是看不到的。

详细信息 关于在 Image 控件中显示图形图像的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

使用 Shape 控件和 Line 控件

Shape 和 Line 控件可用来在窗体表面画图形元素。这些控件不支持任何事件，只用于表面装饰。

有几种属性可以控制 Shape 控件的外观。通过设置 Shape 属性，Shape 控件可显示为矩形、正方形、椭圆型、圆角矩形或圆角正方形。设置 BorderColor 和 FillColor 属性可以改变颜色，BorderStyle、BorderWidth、FillStyle 和 DrawMode 属性控制如何画图。

Line 控件与 Shape 控件相似，但仅用于画线。

详细信息 关于 Shape 和 Line 控件的附加信息，请参阅第七章“使用 Visual Basic 的标准控件”。

Images 应用程序

图 3.18 所示窗体用了四个 Image 控件，一个 Shape 控件，一个 PictureBox 控件和一个 CommandButton 控件。当用户选择扑克牌符号时，Shape 控件使该符号突出显示，并且在 PictureBox 中显示相应的描述。关于本例的工作版本，请参阅 Controls.vbp 示例应用程序中的 Images.frm。

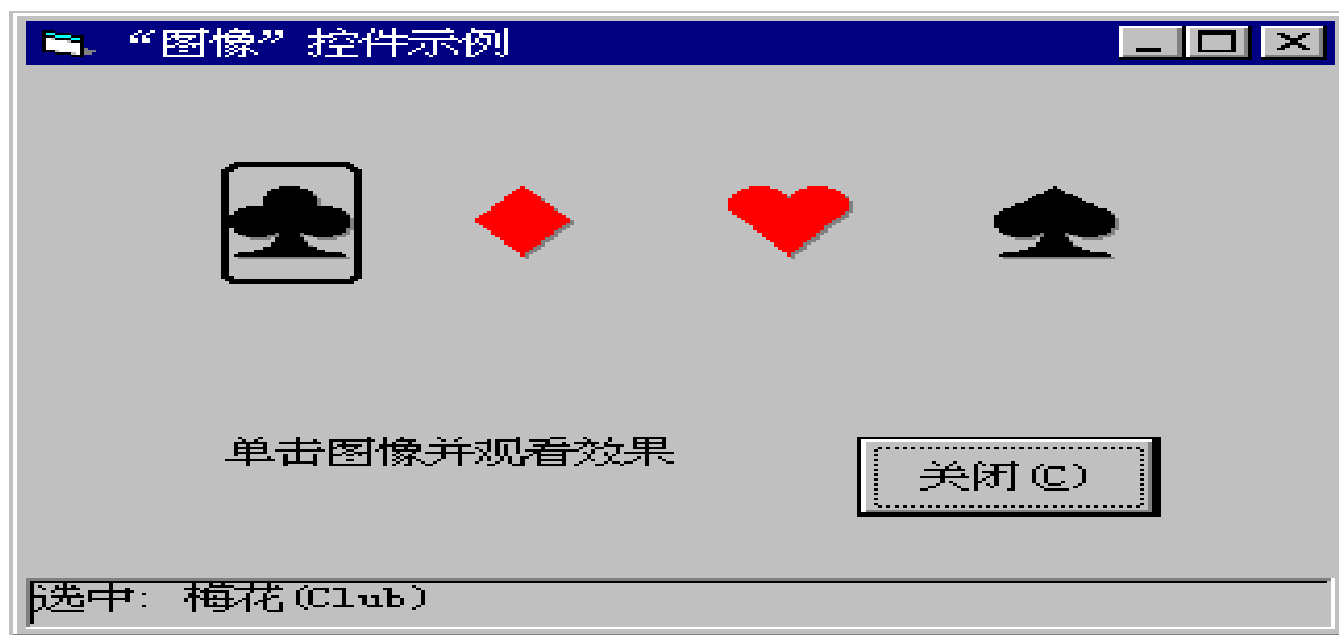


图 3.18 Image 和 Shape 控件的例子

下表列出了应用程序中对象的属性设置。

对象	属性	设置
Picture box	Name	picStatus
	Align	Bottom
First image control	Name	imgClub
	Picture	Spade.ico
Second image control	Name	imgDiamond
	Picture	Diamond.ico
Third image control	Name	imgHeart
	Picture	Heart.ico
Fourth image control	Name	imgSpade
	Caption	Spade.ico
Shape control	Name	shpCard
	Shape	4 - Rounded Rectangle
	BorderWidth	2
	Height	735
	Width	495
CommandButton	Name	cmdClose
	Caption	&Close

Images 应用程序中的事件

Image 应用程序对事件的响应如下：

- 每一个 Image 控件的 Click 事件使 Shape 控件的 Left 属性与它自己的 Left 属性相同，并将图形移到 Image 的顶端。
- 调用 PictureBox 的 Cls 方法清除 StatusBar 的当前标题。
- 调用 PictureBox 的 Print 方法在 StatusBar 上打印新标题。

Image 控件 Click 事件的代码如下：

```
Private Sub imgHeart_Click()  
    shpCard.Left = imgClub.Left  
    picStatus.Cls  
    picStatus.Print "Selected: Club"  
    shpCard.Visible = True  
End Sub
```

注意，Click 事件代码中第一行利用 = 操作符为 Shape 控件的 Left 属性赋一个值（让它等于 Image 控件的 Left 属性）。下面两行调用方法，因而不需要操作符。在第三行中，值 ("Selected: Club") 是 Print 方法的一个参数。

该应用代码中还有一行很重要；它在 Form Load 事件中。

```
shpCard.Visible = False
```

将 Shape 控件的 Visible 属性设为 False，Shape 控件被隐藏起来直至单击

第一幅 Image。在 Image 控件 Click 事件的最后一步将 Visible 属性设为 True。

详细信息 关于属性、方法和事件详细信息，请参阅第五章“编程基础”。

附加控件

Visual Basic 工具箱还包含几个其它的标准控件。有些控件可用于处理外部数据库中的大量数据。有些控件可用于访问 Windows 文件系统。还有其它一些控件难以分类,但却十分有用。

在 Visual Basic 应用程序中还可使用 ActiveX 控件，以前称为自定义控件或 OLE 控件，用法与使用标准控件一样。Visual Basic 的专业版和企业版中的若干 ActiveX 事件同样具有创建自定义控件的功能。可以想像，加上大量厂商制作的附加 ActiveX 控件，几乎可以完成任何工作。

详细信息 关于使用 ActiveX 控件的详细信息，请参阅第四章“工程的管理”。

数据访问控件

现代商业中，大多数信息存放于一个或多个中央数据库中。Visual Basic 提供了多种数据访问控件，可以访问多数流行的数据库，如 Microsoft Access 和 SQL 服务器。

- ADO Data 控件用来连接数据库。可把它看成是数据库与窗体中其它控件之间的一个管道线。Data 控件的属性、方法和事件允许从应用程序内部定位和处理外部数据。
- DataList 控件与 ListBox 控件相似。DataList 控件与 ADO Data 控件一起使用时，它将自动被来自外部数据库某一字段的数据清单填充。
- DataCombo 控件如同 DBList 和 TextBox 的组合。TextBox 区域内选中的文本可以编辑，变更出现在底层数据库中。
- DataGrid 控件显示在网格或表中的数据。与 ADO Data 控件一起使用时，它显示来自于外部数据库多个字段的完全可编辑数据。
- Microsoft Hierarchical FlexGrid 控件是唯一能显示数据的多个视图的控件。可把它看作网格和树或大纲控件的组合。运行时，用户可重新安排行和列以提供数据的不同视图。

详细信息 关于 Data 控件的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”。

文件系统控件

Visual Basic 提供三个控件以增加应用程序中的文件处理能力。通常，这些控件在一起使用以查看驱动器、目录和文件。有一些特殊的属性和事件使它们相互联系。

- DriveListBox 控件看起来像 ComboBox。它提供下拉式驱动器清单，用户可以从其中作出选择。

- DirListBox 与 ListBox 控件相似，但它具有显示当前所选驱动器目录清单的内置能力。
- FileListBox 控件看起来像显示所选目录的文件名清单的 ListBox。

注意 这些控件主要是为在早期 Visual Basic 版本中创建的应用程序的向下兼容性而提供的。CommonDialog 控件提供处理文件访问的简单方法。关于 CommonDialog 控件的详细信息，请参阅本章后面的“其它控件”。

其它控件

Visual Basic 还提供了其它几个标准控件。每个控件都服务于唯一的目的。

- Timer 控件可以在应用中以重复时间间隔产生一个事件。这对不需要与用户交互的代码的执行非常有用。
- OLE 容器控件很容易为应用程序增加链接和嵌入等功能。通过 OLE 容器控件，可以提供对任何 OLE——可用的应用程序（如 Microsoft Excel、Word 和其它许多应用程序）的功能的访问，。
- CommonDialog 控件在应用程序中增加内置对话框，用来选择文件、颜色、字体和打印函数。

详细信息 关于标准控件的附加信息，请参阅第七章“使用 Visual Basic 的标准控件”。

焦点概述

焦点是接收用户鼠标或键盘输入的能力。当对象具有焦点时，可接收用户的输入。在 Microsoft Windows 界面，任一时刻可运行几个应用程序，但只有具有焦点的应用程序才有活动标题栏，才能接受用户输入。在有几个 TextBox 的 Visual Basic 窗体中，只有具有焦点的 TextBox 才显示由键盘输入的文本。

当对象得到或失去焦点时，会产生 GotFocus 或 LostFocus 事件。窗体和多数控件支持这些事件。

事件	描述
GotFocus	对象得到焦点时发生
LostFocus	对象失去焦点时发生。LostFocus 事件过程主要用来对更新进行证实和有效性检查，或用于修正或改变在对象的 GotFocus 过程中建立的条件

下列方法可以将焦点赋给对象。

- 运行时选择对象。
- 运行时用快捷键选择对象。
- 在代码中用 SetFocus 方法。

有些对象，它是否具有焦点是可以看出来的。例如，当命令按钮具有焦点时，标题周围的边框将突出显示（见图 3.19）。

只有当对象的 Enabled 和 Visible 属性为 True 时，它才能接收焦点。Enabled 属性允许对象响应由用户产生的事件，如键盘和鼠标事件。Visible

属性决定了对象在屏幕上是否可见。

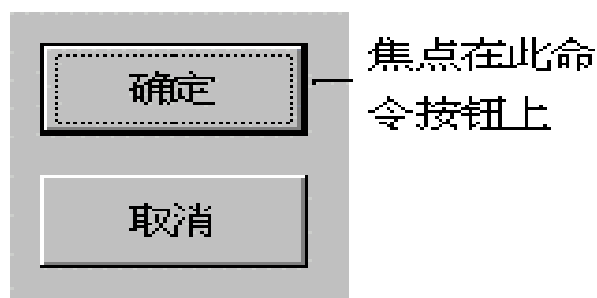


图 3.19 有焦点的命令按钮

注意 只有不包含任何可接收焦点的控件的窗体，才能接收焦点。

控件的 Validate 事件

控件也有一个 `Validate` 事件，在控件失去焦点前发生。然而，仅当即将接受焦点的控件的 `CausesValidation` 属性设置为 `True` 时该事件发生。在许多情况下，由于 `Validate` 事件发生在失去焦点之前，因此对数据验证来说它比 `LostFocus` 事件更适用。详细信息，请参阅第七章“使用 Visual Basic 标准控件”中的“通过限制焦点验证控件数据”。

不能接受焦点的控件

有些控件，例如轻量控件，不能接受焦点。轻量控件包含下面的控件：

- `Frame` 控件

- Image 控件
- Label 控件
- Line 控件
- Shape 控件

此外，运行时不可见的控件，如 Timer 控件，不能接受焦点。

设置 Tab 键顺序

例如，假设您建立了两个名称为 Text1 和 Text2 的 TextBox，然后又建立了一个名称为 Command1 的 CommandButton。应用程序启动时，Text1 具有焦点。按 TAB 键将使焦点按控件建立的顺序在控件间移动，如图 3.20 所示。

设置 TabIndex 属性将改变一个控件的 tab 键顺序。控件的 TabIndex 属性决定了它在 tab 键顺序中的位置。按照缺省规定，第一个建立的控件其 TabIndex 值为 0，第二个的 TabIndex 值为 1，以此类推。当改变了一个控件的 tab 键顺序位置，Visual 自动为其它控件的 tab 键顺序位置重新编号，以反映插入和删除。例如，要使 Command1 变为 tab 键顺序中的首位，其它控件的 TabIndex 值将自动向上调整，如下表所示。

控 件	变化前的 TabIndex 值	变化后的 TabIndex 值
Text1	0	1
Text2	1	2
Command1	2	0

因为编号从 0 开始，TabIndex 的最大值总是比 tab 键顺序中控件的数目少 1。即使 TabIndex 属性值高于控件数目，Visual Basic 也会将这个值转换为控件数减一。

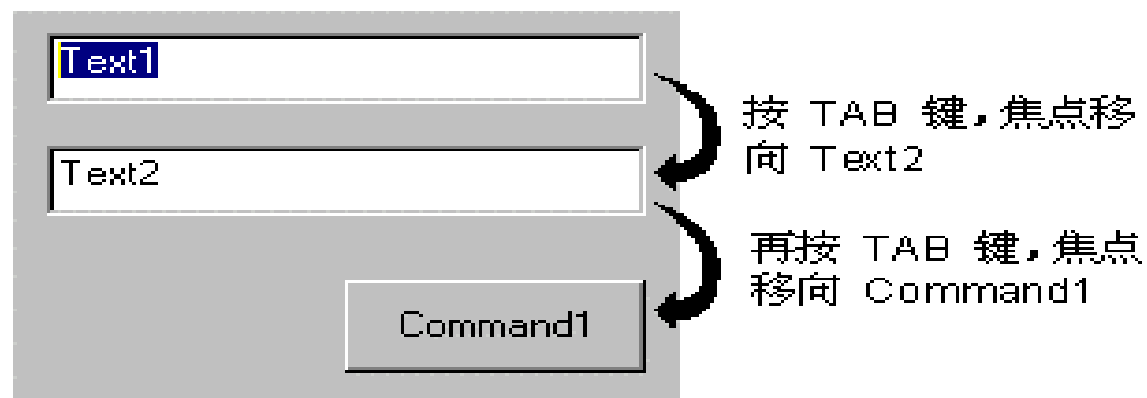


图 3.20 Tab 键例子

注意 不能获得焦点的控件以及无效的和不可见的控件不具有 TabIndex 属性，因而不包含在 tab 键顺序中。按 TAB 键时，这些控件将被跳过。

在 Tab 键顺序中删除控件

通常，运行时按 TAB 键能选择 tab 键顺序中的每一控件。将控件的 TabStop 属性设为 False (0)，便可将此控件从 tab 键顺序中删除。

TabStop 属性已置为 False 的控件，仍然保持它在实际 tab 键顺序中的位置，只不过在按 tab 键时这个控件被跳过。

注意 一个 OptionButton 组只有一个 tab 站。选中的按钮（即 Value 值为 True 的按钮）的 TabStop 属性自动设为 True，而其它按钮的 TabStop 属性为 False。

菜单基础

如果应用程序要为用户提供一组命令，菜单提供一种方便的给命令分组的方法，并使用户容易访问这些命令。

图 3.21 举例说明了无标题窗体的菜单界面元素。

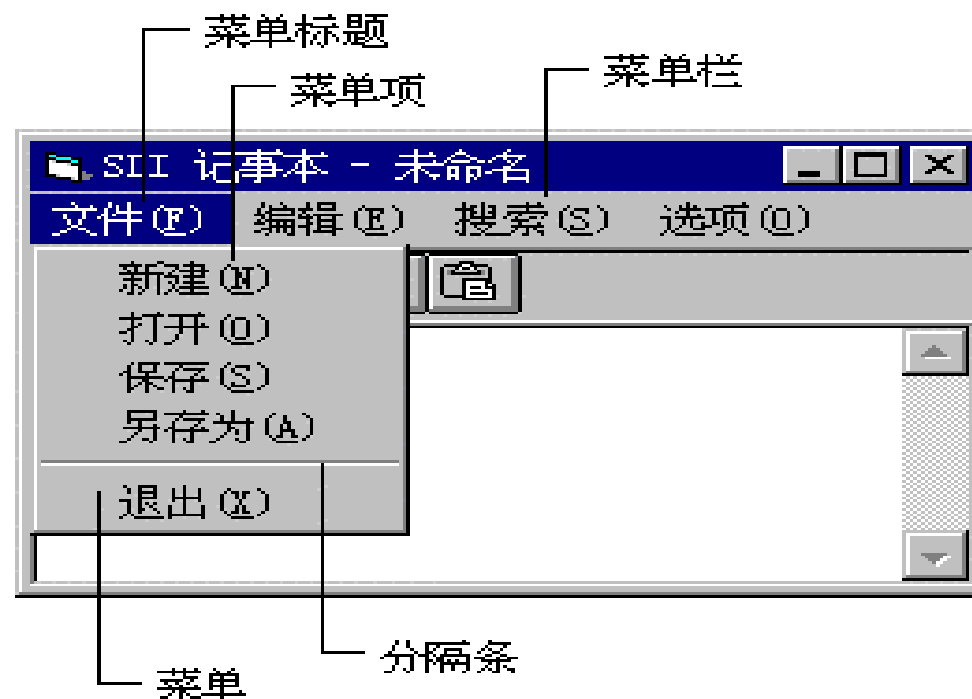


图 3.21 Visual Basic 窗体的菜单界面元素

菜单栏立即出现在窗体的标题栏下面，并包含一个或多个菜单标题。当单击一个菜单标题（如“文件”），包含菜单项目的列表就被拉下来。菜单项可以包括命令（如“新建”和“退出”）、分隔条和子菜单标题。用户看到的每个菜单项和在“菜单编辑器”中定义的一个菜单控件对应（“菜单编辑器”将在本章后面讨论）。

要使您的应用程序简单好用，应该将菜单项按其功能分组。例如，图 3.21 中与文件有关的命令“新建”，“打开”，和“另存为”都列入“文件”菜单。

有些菜单项目直接执行动作，如“文件”菜单中的“退出”菜单项将关闭应用程序。其它菜单项显示一个对话框，即要求用户提供应用程序执行动作所需信息的窗口。应该在这些菜单项后加上省略符 (...)。例如，当从“文件”菜单中选择“另存为...”时，出现“文件另存为”对话框。

菜单控件是一个对象；与其它对象一样，它具有定义它的外观与行为的属性。在设计或运行时可以设置 `Caption` 属性、`Enabled` 和 `Visible` 属性、`Checked` 属性以及其它属性。菜单控件只包含一个事件，即 `Click` 事件，当用鼠标或键盘选中该菜单控件时，将调用该事件。

详细信息 关于菜单控件的详细信息，请参阅第六章“创建用户界面”中的“用菜单编辑器创建菜单”。

弹出式菜单

弹出式菜单是显示于窗体之上，独立于菜单栏的浮动式菜单，如图 3.22 所示。显示在弹出式菜单上的项取决于鼠标右键按下时指针的位置，因此，弹出式菜单又称为上下文菜单（在 Windows 95 中，击鼠标右键可激活上下文菜单。）应该用弹出式菜单提供一种访问公共的上下文命令的高效方法。例如，用鼠标右键单击一个 `TextBox`，将会出现一个上下文菜单（如图 3.22 所示）。

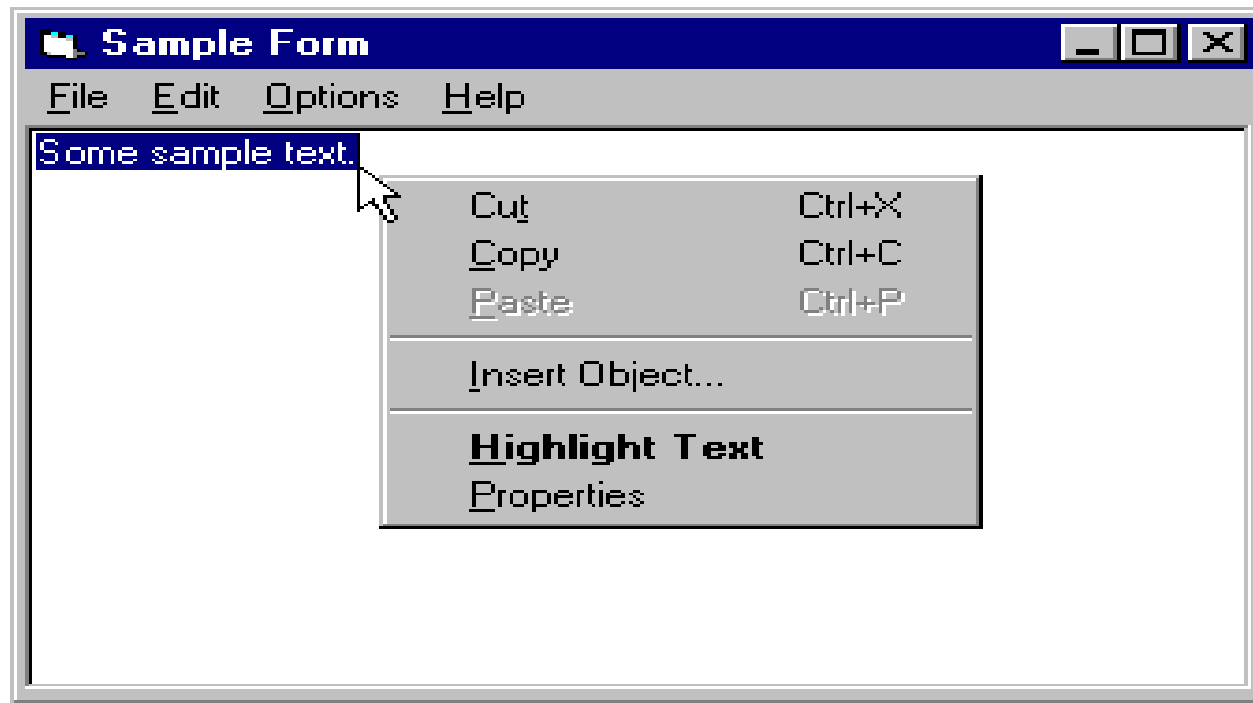


图 3.22 弹出式菜单

任何至少有一个菜单项的菜单，运行时都可以显示为弹出式菜单。使用 `PopupMenu` 方法可显示弹出式菜单。

详细信息 关于弹出式菜单的详细信息，请参阅第六章“创建用户界面”中的“用菜单编辑器创建菜单”。

使用菜单编辑器

使用“菜单编辑器”，可向现存的菜单中增加新命令、用自己的命令替代现存的菜单命令、产生新的菜单和菜单栏，改变和删除现存菜单和菜单栏。“菜

单编辑器”的主要优点是使用方便。可以在只用很少编程的全交互方式中自定义菜单。

要显示“菜单编辑器”，请

- 在“工具”菜单中选择“菜单编辑器”
这将打开“菜单编辑器”，如图 3.23 所示



图 3.23 “菜单编辑器”

大多数菜单控件属性可用“菜单编辑器”设置，同样，所有的菜单属性也可以在“属性”窗口中得到。通常，在“菜单编辑器”中建立菜单，但要快速改变单个属性，一般使用“属性”窗口。

详细信息 关于创建菜单和使用“菜单编辑器”的详细信息，请参阅第六章“创建用户界面”中的“用菜单编辑器创建菜单”。

用对话框提示用户

在基于 Windows 的应用程序中，对话框被用来提示用户提供应用程序继续运行所需的数据或者向用户显示信息。对话框是一种特殊类型的窗体对象，可用以下三种方法之一建立：

- 使用 `MsgBox` 或 `InputBox` 函数的代码可以创建预定义对话框。
- 使用标准窗体或自定义已存在的对话框创建自定义对话框。
- 使用 `CommonDialog` 控件可创建标准对话框，如“打印”和“打开文件”。

图 3.24 是利用 `MsgBox` 函数创建预定义对话框的例子。



图 3.24 预定义对话框

当在代码中调用 `MsgBox` 函数时，该对话框被显示。代码如下：

```
MsgBox "Error encountered while trying to open file," & vbCrLf & "please  
retry.", vbExclamation, "Text Editor"
```

需要给 `MsgBox` 函数提供三条信息或三个参数：消息文本、决定对话框类型的常数（`numeric` 值）和标题。由于样式可以使用按钮和图标的各种各样组合，所以创建对话框更加容易。

由于大多数对话框需要用户的交互作用，所以通常显示为模态对话框。在继续使用应用程序的其它部分以前，必须关闭（隐藏或卸载）模态对话框。例如，若在切换到其它窗体或其它对话框前必须单击“确定”或者“取消”，那么这个对话框就是模态对话框。

非模态对话框不需要关闭就可以使焦点在该对话框和其它窗体之间移动。对话框显示时，可以在当前应用程序的其它地方继续工作。非模态对话框很少；通常因为应用程序继续前需要响应才显示对话框。Visual Basic 中“编辑”菜单的“查找”对话框就是一个非模态对话框。使用非模态对话框显

示常用的命令或信息。

详细信息 关于创建对话框的详细信息，请参阅第六章“创建用户界面”。

第四章 工 程 管 理

为了用 Visual Basic 创建应用程序，应当使用工程。一个工程是用来建造应用程序的文件的集合。本章叙述如何建造和管理工程。

当创建一个应用程序时，通常要创建一些新窗体；也可以利用或修改以前工程所创建的窗体。对于可能纳入工程的其它模块或文件同样如此。来自其它应用程序的 ActiveX 控件和对象也可在工程之间共享。

在工程的所有部件被汇集在一起并完成代码编写之后，便可以编译工程，创建一个可执行文件。

目 录

- 使用工程
- Visual Basic 工程的结构
- 创建、打开和保存工程
- 添加、删除和保存文件
- 在工程中添加控件
- 制作和运行可执行文件
- 设置工程选项

- 使用向导和外接程序

使用工程

在开发应用程序时，要使用工程来管理构成应用程序的所有不同的文件。一个工程包括：

- 跟踪所有部件的工程文件 (.vbp)。
- 每个窗体的文件 (.frm)。
- 每个窗体的二进制数据文件 (.frx)，它含有窗体上控件的属性数据。对含有二进制属性（例如图片或图标）的任何 .frm 文件都是不可编辑的，这些文件都是自动产生的。
- 每个类模块的一个文件 (.cls)，该文件是可选项。
- 每个标准模块的一个文件 (.bas)，该文件是可选项。
- 一个或多个包含 ActiveX 控件的文件 (.ocx)，该文件是可选项。
- 单个资源文件 (.res)，该文件是可选项。

工程文件就是与该工程有关的全部文件和对象的清单，也是所设置的环境选项方面的信息。每次保存工程时，这些信息都要被更新。所有这些文件和对象也可供其它工程共享。

当完成工程的全部文件之后，即可将此工程转换成可执行文件 (.exe)：从“文件”菜单，选取“制作 project.exe”命令。

注意 使用 Visual Basic 的专业版和企业版，还可以创建其它类型的可执

行文件，例如 .ocx 和 .dll 文件。本章假定所论及的是标准的 .exe 工程。与其它工程类型有关的补充信息，请参阅专业版和企业版中的《部件工具指南》。

详细信息 关于创建可执行文件的详细内容，请参阅本章后面的“制作和运行可执行文件”。有关二进制数据文件和工程文件的信息，请参阅附录 A “Visual Basic 的规范、限制和文件格式”。

工程资源管理器

当创建、添加或从一工程中删除可编辑文件时，Visual Basic 会反映工程资源管理器窗口中发生的变化，该窗口包含此工程的当前文件的列表。图 4.1 所示的工程资源管理器窗口中列出可以纳入 Visual Basic 工程的一些文件类型。

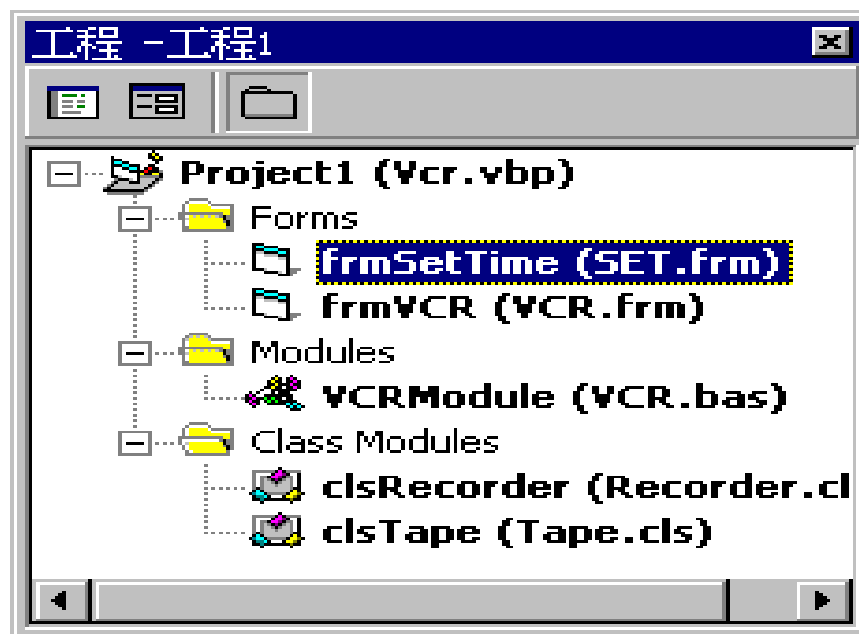


图 4.1 工程资源管理器窗口

工程文件

每次保存工程，Visual Basic 都要更新工程文件 (.vbp)。工程文件包含文件列表，它与出现在工程资源管理器窗口的文件列表相同，工程文件还引用工程中所使用的 ActiveX 控件和可插入对象。

通过双击一个现存工程的图标，或从“文件”菜单中选定“打开工程”命令，或拖动该文件并放入工程资源管理器窗口，可以打开这个现存工程文件。

详细信息 信息的指定格式存储在 .vbp 文件里，该格式在附录 A "Visual

Basic 的规范、限制和文件格式”中描述。

Visual Basic 工程的结构

以下几节讲述的是可以纳入工程的不同类型的文件和对象。

窗体模块

窗体模块（具有 .frm 文件扩展名）包含窗体及其控件的正文描述，包括它们的属性设置。它们也含有窗体级的常数、变量和外部过程的声明、事件过程和一般过程。

详细信息 关于创建窗体的详细内容，请参阅第二章“用 Visual Basic 开发应用程序”和第六章“创建用户界面”。关于窗体文件的格式和内容的信息，请参阅附录 A“Visual Basic 的规范、限制和文件格式”。

类模块

类模块（具有 .cls 文件扩展名）与窗体模块类似，只是没有可见的用户界面。可以使用类模块创建含有方法和属性代码的自己的对象。

详细信息 关于在类模块里编写代码的信息，请参阅第九章“用对象编程”中的“创建自己的类”。

标准模块

标准模块（具有 .bas 文件扩展名）可以包含类型、常数、变量、外部过程和公共过程的公共的或模块级的声明。

详细信息 关于使用模块的信息，请参阅第五章“编程基础”和第九章“用对象编程”。

资源文件

资源文件（具有 .res 文件扩展名）包含着无需重新编辑代码便可以改变的位图、字符串和其它数据。例如，如果计划用一种外语将应用程序本地化，可以将用户界面的全部正文串和位图存放在资源文件里，然后将资源文件本地化，而不是将整个应用程序本地化。一个工程最多包含一个资源文件。

详细信息 关于使用资源文件的详细内容，请参阅本章后面的“使用资源文件”和第十六章“国际发行”。

ActiveX 文档

ActiveX 文档 (.dob) 类似于窗体，但是在互联网资源管理器之类的互联网浏览器中是可以显示的。Visual Basic 的专业版和企业版能够创建 ActiveX 文档。

详细信息 关于 ActiveX 文档的更多信息，请参阅《Microsoft Visual Basic

6.0 部件工具指南》中的“创建 ActiveX 部件”，在专业版和企业版里可找到。

用户控件和属性页模块

用户控件 (.ctl) 和属性页 (.pag) 模块也类似于窗体，但它们被用于创建 ActiveX 控件及与其关联的用来显示设计时属性的属性页。Visual Basic 的专业版和企业版能够创建 ActiveX 控件。

详细信息 关于创建 ActiveX 控件的详细信息，请参阅《Microsoft Visual Basic 6.0 部件工具指南》中的“创建 ActiveX 部件”，在专业版和企业版本里可找到。

部件

除文件和模块以外，还有几个其它类型的部件可以添加到工程中。

ActiveX 控件

ActiveX 控件（具有 .ocx 文件扩展名）是可选的控件，它可以被添加到工具箱中并在窗体里使用。当安装 Visual Basic 时，Visual Basic 包含的含有控件的那些文件被复制到一个公共目录（Windows 95 下面的子目录 \Windows\System）中。可以从多种来源取得附加的 ActiveX 控件。也可以使用 Visual Basic 专业版和企业版创建自己的控件。

详细信息 关于使用被包含的 ActiveX 控件的更多信息，请参阅《Microsoft Visual Basic 6.0 部件工具指南》，在专业版和企业版本里可找到。

可插入的对象

可插入的对象，例如 Microsoft Excel 的工作表对象，是可用作建造集成方案时建造块的部件。一个集成方案可以包含由不同的应用程序创建的不同格式的数据，例如工作表、位图和正文。

详细信息 关于使用其它应用程序的对象的更多信息，请参阅第十章“用部件编程”。

引用

也可以添加能被应用程序使用的外部 ActiveX 部件的引用。通过访问“工程”菜单上的“引用”菜单项，使用“引用”对话框可指定引用。

详细信息 关于引用的更多信息，请参阅本章后面的“使用其它应用程序的对象”。

ActiveX 设计器

ActiveX 设计器是类的设计工具，从类出发可以创建对象。窗体的设计界面是缺省的设计器。从其它的源可取得附加的设计器。

详细信息 关于 ActiveX 设计器的更多信息，请参阅第九章“用对象编程”

中的“ActiveX 设计器”。

标准控件

标准控件是由 Visual Basic 提供的。标准控件，例如命令按钮或框架控件，总是包含在工具箱里，与可从工具箱里删除添加的 ActiveX 控件和可插入对象不同。

详细信息 关于标准控件的更多信息，请参阅第三章“窗体、控件和菜单”和第七章“使用 Visual Basic 的标准控件”。

创建、打开和保存工程

“文件”菜单上的四个命令允许创建、打开和保存工程。

菜单命令	描述
“新建工程”	关闭当前工程，提示用户保存所有修改过的文件。可以从“新建工程”对话框选定一个工程类。Visual Basic 然后创建一个带有单个新文件的新工程
“打开工程”	关闭当前工程，提示用户保存所有改动。随后，Visual Basic 打开一个现有工程，包括其工程文件 (.vbp) 中所列的窗体、模块和 ActiveX 控件。

续表

“保存工程”	更新当前工程的工程文件及其全部窗体、标准和类模块
“工程另存为”	更新当前工程的工程文件，用规定的文件名保存此工程文件。 Visual Basic 还提示用户保存所有修改过的窗体或模块

在工程间文件可以共享。像窗体这样的单个文件可以是多个工程的组成部分。注意，在一个工程中的窗体或模块所做的改变，将会传播到共享这个模块的所有工程。

详细信息 关于共享文件的详细内容，请参阅本章后面的“添加、删除和保存文件”。

使用多个工程

在 Visual Basic 的专业版和企业版中，可以同时打开多个工程。在建造和测试有关用户创建的控件或其它部件的解决方案时，这种功能很有用。在装入了多个工程时，工程资源管理器窗口的标题将变成“工程组”，而所有打开的工程部件都会显示出来。

要向当前工程组添加附加工程，请按照以下步骤执行：

1. 在“文件”菜单中选取“添加工程”。

“添加工程”对话框被显示。

2. 选定现有工程或新的工程类型，并选取“打开”。

要从现有工程组里删除一个工程，请按照以下步骤执行：

1. 在“工程资源管理器”里选定一个工程或一个工程部件。

2. 在“文件”菜单中选取“删除工程”。

详细信息 为获取有关使用多个工程的详细信息，请参阅《Microsoft Visual Basic 6.0 部件工具指南》中的“创建 ActiveX 部件”，在专业版和企业版里可找到。

添加、删除和保存文件

在工程中使用多个文件和使用工程本身类似。

要向工程中添加文件，请按照以下步骤执行：

1. 选定“工程”、“添加 filetype”（这里，filetype 是文件的类型）。

“添加 filetype”对话框（图 4.2）被显示。

2. 选定一个现存的文件或一个新的文件类型，并选取“打开”。

在工程中添加文件时，是简单地将对于该现存文件的引用纳入工程；而不是添加该文件的复制件。因此，如果更改文件并保存它，这个更改会影响包含此文件的任何工程。如若想改变文件而不影响其它工程，应在“工程资源管理器”里选定该文件，从“文件”菜单选取“filename 另存为”，然后以一个新的文件名保存此文件。

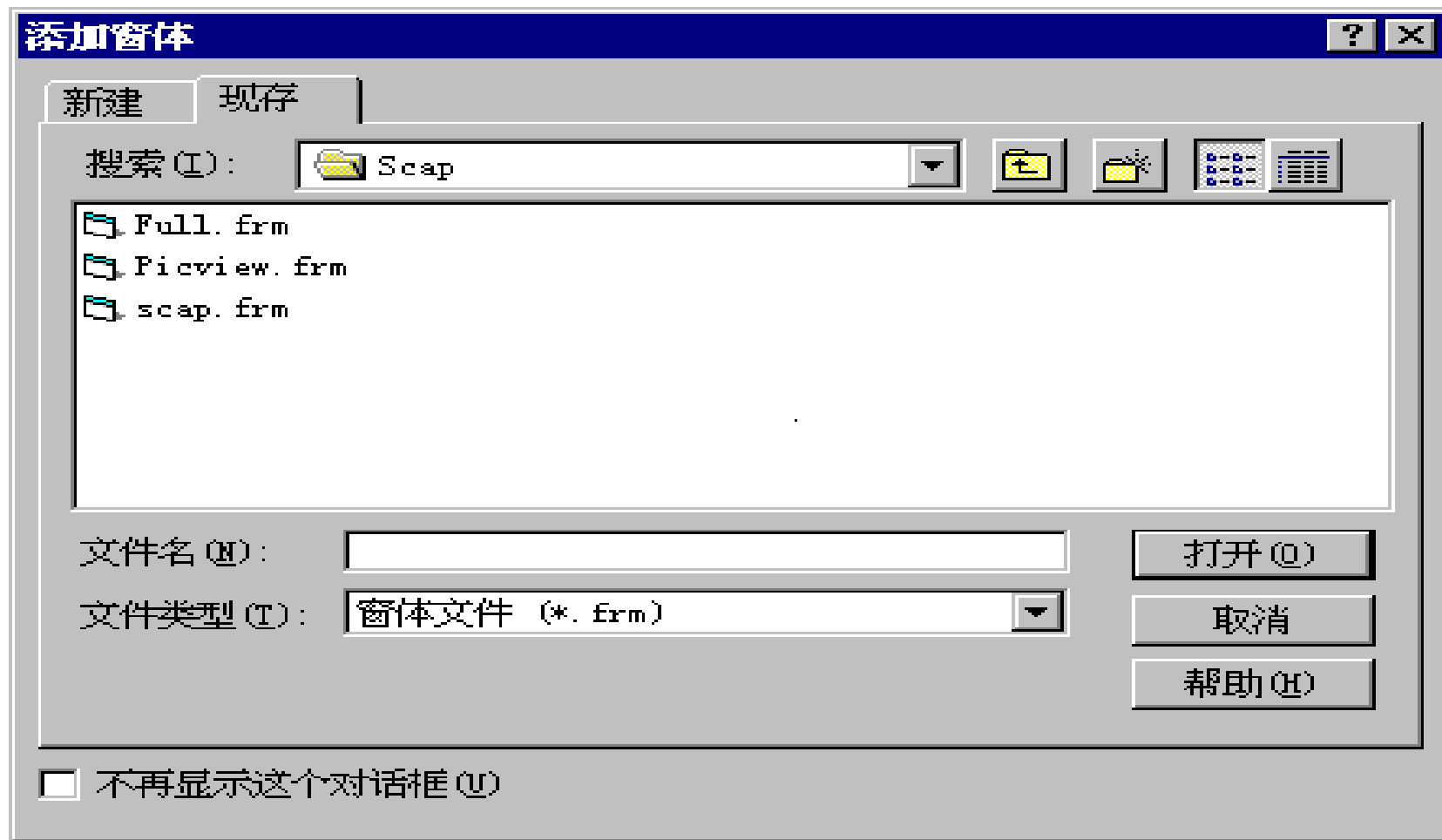


图 4.2 “添加窗体”对话框

注意 可以从 Windows 的“资源管理器”、“文件管理器”或“网上邻居”拖动文件并放入“工程”窗口，将它们添加到一个工程。也可以拖动 .ocx 文件并放入工具箱，以添加新的控件。

要从工程中删除文件，请按照以下步骤执行：

1. 在“工程资源管理器”中选定该文件。
2. 从“工程”菜单，选取“删除 filename”。
3. 此文件将从工程里删除掉，但是仍存在于磁盘上。

如果从工程里删除了文件，在保存此工程时 Visual Basic 更新此工程文件中的这个信息。但是，如果在 Visual Basic 之外删除一个文件，Visual Basic 不能更新此工程文件；因此，当打开此工程时，Visual Basic 将显示一个错误信息，警告一个文件丢失。

要只保存文件而不保存工程，请按照以下步骤执行：

1. 在“工程资源管理器”里选定此文件。
2. 从“文件”菜单，选取“保存 filename”。

合并文本

也能够从其它文件将现存的文本插入到一个代码模块中。在添加常数清单或者添加可能保存在文本文件中的代码段时，这个功能很有用。

要将文本文件插入代码，请按照以下步骤执行：

1. 从“工程”窗口，选定要插入代码的窗体或模块。
2. 选取“查看代码”按钮，将光标移动到代码编辑器中要插入代码的地方。
3. 从“编辑”菜单，选取“插入文件”。
4. 选定欲插入的文本文件名，选取“打开”。

注意 如果您使用 Visual Basic 以外的文本或代码编辑器编辑 Visual

Basic 代码，一定要注意不要改变属性 VB_PredeclaredId 的设置。特别注意，改变了这个属性可能会对 GlobalMultiUse 和 GlobalSingleUse 类造成严重问题。

一般情况下，不要手工编辑这些属性，因为这样做将使模块处于内部不一致的状态。

在工程中添加控件

工具箱里可用的控件集可被每个工程单独定义。任何给定的控件，在将其添加到工程的窗体之前，必须先位于工具箱里。经常在工具箱里出现的标准控件的基本集，将在第三章“窗体、控件和菜单”中描述。

在工程中添加 ActiveX 控件

将 ActiveX 控件和可插入对象添加到工具箱中之后就能将它们添加到工程中。

为了将控件添加到工程的工具箱中，请执行：

1. 从“工程”菜单选取“部件”。

“部件”对话框被显示，如图 4.3 所示。列于此对话框中的项目包含全部登记的 ActiveX 控件、可插入对象和 ActiveX 设计器。

2. 要向工具箱添加控件（具有 .ocx 文件扩展名）或可插入对象，选定控

件名左面的复选框。

为查看具有 .ocx 文件扩展名的控件，选定“控件”选项卡。为查看可插入对象，例如 Microsoft Excel 图表，选定“可插入对象”选项卡。

3. 选取“确定”，关闭“部件”对话框。现在，所选定的全部 ActiveX 控件将出现在工具箱中。

要向“部件”对话框添加 ActiveX 控件，选取“浏览”按钮，搜索其它目录寻找具有 .ocx 扩展名的文件。在可用控件清单中添加 ActiveX 控件时，Visual Basic 自动选定该复选框。

注意 每一个 ActiveX 控件有一个具有 .oca 扩展名的文件。该文件存储有高速缓存型库信息和该控件特有的其它数据。具有 .oca 扩展名的文件通常存储在与 ActiveX 控件相同的目录里，并可在需要时重建（文件大小和日期可以改变）。

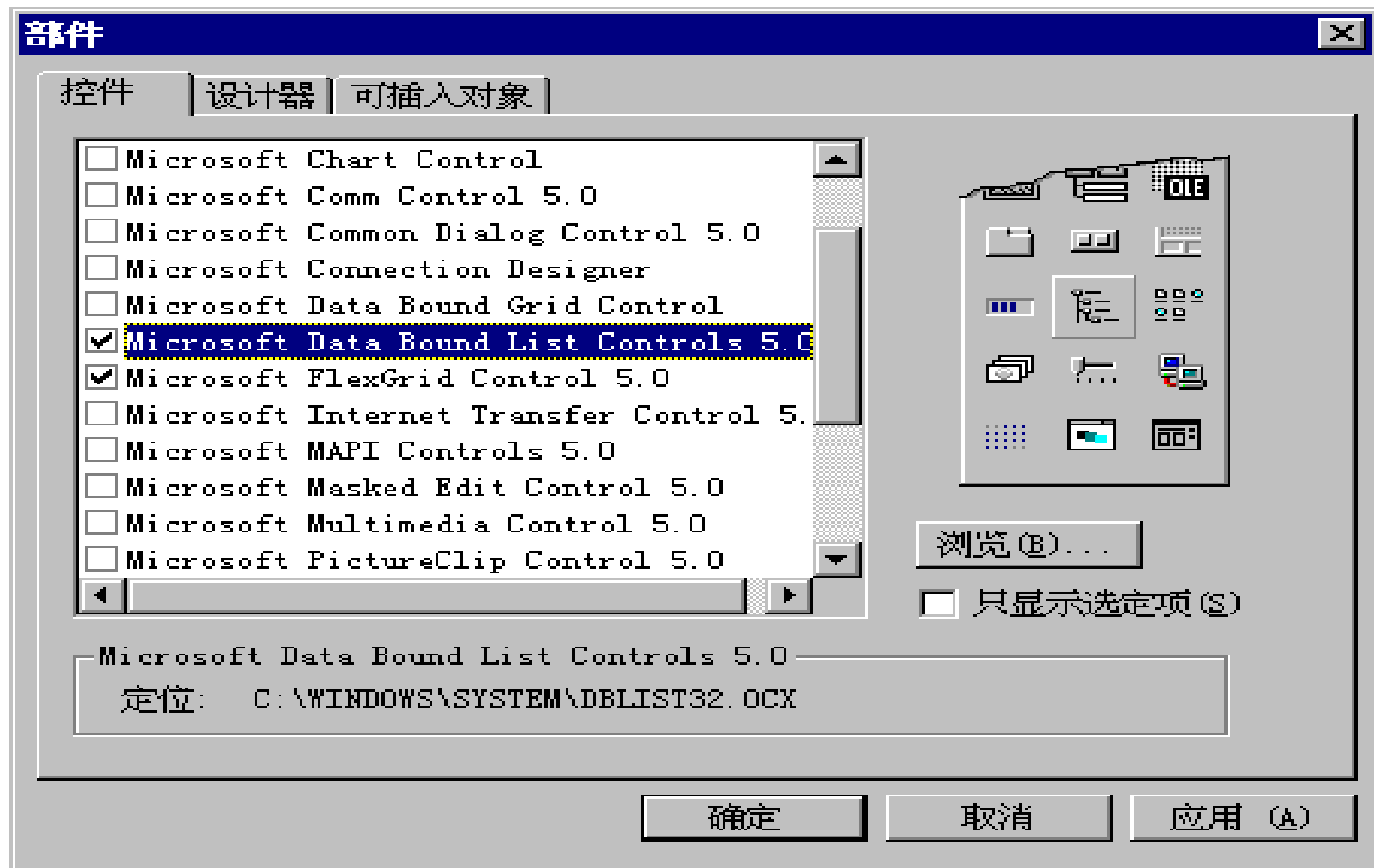


图 4.3 “部件”对话框

从工程中删除控件

要从工程中删除控件，请按照以下步骤执行：

1. 从“工程”菜单，选取“部件”。

显示“部件”对话框。

2. 清除要删除的每一个控件旁边的复选框。

这些控件的图标将从工具箱内删除。

注意 若某个控件的一个实例被这个工程的任何窗体所用，则不能从工具箱中删掉这个控件。

使用其它应用程序的对象

也可以使用来自其它应用程序的对象，例如包含在 Microsoft Excel 对象库中的对象，无论作为工具箱中的控件还是作为代码中的可编程对象都可使用。在工具箱中添加对象，请参阅本章前面的“在工程中添加控件”。

要使其它应用程序的对象在代码中可以使用，但不是作为控件，应设置对那个应用程序的对象库的引用。

要添加对其它应用程序的对象库的引用，请按照以下步骤执行：

1. 从“工程”菜单，选取“引用”。

“引用”对话框被显示，如图 4.4 所示。

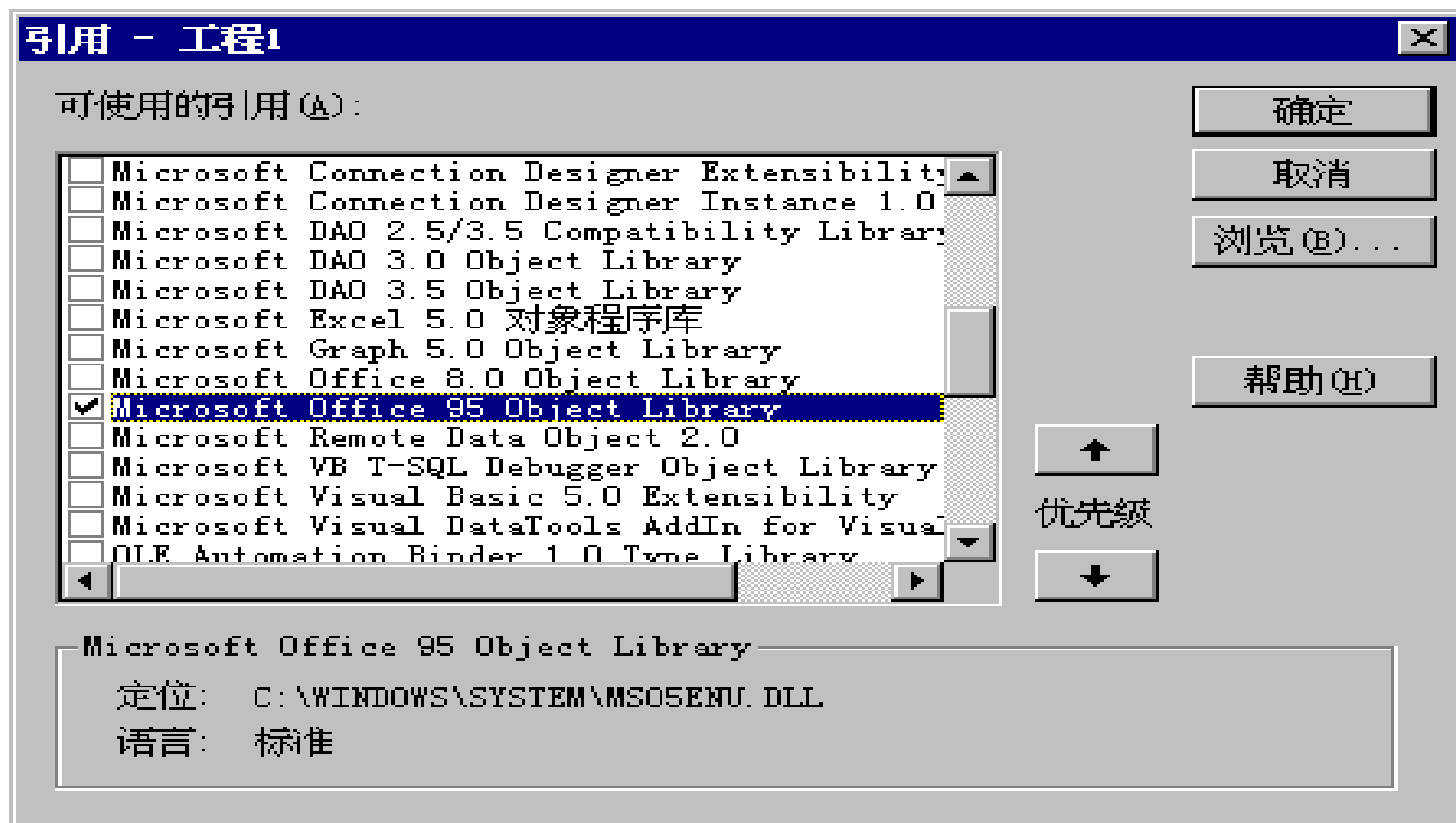


图 4.4 “引用”对话框

2. 选定欲添入工程的每个引用旁边的复选框。
要添加未列入“引用”对话框的应用程序的引用，选取“浏览”按钮，然后选定这个应用程序。
3. 选取“确定”，在工程中添加所选定的引用。

如果不是正在使用引用库中的任何对象，应当清除该引用的复选框，将 Visual Basic 必须解决的对象引用的数量减至最少，这样能减少工程编译占用的时间。

一旦设置了对所需对象库的引用，从“视图”菜单里选取“对象浏览器”，就可以在“对象浏览器”中找到一个特定的对象以及它的方法和属性。在代码中可以使用列在“对象浏览器”中的任何对象。

详细信息 关于“对象浏览器”的信息，请参阅第九章“用对象编程”中的“查找对象信息”。

使用资源文件

资源文件允许将某一应用程序使用的全部特定版本的正文和位图汇集于一处。它可以包含常数声明、图标、屏幕文本和其它可在本地化版本之间或在各种修订版之间或特殊配置间改变的资料。

要在工程中添加文件，请按照以下步骤执行：

1. 从“工程”菜单，选定“添加文件”。

“添加文件”对话框被显示。

2. 选定一个存在的资源文件 (.res) 并选取“打开”。

一个工程只能有一个资源文件；如果添加第二个具有 .res 扩展名的文件，会产生错误。

详细信息 关于资源文件内容的更多信息，请参阅第十六章“国际发行”。

制作和运行可执行文件

使用下面的过程，可以从 Visual Basic 制作可执行文件 (.exe)。

要在 Visual Basic 中制作可执行文件，请按照以下步骤执行：

1. 从“文件”菜单中选取“制作 `projectname.exe`”，这里 `projectname` 是工程的应用程序名。
2. 为了用新版本重写现有的可执行文件，键入文件名或浏览有关目录，选定一个现有文件名。
3. 单击“选项”按钮，可以在“工程属性”对话框里规定一些有关该执行文件特定版本的详细资料。
4. 若要修改工程的版本号，则要设置合适的“主版本号”、“次版本号”和“修订号”。选定“自动升级”，那么每一次运行该工程的“制作 `projectname.exe`”命令时，“修订号”都会自动增加。
5. 为了给应用程序指定新名，在“应用程序”下将新名键入“标题”框。如果要指定新图标，则从清单里选取一个。
6. 通过从列表框中选定主题并在文本框中输入信息，还可以输入“版本信息”框下的各种版本的版本专用注释（注释、公司名、商标和版权信息等等）。
7. 选取“确定”，关闭“工程属性”对话框，再在“制作 `appname.exe`”对话框中选取“确定”，编译和连接该可执行文件。

双击可执行文件的图标，像运行任何其它基于 Windows 的应用程序那样，

可运行这个可执行文件。

注意 如果想要有计划地编译一个工程，在 DOS 会话中从命令行建造可执行文件的方法是非常有用的。在批处理文件里键入：

```
Vb6 /make projectname[.vbp] [exename]
```

对于 projectname，请键入工程文件名。使用变量 exename 是为了给所得到的可执行文件重新命名。

条件编译

条件编译允许有选择地编译程序的某些部分。可以将程序的特殊性能纳入不同版本，例如对于不同语言版本中的某个应用程序，应改变日期和货币显示的过滤器。

详细信息 关于条件编译的详细信息，请参阅第八章“再论编程”中的“使用条件编译”。

设置工程选项

Visual Basic 允许通过设置一些属性来自定义每个工程。使用“工程属性”对话框，通过“工程”菜单上的“工程属性”命令可以设置。属性设置被保存在工程文件(.vbp)中。

下表列出能够设置的某些选项：

选项	描述
“启动对象”	运行时 Visual Basic 显示的第一个窗体，或者 Sub Main ()。
“工程名”	在代码中标识该工程。它不能包含句号 (.)、空格，或以非字母字符开头。对于公共类名，工程名和类名不能超过 37 个字符
“帮助文件”	与工程相关的 Help 文件名
“工程帮助上下文 ID”	在应用程序的对象库在“对象浏览器”中被选中的情况下，当选定了“?”按钮时供调用的特定“帮助”主题的上下文标识符
“工程描述”	工程的外部名。显示于“引用”和“对象浏览器”对话里

还有许多其它选项也是可使用的，包括编译、部件和多线程方面。若想访问某些更高级的选项，可以在联机帮助查找中得到更多信息。

详细信息 欲学习影响所有工程的环境选项设置方面的知识，请参阅第二章“用 Visual Basic 开发应用程序”。

使用向导和外接程序

Visual Basic 允许选择和管理外接程序，这是对 Visual Basic 的扩充。这些扩充增强了 Visual Basic 开发环境的能力，例如，特殊的源代码控制能力。Microsoft 和其它开发者创建了可以用于应用程序的外接程序。向导是一种外接程序，它可以简化某些任务，例如创建窗体。Visual Basic 包含几种向导。

要使外接程序出现在“外接程序管理器”对话框中，外接程序的开发者必

须保证它已正确安装。

使用外接程序管理器

使用外接程序管理器对工程可以添加或删除外接程序，从“外接程序”菜单可以对其进行访问。“外接程序管理器”对话框列出可用的外接程序。

要安装外接程序，请按照以下步骤执行：

1. 从“外接程序”菜单，选取“外接程序管理器”。
2. 从列表中突出显示一个外接程序并单击“加载行为”中想要的行为。要卸载一个外接程序或阻止其加载，请清除所有的“加载行为”框。
3. 当做完选定操作后，选取“确定”。

按照您“加载行为”的不同选择，Visual Basic 连接被选定的外接程序，断开与被清除的外接程序的连接。

Visual Basic 保存在编辑会话期间的外接程序选定。

注意 选择一个外接程序会在 Visual Basic “外接程序”菜单中增加菜单项。

使用向导

向导通过提供具体任务的帮助的方法，使 Visual Basic 的使用更加容易。例如，Visual Basic 所包含的应用程序向导通过提出一系列问题和选择的方法，为创建应用程序框架的工作提供帮助。它根据选择生成窗体和窗体后面的代码，你只需为特有的功能添加代码。

Visual Basic 的专业版和企业版包含一些其它向导，其中有创建用于数据库的窗体的数据窗体向导，和在互联网的应用程序里用于变换窗体的 ActiveX 文档向导。

使用外接程序管理器可安装或删除向导。一旦安装，它们将作为“外接程序”菜单上的选项出现。某些向导也能够以图标的形式出现在有关的对话框中；例如，可使用“新建工程”对话框中的“应用程序向导”图标访问“应用程序向导”。

要启动“应用程序向导”，请

- 从“外接程序”菜单，选取“应用程序向导”。
 - 或者 -
- 1. 从“文件”菜单，选取“新建工程”。
- 2. 选定“应用程序向导”图标。

第五章 编程基础

本章介绍 Visual Basic 语言的基本部件。在利用窗体和控件为应用程序建立界面之后就需要编写代码，定义应用程序的特性。与任何现代编程语言一样，Visual Basic 支持许多公共的编程结构和语言元素。

Visual Basic 是一种基于对象的编程语言。由于很少接触对象，许多程序员可能都感到对象不易接受。但不必过虑：不管是否意识到这一点，人们一生中的大部分时间其实都在和对象打交道。一旦了解一些基本概念就会知道，对象实际上使编程变得比以往任何时候都容易。

如果曾经使用过其它语言编写程序，那么，本章许多内容应已不陌生。尽管 Visual Basic 的大部分构造与其它语言类似，但其事件驱动特性却带来一些微妙差别。试以开放心态研究这些内容；一旦搞明白这些差别，就能更充分地利用它们。

对于编程新手来说，本章内容可以作为编写代码的基本构造模块概论。理解这些基本内容之后，就能用 Visual Basic 编写功能强大的应用程序。

目录

- Visual Basic 应用程序的结构

- 开始编码之前
- 代码编写机制
- 变量、常数和数据类型概述
- 过程概述
- 控件结构概述
- 使用对象

示例应用程序 Vcr.vbp

本章许多代码示例都取自 Vcr.vbp 示例应用程序，该示例应用程序列在 Samples 目录中。

Visual Basic 应用程序的结构

一个应用程序实际上无非是指挥计算机完成任务的指令集。应用程序的结构是组织指令的方法，也就是指令存放的位置和指令的执行顺序。

典型的 "hello world" 例子以及诸如此类的简单应用程序都具有简单结构。对于只有一行的代码来说，组织结构并不十分重要。但应用程序越复杂，对组织或结构的要求也就越明显。试想一下，如果允许应用程序以随机顺序去执行，那将会带来什么样的混乱。除了对应用程序的执行进行控制外，如何在应用程序中轻松查找特定指令，结构也起着很重要的作用。

由于 Visual Basic 应用程序是基于对象的，所以应用程序的代码结构就是该程序在屏幕上物理表示的模型。根据定义，对象包含数据和代码。在屏幕上看到的窗体代表属性，这些属性定义了窗体的外观和内在特性。应用程序中的每个窗体都有一个包含其代码的相关的窗体模块（文件扩展名为 .frm）。

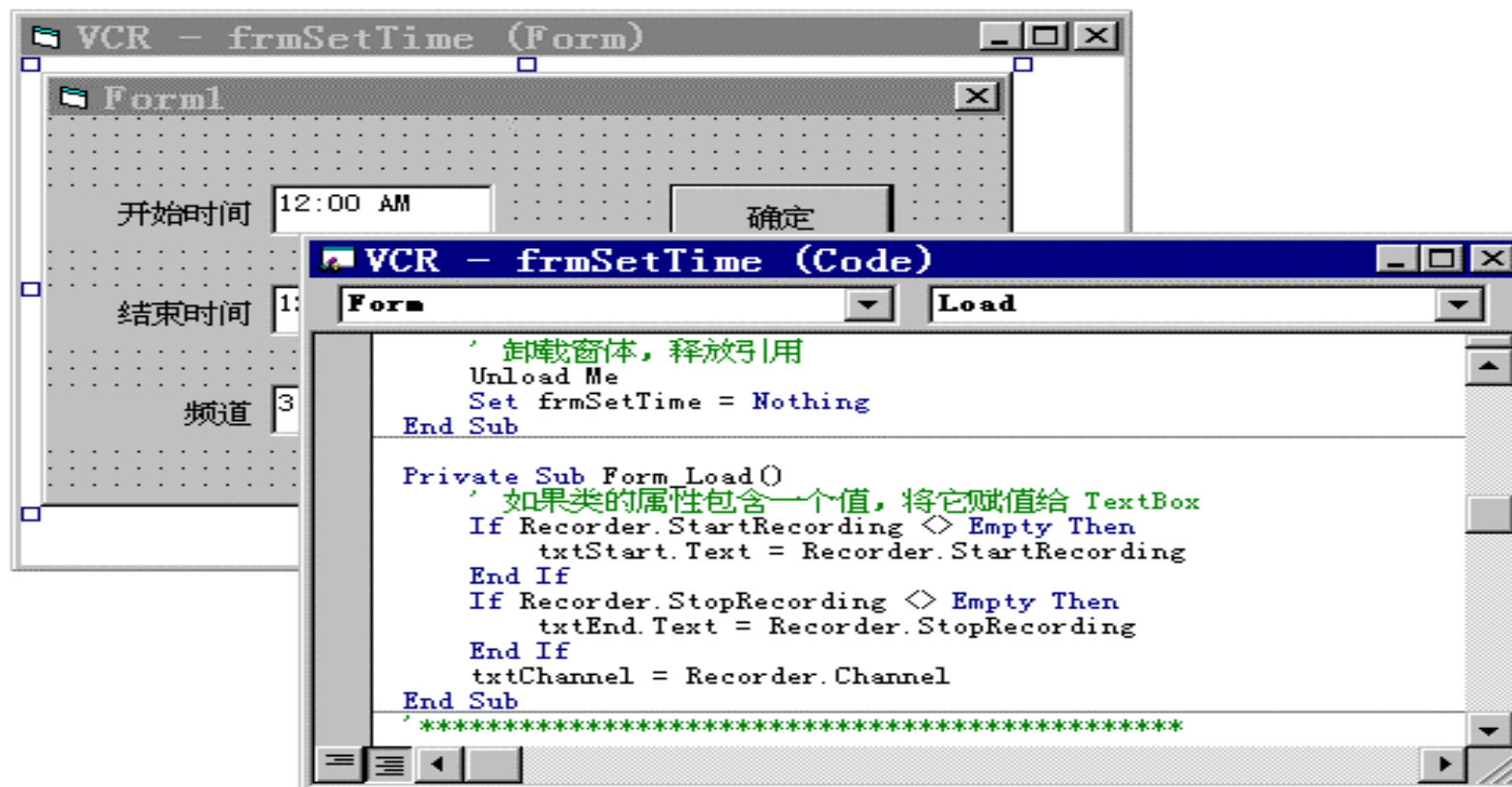


图 5.1 窗体及相关的窗体模块

每个窗体模块都包含事件过程，即代码部分，其中有为响应特定事件而执行的指令。窗体可包含控件。在窗体模块中，对窗体上的每个控件都有一个对应的事件过程集。除了事件过程，窗体模块还可包含通用过程，它对来自任何事件过程的调用都作出响应。

可将那些与特定窗体或控件无关的代码放入另一类型的模块——标准模块（文件扩展名为 `.BAS`）中。一个过程可能用来响应几个不同对象中的事件，应该将这个过程中放在标准模块中，而不应在每一个对象的事件过程中重复相同的代码。

用类模块（文件扩展名为 `.CLS`）创建对象，这些对象可被应用程序内的过程调用。标准模块只包含代码，而类模块既包含代码又包含数据，可视为没有物理表示的控件。

在第四章“工程的管理”中叙述了哪些部件可以添加到应用程序中，本章将说明如何将代码写到构成应用程序的各种部件中。按照缺省规定，工程包含唯一的窗体模块。可根据需要另行添加窗体、类和标准模块。第九章“用对象编程”将讨论类模块。

事件驱动应用程序的工作方式

事件是窗体或控件识别的动作。在响应事件时，事件驱动应用程序执行 Basic 代码。Visual Basic 的每一个窗体和控件都有一个预定义的事件集。如果其中有一个事件发生，而且，在关联的事件过程中存在代码，则 Visual Basic 调用该代码。

尽管 Visual Basic 中的对象自动识别预定义的事件集，但要判定它们是否响应具体事件以及如何响应具体事件则是编程的责任了。代码部分（即事件过程）与每个事件对应。想让控件响应事件时，就把代码写入这个事件的事件过程之中。

对象所识别的事件类型多种多样，但多数类型为大多数控件所共有。例如，大多数对象都能识别 Click 事件——如果单击窗体，则执行窗体的单击事件过程中的代码；如果单击命令按钮，则执行命令按钮的 Click 事件过程中的代码。每个情况中的实际代码几乎完全不一样。

这里是事件驱动应用程序中的典型事件序列：

1. 启动应用程序，装载和显示窗体。
2. 窗体（或窗体上的控件）接收事件。事件可由用户引发（例如键盘操作），可由系统引发（例如定时器事件），也可由代码间接引发（例如，当代码装载窗体时的 Load 事件）。
3. 如果在相应的事件过程中存在代码，就执行代码。
4. 应用程序等待下一次事件。

注意 许多事件伴随其它事件发生。例如，在 DblClick 事件发生时，MouseDown、MouseUp 和 Click 事件也会发生。

开始编码之前

在 Visual Basic 中创建应用程序的最重要的（又往往为人忽视的）部分是

设计阶段。需要为应用程序设计一个用户界面，这是显然的；但需要设计代码的结构就不那么显然了。构造应用程序的方法不同，可造成应用程序的性能及代码的可维护性、可使用性不同。

Visual Basic 应用程序的代码是用分层方式组织的。典型应用程序包括若干模块：应用程序中每个窗体的窗体模块、共享代码的可选标准模块以及可选的类模块。每个模块包含若干含有代码的过程：**event** 过程、**Sub** 子过程或 **Function** 过程，以及 **Property** 过程。

判定什么过程属于什么模块，这与创建的应用程序的类型有一定关系。因为 **Visual Basic** 是基于对象的，所以，利用应用程序代表的对象来考虑应用程序将是有益的。在这一章，示例应用程序 **Vcr.vbp** 的设计建立在盒式录像机和电视机组成的对象基础之上。**VCR** 应用程序包含两个窗体模块、一个标准模块和两个类模块。可用“对象浏览器”来检查工程的结构（图 5.2）。

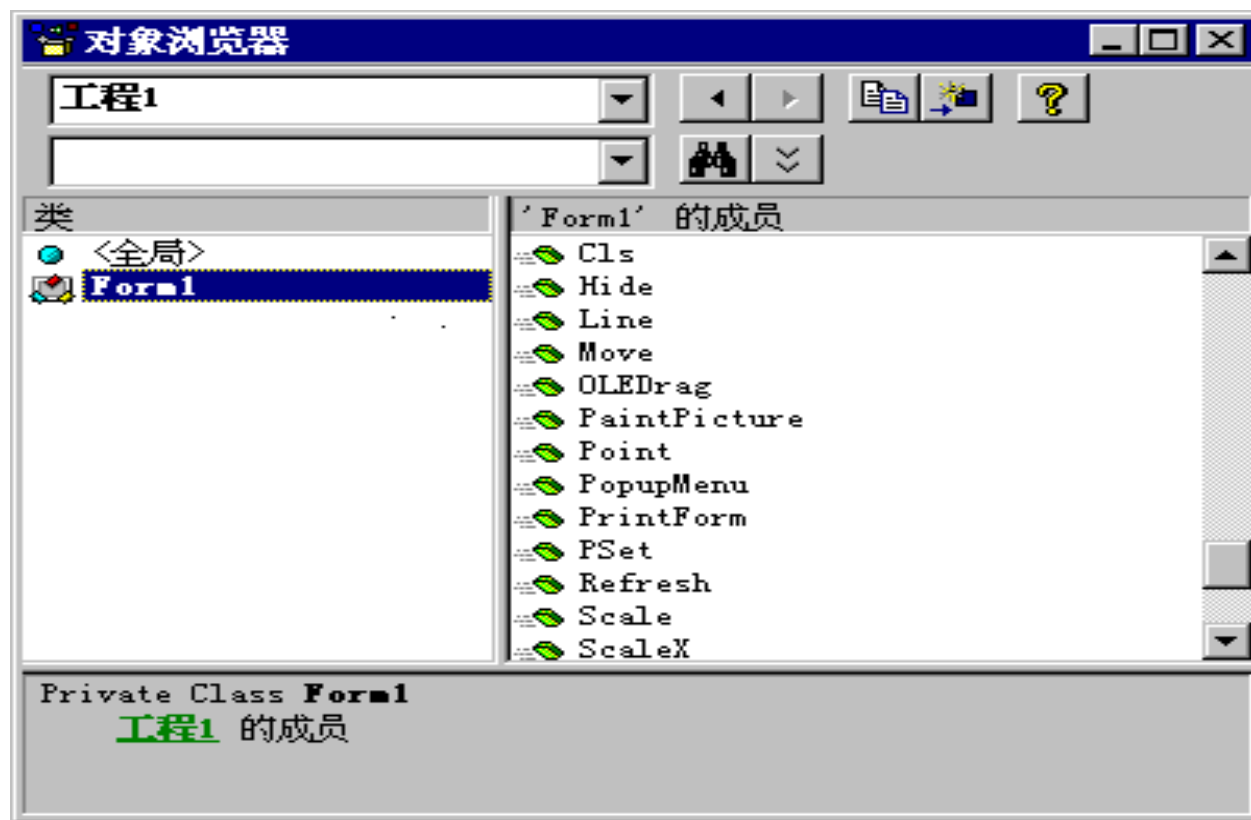


图 5.2 在“对象浏览器”中显示的 VCR 工程结构

VCR 应用程序的主窗体 (frmVCR) 是 VCR 与电视屏幕组合的可视图像 (图 5.3)。它包括几个对象，这些对象以现实世界中的图景为模型。一组命令按钮 (cmdPlay、cmdRecord 等等) 模仿的是操作 VCR 的按钮。软件 VCR 还包括时钟 (lblTime)、通道指示器 (lblChannel)、功能指示器 (shpPlay、shpRecord 等等) 以及显像管 (picTV)。Vcr.frm 窗体模块包括所有这些对象的事件过程。



图 5.3 VCR 应用程序的主窗体

在很多情况中都存在着多个对象共享的重复过程。例如，“按下”“放像”、“倒带”或“录像”按钮时就要求“暂停”和“停止”按钮有效。最好创

建一个共享的子过程，所有按钮都可调用它，而不要在每个按钮的 Click 事件过程中重复这段代码。如果将来要修改这些子程序，则只在一个地方修改就可进行全部修改。标准模块 Vcr.bas 包含各种共享过程。

VCR 的某些部分是不可视的，比如磁带的输送机置或录制电视节目的逻辑便是如此。同样，软件 VCR 的一些功能没有可视图像。这些部分和功能是作为两个类模块 Recorder.cls 和 Tape.cls 实现的。clsRecorder 模块包含启动“录像”进程的代码；而 clsTape 模块则包含控制“磁带”方向和速度的代码。这些模块中所定义类与窗体中的任何对象都没有直接关系。因为它们是独立的代码模块，所以不作任何修改就可轻轻松松重建一个录音程序。

除了设计代码结构之外，建立命名约定也是很重要的。按照缺省规定，Visual Basic 将工程的第一个窗体命名为 Form1，第二个窗体为 Form2，依此类推。如果应用程序中有几个窗体，则最好赋予它们有含义的名字，以免在编写或编辑代码时出现混淆情况。在附录 B“Visual Basic 编码约定”中列举了一些 Visual Basic 建议的命名约定。

在进一步了解对象和学习编写代码时，可参考 VCR 示例应用程序，查找有关各种编码技巧的例子。

代码编写机制

在着手编写代码之前，了解 Visual Basic 编写代码的机制是很重要的。和

任何编程语言一样，Visual Basic 有自身的组织、编辑和格式化代码规则。

代码模块

Visual Basic 的代码存储在模块中。模块有三种类型：窗体、标准和类。简单的应用程序可以只有一个窗体，应用程序的所有代码都驻留在窗体模块中。而当应用程序庞大复杂时，就要另加窗体。最终可能会发现在几个窗体中都有要执行的公共代码。因为不希望在两个窗体中重复代码，所以要创建一个独立模块，它包含实现公共代码的过程。独立模块应为标准模块。此后可以建立一个包含共享过程的模块库。

每个标准模块、类模块和窗体模块都可包含：

- 声明。可将常数、类型、变量和动态链接库 (DLL) 过程的声明放在窗体、类或标准模块的模块级。
- 过程。Sub、Function 或者 Property 过程包含可以作为单元来执行的代码片段。本章后面的“过程概述”一节中将讨论这些内容。

窗体模块

窗体模块（文件扩展名为 .FRM）是大多数 Visual Basic 应用程序的基础。窗体模块可以包含处理事件的过程、通用过程以及变量、常数、类型和外部过程的窗体级声明。如果要在文本编辑器中观察窗体模块，则还会看到窗体及其控件的描述，包括它们的属性设置值。写入窗体模块的代码是该

窗体所属的具体应用程序专用的；它也可以引用该应用程序内的其它窗体或对象。

标准模块

标准模块（文件扩展名为 `.BAS`）是应用程序内其它模块访问的过程和声明的容器。它们可以包含变量、常数、类型、外部过程和全局过程的全局（在整个应用程序范围内有效的）声明或模块级声明。写入标准模块的代码不必绑在特定的应用程序上；如果能够注意不用名称引用窗体和控件，则在许多不同的应用程序中可以重用标准模块。

类模块

在 Visual Basic 中类模块（文件扩展名为 `.CLS`）是面向对象编程的基础。可在类模块中编写代码建立新对象。这些新对象可以包含自定义的属性和方法。实际上，窗体正是这样一种类模块，在其上可安放控件，可显示窗体窗口。

详细信息 关于在类模块中编写代码的更详细的信息，请参阅第九章“面向对象编程”。

注意 Visual Basic 的专业版和企业版也包含 ActiveX 文档、ActiveX 设计器和用户控件。它们引入了具有不同文件扩展名的新模块类型。从编写代码的角度来看，这些模块应视同为窗体模块。

使用“代码编辑器”

Visual Basic “代码编辑器” 是一个窗口，大多数代码都在此窗口上编写。它像一个高度专门化的字处理软件，有许多便于编写 Visual Basic 代码的功能。图 5.4 所示为“代码编辑器”窗口。

因为要操作模块中的 Visual Basic 代码，所以要为每一个从“工程资源管理器”中选择的模块打开一个独立的“代码编辑器”窗口。在每个模块中，对于模块中所包含的每个对象，将模块中的代码再细分出与对象对应的独立部分。用“对象列表框”实现各部分间的切换。在窗体模块中，该列表包含一个通用段，一个属于窗体自身的段以及窗体所包含的每一控件的段。对于类模块，列表包括一个通用段和一个类段；对于标准模块，只有一个通用段被显示。

每一段代码都可包含几个用“过程列表框”访问的不同过程。对窗体或控件的每一个事件过程，窗体模块的过程列表都包含一个独立的段。例如，Label 控件的过程列表就包含 Change 事件段、Click 事件段和 DblClick 事件段等等。类模块只列举类本身的事件过程——初始化和终止。标准模块不列举任何事件过程，因为标准模块不支持事件。



图 5.4 “代码编辑器”窗口

模块通用段的过程列表只包含唯一一段——声明段，其中放置模块级的变量、常数和 DLL 声明。当在模块中添加子过程或函数过程时，那些过程被添加到声明段下方的“过程列表框”中。

代码的两种不同视图都可用于“代码编辑器”窗口。可以一次只查看一个过程，也可以查看模块中的所有过程，这些过程彼此之间用线隔开（如图 5.4 所示）。为了在两个视图之间进行切换，利用编辑器窗口左下角的“查看选择”按钮。

自动完成编码

Visual Basic 能自动填充语句、属性和参数，这些性能使编写代码更加方便。在输入代码时，编辑器列举适当的选择、语句或函数原型或值。通过“工具”菜单上的“选项”命令访问“选项”对话框，在“选项”对话框的“编辑器”选项卡上可用这样的选项，由它们决定是允许还是禁止各代码的设置值。

在代码中输入一控件名时，“自动列出成员特性”会亮出这个控件的下拉式属性表（图 5.5）。键入属性名的前几个字母，就会从表中选中该名字，按 Tab 键将完成这次输入。当不能确认给定的控件有什么样的属性时，这个选项是非常有帮助的。即使选择了禁止“自动列出成员特性”，仍可使用 CTRL + J 组合键得到这种性能。

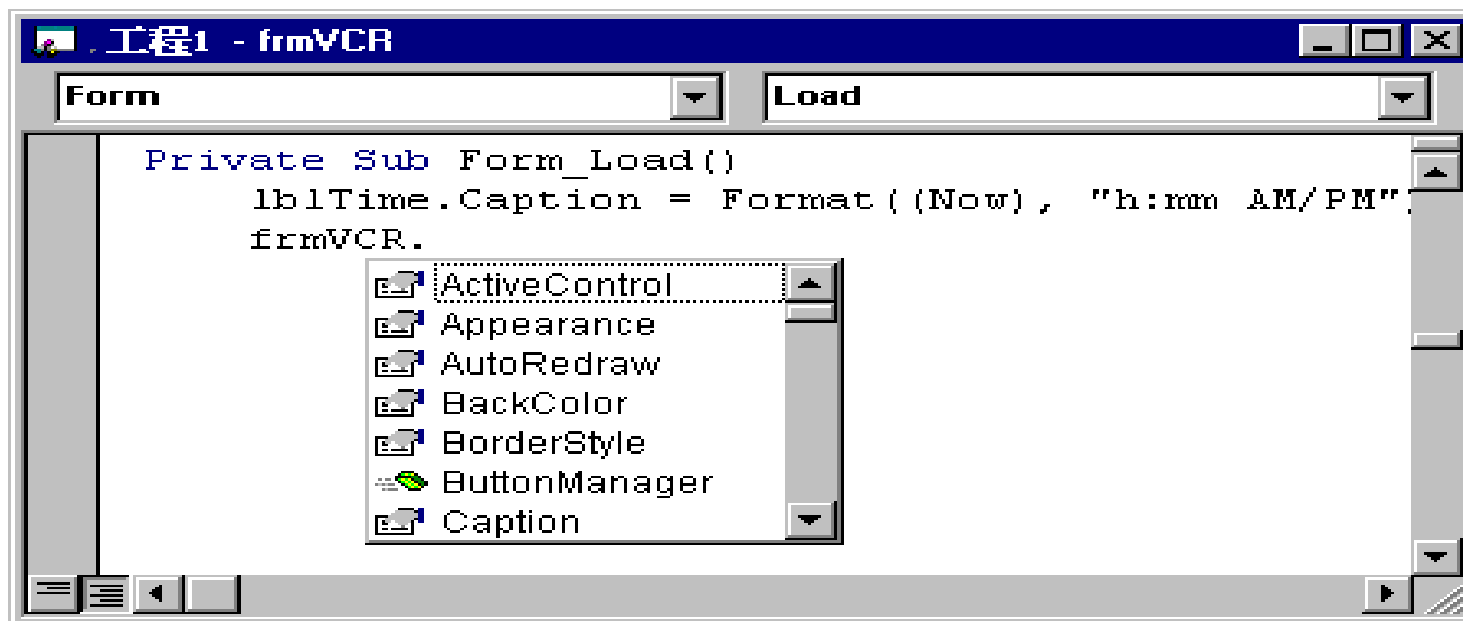


图 5.5 “自动列表成员功能”

“自动快速信息”功能显示语句和函数的语法(图 5.6)。当输入合法的 Visual Basic 语句或函数名之后，语法立即显示在当前行的下面，并用黑体字显示它的第一个参数。在输入第一个参数值之后，第二个参数又出现了，同样也是黑体字。“自动快速信息”也可以用 CTRL+I 组合键得到。

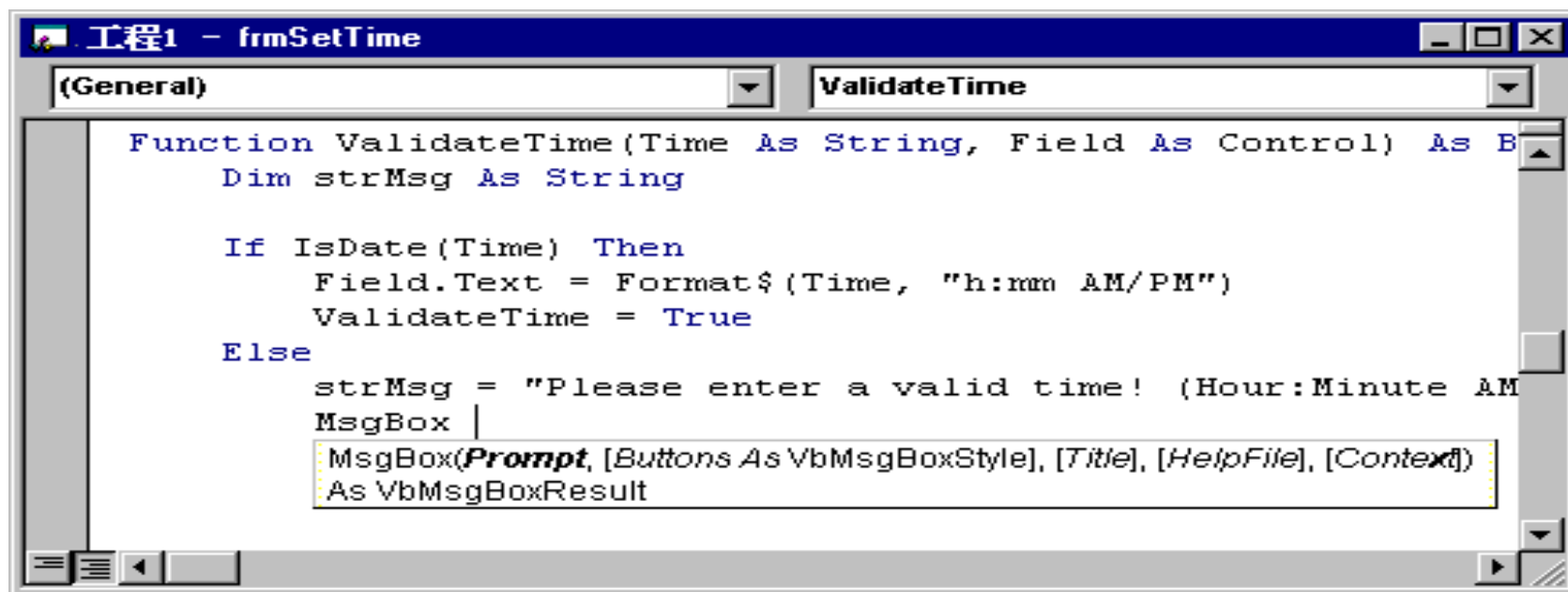


图 5.6 “自动快速信息”

书签

在代码编辑器中书签可用于标记代码的某些行，以便您以后可以很容易地返回这些行。书签开关的切换以及定位到已有书签的命令可以在编辑、书签菜单项或编辑工具栏中得到。

详细信息 关于如何使用组合键使用“代码编辑器”窗口的各种功能的更详细信息，请参阅“代码窗口的快捷键”。

编码基础

这一部分介绍代码编写机制，包括如何中断和合并代码行、如何添加注释、如何使用数字以及 Visual Basic 命名约定。

将单行语句分成多行

可以在“代码”窗口中用续行符（一个空格后面跟一个下划线）将长语句分成多行。由于使用续行符，无论在计算机上还是打印出来的代码都变得易读。下面用续行符（_）将代码分成若干行：

```
Data1.RecordSource = _  
"SELECT * FROM Titles, Publishers" _  
& "WHERE Publishers.PubId = Titles.PubID" _  
& "AND Publishers.State = 'CA'"
```

在同一行内，续行符后面不能加注释。至于在什么地方可以使用续行符，还是有某些限制的。

将多个语句合并到同一行上

通常，一行之中有一个 Visual Basic 语句，而没有语句终结符，但是也可以将两个或多个语句放在同一行，只是要用冒号 (:) 将它们分开。

```
Text1.Text = "Hello" : Red = 255 : Text1.BackColor = _
```

Red

但是，为了便于阅读代码，最好还是一行放一个语句。

详细信息 关于详细信息，请参阅附录 A “Visual Basic 的规范、限制和文件格式”。

在代码中添加注释

阅读本手册的例子时常会遇到注释符 (')。这个符号告诉 Visual Basic，忽略该符号后面的内容。这些内容是代码段中的注释，既是为了方便开发者，也是为了方便以后可能检查源代码的其他程序员。例如：

```
' 这是从屏幕左边
```

```
' 开始的注释。
```

```
Text1.Text = "Hi!"           '在文本框中放
```

```
                               '欢迎词。
```

注释可以和语句在同一行，并写在语句的后面，也可占据一整行。上述代码对这两种情况都作了示范。记住，不能在同一行上将注释接在续行符之后。

注意 您可以通过选中两行或多行代码并在“编辑”工具栏上通过选择“注释块”或“删除注释块”按钮来对该代码块添加或删除注释符号。

理解数字系统

在本文档中大多数的数值都是十进制的（基数为 10）。但有时用十六进制数（基数为 16）或八进制数（基数为 8）更方便。Visual Basic 用前缀 &H 表示十六进制数，而用 &O 表示八进制数。下表说明同一个数的十进制、八进制和十六进制表示。

十 进 制	八 进 制	十 六 进 制
9	&O 11	&H 9
15	&O 17	&H F
16	&O 20	&H 10
20	&O 24	&H 14
255	&O 377	&H FF

通常不必掌握十六进制或八进制数制，因为计算机可以用任何数制的数工作。但是，对某些任务来说，有的数系就比其它数系更合适，例如利用十六进制数设置屏幕和控件的颜色。

Visual Basic 的命名约定

在编写 Visual Basic 代码时，要声明和命名许多元素（Sub 和 Function 过程、变量、常数等等）。在 Visual Basic 代码中声明的过程、变量和常数的名字，必须遵循这些规则：

- 它们必须以字母开头。
- 它们不可以包含嵌入的句号或者类型声明字符（规定数据类型的特殊字符）。
- 它们不能超过 255 个字符。控件、窗体、类和模块的名字不能超过 40 个字符。
- 它们不能和受到限制的关键字同名。

受到限制的关键字是 Visual Basic 使用的词，是语言的组成部分。其中包括预定义语句（比如 If 和 Loop）、函数（比如 Len 和 Abs）和操作符（比如 Or 和 Mod）。

详细信息 关于关键字表的更详细信息，请参阅《Visual Basic 6.0 语言参考手册》。

窗体和控件可以和受到限制的关键字同名。例如，可以将某个控件命名为 Loop。但在代码中不能用通常的方法引用该控件，因为 Visual Basic 会认为 Loop 意味着关键字。例如，下面的代码就会出错。

```
Loop.Visible = True           ' 出错。
```

为了引用那些和受到限制的关键字同名的窗体或控件，就必须限定它们，或者将其用方括号 [] 括起来。例如，下面的代码就不会出错。

```
MyForm.Loop.Visible = True    '用窗体名
                                '将其限定。
```

```
[Loop].Visible = True        '方括号
                                '起了作用。
```

在引用窗体和控件时都可以用这种方式使用方括号，但在声明变量或定义

过程期间，当变量名或过程名与受到限制的关键字相同时，这种方式是不能使用的。方括号还可以用来强制 Visual Basic 接受其它类型库提供的名称，这些名称与受到限制的关键字冲突。

注意 因为键入方括号也是令人厌倦的事情，所以希望少用受到限制的关键字作窗体或控件名。但是，如果 Visual Basic 的新版本定义了与现有窗体或控件冲突的新关键字，那么在为使用新版本而更新代码时，可以使用这个技巧。

变量、常数和数据类型概述

在 Visual Basic 环境下进行计算时，常常需要临时存储数据。例如，可能想要计算几个值，将它们进行比较，并根据比较的结果对它们进行不同的操作。如果想要比较这些值，就要存储它们，但不必把它们存入属性。

像大多数编程语言那样，Visual Basic 使用变量来存储值。变量具有名字（用来引用该变量所含的值的名词）和数据类型（确定变量可以存储的数据的种类）。数组可以用来存储建立了索引的相关变量集。

常数也存储数值，顾名思义，在整个应用程序执行过程中，这些值都保持不变。常数的使用增加了代码的可读性，因为我们看到的是具有含义的名字而不是数字。Visual Basic 有许多内部常数，但也可以建立自定义常数。

在 Visual Basic 中，数据类型控制了数据的内部存储。按照缺省规定，Visual Basic 使用了变体数据类型。在不需要 Variant 提供的灵活性时，还有许多

其它可用的数据类型，它们可用来优化代码的速度和大小。

变 量

在 Visual Basic 中执行应用程序期间，用变量临时存储数值。变量具有名字（用来引用变量所包含的值的词）和数据类型（确定变量能够存储的数据的种类）。

可以把变量看作内存中存放未知值的所在处。例如，假定正在为水果铺编一个销售苹果的软件。在销售实际发生之前并不知道苹果的价格和销量。此时，可以设计两个变量来保存未知数，将它们命名为 `ApplePrice` 和 `ApplesSold`。每次运行程序时，用户就这两个变量提供具体值。为了计算总的销售额，并且将结果显示在名叫 `txtSales` 的文本框中，代码应该是这样的：

```
txtSales.txt = ApplePrice * ApplesSold
```

每次根据用户提供的数值，这个表达式返回不同的金额。由于有了变量，就可以设计一个算式，而不必事先知道实际的输入是多少。

在这个例子中，`ApplePrice` 的数据类型是货币，而 `ApplesSold` 的数据类型是整数。变量还可以表示许多其它数值，比如：文本数值、日期、各种数值类型，甚至对象也在此列。

存储和检索变量中的数据

用赋值语句进行计算，并将结果赋予变量：

ApplesSold = 10	'将值 10
	'传给变量。
ApplesSold = ApplesSold + 1	'变量值
	'增一。

注意，例子中的等号是赋值符，并不是等于操作符；它将数值 (10) 赋予变量 (ApplesSold)。

声明变量

声明变量就是事先将变量通知程序。要用 **Dim** 语句声明变量，**Dim** 语句提供了变量名：

Dim *variablename* [**As** *type*]

在过程内部用 **Dim** 语句声明的变量，只有在该过程执行时才存在。过程一结束，该变量的值也就消失了。此外，过程中的变量值对过程来说是局部的，也就是说，无法在一个过程中访问另一个过程中的变量。由于这些特点，在不同过程中就可使用相同的变量名，而不必担心有什么冲突和意想不到的变故。

变量名：

- 必须以字母开头。
- 不能包含嵌入的（英文）句号或者嵌入的类型声明字符。
- 不得超过 255 个字符。
- 在同一个范围内必须是唯一的。范围就是可以引用变量的变化域，如一个过程、一个窗体等等。

由于 Dim 语句中的可选的 As type 子句，可以定义被声明变量的数据类型或对象类型。数据类型定义了变量所存储信息的类型。String、Integer 和 Currency 都是数据类型的例子。变量也可以包含来自 Visual Basic 或其它应用程序的对象。Object、Form1 和 TextBox 都是 Visual Basic 对象类型或类的实例。

详细信息 关于对象的更详细的信息，请参阅第九章“用对象编程”和第十章“用部件编程”。在本章后面的“数据类型”部分将详细讨论数据类型。声明变量的其它方式：

- 不在过程内部，而在窗体、标准或类模块的声明段声明变量，这将使变量对模块中的所有过程有效。
- 用 Public 关键字声明变量，这将使变量在整个应用程序中有效。
- 用 Static 关键字声明一个局部变量，那么，即使过程结束，变量的值也仍然保留着。

隐式声明

在使用一个变量之前并不必先声明这个变量。例如，可以书写这样一个函

数，在其中就不必在使用变量 TempVal 之前先声明它：

```
Function SafeSqr (num)
    TempVal = Abs (num)
    SafeSqr = Sqr (TempVal)
End Function
```

Visual Basic 用这个名字自动创建一个变量，使用这个变量时，可以认为它就是显式声明的。虽然这种方法很方便，但是如果把变量名拼错了的话，会导致一个难以查找的错误。例如，假定写了这样一个函数：

```
Function SafeSqr (num)
    TempVal = Abs (num)
    SafeSqr = Sqr (TemVal)
End Function
```

乍看起来，这两段代码好像是一样的。但是因为在倒数第二行把 TempVal 变量名写错了，所以函数总是返回 0。当 Visual Basic 遇到新名字，它分辨不出这是意味着隐式声明了一个新变量呢，还是仅仅把一个现有变量名写错了，于是只好用这个名字再创建一个新变量。

显式声明

为了避免写错变量名引起的麻烦，可以规定，只要遇到一个未经明确声明就当成变量的名字，Visual Basic 都发出错误警告。
要显式声明变量，请

- 在类模块、窗体模块或标准模块的声明段中加入这个语句：

`Option Explicit`

-或-

在“工具”菜单中选取“选项”，单击“编辑器”选项卡，再复选“要求变量声明”选项。这样就在任何新模块中自动插入 `Option Explicit` 语句，但不会在已经建立起来的模块中自动插入；所以在工程内部，只能用手工方法向现有模块添加 `Option Explicit`。

如果对包含 `SafeSqr` 函数的窗体或标准模块执行该语句，那么 Visual Basic 将认定 `TempVal` 和 `TemVal` 都是未经声明变量，并为两者都发出错误信息。随后就可以显式声明 `TempVal`。

```
Function SafeSqr(num)
```

```
    Dim TempVal
```

```
    TempVal = Abs(num)
```

```
    SafeSqr = Sqr(TemVal)
```

```
End Function
```

因为 Visual Basic 对拼错了的 `TemVal` 显示错误信息，所以能够立刻明白是什么问题。由于 `Option Explicit` 语句有助于抓住这些类型的错误，所以最好在所有代码中使用它。

注意 `Option Explicit` 语句的作用范围仅限于语句所在模块，所以，对每个需要 Visual Basic 强制显式变量声明的窗体模块、标准模块及类模块，必须将 `Option Explicit` 语句放在这些模块的声明段中。如果选择“要求变量声明”，Visual Basic 会在后续的窗体模块、标准模块及类

模块中自动插入 Option Explicit，但是不会将它加入到现有代码中。必须在工程中通过手工将 Option Explicit 语句加到任何现有模块中。

理解变量的范围

变量的范围确定了能够知晓该变量存在的那部分代码。在一个过程内部声明变量时，只有过程内部的代码才能访问或改变那个变量的值；它有一个范围，对该过程来说是局部的。但是，有时需要使用具有更大范围的变量，例如这样一个变量，其值对于同一模块内的所有过程都有效，甚至对于整个应用程序的所有过程都有效。Visual Basic 允许在声明变量时指定它的范围。

指定变量的有效范围

一个变量在划定范围时被看作是过程级（局部）变量，还是模块级变量，这取决于声明该变量时采用的方式。

范围	专用	公用
过程级	变量对于这种过程是专用的，在该过程中出现了这些变量	不可使用。不能在一个过程中声明公用变量
模块级	变量对于这种过程是专用的，在该过程中出现了这些变量	变量可用于所有模块

过程内部使用的变量

过程级变量只有在声明它们的过程中才能被识别，它们又称为局部变量。用 `Dim` 或者 `Static` 关键字来声明它们。例如：

```
Dim intTemp As Integer
```

- 或者 -

```
Static intPermanent As Integer
```

在整个应用程序运行时，用 `Static` 声明的局部变量中的值一直存在，而用 `Dim` 声明的变量只在过程执行期间才存在。

对任何临时计算来说，局部变量是最佳选择。例如，可以建立十来个不同的过程，每个过程都包含称作 `intTemp` 的变量。只要每个 `intTemp` 都声明为局部变量，那么每个过程只识别它自己的 `intTemp` 版本。任何一个过程都能够改变它自己的局部的 `intTemp` 变量的值，而不会影响别的过程中的 `intTemp` 变量。

模块内部使用的变量

按照缺省规定，模块级变量对该模块的所有过程都可用，但对其它模块的代码不可用。可在模块顶部的声明段用 `Private` 关键字声明模块级变量，从而建立模块级变量。例如：

```
Private intTemp As Integer
```

在模块级，`Private` 和 `Dim` 之间没有什么区别，但 `Private` 更好些，因为很容易把它和 `Public` 区别开来，使代码更容易理解。

所有模块使用的变量

为了使模块级的变量在其它模块中也有效，用 `Public` 关键字声明变量。公用变量中的值可用于应用程序的所有过程。和所有模块级变量一样，也在模块顶部的声明段来声明公用变量。例如：

```
Public intTemp As Integer
```

注意 不能在过程中声明公用变量，只能在模块的声明段中声明公用变量。

详细信息 有关变量的其它信息，请参阅 “高级变量主题”。

高级变量主题

使用多个同名的变量

如果不同模块中的公用变量使用同一名字，则通过同时引用模块名和变量名就可以在代码中区分它们。例如，如果有一个在 `Form1` 和 `Module1` 中都声明了的公用 `Integer` 变量 `intX`，则把它们作为 `Module1.intX` 和 `Form1.intX` 来引用便得到正确值。

为了看清这是如何工作的，在一个新工程中插入两个标准模块，并在窗体

上画上三个命令按钮。

在第一个标准模块 `Module1` 之中声明一个变量 `intX`。`Test` 过程设置它的值：

```
Public intX As Integer      '声明 Module1 的 intX。
Sub Test ()
    '设置 Module1 的 intX 变量的值。
    intX = 1
End Sub
```

在第二个标准模块 `Module2` 中声明了第二个变量 `intX`，它有相同的名字。又是名为 `Test` 的过程设置它的值：

```
Public intX As Integer      '声明 Module2 的 intX。
Sub Test ()
    '设置 Module2 的 intX 变量的值。
    intX = 2
End Sub
```

在窗体模块中声明了第三个变量 `intX`。名为 `Test` 的过程又一次设置它的值。

```
Public intX As Integer      '声明了该窗体的 intX 变量。
Sub Test ()
    ' 设置 form 中的 intX 变量值。
    intX = 3
```



```
End Sub
```

在三个命令按钮的 Click 事件过程中，每一个都调用了相应的 Test 过程，并用 MsgBox 来显示这三个变量的值。

```
Private Sub Command1_Click ()
```

```
    Module1.Test                ' 调用 Module1 中的 Test。
```

```
    MsgBox Module1.intX         ' 显示 Module1 的 intX。
```

```
End Sub
```

```
Private Sub Command2_Click ()
```

```
    Module2.Test                ' 调用 Module2 中的 Test。
```

```
    MsgBox Module2.intX         ' 显示 Module2 的 intX。
```

```
End Sub
```

```
Private Sub Command3_Click ()
```

```
    Test                        ' 调用 Form1 中的 Test。
```

```
    MsgBox intX                 ' 显示 Form1 的 intX。
```

```
End Sub
```

运行应用程序，单击三个命令按钮中的每一个按钮。于是将看到三个公用变量被分别引用。注意在第三个命令按钮的 Click 事件过程中，在调用 Form1 的 Test 过程时不必指定 Form1.Test，在调用 Form1 的 Integer 变量的值时也不必指定 Form1.intX。如果多个过程或变量同名，则 Visual Basic 会取变化更受限制的值，在本例中就是 Form1 变量。

公用变量与局部变量的比较

在不同的范围内也可有同名的变量。例如，可有名为 Temp 的公用变量，然后在过程中声明名为 Temp 的局部变量。在过程内通过引用名字 Temp 来访问局部变量；而在过程外则通过引用名字 Temp 来访问公用变量。通过用模块名限定模块级变量就可在过程内访问这样的变量。

```
Public Temp As Integer
```

```
Sub Test ()
```

```
    Dim Temp As Integer
```

```
    Temp = 2
```

```
    MsgBox Form1.Temp
```

```
End Sub
```

```
' Temp 的值为 2。
```

```
' Form1.Temp 的值为 1。
```

```
Private Sub Form_Load ()
```

```
    Temp = 1
```

```
End Sub
```

```
' 将 Form1.Temp 的值设置成 1。
```

```
Private Sub Command1_Click ()
```

```
    Test
```

```
End Sub
```

一般说来，当变量名称相同而范围不同时，局限性大的变量总会用“阴影”遮住局限性不太大的变量（即优先访问局限性大的变量）。所以，如果还有

名为 Temp 的过程级变量，则它会用“阴影”遮住模块内部的公用变量 Temp。

阴影窗体属性和控件

由于阴影效应，窗体属性、控件、常数和过程皆被视为窗体模块中的模块级变量。窗体属性或控件的名称与模块级变量、常数、自定义类型或过程的名称相同是不合法的，因为它们的范围相同。

在窗体模块内，和窗体中控件同名的局部变量将遮住同名控件。因此必须引用窗体名称或 Me 关键字来限定控件，才能设置或者得到该控件的值或它的属性值。例如：

```
Private Sub Form_Click ()
```

```
Dim Text1 , BackColor
```

```
'假定该窗体有一个控件也叫做 Text1。
```

```
Text1 = "Variable"
```

```
'变量用“阴影”遮住控件。
```

```
Me.Text1 = "Control"
```

```
'要得到控件，必须用‘Me’限定。
```

```
Text1.Top = 0
```

```
'导致出错！
```

```
Me.Text1.Top = 0
```

```
'要得到控件，必须用‘Me’限定。
```

```
BackColor = 0
```

```
'变量用“阴影”遮住属性。
```

```
Me.BackColor = 0
```

```
'要得到窗体属性，必须用‘Me’限定。
```

```
End Sub
```

使用同名的变量和过程

专用模块级变量和公共模块级变量的名字也会和过程名冲突。模块中的变量不能和任何过程同名，也不能和模块中定义的类型同名。但可以和公用过程或其它模块中定义的类型或变量同名。在这种情况下，从别的模块访问这个变量时，就必须用模块名来限定。

虽然上面讨论阴影规则并不复杂，但是用阴影的方法可能会带来麻烦，而且会导致难以查找的错误。因此，对不同的变量使用不同的名称才是一种好的编程习惯。在窗体模块中应尽量使变量名和窗体中的控件名不一样。

静态变量

除范围之外，变量还有存活期，在这一期间变量能够保持它们的值。在应用程序的存活期内一直保持模块级变量和公用变量的值。但是，对于 `Dim` 声明的局部变量以及声明局部变量的过程，仅当过程在执行时这些局部变量才存在。通常，当一个过程执行完毕，它的局部变量的值就已经不存在，而且变量所占据的内存也被释放。当下一次执行该过程时，它的所有局部变量将重新初始化。

但可将局部变量定义成静态的，从而保留变量的值。在过程内部用 `Static` 关键字声明一个或多个变量，其用法和 `Dim` 语句完全一样：

`Static Depth`

例如，下面的函数将存储在静态变量 `Accumulate` 中的以前的运营总值与一个新值相加，以计算运营总值。

```
Function RunningTotal (num)
    Static ApplesSold
    ApplesSold = ApplesSold + num
    RunningTotal = ApplesSold
End Function
```

如果用 `Dim` 而不用 `Static` 声明 `ApplesSold`，则以前的累计值不会通过调用函数保留下来，函数只会简单地返回调用它的那个相同值。

在模块的声明段声明 `ApplesSold`，并使它成为模块级变量，由此也会收到同样效果。但是，这种方法一旦改变变量的范围，过程就不再对变量排他性存取。由于其它过程也可以访问和改变变量的值，所以运营总值也许不可靠，代码将更难于维护。

声明所有的局部变量为静态变量

为了使过程中所有的局部变量为静态变量，可在过程头的起始处加上 `Static` 关键字。例如：

```
Static Function RunningTotal (num)
```

这就使过程中的所有局部变量都变为静态，无论它们是用 `Static`、`Dim` 或 `Private` 声明的还是隐式声明的。可以将 `Static` 放在任何 `Sub` 或 `Function` 过程头的前面，包括事件过程和声明为 `Private` 的过程。

常数

经常会发现代码包含一些常数值，它们一次又一次地反复出现。还可发现，代码要用到很难记住的数字，而那些数字没有明确意义。

在这些情况下，可用常数大幅度地改进代码的可读性和可维护性。常数是有意义的名字，取代永远不变的数值或字符串。尽管常数有点像变量，但不能像对变量那样修改常数，也不能对常数赋以新值。常数有两种来源：

- 内部的或系统定义的常数是应用程序和控件提供的。在“对象浏览器”中的 Visual Basic (VB)、和 Visual Basic for applications (VBA) 对象库中列举了 Visual Basic 的常数。其它提供对象库的应用程序，如 Microsoft Excel 和 Microsoft Project，也提供了常数列表，这些常数可与应用程序的对象、方法和属性一起使用。在每个 ActiveX 控件的对象库中也定义了常数。关于使用“对象浏览器”的详细信息，请参阅第九章“用对象编程”。
- 符号的或用户定义的常数是使用 Const 语句来声明的。下节“创建属于自己的常数”中将说明用户定义的常数。

在 Visual Basic 中，常数名采用大小写混合的格式，其前缀表示定义常数的对象库名。来自 Visual Basic 和 Visual Basic for applications 对象库的常数以 "vb" 开头，例如 vbTileHorizontal。

设计前缀时应尽力防止发生意外冲突，不能出现常数名称相同但表示不同数值的情况。即使使用了前缀，两个对象库也仍可能包含表示不同值的相

同常数。在这种情况下，引用哪个常数取决于哪个对象库具有更高的优先级。关于改变对象库优先级的信息，请参阅“引用对话框”部分。为了绝对确保不发生常数名字冲突，可用以下语法来限定对常数的引用：

[libname.][modulename.]constname

Libname 通常是控件或库的类名。**Modulename** 是定义常数的模块的名字。**Constname** 是常数名。在对象库中定义了每个元素，并能在“对象浏览器”中查看元素。

创建属于自己的常数

声明常数的语法是：

[Public|Private] Const constantname[As type] = expression

参数 **constantname** 是有效的符号名（其规则与建立变量名的规则一样），**expression** 由数值常数或字符串常数以及运算符组成；但在 **expression** 中不能使用函数调用。

Const 语句可以表示数量、日期和时间：

```
Const conPi = 3.14159265358979
```

```
Public Const conMaxPlanets As Integer = 9
```

```
Const conReleaseDate = #1/1/95#
```

也可用 **Const** 语句定义字符串常数：

```
Public Const conVersion = "07.10.A"
```

```
Const conCodeName = "Enigma"
```

如果用逗号进行分隔，则在一行中可放置多个常数声明：

```
Public Const conPi = 3.14, conMaxPlanets = 9, _  
conWorldPop = 6E+09
```

等号（=）右边的表达式往往是数字或文字串，但也可以是其结果为数或字符串的表达式（尽管表达式不能包含函数调用）。甚至可用先前定义过的常数定义新常数。

```
Const conPi2 = conPi * 2
```

一旦已定义常数，就可将其放置在代码中，使代码更可读。例如：

```
Static SolarSystem (1 To conMaxPlanets)  
If numPeople > conWorldPop Then Exit Sub
```

设定用户自定义常数的范围

和变量声明一样，Const 语句也有范围，也使用相同的规则：

- 为创建仅存在于过程中的常数，请在这个过程内部声明常数。
- 为创建一常数，它对模块中所有过程都有效，但对模块之外任何代码都无效，请在模块的声明段中声明常数。
- 为创建在整个应用程序中有效的常数，请在标准模块的声明段中进行声明，并在 Const 前面放置 Public 关键字。在窗体模块或类模块中不能声明 Public 常数。

详细信息 关于范围的更详细信息，请参阅本章前面的“理解变量的范围”一节。

避免循环引用

由于常数可以用其它常数定义，因此必须小心，在两个以上常数之间不要出现循环或循环引用。当程序中有两个以上的公用常数，而且每个公用常数都用另一个去定义时就会出现循环。

例如：

'在 Module1 中：

```
Public Const conA = conB * 2      '在整个应用程序  
                                   '中有效。
```

'在 Module 2：

```
Public Const conB = conA / 2      '在整个应用程序  
                                   '中有效。
```

如果出现循环，在试图运行此应用程序时，Visual Basic 就会产生错误信息。不解决循环引用就不能运行程序。为避免出现循环，可将公共常数限制在单一模块内，或最多只存在于少数几个模块内。

数据类型

变量是用来存储值的所在处，它有名字和数据类型。变量的数据类型决定了如何将代表这些值的位存储到计算机的内存中。在声明变量时也可指定它的数据类型。所有变量都具有数据类型，以决定能够存储哪种数据。

根据缺省规定，如果在声明中没有说明数据类型，则令变量的数据类型为 Variant。Variant 数据类型很像一条变色龙，它可在不同场合代表不同数据类型。当指定变量为 Variant 变量时，不必在数据类型之间进行转换，Visual Basic 会自动完成各种必要的转换。

但是，如果知道变量确实总是存储特定类型的数据，并且还声明了这种特定类型的变量，则 Visual Basic 会以更高的效率处理这个数据。例如，存储人名的变量最好表示成 String 数据类型，因为名字总是由字符组成。

除变量外，数据类型也用于其它场合。在给属性赋值时，这个值就有数据类型；函数的参数也有数据类型。事实上，在 Visual Basic 中，凡是与数据有关的东西就与数据类型有关。

你也可声明任何基本类型的数组。

详细信息 关于更详细信息，请参阅本章后面的“数组”部分。在第十五章“性能和兼容性的设计”中还将讨论通过选择数据类型来提高应用程序的性能。

声明带数据类型的变量

在使用非 Variant 变量之前，必须使用 Private、Public、Dim 或 Static 语句将变量声明为 As type。例如，下列语句分别声明了 Integer、Double、String 和 Currency 类型的变量：

```
Private I As Integer
```

```
Dim Amt As Double
```

```
Static YourName As String
```

```
Public BillsPaid As Currency
```

一个声明语句可将多个声明组合起来，请看下列语句：

```
Private I As Integer,Amt As Double
```

```
Private YourName As String,BillsPaid As Currency
```

```
Private Test,Amount,J As Integer
```

注意 如果不提供数据类型，则指定变量为缺省类型。在上例中，变量 `Test` 和 `Amount` 是 `Variant` 数据类型。如果由于其它编程语言的经验使你感到同一个声明语句中的所有变量都有相同的指定数据类型（此例中为 `Integer`），那么，此处的结果可能会令人感到新鲜。

Numeric 数据类型

Visual Basic 支持几种 `Numeric` 数据类型：`Integer`（整型）、`Long`（长整型）、`Single`（单精度浮点型）、`Double`（双精度浮点型）和 `Currency`（货币型）。

与 `Variant` 类型相比，`Numeric` 类型占用的存储空间通常要少。

如果知道变量总是存放整数（如 12）而不是带小数点的数字（如 3.57），就应当将它声明为 `Integer` 类型或 `Long` 类型。整数的运算速度较快，而且比其它数据类型占据的内存要少。在 `For...Next` 循环内作为计数器变量使用时，整数类型尤为有用。

详细信息 关于控制结构的详细资料，请参阅本章后面的“控制结构概述”。如果变量包含小数，则可将它们声明为 `Single`、`Double` 或 `Currency` 变量。

Currency 数据类型支持小数点右面 4 位和小数点左面 15 位；它是一个精确的定点数据类型，适用于货币计算。浮点 (Single 和 Double) 数比 Currency 的有效范围大得多，但有可能产生小的进位误差。

注意 浮点数值可表示为 mmmEeee 或 mmmDeee，其中 mmm 是尾数，而 eee 是指数（以 10 为底的幂）。Single 数据类型的最大正数值为 3.402823E+38，或 3.4 乘以 10 的 38 次方；Double 数据类型的最大正数值是 1.79769313486232D+308 或 1.8 乘以 10 的 308 次方。用 D 将数值文字中的尾数部分和指数部分隔开，就会导致将该值作为 Double 数据类型来处理。同样，用这种方式使用 E，也会导致将该值作为 Single 数据类型来处理。

Byte 数据类型

如果变量包含二进制数，则将它声明为 Byte 数据类型的数组（本章后面的“数组”中将讨论数组）。在转换格式期间用 Byte 变量存储二进制数据就可保留数据。当 String 变量在 ANSI 和 Unicode 格式间进行转换时，变量中的任何二进制数据都会遭到破坏。在下列任何一种情况下，Visual Basic 都会自动在 ANSI 和 Unicode 之间进行转换：

- 读文件时
- 写文件时
- 调用 DLL 时
- 调用对象的方法和属性时

除一元减法外，所有可对整数进行操作的运算符均可操作 **Byte** 数据类型。因为 **Byte** 是从 0-255 的无符号类型，所以不能表示负数。因此，在进行一元减法运算时，**Visual Basic** 首先将 **Byte** 转换为符号整数。所有数值变量都可相互赋值，也可对 **Variant** 类型变量赋值。在将浮点数赋予整数之前，**Visual Basic** 要将浮点数的小数部分四舍五入，而不是将小数部分去掉。

详细信息 关于 **Unicode** 和 **ANSI** 转换的详细信息，请参阅第十六章“国际化”。

String 数据类型

如果变量总是包含字符串而从不包含数值，就可将其声明为 **String** 类型。

```
Private S As String
```

然后将字符串赋予这个变量，并用字符串函数对它进行操作。

```
S = "Database"
```

```
S = Left(S,4)
```

按照缺省规定，**String** 变量或参数是一个可变长度的字符串。随着对字符串赋予新数据，它的长度可增可减。也可以声明字符串具有固定长度。可用以下语法声明一个定长字符串：

```
String * size
```

例如，为了声明一个长度为 50 字符的字符串，可用下列语句：

```
Dim EmpName As String * 50
```

如果赋予字符串的字符少于 50 个，则用空格将 EmpName 的不足部分填满。如果赋予字符串的长度太长，已不能成为定长字符串，则 Visual Basic 会直接截去超出部分的字符。

因为定长字符串用空格填充尾部多余的空间，所以在处理定长字符串时可发现，删除空格的 Trim 和 RTrim 函数是很有用的。

可将标准模块中的定长字符串声明为 Public 或 Private。在窗体和类模块中，必须将定长字符串声明为 Private。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的“Ltrim、RTrim 和 Trim 函数”。

交换字符串和数字

如果字符串表示数值，则可将字符串赋予数值变量。也可将数值赋予字符串变量。例如，将命令按钮、文本框和列表框放置在窗体中。在命令按钮的 Click 事件中输入下列代码。运行应用程序并单击命令按钮。

```
Private Sub Command1_Click ()
```

```
    Dim intX As Integer
```

```
    Dim strY As String
```

```
    strY = "100.23"
```

```
    intX = strY
```

```
    List1.AddItem Cos (strY)
```

'将字符串传递给数值变量。

'将字符串中数值的余弦值。

```
strY = Cos (strY)
```

```
Text1.Text = strY
```

```
End Sub
```

'添加到列表框中。

'将余弦值传递给字符串变量。

'在文本框中显示字符串。

Visual Basic 会自动强制变量为适当的数据类型。在转换字符串和数值时要小心；如果传送字符串中的值不是数值，则在运行时会出错。

Boolean 数据类型

若变量的值只是“true/false”、“yes/no”、“on/off”信息，则可将它声明为 Boolean 类型。Boolean 的缺省值为 False。在下面的例子中，blnRunning 是 Boolean 变量，存储简单的 yes/no 设置。

```
Dim blnRunning As Boolean
```

```
'查看磁带是否在转。
```

```
If Recorder.Direction = 1 Then
```

```
    blnRunning = True
```

```
End if
```

Date 数据类型

Date 和 Time 值既可包含在特定的 Date 数据类型中，又可包含在 Variant 变量中。一般的 Date 特性适用于这两种类型。

详细信息 请参阅《联机手册》的“高级变量主题”中的“在 Variant 变量中存储 Date/Time 值”部分。

当其它数值数据类型转换为 Date 时，小数点左边的值表示 Date 信息，小数点右边的值则代表 Time。午夜为 0，正午为 0.5。负数表示公元 1899 年 12 月 31 日之前的 Date。

Object 数据类型

Object 变量作为 32 位（4 个字节）地址来存储，该地址可引用应用程序中或某些其它应用程序中的对象。可以随后（用 Set 语句）指定一个被声明为 Object 的变量去引用应用程序所识别的任何实际对象。

```
Dim objDb As Object
```

```
Set objDb = OpenDatabase("c:\Vb5\Biblio.mdb")
```

在声明对象变量时，请试用特定的类，而不用一般的 Object（例如用 TextBox 而不用 Control，或者像上面的例子那样，用 Database 取代 Object）。运行应用程序之前，Visual Basic 可以决定引用特定类型对象的属性和方法。因此，应用程序在运行时速度会更快。在“对象浏览器”中列举了特定的类。

当使用其它应用程序的对象，并在“对象浏览器”中的“类”列表中列举对象时，不要用 Variant 或一般的 Object，而应声明对象。这样可确保 Visual Basic 能够识别引用的特定类型对象，在运行时解决引用问题。

详细信息 关于创建和指定对象以及对象变量的更详细信息，请参阅本章

后面的“创建对象”部分。

转换数据类型

Visual Basic 提供了几种转换函数，用来将值转换成特定数据类型。例如，用 CCur 函数将值转换成 Currency 类型：

```
PayPerWeek = CCur (hours * hourlyPay)
```

转换函数	将表达式转换成
Cbool	Boolean
Cbyte	Byte
Ccur	Currency
Cdate	Date
Cdbl	Double
Cint	Integer
CLng	Long
CSng	Single
CStr	String
Cvar	Variant
CVErr	Error

注意 对目标数据类型，传递到转换函数的值必须是有效的，否则会发生错误。例如，如果想把 Long 型数转换成 Integer 型数，那么，Long 型数必须在 Integer 数据类型的有效范围之内。

详细信息 在联机帮助中查找指定的转换函数。

Variant 数据类型

Variant 变量能够存储所有系统定义类型的数据。如果把它们赋予 Variant 变量，则不必在这些数据的类型间进行转换；Visual Basic 会自动完成任何必要的转换。例如：

```
Dim SomeValue          '缺省为 Variant。
```

```
SomeValue = "17"       'SomeValue 包含 "17"（双字符的串）。
```

```
SomeValue = SomeValue - 15      '现在， SomeValue 包含数值 2。
```

```
SomeValue = "U" & SomeValue    '现在， SomeValue 包含 "U2"（双字符的串）。
```

不必过多关注 Variant 变量中数据的类型就可对 Variant 变量进行操作，但须避免落入陷阱。

- 如果对 Variant 变量进行数学运算或函数运算，则 Variant 必包含某个数。更详细的信息，请参阅《联机手册》中“高级变量主题”中的“变量中存储的 Numeric 值”部分。
- 如果正在连接两个字符串，则用 " & " 操作符而不用 " + " 操作符。更详细的信息，请参阅《联机手册》中“高级变量主题”中的“Variant 变量中存储的 Strings”部分。

除了可以像其它标准数据类型一样操作外，Variants 还包含三种特定值：Empty，Null 和 Error。

Empty 值

有时需要知道是否已将一个值赋予所创建的变量。在赋值之前，Variant 变量具有值 Empty。值 Empty 是异于 0、零长度字符串 ("") 或 Null 值的特定值。可用 IsEmpty 函数测试 Empty 值：

```
If IsEmpty (Z) Then Z = 0
```

当 Variant 变量包含 Empty 值时，可在表达式中使用它；将其作为 0 或零长度字符串来处理，这要根据表达式来定。

只要将任何值（包括 0、零长度字符串或 Null）赋予 Variant 变量，Empty 值就会消失。而将关键字 Empty 赋予 Variant 变量，就可将 Variant 变量恢复为 Empty。

Null 值

Variant 数据类型还可包含一特定值：Null。Null 通常用于数据库应用程序，表示未知数据或丢失的数据。由于在数据库中使用 Null 方法，Null 具有某些唯一的特性：

- 对包含 Null 的表达式，计算结果总是 Null。于是说 Null 通过表达式“传播”；如果表达式的部分之值为 Null，那么整个表达式的值也为 Null。
- 将 Null 值、含 Null 的 Variant 变量或计算结果为 Null 的表达式作为参数传递给大多数函数，将会使函数返回 Null。

- Null 值经由返回 Variant 数据类型的内在函数传播。也可用 Null 关键字指定 Null 值。

```
Z = Null
```

也可用 IsNull 函数测试 Variant 变量是否包含 Null 值。

```
If IsNull (X) And IsNull (Y) Then
```

```
    Z = Null
```

```
Else
```

```
    Z = 0
```

```
End If
```

如果将 Null 值赋予 Variant 以外的任何其它类型变量，则将出现可以捕获的错误。而将 Null 值赋予 Variant 则不会发生错误，Null 将通过包含 Variant 变量的表达式传播（尽管 Null 并不通过某些函数来传播）。可以从任何具有 Variant 返回值的函数过程返回 Null。

除非明确将 Null 赋予变量，否则变量不会设置成 Null 值，所以，如果不在应用程序中使用 Null，就不必书写测试 Null 和处理 Null 的程序。

详细信息 关于如何在表达式中使用 Null 的详细信息，请参阅语言参考的“Null”部分。

Error 值

在 Variant 中，Error 是特定值，指出已发生的过程中的错误状态。但是，与其它类型错误不同，这里并未发生正常的应用程序级的错误处理。因此，

程序员或应用程序本身可根据 `Error` 值进行取舍。利用 `CVErr` 函数将实数转换成错误值就可建立 `Error` 值。

详细信息 关于如何在表达式中使用 `Error` 值，请参阅《Visual Basic 6.0 语言参考手册》中的“`CVErr` 函数”部分。关于错误处理的信息，请参阅第十三章“调试代码和处理错误”。有关 `Variant` 数据类型的更详细信息，请参阅《联机手册》中的“高级变量主题”。

高级变量主题

数值在变量中的内部表示

`Variant` 变量维护它们所存储的数值的内部表示。这个表示将决定 `Visual Basic` 在执行比较操作和其它操作时如何处理这些值。将值赋予 `Variant` 变量时，`Visual Basic` 会用最紧凑的表示正确记录这个值。以后的操作可能使 `Visual Basic` 改变这个用于特定变量的表示（`Variant` 变量不是无类型变量，而是能够随意改变类型的变量）。这些内部表示与本章前面在“数据类型”中讨论的数据类型相一致。

注意 `Variant` 总保持 16 位，无论其中存储了什么数据。对象、字符串和数组，在物理上并非存于 `Variant` 之中；在这些情况下，用四个字节的 `Variant` 来保持对象引用、或者字符串或数组的指针。而真正的数据存在其它地方。

很多时候，不必了解 Visual Basic 为特殊变量使用哪一种变量类型；Visual Basic 会自动转换类型。若想了解 Visual Basic 正在使用哪种变量类型，可使用 VarType 函数。

例如，如果在 Variant 变量中存储二进制值，Visual Basic 就会用 Double 内部表示。如果知道应用程序并不需要 Double 值提供的高精度（以及由此而来的低速度），那么将此值转换成 Single 甚至转换成 Currency 就可以加快计算速度：

```
If VarType (X) = 5 Then X = CSng (X) '转换成单精度。
```

对 Array 变量，VarType 的值等于数组元素与数据类型返回值之和。例如，这个数组包含 Double 值：

```
Private Sub Form_Click ()  
    Dim dblSample (2) As Double  
    MsgBox VarType (dblSample)  
End Sub
```

Visual Basic 的今后版本可能会增加新的 Variant 表示，所以，任何根据 VarType 函数的返回值做出判断的代码都应能适当地处理非当前定义的返回值。

详细信息 关于 VarType 函数的信息，请参阅《Visual Basic 6.0 语言参考手册》中的“VarType 函数”。关于数组的更详细信息，请参阅本章后面的“数组”。关于转换数据类型的详细信息，请参阅本章前面的“数据类型”。

Variant 变量中存储的 Numeric 值

在 Variant 变量中存储整个数时，Visual Basic 使用最紧凑的表示。例如，若是存储不带小数点的微小的数值，Variant 对该值使用 Integer 表示。如果此后赋予一个较大的数，则 Visual Basic 将使用 Long 值，如果这个数非常大或者有小数部分，则 Visual Basic 将使用 Double 值。

有时想对一个数使用特定表示。例如，为了在以后的计算中避免舍入误差，可能希望 Variant 变量将数值作为 Currency 存储起来。Visual Basic 提供了几种转换函数，可以用它们将值转换成指定的类型（参阅本章前面的“转换数据类型”）。例如，用 CCur 函数可将值转换成 Currency 类型：

```
PayPerWeek = CCur (hours * hourlyPay)
```

如果 Variant 变量不含数或者不含可以解释为数的内容，则对 Variant 变量执行数学运算或函数计算就会出错。例如，不能对 'U2' 这个值进行任何算术运算，尽管它包含一个数值字符 '2'，但整个值并非合法的数。同样不能对值 1040EZ 进行任何计算，但是，可以对 +10 或 -1.7E6 进行计算，因为它们是有有效的数。因此经常需要判定 Variant 变量是否包含可以当作数使用的值。IsNumeric 函数可以完成这样的任务：

```
Do
```

```
    anyNumber = InputBox ("Enter a number")
```

```
Loop Until IsNumeric (anyNumber)
```

```
MsgBox "The square root is: " & Sqr (anyNumber)
```

当 Visual Basic 将一个非数值的表示（如包含一个数的字符串）转换成数值时，它会使用地区设置（在 Windows 的“控制面板”中规定）来解释千位分隔符、小数点符号和货币符号。

于是，如果 Windows 的“控制面板”中的国家/地区设置值被设置成美国、加拿大或澳大利亚，则这两个语句将返回 True：

```
IsNumeric("$100")
```

```
IsNumeric("1,560.50")
```

而下面两个语句则返回 false：

```
IsNumeric("DM 100")
```

```
IsNumeric("1,560,50")
```

但是，如果在 Windows 的“控制面板”中的国家/地区设置值被设置成德国，则情况正好相反：前两个语句返回 false，而后两个语句返回 true。

如果把包含数的 Variant 赋给字符串变量或属性，Visual Basic 自动把数的内部表示转换成字符串。如果想显式地将数转换成字符串，可以用 CStr 函数。还可以用 Format 函数将数转换成字符串，字符串包含某种格式，如货币符、千位分隔符和小数点符号。Format 函数按照 Windows“控制面板”中的“地区设置”对话框自动使用相应的符号。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的 "Format 函数" 以及关于转换函数的主题。关于编写分布在国外市场的应用程序的更详细信息，请参阅第十六章“国际化”。

Variant 变量中存储的字符串

一般说来，在 Variant 变量中存储和使用字符串不会出什么问题。但是如前所述，有时在使用两个 Variant 值时，" + " 运算符的结果可能很含糊。如果两个 Variant 变量都包含数，则 " + " 运算符执行加法运算。如果两个 Variant 变量都包含字符串，则 " + " 运算符执行字符串连接。但是，如果一个值代表数，而另一个值代表字符串，则情况就很复杂了。Visual Basic 首先试图把字符串变成数。如果转换成功，那么 " + " 运算符把两者加在一起；如果不成功，则产生一个“类型不匹配”错误。

为了确保执行的动作为字符串连接，不管变量中的表示如何，都用 & 运算符。例如，以下代码：

```
Sub Form_Click ()  
    Dim X ,   Y  
    X = "6"  
    Y = "7"  
    Print X + Y,X & Y  
    X = 6  
    Print X + Y,X & Y  
End Sub
```

在窗体上产生如下结果：

67 67

注意 Visual Basic 在内部用 Unicode 存储字符串。关于 Unicode 的更详细的信息，请参阅第十六章“国际化”。

Variant 变量中存储的 Date/Time 值

Variant 变量还可以包含 Date/Time 值。有几个函数返回 Date/Time 值。例如，DateSerial 返回该年所剩的天数。

```
Private Sub Form_Click ()
    Dim rightnow ,    daysleft ,    hoursleft ,    minutesleft
    rightnow = Now           'Now 返回当前的 Date 和 Time。
    daysleft = Int(DateSerial(Year(rightnow) _
+ 1, 1, 1) - rightnow)
    hoursleft = 24 - Hour (rightnow)
    minutesleft = 60 - Minute (rightnow)
    Print daysleft & " days left in the year."
    Print hoursleft & " hours left in the day."
    Print minutesleft & " minutes left in the hour."
End Sub
```

还可以对 Date/Time 值进行运算。通过加减一个整数来增加或减少天数；通过加减一个分数来增加或减少时间。所以，加 20 就是加 20 天，而减掉 1/24 就是减去一小时。

存储在 Variant 变量中的 Date 值的有效范围从公元 0100 年 1 月 1 日到公元 9999 年 12 月 31 日。计算 Date 时不考虑 Gregorian 历法采用之前的日期。所以如果计算 Gregorian 历法被采用那年（在英国及其殖民地为 1752 年；其它国家/地区则或早一些或晚一些）之前的日期，结果可能会不正确。

在代码中可以采用 Date/Time 文字，只要用一对 " # " 号将它们括起来，如同对字符串用一对双引号 ("") 括起来一样。例如，可以将包含 Date/Time 值的 Variant 和一串表示 Date 的文字作比较：

```
If SomeDate > #3/6/93# Then
```

下面的例子将包含 Date/Time 值的 Variant 变量和一串表示 Date 和 Time 的文字相比较：

```
If SomeDate > #3/6/93 1:20pm# Then
```

如果 Date/Time 值不包括 Time，则 Visual Basic 自动将该值的 Time 部分设定为午夜（一天的开始）。如果 Date/Time 值不包括 Date，则 Visual Basic 自动将该值的 Date 部分设定为公元 1899 年 12 月 30 日。

Visual Basic 接受许多种文字的 Date 和 Time 格式。下面的 Date/Time 值全部有效：

```
SomeDate = #3-6-93 13:20#
```

```
SomeDate = #March 27 , 1993 1:20am#
```

```
SomeDate = #Apr-2-93#
```

```
SomeDate = #4 April 1993#
```

详细信息 有关用国际格式处理 Date 的信息，请参阅第十六章“国际化”。用 IsNumeric 函数可测试一个 Variant 变量是否包含一个值，这个值可以看作有效的数值，同样，用 IsDate 函数可测试一个 Variant 变量是否包含一个值，这个值可以看作有效的 Date/Time 值。然后可用 CDate 函数将该值转换成 Date/Time 值。

例如，以下代码用 IsDate 测试文本框的 Text 属性。如果属性包含被看作有效 Date 的文本，则 Visual Basic 将此文本转换成 Date 并计算到年底还有几天。

```
Dim SomeDate ,    daysleft
If IsDate (Text1.Text) Then
    SomeDate = CDate (Text1.Text)
    daysleft = DateSerial(Year(SomeDate) + _
    1, 1, 1) - SomeDate
    Text2.Text = daysleft & " days left in the year."
Else
    MsgBox Text1.Text & " is not a valid date."
End If
```

详细信息 关于各种 Date 和 Time 函数的信息，请参阅《Visual Basic 6.0 语言参考手册》中的“Date 函数”。

Variant 变量中存储的 Objects

可以将 Objects 存储在 Variant 变量中。当需要高效地处理包括 Objects 在内的各种数据类型时，这可能十分有用。例如，一个数组中的所有元素都必须具有相同的数据类型。将数组的数据类型设置成 Variant 就可将 Objects 连同其它数据类型一起存入同一个数组中。

数组

如果有过用其它语言编程的经历，那么想必会熟悉数组的概念。由于有了数组，可以用相同名字引用一系列变量，并用数字（索引）来识别它们。在许多场合，使用数组可以缩短和简化程序，因为可以利用索引值设计一个循环，高效处理多种情况。数组有上界和下界，数组的元素在上下界内是连续的。因为 Visual Basic 对每一个索引值都分配空间，所以不要不切实际声明一个太大的数组。

注意 这一部分讨论的数组是程序中声明的变量数组。它们不同于控件数组，控件数组是在设计时通过设置控件的 Index 属性规定的。变量数组总是连续的；与控件数组不同的是，不能从一个数组的中部加载或卸载数组元素。

一个数组中的所有元素具有相同的数据类型。当然，当数据类型为 Variant 时，各个元素能够包含不同种类的数据（对象、字符串、数值等等）。可

以声明任何基本数据类型的数组，包括用户定义的类型（请参阅第八章“再论编程”中的“创建自己的数据类型”）和对象变量（请参阅第九章“用对象编程”）。

在 Visual Basic 中有两种类型的数组：固定大小的数组——它总是保持同样的大小，以及在运行时大小可以改变的动态数组。在本章后面的“动态数组”中将要详细讨论动态数组。

声明固定大小的数组

有三种方法声明固定大小的数组，用哪一种方法取决于数组应有的有效范围：

- 建立公用数组，在模块的声明段用 **Public** 语句声明数组。
- 建立模块级数组，在模块的声明段用 **Private** 语句声明数组。
- 建立局部数组，在过程中用 **Private** 语句声明数组。

设定上下界

声明数组时，在数组名之后跟一个用括号括起来的上界。上界不得超过 **Long** 数据类型的范围（-2,147,483,648 到 2,147,483,647）。例如，下列数组声明可出现在模块的声明段：

<code>Dim Counters (14) As Integer</code>	'15 个元素。
<code>Dim Sums (20) As Double</code>	'21 个元素。

为建立公用数组，直接用 `Public` 取代 `Dim`。

```
Public Counters (14) As Integer
```

```
Public Sums (20) As Double
```

在过程之中同样的声明使用 `Dim`：

```
Dim Counters (14) As Integer
```

```
Dim Sums (20) As Double
```

第一个声明建立了一个有 15 个元素的数组，其索引号从 0 到 14。第二个声明建立了一个有 21 个元素的数组，其索引号从 0 到 20。缺省的下界为 0。

为了规定下界，用关键字 `To` 显式提供下界（为 `Long` 数据类型）：

```
Dim Counters (1 To 15) As Integer
```

```
Dim Sums (100 To 120) As String
```

在前述声明中，`Counters` 的索引值范围从 1 到 15，而 `Sums` 的索引值范围从 100 到 120。

包含其它数组的数组

有可能建立 `Variant` 数据类型数组，并与不同数据类型的数组共居一处。以下代码建立两个数组，一个包含整数，而另一个包含字符串。然后声明第三个 `Variant` 数组，并将整数和字符串数组放置其中：

```
Private Sub Command1_Click ()
```

```
Dim intX As Integer '声明计数器变量。
```

```
'声明并放置整数数组。
```

```
Dim countersA (5) As Integer
```

```
For intX = 0 To 4
```

```
countersA (intX) = 5
```

```
Next intX
```

```
'声明并放置字符串数组。
```

```
Dim countersB (5) As String
```

```
For intX = 0 To 4
```

```
countersB (intX) = "hello"
```

```
Next intX
```

```
Dim arrX (2) As Variant
```

```
'声明拥有两个成员的新数组。
```

```
arrX (1) = countersA ()
```

```
'将其它数组移居到数组。
```

```
arrX (2) = countersB ()
```

```
MsgBox arrX (1) (2)
```

```
'显示每一个数组的成员。
```

```
MsgBox arrX (2) (3)
```

```
End Sub
```

多维数组

有时需要追踪记录数组中的相关信息。例如，为了追踪记录计算机屏幕上的每一个像素，需要引用它的 X、Y 坐标。这时应该用多维数组存储值。

可用 Visual Basic 声明多维数组。例如，下面的语句声明了一个过程内的 10×10 的二维数组。

```
Static MatrixA (9, 9) As Double
```

可用显式下界来声明两个维数或两个维数中的任何一个：

```
Static MatrixA (1 To 10, 1 To 10) As Double
```

可以将所有这些推广到二维以上的数组。例如：

```
Dim MultiD (3, 1 To 10, 1 To 15)
```

这个声明建立了三维数组，大小为 $4 \times 10 \times 15$ 。元素总数为三个维数的乘积，为 600。

注意 在增加数组的维数时，数组所占的存储空间会大幅度增加，所以要慎用多维数组。使用 Variant 数组时更要格外小心，因为它们需要更大的存储空间。

用循环操作数组

可以用 For 循环嵌套有效的处理多维数组。例如，在 MatrixA 中基于每个元素在数组中的位置为其赋值：

```
Dim I As Integer, J As Integer
```

```
Static MatrixA(1 To 10, 1 To 10) As Double
```

```
For I = 1 To 10
```

```
    For J = 1 To 10
```

MatrixA (I, J) = I * 10 + J

Next J

Next I

详细信息 关于循环的详细信息，请参阅本章后面的“循环结构”部分。

动态数组

数组到底应该有多大才合适，有时可能不得而知。所以希望能够在运行时具有改变数组大小的能力。

动态数组就可以在任何时候改变大小。在 Visual Basic 中，动态数组最灵活、最方便，有助于有效管理内存。例如，可短时间使用一个大数组，然后，在不使用这个数组时，将内存空间释放给系统。

如果不用动态数组，就要声明一个数组，它的大小尽可能达到最大，然后再抹去那些不必要的元素。但是，如果过度使用这种方法，会导致内存的操作环境变慢。

要创建动态数组，请按照以下步骤执行：

1. （如果希望数组为公用数组，则）用 Public 语句声明数组，或者（如果希望数组为模块级，则）在模块级用 Dim 语句声明数组，或者（如果希望数组为局部数组，则）在过程中用 Static 或 Dim 语句声明数组。给数组附以一个空维数表，这样就将数组声明为动态数组。

Dim DynArray ()

2. 用 ReDim 语句分配实际的元素个数。

```
ReDim DynArray (X + 1)
```

ReDim 语句只能出现在过程中。与 Dim 语句、Static 语句不同，ReDim 语句是一个可执行语句，由于这一语句，应用程序在运行时执行一个操作。

ReDim 语句支持这样的语法，它与固定数组中使用的语法相同。对于每一维数，每个 ReDim 语句都能改变元素数目以及上下界。但是，数组的维数不能改变。

```
ReDim DynArray (4 to 12)
```

例如，用第一次声明在模块级所建立的动态数组 `Matrix1`：

```
Dim Matrix1 () As Integer
```

然后，在过程中给数组分配空间：

```
Sub CalcValuesNow ()
```

```
    .  
    .  
    .
```

```
    ReDim Matrix1 (19, 29)
```

```
End Sub
```

这里的 ReDim 语句给 `Matrix` 分配一个 20×30 的整数矩阵（元素总大小为 600）。还有一个办法，用变量设置动态数组的边界：

```
ReDim Matrix1 (X, Y)
```

注意 您可以将字符串赋值给大小可变的字节数组。一个字节数组也可以

被赋值给一个可变长的字符串。一定要注意字符串中的字节数会随平台而变化。同一个字符串在 Unicode 平台上的字节数是它在非 Unicode 平台上的两倍。

保留动态数组的内容

每次执行 `ReDim` 语句时，当前存储在数组中的值都会全部丢失。Visual Basic 重新将数组元素的值置为 `Empty`（对 `Variant` 数组）、置为 `0`（对 `Numeric` 数组）、置为零长度字符串（对 `String` 数组）或者置为 `Nothing`（对于对象的数组）。

在为新数据准备数组，或者要缩减数组大小以节省内存时，这样做是非常有用的。有时希望改变数组大小又不丢失数组中的数据。使用具有 `Preserve` 关键字的 `ReDim` 语句就可做到这点。例如，使用 `UBound` 函数引用上界，使数组扩大、增加一个元素，而现有元素的值并未丢失：

```
ReDim Preserve DynArray (UBound (DynArray) + 1)
```

在用 `Preserve` 关键字时，只能改变多维数组中最后一维的上界；如果改变了其它维或最后一维的下界，那么运行时就会出错。所以可这样编程：

```
ReDim Preserve Matrix (10, UBound (Matrix, 2) + 1)
```

而不可这样编程：

```
ReDim Preserve Matrix (UBound (Matrix, 1) + 1, 10)
```

详细信息 关于动态数组的更详细信息，请参阅语言参考中的“`ReDim` 函数”。关于对象数组，请参阅第九章“用对象编程”。

过程概述

将程序分割成较小的逻辑部件就可以简化程序设计任务。称这些部件为过程，它们可以变成增强和扩展 Visual Basic 的构件。

过程可用于压缩重复任务或共享任务，例如，压缩频繁的计算、文本与控件操作和数据库操作。

用过程编程有两大好处：

- 过程可使程序划分成离散的逻辑单元，每个单元都比无过程的整个程序容易调试。
- 一个程序中的过程，往往不必修改或只需稍作改动，便可以成为另一个程序的构件。

在 Visual Basic 中使用下列几种过程：

- Sub 过程不返回值。
- Function 过程返回值。
- property 过程返回并指定值，以及设置对象引用。

详细信息 关于属性过程，请参阅第九章“用对象编程”。

Sub 过程

子过程是在响应事件时执行的代码块。将模块中的代码分成子过程后，在应用程序中查找和修改代码变得更容易了。

子过程的语法是：

```
[Private|Public][Static]Sub procedurename (arguments)
```

```
    statements
```

```
End Sub
```

每次调用过程都会执行 **Sub** 和 **End Sub** 之间的 **statements**。可以将子过程放入标准模块、类模块和窗体模块中。按照缺省规定，所有模块中的子过程为 **Public**（公用的），这意味着在应用程序中可随处调用它们。

过程的 *arguments* 类似于变量声明，它声明了从调用过程传递进来的值。在 **Visual Basic** 中应区分通用过程和事件过程这两类子过程。

通用过程

通用过程告诉应用程序如何完成一项指定的任务。一旦确定了通用过程，就必须专由应用程序来调用。反之，直到为响应用户引发的事件或系统引发的事件而调用事件过程时，事件过程通常总是处于空闲状态。

为什么要建立通用过程呢？理由之一就是，几个不同的事件过程也许要执行同样的动作。将公共语句放入一分离开的过程（通用过程）并由事件过程来调用它，诚为编程上策。这样一来就不必重复代码，也容易维护应用程序。例如，**VCR** 示例应用程序使用了一个通用过程，几个不同滚动按钮的 **Click** 事件都调用这个通用过程。图 5.7 说明了通用过程的使用。**Click** 事件中的代码调用按钮管理器的子过程，子过程运行自身的代码，然后将控制返回到 **Click** 事件过程。

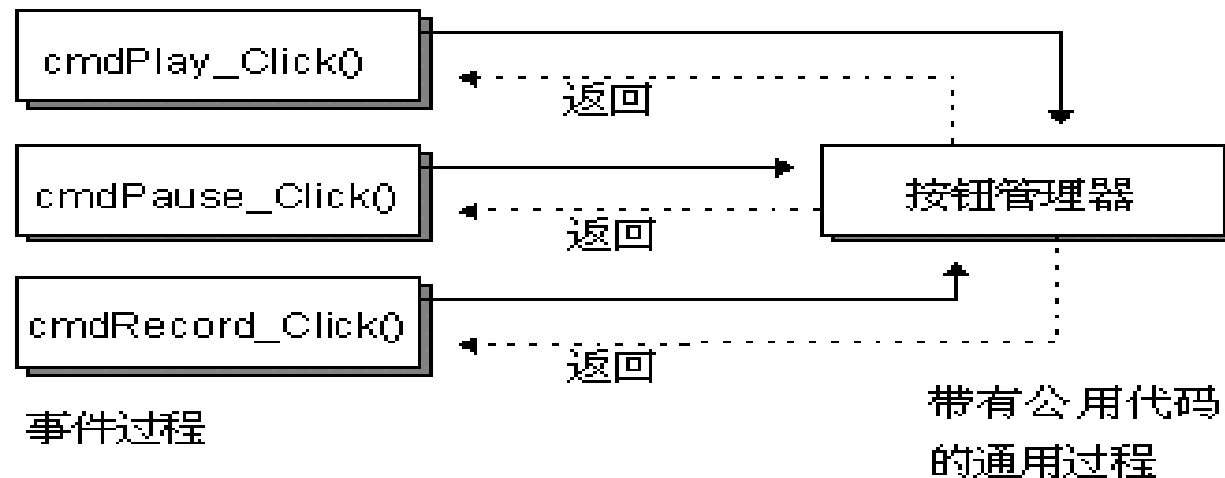


图 5.7 通用过程如何被事件过程调用

事件过程

当 Visual Basic 中的对象对一个事件的发生作出认定时，便自动用相应于事件的名字调用该事件的过程。因为名字在对象和代码之间建立了联系，所以说事件过程是附加在窗体和控件上的。

- 一个控件的事件过程将控件的（在 Name 属性中规定的）实际名字、下划线（_）和事件名组合起来。例如，如果希望在单击了一个名为 cmdPlay 的命令按钮之后，这个按钮会调用事件过程，则使用 cmdPlay_Click 过程。
- 一个窗体事件过程将词汇 "Form"、下划线和事件名组合起来。如果希

望在单击窗体之后，窗体会调用事件过程，则要使用 `Form_Click` 过程。

（和控件一样，窗体也有唯一的名字，但不能在事件过程的名字中使用这些名字。）如果正在使用 `MDI` 窗体，则事件过程将词汇 “`MDIForm`”、下划线和事件名组合起来，如 `MDIForm_Load`。

所有的事件过程使用相同的语法。

控件事件的语法	窗体事件的语法
<pre>Private Sub <i>controlname_eventname</i> (<i>arguments</i>) <i>Statements</i> End Sub</pre>	<pre>Private Sub <i>Form_eventname</i> (<i>arguments</i>) <i>Statements</i> End Sub</pre>

虽然可以自己编写事件过程，但使用 `Visual Basic` 提供的代码过程会更方便，这个过程自动将正确的过程名包括进来。从“对象框”中选择一个对象，从“过程框”中选择一个过程，就可在“代码编辑器”窗口选择一个模板。

在开始为控件编写事件过程之前先设置控件的 `Name` 属性，这不失为一个好主意。如果对控件附加一个过程之后又更改控件的名字，那么也必须更改过程的名字，以符合控件的新名字。否则，`Visual Basic` 无法使控件和过程相符。过程名与控件名不符时，过程就成为通用过程。

详细信息 `Visual Basic` 可识别各种窗体和控件的各种事件。关于所有事件的说明，请参阅语言参考。

Function 过程

Visual Basic 包含内置的或内部的函数，如 Sqr、Cos 或 Chr。此外，还可使用 Function 语句编写自己的 Function 过程。

函数过程的语法是：

```
Private|Public|[Static]Function procedurename (arguments) [As type]
```

```
    statements
```

```
End Function
```

与 Sub 过程一样，Function 过程也是一个独立的过程，可读取参数、执行一系列语句并改变其参数的值。与子过程不同，Function 过程可返回一个值到调用的过程。在 Sub 过程与 Function 过程之间有三点区别：

- 一般说来，让较大的语句或表达式的右边包含函数过程名和参数 (*returnvalue = function*)，这就调用了函数。
- 与变量完全一样，函数过程有数据类型。这就决定了返回值的类型（如果没有 As 子句，缺省的数据类型为 Variant）。
- 给 *procedurename* 自身赋一个值，就可返回这个值。Function 过程返回一个值时，该值可成为较大表达式的一部分。

例如，下面是已知直角三角形两直角边的值，计算第三边（斜边）的函数：

```
Function Hypotenuse (A As Integer, B As Integer) As String
```

```
    Hypotenuse = Sqr (A ^ 2 + B ^ 2)
```

```
End Function
```

在 Visual Basic 中调用 Function 过程的方法和调用任何内部函数的方法是一样的：

```
Label1.Caption = Hypotenuse(CInt(Text1.Text), _  
CInt(Text2.Text))  
strX = Hypotenuse (Width, Height)
```

详细信息 关于函数过程的详细信息，请参阅《Visual Basic 6.0 语言参考手册》中的“Function 语句”。调用各种类型的过程的技巧，在本章后面的“调用过程”部分有详细讨论。

使用过程

创建新过程

要创建新的通用过程，请

- 在“代码”窗口输入过程头并按下回车键。过程头以 Sub 或 Function 打头，再接一个名字。例如，可以用以下任一方式输入：

```
Sub UpdateForm ()
```

```
Function GetCoord ()
```

通过完成新过程的模板， Visual Basic 作出反应。

选择现有过程

要在当前模块中查看过程，请

- 为了查看现有通用过程，在“代码”窗口的“对象框”中选择“通用”，然后在“过程框”中选择过程。

—或者—

为查看事件过程，在“代码”窗口的“对象框”中选择适当的对象，然后在“过程框”中选择事件。

要查看其它模块中的过程，请

1. 在“视图”菜单中选取“对象浏览器”。
2. 在“工程/库”框中选择工程。
3. 在“类/模块”列表中选择模块，并在“成员”列表中选择过程。
4. 选取“查看定义”。

调用过程

调用过程有诸多技巧，它们与过程的类型、位置以及在应用程序中的使用方式有关。下面的章节说明如何调用 Sub 过程和 Function 过程。

调用 Sub 过程

与 Sub 过程不同，在表达式中，Sub 过程不能用其名字调用。调用 Sub 过

程的是一个独立的语句。Sub 过程还有一点与函数不一样，它不会用名字返回一个值。但是，与 Function 过程一样，Sub 过程也可以修改传递给它们的任何变量的值。

调用 Sub 过程有两种方法：

'以下两个语句都调用了名为 MyProc 的 Sub 过程。

```
Call MyProc (FirstArgument, SecondArgument)
```

```
MyProc FirstArgument, SecondArgument
```

注意，当使用 Call 语法时，参数必须在括号内。若省略 Call 关键字，则也必须省略参数两边的括号。

调用函数过程

通常，调用自行编写的函数过程的方法和调用 Visual Basic 内部函数过程（例如 Abs）的方法一样；即在表达式中写上它的名字。

'下面的语句都调用函数 ToDec。

```
Print 10 * ToDec
```

```
X = ToDec
```

```
If ToDec = 10 Then Debug.Print "Out of Range"
```

```
X = AnotherFunction ( 10 * ToDec)
```

就像调用 Sub 过程那样，也能调用函数。下面的语句都调用同一个函数：

```
Call Year (Now)
```

```
Year Now
```

当用这种方法调用函数时，Visual Basic 放弃返回值。

调用其它模块中的过程

在工程中的任何地方都能调用其它模块中的公用过程。可能需要指定这样的模块，它包含正在调用的过程。调用其它模块中的过程的各种技巧，取决于该过程是在窗体模块中、类模块中还是标准模块中。

窗体中的过程

所有窗体模块的外部调用必须指向包含此过程的窗体模块。如果在窗体模块 `Form1` 中包含 `SomeSub` 过程，则可使用下面的语句调用 `Form1` 中的过程：

```
Call Form1.SomeSub(arguments)
```

类模块中的过程

与窗体中调用过程类似，在类模块中调用过程要调用与过程一致并且指向类实例的变量。例如，`DemoClass` 是类 `Class1` 的实例：

```
Dim DemoClass as New Class1
```

```
DemoClass.SomeSub
```

但是不同于窗体的是，在引用一个类的实例时，不能用类名作限定符。必须首先声明类的实例为对象变量（在这个例子中是 `DemoClass`）并用变量

名引用它。

详细信息 可在第九章“用对象编程”中查找关于对象变量和类模块的详细信息。

标准模块中的过程

如果过程名是唯一的，则不必在调用时加模块名。无论是在模块内，还是在模块外调用，结果总会引用这个唯一过程。如果过程仅出现在一个地方，这个过程就是唯一的。

如果两个以上的模块都包含同名的过程，那就有必要用模块名来限定了。在同一模块内调用一个公共过程就会运行该模块内的过程。例如，对于 `Module1` 和 `Module2` 中名为 `CommonName` 的过程，从 `Module2` 中调用 `CommonName` 则运行 `Module2` 中的 `CommonName` 过程，而不是 `Module1` 中的 `CommonName` 过程。

从其它模块调用公共过程名时必须指定那个模块。例如，若在 `Module1` 中调用 `Module2` 中的 `CommonName` 过程，要用下面的语句：

```
Module2.CommonName (arguments)
```

向过程传递参数

过程中的代码通常需要某些关于程序状态的信息才能完成它的工作。信息包括在调用过程时传递到过程内的变量。当将变量传递到过程时，称变量

为参数。

参数的数据类型

过程的参数被缺省为具有 `Variant` 数据类型。不过，也可以声明参数为其它数据类型。例如，下面的函数接受一个字符串和一个整数：

```
Function WhatsForLunch(WeekDay As String, Hour _  
As Integer) As String  
    '根据星期几和时间，返回午餐菜单。  
    If WeekDay = "Friday" then  
        WhatsForLunch = "Fish"  
    Else  
        WhatsForLunch = "Chicken"  
    End If  
    If Hour > 4 Then WhatsForLunch = "Too late"  
End Function
```

详细信息 关于 `Visual Basic` 的数据类型，请参阅本章前面部分。也可以在语言参考中查找指定的数据类型。

按值传递参数

按值传递参数时，传递的只是变量的副本。如果过程改变了这个值，则所

作变动只影响副本而不会影响变量本身。用 **ByVal** 关键字指出参数是按值来传递的。

例如：

```
Sub PostAccounts (ByVal intAcctNum as Integer)
    .
    . '这里放语句。
    .
End Sub
```

按地址传递参数

按地址传递参数使过程用变量的内存地址去访问实际变量的内容。结果，将变量传递给过程时，通过过程可永远改变变量值。按地址传递参数在 **Visual Basic** 中是缺省的。

如果给按地址传递参数指定数据类型，就必须将这种类型的值传给参数。可以给参数传递一个表达式，而不是数据类型。**Visual Basic** 计算表达式，如果可能的话，还会按要求的类型将值传递给参数。

把变量转换成表达式的最简单的方法就是把它放在括号内。例如，为了把声明为整数的变量传递给过程，该过程以字符串为参数，则可以用下面的语句：

```
Sub CallingProcedure ()
    Dim intX As Integer
```



```
intX = 12 * 3
Foo (intX)
End Sub
```

```
Sub Foo (Bar As String)
    MsgBox Bar           'Bar 的值为字符串 ‘ 36 ’。
End Sub
```

使用可选的参数

在过程的参数列表中列入 **Optional** 关键字，就可以指定过程的参数为可选的。如果指定了可选参数，则参数表中此参数后面的其它参数也必是可选的，并且要用 **Optional** 关键字来声明。下面两段示例代码假定有一个窗体，其内有一命令按钮和一列表框。

例如，这段代码提供所有可选参数：

```
Dim strName As String
Dim strAddress As String

Sub ListText(Optional x As String, Optional y _
As String)
    List1.AddItem x
    List1.AddItem y
```

```
End Sub
```

```
Private Sub Command1_Click ()
```

```
    strName = "yourname"
```

```
    strAddress = 12345 '提供了两个参数。
```

```
    Call ListText (strName, strAddress)
```

```
End Sub
```

而下面的代码并未提供全部可选参数：

```
Dim strName As String
```

```
Dim varAddress As Variant
```

```
Sub ListText (x As String, Optional y As Variant)
```

```
    List1.AddItem x
```

```
    If Not IsMissing (y) Then
```

```
        List1.AddItem y
```

```
    End If
```

```
End Sub
```

```
Private Sub Command1_Click ()
```

```
    strName = "yourname" '未提供第二个参数。
```

```
    Call ListText (strName)
```

```
End Sub
```

在未提供某个可选参数时，实际上将该参数作为具有 `Empty` 值的变体来赋值。上例说明如何用 `IsMissing` 函数测试丢失的可选参数。

提供可选参数的缺省值

也可以给可选参数指定缺省值。在下例中，如果未将可选参数传递到函数过程，则返回一个缺省值。

```
Sub ListText(x As String, Optional y As _  
Integer = 12345)  
    List1.AddItem x  
    List1.AddItem y  
End Sub
```

```
Private Sub Command1_Click()  
    strName = "yourname"      '未提供第二个参数。  
    Call ListText(strName)    '添加“yourname”和“12345”。  
End Sub
```

使用不定数量的参数

一般说来，过程调用中的参数个数应等于过程说明的参数个数。可用 `ParamArray` 关键字指明，过程将接受任意个数的参数。于是可以这样来编写计算总和的 `Sum` 函数：

```
Dim x As Integer
Dim y As Integer
Dim intSum As Integer

Sub Sum (ParamArray intNums ())
    For Each x In intNums
        y = y + x
    Next x
    intSum = y
End Sub

Private Sub Command1_Click ()
    Sum 1, 3, 5, 7, 8
    List1.AddItem intSum
End Sub
```

用命名的参数创建简单语句

对许多内建函数、语句和方法，Visual Basic 提供了命名参数方法来快捷传递参数值。对命名参数，通过给命名参数赋值，就可按任意次序提供任意多参数。为此，键入命名参数，其后为冒号、等号和值 (**MyArgument := "SomeValue"**)，可以按任意次序安排这些赋值，它们之间用逗号分开。注

意，下例中的参数顺序和所要参数的顺序相反：

```
Function ListText (strName As String, Optional strAddress As String)
```

```
    List1.AddItem strName
```

```
    List2.AddItem strAddress
```

```
End Sub
```

```
Private Sub Command1_Click ()
```

```
    ListText strAddress:="12345", strName:="Your Name"
```

```
End Sub
```

如果过程有若干不必总要指定的可选参数，则上述内容更为有用。

确定对命名参数的支持

要确定哪一个函数、语句和方法支持命名参数，用“代码”窗口中的“AutoQuickInfo”功能，检查“对象浏览器”，或者参阅语言参考。使用命名参数时要注意以下几点：

- 在 Visual Basic (VB) 对象库中的对象的方法不支持命名参数。而 Visual Basic for applications (VBA) 对象库中的所有的语言关键字都支持命名的参数。
- 在语法中，命名参数是用粗体和斜体字表示的。所有其它参数只用斜体字表示。

重点 使用命名参数时不能省略所需参数的输入。可以只省略可选参数。

对于 Visual Basic (VB) 和 Visual Basic for applications (VBA) 对象库，“对象浏览器”对话框将可选参数用方括号 [] 括起来。

详细信息 请参阅语言参考中的 "ByVal"、"ByRef"、"Optional" 和 "Param Array"。

控制结构概述

有了控制结构就可控制程序执行的流程。如果未复选控制流语句，程序便从左至右、自顶向下地贯穿这些语句。有些简单程序可以只用单向流程来编写，有些流程可以依靠运算符的优先级来控制，但任何编程语言的效力和用途皆由其通过结构和循环改变语句顺序的能力而得。

判定结构

Visual Basic 过程能够测试条件式，然后根据测试结果执行不同的操作。Visual Basic 支持的判定结构有：

- If...Then
- If...Then...Else
- Select Case

If...Then

用 If...Then 结构有条件地执行一个或多个语句。单行语法和多行块语法都可以使用：

If *condition* **Then** *statement*

If *condition* **Then**

statements

End If

condition 通常是比较式，但它可以是任何计算数值的表达式。Visual Basic 将这个值解释为 True 或 False：一个为零的数值为 False，而任何非零数值都被看作 True。若 *condition* 为 True，则 Visual Basic 执行 Then 关键字后面的所有 *statements*。可以使用单行或多行语法有条件地执行一个语句（下面两个例子等价）：

```
If anyDate < Now Then anyDate = Now
```

```
If anyDate < Now Then
```

```
    anyDate = Now
```

```
End If
```

注意：If...Then 的单行格式不用 End If 语句。如果 *condition* 为 True 时要执行多行代码，则必须使用多行块 If...Then...End If 语法。

```
If anyDate < Now Then
```

```
anyDate = Now
```

```
Timer1.Enabled = False
```

'定时器控制失效。

```
End If
```

If...Then...Else

用 If...Then...Else 块定义几个语句块，执行其中一个语句：

```
If condition1 Then
```

```
    [statementblock-1]
```

```
[ElseIf condition2 Then
```

```
    [statementblock-2]] ...
```

```
[Else
```

```
    [statementblock-n]]
```

```
End If
```

Visual Basic 首先测试 *condition1*。如果它为 False，Visual Basic 就测试 *condition2*，依此类推，直到找到一个为 True 的条件。当它找到一个为 True 的条件时，Visual Basic 就会执行相应的语句块，然后执行 End If 后面的代码。作为一个选择，可以包含 Else 语句块，如果条件都不是 True，则 Visual Basic 执行 Else 语句块。

If...Then...ElseIf 只是 If...Then...Else 的一个特例。注意，可以使用任意数量的 ElseIf 子句，或者一个也不用。可以有一个 Else 子句，而不管有没有 ElseIf 子句。

例如，应用程序可以根据菜单控件数组中哪个控件被单击而采取相应的动作：

```
Private Sub mnuCut_Click(Index As Integer)
    If Index = 0 Then                '“剪切”命令。
        CopyActiveControl           '调用通用过程。
        ClearActiveControl
    ElseIf Index = 1 Then            '“复制”命令。
        CopyActiveControl
    ElseIf Index = 2 Then            '“清除”命令。
        ClearActiveControl
    Else                             '“粘贴”命令。
        PasteActiveControl
    End If
End Sub
```

注意，总是可以添加更多的 `ElseIf` 块到 `If...Then` 结构中去。但是，当每个 `ElseIf` 都将相同的表达式比作不同的数值时，这个结构编写起来很乏味。在这种情况下可以使用 `Select Case` 判定结构。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的“`If...Then...Else` 语句”。

Select Case

Visual Basic 提供 **Select Case** 结构替代 **If...Then...Else**，从而可在多个语句块中有选择地执行其中一个。**Select Case** 语句的能力与 **If...Then...Else** 语句类似，但对多重选择的情况，**Select Case** 语句使代码更加易读。

Select Case 在结构的上方处理一个测试表达式并只计算一次。然后，Visual Basic 将表达式的值与结构中的每个 **Case** 的值进行比较。如果相等，就执行与该 **Case** 相关联的语句块。

Select Case *testexpression*

 [**Case** *expressionlist1*

 [*statementblock-1*]]

 [**Case** *expressionlist2*

 [*statementblock-2*]]

 .

 .

 .

 [**Case Else**

 [*statementblock-n*]]

End Select

每一个 *expressionlist* 是一个或几个值的列表。如果在一个列表中有多个值，就用逗号把值隔开。每一个 *statementblock* 中含有零个或多个语句。

如果不止一个 Case 与测试表达式相匹配，则只对第一个匹配的 Case 执行与之相关联的语句块。如果在表达式列表中没有一个值与测试表达式相匹配，则 Visual Basic 执行 Case Else 子句（此项是可选的）中的语句。例如，假定在 If...Then...Else 的例子中要向“编辑”菜单添加命令。为此可以另加一个 ElseIf 子句，或用 Select Case 来写函数：

```
Private Sub mnuCut_Click(Index As Integer)
    Select Case Index
        Case 0
            CopyActiveControl
            ClearActiveControl
            '“剪切”命令。
            '调用通用过程。
        Case 1
            CopyActiveControl
            '“复制”命令。
        Case 2
            ClearActiveControl
            '“清除”命令。
        Case 3
            PasteActiveControl
            '“粘贴”命令。
        Case Else
            frmFind.Show
            '显示找到的对话框。
    End Select
End Sub
```

注意 Select Case 结构每次都要在开始处计算表达式的值。而 If...Then...Else 结构为每个 ElseIf 语句计算不同的表达式。只有在 If 语句和每一个 ElseIf

语句计算相同表达式时，才能用 `Select Case` 结构替换 `If...Then...Else` 结构。

循环结构

循环结构允许重复执行一行或数行代码。Visual Basic 支持的循环结构有：

- `Do...Loop`
- `For...Next`
- `For Each...Next`

`Do...Loop`

用 `Do` 循环重复执行一语句块，且重复次数不定。`Do...Loop` 语句有几种演变形式，但每种都计算数值条件以决定是否继续执行。如同 `If...Then condition` 必须是一个数值或者值为 `True`（非零）或 `False`（零）的表达式。在下面的 `Do...Loop` 循环中，只要 `condition` 为 `True` 就执行 `statements`。

```
Do While condition  
    statements
```

```
Loop
```

当 Visual Basic 执行这个 `Do` 循环时会首先测试 `condition`。如果 `condition` 为 `False`（零），则跳过所有语句。如果 `condition` 为 `True`（非零），则 Visual Basic 执行语句，然后退回到 `Do While` 语句再测试条件。

因此，只要 *condition* 为 **True** 或非零，循环可以随意执行多少次。如果 *condition* 一开始便为 **False**，则不会执行语句。例如，以下过程将计算某一目标字符串在另一字符串中出现的次数，只要发现目标串就执行循环：

```
Function CountStrings (longstring, target)
    Dim position, count
    position = 1
    Do While InStr (position, longstring, target)
        position = InStr(position, longstring, target)_
            + 1
        count = count + 1
    Loop
    CountStrings = count
End Function
```

如果目标字符串未出现在另一个字符串中，则 **InStr** 返回 0，而且不再执行循环。

Do...Loop 语句的另一种演变形式是先执行语句，然后在每次执行后测试 *condition*。这种形式保证 *statements* 至少执行一次：

```
Do
    statements
Loop While condition
```

其它两种演变形式类似于前两个，所不同的是，只要 *condition* 为 **False** 而

不是 **True**，它们就执行循环。

循环零次或多次	至少循环一次
Do Until <i>condition</i>	Do
<i>Statements</i>	<i>Statements</i>
Loop	Loop Until <i>condition</i>

For...Next000

在不知道循环内需要执行多少次语句时，宜用 **Do** 循环。但是，在知道要执行多少次时，则最好使用 **For...Next** 循环。与 **Do** 循环不同，**For** 循环使用一个叫做计数器的变量，每重复一次循环之后，计数器变量的值就会增加或者减少。**For** 循环的语法如下：

```
For counter = start To end [Step increment]
```

```
    statements
```

```
Next [counter]
```

参数 *Counter*、*Start*、*end* 和 *increment* 都是数值型的。

注意 *increment* 参数可正可负。如果 *increment* 为正，则 *Start* 必须小于等于 *end*，否则不能执行循环内的语句。如果 *increment* 为负，则 *Start* 必须大于等于 *end*，这样才能执行循环体。如果没有设置 *Step*，则 *increment* 缺省值为 1。

在执行 **For** 循环时，Visual Basic

1. 设置 *counter* 等于 *start*。

2. 测试 *counter* 是否大于 *end*。若是的话，则 Visual Basic 退出循环。
(若 *increment* 为负，则 Visual Basic 测试 *counter* 是否小于 *end*。)
3. 执行语句。
4. *counter* 增加一，或者增加 *increment* (如果已指定的话)。
5. 重复步骤 2 到步骤 4。

以下代码打印出所有有效的屏幕字体名：

```
Private Sub Form_Click ()  
    Dim I As Integer  
    For i = 0 To Screen.FontCount  
        Print Screen.Fonts (i)  
    Next
```

```
End Sub
```

在 VCR 示例应用程序中，HighlightButton 过程使用 For...Next 循环，一步步经过 VCR 窗体的控件集合，并显示适当的 Shape 控件：

```
Sub HighlightButton (MyControl As Variant)  
    Dim i As Integer  
    For i = 0 To frmVCR.Controls.Count - 1  
        If TypeOf frmVCR.Controls (i) Is Shape Then  
            If frmVCR.Controls (i).Name = MyControl Then  
                frmVCR.Controls (i).Visible = True  
            Else  
                frmVCR.Controls (i).Visible = False  
            End If  
        End If  
    Next i  
End Sub
```

```
        End If
    End If
Next
End Sub
```

For Each...Next

For Each...Next 循环与 **For...Next** 循环类似，但它对数组或对象集合中的每一个元素重复一组语句，而不是重复语句一定的次数。如果不知道一个集合有多少元素，**For Each...Next** 循环非常有用。

For Each...Next 循环的语法如下：

```
For Each element In group
    statements
Next elementt
```

例如，下面的子过程打开 **Biblio.mdb**，把每一个表的名字加到列表框中。

```
Sub ListTableDefs ()
    Dim objDb As Database
    Dim MyTableDef as TableDef
    Set objDb = OpenDatabase("c:\vb\biblio.mdb", _
        True, False)
    For Each MyTableDef In objDb.TableDefs ()
        List1.AddItem MyTableDef.Name
    
```



```
Next MyTableDef  
End Sub
```

请记住使用 For Each...Next 时的几点限制：

- 对集合，*element* 只能是 Variant 变量，或一般的 Object 变量，或“对象浏览器”中列出的对象。
- 对数组，*element* 只能是 Variant 变量。
- For Each...Next 不能与用户自定义类型的数组一起使用，因为 Variant 不可能包含用户自定义类型。

使用控制结构

嵌套控制结构

可以把控制结构放入另一个控制结构之内（例如在 For...Next 循环中的 If...Then 块）。一个控制结构内部包含另一个控制结构叫做 nest（嵌套）。在 Visual Basic 中，控制结构的嵌套层数没有限制。按一般习惯，为了使判定结构和循环结构更具可读性，总是用缩排方式书写判定结构或循环的正文部分。

例如，下面的过程要把打印机和屏幕共有的字体名全部打印出来：

```
Private Sub Form_Click()  
    Dim SFont, PFont
```

```
For Each SFont In Screen.Fonts ()  
    For Each PFont In Printer.Fonts ()  
        If SFont = PFont Then  
            Print SFont  
        End If  
    Next PFont  
Next SFont  
End Sub
```

注意，第一个 **Next** 关闭了内层的 **For** 循环，而最后一个 **For** 关闭了外层的 **For** 循环。同样，在嵌套的 **If** 语句中，**End If** 语句自动与最靠近的前一个 **If** 语句配对。嵌套的 **Do...Loop** 结构的工作方式也是一样的，最内圈的 **Loop** 语句与最内圈的 **Do** 语句匹配。

退出控制结构

用 **Exit** 语句可以直接退出 **For** 循环、**Do** 循环、子过程或函数过程。**Exit** 语句的语法很简单：**Exit For** 在 **For** 循环中出现的次数没有限制，**Exit Do** 在 **Do** 循环中出现的次数也没有限制。

```
For counter = start To end [Step increment]  
    [statementblock]  
[Exit For]  
    [statementblock]
```

```
Next [counter[, counter] [,...]]
```

```
Do [{While | Until} condition]
```

```
    [statementblock]
```

```
    [Exit Do]
```

```
    [statementblock]
```

Loop

Exit Do 语句可以在 **Do** 循环语法的所有版本中使用。

Exit For 和 **Exit Do** 非常有用，因为它有时适于立即退出循环，而且不再执行循环中的任何进一步迭代或者语句。例如，在前面的打印屏幕和打印机共有字体的例子中，程序不断将打印机字体和给定的屏幕字体作比较，甚至在已经找到了一个相符的打字机字体后还在继续寻找。对这个函数有一个效率更高的改进版，在此，只要找到相匹配的字体后就立即退出循环：

```
Private Sub Form_Click ()
```

```
    Dim SFont, PFont
```

```
    For Each SFont In Screen.Fonts ()
```

```
        For Each PFont In Printer.Fonts ()
```

```
            If SFont = PFont Then
```

```
                Print Sfont
```

```
                Exit For
```

'退出内圈循环。

```
            End If
```

```
        Next PFont
```

```
    Next SFont
```

End Sub

正如此例所表明的，Exit 语句几乎总是出现在 If 语句或 Select Case 语句内部，而 If 语句或 Select Case 语句在循环内嵌套。

用 Exit 语句中断循环时，计数器变量的值会因退出循环的方式而不同：

- 在完成循环时，计数器的值等于上限值加上步进值。
- 在提前退出循环时，计数器变量保持其值，并遵从有关取值范围的一般规则。
- 在集合之外叠代时，如果计数器变量为 Object 类型，则其值为 Nothing；如果计数器变量为 Variant 类型，则其值为 Empty。

退出子过程或函数过程

也可从控制结构内部退出过程。Exit Sub 和 Exit Function 的语法，和上一节“退出控制结构”中的 Exit For 和 Exit Do 相似。Exit Sub 可以出现在子过程主体内的任何地方，出现的次数随需要而定。

当过程已完成每个任务并可直接返回时，Exit Sub 和 Exit Function 是非常有用的。例如，如果想改动前面的例子，使得对查找到的打印机和屏幕的共有字体，只打印其中的第一个，则可用 Exit Sub：

```
Private Sub Form_Click ()
```

```
    Dim SFont, PFont
```

```
    For Each SFont In Screen.Fonts ()
```

```
        For Each PFont In Printer.Fonts ()
```

```
        If SFont = PFont Then
            Print Sfont
            Exit Sub           '退出过程。
        End If
    Next PFont
Next SFont
End Sub
```

使用对象

在 Visual Basic 中创建应用程序之际，也就是和对象打交道之时了。可以使用 Visual Basic 提供的对象，比如控件、窗体和数据访问对象。也可以在一个 Visual Basic 应用程序内部控制另一个应用程序对象。甚至还可建立自己的对象，定义它们的属性和方法。

对象是什么

对象是代码和数据的组合，可以作为一个单位来处理。对象可以是应用程序的一部分，比如可以是控件或窗体。整个应用程序也是一个对象。下表列出了在 Visual Basic 中可能用到的几种类型的对象：

示例	描述
命令按钮	窗体上的控件，像命令按钮和框架，它们都是对象
窗体	Visual Basic 工程中的每一个窗体都是独立的对象
数据库	数据库是对象，并且还包含其他对象，如字段、索引等
图表	Microsoft Excel 中的图表是对象

对象从何而来

Visual Basic 中的每个对象都是用类定义的。用饼干模子和饼干之间的关系作比喻，就会明白对象和它的类之间的关系。饼干模子是类。它确定了每块饼干的特征，比如大小和形状。用类创建对象。对象就是饼干。

下面再用两个例子进一步说明 Visual Basic 中类与对象之间的关系。

- 在 Visual Basic 的“工具箱”上，控件代表类。直到在窗体上画出这些被称作控件的对象为止，它们实际上并不存在。在创建控件之时也就是在复制控件类，或建立控件类的实例。这个类实例就是应用程序中引用的对象。
- 在设计时操作的窗体是类。在运行时，Visual Basic 建立窗体的类实例。“属性”窗口显示 Visual Basic 应用程序中的对象的类和 Name 属性，如图 5.8 所示。

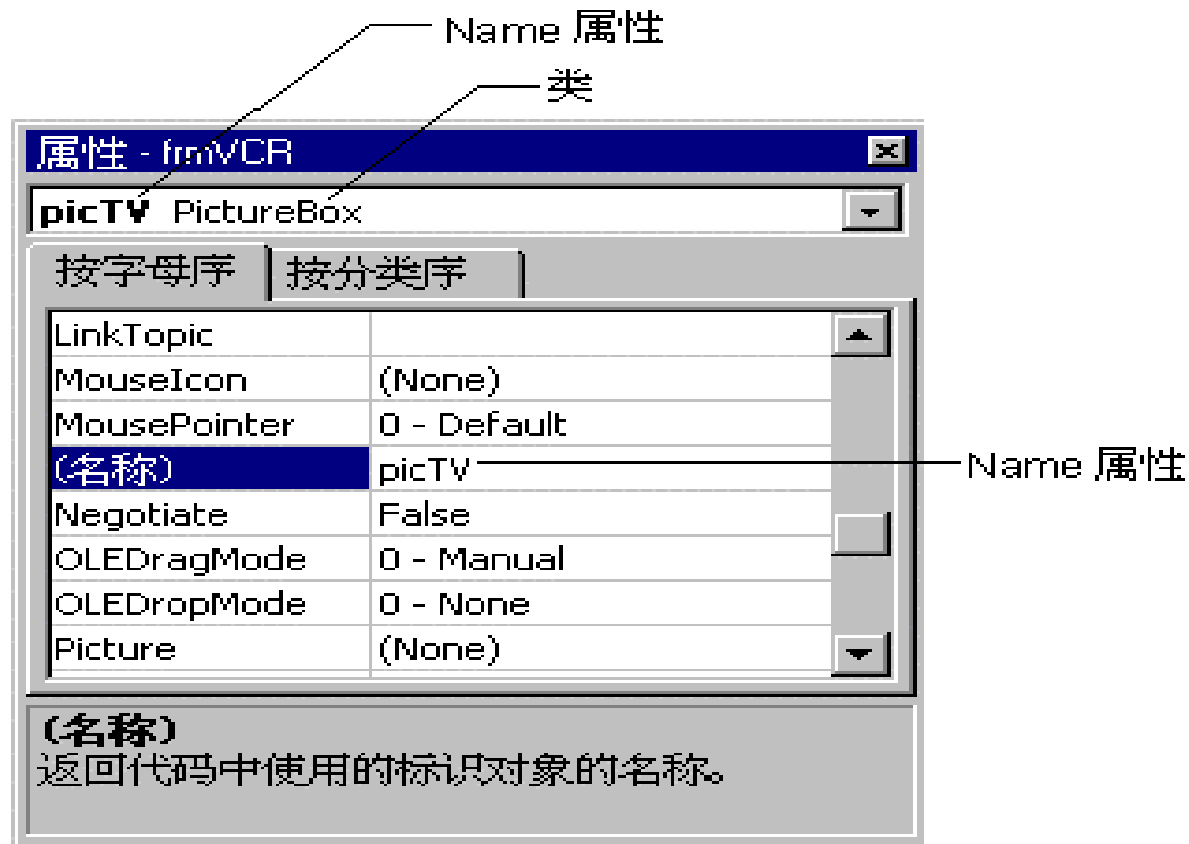


图 5.8 “属性”窗口中显示的对象和类名

把对象看作是类的原原本本的复制品，从而建立所有对象。一旦它们以单个对象的形式存在，属性就可改变。例如，如果在窗体上画了三个命令按钮，则每个命令按钮对象都是命令按钮类的实例。每个对象都具有一组由类定义的公共的特征和功能（属性、方法和事件）。但是，每个对象都有自己的名字，都能分别设置成有效或无效，都能放在窗体的不同位置，等等。

为简单起见，本章之外的大部分内容将不过多引用对象的类。例如，只要记住术语“ListBox 控件”意味着“列表框类的一个实例”。

用对象能做什么

对象可提供现成代码，省却书写麻烦。例如，可以自己创建打开文件和保存文件的对话框，但实际上大可不必。取而代之的是利用 Visual Basic 提供的 CommonDialog 控件（一个对象）。虽然用户也能撰写日程管理和资源管理的程序，但也大可不必。在此亦可使用 Microsoft Project 提供的 Calendar、Resources 和 Task 对象。

Visual Basic 可以组合来自其它来源的对象

Visual Basic 提供了把来自不同资源的对象组合起来的工具。现在可把 Visual Basic 的各种强有力的特性以及支持自动化（先前以 OLE 自动化闻名）的应用程序结合起来，建立定制的解决方法。自动化是部件对象模式 (COM) 的一个特性，它是应用程序使用的工业标准，用来陈列对象以开发工具和其它应用程序。

可把 Visual Basic 内部的控件结合在一起，也可使用其它应用程序提供的对象。考虑把下列对象放入 Visual Basic 窗体：

- Microsoft Excel Chart 对象
- Microsoft Excel Worksheet 对象

- Microsoft Word Document 对象

可用这些对象建立如图 5.9 所示的支票簿应用程序。由于不必书写代码，重建已由 Microsoft Excel 和 Word 等对象提供的功能，因而节省很多时间。

使用对象初步

Visual Basic 对象支持属性、方法和事件。在 Visual Basic 中，称对象的数据（设置和属性）为属性，称各种可在对象上操作的过程为方法。事件是可被对象识别的动作，例如单击鼠标和按下键盘键，还可编写代码来响应事件。

改变对象的属性就可改变对象的特性。用收音机打比方，收音机的一个属性是音量。用 Visual Basic 的行话来说，就是收音机有个“Volume”属性，改变其值就可调节音量大小。假定收音机的音量值可设置在 0 到 10 之间。如果能够通过 Visual Basic 控制收音机，则可在一个过程中写代码，把“Volume”属性值从 3 提高到 5，使声音更响一些：

```
Radio.Volume = 5
```

除了属性以外，对象还有方法。方法和属性都是对象的一部分。一般说来，方法就是要执行的动作，而属性就是要设置或检索的特性。以拨号打电话为例。可以说电话有一个“拨号”方法，拨一个 7 位电话号码的语法就是：

```
Phone.Dial 5551111
```

对象还有事件。当对象的某方面有变动时就触发了事件。例如，收音机可能有“VolumeChange”事件。电话可能有“Ring”事件，等等。

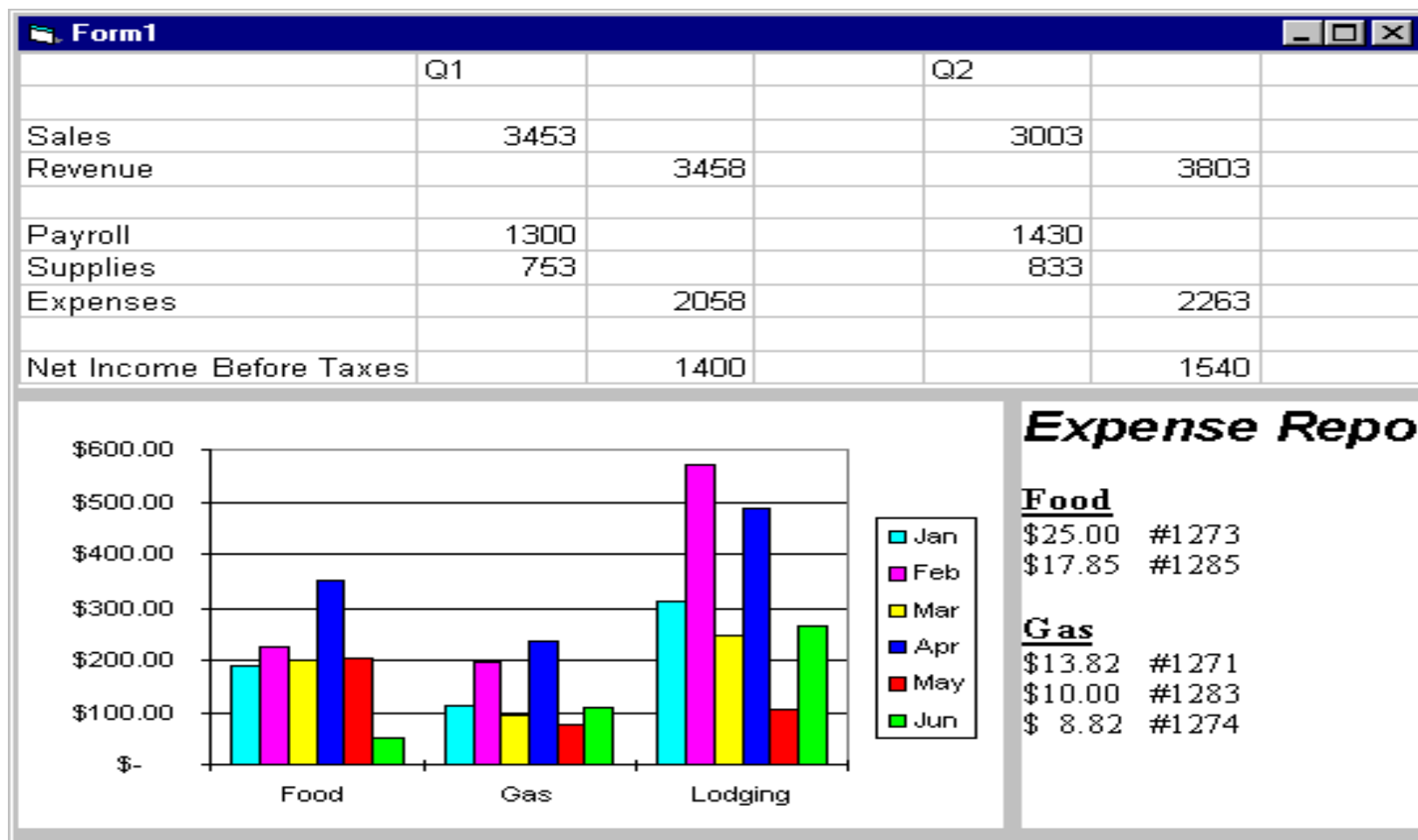


图 5.9 利用其它应用程序的对象

用属性控制对象

个别属性随着可以设置和取得它们的值的时间不同而不同。有的属性可在设计时设置。可在“属性”窗口设置这些属性的值而无需编写任何代码。而有的属性在设计时是不可用的；因此，这些属性只有通过代码在运行时设置。

在运行时可以设置并可获得值的属性叫做读写属性。在运行时只能读取的属性叫做只读属性。

设置属性值

在想改变对象的外观或特性时设置属性的值。例如，通过改变 `TextBox` 控件的 `Text` 属性，就可以改变文本框的内容。

用下列语法设置属性值：

`object.property = expression`

下面是设置属性的语句：

<code>Text1.Top = 200</code>	' 设置 <code>Top</code> 属性为 200 缇 (twips)。
<code>Text1.Visible = True</code>	' 显示文本框。
<code>Text1.Text = "hello"</code>	' 在文本框中显示 "hello"。

读取属性值

想要在代码执行附加动作（例如给另一个对象赋值）之前得知对象的状态，这时就要读取属性值。例如，在运行代码之前能够返回 `TextBox` 控件的 `Text` 属性值，以确定文本框的内容，这里，代码可能改变值。

在大多数情况下可以用以下语法获得属性值：

```
variable = object.property
```

属性值可以作为较复杂的表达式的一部分，而不必将属性赋予变量。下面的代码计算控件数组中的一个新成员的 `Top` 属性，它等于前一个成员的 `Top` 属性加上 400。

```
Private Sub cmdAdd_Click ()  
    '[语句]  
    optButton (n).Top = optButton (n-1).Top + 400  
    '[语句]  
End Sub
```

提示 如果不只一次使用一个属性值，而且将这个值存储到一个变量中，则代码执行起来会更快。

用方法来执行动作

方法能够影响属性值。例如在用收音机打比方的例子中，`SetVolume` 方法改变了 `Volume` 属性。与此类似，在 `Visual Basic` 中列表框具有 `List` 属性，

而 `Clear` 和 `AddItem` 方法可以改变这一属性。

在代码中使用方法

在代码中使用方法时如何书写语句，这取决于该方法要求多少参数，以及是否返回一个值。如果方法不要求参数，则用以下语法编写代码：

`object.method`

下例中，用 `Refresh` 方法重画图片框：

```
Picture1.Refresh           ' 强迫重画控件。
```

有些方法，如上面的 `Refresh`，既无参数又不返回值。

如果方法要用多个参数，就用逗号将它们分开。例如，`Circle` 方法就要用代表窗体上圆的位置、半径和颜色的参数：

```
' 画一个半径为 1200 缇的蓝色圆。
```

```
Form1.Circle (1600, 1800), 1200, vbBlue
```

如果要保存方法的返回值，就必须把参数用括号括起来。例如，`GetData` 方法从剪贴板返回一张图片：

```
Picture = Clipboard.GetData (vbCFBitmap)
```

如果没有返回值，则参数不会出现在括号中。例如，`AddItem` 方法没有返回值。

```
List1.AddItem "yourname"      ' 在列表框中添加 "yourname"。
```

详细信息 关于 `Visual Basic` 对所有方法提供的语法和参数，请参阅《`Visual`

Basic 6.0 语言参考手册》。

对象之间的关系

在同一窗体上放置两个命令按钮时，它们是两个具有不同 `Name` 属性设置值（`Command1` 和 `Command2`）的分开对象，但都分享同一个类——命令按钮。

它们也共享这样一个特征，即它们在同一窗体上。在本章前面已看到，窗体上的控件也包含在窗体中。这就把控件放在一个层次上。要引用控件就必须先引用窗体，就如同打电话拨号一样，首先要拨国家/地区号码或区域号码，然后再拨具体的电话号码。

这两个命令按钮也共享这样一个特征，即它们都是控件。所有控件都具有公共特性，使它们不同于窗体以及 `Visual Basic` 环境中的其它对象。以下章节中将说明 `Visual Basic` 如何用集合使相关的对象分组。

对象的层次

对象层次提供了一种组织结构，它决定了对象间的相互关系以及访问它们的方法。在大多数情况下不必考虑 `Visual Basic` 对象的层次。但是：

- 在操作其它应用程序的对象时，应当熟悉那个应用程序的对象层次。关于定位对象层次的信息，请参阅第十章“用部件编程”。
- 在使用数据访问对象时，应当熟悉数据访问对象层次。

在 Visual Basic 中，一个对象包含其它对象的一般情况时有发生。下面我们要详细讨论。

使用对象集合

集合对象有自己的属性和方法。对象集合中的对象作为集合的成员被引用。集合中的每个成员从 0 开始顺次编号；这就是成员的索引号。例如，控件集合包含已给定窗体上的所有控件，如图 5.10 所示。如果对集合中的所有对象都必须执行相同操作，则可用集合简化代码。

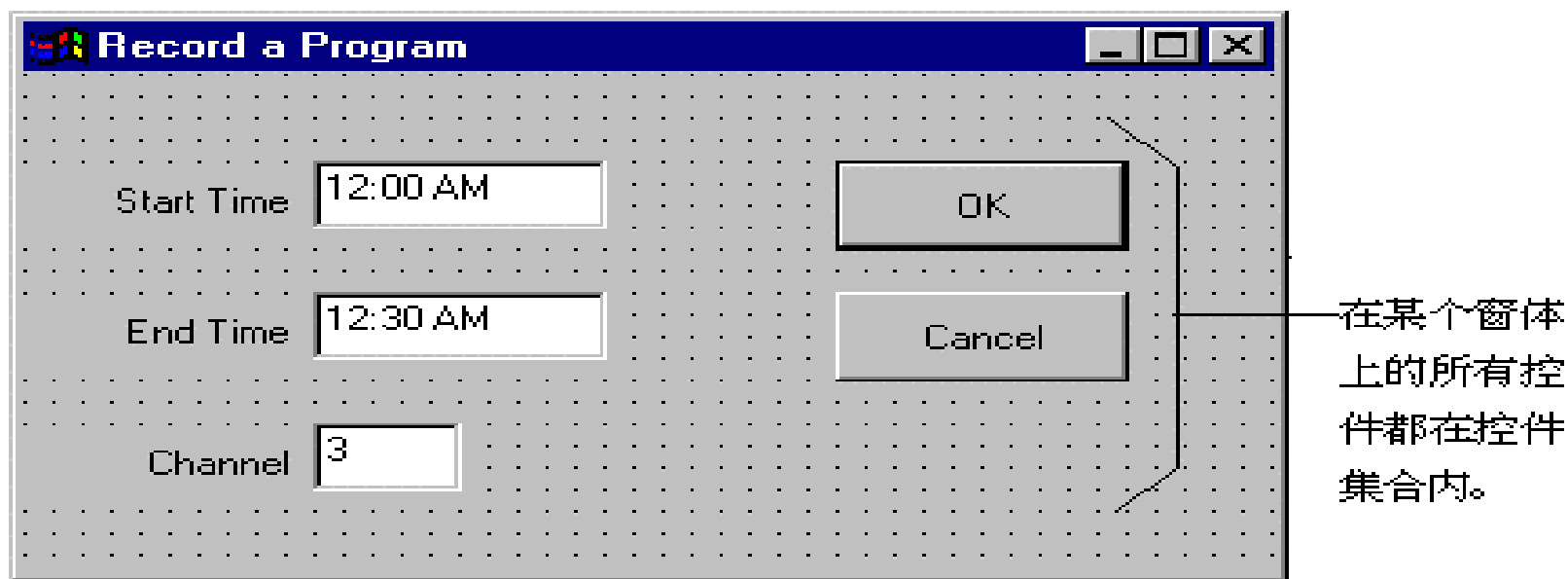


图 5.10 控件集合

例如，下列代码遍历控件集合，并在列表框中列出每个成员的名字。

```
Dim MyControl as Control
For Each MyControl In Form1.Controls
    '把每个控件的名字送到列表框。
    List1.AddItem MyControl.Name
Next MyControl
```

应用集合成员的属性与方法

有两种通用方法可用来对集合对象成员寻址：

- 指定成员的名称。以下两个表达式等价：

```
Controls ("List1")
```

```
Controls!List1
```

- 利用成员的索引号：

```
Controls (3)
```

一旦能够从整体上对全体成员寻址，而且还能够个别地对单个成员寻址，就可用下面随便一种方式应用属性和方法：

'将 ListBox 控件的 Top 属性设置为 200。

```
Controls!List1.Top = 200
```

— 或者 —

```
Dim MyControl as Control
```

```
For Each MyControl In Form1.Controls ()
```

'将每一个成员的 Top 属性设置为 200。


```
MyControl.Top = 200
```

```
Next MyControl
```

包含其它对象的对象

在 Visual Basic 中，一些对象包含其它对象。例如，窗体通常包含一个或数个控件。把一个对象当作其它对象的容器，这样做的好处在于，在代码中引用容器即可明了要使用哪个对象。例如，图 5.11 说明在一个应用程序中可有两个不同的窗体，一个用来输入应付帐交易，另一个用来输入应收帐交易。

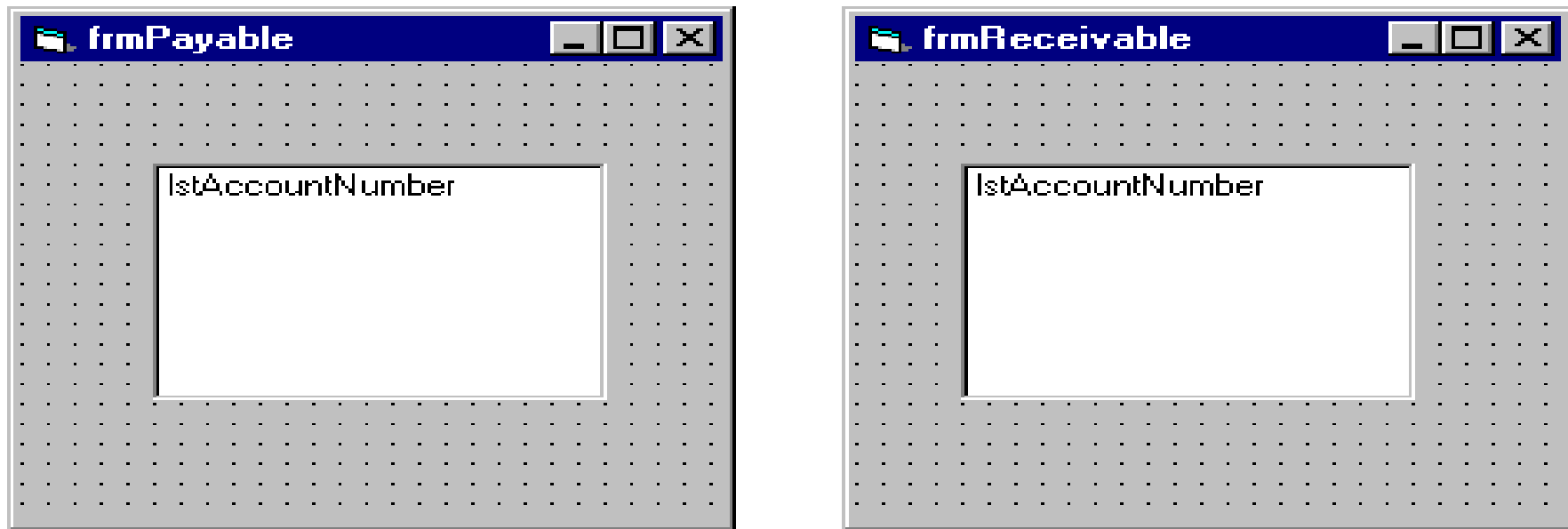


图 5.11 两个不同窗体可以包含同名的控件

两个窗口可以有名为 `IstAcctNo` 的列表框。通过引用包含列表框的窗体就可准确指定要使用的列表框：

```
frmReceivable.lstAcctNo.AddItem 1201
```

— 或者 —

```
frmPayable.lstAcctNo.AddItem 1201
```

Visual Basic 中的公共集合

在 Visual Basic 中一个对象包含其它对象的一般情况时有发生。下表简要叙述 Visual Basic 中最常用的集合。

集合	描述
Forms	包含载入的窗体
Controls	包含窗体上的控件
Printers	包含可用的打印机对象

在 Visual Basic 中也可实现对象控制。

详细信息 关于对象容器的详细信息，请参阅第八章“再论编程”中的“使用集合替代数组”。关于打印机集合的信息，参阅第十二章“使用文本和图形”。关于窗体集合和控件集合的详细信息，请参阅《Visual Basic 6.0 语言参考手册》。

容器属性

在窗体内，可用 `Container` 属性改变对象的容器。以下控件可包含其它控件：

- `Frame` 控件
- `PictureBox` 控件
- `ToolBar` 控件（仅对专业版和企业版而言）

如何在窗体上将命令按钮从一个容器移动到另一个容器，该例对此作了演示。打开新工程，并在窗体上画出 `Frame` 控件、`PictureBox` 控件和 `CommandButton` 控件。

下列代码在窗体单击事件中，这些代码使计数器变量增加，并通过 `Select Case` 循环使命令按钮在容器之间移动。

```
Private Sub Form_Click ()  
    Static intX as Integer  
    Select Case intX  
        Case 0  
            Set Command1.Container = PictureBox1  
            Command1.Top= 0  
            Command1.Left= 0  
  
        Case 1  
            Set Command1.Container = Frame1
```

```
Command1.Top= 0  
Command1.Left= 0  
  
Case 2  
Set Command1.Container = Form1  
Command1.Top= 0  
Command1.Left= 0  
  
End Select  
intX = intX + 1
```

```
End Sub
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的“Container 属性”。

对象之间的通讯

除了在 Visual Basic 中使用和创建对象外，还可与其它应用程序通讯并在自己的应用程序中操作其它应用程序的对象。在应用程序之间共享数据的能力是 Microsoft Windows 操作系统的关键性能之一。有了 Visual Basic，与其它应用程序进行通讯变得极为灵活。

详细信息 关于使用其它应用程序的对象及其通讯的详细信息，请参阅第十章“用部件编程”。

创建对象

创建对象的最简单方法是在“工具箱”中双击控件。但是，对 Visual Basic 中的所有可用对象以及来自其它应用程序的所有可用对象，要想充分体现它们的优势，可在运行时用 Visual Basic 的可编程性能创立对象。

- 用对象变量创建对象引用
- 用类模块“从零开始”创建自己的对象
- 用集合对象创建自己的集合

详细信息 在其它章节将会介绍如何访问对象。例如，在第十章“用部件编程”中将讨论 CreateObject 和 GetObject 函数。

使用对象变量

除了存储值以外，变量可引用对象。如同给变量赋值一样，基于下列相同理由，可把对象赋给变量：

- 变量名往往比变量所含的值（或者，在此情况下，所含值就是它们引用的对象）短而好记。
- 在运行代码时，为了引用其它对象，应改变变量。
- 引用包含对象的变量比反复引用对象本身有更高的效率。

使用对象变量与使用常规变量一样，只不过多了把对象赋予变量这一步：

- 首先声明变量：

```
Dim variable As class
```

- 然后把对象赋予变量：

```
Set variable = object
```

声明对象变量

声明对象变量的方法和声明其它变量一样，要用 `Dim`、`ReDim`、`Static`、`Private` 和 `Public`。仅有的不同在于可选的 `New` 关键字和 `Class` 参数，本章后面将会对 `New` 关键字和 `Class` 参数都作介绍。语法是：

```
{Dim | ReDim | Static | Private | Public} variable As [New] class
```

例如，可以声明一个对象变量，它引用应用程序中的被称做 `frmMain` 的窗体：

```
Dim FormVar As New frmMain '声明一个类型为 frmMain  
                             '的对象变量。
```

也可以声明一个对象变量，它引用应用程序中的任何窗体：

```
Dim anyForm As Form '一般窗体变量。
```

类似地，可以声明一个对象变量，它能够引用应用程序中的任何文本框：

```
Dim anyText As TextBox '可以引用任何文本框  
                        '(但只能是一个文本框)。
```

还可以声明一个对象变量，它能够引用任何类型的控件：

```
Dim anyControl As Control '一般控件变量。
```

注意，可以声明一个引用应用程序中的指定窗体的窗体变量，但不能声明

引用特定控件的控件变量。可以声明能够引用控件的特定类型（如 `TextBox` 或列表框）的控件变量，但不是引用那种类型的一个特定控件（如 `txtEntry` 或 `List1`）。但是，可把特定的控件赋予那种类型的变量。例如，对于具有列表框 `lstSample` 的窗体可以这样编码：

```
Dim objDemo As ListBox
Set objDemo = lstSample
```

赋值对象变量

用 `Set` 语句将对象赋予对象变量：

```
Set variable = object
```

无论何时，只要希望对象变量引用对象，就可使用 `Set` 语句。有时可用对象变量，尤其可用控件变量直接缩短要键入的代码。例如，原来的代码是这样的：

```
If frmAccountDisplay!txtAccountBalance.Text < 0 Then
    frmAccountDisplay!txtAccountBalance.BackColor = 0
    frmAccountDisplay!txtAccountBalance.ForeColor = 255
End If
```

如果使用控件变量，程序就可显著缩短：

```
Dim Bal As TextBox
Set Bal = frmAccountDisplay!txtAccountBalance
If Bal.Text < 0 Then
```

```
Bal.BackColor = 0  
Bal.ForeColor = 255
```

```
End If
```

特定对象类型与一般对象类型

特定的对象变量必须引用特定类型的对象或类。特定的窗体变量只能引用应用程序中的一个窗体（尽管可在该窗体的诸多实例中引用其中之一）。与此相似，特定的控件变量只能引用应用程序中的特定类型的控件，比如 **TextBox** 或者列表框。请看一例，打开新的工程，在窗体中放置一文本框。将下列代码添加到窗体中：

```
Private Sub Form_Click()  
    Dim anyText As TextBox  
    Set anyText = Text1  
    anyText.Text = "Hello"  
End Sub
```

运行应用程序并单击窗体。文本框的 **Text** 属性将变成 "Hello"。

一般对象变量可在多种特定对象类型中引用其中一种。例如，一般窗体变量可引用应用程序中的任何窗体；一般控件变量可引用应用程序中任何窗体上的任何控件。请看一例，打开新的工程并在窗体中按任意顺序放置若干 **Frame**、**Label** 和 **CommandButton** 控件。将下列代码添加到窗体中：

```
Private Sub Form_Click()
```



```
Dim anyControl As Control
Set anyControl = Form1.Controls (3)
anyControl.Caption = "Hello"
```

End Sub

运行程序并单击窗体。刚才放入窗体中的第三个控件的标题将变成 "Hello"。

Visual Basic 中有四种一般对象：

一般对象类型	可引用的对象
Form	应用程序中的任何窗体（包括 MDI 子窗体和 MDI 窗体）
Control	应用程序中的任何控件
MDIForm	应用程序中的 MDI 窗体（如果应用程序有 MDI 窗体的话）
Object	任何对象

在不知道变量运行时是否引用特定类型的对象时，一般对象变量是很有用的。例如，如果要书写代码，并使之在应用程序的任何窗体中都可操作，则必须使用一般窗体变量。

注意 由于应用程序中只能有一个 MDI 窗体，因此不必使用一般的 MDIForm 类型。无论何时，一旦需要声明引用 MDI 窗体的窗体变量，就可用特定的 MDIForm 类型(或者是 MDIForm1,或者是对 MDI 窗体的 Name 属性指定的任何名称)，而不用一般的 MDIForm 类型。实际上，由于在运行应用程序前，Visual Basic 就能够决定引用特定窗体类型的属性和方法，因此总应使用特定的 MDIForm 类型。

只是出于完整性的考虑才提供一般的 MDIForm 类型；在 Visual Basic 的

今后版本中，单个应用程序中可以拥有多个 MDI 窗体，这将是非常有用的。

作为对象的窗体

窗体最常用到的是构造应用程序界面，但也常常是应用程序中其它模块调用的对象。窗体与类模块关系密切。两者之间的主要区别在于，窗体是可视对象，而类模块却没有可视界面。

添加自定义方法和属性

可对窗体添加自定义方法和属性，并可从应用程序中的其它模块访问它们。为了创建窗体的新方法，请添加用 Public 声明的过程。

'自定义 Form1 的方法。

```
Public Sub LateJobsCount ()
```

```
    .
```

```
    . ' <statements>
```

```
    .
```

```
End Sub
```

可用下面这个语句从其它模块调用 LateJobsCount 过程：

```
Form1.LateJobsCount
```

创建窗体的新属性就像在窗体模块中声明一个公用变量那样简单：

```
Public IDNumber As Integer
```

可用下面两个语句从其它模块设置和返回 Form1 的 IDNumber 值：

```
Form1.IDNumber = 3
```

```
Text1.Text = Form1.IDNumber
```

也可用 Property 过程向窗体添加自定义属性。

详细信息 第九章“用对象编程”提供了 Property 过程的详细材料。

注意 不必加载窗体就可调用窗体的变量或自定义方法，还可设置窗体的自定义属性。于是，不必将窗体加载到内存就可运行有关窗体的代码。同样，引用控件而不引用其属性或方法时也不会加载窗体。

使用 New 关键字

用 New 关键字创建新对象，这个对象被视为它的类所定义的对象。可用 New 创建集合、窗体的实例以及类模块中定义的类。

在窗体上使用 New 关键字

在设计时创建的窗体是类。可用 New 关键字创建这个类的新实例。为了看到这一工作过程，请在窗体上画出一个命令按钮和其它几个控件。在“属性”窗口中将窗体的 Name 属性设置成 Sample。在命令按钮的 Click 事件过程中添加下列代码：

```
Dim x As New Sample
```

```
x.Show
```

运行应用程序，并单击命令按钮数次。把最前面的窗体移到一边。因为窗体是具有可视界面的类，所以可看到附加副本。每个窗体在相同位置上都有相同控件，而且这些位置也正是窗体在设计时的位置。

注意 要使窗体变量和已加载窗体的实例一直存在，请用 `Static` 或 `Public` 变量，而不要用局部变量。

也可将 `New` 关键字与 `Set` 语句并用。请试用命令按钮的 `Click` 事件过程中的代码：

```
Dim f As Form1
Set f = New Form1
f.Caption = "hello"
f.Show
```

将 `New` 关键字和 `Set` 语句并用会加快运行速度，因此，这种方法值得推荐。

对其它对象使用 `New` 关键字

可用 `New` 关键字从类模块中定义的类创建集合和对象。试用下例说明这一工作过程。

此例说明了 `New` 关键字是如何创建类的实例的。打开新工程，在 `Form1` 上画出一个 `CommandButton` 控件。从“工程”菜单中选择“添加类模块”命令，从而在工程中添加一个类模块。设置类模块的 `Name` 属性为 `ShowMe`。

`Form1` 中的下列代码创建了类 `ShowMe` 的新实例，还调用了包含在类模块

中的过程。

```
Public clsNew As ShowMe  
Private Sub Command1_Click()  
    Set clsNew = New ShowMe  
    clsNew.ShowFrm  
End Sub
```

类模块中的 `ShowFrm` 过程创建类 `Form1` 的新实例，显示窗体，然后使其最小化。

```
Sub ShowFrm()  
    Dim frmNew As Form1  
    Set frmNew = New Form1  
    frmNew.Show  
    frmNew.WindowState = 1  
End Sub
```

为了应用这个例子，请运行应用程序并单击命令按钮数次。在创建 `ShowMe` 类的每一个新实例时将看到最小化的窗体图标出现在桌面上。

详细信息 关于利用 `New` 创建对象的信息，请参阅第十章“用部件编程”。

New 关键字的限制

下表说明不能用 `New` 关键字做什么。

不能 用 New 创建	不允许的代码的例子
基本数据类型的变量	Dim X As New Integer
任何一般对象类型的变量	Dim X As New Control
任何特定控件类型的变量	Dim X As New ListBox
任何特定控件的变量	Dim X As New lstNames

释放对对象的引用

每个对象都使用内存和系统资源。当不再使用对象时及时释放这些资源，这是良好的编程习惯。

- 用 Unload 从内存中卸载窗体或控件。
- 用 Nothing 释放对象变量占用的资源。用 Set 语句将 Nothing 赋予对象变量。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的“Unload 事件”和“Nothing”部分。

给过程传递对象

可在 Visual Basic 中将对象传递给过程。在下面的代码例子中假定窗体中有一 CommandButton 控件：

```
Private Sub Command1_Click ()
```

```
    '调用 Demo 子过程，并将窗体传递给它。
```

```

    Demo Form 1
End Sub

Private Sub Demo (x As Form 1)
    '在屏幕中使窗体居中。
    x.Left = (Screen.Width - x.Width) / 2
End Sub
```

可以通过引用把对象传递给参数，然后在过程内将参数设置成新对象。为了看到这一工作过程，打开工程并插入第二个窗体。在每个窗体中安放一个图片框。需要改变的属性设置值如下表所示：

对象	属性	设置
Form2 上的 图片框	Name	Picture2
	Picture	c:\vb\icons\arrows\arw01dn.ico

Form1_Click 事件过程调用 Form2 中的 GetPicture 过程，并把一个空的图片框传递给它。

```

Private Sub Form_Click ()
    Form2.GetPicture Picture1
End Sub
```

Form2 中的 GetPicture 过程把 Form2 上的图片框的 Picture 属性赋予 Form1 上的空图片框。

```

Private objX As PictureBox
```

```
Public Sub GetPicture (x As PictureBox)
    '将传递进来的图片框赋予对象变量。
    Set objX = x
    '将 Picture 属性值赋予 Form1 的图片框。
    objX.Picture = picture2.Picture
End Sub
```

为了应用这个例子，请运行程序并单击 Form1。在 Form1 的图片框中将看到出现了 Form2 中的图标。

详细信息 上述主题意在概述对象。更详细内容，请参阅第九章“用对象编程”和第十章“用部件编程”。

第二部分 Visual Basic 能做什么

一旦了解了 Visual Basic 的基础，就要准备迎接新的更大挑战。那么，到底能用 Visual Basic 干什么呢？也许应该问不能用它干什么更恰当一些。答案是：没有什么不能干的！从设计新型的用户界面到利用其它应用程序的对象、从处理文字图像到使用数据库，Visual Basic 提供了完成这些工作的所有工具。

《程序员指南》的第一部分介绍基本概念；第二部分介绍用 Visual Basic 开发由简单到复杂的应用程序所用到的构件。

第六章 创建用户界面

深入介绍用户界面的样式、窗体、菜单、工具栏等等。

第七章 使用 Visual Basic 的标准控件

介绍 Visual Basic 的内部控件及用法。

第八章 再论编程

进一步讨论 Visual Basic 编程的概念、技巧和附加工具。

第九章 用对象编程

深入介绍对象，包括创建类及使用 `Active X` 设计工具。

第十章 用部件编程

介绍在应用程序中使用 `ActiveX` 部件编程的技巧。

第十一章 响应鼠标和键盘事件

讨论包括拖放在内的用户输入的处理。

第十二章 使用文本和图形

介绍使用 `Visual Basic` 的文本和图形方法来显示和打印。

第十三章 调试代码和处理错误

说明在设计时或运行时，如果应用程序没有按预计的设计运行，则应如何处理。

第十四章 处理驱动器、文件夹和文件

使用文件系统的技巧。

第十五章 性能和兼容性设计

介绍使应用程序更快更小的优化技巧。

第十六章 国际化

对不同语言和文化背景下的应用程序进行的思考。

第十七章 发布应用程序

介绍如何使用打包和部署向导 (Package and Deployment Wizard) 来创建安装程序。

第六章 创建用户界面

用户界面是一个应用程序最重要的部分，它是最直接的现实世界。对用户而言，界面就是应用程序，它们感觉不到幕后正在执行的代码。不论花多少时间和精力来编制和优化代码，应用程序的可用性仍然依赖于界面。

设计一个应用程序时，需要做出有关界面的若干决定。应该使用单文档还是多文档样式？需要多少不同的窗体？菜单中将包含什么命令？要不要使用工具栏重复菜单的功能？提供什么对话框与用户交互？需要提供多少帮助？

在开始设计用户界面之前，需要考虑应用程序的目的。经常使用的主要应用程序，其设计应该与只是偶尔使用的不同。用来显示信息的应用程序与用来收集信息的应用程序的需求不同。

预期的用户也应该影响设计。目标是针对初学者的应用程序，它的设计要求简单明了，而针对有经验的用户却可以复杂一些。读者所使用过的其它应用程序也会影响他们对应用程序操作方式的期望。如果计划发布到全球，那么语言与文化也是设计所必须考虑的部分。

用户界面的设计最好是作为一个反复过程进行，很难在第一遍就能提出一个完美的设计。本章将介绍在 Visual Basic 中设计用户界面的过程，同时介绍在为用户创建重要应用程序时需用的一些工具。

目 录

- 界面样式
- 多文档界面 (MDI) 的应用程序
- 再论窗体
- 在应用程序中使用菜单
- 工具栏
- 对话框
- 各种显示类型的设计
- 设计时心里想着用户

示例程序：Mdinote.vbp 和 Sdinote.vbp

本章的许多代码示例都取自 Mdinote.vbp 和 Sdinote.vbp 示例程序，示例应用程序列在 Samples!Alink("vbsamples") 目录中。

界面样式

如果使用过基于 Windows 的应用程序，也许已经注意到不是所有用户界面看上去或操作起来都一样。用户界面样式主要有两种：单文档界面 (SDI) 和多文档界面 (MDI)。SDI 界面的一个示例就是 Microsoft Windows 中的

WordPad（记事本）应用程序（图 6.1）。在 WordPad 中，只能打开一个文档，想要打开另一个文档时，必须先关上已打开的文档。

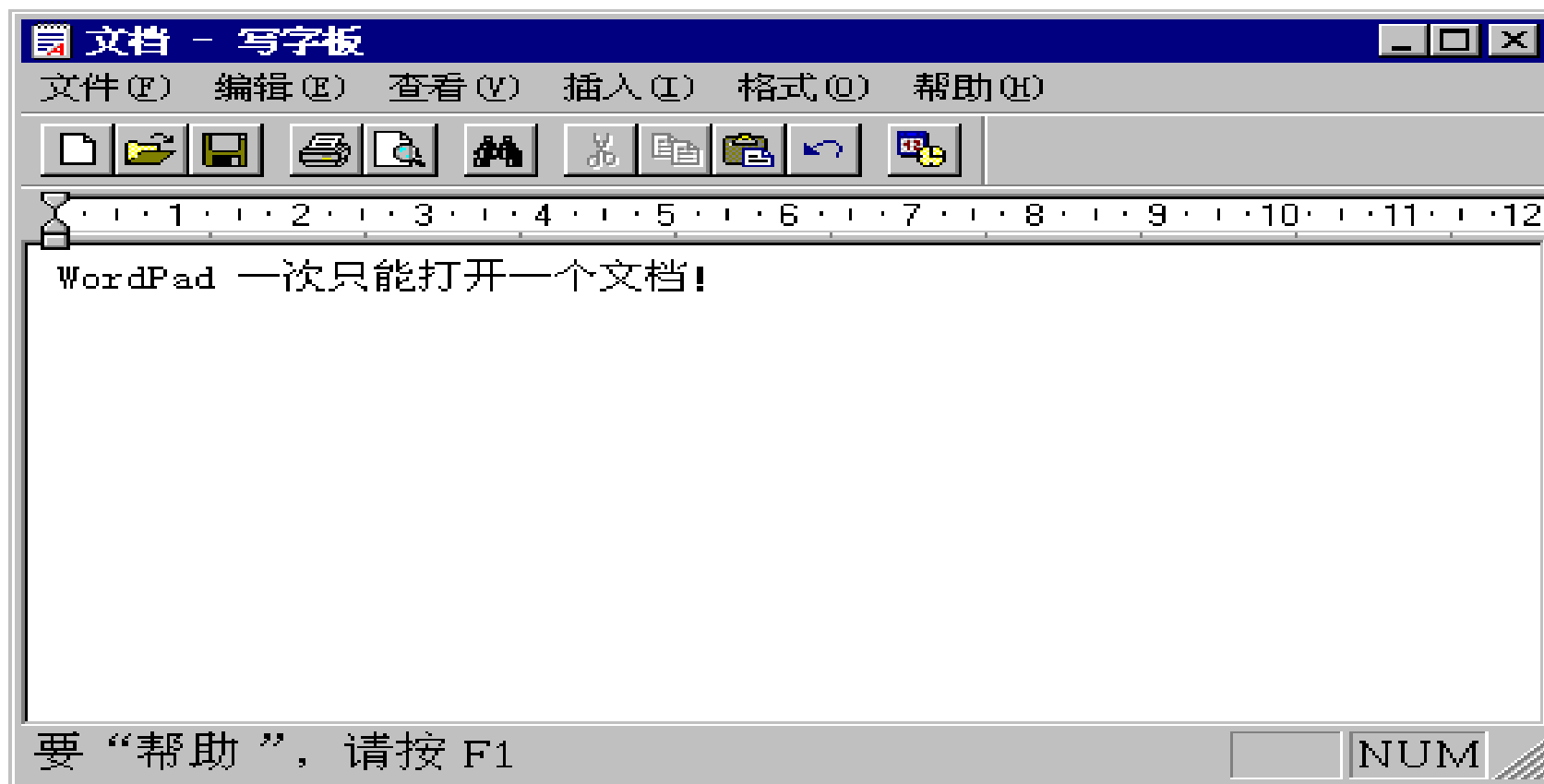


图 6.1 记事本，一个单文档界面 (SDI) 应用程序

象 Microsoft Excel 和 Microsoft Word for Windows 这样的应用程序就是 MDI 界面；它们允许同时显示多个文档，每一个文档都显示在自己的窗口中（图 6.2）。按照是否包含有能在窗口或文档之间进行切换的子菜单的

Windows 菜单项，就能识别出 MDI 应用程序。

要决定哪种界面样式最好，需要看应用程序的目的。一个处理保险索赔的应用程序可能要用多文档界面 MDI 样式，一个职员很可能会同时处理一个以上索赔，或者需要对两个索赔进行比较。另一方面，日历程序最好设成 SDI（单文档界面）样式，因为不大需要同时打开一个以上日历；在极少的情况下，可以再打开一个单文档界面应用程序的实例。

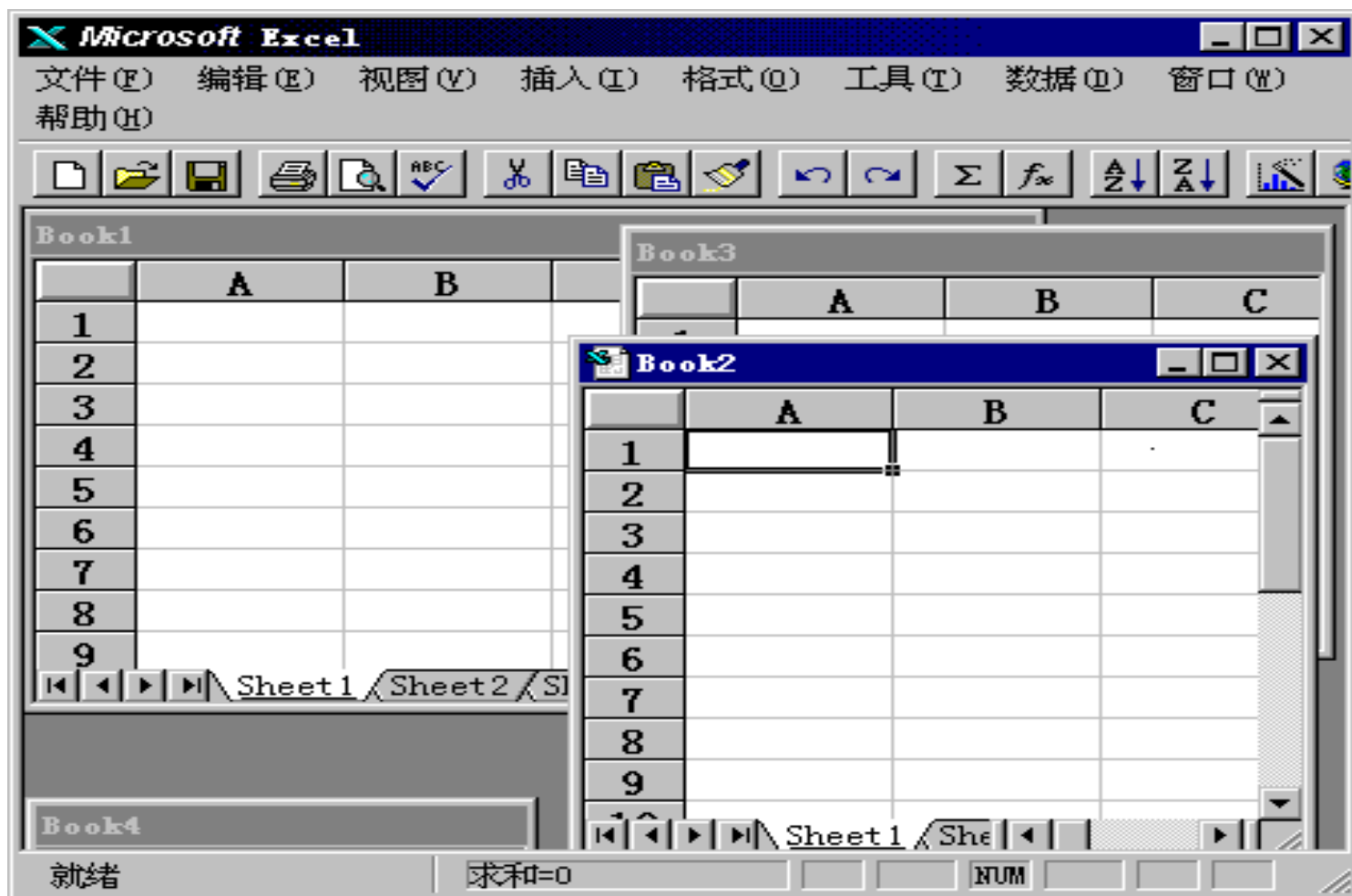


图 6.2 Microsoft Excel, 一个多文档界面 (MDI) 应用程序

SDI 样式更常用,《程序员指南》中的大多数实例都采用 SDI 应用程序。关于创建 MDI 应用程序有一些需要单独考虑的问题和技巧,放在本章后面“多文档界面 (MDI) 应用程序”一节。

除了以上两个最常用的界面样式 SDI 与 MDI 外，第三种界面样式变得越来越流行，这就是资源管理器样式的界面（图 6.3）。资源管理器样式界面是包括有两个窗格或者区域的一个单独的窗口，通常是由左半部分的一个树型的或者层次型的视图和右半部分的一个显示区所组成，如在 Microsoft Windows 的资源管理器中所见到的那样。这种样式的界面可用于定位或浏览大量的文档、图片或文件。

除了本章所附的 MDI 与 SDI 应用程序示例，实用程序向导还提供了一种比较界面样式差别的方法。可以用向导来为每一种样式生成一个框架，然后观察它生成的窗体与代码。



图 6.3 Windows 的资源管理器，资源管理器样式的界面

详细信息 要了解更多的关于 MDI 应用程序的知识，请参阅“多文档界面 (MDI) 应用程序”一节。有关窗体使用的基础，请参阅第三章“窗体、控件与菜单”。有关访问实用程序向导的信息，请参阅第四章“工程的管理”中“使用向导和外接程序”一节。

多文档界面 (MDI) 应用程序

多文档界面 (MDI) 允许创建在单个容器窗体中包含多个窗体的应用程序。像 Microsoft Excel 与 Microsoft Word for Windows 这样的应用程序就具有多文档界面。

MDI 应用程序允许用户同时显示多个文档，每个文档显示在它自己的窗口中。文档或子窗口被包含在父窗口中，父窗口为应用程序中所有的子窗口提供工作空间。例如：Microsoft Excel 允许创建并显示不同样式的多文档窗口。每个子窗口都被限制在 Excel 父窗口的区域之内。当最小化 Excel 时，所有的文档窗口也被最小化，只有父窗口的图标显示在任务栏中。

子窗体就是 MDIChild 属性设置为 True 的普通窗体。一个应用程序可以包含许多相似或者不同样式的 MDI 子窗体。

在运行时，子窗体显示在 MDI 父窗体工作空间之内（其区域在父窗体边框以内及标题与菜单栏之下）。当子窗体最小化时，它的图标显示在 MDI 窗体的工作空间之内，而不是在任务栏中，如图 6.4 所示。



图 6.4 显示在 MDI 窗体工作空间之内的子窗体

注意 一个应用程序也可以包括标准的、不是包含在 MDI 窗体之内的非 MDI 窗体。MDI 应用程序中标准窗体的典型用法是显示模式的对话框。

MDI 窗体类似于具有一个限制条件的普通窗体。除非控件具有 `Align` 属性（如 `PictureBox` 控件）或者具有不可见界面（如 `Timer` 控件），否则不能

将控件直接放置在 MDI 窗体上。

创建 MDI 应用程序

使用下列步骤来创建 MDI 窗体及其子窗体。

要创建 MDI 应用程序，请按照以下步骤执行：

1. 创建 MDI 窗体。

从“工程”菜单中选取“添加 MDI 窗体”。

注意 一个应用程序只能有一个 MDI 窗体。如果工程已经有了一个 MDI 窗体，则该“工程”菜单上的“添加 MDI 窗体”命令就不可使用。

2. 创建应用程序的子窗体。

要创建一个 MDI 子窗体，先创建一个新窗体（或者打开一个存在的窗体），然后把它的 MDIChild 属性设为 True。

设计时使用 MDI 子窗体

在设计时，子窗体不是限制在 MDI 窗体区域之内。可以添加控件、设置属性、编写代码以及设计子窗体功能，就像在其它 Visual Basic 窗体中做的那样。

通过查看 MDIChild 属性或者检查工程资源管理器，可以确定窗体是否是一个 MDI 子窗体。如果该窗体的 MDIChild 属性设置为 True，则它是一个子窗体。Visual Basic 在“工程资源管理器”中为 MDI 窗体与 MDI 子

窗体显示了特定的图标，如图 6.5 所示。

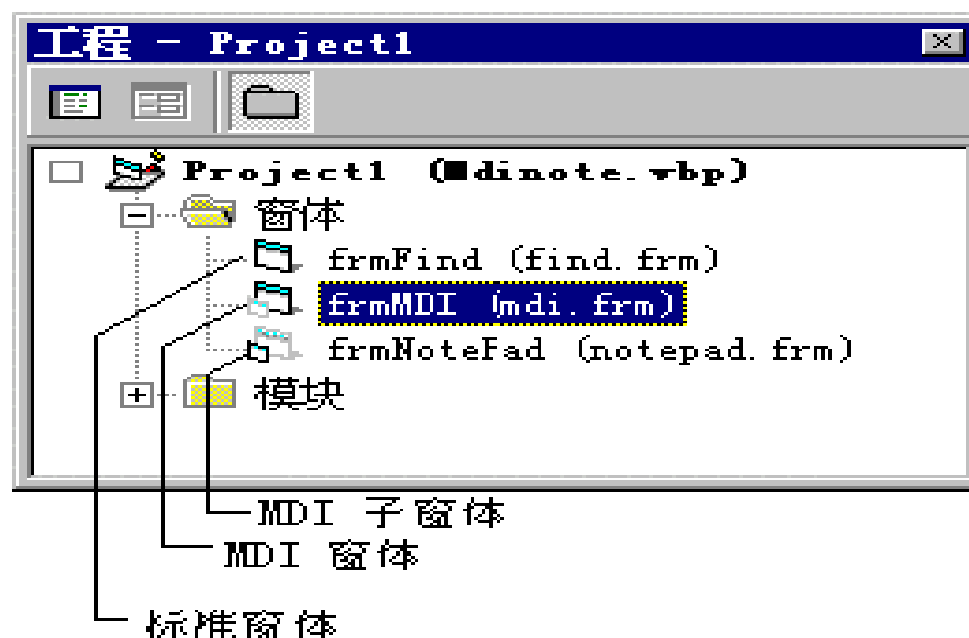


图 6.5 工程资源管理器中的图标标明 MDI 子窗体、标准窗体和 MDI 窗体

MDI 窗体运行时的特性

在运行时，MDI 窗体及其所有的子窗体都呈现特定的性质：

- 所有子窗体均显示在 MDI 窗体的工作空间内。像其它的窗体一样，用户能移动子窗体和改变子窗体的大小，不过，它们被限制于这一工作空间内。

- 当最小化一个子窗体时，它的图标将显示于 MDI 窗体上而不是在任务栏中。当最小化 MDI 窗体时，此 MDI 窗体及其所有子窗体将由一个图标来代表。当还原 MDI 窗体时，MDI 窗体及其所有子窗体将按最小化之前的状态显示出来。
- 当最大化一个子窗体时，它的标题会与 MDI 窗体的标题组合在一起并显示于 MDI 窗体的标题栏上（参见图 6.6）。

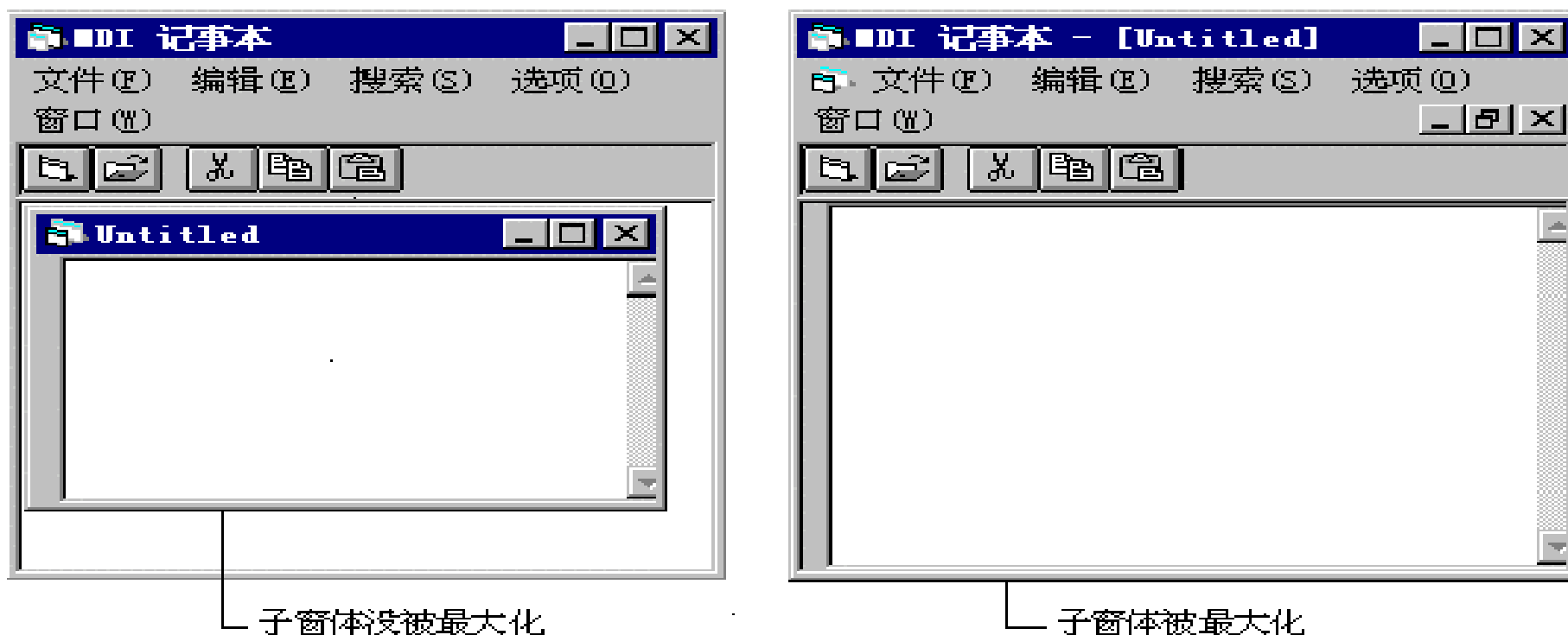


图 6.6 与 MDI 窗体标题相组合的子窗体标题

- 通过设定 `AutoShowChildren` 属性，子窗体可以在窗体加载时自动显示

(True) 或自动隐藏 (False)。

- 活动子窗体的菜单（若有）将显示在 MDI 窗体的菜单栏中，而不是显示在子窗体中。

MDI NotePad 应用程序

MDI NotePad 示例应用程序是一个简单的文本编辑器，它与 Microsoft Windows 中包含的 NotePad 应用程序相似。不过，MDI NotePad 应用程序使用的是一个多文档界面 (MDI)。在运行时，若用户需要一个新文档（用该应用程序的“文件”菜单中的“新建”命令执行），应用程序就会创建子窗体的一个新实例。这就允许按其所需创建多个子窗体或文档。

在 Visual Basic 中为了创建以文档为中心的应用程序，至少需要两个窗体：一个 MDI 窗体和一个子窗体。设计时，应创建一个 MDI 窗体以容纳该应用程序，再创建一个子窗体作为这个应用程序文档的模板。

要创建自己的 MDI NotePad 应用程序，请按照以下步骤执行：

1. 从“文件”菜单中，选取“新建工程”命令。
2. 从“工程”菜单中，选取“添加 MDI 窗体”命令来创建容器窗体。

现在，这个工程应当包含一个 MDI 窗体 (MDIForm1) 和一个标准窗体 (Form1)。

3. 在 Form1 上创建一个文本框 (Text1)。
4. 按下列方式为两个窗体和文本框设置属性。

对 象	属 性	设 置 值
M D IForm 1	Caption	M D I NotePad
Form 1	Caption	无 标 题
	M D IChild	True
Text1	MultiLine	True
	Text	(空 值)
	Left	0
	Top	0

5. 使用“菜单编辑器”(从“工具”菜单中), 为 MDIForm1 创建一个“文件”菜单。

标 题	名 称	缩 进
&File	mnuFile	否
&New	mnuFileNew	是

6. 在 mnuFileNew_Click 过程中增加以下代码:

```
Private Sub mnuFileNew_Click ()
    ' 创建名为 NewDoc 的窗体 Form1 的一个新实例。
    Dim NewDoc As New Form1
    ' 显示此新窗体。
    NewDoc.Show
End Sub
```

这个过程创建并显示 Form1 的名为 NewDoc 的一个新实例 (或其副本)。每当从“文件”菜单中选取“新建”命令时, 将会创建一个与 Form1

完全相同的副本（实例），它包含 Form1 所包含的所有控件和代码。

7. 给 Form1 窗体的 Form_Resize 过程添加以下代码：

```
Private Sub Form_Resize()  
    ' 扩展文本框以放置当前子窗体。  
    Text1.Height = ScaleHeight  
    Text1.Width = ScaleWidth  
End Sub
```

Form_Resize 事件过程的代码，像 Form1 中的所有代码一样，能为 Form1 的每一个实例所共享。当显示窗体的几个副本时，每个窗体都能识别各自的事件。当一个事件出现时，该事件过程的代码就会被调用。由于相同的代码为每个实例所共享，关于调用该代码的窗体是如何引用的，尤其是每个实例都具有相同的名字 (Form1) 时。这个问题将在本章后面的“使用 MDI 窗体及其子窗体”一节中讨论。

8. 按 F5 键可运行该应用程序。

提示 除了本章提到的以外，Mdinote.vbp 示例应用程序还包含许多 MDI 技巧。花一些时间对该示例代码进行全面研究会找到这些技巧。Sdinote.vbp 示例应用程序是同一应用程序转换为 SDI 样式的实现；比较这两个例子可以明白 MDI 和 SDI 技巧的差别。

使用 MDI 窗体及其子窗体

当 MDI 应用程序在一次会话中要打开、保存和关闭几个子窗体时，应当

能够引用活动窗体和保持关于子窗体的状态信息。这个主题描述了一些用来指定活动子窗体或者控件、加载和卸载 MDI 窗体及其子窗体、以及保持子窗体的状态信息的编码技巧。

指定活动子窗体或控件

有时要提供一条命令，它用于对当前活动子窗体上具有焦点的控件进行操作。例如，假设有从子窗体的文本框中把所选文本复制到剪贴板上。在 `Mdinote.vbp` 示例应用程序中，“编辑”菜单上的“复制”项的 `Click` 事件将会调用 `EditCopyProc`，它是把选定的文本复制到剪贴板上的过程。

由于应用程序可以有同一子窗体的许多实例，`EditCopyProc` 需要知道使用的是哪一个窗体。为了指定这一点，使用 MDI 窗体的 `ActiveForm` 属性，该属性可以返回具有焦点的或者最后被激活的子窗体。

注意 当访问 `ActiveForm` 属性时，至少应有一个 MDI 子窗体被加载或可见，否则会返回一个错误。

当一个窗体中有几个控件时，也需要指定哪个控件是活动的。像 `ActiveForm` 属性一样，`ActiveControl` 属性能返回活动子窗体上具有焦点的控件。下边是副本例程的示例，从子窗体菜单、MDI 窗体菜单或者是工具栏按钮上可对其进行调用。

```
Private Sub EditCopyProc ()  
    ' 将选定文本复制到剪贴板上。  
    Clipboard.SetText _
```

```
frmMDI.ActiveForm.ActiveControl.SelText
```

```
End Sub
```

假如，正在编写被多个窗体实例调用的代码，不用窗体标识符访问窗体的控件或属性是一个好办法。例如，用 `Text1.Height` 引用 `Form1` 上文本框的高度，而不是使用 `Form1.Text1.Height`。这样，该代码总是影响当前窗体。

在代码中指定当前窗体的另一种方法是用 `Me` 关键字。用 `Me` 关键字来引用当前其代码正在运行的窗体。当需要把当前窗体实例的引用参数传递给过程时，这个关键字很有用。

详细信息 有关通过 `Dim` 语句用 `New` 关键字来创建多个窗体实例的信息，请参阅第五章“编程基础”中的“变量、常数和数据类型概述”，以及《语言参考》的“`Dim` 语句”。有关通过 `Dim` 语句用 `New` 关键字来创建多个窗体实例的信息，请参阅“编程基础”中的“变量、常数和数据类型概述”，和《语言参考》的“`Dim` 语句”。

加载 MDI 窗体及其子窗体

加载子窗体时，其父窗体（MDI 窗体）会自动加载并显示。而加载 MDI 窗体时，其子窗体并不会自动加载。

在 MDI NotePad 示例中，子窗体是缺省的启动窗体，因而在程序运行时，子窗体和 MDI 窗体两者都会加载。如果在 MDI NotePad 应用程序中改变启动窗体为 `frmMDI`（在“工程属性”的“一般”选项卡上），然后运行应用程序，则只有 MDI 窗体被加载。当从“文件”菜单中选取“新建”命

令时，才会加载第一个子窗体。

`AutoShowChildren` 属性可用于加载隐藏状态的 MDI 子窗口，使它们处于隐藏状态直至用 `Show` 方法把它们显示出来。这就允许在子窗体变成可见之前更新标题、位置和菜单等各种细节。

不能把 MDI 子窗体或者 MDI 窗体显示为模式窗体（用带 `vbModal` 参数的 `Show` 方法）。如果想在 MDI 应用程序中使用模式对话框，可使用 `MDIChild` 属性设置为 `False` 的窗体。

设置子窗体的大小和位置

如果 MDI 子窗体具有大小可变的边框（即 `BorderStyle = 2`），在其装载时，Microsoft Windows 将决定其初始的高度、宽度和位置。边框大小可变的子窗体，其初始大小与位置取决于 MDI 窗体的大小，而不是设计时子窗体的大小。当 MDI 子窗体的边框大小不可变（即 `BorderStyle = 0, 1` 或 `3`）时，则它将用设计时的 `Height` 和 `Width` 属性被载入。

如果设置 `AutoShowChildren` 为 `False`，则在 MDI 子窗体载入以后，把它设为可见状态之前，可以改变其位置。

详细信息 请参阅《语言参考》的“`AutoShowChildren` 属性”和“`Show` 方法”。

维护子窗体的状态信息

在用户决定退出 MDI 应用程序时，必须有保存信息的机会。为了使其能

够进行，应用程序必须随时都能确定自上次保存以来子窗体中的数据是否有改变。

通过在每个子窗体中声明一个公用变量来实现此功能。例如，可以在子窗体的声明部分声明一个变量：

```
Public boolDirty As Boolean
```

Text1 中的文本每改变一次时，子窗体文本框的 Change 事件就会将 boolDirty 设置为 True。可添加此代码以指示自上次保存以来 Text1 的内容已经改变。

```
Private Sub Text1_Change ()
```

```
    boolDirty = True
```

```
End Sub
```

反之，用户每次保存子窗体的内容时，文本框的 Change 事件就将 boolDirty 设置为 False，以指示 Text1 的内容不再需要保存。在下列代码中，假设有一个叫做“保存”(mnuFileSave)的菜单命令和一个用来保存文本框内容的名为 FileSave 的过程：

```
Sub mnuFileSave_Click ()
```

```
    '保存 Text1 的内容。
```

```
    FileSave
```

```
    '设置状态变量。
```

```
    boolDirty = False
```

```
End Sub
```

用 QueryUnload 卸载 MDI 窗体

当用户决定退出应用程序时，boolDirty 标志就很有用了。当用户从 MDI 窗体的“控制”菜单中选取“关闭”，或者从提供的菜单项，例如“文件”菜单上的“退出”命令，上述情况就会出现。如果用户使用 MDI 窗体的“控制”菜单来关闭应用程序，Visual Basic 将试图卸载 MDI 窗体。

当 MDI 窗体被卸载时，QueryUnload 事件首先为 MDI 窗体调用，然后再为每一个打开的子窗体调用。如果在这些 QueryUnload 事件过程中没有代码，则取消 Unload 事件，然后，每一个子窗体被卸载，最后，MDI 窗体被卸载。

由于 QueryUnload 事件在窗体卸载之前被调用，因此在窗体卸载前可以给用户一个机会来保存窗体。下述代码使用 boolDirty 标志来决定是否要提醒用户在子窗体卸载之前进行保存。请注意，可以在工程的任何地方访问公共窗体级变量值。这个代码假定有一个名叫 FileSave 的过程，它将 Text1 的内容保存到一个文件中。

```
Private Sub mnuFExit_Click()
```

```
    '当用户在 MDI 应用程序中选取“文件提出”命令时，卸载
```

```
    'MDI 窗体，为每个打开的子窗体调用 QueryUnload 事件。
```

```
    Unload frmMDI
```

```
End
```

```
End Sub
```

```
Private Sub Form_QueryUnload(Cancel As Integer, _  
    UnloadMode As Integer)  
    If boolDirty Then  
        '调用例程来询问用户且必要时保存文件。  
        FileSave  
    End If  
End Sub
```

详细信息 请参阅《语言参考》的“QueryUnload 事件”。

再论窗体

除窗体设计原理以外，还需考虑应用程序的开始与结束。有一些技巧用于决定应用程序启动时的外观。熟悉应用程序卸载时进行的一些处理也很重要。

设置启动窗体

在缺省情况下，应用程序中的第一个窗体被指定为启动窗体。应用程序开始运行时，此窗体就被显示出来（因而最先执行的代码是该窗体的 `Form_Initialize` 事件中的代码）。如果想在应用程序启动时显示别的窗体，那么就得改变启动窗体。

要改变启动窗体，请按照以下步骤执行：

1. 从“工程”菜单中，选取“工程属性”。
2. 选取“一般”。
3. 在“启动对象”ListBox 中，选取要作为新启动窗体的窗体。
4. 选取“确定”。

没有启动窗体时的启动

有时候也许要应用程序启动时不加载任何窗体。例如：可能想先运行装入数据文件的代码，然后再根据数据文件的内容决定显示几个不同窗体中的哪一个。要做到这一点，可在标准模块中创建一个名为 Main 的子过程，如下面的例子所示。

```
Sub Main()  
    Dim intStatus As Integer  
    '调用一个函数过程来检验用户状态。  
    intStatus = GetUserStatus  
    '根据状态显示某个启动窗体。  
    If intStatus = 1 Then  
        frmMain.Show  
    Else  
        frmPassword.Show  
    End If
```

这个过程必须是一个子过程，且不能在窗体模块内。欲将 Sub Main 过程设为启动对象，可从“工程”菜单中，选取“工程属性”，再选“一般”，然后从“启动对象”框中选定“Sub Main”。

显示启动时的快速显示

如果启动时有一个较长的执行过程，例如要从数据库中装入大量数据或者要装入一些大型位图，这时可能希望在启动时给出一个快速显示。快速显示是一种窗体，它通常显示的是诸如应用程序名，版权信息和一个简单的位图等内容。启动 Visual Basic 时所显示的屏幕就是一个快速显示。要显示快速显示，需用 Sub Main 过程作为启动对象，并用 Show 方法显示该窗体：

```
Private Sub Main()  
    '显示快速显示。  
    frmSplash.Show  
    '在此处添加启动过程。  
    ...  
    '显示主窗体并卸载快速显示。  
    frmMain.Show  
    Unload frmSplash  
End Sub
```

当一些启动例程正在执行时，快速显示能吸引用户的注意，造成应用程序

装载很快的错觉。当这些启动例程完成以后，可以装入第一个窗体并使其快速显示卸载。

对于快速显示的设计来说，尽量简单为好。如果使用大量位图或者大量控件，则快速显示本身装入将会变慢。

结束应用程序

当所有窗体都已关闭并且没有代码正在执行时，事件驱动的应用程序就停止运行。如果最后一个可见窗体关闭时仍有隐藏窗体存在，那么，应用程序表现为已经结束了（因为没有可见的窗体），可实际上应用程序仍在继续运行，直至所有隐藏窗体都关闭为止。之所以出现这种情况，是因为对已卸载窗体的属性或控件的任何访问，都将导致隐含地、不予显示地加载那个窗体。

避免出现关闭应用程序时的这类问题，最好的办法是确保所有的窗体都已卸载。如果有一个以上窗体，可以使用 `Forms` 集合和 `Unload` 语句。例如，在主窗体上可以用一个名为 `cmdQuit` 的命令按钮退出程序。如果应用程序只有一个窗体，则 `Click` 事件过程可简单为：

```
Private Sub cmdQuit_Click ()  
    Unload Me  
End Sub
```

如果应用程序使用多窗体，通过把代码放入主窗体的 `Unload` 事件过程可以卸载这些窗体。可以使用 `Forms` 集合确保找到并关闭所有窗体。下列代

码就是使用窗体集合来卸载所有窗体的：

```
Private Sub Form_Unload (Cancel As Integer)
    Dim i as integer
    '在窗体集合中循环并卸载每个窗体。
    For i = Forms.Count - 1 to 0 Step - 1
        Unload Forms (i)
    Next
End Sub
```

有一些情况需要不顾现存窗体或对象的状态而结束应用程序。为此，Visual Basic 提供了 End 语句。

End 语句使应用程序立即结束：在 End 语句之后的代码不会执行，也不会有事件发生。特别是，Visual Basic 将不执行任何窗体的 QueryUnload、Unload 或 Terminate 事件过程。对象的各个引用将被释放，但如果定义有自己的类，Visual Basic 将不执行由这些类所创建对象的 Terminate 事件。

除 End 语句以外，Stop 语句可以暂停一个应用程序。然而，Stop 语句只能在调试时使用，因为它不释放对象的引用。

详细信息 关于 Stop 语句的信息，请参阅第十三章“调试代码和处理错误”中的“使用中断模式”，以及《语言参考》的“Stop 语句”。关于窗体集合或者释放对象引用的信息，请参阅第九章“用对象编程”。

在应用程序中使用菜单

许多简单的应用程序只由一个窗体和几个控件组成，但是通过增加菜单可以增强 Visual Basic 应用程序的功能。这一节将介绍如何在应用程序中创建菜单和使用菜单。

用菜单编辑器创建菜单

用菜单编辑器可以创建新的菜单和菜单栏、在已有的菜单上增加新命令、用自己的命令来替换已有的菜单命令以及修改和删除已有的菜单和菜单栏。

要显示菜单编辑器，请：

- 从“工具”菜单上，选取“菜单编辑器”。
- 或者 -

在“工具栏”上单击“菜单编辑器”按钮。

这就会打开菜单编辑器，如图 6.7 所示。

尽管大多数菜单控件属性可用菜单编辑器设置，所有的菜单属性在“属性”窗口中也都是可用的。菜单控件的两个最重要的属性是：

- **Name**——这是代码中用来引用菜单控件的名字。
- **Caption**——这是出现在控件上的文本。

菜单编辑器中的其它属性，包括 **Index**、**Checked** 和 **NegotiatePosition**，在

本章稍后介绍。

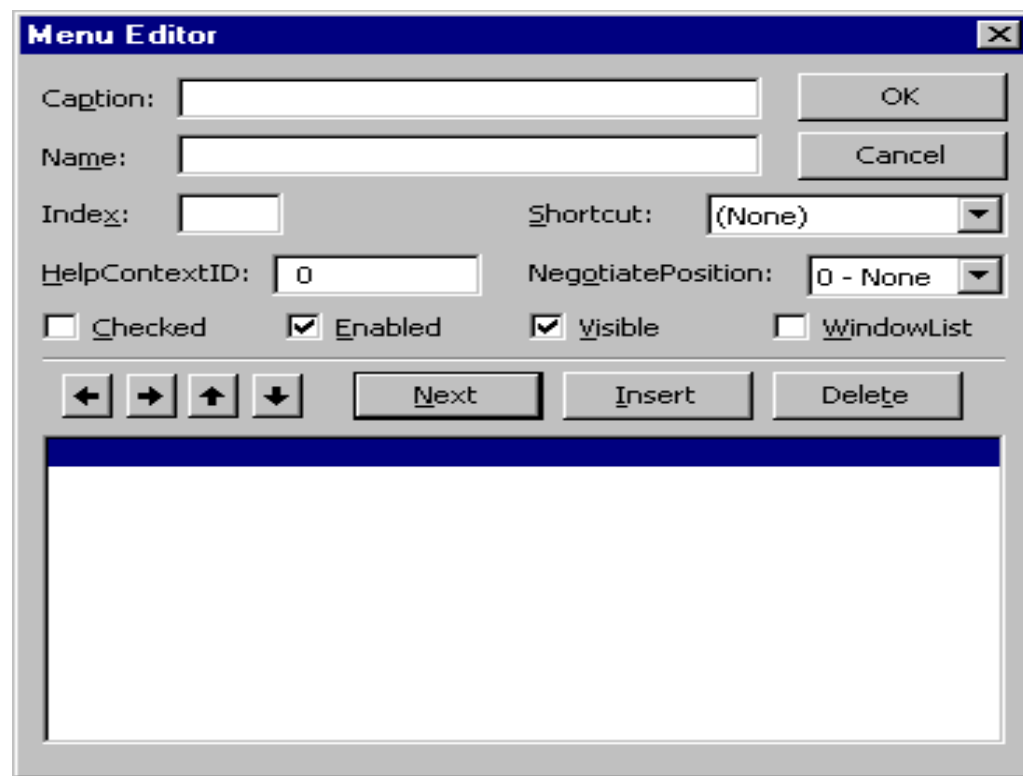


图 6.7 菜单编辑器

使用菜单编辑器中的列表框

菜单控件列表框（位于菜单编辑器的下部）列出当前窗体的所有菜单控件。当在标题文本框中键入一个菜单项时，该项也会出现在菜单控件列表框中。

从列表框中选取一个已存在的菜单控件可以编辑该控件的属性。

例如，图 6.7 表明典型应用程序中“文件”菜单的各种菜单控件。菜单控件在菜单控件列表框中的位置决定了该控件是菜单标题、菜单项、子菜单标题还是子菜单项：

- 位于列表框中左侧平齐的菜单控件作为菜单标题显示在菜单栏中。
- 列表框中被缩进过的菜单控件，当单击其前导的菜单标题时才会在该菜单上显示。
- 一个缩进过的菜单控件，如果后面还紧跟着再次缩进的一些菜单控件，它就成为一个子菜单的标题。在子菜单标题以下缩进的各个菜单控件，就成为该子菜单的菜单项。
- 一个以连字符 (-) 作为它的 Caption 属性的菜单控件，作为一个分隔符条出现。分隔符条可把菜单项划分成若干个逻辑组。

注意 如果菜单控件是一个菜单标题、带有子菜单项、被复选或无效、或者有一个快捷键，那么它就不能作为分隔符条。

要在菜单编辑器中创建菜单控件，请按照以下步骤执行：

1. 选取该窗体
2. 从“工具”菜单中，选取“菜单编辑器”。
 - 或者 -
 - 在“工具栏”上单击“菜单编辑器”按钮。
3. 在“标题”文本框中，为第一个菜单标题键入希望在菜单栏上显示的文本。如果希望某一字符成为该菜单项的访问键，也可以在该字符前面加上一个 (&) 字符。在菜单中，这一字符会自动加上一条下划线。

菜单标题文本显示在菜单控件列表框中。

4. 在“名称”文本框中，键入将用来在代码中引用该菜单控件的名字。请参阅本章后面“菜单标题与命名准则”。
5. 单击向左或向右箭头按钮，可以改变该控件的缩进级。
6. 如果需要的话，还可以设置控件的其他属性。这一工作可以在菜单编辑器中做，也可以以后在“属性”窗口中做。
7. 选取“下一个”就可以再建一个菜单控件。

- 或者 -

单击“插入”可以在现有的控件之间增加一个菜单控件。

也可以单击向上与向下的箭头按钮，在现有菜单控件之中移动控件。

8. 如果窗体所有的菜单控件都已创建，选取“确定”可关闭菜单编辑器。创建的菜单标题将显示在窗体上。在设计时，单击一个菜单标题可下拉其相应的菜单项。

分隔菜单项

分隔符条作为菜单项间的一个水平行显示在菜单上。在菜单项很多的菜单上，可以使用分隔符条将各项划分成一些逻辑组。例如 Visual Basic 的“帮助”菜单，使用分隔符条将其菜单项分成三组，如图 6.8 所示。

要在菜单编辑器中创建分隔符条，请按照以下步骤执行：

1. 如果想在现有菜单中增加一个分隔符条，选取“插入”，在想要分隔开来的菜单项之间插入一个菜单控件。

2. 如有必要，单击右箭头按钮使新菜单项缩进到与它要隔开的菜单项同级。
3. 在“标题”文本框中键入一个连字符 (-)。
4. 设置“名称”属性。
5. 选取“确定”，关闭菜单编辑器。

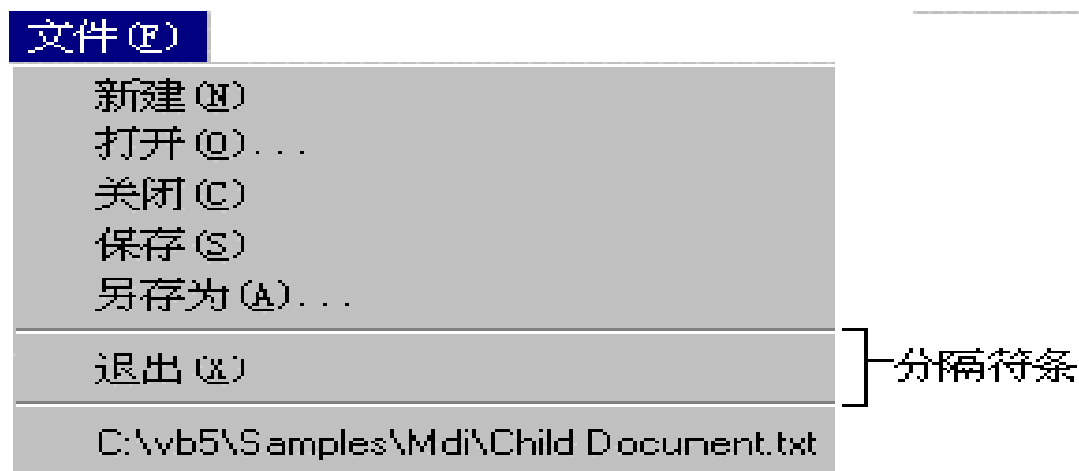


图 6.8 分隔符条

注意 虽然分隔符条是当作菜单控件来创建的，它们却不能响应 Click 事件，而且也不能被选取。

赋值访问键和快捷键

通过定义访问键和快捷键可改进键盘对菜单命令的访问。

访问键

访问键允许按下 ALT 键并打入一个指定字符来打开一个菜单。一旦菜单打开，通过按下所赋值的字符（访问键）可选取控件。例如，按下 ALT + E 键

可打开“编辑”菜单，再按 P 键可选取“粘贴”菜单项。在菜单控件的标题中，一个指定的访问键表现为一个带下划线的字母，如图 6.9 所示。

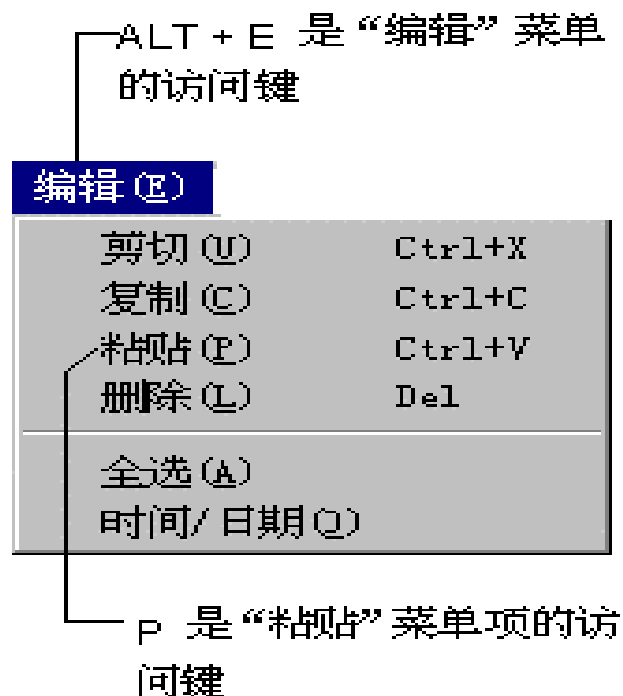


图 6.9 访问键

要在菜单编辑器中给菜单控件赋值访问键，请按照以下步骤执行：

1. 选取要赋值访问键的菜单项。
2. 在其“标题”框中，在要作为访问键字符的前面直接键入一个 (&) 字符。

例如，如果图 6.9 所示的“编辑”菜单被打开，下列 Caption 属性设置值响

应的对应键。

菜单控件标题	标题属性	访问键
Cut	Cu&t	t
Copy	C&opy	o
Paste	&Paste	p
Delete	De&lete	l
Select All	Select &All	a
Time/Date	Time/&Date	d

注意 菜单中不能使用重复的访问键。如果多个菜单项使用同一个访问键，则该键将不起作用。例如，如果 C 同时是“剪切”和“复制”的访问键，那么，当选取“编辑”菜单且键入 C 时，则“复制”命令将被选，但只有按下 ENTER 键以后，应用程序才会执行该命令。而“剪切”命令根本不会执行。

快捷键

快捷键按下时会立刻运行一个菜单项。可以为频繁使用的菜单项指定一个快捷键，它提供一种键盘单步的访问方法，而不是按住 ALT 键、再按菜单标题访问字符，然后再按菜单项访问字符的三步方法。快捷键的赋值包括功能键与控制键的组合，如 CTRL + F1 键或 CTRL + A 键。它们出现在菜单中相应菜单项的右边，如图 6.10 所示。

编辑(E)	
剪切(U)	Ctrl+X
复制(C)	Ctrl+C
粘贴(P)	Ctrl+V
删除(L)	Del
全选(A)	
时间/日期(D)	

图 6.10 快捷健

要对菜单项赋值快捷键，请按照以下步骤执行：

1. 打开“菜单编辑器”。
2. 选取该菜单项。
3. 在“快捷键”组合框中选取功能键或者键的组合。

要删除快捷键赋值，应选取列表顶部的“(none)”。

注意 快捷键将自动出现在菜单上；因此，不需要在菜单编辑器的“标题”框中键入 CTRL+key。

菜单标题与命名准则

为了与其它应用程序保持一致，创建菜单时遵循所确立的命名准则是一个好办法。

设置 Caption 属性

为菜单项进行标题赋值时，应当尽量遵循下列准则：

- 菜单中项目名称应当唯一，但不同菜单中相似动作项目可以重名。
- 项目名称可以是单词、复合词或者多个词。
- 每一个项目名称都应当有一个用键盘选取命令的、一个唯一的记忆访问字符。访问字符应当是菜单标题的第一个字母，除非别的字符更易记；两个菜单标题不能用同一个访问字符。关于指定访问键与快捷键的详细信息，请参阅本章前面的“用菜单编辑器创建菜单”部分。
- 如果命令在完成之前还需要附加信息，则在其名称后面应当有一个省略号（…），比如显示一个对话框的命令（“另存为…”，“首选项…”）。
- 项目名称尽量简短。如果要使应用程序本地化，那么，在外文版中字词长度会增加将近百分之三十，这样也许没有足够空间列出各个菜单项。有关应用程序本地化的更详细内容，请参阅第十六章“国际发行”。

菜单命名约定

为了使代码更可读和更易维护，在菜单编辑器中设置 `Name` 属性时遵循已确定的命名约定是一个好办法。大多数命名约定规则都建议用前缀来标识对象（即，对菜单控件用 `mnu`），其后紧跟顶层菜单的名称（如 `File`）。对于子菜单，其后再紧跟该子菜单的标题（如 `mnuFileOpen`）。

详细信息 有关所建议的命名约定的示例，请参阅附录 B “Visual Basic 编

码约定”。有关所建议的命名约定的示例，请参阅“Visual Basic 编码约定”。

创建子菜单

所创建的每个菜单可以包含最多五级子菜单。子菜单会分支出另一个菜单以显示它自己的菜单项。需使用子菜单的场合有：

- 菜单栏已满。
- 某一特定菜单控件很少被用到。
- 要突出某一菜单控件与另一个的关系。

然而，如果菜单栏中还有空间，最好再创建一个菜单标题而不是子菜单。这样，当菜单拉下时所有控件都可见。限制使用子菜单也是一种好的编程策略，它可以免去查找应用程序菜单界面的负担（大多数应用程序都只使用一级子菜单）。

在菜单编辑器中，在不是菜单标题的菜单控件之下缩进的任何菜单控件，都是子菜单控件。一般来说，子菜单控件可以包括子菜单项、分隔符条和子菜单标题。

要创建子菜单，请按照以下步骤执行：

1. 创建想作为子菜单标题的菜单项。
2. 创建将出现在新子菜单中的各个项目，然后单击右箭头按钮将它们缩进。

在菜单编辑器中每一缩进级前都加了四个点 (....)。要删除一缩进级，单击左箭头按钮。

注意 如果想用多于一级的子菜单，可以考虑使用对话框来替代。对话框允许在一个地方指定好几个选择。关于使用对话框的内容，请参阅本章后面的“对话框”。

创建菜单控件数组

菜单控件数组就是在同一菜单上共享相同名称和事件过程的菜单项目的集合。菜单控件数组使用于：

- 在运行时要创建一个新菜单项，它必须是控件数组中的成员。如 MDI Notepad 示例，它就用一个菜单控件数组来存储新近打开的文件清单。
- 简化代码，因为通用代码块可以被所有菜单项使用。

每个菜单控件数组元素都由唯一的索引值来标识，该值在菜单编辑器上“Index 属性框”中指定。当一个控件数组成员识别一个事件时，Visual Basic 将其 Index 属性值作为一个附加的参数传递给事件过程。事件过程必须包含有核对 Index 属性值的代码，因而可以判断出正在使用的是哪一个控件。

详细信息 关于控件数组的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”中的“使用控件数组”。要在菜单编辑器中创建菜单控件数组，请按照以下步骤执行：

1. 选取窗体。
2. 从“工具”菜单中，选取“菜单编辑器”。
 - 或 -
 - 在“工具栏”上单击“菜单编辑器”按钮。

3. 在“标题”文本框中，键入想出现在菜单栏中的第一个菜单标题的文本。菜单标题文本显示在菜单控件列表框中。
4. 在“名称”文本框中，键入将在代码中用来引用菜单控件的名称。保持“索引”框是空的。
5. 在下一个缩进级，通过设定“标题”和“名称”来创建将成为数组中第一个元素的菜单项。
6. 将数组中第一个元素的“索引”设置为 0。
7. 在第一个的同一缩进级上创建第二个菜单项。
8. 将第二个元素的“名称”设置成与第一个元素相同，且把它的“索引”设置为 1。
9. 对于数组中的后续元素重复步骤 5-8。

重点 菜单控件数组的各元素在菜单控件列表框中必须是连续的，而且必须在同一缩进级上。创建菜单控件数组时，要把在菜单中出现的分隔符条也包括进去。

运行时创建和修改菜单

设计时创建的菜单也能动态地响应运行时的条件。例如，如果菜单项的动作在某些点上成为不适当时，通过使其失效可防止对该菜单项的选取。比如：在 MDI NotePad 应用程序中，如果剪贴板上没有任何文本，则“编辑”菜单中的“粘贴”菜单项变暗，因而就不能被选。

如果有一个菜单控件数组，也可以动态地增加菜单项。这将在本主题后面

的“运行时添加菜单控件”中讲述。

也可以编写应用程序，使用复选标志来指示几个命令中的哪一个是最最后选取的。例如：如果工具栏被显示，MDI NotePad 应用程序工具栏的“选项”菜单项就会显示一个复选标志。本节描述的其它菜单控件功能包括使菜单项可见或不可见以及增加或删除菜单项操作的代码。

使菜单命令有效或无效

所有的菜单控件都具有 Enabled 属性，当这个属性设为 False 时，菜单命令无效使它不响应动作。当 Enabled 设为 False 时，快捷键的访问也无效。一个无效的菜单控件会变暗，如图 6.11 所示的“粘贴”菜单项。

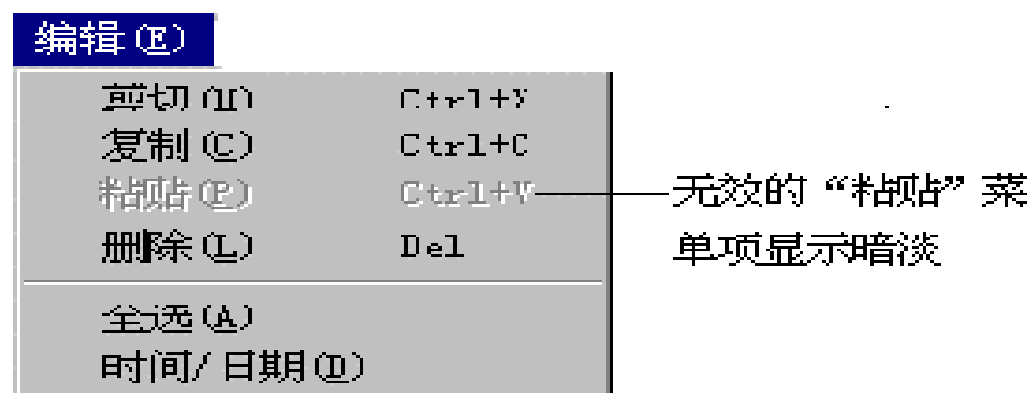


图 6.11 一个无效的菜单项

例如：下列语句使 MDI NotePad 应用程序中“编辑”菜单上的“粘贴”菜单项无效。

```
mnuEditPaste.Enabled = False
```

菜单标题的无效使得整个菜单无效，因为不首先单击菜单标题，就不能访问任何菜单项。例如：下列代码将使 MDI NotePad 应用程序的“编辑”菜单无效。

```
mnuEdit.Enabled = False
```

显示菜单控件上的复选标志

使用 `Checked` 属性，可以把一个复选标志放置在菜单上，以便：

- 表示打开/关闭条件的状态。选取菜单命令可交替地增加和删除此复选标志。
- 指示几个模式中哪一个正在起作用。MDI NotePad 应用程序的“选项”菜单使用复选标志来指示工具栏的状态，如图 6.12 所示。

在 Visual Basic 中可以用 `Checked` 属性来创建复选标志。通过选取被标记为 `Checked` 的复选框来设置菜单编辑器中的 `Checked` 属性的初始值。在运行时要从一个菜单控件上增加或删除复选标志，可以从代码中设置它的 `Checked` 属性。例如：

```
Private Sub mnuOptions_Click ()
```

```
    '根据 Visible 属性设置复选标志的状态。
```

```
    mnuOptionsToolbar.Checked = picToolbar.Visible
```

```
End Sub
```

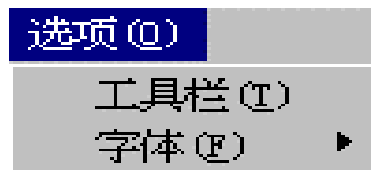


图 6.12 被复选的菜单项

使菜单控件不可见

在菜单编辑器中，通过选取被标记为 `Visible` 的复选框，可以设置菜单控件的 `Visible` 属性的初值。在运行时，要使一个菜单控件可见或不可见，可以从代码中设置其 `Visible` 属性。例如：

```
mnuFileArray(0).Visible = True    ' 使控件可见。
```

```
mnuFileArray(0).Visible = False    ' 使控件不可见。
```

当一个菜单控件不可见时，菜单中的其余控件会上移以填补空出的空间。如果控件位于菜单栏上，则菜单栏上其余的控件会左移以填补该空间。

注意 使菜单控件不可见也产生使之无效的作用，因为该控件通过菜单、访问键或者快捷键都再无法访问。如果菜单标题不可见，则该菜单上所有控件均无效。

运行时添加菜单控件

运行时菜单可以增长。例如：在图 6.13 中作为 SDI NotePad 应用程序中

被打开的文件，会动态地创建菜单项来显示刚打开文件的路径名。

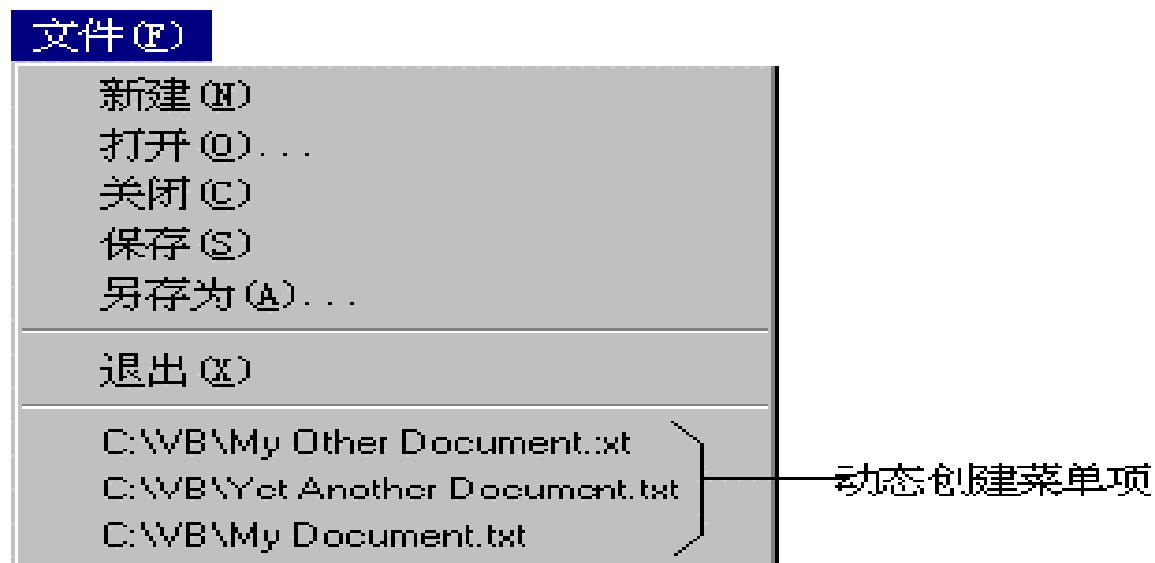


图 6.13 运行时创建并显示的菜单控件数组的元素

运行时为了创建控件必须使用控件数组。因为设计时对 `mnuRecentFile` 菜单控件的 `Index` 属性进行了赋值，它自动地成为控件数组的一个元素——即使还没有创建其它元素。

当创建 `mnuRecentFile(0)` 时，实际上创建了一个在运行时不可见的分隔符条。当运行时用户第一次存储一个文件时，这个分隔符条就会变得可见，且第一个文件名被加到该菜单上。运行时每存储一个文件，则会再装入一个菜单控件到该数组中，从而使该菜单增长。

运行时所创建的控件可以使用 `Hide` 方法或者设置该控件的 `Visible` 属性为 `False` 来隐藏。如果要从内存中删除一个控件数组中的控件，请使用 `Unload`

语句。

编写菜单控件的代码

当用户选取一个菜单控件时，一个 Click 事件出现。需要在代码中为每个菜单控件编写一个 Click 事件过程。除分隔符条以外的所有菜单控件（以及无效的或不可见的菜单控件）都能识别 Click 事件。

在菜单事件过程中编写的代码与在控件任何其它事件过程中编写的代码完全相同。例如：“文件”菜单中的“关闭”菜单项的 Click 事件的代码看上去如下：

```
Sub mnuFileClose_Click()  
    Unload Me  
End Sub
```

一旦菜单标题被选取，Visual Basic 将自动地显示出一个菜单；但是，没有必要为一个菜单标题的 Click 事件过程编写代码，除非想执行其它操作，比如每次显示菜单时使某些菜单项无效。

注意 在设计时，当关闭菜单编辑器时，所创建的菜单将显示在窗体上。在窗体上选取一个菜单项将显示那个菜单控件的 Click 事件过程。

显示弹出式菜单

弹出式菜单是独立于菜单栏而显示在窗体上的浮动菜单。在弹出式菜单上显示的项目取决于按下鼠标右键时指针所处的位置；因而，弹出式菜单也被称为上下文菜单。在 Microsoft Windows 95 中，可以通过单击鼠标右键来激活上下文菜单。

在运行时，至少含有一个菜单项的任何菜单都可以作为弹出式菜单被显示。为了显示弹出式菜单，可使用 `PopupMenu` 方法。这个方法使用下列语法：

```
[object.]PopupMenu menuname [, flags [, x [, y [, boldcommand ]]]]
```

例如：当用户用鼠标右键单击一个窗体时，以下的代码显示一个名为 `mnuFile` 的菜单。可用 `MouseUp` 或者 `MouseDown` 事件来检测何时单击了鼠标右键，虽然标准用法是使用 `MouseUp` 事件：

```
Private Sub Form_MouseUp (Button As Integer, Shift As _  
    Integer, X As Single, Y As Single)  
    If Button = 2 Then      '    检查是否单击了鼠标右键。  
        PopupMenu mnuFile  '    把文件菜单显示为一个弹出式菜单。  
    End If  
End Sub
```

直到菜单中被选取一项或者取消这个菜单时，调用 `PopupMenu` 方法后面的代码才会运行。

注意 每次只能显示一个弹出式菜单。在已显示一个弹出式菜单的情况下，对后面的调用 `PopupMenu` 方法将不予理睬。在一个菜单控件正活动的任何时刻，调用 `PopupMenu` 方法均不会被理睬。

常常会想用一個弹出式菜单来访问那些在菜单栏中不常用的选项。为创建

一个不显示在菜单栏里的菜单，可在设计时使顶级菜单项目为不可见（保证在菜单编辑器里的“Visible”复选框没有被选上）。当 Visual Basic 显示一个弹出式菜单时，指定的顶级菜单的 Visible 属性会被忽略。

Flags 参数

在 PopupMenu 方法中使用 flags 参数可以进一步定义弹出式菜单的位置与性能。下表列出了可用于描述弹出式菜单位置的标志。

位置常数	描述
VbPopupMenuLeftAlign	缺省。指定的 x 位置定义了该弹出式菜单的左边界
vbPopupMenuCenterAlign	弹出式菜单以指定的 x 位置为中心
vbPopupMenuRightAlign	指定的 x 位置定义了该弹出式菜单的右边界

下表列出了可用于描述弹出式菜单性能的标志。

行为常数	描述
vbPopupMenuLeftButton	缺省。只有当用户用鼠标左键单击菜单项时，才显示弹出式菜单
vbPopupMenuRightButton	当用户用鼠标右键或者左键单击菜单项时，显示弹出式菜单

想要指定一个标志，从每组中选取一个常数，再用 Or 操作符将它们连起来。下面的代码是，当用户单击一个命令按钮时，显示一个上边框在窗体中心的弹出式菜单。弹出式菜单触发受到鼠标右键或左键单击的菜单项的 Click 事件。

```
Private Sub Command1_Click ()  
    ' X 变量和 Y 变量的尺寸。
```

```
Dim xloc, yloc
```

```
'设置 X 变量和 Y 变量到窗体中心。
```

```
xloc = ScaleWidth / 2
```

```
yloc = ScaleHeight / 2
```

```
'显示弹出式菜单。
```

```
PopupMenu mnuEdit, vbPopupMenuCenterAlign Or _
```

```
vbPopupMenuRightButton, xloc, yloc
```

```
End Sub
```

Boldcommand 参数

使用 **boldcommand** 参数来指定在显示的弹出式菜单中想以粗体字体出现的菜单控件的名称。在弹出式菜单中只能有一个菜单控件被加粗。

MDI 应用程序中的菜单

在 MDI 应用程序中，每一个子窗体的菜单都显示在 MDI 窗体上，而不是在子窗体本身。当子窗体有焦点时，该子窗体的菜单（如果有的话）就代替菜单栏上的 MDI 窗体的菜单。如果没有可见的子窗体，或者如果带有焦点的子窗体没有菜单，则显示 MDI 窗体的菜单（见图 6.14 和 6.15）。

MDI 应用程序使用几套菜单的情况很普遍。当打开一个文档时，应用程序显示与该类文档相关的菜单。通常，当没有可见子窗体时，会显示一个不同的菜单。例如，当没有文件打开时，Microsoft Excel 只显示“文件”与“帮助”菜单。当用户打开一个文件时，其他菜单就会显示出来（“文件”、“编辑”、“视图”、“插入”、“格式”、“工具”、“数据”和“窗口”等等）。

创建 MDI 应用程序的菜单

通过给 MDI 窗体和子窗体添加菜单控件，可以为 Visual Basic 应用程序创建菜单。管理 MDI 应用程序中菜单的一个方法是把希望在任何时候都显示的菜单控件放在 MDI 窗体上（即使没有子窗体可见时）。当运行该应用程序时，如果没有可见的子窗体，会自动显示 MDI 窗体菜单，如图 6.14 所示。

把应用于子窗体的菜单控件放置到子窗体中。在运行时，只要有一个子窗体可见，这些菜单标题就会显示在 MDI 窗体的菜单栏中。

有些应用程序支持多种类型的文档。例如，在 Microsoft Access 中，可以打开表格、查询、窗体和其它文档类型。要在 Visual Basic 中创建这样的应用程序，应使用两个子窗体。设计一个带有执行工作单任务菜单的子窗体和另一个带有完成图表任务菜单的子窗体。

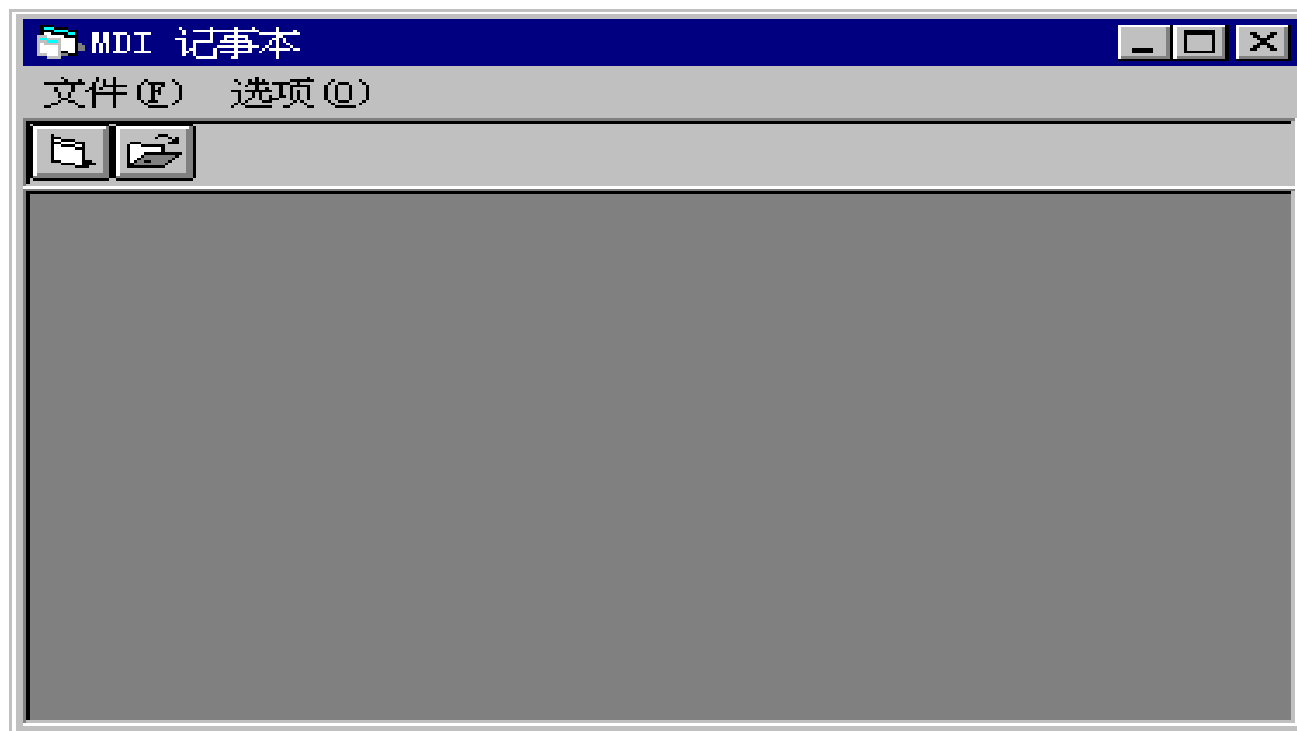


图 6.14 当没有子窗体被加载时，显示 MDI 窗体

在运行时，当工作单窗体的实例有焦点时，则显示工作单菜单，而当选取图表时，则显示图表窗体的菜单。如果所有的工作单和图表都关闭，则显示 MDI 窗体的菜单。关于创建菜单的详细内容，请参阅本章前面的“在应用程序中使用菜单”部分。

创建“窗口”菜单

大多数 MDI 应用程序（例如：Microsoft Word for Windows 与 Microsoft

Excel) 都结合了“窗口”菜单。这是一个显示所有打开的子窗体标题的特殊菜单，如图 6.15 所示。另外，有些应用程序将操纵子窗体的命令，比如“层叠”、“平铺”与“排列图标”，放在这个菜单中。

在 MDI 窗体或者 MDI 子窗体上的任何菜单控件，只要将其 WindowList 属性设置为 True，都可以用于显示打开子窗体的清单。在运行时，Visual Basic 自动管理与显示标题清单，并在当前正有焦点的标题旁边显示一个复选标志。另外，在窗口清单的上方自动放置一个分隔符条。

要设置 WindowList 属性，请按照以下步骤执行：

1. 选取希望菜单出现其上的窗体，从“工具”菜单中，选取“菜单编辑器”。

注意 WindowList 属性只应用于 MDI 窗体和 MDI 子窗体。它对标准窗体（非 MDI）不起作用。

2. 在菜单编辑器列表框中，选取希望打开的子窗体的清单显示在其上的菜单。

3. 选取“WindowList”复选框。

在运行时，这个菜单显示打开的子窗体的清单。另外，这个菜单控件的 WindowList 属性返回 True。

详细信息 请参阅《语言参考》的“WindowList 属性”。



图 6.15 “窗口”菜单显示每个打开子窗体的名称

排列子窗体

就像前面所提到的，有些应用程序把诸如平铺 (Tile)、层叠 (Cascade) 与排列图标 (Arrange Icons) 等操作列在和打开的子窗体列表的同一个菜单上。在 MDI 窗体中使用 `Arrange` 方法来重新对齐子窗体。可以层叠、水平平铺或者沿着 MDI 窗体的下部排列的子窗体图标等方式来显示子窗体。下面的实例示出了“层叠”、“平铺”和“排列图标”菜单控件的 Click 事件

过程。

```
Private Sub mnuWCascade_Click ()
```

```
    '层叠子窗体。
```

```
    frmMDI.Arrange vbCascade
```

```
End Sub
```

```
Private Sub mnuWTile_Click ()
```

```
    '平铺子窗体。
```

```
    frmMDI.Arrange vbTileHorizontal
```

```
End Sub
```

```
Private Sub mnuWArrange_Click ()
```

```
    '排列所有子窗体图标。
```

```
    frmMDI.Arrange vbArrangeIcons
```

```
End Sub
```

注意 内部常数 vbCascade、vbTileHorizontal 和 vbArrangeIcons 都在对象浏览器的 Visual Basic (VB) 对象库中列出。

当平铺或者层叠有着固定边框类型的子窗体时，每个子窗体都像具有可变尺寸边框那样被定位。这使得子窗体可能重叠。

工具栏

可以用工具栏进一步增强应用程序的菜单界面。工具栏含有工具栏按钮，它提供了对于应用程序中最常用的命令的快速访问。例如：Visual Basic 的工具栏含有执行常用命令的 `ToolBarButton`，如“打开”（打开已存在的工程）、“保存”（存储当前的工程），等等。

创建工具栏

工具栏（也称为发条或者控制栏）已经成为许多基于 Windows 的应用程序的标准功能。工具栏提供了对于应用程序中最常用的菜单命令的快速访问。使用 `ToolBar` 控件来创建工具栏非常容易且很方便，它在 Visual Basic 的专业版与企业版中是可用的。如果使用的是 Visual Basic 学习版，则可以像本章后面的“协调菜单与工具栏的外观”所描述的那样用手工来创建工具栏。

下列示例演示为 MDI 应用程序创建工具栏的过程；在标准窗体上创建一个工具栏的过程基本上一样。

要手工创建工具栏，请按照以下步骤执行：

1. 在 MDI 窗体上放置一个图片框。

图片框的宽度会自动伸展，直到填满 MDI 窗体工作空间。工作空间就是窗体边框以内的区域，不包括标题条、菜单栏或所有的工具栏、状态

栏或者可能在窗体上的滚动条。

注意 只能把那些直接支持 `Align` 属性的控件放置在 MDI 窗体上（图片框是支持这一属性的唯一的标准控件）。

2. 在图片框中，可以放置任何想在工具栏上显示的控件。

典型地，用 `CommandButton` 或 `Image` 控件来创建工具栏按钮。图 6.16 表示出了一个含有 `Image` 控件的工具栏。

要在图片框中添加控件，单击工具栏中的控件按钮，然后在图片框中画出它。

注意 当 MDI 窗体中包含了图片框时，该 MDI 窗体的内部区域不包括图片框在内。例如：MDI 窗体的 `ScaleHeight` 属性返回 MDI 窗体的内部高度，这个高度已不包括图片框的高度。

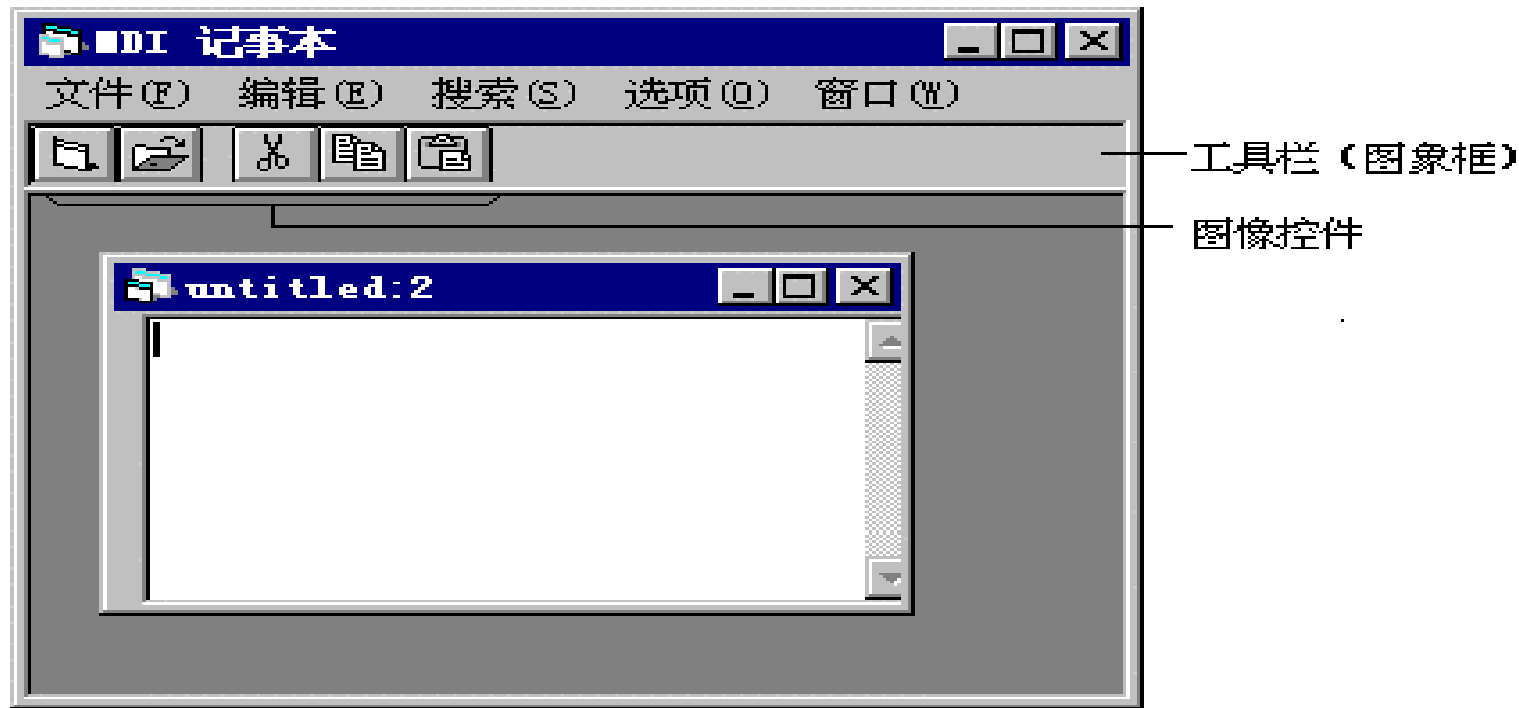


图 6.16 用 Image 控件来创建工具栏按钮

3. 设置设计时属性。

使用工具栏的一个好处是可以显示一个形象的命令图示。Image 控件是作为工具栏按钮的一个很好的选择，因为可以用它来显示一个位图。在设计时设置其 Picture 属性来显示一个位图；这样，当该按钮被单击时，即能提供一个命令执行的可见信息。也可以通过设置按钮的 ToolTipText 属性来使用工具提示，这样，当用户把鼠标指针保持在一个按钮上时，就可以显示出该工具栏按钮的名称。

4. 编写代码

因为工具栏频繁地用于提供对其它命令的快捷访问，因而在大部分时间内都是从每一个按钮的 `Click` 事件中调用其它过程，比如对应的菜单命令。

提示 可用一个不显示工具栏的 `MDI` 窗体来使用在运行时不可见的控件（比如 `Timer` 控件）。为了做到这一点，在 `MDI` 窗体上放置一个图片框，把控件放到图片框中，然后把图片框的 `Visible` 属性设置为 `False`。

编写工具栏的代码

工具栏用于提供访问某些应用程序命令的快捷方法。例如：图 6.16 中工具栏上的第一个按钮就是“新建文件”命令的快捷键。现在，可以在 `MDI NotePad` 示例应用程序中三个地方请求创建新文件。

- 在 `MDI` 窗体上（`MDI` 窗体上“文件”菜单中的“新建”命令）。
- 在子窗体上（在子窗体“文件”菜单上的“新建”命令）。
- 在工具栏上（“新建文件”按钮）。

与其把这个代码重复三次，还不如从子窗体的 `mnuFileNew_Click` 事件取出原代码，然后把它放入子窗体的一个公用过程中。可以从上面任何一个事件过程调用这个过程。以下是一个示例：

这个例程在公共过程中。

```
Public Sub FileNew ()
```

```
    Dim frmNewPad As New frmNotePad
```

```
        frmNewPad.Show  
End Sub
```

'在子窗体的“文件”菜单上选取“新建”。

```
Private Sub mnuchildFileNew_Click()  
    FileNew  
End Sub
```

'在 MDI 窗体的“文件”菜单上选取“新建”。

```
Private Sub mnumdiFileNew_Click()  
    frmNotePad.FileNew  
End Sub
```

'在工具栏上单击“新建文件”按钮。

```
Private Sub btnFileNew_Click()  
    frmNotePad.FileNew  
End Sub
```

协调菜单与工具栏的外观

当由其它应用程序提供的对象在窗体中被激活时，有许多方法让对象的菜单与工具栏出现在容器窗体内；然而，需要规定它们将如何显示。这一过

程叫做用户界面协调，因为 Visual Basic 与已经链接或嵌入的对象必须协调容器窗体中的空间。

控制菜单的外观

通过设置窗体的 `NegotiateMenus` 属性可以决定一个链接或嵌入的对象的菜单是否出现在容器窗体中。如果子窗体的 `NegotiateMenus` 属性置为 `True`（缺省），而且容器有一个定义的菜单栏，那么，当这个对象被激活时，它的菜单就放置在容器的菜单栏中。如果容器没有菜单栏，或者 `NegotiateMenus` 属性被设置为 `False`，那么，当这个对象激活时对象的菜单将不出现。

注意 `NegotiateMenus` 属性不适用于 MDI 窗体。

控制工具栏的外观

MDI 窗体的 `NegotiateToolbars` 属性决定了链接或嵌入对象的工具栏是不固定的调色板还是被放置在父窗体上。这种性能不要求工具栏出现在 MDI 父窗体上。如果 MDI 窗体的 `NegotiateToolbars` 属性设为 `True`，则对象的工具栏出现在 MDI 父窗体上。如果 `NegotiateToolbars` 设为 `False`，则对象的工具栏就为不固定的调色板。

注意 `NegotiateToolbars` 属性只用于 MDI 窗体。

如果 MDI 窗体包含工具栏，它通常被包含在父窗体的 `PictureBox` 控件中。图片框的 `Negotiate` 属性决定了被激活时容器的工具栏是继续显示还

是被对象的工具栏所代替。如果 `Negotiate` 设置为 `True`，则除了容器的工具栏外还显示对象的工具栏。如果 `Negotiate` 设置为 `False`，则对象的工具栏代替容器的工具栏。

注意 菜单与工具栏的协调只发生在支持现场激活的可插入对象之上。有关现场激活的详细信息，请参阅第十章“用部件编程”。用以下过程可以看到这三个属性如何交互作用。

要执行菜单与工具栏的协调，请按照以下步骤执行：

1. 在 MDI 窗体中添加工具栏。这在本章前面的“创建工具栏”中描述过。
2. 在子窗体上放置一个可插入的对象。
3. 设置 `NegotiateMenus`、`NegotiateToolbars` 以及 `Negotiate` 属性。
4. 运行此应用程序，然后双击该对象。

对话框

在基于 Windows 的应用程序中，对话框被用来：

- 提示用户提供应用程序继续执行所需要的数据。
- 向用户显示信息。

例如，在 Visual Basic 中，用“打开文件”对话框来显示已存在的工程。Visual Basic 中的“关于”对话框也是一个如何使用对话框来显示信息的例子。在菜单栏上单击“帮助”，选择“关于 Visual Basic”菜单项，则显示“关于”对话框。

模式与无模式的对话框

对话框不是模式就是无模式的。模式对话框，在可以继续操作应用程序的其他部分之前，必须被关闭（隐藏或卸载）。例如，如果一个对话框，在可以切换到其它窗体或对话框之前要求先单击“确定”或“取消”，则它就是模式的。

Visual Basic 中的“关于”对话框是模式的。显示重要消息的对话框总应当是模式的一一那就是说，在继续做下去之前，总是要求用户应当先关上对话框或者对它的消息作出响应。

无模式的对话框允许在对话框与其他窗体之间转移焦点而不用关闭对话框。当对话框正在显示时，可以在当前应用程序的其他地方继续工作。无模式对话框很少使用。Visual Basic 中“编辑”菜单中的“查找”对话框就是一个无模式对话框的实例。无模式对话框用于显示频繁使用的命令与信息。

要将窗体作为模式对话框显示，请

- 使用 Show 方法，其 style 参数值为 vbModal（一个值为 1 的常数）。
例如：

将 frmAbout 作为模式对话框显示。

```
frmAbout.Show vbModal
```

要将窗体作为无模式对话框显示，请

- 使用不带 style 参数的 Show 方法。

例如：

'将 frmAbout 作为无模式对话框显示。

frmAbout.Show

注意 如果窗体显示为模式对话框，则只有当对话框关闭之后，在 Show 方法后的代码才能执行。然而，当窗体被显示为无模式对话框时，在该窗体显示出来以后，Show 方法后面的代码紧接着就会执行。

Show 方法另有一个可选参数，owner，用来指定窗体的父子关系。可将某个窗体名传给这个参数，使得这个窗体成为新窗体的拥有者。

要显示一个窗体，作为另一个窗体的子窗体，请

- 这里 Show 方法有两个参数：style 和 owner。

例如：

' 将 frmAbout 显示为 frmMain 的无模式子窗体。

frmAbout.Show vbModeless, frmMain

在 Show 方法中使用 owner 参数，确保对话框在它的父窗体最小化时它也最小化，或者在其父窗体关闭时它也卸载。

预定义对话框的使用

在应用程序中添加对话框最容易的方法是使用预定义对话框，因为不必考虑设计、装载或者显示对话框方面的问题。然而，控件在其外观上要受到限制。预定义的对话框总是模式的。

下表列出了 Visual Basic 应用程序中添加预定义对话框时能使用的函数。

使用的函数	功能
InputBox 函数	在对话框中显示一个命令提示，并返回所接收的任何东西
MsgBox 函数	在对话框中显示一条消息，并返回一个表示命令按钮已被单击的值

用输入框来提示输入

应用 InputBox 函数请求提供数据。这个函数显示要求输入数据的模式对话框。图 6.17 所示的文本输入框提示输入要打开文件的名称。

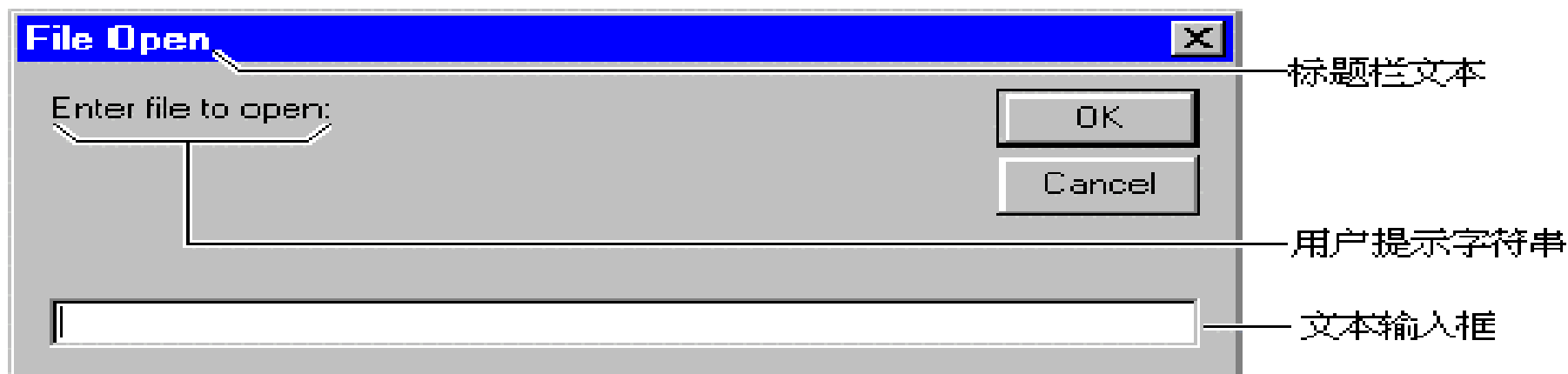


图 6.17 使用 InputBox 函数的对话框

以下代码显示图 6.17 中所示的输入框。

```
FileName = InputBox("Enter file to open:", "File Open")
```

注意 切记当使用 InputBox 函数时，对对话框的各部分的控制非常有限。只能改变标题栏中的文本、显示给用户的命令提示、对话框在屏幕上

的位置以及它是否显示一个“帮助”按钮。

详细信息 请参阅《语言参考》的“InputBox 函数”。

用消息对话框显示信息

可以用 `MsgBox` 函数获得“是”或者“否”的响应，并显示简短的消息，比如：错误、警告或者对话框中的期待。看完这些消息以后，可选取一个按钮来关闭该对话框。

如果文件不能打开，那么名为 `Text Editor` 的应用程序就会显示如图 6.18 所示的消息对话框。



图 6.18 用 `MsgBox` 函数创建的错误消息对话框

以下代码显示如图 6.18 所示的消息框：

```
MsgBox "Error encountered while trying to open file, _  
please retry.", vbExclamation, "Text Editor"
```

注意 所谓模式的，既可以局限于应用程序中，也可以局限于系统中。如

果消息框的模式局限在应用程序中（缺省），则在这个对话框未消失之前不能切换到该应用程序的其他部分，但是可以切换到其他应用程序。在消息框未消失之前系统的模式消息框不允许切换到别的应用程序。

详细信息 请参阅《语言参考》的“MsgBox 函数”。

用窗体作为自定义对话框

自定义对话框就是用户所创建的含有控件的窗体——这些控件包括命令按钮、选取按钮和文本框——它们可以为应用程序接收信息。通过设置属性值来自定义窗体的外观。也可以编写在运行时显示对话框的代码。

要创建自定义对话框，可以从新窗体着手，或者自定义现成的对话框。如果重复过多，可以建造能在许多应用程序中使用的对话框的集合。

要自定义现存的对话框，请按照以下步骤执行：

1. 从“工程”菜单中选取“添加窗体”，在工程中添加一现存的窗体。
2. 从“文件”菜单中选取“filename 另存为”并输入新的文件名（这可以防止改变已存在的窗体版本）。
3. 根据需要自定义窗体的外观。
4. 在代码窗口中自定义事件过程。

要创建新的对话框，请按照以下步骤执行：

1. 从“工程”菜单中选取“添加窗体”。

- 或者 -

在工具栏上单击“窗体”按钮，创建新的窗体。

2. 如有必要，自定义窗体外观。

3. 在“代码”窗口中自定义事件过程。

有很大的自由来定义自定义对话框的外观。它可以是固定的或可移动的、模式或无模式的。它可以包含不同类型的控件；然而，对话框通常不包括菜单栏、窗口滚动条、最小化与最大化按钮、状态条或者尺寸可变的边框。本主题余下的部分将讨论创建典型类型的对话框方法。

添加标题

对话框总应当有标识它的标题。要创建标题，设置该窗体的 `Caption` 属性为将在标题条中显示的文本字符串。通常，这一步是在设计时使用“属性”窗口来完成的，但也可以用代码来完成这一步。例如：

```
frmAbout.Caption = "About"
```

提示 如果想完全删除此标题栏，可以设置窗体的 `ControlBox`、`MinButton` 和 `MaxButton` 为 `False`；设置 `BorderStyle` 为尺寸不可变的设置（0、1 或 3），并设置 `Caption` 为空字符串（""）。

设置标准对话框的属性

一般来说，用户响应对话框时，先提供信息，然后用“确定”或者“取消”命令按钮关闭对话框。因为对话框是临时性的，用户通常不需要对它进行

移动、改变尺寸、最大化或最小化等操作。其结果是：随新窗体出现的可变尺寸边框类型、“控制”菜单框、“最大化”按钮以及“最小化”按钮，在大多数对话框中都是不需要的。

通过设置 `BorderStyle`、`ControlBox`、`MaxButton` 和 `MinButton` 属性，可以删除这些项目。例如，“关于”对话框可能使用以下的属性设置。

属性	设置	效果
<code>BorderStyle</code>	<code>1</code>	改变边框类型为固定的单个边框，因而防止对话框在运行时被改变尺寸
<code>ControlBox</code>	<code>False</code>	删除控制菜单框
<code>MaxButton</code>	<code>False</code>	删除最大化按钮，因而防止对话框在运行时被最大化
<code>MinButton</code>	<code>False</code>	删除最小化按钮，因而防止对话框在运行时被最小化

记住，如果删除“控制”菜单框（`ControlBox = False`），则必须向用户提供退出该对话框的其它方法。实现的办法通常是在对话框中添加“确定”、“取消”或者“退出”命令按钮，并在隐藏或卸载该对话框的 `Click` 按钮事件中添加代码。

添加和放置命令按钮

模式对话框必须至少包含一个退出该对话框的命令按钮。通常用两个命令按钮：其中一个按钮开始动作，而另一个按钮关闭该对话框而不做任何改变。典型状态是，这两个按钮的 `Caption` 属性设置“确定”与“取消”。在这个方案中，“确定”命令按钮的 `Default` 属性设置为 `True`，而 `Cancel` 命

令按钮的 `Cancel` 属性设置为 `True`。虽然“确定”与“取消”是最常用的按钮，其他的按钮标题组合也可使用。

显示消息的对话框通常使用 `Label` 控件来显示错误消息或者命令提示，并且用一至两个命令按钮来执行动作。例如，也许给标识的 `Caption` 属性赋以错误消息或者命令提示，而给两个命令按钮控件的 `Caption` 属性指定“是”与“否”。当用户选取“是”，则发生一个动作，当选取“否”时，则发生另一个动作。

这种类型对话框的命令按钮通常被放置在对话框的底部或右边，而顶部或左边的按钮为缺省按钮，如图 6.19 所示。



图 6.19 对话框中命令按钮的位置

设置缺省、取消和焦点

命令按钮控件提供了以下的属性：

- Default
- Cancel
- TabIndex
- TabStop

Default 按钮就是当用户按下 **ENTER** 键时选中的按钮。在一个窗体上，只能有一个命令按钮的 **Default** 属性可以设置为 **True**。按下 **ENTER** 键就调用缺省命令按钮的 **Click** 事件。这个功能连同编辑控件（比如 **TextBox**）一起工作。例如，可以在文本框中键入数据，然后按下 **ENTER** 键来产生 **Click** 事件，以取代选取“确定”按钮。

“取消”按钮就是当按下 **ESC** 键时选中的按钮。在一个窗体上，只能有一个命令按钮的 **Cancel** 属性可以设置为 **True**。按下 **ESC** 键调用“取消”命令按钮的 **Click** 事件。“取消”按钮也可以为缺省命令按钮。要指定对话框的“取消”按钮，设置该命令按钮的 **Cancel** 属性为 **True**。

提示 一般说来，代表最可靠的或者最安全的操作的按钮应当是缺省按钮。例如，在“文本替换”对话框中，“取消”应当是缺省按钮，而不是“全部替换”。

也可以指定当对话被显示时具有焦点的按钮。当窗体被显示时，带有最低的 **TabIndex** 设置的控件接收焦点。按下 **ENTER** 键会调用缺省命令按钮或者有焦点的命令按钮的 **Click** 事件。要在窗体被显示时将焦点给某个命

令按钮，则设置该命令按钮的 `TabIndex` 为 0，它的 `TabStop` 属性为 `True`。也可以用 `SetFocus` 方法在窗体被显示时将焦点给指定的控件。

详细信息 请参阅《语言参考》的“`TabIndex` 属性”与“`TabStop` 属性”。

使对话框上的控件无效

有时候需要使控件无效，因为它们的动作在当前的上下文中不适用。例如，当 Visual Basic 的“查找”对话框第一次显示时，“查找下一个”按钮应无效，如图 6.20 所示。设置对话框上的控件的 `Enabled` 属性为 `False`，可以使其无效。

要使对话框上的控件无效，请

- 设置每个控件的 `Enabled` 属性为 `False`。例如：

```
cmdFindNext.Enabled = False
```

```
cmdReplace.Enabled = False
```



图 6.20 对话框上的无效控件

显示自定义对话框

使用应用程序中显示其他窗体同样的方法来显示对话框。当应用程序运行时，启动窗体会自动装入。想在应用程序中出现第二个窗体或对话框，要用装入并显示它的代码来实现。同样，想要窗体或对话框消失，也要编写卸载或隐藏它的代码。

以下代码在用户选取“帮助”菜单中的“关于”菜单项时显示“关于”对话框。

```
Private Sub mnuHelpAbout_Click ()
```

```
    '此处用 style = vbModal 的 Show 方法将对话框显示为模式的。
```

```
    frmAbout.Show vbModal
```

```
End Sub
```

显示选项

代码决定对话框将如何装入内存与如何显示。下表描述各种类型的显示任务以及用来执行这些任务的关键字。

任 务	关 键 字
将窗体装入内存，但不显示	用 Load 语句，或者引用窗体上的属性或控件
装入并显示无模式窗体	用 Show 方法
装入并显示模式窗体	用 style = vbModal 的 Show 方法
显示已装入的窗体	设置它的 Visible 属性为 True，或者使用 Show 方法
从视窗中隐藏窗体	设置它的 Visible 属性为 False，或者使用 Hide 方法
从视窗中隐藏窗体并从内存中 卸载	用 Unload 语句

Show 方法装入窗体并设置它的 Visible 属性为 True。传递给 Show 方法的参数标识了该对话框的类型。如果 style 参数省略或者设置为 vbModeless 或 0（缺省），则对话框为无模式类型，如果 style 参数为 vbModal 或 1，则对话框为模式类型。

当选取“确定”或“取消”退出对话框时，可使用 Unload 语句或者 Hide 方法。例如：

```
Unload frmAbout
```

- 或者 -

frmAbout.Hide

Unload 语句把对话框从内存中删除，而 **Hide** 方法只是通过设置对话框的 **Visible** 属性为 **False** 将其从视窗中删除。当卸载窗体时，该窗体本身以及它的控件都从内存中卸载（包括在运行时装入的任何控件）。当隐藏窗体时，该窗体以及它的控件仍留在内存中。

当需要节省内存空间时，最好卸载窗体，因为卸载窗体可以释放内存。如果经常使用对话框，可以选取隐藏窗体。隐藏窗体仍可以保留与它关联的任何数据，包括属性值、打印输出和动态创建的控件。窗体被隐藏后，可以继续从代码中引用隐藏窗体的属性与控件。

各种显示类型的设计

Microsoft Windows 是与设备无关的一基于窗口的应用程序可以在许多不同显示分辨率与颜色浓度的计算机中运行。同样，用 **Visual Basic** 编写的应用程序也会在不同类型的显示器上运行，在设计应用程序时需要考虑这一点。

设计与分辨率无关的窗体

缺省情况下，当改变屏幕分辨率时，**Microsoft Visual Basic** 不会改变窗体与控件的尺寸。这就意味着在分辨率为 1024×768 的屏幕上设计的窗体，在分辨率为 640×480 的屏幕中运行时伸出屏幕的边界之外。如果想创建

不管使用什么样的屏幕分辨率都能有相同比例的窗体和控件，必须在最低的分辨率下设计窗体，或者将改变窗体的代码添加到程序中去。

避免尺寸问题的最简单的方法是在 640×480 的分辨率下设计窗体。如果更喜欢在高一些的分辨率下工作，仍需要考虑窗体在低一些的分辨率下将如何显示。实现这一点的方法是用“Form Layout”窗口预览窗体的大小和位置。您也可以使用“Resolution Guides”观察在低分辨率时屏幕的哪些部分是可见的。要切换到“Resolution Guides”，可以在“Form Layout”窗口单击鼠标右键，从弹出菜单上选择“Resolution Guides”菜单项。

在运行时，Visual Basic 根据设计时的位置来放置窗体。如果设计时在 1024×768 的分辨率上运行，并把窗体放到屏幕的右下角，则当它在比较低的分辨率下运行时该窗体可能看不见。为了避免这种情况的发生，在设计时可从“Form Layout”窗口的弹出菜单选择“Startup Position”菜单项来设置窗体的启动位置。同样，您也可以在运行时用下面的 Form Load 事件中的代码来设置窗体的位置：

```
Private Sub Form_Load()
```

```
    Me.Move 0, 0
```

```
End Sub
```

尽管设置窗体的 Left 与 Top 的属性都为 0 也能有同样的效果，但 Move 方法只要一步就能完成。

Visual Basic 使用与设备无关的度量单位，缇，是用来计算尺寸与位置的单位。Screen 对象的两个属性 TwipsPerPixelX 与 TwipsPerPixelY 可以用来确定运行时的显示尺寸。应用这些属性，可以编写代码来调整窗体以及控

件的尺寸与位置。

```
Private Sub SetControls()  
    Dim X As Integer  
    Dim Y As Integer  
  
    X = Screen.TwipsPerPixelX  
    Y = Screen.TwipsPerPixelY  
    Select Case X, Y  
        Case 15, 15  
            ' 重新改变控件尺寸及移动控件。  
            txtName.Height = 200  
            txtName.Width = 500  
            txtName.Move 200, 200  
            ' 增加为其它分辨率编写的代码。  
            ...  
    End Sub
```

也需要知道在设计时 Visual Basic 自身窗口的位置。如果在高分辨率下把“工程”窗口放置到屏幕的右边，那么会发现在低分辨率下打开工程时，它不再是可访问的了。

设计不同浓度的颜色

在设计应用程序时，也需要考虑运行应用程序计算机可能的颜色显示能力。有些计算机可以显示 256 色或更多的颜色，而另一些只能显示 16 种颜色。如果使用 256 色的调色板来设计窗体，那么在 16 色上显示时，抖动（模仿无效颜色的一种方法）会使窗体上的一些元素消失。

为了避免这种情况，最好把应用程序使用的颜色局限于 Windows 标准的 16 种颜色。这些颜色由 Visual Basic 的颜色常数来表示（如 `vbBlack`、`vbBlue` 和 `vbCyan` 等等）。如果在应用程序中需要用 16 种以上的颜色，那么对于文本、按钮以及其他界面元素仍然应当坚持用标准颜色。

设计时心里想着用户

除非创建 Visual Basic 应用程序完全只供自己使用，否则创作的价值只能由其他人来评价。应用程序的用户界面对用户有极大的影响——无论代码在技术上多么卓越，或者优化得多么的好，如果用户发现应用程序很难使用，那么他们就难于很好地接受它。

作为程序员，毫无疑问，对计算机方面技术已非常熟悉。因而很容易忘记大多数用户不理解（而且也许并不在意）隐藏在应用程序后面的技术。把应用程序看作达到目的工具：完成任务的方法想象中应比没有计算机的帮助更容易。一个设计得好的用户界面把用户与基础技术隔离开来，从而使

完成预定的任务变得很容易。

在设计应用程序用户界面的过程中，需要时时想到用户。如何能无需指导就发现应用程序的各种各样的功能？当错误发生时，应用程序如何响应？在帮助或辅助用户方面将提供些什么？设计是否以一种艺术美来使用户高兴？以上这些问题的答案以及其他有关用户为中心的设计问题，在本节中都将涉及到。

界面设计的基础

不必成为创建用户界面的艺术家——大多数用户界面设计的原则，与任意一门基础艺术课中所讲授的基础设计的原则相同。构图、颜色等的基本的设计原则，就像它们应用在纸张或油画上一样，也能很好地在一台计算机的屏幕上应用。

虽然 Visual Basic 能通过简单地将控件拖动并放置到窗体上而使得创建用户界面非常容易，但是，在设计之前稍微计划一下就能使应用程序的可用性有很大地改观。可能需先在纸上画出窗体开始设计，决定需要哪些控件，不同元素的相对重要性，以及控件之间的关系。

构图：应用程序的观感与感觉

窗体的构图或布局不仅影响它的美感，而且也极大地影响应用程序的可用性。构图包括诸如控件的位置、元素的一致性、动感、空白空间的使用以

及设计的简单性等因素。

控件的位置

在大多数界面设计中，不是所有的元素都一样重要。仔细地设计是很有必要的，以确保越是重要的元素越要很快地显现给用户。重要的或者频繁访问的元素应当放在显著的位置上，而不太重要的元素就应当降级到不太显著的位置上。

在大多数语言中我们习惯于在一页之中从左到右、自上到下地阅读。对于计算机屏幕也如此，大多数用户的眼睛会首先注视屏幕的左上部位，所以最重要的元素应当放在屏幕的左上部位。例如，如果窗体上的信息与客户有关，则它的名字字段应当显示在它最先被看到的地方。而按钮，如“确定”或“下一个”，应当放置在屏幕的右下部位；用户在未完成对窗体的操作之前，通常不会访问这些按钮。

把元素与控件分成组也很重要。尽量把信息按功能或关系进行逻辑地分组。因为他们的功能彼此相关，所以定位数据库的按钮应当被形象地分成一组，而不是分散在窗体的四处。对信息也是一样，名字字段与地址通常分在一组，因为它们联系紧密。在许多情况下，可以使用框架控件来帮助加强控件之间的联系。

界面元素的一致性

在用户界面设计中，一致性是一种优点。一致的外观与感觉可以在应用程

序中创造一种和谐，任何东西看上去都那么协调。如果界面缺乏一致性，则很可能引起混淆，并使应用程序看起来非常混乱、没有条理、价值降低，甚至可能引起对应用程序可靠性的怀疑。

为了保持视觉上的一致性，在开始开发应用程序之前应先创建设计策略和类型约定。诸如控件的类型、控件的尺寸、分组的标准以及字体的选取等设计元素都应该在事先确定。可以创建设计样板来帮助进行设计。

在 Visual Basic 中有大量的控件可供使用，这可能引起有人想使用所有的控件。为了避免这种引诱，选取能很好地适合特定应用程序的控件子集。虽然列表框、组合框、网格以及树等控件都可用来表示信息列表，最好还是尽可能使用一种类型。

还有，尽量恰当地使用控件，虽然 TextBox 控件可以设置成只读并用来显示文本，但 Label 控件通常更适用于该目的。在为控件设置属性时请保持一致性，如果在一个地方为可编辑的文本使用白色背景，除非有很好的理由，否则不要在别的地方又使用灰色。

在应用程序中不同的窗体之间保持一致性对其可用性有非常重要的作用。如果在一个窗体上使用了灰色背景以及三维效果，而在另一个窗体上使用白色背景，则这两个窗体就显得毫不相干。选定一种类型并在整个应用程序保持一致，即使这意味着要重新设计某些功能。

动感：窗体与其功能匹配

动感是对象功能的可见线索。虽然对这个术语也许还不熟悉，但动感的实例四处可见。自行车上的把手，手放在它的上面，动感会将把手用手扣紧

这件事显现出来。按下按钮、旋转旋钮和点亮电灯的开关等都能进行动感表示，一看到它们就可以看出其用处。

用户界面也使用动感。例如，用在命令按钮上的三维立体效果使得他们看上去像是被按下去的。如果设计平面边框的命令按钮的话，就会失去这种动感，因而不能清楚地告诉用户它是一个命令按钮。在有些情况下，平面的按钮也许是适合的，比如游戏或者多媒体应用程序；只要在整个应用程序中保持一致就很好。

文本框也提供了一种动感，用户可以期望带有边框和白色背景的框，框中包含可编辑的文本。显示不带边框的文本框 (`BorderStyle = 0`) 也有可能，这使它看起来像一个标签，并且不能明显地提示用户它是可编辑的。

空白空间的使用

在用户界面中使用空白空间有助于突出元素和改善可用性。空白空间不必非得是白色的——它被认为是窗体控件之间以及控件四周的空白区域。一个窗体上有太多的控件会导致界面杂乱无章，使得寻找一个字段或者控件非常困难。在设计中需要插入空白空间来突出设计元素。

各控件之间一致的间隔以及垂直与水平方向元素的对齐也可以使设计更可用。就像杂志中的文本那样，安排得行列整齐、行距一致，整齐的界面也会使其容易阅读。

Visual Basic 提供了几个工具，使得控件的间距、排列和尺寸的调整非常容易。“排列”、“按相同大小制作”、“水平间距”、“垂直间距”和“在窗体中央”等命令都可以在“格式”菜单中找到。

保持界面的简明

界面设计最重要的原则也许就是简单化。对于应用程序而言，如果界面看上去很难，则可能程序本身也很难。稍稍深入考虑一下便有助于创建看上去（实际上也是）用起来都很简单的界面。从美学的角度来讲，整洁、简单明了的设计常常更可取。

在界面设计中，一个普遍易犯的错误就是力图用界面来模仿真实世界的对象。例如，想象一下要求创建完整的保险单的应用程序。很自然的反应就是在屏幕上设计完全仿照保险单的界面。这样做会出现几个问题：保险单的形状与尺寸和屏幕上的有很大不同，要非常完善地复制这样的表格会将其限制在文本框与复选框中，而对用户并没有真正的好处。

最好是设计出自己的、也能提供原始保险单打印副本（带打印预览）的界面。通过从原始保险单中创建字段的逻辑组，并使用有标签的界面或几个链接的窗体，就可以不要求滚动屏幕而显示所有的信息。也可以使用附加的控件，比如带有选取预装入的列表框，这些控件可以减少打字工作量。

也可以取出不常用的函数并把它们移到它们自己的窗体中去，来简化许多应用程序。提供缺省有时也可以简化应用程序；如果十个用户中有九个选取加粗的文本，就把文本粗体设为缺省值，而不要叫用户每次都选取一遍（不要忘记提供一个选项可以覆盖该缺省值）。向导也有助于简化复杂的或不常用的任务。

简化与否最好的检验就是在应用中观察应用程序。如果有代表性的用户没有联机帮助就不能立即完成想要完成的任务，那么就需要重新考虑设计了。

使用颜色与图像

在界面上使用颜色可以增加视觉上的感染力，但是滥用的现象也时有发生。许多显示器能够显示几百万种颜色，这很容易使人要全部使用它们。如果在开始设计时没有仔细地考虑，颜色也会像其他基本设计原则一样，出现许多问题。

每个人对颜色的喜爱有很大的不同，用户的品味也会各不相同。颜色能够引发强烈的情感，如果正在设计针对全球读者的程序，那么某些颜色可能有文化上的重大意义。一般说来，最好保守传统，采用一些柔和的、更中性化的颜色。

当然，预期的读者以及试图传达的语气与情绪也会影响对颜色的选取。明亮的红色、绿色和黄色适用于小孩子使用的应用程序，但是在银行应用程序中它很难带来财务责任心。

少量明亮色彩可以有效地突出或者吸引人们对重要区域的注意。作为经验之谈，应当尽量限制应用程序所用颜色的种类，而且色调也应该保持一致。如果可能的话，最好坚持标准的 16 色的调色板；在 16 色显示器上观看时，抖动会使得其他一些颜色显示不出来。

使用颜色时另一个需要考虑的问题就是色盲。有一些人不能分辨不同的基色（如红色与绿色）组合之间的差别。对于有这种情况的人，绿色背景上的红色文本就会看不见。

图像和图标

图片与图标的使用也可以增加应用程序的视觉上的趣味，但是，细心的设计也是必不可少的。不用文本，图像就可以形象地传达信息，但常常不同的人对图像的理解也不一样。

带有表示各种功能的图标的工具栏，它是一种很有用的界面设备，但如果不能很容易地识别图标所表示的功能，反而会事与愿违。在设计工具栏图标时，应查看一下其它的应用程序以了解已经创建了什么样的标准。例如，许多应用程序用一张角上有卷边的纸来表示“新建文件”图标。也许还有更好的比喻来表示这一功能，但改用其它的表示方法会引起用户的混淆。

考虑图像文化上的意义也非常重要。许多程序使用田园风格的带一面旗的邮箱（图 6.21）来代表邮件功能。这原本是美国的图标；其他国家/地区或文化的用户也许不把它看作邮箱。



图 6.21 代表信箱的图标

在设计自己的图标与图像时，应尽量使它们简单。具有多种颜色的复杂的图片，作为 16×16 像素的工具栏图标，或者在高分辨率的屏幕上显示时，都不能很好地适应。

选取字体

字体也是用户界面的重要部分，因为它们常常给用户传递重要的信息。需选取在不同的分辨率和不同类型的显示器上都能容易阅读的字体的。最好尽

量坚持使用简单的无衬线字体或者衬线字体。通常手写字体或者其他装饰性字体的打印效果比屏幕上的效果更好，而且字体越小读起来越难。除非计划按应用程序来配置字体，否则应当坚持使用标准 Windows 字体，如 Arial、New Times Roman 或者 System。如果用户的系统没有包含指定的字体，系统会使用替代的字体，其结果可能与设想的完全不一样。如果正在为国际读者设计，需要调查在预想的语言里可用什么字体。还有，在为其他语言设计时，需要考虑文本的扩展——有些语言的文本串可以多占 50% 以上的空间。还有，在选取字体时，设计的一致性非常重要。大多数情况下，不应当在应用程序中使用两种以上字体。太多的字体使得应用程序看上去像罚款通知单。

可用性设计

任何应用程序的可用性基本上由用户决定。界面设计是需多次反复的过程；在为应用程序设计界面时，第一步就设计出非常完美的界面的情况非常少见。用户参与设计过程越早，花的气力越少，创建的界面越好、越可用。

什么是好的界面

设计用户界面时，开始时最好是先看看 Microsoft 或其他公司的一些卖得很好的应用程序。毕竟，界面很差的应用程序不会卖得很好。将会发现许

多通用的东西，比如：工具栏、状态条、工具提示、上下文菜单以及标记对话框。Visual Basic 具有把所有这些东西添加到应用程序中的能力，这并不偶然。

也可以凭借自己使用软件的经验。想一想曾经使用过的一些应用程序，哪些可以工作、哪些不可以以及如何修改它。但要记住个人的喜好不等于用户的喜好，必须把自己的意见与用户的意见一致起来。

还要注意到大多数成功的应用程序都提供选择来适应不同的用户的偏爱。例如，Microsoft Windows “资源管理器”允许用户通过菜单、键盘命令或者拖放来复制文件。提供选项会扩大应用程序的吸引力，至少应该使所有的功能都能被鼠标和键盘所访问。

Windows 界面准则

Windows 操作系统的主要的优点就是为所有的应用程序提供了公用的界面。知道如何使用基于 Windows 的应用程序的用户，很容易学会使用其他应用程序。而与已创建的界面准则相差太远的应用程序不易让人明了。

菜单就是这方面很好的例子——大多数基于 Windows 的应用程序都遵循这样的标准：“文件”菜单在最左边，然后是“编辑”、“工具”等可选的菜单，最右边是“帮助”菜单。如果说 Documents 会比 File 更好，或者“帮助”菜单要放在最前，这就值得讨论一下了。没有任何事情阻止您这样做，但这样做会引起用户的混淆，降低应用程序的可用性。每当在应用程序与其他程序之间切换时，用户都不得不停下来想一想。

子菜单的位置也很重要。用户本期望在“编辑”菜单下找到“复制”、“剪切”与“粘贴”等子菜单，若将它们移到“文件”菜单下会引起用户的混乱。不要偏离已经创建的准则太远，除非有很好的理由这样做。

可用性的检测

测试界面可用性的最好方法是在整个设计过程中请用户参与。不论是正在设计大型的压缩包应用程序，还是小型的有限使用的应用程序，设计的过程应当完全相同。使用已创建的设计准则，界面设计应从纸上开始。

下一步是创建一个或者多个原型，在 Visual Basic 中设计窗体。还需要增加足够的代码来启动原型：显示窗体、用示例数据填充列表框等等。然后准备可用性测试。

可用性测试可以是个不拘形式的过程：与用户一道审查设计；也可以是在已创建的可用性实验室中进行的正式的过程。这两种方法目的是一样的：从用户那儿了解哪儿设计得很好，哪儿还需要改进的第一手材料。放开，让用户与应用程序在一起，然后观察它们；这种方式比询问用户更为有效。当用户试图完成一系列任务时让他们表达其思考过程：“要想打开新文档，所以要在‘文件’菜单中找一找。”记下哪些地方的界面设计没有反应他们的思考过程。与不同类型的用户一起测试，如果发现用户完成某个特定的任务有困难，该任务可能需要多加关照。

下一步，复查一下记录，考虑如何修改该界面使它更加可用。修改界面并再测试。一旦对应用程序可用性满意，就准备开始编码。在开发的过程中

也需要不时地测试来确保对原型的设想是正确的。

功能的可发现性

可用性测试的关键的概念是可发现性。如果用户不能发现如何使用某个功能（或者甚至不知道有此功能存在），则此功能很少有人去使用。例如，Windows 3.1 的大多数用户都从来不知道 ALT 和 TAB 的组合键可以用于在打开的应用程序之间切换。界面中没有任何地方可提供线索来帮助用户发现这一功能。

为了测试功能的可发现性，不解释如何做就要求用户完成一个任务（例如，使用“窗体模板”创建新文档）。如果他们不能完成这个任务，或者尝试了好多次，则此功能的可发现性还需要改进。

当用户或系统出错时与用户交互

在理想世界里，软件与硬件都会无故障地一直工作下去，用户也从不出错。而现实中错误总是难免的。决定当事情出毛病时应用程序如何响应，是用户界面设计的一部分。

常用的响应是显示一个对话框，要求用户输入应用程序该如何处理这个问题。不太常用（但更好）的响应是简单地解决问题而不打扰用户。毕竟，用户主要关心的是完成任务，而不是技术细节。在设计用户界面时，考虑可能出现的错误，并判断哪一个需要用户交互作用，哪一个可以按事先安

排的方案解决。

创建容易理解的对话框

偶尔应用程序中会出现错误，需要为解决这种情况做出判断。这通常作为代码的分支出现——If...Then 语句或者 Case 语句。如果这个判断需要与用户交互，此问题通常用对话框来提交用户。对话框是用户界面的一部分，像界面的其他部分一样，它们的设计在应用程序可用性中发挥了作用。

有时有这样的感觉，好像许多程序对话框的设计员，不会讲使人容易理解的话。比如这样的消息：“硬盘 C 的扇区被损坏或不能访问。中止、重试、忽略？”（见图 6.22）这对一般的用户而言不大好理解。这等于有侍者问顾客：“我们没有汤或者厨房正在生火，中止、重试、忽略？”您会如何回答呢？以用户能理解的方式或短语描述问题（和选择）是重要的。在前面的例子中，更好的消息可以是“在 C 盘上存文件有问题，请把文件存于 A 盘。存不存文件？”



图 6.22 哪个对话框表示的消息最明确

当为应用程序创建对话框时，心里想着用户。这个消息给用户传达了有用的信息吗？它容易理解吗？命令按钮表示的选择明确吗？这选择适合给定的条件吗？记住，仅仅一个讨厌的消息框就会使用户对应用程序产生坏印象。

如果正在设计自定义对话框，尽量坚持用标准类型。如果与标准消息框布局相差太远，用户可能不会把它认作是对话框。

详细信息 关于对话框的详细内容，请参阅本章前面的“对话框”。

不用对话框的错误处理

当错误出现时不一定要打断用户。有时更可取的是不通知用户而用代码来处理错误，或者以不停止用户工作流程的方法来提醒用户。这个技术的很好的例子是 Microsoft Word 中的“自动更正”功能：如果普通单词拼错了，Word 自动修改它；如果不常用单词拼错了，在其下划一条红线提醒用户以后改正。

有大量的技术可以使用；哪些技术适用于应用程序应由自己决定。这里有几个建议：

- 在“编辑”菜单中添加“撤销”功能。对于删除等情况，与其用“确定”对话框来打断用户，还不如确保他们作出正确的决定并提供“撤销”功能以备他们以后改变主意。
- 在状态栏或图标上显示消息。如果错误不影响用户当前的任务，不要停止应用程序。使用状态栏或亮色警告图标来警告用户，当他们准备好后可以处理该问题。
- 改正问题。有时错误的解决办法很显然。例如，当用户试图存文件时磁盘已满，则在其他驱动器中检查系统寻找空间。如果空间可用，则保存该文件；在状态栏中显示一条消息告诉用户做了些什么。
- 保存消息等候处理。因为不是所有的错误都是紧要的，或要求马上注意的；考虑把这些记录到文件中，当用户退出应用程序时或其他方便的时

候再把它们显示给用户。如果用户发生输入错误（如：把 Main St. 写成 Mian St.），记录它。添加“Review Entries”按钮和显示差异的函数，以便用户可以改正它们。

- 不要做任何事。有时错误并不重要，不足以成为警告的原因。例如，LPT1 上的打印机的纸张没准备好这一事实，在准备打印之前并没有多大关系。等待，直到消息合乎当前的任务。

详细信息 关于错误处理技术的详细内容，请参阅第十三章“调试代码与处理错误”。

设计用户辅助模式

不论用户界面设计得多么好，有时用户总需要帮助。应用程序的用户辅助模式包括诸如联机帮助和打印出来的文档等东西；它也可以包括用户辅助设备，如工具提示、状态条、“这是什么”帮助以及向导。

像应用程序的其他任何部分一样，用户辅助模式设计应当在开始开发之前。模式的内容将随着应用程序的复杂程度与预期读者的不同而不同。

帮助与文档

联机帮助是任何应用程序的重要部分，它通常是用户有问题时最先查看的地方。甚至简单的应用程序也应该提供“帮助”。不提供它就好像是假定用户从来不会有问题。

在设计“帮助”系统时，记住它的主要目的是回答问题。创建主题名称与索引条目时尽量用用户的术语，例如，“我如何格式化页面？”比“编辑”，“页格式”菜单更容易找到主题。不要忘记上下文相关性；对大多数用户而言，如果他们按下 F1 键寻求一指定字段的帮助，却发现自己在内容主题上，则他们会感到受挫折。

基本概念的文档，不管是打印的和/或由压缩盘提供的，对所有的应用程序都是有帮助的，除了最简单的以外。它可以提供那些用简短的“帮助”主题难以传达的信息。至少，应该在 ReadMe 文件窗体中提供用户需要时可以打印的文档。

用户辅助设备

在用户界面中，有几种对用户提供辅助的技术。用 Visual Basic 在应用程序中添加工具提示、“这是什么”帮助、状态显示和向导是很容易的。这些设备中的哪些适用于自己的应用程序应由自己决定。

工具提示

当用户在用户界面上搜索时，工具提示（图 6.23）是一种向他们显示信息的好方法。工具提示是个小标签，当鼠标指针在控件上停留会儿即显示，通常包含此控件的功能描述。正常情况下工具提示与工具栏结合使用，它在界面的大多数部分也能很好工作。



图 6.23 Visual Basic 工具栏的工具提示

大多数 Visual Basic 控件都包含用来显示工具提示的属性：ToolTipText。以下代码将对名为“cmdPrint”的命令按钮提供工具提示。

```
cmdPrint.ToolTipText = "Prints the current document"
```

像界面的其它部分一样，要确保此文本能明确地传达给用户的消息。

详细信息 关于工具提示的详细内容，请参阅《语言参考》的“ToolTipText 属性”。

“这是什么”帮助

当用户选取“这是什么”帮助并单击控件上的“这是什么”光标，“这是什么”帮助提供了和弹出式“帮助”主题（见图 6.24）的链接。“这是什么”帮助可以从工具栏按钮、菜单项或者对话框的标题栏上的按钮启动。



“拼写”按钮（“常用”工具栏）

检查活动文档中的拼写。检查范围包括页眉、页脚、脚注、尾注和批注。

图 6.24 “这是什么”帮助的弹出式窗口

要从菜单或工具栏使“这是什么”帮助有效，请按照以下步骤执行：

1. 选取希望为其提供帮助的控件。
2. 在“属性”窗口中，选取 WhatsThisHelpID 属性。
3. 为相关的弹出式“帮助”主题输入上下文标识符号。
4. 为任何其他控件重复步骤 1 到步骤 3。
5. 选取窗体。
6. 在“属性”窗口中，设置该窗体的 WhatsThisHelp 属性为 True。
7. 在菜单或工具栏按钮的 Click 事件中，键入以下代码：

```
formname.W hatsThisHelp
```

当用户单击该按钮或菜单时，鼠标指针会改变为“这是什么”帮助指针。为了使在自定义对话框体的标题栏上的“这是什么”帮助有效，设置该窗体的 WhatsThisButton 与 WhatsThisHelp 属性为 True。

详细信息 关于“这是什么”帮助的详细内容，请参阅《语言参考》的“WhatsThisHelp 属性”与“WhatsThisButton 属性”。

状态显示

状态显示也可用与工具提示差不多的方法来提供用户辅助设备。状态显示是提供那些不太适合工具提示的指令或消息的一种好方法。包括在 Visual Basic 的专业版与企业版中的状态条控件能很好地显示消息；Label 控件也能用作状态显示。

在状态显示中显示的文本可以用以下两种方法中的一种来更新：用控件或窗体的 GotFocus 事件，或者用 MouseMove 事件。如果想把显示用作学习设备，在 Help 菜单中添加一个项目来转换其 Visible 属性的开与关。

要添加状态显示，请按照以下步骤执行：

1. 在窗体中添加 Label 控件。
2. 选取希望为其显示消息的那个控件。
3. 在控件的 MouseMove（或 GotFocus）事件中添加以下代码：

```
Labelname.Caption = "Enter the customer's ID number in this field"
```

当鼠标移到该控件上时，这条消息将显示在此 Label 控件中。

4. 为任何其它的控件重复步骤 2 到步骤 3。

向导

向导是一种用户辅助设备，它引导用自己的实际数据一步一步地实现一个过程。向导通常用来提供任务专用辅助。它们帮助完成需要相当长的（而且令人讨厌的）学习过程的任务，它们给还没有成为专家的用户提供专家信息。

Visual Basic 的专业版与企业版包括了创建向导的工具：向导管理器。

详细信息 关于向导的详细内容，请参阅第四章“工程的管理”中的“使

用向导与外接程序”。

第七章 使用 Visual Basic 的标准控件

控件用来获取用户的输入信息和显示输出信息。应用程序中可用的控件包括文本框、命令按钮和列表框。而通过另外一些控件可访问其它应用程序并处理数据，这时，那些远程应用程序就好象是代码的一部分。每个控件都有一组属性、方法和事件。本章介绍 Visual Basic 中的标准控件。

详细信息 关于 Visual Basic 专业版和企业版中提供的 ActiveX 控件的详细信息，请参阅《部件工具指南》中的“使用 ActiveX 控件”一章。

目录

- Visual Basic 控件简介
- 通过限制焦点验证控件数据
- 使用控件数组
- 使用 ADO 数据控件
- 使用 Check Box 控件
- 使用 Combo Box 控件
- 使用 Command Button 控件
- 使用 Common Dialog 控件

- 使用 Data 控件
- 使用 DataCombo 和 DataList 控件
- 使用 DataGrid 控件
- 使用 文件系统控件 (Directory ListBox、 DriveListBox 和 FileListBox)
- 使用 Frame 控件
- 使用 Hierarchical FlexGrid 控件
- 使用 HScrollBar 和 VScrollBar 控件
- 使用 Image 控件
- 使用 Label 控件
- 使用 Line 控件
- 使用 ListBox 控件
- 使用 OLE 容器控件
- 使用 OptionButton 控件
- 使用 PictureBox 控件
- 使用 RemoteData 控件
- 使用 Shape 控件
- 使用 TextBox 控件
- 使用 Timer 控件

示例应用程序: Alarm.vbp、Calc.vbp、Controls.vbp、Flex.vbp、Winseek.vbp

本章中许多代码示例源于 Alarm.vbp、Calc.vbp、Controls.vbp、Flex.vbp 和 Winseek.vbp 示例应用程序。这些示例应用程序列在 Samples 目录中。

Visual Basic 控件简介

可用 Visual Basic 工具箱中的工具在窗体上绘制控件。



图 7.1 Visual Basic 工具箱

控件分类

Visual Basic 的控件有三种广义分类：

- 内部控件，例如 `CommandButton` 和 `Frame` 控件。这些控件都在 Visual Basic 的 `.exe` 文件中。内部控件总是出现在工具箱中，不象 ActiveX 控件和可插入对象那样可以添加到工具箱中，或从工具箱中删除。
- ActiveX 控件，是扩展名为 `.ocx` 的独立文件，其中包括各种版本 Visual

Basic 提供的控件（DataCombo, DataList 控件等等）和仅在专业版和企业版中提供的控件（例如 ListView、Toolbar、Animation 和 Tabbed Dialog），另外还有许多第三方提供的 ActiveX 控件。

注意 具有文件扩展名 .vbz 的控件使用了老的技术，在 Visual Basic 的早期版本编写的应用程序中可以找到这些控件。当 Visual Basic 打开包含 .vbz 控件的工程时，在缺省情况下用 .ocx 控件取代 .vbz 控件，当然，这只有在控件的 .ocx 版本存在时才可以。关于将控件更新为 .ocx 格式的信息，请参阅本章后面的“更新旧版 Visual Basic 控件”一节。

- 可插入的对象，例如一个包含公司所有雇员的列表的 Microsoft Excel 工作表对象，或者一个包含某工程计划信息的 Microsoft Project 日历对象。因为这些对象能添加到工具箱中，所以可把它们当作控件使用。其中一些对象还支持自动化（正式的名称为 OLE 自动化），使用这种控件就可在 Visual Basic 应用程序中编程控制另一个应用程序的对象。关于自动化的详细信息，请参阅第十章“用部件编程”中的“使用对象编程”。

详细信息 关于 Visual Basic 专业版和企业版中提供的 ActiveX 控件的详细信息，请参阅《部件工具指南》中的“使用 ActiveX 控件”。

内部控件

下表总结了 Visual Basic 工具箱中的内部控件。

图 标	控 件 名	类 名	描 述
	复 选 框	CheckBox	显示 True/False 或 Yes/No 选项。一次可在窗体上选定任意数目的复选框
	组 合 框	ComboBox	将文本框和列表框组合起来。用户可以输入选项，也可从下拉式列表中选择选项
	命 令 按 钮	CommandButton	在用户选定命令或操作后执行它
	数 据	Data	能与现有数据库连接并在窗体上显示数据库中的信息
	目 录 列 表 框	DirListBox	显示目录和路径并允许用户从中进行选择
	驱 动 器 列 表 框	DriveListBox	显示有效的磁盘驱动器并允许用户选择

续表

	文件列表框	<code>FileListBox</code>	显示文件列表并允许用户从中进行选择
	框架	<code>Frame</code>	为控件提供可视的功能化容器
	水平和垂直滚动条	<code>HScrollBar</code> 和 <code>VScrollBar</code>	对于不能自动提供滚动条的控件，允许用户为它们添加滚动条（这些滚动条与许多控件的内建滚动条不同）
	图像	<code>Image</code>	显示位图、图标或 Windows 图元文件、JPEG 或 GIF 文件；单击时类似命令按钮
	标签	<code>Label</code>	为用户显示用户不可交互操作或不可修改的文本

续表



线形

Line

在窗体上添加线段

列表框

ListBox

显示项目列表，用户可从中进行选择



OLE 容器

OLE

将数据嵌入到 Visual Basic 应用程序中



选项按钮

OptionButton

选项按钮控件与其它选项按钮组成选项组，用来显示多个选项，用户只能从中选择一项



图片框

PictureBox

显示位图、图标或 Windows 图元文件、JPEG 或 GIF 文件。也可显示文本或者充任其它控件的可视容器

续表



形状

Shape

向窗体、框架或图片框添加矩形、正方形、椭圆或圆形

文本框

TextBox

提供一个区域来输入文本、显示文本

定时器

Timer

按指定时间间隔执行定时器事件

注意 指针工具（工具箱中的第一个工具）可用来移动窗体和控件，并调整它们的大小。指针工具不是控件。

标准 ActiveX 控件

Visual Basic 学习版包含若干 ActiveX 控件（称为标准 ActiveX 控件），有了这些控件就可在应用程序中引入高级功能。ActiveX 控件的文件扩展名为 .ocx，可手工将它们添加到工具箱中，以便在工程中使用 `aq`。下表总结了 Visual Basic 学习版提供的标准 ActiveX 控件。

图 标	控 件 名	类 名	描 述
	公 共 对 话 框	CommonDialog	提供一组标准对话框，用于打开和保存文件，设置打印选项、选择颜色和字体等操作。
	数 据 组 合	DataCombo	除了提供标准组合框控件的大多数功能之外还提供附加的数据访问功能。
	数 据 列 表	DataList	除了提供标准列表框控件的大多数功能之外还提供附加的数据访问能力。
	Microsoft Hierarchical FlexGrid	MSFlexGrid	一个可以绑定到数据环境设计器以显示层次结构记录集的只读网格控件。

详细信息 关于 Visual Basic 专业版和企业版中提供的 ActiveX 控件的详细信息，请参阅《部件工具指南》中的“使用 ActiveX 控件”。

添加和删除 ActiveX 控件

按照下述步骤在工具箱中添加或删除 ActiveX 控件。

要在工具箱中添加 ActiveX 控件：

1. 在“工程”菜单中选择“部件”。
2. 选定 .ocx 控件名旁边的复选框，然后选择“确定”。将控件放入工具箱后，就象对待内部控件那样将它们添加到窗体上。

要删除 ActiveX 控件：

1. 在工程的窗体上删除控件的所有实例。删除工程代码中对控件的所有引用。若代码中还留有被删除控件的引用，则在编译应用程序时将显示出错信息。
2. 在“工程”菜单中选择“部件”。
清除 .ocx 控件名旁边的复选框，然后选择“确定”。若工程中还有控件的实例，则将显示出错信息。

详细信息 关于在工具箱中添加或删除控件和可插入对象的详细信息，请参阅第四章，“工程的管理”中的“在工程中添加控件”。

更新旧版 Visual Basic 控件

文件扩展名为 .vb3 的 16 位旧版 Visual Basic 控件与此版 Visual Basic 不兼容。若试图加载包含 .vb3 控件的旧工程，Visual Basic 将会警告说控件不

适用或不兼容，此时，可以继续加载无 .vb_x 控件的工程，当然，应用程序将无法正常运行。

如果手中有包含第三方 .vb_x 控件的旧版本 Visual Basic 的工程，请向控件厂商咨询有关 .ocx 替代控件的信息。

控件命名约定

第一次创建对象（窗体或控件）时，Visual Basic 将其 Name 属性设置为缺省值。例如，最初将所有命令按钮的 Name 属性都设置为 Command_n，其中 n 为 1、2、3 等等。Visual Basic 将绘制在窗体上的第一个命令按钮命名为 Command1，第二个为 Command2，第三个为 Command3。

可以保留缺省名称；但是，如果有几个同类型控件时，最好将 Name 属性改成具有描述性的名称。因为很难区分 MyForm 窗体的 Command1 按钮与 YourForm 的 Command1 按钮，所以命名约定十分有益，特别是当应用程序包含了若干窗体、标准和类模块的时候。

可以这样为控件命名：用前缀描述类，其后为控件的描述性名称。使用了命名约定，代码就可自动描述自己并使相似的对象在对象列表框中按字母顺序排列。例如，可以这样来为 CheckBox 控件命名：

chkReadOnly

为窗体和控件命名的名称：

- 必须以字母开头。

- 只能包含字母、数字和下划线字符 (_)；不允许有标点符号字符和空格。
- 不能超过 40 个字符。

详细信息 关于命名约定的详细信息，请参阅附录 B，“Visual Basic 编码约定”。

使用控件值

所有控件都有一个属性，只需引用控件名而无需使用属性名就可利用这个属性来存储或获取数值，称此属性为控件的值，这是控件的最重要或最常用属性。下表列出了每个控件属性，这些属性被看成是控件值。

控件	值
Check Box（复选框）	Value
Combo Box（组合框）	Text
Command Button（命令按钮）	Value
Common Dialog（公共对话框）	Action
Data（数据）	Caption
DataCombo（数据组合）	Text
DataGrid（数据网格）	Text
DataList（数据列表）	Text
Dir List Box（目录列表框）	Path
Drive List Box（驱动器列表框）	Drive
File List Box（文件列表框）	FileName

续表

FlexGrid	Text
Frame (框架)	Caption
HScrollBar (水平滚动条)	Value
Image (图像)	Picture
Label (标签)	Caption
Line (线形)	Visible
List Box (列表框)	Text
Option Button (选项按钮)	Value
Picture Box (图片框)	Picture
Shape (形状)	Shape
Text Box (文本框)	Text
Timer (定时器)	Enabled
VScrollBar (垂直滚动条)	Value

当控件的属性为该控件的值时，无论何时引用这个属性都不必在代码中指定属性名。例如，下行代码设置了 `TextBox` 控件的 `Text` 属性值：

```
Text1 = "This text is assigned to the Text property _  
of Text1"
```

在下例中，只要用户单击文件列表框中的文件，`Label1` 的 `Caption` 属性就被设置成 `File1` 的 `FileName` 属性：

```
Private Sub File1_Click ()  
    Label1 = File1
```

End Sub

注意 由于使用了控件值，代码的可读性略受影响，所以本指南中的示例不使用控件值，而是明确引用控件的属性。两种方法均可用来编写代码，在不引起阅读困难时可考虑使用控件值。

通过限制焦点验证控件数据

`Validate` 事件和 `CausesValidation` 属性是在允许用户将焦点移开控件之前，串联使用以确认对控件的输入的。例如：假设有几个文本框和一个 `Help` 按钮的应用程序，当每个文本框接收焦点时，您想在文本框的特殊验证准则被满足之前阻止用户移动焦点；然而，您也想允许用户在任何时候都能单击 `Help` 按钮。要做到这点，在 `Validate` 事件中设置验证准则，并将 `Help` 按钮的 `CausesValidation` 属性设置为 `False`。假如属性设置为 `True` (缺省设置)，`Validate` 事件将在第一个控件上发生。假如属性设置为 `False`，在第一个控件上的 `Validate` 事件将抢先发生。

`Validate` 事件比 `LostFocus` 事件更适合验证数据输入项，因为 `LostFocus` 事件(按照定义)是在焦点已移动后发生。相反，通过使用 `Validate` 事件，您能防止焦点移动到另一个控件直到验证规则被满足。

可能的使用

- 数据输入项应用程序需要执行比 `Masked Edit` 控件所提供的或在业务规则中发生的有效性验证更复杂的数据输入项有效性验证。

- 窗体需要防止用户使用 **TAB** 键或加速键将控件移走，直到数据已被输入到字段中。
- 在 **Internet Explorer** 中运行的 **ActiveX** 文档需要一种方法，使用户在脚本程序化地移动焦点之前完成在窗体上的操作。

在 **Validate** 事件上控制焦点

Validate 事件包括一个 **keepfocus** 参数。当参数设置为 **True** 时，控件将保留焦点。这样做有效地防止了用户单击其它控件。

使用控件数组

控件数组是一组具有共同名称和类型的控件。它们的事件过程也相同。一个控件数组至少应有一个元素，元素数目可在系统资源和内存允许的范围內增加；数组的大小也取决于每个控件所需的内存和 **Windows** 资源。在控件数组中可用到的最大索引值为 32767。同一控件数组中的元素有自己的属性设置值。常见的控件数组的用处包括实现菜单控件和选项按钮分组。

注意 **Visual Basic** 包括了在运行时动态地将未引用的控件添加到 **Controls** 集合中的能力。本主题仅指在设计时通过将一个控件剪切和粘贴到窗体上添加的引用控件。有关在运行时添加控件的详细信息，请参阅参考主题“**Add 方法 (Controls 集合)**”和“**Add 方法 (Licenses 集合)**”。

为何使用控件数组

在设计时，使用控件数组添加控件所消耗的资源比直接向窗体添加多个相同类型的控件消耗的资源要少。当希望若干控件共享代码时，控件数组也很有用。例如，如果创建了一个包含三个选项按钮的控件数组，则无论单击哪个按钮时都将执行相同的代码。

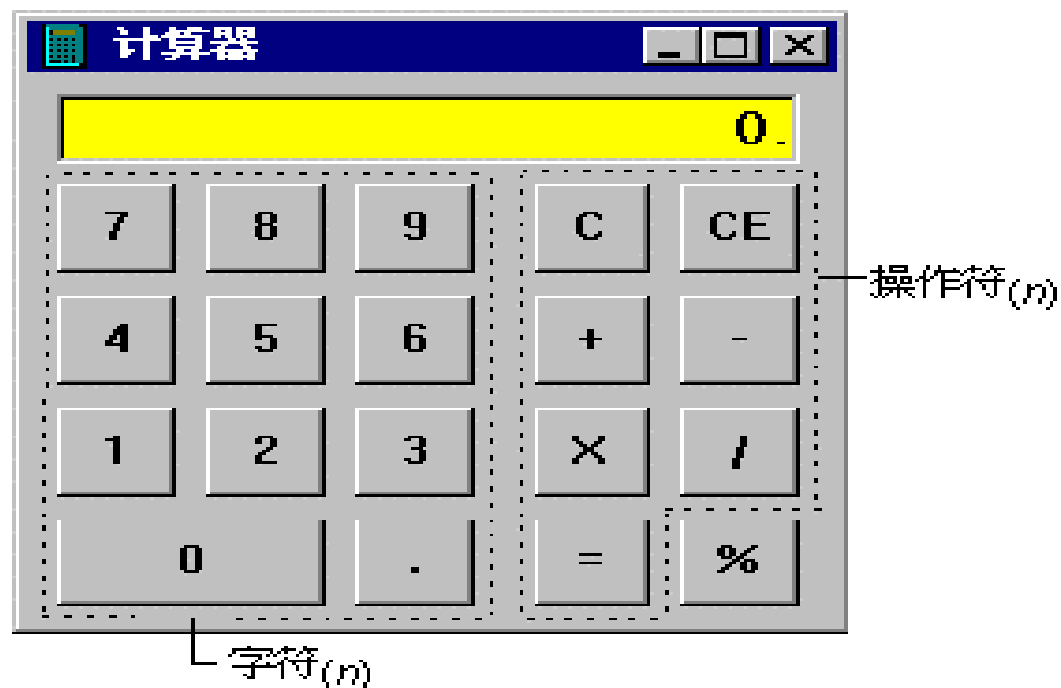
若要在运行时创建一个控件的新实例，则新控件必须是控件数组的成员。

使用控件数组时，每个新成员继承数组的公共事件过程。

使用控件数组机制是不可能运行时创建新控件的，因为每个新控件都继承为数组编写好的事件过程。例如，如果窗体上有若干文本框，而且每个文本框都接受一个日期数值，则可创建一个控件数组，使所有文本框共享同一个合法性检查代码。

示例应用程序：Calc.vbp

图 7.2 所示的 Calculator 示例应用程序（列在 `Samples!Alink("vbsamples")` 目录中）包含两个控件数组——数字按钮和操作按钮。



7.2 控件数组示例

下表列出 Calculator 示例中控件数组的 Name 和 Index 属性值。

数字 (n)	操作符 (n)
0 = Number(0)	+= Operator(1)
1 = Number(1)	-= Operator(2)
2 = Number(2)	X = Operator(3)
3 = Number(3)	/ = Operator(4)
4 = Number(4)	= = Operator(5)

续表

```
5 = Number(5)
```

```
6 = Number(6)
```

```
7 = Number(7)
```

```
8 = Number(8)
```

```
9 = Number(9)
```

请注意示例是如何使用 `object(index)` 语法引用每个控件的。在创建控件时指定索引值。实际上，在设计时为一个控件指定任意的索引，这将使控件成为数组的一部分。

`Index` 属性区分控件数组中的元素。当数组中的一个控件识别了一个事件时，`Visual Basic` 将调用公共事件过程并传递一个参数（`Index` 属性的值），分辨是哪个控件识别事件。

例如，`Number_Click` 事件过程的第一行代码是这样的：

```
Private Sub Number_Click (Index As Integer)
```

如果 `Number(0)` 识别事件，则 `Visual Basic` 将 0 作为 `index` 参数传递，而如果 `Number(1)` 识别事件，则 `Visual Basic` 将 1 作为 `index` 参数传递。与索引值不同，对于 `Number(0)` 到 `Number(9)` 已执行的其余 `Number_Click` 代码都是相同的。

设计时创建控件数组

设计时有三种方法创建控件数组：

- 将相同名字赋予多个控件。
- 复制现有的控件并将其粘贴到窗体上。
- 将控件的 `Index` 属性设置为非 `Null` 数值。

注意 必须在“菜单编辑器”中创建菜单控件数组。关于此项操作的详细信息，请参阅第六章，“创建用户界面”中的“运行时创建和修改菜单”。

要通过改变控件名称添加控件数组元素：

1. 绘制控件数组中要添加的控件（必须为同一类型的控件）决定哪一个控件作为数组中的第一个元素。
2. 选定控件并将其 `Name` 设置值变成数组第一个元素的 `Name` 设置值。
3. 在数组中为控件输入现有名称时，`Visual Basic` 将显示一个对话框，要求确认是否要创建控件数组。此时选择“确定”确认操作。

例如，若控件数组第一个元素名为 `cmdCtlArr`，则选择一个 `CommandButton` 将其添加到数组中，并将其名称设置为 `cmdCtlArr`，此时将显示这样一段信息：“已经存在名为 '`cmdCtlArr`' 的控件。是否要创建控件数组？”。选择“确定”确认操作。

用这种方法添加的控件仅仅共享 `Name` 属性和控件类型；其它属性与最初绘制控件时的值相同。

要通过复制现存控件添加控件数组元素：

1. 绘制控件数组中的控件。
2. 当控件获得焦点时，选择“编辑”菜单中的“复制”命令。

3. 在“编辑”菜单中，选择“粘贴”命令。Visual Basic 将显示一个对话框询问是否确认创建控件数组。选择“确定”确认操作。

指定给控件的索引值为 1。绘制的第一个控件具有索引值 0。

每个新数组元素的索引值与其添加到控件数组中的次序相同。这样添加控件时，大多数可视属性，例如高度、宽度和颜色，将从数组中第一个控件复制到新控件中。

运行时添加控件数组

在运行时，可用 Load 和 Unload 语句添加和删除控件数组中的控件，然而，添加的控件必须是现有控件数组的元素。必须在设计时创建一个（在大多数情况下）Index 属性为 0 的控件，然后在运行时使用如下语法：

Load *object(index%)*

Unload *object(index%)*

参 数	描 述
<i>object</i>	在控件数组中添加或删除的控件名称
<i>index%</i>	控件在数组中的索引值

加载控件数组的新元素时，大多数属性设置值将由数组中具有最小下标的

现有元素复制——本例中是索引值为 0 的元素。因为不会自动把 Visible、Index 和 TabIndex 属性设置值复制到控件数组的新元素中，所以，为了使新添加的控件可见，必须将其 Visible 属性设置为 True。

注意 试图对数组中已存在的索引值使用 Load 语句时，Visual Basic 将生成一个错误。

重点 可用 Unload 语句删除所有由 Load 语句创建的控件，然而，Unload 无法删除设计时创建的控件，无论它们是否是控件数组的一部分。

控件方案：在控件数组中添加和删除控件

如何在运行时添加和删除控件，控件数组示例对此作了演示，这里，控件是选项按钮。根据这个示例，用户可以添加选项按钮，改变图片框背景颜色。

如图 7.3 所示，启动窗体，然后在上面绘制一个图片框、一个标签、两个选项按钮和三个命令按钮。

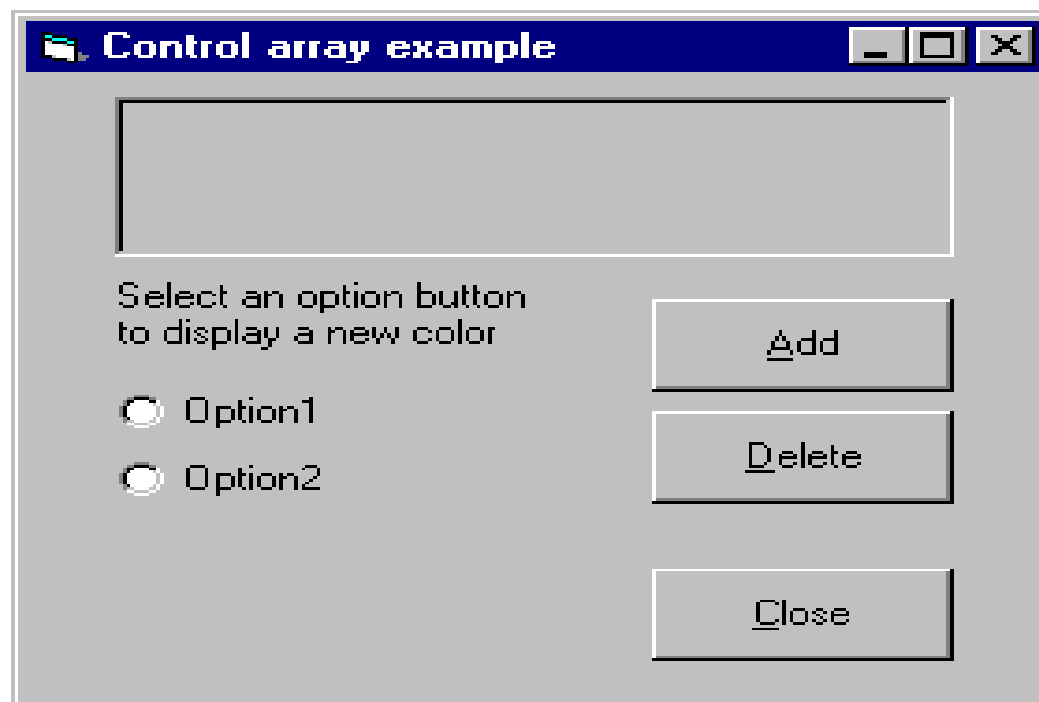


图 7.3 在运行时添加控件

下表列出了应用程序中对象的属性设置值。

对象	属性	设置值
Form	Caption	Control Array Example
Picture Box	Name	picDisplay
Label	Caption	select an option button to display a new color

续表

Option1	Name	optButton
	Index	0
Option2	Name	optButton
	Index	1
第一个 CommandButton	Name	cmdAdd
	Caption	&Add
第二个 CommandButton	Name	cmdDelete
	Caption	&Delete
第三个 CommandButton	Name	cmdClose
	Caption	&Close

控件数组应用程序的事件

接着，必须添加选项按钮和命令按钮的事件过程。添加窗体声明后启动应用程序：

```
Dim MaxId As Integer
```

所有选项按钮共享 Click 事件过程：

```
Private Sub optButton_Click (Index As Integer)
```

```
    picDisplay.BackColor = QBColor(Index + 1)
```

```
End Sub
```

通过“添加”命令按钮的 Click 事件过程添加新的选项按钮。本例中，在

执行 Load 语句前，代码将检查确认加载的选项按钮数不超过十个。加载控件之后，必须将其 Visible 属性设置为 True。

```
Private Sub cmdAdd_Click ()  
    If MaxId = 0 Then MaxId = 1      '设置全部选项按钮。  
    If MaxId > 8 Then Exit Sub      '只允许十个按钮。  
    MaxId = MaxId + 1              '按钮计数递增。  
    Load optButton(MaxId)          '创建新按钮。  
    optButton(0).SetFocus           '重置按钮选项。  
    '将新按钮放置在上一个按钮下方。  
    optButton(MaxId).Top = optButton(MaxId - 1)._  
    Top + 400  
    optButton(MaxId).Visible = True '显示新按钮。  
    optButton(MaxId).Caption = "Option" & MaxId + 1  
End Sub
```

通过“删除”命令按钮的 Click 事件过程删除选项按钮：

```
Private Sub cmdDelete_Click ()  
    If MaxId <= 1 Then Exit Sub '保留最初的两个按钮。  
    Unload optButton(MaxId)     '删除最后的按钮。  
    MaxId = MaxId - 1           '按钮计数递减。  
    optButton(0).SetFocus       '重置按钮选项。  
End Sub
```

通过“关闭”按钮的 Click 事件过程结束应用程序：

```
Private Sub cmdClose_Click ()  
    Unload Me  
End Sub
```

使用 ADO Data 控件

ADO Data 控件使用 Microsoft ActiveX 数据对象 (ADO) 来快速建立数据绑定的控件和数据提供者之间的连接。数据绑定控件是任何具有“数据源”属性的控件。数据提供者可以是任何符合 OLE DB 规范的数据源。使用 Visual Basic 的类模块也可以很方便地创建子集的数据提供者。尽管可以在应用程序中直接使用 ActiveX 数据对象，但 ADO Data 控件有作为一个图形控件的优势（具有“向前”和“向后”按钮），以及一个易于使用的界面，使您可以用最少的代码创建数据库应用程序。



图 7.4 ADO Data 控件

在 Visual Basic 的“工具箱”中，不少控件都可以作为数据绑定的控件，包括复选框、组合框、图像、标签、列表框、图片框以及文本框控件等。此外，Visual Basic 还包括了若干种数据绑定的 ActiveX 控件，诸如 DataGrid、DataCombo、Chart 以及 DataList 控件等。用户也可以创建自己的数据绑定的 ActiveX 控件，或从其他开发商购买控件。

Visual Basic 以前的版本提供了内在的 Data 控件和 Remote Data 控件 (RDC) 来进行数据访问。这两种控件仍包括在 Visual Basic 中，以提供向后兼容。不过，因为 ADO 的适应性更广，因此建议用户使用 ADO Data 控件来创建新的数据库应用程序。

详细信息 关于数据绑定的控件的完整列表位于“绑定到 ADO Data 控件的控件”中。要了解如何使用这些内在的 Data 控件或 Remote Data 控件，请参阅“使用 Data 控件”或“使用 Remote Data 控件”。关于创建数据提供者的详细信息，请参阅“创建数据识别类”。

可能的用法

- 连接一个本地数据库或远程数据库。
- 打开一个指定的数据库表，或定义一个基于结构化查询语言 (SQL) 的查询、或存储过程、或该数据库中的表的视图的记录集合。
- 将数据字段的数值传递给数据绑定的控件，可以在这些控件中显示或更改这些数值。
- 添加新的记录，或根据对显示在绑定的控件中的数据的所有更改来更新一个数据库。

要创建一个客户或前端数据库应用程序，应在窗体中添加 ADO Data 控件，以及其它所需要的任何 Visual Basic 控件。可以根据您的需要在窗体中放置多个 ADO Data 控件。不过，请注意，这种控件是一种相当“昂贵”的创建连接的方法，应在第一个控件至少有两个连接，且以后的每个控件至少多一个连接时使用。

用最少的代码创建一个前端数据库应用程序

通过在设计时设置一些属性，可以用最少的代码来创建一个数据库应用程序。如果正在使用 OLE DB 数据源，则相应的 Microsoft 数据链接名称 (Data Link Name) (.MDL) 必须是在您的机器上创建的。请参阅“创建 Northwind 的 OLE DB Data Link”，以获得一个循序渐进的示例。

要创建一个简单的前端数据库应用程序：

1. 在窗体上放置一个 ADO Data 控件（该图标的工具提示为 "ADODC"）。如果该控件不在“工具箱”中，请按 CTRL+T 键，显示“部件”对话框。在这个“部件”对话框中，单击“Microsoft ADO Data Control”。
2. 在“工具箱”中，单击选定“ADO Data 控件”。然后按 F4 键显示“属性”窗口。
3. 在“属性”窗口中，单击“ConnectionString”显示“ConnectionString”对话框。
4. 如果您已经创建了一个 Microsoft 数据链接文件（Data Link file）(.MDL)，请选择“使用 OLE DB 文件”并单击“浏览”，以找到计算机上的文件。如果使用 DSN，则单击“使用 ODBC 数据源名”，并从框中选择一个 DSN，或单击“新建”创建一个。如果想创建一个连接字符串，请选择“使用 ConnectionString”，单击“生成”，然后使用“数据链接属性”对话框创建一个连接字符串。在创建连接字符串后，单击“确定”。ConnectionString 属性将使用一个类似于下面这一行的字符串来填充：

```
driver={SQL Server};server=bigsmile;uid=sa;pwd=pwd;database=pubs
```

5. 在“属性”窗口中，将“记录源”属性设置为一个 SQL 语句。例如：

```
SELECT * FROM Titles WHERE AuthorID = 72
```

在访问一个表时，应始终包括一个 WHERE 子句。如果这样做失败，则会锁定整个表，这样对其他用户将是一个严重的障碍。

6. 在窗体上再放置一个“文本框”控件，用来显示数据库信息。
7. 在其“属性”窗口中，将 Text1 的“数据源”属性设为 ADO Data 控件的名称 (ADODC1)。这样就将这个文本框和 ADO Data 控件绑定在一起。
8. 在其“属性”窗口中，单击“数据字段”将下拉得到一个可用的字段列表。单击所要显示的字段名称。
9. 对希望访问的其它每个字段重复第 6、7、8 步。
10. 按 F5 键运行该应用程序。用户可以在 ADO Data 控件中使用四个箭头按钮，从而允许用户到达数据的开始、记录的末尾或在数据内从一个记录移动到另一个记录。

在程序中设置 **ConnectionString**、**Source**、**DataSource** 以及 **DataField**

下面的代码演示了如何在程序中设置这四个属性。注意设置 DataSource 属性要使用 Set 语句。

```
Private Sub Form_Load()
```

```
    With ADODC1
```

```
        .ConnectionString = "driver={SQL Server};" & _
```

```
        "server=bigsmile;uid=sa;pwd=pwd;database=pubs"
```

```

        . RecordSource = "Select * From Titles Where AuthorID = 7"
    End With
    Set Text1.DataSource = ADODC1
    Text1.DataField = "Title"
End Sub

```

ADO Data 控件的事件

ADO Data 控件提供了若干个可以编程的事件。下表说明了这些事件及其何时产生，不过这个表不是一个关于这些事件何时发生的所有条件的完整列表。更完整的信息，请参阅各个事件的参考主题。

事件	产生条件
WillMove	当 执 行 Recordset.Open 、 Recordset.MoveNext 、 Recordset.Move 、 Recordset.MoveLast 、 Recordset.MoveFirst 、 Recordset.MovePrevious 、 Recordset.Bookmark 、 Recordset.AddNew 、 Recordset.Delete、Recordset .Requery、Recordset.Resync 方法时
MoveComplete	在 WillMove 事件之后
WillChangeField	在 Value 属性更改之前
FieldChangeComplete	在 WillChangeField 事件之后

续表

WillChangeRecord	当 执 行 Recordset.Update 、 Recordset.Delete 、 Recordset.CancelUpdate 、 Recordset.UpdateBatch、 Recordset.CancelBatch 方法时
RecordChangeComplete	在 WillChangeRecord 事件之后
WillChangeRecordset	在 执 行 Recordset.Requery 、 Recordset.Resync 、 Recordset.Close 、 Recordset.Open、 Recordset.Filter 方法时
RecordsetChangeComplete	在 WillChangeRecordset 事件之后
InfoMessage	当数据提供者返回一个结果时

详细信息 如果试图循序渐进地使用 ADO Data 控件，请参阅“使用 DataGrid 和 ADO 数据控件创建一个简单的数据库应用程序”，“创建一个简单的 DataCombo 应用程序”以及“创建一个连接 DataList 控件的 DataGrid”。

设置 ADO Data 控件的与数据库相关的属性

当创建连接时，您可以使用下列三种源之一：一个连接字符串，一个 OLE DB 文件(.MDL)，或一个 ODBC 数据源名称(DSN)。当您使用 DSN 时，则无

须更改控件的任何其它属性。

不过，如果对数据库技术比较了解，可以更改在 ADO Data 控件中出现的其它的一些属性。下面的列表说明了该控件的与数据库相关的一些属性。这个列表同时也建议了设置这些属性的逻辑顺序。

注意 数据库技术是比较复杂的，下列的建议并不意味着一定要视为规则。

1. **ConnectionString**—**ConnectionString** 属性是一个字符串，可以包含进行一个连接所需的所有设置值。在该字符串中所传递的参数是与驱动程序相关的。例如，ODBC 驱动程序允许该字符串包含驱动程序、提供者、缺省的数据库、服务器、用户名称以及密码等。
2. **UserName**—用户的名称，当数据库受密码保护时，需要指定该属性。和 **Provider** 属性类似，这个属性可以在 **ConnectionString** 中指定。如果同时提供了一个 **ConnectionString** 属性以及一个 **UserName** 属性，则 **ConnectionString** 中的值将覆盖 **UserName** 属性的值。
3. **Password**—在访问一个受保护的数据库时也是必需的。和 **Provider** 属性、**UserName** 属性类似，如果在 **ConnectionString** 属性中指定了密码，则将覆盖在这个属性中指定的值。
4. **RecordSource**—这个属性通常包含一条语句，用于决定从数据库检索什么信息。
5. **CommandType**—**CommandType** 属性告诉数据提供者 **Source** 属性是一条 SQL 语句、一个表的名称、一个存储过程还是一个未知的类型。
6. **CursorLocation**—这个属性指定光标的位置，是位于客户还是位于服务器

上。这一决策将影响您对下面几个属性的设置。

7. **CursorType**—**CursorType** 属性决定记录集是静态类型、动态类型、还是键集光标类型。
8. **LockType**—**LockType** 属性决定当其他人试图更改您正在编辑的数据时，如何锁定该数据。如何设置这个 **LockType** 属性是一个复杂的决策，取决于多个因素。
9. **Mode**—**Mode** 属性决定想用记录集进行什么操作。例如，如果只是想要创建一个报告，可以将该属性设为只读来获得性能的改善。
10. **MaxRecords**—这个属性决定光标的大小。如何决定这个属性的值取决于所检索的记录的大小，以及计算机的可用资源（内存）的多少。一个大的记录（包括很多列以及长字符串）比小记录要花费更多的资源。因此，**MaxRecords** 属性就不能太大。
11. **ConnectionTimeout**—设置等待建立一个连接的时间，以秒为单位。如果连接超时，则返回一个错误。
12. **CacheSize**—**CacheSize** 属性指定从光标中可以检索多少条记录。如果将 **CursorLocation** 设为客户端，则这个属性只能设为一个较小的数目（可能为 1），不会有任何不利的影响。如果光标的位置位于服务器端，则可以对这个数进行调整，将其设为您希望一次可以查看的行数。例如，如果使用 **DataGrid** 控件来查看 30 行，则可以将 **CacheSize** 设为 60，这样不必检索更多的数据就可以进行滚动。
13. **BOFAction**、**EOFAction**—这两个属性决定当该控件位于光标的开始和末尾时的行为。提供的选择包括停留在开始或末尾、移动到第一个或最

后一个记录、或添加一个新记录（只能在末尾）。

绑定到 ADO Data 控件的控件

任何具有 DataSource 属性的控件都可以绑定到一个 ADO Data 控件。下面的内在控件都可以绑定到 ADO Data 控件：

- 复选框 (CheckBox)
- 组合框 (ComboBox)
- 图像 (Image)
- 标签 (Label)
- 列表框 (ListBox)
- 图片框 (PictureBox)
- 文本框 (TextBox)

Visual Basic 的所有版本中也提供下述数据绑定的 ActiveX 控件：

- DataList
- DataCombo
- DataGrid
- Microsoft Hierarchical FlexGrid
- RichTextBox
- Microsoft Chart
- DateTimePicker
- ImageCombo

- **MonthView**

最后，用户可以使用 **DataBinding** 对象创建自己的数据绑定的 **ActiveX** 控件。

详细信息 关于自己创建数据绑定控件的详细内容，请参阅“创建数据绑定用户控件”。

使用 **CheckBox** 控件

选定 **CheckBox** 控件时，这个控件将显示选定标记。通常用此控件提供 **Yes/No** 或 **True/False** 选项。可用分组的 **CheckBox** 控件显示多个选项，用户可从中选择一个或多个选项。



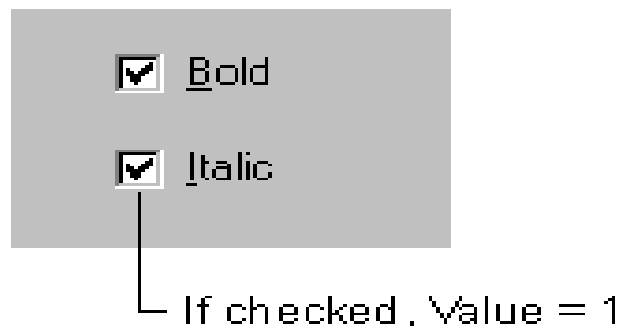
图 7.5 **CheckBox** 控件

CheckBox 控件与 **OptionButton** 控件的相同之处在于，每个都是用来指示用户所作的选择。不同之处在于，对于一组 **OptionButton**，一次只能选定其中的一个，而对 **CheckBox** 控件，则可选定任意数目的复选框。

详细信息 关于 **CheckBox** 控件的简单示例，请参阅第三章“窗体，控件和菜单”中的“用复选框选择单个选项”。

Value 属性

CheckBox 控件的 Value 属性指示复选框处于选定、未选定或禁止状态（暗淡的）中的哪一种。选定时，value 设置值为 1。例如：



下表列出用于设置 Value 属性的数值和相应的 Visual Basic 常数。

设置值	值	常数
Unchecked	0	vbUnchecked
Checked	1	vbChecked
Unavailable	2	vbGrayed

用户单击 CheckBox 控件指定选定或未选定状态，然后可检测控件状态并根据此信息编写应用程序以执行某些操作。

缺省时，CheckBox 控件设置为 vbUnchecked。若要预先在一列复选框中选定若干复选框，则应在 Form_Load 或 Form_Initialize 过程中将 Value 属性设置为 vbChecked。

可将 Value 属性设置为 vbGrayed 以禁用复选框。例如，有时可能希望在

满足某条件之前禁用复选框。

Click 事件

无论何时单击 `CheckBox` 控件都将触发 `Click` 事件，然后编写应用程序，根据复选框的状态执行某些操作。在下例中，每次单击 `CheckBox` 控件时都将改变其 `Caption` 属性以指示选定或未选定状态。

```
Private Sub Check1_Click()  
    If Check1.Value = vbChecked Then  
        Check1.Caption = "Checked"  
    ElseIf Check1.Value = vbUnchecked Then  
        Check1.Caption = "Unchecked"  
    End If
```

```
End Sub
```

注意 如果试图双击 `CheckBox` 控件，则将双击当作两次单击，而且分别处理每次单击；这就是说，`CheckBox` 控件不支持双击事件。

响应鼠标和键盘

在键盘上使用 `TAB` 键并按 `SPACEBAR` 键，由此将焦点转移到 `CheckBox` 控件上，这时也会触发 `CheckBox` 控件的 `Click` 事件。

可以在 `Caption` 属性的一个字母之前添加连字符，创建一个键盘快捷方式来切换 `CheckBox` 控件的选择。例如：



本例中，按 ALT+C 组合键将使控件的状态在选定和未选定之间切换。

增强 CheckBox 控件的视觉效果

CheckBox 控件象 CommandButton 和 OptionButton 控件一样，可通过更改 Style 属性的设置值后使用 Picture、DownPicture 和 DisabledPicture 属性增强其视觉效果。例如，有时可能希望在复选框中添加图标或位图，或者在单击或禁止控件时显示不同的图像。

使用 ComboBox 控件

组合框控件将文本框和列表框的功能结合在一起。有了这个控件，用户可以通过在组合框中输入文本来选定项目，也可从列表选定项目。



图 7.6 ComboBox 控件

组合框向用户提供了供选择的列表。如果项目数超过了组合框能够显示的项目数，控件上将自动出现滚动条。用户即可上下或左右滚动列表。

何时用组合框代替列表框

通常，组合框适用于建议性的选项列表，而当希望将输入限制在列表之内时，应使用列表框。组合框包含编辑区域，因此可将不在列表中的选项输入列区域中。

此外，组合框节省了窗体的空间。只有单击组合框的向下箭头时（样式 1 的组合框除外，它总是处于下拉状态）才显示全部列表，所以无法容纳列表框的地方可以很容易地容纳组合框。

详细信息 关于这些控件的简单示例，请参阅第三章“窗体、控件和菜单”中的“使用 ListBox 和 ComboBox”。有关列表框控件的详细信息，还请参阅本章后面的“使用 ListBox 控件”。

数据绑定特性

Visual Basic 中的标准版和数据绑定版的 ComboBox 控件。虽然通过这两个版本都可以显示、编辑和更新大多数标准类型数据库中的信息，但是 DBCombo 提供了更高级的数据访问特性。DBCombo 控件还支持一组与标准组合框控件不同的属性和方法。

详细信息 关于详细信息，请参阅第七章“使用 Visual Basic 标准控件”中“使用 DataList 和 DataCombo”控件。

组合框的样式

此处有三种组合框样式。每种样式都可在设计时或运行时来设置，而且每种样式都使用数值或相应的 Visual Basic 常数来设置组合框的样式。

样式	值	常数
下拉式组合框	0	vbComboDropDown
简单组合框	1	vbComboSimple
下拉式列表框	2	VbComboDropDownList



图 7.7 组合框样式

下拉式组合框

在缺省设置 (Style = 0) 下，组合框为下拉式。用户可（象在文本框中一样）直接输入文本，也可单击组合框右侧的附带箭头打开选项列表。选定某个选项后，将此选项插入到组合框顶端的文本部分中。当控件获得焦点时，也可按 ALT+ DOWN ARROW 键打开列表。

简单组合框

将组合框 `Style` 属性设置为 1 将指定一个简单的组合框，任何时候都在其内显示列表。为显示列表中所有项，必须将列表框绘制得足够大。当选项数超过可显示的限度时将自动插入一个垂直滚动条。用户可直接输入文本，也可从列表中选择。象下拉式组合框一样，简单组合框也允许用户输入那些不在列表中的选项。

下拉式列表框

下拉式列表框 (`Style = 2`) 与正规列表框相似——它显示项目的列表，用户必须从中选择。但下拉式列表框与列表框的不同之处在于，除非单击框右侧的箭头，否则不显示列表。这种列表框与下拉式组合框的主要差别在于，用户不能在列表框中输入选项，而只能在列表中选择。当窗体上的空间较少时，可使用这种类型的列表框。

添加项目

为在组合框中添加项目，应使用 `AddItem` 方法，其语法如下：

box.AddItem item[, index]

参 数	描 述
<i>box</i>	列表框或组合框名称
<i>item</i>	在列表中添加的字符串表达式。若 <i>item</i> 为文字常数，则用引号括起来
<i>index</i>	指定新项目在列表中的插入位置。 <i>Index</i> 为 0 表示第一个位置。若省略 <i>index</i> ，则项目插入在列表末尾（或正确的排序顺序处）

通常在 **Form_Load** 事件过程中添加列表项目，但也可在任何时候使用 **AddItem** 方法。这样一来就能够动态地（响应用户的操作）在列表中添加项目。

以下代码将 "Chardonnay," "Fum'e Blanc," "Gewürztraminer," 和 "Zinfandel" 放置到名为 **Combo1**，**Style** 属性为 0 (**vbComboDropDown**) 的组合框中：

```
Private Sub Form_Load ()  
    Combo1.AddItem "Chardonnay"  
    Combo1.AddItem "Fum'e Blanc"  
    Combo1.AddItem "Gewürztraminer"  
    Combo1.AddItem "Zinfandel"  
End Sub
```

运行时，只要加载窗体，而且用户单击向下箭头，则将显示如图 7.8 所示的列表。



图 7.8 “葡萄酒清单”组合框

设计时添加项目

在设计时，也可设置组合框控件“属性”窗口的 `List` 属性，从而在列表中添加项目。选定 `List` 属性选项并单击向下箭头后就可输入列表项目，然后按 `CTRL+ENTER` 组合键换到新的一行。

只能将项目添加到列表的末尾。所以，如果要将列表按字母顺序排序，则应将 `Sorted` 属性设置为 `True`。关于详细的信息，请参阅后面“排序列表”。

在指定位置添加项目

为了在列表指定位置添加项目，应在新项目后指定索引值。例如，下行代码将 `"Pinot Noir"` 插入到第一个位置并把其它项目的位置向下调整：

```
Combo1.AddItem "Pinot Noir", 0
```

注意，指定列表中的第一个位置的是 `0` 而不是 `1`（见图 7.9）。

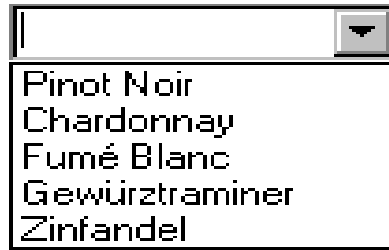


图 7.9 向列表添加项目

排序列表

将 `Sorted` 属性设置为 `True` 并省略索引，则可在列表中指定按字母顺序添加的项目。排序时不区分大小写；所以 `"chardonnay"` 和 `"Chardonnay"` 被看作一个词。

将 `Sorted` 属性设置为 `True` 之后，使用带有 `index` 参数的 `AddItem` 方法将导致不可预料的非排序结果。

删除项目

可在组合框中用 `RemoveItem` 方法删除项目。`RemoveItem` 有一个参数 *index*，它指定要删除的项目：

`box.RemoveItem index`

box 及 *index* 参数和 `AddItem` 中的参数相同。

例如，为了删除列表中的第一个项目，应添加下面一行代码：

```
Combo1.RemoveItem 0
```

为了在组合框中删除所有列表项目，应使用 `Clear` 方法：

```
Combo1.Clear
```

用 `Text` 属性获取列表内容

获取当前选定项目值的最简单的常用方法就是使用 `Text` 属性。在运行时无论向控件的文本框部分输入了什么文本，`Text` 属性都与这个文本相对应。它可以是选定的列表选项，或者是用户在文本框中输入的字符串。例如，如果用户选定列表框中的 `"Chardonnay"`，则通过下列代码显示有关 `Chardonnay` 的信息：

```
Private Sub Combo1_Click ()  
    If Combo1.Text = "Chardonnay" Then  
        Text1.Text = "Chardonnay is a medium-bodied _  
        white wine."  
    End If  
End Sub
```

`Text` 属性包含 `Combo1` 列表框中当前选定的项目。代码查看是否选择了 `"Chardonnay"`，若是如此，则在文本框中显示信息。

用 List 属性访问列表选项

有了 List 属性就可访问列表中所有项目。该属性包含一个数组，而且列表中的每个项目都是数组的元素。每一项都表示为字符串的形式。为了引用列表中的项目，应使用如下语法：

box.List(*index*)

box 参数引用组合框，而 *index* 是项目的位置。顶端项目的索引为 0，下一个项目的索引为 1，依次类推。例如，在文本框中，以下语句显示列表中的第三个项目 (*index* = 2)：

```
Text1.Text = Combo1.List(2)
```

用 ListIndex 属性判断位置

欲知组合框列表选定项目位置，请使用 ListIndex 属性。该属性设置或返回控件中当前选定项目的索引值，而且只在运行时有效。对组合框的 ListIndex 属性进行设置也会触发控件的 Click 事件。

若选定第一个（顶端）项目，则属性值为 0，选定的下一个项目属性值为 1，依次类推。如果未选定项目，或者用户在组合框中输入选项（样式 0 或 1）而不在列表中选择现有项目，则 ListIndex 为 -1。

注意 NewIndex 属性可用来跟踪列表中最后添加的项目的索引。向排序列表插入项目时，这一点很有用。

用 ListCount 属性返回项目数

为了返回组合框中的项目数，应使用 ListCount 属性。例如，下列语句用 ListCount 属性判断组合框中的项目数：

```
Text1.Text = "You have " & Combo1.ListCount & " _  
entries listed"
```

使用 CommandButton 控件

命令按钮控件被用来启动、中断或结束一个进程。单击它时将调用已写入 Click 事件过程中的命令。



图 7.10 CommandButton 控件

大多数 Visual Basic 应用程序中都有命令按钮，用户可以单击按钮执行操作。单击时，按钮不仅能执行相应的操作，而且看起来就象是被按下和松开一样，因此有时称其为下压按钮。

详细信息 关于 CommandButton 控件的简单示例，请参阅第三章，“窗体、控件和菜单”中的“单击按钮执行操作”。

向窗体添加命令按钮

在应用程序中很可能要使用一个或多个命令按钮。就象在其它控件绘制按钮那样,在窗体上添加命令按钮。可用鼠标调整命令按钮的大小,也可通过设置 Height 和 Width 属性进行调整。

设置标题

可用 Caption 属性改变命令按钮上显示的文本。设计时,可在控件的“属性”窗口中设置此属性。在设计时设置 Caption 属性后将动态更新按钮文本。

Caption 属性最多包含 255 个字符。若标题超过了命令按钮的宽度,则会折到下一行。但是,如果控件无法容纳其全部长度,则标题会被剪切。

可以通过设置 Font 属性改变在命令按钮上显示的字体。

创建键盘快捷方式

可通过 Caption 属性创建命令按钮的访问键快捷方式,为此,只需在作为访问键的字母前添加一个连字符 (&)。例如,要为标题 "Print" 创建访问键,应在字母 "P" 前添加连字符,于是得到 "&Print"。运行时,字母 "P" 将带下划线,同时按 ALT+P 键就可选定命令按钮。

注意 如果不创建访问键,而又要使标题中包含连字符但不创建访问键,应添加两个连字符 (&&)。这样一来,在标题中就只显示一个连字符

而不显示下划线。

指定 **Default** 和 **Cancel** 属性

在每个窗体上部可选择命令按钮作为缺省的命令按钮，也就是说，不管窗体上的哪个控件有焦点，只要用户按 **ENTER** 键，就已单击此缺省按钮。为了指定一个缺省命令按钮，应将其 **Default** 属性设置为 **True**。也可指定缺省的取消按钮。在把命令按钮的 **Cancel** 属性设置为 **True** 后，不管窗体的哪个控件有焦点，按 **ESC** 键，就已单击了此缺省按钮。

选定命令按钮

运行时，可用鼠标或键盘通过下述方法选定命令按钮：

- 用鼠标单击按钮。
- 按 **TAB** 键，将焦点转移到按钮上，然后按 **SPACEBAR** 或 **ENTER** 键选定按钮。
- 按命令按钮的访问键（**ALT**+ 带下划线的字母）。
- 若命令按钮是窗体的缺省命令按钮，则可按 **ENTER** 键选定按钮，即使已把焦点转移到其它控件上，情况也是如此。
- 若命令按钮是窗体的缺省取消按钮，则可按 **ESC** 键选定按钮，即使已把焦点转移到其它控件上，情况也是如此。

Value 属性

无论何时选定命令按钮都会将其 Value 属性设置为 True 并触发 Click 事件。False（缺省）指示未选择按钮。可在代码中用 Value 属性触发命令按钮的 Click 事件。例如：

```
cmdClose.Value = True
```

Click 事件

单击命令按钮时将触发按钮的 Click 事件并调用已写入 Click 事件过程中的代码。

单击命令按钮后也将生成 MouseDown 和 MouseUp 事件。如果要在这些相关事件中附加事件过程，则应确保操作不发生冲突。控件不同，这三个事件过程发生的顺序也不同。CommandButton 控件中事件发生的顺序为：MouseDown、Click、MouseUp。

注意 如果用户试图双击命令按钮控件，则其中每次单击都将被分别处理；即命令按钮控件不支持双击事件。

详细信息 关于 MouseDown 和 MouseUp 事件的详细信息，请参阅第十一章，“响应鼠标和键盘事件”。

增强命令按钮的视觉效果

命令按钮象复选框和选项按钮一样，可通过更改 Style 属性设置值后用

Picture、DownPicture 和 DisabledPicture 属性增强视觉效果。会有这样的情况发生，比如要向命令按钮添加图标或位图，或者在单击、禁止控件时显示不同的图像。

使用 CommonDialog 控件

CommonDialog 控件提供诸如打开和保存文件、设置打印选项、选择颜色和字体等操作的一组标准对话框。运行 Windows 帮助引擎时，控件还能够显示帮助。



图 7.11 CommonDialog 控件

CommonDialog 控件在 Visual Basic 和 Microsoft Windows 动态链接库 Commdlg.dll 例程之间提供了接口。为了用该控件创建对话框，必须要求 Commdlg.dll 在 Microsoft Windows\System 目录下。

为了在应用程序中使用 CommonDialog 控件，应将其添加到窗体上并设置属性。控件显示的对话框由控件的方法决定。运行时，调用相应方法后将显示对话框或执行帮助引擎；设计时在窗体上将 CommonDialog 控件显示成一个图标。此图标的大小不能改变。

CommonDialog 控件可以显示如下常用对话框：

- “打开”
- “另存为”
- “颜色”
- “字体”
- “打印”

要使用 `CommonDialog` 控件

1. 若未添加 `CommonDialog` 控件，则应从“工程”菜单中选定“部件”，将控件添加到工具箱中。在标记对话的“控件”中找到并选定控件，然后单击“确定”按钮。
2. 单击工具箱中的“`CommonDialog`”控件并在窗体上绘制该控件。
在窗体上绘制 `CommonDialog` 控件时，控件将自动调整大小。象 `Timer` 控件一样，`CommonDialog` 控件在运行时不可见。
3. 运行时，请适当使用下表所列方法显示需要的对话。

方法	显示的对话
<code>ShowOpen</code>	打开
<code>ShowSave</code>	另存为
<code>ShowColor</code>	颜色
<code>ShowFont</code>	字体
<code>ShowPrinter</code>	打印
<code>ShowHelp</code>	调用 Windows “帮助”

显示“打开”和“另存为”对话框

有了“打开”对话框就可指定驱动器、目录、文件扩展名和文件名。

“另存为”对话框在外观上与“打开”对话框相同，只是对话框的标题和文件名是暗淡的。运行时选定文件并关闭对话框后，可用 `FileName` 属性获取选定的文件名。

要显示“打开”对话框

1. 指定在“文件类型”列表框中显示的文件过滤器列表。

可用下列格式设置 `Filter` 属性：

description1 | filter1 | description2 | filter2...

Description 是列表框中显示的字符串——例如，“Text Files (*.txt)”。`Filter` 是实际的文件过滤器——例如，“*.txt”。每个 *description | filter* 设置间必须用管道符号分隔 (|)。

2. 用 `ShowOpen` 方法显示对话框。

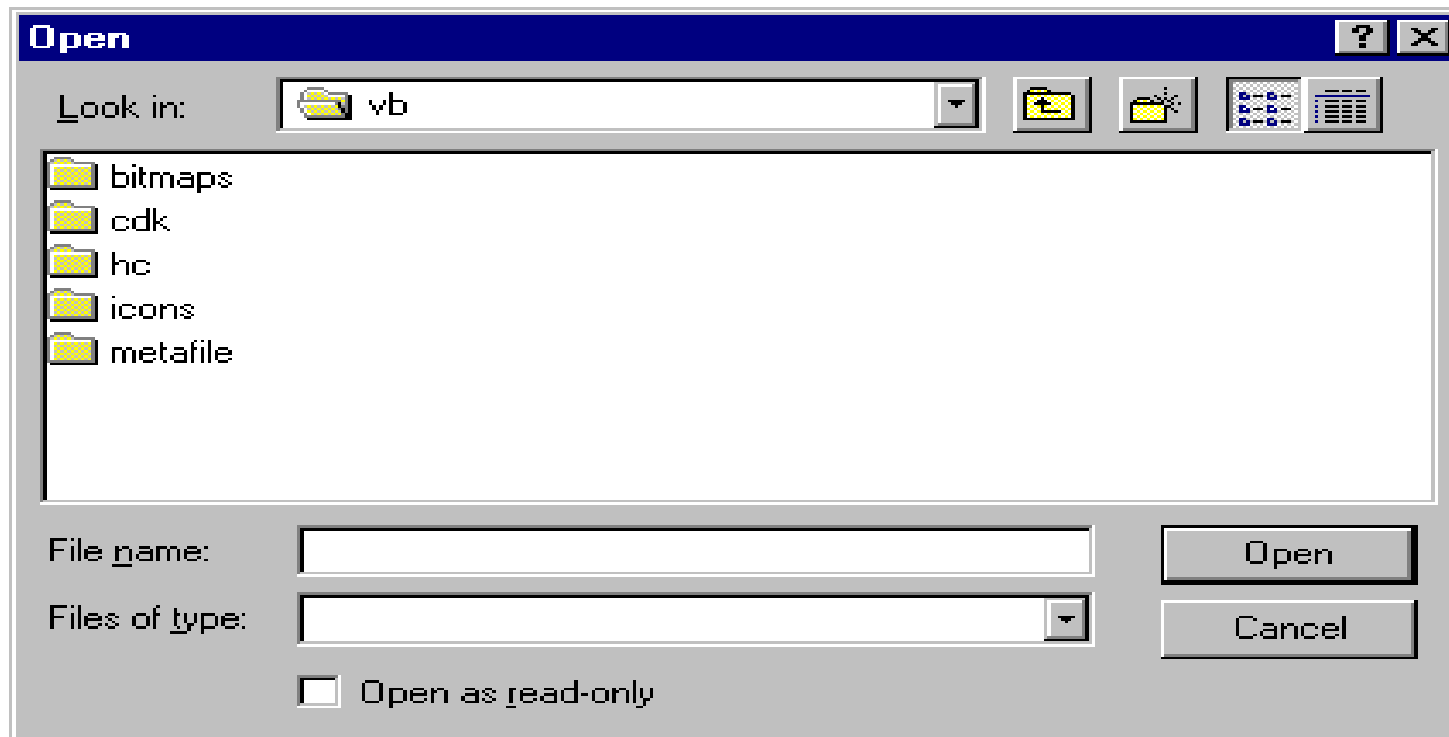


图 7.12 “打开”对话框

选定文件后可用 `FileName` 属性获取选定文件的名称。
对所有公共对话框，当 `CancelError` 属性为 `True`，而且用户单击了对话框的“取消”按钮时将生成一个错误。在显示对话框时捕获错误，以此检测是否按了“取消”按钮。

下列代码显示“打开”对话框并以选定的文件名为打开文件过程的参数：

```
Private Sub mnuFileOpen_Click ()  
    'CancelError 为 True。  
    On Error GoTo ErrHandler
```

'设置过滤器。

```
CommonDialog1.Filter = "All Files (*.*)|*.*|Text _  
Files (*.txt)|*.txt|Batch Files (*.bat)|*.bat"
```

'指定缺省过滤器。

```
CommonDialog1.FilterIndex = 2
```

'显示“打开”对话框。

```
CommonDialog1.ShowOpen
```

'调用打开文件的过程。

```
OpenFile (CommonDialog1.FileName)
```

```
Exit Sub
```

ErrHandler:

'用户按“取消”按钮。

```
Exit Sub
```

```
End Sub
```

使用“颜色”对话框

可用“颜色”对话框在调色板中选择颜色，或者创建并选定自定义颜色。运行时，选定颜色并关闭对话框后可用 `Color` 属性获取选定的颜色。



图 7.13 “颜色”对话框

要显示“颜色”对话框

1. 将 `CommonDialog` 控件的 `Flags` 属性设置成 Visual Basic 常数 `cdlCCRGBInit`。
 2. 用 `ShowColor` 方法显示对话框。
- 可用 `Color` 属性获取选定颜色的 RGB 值。单击“`Command1`”命令按钮时，下列代码将显示“颜色”对话框：

```
Private Sub Command1_Click ()
    '将 Cancel 设置成 True。
    CommonDialog1.CancelError = True
    On Error GoTo ErrHandler
    '设置 Flags 属性。
    CommonDialog1.Flags = cdlCCRGBInit
    '显示 “颜色” 对话框。
    CommonDialog1.ShowColor
    '将窗体的背景颜色设置成选定的'颜色。
    Form1.BackColor = CommonDialog1.Color
    Exit Sub

ErrHandler:
    '用户按了 “取消” 按钮。
    Exit Sub
End Sub
```

使用 “字体” 对话框

“字体”对话框根据大小、颜色、样式选择字体。用户一旦在“字体”对话框中选定字体后，下列属性就会包含有关用户选项的信息。

属 性	决 定
Color	选定的颜色。为使用此属性，必须先将 <code>Flags</code> 属性设置为 <code>cdlCFEffects</code>
FontBold	是否选定“粗体”
FontItalic	是否选定“斜体”
FontStrikethru	是否选定删除线
FontUnderline	是否选定下划线
FontName	选定的字体名称
FontSize	选定的字体大小

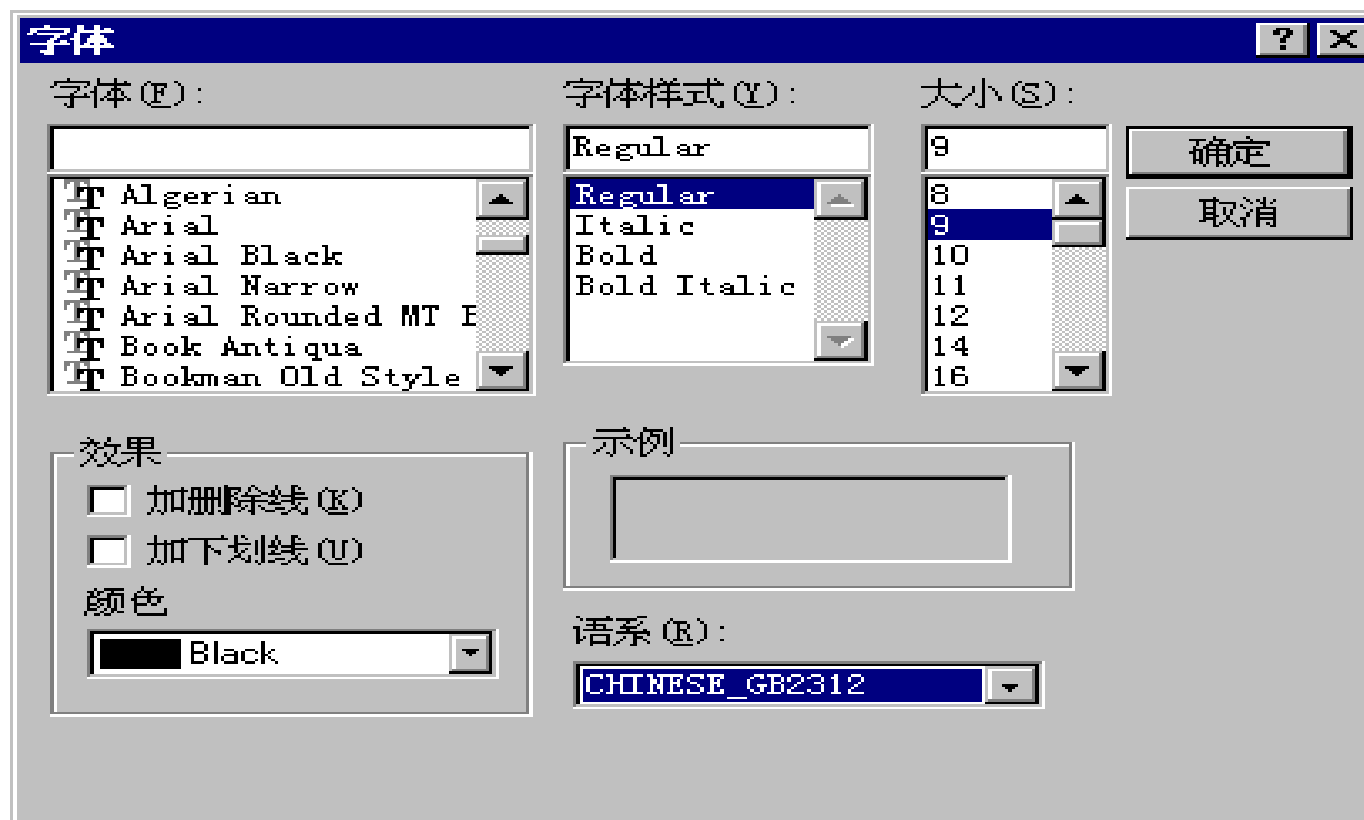


图 7.14 “字体”对话框

要显示“字体”对话框

1. 将 Flags 属性设置为下述 Visual Basic 常数之一：

- `cdlCFScreenFonts`（屏幕字体）
- `cdlCFPrinterFonts`（打印机字体）
- `cdlCFBoth`（既可以是屏幕字体又可以是打印机字体）

警告 在显示“字体”对话框之前必须将 `Flags` 属性设置为这些数值之一，否则将发生字体不存在错误。

2. 用 `ShowFont` 方法显示对话框。

下列代码根据用户在“字体”对话框中的选择来设置文本框的字体属性：

```
Private Sub Command1_Click ()  
    '将 Cancel 设置成 True。  
    CommonDialog1.CancelError = True  
    On Error GoTo ErrHandler  
    '设置 Flags 属性。  
    CommonDialog1.Flags = cdlCFBoth Or cdlCFEffects  
    '显示“字体”对话框。  
    CommonDialog1.ShowFont  
    '根据用户的选择来设置文本属性。  
    Text1.Font.Name = CommonDialog1.FontName  
    Text1.Font.Size = CommonDialog1.FontSize  
    Text1.Font.Bold = CommonDialog1.FontBold  
    Text1.Font.Italic = CommonDialog1.FontItalic  
    Text1.Font.Underline = CommonDialog1.FontUnderline  
    Text1.Font.Strikethru = CommonDialog1.FontStrikethru  
    Text1.ForeColor = CommonDialog1.Color  
    Exit Sub  
ErrHandler:
```

用户按了“取消”按钮。

Exit Sub

End Sub

使用“打印”对话框

“打印”对话框允许用户指定打印输出的方法。用户可指定打印页数范围、打印质量、复制数目等等。此对话框还显示有当前安装的打印机信息，并允许用户进行配置或重新安装新的缺省打印机。

注意 此对话框并不真正地将数据送到打印机上。它允许用户指定如何打印数据。必须编写代码实现用选定格式打印数据。

详细信息 关于打印数据的信息，请参阅第十二章“使用文本和图形”。运行时，当用户在“打印”对话框作出选择后，下述属性将包含用户选项的信息。

属性	决定
Copies	要打印的份数
FromPage	打印的起始页
ToPage	打印的结束页
hDC	选定打印机的设备上下文
Orientation	页面定向（画像或风景）

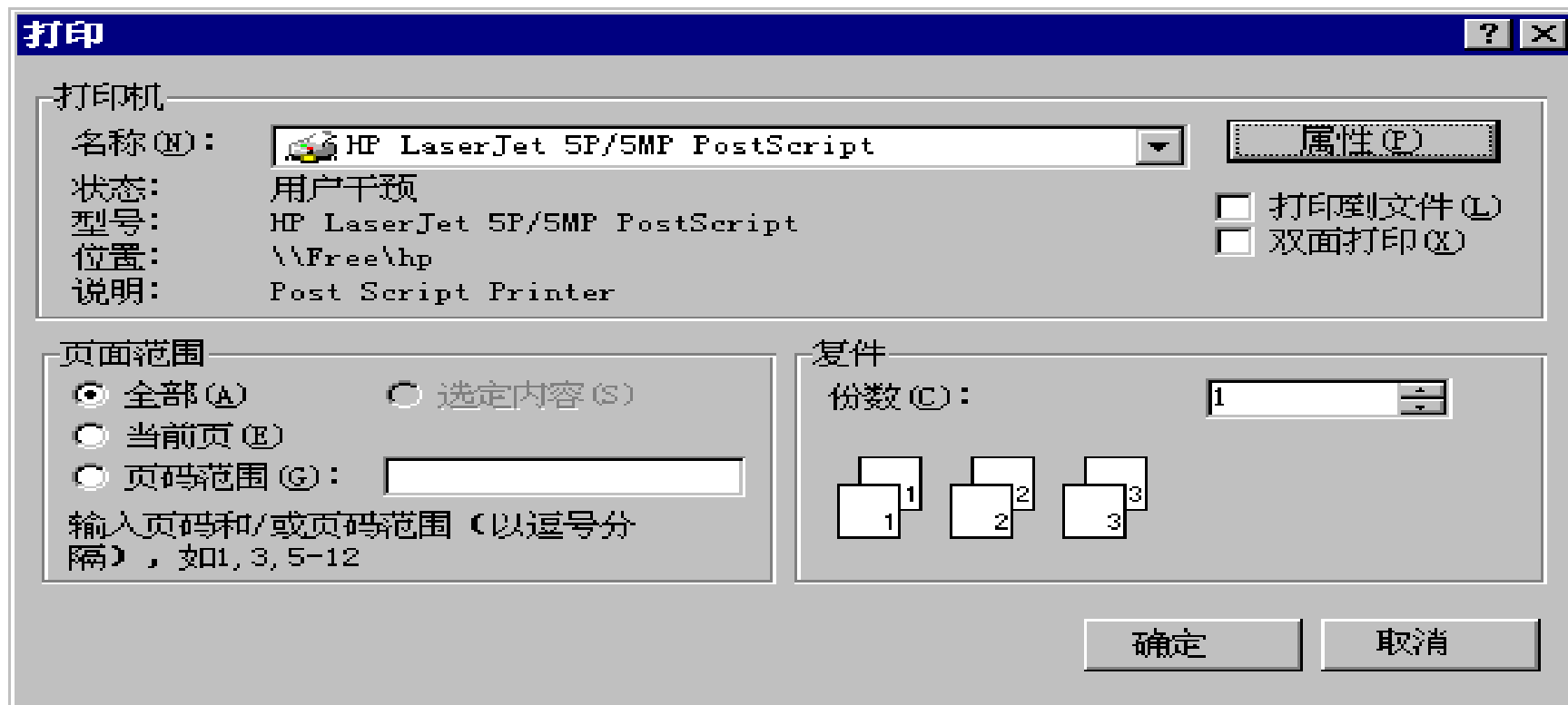


图 7.15 “打印”对话框

要显示“打印”对话框

1. 通过设置相应的“打印”对话框属性，为对话框设置所需缺省设置值。
例如，为在显示对话框时在“份数”框中显示 2，应将 Copies 属性设置为 2：

```
CommonDialog1.Copies = 2
```

2. 用 ShowPrinter 方法显示“打印”对话框。

用户单击“Command1”命令按钮时，下列代码将显示“打印”对话框：

```

Private Sub Command1_Click ()
    Dim BeginPage, EndPage, NumCopies, Orientation.    i
    '将 Cancel 设置成 True。
    CommonDialog1.CancelError = True
    On Error GoTo ErrHandler
    '显示 “打印” 对话框。
    CommonDialog1.ShowPrinter
    '从对话框中获取用户选定数值。
    BeginPage = CommonDialog1.FromPage
    EndPage      = CommonDialog1.ToPage
    NumCopies     = CommonDialog1.Copies
    Orientation=CommonDialog1.Orientation
    For i = 1 to NumCopies
        '在此放置代码，将数据发送到打印机。
    Next
    Exit Sub
ErrHandler:
    '用户按下了 “取消” 按钮。
    Exit Sub
End Sub

```

注意 若将 PrinterDefault 属性设置为 True，则可在 Visual Basic Printer 对象上打印。另外，当 PrinterDefault 属性为 True 时，所有在 “打印”

对话框“设置”部分中作出的变更都将改变用户“打印机”设置中的打印机设置值。

使用 ShowHelp 方法显示帮助文件

可用 CommonDialog 控件的 ShowHelp 方法显示帮助文件。

要使用 ShowHelp 方法显示帮助文件

1. 设置 HelpCommand 和 HelpFile 属性。
2. 用 ShowHelp 方法显示指定的帮助文件。

在单击“Command1”命令按钮时，下列代码将显示指定的帮助文件：

```
Private Sub Command1_Click()  
    '将 Cancel 设置为 True。  
    CommonDialog1.CancelError = True  
    On Error GoTo ErrHandler  
    '设置 HelpCommand 属性  
    CommonDialog1.HelpCommand = cdlHelpForceFile  
    '指定帮助文件。  
    CommonDialog1.HelpFile = "c:\Windows\Cardfile.hlp"  
    '显示 Windows 帮助引擎。  
    CommonDialog1.ShowHelp  
Exit Sub
```

ErrorHandler:

 '用户按下了“取消”按钮。

Exit Sub

End Sub

详细信息 有关显示通用对话框控件的帮助文件的详细信息，请参阅《Visual Basic 6.0 语言参考手册》“HelpCommand 属性”、“HelpFile 属性”和“ShowHelp 方法”。

使用 Data 控件

内在的 Data 控件通过使用 Microsoft 的 Jet 数据库引擎来实现数据访问——与 Microsoft Access 所用的数据库引擎相同。这一技术使用户可以无缝地访问很多标准的数据库格式，而且使用户无需编写任何代码就可以创建数据识别应用程序。这种内在的 Data 控件最适合较小的（桌面）数据库，诸如 Access 和 ISAM 数据库。

可以使用这种内在的 Data 控件创建应用程序来显示、编辑和更新来自多种已有的数据库的信息。这些数据库包括 Microsoft Access、Btrieve、dBASE、Microsoft FoxPro 和 Paradox。也可以使用这种控件如同访问真正的数据库一样来访问 Microsoft Excel、Lotus 1-2-3 以及标准的 ASCII 文本文件。此外，Data 控件也可以访问和操作远程的开放式数据库连接 (ODBC) 数据库，诸如 Microsoft SQL Server 以及 Oracle。

注意 Data 控件和 Remote Data 控件两者都包含在 Visual Basic 中，以提供向后兼容。不过，由于 ActiveX 数据对象 (ADO) 的适应性更广，因此建议使用 ADO Data 控件来创建新的数据库应用程序。详细信息请参阅“使用 ADO Data 控件”。

Data 控件、Remote Data 控件以及 ADO Data 控件在概念上很相似：三者都是将一个数据源连接到一个数据绑定控件的“数据控件”。三者也都共享相同的外观——一组共四个按钮，使用户可以直接到达记录集的开始、记录集的末尾以及在记录集中向前或向后翻卷。

使用 Data 控件创建一个简单的数据库应用程序

要使用 Data 控件创建一个简单的数据库应用程序

1. 在窗体上放置一个 Data 控件。Data 控件是一个内在的控件，因而总是可用的。
2. 单击并选定这个 Data 控件，按 F4 键显示“属性”窗口。
3. 在“属性”窗口中，将“连接”属性设置为想要使用的数据库类型。
4. 在“属性”窗口中，将“DatabaseName”属性设置为想要连接的数据库的文件或目录名称。
5. 在“属性”窗口中，将“记录源”属性设置为想要访问的数据库表的名称。
6. 在该窗体上放置一个文本框控件。
7. 单击并选定这个 TextBox 控件，并在其“属性”窗口中将“数据源”属性设置为该 Data 控件。

8. 在这个“属性”窗口中，将“数据字段”属性设置为在该数据库中想要查看或修改的字段的名称。
9. 对其它的每一个想要访问的字段，重复第 6、7、8 步。
10. 按 F5 键运行这个应用程序。

设置 Data 控件与数据相关的属性

下述与数据相关的属性可以在设计时设置。这个列表给出了设置这些属性的一种逻辑顺序：

注意 数据库技术是一门复杂的科学，下面的这些建议并不意味着要当作规则来使用。

1. **RecordsetType**—RecordsetType 属性决定记录集是一个表、一个动态集 (dynaset) 还是一个快照。这个选择将影响哪些记录集属性是可用的。例如，快照类型的记录集与 dynaset 记录集相比具有更多的限制。
2. **DefaultType**—DefaultType 属性指定所使用的是 JET 工作空间，还是 ODBC Direct 工作空间。
3. **DefaultCursorType**—DefaultCursorType 属性决定光标的位置。可以使用 ODBC 驱动程序来决定光标的位置，或者指定服务器或 ODBC 光标。只有当使用 ODBC Direct 工作空间时，DefaultCursorType 属性才是有效的。
4. **Exclusive**—决定该数据是用于单用户环境，还是多用户环境。
5. **Options**—这个属性决定记录集的特征。例如，在一个多用户环境中，可以设置 Options 属性来禁止他人所做的更改。

6. **BOFAction**、**EOFAction**—这两个属性决定当这个控件位于光标的开始或末尾时的行为。可能的选择包括停留在开始或末尾、移动到第一个或最后一个记录、或者添加一个新的记录（只有在末尾时）。

使用 DataCombo 和 DataList 控件

DataCombo 和 DataList 控件与标准列表框和组合框控件极为相似，但有一些重要的不同之处，这种不同使这两个控件在数据库应用程序中具有极大的适应性和用武之地。这两个控件都可以被这些控件所绑定的数据库字段自动填充。此外，它们还能有选择地将一个选定的字段传递给第二个数据控件，从而适合用于创建“查找表”应用程序。

可能的用法

- 在一个关系数据库中，使用一个表的数据来提供要输入给第二个（相关的）表值。例如，在一个存货清单数据库中，供应商的名称存储在一个表中，每个供应商都有一个唯一的标识符。另一个显示产品的表则使用这些标识符来表明是哪个供应商供应的该产品。可以使用 DataList 控件来显示供应商的名称，而（不可见地）将供应商的标识符提供给产品表。
- 允许用户通过从一个下拉列表中选择一种标准来缩小搜索范围。例如，一个销售报告的数据库应用程序可以使用一个 DataList 控件让用户选择一个州 (State) 或一个销售区域。一旦作出选择，则该选择项将自动

传递给第二个数据控件，这个控件负责查找选定区域的销售记录。与它们对应的内在控件一样，DataList 和 DataCombo 控件之间的主要不同在于 DataCombo 控件提供了一个可以在其中编辑内容的文本框。

详细信息 有关 DataList 和 DataCombo 控件连接数据库表能力的说明，请参阅“使用 DataCombo 和 DataList 控件连接两个表”。要创建一个简单的使用链接表的数据库应用程序，请参阅“创建一个简单的 DataCombo 应用程序”。

值得注意的控件属性

DataList 和 DataCombo 控件的一些重要属性包括：

属性	描述
BoundText	包含在 BoundColumn 属性中所指定的字段的值
SelectedItem	返回一个对所选项目的行的标签
MatchEntry	在一个由 DataCombo 控件生成的列表中启用扩展搜索模式来定位项目

IntegralHeight 将控件的高度设为恰好显示整数个完整的行

VisibleCount 指定在一个列表中可见的数目

注意 DataCombo 控件的 DataFormat 属性是一个 Extender 属性。因此在属性表上它总是可见的，并且可以在代码中设置。然而，DataCombo 控件仅对其列表中最上端的项格式化。对于看到已格式化的顶端项的最终用户来说，这一点可能不太重要，只要从未格式化的列表中选择即可。已格式化的项可能也会误导最终用户，使他们以为项目要在格式化之后再输入数据库。由于这些原因，建议在使用 DataCombo 控件时不要设置 DataFormat 属性。

详细信息 如果要使用一个演示 BoundText 属性用法的循序渐进教程，请参阅“创建一个连接 DataList 控件的 DataGrid”。关于这些控件的属性和方法的完整列表，请参阅“DataList 控件”和“DataCombo 控件”。

使用 DataCombo 和 DataList 控件连接两个表

DataCombo 和 DataList 控件与众不同的特性是具有访问两个不同的表，并且将第一个表的数据链接到第二个表的某个字段的能力。这是通过使用两个数据源完成的（诸如 ADO Data 控件或 Data 环境）。

关系表和“不友好的”值

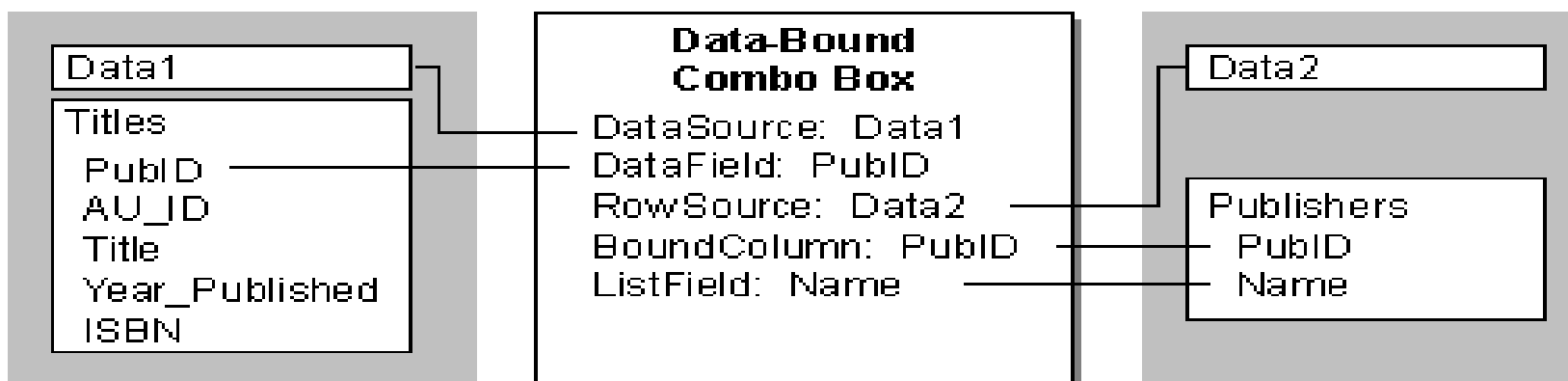
在一个关系数据库中，对于重复使用的信息并不是在多个地方都保存其全部的信息。大多数这种信息都保存在由多个字段组成的一个记录集中；在这些字段中有一个“标识符”字段来唯一地标识这个记录集。例如，Visual Basic 提供的 Biblio 数据库在一个名为 "Publishers" 的表中存储了若干个出版公司的名称。这个表包括很多字段，诸如地址、城市、邮政编码以及电话号码等。但是为了简单起见，只考虑这个表的两个本质字段 Name 和 PubID 字段。Name 字段存储一个出版商的名称，而 PubID 字段则存储一个相对“不友好的”值，如一个数或代码。但这个不友好的值是很重要的，因为这个值唯一地标识该出版商，并且可以作为一种链接整个记录集的手段。此外，这个值会存储在第二个表中的多个记录集中。

第二个表的名称为 "Titles"，其每个记录集包含的信息包括标题、出版年份、国际标准书号 ISBN 等。在这些字段中有一个字段的名称就是 "PubID"。这个字段的名称与 Publishers 表中的相应字段的名称相同，因为这个字段存储了将该标题和一个特定的出版商链接在一起的值。

这种可行方案提出了一个小问题：给定一个允许用户插入新标题的数据库应用程序，用户必须用某种方法输入标识出版商的整数。如果用户能记住每个出版商的唯一标识符，那么也还是可行的，不过如果一方面用户能看到出版商的名称，另一方面存入应用程序的又是数据库中相应的值，则会显得更加方便。而 DataList 和 DataCombo 控件就可以轻松地解决这个问题。

两个数据源、三个字段、无编码

DataList 和 DataCombo 控件使用两个数据源来解决这个问题。在只显示出版商的名称（来自 Publishers 表）的同时，DataList 或 DataCombo 控件只将 PubID 字段的值写入到 Titles 表。通过“属性”窗口，将 RowSource 设置为提供要写入的数据的数据源（即 Publishers 表）。然后将 DataSource 属性设置为要写入数据的数据源（即 Titles 表）。最后，设置 DataField、ListField 以及 BoundColumn 属性。下图演示了如何将两个数据源（以两个 Data 控件的形式）以及三个字段指定给一个 DataCombo 控件：



简要而言，ListField 属性决定该控件所显示的是哪一个字段。在本例中就是出版商的名称。另一方面，BoundColumn 属性则决定 Publishers 表中由哪一个字段向 Title 表供应实际所需的值。注意 Publishers 表中的 PubID 字段不能（也不应该）被编辑。相反，在 PubID 字段中的值将写入到由 DataField 属性所指定的字段。在本例中，这个属性就是 Titles 表中的 PubID 字段。

下表概要地介绍这些属性及其使用方法。

属 性	描 述
DataSource	DataList 或 DataCombo 所绑定的数据控件的名称
DataField	由 DataSource 属性所指定的记录集中的一个字段名称。这个字段将用于决定在列表中高亮显示哪一个元素。如果作出了新的选择，则它就是当移动到一个新记录时所需更新的字段
RowSource	将用于填充列表的数据控件的名称
BoundColumn	由 RowSource 属性所指定的记录集中的一个字段名称。这个字段必须和将用于更新该列表的 DataField 的类型相同
ListField	由将用于填充该列表的 RowSource 所指定的记录集中的一个字段名称

注意 DataList 和 DataCombo 控件也可以与单个数据控件一起使用。要实现这一点，可以将 DataSource 和 RowSource 属性设置为同一个数据控件，并且将 DataField 和 BoundColumn 属性设置为该数据控件的记录集中的同一个字段。在这种情形下，将使用 ListField 的值来填充该列表，且这些值来自于被更新的同一个记录集。如果指定了一个 ListField 属性，但没有设置 BoundColumn 属性，则 BoundColumn 将自动被设置为 ListField 字段。

详细信息 如果想使用 DataCombo 控件循序渐进地创建一个简单的数据库应用程序，请参阅“创建一个简单的 DataCombo 应用程序”。

创建一个简单的 DataCombo 应用程序

下面的示例使用 DataCombo 控件为 Northwind.mdb 示例数据库的 Titles 表创建一个数据输入界面。这个输入界面使用户可以输入新的产品，并通过提供一个包括所有供应商名称的查找表将这些新产品指定到已有的供应商。当用户在输入窗体中要输入供应商字段的值时，他们可以从一个列表框中选择一个供应商。当他们选定一个供应商后，该供应商的 SupplierID 字段值就复制到 Products 表的 SupplierID 字段。

要使用 DataCombo 控件来创建一个查找表

1. 给 Northwind 数据库创建一个 OLEDB 数据源。

如果还没有创建数据源，请按照“创建 Northwind 的 OLEDB 数据源”中的步骤进行。

2. 在 Visual Basic 中创建一个新的标准的 EXE 工程。

如果 DataGrid、DataCombo 或 ADO Data 控件不在“工具箱”中，则右键单击“工具箱”，然后使用“部件”对话框来添加控件。

3. 添加一个 DataCombo 控件、两个 ADO Data 控件以及一个 DataGrid 控件到窗体中。

4. 在“属性”窗口中，如下表所示设置第一个数据控件 (Adodc1) 的属性。

属性	设置值
Name	adoDataSource
ConnectionString	Northwind.mdl
RecordSource	Select * From Products;
Caption	Products

5. 在“属性”窗口中，如下表所示设置第二个数据控件 (Adodc2) 的属性。

属 性	设 置 值
Name	adoRowSource
ConnectionString	Northwind.mdl
RecordSource	Select CompanyName, SupplierID From Suppliers;
Caption	Suppliers
Visible	False

6. 在“属性”窗口中，如下表所示设置 DataGrid 控件的属性。

属 性	设 置 值
Name	GrdProducts
DataSource	AdoDataSource
Caption	Products

7. 在“属性”窗口中，如下表所示设置 DataCombo 控件的属性。

属 性	设 置 值
Name	dcbSuppliers
DataSource	adoDataSource
DataField	SupplierID
RowSource	adoRowSource
ListField	CompanyName
BoundColumn	SupplierID

8. 最后，将下述代码添加到该窗体的代码模块中：

```
Private Sub Form_Load()
```

```
' 在 DataGrid 控件中隐藏 SupplierID 字段，使用户  
' 不会混淆到底该更改哪一个值。  
grdProducts.Columns("SupplierID").Visible = False  
End Sub
```

9. 运行该工程。

可以通过单击可视的 ADO Data 控件上的箭头来浏览记录集。如果这样做，DataCombo 控件将更新和显示每一个产品的供应商的名称。如果要编辑 SupplierID 字段，则单击 DataCombo 控件的箭头来显示一个下拉列表，然后再单击一个不同的供应商，来改变写入到 SupplierID 字段的值。

使用 DataGrid 控件

DataGrid 控件是一种类似于电子数据表的绑定控件，可以显示一系列行和列来表示 Recordset 对象的记录和字段。可以使用 DataGrid 来创建一个允许最终用户阅读和写入到绝大多数数据库的应用程序。DataGrid 控件可以在设计时快速进行配置，只需少量代码或无需代码。当在设计时设置了 DataGrid 控件的 DataSource 属性后，就会用数据源的记录集来自动填充该控件，以及自动设置该控件的列标头。然后您就可以编辑该网格的列；删除、重新安排、添加列标头、或者调整任意一列的宽度。在运行时，可以在程序中切换 DataSource 来察看不同的表，或者可以修改当前数据库的查询，以返回一个不同的记录集合。

注意 DataGrid 控件与 Visual Basic 5.0 中的 DBGrid 是代码兼容的，除了一个例外：DataGrid 控件不支持 DBGrid 的“解除绑定模式”概念。DBGrid 控件包括在 Visual Basic 的 Tools 目录中。

可能的用法

- 查看和编辑在远程或本地数据库中的数据。
- 与另一个数据绑定的控件（诸如 DataList 控件）联合使用，使用 DataGrid 控件来显示一个表的记录，这个表通过一个公共字段链接到由第二个数据绑定控件所显示的表。

使用 DataGrid 控件的设计时特性

可以不编写任何代码，只通过使用 DataGrid 控件的设计时特性来创建一个数据库应用程序。下面的说明概要地说明了在实现 DataGrid 控件的典型应用时的一般步骤。完整的循序渐进的指示，请参阅主题“DataGrid 方案 1：使用 DataGrid 控件创建一个简单数据库应用程序”。

要在设计时实现一个 DataGrid 控件

1. 为要访问的数据库创建一个 Microsoft 数据链接 (.MDL) 文件。请参阅“创建 Northwind OLE DB 数据链接”主题，以获得一个示例。
2. 在窗体上放置一个 ADO Data 控件，并将其 ConnectionString 属性设置为在第 1 步中所创建的 OLE DB 数据源。
3. 在这个 ADO Data 控件的 RecordSource 属性中输入一条将返回一个记

录集的 SQL 语句。例如，

```
Select * From MyTableName Where CustID = 12
```

4. 在窗体上放置一个 DataGrid 控件，并将其 DataSource 属性设置为这个 ADO Data 控件。
5. 右键单击该 DataGrid 控件，然后单击“检索字段”。
6. 右键单击该 DataGrid 控件，然后单击“编辑”。
7. 重新设置该网格的大小、删除或添加网格的列。
8. 右键单击该 DataGrid 控件，然后单击“属性”。
9. 使用“属性页”对话框来设置该控件的适当的属性，将该网格配置为所需的外观和行为。

在运行时更改显示的数据

在创建了一个使用设计时特性的网格后，也可以在运行时动态地更改该网格的数据源。下面介绍实现这一功能的通常方法。

更改 DataSource 的 RecordSource

更改所显示的数据的最通常方法是改变该 DataSource 的查询。例如，如果 DataGrid 控件使用一个 ADO Data 控件作为其 DataSource，则重写 RecordSource 和刷新该 ADO Data 控件都将改变所显示的数据。

'ADO Data 控件连接的是 Northwind 数据库的'Products 表。新查询查找所有

'SupplierID = 12 的记录。

```
Dim strQuery As String
strQuery = "SELECT * FROM Suppliers WHERE SupplierID = 12"
Adodc1.RecordSource = strQuery
Adodc1.Refresh
```

更改 **DataSource**

在运行时，可以将 **DataSource** 属性重新设置为一个不同的数据源。例如，您可能具有若干个 **ADO Data** 控件，每个控件连接不同的数据库，或设置为不同的 **RecordSource** 属性。可以简单地将 **DataSource** 从一个 **ADO Data** 控件重新设置为另一个 **ADO Data** 控件：

- ' 将 **DataSource** 重新设置为一个连接到 **Pubs** 数据库的、
- ' 使用 **Authors** 表的 **ADO Data** 控件。

```
Set DataGrid1.DataSource = adoPubsAuthors
```

重新绑定 **DataSource**

当将 **DataGrid** 控件用于一个远程数据库，诸如 **SQL Server** 时，可以改变表的结构。例如，可以给这个表添加一个字段。在这种情形下，可以调用 **Rebind** 方法根据新的结构来重新创建该网格。注意，如果已经在设计时改变了这个列的布局，**DataGrid** 控件将会试图重新创建当前的布局，包括任何空的列。不过，通过首先调用 **ClearFields** 方法，可以强制该网格重新设置所有的列。

从 DataGrid 返回值

在 DataGrid 被连接到一个数据库后，可能想要监视用户单击了哪一个单元。可以使用 RowColChange 事件——而不是 Click 事件。如下所示：

```
Private Sub DataGrid1_RowColChange(LastRow As Variant, ByVal LastCol As Integer)
    ' 显示用户所单击的单元的文字、行和列的信息。
    Debug.Print DataGrid1.Text; DataGrid1.Row; DataGrid1.Col
End Sub
```

使用 CellText 和 CellValue 方法

当一个列使用 NumberFormat 属性设置格式后，CellText 和 CellValue 属性是很有用的。NumberFormat 属性不必更改实际的数据格式就可以更改任何包含数字的列的格式。例如，给定一个网格，其中包含一个名为 ProductID 的、包含整数的列。下面的代码将使 DataGrid 以 "P-0000" 的格式来显示数据。换句话说，尽管在 ProductID 字段中所包含的实际数值为 "3"，但该网格所显示的值将是 "P-0003"。

```
Private Sub Form_Load()
    DataGrid1.Columns("ProductID").NumberFormat = "P-0000"
End Sub
```

要返回数据库中所包含的实际值，应使用 CellValue 方法，如下所示：

```
Private Sub DataGrid1_RowColChange(LastRow As Variant, ByVal LastCol As Integer)
```

```
Debug.Print _  
DataGrid1.Columns("ProductID").CellValue(DataGrid1.Bookmark)
```

End Sub

注意 上面所用的 `CellValue` 和下面所用的 `CellText` 值，都需要将 `Bookmark` 属性作为一个参数，功能才正确。

相反地，如果要返回该字段的格式化的值，应使用 `CellText` 方法：

```
Private Sub DataGrid1_RowColChange(LastRow As Variant, ByVal LastCol As Integer)  
    Debug.Print _  
    DataGrid1.Columns("ProductID").CellText(DataGrid1.Bookmark)
```

End Sub

注意 上面的 `CellText` 方法等价于使用 `DataGrid` 控件的 `Text` 属性。

下一步

要阅读关于使用该控件创建简单的应用程序的一个循序渐进的过程，请参阅“使用 `DataGrid` 控件创建简单的数据库应用程序”，或“创建一个连接 `DataList` 控件的 `DataGrid`”。

要了解关于 `Split` 对象以及如何对其编程的详细信息，请参阅“操作 `DataGrid` 视图”。

创建 Northwind 的 OLE DB 数据链接

访问数据的一个重要步骤是为想要访问的每个数据库都创建一个 OLE DB 数据源。下面的步骤为 Visual Basic 所提供的 Nwind.mdb (Northwind) 数据库创建这样一个对象。这个数据源被用于 Visual Basic 文档所提供的一些示例过程。在一个计算机上只需要创建一次 OLE DB 数据源。

要创建 Northwind 的 OLE DB 数据源

1. 打开 Windows Explorer 或 Windows NT Explorer。
2. 打开您想要创建 OLE DB 数据源的目录。在该示例中，打开 Program Files、Microsoft Visual Studio 和 VB98。
3. 右键单击 Explorer 的右边窗格，然后单击上下文菜单上的“新建”。从文件类型列表中单击“Microsoft 数据链接”。
4. 重命名新文件 Northwind.MDL。
5. 右键单击文件并单击上下文菜单上的“属性”，以显示“Northwind.MDL Properties”对话框。
6. 单击“连接”选项卡。
7. 单击“提供方”框并选择“Microsoft Jet 3.51 OLE DB Provider”。
8. 在 Data Source 框中输入 nwind.mdb 文件的路径。
9. 单击“测试连接”，检测连接。
10. 如果连接通过，单击“确定”。

注意 也可以通过在“控制面板”中单击“数据链接”图标创建一个 OLE DB 数据源。在“管理数据链接文件”对话框中，单击“新建”创建一个新的数据源。

使用 DataGrid 和 ADO Data 控件创建一个简单的数据库应用程序

只使用一个 DataGrid 和一个 ADO Data 控件，可以创建一个允许最终用户阅读和写入记录集的数据库应用程序。

要使用 ADO 数据控件来创建一个简单的数据库应用程序

1. 为 Northwind 数据库创建一个 OLE DB 数据源。
如果还没有创建数据源，请按照“创建 Northwind 的 OLE DB Data Link”中的步骤操作。
2. 在 Visual Basic 中创建一个新的标准的 EXE 工程。
如果 DataGrid 控件不在“工具箱”中，则用右键单击“工具箱”，然后使用“部件”对话框来添加控件。同时也载入 ADO 控件。
3. 在空窗体上各放置控件的一个实例。
4. 将 ADO 控件的 ConnectionString 属性设置为 Northwind 的数据源。
单击并选定该 ADO Data 控件，并按 F4 键出现“属性”窗口。单击“ConnectionString”，然后单击 OLE DB File。单击 Northwind 的数据源。
5. 设置 ADO 控件的 RecordSource 属性。
在“属性”窗口中，单击“记录源”并输入一条 SQL 语句来填充 DataGrid

控件。在本例中，输入“Select * From Products”。

6. 将 DataGrid 控件的 DataSource 属性设置为这个 ADO Data 控件。

单击并选定该 DataGrid 控件。在其“属性”窗口中，单击“数据源”将出现一个包含所有数据控件的下拉列表——在本例中只有 ADO Data 控件。单击这个控件。

7. 按 F5 键运行这个工程。

创建一个连接 DataList 控件的 DataGrid

DataGrid 的通常用法是显示数据库的一个表所提供的“详细内容”。例如，Northwind (Nwind.mdb) 数据库包括两个表，一个名为 "Suppliers"，另一个名为 "Products"。在本例中，我们使用 DataList 控件来显示 "Suppliers" 表中的供应商的公司名称。当用户单击任意一个公司名称时，这个 DataList 控件将提供该公司的 SupplierID。使用这个标识符，就可以构造一个查询，在 "Products" 表中检索具有相匹配的 SupplierID 的所有记录。换句话说，当用户单击一个公司时（在 DataList 控件中），该公司生产的所有产品将出现在 DataGrid 控件中。

要使用一个指定供应商的产品填充一个 DataGrid 控件

1. 确认在机器上已为 Northwind 数据库建立了一个 OLE DB 数据源；如果还没有创建这样的一个数据源，请按照“创建 Northwind 的 OLE DB Data 连接”的步骤操作。
2. 在 Visual Basic 中创建一个新的标准的 EXE 工程。

如果 DataGrid、DataList 和 ADO Data 控件不在“工具箱”中，则右键单击“工具箱”，然后单击“部件”。在“部件”对话框中双击“Microsoft DataGrid Control”、“Microsoft DataList Controls”以及“Microsoft ADO Control”。

3. 在一个空窗体中各放置一个 DataGrid 和 DataList 控件的实例。
将 DataList 控件放置在该窗体的左上角，然后将 DataGrid 控件放在它的下面的某处。
4. 在窗体放置两个 ADO Data 控件实例。
选择第一个 ADO Data 控件，并按 F4 键来显示其“属性页”。将该控件的 Name 属性设置为 adoSuppliers。选择第二个 ADO Data 控件并将其 Name 属性设置为 adoProducts。将第一个控件直接放在 DataList 控件的下面，把第二个控件直接放在 DataGrid 控件的下面。
5. 将这两个 ADO Data 控件的 ConnectionString 属性设置为 Northwind 的 OLE DB 数据源。
选择名为 adoSuppliers 的控件，然后将其 ConnectionString 属性设置为 Northwind 的 OLE DB data source (Northwind.mdl)。选择名为 adoProducts 的控件，并重复该操作。
6. 设置这两个 ADO Data 控件的 RecordSource 属性。
选择 adoSuppliers 并在其“属性页”上单击“RecordSoure”。输入 Select * From Suppliers。这个查询将指示该 ADO Data 控件返回 Suppliers 表中的所有记录。选择 adoProducts，单击“RecordSoure”，并输入 Select * From Products。这个查询将返回在 Products 表中的所有记录。

7. 将 DataList 控件的 RowSource 属性设置为 adoSuppliers。
RowSource 属性决定由哪一个数据源为 ListField 属性供应数据。
8. 将 DataList 控件的 ListField 属性设置为 CompanyName。
ListField 属性被设置成名为 Suppliers 的表中的字段名称。在运行时，DataList 控件显示在这个属性中所指定的字段的值。在本例中，该属性将显示在 Suppliers 表中找到的一个公司名称。
9. 将 DataList 控件的 BoundColumn 属性设置为 SupplierID。
BoundColumn 属性被设为 Suppliers 表中的第二个字段。在本例中，这个属性就被设为 SupplierID 字段。当单击 DataList 控件时，BoundText 属性返回与在 DataList 控件中所显示的公司相关联的 SupplierID 字段的值。这个值将用于对 Products 表的查询，该查询为 DataGrid 控件提供数据。
10. 将 DataGrid 控件的 DataSource 属性设置为 adoProducts。
DataSource 属性为该控件指定数据源。在本例中，该属性被设置为名为 adoProducts 的 ADO Data 控件，这将返回 Products 表中的所有记录。
11. 在窗体的代码模块中，添加下述内容：

```
Private Sub Datalist1_Click()  
    ' 声明一个用来包含新查询的字符串变量。这个新的  
    ' 查询使用 DataList 控件的 BoundText 属性  
    ' 来提供一个 SupplierID 值。新查询查找所有  
    ' 具有相同的 SupplierID 的产品。这个查询被  
    ' 指定给名为 adoProducts 的 ADO Data 控件
```

' 的 RecordSource 属性。在刷新控件后，DataGrid
' 将使用包含由同一个公司供应的所有产品的新
' 记录集来更新。

```
Dim strQuery As String  
strQuery = "Select * FROM Products WHERE SupplierID = " & _  
Datalist1.BindText
```

```
With adoProducts  
    .RecordSource = strQuery  
    .Refresh  
End With
```

```
With DataGrid1  
    .ClearFields  
    .ReBind  
End With  
End Sub
```

12. 运行该工程。

单击 DataList 控件中的任意公司名称，将自动用该公司所供应的产品更新 DataGrid 控件。

使用列

通过更改 `DataSource` 属性，可以动态地更改在 `DataGrid` 控件中显示的数据。例如，可以显示同一个数据库的不同表。如果这样做，则 `DataGrid` 控件将只根据默认的属性显示数据。

添加、删除或隐藏列

通过使用 `Columns` 集合和 `Column` 对象的属性和方法，可以在程序中添加、删除或隐藏列。

添加和删除一列

要在运行时添加一列，可以使用 `Add` 方法。如果首先声明一个变量，并将新对象赋给该变量，就可以用简明的代码设置各种属性。

```
Private Sub AddColumn()  
    ' 在最右边的位置添加一列。然后设置其 Visible、Width、  
    ' Caption 以及 Alignment 属性。DataField 属性则指定  
    ' 该列将绑定到哪一个字段。  
    Dim c As Column  
    Set c = DataGrid1.Columns.Add(DataGrid1.Columns.Count)  
    With c
```

```
.Visible = True  
.Width = 1000  
.Caption = "我的新列"  
.DataField = Adodc1.Recordset.Fields("ProductName").Name  
.Alignment = dbgRight
```

```
End With
```

```
End Sub
```

可以使用方法来删除任意一列。请确保使用 `ColIndex` 参数来指定要删除的列。下面的代码将删除被单击的列。

```
Private Sub DataGrid1_HeadClick(ByVal ColIndex As Integer)
```

```
    DataGrid1.Columns.Remove ColIndex
```

```
End Sub
```

隐藏一列

通过将 `Visible` 属性设置为 `False`，可以隐藏任意一列。当想要限制用户可以查看或编辑的列时这一功能特别有用。下面的示例在 `Columns` 集合中循环，隐藏除少数列之外的所有列。

```
Private Sub HideColumns()
```

```
    ' 使用 DataField 属性来判别正在测试的是哪一列。
```

```
    ' 只显示三列：ProductName、UnitPrice 以及
```

```
    ' UnitsInStock。
```

```

Dim c As Column
For Each c In DataGrid1.Columns
    Select Case c.DataField
        Case "ProductName"
            c.Visible = True
        Case "UnitPrice"
            c.Visible = True
        Case "UnitsInStock"
            c.Visible = True
            c.Caption = "In Stock" ' 更改这个列的标头。
        Case Else ' 隐藏其它所有的列。
            c.Visible = False
    End Select
Next c
End Sub

```

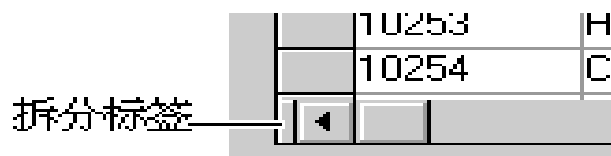
操作 DataGrid 视图

一个“拆分”的网格使最终用户对相同的数据可以拥有多个视图。例如，假设有一个由十个字段组成的大表。在这种情况下，在控件中察看的记录集将有十列宽，除非窗体非常宽，否则用户将无法同时看见所有列的内容。而且，假设用户只对第一列和最后一列感兴趣（例如，第一列是名字，最

后一列是电话号码)。为了能同时看到在两端的列（不重新安排列的顺序），可以对网格进行拆分。

创建一个 Split 对象

在设计时，可以创建一个拆分，具体步骤是：右键单击网格，单击“编辑”，再单击右键，然后单击“拆分”。通过右键单击该控件，并单击“属性”来显示“属性页”对话框，可以编辑这个拆分。可以使用“拆分”选项卡来自定义拆分。要删除一个拆分，右键单击该拆分，并单击“删除”。在运行时，最终用户也可以通过单击位于这个网格控件的左下边的右边的选项卡，以手工方式来拆分该网格（除非不允许这个操作），如下图所示：



默认情况下，DataGrid 控件包含一个 Split 对象。防止最终用户添加拆分的代码为：

```
DataGrid1.Splits(0).AllowSizing = False
```

在程序中添加和删除拆分

DataGrid 控件包含一个 Split 对象的集合。要在程序中添加拆分，可以使用 Add 方法，如下所示：

```
DataGrid1.Splits.Add 1
```

注意 Add 方法需要新的拆分索引作为其参数。要添加一个拆分，应将这个索引参数设置为 Splits 集合的 Count 属性值。

使用 Split 集合的 Add 方法，可以在程序中按照实际需要添加拆分。由于添加多于两个以上的拆分将使网格很难使用，可以使用该集合的 Count 属性来限制拆分的数目。

```
If DataGrid1.Splits.Count < 3 Then ' 添加一个拆分。
```

```
    DataGrid1.Splits.Add DataGrid1.Splits.Count
```

```
End If
```

使拆分同步

当拆分多于一个时，可能希望控制这些拆分如何滚动。例如，在一个具有三个拆分的网格中，可以决定只让第一个和第三个拆分同步，而让中间的拆分独立地滚动。要同步任何两个（或多个）拆分，只需将每个 Split 对象的 ScrollGroup 属性设置为同一个值。

' 使第一个和第三个 Split 对象同步。

```
With DataGrid1
```

```
    .Splits(0).ScrollGroup = 1
```

```
.Splits(1).ScrollGroup = 2
```

```
.Splits(2).ScrollGroup = 1
```

```
End With
```

通过设置 Scrollbars 属性，使同步的拆分组只显示一个滚卷条，从而进一步自定义拆分的外观。

控制 Tab 键和箭头键的行为

使用 `WrapCellPointer`、`TabAcrossSplits` 以及 `TabAction` 属性，可以决定当最终用户按下 `tab` 键或箭头键时网格的行为。

在这三个属性中，`TabAction` 属性级别最高，它决定 `WrapCellPointer` 和 `TabAcrossSplits` 这两个属性是否能生效。`TabAction` 有三个设置值：`Control Navigation`、`Column Navigation` 和 `Grid Navigation`。当该属性设置为 `Control Navigation` 时，按 `Tab` 键根据 `TabIndex` 将焦点切换到下一个控件。这一设置优先于 `WrapCellPointer` 和 `TabAcrossSplits`。

`WrapCellPointer` 属性决定在任何单个的拆分中 `tab` 键和箭头键的行为。如果该属性设置为 `True`，且当前单元位于最后一列，这时最终用户按 `tab` 键则使第一列的下一行变成当前的单元。不过，如果当前单元位于最后一行的最后一列时，这时就没有地方可以“换行”。

`TabAcrossSplits` 属性决定当网格中存在两个或多个拆分时 `tab` 和箭头键的行为。如果该属性设置为 `True`，且当前单元位于任何一个拆分的最后一列，则按 `Tab` 或箭头键将使当前单元“跳”到下一个拆分的第一列。当前单元仍保持相同的行位置。

注意 如果 `WrapCellPointer` 和 `TabAcrossSplits` 属性都设置为 `True`，则只有当前单元位于最后一个拆分的最后一列时才会换行。这时当前单元将换到第一个拆分的下一行。

自定义列集合

每一个 Split 对象都有一个 Columns 属性，允许用户来操作一个 Column 对象的集合。通过这样做，可以更改每个 Split 对象的外观。例如，可以用一个拆分包含显示姓氏字段和名字字段的两个列，而第二个拆分则显示电话字段和地址字段。要实现这一目标，需要将其余的每一列的 Visible 属性设置为 False，如下所示：

' 枚举 Columns 集合，对每一个 Column 对象的 DataField 属性

' 进行测试。如果测试失败，则隐藏这一列。

```
Dim i As Integer
```

' 隐藏除 ProductName 列之外的所有列。

```
For i = 0 To DataGrid1.Splits(0).Columns.Count - 1
```

```
    If DataGrid1.Splits(0).Columns(i).DataField <> "ProductName" Then
```

```
        DataGrid1.Splits(0).Columns(i).Visible = False
```

```
    End If
```

```
Next i
```

' 隐藏除 UnitPrice 列之外的所有列。

```
For i = 0 To DataGrid1.Splits(1).Columns.Count - 1
```

```
    If DataGrid1.Splits(1).Columns(i).DataField <> "UnitPrice" Then
```

```
        DataGrid1.Splits(1).Columns(i).Visible = False
```

```
    End If
```

```
Next I
```

使用 Bookmarks 和 SelBookmarks 跟踪记录

Bookmarks 和 SelBookmarks 提供了标记记录的一种手段。当编写应用程序中的特定功能（诸如允许最终用户手工地选择多个不相邻的记录，进行所选记录的大批更新）时，这就很有必要。在这些情形中，需要标记哪些记录已被选择，因此可以使用 SelBookmarks 集合及其属性。有两个函数，分别是 CellText 和 CellValue 方法，需要标记才能正确执行。

标记用户的选择

SelBookmarks 集合包含所有选定的记录的书签。当最终用户手工选择记录时（即在单击时按住 CTRL 键），每一个选定的记录的书签都会加入到该集合中。使用标准的循环，用户可以知道已经选定了什么，也可以保存书签（因为可能需要恢复某个值），以及执行操作：

```
Dim i as Integer ' 计数器
Dim intCount As Integer
intCount = DataGrid1.SelBookmarks.Count - 1
ReDim arrSelBK(intCount) ' 声明用于保存书签的数组。
For i = 0 To intCount
    ArrSelBK(i) = DataGrid1.SelBookmarks(i)
    ' 在此处执行操作。如果该操作必须被
    ' 取消，则退出该循环，然后使用该数
```


' 组来取消这些更改。

Next i

通过在程序中添加 **SelBookmarks** 集合来选择记录

通过将记录添加到这个集合，也可以在程序中选定记录。例如，可能有一个显示指定的客户所有订货的网格。如果要高亮显示该客户花费超过 \$100 的所有记录，则对记录进行过滤，并将结果书签添加到 **SelBookmarks** 集合。

```
Dim rs As Recordset
```

```
Set rs = Adodc1.Recordset
```

```
While Not rs.EOF
```

```
    If rs!SupplierID = 12 Then
```

```
        DataGrid1.SelBookmarks.Add rs.Bookmark
```

```
    End If
```

```
    rs.MoveNext
```

```
Wend
```

显示计算结果字段

假设在表中有一个名为 "Price" 的字段，并且想使用本地税率来计算表中

每一项的税费。这就是一个计算结果字段，可以通过修改 `DataSource` 的查询来计算这个值，并把这个值返回给 `DataGrid` 控件。

要在 `DataGrid` 控件中创建一个计算结果字段

1. 确认在机器上已为 `Northwind` 数据库建立了一个 `OLE DB` 数据源；如果还没有创建这样的数据源，请按照“创建 `Northwind` 的 `OLE DB Data` 连接”的步骤操作。
2. 在窗体上放置一个 `ADO Data` 控件和一个 `DataGrid` 控件。
3. 将 `ADO Data` 控件的 `ConnectionString` 属性设置为 `Northwind` 的数据源。
4. 设置 `ADO Data` 控件的 `RecordSource` 属性。
在“属性”窗口中，单击“记录源”并输入 `Select ProductName, UnitPrice, (UnitPrice * .082) As Tax From Products`。
5. 将 `DataGrid` 控件的 `DataSource` 属性设置为这个 `ADO Data` 控件。
6. 运行该工程。

与类模块一起使用 `DataGrid` 控件

如果想要访问以自定义格式或以 `ODBC` 驱动程序不直接支持的格式存放的数据，可以创建一个类来封装该数据。然后可以编写该类的自定义函数来检索这些数据。这样该类就变成了一种数据源，可以被任何数据使用者（如 `DataGrid` 控件）使用。

在这个类模块的 `Initialize` 事件中，首先通过声明一个作为 `New`

ADODB.Recordset 的变量，来创建一个 ADODB recordset 对象。在创建了这个 recordset 对象后，再添加字段，每个数据源中的每个字段都要加入。然后使用合适的数据填充这个记录集。

注意 也可以使用 OLEDB 示例提供者来创建一个数据源。关于 OLEDB 示例提供者的详细信息，请参阅“创建带有数据提供方的部件”。

类模块有一个 GetDataMember 事件，只要当数据使用者（诸如 DataGrid 控件）需要数据时就产生该事件。在这个事件中，Data 参数被设置为在 Initialize 事件中所创建的 recordset 对象。

如果要使用这个类模块，应创建一个具有一个 DataGrid 控件的窗体。在该窗体的 Load 事件的代码中，将该控件的 DataSource 属性设置为这个类。

注意 数据类模块在设计时是不可用的。例如，如果使用 DataGrid 控件，则当用户在“属性”窗口中单击“数据源”时，所有可用的数据源都会出现在一个下拉列表中。但其中不会有这个数据类模块，它只能在代码中设置。

使用类模块创建一个数据源

下面的示例使用一个类模块来创建一个简单数据源。然后通过 DataSource 属性将 DataGrid 控件绑定到该模块。

要创建一个用于 DataGrid 的类

1. 创建一个新的标准 Exe 工程。
2. 给窗体添加一个 DataGrid 控件。

如果 DataGrid 控件不在“工具箱”中，则在“工程”菜单中单击“部件”，

再单击“Microsoft DataGrid Control”，然后单击“确定”。

3. 在“工程”菜单中，单击“引用”。在“引用”对话框中，单击“Microsoft ActiveX Data Objects 2.0 Library”。

4. 在“工程”菜单中，单击“添加类模块”来给工程添加一个数据类模块。

5. 在“工程资源管理器”窗口中，单击并选定“类”图标，并按 F4 键显示“属性”窗口。

6. 在“属性”窗口中，将类的名称更改为 NamesData。

7. 在“属性”窗口中，单击“DataSourceBehavior”并将该属性更改为 vbDataSource。

8. 在该类模块的 Declarations 部分，创建一个 ADODB Recordset 变量，如下所示：

```
Option Explicit
```

```
Private WithEvents rsNames As ADODB.RecordSet
```

使用 WithEvents 关键词来声明该变量，使用户可以对 RecordSet 对象的事件编程。

9. 在该类的 Initialize 事件中，添加下述代码：

```
Private Sub Class_Initialize()
```

```
    ' 将新的数据成员的名称添加到 DataMember 集合
```

```
    ' 这使其它对象可以看见这些可用的 DataMembers
```

```
    DataMembers.Add "Names"
```

```
    Set rsNames = New ADODB.RecordSet ' 设置对象变量。
```

```
' 创建一个具有两个字段的 recordset, 并打开该 recordset。  
' 第一个记录具有一个整数的数据类型, 第二个记录是一个最大可  
' 达 256 个字符的字符串。CursorType 被设置为 OpenStatic  
' —— 一个可更新的对一组记录的快照。LockType 被设置为  
' LockOptimistic, 以允许对该 recordset 进行更新。
```

```
With rsNames
```

```
    .Fields.Append "ID", adInteger  
    .Fields.Append "Name", adBSTR, 255  
    .CursorType = adOpenStatic  
    .LockType = adLockOptimistic  
    .Open
```

```
End With
```

```
Dim i As Integer
```

```
For i = 1 to 10 ' 添加十条记录。
```

```
    rsNames.AddNew  
    rsNames!ID = i  
    rsNames!Name = "Name " & i  
    rsNames.Update
```

```
Next i
```

```
rsNames.MoveFirst ' 移到该记录集的开始。
```

```
End Sub
```

这部分代码首先创建 `recordset` 对象，然后给该对象添加两个字段。代码接着给 `recordset` 添加十条记录。

10. 在该类的 `GetDataMember` 事件中，添加下述代码：

```
Private Sub Class_GetDataMember(ByVal DataMember As String, _  
Data As Object)  
    Set Data = rsNames
```

```
End Sub
```

只要发生该事件——即当该类对象被绑定到一个数据使用者，如 `DataGrid` 控件时，代码将返回该 `recordset` 对象。

11. 在 `Form` 对象的代码模块中，声明一个数据类的对象变量：

```
Option Explicit
```

```
Private datNames As NamesData ' 类变量
```

12. 在 `Form` 对象的 `Load` 事件的代码中，将 `DataGrid` 控件的 `DataSource` 设置为该类对象。

```
Private Sub Form_Load()  
    ' 创建一个新的 NamesData 对象  
    Set datNames = New NamesData  
  
    ' 将这个 DataGrid 绑定到新的数据源 datNames  
    Set DataGrid1.DataSource = datNames  
End Sub
```

13. 按 F5 键运行该工程。

RecordSet 事件的编程

也可以对这个 Recordset 对象的事件进行编程。在该类模块中，单击“对象”框（在位于左上角），然后单击“rsNames”。在“过程/事件”框（在右上角）中，下拉列表将显示这个 Recordset 对象的所有事件。

给类添加一个属性

类模块也可以进行修改，来响应事件或函数调用。下面的代码演示了如何先给类添加属性。当从其它对象调用该对象时，这一属性将返回该类的 RecordCount。

```
Public Property Get RecordCount() As Long
    RecordCount = rsNames.RecordCount
End Sub
```

使用 DataMember 属性

GetDataMember 事件也包括 DataMember 参数。使用这个参数，可以在类模块中包括多个记录集，并使用带 DataMember 参数的 Select Case 语句来返回相应的记录集：

```
Private Sub Class_GetDataMember(ByVal DataMember As String, Data As _
Object)
```

```

Select Case DataMember
Case "Names"
    Set Data = rsNames
Case "Dates"
    Set Data = rsDates
Case Else
    ' 设置一个默认的数据成员
    Set Data = rsYears
End Select

```

End Sub

如果要指定所需的 DataMember，请将数据使用者的 DataMember 属性设置为适当的字符串，然后如平常一样设置 DataSource。对于 DataGrid 控件，可以采用如下方法：

```

Private Sub Form_Load()
    ' 创建一个新的 NamesData 对象
    Set datNames = New NamesData

    ' 指定所需的 DataMember，然后设置 DataSource。
    DataGrid1.DataMember = "Names"
    Set DataGrid1.DataSource = datNames

```

End Sub

使用文件系统控件（Directory List Box、Drive List Box 和 File List Box）

许多应用程序必须显示关于磁盘驱动器、目录和文件的信息。为使用户能够利用文件系统，Visual Basic 提供了两种选择。可以使用由 CommonDialog 控件提供的标准对话框，或者使用 DirListBox、DriveListBox 和 FileListBox 这三种特殊的控件的组合创建自定义对话。

文件系统控件使用户能在应用程序中检查可用的磁盘文件并从中选择。若只需要标准的“打开文件”或“保存”对话框，则应考虑使用 CommonDialog 控件。

详细信息 请参阅本章前面的“使用 CommonDialog 控件”。

示例应用程序：Winseek.vbp

许多代码示例都来自 WinSeek 示例应用程序 (Winseek.vbp)，该程序列在 Samples 目录中。

检查文件系统

每个文件系统控件都经过精心设计，将灵活、复杂的文件系统检查功能与

简易的编程方法结合起来。每个控件都自动执行文件数据获取任务，但也可编写代码自定义控件外观并指定显示的信息。

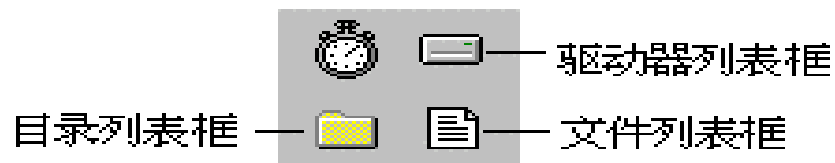


图 7.16 文件系统控件

可单独使用文件系统控件，也可组合起来使用。组合使用时，可在各控件的事件过程中编写代码来判断它们之间的交互方式。也可让它们独立操作。图 7.17 显示了一起使用的三个控件。

文件系统控件自动从操作系统获取一切信息；可访问此信息或判断每个控件通过其属性显示的信息。例如，在缺省时显示当前工作目录的内容（即启动应用程序的目录，或者由 `ChDir` 语句改变的当前目录）。

应用程序也可显示文件列表，这些文件的名称匹配一种模式，例如 `*.frm`。只需在窗体上绘制一个文件列表，设置其 `Pattern` 属性为 `*.frm`。运行时，可用下列代码指定 `Pattern` 属性：

```
File1.Pattern = "*.FRM"
```

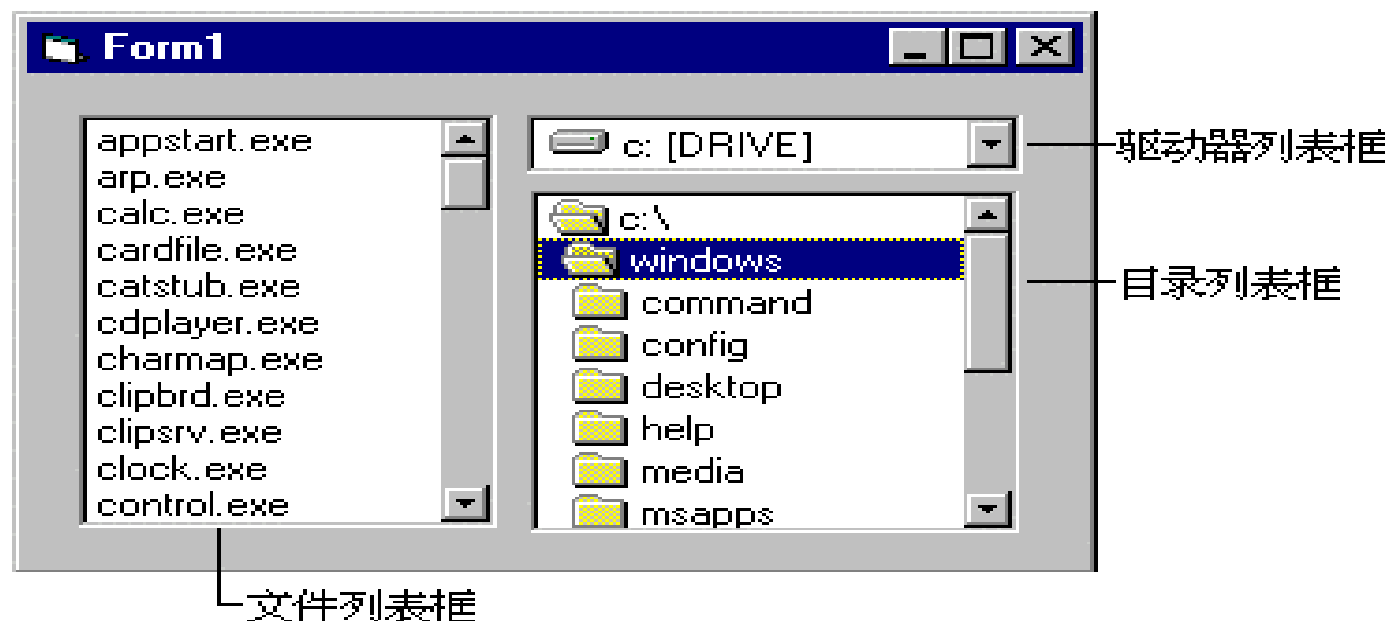


图 7.17 一起使用的三个文件系统控件

文件系统控件使操作非常灵活，这是 `CommonDialog` 控件无法做到的。可以用多种方法混合、匹配它们，并可控制它们的外观和交互方式。如果目的就是要让用户能够打开和保存文件，则 `CommonDialog` 控件为这些操作提供了现成的一组对话框。这些对话框也为许多其它基于 `Microsoft Windows` 的应用程序所使用，因此具有标准化的外观。这些控件也能识别可用的网络驱动器。

详细信息 请参阅本章前面的“使用 `CommonDialog` 控件”。

驱动器列表框

驱动器列表框是下拉式列表框。在缺省时在用户系统上显示当前驱动器。当该控件获得焦点时，用户可输入任何有效的驱动器标识符，或者单击驱动器列表框右侧的箭头。用户单击箭头时将列表框下拉以列举所有的有效驱动器。若用户从中选定新驱动器，则这个驱动器将出现在列表框的顶端。可用代码检查 **Drive** 属性来判断当前选择的驱动器。应用程序也可通过下述简单赋值语句指定出现在列表框顶端的驱动器：

```
Drive1.Drive = "c:\"
```

驱动器列表框显示可用的有效驱动器。从列表框中选择驱动器并不能自动地变更当前的工作驱动器；然而可用 **Drive** 属性在操作系统级变更驱动器，这只需将它作为 **ChDrive** 语句的参数：

```
ChDrive Drive1.Drive
```

目录列表框

目录列表框从最高层目录开始显示用户系统上的当前驱动器目录结构。起初，当前目录名被突出显示，而且当前目录和在目录层次结构中比它更高层的目录一起向根目录方向缩进。在目录列表框中当前目录下的子目录也缩进显示。在列表中上下移动时将依次突出显示每个目录项。

标识单个目录

列表框中的每个目录关联一个整型标识符，可用它来标识单个目录。CommonDialog 控件没有提供这个功能。Path 属性 (Dir1.Path) 指定的目录总是有 ListIndex 值 -1。紧邻其上的目录具有 ListIndex 值 - 2，再上一个为 ListIndex- 3，依次类推。Dir1.Path 的第一个子目录具有 ListIndex 值 0。如图 7.18 所示，若第一级子目录有多个目录，则每个目录的 ListIndex 值按 1、2、3… 的顺序依次排列。



图 7.18 目录列表框中显示的目录结构

设置当前目录

可用目录列表框的 Path 属性设置或返回列表框中的当前目录 (ListIndex = -1)。例如，若为图 7.18 中的 Drive1.Path 赋以 "c:\payroll"，则目录 \Pagroll 将成为当前工作目录。

同样，可把驱动器列表框的 `Drive` 属性赋予目录列表框的 `Path` 属性：

```
Dir1.Path = Drive1.Drive
```

执行赋值语句时，目录列表框将显示此驱动器上所有有效的目录和子目录。缺省时，目录列表框将显示驱动器的当前目录的所有上级目录以及下一级子目录，而驱动器是被指定给 `Dir1.Path` 属性的。目录列表框并不在操作系统级设置当前目录；它只是突出显示目录并将其 `ListIndex` 值设置为 `-1`。

为设置当前工作目录应使用 `ChDir` 语句。例如，下列语句将当前目录变成目录列表框中显示的一个目录：

```
ChDir Dir1.Path
```

在使用文件控件的应用程序中，可用 `Application` 对象将当前目录设置成应用程序的可执行（.exe）文件所在目录：

```
ChDrive App.Path '设置驱动器。
```

```
ChDir App.Path '设置目录。
```

注意 `Path` 属性只在运行时可用，在设计时不可用。

详细信息 关于 `Application` 对象的详细信息，请参阅《Visual Basic 6.0 语言参考》中“App 对象”。

单击目录项目

单击目录列表框中的某个项目时将突出显示该项目。而双击项目时则把它赋予 `Path` 属性并把其 `ListIndex` 属性设置为 `-1`，然后重绘目录列表框以

显示直接相邻的下级子目录。

查找目录的相对位置

ListCount 属性返回当前扩展目录下的目录数目，而不是目录列表框中的目录总数。因为当前扩展目录的 **ListIndex** 值总为 - 1，所以可以编程来判断在目录层次结构中，当前扩展目录与根目录有多大距离。例如：

'初始化当前扩展目录。

GoHigher = 0

'若目录不存在，

'则 **Dir1.List(x)** 返回空字符串。

Do Until Dir1.List(GoHigher) = ""

GoHigher = GoHigher - 1

Loop

'如有必要可转换成正数。

LevelsAbove = Abs(GoHigher)

文件列表框

文件列表框在运行时显示由 **Path** 属性指定的包含在目录中的文件。可用下列语句在当前驱动器上显示当前目录中的所有文件：

File1.Path = Dir1.Path

然后，可设置 **Pattern** 属性来显示这些文件的子集——例如，设置为 ***.frm** 后将只显示这种扩展名的文件。**Pattern** 属性也接受由分号分隔的列表。例如，下列代码行将显示所有扩展名为 **.frm** 和 **.bas** 的文件：

```
File1.Pattern = "*.frm;*.bas"
```

Visual Basic 支持 **?** 通配符。例如，**???.txt** 将显示所有文件名包含三个字符且扩展名为 **.txt** 的文件。

使用文件属性

文件列表框的属性也提供当前选定文件的属性（**Archive**、**Normal**、**System**、**Hidden** 和 **ReadOnly**）。可在文件列表框中用这些属性指定要显示的文件类型。**System** 和 **Hidden** 属性的缺省值为 **False**。**Normal**、**Archive** 和 **ReadOnly** 属性的缺省值为 **True**。

例如，为了在列表框中只显示只读文件，直接将 **ReadOnly** 属性设置为 **True** 并把其它属性设置为 **False**：

```
File1.ReadOnly = True
```

```
File1.Archive = False
```

```
File1.Normal = False
```

```
File1.System = False
```

```
File1.Hidden = False
```

当 **Normal = True** 时将显示无 **System** 或 **Hidden** 属性的文件。当 **Normal = False** 时也仍然可显示具有 **ReadOnly** 和/或 **Archive** 属性的文件，只需将

这些属性设置为 `True`。

注意 不使用 `attribute` 属性设置文件属性。应使用 `SetAttr` 语句设置文件属性。

缺省时，在文件列表框中只突出显示单个选定文件项。要选定多个文件，应使用 `MultiSelect` 属性。

详细信息 关于 `SetAttr` 语句的详细信息，请参阅《Visual Basic 6.0 语言参考》的“`SetAttr` 语句”和“`MultiSelect` 属性”。

使用文件系统控件的组合

如果使用文件系统控件的组合，则可同步显示信息。例如，若有缺省名为 `Drive1`、`Dir1` 和 `File1` 的驱动器列表框、目录列表框和文件列表框，则事件可能按如下顺序发生：

1. 用户选定 `Drive1` 列表框中的驱动器。
2. 生成 `Drive1_Change` 事件，更新 `Drive1` 的显示以反映新驱动器。
3. `Drive1_Change` 事件过程的代码使用下述语句，将新选定项目（`Drive1.Drive` 属性）赋予 `Dir1` 列表框的 `Path` 属性：

```
Private Sub Drive1_Change ()  
    Dir1.Path = Drive1.Drive  
End Sub
```

4. `Path` 属性赋值语句生成 `Dir1_Change` 事件并更新 `Dir1` 的显示以反映新驱动器的当前目录。

5. Dir1_Change 事件过程的代码将新路径（Dir1.Path 属性）赋予 File1 列表框的 File1.Path 属性：

```
Private Sub Dir1_Change ()  
    File1.Path = Dir1.Path  
End Sub
```

6. File1.Path 属性赋值语句更新 File1 列表框中的显示以反映 Dir1 路径指定。

用到的事件过程及修改过的属性与应用程序使用文件系统控件组合的方式有关。下节“文件系统控件方案：文件搜索器应用程序”演示了这里描述的控件同步。

文件系统控件方案：文件搜索器应用程序

因为用户常常希望快速查找应用程序的可用文件或文件组，所以，许多应用程序都提供查询文件系统的功能。Winseek.vbp 示例应用程序协助用户浏览驱动器 and 目录并显示所有类型的文件。

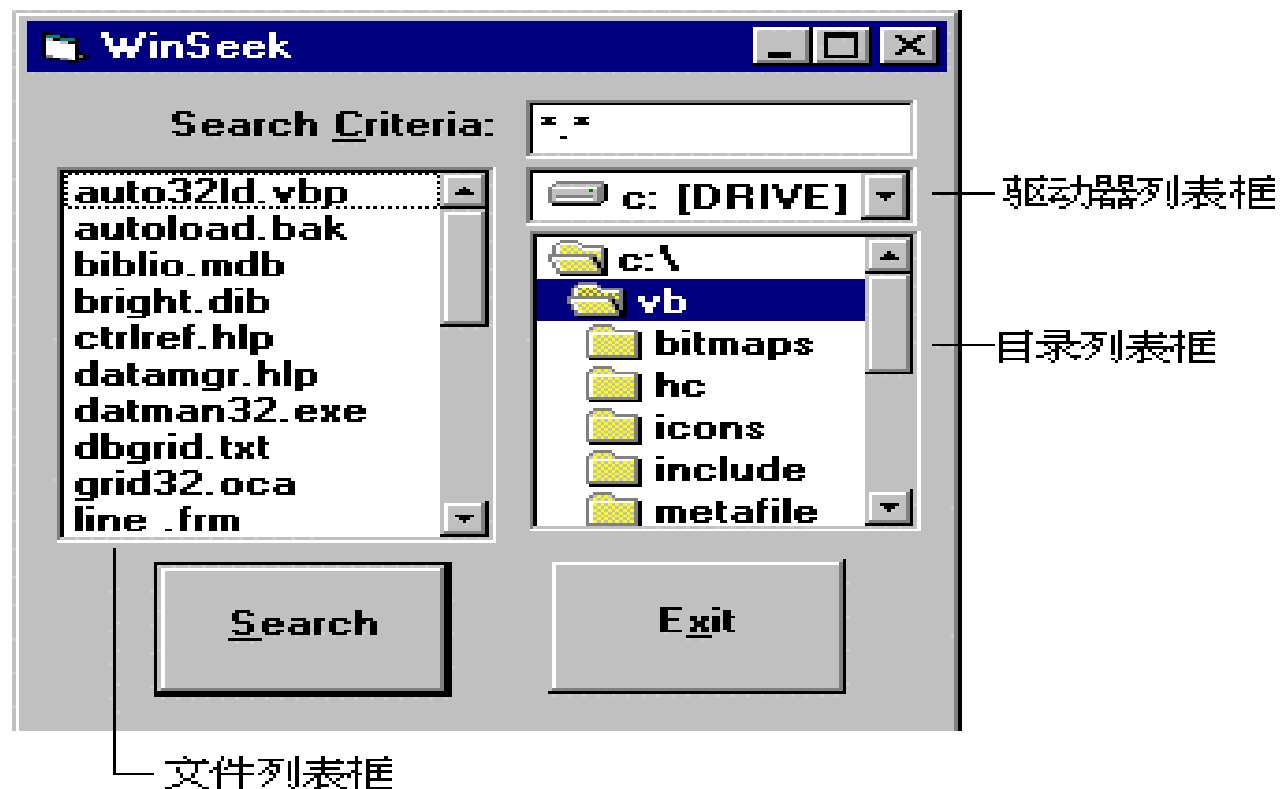


图 7.19 WinSeek 应用程序中的文件系统控件

下表总结了 WinSeek 应用程序在 Seek.frm 窗体中使用的控件。

控 件	属 性	设 置 值
DriveListBox	Name	drvList
DirListBox	Name	dirList
FileListBox	Name	FilList
	Pattern	*.*

第 一 个	Name	CmdSearch
CommandButton	Caption	& Search
	Default	True
第 二 个	Name	CmdExit
CommandButton	Caption	E& xit
ListBox	Name	LstFoundFiles

注意 文件系统控件没有 `caption` 属性，虽然可为它们加标签并指定访问键。关于这种使用标签的方法的详细信息，请参阅本章后面的“使用 `Label` 控件”一节。

编写 WinSeek 应用程序的代码

用鼠标单击驱动器列表框中的项目就会触发 `Change` 事件。当用户选定项目并接着改变窗体上的焦点时也将触发 `Change` 事件。在目录列表框中，需要 `Db1Click` 事件生成 `Change` 事件。

当用户不想用鼠标变更目录时，通常使用箭头键选定需要的目录，然后按 `ENTER` 键。

因为 `ENTER` 键通常与缺省 `CommandButton` 控件关联，所以，`WinSeek` 必须识别用户在何时只是想变更目录而不搜索文件。

通过判断 `dirList` 列表框的路径是否不同于当前突出显示的目录，`WinSeek` 应用程序解决了这个问题。当用户单击目录列表框中的项目或用箭头键在

目录列表框中移动时，可能发生这种情况。下列代码判断 `dirList.Path` 是否与突出显示的目录不同。若不同，则更新 `dirList.Path`。若相同，则执行搜索操作。

```
Private Sub cmdSearch_Click()
```

```
    .
```

```
    .
```

```
    .
```

```
'若 dirList.Path 与当前选定目录不同，
```

```
'则执行更新操作；否则执行'搜索操作。
```

```
If dirList.Path <> dirList.List(dirList.ListIndex) Then
```

```
    dirList.Path = dirList.List(dirList.ListIndex)
```

```
    Exit Sub
```

```
End If
```

```
'继续搜索。
```

```
    .
```

```
    .
```

```
    .
```

```
End Sub
```

WinSeek 应用程序使用下述过程处理重要的事件：

- `drvList_Change` 过程
- `dirList_Change` 过程
- `cmdSearch_Click` 过程

DriveListBox 控件的 Change 事件

当用户单击驱动器列表框的项目时就会生成控件的 **Change** 事件。此时将调用 **drvList_Change** 事件过程并运行下述代码：

```
Private Sub drvList_Change ()  
    On Error GoTo DriveHandler  
    '若选定新驱动器，则 Dir1 列框'更新显示。  
    dirList.Path = drvList.Drive  
    Exit Sub  
'若发生错误，则用 dirList.Path 重新设置 drvList.Drive。  
DriveHandler:  
    drvList.Drive = dirList.Path  
    Exit Sub  
End Sub
```

注意，无论用鼠标单击还是移动选项（例如，利用箭头键移动），在选定新驱动器时都将在驱动器列表框中发生 **Change** 事件。试图访问未关闭驱动器门的软盘驱动器，或者选定一个无意中已切断连接的网络驱动器，诸如此类的操作都将触发错误处理程序。因为出错后不能按原先的安排赋值，所以 **dirList.Path** 仍包含先前的有效驱动器。将 **dirList.Path** 重新赋予 **drvList.Drive** 后就可更正这个错误。

详细信息 请参阅第十三章“调试代码和处理错误”。

DirListBox 控件的 Change 事件

如果用户双击目录列表框中的项目或在代码中（如 `drvList_Change` 过程中）更改 `dirList` 的 `Path` 属性，就会启动 `dirList_Change` 事件。下列代码响应此事件：

```
Private Sub dirList_Change ()  
    '更新文件列表框，以便与目录列表框同步。  
    filList.Path = dirList.Path  
End Sub
```

此事件过程将 `dirList` 列表框的 `Path` 属性赋予 `filList` 列表框的 `Path` 属性。这将在 `filList` 列表框中触发重新绘制的 `PathChange` 事件；不必在 `filList_PathChange` 过程中添加代码，因为此应用程序中的事件在 `filList` 列表框结束。

CommandButton 控件的 Click 事件

此事件过程判断 `dirList` 列表框中突出显示的项目是否和 `dirList.Path` 相同。若项目不同，则更新 `dirList.Path`；否则执行搜索操作。

```
Private Sub cmdSearch_Click ()  
    .  
    .  
    .  
    '若 dirList.Path 与当前选定的目录不同，
```

'则更新；否则执行搜索。

```
If dirList.Path <> dirList.List _
```

```
(dirList.ListIndex) Then
```

```
    dirList.Path = dirList.List(dirList.ListIndex)
```

```
    Exit Sub
```

```
End If
```

'继续搜索。

.

.

.

```
End Sub
```

注意 可用附加功能增强 WinSeek 应用程序。例如会出现这样的情况，希望使用文件控件的属性。这时可通过使用复选框使用户设置文件属性的不同组合，以使文件列表框显示 Hidden、System 等等属性的文件。这将把搜索限制在满足条件的文件中。

使用 Frame 控件

Frame（框架）控件可以用来对其他控件进行分组，以便于用户识别。使用框架控件可以将一个窗体中的各种功能进一步进行分类，例如，将各种选项按钮控件分隔开。



图 7.20 Frame 控件

在大多数的情况下，框架控件的用法是比较“消极的”，我们用它对控件进行分组，但是通常没有必要响应它的事件。不过，它的 `Name`、`Caption` 和 `Font` 属性是经常被修改的。

详细信息 请参阅第三章“窗体、控件和菜单”中的“使用选项按钮分组选项”，该部分简单地演示了如何使用框架控件对选项按钮进行分组。

在窗体中添加一个 Frame 控件

在使用框架控件分组其他选项的时候，请首先绘出框架控件，然后再绘制它内部的其他控件。这样在移动框架的时候，可以同时移动它包含的控件。

在框架内部控制控件

要将控件加入到框架中，只需将它们绘制在框架的内部即可。如果将控件

绘制在框架之外，或者在向窗体添加控件的时候使用了双击方法，然后将它移动到框架控件内部，那么控件将仅仅“位于”框架的顶部，在进行移动的时候将不得不分别移动框架和控件。

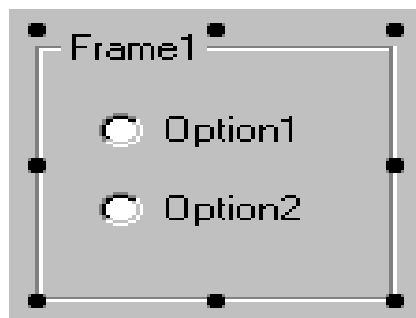


图 7.21 位于框架内部的控件

注意 如果希望将已经存在的若干控件放在某个框架中，可以先选择所有控件，将它们剪贴到剪贴板上，然后选定框架控件并把它们粘贴到框架上。

选择框架中的多个控件

要选择框架中的多个控件，在使用鼠标拉框包围控件的时候需要按下 **CTRL** 键。在释放鼠标的时候，位于框架之内的控件将被选定，如图 7.22 所示：

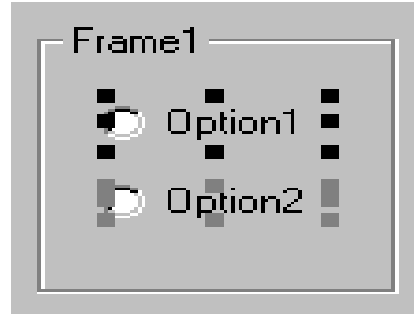


图 7.22 选择框架内部的控件

使用 Microsoft Hierarchical FlexGrid 控件

Microsoft Hierarchical FlexGrid (MSHFlexGrid) 和 Microsoft FlexGrid (MSFlexGrid) 控件以网格的形式显示 Recordset 数据，数据可以来自单个表或者多个表。

Hierarchical FlexGrid 控件提供了在网格中显示数据的高级功能。它与 Microsoft Data Bound 网格 (DataGrid) 控件类似，但也有显著区别：Hierarchical FlexGrid 控件不允许用户对它绑定或包含的数据进行编辑。因此，这种控件在显示数据的同时能够确保原始数据的安全，使数据不被用户修改。不过，通过将它与文本框结合起来使用，Hierarchical FlexGrid 控件的单元格编辑能力也是可以实现的。

尽管 Hierarchical FlexGrid 控件是 Visual Basic 5.0 中使用的 FlexGrid 控

件， Hierarchical FlexGrid 控件是比较灵活的。 Hierarchical FlexGrid 控件还提供了更多的显示选项， 利用这些选项我们可以定义出最适合于自己需要的自定义格式。

这里的各个主题主要集中在如何使用 Hierarchical FlexGrid 上。关于早期的 FlexGrid 控件的详细信息， 请参阅有关的 Visual Basic 5.0 文档。

Visual Basic 的 FlexGrid 控件

图 标	缩 写	控 件 名 称
	M SHFlexGrid 控件	Microsoft Hierarchical FlexGrid 控件
	M SFlexGrid 控件	Microsoft FlexGrid 控件

Hierarchical FlexGrid 控件支持以下功能：

- 只读的数据绑定。
- 动态重新安排列和行。
- 在进行列调整的过程中重新对数据进行分组。
- 利用现有的、为 DBGrid 控件编写的 Visual Basic 代码。
- 每个单元格中可以包含文本、图片或者同时包含文本和图片。
- 在代码中或者在运行时 defRunTime 改变当前的单元格文本。
- 当 Hierarchical FlexGrid 被指派到一个数据控件时，自动读取数据。
- 单元格中的文本换行功能。

- 当控件的 DataSource 和 DataMember 属性被绑定到特定的数据提供者时，提供 ActiveX Data Binding。
- 通过 Visual Basic 中的 Data Binding Manager 进行绑定。
- 直接绑定到 Command 分层结构中的分组的和相关的 ADO Recordsets。
- 当 Hierarchical FlexGrid 被绑定到一个 Recordsets 分层结构时，提供其他显示选项。这些附加的选项支持以多种方式显示分组的和相关的 Recordsets，包括带区。

注意 如果要利用标记了星号 (*) 的功能，必须使用 Hierarchical FlexGrid 控件。这些功能是 FlexGrid 控件所没有的。

由于以往的 FlexGrid 控件的某些局限，该控件不能够访问包括带区在内的某些功能。如果希望访问所有的功能，在创建新的数据绑定网格控件的时候，建议使用 Hierarchical FlexGrid 控件。FlexGrid 控件不能够自动升级到 Hierarchical FlexGrid 控件。

访问 Hierarchical FlexGrid 控件

要在 Visual Basic 中安装并访问 Hierarchical FlexGrid 控件，请使用以下步骤。

要安装和访问 Hierarchical FlexGrid 控件

1. 在“工程”菜单中，选择“部件”。出现“部件”对话框。
2. 在“控件”选项卡中，选择“Microsoft Hierarchical FlexGrid Control 6.0”，然后单击“确定”。MSHFlexGrid 控件被添加到 Visual Basic 工具箱中。

3. 在 Visual Basic 工具箱中，单击 MSHFlexGrid 控件，然后将其拖到一个 Visual Basic 窗体上。

-或者-

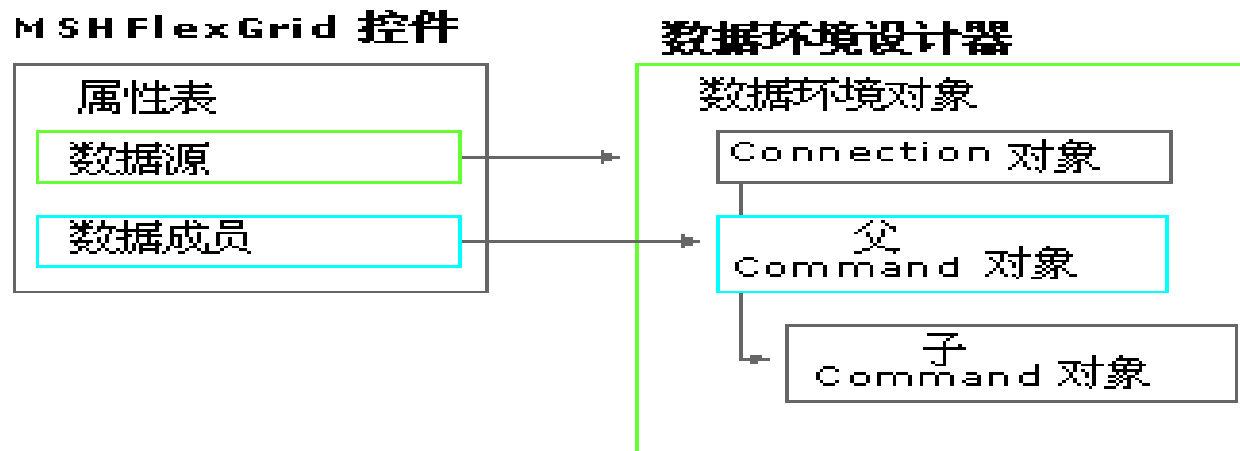
在 Visual Basic 工具箱上，双击 MSHFlexGrid 控件，将其添加到窗体上。

将数据绑定到 Hierarchical FlexGrid

在开始使用它的功能之前，必须先将数据绑定到 Hierarchical FlexGrid。要将数据绑定到控件，可以使用 Visual Basic 新的 Data Binding Manager，或者通过编程实现。

在将 Hierarchical FlexGrid 绑定到数据源之后，Hierarchical FlexGrid 在设计时屏幕显示是一个空白列和一个空白行。字段和带区信息不是自动提取的（要获得此类信息，请参阅取得结构信息）。如果 Hierarchical FlexGrid 在没有字段和带区信息的情况下运行，那么在显示数据的时候将使用缺省的属性设置。就是说，如果 Hierarchical FlexGrid 被绑定到一个分层结构的 Command，那么显示出的数据带区将是水平排列的，每个带区中包含一列，分别对应于 Recordset 中的每一个字段。

绑定到数据源的 Hierarchical FlexGrid



使用 Visual Basic Data Binding Manager 将数据绑定到 Hierarchical FlexGrid

本节说明如何使用 Visual Basic Data Binding Manager 将数据绑定到 Hierarchical FlexGrid。Data Binding Manager 提供了一种便于进行数据绑定的用户界面。

使用 Visual Basic Data Binding Manager 设置 DataSource

1. 为 Hierarchical FlexGrid 创建数据源。

数据源可以是 DataEnvironment 对象或者 ActiveX Data Object (ADO) Data Control，或者是一种新的 Visual Basic 功能。在本例中，将数据源创建为 DataEnvironment 对象。

2. 在 Visual Basic 工具箱上，单击 MSHFlexGrid 控件，然后将其拖到一个 Visual Basic 窗体上。
-或者 -
在 Visual Basic 工具箱上，双击 MSHFlexGrid 控件将其拖到一个 Visual Basic 窗体上。
3. 在 Visual Basic “属性”窗口中，将 DataSource 属性设置为包含了希望被绑定到 Hierarchical FlexGrid 的 Command 对象的 DataEnvironment 对象。
警告 如果 DataSource 被重新设置，Hierarchical FlexGrid 单元格中的所有用户定义的、修改过的数据都将被丢失。
4. 在 Visual Basic “属性”窗口中，将 DataMember 属性设置为 DataEnvironment 中的一个 Command 对象。如果希望在 Hierarchical FlexGrid 中查看分层结构的数据，那么必须指定 Command 分层结构中最顶部的父 Command 对象作为 DataMember。
5. 要查看 Hierarchical FlexGrid 中的数据，请在“运行”菜单中选择“开始”。
-或者 -
按下 F5 键。

以编程方式将数据绑定到 Hierarchical FlexGrid

本节描述如何以编程方式将数据绑定到 Hierarchical FlexGrid。

要以编程方式设置 DataSource

1. 在 Visual Basic 工具箱中，双击 MSHFlexGrid 控件，将其放在一个 Visual Basic 窗体上。
2. 右击该 Hierarchical FlexGrid，然后从快捷菜单中选择“查看代码”。出现“代码编辑器”窗口。
3. 在 Form_Load 事件中，添加代码来创建一个 ADO Recordset 并将其赋予 Hierarchical FlexGrid。该代码将在下面的分步骤中提供。

注意 要以编程方式设置数据源，工程中必须要有指向 Microsoft ActiveX Data Objects 的引用：在“工程”菜单中，选择“引用”，然后选择“Microsoft ActiveX Data Objects 2.0 Library”。

- 插入下列代码，创建一个 ADO Connection 和 Recordset，请根据实际情况替换其中的注释（例如，将 <myDataSource> 替换为数据源的实际名称）：

```
DIM Cn As New Connection, Rs As New Recordset
```

```
' 你需要将 <myDataSource> 替换为你的系统  
' 中的一个有效的 DSN。
```

```
Cn.ConnectionString = "DSN=<myDataSource>"
```

```
' 使用下列代码用于 SHAPE Commands
```

```
Cn.Provider = "MSDataShape"
```

```
Cn.CursorLocation = adUseNone
```

' 第二种办法，对于 SQL Commands 可使用下列代码

```
Cn.CursorLocation = adUseNone
```

```
Cn.Open
```

' 需要为上面创建的 Connection 的 Recordsets 指定

' 有效的数据源

```
Rs.Source = "<valid SQL SELECT command>"
```

' 现在将 Command 与 Connection 关联

' 起来并执行它们。

```
Set Rs.ActiveConnection = Cn
```

```
Rs.Open
```

- 插入下面的代码，将 Rs 中打开的 Recordset 赋予 Hierarchical FlexGrid:

```
Set MSHFlexGrid1.DataSource = Rs
```

4. 要查看 Hierarchical FlexGrid 中的数据，请在“运行”菜单中选择“开始”。

-或者-

按下 F5 键。

Hierarchical FlexGrid 与分层结构的 Recordset

将 Hierarchical FlexGrid 和一个分层结构的 Recordset 结合起来可以查看关系类型的信息。在将这些信息显示给用户的时候，可以确保原始数据的安全，使之不被用户修改；也可以通过将文本框添加到窗体 defForm 来增加 Hierarchical FlexGrid 的单元格编辑功能。当 Hierarchical FlexGrid 被绑定到分层结构的 Recordsets 上时，可以使用带区来显示分组的和相关的 Recordsets。

注意 在执行本步骤之前，必须先将数据绑定到 Hierarchical FlexGrid。为此，请参阅将数据绑定到分层结构的 FlexGrid。

要使用带区在 Hierarchical FlexGrid 中显示分层结构的 Recordsets

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“通用”选项卡中，设置“带区显示”。关于每一种带区显示的描述，请参阅带区的格式设置。
3. 在“带区”选项卡中，从“带区”框中选择一个带区。这里列出的可用带区是根据 Command 分层结构中的 Recordsets 确定的。对于每一个带区，生成 Recordset 的 Command 的名称被显示在括号中。
4. 在必要的时候修改每个带区的属性。与此有关的详细信息，请参阅带区的格式设置。
5. 单击“确定”，将带区属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

带区的格式设置

在 Hierarchical FlexGrid 中，可以利用带区格式设置功能来指定某个 Recordset 的显示选项。ADO 分层结构 Recordset 中的每一个 Recordset 都将被创建一个带区。例如，如果将一个 Hierarchical FlexGrid 绑定到一个包含 Customers 和 Orders 的 ADO 分层结构，那么 Hierarchical FlexGrid 在开始的时候将包含两个带区。

通过设置带区的格式，可以定制 Hierarchical FlexGrid 的屏幕外观。这样就可以突出显示多个带区中的重要信息。允许进行格式设置的带区元素包括：列标头、网格线、颜色和缩进。

如果绑定到非分层结构的 Recordset，那么控件中将只有一个带区，而且该带区被称为 Band 0。没有其他带区，因为带区以 Command 分层结构中的 Recordsets 为基础。

改变带区的布局

通过改变带区的布局，带区内的字段显示方式将得以改变。在缺省情况下，带区的诸字段将沿水平方向排列，同标准的网格一样。

水平方式的带区显示

	CustomerID	CompanyNa	ContactName	ContactTitle	Address	City
☐	ALFKI	Alfreds Futter	Maria Anders	Sales Representative	Obere Str. 5	Berlin
☐	ANATR	Ana Trujillo E	Ana Trujillo	Owner	Avda. de la	México
☐	ANTON	Antonio Mor	Antonio Mor	Owner	Mataderos :	México

垂直显示方式将导致带区在高度方向上进行扩展，以便能够在带区中容纳所有的字段。Hierarchical FlexGrid 中显示的其他带区也将被扩展，确保所有的带区的高度相同。

垂直方式的带区显示

	CustomerID	CompanyNa	ContactNam	ContactTitle	Address
[-]	ALFKI	Alfreds Futter	Maria Ander	Sales Repre	Obere Str. 5
	10643	ALFKI	6	1995-09-25	1995-10-23
	10692	ALFKI	4	1995-11-03	1995-12-01
	10702	ALFKI	4	1995-11-13	1995-12-25
	10835	ALFKI	1	1996-02-15	1996-03-14
	10952	ALFKI	1	1996-04-15	1996-05-27
	11011	ALFKI	3	1996-05-09	1996-06-06
[-]	ANATR	Ana Trujillo &	Ana Trujillo	Owner	Avda. de la
	10308	ANATR	7	1994-10-19	1994-11-16
	10625	ANATR	3	1995-09-08	1995-10-06
	10759	ANATR	3	1995-12-29	1996-01-26

要设置带区的布局

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“通用”选项卡中设置“带区显示”。
3. 单击“确定”，将带区的显示属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

显示列标头

如果带区以水平方式显示，那么可以为 Hierarchical FlexGrid 设置标头。标

头直接显示在带区之上，并且为 Hierarchical FlexGrid 中的每一个带区分别显示一次。如果在 Hierarchical FlexGrid 的顶部和左边只需要为每个带区显示一组标头，请使用固定单元格取代标头。关于固定单元格的信息，请参阅“自定义固定外观”（在主题自定义 Hierarchical FlexGrid 区域中）。

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“实现”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“带区”选项卡中，选择“列标头”，然后在“TextStyleHeader”列表框中选择列标头样式。
3. 单击“确定”，将列标头属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

水平的列标头

	CustomerID	CompanyNa	ContactNam	ContactTitle	Address
	ALFKI	Alfreds Futter	Maria Ander	Sales Repre	Obere Str. 5
	ANATR	Ana Trujillo E	Ana Trujillo	Owner	Avda. de la

改变列的顺序

Hierarchical FlexGrid 带区中的各列的顺序是可以改变的。

要改变带区内的列顺序

1. 选定需要移动的列。
2. 使用上下箭头键将列移动到带区内的新位置上。

修改颜色和网格线

设置带区的颜色和网格线信息的方法有两种：全体方式和个别方式。修改颜色和网格线的目的通常是为了突出地显示 Hierarchical FlexGrid 内的某些重要信息，便于用户阅读数据。另外，还可以指定带区内的单元格之间是否需要显示网格线。使用下面的过程可以修改 Hierarchical FlexGrid 内所有带区的颜色和网格线。

注意 如果需要改变单个带区的颜色信息，必须使用 BackColorBand 属性以编程方式进行改变。

要改变全局的颜色和网格线

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“带区”选项卡中，从“网格线”框中选择一种样式，然后单击“应用”。对于选定的带区，该样式确定了 Hierarchical FlexGrid 中位于标准的填充文本区域之间的线条的类型。
3. 在“样式”选项卡中，从“固定网格线”中选择一种样式。然后从“非置入网格线”中选择一种样式并单击“应用”。这些样式确定了 Hierarchical FlexGrid 中位于固定的与未填充区域之间的线条的类型。
4. 在“颜色”选项卡中，为每一种网格线属性指定一种颜色。为此，请首先选择一个“颜色设置”。然后选择要修改的属性（例如 GridColor），从“调色板”中选择一种颜色，最后单击“应用”。对于需要改变的每一条 Hierarchical FlexGrid 网格线都需要重复该操作。

注意 如果使用了 Windows 缺省颜色，那么 Hierarchical FlexGrid 中显示的颜色将是在“显示控制面板”中指定的颜色。除了将 Hierarchical FlexGrid 属性的颜色改变为标准的或 Windows 缺省颜色之外，也可以创建自己的颜色定义（单击“编辑自定义颜色”，使用随之弹出的“颜色”对话框）。

5. 单击“确定”，将网格线和颜色属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

缩进带区

如果带区以垂直方式显示，那么可以使带区缩进若干列。这样就可以使用户查看带区信息的时候更加清楚。每个带区前面被缩进的列中包含了空白的、未被使用的单元格。因此，用户不能够将焦点移到这些区域中。这些单元格的格式特点是由若干缩进格式定义属性，如 GridLinesIndent 定义的。请参阅前面的“改变带区的布局”中的“垂直方式的带区显示”图片。

要缩进带区

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“带区”选项卡中，选择“带区缩进”并指定带区需要缩进的列数。
3. 单击“确定”，将带区属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

对非分层结构的 Recordset 使用带区功能

使用 Hierarchical FlexGrid 的“属性页”对话框，也可以设置非分层结构的 Recordset 带区的格式。

非分层结构的 Recordset 中只包含一个带区，即 Band 0。没有其他的带区，因为带区是以 Command 分层结构中的 Recordsets 为基础的。

使用带区的扩展和收缩功能

扩展和收缩功能有助于方便地查看 Recordset 的组织情况，便于滚动查看 Hierarchical FlexGrid。该功能使用户既能够查看大量的数据，也可以查看压缩形式的信息。如果一个带区是可开展的，那么该带区的左上角会显示出一个缺省的扩展 (+) 或收缩 (-) 位图。无论垂直带区还是水平带区都可以使用扩展和收缩功能。

当带区处于扩展状态的时候，出现的将是收缩 (-) 位图。在扩展状态下，带区显示出最大数量的数据。当带区处于收缩状态的时候，出现的将是收缩 (+) 位图。在扩展状态下，带区显示出最少量的数据。

在收缩状态下，带区中可能显示未被填充数据的区域。对未填充区域的格式也有专门的规定。

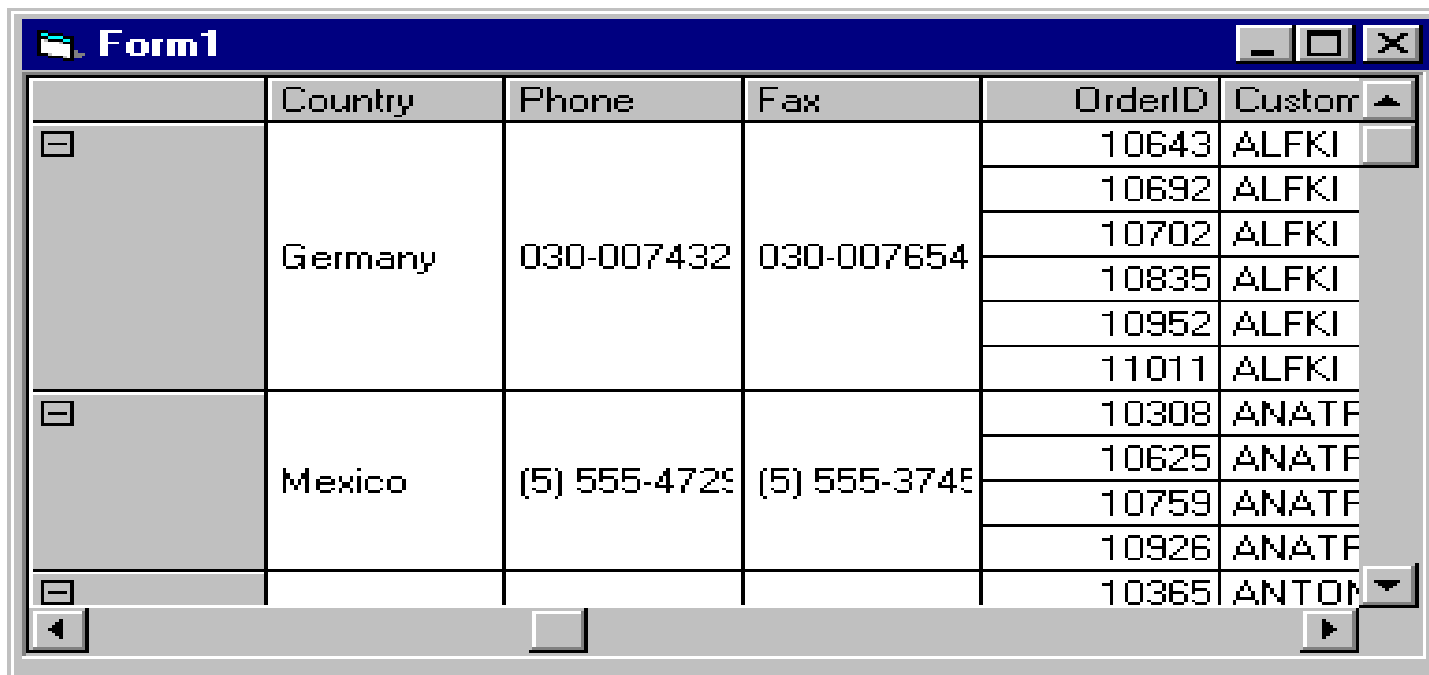
收缩状态的带区

The screenshot shows a Windows application window titled "Form1". Inside the window is a data grid with the following columns: Country, Phone, Fax, OrderID, CustomerID, and Emj. The grid contains 12 rows of data, each with a "+" icon in the first column. The data is as follows:

	Country	Phone	Fax	OrderID	CustomerID	Emj
+	Germany	030-007432	030-007654			
+	Mexico	(5) 555-4729	(5) 555-3745			
+	Mexico	(5) 555-3932				
+	UK	(171) 555-77	(171) 555-67			
+	Sweden	0921-12 34	0921-12 34			
+	Germany	0621-08460	0621-08924			
+	France	88.60.15.31	88.60.15.32			
+	Spain	(91) 555 22	(91) 555 91			
+	France	91.24.45.40	91.24.45.41			
+	Canada	(604) 555-47	(604) 555-37			
+	UK	(171) 555-12				

当一个带区处于收缩状态时，它的任何一个记录都不会显示出来，该带区中的列也不会显示出来。当用户将带区展开时，这些列将显示出来。展开状态的带区具有统一的高度。在必要的情况下，较小的带区将被扩大以便与较大带区保持一致的高度。

扩展状态的带区



The screenshot shows a Windows application window titled "Form1" containing a hierarchical table. The table has columns: Country, Phone, Fax, OrderID, and Custom. It is organized into three bands. The first band, labeled "Germany", is expanded and contains six rows of data. The second band, labeled "Mexico", is also expanded and contains four rows of data. The third band is collapsed. Each band has a small square icon in its header cell to toggle its state. The table is scrollable, with horizontal and vertical scroll bars visible at the bottom.

	Country	Phone	Fax	OrderID	Custom
[-]	Germany	030-007432	030-007654	10643	ALFKI
				10692	ALFKI
				10702	ALFKI
				10835	ALFKI
				10952	ALFKI
				11011	ALFKI
[-]	Mexico	(5) 555-4729	(5) 555-3745	10308	ANATP
				10625	ANATP
				10759	ANATP
				10926	ANATP
[-]				10365	ANTON

带区的扩展和收缩

本节说明如何为 Hierarchical FlexGrid 中的带区添加扩展和收缩功能。它还说明如何使用扩展和收缩功能。

要在带区中添加扩展与收缩功能

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。

2. 在“带区”选项卡中，选择“可扩充带区”。其次，从“带区”列表中选择要扩展的带区。

注意 要使一个带区成为可扩展的，它至少要有有一个子带区。因此，如果只有 Band 0，那么“可扩充带区”复选框将是不可使用的。

3. 单击“确定”，将带区属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

要使用扩展和收缩功能

1. 在通过“属性页”对话框将带区功能添加到 Hierarchical FlexGrid 之后，请从“运行”菜单中选择“启动”。

-或者-

按下 F5 键。

2. 单击带区左上角的扩展 (+) 图形，查看最大化的（扩展的）Recordset 信息。

3. 单击带区左上角的收缩 (-) 图形，查看最小化的（收缩的）Recordset 信息。

注意 如果被扩展的带区中包含一个被收缩的子带区，那么在将子带区展开之前它将保持收缩状态。

获取结构信息

Hierarchical FlexGrid 的结构信息中包含了与带区和列的设置有关的详细信息。在缺省设置的情况下，每个带区中的列顺序与对应的 ADO Recordset 中

的顺序相同。在获取该信息之后，可以使用它们来控制数据如何显示在 Hierarchical FlexGrid 中。

要获取与带区和字段信息有关的信息

- 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“检索结构”。
-或者-

访问 Hierarchical FlexGrid 的“属性页”对话框并选择“带区”选项卡。

该选项卡中包含了 Hierarchical FlexGrid 的结构信息。

在获取了数据的结构之后，信息与 Hierarchical FlexGrid 保存在一起。在获取了结构信息之后，设计时 Hierarchical FlexGrid 中将显示出每个带区和字段更新之后的信息。

注意 如果 DataSource 的结构发生了变化，那么只有重新提取结构才能使这些变化在 Hierarchical FlexGrid 中反映出来。

清除带区与列信息

如果清除了带区与列信息，它们将恢复为缺省的设置。

要清除带区与字段信息

- 右击 Hierarchical FlexGrid，然后选择从快捷菜单中选择“清除结构”。

注意 如果对 Hierarchical FlexGrid 的带区或列属性进行了修改，将会出现一条警告信息。这时，可以单击“确定”，将 Hierarchical FlexGrid 重新设置为缺省状态，取消所有的用户定义设置值。

定制 Hierarchical FlexGrid 中的各个区域

Hierarchical FlexGrid 中包含多个可以定制的区域。要定制这些区域，可以使用 Hierarchical FlexGrid 的“属性页”对话框，也可以使用“代码编辑器”窗口，以编程方式实现。对这些区域的定制将有助于提高 Hierarchical FlexGrid 界面的易读性和实用性。

这些 Hierarchical FlexGrid 区域包括：

- 标准

Hierarchical FlexGrid 的标准区域是那些包含数据绑定信息的单元格。

- 带区

带区区域包含了 Hierarchical FlexGrid 中每个 Recordset 的显示信息。与此有关的详细说明，请参阅带区的格式设置。

- 固定的

固定区域中包含固定的或者静态的行与列。

- 标头

列标头区域确定与被绑定到 Hierarchical FlexGrid 的 Recordset 有关的信息。在使用时，Hierarchical FlexGrid 中的每个带区都将重复这些列标头。与此有关的详细信息，请参阅带区的格式设置。

- 缩进

在以垂直方式显示带区时，Hierarchical FlexGrid 的缩进区域就是将一个数据带区缩进若干列的区域。每个带区前面的缩进列中包含空白的、未使用的

单元格。与此有关的详细说明，请参阅带区的格式设置。

- 未填充的

Hierarchical FlexGrid 的未填充区域包括 Hierarchical FlexGrid 工作区域的右边和下边的单元格。该区域是空白的，包含未被使用的单元格。

定制标准区域的显示方式

Hierarchical FlexGrid 标准区域的显示方式是可以定制的，这样提供给用户的信息将更加清晰、便于访问。标准区域中包含被绑定到数据的单元格。

要定制 Hierarchical FlexGrid 标准区域的显示方式

1. 单击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“通用”选项卡上，设置下列标准属性：

属性	描述
Rows	Hierarchical FlexGrid 中的行数
Columns	Hierarchical FlexGrid 中的列数
AllowBigSelection	允许用户通过单击列或行标头来选择整个列或者行。
MousePointer	确定鼠标指针的类型
注意 如果要使用定制的鼠标指针，必须在“图片”选项卡中为“MouseIcon”属性指定一个定制的图标	

续表

FillStyle	确定文本和单元格的格式信息是应用于单个单元格还是被选定的单元格
SelectionMode	确定 Hierarchical FlexGrid 允许用户如何进行选择
AllowUserResizing	确定是否允许用户使用鼠标重新设置行与列的尺寸
ScrollBars	确定 Hierarchical FlexGrid 是否有滚动条，如果有的话，它还说明是滚动条是水平的、垂直的，还是二者兼而有之
Highlight	确定单元格应该在什么时候被突出显示
FocusRect	定义 Hierarchical FlexGrid 在当前单元格附近绘制的焦点矩形的类型

3. 在 Style 选项卡上，设置下列标准属性：

属性	描述
MergeCells	具有相同内容的各个单元格是否被归类为跨越多个单元格或行的一个单元格
RowHeightMin	整个控件的最小行高，以 Twip 为单位
PictureType	显示的图片为彩色的还是单色的
WordWrap	Hierarchical FlexGrid 是否允许单元格内的文本回绕或者有多行

4. 在 Font 选项卡上，设置下列标准属性：

属 性	描 述
Font	字 体 风 格 的 名 称
Size	字 体 的 大 小 ， 以 磅 为 单 位
Effects	文 本 的 效 果 ： 下 划 线 或 删 除 线 ， 或 者 文 本 的 样 式 ： 粗 体 或 斜 体 。 最 多 可 以 选 择 四 个 “ 效 果 ” 复 选 框
Sample Text	字 体 选 择 的 效 果 预 览

5. 单击“确定”，将标准属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

定制固定区域的显示方式

可以创建固定显示在 Hierarchical FlexGrid 中的行或者列。为了便于用户阅读，应该使用一个固定的区域来显示一个标头集合，而不是在每个带区中使用重复的列标头。该区域将是静态的，它标识了下面的行或者右边的列。
要定制 Hierarchical FlexGrid 的固定区域的显示方式

1. 单击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“通用”选项卡上，指定“固定行”和“固定列”文本框中各自的行数和列数，然后单击“应用”。
3. 在“样式”选项卡上，从“固定文本样式”框中为固定文本选择显示风格，然后单击“应用”。
4. 在“字体”选项卡中，设置下列标准属性：

属 性	描 述
Font	字 体 风 格 的 名 称
Size	字 体 的 大 小 ， 以 磅 为 单 位
Effects	文 本 的 效 果 ： 下 划 线 或 删 除 线 ， 或 者 文 本 的 样 式 ： 粗 体 或 斜 体 。 最 多 可 以 选 择 四 个 “ 效 果 ” 复 选 框
Sample Text	字 体 选 择 的 效 果 预 览

5. 从 “ 颜 色 ” 选 项 卡 中 ， 为 每 一 个 固 定 属 性 指 定 一 个 颜 色 。 为 此 ， 首 先 需 要 选 择 一 个 “ 颜 色 设 置 ” 。 然 后 ， 单 击 需 要 改 变 的 属 性 （ 例 如 BackColorFixed）， 单 击 “ 调 色 板 ” 中 的 一 种 颜 色 ， 最 后 单 击 “ 应 用 ” 。 对 需 要 改 变 的 每 一 个 固 定 区 域 重 复 上 述 操 作 。

注 意 如 果 使 用 了 “ Windows 缺 省 ” 颜 色 ， 那 么 Hierarchical FlexGrid 中 显 示 的 颜 色 将 是 在 “ 显 示 控 制 面 板 ” 中 指 定 的 颜 色 。 除 了 将 Hierarchical FlexGrid 属 性 的 颜 色 改 变 为 标 准 的 或 Windows 缺 省 颜 色 之 外 ， 也 可 以 创 建 自 己 的 颜 色 定 义 （ 单 击 “ Edit Custom Color”， 使 用 随 之 弹 出 的 “ 颜 色 ” 对 话 框 。 ） 关 于 每 个 属 性 的 有 关 信 息 ， 请 参 阅 Hierarchical FlexGrid 的 属 性 主 题 。

6. 单 击 “ 确 定 ” ， 将 固 定 属 性 应 用 于 Hierarchical FlexGrid 并 关 闭 “ 属 性 页 ” 对 话 框 。

定制标头的显示方式

在 Hierarchical FlexGrid 中 ， 可 以 改 变 带 区 标 头 的 格 式 和 显 示 属 性 。

要定制 Hierarchical FlexGrid 的标头显示方式

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。出现 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“带区”选项卡中，从“文本样式标头”框中选择一种标头样式，然后选择“ColumnHeaders”。

注意 为了防止 Hierarchical FlexGrid 显示重复的标头，必须在“通用”选项卡中将 Fixed Row 文本框设置为 0。此外，如果希望在 Hierarchical FlexGrid 的顶部只显示带区的一组标头，那么必须使用固定行，而不是列标头。

3. 在“列标题”和“列名称”列表框中，指定需要显示的各列。在缺省的情况下，该列表中包含了 Recordset 的所有字段以及它们的字段名。如果需要停止显示某个 Field 对象，或者希望改变其名称，请使用“列标题”列表。如果需要取消某个字段，请单击对应的选择符号。如果希望改变字段的名称，请单击名称将其选定，然后再次单击它，进入编辑模式进行重命名。
4. 单击“确定”，将带区属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

定制未填充数据的区域

在 Hierarchical FlexGrid 中，可以改变带未填充数据区域的格式和显示属性。这种未填充区域是空白的，不包含任何数据。

要定制 Hierarchical FlexGrid 的未填充数据区域的显示方式

1. 右击 Hierarchical FlexGrid，然后从快捷菜单中选择“属性”。显示出 Hierarchical FlexGrid 的“属性页”对话框。
2. 在“样式”选项卡上，改变“非置入网格线”属性。
3. 在“颜色”选项卡上，为每一个固定属性指定一个颜色。为此，首先需要选择一个“颜色设置”。然后，单击需要改变的属性（例如 BackColorUnpopulated），单击“调色板”中的一种颜色，最后单击“应用”。对需要改变的每一个未填充区域重复上述操作。

注意 如果使用了“Windows 缺省”颜色，那么 Hierarchical FlexGrid 中显示的颜色将是在“显示控制面板”中指定的颜色。除了将 Hierarchical FlexGrid 属性的颜色改变为标准的或 Windows 缺省颜色之外，也可以创建自己的颜色定义（单击“Edit Custom Color”，使用随之弹出的“颜色”对话框）。

4. 单击“确定”，将固定属性应用于 Hierarchical FlexGrid 并关闭“属性页”对话框。

将 Hierarchical FlexGrid 绑定到一个 Data Environment 上

在使用 Hierarchical FlexGrid 中的带区之前，必须先将 FlexGrid 绑定到一个 Data Environment（数据环境）上，该数据环境中包含了一个顶层的父 Command 对象。

要将 Hierarchical FlexGrid 控件绑定到一个 Data Environment 上

1. 将一个 MSHFlexGrid 控件拖到 Visual Basic 窗体上。
2. 在 Visual Basic Properties 窗口中，将 DataSource 属性设置为包含了希望被绑定到 Hierarchical FlexGrid 控件的 Command 对象的 DataEnvironment 对象。例如 DataEnvironment1。
3. 在 Visual Basic Properties 窗口中，将 DataMember 属性设置为 DataEnvironment 中的一个顶层的父 Command 对象。于是数据被绑定到 Hierarchical FlexGrid 控件上。
4. 如果要查看 Hierarchical FlexGrid 中的数据，请从“运行”菜单中选择“启动”，或者按 F5 键。获取到的结构将出现在 Hierarchical FlexGrid 中。

对 Hierarchical FlexGrid 中的数据进行合并与排序

在本节的应用方案中介绍了如何对 Hierarchical FlexGrid 中的数据进行排序和合并。在大多数情况下，可以通过将数据库下载到 Hierarchical FlexGrid 的方式来访问数据。不过，在这里示例数据通过“代码编辑器”窗口填写到 Hierarchical FlexGrid 的行与列中。

要建立数据显示

1. 设置 Hierarchical FlexGrid 的各种属性。
2. 建立数据。
3. 定义各种子程序，以计算索引值和进行排序。
4. 定义子程序，将数据（来自第二步）填入 Hierarchical FlexGrid。

5. 允许控件切换到数据组织视图。
要完成本方案，请依次执行本节中的各个步骤。

设置控件的各种属性

使用下表中的设置值，设置列数与行数、字体信息，并为 Hierarchical FlexGrid 创建列标头。

MSHFlexGrid 控件

属 性	设 置 值
Name	Fg1
Cols	4
Rows	20
MergeCells	2 – Restrict Rows
FormatString	<Region <Product <Employees >Sales
FontName	Arial

数据的排序与合并

使用下面的过程完成 Hierarchical FlexGrid 中的数据排序与合并。
排序与合并数据

1. 创建一个数组用来存储数据。为此，首先在“代码编辑器”窗口的 Form_Load 事件中加入下面的子程序：

```
Sub Form_Load ()
```



```
Dim I As Integer
```

```
'创建数组。
```

```
For i = Fg1.FixedRows To Fg1.Rows - 1
```

```
    '地区。
```

```
    Fg1.TextArray(fgi(i, 0)) = RandomString(0)
```

```
    '产品。
```

```
    Fg1.TextArray(fgi(i, 1)) = RandomString(1)
```

```
    '雇员。
```

```
    Fg1.TextArray(fgi(i, 2)) = RandomString(2)
```

```
    Fg1.TextArray(fgi(i, 3)) = _
```

```
    Format(Rnd * 10000, "#.00")
```

```
Next
```

```
'设置合并。
```

```
Fg1.MergeCol(0) = True
```

```
Fg1.MergeCol(1) = True
```

```
Fg1.MergeCol(2) = True
```

```
'排序以查看效果。
```

```
DoSort0
```

```
'Format Grid
```

```
Fg1.ColWidth(0) = 1000
```

```
Fg1.ColWidth(1) = 1000
```

```
Fg1.ColWidth(2) = 1000
```

```
Fg1.ColWidth(3) = 1000
```

```
End Sub
```

2. 计算一个索引并完成排序。要这样做，请定义一个例程来计算索引并排序。在索引时同 `TextArray` 属性一起使用来对数据排序。插入下列例程计算索引：

```
Function Fgi (r As Integer, c As Integer) As Integer
```

```
    Fgi = c + Fg1.Cols * r
```

```
End Function
```

```
Sub DoSort ()
```

```
    Fg1.Col = 0
```

```
    Fg1.ColSel = Fg1.Cols - 1
```

```
    Fg1.Sort = 1      '一般为升序。
```

```
End Sub
```

3. 将数据输入到 `Hierarchical FlexGrid`。要这样做，请定义一个例程使用示例数据充填 `Hierarchical FlexGrid`：

```
Function RandomString (kind As Integer)
```

```
    Dim s As String
```

```
    Select Case kind
```

```
        Case 0      '地区。
```

```
            Select Case (Rnd * 1000) Mod 5
```

```
Case 0: s = "1. Northwest"  
Case 1: s = "2. Southwest"  
Case 2: s = "3. Midwest"  
Case 3: s = "4. East"  
Case Else: s = "5. Overseas"
```

```
End Select
```

```
Case 1      '产 品 。
```

```
    Select Case (Rnd * 1000) Mod 5
```

```
        Case 0: s = "1. Chai"  
        Case 1: s = "2. Peppermint"  
        Case 2: s = "3. Chamomile"  
        Case Else: s = "4. Oolong"
```

```
    End Select
```

```
Case 2      '雇 员 。
```

```
    Select Case (Rnd * 1000) Mod 4
```

```
        Case 0: s = "Clare"  
        Case 1: s = "Tiffany"  
        Case 2: s = "Sally"  
        Case Else: s = "Lori"
```

```
    End Select
```

```
End Select
```

```
RandomString = s
```

```
End Function
```

如果在此时运行工程，则将显示窗体如下：

Region	Product	Employees	Sales
1. Northwest	1. Chai	Clare	5338.73
		Lori	7988.84
	3. Chamomile	Sally	5924.58
	4. Oolong	Tiffany	3262.06
		Sally	9193.77
2. Southwest	2. Peppermin	Sally	3640.19
	4. Oolong	Clare	157.04
		Sally	2613.68
		Clare	2895.63
			4013.74
3. Midwest	1. Chai	Sally	7607.24
4. East	1. Chai	Lori	6465.87
		Sally	8818.58

接着，您需要允许用户重新组织数据。也就是说，您必须允许 Hierarchical FlexGrid 切换数据组织视图。

4. 通过添加下列例程重新组织数据，该例程将列拖动到新的位置。当用户按下鼠标按钮来触发 `MouseDown` 事件时，该例程用 `Tag` 属性保存列数目：

```
Sub Fg1_MouseDown (Button As Integer, _
```

```
Shift As Integer, X As Single, Y As Single)
```

```

Fg1.Tag = ""
If Fg1.MouseRow <> 0 Then Exit Sub
Fg1.Tag = Str(Fg1.MouseCol)
MousePointer = vbSizeWE
End Sub

```

5. 添加下列例程以重新调整列，当用户释放鼠标按钮触发 `MouseUp` 事件时，该例程重新调整列并将数据排序：

```

Sub Fg1_MouseUp (Button As Integer, Shift As _
Integer, X As Single, Y As Single)
    MousePointer = vbDefault
    If Fg1.Tag = "" Then Exit Sub
    Fg1.Redraw = False
    Fg1.ColPosition(Val(Fg1.Tag)) = Fg1.MouseCol
    DoSort
    Fg1.Redraw = True
End Sub

```

一旦完成了该方案中的过程，则当您在运行时无论何时将列拖动到新位置时，数据都重新组织。例如，如果您将 `Employee` 列拖动到左边，它将会以如下方式出现：

Employees	Region	Product	Sales	
Clare	1. Northwest	1. Chai	5338.73	
	2. Southwest	4. Oolong	157.04	
			2895.63	
			4013.74	
	5. Overseas	2. Peppermin	3820.11	
Lori	4. East	1. Chai	6465.87	
	5. Overseas	2. Peppermin	2268.66	
	1. Northwest	1. Chai	7988.84	
	5. Overseas	4. Oolong	6478.21	
Sally	4. East	1. Chai	9619.53	
		4. Oolong	5833.59	
	2. Southwest	4. Oolong	2613.68	
	5. Overseas	4. Oolong	8246.82	

在 Hierarchical FlexGrid 电子表格中编辑单元

在该方案中，您可以在 Hierarchical FlexGrid 电子表格中编辑单元。该方案举例说明了 Hierarchical FlexGrid 的控件事件和容器的一些能力，并说明了如何使用该控件创建一个电子表格，可以在该电子表格中的单元中使用标准 Visual Basic 控件进行编辑。

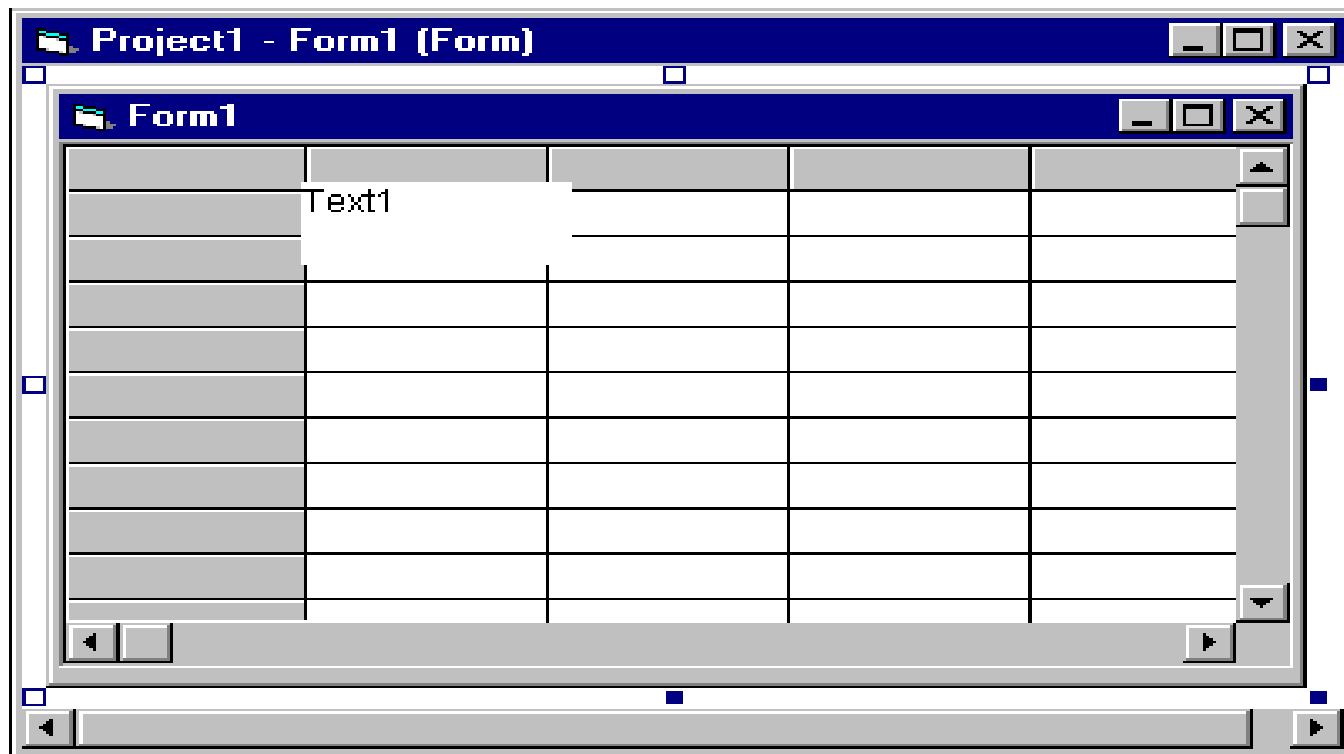
注意 示例应用程序 (Flex.vbp) 举例说明了漫游和选择单元范围所必需的能力。

要创建该数据显示

1. 创建 Hierarchical FlexGrid 和 TextBox 控件。
 2. 设置这些控件的属性。
 3. 对 Hierarchical FlexGrid 添加行和列标头。
 4. 添加单元内编辑到 Hierarchical FlexGrid。
 5. 添加功能到 TextBox 控件（“编辑”框），以便更新数据。
 6. 将数据从文本框复制到 Hierarchical FlexGrid。
- 要完成该方案，请按照该部分中的步骤顺序执行。

创建控件

添加一个 Hierarchical FlexGrid 到工程，然后添加一个 TextBox 控件以创建一个父子关系，如下所示：



设置控件的属性

如下所示设置 Hierarchical FlexGrid 和 TextBox 控件的属性：

MSHFlexGrid 控件

属性	设置值
Name	Fg2
Cols	6

续表

Rows	20
FillStyle	1 - Repeat
FocusRect	2 - Heavy
FontName	Arial
FontSize	9

TextBox 控件

属 性	设 置 值
Name	TxtEdit
FontName	Arial
FontSize	9
BorderSize	0 – None
Visible	False

在电子表格中编辑单元

使用下列过程在 Hierarchical FlexGrid 中编辑单元。

要在电子表格中编辑单元

1. 通过将下列代码添加到“代码编辑器”窗口中的 Form_Load 过程来修改您的 Hierarchical FlexGrid，使其代表一个电子表格：

```
Sub Form_Load ()  
    Dim i As Integer
```

'使第一列较窄。

```
Fg2.ColWidth(0) = Fg2.ColWidth(0) / 2
```

```
Fg2.ColAlignment(0) = 1    '放置在正中。
```

'标记行和列。

```
For i = Fg2.FixedRows To Fg2.Rows - 1
```

```
    Fg2.TextArray(fgi(i, 0)) = i
```

```
Next
```

```
For i = Fg2.FixedCols To Fg2.Cols - 1
```

```
    Fg2.TextArray(fgi(0, i)) = i
```

```
Next
```

'初始化编辑框（于是，现在可加载它）。

```
txtEdit = ""
```

```
End Sub
```

2. 如下创建计算 TextArray 属性索引值的函数：

```
Function Fgi (r As Integer, c As Integer) As Integer
```

```
    Fgi = c + Fg2.Cols * r
```

```
End Function
```

3. 将下列代码添加到 Hierarchical FlexGrid 的 KeyPress 和 DblClick 事件：

```
Sub Fg2_KeyPress (KeyAscii As Integer)
    MSHFlexGridEdit Fg2, txtEdit, KeyAscii
End Sub
```

```
Sub Fg2_DblClick ()
    MSHFlexGridEdit Fg2, txtEdit, 32 '模拟一个空格。
End Sub
```

4. 添加下列例程以初始化文本框，并将焦点从 Hierarchical FlexGrid 传递到 TextBox 控件：

```
Sub MSHFlexGridEdit (MSHFlexGrid As Control, _
    Edt As Control, KeyAscii As Integer)
```

 '使用已输入的字符。

```
    Select Case keyascii
```

 '空格表示编辑当前的文本。

```
        Case 0 To 32
```

```
            Edt = MSHFlexGrid
```

```
            Edt.SelStart = 1000
```

 '其它所有字符表示取代当前的文本。

```
        Case Else
```

```
        Edt = Chr(keyascii)
        Edt.SelStart = 1
    End Select
```

在合适的位置显示 Edt。

```
Edt.Move MSHFlexGrid.Left + MSHFlexGrid.CellLeft, _
        MSHFlexGrid.Top + MSHFlexGrid.CellTop, _
        MSHFlexGrid.CellWidth - 8, _
        MSHFlexGrid.CellHeight - 8
Edt.Visible = True
```

启动工作。

```
Edt.SetFocus
```

```
End Sub
```

5. 通过将下列例程添加到 TextBox 的 KeyPress 和 DblClick 事件中来添加更新数据功能性：

```
Sub txtEdit_KeyPress (KeyAscii As Integer)
    '删除回车符，以消除嘟嘟声。
    If KeyAscii = Asc(vbCr) Then KeyAscii = 0
End Sub
```

```
Sub txtEdit_KeyDown (KeyCode As Integer, _
```

Shift As Integer)

 EditKeyCode Fg2, txtEdit, KeyCode, Shift

End Sub

Sub EditKeyCode (M SHFlexGrid As Control, Edt As _
Control, KeyCode As Integer, Shift As Integer)

 '标准编辑控件处理。

 Select Case KeyCode

 Case 27 'ESC:隐藏焦点并将其返回 M SHFlexGrid。

 Edt.Visible = False

 M SHFlexGrid.SetFocus

 Case 13 'ENTER 将焦点返回 M SHFlexGrid。

 M SHFlexGrid.SetFocus

 Case 38 '向上。

 M SHFlexGrid.SetFocus

 DoEvents

 If M SHFlexGrid.Row > M SHFlexGrid.FixedRows Then

 M SHFlexGrid.Row = M SHFlexGrid.Row - 1

```
End If
```

```
Case 40 '向下。
```

```
MSFlexGrid.SetFocus
```

```
DoEvents
```

```
If MSHFlexGrid.Row < MSHFlexGrid.Rows - 1 Then
```

```
MSHFlexGrid.Row = MSHFlexGrid.Row + 1
```

```
End If
```

```
End Select
```

```
End Sub
```

接着，当把数据输入到 `TextBox` 中时，您需要先告诉 `Hierarchical FlexGrid` 控件应该对数据做什么。当您输入数据并按下 `ENTER` 键，或用鼠标单击 `Hierarchical MSFlexGrid` 控件中的另一个单元时，焦点将返回此控件。这时 `TextBox` 中的文本被复制到活动单元中；要这样做，请继续执行下列步骤。

6. 通过将下列代码添加到 `GotFocus` 和 `LeaveCell` 事件过程中，将数据从文本框复制到 `Hierarchical FlexGrid`：

```
Sub Fg2_GotFocus ()
```

```
If txtEdit.Visible = False Then Exit Sub
```

```
Fg2 = txtEdit
```

```
txtEdit.Visible = False
```

```
End Sub
```

```

Sub Fg2_LeaveCell ()
    If txtEdit.Visible = False Then Exit Sub
    Fg2 = txtEdit
    txtEdit.Visible = False
End Sub

```

一旦方案中的过程完成，则可以在运行时将数据输入到单个的单元中，如下面的图表所示。

1	2	3	4	
Welcome to				
	FlexArray	FlexArray	FlexArray	
	FlexArray	FlexArray	FlexArray	
		Place your		
		cursor in a		
		cell and type		

使用 Hierarchical FlexGrid 显示一个带有标头的大纲

在该方案中，您可以使用 Hierarchical FlexGrid 创建一个大纲样式的显示，该显示带有可以折叠和扩展的标头。

要创建该数据显示，请按照以下步骤执行：

- 1. 设置 Hierarchical FlexGrid 的属性。
- 2. 创建数据。
- 3. 添加折叠和扩展功能性。

要完成该方案，请按照该部分中的顺序执行下列步骤。

设置控件的属性

使用 FormatString 属性设置四个列标头。将第一列设置为同其他列一样窄和空（象一个电子表格），以包含数据（包括每个标头间的空间）。此外，还要按照以下设置：

MSHFlexGrid 控件

属 性	设 置 值
Name	Fg3
Cols	4
Rows	2
Selection Mode	1 - By Row
FillStyle	1 – Repeat
FocusRect	0 – None

续表

GridLines	0 – None
GridLinesFixed	2 – Inset
FormatString	^ Description >Date >Amount
FontName	Arial

创建一个带有标头的大纲显示

使用下列步骤完成该方案，使用 Hierarchical FlexGrid 创建一个带有标头的大纲显示。

要创建带有标头的大纲显示，请按照以下步骤执行：

1. 创建示例数据，并设置标头、列和行属性。要这样做，请将下列代码插入“代码编辑器”窗口的 `Form_Load` 事件。这就创建了一个示例数据，设置并计算控件顶部的标头，然后设置 `Col` 和 `Row` 属性，以在运行时选择第一个单元。

```
Sub Form_Load ()  
    Dim i As Integer, tot As Integer  
    Dim t As String, s As String  
  
    '创建示例数据。  
    t = Chr(9)  
    Fg3.Rows = 1
```

```
Fg3.AddItem "*" + t + "AirFare"
s = "" + t + "SFO-JFK " + t + "9-Apr-95" + t + "750.00"
For i = 0 to 5
    Fg3.AddItem s
Next
```

```
Fg3.AddItem "*" + t + "Meals"
s = "" + t + "Flint's BBQ " + t + "25-Apr-95" _
+t + "35.00"
For i = 0 to 5
    Fg3.AddItem s
Next
```

```
Fg3.AddItem "*" + t + "Hotel"
s = "" + t + "Center Plaza" + t + "25-Apr-95" _
+t + "817.00"
For i = 0 to 5
    Fg3.AddItem s
Next
```

添加总计和格式标头项。

```

For i = Fg3.Rows - 1 To 0 Step -1
    If Fg3.TextArray(i * Fg3.Cols) = "" Then
        tot = tot + Val(Fg3.TextArray_
            (i * Fg3.Cols + 3))
    Else
        Fg3.Row = i
        Fg3.Col = 0
        Fg3.ColSel = Fg3.Cols - 1
        Fg3.CellBackColor = &HC0C0C0
        Fg3.CellFontBold = True
        Fg3.CellFontWidth = 8
        Fg3.TextArray(i * Fg3.Cols + 3) = _
            Format(tot, "0")
        tot = 0
    End If
Next
Fg3.ColSel = Fg3.Cols - 1

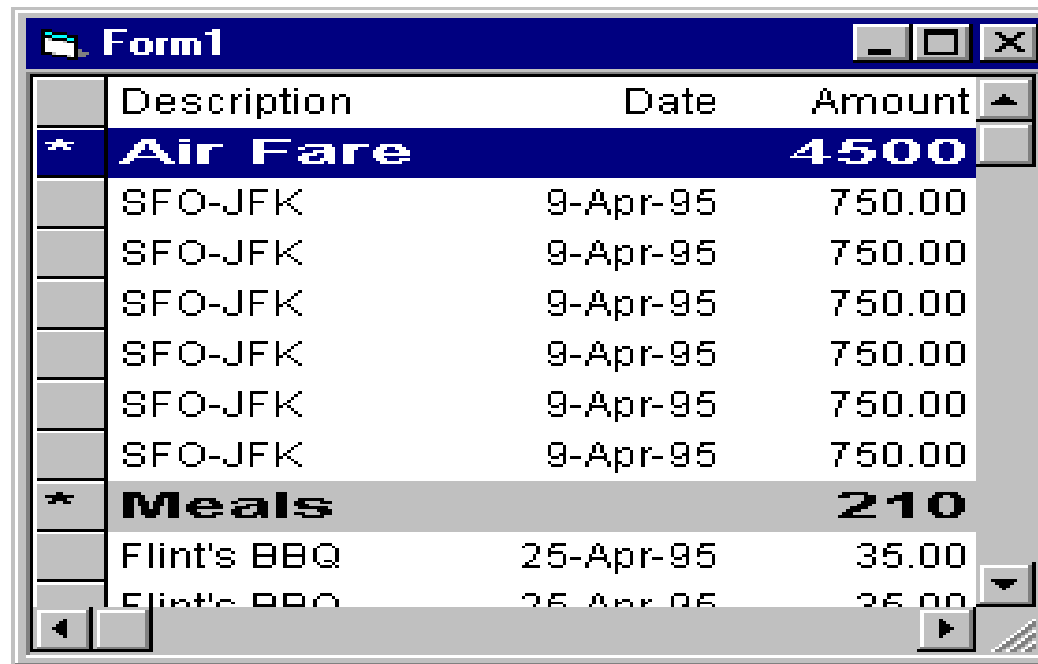
' 格式化网格
Fg3.ColWidth(0) = 300
Fg3.ColWidth(1) = 1500
Fg3.ColWidth(2) = 1000

```

```
Fg3.ColWidth(3) = 1000
```

```
End Sub
```

运行时，将行按其标头（Air Fare、Meals 和 Hotels）分成三部分，如下图所示：



	Description	Date	Amount
*	Air Fare		4500
	SFO-JFK	9-Apr-95	750.00
	SFO-JFK	9-Apr-95	750.00
	SFO-JFK	9-Apr-95	750.00
	SFO-JFK	9-Apr-95	750.00
	SFO-JFK	9-Apr-95	750.00
	SFO-JFK	9-Apr-95	750.00
*	Meals		210
	Flint's BBQ	25-Apr-95	35.00
	Flint's BBQ	25-Apr-95	35.00

2. 通过将下列代码插入到 Hierarchical FlexGrid 的 DblClick 事件过程中对行标头添加折叠和扩展功能性：

```
Sub Fg3_DblClick ()
```

```
Dim i As Integer, r As Integer
```

'忽略顶端的行。

```
r = Fg3.MouseRow
```

```
If r < 1 Then Exit Sub
```

'查找要折叠或扩充的字段。

```
While r > 0 And Fg3.TextArray(r * Fg3.Cols) = ""
```

```
    r = r - 1
```

```
Wend
```

'在第一列显示折叠/扩充符号图。

```
If Fg3.TextArray(r * Fg3.Cols) = "*" Then
```

```
    Fg3.TextArray(r * Fg3.Cols) = "+"
```

```
Else
```

```
    Fg3.TextArray(r * Fg3.Cols) = "*"
```

```
End If
```

'在当前标头下扩充项目。

```
r = r + 1
```

```
If Fg3.RowHeight(r) = 0 Then
```

```
    Do While Fg3.TextArray(r * Fg3.Cols) = ""
```

```
        Fg3.RowHeight(r) = -1' Default row height.
```

```
        r = r + 1
    If r >= Fg3.Rows Then Exit Do
Loop
```

'在当前标头下折叠项目。

Else

```
    Do While Fg3.TextArray(r * Fg3.Cols) = ""
        Fg3.RowHeight(r) = 0 '隐藏行。
        r = r + 1
    If r >= Fg3.Rows Then Exit Do
Loop
```

End If

End Sub

一旦该方案中的所有步骤都完成，您就可以在运行时通过双击第一列中的符号 "+" 或 "*" 来展开或折叠行标头，如下例所示：

	Description	Date	Amount
+	Air Fare		4500
+	Meals		210
*	Hotel		4902
	Center Plaza	25-Apr-95	817.00
	Center Plaza	25-Apr-95	817.00
	Center Plaza	25-Apr-95	817.00
	Center Plaza	25-Apr-95	817.00
	Center Plaza	25-Apr-95	817.00
	Center Plaza	25-Apr-95	817.00

注意 很容易改动此例，使之显示图像而不是 "+" 和 "*" 字符，或者在概要中添加附加的层次。

使用 HScrollBar 和 VScrollBar 控件

有了滚动条，就可在应用程序或控件中水平或垂直滚动，相当方便地巡视一长列项目或大量信息。滚动条是 Windows 95 和 Windows NT 界面上的共同元素。





图 7.23 HScrollBar 和 VScrollBar 控件

水平、垂直滚动条控件不同于 Windows 中内部的滚动条或 Visual Basic 中那些附加在文本框、列表框、组合框或 MDI 窗体上的滚动条。无论何时，只要应用程序或控件所包含的信息超过当前窗口（或者在 ScrollBars 属性被设置成 True 时的文本框和 MDI 窗体）所能显示的信息，那些滚动条就会自动出现。

在较早的 Visual Basic 版本中，通常用滚动条作为输入设备。但目前的 Windows 界面指南则建议用滑块取代滚动条作为输入设备。Visual Basic 专业版和企业版都包括 Windows 95 的滑块控件。

滚动条在 Visual Basic 中仍然有价值，因为它为那些不能自动支持滚动的应用程序和控件提供了滚动功能。关于使用滚动条的信息，请参阅“Scroll Bar 控件方案：创建可滚动的图形视口”。

Scroll Bar 控件如何工作

滚动条控件用 Scroll 和 Change 事件监视滚动框（有时用拇指替代）沿滚动条的移动。

事件	描述
Change	在滚动框移动后发生
Scroll	在移动滚动框时发生。在单击滚动箭头或滚动条时不发生

可用 `Scroll` 事件访问滚动条被拖动后的数值。在释放滚动框或单击滚动条或滚动箭头时，`Change` 事件就会发生。

Value 属性

`Value` 属性（缺省值为 0）是一个整数，它对应于滚动框在滚动条中的位置。当滚动框位置在最小值时，它将移动到滚动条的最左端位置（水平滚动条）或顶端位置（垂直滚动条）。当滚动框在最大值时，它将移动到滚动条的最右端或底端位置。同样，滚动框取中间数值时将位于滚动条的中间位置。除了可用鼠标单击改变滚动条数值外，也可将滚动框沿滚动条拖动到任意位置。结果取决于滚动框的位置，但总是在用户所设置的 `Min` 和 `Max` 属性之间。

注意 如果希望滚动条显示的信息从较大数值向较小数值变化，可将 `Min` 设置成大于 `Max` 的值。

LargeChange 和 SmallChange 属性

为了指定滚动条中的移动量，对于单击滚动条的情况可用 `LargeChange` 属性，对于单击滚动条两端箭头的情况可用 `SmallChange` 属性。滚动条的 `Value` 属性增加或减少的长度是由 `LargeChange` 和 `SmallChange` 属性设置的数值。要设置滚动框在运行时的位置，可将 `Value` 属性设为 0 到 32,767 中的某个数值（包括 0 和 32,767）。

Scroll Bar 控件方案：创建可滚动的图形视口

除了 PictureBox 控件之外，也可用水平、垂直滚动条创建可滚动的图形视口应用程序。当所包含的图形超过控件范围时，单独一个 PictureBox 控件无法实现滚动功能——因为 PictureBox 控件不能自动添加滚动条。应用程序使用两个图片框。称第一个为平稳的父 PictureBox 控件。称第二个为子 PictureBox 控件，它包含在父图片框中。子图片框中包含图形图像，可用滚动条控件在父图片框中移动子图片框。

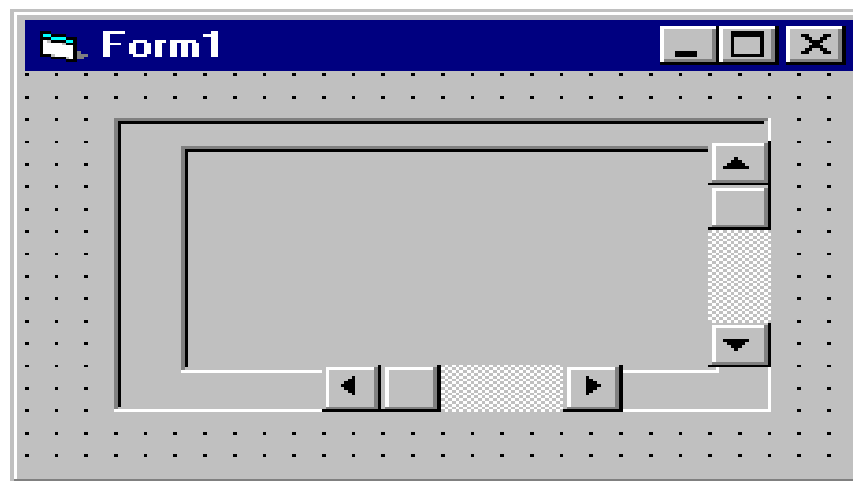


图 7.24 在设计时添加滚动条控件

如图 7.24 所示，先创建一个新工程，然后在窗体上绘制两个图片框、一个水平滚动条和一个垂直滚动条。

这里，用窗体的 `Form_Load` 事件设置比例模型，在父图片框中调整子图片框的大小，水平、垂直滚动条将定位并调整它们的大小，然后加载位图图形。将下列代码添加到窗体的 `Form_Load` 事件过程中：

```
Private Sub Form_Load()
```

```
    '设置 ScaleMode 为像素。
```

```
    Form1.ScaleMode = vbPixels
```

```
    Picture1.ScaleMode = vbPixels
```

```
    '将 AutoSize 设置为 True，以使 Picture2 的边界
```

```
    '扩展到实际的位图大小。
```

```
    Picture2.AutoSize = True
```

```
    '将每个图片框的 BorderStyle 属性设置为 None。
```

```
    Picture1.BorderStyle = 0
```

```
    Picture2.BorderStyle = 0
```

```
    '加载位图。
```

```
    Picture2.Picture = _
```

```
    LoadPicture("c:\Windows\Winlogo.bmp")
```

'初始化两个图片框的位置。

```
Picture1.Move 0, 0, ScaleWidth - VScroll1.Width, _  
ScaleHeight - HScroll1.Height  
Picture2.Move 0, 0
```

'将水平滚动条定位。

```
HScroll1.Top = Picture1.Height  
HScroll1.Left = 0  
HScroll1.Width = Picture1.Width
```

'将垂直滚动条定位。

```
VScroll1.Top = 0  
VScroll1.Left = Picture1.Width  
VScroll1.Height = Picture1.Height
```

'设置滚动条的 Max 属性。

```
HScroll1.Max = Picture2.Width - Picture1.Width  
VScroll1.Max = Picture2.Height - Picture1.Height
```

'判断子图片框是否将充满屏幕。

'若如此，则无需使用滚动条。

```
VScroll1.Visible = (Picture1.Height < _
```

```
Picture2.Height)  
HScroll1.Visible = (Picture1.Width < _  
Picture2.Width)
```

```
End Sub
```

水平和垂直滚动条的 **Change** 事件用来在父图片框中上、下、左、右移动子图片框。请将下列代码添加到两个滚动条控件的 **Change** 事件中：

```
Private Sub HScroll1_Change()  
    Picture2.Left = -HScroll1.Value  
End Sub
```

```
Private Sub VScroll1_Change()  
    Picture2.Top = -VScroll1.Value  
End Sub
```

将子图片框的 **Left** 和 **Top** 属性分别设置成水平和垂直滚动条数值的负值，这样，当上、下、左、右滚动时，图形可正确地移动。运行时，显示的图形如图 7.25 所示。

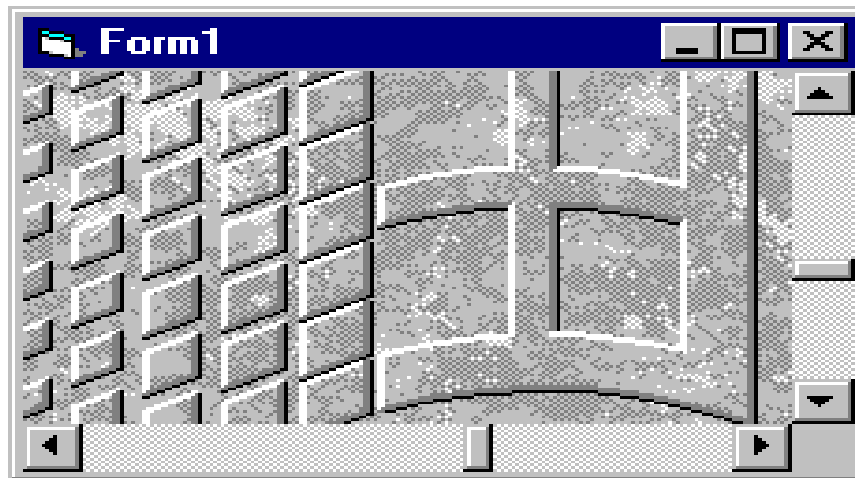


图 7.25 运行时滚动位图

运行时调整窗体大小

在上例中，窗体的初始大小限制图形的可视大小。在运行时当用户调整窗体大小时，为了调整图形视口应用程序的大小，可将下列代码添加到窗体的 `Form_Resize` 事件过程中：

```
Private Sub Form_Resize()
```

```
    '调整窗体大小时，改变 Picture1 的尺寸。
```

```
    Picture1.Height = Form1.Height
```

```
    Picture1.Width = Form1.Width
```

```
    '重新初始化图片和滚动条的位置。
```

```
Picture1.Move 0, 0, ScaleWidth - VScroll1.Width, _  
ScaleHeight - HScroll1.Height  
Picture2.Move 0, 0  
HScroll1.Top = Picture1.Height  
HScroll1.Left = 0  
HScroll1.Width = Picture1.Width  
VScroll1.Top = 0  
VScroll1.Left = Picture1.Width  
VScroll1.Height = Picture1.Height  
HScroll1.Max = Picture2.Width - Picture1.Width  
VScroll1.Max = Picture2.Height - Picture1.Width
```

检查是否需要滚动条。

```
VScroll1.Visible = (Picture1.Height < _  
Picture2.Height)  
HScroll1.Visible = (Picture1.Width < _  
Picture2.Width)
```

```
End Sub
```

使用 Image 控件

Image 控件用来显示图形。它可显示下面几种格式的图形：位图、图标、图元文件、增强型图元文件、JPEG 或 GIF 文件。



图 7.26 Image 控件

除此之外，Image 控件还响应 Click 事件，并可用 Image 控件代替命令按钮或作为工具条的项目，还可用来制作简单动画。

详细信息 关于使用 Image 控件作为命令按钮的简单示例，请参阅第三章“窗体、控件和菜单”中的“轻图形控件”。关于使用 Image 控件创建工具条的信息，请参阅第六章“创建用户界面”中的“创建 ToolBar”。关于使用 Image 控件创建简单动画的详细信息，请参阅第十二章“使用文本和图形”中的“创建简单动画”。

何时使用 Image 控件而不使用 PictureBox 控件

Image 控件使用的系统资源比 PictureBox 控件少而且重新绘图速度快，但它仅支持 PictureBox 控件的一部分属性、事件和方法。两种控件都支持相同的图片格式，但是，在 Image 控件中可以伸展图片的大小使之适合控件的大小。在 PictureBox 控件中不能这样做。

详细信息 关于使用 Image 控件节省系统资源的详细信息，请参阅第十五章“性能和兼容性设计”中的“修剪图形”。

支持的图形格式

Image 控件可显示下述标准格式的图片文件。

图 片 格 式	描 述
Bitmap	位图将图像定义为点（像素）的图案。位图的文件扩展名是 .bmp 或 .dib。也称位图为“画图类型”("paint-type") 的图形 可用多种颜色深度：包括 2、4、8、16、24 和 32 位的颜色深度，但是只有当显示设备支持位图使用的颜色深度时才能正确显示位图。例如，每像素 8 比特（256 色）的位图在每像素 4 比特（16 色）的设备上只能显示出 16 种颜色
Icon	图标是特殊类型的位图。图标的最大尺寸为 32 像素 × 32 像素，但在 Microsoft Windows 95 下，图标也可 为 16 × 16 像素大小。图标的文件扩展名为 .ico
Cursor	游标，像图标一样，实质上是位图。然而游标也包含热点，通过 X 和 Y 坐标跟踪游标位置的像素。游标的扩展文件名 .cur

续表

Metafile	图元文件将图形定义为编码的线段和图形。普通图元文件扩展名为 .wmf。增强型图元文件扩展名为 .emf。只能加载与 Microsoft Windows 兼容的图元文件。图元文件也称作“绘图类型”的图形
JPEG	JPEG 是一种支持 8 位和 24 位颜色的压缩位图格式。它是 Internet 上一种流行的文件格式
GIF	GIF 是最初由 CompuServe 开发的一种压缩位图格式。它可支持多达 256 种的颜色，是 Internet 上一种流行的文件格式

将图形加载到 Image 控件中

设计时从控件的“属性”窗口中选择 Picture 属性，或在运行时使用 Picture 属性和 LoadPicture 函数都可将图形加载到 Image 控件。

```
Set Image1.Picture = LoadPicture("c:\Windows\Winlogo.cur", vbLPLarge, vbLPColor)
```

向 Image 控件加载图片时，控件能够自动调整大小以适应控件——不管在窗体上把 Image 控件画得有多大或有多小。您可能希望使用包含大小不同和颜色深度不同的独立图像以支持多种显示设备的图标 (.icon) 和游标 (.cur) 文件。LoadPicture 函数的设置允许您从 .ico 或 .cur 文件中选择特定颜色深度和大小的图像。假如没有要求设置的匹配，LoadPicture 将加载可获得的最匹配图像。

为了清除 Image 控件中的图形，可使用不指定文件名的 LoadPicture 函数。例如：

```
Set Image1.Picture = LoadPicture
```

这样将清除 `Image` 控件中的内容，即使对于在设计时加载 `Picture` 属性中的图形亦如此。

使用剪贴板

在设计时可粘贴其它应用程序中的图形，由此即可将图形加载到 `Image` 控件中。例如，有时可能希望添加一个由 `Windows Paint` 创建的位图图像。只需把图像复制到剪贴板，选定 `Image` 控件，然后使用键盘快捷键 `CTRL+V` 或者“编辑”菜单中的“粘贴”命令。

Stretch 属性

在设计时调整 `Image` 控件大小，这时，`Stretch` 属性决定是否使图片伸缩。若将属性设置为 `True`，则将伸缩 `Picture` 属性加载的图片。如图 7.27 所示，伸缩图片（特别是位图格式的图片）可能导致图像质量的降低。然而，图元文件这种“绘图类型”的图形却非常适合伸缩。

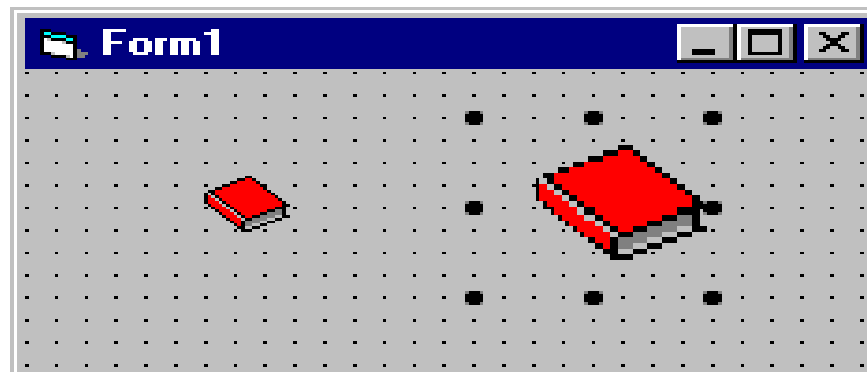


图 7.27 扩展位图图像

使用 Label 控件

Label 控件用来显示文本，但用户不能编辑这些控件。可用此控件标识窗体的对象——例如，说明单击控件时将执行何种操作——或者在运行时为响应应用程序的事件或进程而显示相应的信息。



图 7.28 Label 控件

使用标签的情况很多，而且目的也不相同。通常用标签来标注本身不具有 Caption 属性的控件。例如，可用 Label 控件为文本框、列表框、组合框

等等控件添加描述性的标签。也可用它们为窗体添加说明文字，例如向用户提供帮助信息。

还可编写代码改变 Label 控件显示的文本内容以响应运行时的事件。例如，若应用程序需要用几分钟处理某个变更，则可用标签显示处理状况消息。因为 Label 控件不接受焦点，所以被用来为其它控件创建访问键。

设置标签的标题

为了改变 Label 控件中显示的文本，可使用 Caption 属性。设计时，可从控件的“属性”窗口中选定并设置此属性。

Caption 属性的长度最长可设置成 1024 字节。

排列文本

可使用 Alignment 属性，将 Label 控件中文本的排列方式设置为 Left Justify（0，缺省），Center(1)或者 Right Justify(2)。

AutoSize 和 WordWrap 属性

缺省情况下，当输入到 Caption 属性的文本超过控件宽度时，文本会自动换行，而且在超过控件高度时，超出部分将被裁剪掉。

为使控件能够自动调整以适应内容多少，可将 AutoSize 属性设置为 True。这样控件可水平扩充以适应 Caption 属性内容。为使 Caption 属性的内容自动换行并垂直扩充，应将 WordWrap 属性设置为 True。

详细信息 关于 `AutoSize` 和 `WordWrap` 属性的简单示例，请参阅第三章“窗体、控件和菜单”中的“用标签显示文本”。

用标签创建访问键

如果要将 `Caption` 属性中的字符定义成访问键，应将 `UseMnemonic` 属性设置为 `True`。定义了 `Label` 控件的访问键后，用户按 `ALT+` 键指定的字符，就可将焦点按 `tab` 键次序移动到下一个控件。

在作为访问键的字母之前添加一个连字符 (`&`)，就可为其它具有 `Caption` 属性的控件创建访问键。为了对没有标题的控件指定访问键，可使用标签。标签不接受焦点，因此焦点会按照 `tab` 键次序自动移动到下一控件处。可用这种技术为文本框、图片框、组合框、列表框、驱动器列表框、目录列表框、网格和图像指定访问键。

要将标签指定为控件的访问键，请执行下述步骤：

1. 首先绘制标签，然后再绘制控件。

-或者-

以任意顺序绘制控件，并将标签的 `TabIndex` 属性设置为控件的 `TabIndex` 属性减 1。

2. 在标签的 `Caption` 属性中用连字符为标签指定访问键。

注意 有时可能要在 `Label` 控件中显示连字符而不是用它们创建访问键。如果在一次记录集中，数据包含连字符，而且要将 `Label` 控件绑定到记录集的某个字段，就会出现所说情况。为在 `Label` 控

件中显示连字符，应将 UseMnemonic 属性设置为 False。

使用 Line 控件

直线控件用来在窗体、框架或图片框中创建简单的线段。



图 7.29 Line 控件

可控制 Line 控件的位置、长度、颜色和样式来自定义应用程序的外观。如图 7.30 所示，用 Line 控件将包含文本 "Company Name" 的标签与窗体上的其它控件分隔。

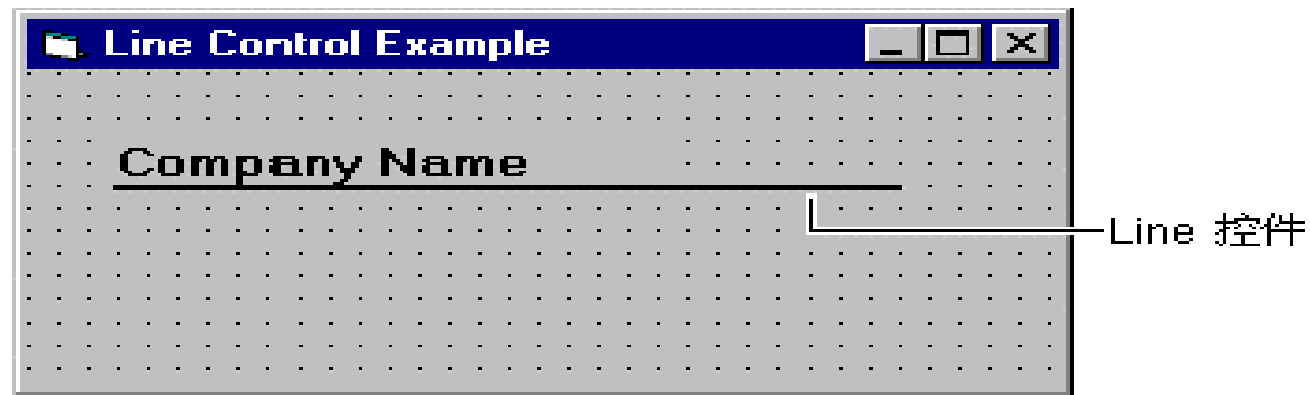


图 7.30 窗体上的 Line 控件

Line 控件功能有限，只用来完成简单的任务——显示和打印。例如，不能把直线段连接成其它图形。要完成高级的功能，应使用 Line 方法。

详细信息 关于使用 Line 方法在运行时绘制直线、矩形、填充的方框，或者关于使用 Circle 方法绘制圆、椭圆、圆弧的详细信息，请参阅第十二章“使用文本和图形”中的“使用 Graphics 方法”。

设置边界样式和颜色

设置线段的颜色和样式应使用 BorderStyle 和 BorderColor 属性。

BorderStyle 属性提供六种直线样式：

- 透明
- 实线
- 虚线
- 点线
- 点划线
- 双点划线
- 内实线

要指定直线样式，在设计时可从直线控件的“属性”窗口中选择并设置 BorderStyle 属性，而在运行时可在代码中使用相应的 Visual Basic 常数指定样式。

BackColor 属性用来指定直线的颜色。

在设计时从直线控件的“属性”窗口中选择 BorderColor 属性，然后从提

供的调色板或系统颜色中选择颜色，这样就可设置直线颜色。

为了在运行时设置颜色，应使用 Visual Basic 颜色常数（例如，vbGreen）或系统颜色常数（例如，vbWindowBackground），或使用 RGB 函数指定边界颜色。

注意 在将 BorderStyle 设置为 0（透明）时会忽略 BorderColor 属性。

详细信息 关于 Visual Basic 颜色常数表的信息，请参阅《Visual Basic 6.0 语言参考手册》的“颜色常数”。关于如何指定 RGB 颜色的信息，请参阅《Visual Basic 6.0 语言参考手册》的“RGB 函数”。同时，关于在 Visual Basic 中创建图形的详细信息，请参阅第十二章“使用文本和图形”。

移动和调整线段大小

在运行时，可更改直线控件的 X1、X2、Y1 和 Y2 属性来移动控件或调整直线控件的大小。用 X1 和 Y1 属性设置直线控件左端点的水平位置和垂直位置。用 X2 和 Y2 属性设置直线控件右端点的水平位置和垂直位置。不能用 Move 方法移动直线。

在窗体上画线

可用 Line 控件在窗体上绘制简单直线。

在窗体上绘制直线

1. 在工具箱中选择 Line 控件。

指针移动到窗体上时将变成一个十字形。

2. 在窗体上单击预想的直线起始处并按住鼠标按钮。
3. 拖动十字形到预想的直线末端处并松开鼠标按钮。
4. 若要改变直线的外观，则应从“属性”窗口中选定“边界样式”属性。
5. 在“设置”框中，选择所需样式。

使用 ListBox 控件

ListBox 控件显示项目列表，用户可从中选择一个或多个项目。



图 7.31 ListBox 控件

列表框为用户提供了选项的列表。虽然也可设置多列列表，但在缺省时将在单列列表中垂直显示选项。如果项目数目超过了列表框可显示的数目，控件上将自动出现滚动条。这时用户可在列表中上、下、左、右滚动。图 7.32 所示为一个单列列表框。



图 7.32 单列列表框

数据绑定特性

Visual Basic 包含 ListBox 控件的标准版本 (ListBox) 和数据绑定版本 (DBList)。虽然两种版本的控件都能显示、编辑和更新大多数标准类型数据库的信息，但是 DataList 提供了更高级的数据访问功能。DataList 控件还支持一套与标准 ListBox 控件不同的属性和方法。

详细信息 请参阅“使用 DataCombo 和 DataList 控件”以获得有关 Listbox 控件的数据绑定版本的详细信息。

Click 和 Double-Click 事件

对于列表框事件，特别是当列表框作为对话框的一部分出现时，建议添加一个命令按钮，并把该按钮同列表框并用。按钮的 Click 事件过程应该使用列表框的选项执行适于应用程序的操作。

双击列表中的项目与先选定项目然后单击命令按钮，这两者应该具有相同的效果。为此，应在 `ListBox` 控件的 `DblClick` 过程中调用命令按钮的 `Click` 过程：

```
Private Sub List1_DblClick ()  
    Command1_Click  
End Sub
```

也可将命令按钮的 `Value` 属性值设置为 `True`，这就将自动调用事件过程：

```
Private Sub List1_DblClick ()  
    Command1.Value = True  
End Sub
```

这将为使用鼠标的用户提供快捷方式，同时也没有妨碍使用键盘的用户执行同样的操作。注意，没有与 `DblClick` 事件等价的键盘命令。

向列表添加项目

为了向列表框中添加项目，应使用 `AddItem` 方法，其语法如下：

```
box.AddItem item[, index]
```

参数	描述
box	列表框的名称
item	添加到列表中的字符串表达式。若 <code>item</code> 是文字常数，则用引号将它括起来
index	指定在列表中插入新项目的位置。 <code>index</code> 为 0 表示第一个位置。若省略 <code>index</code> ，则将项目插入在末尾（或按排序次序插入在适当的位置）

通常在 `Form_Load` 事件过程中添加列表项目，但也可在任何时候使用 `AddItem` 方法添加项目，于是可动态（响应用户的操作）添加项目。下列代码将 "Germany"、"India"、"France" 和 "USA" 添加到名为 `List1` 的列表框中：

```
Private Sub Form_Load ()  
    List1.AddItem "Germany"  
    List1.AddItem "India"  
    List1.AddItem "France"  
    List1.AddItem "USA"  
End Sub
```

只要在运行时加载窗体就会出现如图 7.33 所示的列表。

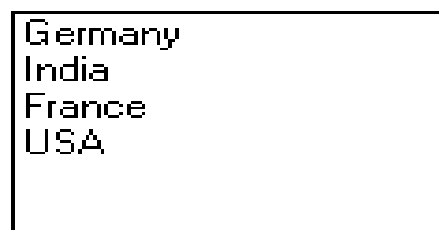


图 7.33 "Countries" 列表框

在指定位置添加项目

为了在指定位置添加项目，应对新项目指定索引值。例如，下行代码将 "Japan" 插入到第一个位置并把其它项目向下调整：

```
List1.AddItem "Japan", 0
```

注意，是 0 而不是 1 指定列表中的第一个位置（见图 7.34）。

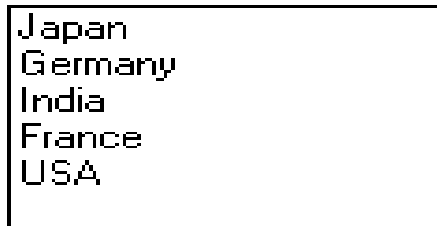


图 7.34 向列表添加项目

设计时添加项目

通过设置 `ListBox` 控件“属性”窗口的 `List` 属性还可在设计时向列表添加项目。在选定了 `List` 属性选项并单击向下箭头时，可输入列表项目并按 `CTRL+ENTER` 组合键换行。

只能在列表末端添加项目。所以，如果要将列表按字母顺序排序，则应将 `Sorted` 属性设置成 `True`。关于详细的信息，请参阅“排序列表”。

排序列表

可以指定要按字母顺序添加到列表中的项目，为此将 `Sorted` 属性设置为 `True` 并省略索引。排序时不区分大小写；因此单词 "japan" 和 "Japan" 将被同等对待。

`Sorted` 属性设置为 `True` 后，使用带有 `index` 参数的 `AddItem` 方法可能会导致不可预料的非排序结果。

从列表中删除项目

可用 `RemoveItem` 方法从列表框中删除项目。`RemoveItem` 有一参数 `index`，它指定删除的项目：

```
box.RemoveItem index
```

`box` 和 `index` 参数与 `AddItem` 中的参数相同。

例如要删除列表中的第一个项目，可添加下行代码：

```
List1.RemoveItem 0
```

要删除连结版或标准版的列表、组合框中的所有项目，应使用 `Clear` 方法：

```
List1.Clear
```

通过 `Text` 属性获取列表内容

通常，获取当前选定项目值的最简单方法是使用 `Text` 属性。`Text` 属性总是对应用户在运行时选定的列表项目。

例如，下列代码在用户从列表框中选定 `"Canada"` 时显示有关加拿大人口的信息：

```
Private Sub List1_Click ()
```

```
    If List1.Text = "Canada" Then
```

```
Text1.Text = "Canada has 24 million people."
```

```
End If
```

```
End Sub
```

Text 属性包含当前在 **List1** 列表框中选定的项目。代码检查是否选定了 "Canada"，若已选定，则在 **Text** 框中显示信息。

用 **List** 属性访问列表项目

可用 **List** 属性访问列表的全部项目。此属性包含一个数组，列表的每个项目都是数组的元素。每个项目以字符串形式表示。引用列表的项目时应使用如下语法：

```
box.List(index)
```

box 参数是列表框的引用，**index** 是项目的位置。顶端项目的索引为 0，接下来的项目索引为 1，依此类推。例如，下列语句在一个文本框中显示列表的第三个项目（**index** = 2）：

```
Text1.Text = List1.List(2)
```

用 **ListIndex** 属性判断位置

如果要了解列表中已选定项目的位置，则用 **ListIndex** 属性。此属性只在运行时可用，它设置或返回控件中当前选定项目的索引。设置列表框的 **ListIndex** 属性也将触发控件的 **Click** 事件。

如果选定第一个（顶端）项目，则属性的值为 0，如果选定下一个项目，则属性的值为 1，依此类推。若未选定项目，则 `ListIndex` 值为 -1。

注意 `NewIndex` 属性可用来跟踪添加到列表的最后一个项目的索引。在向排序列表插入项目时，这一点十分有用。

使用 `ListCount` 属性返回项目数

为了返回列表框中项目的数目，应使用 `ListCount` 属性。例如，下列语句用 `ListCount` 属性判断列表框中的项目数：

```
Text1.Text = "You have " & List1.ListCount & " _  
entries listed"
```

创建多列和多项列表框

可用 `Columns` 属性指定列表框中的列数目。此属性取值如下：

值	描述
0	垂直滚动的单列列表框
1	水平滚动的单列列表框
>1	水平滚动的多列列表框

如有必要，`Visual Basic` 可自动换行显示列表项目并为列表添加水平滚动条；若列表只填充在单列中，则不添加滚动条。`Visual Basic` 可根据需要自动换列显示。注意，若列表框的项目比列宽度要宽，则会截去文本超出的

部分。

用户可从列表中选择多个项目。设置 `MultiSelect` 属性来处理标准列表框中的多项选择，`MultiSelect` 属性取值如下：

值	选项类型	描述
0	无	标准列表框
1	简单多项选择	单击或按 SPACEBAR 键选定列表中的附加项目，或撤销对附加项目所作的选定
2	扩充的多项选择	可用 SHIFT+ 单击或 SHIFT+ 箭头键选定从上一个选定项到当前的选项之间的所有选项。CTRL+ 单击将选定（或撤销选定）列表中的项目

详细信息 关于 Columns 和 MultiSelect 属性的详细信息，请参阅本章后面的“列表框控件方案 2：创建多列列表框”。

列表框控件方案 1：添加和删除项目

本示例说明了如何使用 AddItem、RemoveItem、Clear 方法和 ListIndex、ListCount 属性在运行时添加或删除列表项目。在图 7.35 所示示例中，用户可把客户名输入文本框，然后单击“添加”按钮将输入项添加到列表中。为删除当前列表项目，可选定项目并单击“删除”按钮，要清除列表中所有的项目，可单击“清除”按钮。

在一个看似文本框（BorderStyle 设置为 1-Fixed Single）的标签中显示列表框中的客户数。每添加一个客户或删除一个客户时，标签都会更新。列表框的 Sorted 属性已设置为 True，所以按字母顺序添加项目。

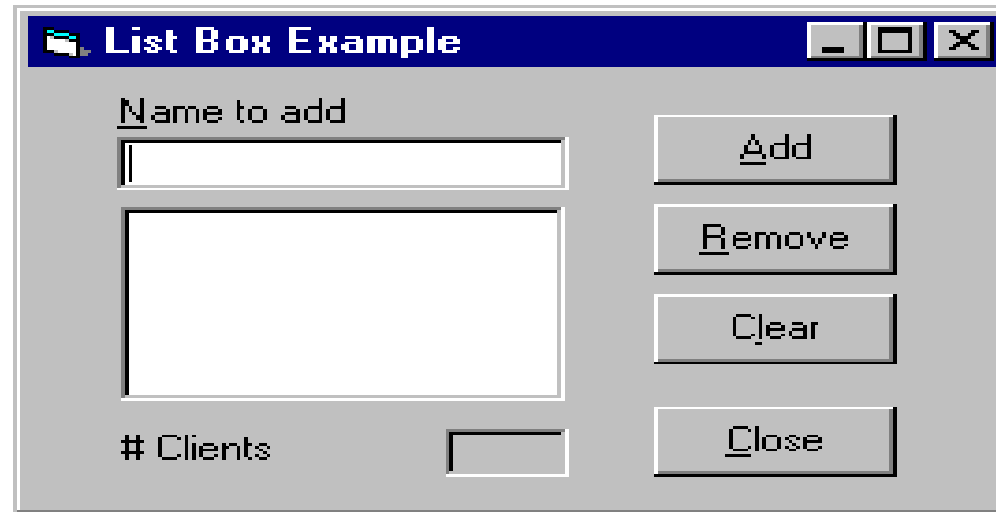


图 7.35 使用了 AddItem、RemoveItem 和 Clear 方法的列表框

创建一个窗体，其上有一个文本框、一个列表框、三个标签和四个命令按钮。下表列出了应用程序中对象的属性设置值。

对象	属性	设置值
顶端的文本框	Name	txtName
	Text	(Empty)
顶端的标签	Name	lblName
	Caption	&Name to add
列表框	Name	lstClient
	Sorted	True
底部的标签	Name	lblClients
	Caption	# Clients

续表

客户数标签 (看似一个文本框)	Name Caption BorderStyle	lblDisplay (Empty) 1-Fixed Single
第一个命令按钮	Name Caption	cmdAdd &Add
第二个命令按钮	Name Caption	cmdRemove &Remove
第三个命令按钮	Name Caption	cmdClear &Clear
第四个命令按钮	Name Caption	cmdClose &Close

列表框应用程序中的事件

将下列代码添加到 cmdAdd_Click 事件过程中：

```
Private Sub cmdAdd_Click ()  
    lstClient.AddItem txtName.Text '添加到列表中。  
    txtName.Text = "" '清除文本框。  
    txtName.SetFocus  
    '显示数目。  
    lblDisplay.Caption = lstClient.ListCount
```

```
End Sub
```

将下列代码添加到 cmdRemove_Click 事件过程中：

```
Private Sub cmdRemove_Click ()
```

```
    Dim Ind As Integer
```

```
    Ind = lstClient.ListIndex      '获取索引。
```

```
    '确保选定列表项目。
```

```
    If Ind >= 0 Then
```

```
        '将其从列表中删除。
```

```
        lstClient.RemoveItem Ind
```

```
        '显示数目。
```

```
        lblDisplay.Caption = lstClient.ListCount
```

```
    Else
```

```
        Beep
```

```
    End If
```

```
    '若列表中没有项目，则禁用此按钮。
```

```
    cmdRemove.Enabled = (lstClient.ListIndex <> -1)
```

```
End Sub
```

将下列代码添加到 cmdClear_Click 事件过程中：

```
Private Sub cmdClear_Click ()
```

```
    '清空列表框。
```

```
    lstClient.Clear
```

'禁用“删除”按钮。

`cmdRemove.Enabled = False`

'显示数目。

`lblDisplay.Caption = lstClient.ListCount`

`End Sub`

将下列代码添加到 `cmdClose_Click` 事件过程中：

`Private Sub cmdClose_Click ()`

`Unload Me`

`End Sub`

将下列代码添加到 `lstClient_Click` 事件过程中：

`Private Sub lstClient_Click ()`

`cmdRemove.Enabled = lstClient.ListIndex <> -1`

`End Sub`

将下列代码添加到 `txtName_Change` 事件过程中：

`Private Sub txtName_Change ()`

'当 `txtName` 文本框中至少有一个字符时，

'允许“添加”按钮。

`cmdAdd.Enabled = (Len(txtName.Text) > 0)`

`End Sub`

列表框控件方案 2：创建多列列表框

要创建多列、多选择列表框，有必要设置列表框的 `Columns` 和 `MultiSelect` 属性。下例中用这些属性创建这种列表框。

如图 7.36 所示，在运行应用程序时将会看到，列表框包含两列。

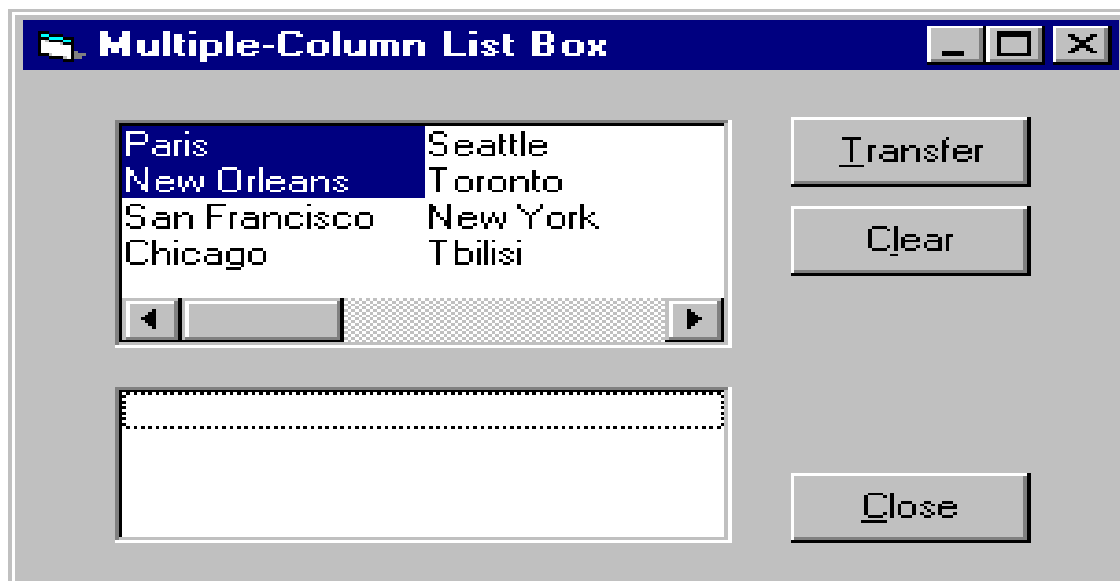


图 7.36 多列列表框

如果绘制的列表框大到可在一列中容纳所有项目，则第二列为空列；只有在列表框长度不够时，其它的项目才换行显示并自动添加水平滚动条。请调整顶端列表框的大小并添加附加的列表项目，以观察 Visual Basic 如何自动处理多个列。

示例中使用 `Selected` 属性——一个包含列表框选定状态的布尔数组——判断选定了哪些项目。数组中的每个元素对应一个列表项，选定项目的数组元素值为 `True`，未选定项目的数组元素值为 `False`。从列表中选定项目后，

代码将检查每个数组元素是否被设置为 **True**。如果项目的对应数组元素被设置为 **True**，则用 **AddItem** 方法将此项目添加到第二个列表（正常的单列列表框）中。

如下图所示，设置示例中控件的属性。

对象	属性	设置值
窗体	Caption	Multiple-Column 列表框
顶端的列表框	Name,Columns,MultiSelect	lstTop,2,2-Extended
底部的列表框	Name	LstBottom
第一个命令按钮	Name Caption	cmdTransfer & Transfer
第二个命令按钮	Name Caption	cmdClear & Clear
第三个命令按钮	Name Caption	cmdClose & Close

MultiSelect 属性用来选定列表框中的一组值。如果单击第一个列表项目，然后按 **SHIFT** 键并单击最后一个项目（或用 **SHIFT+DOWN** 方向键），则将选定此范围内的所有项目。

多列列表框应用程序中的事件

将下列代码添加到 **Form_Load** 过程中以初始化顶端的列表，**lstTop**:

```
Private Sub Form_Load ()  
    lstTop.AddItem "Paris"  
    lstTop.AddItem "New Orleans"  
    lstTop.AddItem "San Francisco"
```

```
lstTop.AddItem "Chicago"  
lstTop.AddItem "Seattle"  
lstTop.AddItem "Toronto"  
lstTop.AddItem "New York"  
lstTop.AddItem "Tbilisi"  
lstTop.AddItem "Moscow"  
lstTop.AddItem "Portland"  
'选定一对项目。  
lstTop.Selected(0) = True  
lstTop.Selected(1) = True
```

End Sub

注意 向“属性”窗口的 List 属性中输入项目,这样,不必重复使用 AddItem 方法就可在列表框中添加项目。在输入每个项目后按 CTRL+ENTER 键换行。用这种方法就可在多列列表框中输入多个项目。

将下列代码添加到 lstTop_DblClick 事件过程中:

```
Private Sub lstTop_DblClick ()  
    cmdTransfer.Value = True '按“传送”按钮。
```

End Sub

将下列代码添加到“传送”命令按钮的 Click 事件过程中:

```
Private Sub cmdTransfer_Click ()  
    For n = 0 To (lstTop.ListCount - 1)
```

'若被选定,则添加到列表中。

```
        If lstTop.Selected(n) = True Then
            lstBottom.AddItem lstTop.List(n)
        End If
    Next
    cmdClear.Enabled = True
End Sub
```

注意数组索引值以什么方式从 0 开始，并向 ListCount-1 转变。
将下列代码添加到“清除”命令按钮的 Click 事件过程中：

```
Private Sub cmdClear_Click ()
    lstBottom.Clear
    cmdClear.Enabled = False
End Sub
```

将下列代码添加到“关闭”命令按钮的 Click 事件过程中。

```
Private Sub cmdClose_Click ()
    Unload Me
End Sub
```

使用 OLE 容器控件

可将支持自动化（正式名称为 OLE 自动化）的对象链接或嵌入到 OLE 容器控件中。通过此控件，Visual Basic 应用程序可显示和操作其它基于

Windows 的应用程序（例如 Microsoft Excel 和 Microsoft Word for Windows）中的数据。



图 7.37 OLE 容器控件

OLE 容器控件通常用来创建以文档处理为中心的应用程序。在这种应用程序中，用户将不同应用程序的数据组合起来创建单个文档。这种类型的应用程序可能是字处理器，允许用户输入文本之后再嵌入一个工作表或图表。可用 OLE 容器控件把其它应用程序的对象添加到 Visual Basic 应用程序中。使用此控件可以：

- 在应用程序中创建一个对象的占位符。可创建运行时出现在 OLE 容器控件中的对象，也可在设计时改变放置在 OLE 容器控件上的对象。
- 在应用程序中创建链接的对象。
- 将 OLE 容器控件与某个数据库绑定。
- 如果用户移动、调整、更新 OLE 容器控件中的对象则执行相应的操作。
- 由数据来创建对象，这里的数据是已复制到剪贴板上的数据。
- 把对象显示为图标。
- 对于包含多个 OLE 容器控件（在以前的 Visual Basic 版本中称作 OLE 客户控件）的应用程序，提供与该应用程序向后兼容的能力。

详细信息 关于使用 OLE 容器控件的详细讨论和示例，请参阅第十章，“用部件编程”中“使用部件的可视界面”。

使用 `OptionButton` 控件

选项按钮被用来显示选项，通常以选项按钮组的形式出现，用户可从中选择一个选项。



图 7.38 `OptionButton` 控件

`OptionButton` 控件和 `CheckBox` 控件看起来功能相似，但却有一个重要区别：当用户选定一个选项按钮时，同组中的其它选项按钮会自动失效。与此对照的是，用户可选定任意数目的复选框。

详细信息 关于使用选项按钮的示例，请参阅第三章，“窗体、控件和菜单”中“将用选项按钮的选项分组”。

创建选项按钮组

要将选项按钮分组，可把它们绘制在不同的容器控件中，像 `Frame` 控件、`PictureBox` 控件，或窗体这样的容器控件中。运行时，用户在每个选项组中只能选定一个选项按钮。例如，如果把选项按钮分别添加到窗体和窗体上的一个 `Frame` 控件中，则相当于创建两组不同的选项按钮。

所有直接添加到窗体的选项按钮成为一组选项按钮。要添加附加按钮组，应把按钮放置在框架或 `PictureBox` 控件中。

要将框架或图片框中的 `OptionButton` 控件分组，应首先绘制框架或图片框，然后在内部绘制 `OptionButton` 控件。设计时，可选择 `Frame` 控件或 `PictureBox` 控件中的选项按钮，并把它们作为一个单元来移动。

要选定 `Frame` 控件、`PictureBox` 控件或窗体中所包含的多个控件时，可在按住 `CTRL` 键的同时用鼠标在这些控件周围绘制一个方框。

详细信息 关于在框架内部绘制控件的详细信息，请参阅本章后面的“使用 `Frame` 控件”。

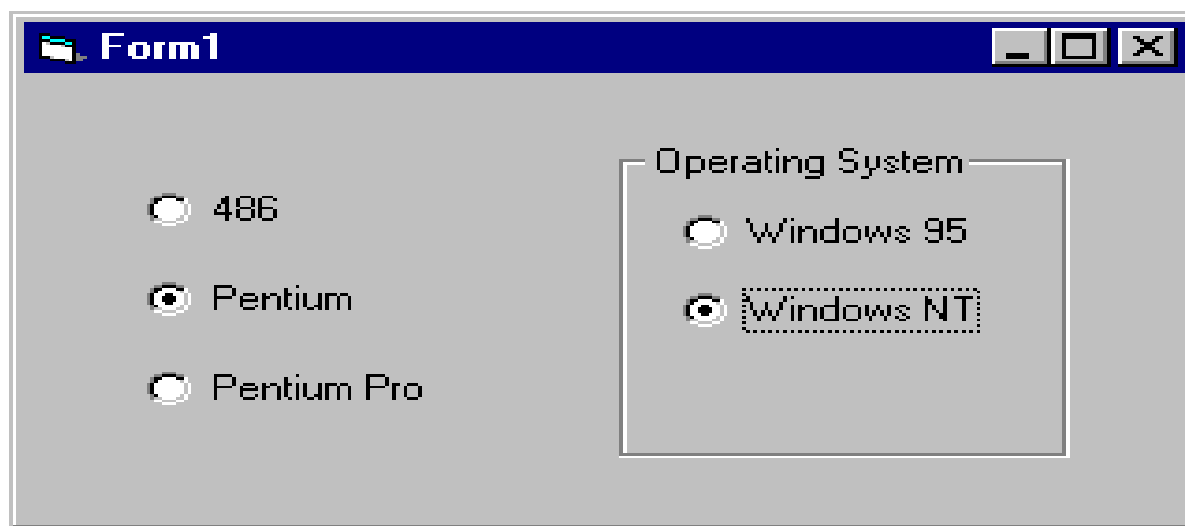


图 7.39 创建选项按钮组

运行时选择选项按钮

在运行时，有若干种选定选项按钮的方法：用鼠标单击按钮，用 `TAB` 键将

焦点转移到控件，用 **TAB** 键选择一组选项按钮后再用方向键从组中选定一个按钮，在选项按钮的标题上创建快捷键，或者在代码中将选项按钮的 **Value** 属性设置为 **True**。

Click 事件

选定选项按钮时将触发其 **Click** 事件。是否有必要响应此事件，这将取决于应用程序的功能。例如，当希望通过更新 **Label** 控件的标题向用户提供有关选定项目的信息时，对此事件作出响应是很有益的。

Value 属性

选项按钮的 **Value** 属性指出是否选定了此按钮。选定时，数值将变为 **True**。可在代码中设置选项按钮的 **Value** 属性来选定按钮。例如：

```
optPentium.Value = True
```

要在选项按钮组中设置缺省选项按钮，可在设计时通过“属性”窗口设置 **Value** 属性，也可在运行时在代码中用上述语句来设置 **Value** 属性。

在向用户显示包含选项按钮的对话框时将要求他们选择项目，确定应用程序下一步做什么。可用每个 **OptionButton** 控件的 **Value** 属性判断用户选定的选项并作出相应的响应。

创建键盘快捷方式

可用 **Caption** 属性为选项按钮创建访问键快捷方式，这只要在作为访问键

的字母前添加一个连字符 (&)。例如，要为选项按钮标题 "Pentium" 创建访问键，应在字母 "P" 前添加连字符： "&Pentium" 。运行时，字母 "P" 将带下划线，同时按 ALT+P 组合键就可选定此命令按钮。

注意 要使标题包含连字符但不创建访问键，就应使标题包含两个连字符 (&&)。这样，标题中将显示一个连字符，而且没有字符带下划线。

禁止选项按钮

要禁止选项按钮，应将其 Enabled 属性设置成 False。运行时将显示暗淡的选项按钮，这意味着按钮无效。

增强 OptionButton 控件的视觉效果

可以改善 OptionButton 控件的外观，这只要改变 Style 属性的设置，然后使用 Picture、DownPicture 和 DisabledPicture 属性。

使用 PictureBox 控件

PictureBox 控件被用来显示图形，作为其它控件的容器，显示图形方法的输出或显示 Print 方法输出的文本。



图 7.40 PictureBox 控件

PictureBox 控件和 Image 控件相似，每个控件都可用来说明应用程序中的图形——每个都支持相同的图形格式。但是，PictureBox 控件包含了 Image 控件不具有的功能，例如：作为其它控件的容器的功能和支持图形方法的功能。

详细信息 关于 PictureBox 控件的示例，请参阅第三章“窗体、控件和菜单”中的“使用 PictureBox 控件”。

支持的图形格式

PictureBox 控件可显示下述任何格式的图片文件：位图、图标、图元文件、增强型图元文件、JPEG 或 GIF 文件。

详细信息 关于这些图形格式详细的说明，请参阅本章前面的“使用 Image 控件”。

将图形加载到 PictureBox 控件中

在设计时从“属性”窗口中选定并设置 Picture 属性就可将图片加载到 PictureBox 控件中，也可在运行时用 Picture 属性或 LoadPicture 函数做到这一点。

```
Set Picture1.Picture = _
```

```
LoadPicture("c:\Windows\Winlogo.cur", vbLPLarge, vbLPColor)
```

您可能希望使用包含大小不同和颜色深度不同的图像以支持多种显示设备的独立图像的图标（.icon）和光标（.cur）文件。LoadPicture 函数的设置允许您从 .ico 或 .cur 文件中选择特定颜色深度和大小的图像。假如没有要求设置的准确匹配，LoadPicture 将加载可获得的最匹配图像。

为清除 PictureBox 控件中的图形，应使用不指定文件名的 LoadPicture 函数。例如：

```
set Picture1.Picture = LoadPicture
```

这将清除 PictureBox 控件，即使在设计时向 Picture 属性加载了图形亦如此。

使用剪贴板

设计时也可这样向 PictureBox 控件添加图形：从其它应用程序中复制图形后把它粘贴到 PictureBox 控件中。例如，有时可能希望添加由 Windows Paint 创建的位图图像。直接把图像复制到剪贴板，选定 PictureBox 控件，然后使用键盘快捷方式 CTRL+V 或使用“编辑”菜单的“粘贴”命令。

调整图片的大小

缺省时，加载到图片框中的图形保持其原始尺寸，这意味着如果图形比控件大，则超过的部分将被剪裁掉——PictureBox 控件不提供滚动条。要使 PictureBox 控件自动调整大小以显示完整图形，应将其 AutoSize 属性设置为 True。这样控件将自动调整大小以适应加载的图形。

与 Image 控件不同， PictureBox 控件不能伸展图像以适应控件尺寸。

详细信息 关于使用图片框创建可滚动的图形视窗的信息，请参阅本章前面的“ScrollBar 控件方案：创建可滚动的图形视窗”。

用 PictureBox 控件作容器

可用 PictureBox 控件作为其它控件的容器。例如，因为可将 PictureBox 控件放置到 MDI 窗体的内部区域，所以通常用它手工创建工具条或状态条。

详细信息 关于用 PictureBox 控件作其它控件容器的详细信息，请参阅第六章“创建用户界面”中的“创建 Toolbar”。

图形方法

图片框像窗体一样，可用来显示图像方法（例如 Circle、Line 和 Point）的输出。例如，将 PictureBox 控件的 AutoRedraw 属性设置为 True 后，就可用 Circle 方法在控件上绘制一个圆。

```
Picture1.AutoRedraw = True
```

```
Picture1.Circle (1200, 1000), 750
```

将 AutoRedraw 设置成 True 后，可将图形方法的输出显示在控件上，而且，在调整 PictureBox 控件大小或移去隐藏图片框的对象重新显示图片框时，控件可自动重新绘制显示这些输出。

详细信息 关于在 PictureBox 控件中使用图形方法的详细信息，请参阅第

十二章，“使用文本和图形”中的“使用图形方法”。

使用 Print 方法

将 `AutoRedraw` 属性设置为 `True` 并使用 `Print` 方法就可在 `PictureBox` 控件上输出文本。例如：

```
Picture1.Print "A text string"
```

使用 `Print` 方法时可修改字体样式和大小，也可使用 `CurrentX`、`CurrentY`、`Height` 和 `Width` 属性对齐图片框中的文本。

详细信息 请参阅第十二章“使用文本和图形”以及“`Print` 方法”。

使用 RemoteData 控件

`Remote Data` 控件通过使用 `Microsoft` 的远程数据对象来实现数据访问。该技术提供了对标准数据库格式的无缝访问，用户不必编写任何代码就可以创建数据识别的应用程序。`Remote Data` 控件适合于较大的客户-服务器数据库，包括开放式数据库连接性（`ODBC`）数据库，诸如 `Microsoft SQL Server` 以及 `Oracle`。

需要注意的是，`RemoteData` 控件不支持 `Image` 数据类型。

注意 `Remote Data` 控件和 `Data` 控件二者都包括在 `Visual Basic` 中，以提供向后兼容性。不过，由于 `ActiveX` 数据对象（`ADO`）的适应性更广，

建议使用 ADO Data 控件来创建新的数据库应用程序。详细信息，请参阅“使用 ADO Data 控件”。

Data 控件、Remote Data 控件以及 ADO Data 控件在概念上很相似：三者都是将一个数据源连接到一个数据绑定控件的“数据控件”。而且三者共享相同的外观——一组四个的按钮，使用户可以迅速到达记录集的开始、记录集的末尾、或在记录集中向前或向后翻卷。

使用 Remote Data 控件创建一个简单的数据库应用程序

下面的操作过程使用这种 Data 控件创建了一个简单的数据库应用程序。

要使用 Remote Data 控件创建一个简单的数据库应用程序

1. 在窗体上放置一个 Remote Data 控件（该图标的工具提示控件为“MSRDC”）。

如果 Remote Data 控件不在“工具箱”中，则按 CTRL+T 键来显示“部件”对话框。在“部件”对话框中，单击“Microsoft RemoteData Control”。然后单击“确定”将其添加到“工具箱”中。

2. 单击选定该 Remote Data 控件，然后按 F4 键显示“属性”窗口。
3. 在“属性”窗口中，将“DataSourceName”属性设置为要连接的数据库的 DSN。
4. 在“属性”窗口中，单击“SQL”属性并输入一个 SQL 语句，例如：

```
SELECT * FROM Products WHERE AuthorID = 72
```

在访问一个表时，应始终包括一个 WHERE 子句。不这样做就会锁定整个表，这对其他用户将是一个严重的障碍。

5. 在窗体上再放置一个 `TextBox` 控件。
6. 单击并选定该文本框控件，然后在其“属性”窗口将“数据源”属性设置为该 `RemoteData` 控件。
7. 在“属性”窗口中，将“数据字段”属性设置为要查看或修改的数据库中的字段名称。
8. 对希望访问的其他的每个字段重复第 5、6、7 步。
9. 按 `F5` 键运行该应用程序。

设置 `RemoteData` 控件的与数据相关的属性

下述与数据相关的属性可以在设计时设置。下表给出了设置这些属性的一种逻辑顺序：

1. **Connect**——“连接”属性是一个字符串，该字符串可以包含进行一个连接所需的所有设置值。在该字符串中所传递的参数是与驱动程序相关的。例如，`ODBC` 驱动程序允许该字符串包含驱动程序、数据库、用户名称以及密码。
2. **UserName**——对一个受保护的数据库标识一个用户。该用户必须使用一个合法的、该数据库管理系统能识别的密码。用户名称也可以包含在“连接”属性值中，此时本属性中的设置就是多余的。
3. **Password**——与用户名称一起使用，密码使用户可以访问受保护的数据。密码也可以包含在“连接”属性值中，此时本属性中的设置就是多余的。
4. **SQL**——`SQL` 属性包含了用于检索一个结果集的 `SQL` 语句。结果集的大小可以决定是使用客户端光标还是使用服务器端的光标。例如，一个

小的结果集可以用一个客户端的光标来管理。

5. **RowsetSize**——设置在结果集返回的行数，如果光标为键集光标。可以根据计算机的资源（内存）调整这个数目，以获得性能的改善。
6. **ReadOnly**——指定数据是否可以写入。如果正在编写的数据是不需要的，将这个属性设为“真”可以获得性能的改善。
7. **CursorDriver**——决定驱动程序的位置和类型。这个属性的设置值将影响其它属性的设置。例如，只要结果集比较小，则选择 ODBC 客户端光标可以提高性能。
8. **LockType**——LockType 属性决定当其他人试图更改数据时如何锁定该数据。如果不希望其他人更改该数据（当您正在查阅时），可以将 LockType 设置为 "optimistic"——即其他人可以自由地查看和更改该数据。如果将该属性设置为 "pessimistic"，那么在您访问该数据期间，其他人不能访问该数据。
9. **BOFAction**、**EOFAction**——这两个属性决定当该控件位于光标的开始和末尾时的行为。提供的选择包括停留在开始或末尾、移动到第一个或最后一个记录、或添加一个新记录（只有在末尾时）。
10. **ResultSetType**——决定光标是静态类型还是键集类型。
11. **KeySetSize**——如果光标为键集类型，则可以使用 KeySetSize 属性来优化返回的结果集的大小。
12. **LoginTimeout**——设置等待的秒数，超时则返回一个错误。
13. **MaxRows**——指定光标的大小。如何确定这个属性取决于所检索的记录大小，以及您的计算机上可用资源（内存）的多少。大的记录（具有很

多列以及字符串) 与小的记录相比会消耗更多的资源。MaxRows 属性就应该相应地减少。

- 14. **Options**——指定该控件是否异步地执行查询。当估计一个查询可能要花费好几分钟来执行时，应使用异步操作。
- 15. **Prompt**——在 RDO 基于该 RemoteData 控件的参数打开一个连接时，Connect 属性应当包含足够的用于建立该连接的信息。如果没有提供像数据源名称、用户名称、或密码等这样的信息，则 ODBC 驱动程序管理器将显示一个或多个对话框，以便从用户取得这些信息。如果不想让这些对话框出现，可以相应地设置这个 Prompt 属性来取消这一功能。
- 16. **QueryTimeout**——设置等待一个查询完成的秒数，超时则返回一个错误。
- 17. **BatchSize**——这个属性决定在一个批处理中可以发送多少条语句——如果驱动程序允许使用批处理语句。

使用 Shape 控件

可用 Shape 控件在窗体、框架或图片框中创建下述预定义形状：矩形、正方形、椭圆形、圆形、圆角矩形或圆角正方形。



图 7.41 Shape 控件

可以设置绘制在窗体上的任意形状的形状样式、颜色、填充样式、边框颜色和边框样式。

完成简单功能时，可用形状控件创建多种形状而无需编写代码。要实现更高级的功能，应使用 `Line` 和 `Circle` 方法。

详细信息 关于在运行时使用 `Line` 方法绘制直线、矩形、填充的方框，或者关于使用 `Circle` 方法绘制圆、椭圆、圆弧的详细信息，请参阅第十二章，“使用文本和图形”中的“使用图形方法”。

预定义的形状

`Shape` 控件的 `Style` 属性提供了六种预定义的形状。下表列出所有预定义形状、形状值和相应的 `Visual Basic` 常数：

形状	样式	常数
矩形	0	<code>vbShapeRectangle</code>
正方形	1	<code>vbShapeSquare</code>
椭圆形	2	<code>vbShapeOval</code>
圆形	3	<code>vbShapeCircle</code>
圆角矩形	4	<code>vbShapeRoundedRectangle</code>
圆角正方形	5	<code>vbShapeRoundedSquare</code>

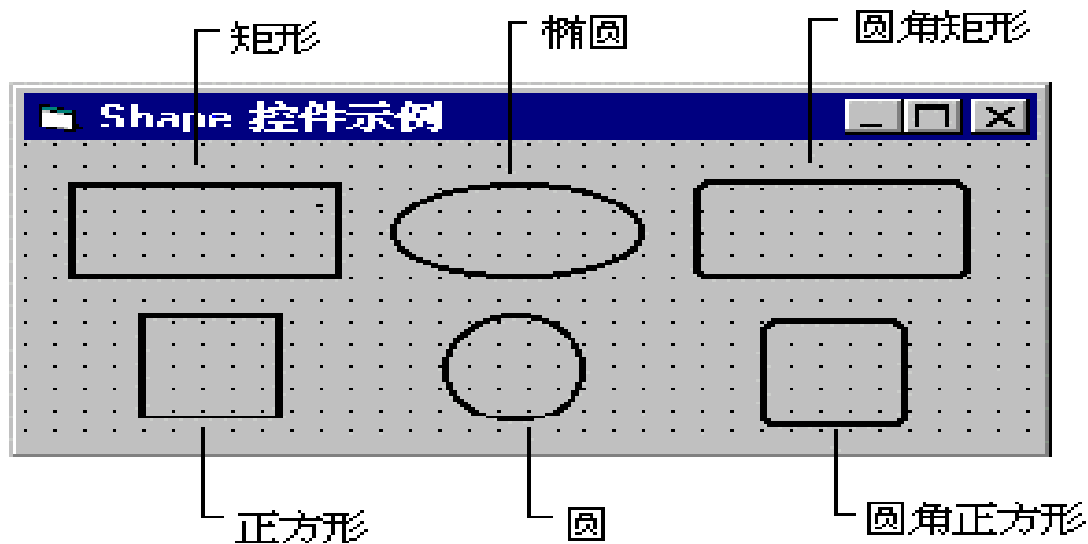


图 7.42 预定义的形状

Fill 和 Line 样式

可用 `FillStyle` 和 `BorderStyle` 属性对绘制在窗体上的任意形状设置填充样式和边框样式。

`FillStyle` 属性和 `Style` 属性同样都提供了若干预定义的填充样式图案，其中包括：实线、透明、水平线、垂直线、向上对角线、向下对角线、十字线和对角十字线。

`BorderStyle` 属性提供若干预定义的边框样式，其中包括：透明、实线、虚线、点线、点划线、双点划线和内实线。

详细信息 `FillStyle` 和 `BorderStyle` 属性都提供了代表上面所列样式的常

数。关于详细信息，请参阅《Visual Basic 6.0 语言参考手册》的“FillStyle 属性”和“BorderStyle 属性”。

设置 Color 属性

可用 BackColor 和 FillColor 属性为形状和形状的边框添加颜色。

设计时，可从 Shape 控件的“属性”窗口中选定填充或边框颜色属性，然后从提供的调色板或系统颜色中选择要设置的颜色。

为在运行时设置颜色，可使用 Visual Basic 颜色常数（如，vbGreen）或系统颜色常数（例如 vbWindowBackground），还可使用 RGB 函数指定填充颜色。

注意 当把 FillStyle 或 BackStyle 属性设置为 1 (Transparent) 时将忽略 FillColor 和 BackColor 属性。

详细信息 关于 Visual Basic 颜色常数列表，请参阅《Visual Basic 6.0 语言参考手册》的“颜色常数”。关于指定 RGB 颜色的信息，请参阅《Visual Basic 6.0 语言参考手册》的“RGB 函数”。同时，关于在 Visual Basic 中创建图形的详细信息，请参阅第十二章“使用文本和图形”。

在窗体上绘制形状

可用 Shape 控件在窗体上绘制（普通或圆角）矩形、（普通或圆角）正方形、椭圆形和圆形。

要在窗体上绘制形状，请按以下步骤执行：

18. 在工具箱中，选定 **Shape** 控件。

当指针移动到窗体上时就变成了一个十字。

19. 单击并拖动十字使所绘制的形状达到所需大小。

20. 在“属性”窗口中选定 **Shape** 属性。

21. 在“设置值”框中选择所需样式。

调整 **Shape** 控件大小的方法和调整其它控件的方法相同：选定并拖动控件，将其调整到所需尺寸，或者设置控件的 **Height** 和 **Width** 属性。

使用 **TextBox** 控件

TextBox 控件被用来在运行时显示用户输入的信息，或者在设计或运行时为控件的 **Text** 属性赋值。



图 7.43 **TextBox** 控件

TextBox 控件可用于编辑文本，虽然也可将其 **Locked** 属性设置为 **True** 使其成为只读的。还可用文本框实现多行显示、根据控件的尺寸自动换行以及添加基本格式的功能。

Text 属性

Text 属性包含输入到 **TextBox** 控件中的文本。缺省时，文本框中输入的字符最多为 2048 个。若将控件的 **MultiLine** 属性设置为 **True**，则可输入多达 32K 的文本。

格式化文本

当文本超过控件边界时可将 **MultiLine** 属性设置为 **True**，使控件自动换行，并可将 **ScrollBars** 属性设置成添加水平滚动条或垂直滚动条（或者两种都添加），由此即添加了滚动条。但是，如果添加滚动条，那么，由于出现滚动条而使水平编辑区域增大，自动文本换行功能就会失败。

当把 **MultiLine** 属性设置为 **True** 时，可将文本的对齐方式调整为左对齐、中央对齐或右对齐。缺省设置为左对齐。如果 **MultiLine** 属性为 **False**，则 **Alignment** 属性无效。

详细信息 关于 **MultiLine**、**ScrollBar** 和 **Alignment** 属性的示例，请参阅第三章“窗体、控件和菜单”中的“使用 **TextBox**”。

选择文本

可通过 **SelStart**、**SelLength** 和 **SelText** 属性控制文本框中的插入点和文本选定操作。

详细信息 关于 SelStart、SelText 和 SelLength 属性的示例，请参阅第三章“窗体、控件和菜单”中的“使用 TextBox”。

创建密码文本框

密码框是一个文本框允许在用户输入密码的同时显示星号之类的占位符。Visual Basic 提供 PasswordChar 和 MaxLength 这两个文本框属性，大大简化了密码文本框的创建。

PasswordChar 指定显示在文本框中的字符。例如，若希望在密码框中显示星号，则可在“属性”窗口中将 PasswordChar 属性指定为 *。如图 7.48 所示，无论用户输入什么字符，文本框中都显示星号。

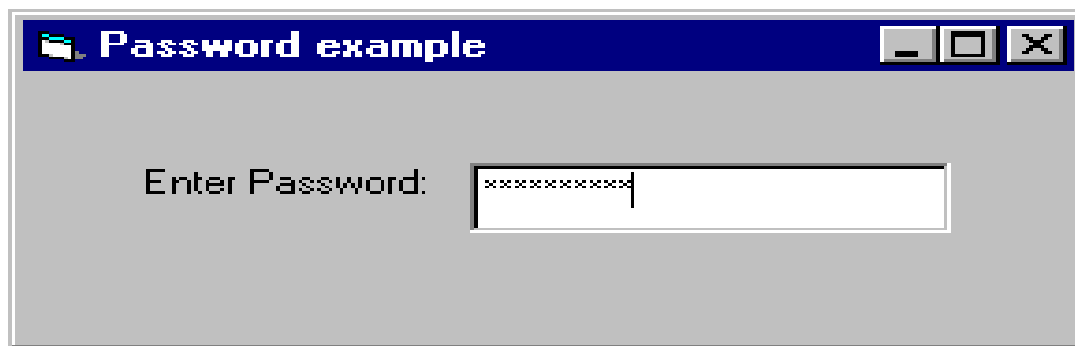


图 7.44 密码示例

可用 MaxLength 设定输入文本框的字符数。输入的字符数超过 MaxLength 后，系统不接受多出的字符并发出嘟嘟声。

取消文本框中的击键值

可用 `KeyPress` 事件限制或转换输入的字符。`KeyPress` 事件使用一个参数 `keyascii`。此参数为整型数值，代表输入到文本框中字符的数值 (`ASCII`)。

下例对如何在输入字符时取消击键值进行演示。若输入的字符不在指定范围内，则该过程通过将 `KeyAscii` 设置为 0 取消这个字符。在本例中，文本框的名称为 `txtEnterNums`，该过程将不允许文本框接受任何非数字的字符。示例中把 `KeyAscii` 与多个字符的数值型 (`Asc`) 值直接比较。

```
Private Sub txtEnterNums_KeyPress (KeyAscii As Integer)
    If KeyAscii < Asc("0") Or KeyAscii > Asc("9") Then
        KeyAscii = 0           '取消字符。
        Beep                   '发出错误信号。
    End If
End Sub
```

详细信息 关于 `KeyPress` 事件的详细信息，请参阅第十一章，“响应鼠标和键盘事件”中“响应键盘事件”。

创建只读文本框

可用 `Locked` 属性防止用户编辑文本框内容。将 `Locked` 属性设置为 `True` 后，用户就可滚动文本框中的文本并将其突出显示，但不能作任何变更。将 `Locked` 属性设置为 `True` 后就可可在文本框中使用“复制”命令，但不能

使用“剪切”和“粘贴”命令。`Locked` 属性只影响运行时的用户交互。这时仍可变更 `Text` 属性，从而在运行时通过程序改变文本框的内容。

打印字符串中的引号

引号 (" ") 有时出现在文本的字符串中。

```
She said, "You deserve a treat!"
```

因为赋予变量或属性的字符串都用引号 (" ") 括起来，所以对于字符串中要显示的一对引号，必须再插入一对附加的引号。`Visual Basic` 将并列的两对引号解释为嵌入的引号。

例如，要显示下面的字符串就应使用下述代码：

```
Text1.Text = "She said, ""You deserve a treat!"" "
```

可用引号的 `ASCII` 字符 (34) 达到相同效果：

```
Text1.Text = "She said, " & Chr(34) + "You deserve a treat!" & Chr(34)
```

使用 `Timer` 控件

`Timer` 控件响应时间的流逝。它们独立于用户，编程后可用来在一定的时间间隔内执行操作。此控件的一个一般用处是检查系统时钟，判断是否该执行某项任务。对于其它后台处理，`Timer` 控件也非常有用。



图 7.45 Timer 控件

每个 Timer 控件都有 Interval 属性，指定定时器事件之间的毫秒数。除非禁止此属性，否则定时器在大致相等的时间间隔内不断接受事件（称作定时器事件会更贴切）。

在为 Timer 控件编程时应考虑对 Interval 属性的几条限制：

- 如果应用程序或其它应用程序正在进行对系统要求很高的操作——例如长循环、高强度的计算或者正在访问驱动器、网络或端口——则应用程序定时器事件的间隔可能比 Interval 属性指定的间隔长。
- 间隔的取值可在 0~64,767 之间（包括这两个数值），这意味着即使是最长的间隔也不比一分钟长多少（大约 64.8 秒）。
- 间隔并不一定十分准确。要保证间隔准确，应在需要时才让定时器检查系统时钟，而不在内部追踪积聚的时间。
- 系统每秒生成 18 个时钟信号——所以即使用毫秒衡量 Interval 属性，间隔实际的精确度不会超过 1/18 秒。

每个 Timer 控件必须要与窗体关联，因此要创建定时器应用程序就必须至少创建一个窗体（如果不需要窗体完成其它操作就不必使窗体可见）。

注意 在 Visual Basic 中，单词 "timer" 有好几种用法，每种都与 Timer 控件的工作有关。除了控件名和控件类型以外，"timer" 还用在定时器事件和定时器函数中。

示例应用程序：Alarm.vbp

Alarm 示例 (Alarm.vbp，列在 Samples 目录中)应用程序中包含了使用 Timer 控件的技巧。

在窗体上放置 Timer 控件

在窗体上放置 Timer 控件的方法与绘制其它控件的方法相同：单击工具箱中的定时器按钮并将它拖动到窗体上。

Timer 控件只在设计时出现在窗体上，所以可以选定这个控件，查看属性，编写事件过程。运行时，定时器不可见，所以其位置和大小无关紧要。

初始化 Timer 控件

Timer 控件有两个关键属性。

属 性	设置值
Enabled	若希望窗体一加载定时器就开始工作，应将此属性设置为 True，否则，保持此属性为 False。有时可能选择由外部事件（例如单击命令按钮）启动定时器操作
Interval	定时器事件之间的毫秒数

注意，定时器的 Enabled 属性不同于其它对象的 Enabled 属性。对于大多数对象，Enabled 属性决定对象是否响应用户触发的事件。对于 Timer 控件，将 Enabled 设置为 False 时就会暂停定时器操作。

记住，Timer 事件是周期性的。Interval 属性主要是决定“多少次”而不是“多久”。间隔的长度取决于所需精确度。因为存在某些内部的错误可能性，所以应将间隔设置为精确度的一半。

注意 定时器事件生成越频繁，响应事件所使用的处理器事件就越多。这将降低系统综合性能。除非有必要，否则不要设置过小的间隔。

Timer 控件方案：响应 Timer 事件

经历了 Timer 控件的时间间隔后，Visual Basic 将生成 Timer 事件。通常，在响应此事件时将检查某些一般条件，例如系统时钟。

数字型时钟是涉及 Timer 控件的简单而有用的应用程序。一旦理解此应用程序的工作方式，就可增强它的性能，实现闹钟、跑表或其它定时设备的功能。

Digital Clock 应用程序包含一个定时器和一个有边框的标签。设计时，应用程序如图 7.46 所示。

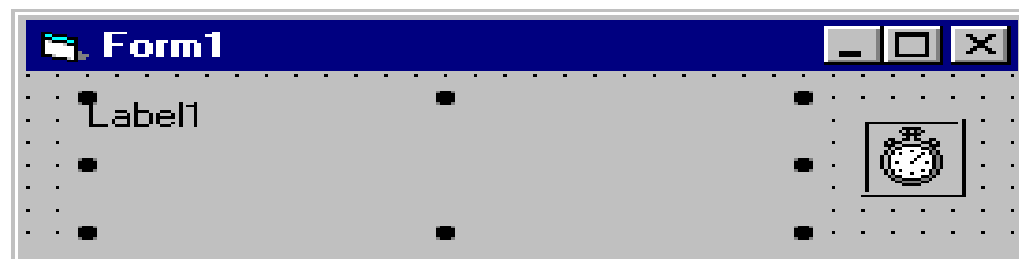


图 7.46 Digital Clock 应用程序

定时器在运行时不可见。下表列出了 Digital Clock 应用程序中的属性设置值。

控 件	属 性	设 置 值
Label1	BorderStyle	Fixed Single
Timer1	Interval	500（半秒）
Timer1	Enabled	True

下列应用程序中的唯一过程是定时器的事件过程：

```
Private Sub Timer1_Timer ()  
    If lblTime.Caption <> CStr(Time) Then  
        lblTime.Caption = Time  
    End If  
End Sub
```

过程调用内在的 Time 函数来显示系统时间。此函数返回一个 Variant，以日期/时间数值 (VarType 7) 的形式包含当前时间。将它赋予某个字符串变量或属性时，比如赋予本例中的 Caption 属性时，Visual Basic 将用“控制面板”中指定的格式将其转换成一个字符串。若要用其它格式显示，可使用 Format 函数。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的“Format 属性”。将定时器的 Interval 属性设置为 500，按照上述原则将 Interval 设置为要区分的最短时间（本例中为一秒钟）的一半。这会使 Timer 控件代码每半秒更新一次标签。这样做不仅浪费而且可能导致视觉颤动，因此代码在改变标题之前要检查当前时间是否与在标签上显示的时间不同。

无需另外编写语句就可自定义数字时钟的外观。例如，可为标签选择其它字体或改变窗体的 `BorderStyle` 属性。

第八章 再 论 编 程

本章在 Visual Basic 编程的基本功能之外介绍其它各种特色，这些特色能使您方便地创建强大而灵活的应用程序。

例如，可以将多个工程加载到编程环境的单个会话中，利用 Windows 的注册设置值进行工作，或者有选择地编译程序的某些部分。

除了编写代码的基本功能以外，Visual Basic 还提供了多种能增强代码功能的语言元素。本章的最后四个主题将讨论其中四种语言元素：用户自定义类型、枚举常数、数组和集合。

目 录

- 利用多个工程进行工作
- 管理应用程序设置值
- 使用条件编译
- 利用资源文件进行工作
- 利用模板进行工作
- 利用命令行开关进行工作
- 将工程编译成本机代码

- 创建自己的数据类型
- 通过枚举使用常数集进行工作
- 用集合替代数组

利用多个工程进行工作

可以通过单个工程进行工作来创建许多应用程序。但是，随着应用程序越来越复杂，也许希望在编程环境的同一个会话中利用多个工程进行工作。例如，也许想利用一个工程建立一个应用程序的可执行文件，利用第二个工程作为“scratch-pad”以便在将代码添加到应用程序之前测试该代码。通过将一个新的或已有的工程添加至工程组，可以将其添加到当前的编辑会话中。然后可以保存该工程组并在以后的编辑会话中利用其进行工作。可以打开整个工程组或工程组中的单个工程，或者将该工程组或其中的单个工程添加至另一个工程组中。

在工程组中，一个可执行的工程被当作启动工程。当一个工程组处于打开状态并且从“运行”菜单中选择了“启动”，在工具栏单击“启动”按钮或者按 F5 键，Visual Basic 将运行启动工程。

在专业版和企业版中，可以使用工程组创建并调试多部件的应用程序。例如，可以创建和调试包含标准可执行工程、ActiveX 可执行工程、ActiveX 动态链接库工程或 ActiveX 控件工程的工程组。详细内容，请参阅随 Visual Basic 专业版和企业版提供的《部件工具指南》中的“创建 ActiveX 部件”。

添加或删除工程

当在启动时通过在“文件”菜单中选择“新建工程”命令创建一个新的工程时，Visual Basic 自动创建一个工程组。然后可以将新的或现有的工程添加到该工程组中。

要向工程组中添加新的工程，请执行下述步骤：

- 单击工具栏中的“添加工程”按钮或从“文件”菜单中选择“添加工程”。
在 Visual Basic 学习版中，Visual Basic 自动向工程组中添加一个新的可执行工程。在专业版和企业版中，则可以从“添加工程”按钮的弹出式菜单中或者从显示在“添加工程”对话框的“新建”选项卡上的图标中选择希望添加的工程类型。

要向工程组中添加现有的工程，请按照以下步骤执行：

1. 从“文件”菜单选择“添加工程”。

Visual Basic 将显示“添加工程”对话框。

2. 单击“现存”选项卡。

3. 选择工程文件，然后选择“打开”。

Visual Basic 将选择的工程添加至工程组中。

Visual Basic 在工程窗口中有层次地显示多个工程。每一个工程显示在最上一级，工程的窗体、模块、控件、属性页或者文档对象成组地按层次排列在工程下。

要从工程组中删除工程，请按照以下步骤执行：

1. 在“工程”窗口中选择工程的名字。

2. 从“文件”菜单中选择“删除工程”。

Visual Basic 将从工程组中删除选择的工程。

在 Visual Basic 的专业版和企业版中，可以改变工程的类型，方法是在“工程”菜单中选择“工程属性”命令，然后改变在“工程属性”对话框中“通用”选项卡上的“工程类型”选项。



图 8.1 显示多个工程的工程窗口



图 8.2 “工程属性”对话框中的“通用”选项卡

指定一个启动工程

因为一个工程组包含了多个工程，因此当从“运行”菜单中选择“启动”、

或者在工具栏中单击“启动”按钮或者按 F5 键时，Visual Basic 需要知道运行哪一个工程。在缺省情况下，Visual Basic 运行添加到工程组中的第一个可执行的 (.exe) 工程。但是，可以指定一个不同的启动工程。

要指定启动工程，请按照以下步骤执行：

1. 在“工程”窗口，选择一个工程。
2. 单击鼠标右键并从上下文菜单中选择“设置为启动”。

Visual Basic 在工程窗口中以粗体显示启动工程的名称。

详细信息 有关多个工程的调试详见随 Visual Basic 专业版和企业版提供的《部件工具指南》中“创建 ActiveX 部件”的第七章“调试、测试和开发部件”。

管理应用程序设置值

在 Microsoft Windows 3.1 和 Windows 的早期版本中，诸如窗口位置、使用的文件以及其它项目等程序设置值是存储在各种 .ini 文件中。在 Windows NT、Windows 95 以及 Windows 的较新版本中，这些程序设置值则是存储在系统注册位置中。

Visual Basic 提供了一个标准的注册位置以存储创建于 Visual Basic 的应用程序的程序信息：

HKEY_CURRENT_USER\Software\VB and VBA Program Settings\appname\section\key

Visual Basic 也提供了四个语句或函数来处理存储在应用程序注册位置的程

序设置值。

函数或语句	描述
GetSetting 函数	检索注册表设置值
SaveSetting 语句	保存或创建注册表设置值
GetAllSettings 函数	返回一个包含多项注册表设置值的数组
DeleteSetting 语句	删除注册表设置值

注意 要查阅注册项，请使用 Windows 95 和 Windows NT 中包含的 Regedit 应用程序。

创建或保存应用程序设置值

可以使用 `SaveSetting` 语句将一个新值保存至存储于应用程序注册位置中的注册表项中。例如，可以向应用程序的主窗体中的 `Form_Unload` 事件添加代码以便在停机时保存设置值，或者向“选项”对话框的 `Form_Unload` 事件添加代码以便更新的首选项。

按以下语法使用 `SaveSetting` 语句：

`SaveSetting appname, section, key, value`

下列代码保存 `Backup` 和 `LastEntry` 表项的新值，这些表项包含在名为

“RegCust”的应用程序的注册位置的启动部分。该代码假设变量 `strDate` 和 `intLastEntry` 包含有新的值。

```
Private Sub Form_Unload(Cancel As Integer)
    SaveSetting "RegCust", "Startup", "Backup", strDate
    SaveSetting "RegCust", "Startup", "LastEntry", _
        intLastEntry
End Sub
```

如果在注册位置的 `Software/Microsoft` 部分没有应用程序“RegCust”的注册项、或者没有这些指定的部分或注册表项，该代码将创建它。

详细信息 请参阅“SaveSetting 语句”。

检索应用程序设置值

可以使用 `GetSetting` 和 `GetAllSettings` 函数检索存储在应用程序注册位置的注册值。例如，应用程序能够检索注册表设置值以便在它被关闭时重新建立其条件。

一次检索一项设置值

如果仅检索一项注册表设置值，通过以下的语法使用 `GetSetting` 函数：

```
GetSetting(appname, section, key[, default])
```

下列代码检索“RegCust”应用程序启动部分的 `LastEntry` 表项的值，并将

该值显示在立即窗口中。

```
Private Sub Form_Load()  
    Dim intLastEntry As Integer  
    intLastEntry = GetSetting("RegCust", "Startup", _  
        "LastEntry", "0")  
    Debug.Print intLastEntry  
End Sub
```

应注意，可以使用可选的参数 `default` 设置，当注册位置中指定的表项没有值时由 `Visual Basic` 返回的值。

一次检索多项设置值

如果检索多项注册表项的值，通过以下的语法使用 `GetAllSettings` 函数：

```
GetAllSettings(appname, section)
```

下列代码检索“`RegCust`”应用程序启动部分的两栏注册表项的值，并将其结果显示在立即窗口中。

```
Private Sub Form_Load()  
    Dim avntSettings As Variant  
    Dim intX As Integer  
    avntSettings = GetAllSettings("RegCust", "Startup")  
    For intX = 0 To UBound(avntSettings, 1)  
        Debug.Print avntSettings(intX, 0), _
```

```
        avntSettings(intX, 1)
    Next intX
End Sub
```

详细信息 请参阅“GetSetting 函数”和“GetAllSettings 函数”。

删除应用程序设置值

可以使用 **DeleteSetting** 语句删除注册表项、注册部分或者应用程序注册位置。例如，当卸载一个应用程序时，也许希望删除该应用程序的所有注册表信息。

通过以下语法使用 **DeleteSetting** 语句：

```
DeleteSetting(appname, section, key)
```

下列代码删除“RegCust”应用程序启动部分中的 **LastEntry** 注册表项。

```
Private Sub cmdDelKey_Click()
    DeleteSetting "RegCust", "StartUp", "LastEntry"
End Sub
```

下列代码删除“RegCust”应用程序注册位置的整个 **StartUp** 部分。

```
Private Sub cmdDelSection_Click()
    DeleteSetting "RegCust", "StartUp"
End Sub
```

下列代码删除“RegCust”应用程序整个注册位置。

```
Private Sub cmdUnInstall_Click()  
    DeleteSetting "RegCust"  
End Sub
```

详细信息 请参阅“DeleteSetting 语句”。

使用条件编译

条件编译允许有选择地编译程序的某些部分。在不同的版本中可以包括应用程序的某些特别的功能，例如，设计一个可以在不同的平台上运行的应用程序，或者改变一个以多种语言发布的应用程序中的日期和货币的显示过滤器。

用于条件编译的结构化代码

要有条件地编译代码的某一部分，应将该部分代码封闭在 `#If...Then` 和 `#EndIf` 语句之间，使用一个布尔常数作为分支检测。要编译被封闭的代码，可将该布尔常数设置为 `-1 (True)`。

例如，要从同一份源代码创建同一个应用程序的法语版本和德语版本，可使用预定义常数 `conFrenchVersion` 和 `conGermanVersion` 将与平台有关的代码部分嵌入 `#If...Then` 语句之间。

```
#If conFrenchVersion Then
```



```
'<适用于法语版本的代码>。  
#ElseIf conGermanVersion then  
    '<适用于德语版本的代码>。  
#Else  
    '<适用于其它版本的代码>。  
#End If
```

如果在编译时常数 `conFrenchVersion` 的数值被设置为 `True`，则适用于法语的代码将被编译。如果常数 `conGermanVersion` 的数值被设置为 `True`，则编译器使用适用于德语版本的代码。

声明条件编译常数

设置条件编译常数有三种方法：在“工程属性”对话框的“制作”选项卡的“条件编译参数”字段设置条件编译常数、在命令行或在代码中设置条件编译常数。

条件编译常数有特别的作用范围，不能从标准代码中访问。如何设置条件编译常数可能取决于希望常数具有怎样的作用范围。

如何设置	范围
“工程属性”对话框	Public 作用于工程中的所有模块
命令行	Public 作用于工程中的所有模块
代码中的 #Const 语句	Private 仅作用于声明该常数的模块

在“工程属性”对话框中设置常数

在创建可执行文件前，从“工程”菜单选择“工程属性”，单击“工程属性”对话框中的“生成”选项卡，在“条件编译参数”字段中键入一个参数，例如 `conFrenchVersion = -1`（如果希望将应用程序编译成法语版本的话）。当程序被编译时，该参数将满足 `#If...Then` 的条件，处于 `#If...Then` 和 `#EndIf` 之间的代码将包含进编译后的程序中。

如果是包含一个或多个 `#ElseIf` 语句的复杂 `#If...Then` 语句，则需要设置另外的常数。可参照以下示例通过冒号间隔来设置多个常数：

```
conFrenchVersion=-1:conANSI=0
```

在命令行设置常数

如果要从命令行启动编译，可以使用 `/d` 开关键入条件编译常数，如下所示：

```
vb.exe /make MyProj.vbp /d conFrenchVersion=-1:conANSI=0
```

在 `/d` 开关与第一个常数之间不需要空格。命令行的常数声明将覆盖在“工程属性”对话框中键入的常数声明，但并不清除它们；在“工程属性”对话框中设置的参数对后续的编译保持有效。

详细信息 请参阅“`#If...Then_#Else` 伪指令”和“`#Const` 语句”。

利用资源文件进行工作

资源文件允许将应用程序特定版本的全部文本和位图收集到一起。这可以包括常数声明、图标、屏幕文本，以及可能在本地化版本之间或在修订版本或特定的配置之间发生变化的其它材料。

向工程中添加资源

可以使用资源编译器外接程序创建资源文件。编译过的资源文件带有 `.res` 扩展名。每个工程只能包含一个资源文件。

资源文件实际上是由一系列独立的字符串、位图或者其它项目组成的，其中每一项都有一个唯一的标识符。标识符可能是 `Long` 类型或 `String` 类型，取决于资源所代表的数据类型。例如，字符串的标识符为 `Long` 型，而位图的标识符则为 `Long` 型或 `String` 型。如想在代码中检索资源，应知道每个资源的标识符。引用资源的函数参数可以使用 `Variant` 数据类型。

要将新资源文件添加到您的工程中，请按照以下步骤执行：

1. 从“工具”菜单中选择“资源编辑器”，将在资源编辑器窗口打开一个空的资源文件。

注意 必须安装资源编辑器外接程序。有关安装外接程序的信息，请参阅“管理工程”中的“使用向导和外接程序”。

2. 在资源编辑器工具栏中选择“保存”按钮以保存资源文件。文件将被添

加到“相关文档”部分下的“工程资源管理器”中。

要将已存在的资源文件添加到您的工程中，请

- 从“工程”菜单中选择“添加新资源”。您的工程中任何已存在的资源文件将被替换。

警告 假如您对已存在的资源文件做任何修改，将可能影响使用此资源文件的其它工程。确保您将文件保存在一个新文件名下。

注意 必须安装资源编辑器外接程序。有关安装外接程序的信息，请参阅“管理工程”中的“使用向导和外接程序”。

详细信息 关于资源文件的详细信息，请参阅第十六章“国际化”中的“使用本地化资源文件”。

注意 Windows 资源文件随 16 位或 32 位应用程序而不同，如果试图将一个 16 位资源文件添加到一个工程中，Visual Basic 将产生一条错误信息。

为了将资源文件添加到工程中，可从“工程”菜单中选择“添加文件”命令，这与将别的文件添加至工程中完全一样。一个工程只能接受一个资源文件，如果试图添加第二个扩展名为 .res 的文件，将产生错误。

在代码中使用资源

Visual Basic 提供了三个函数从资源文件中检索数据以供在代码中使用。

函数	描述
LoadResString	返回一个文本字符串
LoadResPicture	返回一个 Picture 对象，例如位图、图标或光标
LoadResData	返回一个 Byte 数组，用于像 .wav 之类的文件

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的相应函数主题。

利用模板进行工作

Visual Basic 提供了各种模板以方便创建通用的应用程序部件。建立应用程序可以从定制现有的模板开始，而不必从头开始一点一点地创建应用程序的各个片段。也可以通过建立自己的专用模板在多个应用程序中重用定制的部件。

当创建一个新的窗体、模块、控件、属性页或者文档时，可以在“添加对象”对话框中选择模板的图标打开一个现有的模板。例如，Visual Basic 提供了内置的窗体模板以便创建“关于”对话框、“选项”对话框或者快速显示。

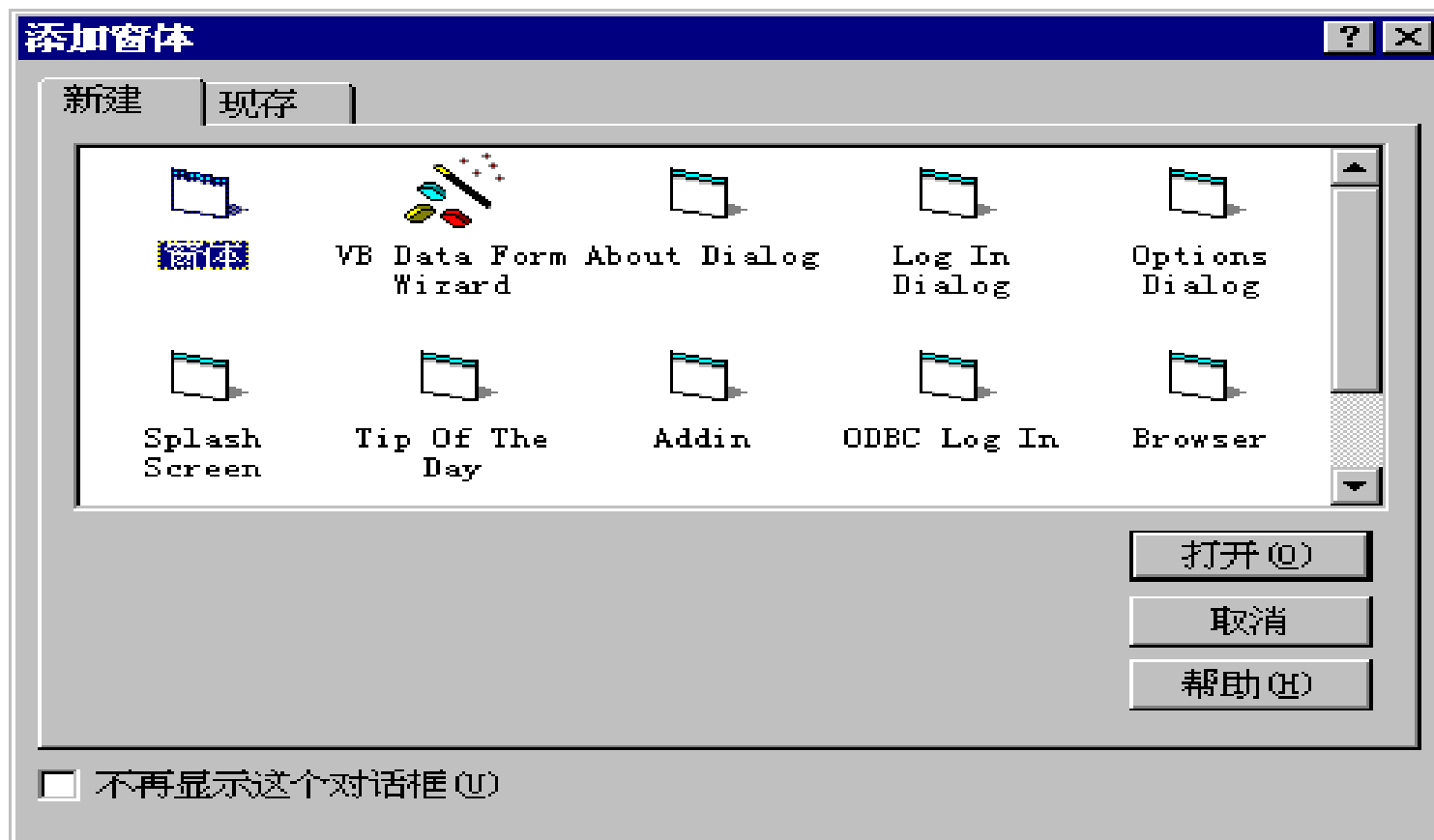


图 8.3 “添加窗体”对话框

当一个模板被打开时，Visual Basic 利用可定制的占位符显示对象。例如，为了创建一个“关于”对话框，可打开“‘关于’对话框”模板，然后将“应用程序标题”、“版本”和“应用程序描述”占位符替换成适合应用程序的具体内容。

为了创建个人专用模板，可保存准备用作模板的对象，然后将它复制到

Visual Basic 下的 Template 目录中适当的子目录下。例如，为了创建一个定制的 MyForm 窗体模板，可以保存一个名为 MyForm 的窗体，然后将 MyForm.frm 文件复制到 \VB\Template\Forms 目录中。当从“工程”菜单中选择“添加窗体”命令时，如图 8.3 所示，Visual Basic 在“添加窗体”对话框中显示 MyForm 模板。

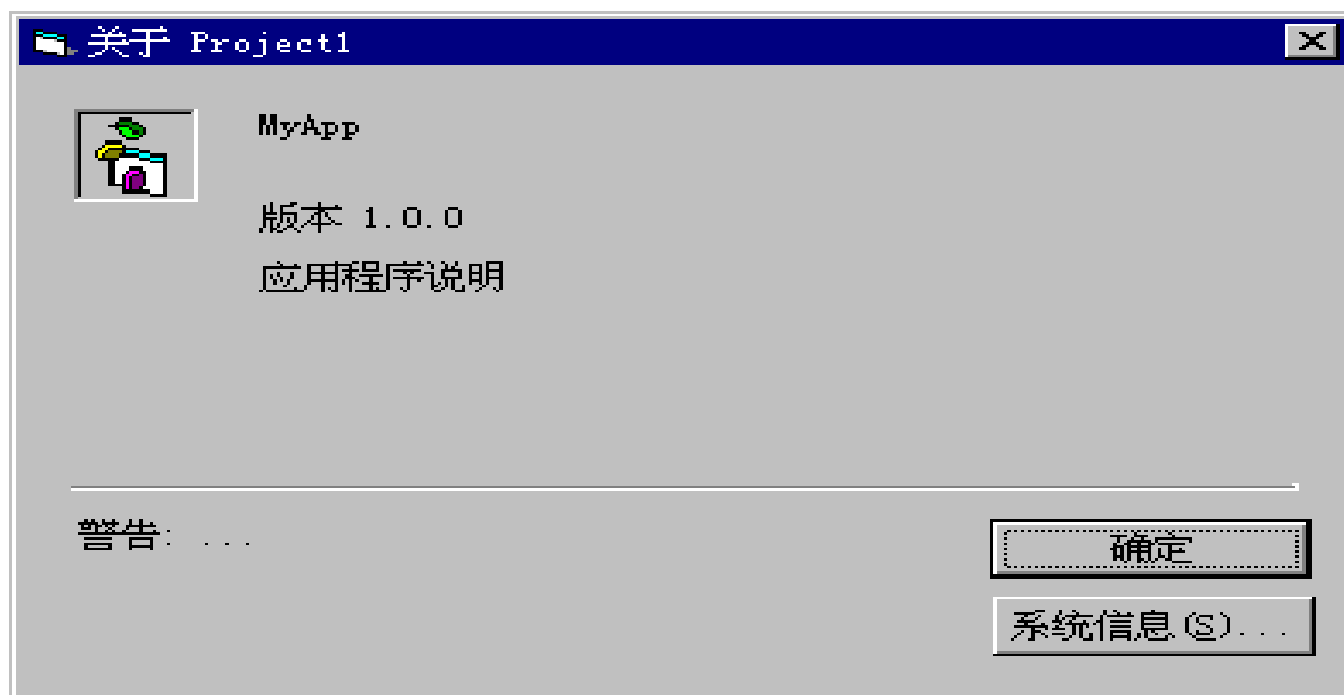


图 8.4 “关于”对话框窗体模板



图 8.5 “选项”对话框中的“环境”选项卡

在“工具”菜单中选择“选项”命令并且清除在“选项”对话框中的“环境”选项卡的“显示模板”选项，可以禁止在“添加对象”对话框中显示模板。例如，为了不显示窗体模板，可在对话框中清除“窗体”选项。

利用命令行开关进行工作

命令行开关提供了一种控制 Visual Basic 如何工作的方式。通过使用命令行开关可以启动一个 Visual Basic 实例并且运行指定的工程、制作一个可执行文件或动态链接库，或者指定一个准备传递给 Command\$ 函数的字符串。

例如，为了运行 MyProject.vbp 工程并随后自动退出，可用以下命令行启动 Visual Basic：

```
c:\Program Files\Microsoft Visual Studio\VB\vb6.exe /runexit MyProj.vbp
```

下表汇总了 Visual Basic 的命令行开关。

开关	描述
/cmd cmdstring	指定一个准备传递给 Command\$ 函数的命令字符串。如使用，它必须是在命令行中的最后一个开关
/d compileconst	指定一个或多个条件编译常数配合 /make or /makedll 开关使用。
/make projectname	将指定的工程制作成一个可执行文件
/makedll projectname	将指定的工程制作成一个动态链接库
/mdi	使用多文档界面 (MDI) 编程环境启动 Visual Basic

续表

<code>/out filename</code>	当与 <code>/make</code> 或 <code>/makedll</code> 开关一起使用时将错误输出到文件中
<code>/run projectname</code>	运行指定的工程
<code>/runexit</code>	运行指定的工程并且随后自动退出
<code>projectname</code>	
<code>/sdi</code>	使用单文档界面 (SDI) 编程环境启动 Visual Basic
<code>/?</code>	列表显示有效的命令行开关

将工程编译成本机代码

如果拥有 Visual Basic 的专业版或企业版，既可以将代码编译成标准的 Visual Basic 伪代码格式，也可以编译成本机代码格式。本机代码编译提供了伪代码编译所没有的关于优化和调试的几个选项。

P-code 或伪代码，是介于 Basic 程序中的高级指令和计算机处理器执行的低级本机代码之间的一种中间步骤。在运行时刻，Visual Basic 将每一句伪代码转换成本机代码。如果将程序直接编译成本机代码，则取消了伪代码这一中间步骤。

可以使用像在 Visual C++ 中提供的调试环境之类的标准的本机代码调试工具来调试编译后的本机代码。也可以使用例如在 Visual C++ 的语言中提供的优化和调试本机代码的选项。例如，可以根据代码运行速度和大小优化

代码。

注意 所有用 Visual Basic 创建的工程都使用运行时 DLL (MSVBVM60.DLL) 服务。在 DLL 所提供的服务当中有您应用程序的启动和关闭代码、窗体和内部控件的功能，以及像 Format 和 CLng 的运行时函数。

用“本机代码”选项来编译工程意味着您写的代码将完整地编译为处理器芯片的本地指令，而不是编译为 p-code。这将大大加速循环和数学计算，并可以加速对 MSVBVM60.DLL 提供的服务程序的调用。然而，这并不能减少对 DLL 的需要。

要将工程编译成本机代码，请按照以下步骤执行：

1. 在“工程”窗口，选择希望编译的工程。
2. 从“工程”菜单选择“工程属性”。
3. 在“工程属性”对话框，单击“编译”选项卡。



图 8.6 在“工程属性”对话框中的“编译”选项卡

4. 选择“编译为本机代码”。

Visual Basic 提供几个选项用于定制和优化可执行文件。例如，当希望创建以代码大小为优化目标的编译代码时，可选择“小代码优化”选项。如使用附加的高级优化选项，单击“高级优化”按钮。

5. 选择所希望的选项，然后单击“确定”。

6. 从“文件”菜单选择“制作 Exe”或者“生成工程组”。

下表描述本机代码优化选项。

选项	描述
禁止别名引用（高级优化）	通知编译器应用程序不使用别名引用。选择本选项将允许编译器实施优化，如将变量存储在寄存器中并且实现循环优化
创建 Symbolic Debug Info	产生 .pdb 文件和 .exe 或 .dll 文件，其中包含允许用 Microsoft Visual C++ 5.0 或其它兼容调试器进行调试的信息
支持 Pentium Pro 不优化	优化代码，使其支持在 Pentium Pro 处理器上运行 取消所有的优化
快速代码优化	通知编译器侧重于运行速度而非代码大小进行优化，使 .exe 和 .dll 文件的速度达到最快
小代码优化	通知编译器侧重于代码大小而非运行速度进行优化，使 .exe 和 .dll 文件的大小达到最小
取消数组边界检查（高级优化）	禁用 Visual Basic 的数组边界检查
取消浮点数错误检查（高级优化）	禁用 Visual Basic 的浮点数错误检查
取消整数溢出检查（高级优化）	禁用 Visual Basic 的整数溢出检查
取消 Pentium FDIV 安全性检查 （高级优化）	禁用对 Pentium 处理器浮点数除法的安全检查

详细信息 关于本机代码选项的详细信息，请参阅附录 C “本机代码编译器开关”。

创建自己的数据类型

不同类型的变量可以组合起来用来创建用户定义的类型（如熟知的 C 编程语言中的 `structs`）。当需要创建单个变量来记录多项相关的信息时，用户定义类型是十分有用的。

可以用 `Type` 语句创建用户定义的类型，该语句必须置于模块的声明部分。用户定义类型可以用适当的关键字声明为 `Private` 或 `Public`，例如：

```
Private Type MyDataType
```

-或-

```
Public Type MyDataType
```

例如，可以创建一个记录有关计算机系统的信息的用户定义类型：

声明（一个标准模块的）。

```
Private Type SystemInfo
```

```
    CPU As Variant
```

```
    Memory As Long
```

```
    VideoColors As Integer
```

```
    Cost As Currency
```

```
    PurchaseDate As Variant
```

```
End Type
```

声明用户定义类型的变量

对同一种用户定义类型，可以声明为局部的、私有的或公用的模块级变量：

```
Dim MySystem As SystemInfo, YourSystem As SystemInfo
```

下表阐明可以在什么地方声明用户定义类型和该类型的变量及其作用范围。

过程 / 模块	将用户定义类型创建成为 ...	用户定义类型的变量可以被声明成 ...
过程	不适用	仅局部
标准模块	私有或公用	私有或公用
窗体模块	仅私有	仅私有
类模块	私有或公用	私有或公用

注意 假如声明了使用 Dim 关键字，标准或类模块中的用户自定义类型将缺省为公有的。假如您希望自定义类型为私有的，确保您用 Private 关键字将其声明为私有。

赋值和检索值

对这种变量的元素赋值和从元素中检索值类似于属性的设置和获取：

```
MySystem.CPU = "486"
```

```
If MySystem.PurchaseDate > #1/1/92# Then
```

如果两个变量都属于同一个用户定义类型，也可以将其中一个变量赋给另

一个变量。这种赋值是将一个变量的所有元素赋给另一个变量的对应元素。

```
YourSystem = MySystem
```

包含数组的用户定义类型

用户定义类型可以包含一个普通数组（固定大小的数组）。例如：

```
Type SystemInfo
```

```
    CPU As Variant
```

```
    Memory As Long
```

```
    DiskDrives(25) As String    '固定大小的数组。
```

```
    VideoColors As Integer
```

```
    Cost As Currency
```

```
    PurchaseDate As Variant
```

```
End Type
```

用户类型中也可以包含动态数组。

```
Type SystemInfo
```

```
    CPU As Variant
```

```
    Memory As Long
```

```
    DiskDrives() As String      '动态数组。
```

```
    VideoColors As Integer
```

```
    Cost As Currency
```

```
    PurchaseDate As Variant
```



```
End Type
```

可以采用访问对象中的属性的方法来访问用户定义类型中数组的值。

```
Dim MySystem As SystemInfo
```

```
ReDim MySystem.DiskDrives(3)
```

```
MySystem.DiskDrives(0) = "1.44 M B"
```

也可以声明属于用户定义类型的数组：

```
Dim AllSystems(100) As SystemInfo
```

可以采用同样的规则访问这一数据结构中的元素。

```
AllSystems(5).CPU = "386 S X"
```

```
AllSystems(5).DiskDrives(2) = "100M S C S I"
```

向过程传递用户定义类型

可以使用用户定义类型传递过程参数。

```
Sub FillSystem (SomeSystem As SystemInfo)
```

```
    SomeSystem.CPU = lstCPU.Text
```

```
    SomeSystem.Memory = txtMemory.Text
```

```
    SomeSystem.Cost = txtCost.Text
```

```
    SomeSystem.PurchaseDate = Now
```

```
End Sub
```

注意 如果要传递窗体或类模块中的用户定义类型，则过程必须是私有的。可以从函数中返回用户定义类型，也可以将属于用户定义类型的变量作为

参数传递到过程中。用户定义类型总是通过引用传递，所以被调用过程可以修改参数并将其返回到调用过程中，这一点在前面的示例中已有说明。

注意 由于用户定义的类型总是由引用来传递的，包含在用户定义类型中的所有数据都将传递到过程并从过程中返回。对于包含大数组的用户定义类型，这将会导致较差的性能，特别是在客户/服务器应用程序中，在客户/服务器应用程序中，过程可能运行在一个远程机器上。在这种情况下，最好只将必要的数据从用户定义类型中抽取并传递。

详细信息 关于通过引用传递参数的详细信息，请参阅第五章“编程基础”中的“向过程传递参数”。

包含对象的用户定义类型

用户定义类型也可以包含对象。

```
Private Type AccountPack
    frmInput as Form
    dbPayRollAccount as Database
End Type
```

提示 因为 Variant 数据类型能存储许多不同类型的数据，所以当希望使用用户定义类型时，在许多情况下也可以使用 Variant 数组。实际上，Variant 数组比用户定义类型更灵活，因为存储在每个元素中的数据的类型可以随时改变，而且还可以将数组定义为动态的，必要时可以改变其大小。但是，Variant 数组使用的内存总是要多于相当的用户定义类型。

嵌套数据结构

嵌套数据结构可以达到所希望的复杂程度。事实上，正如以下的示例所显示的那样，用户定义类型可以包含其他的用户定义类型。为了保证代码的可读性和易于调试，应尽量将定义用户定义数据类型的所有代码限制在一个模块的范围内。

```
Type DriveInfo
```

```
    Type As String
```

```
    Size As Long
```

```
End Type
```

```
Type SystemInfo
```

```
    CPU As Variant
```

```
    Memory As Long
```

```
    DiskDrives(26) As DriveInfo
```

```
    Cost As Currency
```

```
    PurchaseDate As Variant
```

```
End Type
```

```
Dim AllSystems(100) As SystemInfo
```

```
AllSystems(1).DiskDrives(0).Type = "Floppy"
```

通过枚举使用常数集进行工作

枚举类型提供了方便的方法处理有关的常数和使名称与常数数值相关联。例如，可以为与星期日期相关联的一组整数常数声明一个枚举类型，然后在代码中使用星期的名称而不使用其整数数值。

枚举可以通过在标准模块或公用类模块中的声明部分用 `Enum` 语句声明一个枚举类型来创建。枚举类型可以用适当的关键字声明为 `Private` 或 `Public`，例如：

```
Private Enum MyEnum
```

-或-

```
Public Enum MyEnum
```

在缺省情况下，在枚举中的第一个常数被初始化为 0，其后的常数则初始化为比其前面的常数大 1 的数值。例如在下面的枚举 `Days` 中，包含了一个数值为 0 的常数 `Sunday`，数值为 1 的常数 `Monday`，数值为 2 的常数 `Tuesday` 等等。

```
Public Enum Days
```

```
    Sunday
```

```
    Monday
```

```
    Tuesday
```

```
    Wednesday
```

```
    Thursday
```

```
Friday  
Saturday
```

```
End Enum
```

提示 Visual Basic 提供了内置的枚举 `vbDayOfWeek`，包括了与星期的七天相对应的常数。如希望查阅预定义的枚举常数，在代码窗口键入 `vbDayOfWeek`，后跟一个句号，Visual Basic 将自动显示该枚举常数的内容。可以使用赋值语句显式地给枚举中的常数赋值。可以赋值为任何长整数，包括负数。例如，可能希望常数数值小于 0 以便代表出错条件。在以下的枚举中，常数 `Invalid` 被显式地赋值 -1，而常数 `Sunday` 被赋值 0。因为 `Saturday` 是枚举中的第一个元素，所以也被赋值 0。`Monday` 的数值为 1（比 `Sunday` 的数值大 1），`Tuesday` 的数值为 2 等等。

```
Public Enum WorkDays
```

```
    Saturday  
    Sunday = 0  
    Monday  
    Tuesday  
    Wednesday  
    Thursday  
    Friday  
    Invalid = -1
```

```
End Enum
```

注意 Visual Basic 将枚举中的常数数值看作长整数。如果将一个浮点数值

赋给一个枚举中的常数，Visual Basic 会将该数值取整为最接近的长整数。

通过将相关的常数集组织进枚举类型中，就可以在不同的上下文环境中使用同一个常数名称。例如，可以使用在枚举 `Days` 和 `WorkDays` 中的同一个代表星期日期的名称。

当引用单个常数时，为了避免模糊引用，应在常数名称前冠以枚举名。下列代码引用 `Days` 和 `WorkDays` 枚举中的 `Saturday` 常数，并在立即窗口中显示它们的不同的数值。

```
Debug.Print "Days.Saturday = " & Days.Saturday
```

```
Debug.Print "WorkDays.Saturday = " & WorkDays.Saturday
```

当向一个枚举中的常数赋值时，也可以使用另一个枚举中的常数的数值。例如，下述 `WorkDays` 枚举的声明与前述的声明是等同的。

```
Public Enum WorkDays
```

```
    Sunday = 0
```

```
    Monday
```

```
    Tuesday
```

```
    Wednesday
```

```
    Thursday
```

```
    Friday
```

```
    Saturday = Days.Saturday - 6
```

```
    Invalid = -1
```

```
End Enum
```

声明枚举类型后，就可以声明该枚举类型的变量，然后使用该变量存储枚举常数的数值。下列代码使用 `WorkDays` 类型的变量存储与 `WorkDays` 枚举中的常数相关联的整数数值。

```
Dim MyDay As WorkDays
```

```
MyDay = Saturday           'Saturday 的数值为 0。
```

```
If MyDay < Monday Then    'Monday 的数值为 1，  
                           '所以 Visual Basic 显示一个消息框。
```

```
    MsgBox "It's the weekend. Invalid work day!"
```

```
End If
```

注意，当在代码窗口中键入示例中的第二行代码时，Visual Basic 自动在“自动列出成员”列表中显示 `WorkDays` 枚举的常数。

因为常数 `Sunday` 的数值也为 0，所以如果在示例中的第二行用“星期日”替换“星期六”，Visual Basic 也将显示消息框。

```
MyDay = Sunday             'Sunday 的数值也为 0。
```

注意 尽管通常只将枚举常数数值赋给枚举类型的变量，但也可以将任何长整数数值赋给该变量。当对与枚举常数不相关联的变量赋值时，Visual Basic 不会产生错误。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》“Enum 语句”。同时请参阅《部件工具指南》的第二部分“创建 ActiveX 部件”中的第六章“部件设计的一般准则”中的“给部件提供命名的常数”，该书在专业版和企业版中都有提供。

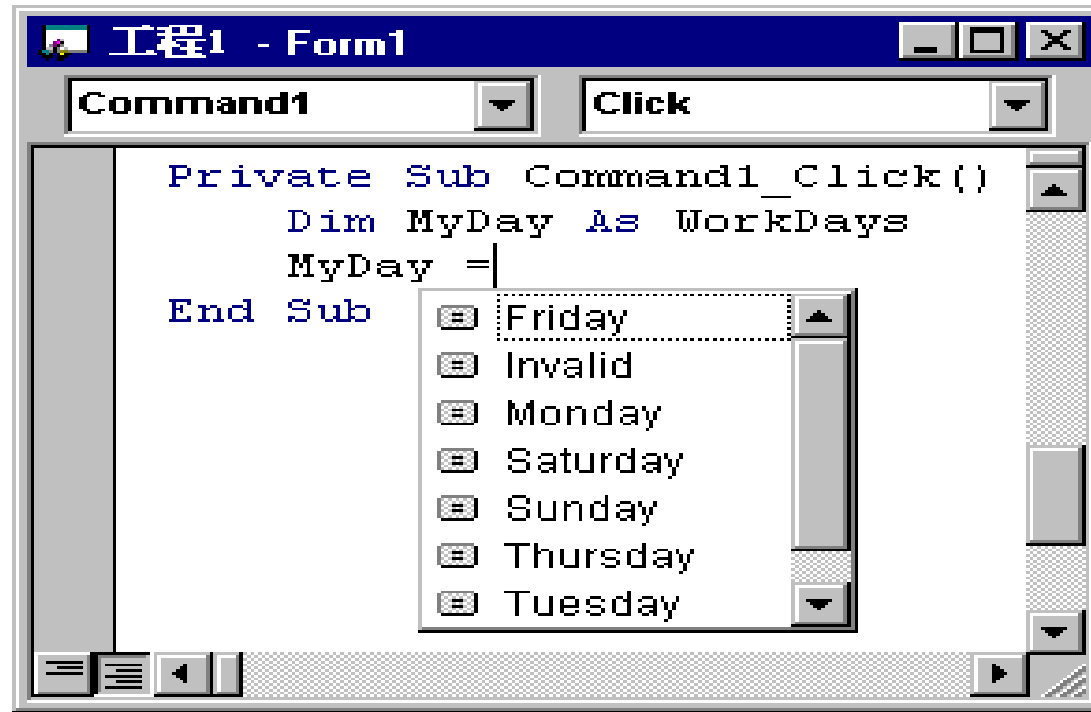


图 8.7 Visual Basic 自动显示枚举的常数

数组的高级功能

尽管数组最通常被用来存储成组的变量，但是在别的一些方面数组也是很有用的。可以将一个数组的内容赋值给另一个数组、创建返回数组的函数，还可以创建返回数组的属性。在许多情况下，这些技术能改进应用程序的性能。

数组赋值

正如可以将一个变量的值赋给另一个变量，例如 `strA = strB`，也可以将一个数组的内容赋给另一个数组。试想，例如，要将一组字节从一个位置复制到另一个位置。可以通过每次复制一个字节来实现，如下所示：

```
Sub ByteCopy(oldCopy() As Byte, newCopy() As Byte)
    Dim i As Integer
    ReDim newCopy (Lbound(oldCopy) To UBound(oldCopy))

    For i = Lbound(oldCopy) To Ubound(oldCopy)
        newCopy(i) = oldCopy(i)
    Next
End Sub
```

一个更有效的方法就是将一个数组赋给另外一个数组：

```
Sub ByteCopy(oldCopy() As Byte, newCopy() As Byte)
    newCopy = oldCopy
End Sub
```

关于变量赋值有一些规则需要铭记。例如，虽然可以将一个声明为整型的变量赋给一个声明为长整型的变量而不会产生任何问题，但是将一个长整型变量赋给一个整型变量很容易导致溢出错误。除了有关数据类型的规则外，数组赋值还要遵从另外一些规则，包括数组维数，每一维的大小，以

- 及数组是固定的还是动态的。
- 对维数和/或数据类型不同的数组赋值的尝试可能成功，也可能失败，这取决于以下几个因素：
- 赋值符左边的数组类型：固定数组 (Dim x(1 to 10) As Integer) 或者动态数组 (Dim x() As Integer)。
 - 赋值符左边数组的维数是否和赋值符右边数组的维数匹配。
 - 赋值符两边数组的每一维的数组元素个数是否匹配。即使数组的声明不同，维数也可能匹配。比如一个数组的每一维元素从 0 开始编号而另一个则从 1 开始，维数也可能匹配。
 - 赋值符两边所有元素的数据类型必须是相容的。这些规则和变量赋值的规则是一致的。

下表显示了这些因素的影响：

左 边 数 组	维 数 是 否 匹 配	元 素 个 数 是 否 匹 配	赋 值 的 结 果
动 态	否	是 或 否	成功。如果需要，左边将重新声明 (ReDim) 使其与右边匹配。
动 态	是	否	成功。如果需要，左边将重新声明 (ReDim) 使其与右边匹配。
动 态	是	是	成功。
固 定	是 或 否	是 或 否	失败并返回一个编译错误。

错误可能发生在编译时，也可能发生在运行时（例如，如果数据类型不能强制转换或赋值试图重新声明 (ReDim) 一个固定大小的数组）。作为程序

员，其任务就是添加错误处理以确保数组在赋值之前是相容的。

从函数返回数组

从一个函数返回一组值是可能实现的。例如，从一个函数返回一组字节,而不必将其先转换为一个字符串然后再转换回来。

从一个函数返回数组与返回单个值略有不同：它要求在函数说明中显示定义其类型，并要求用 `Exit Function` 语句传递返回值。

下面是一个返回字节数组的函数的简单示例：

```
Private Sub Form_Load()  
    Dim i As Integer  
    Dim ReturnArray() As Integer  
  
    i = 1  
    ReturnArray() = ArrayFunction(i)  
End Sub  
  
Public Function ArrayFunction(i As Integer) As Integer()  
    Dim x(2) As Integer  
    x(0) = i  
    x(1) = i + 1
```

'Exit Function 语句传递值。

```
Exit Function x
```

```
End Function
```

在运行以上示例后，ReturnArray() 是一个包含 1 和 2 的两元素数组 ArrayFunction 中数组的值。Exit Function 语句将一个数组作为参数传递；且数组的数据类型必须和函数的数据类型相同（在本例中是整数）。因为这是一个函数调用，传递数组时不必带括号。

注意 尽管可以通过赋值给另一个数组 (ArrayFunction = x()) 来返回一个数组，但出于性能方面的考虑，并不推荐使用这种方法。

必须为返回数组的函数指定一个类型；这个类型可以是 Variant。这样一来，Function X() As Variant() 是有效的而 FunctionX() As () 将失败。

当调用一个返回数组的函数时，用来保存返回值的变量也必须是一个数组，而且其数据类型必须和函数相同，否则将显示一个“类型不匹配”的错误。

详细信息 关于使用数组的详细内容，请参阅第五章“编程基础”中的“数组”部分。关于从属性返回数组的信息，请参阅第九章“使用对象编程”中的“使用属性过程工作”。

用集合替代数组

尽管集合最常用来处理对象，不过也可用它处理任何数据类型。在某些情况下，将项目存储在集合中可能比存储在数组中效率更高。

如果处理较小的、动态的项目集时，可能希望使用集合。下列代码片段演示了如何使用集合来保存和显示 URL 地址列表。

'模块级集合。

```
Public colURLHistory As New Collection
```

'将指定的 URL 地址添加到

'集合中的代码。

```
Private Sub SaveURLHistory(URLAddress As String)
```

```
    colURLHistory.Add URLAddress
```

```
End Sub
```

'在 “立即” 窗口中

'显示 URL 地址的代码列表。

```
Private Sub PrintURLHistory()
```

```
    Dim URLAddress As Variant
```

```
    For Each URLAddress in colURLHistory
```

```
        Debug.Print URLAddress
```

```
    Next URLAddress
```

```
End Sub
```

详细信息 关于使用集合的详细信息，请参阅第九章“用对象编程”中的“用自己的对象编程”，关于数组的详细信息，请参阅第五章“编程基础”中的“数组”。

第九章 用对象编程

对象是 Visual Basic 程序设计的核心。窗体和控件都是对象，数据库也是对象，到处都有对象存在。

如果曾经使用过 Visual Basic，或着使用过本书前五章中的示例，那么就已用对象进行过程序设计了，但是有比至今所知道的对象更多的对象。

在本章中，能接受用户个性的用户定义类型变成类。将会看到，从已定义的类中创建自己的对象非常容易，而且使用对象能简化编码，并增强代码的重用。

目录

以下主题介绍开始使用对象进行程序设计的一些问题。

- Visual Basic 中的对象
- 查找对象信息
- 创建自己的类
- 向类中添加属性和方法
- 向类中添加事件
- 创建数据识别类

- 命名属性、方法和事件
- 多态
- 用自己的对象编程
- 对象模型
- 创建自己的集合类
- ActiveX 设计器

到此并没有结束。下一步是第十章“使用部件的程序设计”中，介绍如何使用 Visual Basic 控制其它应用程序所提供的对象。

示例应用程序 ProgWOb.vbg

本章的一些代码示例来自“使用对象的程序设计”(ProgWOb.vbg)和数据识别类(Dataware.vbp)示例。它们在示例目录中。

Visual Basic 中的对象

Visual Basic 使得对象的使用变得很容易，但是更重要的是，它使在程序性编码和使用对象的程序设计之间进行平稳转换成为可能。

当然，只要一直在使用 Visual Basic，它对使用对象是有帮助的。

与对象有关的术语

下面是一些快速浏览的术语，在讨论 Visual Basic 的对象及其功能时会遇到。如果是从其它程序设计语言或者从使用 ActiveX（从前称为 OLE）术语转到 Visual Basic，那么这些主题对实现转换是有帮助的。

若是刚开始接触对象，可能会感到有点混乱。这是正常的，通过对将要遇到的这些术语的快速浏览，就会对它们如何互相配合有个大概了解。随着在本章其余部分接触更多的有关对象的信息，再回头浏览这些主题，就能将各部分零散的信息形成一个总体。

由此开始

对象是被封装的——也就是说，它同时包含其代码和数据，这比传统的编写代码方法更容易维护。

Visual Basic 对象具有属性、方法和事件。属性是描述对象的数据。方法告诉对象应做的事情。事件是对象所产生的事情，事件发生时可以编写代码进行处理。

在 Visual Basic 中，对象是由类创建的，因此对象被说成是类的一个实例。类定义对象的界面，不管对象是否是公共的和在什么环境下建立的都是这样。类的描述保存在类型库中，而且通过对象浏览器可以查看。

为了使用对象，必须在对象变量中保存其引用。绑定的类型决定使用对象变量访问对象方法的速度。对象变量可以是后期绑定（最慢），或者是事前绑定。事前绑定可以是 DispID 绑定或者 vtable 绑定（最快）。

属性和方法的集合称作界面。Visual Basic 对象的缺省界面是支持所有三种绑定窗体的双重接口。如果对象变量已经声明了类型（即 Dim ... As

`classname`)，它将使用最快的绑定窗体。

除缺省界面外，Visual Basic 对象能够为实现多态提供额外的界面。多态可以操作不同种类的对象，而不用担心每个对象是什么类型。多界面是部件对象模型 (COM) 的一个特征，它允许展开过时的程序，添加新功能而不需拆散老代码。

Visual Basic 中的类也可以是数据识别的。通过直接绑定到一个外部数据源，类可以作为数据使用者，或者通过从一个外部源提供数据，它还可以作为其他对象的数据源。

步入和谐的大厅

如果对所有这些似乎已经很熟悉了，那么可浏览本章的其余部分。如果不熟，也不必担心，所有这些术语的说明都有策略地分布在本文中（并不是很凌乱地出现）。

找出对象所属的类

通用对象变量（即，用 `As Object` 声明的变量）可以保持许多不同类的变量。同样，用 Visual Basic 内建的 `Form` 和 `Control` 类型声明的变量可以包含不同类的窗体和控件。

在使用这些变量时，可能要根据对象的不同类而采取不同的操作。例如，某些对象可能不支持个别的属性和方法。Visual Basic 提供了两种方法：

`TypeOf` 关键字和 `TypeName` 函数。

`TypeOf` 关键字只能用在 `If ... Then ... Else` 语句中。必须在代码中直接包含类名。例如，`If TypeOf MyControl Is CheckBox Then`。

`TypeName` 函数更灵活些。代码中到处可以使用，并且因为它用字符串返回类名，所以可将它与一字符串变量的值相比较。

使用字符串名调用属性和方法

大多数情况下，您可以在设计时发现对象的属性和方法，并且编写代码以处理它们。但是，在少数情况下，您可能不知道某个对象的一些高级属性和方法，或者，您可能只是想要这种灵活性，即允许最终用户在运行时指定属性或执行方法。

例如，请考虑下面的情况，某个客户端应用程序通过传递一个运算符到服务器应用程序来计算由用户输入的表达式。现在，假定您要经常性地向需要新运算符的服务器添加新函数。那么，遗憾的是，在客户端应用程序能够使用新的运算符之前，您需要重新编辑和重新发布它。为了避免这种情况，可以使用 `CallByName` 函数把新运算符当作字符串来传递，而不需要更改应用程序。

`CallByName` 函数允许您使用一个字符串在运行时指定一个属性或方法。

`CallByName` 函数的用法如下：

```
Result = CallByName(Object, ProcedureName, CallType, Arguments())
```

`CallByName` 的第一个参数包含要对其执行动作的对象名。第二个参数 `ProcedureName` 是一个字符串，包含将要调用的方法或属性过程名。`CallType` 参数包含一个常数，代表要调用的过程的类型：方法 (`vbMethod`)、property let (`vbLet`)、property get (`vbGet`)，或 property set (`vbSet`)。最后一个参数是可选的，它包含一个变量数组，数组中包含该过程的参数。

假定您有一个服务器应用程序，`MathServer`，并且它具有一个新的函数 `SquareRoot`。该应用程序中有两个 `TextBox` 控件：`Text1` 包含要计算的表达式；`Text2` 用于输入该函数的名称。要对 `Text1` 中的表达式调用 `SquareRoot` 函数，您可以在一个命令按钮的 `Click` 事件中使用下面的代码：

```
Private Sub Command1_Click()  
    Text1.Text = CallByName(MathServer, Text2.Text, vbMethod, Text1.Text)  
End Sub
```

如果用户在 `Text1` 中输入 "64 / 4"，在 `Text 2` 中输入 "SquareRoot"，则上面的代码将调用 `SquareRoot` 函数（要求一个必需的参数，它是一个包含将要计算的表达式的字符串），并且在 `Text1` 中返回 "4"（16 或 64/4 的平方根）。当然，如果用户在 `Text2` 中输入了一个无效的字符串，或者该字符串包含一个属性名而不是方法名，或者该方法要求附加的必需的参数，那么将会产生一个运行时错误。可以猜到，当您使用 `CallByName` 来预处理这些或其他错误时，需要添加强健的错误处理程序代码。

尽管 `CallByName` 函数在某些情况下可能很有用，但是，还是需要在衡量其实用性时考虑对性能可能的影响——同后期绑定相比，使用 `CallByName`

来调用一个过程只会稍稍慢一点儿。如果您正调用的函数将要被重复调用很多次，例如在一个循环内部，则 `CallByName` 将会对性能产生严重的影响。

对一个对象执行多个动作

经常需要在同一对象中执行多个不同的动作。例如，需要对同一对象设置几个属性。途径之一是使用多条语句。

```
Private Sub Form_Load()  
    Command1.Caption = "OK"  
    Command1.Visible = True  
    Command1.Top = 200  
    Command1.Left = 5000  
    Command1.Enabled = True  
End Sub
```

应注意的是所有这些语句使用同一个对象变量 `Command1`。通过使用 `With...End With` 语句，可使该代码更容易编写、阅读和更有效地运行。

```
Private Sub Form_Load()  
    With Command1  
        .Caption = "OK"  
        .Visible = True
```

```
.Top = 200
.Left = 5000
.Enabled = True
End With
End Sub
```

还可以通过将 `With...End With` 语句置入另一个 `With...End With` 语句内实现嵌套的 `With` 语句。

使用缺省属性

很多对象具有缺省属性。使用缺省属性能够简化代码，因为设置属性值时，不必详细指明该属性。`Value` 是对象缺省属性，这两个语句是等价的：

```
object = 20
```

和

```
object.Value = 20
```

为了说明它们是如何工作的，在窗体上绘制命令按钮和文本框。将下列语句添加到命令按钮的 `Click` 事件中：

```
Text1 = "hello"
```

运行该应用程序，并单击命令按钮。因为 `Text` 是文本框的缺省属性，所以文本框将显示文本“hello”。

通过对象变量使用缺省属性

当对象的引用被保存在对象变量中时，仍然可以使用缺省属性。下面的代码段说明了这一点。

```
Private Sub Command1_Click()  
    Dim obj As Object  
    '把对 Text1 的引用放入对象变量。  
    Set obj = Text1  
    '设置缺省属性 (Text) 的值。  
    obj = "hello"  
End Sub
```

在上面的代码中，`obj = "hello"` 和 `obj.Text = "hello"` 是完全相同的。

通过 Variants 使用缺省属性

当对象引用保存在 `Variants` 类型的变量中，而不是在对象变量中时，访问缺省属性是不同的。这是因为 `Variants` 能包含多种不同类型的数据。

例如，能够使用 `Variants` 中的引用读取 `Text1` 的缺省属性，但是试图将字符串“`goodbye`”赋给缺省属性的做法是行不通的。相反，它将用该字符串代替该对象引用，并改变 `Variants` 的类型。

为了说明这一点，以前面示例为例，在命令按钮的 `Click` 事件中输入下面的代码：

```

Private Sub Command1_Click()
    Dim vnt As Variant
    '设置缺省属性 (Text) 为 “hello”。
    Text1 = "hello"
    '把对 Text1 的引用放入 Variants 。
    Set vnt = Text1
    '显示 Text1 的缺省属性，并且给出
    '    Variants 包含对象引用。
    MsgBox vnt, , "IsObject? " & IsObject(vnt)
    '试图设置 Text1 的缺省属性。
    vnt = "goodbye"
    MsgBox vnt, , "IsObject? " & IsObject(vnt)
End Sub

```

当运行该应用程序并单击命令按钮时，首先出现的信息框显示 Text1 缺省属性的当前值 “hello”，它通过查看 Text1 可得到证实。信息框的标题证明 Variant 包含有一个对象引用——即对 Text1 的引用。

当单击信息框中的“确定”按钮时，“goodbye”赋给该 Variants，并破坏了 Text1 的引用。接着显示另一信息框，并显示该 Variants 的内容——正如所看到的那样，它与 Text1.Text 的当前值不匹配。

信息框的标题证明，Variants 不再包含对象引用——它现在包含字符串 “goodbye”。

详细信息 有关 Variants 和其它数据类型的详细信息，请参阅第五章“编

程基础”中的“变量、常数和数据类型概述”。

使用对象和 Variants 的其它内容将在“Visual Basic 集合对象”中讨论。

创建对象数组

正像声明和使用任何数据类型的数组那样，你能够声明和使用对象类型的数组。这些数组既可以是固定大小的，也可以是动态的。

窗体变量数组

与声明其它任何类型数组的方法相同，可以使用 Private、Dim、ReDim、Static 或 Public 声明窗体数组。如果数组声明使用了 New 关键字，则当使用该数组的元素时，Visual Basic 将自动为数组中每个元素创建一个新窗体实例。

```
Private Sub Command1_Click ()  
    Dim intX As Integer  
    Dim frmNew(1 To 5) As New Form1  
    For intX = 1 To 5  
        frmNew(intX).Show  
        frmNew(intX).WindowState = vbMinimized  
        '为了不让它们第一次以正常大小出现  
        '创建最小化窗体，可改变
```


上面两行的顺序。

Next

End Sub

按命令按钮执行上述代码，将建立五个最小化的 Form1 实例。

注意 如果查看任务栏，可看到六个 Form1。额外的 Form1 实例不是最小化的

——它是刚开始使用的那个。

控件变量数组

与声明任何其它类型数组的方法相同，可以使用 Private、Dim、ReDim、Static 或 Public 声明控件数组。然而与窗体数组不同，控件数组不能使用 New 关键字声明。例如，可以声明数组为特定的控件类型：

```
ReDim ActiveImages(10) As Image
```

当声明数组为特定控件类型时，只能将该类型的控件赋给数组。例如，在前面的声明中，只能将图像控件赋给该数组——但是这些图像控件可以来自不同的窗体。

将其与内置 Controls 集合进行比较，内置 Controls 集合能包含很多不同类型的控件——所有这些控件必须在同一窗体中。

另外，可以声明类属控件变量数组。例如，若要跟踪拖放到某一特定控件中的每个控件，而且不允许任何控件被拖放多次，则可通过对控件变量的动态数组的管理来做到这点，只要让该动态数组包含每个被拖放控件的引

用即可。

```
Private Sub List1_DragDrop(Source As VB.Control, _
    X As Single, Y As Single)
    Dim intX As Integer
    Static intSize As Integer
    Static ctlDropped() As Control
    For intX = 1 To intSize
        '如果数组中有被拖放的控件，则它
        '已经被拖放到这儿一次
        If ctlDropped(intX) Is Source Then
            Beep
            Exit Sub
        End If
    Next
    '扩展数组。
    intSize = intSize + 1
    ReDim Preserve ctlDropped(intSize)
    '保存被拖放控件的引用。
    Set ctlDropped(intSize) = Source
    '给列表框添加控件名。
    List1.AddItem Source.Name
End Sub
```

该示例使用 Is 操作符将控件数组中的变量和控件参数进行比较。Is 操作符可用于测试 Visual Basic 对象引用的一致性：如果两个不同引用的比较结果为同一对象，Is 操作符返回 True。

该示例还使用 Set 语句将 Source 参数中的对象引用赋予数组中的元素。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》中的“Is 操作符”。

在第五章“编程基础”的“数组”和“动态数组”中介绍数组。

关于跟踪对象的更简易的方法，请参阅本章后面的“创建对象集合”。

创建对象集合

集合提供了跟踪对象很有用的方法。与数组不同，当增加和删除集合对象的成员时，它的大小不会改变。

例如，若要跟踪拖放到某一特定控件中的每个控件，并且不允许多次拖放任何控件。通过对集合进行管理可做到这点，该集合包含每个被拖放控件的引用：

```
Private Sub List1_DragDrop(Source As VB.Control, _  
X As Single, Y As Single)  
    Dim vnt As Variant  
    Static colDroppedControls As New Collection  
    For Each vnt In colDroppedControls  
        '如果集合中有被拖放的控件，则  
        '它已经被拖放到这儿一次。
```

```
        If vnt Is Source Then
            Beep
            Exit Sub
        End If
    Next
    '保存被拖放控件的引用。
    colDroppedControls.Add Source
    '在列表框中添加控件名。
    List1.AddItem Source.Name
End Sub
```

该示例使用 `Is` 操作符将 `colDroppedControls` 集合中的对象引用和包含被拖放控件引用的事件参数进行比较。`Is` 操作符可用于测试 Visual Basic 对象引用的一致性：如果两个不同引用的比较结果为同一对象，`Is` 操作符将返回 `True`。

该示例还使用集合对象中的 `Add` 方法将对被拖放控件的引用放到集合中。与数组不同，集合本身也是对象。变量 `colDroppedControls` 被声明为 `As New`，因此在代码中变量第一次被引用时将创建该集合类的实例。由于变量被声明为 `Static`，因此当事件过程结束时，该集合对象不会消失。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`Is` 操作符”。

本章后面的“Visual Basic 集合对象”中将讨论 `Collection` 对象的属性和方法。

有关上述代码与需要使用数组的代码的比较，请参阅本章前面的“创建对

象数组”。

要了解如何在自己集合类中包括集合对象，建立更强健的集合，请参阅本章后面的“创建自己的集合类”。

本章前面的“Visual Basic 中的对象”讨论如何建立和破坏对象。

Visual Basic 集合对象

集合是将一系列相关的项构成组的一种方法。Visual Basic 中集合可用于跟踪很多事情，例如程序中加载的窗体（窗体集合），或者在窗体中的所有控件（控件集合）。

Visual Basic 提供的一般类属 `Collection` 类用来定义自己的集合。需要多少集合对象，就可以建立多少 `Collection` 对象——即 `Collection` 类的实例。还可使用集合对象作为自己的集合类和对象模型的基础，它将在本章后面的“创建自己的集合类”和“对象模型”中讨论。

例如，集合是跟踪多窗体的最好方法。在第六章“创建用户界面”中的“多文档界面 (MDI) 应用程序”里讨论的应用程序中，用户可以打开任何数目的文档窗口。以下代码段演示如何使用集合对象的 `Add` 方法，对用户建立的 MDI 子窗体列表进行管理。该代码假定已经存在一个名为 `mdiDocument` 的窗体，其 `MDIChild` 属性设置为 `True`。

父 `MDIForm` 中的模块级集合。

```
Public colDocuments As New Collection
```

'建立新的 MDI 子文档窗体的代码。

```
Private Sub mnuFileNew()  
    Dim f As New mdiDocument  
    Static intDocumentNumber As Integer  
    intDocumentNumber = intDocumentNumber + 1  
    '下面的语句创建窗体。  
    f.Caption = "Document" & intDocumentNumber  
    '给集合添加对象引用。  
    colDocuments.Add f  
    f.Show  
End Sub
```

`colDocuments` 的作用像是内置窗体集合的一个子集，它只包含窗体 `mdiDocument` 的一些实例。每添加一个新窗体，自动调节集合的大小。可使用 `For Each ... Next` 在集合内进行迭代。如果赋予窗体一个可以检索的键，那么可提供一文本字符串作为 `Add` 方法的第二个参数，如本节后面部分所述。

在变量 `colDocuments` 的声明中，`New` 关键字导致当该变量第一次被引用时，在代码中创建集合对象。因为集合是类，而不是数据类型，所以必须建立集合实例，并且在变量中记录对该实例（对象）的引用。

象其它任何对象一样，当包含其引用的最后一个变量被设置成 `Nothing` 或不可见时，`Collection` 对象将被撤销。它包含的所有对象引用将被释放。因此，变量 `colDocuments` 在父 `MDIForm` 中声明，于是它将存在于程序存

活期中的整个过程。

注意 如果使用集合来跟踪窗体,当窗体被卸载之后,应使用集合的 `Remove` 方法从集合中删除该对象引用。只要对该窗体的引用仍然存在,就不能收回窗体使用的内存,并且 `Collection` 对象保存的引用象对象变量中的引用一样完好。

集合对象的构成

`Collection` 对象将每一项存储于 `Variants` 对象中。于是,能够添加到 `Collection` 对象里的内容列表就和能够存储到 `Variants` 中的内容列表是相同的。这包括标准数据类型、对象和数组——但不包括用户定义类型。

不管 `Variants` 中存储的是什麼,它都占 16 字节,因此使用 `Collection` 对象不如使用数组的效率。然而,不能 `ReDim` 一个 `Collection` 对象,`ReDim` `Collection` 对象能够使代码更清楚、更容易维护。此外,集合对象按键能进行快速的查找,而数组却不能。

注意 准确一点说,即使是把数据存储在其他地方,`Variants` 也总是只占 16 个字节。例如,如果把字符串或数组赋给 `Variants`,`Variants` 将包含一个指向字符串和数组数据备份的指针。在 32 位系统中,指针只使用了 `Variants` 的 4 个字节,而且 `Variants` 中实际上不存在数据。若要存储一个对象,`Variants` 将包含该对象引用,就像对象变量所做的一样。对字符串和数组来说,也只使用 `Variants` 的 4 个字节。数字数据类型存储在 `Variants` 中。不管什么样的数据类型,`Variants`

仍然占 16 个字节。

除了 Variants 大小之外，还有很多情况需要使用 Collection 对象来存储上面列出的所有数据类型。应该知道进行权衡：使用 Collection 对象能够编写清楚且容易维护的代码——所花费的仅是将一些项存储在 Variant 中。

集合对象的属性和方法

每个集合对象都有属性和方法，使用它们能够插入、删除和检索该集合中的项。

属性或方法	描述
Add 方法	给集合添加项。
Count 属性	返回集合中项的数目，只读。
Item 方法	通过索引或关键字，返回项。
Remove 方法	通过索引或关键字，从集合中删除项。

这些属性和方法仅仅是集合最基本的功能。例如，要确保集合只包含一种对象，Add 方法并不能检查将要添加到集合里的对象类型。通过建立自己的集合类能够提供更强健的功能，以及额外的属性、方法和事件，正如本章后面的“创建自己的集合类”中所述。

集合中的增加、删除和检索的基本功能需要关键字和索引。关键字是一个

`String` 的值。它可以是能转换成字符串的名字、驾驶执照号、社会保险号或者简单的整数。`Add` 方法允许将关键字与项相关联，如本节后面部分所述。

索引是长整型，介于 1 和集合中项的数目之间。使用 `before` 和 `after` 命名的参数，可以控制项索引的初始值，但是随着其它项的增加和删除，其值会发生改变。

注意 索引从 1 开始的集合称为基于 1，如“Visual Basic 中的集合”中所述。

使用索引能够在进行集合的项中迭代。例如，假定变量 `mcolEmployees` 中包含对集合对象的一个引用，下列代码使用两种方法把 `Employee` 对象集合中所有雇员的薪水提高 10%。

```
Dim lngCt As Long
For lngCt = 1 To mcolEmployees.Count
    mcolEmployees(lngCt).Rate = _
        mcolEmployees(lngCt).Rate * 1.1
Next
```

```
Dim emp As Employee
For Each emp In mcolEmployees
    emp.Rate = emp.Rate * 1.1
Next
```

提示 为改善性能，应使用 `For Each` 对 `Collection` 对象中的项进行迭代。

使用 `For Each` 迭代要比使用索引快很多倍。这并不是对所有集合都正确——它取决于集合内部存储数据的方法。

给集合添加项

`Add` 方法用于将项添加到集合。语法是：

```
Sub Add (item As Variant [, key As Variant] [, before As Variant]  
[, after As Variant])
```

例如，使用工作订单 `ID` 属性作为关键字，将工作订单对象添加到工作订单集合里，代码可编写为：

```
colWorkOrders.Add woNew, woNew.ID
```

这里假定 `ID` 属性是字符串。如果属性是数字（例如 `Long`），则应使用 `CStr` 函数将它转换成关键字所要求的字符串值：

```
colWorkOrders.Add woNew, CStr(woNew.ID)
```

`Add` 方法支持命名的参数。为了添加一项作为第三个元素，代码可编写为：

```
colWorkOrders.Add woNew, woNew.ID, after:=2
```

可以使用 `before` 和 `after` 命名的参数管理有序的对象集合。例如，`before:=1` 将在集合的开始插入一项，因为集合对象是基于 1 的。

从集合中删除项

`Remove` 方法用于从集合中删除项。语法是：

object.Remove index

index 参数可以是所删除项在集合中的位置，或者是该项的关键字。如果集合中第三个元素的关键字是“W017493”，则可使用这两条语句中的任何一条删除它：

```
colWorkOrders.Remove 3
```

- 或 -

```
colWorkOrders.Remove "W017493"
```

从集合中检索项

Item 方法用于从集合中检索特定项。语法是：

```
[Set] variable = object.Item(index)
```

和 **Remove** 方法一样，**index** 可以检索项在集合中的位置，或者是该项的关键字。使用 **Remove** 方法中的相同示例，这两条语句中的任一条都可检索集合中的第三个元素：

```
Set woCurrent = colWorkOrders.Item(3)
```

- 或 -

```
Set woCurrent = colWorkOrders.Item("W017493")
```

如果使用整数作为关键字，在将它们传递到 **Item** 或 **Remove** 方法之前，必须使用 **CStr** 函数将其转换成字符串。通常集合对象都假定整数是索引。

提示 **Collection** 对象不能决定所传递的值是索引还是键。如果想把值解释为键，并且包含该值的变量是除 **String** 以外的任何值，可以使用 **CStr** 函

数进行转换。如果想把值解释为索引，并且包含该值的变量不是整型数据类型中的一种，使用 `CLng` 进行转换。

Item 是缺省方法

对于集合对象来说，`Item` 方法是缺省方法，因此当访问集合中的项时，可忽略它。所以前面的代码示例也可以写成：

```
Set woCurrent = colWorkOrders(3)
```

- 或 -

```
Set woCurrent = colWorkOrders("W 017493")
```

重点 当添加和删除集合对象元素时，集合对象会自动地维护其数字索引编号。因此给定元素的数字索引随之改变。不能在程序中存储数字索引值，并用它检索同一个元素。为达到该目的，应使用键。

使用 **Item** 方法调用属性和方法

为了使用对象引用，不能从集合中检索该对象引用，并把它放到对象变量中。当对象仍然在集合里时能够使用其引用。

例如，假设上述代码中的 `WorkOrder` 对象具有 `Priority` 属性。下面的语句都可用于设置工作定单的优先级：

```
colWorkOrders.Item("W 017493").Priority = 3
```

```
colWorkOrders("W 017493").Priority = 3
```

可行的原因是 `Visual Basic` 要从左到右计算表达式。当遇到 `Item` 方法时—

—显式的或隐式的——Visual Basic 为指定的项（此时是键为 W017493 的 WorkOrder 对象）取得一个引用，并使用该引用计算该行的其余部分。

提示 如果想调用集合中对象的多个属性或方法，首先将该对象引用复制到一个强类型的对象变量中。把引用放到一个强类型的对象变量中，然后使用，比使用集合里对象的引用要快（例如，Dim woCurrent As WorkOrder），因为 Collection 对象是把项存储在 Variants 中。Variants 中的对象引用总是最后绑定。

详细信息 对很多一般的程序设计任务来说，集合对象也是数组的很有用的可选方案。请参阅第八章“再论编程”中的“用集合替代数组”。

Visual Basic 中的集合

什么是集合？在“Visual Basic 集合对象”中，集合定义为一种把有相关的对象分组的方法。这种定义留下很多解释余地，与其说它是一种定义还不如说它是一个概念更合适。

实际上，当比较集合时就会发现，Visual Basic 所提供的各种集合之间都有很大的差别。例如，下列代码会引起错误：

```
Dim col As Collection
```

```
Set col = Forms          '错误！
```

这是怎么回事？Forms 集合是集合，变量 col 也声明为 Collection，为什么不能将指向 Forms 的引用赋给变量 col？

其原因是 `Collection` 类和 `Forms` 集合都不是多态的，即它们不能相互交换，因为是从各自的代码库开发的。它们没有相同的方法，没有用相同的方法存储对象引用，或者没有使用相同种类的索引值。这使得 `Collection` 类名似乎是多余的，因为它仅仅代表许多可能实现集合方法的一种。该主题探讨一些实现方法的差别，后面将会遇到这些差别。

基于 0 和基于 1 的集合

集合是基于 0 还是基于 1，要取决于开始索引是哪一个。可以猜想，前者的意思是集合中第一项的索引为 0，后者的意思是集合中第一项的索引为 1。基于 0 的集合示例是 `Forms` 和 `Controls` 集合。基于 1 的集合的示例是集合对象。

`Visual Basic` 中旧的集合大多是基于 0 的，而新添加的集合大多基于 1。基于 1 的集合使用时更加直观，因为索引的范围是从 1 到 `Count`，其中，`Count` 是能返回集合中项数目的属性。

对应基于 0 的集合索引来说，其范围则是从 0 到 `Count-1`。

索引和键值

正像 `Visual Basic` 中的 `Collection` 对象那样，`Visual Basic` 中很多集合可以使用数字索引或字符串键来访问其中的项（然而，`Visual Basic` 中的 `Collection` 对象不用指定键就能添加项）。

相反，`Forms` 集合只能使用数字索引来访问。这是因为没有唯一的字符串

值与窗体关联。例如，多个窗体可以有相同的标题，或者加载的多个窗体有相同的 `Name` 属性。

添加和删除项

在能否向集合中添加项这点上，集合之间也是有区别的，如果可以，这些项是如何添加到集合中的。例如，不能使用 `Visual Basic` 代码向 `Printers` 集合添加打印机。

因为 `Collection` 对象是一般性程序设计工具，它比其它集合更加灵活。它具有 `Add` 方法可以将项添加到集合里，也具有 `Remove` 方法可以将项从集合中删除。

与之形成对比的是，`Forms` 集合获取窗体的唯一方法是加载窗体。如果使用 `New` 操作符建立窗体，或者通过引用一个声明为 `As New` 的变量建立窗体，那么该窗体是不能被添加到窗体集合里的，除非使用 `Load` 语句进行加载。

`Forms` 和控件集合没有 `Remove` 方法。可以使用 `Load` 和 `Unload` 语句间接的向这些集合中添加和从这些集合中删除窗体和控件。

窗体集合中的成员

正如上面所述，窗体在加载之前不能添加到 `Forms` 集合中。因此 `Forms` 集合的最准确的说明是：它包含的是程序中当前加载的所有窗体。

即使它不完全准确。如果工程中使用 `Microsoft Forms`（为了与 `Microsoft`

Office 兼容)，就会发现这些窗体都在一个名为 `UserForms` 的单独集合里。因此该 `Forms` 集合所包含的就是程序中当前加载的所有 `Visual Basic` 窗体。

而 `Collection` 类十分准确指定的内容是：能够存储到 `Variants` 里的任何东西。于是集合对象能包含对象或者整数，但不能包含用户定义类型。

遗憾的是，该说明的覆盖范围很广——集合类给定的一个实例能够存储任何数据类型、数组和对象的混合组合。

提示 正如“创建自己的集合类”中所述，建立自己的集合类的最重要原因是能够控制自己的集合中的内容——称为类型安全性概念。

集合的枚举

可以使用 `For Each ... Next` 对集合中的项枚举，而不用担心集合是基于 0 还是基于 1。当然，这很难作为集合的一个定义特性，因为 `Visual Basic` 也允许使用 `For Each ... Next` 对数组中的项枚举。

使 `For Each ... Next` 能工作的是一个称为 `enumerator` 的很小的对象。枚举算子记录集合中项的位置，并在需要时返回下一项。

枚举数组时，在运行中 `Visual Basic` 创建一个数组枚举算子对象。集合具有自己的枚举算子对象，它也是需要时创建的。

枚举算子不会跳过项

`Visual Basic` 中集合的枚举算子不会跳过项。例如，假设枚举一个包含 "A"、

"B" 和 "C" 的集合，并且进行枚举时删除了 "B"，则在操作时 Visual Basic 不会跳到 "C" 上。

枚举算子可能不捕捉添加的项

当枚举集合时，如果给集合添加了项，一些枚举算子将能包含添加的项，而一些枚举算子则不会。例如，枚举 `Forms` 集合时就不枚举刚加载的任何窗体。

如果将它们添加到集合末尾，则在枚举集合时，对象就能枚举添加的项。因此下列循环永远不会终止（直到按 `CTRL+BREAK` 组合键）：

```
Dim col As New Collection
Dim vnt As Variant
col.Add "Endless"
col.Add "Endless"
For Each vnt In col
    MsgBox vnt
    col.Add "Endless"
Next
```

另一方面，添加到集合开始的项不会包含在枚举里。

```
Dim col As New Collection
Dim vnt As Variant
col.Add "Will be enumerated"
For Each vnt In col
```

```
MsgBox vnt
```

'在开始处添加项。

```
col.Add "Won't be enumerated", Before:=1
```

```
Next
```

为什么需要枚举算子？

每次 For Each ... Next 开始时，通过发出一新枚举算子，集合允许嵌套的枚举。例如，假定在变量 mcolStrings 中有一个 Collection 对象的引用，并且该集合只包含字符串。下列代码打印两个不同字符串的所有组合：

```
Dim vnt1 As Variant
```

```
Dim vnt2 As Variant
```

```
For Each vnt1 In mcolStrings
```

```
    For Each vnt2 In mcolStrings
```

```
        If vnt1 < > vnt2 Then
```

```
            Debug.Print vnt1 & " " & vnt2
```

```
        End If
```

```
    Next
```

```
Next
```

详细信息 请参阅本章后面的“创建自己的集合类”。

查找对象信息

对象浏览器是基于类型库的，类型库是包含属性、方法、事件、命名常数以及其它许多方面信息在内的、类的详细描述的资源。

Visual Basic 为创建的类建立类型库信息，为其包含的对象提供类型库，并且允许访问其它应用程序提供的类型库。

可以使用对象浏览器显示工程和库中的有效的类，包括自己定义的类。用这些类所创建的对象都拥有同样的成员——属性、方法和事件——这些都可以在对象浏览器中看到。

要显示对象浏览器，请

- 从“查看”菜单选择“对象浏览器”。

- 或 -

按 F2 键。

- 或 -

在工具栏上单击“对象浏览器”。

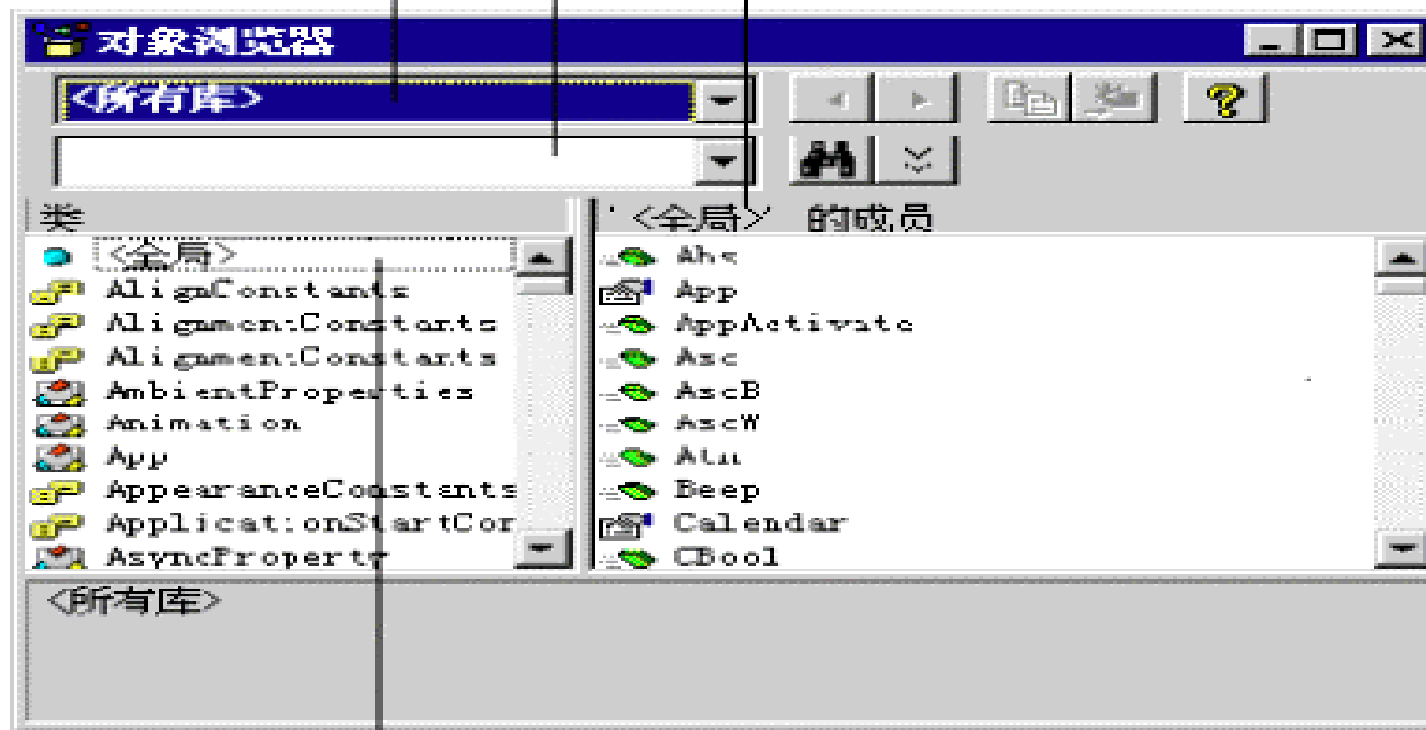
按照缺省规定，对象浏览器不能接连接其它窗口。这样可以使用 CTRL+TAB 在对象浏览器和代码窗口之间切换。也可以通过在对象浏览器中单击右键打开上下文菜单，并单击“可连接的”改变这种情况。

注意 当对象浏览器处于可以连接状态时，不能使用 CTRL+TAB 从代码窗口切换至对象浏览器。

工程/库框允许选择单个库或工程或浏览全部库和工程。

搜索文字框可以查对象及其成员

成员列表显示在其它列表中选定类的属性方法和事件（成员）。



列表中的<全局>项是占位符，可查看选定工程或库的全局对象、集合、函数和语句。

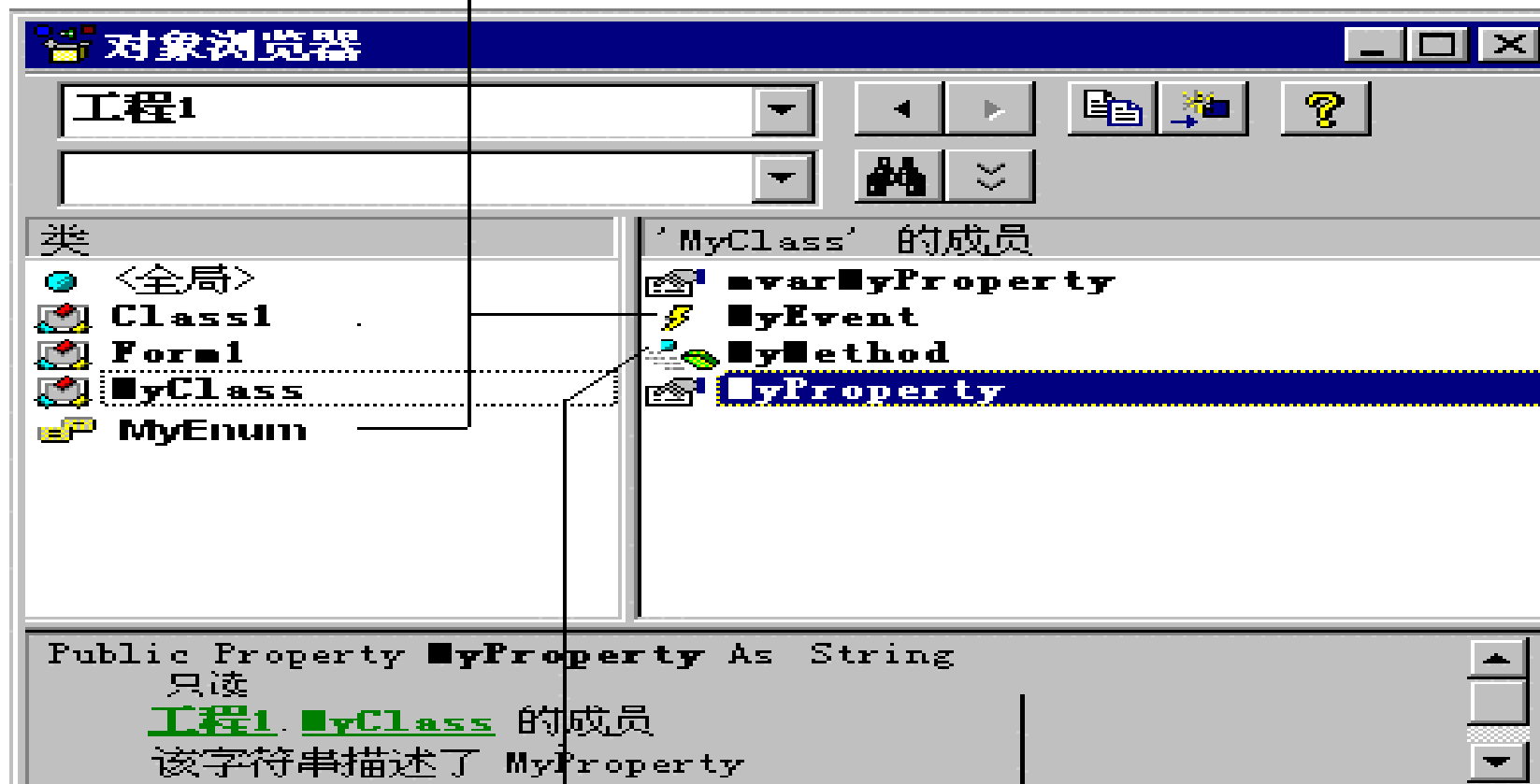
图 9.1 对象浏览器

对象浏览器的内容

对象浏览器用三级层次显示信息，如图 9.2 所示。使用“工程/库”框从顶部开始，可在包括自己 Visual Basic 的工程在内的、已有的工程和库中进行选择。

- 在“类”列表中单击类，然后在底部的描述面板中查看其中的描述。类的属性、方法、事件和常数将在右边的“成员”列表中显示。可查看的类是从“工程/库”框中所选择的工程或库中抽出来的，如果选择了“所有库”，则是从所有的工程和库中抽出来的。
- 可以在“成员”列表中单击成员，查看所选择的类成员的参数和返回值。对象浏览器底部的描述面板将显示对应的信息。
- 可以单击描述面板中的库名或对象名跳转至包含该成员的库或对象。单击对象浏览器顶部的“向后”按钮可以返回至先前的位置。

属性、方法、
事件和枚举有
各自的符号。



小的兰色球体确
定缺省属性或方
法。

描述窗格显示属性、方法的类型
和参数以及对它们的描述。仅是
提供属性 Get, MyProperty 为只读
字符串属性。

图 9.2 在对象浏览器中查看类的成员

提示 当处于“类”列表或“成员”列表中时，键入一个名字的首字母将移至以该字母开头的下一个名字处。

控制对象浏览器的内容

在图 9.3 所示的上下文菜单中，提供了对象浏览器中的“复制”和“查看定义”按钮的替代。也允许打开“引用”对话框，如果选择了一个类或成员，还允许查看所选项目的属性。利用这些菜单项可以为自己的对象设置描述，正如“添加对象的描述”所描述的那样。

在对象浏览器中单击右键可调出上下文菜单。除了前述的功能外，上下文菜单还能控制“类”列表和“成员”列表的内容。

- 当选择了“组成员”时，对象的所有属性都组织在一起，所有的方法也组织在一起，等等。当没有选择“组成员”时，“成员”列表是按字母顺序排列的。
- 当选择了“显示隐含成员”时，在“类”列表和“成员”列表中将显示类型库中被标记为隐藏的那些信息。通常不必参阅这些信息。隐藏的成员以浅灰色显示。

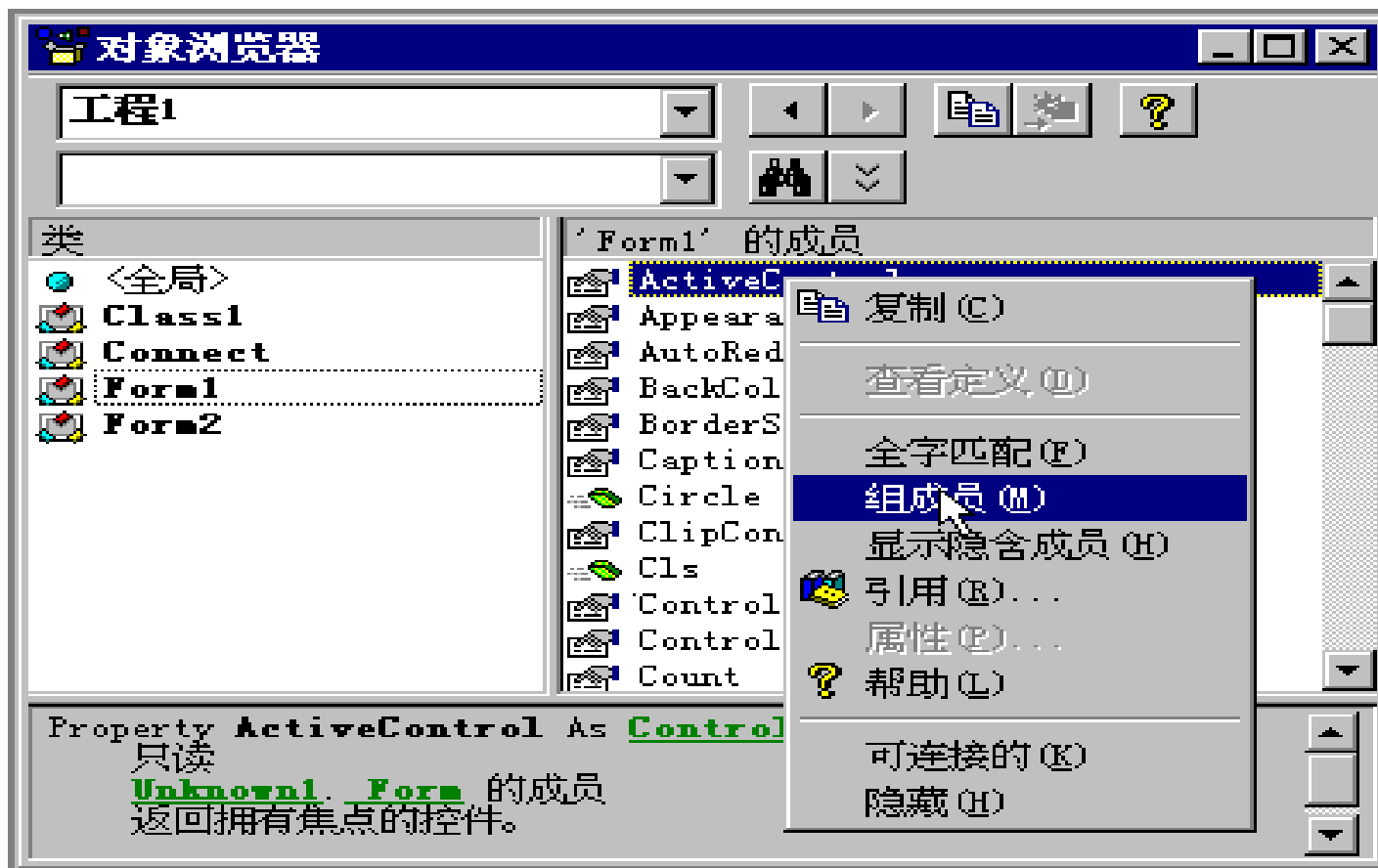


图 9.3 对象浏览器的上下文菜单

提示 选择“组成员”时，键入一个名字的首字母可跳转至以该字母开头的下一个名字处，即使该名字是在另一个组中。

查找和浏览对象

使用对象浏览器可以查找对象及其成员，还可以确定它们所属的工程或库。在“搜索文本”框中键入文本然后单击“搜索”按钮（或按 ENTER 键）。包含键入那些名字的文本的类和成员将出现在“搜索结果”列表中。例如，图 9.4 显示出在“搜索文字”框中键入“打印机”，然后单击“搜索”按钮后的结果。

可以在“搜索结果”列表中选择个项目，然后在对象浏览器底部的描述面板中查看其中的描述。单击描述面板中的带下划线的跳转项目可以选择所指定的库或对对象或成员进行定位。

可以通过在上下文菜单中选择“全字匹配”来限定只搜索与“搜索”框中的字符串精确匹配的项目。

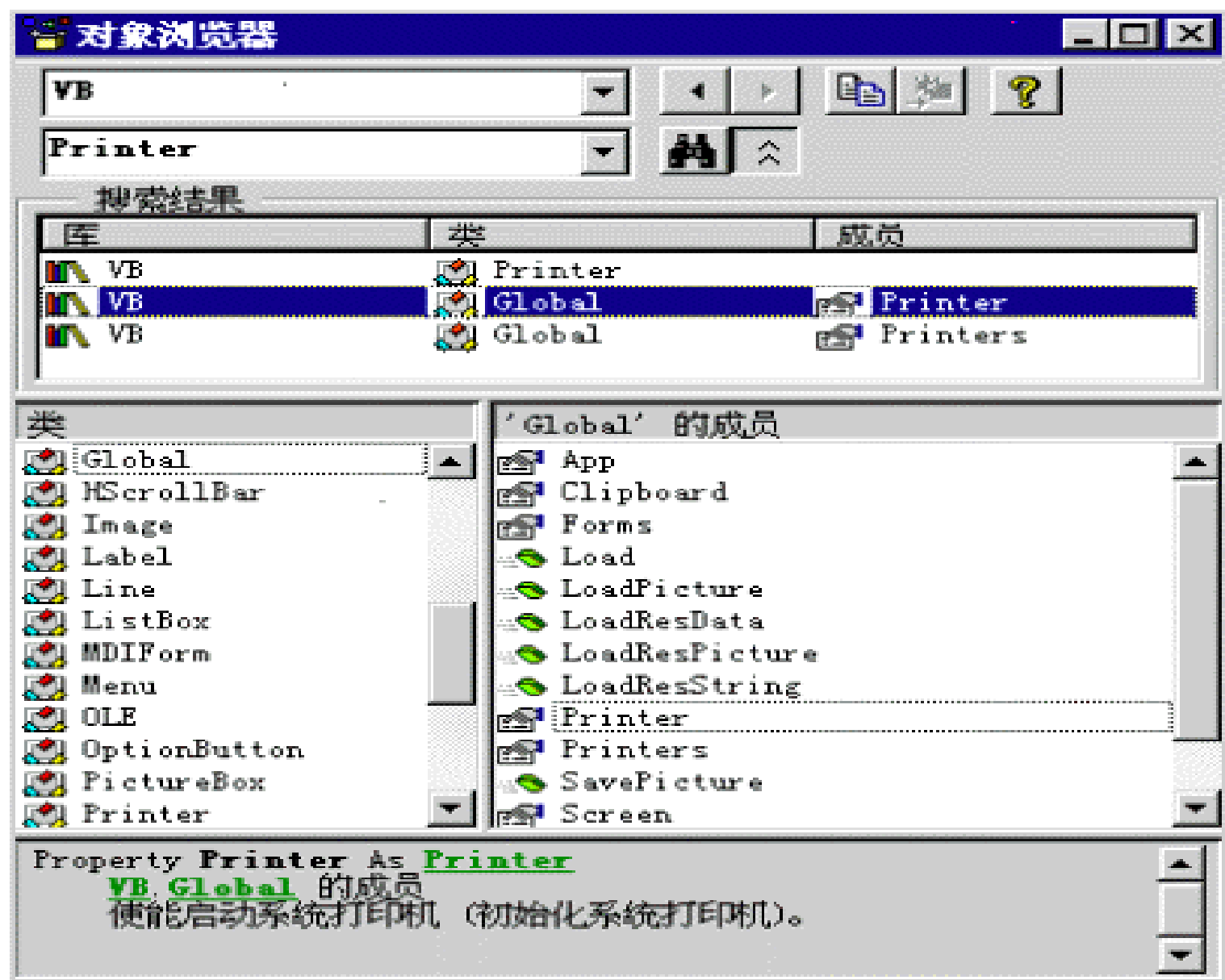


图 9.4 使用“搜索”按钮

添加对象的描述

可以使用对象浏览器给自己的过程、模块、类、属性和方法添加描述及“HelpContextIDs”。使用自己的类时就能找到这些有用的描述。

注意 也可以从“工具”菜单中使用“过程属性”对话框给属性、方法和事件添加描述。

要输入描述字符串，并将类及其成员与帮助主题进行链接，请按照以下步骤执行：

1. 按 F2 键打开“对象浏览器”。在“属性/库”框中选择所需的工程。
2. 在“类”列表中，在类名处单击右键弹出上下文菜单，然后单击“属性”打开“成员选项”对话框。

或者，也可以在“成员”列表中对已添加到类中的属性、方法或事件名单击右键。在上下文菜单中，单击“属性”。如果成员是 `Private` 的或 `Friend` 的，则打开“成员选项”对话框。如果成员是 `Public` 的——即类的界面部分——将代之以打开“过程属性”对话框。

注意 这两种对话框的区别是“过程属性”对话框有一个“高级”按钮，可用来将一个成员指定为该类的缺省成员，这一点将在本章后边的“使属性或方法成为缺省的”中进行描述。

3. 为了当这个类或成员在“对象浏览器”中被选中，并单击“?”时，该帮助主题的上下文标识符能被显示，应在“帮助上下文标识符”框中键入该帮助主题的上下文标识符。

注意 可以创建一个帮助文件以备自用，并将主题连接到自己的类及其成员中。如果希望为自己的工程指定一个帮助文件，可在“工程”菜单中打开“工程属性”对话框，使用其中的“通用”选项卡。

4. 在“描述”框中，键入关于该类或成员的简短描述。
5. 单击“确定”返回“对象浏览器”。键入的描述字符串应该出现在浏览器底部的描述面板中。
6. 对每一个类和其中的每一个成员重复第 2 步到第 5 步。

注意 不能为枚举添加浏览器字符串或帮助主题。

详细信息 枚举在第八章“再论编程”中的“通过枚举使用常数集进行工作”中进行介绍。

在过程之间移动

可以使用对象浏览器向工程的类代码、模块代码或过程代码进行快速移动。要移至类、模块或过程，请按照以下步骤执行：

1. （可选）从“工程/库”框中选择工程。
如果在“工程/库”框中选择了“所有库”，第一步就是可选的，因为已经包括了全部的工程。
2. 属于该工程的类、模块和成员的名称将以粗体显示。双击任何一个以粗体显示的名字以便移至该类、模块或成员（或者对一个名字单击右键，然后从上下文窗口中选择“查看定义”）。
被选择的项目将显示在“代码”窗口。

浏览其它应用程序的对象

在 Visual Basic 中，可以访问和控制由其它应用程序提供的对象。例如，如果在系统中存在 Microsoft Project 和 Microsoft Excel，则可利用 Microsoft Excel 中的图形对象和 Microsoft Project 中的日历对象作为应用程序的一部分。

可以利用对象浏览器检测其它应用程序的类型库。类型库提供了关于其它应用程序中的对象的信息。

注意 在“工程/库”列表中，有关于 Visual Basic (VB) 和 Visual Basic for Applications (VBA) 的单独项。尽管提到“由 Visual Basic 提供的对象”，但可以注意到 Collection 对象是由 VBA 提供的。

在对象浏览器的上下文菜单中选择“引用”，打开“引用”对话框，可以向工程中添加库。

详细信息 关于使用自动化去组合和处理来自其它应用程序的对象的详细信息，请参阅第十章“用部件编程”。

创建自己的类

如果是一位有经验的编程者，经过多年的编程已经积累了一个有用的函数库。对象并不会代替函数——实用程序的函数仍然需要编写和使用——但

对象对于组织过程和数据提供了一个方便和逻辑的方法。
特别是，借以创建对象的类能将数据和过程组织成一个整体。

类：将用户定义类型和过程组织在一起

用户定义类型是一个使相关数据项成组的强大工具。例如，假设此处要定义名为 `udtAccount` 的用户定义类型：

```
Public Type udtAccount
    Number As Long
    Type As Byte
    CustomerName As String
    Balance As Double
End Type
```

可以声明一个 `udtAccount` 类型的变量，分别给各个字段赋值，然后将整个记录传递给打印它的过程，并将其保存至数据库中，对其执行计算，验证各字段的数值，等等。

用户定义类型尽管功能强大，但也给编程者带来了一些问题。例如所创建的提款过程可能出错，如果当提款超过帐上余额，程序中又没有措施防止其它代码对余额字段减时就会出错。

换句话说，过程和用户定义类型的结合效果取决于规则、内存和维护代码的编程者的知识水平。

对象：具有态的用户定义类型

面向对象的编程通过将数据和过程结合成一个单一实体解决了上述问题，如图 9.5 所示。

当用户定义类型 `udtAccount` 变成 `Account` 类时，其数据变成了私有型，访问它们的过程移动类中并变成了属性和方法。这就是所谓的封装的意思——这就是说，对象是一个包括了代码和数据的单元（如果愿意，也可称其为密封舱）。

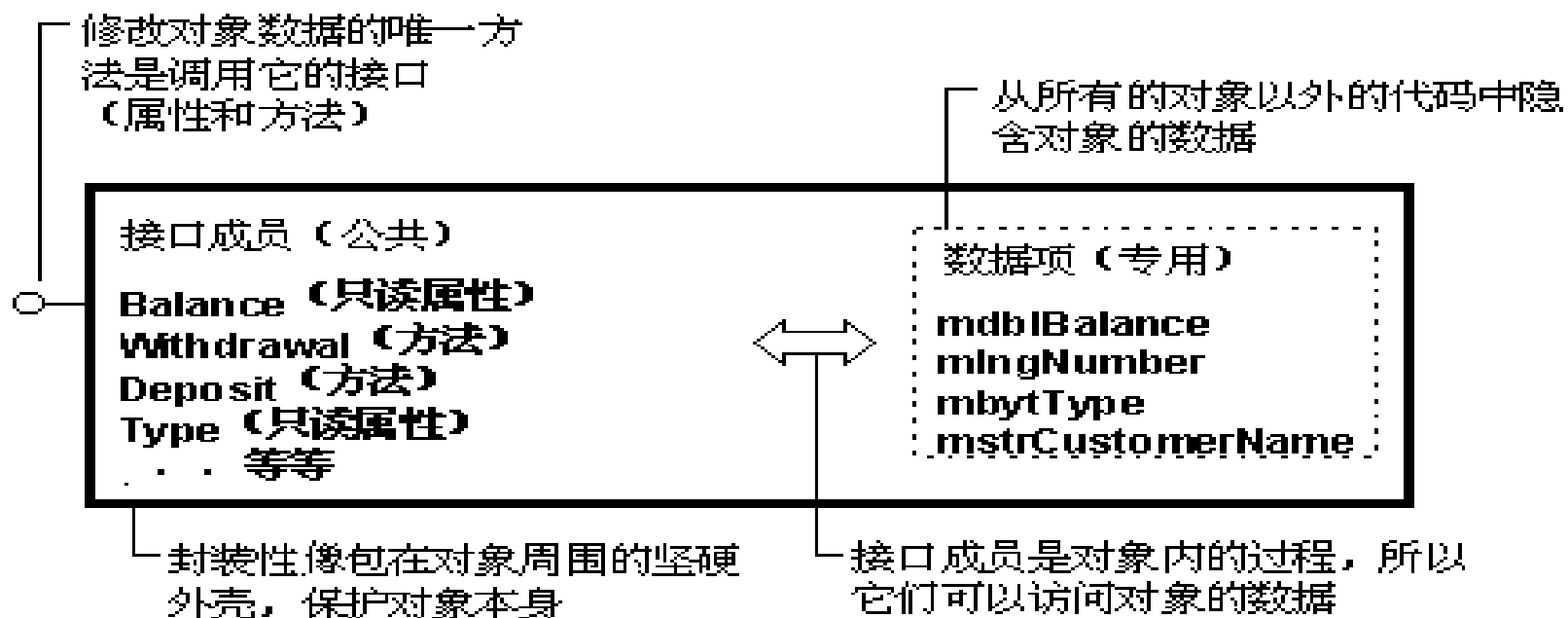


图 9.5 对象使数据和过程相结合

当从 `Account` 类创建一个 `Account` 变量时，访问其数据的唯一途径就是通过构成其界面的属性及方法。以下的代码段说明在 `Account` 类中的过程是如何支持封装的：

'帐户余额对外界代码是隐藏的。

```
Private mdblBalance As Double
```

'帐户余额的只读属性允许外部的代码

'获取帐户余额。

```
Public Property Get Balance() As Double
```

```
    Balance = mdblBalance
```

```
End Property
```

'提取存款的方法改变了帐户的余额，

'但仅限于不发生透支错误的情况。

```
Public Sub Withdrawal(ByVal Amount As Double)
```

```
    If Amount > Balance Then
```

```
        Err.Raise Number:=vbObjectError + 2081, _
```

```
        Description:="Overdraft"
```

```
    End If
```

```
    mdblBalance = mdblBalance - Amount
```

```
End Sub
```

现在，暂不要考虑如何将过程放进类中，也不要考虑如何理解属性过程和

私有变量的语法。要记住的重点就是可以定义一个对象来封装数据和确保数据的可靠。

有了 `Account` 对象，就永远不必关心是否调用了正确的过程更新帐户，因为可以调用的那些过程都已经组装进对象中了。

详细信息 “定制窗体类”部分将属性和方法的创建放入已熟悉的框架。然后，“给类添加属性和方法”部分将解释其语法。

可以参阅第八章“再论编程”中的“创建自己的数据类型”中关于用户定义类型的内容。

关于子过程和函数过程的详细信息，请参阅第五章“编程基础”中的“过程概述”。

定制窗体类

如果说创建类的历史与使用 `Visual Basic` 编程的历史一样长，也许会感到惊奇。但这是真的：`Form1`，每次开始一个工程都要碰到的这个老相识，实际上就是一个类。

为了说明这一点，打开一个新的“标准 `EXE`”工程。添加一个按钮到 `Form1`，并在单击事件中放置如下的代码：

```
Private Sub Command1.Click()  
    Dim f As New Form1  
    f.Show  
End Sub
```

按 F5 键运行这个工程，并单击该按钮。啊！出现了 Form1 的另一个实体！单击其中的按钮，又有另一个窗体！所创建的每一个实体看上去都是一样的，并且有同样的表现，因为它们都是 Form1 类的实体。

这里发生了什么事情？

如果已阅读了第五章“编程基础”中的“使用对象”，就会知道一个被声明成“`As New`”的对象变量，在代码中第一次被引用之前所包含的是“`Nothing`”。当变量第一次被使用时，Visual Basic 注意到其中包含特定数值“`Nothing`”，就要创建一个该类的实例（这也是一件好事，否则 `f.Show` 将引起错误）。

隐藏全局变量的 `Me` 和 `My`

也许会有点奇怪，在代码中怎样就能引用 Form1，它好像是一个对象变量。这并不离奇，Visual Basic 为每一个窗体类创建了一个隐藏的全局对象变量。看起来就像 Visual Basic 向工程添加了下列声明：

```
Public Form1 As New Form1
```

当选择 Form1 作为启动对象时，或者在代码中键入 `Form1.Show`，就引用了这个隐藏的全局对象变量。因为它被声明为“`As New`”，因此当代码中第一次使用这个预先声明的变量时，将创建 Form1 类的一个实例。

隐藏该声明的理由是每次更改窗体的 `Name` 属性时，Visual Basic 都要改变它。通过这种方法，隐藏的变量总是与窗体类有同样的名字。

一个小问题

在上述练习中创建的 `Form1` 的哪一个实例与隐藏的全局变量相关联？如果认为是第一个，则是对的。`Form1` 是工程的缺省启动对象，对 `Visual Basic` 而言它和在代码中使用预先声明的全局变量一样。

提示 在一个窗体卸载后，总是必须将对该窗体的任何引用设置为“`Nothing`”，以便释放其所使用的内存和资源。最经常被忽略的引用是这个隐藏的全局窗体变量。

关于 `Form1` 的其它实例

在第五章“编程基础”中，已经了解到要引用一个对象，需要一个对象变量，一个对象只有当至少存在一个对象变量包含对它的引用时才能存在。

`Form1` 的第二个实例，及以后的其它实例，都有一个对象变量，只要是调用其 `Show` 方法时就一直存在。然后该变量退出可见范围，并且被设置为“`Nothing`”。但是 `Visual Basic` 保持一个名为 `Forms` 的特殊的集合，这可以在第六章“创建用户界面”中的“再论窗体”中看到。该 `Forms` 集合包含了对工程中每一个已被装载的窗体的引用，所以总是可以找到和控制它们。

注意 正如所了解的那样，并不是对所有的类来说都是这样。例如，自己设计的类将不会有隐藏的全局变量或全局集合来跟踪它们——这些是窗体类的特定功能。但是，可以声明自己的全局变量，也可以创建自己的集合——就像在“创建自己的集合类”中所描述的那样。

窗体类的属性、方法和事件

第一次为窗体类添加属性时，可以直接把一个命令按钮（或其它的控件）拖放到 `Form1` 中。这样就添加了一个只读的 `Command1` 属性给窗体类。此后，当需要调用命令按钮的方法或属性时就是引用 `Form1` 的这一属性：

```
Command1.Caption = "Click Me"
```

在改变窗体中任何控件的 `Name` 属性时，`Visual Basic` 悄悄地改变了只读属性的名称以使二者匹配。

如果以前练习示例的工程还打开着，可以按 `F2` 键打开对象浏览器能见到 `Command1` 属性。在“工程/库”框中，选中 `Project1`，可以在类窗格中见到 `Form1`。向下滚动成员窗格直到发现 `Command1`，选中它。

`Command1` 边上有一个属性符号。从说明窗格中可以知道它是 `WithEvents` 属性。正如在“向类中添加事件”中所述，这表示属性（或对象变量）具有与之相关的事件过程。作为事件过程之一的 `Command1_Click()` 也许是自编写 `Visual Basic` 代码以来的第一个。

其它

把控件放到窗体上不是向窗体类添加成员的唯一方法。还可以在建立新变量和过程时添加定制的属性、方法和事件。

为了说明这点，请把下列代码添加到 `Form1` 的声明部分：

```
'Form1 类的 Comment 属性
```

```
Public Comment As String
```

请把下列代码添加到 Form1 的 Click 事件：

```
Private Sub Form_Click()
```

```
    MsgBox Comment, "My comment is:"
```

```
End Sub
```

最后，如下所示向 Command1_Click() 事件过程添加一行代码：

```
Private Sub Command1_Click()
```

```
    Dim f As New Form1
```

```
    f.Comment = InputBox("What's my comment?")
```

```
    f.Show
```

```
End Sub
```

按 F5 键运行该工程。单击 Command1，出现输入框时，输入一些有趣的注释然后单击 OK。当 Form1 的新实例出现时，单击它以显示其 Comment 属性。

单击第一个 Form1 实例，可以看到其 Comment 属性是空白的。这是由于 Visual Basic 把该实例当作启动对象，不可能有机会设置其 Comment 属性。

窗体间可以相互调用其方法

如果仔细观察可以发现，添加到 Form1 类中代码并不设置该对象本身的 Comment 属性——它只设置已被创建的 Form1 新实例的 Comment 属

性。

设置其它窗体的属性和调用其它窗体的方法是一项非常有用的技术。例如，一个 `MDIForm` 在打开一个新的子窗口时可以通过设置其属性、调用其方法来对它初始化。

还可将此技术用于在窗体之间传递信息。

提示 可以为窗体创建定制的事件。本章后面的“向窗体中添加事件”中将逐步介绍。

其它类型的模块

通过向其代码模块添加代码的方式可以给窗体类添加属性、方法和事件。还可以用同样方法向类模块添加属性、方法和事件。对于 Visual Basic 专业版或企业版，该方法还可添加到 `UserControl` 和 `UserDocument` 代码模块。

当读到“向类中添加属性和方法”及“向类中添加事件”这些节时，应记住文中所描述的对窗体类所做的每一件事同样适用于类模块。

详细资料 什么是类模块？在“类模块循序渐进”中介绍了如何定义类并列举了通过类创建的对象存活期。

类模块的循序渐进

本示例演示了如何用类模块定义类（然后可以用它创建对象）。还将展示如

何为新的类创建属性和方法并演示对象是如何创建和撤销的。
打开一个新的“标准 EXE”工程，在“工程”菜单中选中“添加类模块”插入一个类模块。在窗体上画四个按钮。下表列出了本示例中要设置的对象属性值。

对象	属性	设置值
Class 类模块	Name	Thing
Command1	Caption	显示该 Thing
Command2	Caption	颠倒该 Thing 的名称
Command3	Caption	创建新的 Thing
Command4	Caption	临时的 Thing

注意 类模块保存在扩展名为 .cls 的文件中。
添加下列代码到类模块的声明段：

```
Option Explicit
Public Name As String
Private mdtmCreated As Date
```

变量 Name 将成为 Thing 对象的属性，因为它声明为 Public。

注意 不要把这个 Name 属性和上表设置的类模块的 Name 属性混淆（类模块的 Name 属性为 Thing 类提供了名称）。为什么要给 Thing 类命名呢？最好应该反过来问，为什么不？应该给 Thing 类一个名称因为事物总是有名称的！应该知道 Visual Basic 使用属性和方法的名称并没有什么特殊点。可以为自己不同的类使用相同的属性和方法名。

变量 `mdtmCreated` 是私有数据成员，用于存储只读的 `Created` 属性的值。`Created` 属性返回 `Thing` 对象产生的日期和时间。为了实现 `Created` 属性，将以下 `Property Get` 代码添加到类模块的声明部分：

```
Property Get Created() As Date
```

```
    Created = mdtmCreated
```

```
End Property
```

注意 如果通过“工具”菜单下的“添加过程”对话框添加属性过程，应确认删除了 `Property Let` 声明，它是通过本对话框自动添加的。

`Property Let` 只用于读写属性，参见“运用属性过程”部分。

`Thing` 对象有一个方法 `ReverseName`，用于颠倒 `Name` 属性的字母顺序。它不返回值，因此是一个 `Sub` 过程。向类模块添加以下 `Sub` 过程。

```
Public Sub ReverseName()
```

```
    Dim intCt As Integer
```

```
    Dim strNew As String
```

```
    For intCt = 1 To Len(Name)
```

```
        strNew = Mid$(Name, intCt, 1) & strNew
```

```
    Next
```

```
    Name = strNew
```

```
End Sub
```

类模块有两个事件：`Initialize` 和 `Terminate`。从类模块的“对象”下拉列表中，选择“类”。下拉出的“过程”列表将显示事件。将以下代码放到事件过程中：


```
Private Sub Class_Initialize()  
    '设置对象创建的日期/时间，  
    '由只读的 Created 属性返回。  
    mdtmCreated = Now  
    '显示对象属性。  
    MsgBox "Name: " & Name & vbCrLf & "Created: " _  
        & Created, , "Thing Initialize"  
End Sub
```

```
Private Sub Class_Terminate()  
    '显示对象属性。  
    MsgBox "Name: " & Name & vbCrLf & "Created: " _  
        & Created, , "Thing Terminate"  
End Sub
```

一般 **Initialize** 事件包含了对象创建时所需的所有代码，如为 **Created** 属性提供时间标记。**Terminate** 事件包含撤销对象后进行清理的所有代码。本示例中，两个事件主要用来提供 **Thing** 对象创建或撤销的可视化指示。

Thing 对象的使用

添加以下代码到窗体模块的声明段：

```
Option Explicit  
Private mth As Thing
```

变量 `mt` 保存了对一个 `Thing` 对象的引用，该对象是在窗体的 `Load` 事件中产生的。将以下代码分别添加到 `Form_Load` 事件过程及四个按钮的 `Click` 事件过程中。

```
Private Sub Form_Load()  
    Set mt = New Thing  
    mt.Name = InputBox("Enter a name for the Thing")  
End Sub
```

'按钮“显示该 `Thing`”

```
Private Sub Command1_Click()  
    MsgBox "Name: " & mt.Name & vbCrLf _  
        & "Created: " & mt.Created, , "Form Thing"  
End Sub
```

'按钮“颠倒该 `Thing` 的名称”

```
Private Sub Command2_Click()  
    mt.ReverseName  
    '单击“显示该 Thing”  
    Command1.Value = True  
End Sub
```

'按钮“创建新的 `Thing`”

```
Private Sub Command3_Click()  
    Set mth = New Thing  
    mth.Name = InputBox( _  
        "Enter a name for the new Thing")  
End Sub
```

按钮“临时的 Thing”。

```
Private Sub Command4_Click()  
    Dim thTemp As New Thing  
    thTemp.Name = InputBox( _  
        "Enter a name for the Temporary Thing")  
End Sub
```

运行工程

按 F5 键运行该工程。观察 Form_Load 事件过程可知 New 操作符用于创建 Thing 对象。对该对象的引用赋给了变量 mth。

InputBox 函数要求输入 Thing 的名称。输入名称并按 ENTER 键后，返回值被赋予 Thing 对象的 Name 属性。

显示窗体 **Thing**

可以通过按下第一个按钮“显示该 Thing”来验证所赋的 Name 属性值，

此时将显示一个包含了 `Thing` 对象所有属性的消息框。

颠倒 `Thing` 的名称

按第二个按钮“颠倒该 `Thing` 的名称”。该按钮调用 `ReverseName` 方法把 `Thing` 对象的名称倒过来，然后按第一个按钮显示更新过的属性值。

创建新的 `Thing`

单击“创建新的 `Thing`”按钮撤销已存在的 `Thing` 对象后再创建一个新的对象（或，反过来进行，创建一个新的 `Thing` 对象然后撤销旧的）。

`New` 操作符创建一个新的 `Thing`，因此可以看见新 `Thing` 的 `Initialize` 事件显示的消息框。单击“确定”，对新 `Thing` 的引用被赋给了窗体级的变量 `mt`。

这样做清除了对旧的 `Thing` 的引用。由于不再有引用，所以它被撤销，因此可以看到 `Terminate` 事件的消息框。单击“确定”后，`InputBox` 语句要求输入新 `Thing` 的名称。

注意 如果希望在创建新的 `Thing` 之前撤销旧的，可以添加代码 `Set mt = Nothing` 到事件过程的开始部分。

临时的 `Thing`

第四个按钮演示了对象存活期的另一个方面。按下后，将提示输入临时 `Thing` 的名称。

可以等一等，这还不是一个临时的 `Thing` 对象。还没有见到其 `Initialize` 消息框，怎么赋给它名称呢？

由于变量 `thTemp` 已被 `As New` 声明过了，因此当一个属性或方法被使用时就会创建 `Thing` 对象。这将发生在输入框返回的值被赋给 `Name` 属性时。在输入框键入名称后单击“确定”。

此时可以看到 `Initialize` 的消息框，其中的 `Name` 属性还是空白。当单击“确定”关闭消息框后，`InputBox` 语句的值终于赋给了 `Name` 属性。看看这一行代码作了多少事。

当然，当完成了这一切后，`Click` 事件过程也结束了，变量 `thTemp` 也超出可见范围，对临时 `Thing` 的对象引用也被释放，因此可以看到 `Thing` 的 `Terminate` 消息框，注意其中包含所提供的名称。

每次单击该按钮将创建另一个临时 `Thing`，并进行命名，将它撤销。

关闭程序

单击窗体的“关闭”按钮可终止应用程序。不要使用工具栏上的 `End` 按钮。应用程序终止时，`Form1` 被撤销。变量 `mth` 也超出可见范围，`Visual Basic` 清除对 `Thing` 的引用。由于已不存在对 `Thing` 的引用，因此被撤销并显示 `Terminate` 事件的消息框。

再次运行应用程序，这次用工具栏上的 `End` 按钮终止程序。注意这时没有显示 `Thing` 对象的 `Terminate` 消息框。

应特别记住使用 `End` 按钮终止应用程序或在代码中用 `End` 语句立即退出应用程序，并不能运行任何对象的 `Terminate` 事件。最好能通过卸载所有

的窗体来关闭应用程序。

通过 F8 键以每次运行一行代码的形式运行示例，是一个很有用的方法。这有利于理解创建和撤销对象的事件的顺序。

重点 在实际的应用程序中，Initialize 和 Terminate 事件不应包含消息框或其它任何使 Windows 显示消息的代码。一般来说，调试对象的存活期时最好用 Debug.Print 语句。

详细信息 窗体和控件与其它对象有一些差别，这一点将在“Visual Basic 窗体的存活期”中讨论。

在“向类中添加属性和方法”和“向类中添加事件”这些节中可以读到更多有关使用类和类模块的信息。

调试类模块

调试类模块与调试普通应用程序有一些区别。这是因为类模块的属性或方法中的错误，都是可处理的（即，在调用栈中总有一个能处理该错误的过程，该过程要调用该类模块的属性或方法）。

为了补偿这些区别，Visual Basic 除了提供以前的在无法处理错误点时中断和在所有错误点中断选项之外，还提供了类模块内的可捕获错误中断选项。

注意 可以在“工具”菜单下“选项”对话框的“通用”选项卡中设置缺省的错误捕获状态。所选择的选项影响当前的会话期，并且变成 Visual Basic 的所有后续实例的缺省状态。改变设置只影响当前会话期，不影响缺省状态，从“代码窗口文本”菜单选择“切换”（在“代码”

窗口上单击鼠标右键)。

例如，假设类模块 `Class1` 包含以下代码：

```
Public Sub Oops()  
    Dim intOops As Integer  
    intOops = intOops / 0  
End Sub
```

假设在其它类模块、窗体或标准模块中有一个过程调用成员 `Oops`：

```
Private Sub Command1_Click()  
    Dim c1 As New Class1  
    c1.Oops  
End Sub
```

如果可捕获错误选项设为在无法处理错误点时中断，程序将不会停止在被零除的地方。相反该错误能在调用过程 `Command1_Click` 中发生，程序将在调用 `Oops` 方法处停止。

可以设置在所有错误点中断来使程序停止于被零除，但对大多数应用来说设置在所有错误点中断很不方便。这样做每遇到一个错误就会停下来，即使已经编写了错误处理程序。

类模块内中断是折衷的设置：

- 已编写了错误处理的地方，执行将不会停止在类模块中。
- 执行只停止于类模块中不能处理的错误上，并返回到方法的调用处。
- 启动 `Visual Basic` 开发环境后，缺省地设置为类模块内中断。
- 如果没有使用类模块，在类模块内中断与在无法处理错误点时中断完全

相同。

提示 使用类模块内的中断或在无法处理错误点时中断，当遇到中断点时，可以 ALT+F8 键或 ALT+F5 键单步或跳过运行该错误，即进入自己编写的错误处理代码或进入在其中发生错误调用过程的代码中。

详细信息 在第十三章“调试代码和处理错误”中将详细讨论调试。

Visual Basic 窗体的存活期

由于窗体和控件是可见的，所以它们与其它对象的存活期不同。例如，即使释放了对窗体的所有引用，也不会关闭该窗体。Visual Basic 维护整个工程所有窗体的全局集合，只有当窗体卸载时才能从集合中删除该窗体。

同样的，Visual Basic 为每个窗体维护一个控件集合。可以从控件数组中加载或卸载控件，但简单地释放对控件的所有引用并不能撤销它。

详细信息 有关窗体和控件集合的内容已在本章的“Visual Basic 中的集合”中作了讨论。

Visual Basic 窗体经历的状态

通常地，Visual Basic 窗体在整个存活期中有四种状态：

1. 创建，但不加载。
2. 加载，但不显示。
3. 显示。

4. 内存和资源完全收回。

在一定环境下，窗体可有第五种状态：当其中有一个控件仍被引用时，窗体处于卸载和未引用状态。

本主题将描述这些状态及状态之间的转换。

创建，但不加载

`Initialize` 事件是该状态开始的标志。因而，放在 `Form_Initialize` 事件过程中的代码，就是窗体创建时最先执行的代码。

处于这种状态时，窗体是作为一个对象而存在，但还没有窗口。而且它的控件也不存在。虽然该状态可能很短暂，但任何窗体都要经过该状态。

例如，如果执行 `Form1.Show`，则窗体被创建，`Form_Initialize` 开始执行；一旦 `Form_Initialize` 执行完毕，该窗体被加载，这是下一个状态。

在指定窗体为启动对象时会发生同样的过程。在“工程”菜单中，选取“工程属性”对话框，然后选定“通用”选项卡，就可以指定一个窗体为启动对象。一旦窗体被指定为启动对象，该窗体在工程启动时就被创建，并立即加载和显示。

注意 正如下面描述的，通过调用窗体的 `Show`，或使用内置的属性和方法，都可以从 `Form_Initialize` 中加载窗体。

保持创建，但不加载的状态

相反的，以下的代码创建 `Form1` 的实例，但不让其进入加载状态：

```
Dim frm As Form1
```

```
Set frm = New Form1
```

一旦 `Form_Initialize` 结束，在不强制加载窗体的情况下，所能执行的过程只有能添加到该窗体代码窗口的 `Sub`、`Function` 和 `Property` 过程。例如，可以给 `Form1` 添加以下的方法：

```
Public Sub ANewMethod()
```

```
    Debug.Print "Executing ANewMethod"
```

```
End Sub
```

使用变量 `frm`（也就是 `frm.ANewMethod`），可以在窗体不强制进入下一状态的情况下调用该方法。同样的，可以调用 `ANewMethod` 创建窗体：

```
Dim frm As New Form1
```

```
frm.ANewMethod
```

由于 `frm` 声明为 `As New`，所以直到代码中首次使用该变量之前，上述情况是调用 `ANewMethod`，该窗体不能创建。上面的代码执行后，该窗体保持在已创建状态，但没有加载。

注意 只要运行 `Form1.ANewMethod`，无需声明窗体变量，也可以达到上述示例的效果。正如“定制窗体类”中所解释的，`Visual Basic` 为每一个窗体类创建一个隐含的全局变量。该变量名和类名相同，这可想像为 `Visual Basic` 进行了 `Public Form1 As New Form1` 声明。

可以在不加载窗体的情况下，任意运行定制的属性和方法。然而，一旦访问了窗体内置的任一属性或控件，该窗体就进入下一状态。

注意 把一个窗体分为两部分是非常有用的，一部分是代码部分，另一部分是可视部分。窗体加载前，只有代码部分在内存中。这样，可以不加载窗体的可视部分，而在代码中任意调用过程。

所有窗体都需经过的唯一状态

创建完毕但未加载，是所有窗体都需经过的唯一状态。如果上述示例中变量 `frm` 被设置为 `Nothing`，则该窗体在进入下一状态前就撤销了：

```
Dim frm As New Form1
```

```
frm.ANewMethod
```

```
Set frm = Nothing          '窗体被撤销。
```

这样使用的窗体不会比类模块好，所以绝大部分窗体进入下一状态。

加载但不显示

`Load` 事件标志着此状态的开始。一旦窗体进入加载状态，`Form_Load` 事件过程中的代码就开始执行。

`Form_Load` 事件过程开始后，窗体上的所有控件都被创建和加载，而且该窗体有了一个窗口——是通过窗口句柄 (`hWnd`) 和设备描述体 (`hDC`) 完成的，尽管该窗口还未被显示。

任何窗体只有加载后才能可见。

很多窗体自动从创建但不加载状态进入加载但不显示状态。窗体如果满足以下的条件就会自动加载：

- 该窗体在“工程属性”对话框的“通用”选项卡中被指定为启动对象。
- 窗体中首先被调用的属性或方法是 `Show` 方法，正如示例 `Form1.Show`。
- 首先被调用的窗体属性或方法是窗体内置的成员，如上述例子中是 `Move` 方法。

注意 因为每个控件定义了窗体的一个属性，所以这种情况包括了窗体的所有控件；也就是说，为了访问 `Command1` 的 `Caption` 属性，就必须经过窗体的 `Command1` 属性：`Command1.Caption`。

- 正如早些时候所述的，可以用 `Load` 语句加载窗体，而无需首先使用 `New` 或 `As New` 创建该窗体。

从不显示的窗体

上述的前两种情况，一旦 `Form_Load` 执行完毕，窗体就直接可见。而后面的两种情况，窗体将保持加载状态，但不显示。

在 `Visual Basic` 中编程时，常常加载了某一窗体但从未予显示。这样做有以下的原因：

- 用时钟控件产生计时事件
- 用功能控件，而不是用户界面控件。例如，串行通信或访问系统文件。
- 执行 `DDE` 事务

注意 对于专业版和企业版，可以创建 `ActiveX` 部件（以前称为 `OLE` 服务端），这比控件更有利于提供纯代码功能。请参阅《部件工具指南》中的“创建 `ActiveX` 部件”。

总是返回的根状态

任何时候，只要隐藏了窗体，它就总是从可见状态回到加载状态。回到加载状态并不重新执行 Load 事件。Form_Load 在窗体的存活期中只运行一次。

显示状态

一旦窗体可见，用户就能和它交互作用。当然，窗体在卸载前可以任意隐藏及显示。

其它事项：卸载前的准备

窗体在卸载时可以是隐藏的，也可以是可见的。若没隐藏，则它将保持可见直到卸载完毕。

窗体卸载前最后发生 Unload 事件。该事件发生前，有另一个重要的事件发生，即 QueryUnload。QueryUnload 提供了停止窗体卸载的机会。如果某些数据希望保存，则此时将提示保存或忽略所做的更改。

重点 把 QueryUnload 的参数 Cancel 设置为 True，就会忽略 Unload 语句，从而不卸载窗体。

QueryUnload 事件的一个重要功能是还要了解窗体的卸载是什么原因造成的：是单击“关闭”按钮，或是程序中执行 Unload 语句，或在应用程序中关闭，或者是在 Windows 中的关闭。所以 QueryUnload 提供了取消关

闭窗体的机会，同时也允许在需要时从代码中关闭窗体。

重点 在一些情况下，窗体不会接收到 `QueryUnload` 事件。例如，使用了 `End` 语句来结束程序，或在开发环境中单击“结束”按钮（或从“运行”菜单中，选取“结束”按钮）。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`QueryUnload` 事件”。

返回到创建但不加载状态

窗体卸载后，Visual Basic 把它从 `Forms` 集合中删除掉。除非使用变量保持对窗体的引用，否则该窗体将被撤销，其所占内存和资源会被 Visual Basic 收回。

如果使用变量保持了对窗体的引用，例如“定制窗体类”中描述的隐含全局变量，窗体就会回到创建但不加载状态。窗体的窗口、控件不再存在。

对象将继续占有资源和内存。所有窗体代码部分模块级变量中的数据继续存在（但是，事件过程中的 `Static` 变量将消失）。

可以使用已有的引用调用窗体中所添加的方法和属性。但如果调用了内置的成员，或访问其控件，该控件将再次加载，并执行 `Form_Load`。

完全释放内存和资源

释放内存和资源的唯一办法就是卸载窗体，并把所有引用设置为 `Nothing`。这种做法常常会漏掉那些隐含的全局变量引用。如果使用了类名（正如“属性”窗口中的 `Name` 属性所示）来引用窗体，就等于使用隐含全局变量。

为了释放窗体占用的内存，必须把该变量设置为 `Nothing`。例如：

```
Set Form1 = Nothing
```

该窗体在撤销前会接收到 `Terminate` 事件。

提示 很多专业编程人员都避免使用隐含全局变量，而趋向于声明自己的窗体变量（例如，`Dim dlgAbout As New frmAboutBox`）。

注意 执行 `End` 语句将卸载窗体并把所有的对象变量设置为 `Nothing`。然而，这种中断程序的方法非常唐突。所有的窗体都不会发生 `QueryUnload`、`Unload` 或 `Terminate` 事件，所创建的对象也不会发生 `Terminate` 事件。

卸载未被引用，但有控件仍被引用

为了进入这一状态，就必须在卸载和释放窗体时保持对其中某一控件的引用：

```
Dim frm As New Form1
```

```
Dim obj As Object
```

```
frm.Show vbModal
```

'模态窗口解体，保存对其上一个控件的引用。

```
Set obj = frm.Command1
```

```
Unload frm
```

```
Set frm = Nothing
```

窗体已卸载，则释放对它的所有引用。但只要还引用了其中的一个控件，

其代码部分将仍然保存在内存中。一旦调用该控件的任一属性或方法，该窗体将被再次加载：

```
obj.Caption = "Back to life"
```

模块级变量将保留它们的值，但所有控件的属性被设置为缺省值，好像该窗体是首次加载一样。Form_Load 将被执行。

注意 在 Visual Basic 以前的有些版本中，窗体不能完整地重新初始化，Form_Load 也不会再次执行。

注意 不是所有的窗体都像 Visual Basic 窗体。例如，Microsoft Office 中提供的 Microsoft Forms，就没有 Load 和 Unload 事件；一旦这些窗体接收到初始化事件，则它们所有的控件就开始存在，并可以使用。

详细信息 有关窗体的内容，在第三章“窗体、控件和菜单”中的“设计窗体”及第六章“创建用户界面”中的“再论窗体”中都作了讨论。

类模块和标准模块的比较

类模块和标准模块的不同点在于存储数据方法的不同。标准模块的数据只有一个备份。这意味着标准模块中一个公共变量的值改变以后，在后面的程序中再读取该变量时，它将得到同一个值。

而类模块的数据，是相对于类实例（也就是，由类创建的每一对象）而独立存在的。

同样的，标准模块中的数据在程序作用域内存在，也就是说，它存在于程

序的存活期中；而类实例中的数据只存在于对象的存活期，它随对象的创建而创建，随对象的撤销而消失。

最后，当变量在标准模块中声明为 `Public` 时，则它在工程中任何地方都是可见的；而类模块中的 `Public` 变量，只有当对象变量含有对某一类实例的引用时才能访问。

上面的比较，同样适用于标准模块和类模块中的公共过程。用下面的例子可以说明。打开一个新 `Standard Exe` 工程，并在“工程”菜单中添加一个标准模块和一个类模块，然后运行以下的代码：

把下面的代码放在 `Class1` 中；

'下面是 `Class1` 对象的一个属性。

```
Public Comment As String
```

'下面是 `Class1` 对象的一个方法。

```
Public Sub ShowComment()
```

```
    MsgBox Comment, , gstrVisibleEverywhere
```

```
End Sub
```

把下面的代码放在 `Module1` 中：

'标准模块中的代码是全局的。

```
Public gstrVisibleEverywhere As String
```

```
Public Sub CallableAnywhere(ByVal c1 As Class1)
```

```
    '下行改变一个全局变量，
```

'这是一个 Class1 实例的（属性）。

'只有传递给该过程的个别对象才受到影响。

c1.Comment = "Touched by a global function."

End Sub

把两个命令按钮放在 Form1 上，并在 Form1 中添加以下的代码：

Private mc1First As Class1

Private mc1Second As Class1

Private Sub Form_Load()

 '创建两个 Class1 类的实例。

 Set mc1First = New Class1

 Set mc1Second = New Class1

 gstrVisibleEverywhere = "Global string data"

End Sub

Private Sub Command1_Click()

 Call CallableAnywhere(mc1First)

 mc1First.ShowComment

End Sub

Private Sub Command2_Click()

 mc1Second.ShowComment

End Sub

按 F5 键，运行该工程。当 Form1 加载时，它创建两个 Class1 类实例，每个实例有自己的数据。同时，Form1 设置了下面全局变量

gstrVisibleEverywhere

的值。

按下 Command1，调用全局过程并传递引用给第一个 Class1 对象。全局过程设置 Comment 属性，然后 Command1 调用 ShowComment 方法显示该对象的数据。

正如图 9.6 所示，结果信息框演示了全局过程 CallableAnywhere 如何设置对象的 Comment 属性，而且全局字符串在 Class1 内部是可见的。



图 9.6 第一个 Class1 对象的信息框

按下 Command2，调用第二个 Class1 类实例的 ShowComment 方法。

如图 9.7 所示，两个对象都访问了全局字符串变量；然而，第二个对象的 Comment 属性是空的，因为对全局过程 CallableAnywhere 的调用只改变第一个对象的 Comment 属性。

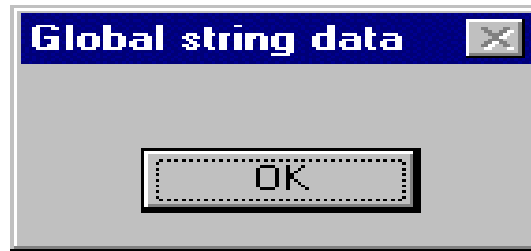


图 9.7 第二个 Class1 对象的信息框

重点 要避免类的代码依赖于全局变量，也就是标准模块中的公共变量。一个类的许多实例可以同时存在，所有这些对象在程序中共享全局数据。在类模块代码中使用全局变量也违背了面向对象封装的编程原则，因为由这样的类所创建的对象并没有包含它们的所有数据。

静态类数据

有时，希望由一个类模块所创建的一些对象共享某个数据项。这就是所谓的静态类数据。

不能在 Visual Basic 类模块中实现真正的静态类数据。但是，通过使用 Property 过程在标准模块中设置及返回 Public 数据成员的值，可以仿真静态类数据。如以下的代码：

'只读属性返回应用程序的名字。

```
Property Get CommonString() As String
```

```
    '变量 gstrVisibleEverywhere 保存在
```

```
    '标准模块中，并声明为 Public。
```

```
CommonString = gstrVisibleEverywhere
```

```
End Property
```

注意 在类模块中，对模块级的变量不能声明为 `Static`。`Static` 数据只能在过程中使用。

由 `Property Let` 过程（对于含有对象引用的属性，使用 `Property Set`）对标准模块数据成员赋予新的值，可以仿真非只读的静态类数据。然而，这种使用全局变量的方式违背了封装编程的原则，而且不推荐使用。

例如，变量 `gstrVisibleEverywhere` 可以在工程中任何地方赋值，甚至可以从别的类代码中赋值，只要该类具有 `CommonString` 属性。这样做能导致程序中非常微妙的错误。

详细信息 `ActiveX` 部件中的全局数据，与普通程序中的处理方式不同。如果有 `VBP` 或 `VBE`，请参阅《部件工具指南》中的“创建 `ActiveX` 部件”，第六章“部件设计的一般准则”中的“标准模块与类模块”。

向类中添加属性和方法

类的缺省界面由类的属性和方法组成。该界面是操作对象的最常用的方式。一般来说，属性表示对象的数据，方法表示对象能执行的动作。或者说，属性提供了对象的描述，而方法则是它的行为。

注意 事件不属于缺省界面。事件是调出接口界面（即，该界面可连接其它对象），而属性和方法属于调入接口（即，该界面的成员被其它对

象引用)。Visual Basic 对象的缺省界面是入端界面。

重点 下面的名字不能作为属性名或方法名，因为它们属于下一级的 IUnknown 和 IDispatch 接口：QueryInterface、AddRef、Release、GetTypeInfoCount、GetTypeInfo、GetIDsOfNames 和 Invoke。这些名字将引起编译错误。

详细信息 在本章后面的“向类中添加事件”中将讨论事件。

向类中添加属性

类属性定义最简单的方法，是向类的模块添加公共变量。例如，只要在名为 Account 的类中声明两个公共变量，可非常简单的创建 Account 类：

```
Public Balance As Double
```

```
Public Name As String
```

给类创建私有数据也很简单，只需声明一个 Private 变量，它就能被该类模块中的编码所访问：

```
Private mstrMOthersMaidenName As String
```

```
Private mintWithdrawalsMonthToDate As Integer
```

数据隐藏

数据隐藏是指保护对象的部分数据，而其余数据则以属性形式公开。这是面向对象的封装原则的一个方面，已在“类：将用户定义类型和过程组织

在一起”中进行了解释。

数据隐藏意味着可以改变类而不影响已存在的编码。例如，将 `Account` 类的私有变量 `mintWithdrawalsMonthToDate` 的类型从 `Integer` 改变为 `Long`，而不改变调用 `Account` 对象已存在的编码。

数据隐藏也允许定义只读属性。例如，可用 `Property Get` 属性过程返回包含了一个月的取款次数的私有变量，该变量只能在 `Account` 对象编码中递增。这就产生了属性过程。

属性过程

如果只能通过声明公共变量来创建属性，那么数据隐藏就没有多大用处。如果引用 `Account` 对象的任何代码将帐号类型设置为任意值，那么给 `Account` 类添加 `Type` 属性究竟有多大的好处呢？

当设置或检索属性值时，就可执行属性过程的代码。例如，可用一对属性过程来实现 `Account` 对象的 `Type` 属性：

```
Public Enum AccountTypes
    atSavings = 1
    atChecking
    atLineOfCredit
End Enum
```

'存储 `Type` 属性的私有数据。

```

Private mbytType As AccountTypes

Public Property Get Type() As AccountTypes
    Type = mbytType
End Property

Public Property Let Type(ByVal NewType As AccountTypes)
    Select Case NewType
        Case atChecking, atSavings, atLineOfCredit
            '若 NewType 有效，则无需做任何事。
        Case Else
            Err.Raise Number:=vbObjectError + 32112, _
                Description:="Invalid account type"
    End Select
    If mbytType > NewType Then
        Err.Raise Number:=vbObjectError + 32113, _
            Description:="Cannot downgrade account type"
    Else
        mbytType = NewType
    End If
End Property

```

假设有一个名为 `acct` 的变量，引用 `Account` 对象。当执行 `x = acct.Type`

编码时，调用 `Property Get` 过程，返回存于类模块私有成员 `mbytType` 的值。

当执行 `acct.Type = atChecking` 代码时，将调用 `Property Let`。若对象是全新的，则 `mbytType` 为零，并赋给任何有效的帐号类型。如果现在的帐号类型是 `atSavings`，则该帐号将会升级。

但是，如果当前的帐号类型是 `atLineOfCredit`，则 `Property Let` 为防止降级将给出错误信息。同样地，执行 `acct.Type = 0` 代码，`Property Let` 的 `Select` 语句将检测到非法的帐号类型值，并指出错误。

总之，属性过程可允许对象保护并验证自己的数据。

详细信息 任何情况下使用公共变量都好吗？“属性过程与公共变量比较”描述了二者正确使用的要点。

在“运用属性过程”中进一步给出了属性过程的功能。

属性过程与公共变量的比较

属性过程具有封装功能，它是如此强大的工具，以致于有时会觉得公共变量有点没用。对于编程来说答案是，“当然——有时是的”。以下是一些基本原则：

以下情况应使用属性过程：

- 属性为只读，或一旦设置就不能改变。
- 属性已设置的值需要验证。
- 超出特定范围的值。例如，负数，虽符合属性的数据类型，但属性如果

允许这样的假设值出现，就会导致程序出错。

- 属性的设置可导致一些对象状态的明显变化，例如，Visible 属性。
- 属性设置可改变内部变量或其它属性的值。

以下场合应使用只读属性的公共变量：

- 属性是自验证类型。例如，如果一个不是 True 或 False 的值被赋给 Boolean 变量，则或者出错，或者数据自动转换。
- 在数据类型所支持的范围内的值都是有效的。像许多 Single 或 Double 类型的属性。
- 属性是 String 数据类型，并没有大小或字符串取值的限制。

注意 不要仅仅为了避免函数调用的额外开销而用公共变量来实现一个属性。其实，由于类型库的要求，Visual Basic 在类模块中以任意方式将公共变量作为属性过程对使用。

运用属性过程

Visual Basic 提供下表所列的三种属性过程。

过程	目的
Property Get	返回属性的值
Property Let	设置属性的值
Property Set	设置对象属性的值（即，该属性含有对象引用）

从上表可见，每一个属性过程都有自己特殊的功能。典型的属性由一对属

性过程组成：Property Get 检索属性的值，Property Let 或 Property Set 给属性赋值。

这些作用在有些情况下可能重叠。有两类赋值的属性过程的原因，是因为将对象引用赋给对象变量时，Visual Basic 有特殊的语法：

```
Dim wdg As Widget
```

```
Set wdg = New Widget
```

规则很简单：如果有 Set 语句，则 Visual Basic 调用 Property Set，如果没有则调用 Property Let。

提示 返回到经典的 Basics 语法，为了正确使用 Property Let 和 Property Set 保持清洁整齐，应键入 Let x = 4（直到今天 Visual Basic 还支持语法）而不是 x = 4。Visual Basic 总是根据赋值类型决定调用属性过程 - 对于 Let x = 4 调用 Property Let，对于 Set c1 = New Class1（对象属性）则调用 Property Set。

详细信息 第五章“编程基础”中“对象使用”解释 Set 语句对对象变量的使用。

读写属性

下列编码显示了典型的读写属性：

属性值的私有存储区。

```
Private mintNumberOfTeeth As Integer
```

```
Public Property Get NumberOfTeeth() As Integer
```

```
        NumberOfTeeth = mintNumberOfTeeth
End Property

Public Property Let NumberOfTeeth(ByVal New Value _
    As Integer)
    ' (省略验证属性值代码。)
    mintNumberOfTeeth = New Value
End Property
```

存储了属性值的私有变量名，由作用域前缀 (m)，类型前缀 (int) 和名称 (NumberOfTeeth) 组成，其中，作用域前缀 (m) 指明变量是模块级的。使用属性相同的名称可指明该变量和属性是相关的。

毫无疑问，已经注意到，这里和以前的示例一样，含有读写属性的属性过程名必须是相同的。

注意 属性过程缺省为公共的，因此，如果省略 Public 关键词，则属性过程仍是公共的。如果想要使属性为私有（只有对象内部才可访问），必须用 Private 关键词声明。即使不是必须，最好还是使用 Public 关键词，因为这可使意图清楚。

运行和播放属性过程

单步运行一些属性过程代码是有益处的。使用“工程”菜单，打开一个新的 Standard Exe 工程，并增添一个类模块。如上所示，从 NumberOfTeeth 属性复制代码到 Class1。

切换到 Form1，将下列代码添加到 Load 事件中：

```
Private Sub Form_Load()  
    Dim c1 As Class1  
    Set c1 = New Class1  
    '赋给新的属性值。  
    c1.NumberOfTeeth = 42  
    '显示属性值。  
    MsgBox c1.NumberOfTeeth  
End Sub
```

按 F8 键，单步运行代码。注意，当属性被赋值时，运行跳到 Property Let，当检索属性值时，运行跳到 Property Get。将会发现，对属性过程其它组合，重复做这种练习是很有用的。

属性过程对的参数必须匹配

至今所见的属性过程示例都很简单，对多数属性而言属性过程也将是这样。但是，属性过程可有多个参数——甚至可选参数。多个参数中最有用的是数组，下面将讨论。

当使用多个参数时，属性过程对的参数必须匹配。下表给出了在属性过程声明中参数的要求。

过程	声明语法
Property Get	Property Get propertyname(1,..., n) As type

续表

Property Let	Property Let propertyname(1,..., n, n+1)
Property Set	Property Set propertyname(1,..., n, n+1)

在 Property 过程中，从第一个参数到倒数第二个参数 (1,..., n)，必须具有与过程相同的名称和数据类型。就像其它过程类型一样，列表中必需参数应排在第一个可选参数之前。

可能注意到，Property Get 过程的声明比相关的 Property Let 或 Property Set 少一个参数。Property Get 过程的数据类型的声明必须与 Property Let 或 Property Set 最后一个参数 (n+1) 的数据类型一样。

例如，一个类似二维数组的属性的 Property Let 声明：

```
Public Property Let Things(ByVal X As Integer, _
```

```
ByVal Y As Integer, ByVal Thing As Variant)
```

```
    ' (省略给数组元素赋值的代码。)
```

```
End Property
```

Property Get 的参数声明必须与 Property Let 过程的参数名称和数据类型相同：

```
Public Property Get Things(ByVal X As Integer, _
```

```
ByVal Y As Integer) As Variant
```

```
    ' (省略从数组取值的代码。)
```

```
End Property
```

在 Property Set 声明中的最后参数的数据类型，要么是对象类型，要么是

Variant 类型。

参数匹配

图 9.8 给出了这些参数匹配规则的理由。并显示了 Visual Basic 如何将赋值语句部分与 Property Let 参数相匹配。

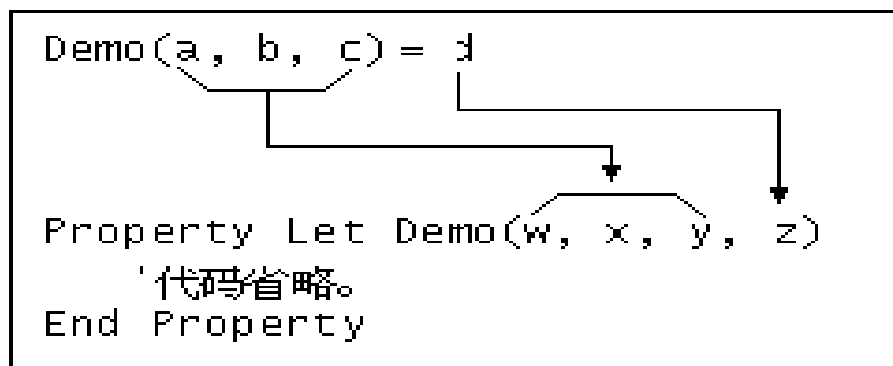


图 9.8 Property Let 过程的调用

带多参数的属性过程最常用于创建属性数组。

只读属性

要创建只读属性，只需简单地省略 Property Let 或（对于对象属性）Property Set。

对象属性

若要创建一个读写对象属性，应使用 Property Get 和 Property Set 如：

```
Private mwdgWidget As Widget
```

```
Public Property Get Widget() As Widget
```

```
    'Set 语句被用来返回一个对象引用。
```

```
    Set Widget = mwdgWidget
```

```
End Property
```

```
Public Property Set Widget(ByVal NewWidget As Widget)
```

```
    Set mwdgWidget = NewWidget
```

```
End Property
```

Variant 属性

Variant 数据类型的读写属性是最复杂的。如下所示，使用三种属性过程类型：

```
Private mvntAnything As Variant
```

```
Public Property Get Anything() As Variant
```

```
    'Set 语句只用于包含了对对象引用的任何属性。
```

```
    If IsObject(mvntAnything) Then
```



```

        Set Anything = mvntAnything
    Else
        Anything = mvntAnything
    End If
End Property

Public Property Let Anything(ByVal NewValue As Variant)
    ' (省略验证代码。)
    mvntAnything = NewWidget
End Property

```

```

Public Property Set Anything(ByVal NewValue As Variant)
    ' (省略验证代码。)
    Set mvntAnything = NewWidget
End Property

```

在正确环境中，**Property Set** 和 **Property Let** 总是很正常的使用。但是，使用 **Property Get** 必须考虑以下两种情况：

```

strSomeString = objvar1.Anything
Set objvar2 = objvar1.Anything

```

在第一种情况中，**Anything** 属性包含了字符串，它赋给了 **String** 变量。在第二个情况中，**Anything** 属性包含了对象引用，它被赋给对象变量。通过使用 **IsObject** 函数，在返回值之前测试私有 **Variants** 变量，可对

Property Get 进行编程来处理这两种情况。

自然，当属性包含对象引用时，如果第一条编码被调用，将会出现错误，但这不是 Property Get 问题，而是使用 Variants 属性的问题。

仅写一次属性

属性过程有几种可能的组合。所有的组合都是有效的，但有些相对来说不太常用，比如只写属性（只有 Property Let，而无 Property Get）。有些则取决于另外的因素，而不是所组合的属性过程类型。

例如，如本章后“对象模型”所描述的那样，当通过创建一个对象模型在程序中组织对象时，可能想让该对象能引用包含它的对象。使用 Parent 属性可这样做。

当对象创建时，应设置 Parent 属性，但此后，可能想防止其被有意或无意地改变。以下示例显示了 Account 对象如何使用 Parent 属性。该属性指向含有帐号的 Department 对象。

'Parent 属性要用私有数据存储

```
Private mdeptParent As Department
```

```
Property Get Parent() As Department
```

 '使用 Set 语句引用对象。

```

        Set Parent = mdeptParent
End Property

'属性值只能被设置一次。
Public Property Set Parent(ByVal NewParent _
As Department)
    If deptParent Is Nothing Then
        '赋初值。
        Set mdeptParent = NewParent
    Else
        Err.Raise Number:=vbObjectError + 32144, _
        Description:="Parent property is read-only"
    End If
End Property

```

当访问 Account 对象的父对象时，例如通过代码 `strX = acctNew.Parent.Name` 获取部门名称，Property Get 被用于返回对父对象的引用。

该示例中，编入 Property Set 是为了使 Parent 属性只能被设置一次。例如，当 Department 对象创建一个新的帐号时，可以执行代码 `Set acctNew.Parent = Me` 设置属性。此后，该属性为只读的。

详细信息 因为在 Visual Basic 中窗体是类，故可向窗体添加用户属性。请参阅本章前面“定制窗体类”的内容。

向类中添加方法

类的方法就是所声明的 `Sub` 或者 `Function` 公用过程。既然按照缺省规定 `Sub` 过程和 `Function` 过程都是公用的，那么在创建方法时就更加无需指明 `Public` 关键字。

例如，为了给 `Account` 类创建 `Withdrawal` 方法，可将下面的 `Public Function` 过程添加到类模块中：

```
Public Function Withdrawal(ByVal Amount As Currency, _  
    ByVal TransactionCode As Byte) As Double
```

```
    '（提款并返回新余额，或者产生透支错误的代码。）
```

```
End Function
```

提示 尽管并不需要键入 `Public` 关键字，但是这样做是一种好的编程习惯，可使以后维护代码的人明确其意图。

将方法声明为 `Public` 子过程

返回新余额是可选的，原因在于调用 `Withdrawal` 方法后，可再调用 `Account` 对象的 `Balance` 属性。这样就可以将 `Withdrawal` 代码编写为一个 `Public Sub` 过程。

提示 如果发现几乎每次调用 `Withdrawal` 时，都要调用 `Balance`，那么返回新余额会更有效一些。这是因为，就像在“添加类模块属性”中所说明的那样，任何属性访问，甚至读取一个公有的变量，都意味着一个函数调用——显式或者隐式的 `Property Get`。

重点 下面的名字不能作为属性名或方法名，因为它们属于下一级的 `IUnknown` 和 `IDispatch` 接口：`QueryInterface`，`AddRef`，`Release`，`GetTypeInfoCount`，`GetTypeInfo`，`GetIDsOfNames` 和 `Invoke`。这些名字将引起编译错误。

详细信息 关于 `Sub` 和 `Function` 过程的详细信息，请参阅第五章“编程基础”中的“过程概述”。

保护实现的细节

类的公共接口是由类模块的属性和方法声明来定义的。同数据隐藏一样，声明为 `Private` 的过程也不是接口的组成部分。这意味着可以更改类模块内部使用的实用程序的过程，而不影响使用该对象的代码。

更重要的是，也可以更改方法实现的内部公用 `Sub` 或者 `Function` 过程的代码，而不影响使用该方法的代码。只要不改变过程参数的数据类型，或者不改变 `Function` 过程所返回的数据类型，接口就不会改变。

将对象的实现细节隐藏在接口之后，是封装的另一方面。封装可以增强方法的性能，或者完全改变方法的实现途径，而不必对使用该方法的代码进行更改。

注意 接口元素命名指南——这在“命名属性、方法和事件”中作了讨论——不仅仅适用于属性名和方法名，而且适用于定义方法的 `Sub` 和 `Function` 过程中的参数名。当在“对象浏览器”中查看这些方法时，这些参数名是可见的，而且当调用该方法时，可以将之作为命名的参数来使用（即，`parametername:=value`）。

详细信息 命名的参数是在第五章“编程基础”中的“向过程传递参数”中引入的。

将方法添加到窗体类中，是一种功能强大的编程技巧，这在“定制窗体类”中作了讨论。

有时，某个成员应该选择属性还是方法，是不太清楚的。下面的“选择属性还是方法？”提供了指南。

是属性还是方法

一般来说，属性是关于某个对象的数据，而方法则是该对象可能被要求去执行的动作。有些的属性比较明显，比如 `Color` 和 `Name`，而另一些则明显的是方法，比如 `Move` 和 `Show`。

不过，就跟人类行为的各个方面一样，这里也存在一个界限不清的区域，在该区域一个参数可作为二者之中的任一个。

例如，为什么 `Visual Basic` 的 `Collection` 类的 `Item` 方法是一种方法而不是一种索引属性呢？该集合中的这些项不恰恰是数据吗？一个设想的 `Widgets` 集合类的 `Item` 方法可能以任一种方式实现，如下所示：

' `Widgets` 集合中对象的私有存储（这对于两种实现方法来说是相同的）。

```
Private mcol As New Collection
```

```
Public Property Get Item(Index As Variant) As Widget
```

```
    Set Item = mcol.Item(Index)
```

```
End Function
```

- 或 -

```
Public Function Item(Index As Variant) As Widget
```

```
    Set Item = mcol.Item(Index)
```

```
End Function
```

这两种实现并非完全大相径庭。二者都是只读的，因此为了把 `Widget` 对象放到集合中，二者都依赖于 `Widgets` 类的 `Add` 方法。二者都将每一件事情委派给 `Collection` 对象去做——甚至它们的错误也都是由该 `Collection` 产生的！

详细信息 关于委派，在本章的后面的“代码重用的许多（内部）方面”和“创建自己的集合类”中作了解释。

在想方设法搞清某个成员是对象的数据，还是对象的行为的过程中，可能会真地变得吹毛求疵。例如，可能会争辩说那个 `Item` 是一种方法，因为该集合的确是在做什么事——浏览所需要的 `Widget`。不过，这种论点通常对任何一种实现来说，都是有道理的。

可能会发现将这种论点转向其头部，并问问自己想怎样看待这些成员，这样会更有用一些。如果希望人们把它当作关于对象的数据来看待，就使之成为一种属性。如果希望人们把它作为对象所做的什么事情来看待，就使之成为一种方法。

从语法的角度

用属性过程来实现成员的一个很强的理由，依赖于想要在代码中使用该成员的方式。也就是说，允许 `Widgets` 集合的用户象下面这样编写代码吗？

```
Set Widgets.Item(4) = wdgMyNewWidget
```

如果允许的话，那么就以读写属性来实现该成员（用“`Property Get`”和“`Property Set`”），因为方法不支持这种语法。

注意 在所碰到的大多数集合实现中，这种语法是不允许的。为集合实现 `Property Set` 并不像看起来那么容易。

从“属性”窗口的角度

也可以暂时假定对象像一个控件。能想像到成员在“属性”窗口中，或者在某个属性页上显示吗？如果那样做并没有任何意义的话，就不要作为一种属性来实现该成员。

从可以感觉到错误的角度

已经将某个 `Item` 设置为只读的属性后，如果忘记了这一点，并试图将一个值赋给它，很可能就会发现 `Visual Basic` 为某个 `Property Get` 所产生的错误信息——“不能给只读属性赋值”，这跟为某个 `Function` 过程所产生的错误信息——“赋值号左边的函数调用必须返回变体或对象”相比较，理解起来要更容易一些。

作为最后一招

作为最后一招，往上抛硬币来作出决定。如果在本节主题中的其它参数看起来都不是那么让人确信的话，那么它就可能没有多大意义。

详细信息 属性过程是在本章前面的“向类中添加属性”引入的。而方法则在“向类中添加方法”作了讨论。

使属性或方法成为缺省的

可以为自己的类所创建的对象赋以缺省的属性，就像 Visual Basic 所提供的对象缺省属性一样。缺省成员的最佳选择，是最常使用的那个。

要将某个属性或者方法设置为缺省，请按照以下步骤执行：

1. 在“工具”菜单上，选择“过程属性”选项，打开“过程属性”对话框。
2. 单击“高级”，扩展“过程属性”对话框。
3. 在“名称”框中，选择类的当前缺省的属性或者方法。如果类当前没有任何缺省的成员，那么跳到第 5 步。

注意 可以用“对象浏览器”来找出某个类当前缺省的成员是什么。当在“Classes”列表选择了类时，可以滚动“Members”列表中的成员；缺省的成员将在其图标旁边用一个蓝色小球作标记。

4. 在“过程标识符”框中，选择 **None** 来删除属性或者方法的缺省状态。
5. 在“名称”框中，选择想使之成为新的缺省项的属性或方法。
6. 在“过程标识符”框中，选择“缺省”，然后单击“确定”。

重点 一个类只可能有一个缺省成员。如果已经将某个属性或者方法标记为缺省，那么在将另一属性或者方法设置为缺省项之前，必须将原来的过程标识符复位为 **None**。如果有两个成员被标记为缺省的，那么不会出现任何编译错误，但是没有办法来预测 Visual Basic 将选择哪一个来作为缺省项。

也可以从“对象浏览器”中打开“过程属性”对话框。这在改变某个类的缺省成员时是很方便的，因为这样可以迅速定位现有的缺省成员。

要用“对象浏览器”来改变缺省属性，请按照以下步骤执行：

1. 按 **F2** 键，打开“对象浏览器”。
2. 在“类”列表中，选择想改变缺省成员的类。
3. 在“成员”列表中，右键单击图标旁边有蓝色小球的成员，打开上下文菜单。单击“属性”来显示“属性特性”对话框。
4. 单击“高级”，扩展“过程属性”对话框。
5. 在“过程标识符”框中，选择 **None** 来删除属性或者方法的缺省状态，然后单击“确定”。
6. 在“成员”列表中，右键单击想使之成为新缺省项的成员，来打开上下文菜单。单击“属性”，显示“属性特性”对话框。
7. 单击“高级”，扩展“过程属性”对话框。
8. 在“过程标识符”框中，选择“缺省”，然后单击“确定”。

注意 不能用“过程属性”对话框改变 Visual Basic 所提供类的缺省成员。

修改声明为 **Private** 或 **Friend** 缺省值

“过程属性”对话框仅允许选择公有的属性和方法作为类的缺省。如果您使一个公有的属性或方法成为类的缺省，而后来又改变声明为 **Private** 或 **Friend**，则该属性或方法的行为会仍然与声明为 **Public** 时一样。

要解决这个问题，必需重新声明该属性或方法为 **Public**，因为“过程属性”对话框不会显示声明为 **Private** 和 **Friend** 的过程。一旦把声明改回为 **Public**，您就可以使用“过程属性”对话框来删除缺省特性。然后，就可以把声明改回为 **Friend** 或 **Private** 了。

友元属性和方法

除了将属性和方法声明为 **Public** 和 **Private** 以外，也可以将它们声明为 **Friend**。**Friend** 成员对于工程中其它对象来说，看起来就像 **Public** 成员。也就是说，它们以类的接口部分的形式出现。但它们并不是类接口的一部分。

在用 Visual Basic 的专业版和企业版可以创建的 ActiveX 部件中，**Friend** 成员起了重要作用。因为它们并非某个对象的接口的一部分，所以不能被使用该部件对象的程序所访问。不过，对于该部件内所有其它对象来说，它们都是可见的，因此它们允许在该部件内进行安全的内部通信。

重点 因为 Friend 成员并非某个对象公用接口的一部分，它们不能被后期绑定访问——也就是说，通过 As Object 声明为对象的变量不能访问。为了使用 Friend 成员，必须用前期绑定声明的变量——也就是说，As classname。

“标准 EXE”工程不可能是 ActiveX 部件，因为它们的类模块不可能是 Public，所以不能被其它应用程序使用。因此在一个“标准 EXE”工程中，对象间所有通信都是私有的，并不需要 Friend 成员。

不过，Friend 成员有个特别有用的特点。因为它们不是某个 ActiveX 接口的一部分，它们不能用来在对象间传递用户定义类型。例如，在下面的标准模块中，假设有用户定义类型：

```
Public Type udtDemo
    intA As Integer
    lngB As Long
    strC As String
End Type
```

可以在 Class1 中定义下面的私有变量和 Friend 成员：

```
Private mDemo As udtDemo

Friend Property Get Demo() As udtDemo
    Demo = mDemo
End Property
```

注意 udtDemo 必须通过引用来传递。

```
Friend Property Let Demo(NewDemo As udtDemo)
```

```
    mDemo = NewDemo
```

```
End Property
```

```
Friend Sub SetDemoParts(ByVal A As Integer, _
```

```
    ByVal B As Long, ByVal C As String)
```

```
    mDemo.intA = A
```

```
    mDemo.lngB = B
```

```
    mDemo.strC = C
```

```
End Sub
```

```
Public Sub ShowDemo()
```

```
    MsgBox mDemo.intA & vbCrLf _
```

```
    & mDemo.lngB & vbCrLf & mDemo.strC
```

```
End Sub
```

注意 当传递 Sub、Function，或属性过程参数这样的用户定义类型时，必须通过引用来传递它们（ByRef 是过程参数的缺省值）。

然后可以写出下列代码使用 Class1：

```
Private Sub Command1_Click()
```

```
    Dim c1A As New Class1
```

```
    Dim c1B As New Class1
```

```
    c1A.SetDemoParts 42, 1138, "Howdy"
```

```
c1B.Demo = c1A.Demo
```

```
c1B.ShowDemo
```

```
End Sub
```

信息框将显示 42、1138 和 "Howdy"。

注意 因为 Friend 过程并非类接口的一部分，所以当使用 Implements 语句来实现多接口时，它们并不包括在里面，如下面“多态”中所描述的。

详细信息 关于 Friend 成员在部件中的使用，在《部件工具指南》的第六章“部件设计的一般准则”中的“对象之间的私有通信”中作了讨论。

向类中添加事件

现在，假定已经用 Stegosaur、Triceratops 和 Tyrannosaur 类创建了一个恐龙模拟。最后要完成的是，想让 Tyrannosaur 咆哮，并且当它这样做的时候，想让这个模拟中的其它每只恐龙都突然警觉起来。

如果 Tyrannosaur 类有一个 Roar 事件，就可以在所有其它恐龙类中处理该事件。本节主题讨论的就是，在类模块中事件的声明和处理。

注意 不要在家里试验这些，至少不要用许多恐龙来作试验。若是用事件将每只恐龙跟其它每一只连接起来，可能会使得恐龙非常之慢，以至于哺乳类对象将统治这个模拟。

有人说属性和方法属于入端接口，因为它们是从对象外面调用的。相对而

言，事件被叫做出端接口，因为它们是在对象里边产生，在其它地方处理。

详细信息 随专业版和企业版一起提供的《部件工具指南》中的“创建 ActiveX 部件”，讨论了设计自己的软件部件时事件的使用问题。

关于处理恐龙问题的更好办法的讨论，请参阅本章后面的“多态”。

声明和引发事件

暂且假设有一个 `Widget` 类。该 `Widget` 类有一种方法，该方法可能花费很长时间来执行，同时还希望自己的应用程序能够显示出一些类型的完成标志。

当然，可以让 `Widget` 对象显示一个用来表示完成百分比的对话框，但是以后在每个使用 `Widget` 对象的工程中，就会都离不开那个对话框了。对象设计的一个较好的原则是让使用该对象的应用程序来处理用户接口——除非设计该对象的全部意图是管理一个窗体或者一个对话框。

`Widget` 的作用是执行其它任务，因此合理的做法是：把一个 `PercentDone` 事件给它，然后让调用 `Widget` 方法的过程来处理那个事件。`PercentDone` 事件也可以提供一种机制来取消任务。

现在，可以开始为该主题来建立代码示例了，方法是打开一个“标准 EXE”工程，将两个按钮和一个标签添加到“`Form1`”中。在“工程”菜单上，选择“添加类模块”，将一个类模块添加到工程中。像下表中所示的那样给对象命名。

对 象	属 性	设 置 值
类 模 块	Name	Widget
第 一 个 按 钮	Caption	启 动 任 务
第 二 个 按 钮	Caption	取 消
标 签	NameCaption	lblPercentDone"0"

Widget 类

在类模块的声明部分，用 `Event` 关键字来声明事件。事件可以有 `ByVal` 和 `ByRef` 参数，就像 `Widget` 的 `PercentDone` 事件所演示的那样：

```
Option Explicit
```

```
Public Event PercentDone(ByVal Percent As Single, _
```

```
ByRef Cancel As Boolean)
```

当调用对象得到 `PercentDone` 事件时，`Percent` 参数包含了任务完成的百分比。可以将 `ByRef Cancel` 参数设置为 `True`，以取消引发该事件的方法。

注意 可以像声明过程参数那样声明事件参数，所不同的是：事件不能有命名的参数、可选的参数或 `ParamArray` 参数。事件没有返回值。

引发 `PercentDone` 事件

`PercentDone` 事件是由 `Widget` 类的 `LongTask` 方法引发的。`LongTask` 方法接受两个参数：该方法工作所要求的时间长度，以及 `LongTask` 在暂停之前产生 `PercentDone` 事件的最小时间间隔。

```
Public Sub LongTask(ByVal Duration As Single, _
```



```
ByVal MinimumInterval As Single)
```

```
Dim sngThreshold As Single
```

```
Dim sngStart As Single
```

```
Dim blnCancel As Boolean
```

'Timer 函数按 Single 返回午夜以来秒数的分数值。

```
sngStart = Timer
```

```
sngThreshold = MinimumInterval
```

```
Do While Timer < (sngStart + Duration)
```

'在实际应用程序中，每次循环时将会在这里

'做某些单元的工作。

```
If Timer > (sngStart + sngThreshold) Then
```

```
    RaiseEvent PercentDone( _
```

```
    sngThreshold / Duration, blnCancel)
```

'检查一下，看看操作是否被取消。

```
If blnCancel Then Exit Sub
```

```
    sngThreshold = sngThreshold + MinimumInterval
```

```
End If
```

```
Loop
```

```
End Sub
```

每隔 `MinimumInterval` 秒，都会引发 `PercentDone` 事件。当该事件返回时，`LongTask` 将检查 `Cancel` 参数是否设置为 `True`。

注意 为了简单起见，`LongTask` 假定事先已经知道任务将持续多长时间。但是，这几乎是不可能的。将任务分解成平均大小的块，可能是比较困难的，而且经常困扰用户的问题是，在他们得到某件事已经发生的指示前，真正经过的时间量。

处理对象的事件

引发事件的对象叫做事件源。为了处理事件源所引发的事件，可以用 `WithEvents` 关键字声明对象类的变量。

为了处理 `Widget` 的 `PercentDone` 事件，将下面的代码放置到 `Form1` 的声明部分：

```
Option Explicit
```

```
Private WithEvents mWidget As Widget
```

```
Private mblnCancel As Boolean
```

`WithEvents` 关键字指定：变量 `mWidget` 将用于处理对象的事件。可以通过提供类名来指定对象类型，该类是创建这个对象的类。

变量 `mWidget` 是在“`Form1`”的声明部分所声明的，因为 `WithEvents` 变量必须是模块级的变量。这是正确的，而不管将它们放置其中的模块类型如何。

变量 `mblnCancel` 将用于取消 `LongTask` 方法。

对 `WithEvents` 变量的一些限制

使用 `WithEvents` 变量时，应该注意下面这些限制条件：

- `WithEvents` 变量不能是派生对象变量。也就是说，不能把它声明为 `As Object`——当声明该变量时必须指定类名。
- 不能把 `WithEvents` 变量声明为 `As New`。必须明确地创建事件源对象，并将它赋给 `WithEvents` 变量。
- 不能在标准模块中声明 `WithEvents` 变量。只能在类模块、窗体模块以及其它定义类的模块中声明。
- 不能创建 `WithEvents` 变量数组。

编写处理事件的代码

一旦声明了 `WithEvents` 变量，变量名就出现在模块“代码”窗口左边的下拉菜单上。当选择了 `mWidget` 时，`Widget` 类的事件将出现在右边下拉菜单上，如下图 9.9 中所示。

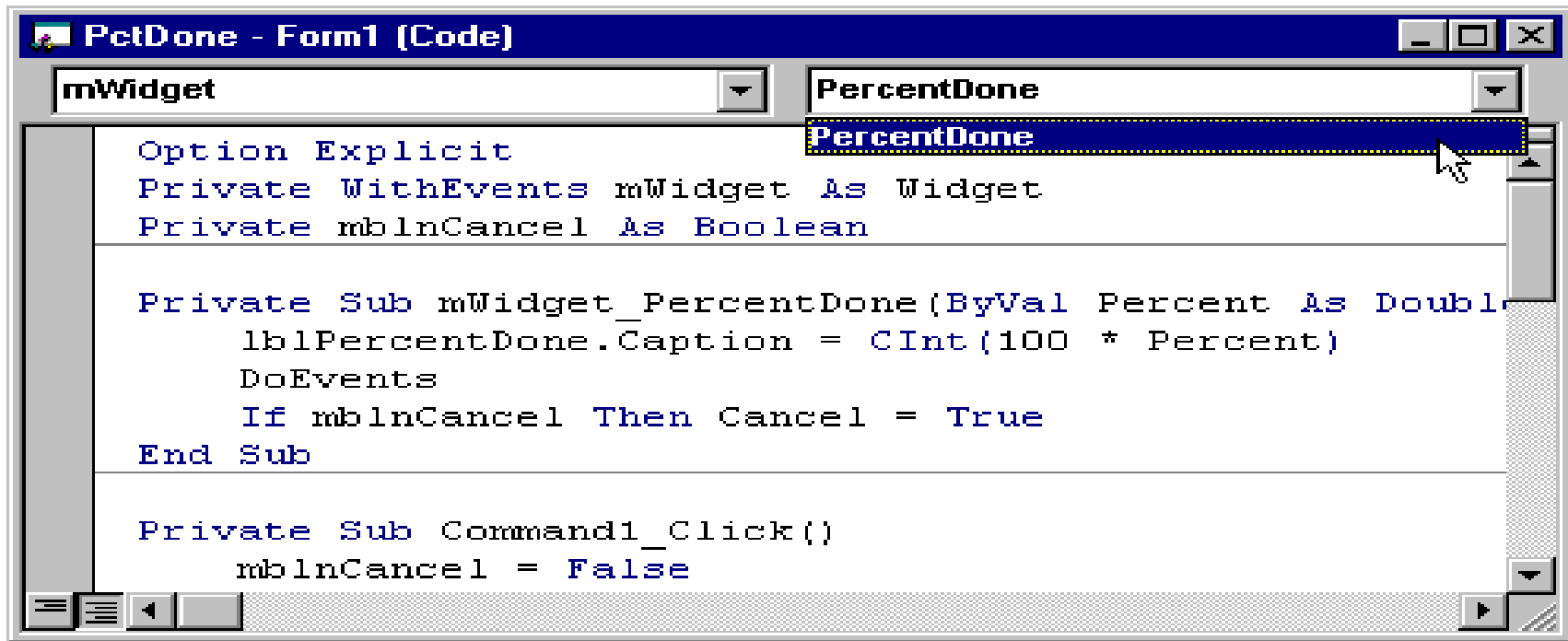


图 9.9 与 WithEvents 变量相关联的事件

选定一个事件，将显示相应的事件过程，以 `mWidget_` 为前缀。所有跟 `WithEvents` 变量相关联的事件过程，都将该变量名为前缀。将下面的代码添加到 `mWidget_PercentDone` 事件过程中。

```
Private Sub mWidget_PercentDone(ByVal Percent As _
Single, Cancel As Boolean)
    lblPercentDone.Caption = CInt(100 * Percent) & "% "
    DoEvents
    If mblnCancel Then Cancel = True
End Sub
```

```
End Sub
```

不论何时，引发 `PercentDone` 事件时，事件过程就在 `Label` 控件中显示完成的百分比。`DoEvents` 语句允许重新画出该标签，同时也给用户单击“取消”按钮的机会。将下面的代码添加到标题为“取消”按钮的 `Click` 事件中。

```
Private Sub Command2_Click()
```

```
    mblnCancel = True
```

```
End Sub
```

当 `LongTask` 正在运行时，如果单击了“取消”按钮，一旦 `DoEvents` 语句允许出现对事件的处理，那么将会执行 `Command2_Click` 事件。模块级的变量 `mblnCancel` 设置为 `True`，`mWidget_PercentDone` 事件将对其进行测试，并将 `ByRef Cancel` 参数设置为 `True`。

连接 **WithEvents** 变量与对象

`Form1` 已经设置了对 `Widget` 对象事件的处理。剩余工作就是寻找某个地方的 `Widget`。

在设计时，声明 `WithEvents` 变量时，并没有任何对象与之相关联。`WithEvents` 变量与任何其它对象变量相同。必须创建对象，并将对该对象的引用赋给这个 `WithEvents` 变量。将下面的代码添加到 `Form_Load` 事件过程中，以创建 `Widget`。

```
Private Sub Form_Load()
```

```
    Set mWidget = New Widget
```

```
End Sub
```

当执行上面的代码时，Visual Basic 将创建一个 Widget，并将它的事件跟 mWidget 相关联的事件过程连接起来。从那时起，一旦 Widget 产生了 PercentDone 事件，都将执行 mWidget_PercentDone 事件过程。

为了调用 LongTask 方法，将下面的代码添加到标题为“启动任务”按钮的 Click 事件中。

“启动任务”按钮。

```
Private Sub Command1_Click()
```

```
    mblnCancel = False
```

```
    lblPercentDone.Caption = "0% "
```

```
    lblPercentDone.Refresh
```

```
    Call mWidget.LongTask(14.4, 0.66)
```

```
    If Not mblnCancel Then lblPercentDone.Caption = 100
```

```
End Sub
```

在调用 LongTask 方法之前，完成百分比的显示标签必须被初始化，而且必须将模块级的 Boolean 标记（其作用是取消方法）设置为 False。

用 14.4 秒任务延迟调用 LongTask。每隔三分之二秒引发一次 PercentDone 事件。每次引发该事件时，都将执行 mWidget_PercentDone 事件过程。

LongTask 结束以后，都要测试 mblnCancel，以了解 LongTask 是否正常结束，或者是否因为 mblnCancel 设置为 True 而停止。只有在前一种情况下，

完成的百分比才会被更新。

运行程序

按 F5 键，就使工程处于运行模式。单击“启动任务”按钮。每次引发 PercentDone 事件，带有完成百分比的标签将会被更新。单击“取消”按钮，可以停止任务。

注意 单击“取消”按钮时，该按钮的外观并不会立即改变。直到 DoEvents 语句允许事件处理时，才引发 Click 事件。

可能会发现用 F8 键来运行程序是有益的——这样做的话，每次将执行一行代码。可以清楚地看到是怎样执行到 LongTask 的，然后在每次引发 PercentDone 事件时，能直接重新进入 Form1。

当执行回到 Form1 中的代码时，如果再次调用 LongTask 方法，将会发生什么样的情况？混乱、混沌，并最终（如果每次产生事件时都发生这种情况）出现堆栈溢出。

处理不同的 Widget 事件

通过将对新 Widget 的引用赋给 mWidget 变量，可以使得 mWidget 变量为不同 Widget 对象处理事件。事实上，每次单击按钮时，通过添加下面两行代码，可以让 Command1 中的代码来做这些：

```
Set mWidget = New Widget      ' <- 新行。
```

```
Call mWidget.LongTask(14.4, 0.66)
```

Set mWidget = Nothing ' <- 新行。

每次按下按钮时，上面的代码都将创建一个新的 Widget。一旦 LongTask 方法结束，对 Widget 的引用就会通过将 mWidget 设置为 Nothing 而释放，而且这个 Widget 也被撤销。

WithEvents 变量每次只能包含一个对象引用，因此如果将不同的 Widget 对象赋给 mWidget，先前 Widget 对象的事件将不会再得到处理。如果 mWidget 是唯一包含对旧 Widget 引用的对象变量时，那么该对象将被撤销。

注意 需要多少 WithEvents 变量，就可以声明多少，但是不支持 WithEvents 变量数组。

终止 WithEvents 变量事件的处理

只要有一个赋给变量 mWidget 的 Widget 对象，不论何时，当该 Widget 产生事件时，都将调用跟 mWidget 相关联的事件过程。为了终止事件处理，可以将 mWidget 设置为 Nothing，就像在下面代码块中所示。

'终止 mWidget 事件处理。

Set mWidget = Nothing

当把 WithEvents 变量设置为 Nothing 时，Visual Basic 将拆除对象的事件和与该变量相关联的事件过程的连接。

重点 WithEvents 变量与任何其它对象变量一样，包含一个对象引用。这个对象引用所起的作用，是保持该对象是活动的。当把对象的所有引用都

设置为 `Nothing` 以撤销它时，不要忘记声明为 `WithEvents` 的那个变量。

WithEvents 与窗体上的控件事件的比较

可能已经注意到，在使用 `WithEvents` 变量的方式和处理某个窗体上控件所产生的事件的方式这二者之间，存在某些相似之处。在两种情况下，当在代码窗口右边下拉菜单上选择事件时，都将得到包含事件正确参数的事件过程。

事实上，确切地说二者机制是相同的。控件是作为窗体类的属性来处理的，并且这个属性的名字，就是在“属性”窗口中赋给控件 `Name` 属性的那个值。

看起来似乎有一个与控件同名的 `Public` 模块级的变量，而且所有控件的事件过程名，都以这个变量的名字为起始，就像 `WithEvents` 变量情况下，它们都以 `WithEvents` 变量名为起始一样。

通过将 `mWidget` 变量声明为 `Public` 而不是 `Private`，可以很容易地看到这一点。这样做时，`mWidget` 将作为 `Form1` 的一种属性出现在“对象浏览器”中，像在另一种情况下，控件出现在窗体上一样。

两种情况的不同之处在于：当创建窗体时，`Visual Basic` 自动创建该窗体上所有控件的实例，而在另一种情况下，必须创建自己的类（这些类的事件是想要处理的）的实例，并且应把那些对象的引用赋给 `WithEvents` 变量。

向窗体添加事件

下面的循序渐进的过程显示了如何给窗体创建自定义事件。要试用此练习，可打开一个新的“标准 EXE”工程，并按如下步骤去做：

要将事件添加到 Form1 中，请按照以下步骤执行：

1. 在“工程”菜单上，选择“添加类模块”，将类模块添加到工程中。将下面的代码放置到 Class1 的“声明”部分：

```
Public Property Get Form1() As Form1  
    Set Form1 = mForm1  
End Property
```

```
Public Property Set Form1(ByVal NewForm1 As Form1)  
    Set mForm1 = NewForm1  
End Property
```

如果使用过程视图，则不能同时查看属性过程。单击代码窗口左下角的“全模块视图”按钮，转换到全模块视图。单击过程视图旁边的“过程视图”按钮，可以返回过程视图（将鼠标在各个按钮上移动一下，看看哪个是哪个）。

2. 将下面的代码添加到 Form1 的“声明”部分：

```
Event Gong  
Private mc1 As Class1
```

因为已经创建了 `Class1`，所以创建 `Class1` 类型的变量是可能的。这个过程要在 `Form1` 和 `Class1` 之间切换几次，因为在模块中的操作步骤，需要首先给另一个模块添加代码。

3. 返回 `Class1`，并将下面的代码添加到“声明”部分。

```
Private WithEvents mForm1 As Form1
```

就像在“向类中添加事件”所讨论的那样，`WithEvents` 关键字意味着 `Form1` 的这个实例是与事件相关联的。注意，一旦已经创建了 `Gong` 事件，这一步成为可能。

4. 在 `Class1` 的“代码”窗口上的左边“对象”下拉菜单上，选择“`mForm1`”来为 `Gong` 事件得到事件过程。将下面的代码添加到该事件过程中：

```
Private Sub mForm1_Gong()
```

```
    MsgBox "Gong!"
```

```
End Sub
```

5. 返回 `Form1`。在“对象”下拉菜单上，选择“添加窗体”，在右边“过程”下拉菜单上，选择“加载”。将下面的代码添加到事件过程中：

```
Private Sub Form_Load()
```

```
    Set mc1 = New Class1
```

```
    Set mc1.Form1 = Me
```

```
End Sub
```

第一行创建了一个 `Class1` 对象，而第二行则将对 `Form1`（也就是说，是 `Me`——在 `Form1` 的“代码”窗口中，`Me` 指的是 `Form1`；而在 `Class1` 的“代码”窗口中，`Me` 指的是 `Class1`）的引用赋给其 `Form1` 属性（在

第一步所创建的)。

6. 将三个文本框放置到 `Form1` 上。在“对象”和“过程”下拉菜单上，依次为每个控件选择“`Change`”事件过程，并将相同代码行放置到它们中的每个里面：

```
Private Sub Text1_Change()  
    RaiseEvent Gong  
End Sub
```

每次当某个文本框的内容改变时，都将引发窗体的 `Gong` 事件。

7. 按 `F5` 键，运行工程。每次在其中一个文本框内键入字符时，信息框都将响铃。这是很令人恼火的，但它显示了怎样将一个事件添加到窗体中，然后从几个控件得到通知。

就像在上面的“声明和引发事件”所述，可以将参数添加到事件中。例如，可能将控件名——或者更好是用对控件的引用——传递给事件的接收者。

声明、引发以及处理事件的总结

为了将某个事件添加到一个类中，然后使用该事件，必须这样做：

- 在定义类的类模块声明部分，用 `Event` 语句来声明事件——该事件带有希望它带有的任何参数。事件总是 `Public`。

注意 事件不能有命名的参数、可选的参数或 `ParamArray` 参数。事件没有返回值。

- 在类模块代码中的合适地方，用 `RaiseEvent` 语句来引发事件，并提供所需要的参数。
- 在将要处理事件的模块声明部分，使用 `WithEvents` 关键字，添加该类类型的变量。它必须是一个模块级的变量。
- 在代码窗口左边的下拉菜单上，选择声明为 `WithEvents` 的变量。
- 在代码窗口右边的下拉菜单上，选择希望处理的事件（可以为类声明多个事件）。
- 使用所提供的参数，将代码添加到事件过程中。

创建数据识别类

如果已经阅读了前面有关创建类的内容，那么现在您就应该知道，类是一个封装了数据和代码的对象，而类的属性就是描述对象的数据。而且还知道可以使用属性过程或者公用属性来显露那些属性所表示的数据。

一切顺利——到目前为止，所有示例都已经处理过临时性的数据，即在运行时被创建和使用的数据。对于许多程序来说，这可能就是用户所需要的一切东西，但如果用户需要在过程之间存储数据或者利用已存在于用户程序之外的数据，那该怎么办呢？为了处理外部数据源，用户需要使自己的类成为数据识别类。

数据识别类可分为两种——数据使用者和数据源。类模块有两种设计时属性，即 `DataBindingBehavior` 和 `DataSourceBehavior`，它们定义了某个类是

如何和外部数据交互作用的。BindingCollection 对象被用来将数据识别类绑定到控件上，或者互相绑定。

数据源

数据源是一个从外部源提供数据的类，这些数据将被其他对象所使用。Data 控件实际上是一个作为数据源的类的实例，但是，已被设置为数据源的类比起一个 Data 控件来功能要强大的多。与 Data 控件不同，数据识别类不是必须有可视的外在表示，也不局限于某个特定的数据接口——如 Data Access Objects (DAO) 或者 Remote Data Objects (RDO)。事实上，数据识别类可以作为任何类型数据的数据源，包括传统的 ODBC 源、ActiveX Data Objects (ADO)，或者任何 OLE DB 提供程序。

DataSourceBehavior 属性决定了某个类是否可以作为数据源。将 DataSourceBehavior 设置为 1 (vbDataSource)，则类就可以作为其他对象的数据源。

数据使用者

简单说来，数据使用者是这样一类——它可被绑定到某个外部数据源上，在很大程度上就像 TextBox 控件可以被绑定到 Data 控件上一样。在 Visual Basic 的早期版本中，控件是唯一可以被绑定到数据源上的对象。设置一个数据识别类为数据使用者，允许您将任何对象绑定到被设置为数据源的类所创建的任何对象上去。

`DataBindingBehavior` 属性允许类绑定到外部数据。设置该属性为 1 (`vbSimpleBound`), 则该类所创建的对象将被绑定到外部数据源中的某个单个数据字段。将 `DataBindingBehavior` 设置为 2 (`vbComplexBound`), 该类将被绑定到外部数据源的一行数据上。可以这样想——如果您的对象是控件, 那么一个 `TextBox` 控件将是一个简单绑定, 而一个网格控件将是一个复杂绑定。

BindingCollection 对象

就像您需要通过 `Data` 控件把一个控件绑定到某个数据库上一样, 数据识别类需要一个中心对象将它们绑定到一起。这个对象就是 `BindingCollection` 对象。顾名思义, `BindingCollection` 是某个数据源和一个或多个数据使用者之间的绑定的集合。

要使用 `BindingCollection` 对象, 您必须首先添加对 `Microsoft Data Binding Collection` 的引用, 方法是在“工程”菜单上的“引用”对话框中选择之。对于任何一个对象, 需要在运行时创建 `BindingCollection` 对象的一个实例。

`BindingCollection` 对象的 `DataSource` 属性是用来指定将提供数据的对象的。该对象必须是一个类或者 `UserControl`, 而且其 `DataSourceBehavior` 属性要设置为 `vbDataSource`。

一旦 `BindingCollection` 已被实例化, 而且设置了 `DataSource`, 就可以使用 `Add` 方法来定义绑定关系了。`Add` 方法有三个必需的参数: 使用者对象名、被绑定到源的对象的属性, 以及将被绑定到该属性的源中的字段。您可以

将多个绑定添加到 `BindingCollection` 中，方法是重复使用 `Add` 方法，可以使用 `Remove` 方法删除一个绑定。

详细信息 关于一步一步创建数据识别类的示例，请参阅“创建数据源”和“创建数据使用者”。

创建数据源

在本节中，我们将一步一步地学习创建作为数据源的数据识别类的过程。本例将绑定一个 `TextBox` 控件到数据源类中，以显示数据。下一节“创建数据使用者”，将演示如何将数据源类绑定到某个数据使用者类。

本节中示例的代码来自于数据识别类示例(`Dataware.vbp`)。用户 `Samples` 目录中找到它。

创建一个数据源的过程分为两步。第一步将创建数据源类；第二步将把它挂接到某个 `TextBox` 控件上，以显示输出。

创建源类

创建源类的第一步是定义一个新类，并给予其必要的属性和方法以提供数据：

1. 打开一个新的标准 `EXE` 工程，并插入一个类模块，方法是在“工程”菜单中选择“添加类模块”。
2. 在“属性”窗口中，对类的属性进行如下设置：

属 性	设 置 值
Name	MySource
DataSourceBehavior	VbDataSource

当 DataSourceBehavior 被设置为 vbDataSource 时，一个新的 Sub 过程 GetDataMember 就被添加到类模块中。用户只要在代码编辑器中选择“对象”列表中的“类”，然后选择“事件”列表，即可看到此情况。

3. 选择“工程”菜单的“引用”，然后添加对 Microsoft ActiveX Data Objects 2.0 Library 的引用。
4. 将下面的代码添加到类模块的“声明”部分：

```
Option Explicit
Private rs As ADODB.Recordset
```

这声明一个对象变量为 ADO Recordset 对象。

5. 将下面的代码添加到类模块的 Initialize 事件过程中：

```
Private Sub Class_Initialize()
    Dim strPath As String, strName As String
    Dim i As Integer
    ' 创建 Recordset 的一个实例。
    Set rs = New ADODB.Recordset
    ' 设置 Recordset 的属性。
    With rs
        .Fields.Append "DirID", adInteger
        .Fields.Append "Directory", adBSTR, 255
```

```

        .CursorType = adOpenStatic
        .LockType = adLockOptimistic
        .Open
End With

' 在各目录中循环，并充填 Recordset
strPath = "C:\"
strName = Dir(strPath, vbDirectory)
i = 0
Do While strName <> ""
    If strName <> "." And strName <> ".." Then
        If (GetAttr(strPath & strName) And _
            vbDirectory) = vbDirectory Then
            i = i + 1
            With rs
                .AddNew
                .Fields.Item("DirID") = i
                .Fields.Item("Directory") = strName
                .Update
            End With
        End If
    End If
End While

```

```

        strName = Dir
    Loop

    ' 返回到第一个记录。
    rs.MoveFirst
End Sub

```

在本例中用户将迅速创建一个 ADO Recordset 对象，并分布一个目录列表来充填之。此外，用户也可以使用一个现存的 recordset，方法是把它分配 Initialize 事件中 ADO Recordset 的 Connect 属性。

6. 在“代码编辑器”中选择“对象”列表的“类”，然后选择“事件”列表中的 GetDataMember。将下面的代码添加到 GetDataMember Sub 过程中：

```

Private Sub Class_GetDataMember(DataMember As String, Data As Object)
    ' 将 Recordset 分配给 Data 对象。
    Set Data = rs
End Sub

```

GetDataMember 过程设置类的数据源。数据源类可以提供多个数据源，方法是将一个 Select Case 语句添加到 GetDataMember 过程中，并在 DataMember 参数中传入一个源名。

7. 将一个新的 Sub 过程，以提供在 Recordset 中循环的公共方法：

```

Public Sub Cycle()

```

```

' 在 Recordset 中循环。
rs.MoveNext
If rs.EOF = True Then
    rs.MoveFirst
End If
End Sub

```

为了在 recordset 中移动，用户需要显露所创建类的浏览方法。为了简单起见，本示例只能在 recordset 中向前循环。为了使得该类更加实用，您可能想要显露像 MoveFirst、MoveNext、Add，以及 Delete 这样的方法。

使用源类

现在已对源类作了定义，我们可以用它来做些有用的事情了。在本例中，我们将把它绑定到某个 TextBox 控件上，这样就可以看到其输出了；我们还将使用一个 CommandButton 来执行我们的 Cycle 方法。

1. 选择 Form1，并添加一个 TextBox 控件和一个 CommandButton 控件到窗体中。
2. 在“属性”窗口中，如下设置 TextBox 的属性：

属 性	设 置 值
Name	TxtConsumer
Text	(blank)

3. 在“属性”窗口中，如下设置 CommandButton 的属性：

属 性	设 置 值
Name	CmdCycle
Caption	Cycle

- 选择“工程”菜单中的“引用”，并添加对 Microsoft Data Binding Collection 的引用。

Data Binding Collection 所提供的 DataBinding 对象是将一个数据源绑定到某个数据使用者的“粘合剂”。

- 将下面的代码添加到类模块的“声明”部分：

```
Option Explicit
```

```
Private objSource As MySource
```

```
Private objBindingCollection As BindingCollection
```

需要用事前绑定来声明源类(MySource)和 BindingCollection 对象。

- 将下面的代码添加到 Form_Load 事件过程中：

```
Private Sub Form_Load()
```

```
    Set objSource = New MySource
```

```
    Set objBindingCollection = New BindingCollection
```

```
    ' 将源类分配给 BindingCollection 的 DataSource 属性。
```

```
    Set objBindingCollection.DataSource = objSource
```

```
    ' 添加一个绑定。
```

```
    ObjBindingCollection.Add txtConsumer, "Text", "Directory"
```

在 Load 事件中，我们创建了源类和 BindingCollection 对象的实例，然后

将源对象分配给 `BindingCollection` 的 `DataSource` 属性。最后，我们添加一个绑定，方法是指定使用者名 (`txtConsumer`)、被绑定的使用者的属性（即 `Text` 属性），以及要绑定到的源对象的 `Field` 属性 (`Directory`)。

7. 将下面的代码添加到 `cmdCycle Click` 事件过程中：

```
Private cmdCycle_Click()  
    ' 调用数据源的 Cycle 方法。  
    ObjSource.Cycle  
End Sub
```

这将执行源类的 `Cycle` 方法。

8. 按下 `F5` 键，运行该工程。

单击“`Cycle`”按钮时，源类所创建的 `recordset` 的目录名将出现在 `TextBox` 中。祝贺您——您刚刚在未使用 `Data` 控件的情况下，将一个控件绑定到某个数据源类上！

9. 保存工程。当提示输入文件名时，使用下面的名字。

将源类保存为 `"MySource.cls"`。

将窗体保存为 `"Dataform.frm"`。

将工程保存为 `"Dataware.vbp"`。

这些文件都将在后面的“创建数据使用者”中用到。

在下一节“创建数据使用者”中，我们将讨论创建一个作为数据使用者的数据识别类的过程。

创建数据使用者

在本节中，我们将一步一步地学习创建作为数据使用者的数据识别类的过程。前一节“创建数据源”演示了如何创建一个可绑定数据使用者的数据源。本示例显示了如何创建一个数据使用者类并将之绑定到前一节所创建的数据源上。

本节示例中的代码来自于数据识别类示例(Dataware.vbp)。您可以在 Samples 目录下找到它。

将一个数据使用者绑定到某个数据源对象

本例演示了如何创建一个数据使用者类，并将之绑定到某个数据源类。它使用了“创建数据源”中所创建的 MySource 类。

1. 打开 Dataware.vbp 工程。（在“文件”菜单中选择“打开工程”。）

注意 如果您先前没有完成“创建数据源”示例，那么将看不到这个工程。您也可以找到 Dataware.vbp 工程的一个已完成版本。

2. 插入一个新的类模块，方法是选择“工程”菜单中的“添加类模块”。

3. 在“属性”窗口中，如下设置新类的属性：

属性	设置值
Name	MyConsumer
DataBindingBehavior	VbSimpleBound

4. 将下面的代码添加到类模块的“声明”部分：

```
Option Explicit
```

```
Private mDirectory As String
```

5. 为公共 DirName 属性添加一对 Property Get / Property Let 过程：

```
Public Property Get DirName() As String
```

```
    DirName = mDirectory
```

```
End Property
```

```
Public Property Let DirName(mNewDir As String)
```

```
    mDirectory = mNewDir
```

```
    ' 在立即窗口中显示新值。
```

```
    Debug.Print mDirectory
```

```
End Property
```

因为 MySource 不是一个可视的类，所以需要在 Property Let 过程中用一条 Debug.Print 语句来证明它正在检索数据源中的新值。

6. 选择 Form1，并将下面的代码添加到“声明”部分：

```
Option Explicit
```

```
Private objSource As MySource
```

```
Private objBindingCollection As BindingCollection
```

```
Private objConsumer As MyConsumer
```

新的代码行添加一个对使用者类的引用。

7. 将下面的代码添加到 Form_Load 事件过程中：

```
Private Sub Form_Load()
```

```
    Set objSource = New MySource
```



```
Set objBindingCollection = New BindingCollection
```

```
Set objConsumer = New MyConsumer
```

```
' 将源类分配给 BindingCollection 的 DataSource 属性。
```

```
Set objBindingCollection.DataSource = objSource
```

```
' 添加一个绑定。
```

```
objBindingCollection.Add txtConsumer, "Text", "Directory"
```

```
objBindingCollection.Add objConsumer, "DirName", "Directory"
```

新代码创建客户类的一个实例，并把它添加到 BindingCollection 中，同时将使用者的 DirName 属性绑定到数据源的 Directory 字段。

8. 按下 F5 键，运行工程。确保“立即”窗口是可见的。

单击“Cycle”按钮时，MySource 所提供的目录名将出现在“文本框”和“立即”窗口中，证明 MyConsumer 被绑定到 MySource。

命名属性、方法和事件

添加到类模块中的属性、方法和事件定义了一个接口——该接口将用来处理从该类创建的对象。在命名这些元素以及它们的参数时，遵循下面几个简单规则是有帮助的。

- 如果可能的话，尽量用完整的词，就像例子 SpellCheck 那样。缩写可能有许多形式，并因此可能使人混淆。如果完整的单词太长的话，可以

用完整的第一个音节。

- 对于标识符，可以用大小写混合的写法，每个单词或者音节的首字符大写，例如 `ShortcutMenus` 或者 `AsyncReadComplete`。
- 对于集合类名，可以用合适的复数写法，例如 `Worksheets`、`Forms` 或者 `Widgets`。如果集合中有这样名字的对象——以 "s" 为结尾，那么可将单词 "Collection" 附加在其后面，就像例子 `SeriesCollection` 那样。
- 对于方法名，可以比较一致地用动词/对象或者对象/动词的顺序。也就是说，使用 `InsertWidget`、`InsertSprocket`，等等，或者总是将对象放在前面，就像例子 `WidgetInsert` 和 `SprocketInsert` 那样。

注意 虽然在属性名中可以包含下划线，但是在事件名中包含下划线会引起错误。在事件过程中下划线用来分开对象名和事件名。因此，在命名属性、方法和事件时，最好都不要使用下划线。

对象编程的一个主要好处是可以进行代码重用。遵循上面的规则——这是 `ActiveX` 对接口指导的一部分，会使得很容易记住属性、方法以及事件的名字和用途。

详细信息 如果手头有 `Visual Basic` 专业版或者企业版，请参阅《部件工具指南》的第六章“创建 `ActiveX` 部件”中，“部件设计的一般准则”中，“命名的学问”中的扩展列表。

多态

多态意味着许多类可以提供同样的属性或者方法，而且调用者在调用这些属性或方法之前，不必知道某个对象属于什么类。

例如，Flea 类跟 Tyrannosaur 类可能都有 Bite 方法。多态意味着可以调用 Bite 方法，而不必知道某个对象是 Flea 还是个 Tyrannosaur——尽管以后当然会搞清这一点的。

详细信息 对于 Visual Basic 的专业版和企业版，多态已经变成了发展软件部件系统的一种强大的机制。这在《部件工具指南》中“创建 ActiveX 部件”中的第六章“部件设计的一般准则”中作了讨论。

Visual Basic 是如何提供多态的

大多数面向对象的程序设计系统，都是通过继承来提供多态的。也就是说，设想的 Flea 和 Tyrannosaur 类可能都是从某个 Animal 类继承来的。每个类都将重写 Animal 类的 Bite 方法，以提供自己的噬咬特点。

多态来自于这个事实：可以调用属于某类对象的 Bite 方法-该类可以是 Animal 中派生出来的任意类，而不必知道该对象到底属于哪一个类。

提供多态接口

Visual Basic 不用继承来提供多态。Visual Basic 是通过多重 ActiveX 接口来提供多态的。在构成 ActiveX 规格说明基础的部件设计模型 (COM) 中，多重接口允许软件部件系统在不扩散现有代码的情况下进行展开。

一个接口是一组相关的属性和方法。ActiveX 规格说明的许多内容是关于实现一些标准接口的，这些标准接口是用来获得系统服务或者为其它程序提供功能。

在 Visual Basic 中，可以创建一个 Animal 接口，并用自己的 Flea 和 Tyrannosaur 类予以实现。然后就可以调用任何一种对象的 Bite 方法，而无需知道它到底是哪一种对象。

多态和性能

从性能方面考虑，多态是很重要的。为了认识这一点，考虑下面的函数：

```
Public Sub GetFood(ByVal Critter As Object, _  
    ByVal Food As Object)  
    Dim dblDistance As Double  
    ' 计算到食物所在处距离的代码（略）。  
    Critter.Move dblDistance    ' 后期绑定  
    Critter.Bite Food           ' 后期绑定  
End Sub
```

对 Critter 来说，Move 方法和 Bite 方法是后期绑定的。后期绑定在 Visual Basic 编译时不能决定变量包含何种对象时发生。在本示例中，Critter 参数被声明为 As Object，因此在运行时，它可能包含对任何类型对象引用，如

Car 或者 Rock。

因为它不可能指明对象将是什么，所以 Visual Basic 编译一些附加的代码，用这些代码来询问该对象是否支持已经调用的方法。如果该对象支持这种方法的话，那么附加的代码将调用它；反之，附加的代码将会产生一个错误。每种方法或者属性的调用都会引入这个额外开销。

相比较而言，接口则允许前期绑定。当 Visual Basic 在编译时明确知道正在调用什么样的接口时，它将检查一下类型库，看看那个接口是否支持该方法。然后 Visual Basic 就可以用一张虚拟函数表 (vtable)，按直接跳转到该方法进行编译。这样做比起后期绑定来要快许多倍。

现在，假设 Move 和 Bite 方法属于 Animal 接口，而且动物的所有类都提供该接口。Criticter 参数现在就可以被声明为 Animal 了，而且 Move 和 Bite 方法也将是前期绑定的：

```
Public Sub GetFood(ByVal Critter As Animal, _  
    ByVal Food As Object)  
    Dim dblDistance As Double  
    '计算到食物所在处距离的代码（略）。  
    Critter.Move dblDistance    '前期绑定 (vtable)。  
    Critter.Bite Food           '前期绑定 (vtable)。  
End Sub
```

创建和实现接口

如上面“Visual Basic 是如何提供多态的”中所解释的，所谓接口，就是一组属性和方法。在下面的代码示例中，将创建一个 `Animal` 接口，并在两个类——`Flea` 和 `Tyrannosaur` 中实现它。

创建该 `Animal` 接口的方法是：将一个类模块添加到工程中，将它命名为 `Animal`，并插入到下面的代码中：

```
Public Sub Move(ByVal Distance As Double)
```

```
End Sub
```

```
Public Sub Bite(ByVal What As Object)
```

```
End Sub
```

注意，在这些方法中并没有任何代码。`Animal` 是一个抽象类，不包含实现的任何代码。抽象类不是用来创建对象的——其用途是为添加其它类中的接口提供模板（尽管已经证明，有些时候实现一个非抽象类的接口是有用的；这将在本节主题的后面进行讨论）。

注意 确切地说，一个抽象的类是不能从中创建对象的类的。从 `Visual Basic` 的类中总是可以创建对象的，即使它们不含有代码，它们也不是真正抽象的。

现在，就可以添加另外两个类模块了，这两个类模块一个叫 `Flea`，另一个叫 `Tyrannosaur`。为了在 `Flea` 类中实现 `Animal` 接口，要用到 `Implements` 语句：

Option Explicit

Implements Animal

一旦将这行代码添加进去，就可以单击代码窗口中左边（“对象”）下拉菜单。其中一个登录项将是 `Animal`。当选择了它时，右边（“过程”）下拉菜单上将显示 `Animal` 接口的方法。

依次选择每种方法，可为所有的方法创建空白过程模板。这些模板将具有正确的参数和数据类型，就像在 `Animal` 类中所定义的那样。每个过程名都将以 `Animal_` 前缀来标识该接口。

重点 一个接口就像一个契约。实现接口后，当调用该接口的任何属性或者方法时，一个类已经约定好要作出反应。因此，必须实现接口的所有属性和方法。

现在，就可以将下面的代码添加到 `Flea` 类中了：

```
Private Sub Animal_Move(ByVal Distance As Double)
```

```
    '(跳过多少英寸的代码，略。)
```

```
    Debug.Print "Flea moved"
```

```
End Sub
```

```
Private Sub Animal_Bite(ByVal What As Object)
```

```
    '(吃了多少生命的代码，略。)
```

```
    Debug.Print "Flea bit a " & TypeName(What)
```

```
End Sub
```

可能想知道为什么这些过程被声明为 `Private`。如果它们是 `Public`，那么

`Animal_Jump` 和 `Animal_Bite` 过程将成为 `Flea` 界面的一部分，这样将被困在跟原先所在的同样的圈子中，将 `Critter` 参数声明为 `As Object`，这样它就可能包含一个 `Flea` 或者一个 `Tyrannosaur`。

多重接口

现在，`Flea` 类有了两个接口：刚刚实现的 `Animal` 接口——它有两个成员，以及缺省的 `Flea` 接口——它没有任何成员。该示例的后面，将把一个成员添加到缺省接口的其中一个。

类似地，可以为 `Tyrannosaur` 类实现 `Animal` 接口：

```
Option Explicit
```

```
Implements Animal
```

```
Private Sub Animal_Move(ByVal Distance As Double)
```

```
    '(跳起多少码的代码，略。)
```

```
    Debug.Print "Tyrannosaur moved"
```

```
End Sub
```

```
Private Sub Animal_Bite(ByVal What As Object)
```

```
    '(拿起一磅肉的代码，略。)
```

```
    Debug.Print "Tyrannosaur bit a " & TypeName(What)
```

```
End Sub
```


练习 Tyrannosaur 和 Flea

将下列代码添加到“Form1”的 Load 事件中：

```
Private Sub Form_Load()  
    Dim fl As Flea  
    Dim ty As Tyrannosaur  
    Dim anim As Animal  
  
    Set fl = New Flea  
    Set ty = New Tyrannosaur  
    '首先看一下 Flea。  
    Set anim = fl  
    Call anim.Bite(ty)      'Flea 咬 dinosaur。  
    '现在轮到 Tyrannosaur。  
    Set anim = ty  
    Call anim.Bite(fl)      'Dinosaur 咬 flea。  
End Sub
```

按 F8 键，单步执行代码。注意“立即”窗口中的信息。当变量 anim 包含对 Flea 的引用时，就调用该 Flea 的 Bite 实现，同样的，对于 Tyrannosaur 来说，情况也是如此。

变量 anim 可以包含对实现 Animal 接口的任何对象的引用。事实上，它只能包含对这种类型对象的引用。如果试图将一个 Form 或者一个

PictureBox 对象赋给 anim 的话，那么将会产生错误。
当通过 anim 来调用 Bite 方法时，该方法是前期绑定的，因为 Visual Basic 在编译时知道，不论将什么对象赋给 anim，该对象都将有一个 Bite 方法。

将 **Tyrannosaurs** 和 **Fleas** 传递给过程

回忆一下“Visual Basic 如何提供多态？”("How Visual Basic Provides polymorphism?")中的 GetFood 过程。可以将 GetFood 过程的第 2 个版本——演示多态的版本——添加到“Form1”中，并用下面的代码来取代 Load 事件中的代码：

```
Private Sub Form_Load()  
    Dim fl As Flea  
    Dim ty As Tyrannosaur  
  
    Set fl = New Flea  
    Set ty = New Tyrannosaur  
    'Flea 在恐龙上吃饭。  
    Call GetFood(fl, ty)  
    '相反的情况。  
    Call GetFood(ty, fl)  
End Sub
```

单步执行上述代码，显示作为传递给另一接口类型参数的对象引用，是怎样转换为对第二个接口（在这里是 **Animal**）的引用的。所发生的情况是：

Visual Basic 查询该对象，看看该对象是否支持第二个接口。如果该对象支持的话，它将返回对该接口的引用，于是 Visual Basic 将返回的引用放置到参数变量中。如果该对象不支持第二个接口，将会出现错误。

实现返回值的方法

假设 Move 方法返回一个值。不管怎么说，自己知道想让 Animal 移动多远，但是，各个实例不可能都移动那么远。它可能又老又弱，或者在路上可能恰好有一堵墙。可以用 Move 方法的返回值来说明该 Animal 实际移动了多远的距离。

```
Public Function Move(ByVal Distance As Double) _  
    As Double
```

```
End Function
```

在 Tyrannosaur 类中实现该方法时，把返回值赋给过程名，这跟处理任何其它 Function 过程一样：

```
Private Function Animal_Move(ByVal Distance _  
    As Double) As Double
```

```
    Dim dblDistanceMoved As Double
```

```
    '计算能弹跳多远（基于对年龄、健康状态和障碍物的
```

```
    '考虑）的代码，略。
```

```
    '该示例假设已经将结果放置到变量 dblDistanceMoved 中。
```

```
    Debug.Print "Tyrannosaur moved"; dblDistanceMoved
```

```
Animal_Move = dblDistanceMoved  
End Function
```

为了返回值的赋值，使用全过程名，包括接口前缀。

实现属性

假设要将 `Age` 属性赋予 `Animal` 类——通过将一个 `Public` 变量添加到声明部分：

```
Option Explicit  
Public Age As Double
```

现在，在 `Tyrannosaur` 和 `Flea` 类的代码模块的“过程”下拉菜单上，将包含实现 `Age` 属性的属性过程，如下图 9.10 所示。

这里图示了本章前面的“向类中添加属性”中的一个要点。严格说来，用公有变量实现属性对于编程者来说是很方便的。此含义的背后，`Visual Basic` 是用一对属性过程来实现该属性。

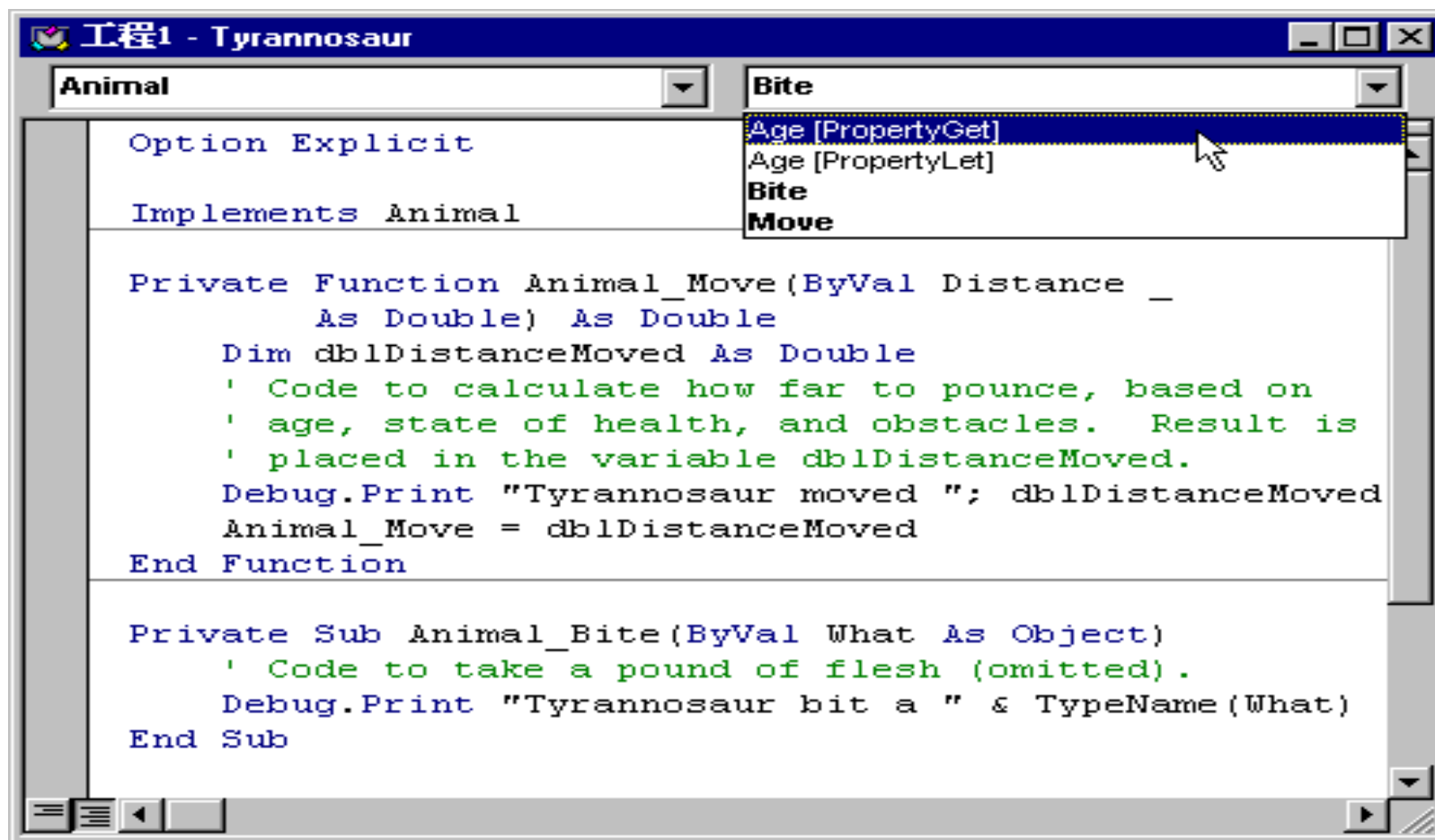


图 9.10 属性过程的实现

必须实现两个过程。通过在私有数据成员中保存值，实现属性很容易，如下所述：

```
Private mdblAge As Double
```

```
Private Property Get Animal_Age() As Double
```

```
    Animal_Age = mdblAge
```

```
End Property
```

```
Private Property Let Animal_Age(ByVal RHS As Double)
```

```
    mdblAge = RHS
```

```
End Property
```

私有数据成员是一个实现细节，必须自己添加。

注意 当 Implements 为 Property Set 或者 Property Let 提供模板时，并没有任何办法来确定最后一个参数的名字，因此代之以名字 RHS，如上述代码示例所述。

在作为公有数据成员实现的属性上，没有对数据的有效性进行验证，但是那并不意味着不能将 Animal_Age 有效性验证的代码添加到 Property Let 中。例如，可能想分别为 Tyrannosaur 或者 Flea 限制其值为合适的年龄。事实上，这意味着接口和实现的独立性。只要接口跟类型库中的描述相匹配，实现可以是任意的。

在进行下一步之前，从两个类模块中删除读写 Age 属性的实现部分。

实现只读属性

当然，允许任意设定一个动物的年龄，是一种糟糕的对象设计。对象应该知道自己的年龄，并将这个年龄作为一种只读属性提供给用户。从 Animal 类中删除公有变量 Age，并为只读年龄属性添加模板，如：

```
Public Property Get Age() As Double
```

```
End Property
```

现在，Tyrannosaur 和 Flea 类代码窗口的“过程”下拉菜单上，仅包含一个登录项，即 Age [PropertyGet]。对 Tyrannosaur 可这样实现：

```
Private mdblBirth As Double
```

```
Private Property Get Animal_Age() As Double
```

```
    Animal_Age = Now - mdblBirth
```

```
End Property
```

上述代码返回 Tyrannosaur 按日计算的年龄。可以在 Tyrannosaur 类的 Initialize 事件中设置 mdblBirth，如：

```
Private Sub Class_Initialize()
```

```
    mdblBirth = Now
```

```
End Sub
```

当然，也可以用更常用的单位来返回属性值，比如犬类的寿命。

关于对象和接口的简要补充讨论

这个 Tyrannosaur 和 Flea 代码示例，看起来似乎用接口和对象来解决很快，也很轻松。把对象的引用赋给一个对象变量，把接口的引用赋给另一

个对象变量。

事实上，所有的引用都是对象的引用。对一个接口的引用也是对实现接口的对象的引用。而且，一个对象可能有多个接口，但在底层它仍然是同一对象。

在 Visual Basic 中，每个类都有一个缺省的接口，该接口具有与类相同的名字。是的，几乎都是同样的。按照惯例，在类名前加一条下划线。该下划线指明了这个接口是隐藏在类型库里的。

这样，Tyrannosaur 类就有一个缺省名为 Tyrannosaur 的接口。因为 Tyrannosaur 同时实现 Animal，所以该类还有第二个接口（其名为 Animal）。

不过，在它的所有底层，对象仍然是 Tyrannosaur。将一个命令按钮放置到“Form1”上，并添加下面的代码：

```
Private Sub Command1_Click()  
    Dim ty As Tyrannosaur  
    Dim anim As Animal  
  
    Set ty = New Tyrannosaur  
    Set anim = ty  
    MsgBox TypeName(anim)  
End Sub
```

可能期望信息框显示 "Animal"，但是，事实上显示的是 "Tyrannosaur"。

查询接口

当把 Tyrannosaur 对象赋给 Animal 类型的变量时，Visual Basic 将询问该 Tyrannosaur 对象它是否支持 Animal 接口（为此所用的方法为 QueryInterface，或者简称 QI；有时可能遇到 QI 作为动词使用）。如果回答是否定的，将会产生错误。如果回答是肯定的，该对象将被赋给变量。只有通过这个变量 Animal 接口的方法和属性才可被访问。

类属对象变量和接口

如果将对象引用赋给类属对象变量（如下列代码所示），将会出现什么情况？

```
Private Sub Command1_Click()  
    Dim ty As Tyrannosaur  
    Dim anim As Animal  
    Dim obj As Object  
  
    Set ty = New Tyrannosaur  
    Set anim = ty  
    Set obj = anim
```

```
MsgBox TypeName(obj)
```

```
End Sub
```

结果得到另一个 `Tyrannosaur`。现在，通过变量 `obj` 调用属性和方法时，得到什么接口？将下面的方法添加到 `Tyrannosaur` 类中：

```
Public Sub Growl()
```

```
    Debug.Print "Rrrrrr"
```

```
End Sub
```

`Growl` 方法属于 `Tyrannosaur` 对象的缺省接口。在命令按钮的 `Click` 事件的代码中，用下面的两行代码来取代 `MsgBox` 语句：

```
obj.Move 42
```

```
obj.Growl
```

当运行这个工程并单击按钮时，执行将在 `Growl` 方法上停止，并返回错误信息“对象不支持此属性或方法”。很明显，接口仍然是 `Animal`。

`Object` 类型的变量和具有多重接口的对象一起使用时，必须记住某些事情。该变量将访问的接口是最后赋值的接口。例如：

```
Private Sub Command1_Click()
```

```
    Dim ty As Tyrannosaur
```

```
    Dim anim As Animal
```

```
    Dim obj As Object
```

```
    Set ty = New Tyrannosaur
```

```
    Set anim = ty
```

```
Set obj = anim
obj.Move 42          '成功
obj.Growl            '失败

Set obj = ty
obj.Move 42          '失败
obj.Growl            '成功
```

```
End Sub
```

幸好，很少将这种比较慢的、后期绑定的 **Object** 数据类型和具有多重接口的对象一起使用。使用多重接口的一个主要原因是，通过多态可以得到前期绑定的好处。

其它接口源

Visual Basic 的类模块不是唯一的实现接口的源。可以实现类型库包含的各种接口，只要该接口支持 **Automation**。

如果有 **Visual Basic** 专业版或企业版，那么可以创建自己的抽象类的类型库。这些类型库可能在许多工程中用到，就像在《部件工具指南》中，“创建 **ActiveX** 部件”中的第六章，即“部件设计的一般准则”中所描述的那样。

专业版和企业版在 **Tools** 目录都包括有 **MkTypLib** (**Make Type Library**) 实用工具。如果已经用 **Microsoft Visual C++** 使用过该实用工具，就会发现这是创建接口的一条更适宜的途径。

在工程中使用接口

为了在工程中使用接口，可以在“工程”菜单上单击“引用”，打开“引用”对话框。如果类型库已经注册，那么它将出现在引用列表中，于是可以复选它。如果类型库不在该列表中，可以使用“浏览”按钮对它定位。一旦有了对某个类型库的引用，就可以使用 `Implements` 来实现该类型库包含的任何 `Automation` 接口。

代码重用的许多（内部）方面

有两种主要的代码重用窗体——二进制和源代码。二进制代码重用，是通过创建和使用一个对象来实现的，而源代码重用，则是通过继承来实现的——`Visual Basic` 不支持（源代码重用也可以通过复制和修改源代码来实现，但这种技术没有什么新意，而且存在许多人所共知的问题）。

`Visual Basic` 已成为二进制代码重用的先驱——控件就是典型的例子。通过将控件的实例放置到窗体上就可以重用该控件中的代码。这被称为包含 (containment) 关系或者具有 (has-a) 关系；也就是说，该窗体包含或者具有 `CommandButton`。

详细信息 包含关系在本章后面的“对象模型”作了讨论。

委派给实现的对象

`Implements` 提供了代码重用的一种强有力的新途径。可以实现一个抽象类（就像在“创建和实现接口”中所讨论的那样），或者也可以实现一个全功能类的接口。可以在外部对象（即，实现内部对象接口的那个对象）的 `Initialize` 事件中创建内部对象（即，所实现的对象）。

就像在“创建和实现接口”中所提到的那样，接口就像契约一样——必须在外部对象的类模块中实现内部对象接口的所有成员。不过，在委派内部对象的属性和方法为代表方面，可以有很多选择。在一个方法中，可能直接委派一个内部对象为代表，传递未更改的参数，而在另一种方法中，可能执行在调用该内部对象之前自己的部分代码，在第三种方法中仅仅执行自己的代码，而完全忽略内部对象！

例如，假设有一个 `OneManBand` 类和一个 `Cacophony` 类，这二者都产生声音。希望把 `Cacophony` 类的功能添加到 `OneManBand` 类中，并重用 `Cacophony` 类方法的一些实现。

'OneManBand 实现 Cacophony 接口。

```
Implements Cacophony
```

'保存引用的对象变量。

```
Private mcac As Cacophony
```

```
Private Sub Class_Initialize()
```

'创建对象。

```
Set mcac = New Cacophony
```

```
End Sub
```

现在，就可以在“对象”下拉菜单上选择 **Cacophony**，然后为 **Cacophony** 接口的方法获得过程模板。为了实现这些方法，可以委派给 **Cacophony** 对象。例如，**Beep** 方法可能看起来如下所示：

```
Private Sub Cacophony_Beep(ByVal Frequency As Double, _
```

```
ByVal Duration As Double)
```

```
    '委派给内部的 Cacophony 对象。
```

```
    Call mcac.Beep(Frequency, Duration)
```

```
End Sub
```

上面的实现是非常简单的。外部对象 (**OneManBand**) 直接委派给内部对象 (**Cacophony**)，不作任何更改即可重用 **Cacophony** 对象的 **Beep** 方法。这是一件好事，但仅仅只是个开端。

Implements 语句对于代码重用来说，是一个非常强大的工具，因为它给予很大的灵活性。可能想更改 **OneManBand** 类的 **Beep** 方法的效果，方法是在对内部 **Cacophony** 对象的调用之前（或之后），插入自己的代码：

```
Private Sub Cacophony_Beep(ByVal Frequency As Double, _
```

```
ByVal Duration As Double)
```

```
    '撞击每一件东西升高八度。
```

```
    Frequency = Frequency * 2
```

```
    '基于 OneManBand 类的另一个属性，即 Staccato，
```

'分割每个嘟嘟声的持续时间。

```
If Staccato Then Duration = Duration * 7 / 8
```

```
Call mcac.Beep(Frequency, Duration)
```

'甚至可以调用 OneManBand 的其它方法。

```
If Staccato Then Pause(Duration * 1 / 8)
```

```
End Sub
```

对于这些方法来说，实现可能会直接委派内部 Cacophony 对象为代表，而对于另外一些来说，可能在委派之前或之后插入自己的代码——甚至完全忽略委派，而完全用自己的代码来实现一种方法。

因为 OneManBand 类实现 Cacophony 接口，所以可以将它和调用该接口的任何音乐应用程序一起来使用。其实现细节可从调用应用程序处隐藏起来，但是结果发出的声音都是自己的。

注意 COM 提供了另一个机制进行二进制代码重用，该机制叫凝聚。在集合中，无更改地重用整个接口，而且该实现是由被凝聚的类的一个实例所提供的。Visual Basic 不支持这种代码重用的窗体。

这样一来不冗长乏味吗？

写委派代码的确可能变得冗长乏味，如果大部分外部对象的属性和方法，只是简单地直接委派给相对应的内部对象的属性和方法的时候，尤其是这样。

如果有 Visual Basic 专业版或企业版，就可以使用“Visual Basic 扩展性”模型，创建自己的委派向导，使任务自动化，类似于专业版和企业版中的

“类向导”一样。

详细信息 在部件软件中多态和多重接口的使用，在《部件工具指南》中，“创建 ActiveX 部件”中的第六章，即“部件设计的一般准则”中作了讨论。

关于“扩展性模型”的使用，在《部件工具指南》中的“用外接程序来扩展 Visual Basic 环境”中作了文档说明。

用自己的对象编程

现在可以开始逐渐使用对象，寻找有用的一些任务——对这些任务来说，将代码和数据结合起来是一个优点。可以使用这些对象的功能，通过声明对象变量、将新对象赋给它们、并调用这些对象的属性和方法。

随着越来越多的对象添加到自己的程序中，将会开始看到它们之间的关系。就可以开始进行更多依赖于对象及其之间关系的程序设计了，并可开始使用更强健的技巧——比如创建自定义集合类——这是为了在代码中表达那些关系。

在某些点上，将会突然发现将对象链接在一起是如何改变了程序本质，这样，就已经有了开始设计基于对象的程序的条件。

下面的主题，概述了编程风格中这些重大改变。现在，读一读它们，给自己一个思考的图片，在对基于对象的程序设计的概念有所了解时，再读读它们。

详细信息 ActiveX 部件开创了代码重用和基于对象程序设计的另一新领域。如果有 Visual Basic 专业版或企业版，就可以通过《部件工具指南》中的“创建 ActiveX 部件”来开始探索这一新领域。

对象引用和引用计数

对象存活期的基本规则是非常简单的：当对某个对象的最后引用被释放时，该对象就被撤销。不过，伴随如此长的存活期，简单并不总意味着容易。随着使用更多对象，并保存了更多包含对那些对象引用的变量的时候，那么，可能要经历这样一些时期，那时，要撤销自己的对象，而且觉得应能撤销时，实际上却又不可能撤销。

在某个点上，应当能想到，Visual Basic 必须记录对象引用——否则它怎么能知道对象的最后引用何时释放呢？可能开始意识到了：如果只有自己能对 Visual Basic 的引用计数进行访问，那么调试将变得非常容易。

不幸的是，事实并非如此。为了更有效地使用对象，部件对象模型 (COM) 对其引用计数的规则指定了许多复杂的快捷方式。所造成的后果是让人不能信赖引用计数的值——即使已经得到了它。

根据 COM 规则，可以信赖的信息仅为，引用值是否为 0。可以知道什么时候该引用值到达 0，因为对象的 Terminate 事件出现了。除了那个以外，从引用计数中就再也搜集不到任何可靠的信息了。

注意 不记住 COM 引用计数的规则并非一件小事情。自己管理引用计数，比起记录程序中包含对象引用的对象变量来，更加困难。

提示 将对象变量声明为类类型，而非对象。这样的话，是否还有没有终止的 `Widget` 对象，所需要担心的变量仅仅是那些声明为 `As Widget` 的变量。

对于对象引用的集合来说，不要使用该 `Visual Basic Collection` 对象本身。在一个 `Visual Basic Collection` 对象中，对象的引用是存储在 `Variants` 中的一——它声明为 `As Object` 的变量一样，可以保存对任何类对象的引用。正确做法是：创建自己的接受唯一类对象的集合类，就像在“创建自己的集合类”中所描述的那样。这样，为了找到 `Widget` 对象，所需要搜索的集合仅仅是那些 `Widget` 类型的集合。

把对象组织成分层结构，就像在“对象模型”中所描述的那样。如果所有对象都能连接起来，那么写一个穿过整个模型并报告现存所有对象的过程，是一件很容易的事。

不要将变量声明为 `As New`。它们就像吹灭后会重新点燃的生日蜡烛一样：如果在将它设置为 `Nothing` 后又使用，那么 `Visual Basic` 会强制性地创建另一个对象。

详细信息 循环引用对于利落地关闭来说，是最麻烦的一种情况。请参阅“对象模型”。

对象模型

通过创建类模块并赋以属性和方法，就可以定义了类，接着就可以由该类

创建任意数量的对象。如何记录所创建的这些对象呢？

记录对象最简单的办法，莫过于为创建的每个对象都声明一个对象变量。当然，这样对能够创建对象的数量就有了限制。

可以在某个数组或者集合中保持多个对象引用，就像在本章前面的“创建对象数组”和“创建对象集合”中所讨论的那样。

刚开始时，可能要定位窗体或标准模块中的对象变量、对象数组以及对象集合，就跟处理普通变量一样。但是，随着添加更多的类，可能会发现正使用的这些对象之间有明确的关系。

对象模型表达了包含关系

对象模型给出了基于对象程序的结构。通过定义程序中所使用的对象之间的关系，对象模型能够以一种使编程变得更容易的方式来组织对象。

一般来说，对象模型表达了这样一个事实：即某些对象是“更大的”，或者说比其它对象更重要一些——可以认为这些对象是包含其它对象的对象，或者是由其它对象所组成的对象。

例如，在编程时，可能创建一个 `SmallBusiness` 对象来作为程序的核心。可能想让这个 `SmallBusiness` 对象包含与其关联的其它类型的对象，比如 `Employee` 对象和 `Customer` 对象。可能同时也希望它包含一个 `Product` 对象。在图 9.11 中显示了该程序的对象模型。

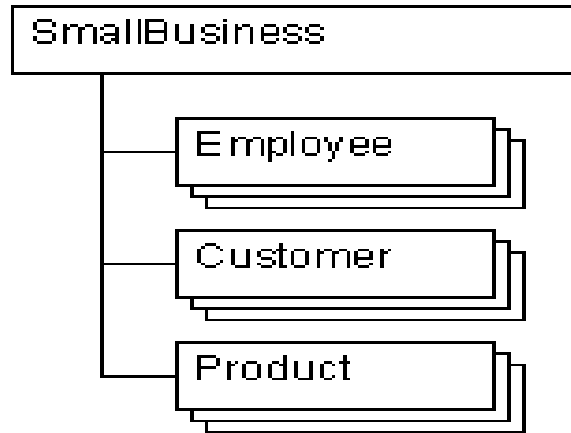


图 9.11 对象模型

可以定义四个类模块，分别叫做 `SmallBusiness`、`Employee`、`Customer` 和 `Product` 类模块，并给它们中的每个添加合适的属性和方法，但是怎样在对象之间建立连接呢？有两种工具可以达到这个目的：即 `Object` 属性和 `Collection` 对象。下列这段代码显示了实现图 9.11 分层结构的一种办法。

' `SmallBusiness` 类模块中声明部分的代码。

```
Public Name As String
```

```
Public Product As New Product
```

```
Public Employees As New Collection
```

```
Public Customers As New Collection
```

第一次引用 `Product` 属性时，将创建对象，因为已经把它声明为 `As New`。例如，可以用下面的代码创建并设置 `SmallBusiness` 对象的 `Product` 对象的名称和价格。

' 一个标准模块的代码。

```
Public sbMain As New SmallBusiness
```

```
Sub Main
```

```
    sbMain.Name = "Velociraptor Enterprises, Inc."
```

```
    ' 在代码中首次使用 Product 变量时，创建 Product 对象。
```

```
    sbMain.Product.Name = "Inflatable Velociraptor"
```

```
    sbMain.Product.Price = 1.98
```

```
    .
```

```
    ' 初始化并显示主窗体的代码。
```

```
    .
```

```
End Sub
```

注意 用公共变量来实现一个对象属性并不简洁。如果在代码中的某处，将该属性设置为 `Nothing` 的话，可能会无意中将 `Product` 对象撤销。更好的办法是创建对象属性时，将之设置为只读属性，如下列代码所示。

'为更强健的对象属性所编制的代码。该属性的存储是私有的，

'因此不能从对象的外部将之设置为 `Nothing`。

```
Private mProduct As New Product
```

```
Property Get Product() As Product
```

```
    '首次调用这个属性时，mProduct 包含 Nothing，
```

```
    '因此 Visual Basic 将创建一个 Product 对象。
```

```
        Set Product = mProduct
End If
```

一对多对象关系

当对象之间关系是一对一时，对象属性可以正常工作。然而，经常出现的情况却是，某种类型的一个对象包含另一种类型的一些对象。在 SmallBusiness 对象模型中，Employees 属性是作为一个 Collection 对象来实现的，因此，这个 SmallBusiness 对象可以包含多个 Employee 对象。下列代码显示了怎样把新的 Employee 对象添加到这个集合中。

```
Public Function NewEmployee(Name, Salary, HireDate, _
ID) As Employee
    Dim empNew As New Employee
    empNew.Name = Name           '对象的隐含创建。
    empNew.Salary = Salary
    empNew.HireDate = HireDate
    '添加到集合中，并使用 ID 作为一个键。
    sbMain.Employees.Add empNew, CStr(ID)
    '返回对新 Employee 的引用。
    Set NewEmployee = empNew
End Function
```

在创建 SmallBusiness 对象所代表的企业雇员时，需要调用这个 NewEmployee 函数多少次，就可以调用多少次。任何时候，通过遍历

Employees 集合，都可将现有的雇员列出。

注意 再次说明，并没有非常强健的实现方法。比较好的实践是创建自己的集合类，并将它们按只读属性给出。这在“创建自己的集合类”中作了讨论。

提示 在 Visual Basic 的专业版和企业版中，包括 Class Builder 实用工具，利用这个实用工具，可以产生实现对象模型时需要的大量代码。Class Builder 可创建强健的对象属性和集合类，并且可以很容易地重新组织对象模型。

Parent 属性

具有对象的引用时，通过使用对象属性和对象集合，就可以到达这个引用所包含的对象。能够向分层结构的上层移动也是非常有用的，因为可以到达包含所引用对象的那个对象。

向上移动通常是用 Parent 属性来完成的。Parent 属性返回对象容器的引用。关于对象模型定位的讨论，请参阅第十章“用部件编程”中的“定位对象模型”。

可以在本章前面的“向类中添加属性”找到使用 Parent 属性的示例。

提示 当把 Parent 属性赋给集合中的对象时，不要使用对 Collection 对象的引用。这个对象真正的父对象是包含该集合的对象。如果 Parent 属性指向了该集合，将不得不使用两级间接指针，才能到达真正的父对象——也就是说，要用 obj.Parent.Parent，而不是 obj.Parent。

Parent 属性、循环引用，以及对象拆卸

使用 Parent 属性的一个最大问题是它们会造成循环引用。“较大”的对象具有对所包含对象的引用，而被包含的对象通过其 Parent 属性也有引用，这样就创建了一个环，如图 9.12 所示。

在这幅图片上有什么错误？去除这些对象的办法，是在用对象完成工作后，释放所有对它们的引用。假设对 SmallBusiness 对象的引用是在一个名为 sbMain 的变量中，就像在本节主题前面讨论的那样，可能会写出象下面这样的代码：

```
Set sbMain = Nothing
```

不幸的是，仍然有一个对 SmallBusiness 对象的引用——事实上，可能有许多引用，因为每个 Employee 对象的 Parent 属性都将包含对这个 SmallBusiness 对象的引用。

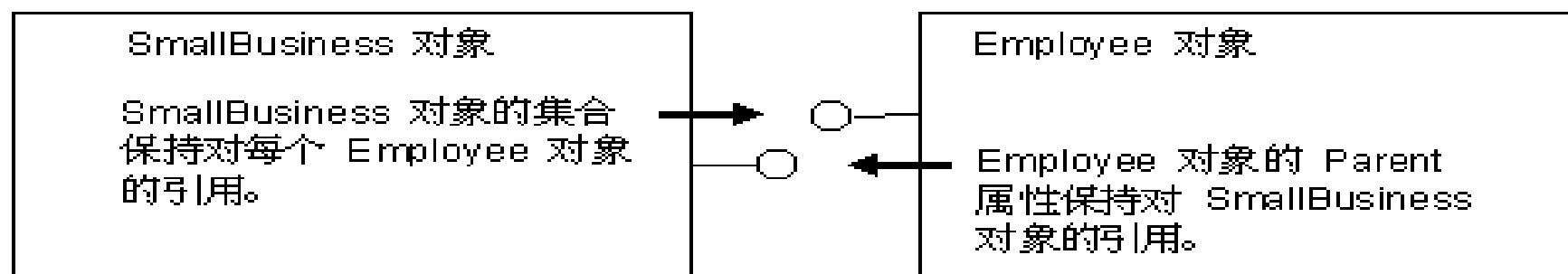


图 9.12 循环引用的一个例子

由于 SmallBusiness 对象的 Employees 集合包含有对每个 Employee 对象

的引用，因此任何对象都不会被撤销。

TearDown 方法

一个解决这个问题的办法是，把 TearDown 方法添加给该 SmallBusiness 对象。这样，就可以将所有 SmallBusiness 对象的对象属性设置为 Nothing，同时也将所有 Collection 对象 (Employees, Customers) 设置为 Nothing。

当 Collection 对象被撤销时，Visual Basic 就把它所包含的所有对象引用设置为 Nothing。如果没有对包含在 Employees 和 Customers 集合中的 Employee 和 Customer 对象其它的引用，那么它们将被撤销。

当然，如果 Employee 对象是由更小的对象构成的，则将会存在其父对象所具有的同样的循环引用问题。在那种情况下，必须将一个 TearDown 方法赋给 Employee 类。要做的事情并不仅仅是将 Employees 的 Collection 对象设置为 Nothing，SmallBusiness 对象还将不得不首先遍历该集合，调用每个 Employee 对象的 TearDown 方法。

其它问题

即使这样，也并非所有对象都能被撤销。如果在程序的某个地方还有一些变量，这些变量仍然包含对 SmallBusiness 对象，或者对 SmallBusiness 对象所包含的任何对象的引用，那么这些对象将不会被破坏。程序中必须有一部分清理代码，以确保各个地方的所有对象变量都被设置为 Nothing。

为了测试是否正发生这种情况，可能需要将一些调试代码添加到对象中。例如，可以将下列代码添加到标准模块中：

' 全局调试集合

Public gcolDebug As New Collection

' 全局函数，赋给每个对象一个唯一的 ID。

```
Public Function DebugSerial() As Long
    Static lngSerial As Long
    lngSerial = lngSerial + 1
    DebugSerial = lngSerial
End Function
```

在每个类模块中，都可以加入类似下面这样的代码。每个类在“Product”出现的地方提供它自己的名字。

'调试 ID 的存储。

```
Private mlngDebugID As Long

Property Get DebugID() As Long
    DebugID = mlngDebugID
End Property

Private Sub Class_Initialize()
    mlngDebugID = DebugSerial
    '将一个字符串登录项添加到全局集合中。
    gcolDebug.Add "Product Initialize; DebugID=" _
        & DebugID, CStr(DebugID)
End Sub
```

```
Private Sub Class_Terminate()
```

```
    '删除字符串登录项，这样，就会知道对象不再存在了。
```

```
    gcolDebug.Remove CStr(DebugID)
```

```
End Sub
```

随着每个对象的创建，该对象将一个字符串放到全局集合中；当该对象撤销时，将删除该字符串。在任何时候，都可以遍历全局集合，看看什么对象还没有撤销。

详细信息 用 Visual Basic 的专业版或者企业版来创建 ActiveX 部件时，对象模型承担新的重要性，并且是另一个问题集合。请参阅《部件工具指南》中，“创建 ActiveX 部件”中的第六章，即“部件设计的一般准则”。

创建自己的集合类

一般可以采取三种办法用集合来实现对象包含。以上面“对象模型”中所讨论的 SmallBusiness 对象的 Employees 集合为例。为了实现该集合，可以这样做：

- 在 SmallBusiness 类模块中，将 Employees 变量声明为 As Collection，并使之成为 Public。这是最简单的解决方案。
- 在 SmallBusiness 类模块中，将 mcolEmployees 变量声明为 As Collection，并使之成为 Private。将添加和删除对象的一组方法赋给

SmallBusiness 对象。这是三种设计方案中面向对象概念用得最少的办法。

- 通过创建名为 Employees 的集合类模块，来实现自己的集合类，就像在本章后面所描述的那样。将 Employees 类的只读属性赋给 SmallBusiness 对象。

下面以强健性的逐渐增加为顺序，列出了三种策略。它们的特点可以分别用稻草盖的房子、树枝盖的房子和砖块盖的房子来作类比。

公有集合示例：稻草盖的房子

为了创建此示例，打开一个新工程并插入两个类模块。在窗体上画出五个命令按钮、一个列表框、两个文本框，以及两个标签，如图 9.13 所示。



图 9.13 Employees 集合示例

下表列出了该示例所需设置的属性值。

对象	属性	设置值
类模块	Name	Employee
类模块	Name	SmallBusiness
窗体	Caption	雇员集合
第一个命令按钮	Caption	添加
	Name	cmdAddEmployee

续表

第二个命令按钮	Caption Name	删除 cmdDeleteEmployee
第三个命令按钮	Caption Name	刷新表 cmdListEmployees
第四个命令按钮	Caption Name	有问题 cmdTrouble
第五个命令按钮	Caption Name	关闭 cmdClose
第一个标签控件	Caption	Name
第二个标签控件	Caption	Salary
第一个文本框	Name Text	TxtName (blank)
第二个文本框	Name Text	TxtSalary (blank)
列表框	Name	lstEmployees

在 Employee 类模块中，将下面的声明和属性过程添加进去：

Option Explicit

'Employee 类的属性。

Public Name As String

Public Salary As Long

'只写一次 ID 属性的私有数据。

```
Private mstrID As String
```

```
Property Get ID() As String
```

```
    ID = mstrID
```

```
End Property
```

'首次设置 ID 属性时，静态的 Boolean 变量也要设置。

'后续的调用什么也不做。

'（若不是这样，产生一个错误会更好。）

```
Property Let ID(strNew As String)
```

```
    Static blnAlreadySet As Boolean
```

```
    If Not blnAlreadySet Then
```

```
        blnAlreadySet = True
```

```
        mstrID = strNew
```

```
    End If
```

```
End Property
```

ID 属性是从集合中检索或者删除 Employee 对象的键，因此必须对它进行一次性设置并不再更改。这是用 Static Boolean 变量来完成的，该变量在首次设置 ID 属性的时候被设置为 True。该属性总是可读的，因为有一个 Property Get。

在 SmallBusiness 类模块中，将下面的声明添加进去。当代码中引用

Employees 变量时，将创建集合对象。

```
Option Explicit
```

```
Public Employees As New Collection
```

窗体所做的所有工作

所有的剩余代码都在窗体模块中。在声明部分添加下面的声明。

```
Option Explicit
```

```
Public sbMain As New SmallBusiness
```

cmdEmployeeAdd_Click 事件中的代码，将一个成员添加到集合。

```
Private Sub cmdEmployeeAdd_Click()
```

```
    Dim empNew As New Employee
```

```
    Static intEmpNum As Integer
```

```
    '用 With 语句，使代码更快，也更简洁
```

```
    ' (.ID 对应 empNew.ID)。
```

```
    With empNew
```

```
        '为新的雇员产生一个唯一的 ID。
```

```
        intEmpNum = intEmpNum + 1
```

```
        .ID = "E" & Format$(intEmpNum, "00000")
```

```
        .Name = txtName.Text
```

```
        .Salary = CDbl(txtSalary.Text)
```

```
        '将 Employee 对象引用添加到集合中，
```

```
        '用 ID 属性作为键。
```



```
        sbMain.Employees.Add empNew, .ID
    End With
    txtName.Text = ""
    txtSalary.Text = ""
    '单击“刷新表”按钮。
    cmdListEmployees.Value = True
```

```
End Sub
```

在 `cmdListEmployees_Click` 事件过程中的代码，用一个 `For Each ... Next` 语句将所有的雇员信息添加到 `Listbox` 控件中。

```
Private Sub cmdListEmployees_Click()
    Dim emp As Employee
    lstEmployees.Clear
    For Each emp In sbMain.Employees
        lstEmployees.AddItem emp.ID & ", " & emp.Name _
            & ", " & emp.Salary
    Next
End Sub
```

`cmdEmployeeDelete_Click` 事件用 `Collection` 对象的 `Remove` 方法，删除当前 `Listbox` 控件中选定的集合对象。

```
Private Sub cmdEmployeeDelete_Click()
    '检查一下，以确保有被选中的雇员。
    If lstEmployees.ListIndex > -1 Then
```

'前六个字符是 ID。

sbMain.Employees.Remove _

Left(lstEmployees.Text, 6)

End If

'单击“刷新表”按钮。

cmdListEmployees.Value = True

End Sub

将下面的代码添加给“有问题”按钮。

Private Sub cmdTrouble_Click()

'说些什么!?

sbMain.Employees.Add Me

End Sub

cmdClose_Click 事件关闭该应用程序。当关闭使用对象的工程时，卸载所有窗体，确保类模块中的所有 **Terminate** 事件过程都将得到执行。相反，使用 **End** 语句终止程序显得生硬，不会执行 **Terminate** 事件。

Private Sub cmdClose_Click()

Unload Me

End Sub

在示例中，为了添加雇员，可以运行应用程序，在两个文本框中输入值，然后选取“添加”按钮。添加几个雇员，然后用删除和列表按钮来作一下实验。

强健性相当于稻草盖的房子

这种简单的实现方法不是很强健。因为 `Employees` 属性仅仅是一个公有的 `Collection` 对象，可能无意中在程序中的任何地方访问它。而且，`Collection` 对象的 `Add` 方法不作任何类型检查。例如，在“有问题”按钮的 `Click` 事件中的代码，会轻率地将对窗体的对象引用插入到雇员集合中。

单击“有问题”按钮，会注意到未出现任何错误。现在，单击“刷新表”按钮。当 `For Each ... Next` 循环遇到意外的对象类型时，它将导致类型为 13 的错误发生，即“类型不匹配”。

这是用公用的 `Collection` 对象建立对象模型时，可能会碰到的错误的一个示例。可以从工程的任何地方添加对象，而且对于它们是否将被正确地初始化，没有任何保证。如果程序员复制代码以添加雇员，而且原始代码后来又有所改动，那么要找到产生的错误非常困难。

私有集合示例：树枝盖的房子

将 `Employee` 对象和 `SmallBusiness` 对象连接起来的一种更强健一些的办法是使 `Collection` 对象私有化。对于该示例来说，将从上面的“公有集合”示例中重用窗体和大部分代码。

`Employee` 类模块没有改变。然而，`SmallBusiness` 类模块被完全翻新了。用下面的声明语句来取代公有的 `Collection` 对象的声明，并把下面几个段

落中所描述的 Sub 和 Function 过程添加进去。

Option Explicit

Private mcolEmployees As New Collection

跟前面的一样，添加一个雇员的代码做大部分工作（可以从前面示例中的 cmdEmployeeAdd_Click 事件过程中把点线之间的代码块取出来）。

重要的改动在于：Collection 对象的 Add 方法不能再从程序的任意模块来调用了，因为 mcolEmployees 是 Private。只能用 EmployeeAdd 方法来添加 Employee 对象，这将会使新对象正确地初始化：

'SmallBusiness 的类方法。

Public Function EmployeeAdd(ByVal Name As String, _

ByVal Salary As Double) As Employee

'-----

Dim empNew As New Employee

Static intEmpNum As Integer

'用 With 语句，使代码更快，也更简洁

' (.ID 对应 empNew.ID)。

With empNew

 '为新雇员产生一个唯一的 ID。

 intEmpNum = intEmpNum + 1

 .ID = "E" & Format\$(intEmpNum, "00000")

 .Name = Name

 .Salary = Salary

'将 Employee 对象引用添加到集合中，
'用 ID 属性作为键。

'-----

mcolEmployees.Add empNew, .ID

End With

'给新雇员返回一个引用。

Set EmployeeAdd = empNew

End Function

EmployeeAdd 方法为新添加的 Employee 对象返回一个引用。这是一个好的实践，因为一旦创建了一个对象，很可能就想用它来做点什么事。

EmployeeCount、EmployeeDelete 以及 Employees 方法委派 Collection 对象的相应方法为代表。委派意味着 Collection 对象做全部工作。

'SmallBusiness 类的方法。

Public Function EmployeeCount() As Long

EmployeeCount = mcolEmployees.Count

End Function

Public Sub EmployeeDelete(ByVal Index As Variant)

mcolEmployees.Remove Index

End Sub

Public Function Employees(ByVal Index As Variant) _

```
As Employee
```

```
    Set Employees = mcolEmployees.Item(Index)
```

```
End Function
```

注意 可以将附加的功能添加到这些方法中。例如，如果一条索引项是无效的，能够引发自己的错误。

最后的方法是 **Trouble**。该方法试图将未初始化的 **Employee** 对象添加到集合中。能猜测到将会发生什么事吗？

'SmallBusiness 类的方法。

```
Public Sub Trouble()
```

```
    Dim x As New Employee
```

```
    mcolEmployees.Add x
```

```
End Sub
```

对窗体的更改

现在，将不得不对窗体模块作一些改动。可以使用前面示例所使用的同样的模块级声明，而且“关闭”按钮的 **Click** 事件也是相同的，但是，其它事件过程已经改变了一一“添加”按钮的代码短了许多，而“删除”和“列出雇员的表”按钮的代码则已经改变为一种很小但却很有效的方式：

```
Private Sub cmdEmployeeAdd_Click()
```

```
    sbMain.EmployeeAdd txtName.Text, txtSalary.Text
```

```
    txtName.Text = ""
```

```
    txtSalary.Text = ""
```

```
        cmdListEmployees.Value = True
End Sub

Private Sub cmdEmployeeDelete_Click()
    '检查一下，以确保有雇员被选中。
    If lstEmployees.ListIndex > -1 Then
        '前六个字符是 ID。
        sbMain.EmployeeDelete Left(lstEmployees.Text, 6)
    End If
    cmdListEmployees.Value = True
End Sub

Private Sub cmdListEmployees_Click()
    Dim lngCt As Long
    lstEmployees.Clear
    For lngCt = 1 To sbMain.EmployeeCount
        With sbMain.Employees(lngCt)
            lstEmployees.AddItem .ID & ", " & .Name _
                & ", " & .Salary
        End With
    Next
End Sub
```

但是，在 `cmdListEmployees_Click` 中的所有这些附加代码是什么？不幸的是，在追求强健性的过程中，就已经放弃了使用 `For Each ... Next` 来遍历集合中各项的能力，因为现在 `Collection` 对象已经声明为 `Private`。如果试图像下面这样编写代码，那么将会产生错误：

不会正常工作，因为 `Employees` 不是一个真正的集合。

```
For Each emp In sbMain.Employees
```

幸运的是，可以用 `EmployeeCount` 方法来为遍历范围分界。

“有问题”按钮也作了少许更改，但它仍然是“有问题”。

```
Private Sub cmdTrouble_Click()
```

```
    sbMain.Trouble
```

```
End Sub
```

运行该工程，并用“添加”、“删除”以及“刷新表”按钮作一下试验。每件事情工作起来，都跟以前是一样的。

当单击“有问题”按钮时，还是不产生任何错误。但是，如果现在单击“刷新表”按钮的话，就可以看到未初始化的 `Employee` 对象已经以某种方式被添加到集合中了。

怎么会发生这种情况？使 `Collection` 对象私有化以后，就保护它不受程序中所有在 `SmallBusiness` 对象外部的代码的影响了，但并没有防范内部代码的影响。在这种情况下，`SmallBusiness` 对象就可能由于内部有大量代码而变得很大，而且很复杂。例如，它将很可能具有象 `CustomerAdd`、`ProductAdd` 等等这些方法。

代码错误，或者 `EmployeeAdd` 方法的重复创建，仍然可能造成错误数据一

——甚至无效的对象——被插入到集合中，因为私有的变量在整个类模块中都是可见的。

创建自己的集合类：砖块盖的房子

实现集合的最强健方法，是使其成为类模块。与前面的示例相比较，将对象创建的所有代码移到集合类中，这就较好符合了面向对象程序设计的原则。

该例使用了跟前面的例子同样的窗体和同样的 `Employee` 类模块。插入新的类模块，并将其 `Name` 属性设置为“`Employees`”。将下面的声明和代码插入到这个新的类模块中。

```
Option Explicit
```

```
Private mcolEmployees As New Collection
```

`Employees` 类的 `Add`、`Count` 以及 `Delete` 方法，本质上都跟旧的 `SmallBusiness` 类中的那些方法是相同的。可以简单地把它们从 `SmallBusiness` 类模块中删除，然后粘贴到 `Employees` 类模块中，并更改一下它们的名字。

名字都可以更改，如不再需要将 `EmployeeAdd` 跟 `CustomerAdd` 区别开来。所实现的每个集合类都有自己的 `Add` 方法。

'`Employees` 集合类的方法。

```
Public Function Add(ByVal Name As String, _
```

```

ByVal Salary As Double) As Employee
    Dim empNew As New Employee
    Static intEmpNum As Integer
    '用 With 语句，使代码更快，也更简洁
    ' ( .ID 相对 empNew.ID)。
    With empNew
        '为新雇员产生一个唯一的 ID。
        intEmpNum = intEmpNum + 1
        .ID = "E" & Format$(intEmpNum, "00000")
        .Name = Name
        .Salary = Salary
        '将 Employee 对象引用添加到集合中，
        '用 ID 属性作为键。
        mcolEmployees.Add empNew, .ID
    End With
    '为新的 Employee 返回一个引用。
    Set Add = empNew
End Function

Public Function Count() As Long
    Count = mcolEmployees.Count
End Function

```

```
Public Sub Delete(ByVal Index As Variant)
```

```
    mcolEmployees.Remove Index
```

```
End Sub
```

SmallBusiness 对象的 **Employees** 方法变成了集合类中的 **Item** 方法。为了通过索引或者键来检索成员，它仍然委派 **Collection** 对象为代表。

'**Employees** 集合类的方法。

```
Public Function Item(ByVal Index As Variant) _
```

```
As Employee
```

```
    Set Item = mcolEmployees.Item(Index)
```

```
End Function
```

可以在这里添加一个很好的技巧。通过将 **Item** 设置为 **Employees** 类的缺省方法，就得到了为 **Employees("E00001")** 编写代码的能力，就像用 **Collection** 对象做的一样。

要使 **Item** 成为缺省的属性，请按照以下步骤执行：

1. 在“工具”菜单上，单击“过程属性”，打开“过程属性”对话框。在“名称”框中，选择 **Item** 方法。
2. 单击“高级”，显示高级功能。在“过程标识符”框中，选择“(缺省)”，使 **Item** 方法成为缺省的。单击“确定”。

注意 一个类只能有一个缺省成员（属性或者方法）。

使 **For Each ... Next** 有效

伴随着强健性而来的，是重新获得了 For Each ... Next。通过添加下面的方法，可以再一次将所有工作委派给 Collection 对象：

NewEnum 必须返回一个集合的枚举算子的 IUnknown 接口。

```
Public Function NewEnum() As IUnknown
```

```
    Set NewEnum = mcolEmployees.[_NewEnum]
```

```
End Function
```

委派给 Collection 对象的最重要的东西是它的枚举算子。枚举算子是一个小对象，该对象知道如何遍历集合中的各项。不能用 Visual Basic 来书写枚举算子对象，因为 Employees 类是基于 Collection 对象的，可以返回 Collection 对象的枚举算子——它自然知道如何枚举 Collection 对象所保存的各项。

包围 Collection 对象的 _NewEnum 方法的方括号是必要的，这是因为方法名前面的下划线。前面的这个下划线是一个约定，它指明该方法是隐藏在类型库中的。不能把自己的方法命名为 _NewEnum，但是可以把它隐藏在类型库中，并将 For Each ... Next 所需要的过程 ID 赋给它。

要隐藏 NewEnum 方法，并将所需的过程 ID 赋给它，请按照以下步骤执行：

1. 在“工具”菜单上，单击“过程属性”，打开“过程属性”对话框，在“名称”框中，选择 NewEnum 方法。
2. 单击“高级”，显示一些高级功能。选中“隐藏该成员”，使 NewEnum 隐藏在类型库中。
3. 在“过程标识符”框中，键入 -4，将 For Each ... Next 所需要的过程标

标识符赋给 `NewEnum`。单击“确定”。

重点 为了使自己的集合类使用 `ForEach ... Next` 来工作，必须提供有正确过程标识符的隐藏 `NewEnum` 方法。

SmallBusiness 类留下的东西不多了

到现在为止，`SmallBusiness` 类中还只有相当少的代码。为了取代所删除了的 `Collection` 对象和所有方法，有新的声明和只读属性：

```
Option Explicit
```

```
Private mEmployees As New Employees
```

```
Public Property Get Employees() As Employees
```

```
    Set Employees = mEmployees
```

```
End If
```

这需要简单解释一下。假设曾经省略了 `Property Get`，那么可以简单地声明 `Public Employees As New Employees`。

只要没有人犯什么错误，每件事物都将很好地工作，但是如果偶然编写代码为 `Set sbMain.Employees = Nothing`，将会发生什么情况？对，`Employees` 集合将被撤销。若是将 `Employees` 设置为只读属性，则可以避免这种可能性。

对窗体的改动

窗体模块的代码跟前面示例中的非常类似。可用模块级相同的声明，而且“关闭”按钮的 Click 事件也是同样的。在大多数事件过程中的唯一改动是：用 Employees 集合对象的新方法取代 SmallBusiness 类的旧方法：

```
Private Sub cmdEmployeeAdd_Click()  
    sbMain.Employees.Add txtName.Text, txtSalary.Text  
    txtName.Text = ""  
    txtSalary.Text = ""  
    cmdListEmployees.Value = True  
End Sub
```

```
Private Sub cmdEmployeeDelete_Click()  
    '检查一下，以确保有雇员被选中。  
    If lstEmployees.ListIndex > -1 Then  
        '前六个字符是 ID。  
        sbMain.Employees.Delete _  
            Left(lstEmployees.Text, 6)  
    End If  
    cmdListEmployees.Value = True  
End Sub
```

```
Private Sub cmdListEmployees_Click()
```

```
Dim emp As Employee
lstEmployees.Clear
For Each emp In sbMain.Employees
    lstEmployees.AddItem emp.ID & ", " & emp.Name _
        & ", " & emp.Salary
Next
End Sub
```

注意，又可以用 `For Each ... Next` 来列出雇员了。
运行该工程，并验证每件事物都能正常工作。这次“有问题”按钮没有任何代码，因为封装已经消除了问题。

详细信息 为了得到关于背景集合的信息，请阅读“Visual Basic 集合对象”和“Visual Basic 集合”。

在专业版和企业版中，包括了 `Class Builder` 实用工具，该实用工具可以创建集合类。

好的面向对象程序设计的优点

创建 `Employees` 集合类，其结果是产生了一种非常清晰、模块化的编程风格。这个集合的所有代码都在集合类中（封装），从而减小了 `SmallBusiness` 类模块的大小。如果 `Employee` 对象的集合出现在对象分层结构的不止一处，那么重用集合类将不需要复制任何代码。

增强集合类

可以为自己的集合类实现附加的方法和属性。例如，可以实现 `Copy` 和 `Move` 方法，或者包含 `SmallBusiness` 对象引用的只读 `Parent` 属性。也可以添加事件。例如，每当 `Add` 或者 `Remove` 方法更改集合中的项数时，可以产生一个 `CountChanged` 事件。

强健性，强健性，强健性

并不总是必须以可能的最强健的方式来实现集合。不过，面向对象编程的一个优点是代码重用；重用对象比起复制源代码来要容易的多，而且如果使用的是强健的、封装的代码，则会更安全些。一位聪明的人说过：“如果想书写真正强健的代码，一定要假设会发生真正糟糕的事情。”

集合类和部件软件

如果使用的是 `Visual Basic` 的专业版或者企业版，可以将自己的工程转变成 `ActiveX` 部件，这样，同组的其他程序员就可以使用已经创建的对象了。

实现集合类的步骤

下面的清单总结了创建集合类所需要的步骤。

1. 将类模块添加到工程中，并为它起一个名字——通常是该集合类将包含

的对象名的复数形式（请参阅本章前面的“命名属性、方法和事件”一节）。

2. 将私有变量添加进去，该私有变量包含对属性和方法所委派的 `Collection` 对象的引用。
3. 在 `Class_Initialize` 事件过程中，创建 `Collection` 对象（如果希望将该对象的创建时间推迟到需要时，可以在第二步中将私有变量声明为 `AS New Collection`。这样每次访问 `Collection` 时会增加少量开销）。
4. 将 `Count` 属性和 `Add`、`Item` 以及 `Remove` 方法添加到类模块中；在每种情况下，通过调用其相应的成员，委派给私有的 `Collection`。
5. 实现 `Add` 方法时，通过只接受一种类型的对象，可以覆盖 `Collection` 对象的无鉴别力的 `Add` 方法。甚至可以让外部创建的对象添加到自己的集合中，这样 `Add` 方法就完全控制了对对象的创建和初始化。
6. 用“过程属性”对话框，使“`Item`”方法成为集合类中缺省的方法。
7. 像下面显示的那样，添加一个“`NewEnum`”方法。用“过程属性”对话框来将其标志为隐藏的，并将一个值为 `-4` 的过程标识符给它，这样，它就可以使用 `For Each ... Next`。

```
Public Function NewEnum() As IUnknown  
    Set NewEnum = mcol.[_NewEnum]  
End Function
```

注意 上面的代码假设第二步中的私有变量被命名为 `mcol`。

8. 将自定义属性、方法和事件添加到集合类中。

注意 在 Visual Basic 的专业版和企业版所包括的 `Class Builder` 实用工

具，可用来创建集合类。就可以定制所得到的源代码。

详细信息 可以在《部件工具指南》中的“创建 ActiveX 部件”中读到更多关于软件部件的信息。

ActiveX 设计器

设计器在 Visual Basic 开发环境中提供了一个可视化的设计窗口。可以用这个窗口来可视化地设计新的类。Visual Basic 为窗体和——在专业版和企业版中——ActiveX 控件和文档提供了内置的设计器。

在以这种方式设计的类中所创建的对象，其具有设计时和运行时不同的行为和外观，尽管许多对象——比如窗体和控件——在这两种模式中看起来非常相似。

除了内置的设计器外，Visual Basic 还允许第三方来开发设计器，以在 Visual Basic 开发环境中使用。这些 ActiveX designers 工作起来就跟 Visual Basic 中内置的设计器一样，使得它们易学和易用。

什么是 ActiveX 设计器

ActiveX 设计器可以为任务提供可视化的接口——否则的话可能需要大量的代码。例如，在 Visual Basic 企业版中包括 UserConnection 设计器，该

设计器为定义复杂的数据库查询提供了可视化工具。这样，在运行时，就可以用很少的代码来调用这些查询。

ActiveX 设计器和内置设计器之间的相似之处

ActiveX 设计器在下面这些方面就像窗体设计器一样：

- ActiveX 设计器产生可以从中创建对象的类。这些类出现在“工程”窗口内，就像窗体类一样。
- 用 ActiveX 设计器所创建的每个类，都有自己的代码模块，在该代码模块中可以为该设计器所提供的事件过程书写代码。
- 通过将自己的属性、方法以及事件添加到 ActiveX 设计器所提供的属性、方法以及事件之中，可以定制自己的类。
- 从自己设计的类中创建的对象可能在设计时和运行时有不同的特点。
- ActiveX 设计器的设计窗口是完全集成到开发环境中的。可以调整其大小和对其进行安排，就像对待内置的设计窗口那样。
- 工程中需要多少 ActiveX 设计器的实例，就可以将多少这种实例添加到工程中，这就像需要多少窗体设计器，就可以添加多少一样。

图 9.14 将内置的 Visual Basic 窗体设计器跟 UserConnection 设计器（包括在 Visual Basic 企业版中的一种 ActiveX 设计器）作了比较。

ActiveX 设计器类跟其它可视化设计类的比较

ActiveX 设计器是非常灵活的。其中有一些，如 UserConnection 设计器，

创建的类的实例在运行时是可编程的，但却是不可见的。另外一些，如 Microsoft Office 所使用的 Microsoft Forms 设计器，可产生跟 Visual Basic 窗体类似的可见对象。

具有可见的运行时部件的 ActiveX 设计器，可以成为宿主 ActiveX 控件。事实上，它们变成了另一种窗体包，除了 Visual Basic 本地窗体外，也可使用该包。

下面的列表将 ActiveX 设计器产生的类跟用内置的 Visual Basic 设计器所产生的类作了比较。

- 如果从 ActiveX 设计器类中创建出来的对象在运行时是可见的，那么它有自己的窗口。它并不像 ActiveX 控件那样，被包含在另一个窗体中。
- 跟窗体类一样，但跟 ActiveX 控件不一样，ActiveX 设计器所产生的类都是私有类。如果用 Visual Basic 的专业版或者企业版来创建 ActiveX 部件，那么就不能将使用这些类的公用的方法声明为参数类型或者返回类型。

例如，如果象下面这样的方法声明出现在公用类中的话，那么将会产生编译错误：

```
Public Function A() As UseConnection1          '错误
```

```
Public Sub B(Callback As UseConnection1)       '错误
```

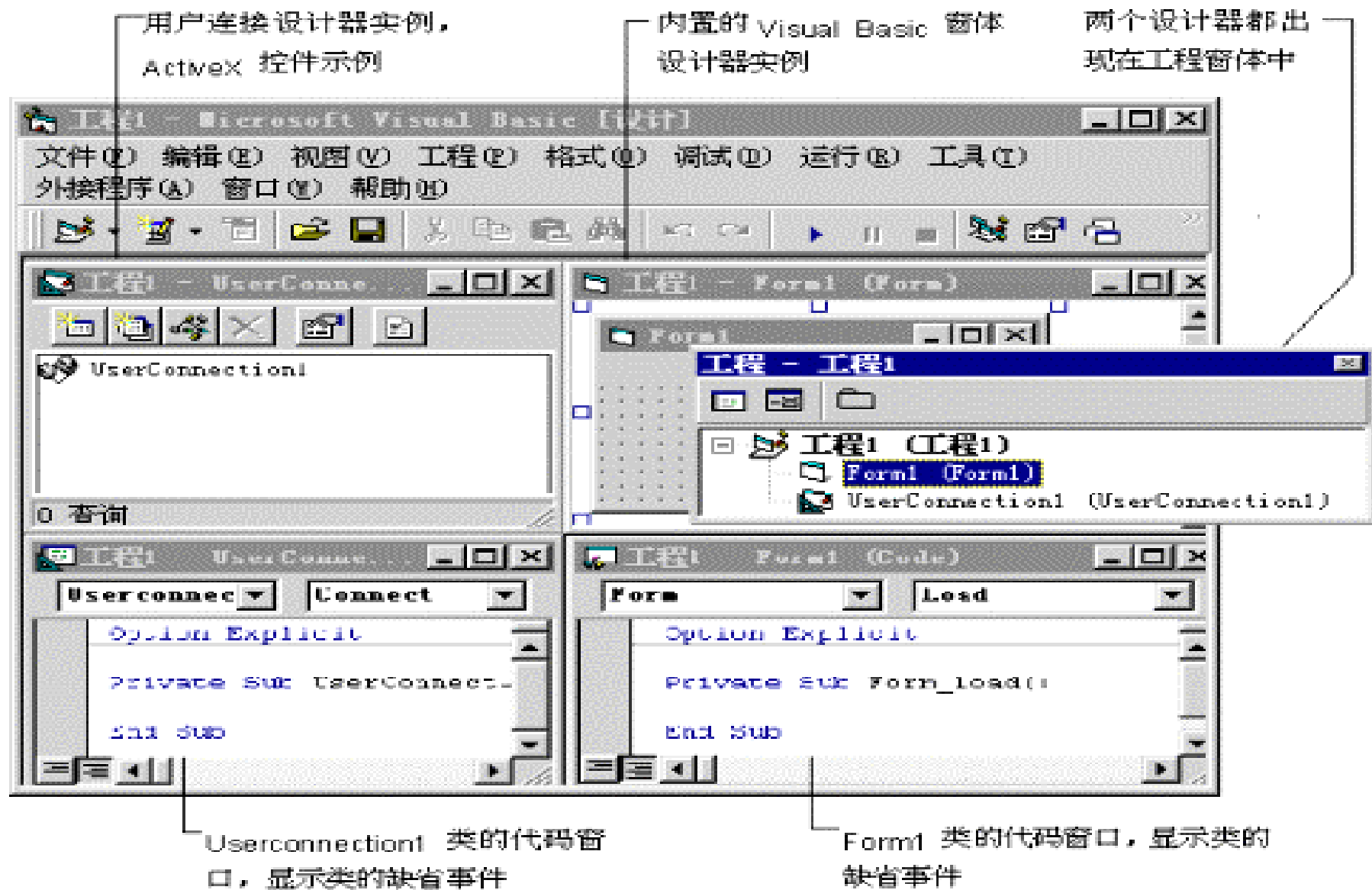


图 9.14 ActiveX 设计器和内置的 Visual Basic 设计器

警告 尽管通过将返回值声明为 `As Object`，可以将引用传递给工程外的私有对象，但这是一种很糟糕的编程实践，而且可能会使得程序不稳定。关于更多的信息，请参阅《Visual Basic 6.0 部件工具指南》中的“创建 ActiveX 部件”部分。

运行时使用 ActiveX 设计器对象

跟内置的窗体设计器一样，ActiveX 设计器也仅仅在开发环境中是可用的。一旦将工程连编成可执行文件，它仅仅使用 ActiveX 设计器的运行时的 .dll。这可能比设计时的 .dll 小得多，因为它不包括可视化的设计工具。图 9.15 图示了概念。

前面已经提到过，ActiveX 设计器可产生在运行时不可见对象的类。图 9.14 中所显示的 UserConnection 设计器就是一个这样的例子。UserConnection 设计器产生一些这样的类——其对象管理运行时跟 SQL 数据库的连接。没有理由让这些对象在运行时可见。

为了使用用 UserConnection 设计器所创建的类，应为该类声明一个类型变量，并为该类创建一个实例。例如，如果添加一个 UserConnection 设计器，并将其 Name 属性设置为 GeneralLedger，那么就可以创建一个 GeneralLedger 对象，如下面这段代码中所示：

'标准模块中的一个全局变量，这是为了保存

'对 GeneralLedger 对象的引用。

```
Public gGeneralLedger As GeneralLedger
```

'在 Form 模块中的代码，这是为了创建 GeneralLedger
'对象并建立一个数据库连接。

```
Private Sub Command1_Click()  
    Set gGeneralLedger = New gGeneralLedger  
    gGeneralLedger.EstablishConnection  
    '（使用该对象的代码。）  
End Sub
```

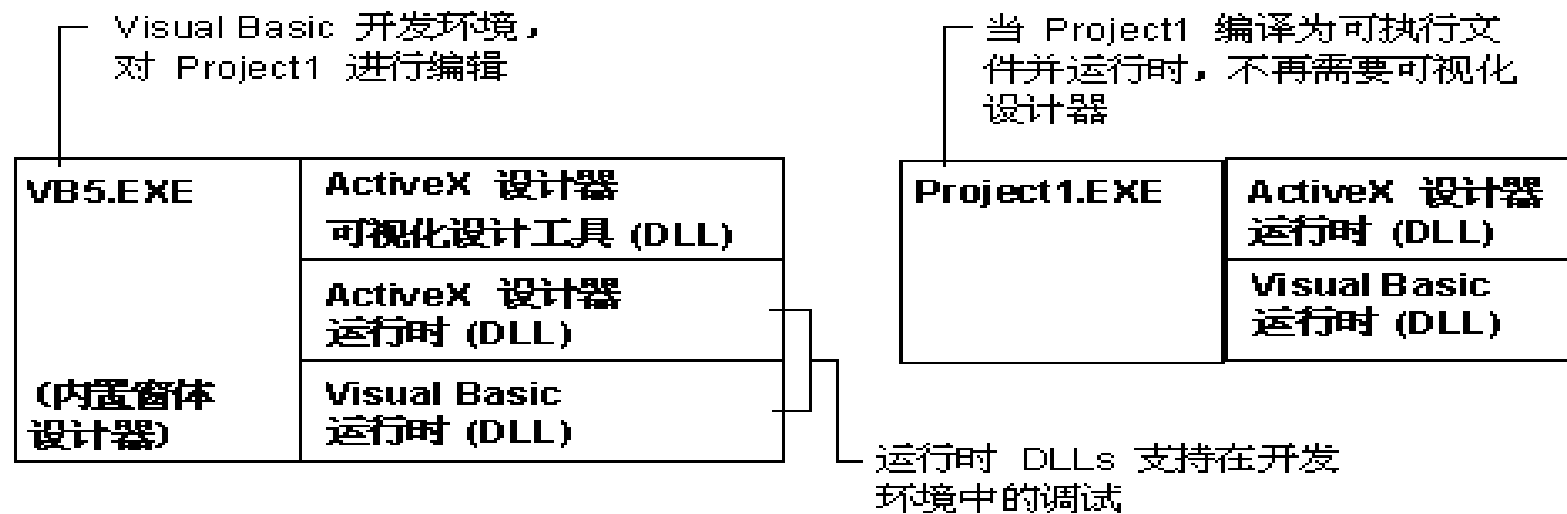


图 9.15 内存中的设计器部件

创建 ActiveX 设计器

使用 ActiveX 设计器的 Software Development Kit，可以用来为使用 Visual Basic 而创建新的 ActiveX 设计器。SDK 包括全部指令以及示例代码。您可以在 Microsoft Development Network 的“SDK Documentation”标题下找到它。

注意 ActiveX Designer SDK 需要一个 C++ 编译器，如 Microsoft Visual C++。ActiveX 设计器不能用 Visual Basic 来编写。

详细信息 在“向‘工程’菜单中添加 ActiveX 设计器”和“插入 ActiveX 设计器的新实例”中提供了将 ActiveX 设计器结合到工程中去的过程。

向“工程”菜单中添加 ActiveX 设计器

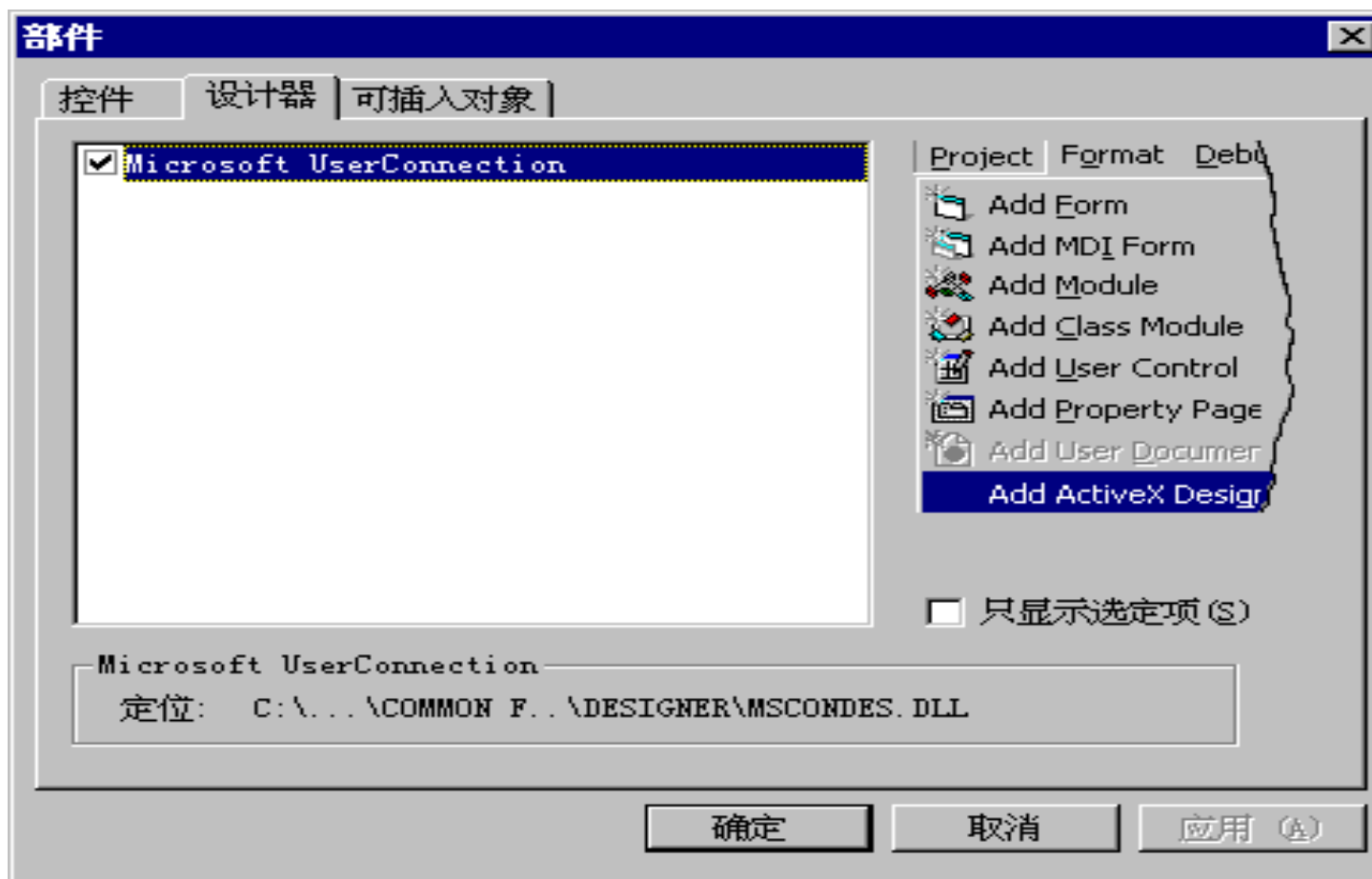
用代理商提供的 Setup 程序安装了一个新的 ActiveX 设计器后，必须将设计器添加到“工程”菜单中，才可以在工程中使用它。

安装 ActiveX 设计器时，应当在 Windows 注册表中适当的部件类别下对它进行注册。然后就可以从“部件”对话框的“设计器”选项卡上，使用该设计器了。

要将 ActiveX 设计器添加到“工程”菜单中，请按照以下步骤执行：

1. 在“工程”菜单上，单击“部件”，打开“部件”对话框。
2. 单击“设计器”选项卡，并选择想要使用的那个设计器，如下图所示。

单击“确定”。



注意 Microsoft Forms 设计器包括在所有版本的 Visual Basic 中，这是为了使得在 Microsoft Office 应用程序中所创建的窗体很容易接口。跟所有的设计器一样，Microsoft Forms 设计器有自己的运行时 .dll。因此，在 Visual Basic 工程中使用该设计器将会增加最后生成的可执行

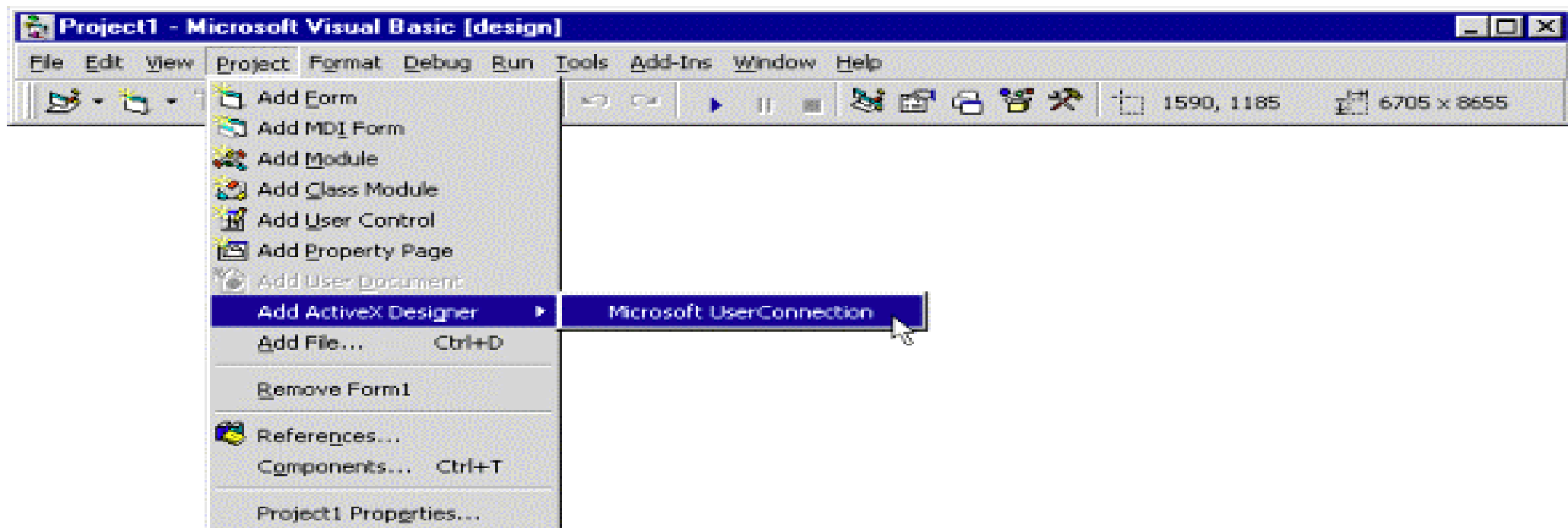
文件的内存需求量。

插入 ActiveX 设计器的新实例

一旦使用“部件”对话框将一个设计器添加到“工程”菜单中以后，需要多少该设计器的实例，就可以插入多少这种实例。

要插入 ActiveX 设计器的实例，请：

- 在“工程”菜单上，单击“添加 ActiveX Designer”（其中，designer 是设计器的名称）以显示已安装设计器的清单。从清单中选择想要使用的设计器如下图所示。



一旦将 ActiveX 设计器的实例添加到自己的工程中，就可以使用其可视化的接口来设计类了。可以单击“工程”窗口上的“代码”按钮来为该设计器打开代码模块，并将代码添加到事件过程中。可以通过添加进自己的属性、方法以及事件，来进一步定制类，就跟在 Visual Basic 类模块中所做的一样。

详细信息 设计器是在“ActiveX 设计器”中引入的。为了将一个设计器添加到“工程”菜单中，请参阅上面的“向‘工程’菜单中添加 ActiveX 设计器”。

第十章 用部件编程

您需要 Visual Basic 应用程序具有 Microsoft Excel 一样的分析与计算功能吗？或者，也许要用 Microsoft Word 的格式化工具来格式化文档，或用 Microsoft Jet 数据库引擎来存储管理数据。更理想地，您想不想能够创建或购买标准部件，并且不加修改地用在多个应用程序中？

所有这些，或者比这更强的功能，通过使用 ActiveX 部件创建自己的应用程序，都可以实现。ActiveX 部件是一段可重复使用的编程代码和数据，它是由用 ActiveX 技术创建的一个或多个对象所组成。应用程序可以使用现有的部件，比如包含在 Microsoft Office 应用程序中部件、各种各样制造厂商所提供的代码部件、ActiveX 文档或 ActiveX 控件（通常称为 OLE 控件）中含有的部件。或者，如果您有 Visual Basic 专业版或企业版，就能开发自己的 ActiveX 控件。

对于支持对象链接和嵌入的部件，可以通过部件的可视界面，在自己的应用程序中插入对象，而不必写任何代码。通过使用 OLE 容器控件或在工具箱中添加对象类，可以在自己的应用程序中插入 OLE-enabled 对象。

为了充分理解 ActiveX 部件，首先应当熟悉如何操作类、对象、属性和方法，这在第九章“用对象编程”中已有说明。

目录

- ActiveX 部件的类型
- 进程内的服务器和进程外的服务器
- 使用 ActiveX 部件
- 创建对对象的引用
- 使用 ActiveX 部件的属性、方法和事件
- 释放 ActiveX 部件
- 定位对象模型
- 处理 ActiveX 部件中的运行时错误
- 处理挂起 ActiveX 部件的请求
- 使用虚拟界面部件

示例应用程序 Geofacts.vbp 和 Olecont.vbp

本章中的很多概念，在示例 Geofacts.vbp 和 Olecont.vbp 中都有说明。该示例应用程序列在 Samples!Alink("vbsamples") 目录中。

ActiveX 部件的类型

ActiveX 部件是将现已存在的、完善的应用程序片断连在一起的强有力手

段。Visual Basic 应用程序可以包含各种类型的 ActiveX 部件：

- 支持 ActiveX 技术的应用程序，如 Microsoft Excel, Microsoft Word 和 Microsoft Access，提供了能从 Visual Basic 应用程序内部来程序化操纵的对象。例如，在应用程序中，可以使用 Microsoft Excel 的电子数据表、Microsoft Word 的文档或者 Microsoft Access 数据库的属性、方法和事件。
- 代码部件提供了可编程对象的库。例如，一个代码部件包含为电子数据表用户提供的财务专用函数库，或诸如对话框那样的对多种应用程序通用的用户界面元素。和 ActiveX-enabled 应用程序中的对象不同的是，代码部件中的对象和应用程序运行在同一进程中，所以能较快地访问到对象。
- 可以利用 ActiveX 控件作为部件来增加功能，而不用自己去创建。不同厂商制造的 ActiveX 控件提供了许多特殊功能，比如在窗体上显示日历牌，用特定的格式读取数据等。
- ActiveX 文档可创建交互式的 Internet 应用程序。您可以创建包含在 Internet Explorer 中的窗体。ActiveX 文档可以显示信息框及次级窗体，且包含 ActiveX 控件。ActiveX 文档还可以有象代码部件那样的功能。为了逐步介绍 ActiveX 文档，请参阅《部件工具指南》中的“创建 ActiveX 文档”，这些内容，在专业版和企业版中都有。

注意 除了使用 ActiveX 文档的 Internet 应用程序外，您也可以使用 Visual Basic 代码和 HTML 页面的组合创建基于客户和基于服务器的 Internet 应用程序。有关按您的选择创建 Internet 和 intranet 应用程

序的详细信息，请参阅《部件工具指南》中《建立 Internet 应用程序》一书中的“Internet 应用程序介绍”。这些在企业版和专业版中可以得到。

有些 ActiveX 部件与应用程序运行在相同的进程中，而另一些则运行在独立进程中。详细信息，请参阅“进程内的服务器和进程外的服务器”。

除了现存的 ActiveX-enabled 应用程序中的部件、代码部件库、ActiveX 控件和 ActiveX 文档外，还可以创建自己的部件。关于创建自己的 ActiveX 部件的更详细的信息，请参阅《Visual Basic 6.0 部件工具指南》中的“创建 ActiveX 部件”，这些内容在专业版和企业版中都有。

进程内的服务器和进程外的服务器

ActiveX 部件通过客户/服务器关系与应用程序——及与部件相互之间——交互作用。客户端是使用部件功能的应用程序代码或部件。服务器是部件及其关联的对象。例如，假设应用程序使用 ActiveX 控件来提供一个标准的雇员窗体，供公司的多种应用程序使用。提供雇员窗体的 ActiveX 控件就是服务器，使用这个控件的应用程序就是服务器的客户端。

ActiveX 部件可能和客户端的应用程序运行在同一个进程中，也可能运行在不同的进程中，这要取决于该 ActiveX 部件是如何实现的。例如，如果应用程序使用一个部件，该部件是一个 ActiveX-enabled 应用程序的一部分，则它在一个单独的进程中运行。如果该部件已作为动态链接库（.dll 文

件) 中的一个可编程对象被实现, 则它运行在和应用程序同一进程中。通常, 如果 **ActiveX** 部件作为一个可执行文件 (.exe 文件) 的一部分被实现, 它就是一个进程外的服务器, 并且运行在它自己的进程中。如果, 它已作为一个动态链接库被实现, 则它是一个进程内的服务器, 并且和客户端应用程序运行在同一个进程中。通常, 使用进程内的服务器的应用程序比使用进程外的服务器的应用程序执行起来要快些, 因为应用程序不必跨越进程的边境来使用对象的属性, 方法和事件。

下列表示出如何实现不同类型的部件:

部件	服务器类型
ActiveX-enabled 应用程序	进程外的服务器
代码部件	进程内的服务器或者进程外的服务器
ActiveX 控件	进程内的服务器
ActiveX 文档	进程内的服务器或者进程外的服务器

使用进程内的服务器部件, 是优化应用程序性能的一种方法。另一种方法是使用事前绑定。详细信息, 请参阅本章后面的“加速对象引用”。

使用 **ActiveX** 部件

使用 **ActiveX** 部件提供的对象与操作其他对象的方法十分相似。首先将一个对象引用赋值给变量, 然后编写使用对象的方法、属性与事件的代码。

但是，在操作部件提供的对象时，还是有一些事情需要注意的。
这个主题综述了操作部件提供的对象的顶层任务，及使用 `ActiveX-enabled` 应用程序中的对象的例子。关于每个任务的细节，请参阅每个任务项目下描述的有关主题。

要使用 `ActiveX` 部件提供的大多数对象，请按照以下步骤执行：

1. 创建对要使用的对象的引用。如何做这件事，取决于对象的类型及 `ActiveX` 部件是否提供类型库。

关于更详细的信息 请参阅本章后面的“创建对对象的引用”。

2. 用对象的方法、属性与事件编写代码。

关于更详细的信息 请参阅本章后面的“使用对象的属性、方法与事件”。

3. 当对象使用完毕后，将其释放。

关于更详细的信息 请参阅本章后面的“释放 `ActiveX` 部件”。

4. 创建错误处理程序

关于更详细的信息 请参阅本章后面的“在 `ActiveX` 部件中处理运行时错误”。

例如，假设已经创建一个包含三个文本框 (`Text1`, `Text2`, `Text3`) 和一个命令按钮 (`Command1`) 的窗体，并在工程中添加了对 `Microsoft Excel 8.0` 对象库的引用。接着，可将代码加到命令按钮的 `Command1_Click` 事件过程中，该过程用 `Microsoft Excel Formula` 方法将 `Text1` 和 `Text2` 中的两个数相加，在 `Text3` 中显示结果。（为避免类型不匹配错误，可能要删除每个文本框的缺省文本值，方法是将其 `Text` 属性置为空字符串）：

```
Private Sub Command1_Click()
```

'为 Microsoft Excel 应用程序、工作簿
'和工作表对象声明对象变量。

```
Dim xlApp As Excel.Application  
Dim xlBook As Excel.Workbook  
Dim xlSheet As Excel.Worksheet
```

'将对象引用赋值给这些变量。
'用 Add 方法 创建新的工作簿和工作表对象。

```
Set xlApp = New Excel.Application  
Set xlBook = xlApp.Workbooks.Add  
Set xlSheet = xlBook.Worksheets.Add
```

'将文本框中的值赋给 Microsoft Excel 单元。

```
xlSheet.Cells(1, 1).Value = Text1.Text  
xlSheet.Cells(2, 1).Value = Text2.Text
```

'用 Formula 方法将 Microsoft Excel 中的值相加。

```
xlSheet.Cells(3, 1).Formula = "=R1C1 + R2C1"  
Text3.Text = xlSheet.Cells(3, 1)
```

'保存工作表

```
xlSheet.SaveAs "c:\Temp.xls"
```

' 关闭工作表。

```
xlBook.Close
```

'用 Quit 方法关闭 Microsoft Excel

```
xlApp.Quit
```

'释放对象

```
Set xlApp = Nothing
```

```
Set xlBook = Nothing
```

```
Set xlSheet = Nothing
```

```
End Sub
```

为了简单起见，这个例子不包括错误处理。然而，在使用由 **ActiveX** 部件提供的对象的应用程序中，推荐您包括错误处理。

创建对对象的引用

在应用程序中，在使用对象的属性、方法和事件之前，必须先声明对象变量，然后将对象引用赋予该变量。如何赋值对象引用取决于两个因素；

- **ActiveX** 部件是否提供类型库。**ActiveX** 部件的类型库包含部件提供的全部对象的定义，包括全部可用的方法、属性和事件的定义。如果 **ActiveX** 部件提供类型库，在使用库的对象前，需要在 **Visual Basic** 工程中添加

一个对类型库的引用。

- 该对象是顶层、外部可创建对象，还是从属对象。对于外部创建的对象
的引用，可以直接赋值，而对从属对象的引用则间接赋值。

如果对象是外部可创建的，可在 **Set** 语句中用 **New** 关键字、**CreateObject** 或 **GetObject** 从部件外面将对象引用赋予变量。如果对象是从属对象，则需使用高层对象的方法，在 **Set** 语句中指定一个对象引用。

例如，在 **Microsoft Excel** 中，应用程序对象是外部可创建的对象，可以从 **Visual Basic** 应用程序中，通过在 **Set** 语句中使用的 **New** 关键字、**CreateObject** 或 **GetObject** 直接赋值引用。相反，**Range** 对象是从属对象，则通过在 **Set** 语句中使用 **Worksheet** 对象的 **Cells** 方法赋值引用。关于外部可创建的对象与从属对象的详细信息，请参阅本章后面的“定位对象模型”。

如果，对象的类包括在类型库中，那么用特定类的变量来创建对象引用，能使应用程序运行得更快。否则，必须使用一般的 **Object** 类的变量，它导致后期绑定。更详细的信息，请参阅“加速对象引用”一节。

要创建对于在类型库中定义的对象引用，请按照以下步骤执行：

1. 从“工程”菜单中选择“引用”。
2. 在“引用”对话框中，选择 **ActiveX** 部件的名称，它包含在应用程序中使用的对象的。
3. 可以使用“浏览”按钮来搜索包含所需对象的类型库文件。类型库可以有 **.tlb** 或 **.olb** 扩展文件名。可执行 (**.exe**) 文件与动态链接库 (**dlls**) 也可以提供类型库，所以也能使用这些文件的扩展名搜索文件。

如果不能肯定应用程序是不是 **ActiveX-enabled** 和能否提供类型库，可以用“浏览”按钮添加一个对于它的引用。如果引用失效，**Visual Basic** 显示错误信息“不能添加对指定文件的引用”，表示该类型库不存在。关于操作与类型库不相关联对象的详细内容，请参阅“创建对对象的引用”一节。

4. 从“查看”菜单中，选择“对象浏览器”查看引用的类型库。从“工程/库”清单中选择适当的类型库。在应用程序中可以使用在“对象浏览器”中列出的全部对象、方法和属性。

关于使用“对象浏览器”的更详细的信息 请参阅“浏览 **ActiveX** 部件类型库”。

5. 声明对象类的对象变量。例如，可以声明类 **Excel Chart** 的变量来引用 **Microsoft Excel Chart** 对象。

```
Dim xlChart As Excel.Chart
```

关于更详细的信息 请参阅本章后面的“声明对象变量”。

6. 用 **Set** 语句中的 **New** 关键字、**CreateObject** 或 **GetObject** 将对象引用赋予变量。关于更详细的信息，请参阅本章后面的“将对象引用赋予变量”。如果对象是从属对象，则需使用高层对象的方法，在 **Set** 语句中指定一个对象引用。

要创建对未在类型库中定义的对象引用，请按照以下步骤执行：

1. 声明 **Object** 数据类型的对象变量。

因为对象与类型库不相关联，所以不能用“对象浏览器”查看该对象的属性、方法和事件。需要知道对象所提供的是什么样的属性、方法和事

件，以及创建对从属对象的引用的任何方法。

关于更详细的信息 请参阅本章后面的“声明对象变量”。

2. 在 `Set` 语句中使用 `CreateObject` 或 `GetObject` 给变量赋值对象引用。

关于更详细的信息，请参见本章后面的“将对象引用赋予变量”。

如果对象是从属对象，则需使用高层对象的方法，在 `Set` 语句中指定一个对象引用。

二义性引用与引用优先权

当代码中引用常数或对象时，在每一个“引用”对话框中选中的类型库中，`Visual Basic` 将按照类型库被显示的次序，搜索常数或对象类。如果两个类型库包含名字相同的常数或类，`Visual Basic` 使用列在“可使用的引用”框中较高位置上的类型库提供的定义。

明确地指定在使用时提供常数或类的类型库，是处理可能的二义性引用的最好方法。例如常数 `vbCancel`，在 `Visual Basic` 与 `Visual Basic Applications` 类型库中代表不同的值。以下代码展示了对常数 `vbCancel` 的充分限定和二义性引用。

```
'打印 Visual Basic vbCancel.
```

```
Debug.Print "VB.vbCancel = "; VB.vbCancel
```

```
'打印 Visual Basic for Applications vbCancel.
```

```
Debug.Print "VBA.vbCancel = "; VBA.vbCancel
```

```
'二义性引用打印 "Available Reference"
```

'清单最高端的 Visual Basic for Applications

'中的常数 vbCancel 的值。

```
Debug.Print "vbCancel = "; vbCancel
```

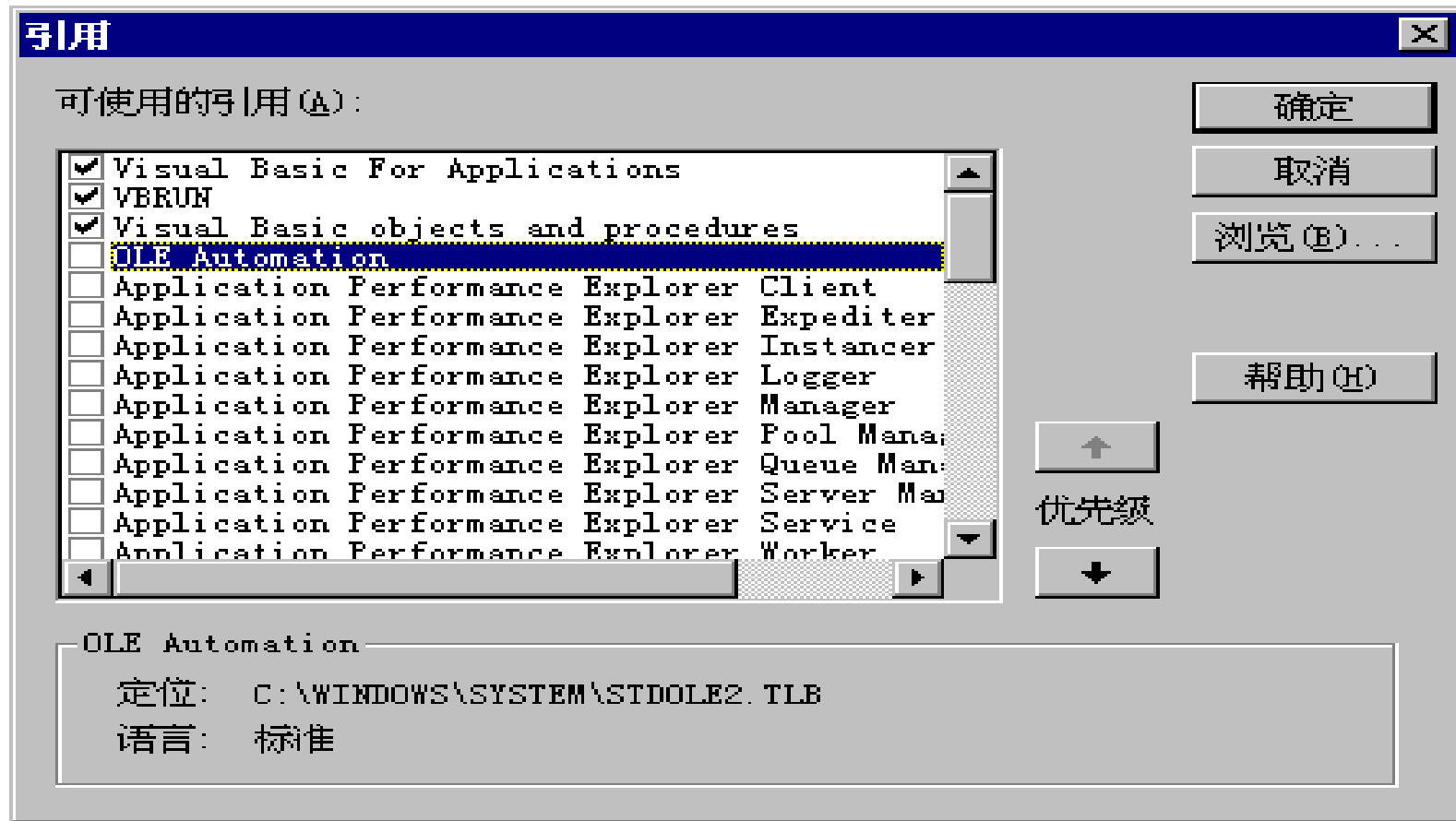


图 10.1 “引用”对话框

下列代码示例表示对一个 Application 对象变量的充分限定与二义性声明。

假定在“可使用的引用”框中，Microsoft Word 出现在 Microsoft Excel 之前，xlApp2 是用 Microsoft Word Application 类加以声明，而不是用 Microsoft Excel Application 类声明的。

'充分限定的对象变量声明

```
Dim xlApp1 As Excel.Application
```

'二义性对象变量声明

```
Dim xlApp2 As Application
```

'赋值对象引用

```
Set xlApp1 = New Excel.Application
```

'下面产生类型不匹配错误。

```
Set xlApp2 = xlApp1
```

您可能会想要利用改变 Visual Basic 搜索引用的顺序，来处理可能的二义性引用。“引用”对话框有两个“优先级”按钮，用这两个按钮可以把一个类型库移到清单中的较高的地方，这样，它的常数和类会比在清单较低位置而名字相同的常数和类较快地被找到。但是，如果还有其他二义性的引用存在，改变优先权顺序，会在应用程序中引起预料不到的问题。通常，最好还是在引用中明确规定类型库。

注意 在 Microsoft Excel 97 以前的版本不支持引用 Microsoft Excel Application 类的 Excel Application 语法。要在 Microsoft Excel 5.0 和 Microsoft Excel 95 之中引用 Microsoft Excel Application 类，需用 [_ExcelApplication] 语法。例如：


```
Set xlApp = New [_ExcelApplication]
```

浏览 ActiveX 部件类型库

如果 ActiveX 部件提供一个类型库，可以用“对象浏览器”查看部件的类以及与每个类的对象相关的属性、方法、事件和常数。

要查看在 ActiveX 部件的类型库中有效的类，请按照以下步骤执行：

1. 如果还没这样做，则在 Visual Basic 工程中添加一个对类型库的引用。
详细信息，请参阅本章后面的“创建对对象的引用”一节。
2. 打开“对象浏览器”，并从“工程/库”清单中选择类型库名。
“对象浏览器”在“类”清单中显示有效类。

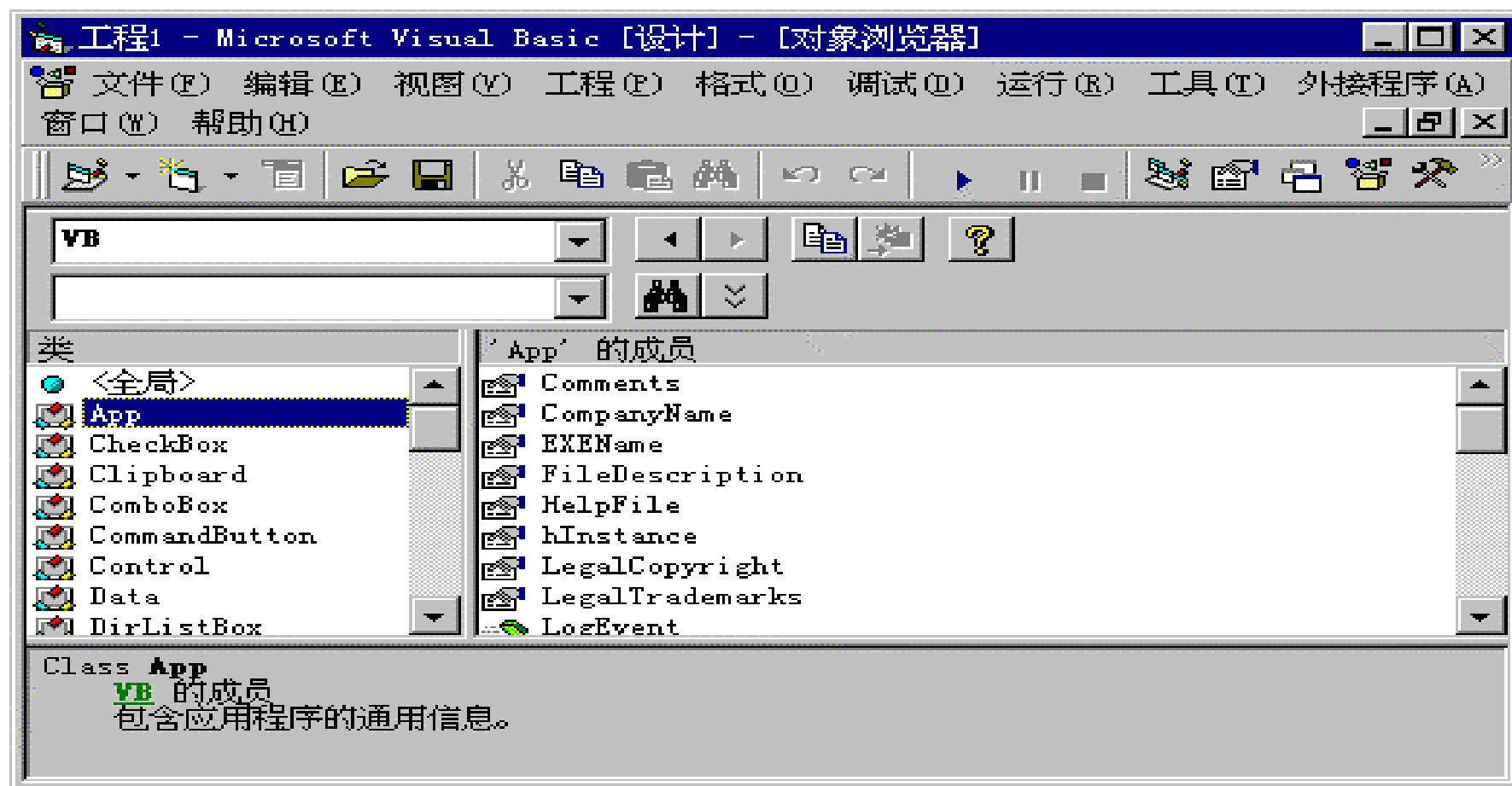


图 10.2 “对象浏览器”

例如，为了查看在 Data Access Object (DAO) 类型库中有效的类，可在“引用”对话框中添加对该库的引用，然后在“对象浏览器”的“工程/库”清单中选择 DAO。

要查看类的成员，请：

- 在“对象浏览器”的“类”清单中选择类名。
“对象浏览器”在“成员”清单中显示类的成员。
- 如果要在类型库中寻找关于特定类或成员的信息，使用“对象浏览器”的“搜索”功能。
- 要使用“搜索”功能，请
- 在“搜索文字”框中输入要寻找的内容，然后单击“搜索”按钮。
“对象浏览器”显示“搜索结果”框，示出由搜索返回的库、类和成员。

声明对象变量

在使用由 **ActiveX** 部件提供的对象的属性、方法和事件之前，必须首先声明一个对象变量。声明对象变量的方法，取决于 **ActiveX** 部件是否提供类型库。

要为在类型库中定义的对象声明变量，请按照以下步骤执行：

1. 在 **Visual Basic** 工程中添加一个对类型库的引用。关于添加对类型库的引用的更详细的信息，请参阅本章前面的“创建对对象的引用”一节。
2. 在变量声明中指定由那个类型库提供类名。声明一个具体类的对象变量可加速对象引用。使用下列语法：

```
Dim variable As [New] class
```

class 参数由两部分组成，其格式为 **component.class**。

部 分	说 明
component	提供对象的部件名。选择的内容被显示在“对象浏览器”的“工程/库”清单中
class	对象的类名（由部件类型库提供）。选择的内容被显示在“对象浏览器”的“类/模块”框中

例如，可以使用下列两种方式之一为 Microsoft Excel Chart 对象声明一个变量：

```
Dim xlChart As Chart
```

```
Dim xlChart As Excel.Chart
```

如果用 **New** 关键字声明一个对象变量，Visual Basic 将自动创建一个对象，并在第一次使用该变量时赋值对象引用。例如下列语句将对新的 DAO 表对象的引用赋予变量 **tdfOrders**，表的 **Name** 属性被置为 “Orders”：

```
Dim tdfOrders As New TableDef
```

```
tdfOrders.Name = "Orders"
```

注意 使用由 **New** 关键字声明的变量会减慢应用程序的运行速度。因为 Visual Basic 每次遇到用 **New** 声明的变量时，都要测试是否已将对象引用赋给该变量。

要为未在类型库中定义的对象声明对象变量，请：

- 声明一般的 **Object** 类的对象变量，使用如下语法：

```
Dim variable As Object
```

例如，下面声明中的变量 **objAny** 可以被 Microsoft Excel Chart 对象或

ActiveX 部件提供的任何其他对象所使用。

`Dim objAny As Object`

声明特定类变量与声明一般的 `Object` 类的变量不同，区别在于 ActiveX 如何将变量和对象绑定。在声明一般的 `Object` 类变量时，ActiveX 必须用后期绑定。在声明特定类的对象变量时，ActiveX 用事前绑定，这样能加速对象引用。详细信息，请参阅本章后面的“加速对象引用”一节。

详细信息 关于声明对象变量的更详细的信息，请参阅“`Dim` 语句”。关于给变量赋值对象引用的更详细的信息，请参阅“将对象引用赋予变量”。

将对象引用赋予变量

在声明对象变量之后，必须给变量赋值对象引用，才能使用对象的属性、方法及事件。可以用几种方法赋值一个新的对象引用：

- 如果用 `New` 关键字声明了变量，在第一次使用该变量时，Visual Basic 会自动地赋值一个新的对象引用。
- 可以用 `New` 关键字或 `CreateObject` 函数，在 `Set` 语句中赋值对一个新的对象的引用。
- 可以用 `GetObject` 函数，在 `Set` 语句中赋值对新的或已有的对象的引用。

用 New 关键字赋值对象引用

如果 ActiveX 部件提供一个类型库，就可以在变量声明或 Set 语句中，用 New 关键字创建一个新对象并将对象引用赋予对象变量。

如果用 New 关键字声明一个对象变量，在第一次使用该变量时，Visual Basic 会自动地创建一个新对象。关于更详细的信息，请参阅“声明对象变量”。

也可以在 Set 语句中，使用 New 关键字赋值对特定类的新对象的引用。例如，下列语句将对于新的 DAO 表对象的引用赋予变量 tdfOrders，将表的 Name 属性设置为“Orders”：

```
Dim tdfOrders As DAO.TableDef
Set tdfOrders = New DAO.TableDef
tdfOrders.Name = "Orders"
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》书的“Dim 语句”或“Set 语句”。

用 CreateObject 赋值对象引用

不管 ActiveX 是否提供一个类型库，可以在 Set 语句中使用 CreateObject 函数创建一个新对象，并将对象引用赋予一个对象变量。必须规定对象的编程标识符作为函数的参数，而且想要访问的对象必须是外部可创建的。

要用 CreateObject 赋值对象引用，请

- 对 `CreateObject` 使用下列语法。

```
Set objectvariable = CreateObject ("progID")
```

通常，`progID` 参数是正在创建对象的完全限定类名，例如 `Word.Document`。但是，`progID` 也可以与类名不同。例如，Microsoft Excel 对象的 `progID` 是“`Sheet`”而不是“`Worksheet`”。可以指定可选的 `servername` 参数来创建一个在跨网络的远程机器上的对象。它占用共享名称的 `Machine Name` 部分。例如，带有网络共享名称的 `\\MyServer\Public`，`servername` 参数将会是“`MyServer`”。

下列代码例子启动 Microsoft Excel（如果 Microsoft Excel 尚未运行）并创建变量 `xlApp` 来引用一个 `Application` 类的对象。参数“`Excel.Application`”充分限定 `Application` 为 Microsoft Excel 定义的类：

```
Dim xlApp As Excel.Application
```

```
Set xlApp = CreateObject("Excel.Application")
```

详细信息 请参阅 “`CreateObject` 函数”。

用 `GetObject` 赋值对象引用

虽然 `GetObject` 函数也能用于赋值对一个新对象的引用，但是它一般还是用于赋值对已经存在的对象的引用。

使用下列语法，赋值对现存的对象的引用。

```
Set objectvariable = GetObject([pathname] [, progID])
```

`pathname` 参数可能是一个现存文件的路径、一个空字符串，也可以完全省

略。如果被省略，则要求 progID。指定一个现存文件的路径，将引起 GetObject 使用存于文件中的信息创建一个对象。如第一个参数为空字符串，则 GetObject 的作用与 CreateObject 一样，将创建编程标识符是 progID 的类的一个新对象。下列表格描述了使用 GetObject 的结果。

如果 ActiveX 部件正在运行	结果
Set X = GetObject("MySrvr.Application")	X 引用一个现存的 Application 对象
Set X = GetObject("", "MySrvr.Object")	X 引用一个新的，外部可创建的对象
如果 ActiveX 部件没在运行	结果
Set X = GetObject("MySrvr.Object")	返回一个错误
Set X = GetObject("", "MySrvr.Object")	ActiveX 部件 (MySrvr) 被启动，X 引用一个新的对象。

例如，变量 wrdApp 引用一个正在运行的 Microsoft Word Application:

```
Dim wdApp As Word.Application
Set wdApp = GetObject("", "Word.Application")
```

同 CreateObject 一样，参数 "Word.Application" 是一个由 Microsoft word 定义的 Application 类的编程标志符。如果有多个 Microsoft Word 实例在运行，不能预先指出 wdApp 将引用那个实例。

重点 可以用 GetObject 赋值对复合文档文件中的对象引用。一个复合文档文件含有对多个对象类型的引用。例如，一个复合文档文件可以含有电子数据表，文本与位图。

如果电子数据表应用程序还没有运行，下列例子就启动该程序并打开文件

Revenue.xls:

```
Dim xlBook As Excel.Workbook
```

```
Set xlBook = GetObject("C:\Accounts\Revenue.xls")
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》一书的“GetObject 函数”。

加速对象引用

可以通过优化 Visual Basic 处理对象引用的方法，使得 Visual Basic 应用程序运行得更快。Visual Basic 处理对象引用的速度能受到下列因素影响：

- ActiveX 部件是作为进程内的服务器，还是作为进程外的服务器来实现的。
- 对象引用是事前绑定的还是后期绑定的。

一般说来，如果一个部件已作为可执行文件（.exe 文件）的一部分被实现，那么它是一个进程外的服务器，且在其自己的进程中运行。如果它作为一个动态链接库被实现，那么它是一个进程内的服务器，而且在和客户端应用程序同一进程中运行。

使用进程内的服务器的应用程序比使用进程外的服务器的应用程序，运行速度要快些，因为应用程序不需要跨越过程的边界去使用对象的属性、方法和事件。关于进程内的服务器与进程外的服务器的更详细的信息，请参阅“进程内的服务器与进程外的服务器”一节。

如果对象引用使用的对象变量被声明为一个特定类的变量，则对象引用是事前绑定。如果对象引用使用的对象变量被声明为一个一般的 `Object` 类的变量，则对象引用是后期绑定。通常，使用事前绑定变量的对象引用比使用后期绑定变量，运行得快些。

例如，可以将对于 `Excel` 对象的引用赋予下列变量中的任意一个：

```
Dim xlApp1 As Excel.Application
```

```
Set xlApp1 = New Excel.Application
```

```
Dim xlApp2 As Object
```

```
Set xlApp2 = CreateObject("Excel.Application")
```

使用变量 `xlApp1` 的代码是事前绑定，它比使用后期绑定变量 `xlApp2` 的代码执行的更快。

后期绑定

当声明一个变量 `As Object`，`Visual Basic` 在编译期间不能决定该变量含有的对象引用是什么性质的。在这种情况下，`Visual Basic` 必须使用后期绑定——即 `Visual Basic` 必须在运行期间才能决定，对象是否实际上具有代码中要使用的属性和方法。

例如，即便 `Visual Basic` 引用并不存在的方法，它也会不出错误地编译下列代码，这是因为它使用了后期绑定对象变量。直至运行时之前，它并不检查方法的存在性，所以将产生运行时错误。

```
Dim xlApp As Object  
Set xlApp = CreateObject("Excel.Application")  
xlApp.TheImpossibleMethod' Method doesn't exist.
```

这个代码比使用事前连接对象变量的代码运行得要慢些，因为 Visual Basic 在编译好的可执行代码中，还必须含有一段代码，它在运行时决定 Microsoft Excel Application 对象是否具有 TheImpossibleMethod 方法。

虽然后期绑定是调用对象的属性和方法的一种最慢的方法，但有时它又是必要的。例如，有时可能需要写这样一个函数，它用一个对象变量作用于若干不同类型对象的任一个。因为，在事先并不知道什么类的对象会赋给这个变量，所以要使用 As Object 声明它是后期绑定的变量。

事前绑定

如果 Visual Basic 在编译期间可以检测属性或方法属于什么对象，那么引用对象的问题在编译期间就能解决。编译好的可执行的代码仅包含引用对象的属性、方法和事件的代码。这被称为事前绑定。

当使用定义对象的类声明对象变量时，该变量可仅包含对那个类的对象的引用。对使用该变量的任何代码，Visual Basic 都可使用事前绑定。

事前绑定极大地减少了设置或恢复属性值所需的时间，因为调用的开销会占据总时间的大部分。对于方法的调用，这种改善的程度取决于所调用的方法的工作量。短的方法，因为调用的开销可以与完成该任务所需的时间相比拟，所以收效最大。

使用对象的属性、方法和事件

将对象引用赋予对象变量后，即可用此变量操作对象的属性和方法。也可用 `WithEvents` 关键字声明对象变量，并用它使应用程序响应该对象的事件。

使用对象的属性和方法

可以用 *object.property* 语法设置并返回对象的属性值，或用 *object.method* 语法使用该对象的方法。例如，可以用下面的代码设置 `Application` 对象的 `Caption` 属性：

```
Dim xlApp As Excel.Application
Set xlApp = New Excel.Application
xlApp.Caption = "MyFirstObject"
```

注意 在 Microsoft Excel 97 以前的版本不支持引用 Microsoft Excel Application 类的 `Excel Application` 语法。要在 Microsoft Excel 5.0 和 Microsoft Excel 95 之中引用 Microsoft Excel Application 类，要用 `[_ExcelApplication]` 语法。例如：

```
Set xlApp = New [_ExcelApplication]
```

可以这样调用 Microsoft Excel Application 对象的 `Quit` 方法：

```
xlApp.Quit
```

通常，引用被其它应用程序或工程定义的对象的方法或属性时，尽可能使之具体是一个好办法。例如：

'用充分限定的属性名设置 Microsoft Project 窗口标题。

```
Dim pjWindow As Project.Window
```

'获得对于第一个 Window 对象的引用。

```
Set pjWindow = ActiveProject.Windows(1)
```

```
pjWindow.Caption = "Project Caption"
```

'没有限定的名字，使 Visual Basic 用'名为 Caption 的属性找到的第一个对象，

'在这种情况下就是 Form1。

```
Caption = "Microsoft Form1 Caption"
```

注意 如果需要在 Visual Basic 应用程序中输入二进制数据，并计划在使用 ActiveX 的应用程序之间共享该数据，请使用 Byte 数组存储该数据。如果将二进制数据赋予一个字符串，然后想将这些数据传给使用字符串的 Automation 对象，数据可能不会正确地转换。关于数据类型的更详细的信息，请参阅第五章“编程基础”。

详细信息 关于使用对象属性和方法的更详细的信息，请参阅第九章“用对象编程”。

响应对象的事件

除了响应 Visual Basic 对象发生的事件，应用程序还可以响应由 ActiveX

部件提供的对象中的事件。例如，如果一个事件发生在 Microsoft Excel 工作簿中，Visual Basic 应用程序能显示一个消息框。

通过在对象的事件过程中添加代码，使得应用程序可以响应该对象的事件。但是，由部件提供的对象的事件过程在 Visual Basic 中不是自动可用的。必须首先使用 WithEvents 关键字声明一个对象变量。

在使用 WithEvents 声明一个对象变量之后，Visual Basic 代码窗口使用该变量来显示该对象的事件过程。然后可以把代码加到这些事件过程中以响应对象的事件。当把一个对象引用赋予这个变量时，就创建了运行时在变量和对象之间的连接。

要为部件提供的对象创建事件过程，请按照以下步骤执行：

1. 在 Visual Basic 工程中添加对于部件的类型库的引用。关于添加对类型库的引用更详细的信息，请参阅“创建对象引用”一节。
2. 在窗体或类模块的 Declarations 部分中，使用 WithEvents 关键字声明对象变量。例如：

```
Dim WithEvents xlBook As Excel.Workbook
```

Visual Basic 把对象变量名加到代码窗口的“对象”框。当选择变量名时，Visual Basic 在“过程”列表框中显示对象的事件过程。

3. 选择一个事件过程，然后将代码加入到当事件发生时，应用程序所要运行的过程。

例如，假设 Visual Basic 应用程序依赖于 Microsoft Excel workbook 中显示的数据，而且已经为此 workbook 声明了 WithEvents 变量 xlBook。当其他用户试图关闭 workbook 时，您可以通过在应用程序中

把下列代码加到 `xlBook_BeforeClose` 事件过程来显示一个消息，并使该 `workbook` 不被关闭。

```
Private Sub xlBook_BeforeClose(Cancel As Boolean)
    ' 隐藏 Microsoft Excel 窗口，使得该消息可见。
    xlBook.Application.Visible = False
    ' 显示消息。
    MsgBox "This workbook must remain open."
    ' 不再隐藏 Microsoft Excel 窗口。
    xlBook.Application.Visible=True
    ' 将设计过程的 Cancel 参数设置为 True，取消该事件。
    Cancel = True
End Sub
```

4. 把对象引用赋予 `WithEvents` 对象变量。

例如，可以把下面的内容加到 Visual Basic 窗体的 `Form_Load` 事件过程中，给变量 `xlBook` 赋值对 Microsoft Excel workbook, Sales.xls 的引用：

```
Private Sub Form_Load()
    Set xlBook = GetObject("Sales.xls")
    ' 显示 Microsoft Excel 和 Worksheet 窗口。
    xlBook.Application.Visible = True
    xlBook.Windows(1).Visible = True
End Sub
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》一书的“Dim 语句”。

释放 ActiveX 部件

使用完对象之后，要清除所有引用这个对象的变量，使对象从内存中释放。要清除对象变量，将其设置为 **Nothing**。例如：

```
Dim acApp As Access.Application
Set acApp = New Access.Application
MsgBox acApp.SysCmd(acSysCmdAccessVer)
Set acApp = Nothing
```

所有的对象变量，当它们越界时，就会被自动清除。如果想让变量跨越过程保持其值，就要使用公共变量或窗体级变量，或者创建返回该对象的过程。下列代码说明如何使用公共变量：

```
Public wdApp as Word.Application
.
.
.
' 创建 Word 对象并启动 Microsoft Word。
Set wdApp = New Word.Application
.
.
```


·
' 在应用程序结束或该引用设置为 `Nothing` 之前，

' Microsoft Word 不关闭：

```
Set wdApp = Nothing
```

另外，当所有对象（包括从属对象）引用结束后，一定要将其设为 `Nothing`。

例如：

```
Dim xlApp As Excel.Application
```

```
Dim xlBook As Excel.Workbook
```

```
Set xlApp = New Excel.Application
```

```
Set xlBook = xlApp.Workbooks.Add
```

```
Set xlApp = Nothing '注意！xlBook 仍会包含对象引用。
```

```
Set xlBook = Nothing '现在全部引用被清除。
```

定位对象模型

一旦理解了如何使用部件提供的对象，就可以使用部件显露的任何对象了。部件可用的范围涵盖了从简单代码部件、ActiveX 控件到大型部件，例如，Microsoft Excel 和 Microsoft Data Access Object (DAO) 编程界面就显露了许多对象。

每一个对象都存在于部件的对象层次中的某处，可以用两种方法访问这些对象：

- 如果对象是外部可创建的，可以直接访问。
- 如果对象是从属对象，则间接访问。可以从部件的层次中较高层的另一对象中得到对它的引用。

定位对象层次的最佳方法是使用“对象浏览器”（如果部件提供一个对象库）。

定位对象层次

如所见到的那样，通过外部可创建对象设置对从属对象的引用，可以向下定位一个对象层次。还可以对集合对象使用一种方法来返回单独的对象。关于详细信息，请参阅“使用外部可创建对象和从属对象”。

图 10.3 显示在 Microsoft Excel 应用程序中对象定位路径。

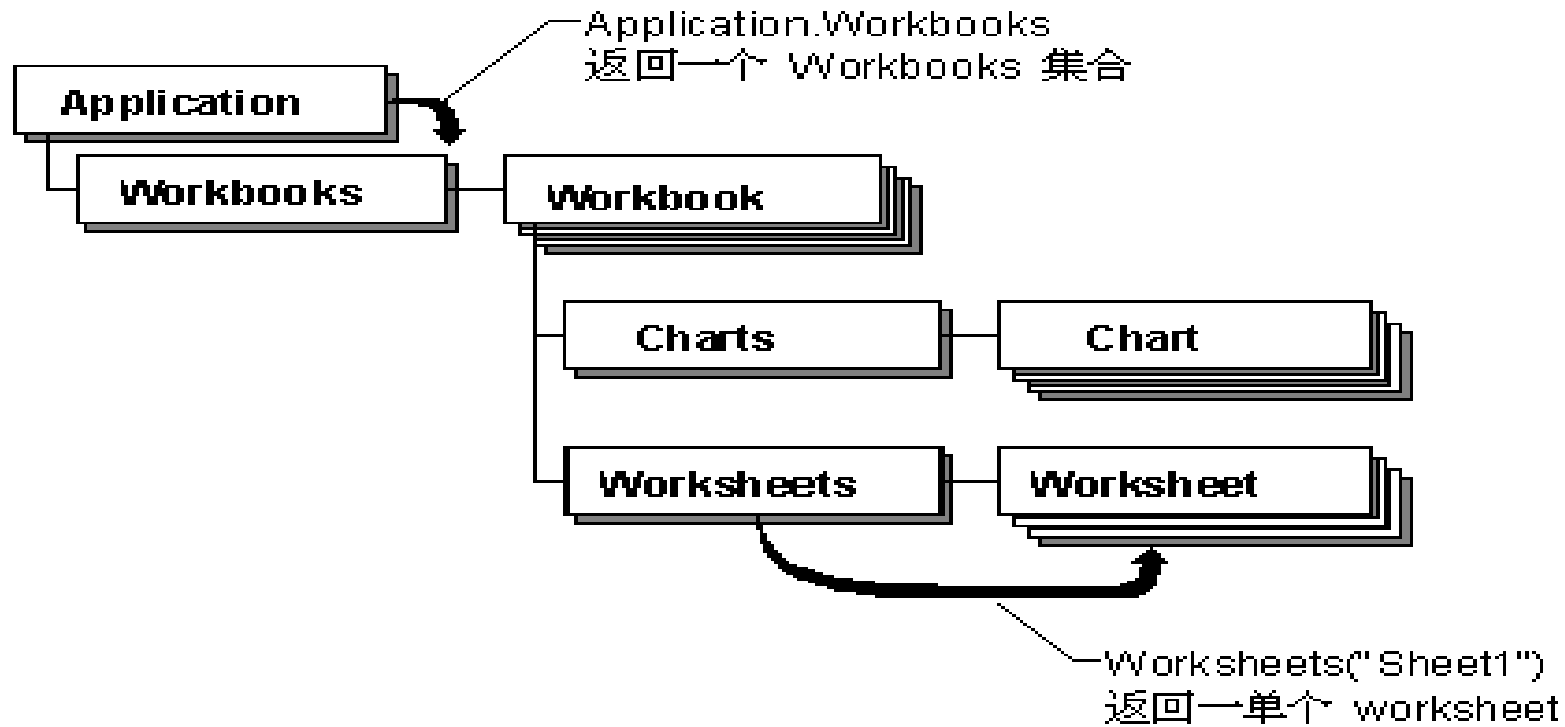


图 10.3 使用集合向下定位 Microsoft Excel 对象层次

要向上定位，大多数应用程序使用 Parent 和 Application，如图 10.4 所示。

集合对象

集合对象是其它对象组的容器。这些对象提供了一个简易的方法来跟踪相同类型的一组对象。例如，可使用 Menus 集合对象来访问一个应用程序中全部 Menu 对象的集合。可以使用下列代码来引用当前装入 Microsoft Excel 中的全部工作簿：

Application.Workbooks

注意 Workbooks 是复数。因为集合对象的标准命名约定是：组成集合的对象类型的复数。使用 For Each 语句可以定位集合中的所有对象，如下所示：

```
Dim xlBook As Excel.Workbook
```

```
.
```

```
.
```

```
.
```

```
For Each xlBook In Application.Workbooks
```

```
    '显示每个工作簿名。
```

```
    MsgBox xlBook.FullName
```

```
Next xlBook
```

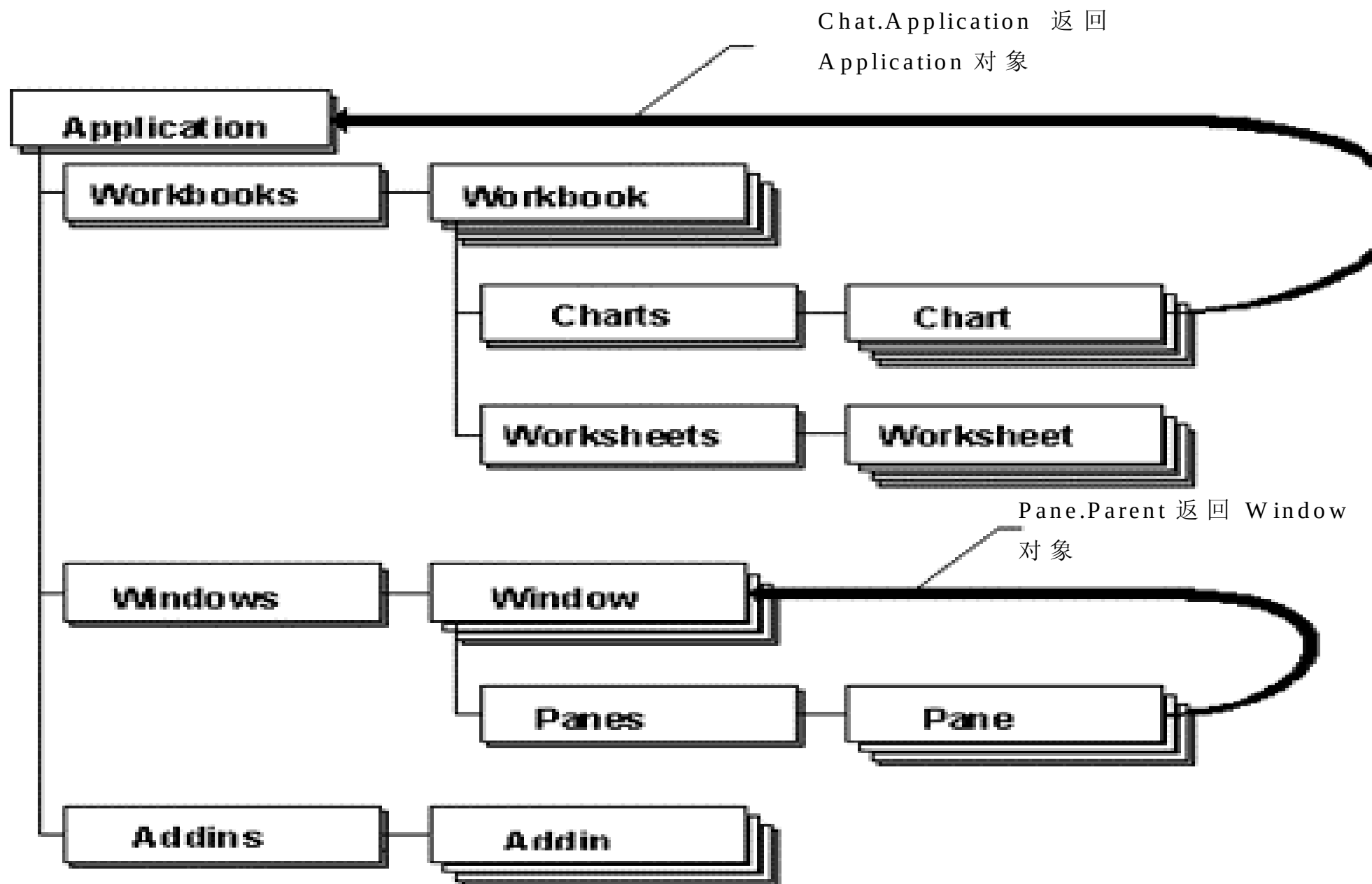


图 10.4 使用 Parent 和 Application 属性向上定位 Microsoft Excel 对象层次

还可以通过名称或者通过它们在集合中的索引顺序引用在许多集合中的单独对象。下面的例子说明如何引用名为 “Normal”、“Example” 和 “Heading” 的 Style 对象：

```
xlBook.Styles("Normal")
```

```
xlBook.Styles("Example")
```

```
xlBook.Styles("Heading")
```

假设这些对象是 Styles 中的前三个对象，并且集合是基于零的，那么，就可以用下面的方法引用它们：

```
xlBook.Styles(1)      ' 引用 Normal Style 对象。
```

```
xlBook.Styles(2)      ' 引用 Example Style 对象。
```

```
xlBook.Styles(3)      ' 引用 Heading Style 对象。
```

详细信息 关于操作集合对象的更详细的信息，请参阅第九章“用对象编程”。

使用外部可创建对象和从属对象

如何创建由部件提供的对象引用，取决于对象是一个外部可创建对象还是从属对象。可以直接创建对外部可创建对象的引用；使用部件对象层次中较高层对象的方法，可间接创建对从属对象的引用。

外部可创建对象

大多数大型 ActiveX-enabled 应用程序和其它 ActiveX 部件，在它们的对象层次中提供了一个顶层外部可创建对象：

- 提供了对该层次中其它对象的访问。
- 提供对整个应用程序起作用的方法和属性。

例如，每个 Microsoft Office 应用程序提供一个顶层 Application 对象。下面的例子显示如何赋值对于 Microsoft Excel、Microsoft Word 和 Microsoft Access 的 Application 对象的引用：

```
Dim xlApp As Excel.Application
Dim wdApp As Word.Application
Dim acApp As Access.Application
Set xlApp = New Excel.Application
Set wdApp = New Word.Application
Set acApp = New Access.Application
```

然后，可以用这些变量来访问在每个应用程序中的从属对象、以及这些对象的属性和方法。更详细的信息，请参阅本章前部的“创建对对象的引用”。

注意 早期 Microsoft Excel 97 的版本不支持引用 Microsoft Excel Application 类的语法 Excel.Application。必须用 [_ExcelApplication] 语法来引用 Microsoft Excel 5.0 和 Microsoft Excel 95 中的 Microsoft Excel Application 类。例如：

```
Set xlApp = New [_ExcelApplication]
```

除了这些顶层外部可创建对象外，ActiveX 部件也可以提供部件对象层次的较低层次上的外部可创建对象。这些对象可以被访问。如果作为外部可创建对象，则直接访问；如果作为较高层的外部可创建对象的从属对象，则间接访问。例如，可以直接或间接地创建对 DAO TableDef 对象的引用：

' 创建对 daoTable1 的直接引用。

```
Dim daoTable1 As DAO.TableDef
Set daoTable1 = New DAO.TableDef
daoTable1.Name = "Table1"
```

' 创建对 daoTable2 的间接引用，
' 作为 DAO DBEngine 对象的从属对象。

```
Dim daoDBE As DAO.DBEngine
Dim daoWs As DAO.Workspace
Dim daoDb As DAO.Database
Dim daoTable2 As DAO.TableDef

Set daoDBE = DAO.DBEngine
Set daoWs = daoDBE.Workspaces(0)
Set daoDb = daoWs.CreateDatabase("db1.mdb", _
    dbLangGeneral)
Set daoTable2 = daoDb.CreateTableDef("Table2")
```

有些对象提供一个 Application 对象，但是，使用另外的名字。例如，Microsoft

Access 中的 Microsoft Jet 数据库引擎把其顶层对象称为 DBEngine 对象。

从属对象

仅能用一种方法获得对从属对象的引用，即通过使用一个外部可创建对象的属性或方法返回对从属对象的引用。从属对象位于对象层次中的较低层，并且仅可通过使用外部可创建对象的方法被访问。例如，假设要从 Microsoft Excel 引用 Button 对象。就不可以使用下列代码（否则会产生错误的结果）：

```
Dim xlButton As Excel.Button  
Set xlButton = New Excel.Button
```

而是使用下列代码，来获得对 Button 对象的引用：

```
Dim xlApp As Excel.Application  
Dim xlBook As Excel.Workbook  
Dim xlSheet As Excel.Worksheet  
Dim xlButton As Excel.Button  
  
Set xlApp = New Excel.Application  
Set xlBook = xlApp.Workbooks.Add  
Set xlSheet = xlBook.Worksheets.Add  
Set xlButton = xlSheet.Buttons.Add(44, 100, 100, 44)
```

'现在可以使用 Button 对象属性。

```
xlButton.Caption = "FirstButton"
```

图 10.5 说明了 Visual Basic 应用程序如何获得对 Button 对象的引用。

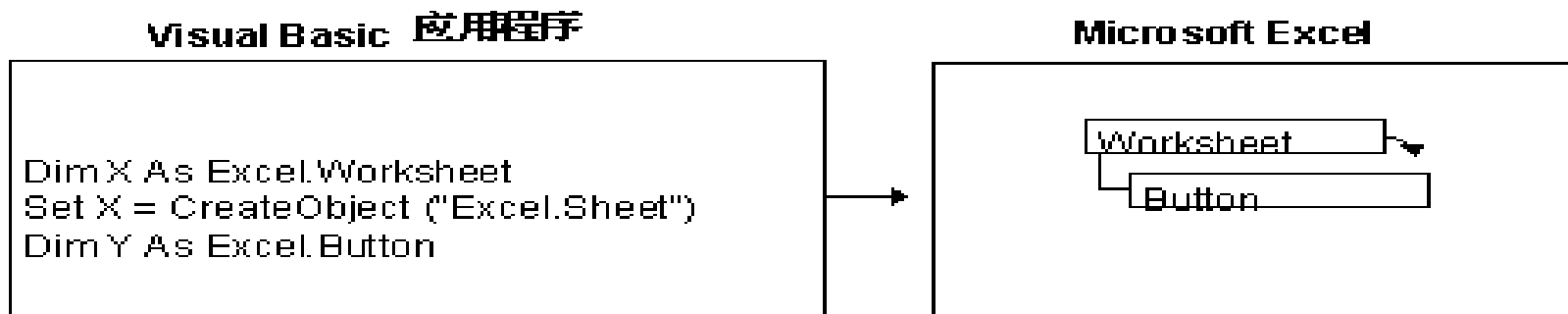


图 10.5 访问从属对象

处理 ActiveX 部件中的运行时错误

当使用 ActiveX 部件时，处理错误的代码尤为重要，因为来自部件的代码用于 Visual Basic 应用程序的内部。在一切可能的地方，应当包含处理部件可能产生错误的代码。例如，检查用户无意中关闭一个部件应用程序而引起的错误，是一个好习惯：

```
Function StartWord()
```

'启动 Microsoft Word。

On Error Goto ErrorTrap

'声明 Microsoft Word Application 变量
'和为错误陷阱使用的整型变量。

```
Dim wdApp As Word.Application
```

```
Dim iTries As Integer
```

'赋值对象引用。

```
Set wdApp = New Word.Application
```

'释放对象变量。

```
Set wdApp = Nothing
```

```
Exit Function
```

ErrorTrap:

'因 Microsoft Word 不能启动而产生的错误的陷阱。

```
Select Case Err.Number
```

```
    Case 440                                ' Automation 错误
```

```
        iTries = iTries + 1
```

'做五次重新启动 Word 的尝试。

```
If iTries < 5 Then
```

```
        Set wdApp = New Word.Application
        Resume
    Else
        Err.Raise Number:=VbObjectError + 28765, _
            Description:= "Couldn't restart Word"
    End If
Case Else
    Err.Raise Number:= Err.Number
End Select
End Function
```

如果在前面的例子中，有 **error 440** 以外的错误发生，过程显示该错误并产生一个新错误。提供对象的应用程序可能把其自身的错误传回去。有些情况下，应用程序可能使用与 **Visual Basic** 用于另一错误的同样的错误代码。在这些情况下，应在可能产生错误的每行后使用 **On Error Resume Next** 并立即检查错误。这种错误检查称作在线错误处理。

测试对象引用

在代码中使用对象变量之前，可能想确认变量是否拥有一个有效的对象引用。可以使用 **Is Nothing** 来确定对象引用是否已经赋给了变量。例如，下列代码检查对象引用是否已经赋给了变量 **wdDoc**：

```
If wdDoc Is Nothing Then MsgBox "No object reference."
```

但是，Is Nothing 不能检测有效对象引用是否已变成无效。例如，如果将 Microsoft Word 对象引用赋予对象变量，而 Microsoft Word 变成无效，该变量将仍保持有效的对象引用。在这种情况下，使用错误处理程序捕获当代码要使用该变量时产生的错误。

详细信息 关于特定应用程序可能送回的错误的信息，请参阅应用程序的文档。关于捕获错误的详细内容，请参阅第十三章“调试代码和处理错误”。

处理挂起 ActiveX 部件的请求

设置一个属性或调用一个对象方法通常只要零点几秒。但是这些请求常常不会立即完成。请看以下几种情况：

- 当用户有一个对话框打开时，如果调用 Microsoft Excel 中 Workbook 的 Close 方法，Microsoft Excel 打出表示忙，不能执行请求的信号。这将导致部件忙状态。
- 如果调用一个执行长操作的方法，例如当数据库很活跃时的大量数据库工作，可能在第一个操作仍为挂起时试图执行另一个操作。这将导致请求挂起 (request pending) 状态。
- 如果有两个或更多程序同时调用一个共享部件，那么一个调用必须在另一调用开始前被完成。部件用请求序列化 (serializing) 方法来处理这样的矛盾，也就是让它们排队等候。这也会导致请求挂起状态。

部件忙就如同打电话时的占线信号（忙音）。知道打不通，您会挂机稍等之

后再试。

请求挂起就象电话打通了，但是去找接电话的人让您等待的时间比您预计的要长得多。如果把您的请求序列化，那么请求挂起就如同另一方提起了话机但马上说“请等候，别挂机。”

部件忙的情况

部件可以拒绝您的请求因为有一个模态对话框打开着，或者因为用户的编辑操作在进行中。例如，当电子数据表单元在编辑时，Microsoft Excel 拒绝客户端应用程序的请求。

Visual Basic 假设忙态是暂时的，并以一个规定的超时间隔不停地发出请求。当时间到，Visual Basic 显示“部件忙”对话框，如图 10.6 所示。

用户可以请求再试、请求取消或转换部件并解决问题（例如，取消对话框）。如果用户选择“取消”，则在发出请求的过程中发生错误 `&h80010001 (RPC_E_CALL_REJECTED)`。



图 10.6 “部件忙”对话框

请求挂起状态

一旦部件接受了应用程序的请求，应用程序必须等到请求被完成。如果请求要用较长的时间，用户可试验用最小化程序或重新调整其大小的办法来解决。

在一个短的超时间隔之后，Visual Basic 显示“部件请求挂起”对话以响应这一试验。

“部件请求挂起”对话框的外观和“部件忙”对话框稍有不同。“取消”按钮被取消了，如图 10.7 所示，因为在进行中的请求不能被取消。

仅该部件暂停，显示作为请求结果的错误消息时，转换到该部件才是有用的。这并不经常发生，因为部件的正常行为是给调用部件的程序返回一个

错误状态。



图 10.7 “部件请求挂起”对话框

详细信息 请参阅本章后面的“改变部件忙或请求挂起消息”、“控制超时间隔”和“产生部件忙超时错误”。

改变部件忙或请求挂起消息

“部件忙”对话框和“部件请求挂起”对话框作为简单的缺省消息由 Visual Basic 提供。在许多情况下这些对话框可能不满足需要。如：

- 程序可以调用一个由无用户界面的部件提供的对象的方法。例如，用 Visual Basic 专业或企业版创建的部件可以在无任何可见窗体的背景下运行。
- 调用的部件可以用企业版的 Visual Basic 的远程自动化 (Remote Automation) 功能创建，并且可以在远离用户的另一个计算机上运行。

- 如果程序已使用 `GetObject` 函数装入 Microsoft Excel 的工作簿，当用户转换至 Microsoft Excel 时，工作簿是看不见的。实际上，Microsoft Excel 本身是不可见的，在这种情况下，“切换到”按钮什么也不做。

在这些情况下，“切换到”按钮是不适合的，可能引起程序用户的混乱。这时可以为两个或其中的一个超时指定代用消息。消息会显示在一个没有“切换到”按钮的简单消息框中。

对请求挂起状态，消息框仅有一个“确定”按钮。对部件忙状态，提供“确定”按钮和“取消”按钮。如果用户按“取消”，在提出请求的过程中会出现错误 `error-2147418111 (&h80010001)`。

下列 `App` 对象的属性决定了“部件忙”或“部件请求挂起”对话框是否被消息框所取代，以及是否允许指定消息框的文本与标题。

OLEServerBusyMsgText 属性

该属性规定消息文本在部件忙状态发生时被显示。设置这个属性使得常用“部件忙”对话框的地方使用了替换的消息框。

OLEServerBusyMsgTitle 属性

如果在部件忙状态下提供了一个替换消息框，请确定要用的标题（仅设置这个属性不会引起替换的消息框被使用）。

OLERequestPendingMsgText 属性

当请求挂起状态发生时，确定要显示的消息文本。设置这个属性使得替换的消息框用在常用“部件请求挂起”对话框的地方。

OLERequestPendingMsgTitle 属性

如果为请求挂起状态提供替换的消息，请确定要使用的标题（仅设置这个属性不会使替换的消息框被使用）。

下面的例子给部件忙和请求挂起两种状态设置标题和消息文本，完全替换“部件忙”或“部件请求挂起”对话框。

```
Public Const APP_TITLE = "Demo Application"
```

```
Private Sub cmdLongTransaction_Click()
```

```
    On Error Goto LongTransaction_Error
```

```
    ' 可能希望在 Sub Main 中设置标题一次。
```

```
    App.OLEServerBusyMsgTitle = APP_TITLE
```

```
    App.OLERequestPendingMsgTitle = APP_TITLE
```

```
    '针对这次请求的消息文本。
```

```
    App.OLEServerBusyMsgText = "The component for _
```

```
        the " & "Long Transaction has not responded. _
```

```
        If " & "you have been waiting more than five " _  
        & "minutes, you may wish to cancel this " _  
        & "request and try it later." & vbCrLf _  
        & "Call Network Services to verify that the " _  
        & "component is running, or to report problems."  
App.OLERequestPendingMsgText = "Your request " _  
    & "is still executing. " & vbCrLf _  
    & "Call Network Services to verify that the " _  
    & " component is running, or to report _  
    problems."
```

'提出请求并使用结果的代码

' ...

LongTransaction_Cleanup:

'完成所有必要清理的代码 ...

' ...

Exit Sub

LongTransaction_Error:

If Err.Number = &h80010001 Then

MsgBox "Transaction cancelled"

Else

'处理其他错误的代码。

```
End If
Resume LongTransaction_Cleanup
End Sub
```

重点 消息的长度可能受操作系统的限制。当目标操作系统是 Windows NT 或 Windows 95 时，可使用长于一千字符的消息。

控制超时间隔

可以使用 App 对象的两个属性设置，决定 Visual Basic 何时显示“部件忙”和“部件请求挂起”对话框的超时间隔。

OLEServerBusyTimeout 属性

决定在显示“部件忙”对话框之前，Visual Basic 反复尝试 Automation 请求的时间。缺省值是 10000 毫秒（10 秒）。

OLERequestPendingTimeout 属性

决定在显示“部件请求挂起”对话框期间，在响应鼠标单击、按键事件和其它事件之前，Visual Basic 要等待多久。缺省值是 5000 毫秒（5 秒）。下面例子演示，为调用假设的 BusinessRules 对象的 StockAnalysis 方法，如何调整和重新设置超时值。

```
Public Sub SetTimeouts(ByVal lngComponentBusy As _  
Long, ByVal lngRequestPending As Long)  
    App.OLEServerBusyTimeout = lngComponentBusy  
    App.OLERequestPendingTimeout = lngRequestPending  
End Sub
```

```
Public Sub ResetTimeouts()  
    App.OLEServerBusyTimeout = 10000  
  
    App.OLERequestPendingTimeout = 5000  
End Sub
```

```
Private Sub cmdFullAnalysis_Click()  
    On Error Goto FullAnalysis_Error  
    '设置很短的超时。2 秒钟之后用户将被提醒，  
    '并且按键或单击将显示“部件忙”和“部件请求挂起”对话框。  
    SetTimeouts 2, 2  
    Me.MousePointer = vbHourglass  
    gobjBusinessRules.StockAnalysis txtNYSECode.Text, _  
    ATYPE_FULL  
FullAnalysis_Cleanup:  
    Me.MousePointer = vbDefault
```

```
ResetTimeouts
```

```
Exit Sub
```

```
FullAnalysis_Error:
```

```
    If Err.Number = &h80010001 Then
```

```
        MsgBox "Analysis cancelled"
```

```
    Else
```

```
        '处理其它错误的代码...
```

```
    End If
```

```
    Resume FullAnalysis_Cleanup
```

```
End Sub
```

可以把这些超时中的任一个设置成很大的值，因为它们作为长整型数据 (Longs) 存储。例如，86,400,000 毫秒是一天，这等同于无穷超时。但是当这样做时，在部件不再忙或请求挂起完成之前，有把程序锁死的危险。

重点 由于这些超时值是 App 对象的属性，它们也对用 OLE 容器控件或工具箱链接或嵌入的文档起作用。如果正在使用链接的或嵌入的文档，并且为 Automation 请求改变了这些属性，那么完成以后应重新设置其值。

产生部件忙超时错误

对于部件忙状态，可以通过把 App 对象的布尔 OLEServerBusy RaiseError

属性设置成 `True`，来越过“部件忙”对话框和置换消息。`Visual Basic` 将在 `OLEServerBusyTimeout` 属性规定的时间长度期间再试请求，然后在提出 `Automation` 请求的过程中产生一个错误，同用户在“部件忙”对话框上按了“取消”按钮一样。

返回的错误是 `-2147418111 (&h80010001)`。然后可以在过程的错误处理器采用最合适的行动。例如，可以显示一个复杂对话框，提供用户几个重试的选择或比较方案。

对于使用了 `Visual Basic` 企业版的 `Remote Automation` 功能、并运行在远程网络计算机上的部件的设计，这个属性尤为有用。这样一个部件可以被另外的部件所调用，并且它必须处理在那些调用中的错误而不显示任何窗体。

对于请求挂起状态，没有对应的属性。一旦 `Automation` 请求被部件所接受，客户端程序必须等待直到请求完成。

使用部件的可视界面

如果部件支持对象链接和嵌入 (OLE)，使用部件的可视界面，可以不用任何代码就把对象链接或嵌入应用程序。使用下面两种方法中的一种来使用部件的可视界面：

- 在应用程序中添加 `OLE` 容器控件，然后在该控件中插入对象。
- 在工具箱中添加对象的类，然后就象在窗体内添加控件那样在应用程序

中添加那个类的对象。

用 OLE 容器控件插入对象

OLE 容器控件为使用对象的可视界面提供了最大的灵活性。使用 OLE 容器控件，可以：

- 在应用程序中为对象创建一个占位符。在运行时创建出现在 OLE 容器控件中的对象，或者改变已在设计时置于 OLE 容器控件中的对象。
- 在应用程序中创建链接对象。
- 把 OLE 容器控件与数据库绑定。
- 如果用户在 OLE 容器控件中移动、调整对象尺寸或更新对象，执行相应的操作。
- 从复制到剪贴板上的数据中创建对象。
- 把对象显示为图标。

在任何时刻，一个 OLE 容器控件内只能有一个对象。在 OLE 容器控件中创建链接或嵌入的对象有几种方法。选择哪一种，取决于是在设计时还是在运行时创建链接或嵌入的对象。一旦有 OLE 容器控件被放在窗体上，就可以用下面的步骤在容器控件中插入对象：

- 使用“插入对象”或“选择性粘贴”对话框。参阅“设计时用 OLE 容器控件插入对象”和“运行时用 OLE 容器控件创建对象”。
- 在“属性”窗口中设置 Class、SourceDoc 和 SourceItem 属性。参阅“运行时用 OLE 容器控件创建对象”。

- 调用 CreateEmbed 或 Create Link 方法。参阅“运行时用 OLE 容器控件创建对象”。

详细信息 关于使用 OLE 容器控件的更详细的信息，请参阅第七章“使用 Visual Basic 的标准控件”中的“使用 OLE 容器控件”和“控件的容器”。

通过在工具箱中添加对象的类来插入对象

与使用工具箱将一个 Visual Basic 的内部控件加入应用程序一样，可以使用工具箱来加入对象。首先，把对象的类加入工具箱，然后把对象加入窗体。

要把对象的类加入工具箱，请按照以下步骤执行：

1. 从“工程”菜单中，选择“部件”。
2. 在“部件”对话框中，单击“可插入对象”选项卡。
3. 选择想要加入工具箱的类，然后单击“确定”。Visual Basic 把那个类的按钮加入工具箱。

例如，要把“Microsoft Excel Worksheet”按钮加入工具箱，选择“Microsoft Excel Worksheet”。

一旦把对象类加入了工具箱，就可以把它放到窗体上来创建那个类的对象。例如，把“Microsoft Excel Worksheet”按钮加入工具箱之后，可以把它拖放到窗体上，在该窗体上创建工作表对象。

比较链接对象与嵌入对象

通过把数据链接或嵌入 Visual Basic 应用程序，用部件的可视界面包含来自另一个应用程序的数据。链接的和嵌入的对象之间的主要区别是：储存数据的地方不同。例如，与链接对象相关的数据是由创建它的应用程序管理，而存储在 OLE 容器控件之外。与嵌入对象相关的数据包含在 OLE 容器控件中，并可与 Visual Basic 应用程序一起存储。

当创建链接或嵌入的对象时，它包含提供该对象的应用程序的名称、数据（或在链接对象的情况下，对该数据的引用）和数据的图像。

注意 为了把一个对象放在 OLE 容器控件中，提供对象的部件必须已在系统注册过。若某应用程序提供用于工程中的对象，当安装它时，应当在系统上注册其对象库，使应用程序对象出现在“插入对象”对话框中。可以通过使用 Regedit.exe 搜寻系统注册簿来寻找对象，但是注意不要改变注册簿的内容。

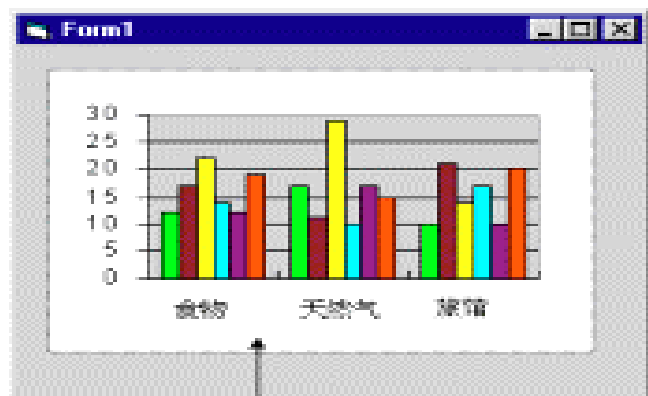
链接的对象

链接一个对象时，实际上是在应用程序中插入链接对象的占位符（并非实际数据本身）。例如，当把电子数据表单元的范围与 Visual Basic 应用程序链接时，与该单元相关的数据是存储在另一文件中的；在 OLE 容器控件中，只存储数据的链路和数据的图像。在使用 Visual Basic 应用程序时，用户可以激活链接对象（例如，双击对象），电子数据表应用程序便自动启

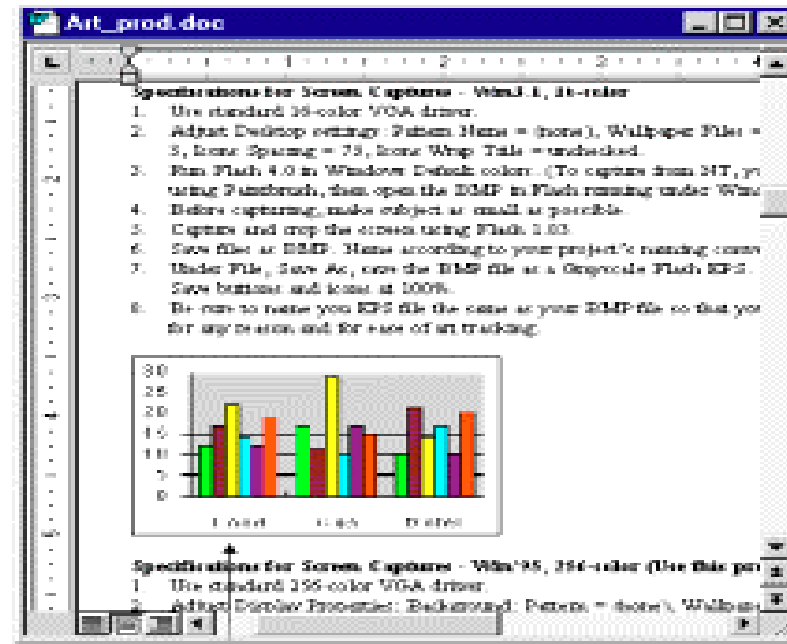
动。然后用户可以使用电子数据表应用程序来编辑那些电子数据表单元。当编辑一个链接对象时，编辑操作是在 OLE 容器控件外的单独窗口中完成的。

当对象与 Visual Basic 应用程序链接时，可以从其它的、包含那个数据链路的应用程序中查看对象的当前数据。数据仅存在于一个地方——ActiveX 部件——就是提供对象的源应用程序。例如，在图 10.8 中，Visual Basic 包含 Graph 应用程序的链路。Microsoft Word 也包含这个 Graph 应用程序的链路。如果图表的数据被随便哪个应用程序改变，改变了的图表将在 Visual Basic 应用程序和 Microsoft Word 文档中都出现。

Visual Basic 应用程序



Word for Windows



链接数据

MS Graph

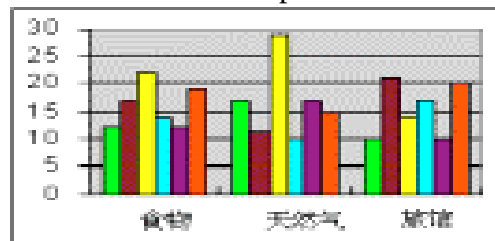


图 10.8 对象的数据可以从包含该数据链路的许多不同的应用程序中访问

正如所见的那样，链接使跟踪出现在多于一个的应用程序中的相同信息变得简单了。当要维护几个应用程序都可访问的一组数据时，链接是有效的手段。

段。

嵌入的对象

要创建一个嵌入对象，可以使用 OLE 容器控件，也可以把一个对象的类加入工具箱。使用嵌入的对象，与对象相关的全部数据都将被复制和纳入 OLE 容器控件。当把控件的内容存入文件时，该文件含有创建对象的应用程序名称、对象数据和对象的源文件图像。由于这个原因，嵌入对象可能会大大增加文件的大小。

与链接对象不同，其它应用程序不能访问嵌入对象中的数据。当想要应用程序保持在另一应用程序中创建和编辑的数据，嵌入是有效的手段，如图 10.9 所示。

当用户激活对象 (graph) 时，创建该对象 (Microsoft Graph) 的 ActiveX 部件被容器应用程序 (Visual Basic 应用程序) 所调用，该对象的数据被打开和编辑。另外，还在容器应用程序中显示对象的用户界面和菜单系统，使用户能现场控制对象。关于现场激活的详细信息，参阅本章后面的“激活 OLE 容器控件中的对象”。

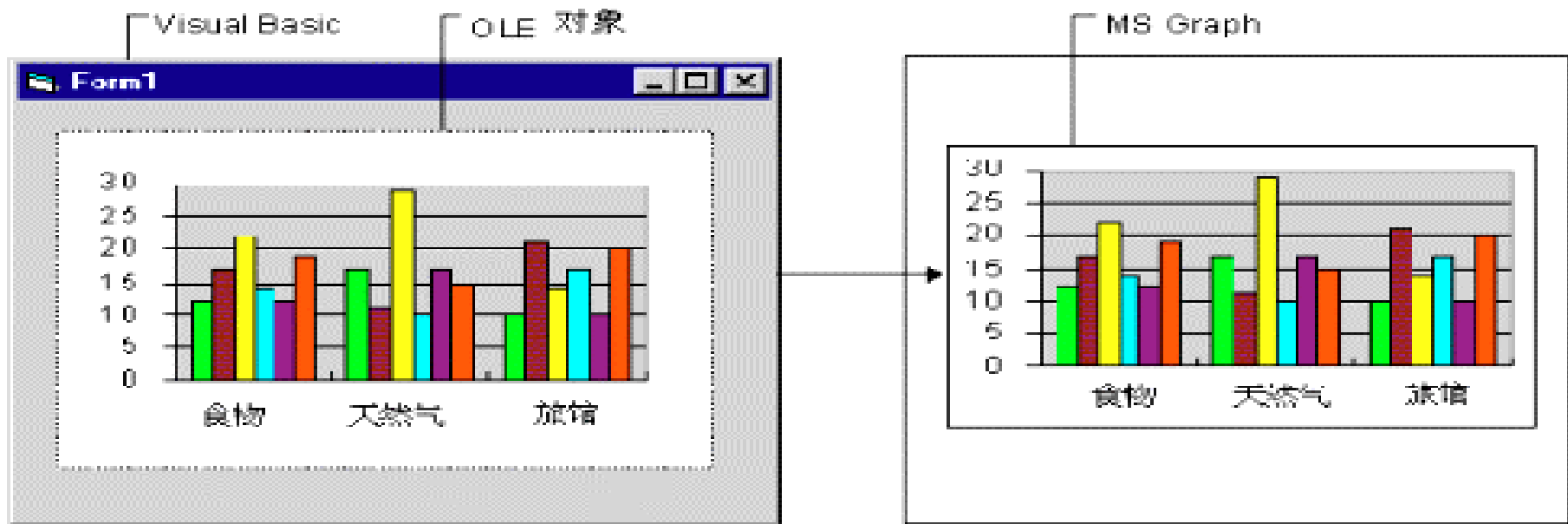


图 10.9 应用程序保持嵌入的对象的数据

设计时用 OLE 容器控件插入对象

每当在窗体上拖放一个 OLE 容器控件，Visual Basic 都显示“插入对象”对话框。如图 10.10 所示，在设计时使用这个对话框来插入链接或嵌入的对象。“插入对象”对话框中，出现一个能链接或嵌入应用程序的可用对象的清单。

当在设计中把对象插入 OLE 容器控件时，Class、SourceDoc 和 SourceItem 属性就已经自动设置好了。这些属性标识：提供该对象的应用程序、源文

件名和该文件内部链接的所有特定的数据。关于用于 OLE 容器控件的各种属性和方法的更详细的信息，参阅“运行时插入对象”。

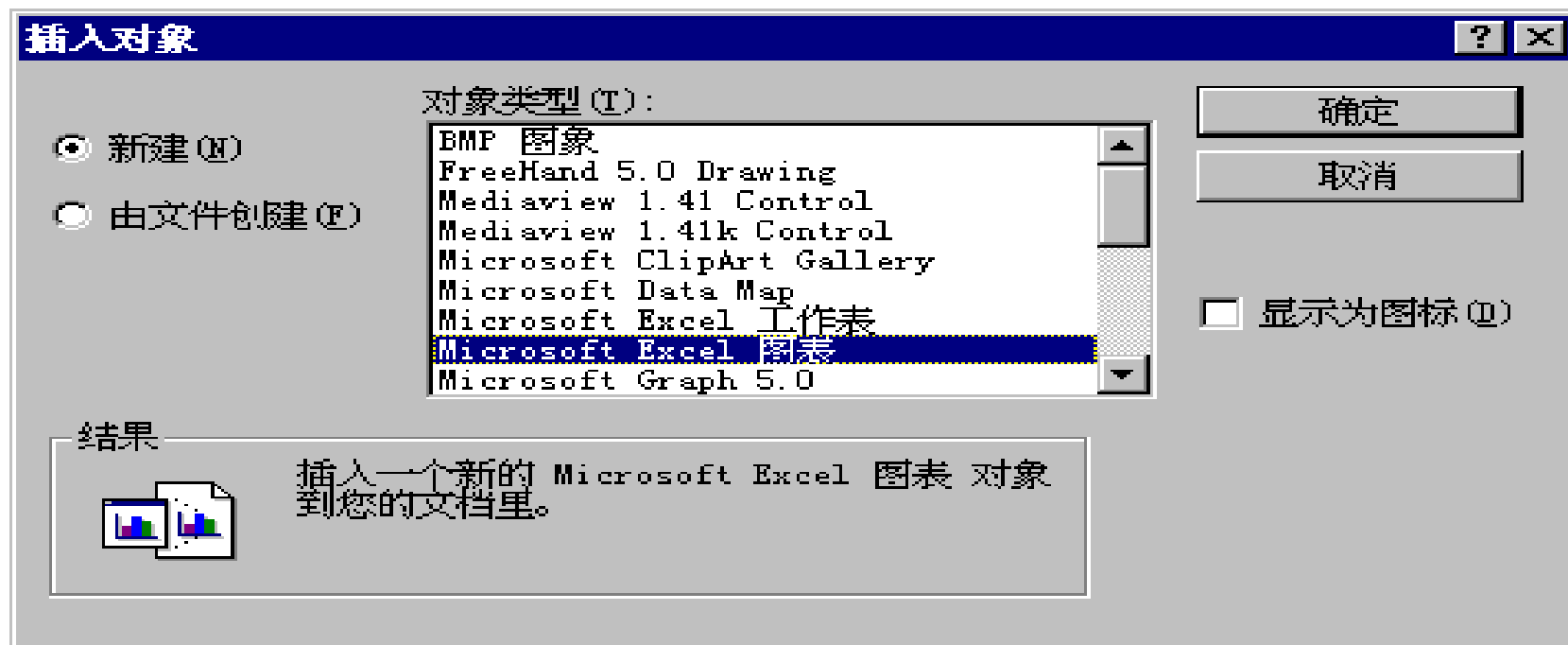


图 10.10 “插入对象”对话框

设计时插入链接对象

当插入一个链接对象时，显示在 OLE 容器控件中的数据只存在于一个地方——源文件中。从任何其它的、与对象的当前数据相链接的应用程序都可以查看那个数据。OLE 容器控件保持着对象的链接信息，例如提供该对

象的应用程序名、链接文件名和链接数据的图像。

要使用“插入对象”对话框来插入链接对象，请按照以下步骤执行：

1. 拖放“OLE 容器控件”到窗体上。

显示“插入对象”对话框。也可以在任何时候单击鼠标右键“OLE 容器控件”，然后选择“插入对象”命令来显示此对话框。

2. 选择“从文件中创建”选项按钮。

3. 选择“浏览”按钮。

显示“浏览”对话框。

4. 选择要链接的文件。

5. 单击“插入”以返回“插入对象”对话框。

6. 在“插入对象”对话框中选择“链接”复选框，然后选择“确定”创建链接对象。

使用链接对象时，每一个运行这个应用程序的用户都必须能访问（有效路径）该链接文件，并具有创建该文件的应用程序的复件。否则，当应用程序运行时，将显示原始数据的图像，但用户不能更改数据，也不能看到其他用户对链接数据进行的更改。如果此应用程序是在网上运行，就会出现问题了。

如果应用程序包含一个链接对象，当应用程序没运行时，对象的数据可能被另一个应用程序更改。下次应用程序运行时，对源文件的更改不会自动出现在 OLE 容器控件中。若要在 OLE 容器控件中显示当前数据，则使用控件的 Update 方法：

oleObj.Update

详细信息 请参阅“Update 方法（OLE 容器）”。

如果用户想保存对链接对象的更改，用户必须从 ActiveX 部件的菜单里保存它。OLE 容器控件的 SaveToFile 方法仅应用于嵌入对象。

设计时创建嵌入对象

当创建嵌入对象时，既可以从文件中嵌入数据，也可以创建一个新的、可被以后的数据填充的空对象。当从文件嵌入数据时，OLE 容器控件中显示的是指定文件的数据复件。当创建新对象时，调用创建对象的应用程序并可以把数据输入对象。

一般来说，在设计时创建显示已存在的数据的嵌入对象。这样可以查看对象的数据，同显示给用户的一样。然后，移动和调节 OLE 容器控件和窗体上其它控件的大小，创建自己的应用程序的用户界面。

要显示存在于嵌入对象中的数据，以现存的文件为模板来创建对象。OLE 容器控件将包含该文件中数据的图像。使用嵌入对象显示数据的应用程序，将比使用链接对象显示同样数据的应用程序大，因为有嵌入对象的应用程序实际上包括了源文件的数据。

要用现存文件创建嵌入对象，请按照以下步骤执行：

1. 在窗体上拖放“OLE 容器控件”。
自动显示“插入对象”对话框。
2. 选择“从文件中创建”选项按钮。
3. 选择“浏览”按钮。

显示“浏览”对话框。

4. 选择想要嵌入的文件。

5. 选择“插入”返回“插入对象”对话框。

6. 在“插入对象”对话框中，选择“确定”创建嵌入对象。

与链接对象中的数据不同，嵌入对象中的数据是不持久的。换句话说，如果让用户输入的更改在下次应用程序运行时出现，就必须使用 `SaveToFile` 方法来保存数据。关于把嵌入数据存入文件的详细内容，请参阅本章后面的“保存和恢复嵌入的数据”。

使用“选择性粘贴”对话框创建对象

在设计时创建对象的另一个方法是使用“选择性粘贴”对话框（如图 10.11 所示）。如果只想利用文件的一部分，例如，电子数据表中一定范围内的单元，或 Word 文档中的一段，那么这个对话框就很有用。

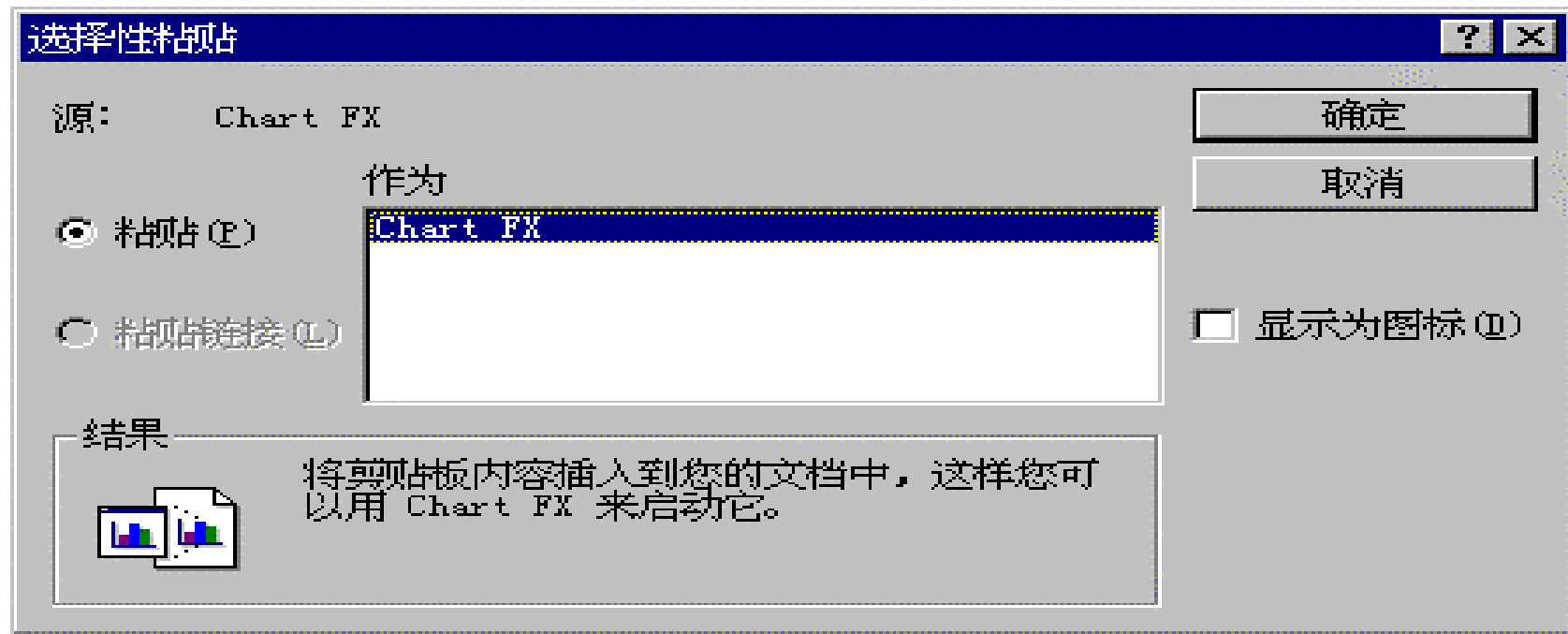


图 10.11 “选择性粘贴”对话框

要使用“选择性粘贴”对话框来创建对象，请按照以下步骤执行：

1. 运行包含要链接或嵌入的数据的应用程序。
2. 选择要链接或嵌入的数据。
3. 从 ActiveX 部件的“编辑”菜单中，选择“复制”。
数据被复制到剪贴板上。
4. 在 Visual Basic 中，用鼠标右键单击“OLE 容器控件”，并从弹出式菜单中选择“选择性粘贴”命令。
5. 若要创建嵌入对象，就选择“粘贴”选项按钮。

-或者-

若要创建链接对象就选择“链接粘贴”选项按钮。

如果控件中已经有了嵌入或链接的对象，就会有一个消息来询问是否要删除现有的对象，并在其位置中创建一新对象。

6. 要创建对象，选择“确定”。

运行时用 OLE 容器控件创建对象

要在运行时创建链接或嵌入对象，需要在代码中使用方法和属性。OLE 容器控件有各种用来操作链接或嵌入对象的属性和方法。关于应用于 OLE 容器控件的全部属性和方法的清单，请参阅“OLE 容器控件”。

使用 Object 属性

通过使用 OLE 容器控件的 Object 属性，可以使用链接或嵌入对象的属性和方法。Object 属性是运行时、只读属性，它保持对 OLE 容器控件中的对象的引用。使用该属性执行 OLE 容器控件的自动化任务，以及程序化操作对象支持的属性和方法：

```
strObjName = oleObj1.Object.Name
```

为使用此属性，OLE 容器控件必须包含可编程的对象。关于可编程对象的详细信息，请参阅“ActiveX 部件的类型”。

运行时创建链接对象

可以用 OLE 容器控件的 `CreateLink` 方法，在运行时从文件中创建一个链接对象。这种方法需要一个文件参数 `sourcedoc`，从这个文件创建对象；还需要一个可选参数 `sourceitem`，它规定了想要从源文件中链接的数据。运行时，下面代码创建链接对象：

```
oleObj1.CreateLink "C:\Excel\Test.xls"
```

注意 如果用“`CreateLink`”创建一个链接的对象，不必在“属性”窗口设置“类”、“`SourceDoc`”和“`SourceItem`”属性。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》一书的“`CreateLink` 方法”。

在运行时创建嵌入对象

要想在运行时，从文件中创建一个嵌入的对象，可以使用 `CreateEmbed` 方法。这种方法有两个参数，`sourcedoc` 和 `class`（如果指定 `SourceDoc`，`class` 是可选的）。`Sourcedoc` 决定对象的模板，而 `class` 决定对象的类型。当用 `CreateEmbed` 时，不需要设置 `SourceDoc` 和 `Class` 属性。

下列代码，用现存文件作为对象的模板创建嵌入的对象。

```
oleObj1.CreateEmbed "Q1profit.xls"
```

详细信息 请参阅“`CreateEmbed` 方法”。

当创建一个空的嵌入的对象时，最好激活将要为对象提供数据的 `ActiveX`

部件。此时，可以使用 **DoVerb** 方法。它允许用户在运行时将任何数据输入应用程序中。然后，用户选择 **ActiveX** 部件的 **Update** 命令（这种菜单命令将出现在部件的“文件”菜单上），显示这些最新进入 **OLE** 容器控件中的数据。

要在运行时创建一个空的嵌入对象，请按照以下步骤执行：

1. 不指定源文档，用 **CreateEmbed** 方法创建一个空的嵌入对象。例如，下面的代码在 **OLE** 容器控件中插入一个 **Microsoft Excel Worksheet** 的文件模板。

```
oleObj1.CreateEmbed "", "Excel.Sheet"
```

2. 使用 **DoVerb** 的方法。**DoVerb** 方法的缺省动词依应用程序而定。对 **Microsoft Excel**，缺省的动词是 **Edit**。

例如，下列代码创建一个空的嵌入对象，然后激活使用缺省 **DoVerb** 操作创建该对象的应用程序。

```
oleObj1.CreateEmbed "", "Excel.Sheet"
```

```
oleObj1.DoVerb -5 '激活
```

当创建一个包含有不同应用程序的多种信息的文档应用程序时，提供空的嵌入对象是有效的方法。关于更详细的信息，请参阅“运行时让用户规定对象”。

将数据库与 **OLE** 容器控件绑定

可以将 **OLE** 容器控件与存储在 **Microsoft Jet** 数据库引擎或 **Microsoft**

Access 数据库中的数据绑定。如果有一个雇员照片表的数据库，就可能要这么做。如果这些照片作为对象被存储，可以将它们与 OLE 容器控件绑定，并将它们作为用数据控件访问的每个记录显示在窗体上。为了将数据与这些数据库中的数据绑定，需要在 DataSource 属性中规定数据源（数据集名），并在 OLE 容器控件的 DataField 属性中规定来自数据源的字段的名称。当从数据库显示一个对象时，OLE 容器控件允许用户激活、编辑和更新对象。如同任何被绑定的控件一样，当记录位置被改变时，被更新的对象将自动写回数据库。

运行时让用户规定对象

通过运行时显示“选择性粘贴”和“插入对象”对话框，允许用户创建一系列对象。当创建以文档为中心的应用程序时就可以这样做。在这种应用程序中，用户结合来自不同应用程序的数据创建一个单一的文档。例如，这个应用程序可能是一个字处理器，在这个处理器中，用户可能输入一些正文，然后使用“插入对象”或“选择性粘贴”对话框嵌入电子数据表和图表。

用 OLE 容器控件的 InsertObjDlg 方法显示“插入对象”对话框，或用 PasteSpecial Dlg 方法显示“选择性粘贴”对话框。这两个对话框让用户决定什么对象进入 OLE 容器控件。

- “插入对象”对话框提供可用的对象清单，并根据用户的选择创建对象。
- “选择性粘贴”对话框允许用户从系统剪贴板中将对象粘贴到 OLE 容

器控件。

通过在事件——比如一个菜单的 Click 事件——中调用适当的方法，在运行时显示这些对话框：

```
Private Sub cmdInsert_Click ()
```

```
    '显示“插入对象”对话框。
```

```
    oleObj1.InsertObjDlg
```

```
    '检查是否用 OLEType 属性
```

```
    '创建一个对象。
```

```
    If oleObj1.OLEType = vbOLENone Then
```

```
        MsgBox "Object Not Created."
```

```
    End If
```

```
End Sub
```

```
Private Sub oleObj1_Click ()
```

```
    '决定包含在剪贴板中的数据能否被粘贴进 OLE 容器控件。
```

```
    If oleObj1.PasteOK Then
```

```
        '显示“选择性粘贴”对话框。
```

```
        oleObj1.PasteSpecialDlg
```

```
        '检查对象是否已被创建。
```

```
        If oleObj1.OLEType = vbOLENone Then
```

```
            MsgBox "Object Not Created."
```

```
        End If
```


End If

End Sub

一旦对话框被显示，就不再需要写更多的代码去创建对象。用户只要在对话框中作出选择，再选“确定”，便可创建对象。如果用户取消对话，将不能创建对象。

注意 在显示“插入对象”或“选择性粘贴”对话框之前，您可能想确定 OLEType 属性的值，看看 OLE 容器控件是否包含链接对象、嵌入对象或没有对象，正如前面的代码示例中演示的那样。

常数 vbOLENone 和其它固有的常数被列在“对象浏览器”的 Visual Basic (VB) 对象库中。

决定在 OLE 容器控件中如何显示对象

可以用 OLE 容器控件的 DisplayType 属性标识对象是否将作为图标出现（设置 DisplayType = 1），或者对象的数据是否将被显示在控件中（设置 DisplayType = 0）。这个属性还决定了在运行和设计时，当显示“插入对象”和“选择性粘贴”对话框时，“显示为图标”复选框的缺省设置。

注意 一旦 OLE 容器控件含有一个对象，则不能改变它的显示形式。然而，能够删除链接的或嵌入的对象、设置 DisplayType 属性并插入一个新对象。

如果该控件不是 UI（用户接口）正在使用的，就可以用 SizeMode 属性

决定对象的图标或数据的图像在 OLE 容器控件中是如何显示的。设置 0-Clip 或 3-Zoom，将剪切图像去适合控件，但是，这并不改变图像的实际尺寸（编辑时可能看不见图像的全部）。一个比控件小的对象在比控件小的区域内被编辑。一个比控件大的对象充满整个容器区域，而且如果对象比控件区还大就可能被剪切。另一方面，设置 SizeMode 为 2-AutoSize，将调整控件的大小以适合图像。

激活 OLE 容器控件中对象

在运行时，OLE 容器控件的 DoVerb 方法激活一个对象，此时，可以用 AppIsRunning 属性决定提供对象的应用程序是否激活和运行。可以设置 AppIsRunning 为 True，启动 ActiveX 部件，从而使对象更快地激活。也可以设置这个属性为 False，在对象失去焦点时关闭应用程序或采取其它适当的行动。

现场激活

有些嵌入的对象可以从 OLE 容器控件内部被编辑（激活）。这称为现场激活，因为用户可以双击应用程序中的对象并与提供对象的应用程序交互，而不用切换到其它的应用程序或窗口。

对于支持现场激活的对象，可以设置 AutoActive 属性，那么用户在任何时刻都能激活对象。也就是说，当 OLE 容器控件的 AutoActive 属性设置为

Double-Click 时，用户可以用双击来激活控件。要注意，如果该对象的应用程序还没运行，激活对象将引发对象的应用程序。

注意 在运行中，想要在用户单击 OLE 容器控件时显示 ActiveX 部件的菜单，必须为窗体至少定义一个菜单项并把它的 Visible 属性设置成 False。如果不想显示任何菜单，这个菜单可以是不可见的。关于当对象在运行时被激活时，在容器应用程序中显示 ActiveX 部件的菜单和工具栏的详细信息，请参阅第六章“创建用户界面”。

响应移动容器或调整容器大小

OLE 容器控件具有 ObjectMove 事件，当移动与 OLE 容器控件相关的对象以及调整其大小时，会触发它。ObjectMove 的参数代表对象容器内对象的坐标（不包括边框）。如果对象移出窗体，参数的值代表与窗体左上角的相对位置。这些值可正可负。如果 ActiveX 部件的宽或高被改变，将通知 OLE 容器控件。

ObjectMove 事件，是 OLE 容器控件确定对象是否已经移动或调整大小的唯一方法。当用户移动包含在 OLE 容器控件中对象或调整其大小时，ObjectMove 事件发生。例如：

```
Private Sub oleObj1_ObjectMove(Left As Single, Top As _  
    Single, Width As Single, Height As Single)  
    '此法重新调整 OLE 容器控件的大小为新对象的大小。  
    oleObj1.Move oleObj1.Left, oleObj1.Top, _
```

```
        Width, Height
'此法移动 OLE 容器控件到新对象的位置。
oleObj1.Move Left, Top, _
        oleObj1.Width, oleObj1.Height
' Repaints the form.
    Me.Refresh
End Sub
```

保存和检索嵌入的数据

与嵌入的对象相关的数据不是持久的，即当含有 OLE 容器控件的窗体被关闭时，与该控件相关的数据的任何变化将丢失。要将更改的数据从对象保存至文件中，需用 OLE 容器控件的 `SaveToFile` 方法。数据被保存在文件后，可以在需要的时候打开文件并恢复对象。

如果一个对象被链接 (`OLEType = 0-Linked`)，实际上只有链接的信息和数据图像被保存在指定的文件里。对象的数据由创建该对象的应用程序来维护。如果用户想把数据改变保存到链接的文件，用户必须从 `ActiveX` 部件的“文件”菜单选择“保存”指令，因为 `SaveToFile` 方法只应用于嵌入的对象。

如果对象被嵌入 (`OLEType = 1-Embedded`)，对象的数据由 OLE 容器控件维护，并能用 `Visual Basic` 应用程序保存。

OLE 容器控件中的对象只能保存于打开的二进制文件。

要将数据从对象保存到文件中，请按照以下步骤执行：

1. 打开二进制文件。
2. 使用 `SaveToFile` 方法。

`cmdSaveObject_Click` 事件过程说明这些步骤：

```
Private Sub cmdSaveObject_Click ()  
    Dim FileNum as Integer  
    '取文件号。  
    FileNum = FreeFile  
    '打开要保存的文件。  
    Open "TEST.OLE" For Binary As #FileNum  
    '保存文件。  
    oleObj1.SaveToFile FileNum  
    '关闭文件。  
    Close #FileNum  
End Sub
```

一旦对象被保存至文件中，它将能被打开并显示在 OLE 容器控件里。

注意 当用 `SaveToFile` 或 `ReadFromFile` 方法时，将文件定位在紧接对象之后。因此，如果保存多个对象到一个文件里，读取它们的顺序应与写它们的顺序相同。

要将数据从文件读入 OLE 容器控件，请按照以下步骤执行：

1. 打开二进制文件。
2. 使用对象的 `ReadFromFile` 方法。

cmdOpenObject_Click 事件过程说明了这些步骤：

```
Private Sub cmdOpenObject_Click ()  
    Dim FileNum as Integer  
    '取文件号。  
    FileNum = FreeFile  
    '打开文件。  
    Open "TEST.OLE" For Binary As #FileNum  
    '读文件。  
    oleObj1.ReadFromFile FileNum  
    '关闭二进制文件。  
    Close #FileNum  
End Sub
```

每当对象的内容被改变，就调用 Updated 事件。这个事件用于决定对象的数据在最后一次保存时是否被更改。为此，在 Updated 事件中设一个全局的变量，指示需要保存的对象。保存对象时，重新设置该变量。

第十一章 响应鼠标和键盘事件

Visual Basic 应用程序能够响应多种鼠标事件和键盘事件。例如，窗体、图片框与图像控件都能检测鼠标指针的位置，并可判定其左、右键是否已按下，还能响应鼠标按钮与 SHIFT、CTRL 或 ALT 键的各种组合。利用键盘事件可以编程响应多种键盘操作，也可以解释、处理 ASCII 字符。

此外，Visual Basic 应用程序还可同时支持事件驱动的拖放功能和 OLE 的拖放功能。可用 Drag 方法连同某些属性及事件来启用诸如拖放控件的操作。OLE 拖放使应用程序在 Windows 环境下进行数据交换时功能大增，无需编写代码就可将其中大多数技术用于应用程序。

还可利用鼠标或键盘管理长时间的后台任务处理，这就使用户得以向其它应用程序进行切换或中断后台处理。

本章未讨论涉及鼠标或键盘的其它操作和事件（例如 Click 事件、DblClick 事件、Focus 事件和 Scroll 事件）。有关 Click 与 DblClick 事件的详细信息，请参阅第三章“窗体、控件和菜单”中“单击按钮执行操作”和“焦点概述”两主题及“语言参考”的“Click 事件”和“DblClick 事件”。同时请参阅联机“Scroll 事件”。

目 录

- 响应鼠标事件
- 检测鼠标按钮
- 检测 SHIFT、CTRL 及 ALT 键的状态
- 拖放
- OLE 拖放
- 自定义鼠标指针
- 响应键盘事件
- 中断后台处理

示例应用程序：Mouse.vbp

在这一章，许多代码示例都来自 Mouse.vbp 示例应用程序。该示例应用程序列在 Samples!Alink("vbsamples") 目录中。

响应鼠标事件

可通过 MouseDown、MouseUp、MouseMove 事件使应用程序对鼠标位置及状态的变化作出响应（其中不包括拖动事件；本章后面的“拖放”一节中将介绍拖动事件）。大多数控件能够识别这些鼠标事件。

事件	描述
MouseDown	按下任意鼠标按钮时发生
MouseUp	释放任意鼠标按钮时发生
MouseMove	每当鼠标指针移动到屏幕新位置时发生

当鼠标指针位于无控件的窗体上方时，窗体将识别鼠标事件。当鼠标指针在控件上方时，控件将识别鼠标事件。

如果按下鼠标按钮不放，则对象将继续识别所有鼠标事件，直到用户释放按钮。即使此时指针已移离对象，情况也是如此。

三种鼠标事件使用下列参数。

参数	描述
<i>Button</i>	一个位域参数，其中最不明显的三位描述鼠标按钮的状态。
<i>Shift</i>	一个位域参数，其中最不明显的三位描述 SHIFT、CTRL 与 ALT 键的状态。
<i>x, y</i>	鼠标指针的位置，这里用到了接受鼠标事件的对象坐标系统描述的鼠标指针位置。

位域参数返回单个位的信息，每个位的信息都指出某状态是打开的还是关闭的。使用二进制的记法时，称最左侧三位为最明显位，而称最右侧三位为最不明显位。本章后面的“检测鼠标按钮”、“检测 SHIFT、CTRL 和 ALT 键的状态”两节介绍了使用这些参数进行程序设计的方法。

AMouseDown 事件

`MouseDown` 是三种鼠标事件中最常使用的事件。例如，在运行时可用它调整控件在窗体上的位置，也可用它实现某些图形效果。按下鼠标按钮时就可触发此事件。

注意 鼠标事件被用来识别和响应各种鼠标状态，并把这些状态看作独立的事件，不应将鼠标与 `Click` 事件和 `DbClick` 事件混为一谈。在按下鼠标按钮并释放时，`Click` 事件只能把此过程识别为一个单一的操作——单击操作。鼠标事件不同于 `Click` 事件和 `DbClick` 事件之处还在于，鼠标事件能够区分各鼠标按钮与 `SHIFT`、`CTRL`、`ALT` 键。

结合 `Move` 方法使用 `MouseDown` 事件

将 `MouseDown` 事件与 `Move` 方法联合起来使用就可将命令按钮移动到窗体的不同位置。鼠标指针的位置决定按钮的新位置：在单击窗体的任意位置（除控件所在处外）时控件将移动到光标位置。

单一过程 `Form_MouseDown` 将执行此操作：

```
Private Sub Form_MouseDown (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    Command1.Move X, Y  
End Sub
```

`Move` 方法将命令按钮控件的左上角放置在由 `x` 和 `y` 参数指出的鼠标指针位置。可修改此过程，以便将控件的中心放置在鼠标位置：

```
Private Sub Form_MouseDown (Button As Integer, _
```

```
        Shift As Integer, X As Single, Y As Single)
    Command1.Move (X - Command1.Width / 2), _
        (Y - Command1.Height / 2)
End Sub
```

结合 Line 方法使用 MouseDown 事件

在先前的绘制位置与鼠标指针的新位置之间画一条直线，由此，Click-A-Line 示例应用程序就对鼠标单击作出响应。这个应用程序中使用了 MouseDown 事件和 Line 方法。使用以下语法，Line 方法将绘制一条从上次绘制点到点 (x2, y2) 的直线：

```
Line - (x2, y2)
```

Click-A-Line 使用一个空窗体及一个过程 — Form_MouseDown：

```
Private Sub Form_MouseDown (Button As Integer, _
    Shift As Integer, X As Single, Y As Single)
    Line -(X, Y)
End Sub
```

第一条直线始于缺省起点，也就是左上角。因此无论何时，只要按下鼠标，应用程序就会绘制一条从先前直线到鼠标目前位置的直线。如图 11.1 所示，程序运行的结果是一系列连接的直线。

详细信息 请参阅《Visual Basic 6.0 手册》的“MouseDown 事件”。

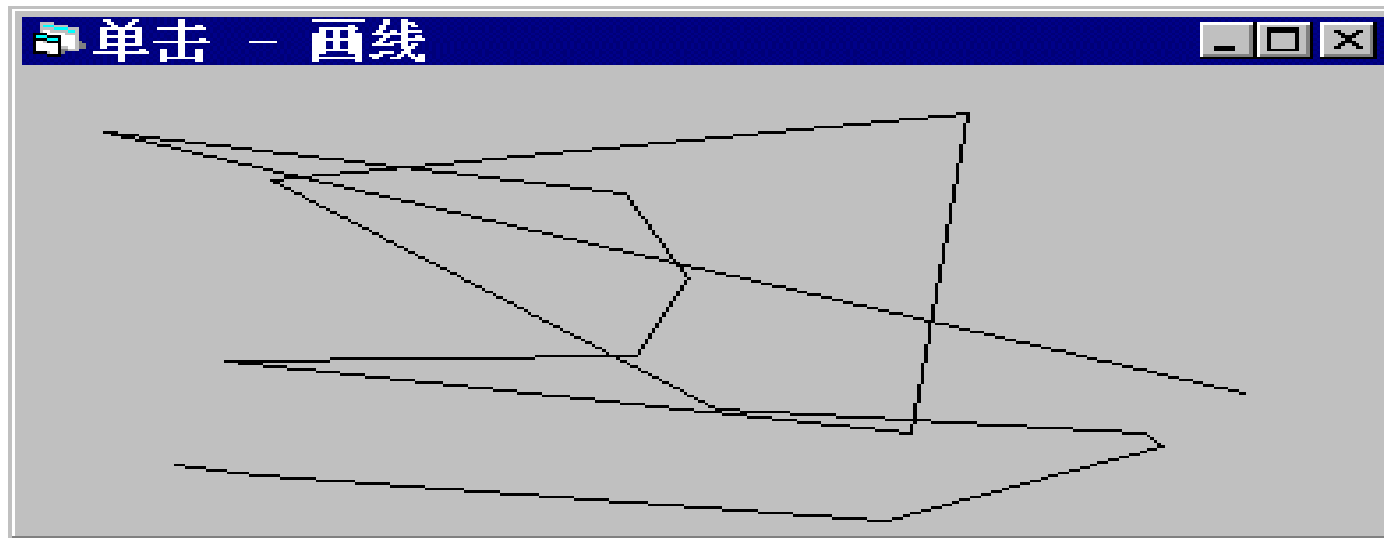


图 11.1 调用 `MouseDown` 后即绘制互相连接的直线

MouseMove 事件

当鼠标指针在屏幕上移动时就会发生 `MouseMove` 事件。当鼠标指针处在窗体和控件的边框内时，窗体和控件均能识别 `MouseMove` 事件。

结合 `Line` 方法使用 `MouseMove` 事件

使用图形方法在 `MouseMove` 过程中的效果与在 `MouseDown` 过程中的效果截然不同。例如，在本章前面的“`MouseDown` 事件”主题中，`Line` 方

法绘制连接的直线段，而在下面的 Scribble 应用程序中，在 Form_MouseMove 过程中使用相同方法绘制的是曲线而不是相互连接的线段。

在 Scribble 应用程序中，每当鼠标指针位置变更时都要识别 MouseMove 事件。以下代码在当前位置和先前位置之间绘制一条线。

```
Private Sub Form_MouseMove (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    Line -(X, Y)  
End Sub
```

与MouseDown过程相同，MouseMove过程创建的直线也从左上角开始，如图 11.2 所示。

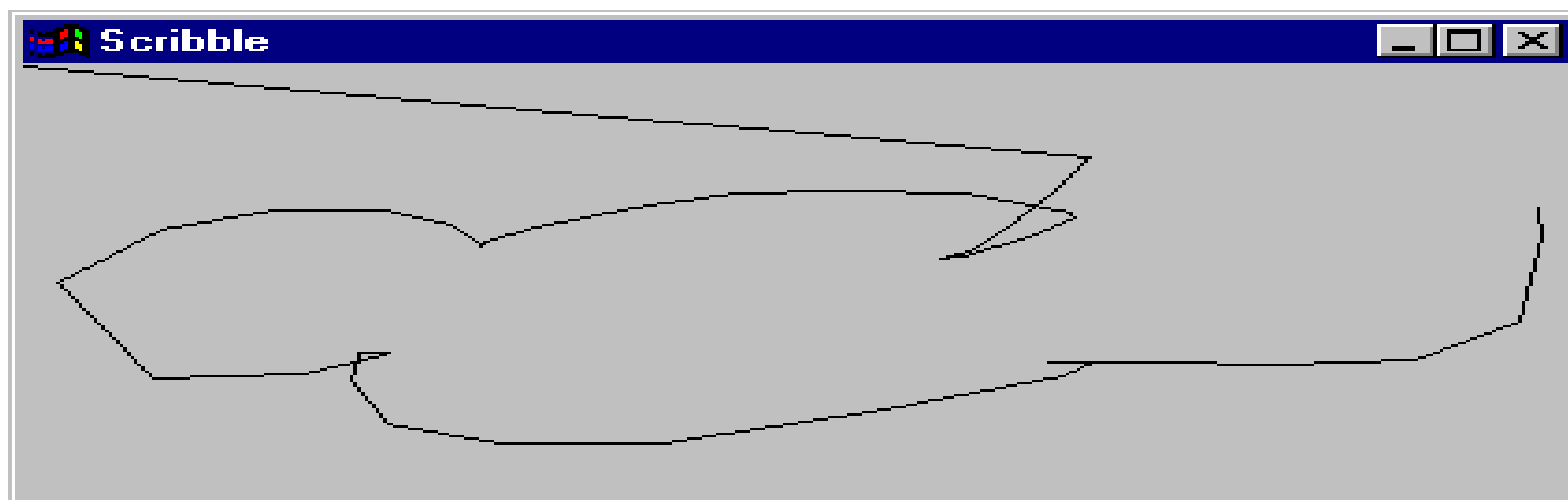


图 11.2 MouseMove 事件与 Line 方法创建的简单绘图程序

MouseMove 如何工作

当指针移过屏幕时要调用多少次 MouseMove 事件？或者换个说法，指针由屏幕顶端移动到底端时将经过多少个位置？

并不是对鼠标经过的每个像素，Visual Basic 都会生成 MouseMove 事件。操作环境每秒生成有限多个鼠标消息。为了看到实际上有多少次识别 MouseMove 事件，可用下述代码增强 Scribble 应用程序，从而在每次识别 MouseMove 事件之处，应用程序都绘制一个小圆圈。运行结果如图 11.3 所示。

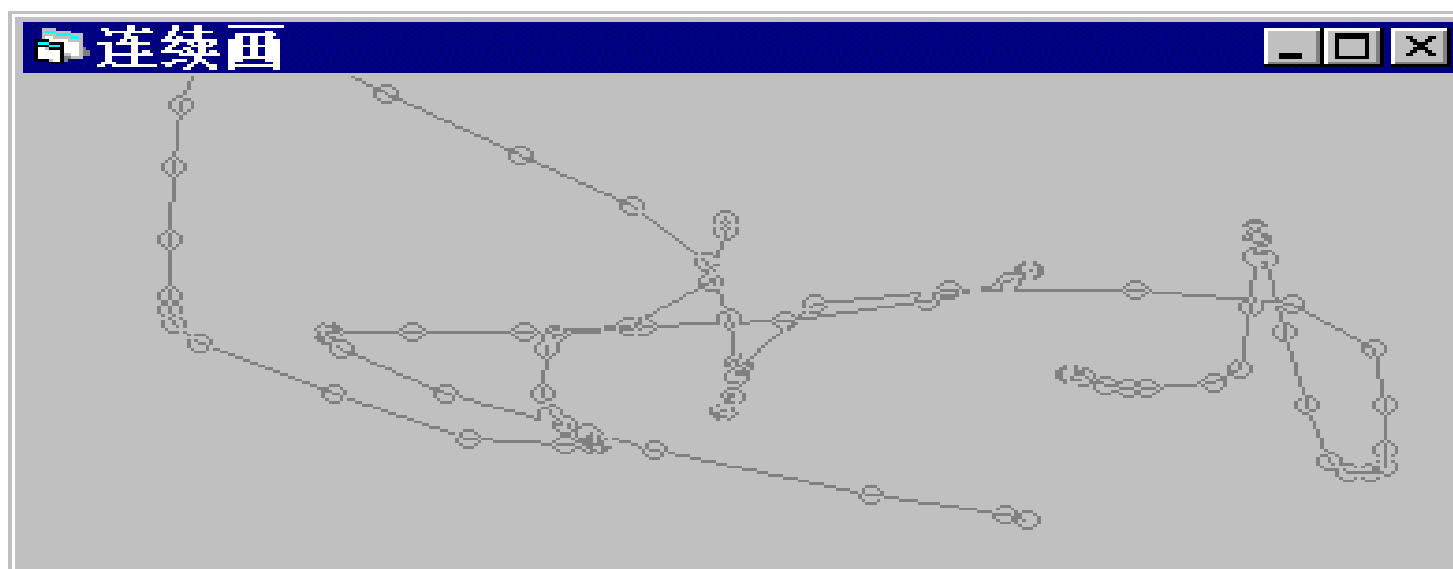


图 11.3 MouseMove 事件发生位置的演示

```
Private Sub Form_MouseMove (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    Line -(X, Y)  
    Circle (X, Y), 50  
End Sub
```

注意，指针移动的越快，在任何两点之间所能识别的 `MouseMove` 事件越少。众多圆圈挤在一起表明鼠标在此位置移动缓慢。

应用程序能接二连三迅速识别大量 `MouseMove` 事件。因此，一个 `MouseMove` 事件过程不应去做那些需要大量计算时间的工作。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`MouseMove` 事件”。

MouseDown 事件

释放鼠标按钮时，`MouseDown` 事件将会发生。`MouseDown` 与 `MouseDown` 和 `MouseMove` 事件搭配使用，往往相得益彰。下例说明如何将三个事件联合使用。

如果 `Scribble` 程序能够做到，只有在按住鼠标按钮时才绘图，而在释放按钮时停止绘图，则这个程序将会更有用。为此，应用程序必须响应三种操作：

- 按下鼠标按钮 (`MouseDown`)。
- 移动鼠标指针 (`MouseMove`)。

- 释放鼠标按钮 (MouseUp)。

MouseDown 与 MouseUp 将通知应用程序开始绘图与停止绘图。通过创建的一个代表绘图状态的窗体级变量就可指定这一点。请在窗体代码模块的声名部分输入如下代码：

```
Dim DrawNow As Boolean
```

DrawNow 代表两种状态值：True 的意思是“绘制直线”，False 的意思是“停止绘制”。

因为变量的缺省初始值为 0 (False)，所以应用程序启动时的绘图状态是关闭的。通过设置窗体级变量 DrawNow 的值，MouseDown 和 MouseUp 过程的第一行代码将响应打开和关闭绘图状态：

```
Private Sub Form_MouseDown (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    DrawNow = True  
    CurrentX = X  
    CurrentY = Y  
End Sub
```

```
Private Sub Form_MouseUp (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    DrawNow = False  
End Sub
```

只有当 DrawNow 为 True 时，MouseMove 过程才绘制直线，否则不执行任

何操作：

```
Private Sub Form_MouseMove (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If DrawNow Then Line -(X, Y)  
End Sub
```

每次按鼠标按钮时都会执行 `MouseDown` 事件过程并打开绘图状态。于是，按住鼠标按钮并在屏幕上拖动指针时将不断重复执行 `MouseMove` 事件过程。

注意，`Line` 方法忽略了第一个端点，这使 `Visual Basic` 从鼠标指针坐标的当前位置开始绘图。在缺省状态下，绘图坐标是相对于上一次的绘图点的；在 `Form_MouseDown` 过程中重新设置了窗体的 `CurrentX` 与 `CurrentY` 属性。

详细信息 请参阅《`Visual Basic 6.0` 语言参考手册》的“`MouseUp` 事件”。

检测鼠标按钮

编写代码，使之对鼠标事件作出不同的响应，这取决于用到哪个鼠标按钮或者是否按下了 `SHIFT`、`CTRL`、`ALT` 键，这样一来应用程序的功能将更强大。为提供这些选择，请使用 `MouseDown`、`MouseUp` 和 `MouseMove` 事件过程及 `button`、`shift` 参数。本章后面的“检测 `SHIFT`、`CTRL` 和 `ALT` 键的状态”介绍了使用 `shift` 参数的方法。

MouseDown、MouseUp 和 MouseMove 事件用 button 参数判断按下的是哪个鼠标按钮或哪些鼠标按钮。button 参数是位域参数——一个值，其每位代表一个状态或条件。这些值被表示成整数。如图 11.4 所示，三个最不明显位（最低位）分别表示鼠标的左按钮、右按钮和中按钮。每一位的缺省值为 0 (False)。如果未按下任何按钮，则三位的二进制值为 000。如果按下左按钮，则二进制值（或模式）变为 001。左按钮的位值由 0 (False) 变为 1 (True)。

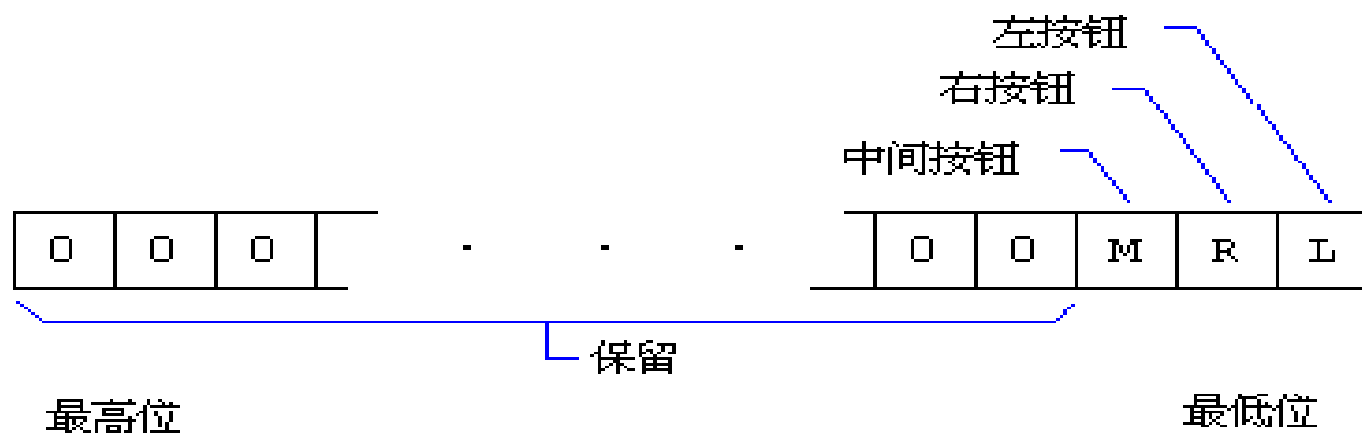


图 11.4 如何用位表示鼠标的状态

button 参数用十进制数值或常数表示这些二进制模式。下表列出了位的二进制值、相应的十进制值和 Visual Basic 常数：

二进制值	十进制值	常数	意义
001	1	VbLeftButton	按下左按钮
010	2	VbRightButton	按下右按钮
100	4	VbMiddleButton	按下中间按钮

注意 Visual Basic 提供常数来表示 button 和 shift 参数的二进制值。这些常数可与相应的十进制数值互换使用。但是，并非所有值都有对应常数。直接将十进制值相加就可导出某些按钮值或按钮与/或 shift 键组合的值。

将十进制数值 4 分配给鼠标中间按钮。同时按下左、右按钮就会产生单个数值 3 (1+2)。对于三按钮鼠标，同时按下三个按钮将产生十进制数值 7 (4+2+1)。下表中列出从可能的按钮组合中导出的其它按钮值：

二进制值	十进制值	常数	意义
000	0		未按下任何按钮
011	3	vbLeftButton + vbRightButton	按下左、右按钮
101	5	vbLeftButton + vbMiddleButton	按下左按钮和中间按钮
110	6	vbRightButton + vbMiddleButton	按下右按钮和中间按钮
111	7	vbRightButton + vbMiddleButton + vbLeftButton	按下三个按钮

结合MouseDown和MouseUp事件使用Button参数

用button参数与MouseDown可判断按下了哪个按钮，而用它与MouseUp就可判断释放了哪个按钮。因为对每个事件只设置一位，所以无法检测是否同时按下了两个以上按钮。换句话说，MouseDown和MouseUp一次只能识别一个按钮。

注意 与此成对照的是，可用MouseMove事件检测是否同时按下了两个以上按钮。MouseMove还能检测是否按下某个特定的按钮，而不管是否同时还有其它按钮被按下。关于详细信息，请参阅本章后面的“结合MouseMove使用Button参数”。

可用简单代码指明哪个按钮触发了MouseDown事件或MouseUp事件。以下过程对button是否等于1、2或4进行检测：

```
Private Sub Form_MouseDown (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button = 1 Then Print "You pressed _  
        the left button."  
    If Button = 2 Then Print "You pressed _  
        the right button."  
    If Button = 4 Then Print "You pressed _  
        the middle button."  
End Sub
```

如果按下多个按钮，Visual Basic 就会将操作解释为两个或多个独立的 `MouseDown` 事件。它为按下的第一个按钮设置位并在窗体上显示此按钮的信息，然后对下一个按钮完成相同的工作。同样，Visual Basic 也将两个或多个按钮释放操作解释为独立的 `MouseUp` 事件。

当释放已被按下的按钮时，以下过程将显示一条信息：

```
Private Sub Form_MouseUp(Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button = 1 Then Print "You released _  
        the left button."  
    If Button = 2 Then Print "You released _  
        the right button."  
    If Button = 4 Then Print "You released _  
        the middle button."  
End Sub
```

结合 MouseMove 使用 Button 参数

对于 `MouseMove` 事件，`button` 指出了鼠标按钮的全部状态——不象在 `MouseDown` 和 `MouseUp` 事件中那样仅仅指出哪个按钮触发事件。可以设置所有位或某些位，或者不设置任何位，所以还能够提供附加信息。这与 `MouseDown` 和 `MouseUp` 过程中一个事件只能设置一位形成对照。

检测单个按钮

如果检测 `MouseMove` 是否等于 001（十进制 1），则意味着检测在移动鼠标时是否只按住左按钮。如果还有其它按钮与左按钮一同被按住，则下列代码不显示任何信息：

```
Private Sub Form_MouseMove (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button = 1 Then Print "You're pressing _  
        only the left button."  
End Sub
```

为了检测是否按下某个特定按钮，应使用 `And` 操作符。下列代码显示每个被按下的按钮的信息，而不管是否还按下其它按钮：

```
Private Sub Form_MouseMove (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button And 1 Then Print "You're pressing _  
        the left button."  
    If Button And 2 Then Print "You're pressing _  
        the right button."  
End Sub
```

同时按下两个按钮后，在窗体上将显示两条信息。`MouseMove` 事件能够识别多个按钮状态。

检测多个按钮

在大多数情况下，为了将已按下的按钮隔离开，应使用 `MouseMove` 事件。在上一个程序的基础上可用 `If...Then...Else` 语句判断左、右按钮中的哪一个（或者全部）被按下。以下示例检测三种按钮状态（按下左按钮、按下右按钮、同时按下两个按钮）并显示相应信息。请将下列代码添加到窗体的 `MouseMove` 事件中：

```
Private Sub Form_MouseMove(Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button = 1 Then  
        Print "You're pressing the left button."  
    ElseIf Button = 2 Then  
        Print "You're pressing the right button."  
    ElseIf Button = 3 Then  
        Print "You're pressing both buttons."  
    End If  
End Sub
```

也可用 `And` 操作符与 `Select Case` 语句判断 `button` 和 `shift` 状态。`And` 操作符与 `Select Case` 语句结合可隔离三按钮鼠标可能具有的按钮状态并显示相应信息。

在窗体声明部分创建名为 `ButtonTest` 的变量：

```
Dim ButtonTest as Integer
```

将下列代码添加到窗体的 `MouseMove` 事件中：

```
Private Sub Form_MouseMove(Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    ButtonTest = Button And 7  
    Select Case ButtonTest  
        Case 1 '或 vbLeftButton  
            Print "You're pressing the left button."  
        Case 2 '或 vbRightButton  
            Print "You're pressing the right button."  
        Case 4 '或 vbMiddleButton  
            Print "You're pressing the middle button."  
        Case 7  
            Print "You're pressing all three buttons."  
    End Select  
End Sub
```

使用 Button 参数增强图形鼠标应用程序

可用 `button` 参数增强 `Scribble` 应用程序的功能，在本章前面的“`MouseMove` 事件”中已描述了该应用程序。当按下左按钮时绘制连续线，释放按钮时则停止绘制，除此之外，当用户按下右按钮时，应用程序将绘制一条从上次绘制点出发的直线。

在编写代码时，留意每个相关事件和预期响应是非常有益的。此处，三个相关事件是鼠标事件：

- **Form_MouseDown**: 根据鼠标按钮状态的不同，事件执行不同的操作：如果按下左按钮，则设置 **DrawNow** 为 **True** 并重置绘图坐标；如果按下右按钮，则绘制直线。
- **Form_MouseUp**: 如果释放左按钮，则设置 **DrawNow** 为 **False**。
- **Form_MouseMove**: 如果 **DrawNow** 为 **True**，则绘制一条直线。

在窗体的声明部分声明变量 **DrawNow**：

```
Dim DrawNow As Boolean
```

根据触发事件的按钮是左按钮还是右按钮，**MouseDown** 过程执行不同的操作：

```
Private Sub Form_MouseDown (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button = vbLeftButton Then  
        DrawNow = True  
        CurrentX = X  
        CurrentY = Y  
    ElseIf Button = vbRightButton Then  
        Line -(X, Y)  
    End If  
End Sub
```

仅当释放左按钮后，**MouseUp** 过程才停止绘图：

```
Private Sub Form_MouseUp (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If Button = vbLeftButton Then DrawNow = False  
End Sub
```

注意，在 MouseUp 过程中，将位设置为 1 (vbLeftButton)，这个位就指出已释放鼠标左按钮，因此应停止绘图。

以下 MouseMove 过程和本章前面的“MouseMove 事件”中的 Scribble 应用程序里的过程相同。

```
Private Sub Form_MouseMove (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    If DrawNow Then Line -(X, Y)  
End Sub
```

检测 SHIFT、CTRL 及 ALT 键的状态

鼠标和键盘事件用 shift 参数判断是否按下了 SHIFT、CTRL 和 ALT 键，以及以什么样的组合（如果存在）按下这些键。如果按 SHIFT 键，则 shift 为 1；如果按 CTRL 键，则 shift 为 2；如果按 ALT 键，则 shift 为 4。应使用这些键值的总和来判断这些组合。例如，同时按下 SHIFT 和 ALT 键时 shift 等于 5 (1 + 4)。

如图 11.5 所示，shift 中的三个最不明显位对应 SHIFT、CTRL 和 ALT 键的

状态。

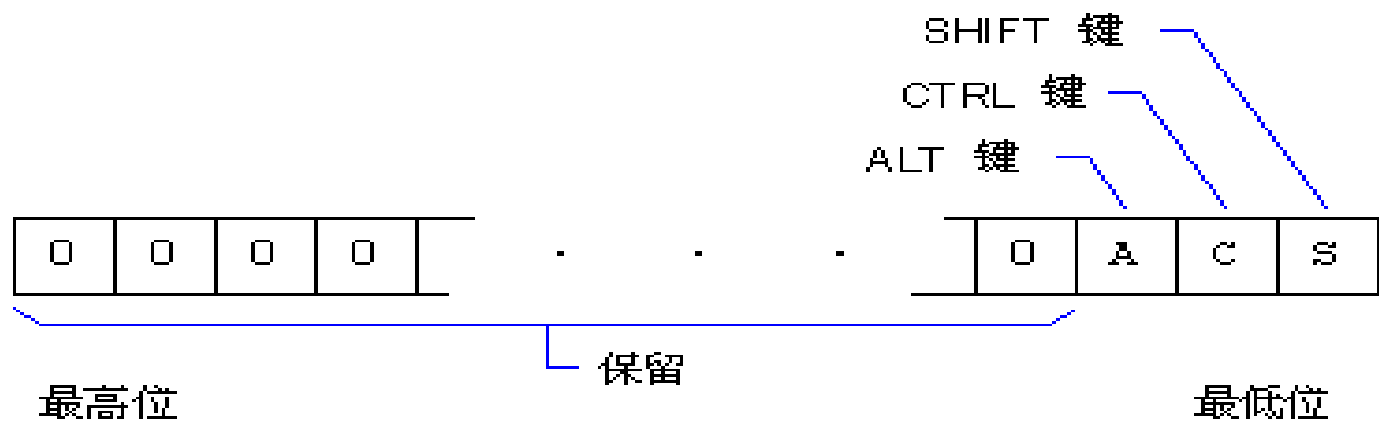


图 11.5 位如何表示 SHIFT、CTRL 和 ALT 键的状态

根据 SHIFT、CTRL 和 ALT 键的状态可在 shift 中设置任一位或设置所有位。下表列出这些值和常数：

二进制值	十进制值	常数	意义
001	1	vbShiftMask	按 SHIFT 键
010	2	vbCtrlMask	按 CTRL 键
100	4	vbAltMask	按 ALT 键
011	3	vbShiftMask + vbCtrlMask	按 SHIFT 键和 CTRL 键
101	5	vbShiftMask + vbAltMask	按 SHIFT 键和 ALT 键
110	6	vbCtrlMask + vbAltMask	按 CTRL 键和 ALT 键
111	7	vbCtrlMask + vbAltMask + vbShiftMask	按 SHIFT、CTRL 和 ALT 键

象对鼠标事件的 `button` 参数那样，可将 `If...Then...Else` 语句或 `And` 操作符与 `Select Case` 语句组合使用以判断是否按下 `SHIFT`、`CTRL` 和 `ALT` 键以及以什么样的组合（若存在）按下这些键。

打开一个新工程并将变量 `ShiftTest` 添加到窗体的声明部分：

```
Dim ShiftTest as Integer
```

将下列代码添加到窗体的 `MouseDown` 事件中：

```
Private Sub Form_MouseDown(Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    ShiftTest = Shift And 7  
    Select Case ShiftTest  
        Case 1 ' 或 vbShiftMask  
            Print "You pressed the SHIFT key."  
        Case 2 '或 vbCtrlMask  
            Print "You pressed the CTRL key."  
        Case 4 '或 vbAltMask  
            Print "You pressed the ALT key."  
        Case 3  
            Print "You pressed both SHIFT and CTRL."  
        Case 5  
            Print "You pressed both SHIFT and ALT."  
        Case 6  
            Print "You pressed both CTRL and ALT."
```

Case 7

Print "You pressed SHIFT, CTRL, and ALT."

End Select

End Sub

拖 放

在设计 Visual Basic 应用程序时可能经常要在窗体上拖动控件。Visual Basic 的拖放功能使用户在程序运行时也能具有这种能力。称按下鼠标按钮并移动控件的操作为拖动，称释放按钮的操作为放下。

注意 在运行时拖动控件并不能自动改变控件位置——必须亲自编程来重新放置控件，“改变控件位置”一节中介绍了这种方法。通常只用拖动指出应该完成某项操作；释放鼠标按钮后，控件将保持其初始位置。用下列拖放属性、事件和方法能够指定拖动操作的意义，而且能指定对于给定控件启动拖动操作的方法。

类别	项目	描述
属性	DragMode	启动自动拖动控件或手工拖动控件
	DragIcon	指定拖动控件时显示的图标
事件	DragDrop	识别何时将控件拖动到对象上
	DragOver	识别何时在对象上拖动控件
方法	Drag	启动或停止手工拖动

除 menus、timers、lines、shapes 以外的所有控件均支持 DragMode、DragIcon 属性和 Drag 方法。窗体识别 DragDrop 和 DragOver 事件，但不支持 DragMode、DragIcon 属性或 Drag 方法。

注意 只有在控件没有焦点时才会被拖动。为防止控件获得焦点，请将 TabStop 属性设置为 False。

启动自动拖动模式

为使用户拖动控件，请将控件 DragMode 属性设置为 1-自动化。在将拖动设置为“自动化”后，拖动就总是“打开”的。若要进一步控制拖动操作，请使用本章后面的“启动拖动或停止拖动时的控制”一节中描述的 0-Manual 设置。

注意 在自动拖动操作发生时，正被拖动的控件不识别其它鼠标事件。

改变拖动图标

拖动控件时，Visual Basic 将控件的灰色轮廓作为缺省的拖动图标。对 DragIcon 属性进行设置，就可用其它图像代替该轮廓。此属性包含对应图形图像的 Picture 对象。

设置 DragIcon 属性的最简单方法就是使用属性窗口。选定 DragIcon 属性后单击“属性”按钮，再从“加载图标”对话框中选择包含图形图像的文

件。

可将 Visual Basic 图标库中的图标分配给 DragIcon 属性（图标位于 \Program files\Microsoft Visual Basic\Icons 目录下）。也可用图像程序创建自己的拖动图标。

在运行时将一个控件的 DragIcon 属性赋给另一个控件的同一属性，由此就可选择拖动图标图像：

```
Set Image1.DragIcon = Image2.DragIcon
```

运行时将一个控件的 Picture 属性赋给另一控件的 DragIcon 属性，由此也可设置 DragIcon 属性：

```
Set Image1.DragIcon = Image3.Picture
```

还可使用 LoadPicture 函数：

```
Set Image1.DragIcon = LoadPicture("c:\Program _  
files\Microsoft Visual Basic\Icons _  
\Computer\Disk04.ico")
```

详细信息 关于 Picture 属性和 LoadPicture 函数的详细信息，请参阅第十二章，“使用文本和图形”。同时请参阅《Visual Basic 6.0 语言参考手册》的“Picture 属性”和“LoadPicture 函数”。

放下对象时的响应

在拖动对象后释放鼠标按钮时，Visual Basic 生成 DragDrop 事件。可用多

种方法响应此事件。应记住，控件无法自动移动到新位置，但可编写代码将控件重新放到新位置（此位置由灰色轮廓的最后位置指示）。关于详细信息，请参阅“改变控件位置”。

在讨论拖放操作时，有两个术语非常重要：源和目标。

术语	意义
源	被拖动的控件。此控件是 menu、timer、line 或 shape 外的任一对象
目标	其上放控件的对象。此对象可为窗体或控件，能识别 DragDrop 事件

鼠标指针位于某控件边框内时释放按钮，控件成为目标。指针位于窗体上无控件的区域上时，窗体成为目标。

DragDrop 事件提供三个参数：source，x 和 y。source 参数引用放到目标上的控件。

因为将源声明为 As Control，所以可象使用控件一样使用它——可引用其属性或调用其方法。

以下的示例说明了源和目标如何相互作用。示例中的源是 Image 控件，设置其 Picture 属性，以加载代表新文件夹的样本图标文件。它的 DragMode 属性已被设置为 1-Automatic 而其 DragIcon 属性被设置为一个样本拖放图标文件。目标也是 image 控件，它包含一个打开着的文件柜的图片。

将以下过程添加到第二个 image 控件的 DragDrop 事件中：

```
Private Sub Image2_DragDrop(Source As Control, _
    X As Single, Y As Single)
    Source.Visible = False
    Image2.Picture = LoadPicture("c:\Program _
```



```
Files\Microsoft Visual _  
Basic\Icons\Office\Files03a.ico")
```

```
End Sub
```

把 Image1 拖放到 Image2 后，Image1 就会消失，而 Image2 会将其图片变为关闭着的文件柜图片。使用源参数将 Image1 的 Visible 属性变成 False。

注意 应小心使用源参数。虽然知道它总是引用控件，但不一定清楚引用哪种控件。例如，如果控件是文本框，并试图引用 Source.Value，则由于文本框没有 Value 属性而导致运行时错误。

可用 If...Then...Else 语句及 TypeOf 关键字判断放了哪种类型的控件。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》及第九章“用对象编程”中的“ If...Then...Else”。

启动拖动或停止拖动时的控制

对于 DragMode 属性，Visual Basic 的手动设置比自动设置提供更多控制。手动设置允许指定可以拖动控件的时间以及不可拖动控件的时间（在将 DragMode 设置成“自动化”时，只要设置不变，就总能拖动控件）。

比如要在响应 MouseDown 和 MouseUp 事件或响应键盘命令或菜单命令时得以进行拖动。有了手动设置，还可在开始拖动前识别 MouseDown 事件，这样就可以记录鼠标的位置。

为在代码中启动拖动，应将 `DragMode` 保持为缺省设置 (0-`Manual`)。然后，无论何时开始拖动或停止拖动都请使用 `Drag` 方法。请用下述 Visual Basic 常数指定 `Drag` 参数的操作。

常数	值	意义
<code>VbCancel</code>	0	取消拖动操作
<code>VbBeginDrag</code>	1	开始拖动操作
<code>VbEndDrag</code>	2	结束拖动操作

`Drag` 方法的语法如下：

`[object.]Drag action`

如果将操作设置为 `vbBeginDrag`，则由 `Drag` 方法启动控件的拖动。如果将操作设置为 `vbEndDrag`，则放下控件并引发 `DragDrop` 事件。如果将操作设置为 `vbCancel` 则取消操作。这里的效果与设置成 `vbEndDrag` 值的情况类似，不同之处在于不引发 `DragDrop` 事件。

本章前面的“放下对象时的响应”中有一示例，在此示例的基础上可对 `Image1` 添加 `MouseDown` 事件，其中，`Image1` 对 `Drag` 方法进行说明。将 `Image1` 的 `DragMode` 属性设置为 0-`Manual`，然后添加以下过程：

```
Private Sub Image1_MouseDown(Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    Image1.Drag vbBeginDrag  
    Set Image1.DragIcon = LoadPicture("c:\Program _  
        files\ Microsoft Visual _  
        Basic\Icons\Dragdrop\Dragfldr.ico")
```

```
End Sub
```

将 `DragOver` 事件过程添加到 `Image2` 中，则当源进入目标时可终止拖动。在移动 `Image1` 经过 `Image2` 时，本例将关闭文件柜。

```
Private Sub Image2_DragOver(Source As Control, _  
    X As Single, Y As Single, State As Integer)  
    Source.Drag vbEndDrag  
    Source.Visible = False  
    Image2.Picture = LoadPicture("c:\Program _  
        files\Microsoft Visual _  
        Basic\Icons\Office\Files03a.ico")
```

```
End Sub
```

将第三个 `Image` 控件添加到窗体上则可对取消拖动操作进行演示。在本例中，`Image3` 的 `Picture` 属性包含一个垃圾桶图标。拖动文件经过 `Image3` 并使用 `DragOver` 事件与 `source` 参数则可取消拖动操作。

```
Private Sub Image3_DragOver(Source As Control, _  
    X As Single, Y As Single, State As Integer)  
    Source.Drag vbCancel
```

```
End Sub
```

改变控件的位置

在释放鼠标按钮后可能会希望改变源控件的位置。为把控件移动到鼠标的新位置处，应对任何已被允许拖动的控件使用 `Move` 方法。

在窗体上将控件拖放到任意不被其它控件占据的位置，此时就可将控件重新定位。为说明这一点，应启动一个新的 `Visual Basic` 工程，在窗体上添加 `Image` 控件并设置 `Picture` 属性，从而将图标或位图分配给 `Image` 控件，然后将 `Image` 控件的 `DragMode` 属性改成 `1-Automatic`。

将以下过程添加到窗体的 `DragDrop` 事件中：

```
Private Sub Form_DragDrop (Source As Control, _  
    X As Single, Y As Single)  
    Source.Move X, Y  
End Sub
```

此代码并未完全按照意图产生预期效果，因为它将控件的左上角放在鼠标位置。下列代码则将控件的中央放在鼠标位置：

```
Private Sub Form_DragDrop (Source As Control, _  
    X As Single, Y As Single)  
    Source.Move (X - Source.Width / 2), _  
        (Y - Source.Height / 2)  
End Sub
```

当 `DragIcon` 属性的设置值不是缺省值（灰色矩形）时，代码的工作状态最佳。当使用灰色矩形时，用户往往希望将控件准确地移入灰色矩形的最终位置。为此，应记录鼠标在源控件中的初始位置。然后将此位置值作为移动控件时的偏移量来使用。

记录鼠标的起始位置：

1. 指定手工拖动控件。
2. 声名两个窗体级变量 DragX 和 DragY。
3. 当 MouseDown 事件发生时打开拖动状态。
4. 将 x 和 y 的值存储在此事件的窗体级变量中。

下例说明如何为名为 Image1 的 image 控件产生拖动动作。在设计时应将控件的 DragMode 属性设置为 0-Manual。声名部分包括窗体级变量 DragX 和 DragY，这两个变量将鼠标在 Image 控件中的起始位置记录下来：

```
Dim DragX As Single, DragY As Single
```

控件的 MouseDown 和 MouseUp 过程分别执行打开拖动状态和放下控件的操作。此外，MouseDown 过程还将拖动开始时鼠标在控件中的位置记录下来：

```
Private Sub Image1_MouseDown (Button As Integer, _  
    Shift As Integer, X As Single, Y As Single)  
    Image1.Drag 1  
    DragX = X  
    DragY = Y  
End Sub
```

Form_DragDrop 过程确实移动了控件。为简化此例，假设 Image1 是窗体上的唯一控件，因而目标只是窗体本身。Form_DragDrop 过程用 DragX、DragY 偏移量将控件再定位：

```
Private Sub Form_DragDrop (Source As Control, _
```

X As Single, Y As Single)

Source.Move (X - DragX), (Y - DragY)

End Sub

注意，本例假定 Image1 和窗体的坐标系统使用相同的单位。若不相同，则应进行单位转换。

详细信息 关于坐标系统的详细信息，请参阅《Visual Basic 6.0 语言参考手册》的第十二章“使用文本和图形”和“ScaleMode 属性”。

OLE 拖放

可在 Visual Basic 应用程序中添加的最强大、最有用的功能之一就是在控件和控件之间、在控件和其它 Windows 应用程序之间拖动文本和图形。有了 OLE，就可将这种功能引入到应用程序中。

使用 OLE 拖放时，并不是把一个控件拖动到另一个控件并调用代码（象本章前面讨论的拖放一样）；而是将数据从一个控件或应用程序移动到另一个控件或应用程序。例如，您可以选择并拖动 Excel 中的一个单元范围，然后将它们放到应用程序的 DataBoundGrid 控件上。

Visual Basic 的几乎所有控件都在某种程度上支持 OLE 拖放。此外，一些标准的和 ActiveX 控件（由 Visual Basic 专业版和企业版提供的）还提供对 OLE 拖放的自动支持，这意味着控件支持在 OLEDragMode 和 OLEDropMode 属性中的自动设置，并且无论是从控件拖出还是在控件内放

入都不需要编写代码。这一点与手动拖放相反，在手动拖放中，您必须为拖放编程：

有些支持自动 `OLEDragMode` 和 `OLEDropMode` 的控件包括 `PictureBox`、`Label`、以及 `TextBox` 控件。为对这些控件启动自动 OLE 拖放，应将 `OLEDragMode` 和 `OLEDropMode` 都设置为“自动化”。

有些控件支持自动 OLE 拖动，但只支持手动放下，有些支持自动放下，但只支持手动拖动。例如，`ComboBox` 控件支持手动和自动拖动，但不支持自动放下。这是由于如果您将一个项目拖动到 `ComboBox` 中，Visual Basic 无法知道将新项目放下的确切位置。然而，可以使用手动放下

，这样您就可以将项目程序化的放在您希望在 `ComboBox` 中的位置了。要启动这些控件的自动拖动，应将 `OLEDragMode` 属性设置为“自动化”。

有些控件只支持手动 OLE 拖放事件，这意味着可用代码对它们进行编程，使之成为 OLE 拖放操作的源，也可使之成为 OLE 拖放操作的目标。

注意 为判断其它 ActiveX 控件是否支持 OLE 拖放，应在 Visual Basic 中加载控件并检查 `OLEDragMode` 和 `OLEDropMode` 属性是否存在，或检查 `OLEDrag` 方法是否存在（不自动支持 OLE 拖动的控件也将不具有 `OLEDragMode` 属性，但是，如果控件通过代码支持 OLE 拖动，则其具有 `OLEDrag` 方法）。

注意 窗体、MDI 窗体、文档对象、用户控件和属性页都包含 `OLEDropMode` 属性，而且仅支持手工拖放。

可用下列 OLE 拖放属性、事件和方法指定已知控件响应拖放的方式。

类 别	项 目	描 述
属 性	OLEDragMode	启动控件的自动拖动或手工拖动（若控件支持手工拖动但不支持自动 OLE 拖动，则它不具有此属性，但支持 OLEDrag 方法和 OLE 拖放事件）
	OLEDropMode	指定控件如何响应放操作
事 件	OLEDragDrop	识别源对象何时被放到控件上
	OLEDragOver	识别源对象何时被拖动经过控件
	OLEGiveFeedback	以源对象为基础向用户提供自定义拖动图标反馈
	OLEStartDrag	在启动拖动时，源支持哪种数据格式和放效果（复制、移动或拒绝数据）
	OLESetData	在放源对象时提供数据
方 法	OLECompleteDrag	当把对象放到目标时通知被执行的操作的源
	OLEDrag	启动手工拖动

自动拖放与手工拖放的比较

将 OLE 拖放设想成自动实现或手工实现，这是非常有用的想法。举例来说自动拖放意味着，只需直接把文本框控件的 OLEDragMode 和 OLEDropMode 属性设置为“自动化”，就可将文本从一个文本框控件拖动到另一个文本框控件：无需编写任何代码来响应 OLE 拖放事件。当把一

列单元从 Excel 拖动到 Word 文档时，就已执行自动拖放操作了。自动拖动数据的工作可能会成为最好的、最简单的方法，这取决于已知的控件或应用程序如何支持 OLE 拖放以及拖动了什么类型的数据。

手工拖放意味着已选择（或被迫使用）手工处理一个或若干 OLE 拖放事件。为进一步控制拖放过程的每步操作，为向用户提供自定义的视觉反馈，为创建自己的数据格式，在这些时候，手工实现 OLE 拖放可能是较好的方法。当控件不支持自动拖放时，手工实现是唯一选择。

定义 OLE 拖放操作的整体模型也是非常有用的。在拖放操作中，从对象中拖动数据，就算该对象为源。称放入数据的对象为目标。Visual Basic 提供属性、事件和方法，控制并响应那些影响源和目标的操作。能够辨别出源和目标是在不同的应用程序中还是在同一个应用程序甚至在同一个控件中，这是很有用的。根据具体情况的不同，可能需要为源或目标或同时为两者编写代码。

启动自动 OLE 拖放

若控件支持自动拖放，则可将其 OLEDragMode 和 OLEDropMode 属性中的至少一个设置为“自动化”，从而对 Visual Basic 控件向外拖动数据或向内放数据。例如，有时可能希望将文本从文本框控件拖动到 Word for Windows 文档中，或希望文本框控件能够接受从 Word for Windows 文档中拖动来的数据。

为了得以从文本框控件拖动数据，应将 OLEDragMode 属性设置为“自动

化”。运行时，可选定输入到文本框控件中的文本，并将它拖动到打开的 Word for Windows 文档中。

按照缺省规定，在将文本从文本框控件拖动到 Word for Windows 文档时，文本被移动到文档而不是被复制。若放文本的同时按下 CTRL 键，则数据被复制到（而不是移动到）文档。对所有支持 OLE 拖放的对象或应用程序，这都是一个缺省性能。为限制此操作，使之只移动数据或只复制数据，需使用手工拖放技术修改自动性能。关于更详细的信息，请参阅“使用鼠标和键盘修改放下效果和用户反馈”。

为使文本框控件在 OLE 拖放操作中自动获得数据，应将其 OLEDropMode 属性设置为“自动化”。运行时，除非在拖动的同时按下 CTRL 键或通过代码修改控件的缺省性能，否则，对于从允许 OLE 操作的应用程序拖动到文本框控件的数据，进行的是移动而不是复制操作。

自动支持拖放操作也有其局限性，其中某些来自控件功能本身。例如，如果将 Word for Windows 文档中的文本拖动到文本框控件，则 Word 文档中的所有丰富文本格式将消失，因为文本框控件不支持这种格式。大多数控件都有大致相同的限制。对自动操作的另一个限制是：无法对所拖放数据的类型进行全面控制。

注意 在拖动数据时可能会注意到鼠标的指针，对于所拖动数据的类型，指出指针经过的对象是否支持 OLE 拖放。如果对象支持 OLE 拖放，则显示“放”指针。如果对象不支持 OLE 拖放，则显示“不放”指针。

OLE 拖放 DataObject 对象

OLE 拖放使用的源、目标模型与“拖放”一节中讨论的简单事件驱动拖放技术使用的模型相同。但在这种情况下并不是将一个控件拖动到另一个控件以调用某些代码；而是将一个控件或应用程序中的数据移动到另一个控件或应用程序中。例如，用户选定 Excel（源）中的一列单元并将它们拖动到应用程序的 DataGrid 控件（目标）中。

在 Visual Basic 中，数据的媒介或存储处是 DataObject 对象——这意味着通过它将数据从源移动到目标。为了做到这一点，DataObject 对象提供了为存储、获取与分析数据所需的方法。下表列出了 DataObject 对象使用的属性及方法：

类别	项目	描述
属性	Files	保存出入于 Windows Explorer 的被拖动的文件名
方法	Clear	清除 DataObject 对象的内容
	GetData	从 DataObject 对象中获取数据
	GetFormat	判断在 DataObject 中是否可用特定数据格式
	SetData	将数据放到 DataObject 对象中，或指出，在请求时可用的特定格式

将这些方法与 OLE 拖放事件并用就可管理源端和目标端（若两端均在 Visual Basic 应用程序中的话）的 DataObject 对象中的数据。例如，可用 SetData 方法将数据放到源端的 DataObject 对象中，然后用 GetData 方法接受目标端的数据。

在触发 `OLEStartDrag` 事件时可用 `Clear` 方法清除 `DataObject` 对象的内容。当用自动拖动操作从控件拖动数据时，在触发 `OLEStartDrag` 事件之前就把数据格式放到 `DataObject` 对象中。若不希望使用缺省数据格式，则请用 `Clear` 方法。若希望在缺省数据格式的基础上添加数据，则不使用 `Clear` 方法。

通过 `Files` 属性可存储一系列文件名，这些文件能被拖动到拖放目标。关于此属性的详细信息，请参阅“从 Windows ‘资源管理器’ 中拖动文件”。

还可指定要传送数据的格式。`SetData` 和 `GetData` 方法使用下述参数将数据放到 `DataObject` 对象或从 `DataObject` 对象中获取数据：

参数	描述
<i>Data</i>	用来指定放到 <code>DataObject</code> 对象中的数据类型（如果设置了 <i>format</i> 参数，则此项是可选参数；否则，此项是必要参数）
<i>Format</i>	允许设置若干可由源支持的不同格式，而无需对每种格式都加载数据（若设置了 <i>data</i> 参数或 Visual Basic 能够处理数据格式，则此项是可选的参数；否则此项是必要的参数）

注意 当把数据放到目标而未指定格式时，Visual Basic 能够检测数据是否为位图、元文件、增强型的元文件或文本。必须明确指定所有其它格式，否则将产生错误。

format 参数使用下述常数或数值指定数据的格式：

常数	值	意义
<code>vCFText</code>	1	文本

续表

vbCFBitmap	2	位图 (.bmp)
vbCFMetafile	3	元文件 (.wmf)
vbCFEMetafile	14	增强型元文件 (.emf)
vbCFDIB	8	与设备无关的位图 (.dib or .bmp)
vbCFPalette	9	调色板
vbCFFiles	15	文件列表
vbCFRTF	-16639	丰富文本格式 (.rtf)

SetData、GetData 和 GetFormat 方法使用 *data* 参数和 *format* 参数返回 DataObject 对象中的数据类型，或者，如果格式与目标兼容，则获取数据。例如：

```
Private Sub txtSource_OLEStartDrag(Data As _  
    VB.DataObject, AllowedEffects As Long)  
    Data.SetData txtSource.SelText, vbCFText  
End Sub
```

本例中，数据是在文本框中选定的文本，指定的格式是文本 (vbCFText)。

注意 在大多数情况下应使用 vbCFDIB 数据格式而不使用 vbCFBitmap 和 vbCFPalette。VbCFDIB 格式包含位图和调色板，因此是传送位图图像的首选方法。但是，因顾及完整性也可指定 vbCFBitmap 和 vbCFPalette。如果决定不使用 vbCFDIB 格式，则必须同时指定 vbCFBitmap 和 vbCFPalette 格式，使得能够将位图与调色板正确地放入 DataObject 对象。

详细信息 关于定义自己的数据格式的信息，请参阅“创建自定义数据格式”。

OLE 拖放如何工作

当执行 OLE 拖放操作时，在源端和目标端会产生某些事件。无论拖放操作是自动操作还是手工操作，都总要生成与源对象关联的事件。但是，只在手工拖放操作时才生成目标端事件。下图说明在拖动源会有什么事件发生并可得到响应，以及在放目标会有什么事件发生并可得到响应。

需要响应哪个事件取决于如何实现拖放功能。例如，可能已创建了一个具有文本框的应用程序并希望该文本框能够自动接受从其它应用程序拖动来的数据。这种情况下直接将控件的 `OLEDropMode` 属性设置为“自动化”。如果还希望能从文本框控件中自动拖动数据，则应将其 `OLEDragMode` 属性设置为“自动化”。

但是，如果希望改变缺省鼠标光标或增强按钮状态和 `shift` 键的功能，则需手工响应源端和目标端事件。同样，如果希望在将数据放到控件之前就对其进行分析（例如检查数据是否兼容），或者在将数据加载到 `DataObject` 对象时延迟若干时间（这样，在开始时就不必加载多个格式），则必须使用手工 OLE 拖放操作。

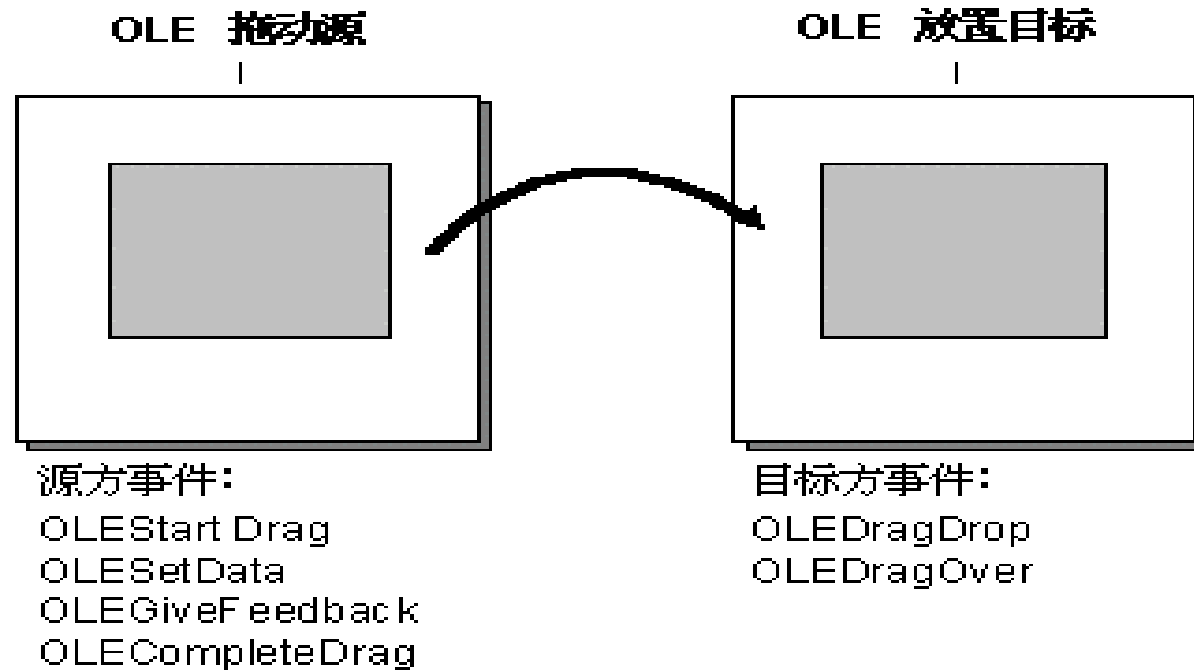


图 11.6 源端事件和目标端事件

因为可将数据拖动到众多具有不同限制和要求的 Visual Basic 控件或 Windows 应用程序中，所以实现 OLE 拖放的难度，从简单到复杂，极为不同。当然，最简单的实现方式是在两个自动对象之间拖放，无论对象是 Word 文档、Excel 电子数据表，还是应用程序中设置为“自动化”的控件。对拖放目标指定多个可接受的数据格式，这将更加复杂。

启动拖动

在 Visual Basic 应用程序的基本手工 OLE 拖放操作过程中究竟发生了什么？

当用户选定数据并按下鼠标左按钮，因而从 OLE 拖动源（例如，一个文本框控件）拖动数据时将触发 `OLEStartDrag` 事件，然后就可存储数据，也可直接指定源所支持的格式。还需指明，源允许复制数据还是允许移动数据，还是两种方法都允许。

详细信息 关于 `OLEDrag` 方法、`OLEstartDrag` 事件、如何使用 `SetData` 方法指定可支持的数据格式和如何将数据放到 `DataObject` 中的详细信息，请参阅“启动 OLE 拖动操作”。

在目标上拖动数据

在拖动数据经过目标时会触发目标的 `OLEDragOver` 事件，这表明源位于目标的边界内。而后，如果在这里放下数据，则应指定目标要执行的操作——或者是复制、移动，拒绝数据。按照约定，缺省操作通常是移动，但也可以是复制。

如果在这里放下源，则目标指定所产生的拖放效果，此时将触发一个 `OLEGiveFeedback` 事件。用这个事件向用户提供视觉反馈，当放下选定数据时，按照该反馈执行操作——也就是说，鼠标指针将发生变化，以指示操作的类型是复制、移动，还是“不放”。

当源在目标边界内移动时——或者若用户按住鼠标按钮的同时按下了 `SHIFT`、`CTRL` 或 `ALT` 键——拖放效果可能会改变。例如，数据可能被拒绝，而不是被复制或移动。

例如，如果移动时超出目标的边界或按 `ESC` 键，则可能会取消或修改拖动操作（鼠标指针可能发生变化，指出当前所经过的对象不接受数据）。

详细信息 关于 `OLEDragOver` 和 `OLEGiveFeedback` 事件详细信息，请参阅“将 OLE 拖动源拖到 OLE 拖放目标的上方”。

完成拖动

当把源放到目标时将触发目标的 `OLEDragDrop` 事件。目标就源所包含的数据的格式（若拖动开始时未将数据放在源中，则就源所支持的格式）查询源，然后获取数据或拒绝数据。

如果在拖动开始时就已存储数据，则目标将使用 `GetData` 方法获取数据。如果在拖动开始时未存储数据，则通过触发 `OLESetData` 事件并使用 `SetData` 方法获取数据。

在接受数据或拒绝数据时将触发 `OLECompleteDrag` 事件，而后，源将执行相应操作：例如，如果接受数据并指定一个移动，则源将删除数据。

详细信息 关于 `OLEDragDrop` 事件、`OLECompleteDrag` 事件及如何使用 `GetFormat` 和 `GetData` 方法从 `DataObject` 对象中获取数据的详细信息，请参阅“将 OLE 拖动源放到 OLE 拖放目标上”。

启动 OLE 拖动操作

如果想要指定所支持的是哪种数据格式或拖放效果（复制、移动或不放），或者，如果想从中拖出数据来的控件不支持自动拖动，则需使用手工的 OLE 拖动操作。

手工拖动操作的第一阶段是调用 `OLEDrag` 方法，设置允许的拖动效果，指定支持的数据格式，以及有选择地将数据放到 `DataObject` 对象中。使用 `OLEDrag` 方法手工启动拖动操作，并使用 `OLEStartDrag` 事件指定允许的拖动操作效果及所支持的数据格式。

OLEDrag 方法

一般来说，在选定数据后，按住鼠标左按钮并移动鼠标时，从对象的 `MouseMove` 事件调用 `OLEDrag` 方法。

`OLEDrag` 方法未提供任何参数。其主要目的是启动手工拖动，然后允许 `OLEStartDrag` 事件设置拖动操作的条件（例如，指定在将数据拖动到另一个控件时发生的事情）。

如果源控件支持 `OLEDragMode` 属性，则为了手工控制拖动操作，必须将此属性设置为“手工”并在控件处使用 `OLEDrag` 方法。如果控件支持手工 OLE 拖动而不支持自动 OLE 拖动，则其不具有 `OLEDragMode` 属性，但支持 `OLEDrag` 方法和 OLE 拖放事件。

注意 如果将源控件 `OLEDragMode` 属性设置为“自动化”，则 `OLEDrag` 方法仍有效。

指定拖动效果和数据格式

在手工 OLE 拖动操作中，当用户开始拖动源并调用 `OLEDrag` 方法时，控件的 `OLEStartDrag` 事件将会发生。用此事件指定拖动效果和源所支持的

数据格式。

OLEStartDrag 事件用两个参数指定支持的数据格式，和放下数据时指定是复制数据还是移动数据（放下效果）。

注意 若未在 OLEStartDrag 事件中指定放下效果和数据格式，则将不启动手工拖动。

AllowedEffects 参数

allowedeffects 参数指定拖动源所支持的拖放效果。例如：

```
Private Sub txtSource_OLEStartDrag(Data As _
    VB.DataObject, AllowedEffects As Long)
    AllowedEffects = vbDropEffectMove Or _
        vbDropEffectCopy
End Sub
```

然后，目标可向拖动源查询此信息并作相应的响应。

Allowedeffects 参数使用下述数值指定放下效果：

常 数	值	描 述
vbDropEffectNone	0	放下目标无法接受数据
vbDropEffectCopy	1	放下的结果为复制。拖动源未改变原始数据
vbDropEffectMove	2	拖动源删除了数据

Format 参数

可通过设置 `OLEStartDrag` 事件的 *format* 参数指定对象支持的数据格式。为此应使用 `SetData` 方法。例如，用丰富文本框控件作源，并用文本框控件作目标，就可指定下述支持格式：

```
Private Sub rtbSource_OLEStartDrag(Data As _  
    VB.DataObject, AllowedEffects As Long)  
    AllowedEffects = vbDropEffectMove Or _  
        vbDropEffectCopy  
  
    Data.SetData , vbCFText  
    Data.SetData , vbCFRTF  
End Sub
```

目标可查询源，从而可判断支持何种数据格式并作出相应的响应——举例来说，若目标不支持所放下数据的格式，它就会拒绝放下的数据。在这种情况下，只有源所支持的数据格式才是文本格式与丰富文本格式。

详细信息 关于 `SetData` 方法的格式值的详细信息，请参阅“OLE 拖放 `DataObject` 对象”。

将数据放到 `DataObject` 对象中

在多数情况下，特别是在源支持两种以上的格式或花费大量时间创建数据时，可能会希望只在目标请求数据时再将数据放入 `DataObject` 对象。但是，可在开始拖动操作时使用 `OLEStartDrag` 事件中的 `SetData` 方法将数据放

入 DataObject 对象。例如：

```
Private Sub txtSource_OLEStartDrag(Data As _
    VB.DataObject, AllowedEffects As Long)
    Data.Clear
    Data.SetData txtSource.SelText, vbCFText
End Sub
```

此例用 Clear 方法清除 DataObject 对象中的缺省数据格式，并指定所选定数据的数据格式（文本），然后又用 SetData 方法将数据放入 DataObject 对象。

将 OLE 拖动源拖到 OLE 拖放目标的上方

使用手工目标，可判断目标中源数据的位置并对其作出响应，还可对鼠标按钮及 SHIFT、CTRL、ALT 键的状态作出响应。源与目标均为手工时可修改缺省的鼠标视觉性能。

为了 . . .	使用 . . .
判断并响应源对象的位置	OLEDragOver 事件的 <i>state</i> 参数
响应鼠标按钮的状态	OLEDragDrop 和 OLEDragOver 事件的 <i>button</i> 参数
响应 SHIFT、CTRL 及 ALT 键的状态	OLEDragDrop 和 OLEDragOver 事件的 <i>shift</i> 参数
修改缺省的鼠标视觉性能	OLEDragOver 事件的 <i>effect</i> 参数和 OLEGiveFeedback

详细信息 关于改变鼠标光标的详细信息，请参阅“将 OLE 拖动源拖到 OLE 拖放目标的上方”。关于使用 *button* 与 *shift* 参数的详细信息，请参阅“使用鼠标与键盘修改放下效果和用户反馈”。

OLEDragOver 事件状态参数

根据位置的不同，*effect* 参数可能会相应变化，以指示当前可接受的拖放效果。

有了 OLEDragOver 事件中的 *state* 参数，就可对数据进入、经过以及离开目标控件诸状态作出响应。例如，当源数据进入目标控件时，将 *state* 参数设置为 *vbEnter*。

在放下目标边界内移动拖动源时将 *state* 参数设置为 *vbOver*。有时希望根据鼠标指针的位置（*x* 和 *y* 参数）的不同改变拖动效果。注意，每秒都会数次生成 OLEDragOver 事件，就是在鼠标静止时也不例外。

通过使用下列常数，用 OLEDragOver 事件中的 *state* 参数指定数据何时进入、经过和离开目标：

常数	值	意义
<i>vbEnter</i>	0	已将数据拖入目标的范围
<i>vbLeave</i>	1	已将数据拖出目标的范围
<i>vbOver</i>	2	数据仍在目标的范围内，而且移动了鼠标或者改变了鼠标按钮或键盘的状态，或者消耗了一段由系统决定的时间

向用户提供自定义的视觉反馈

若希望改变 OLE 拖放操作中缺省的鼠标视觉性能，可在目标端操作 `OLEDragOver` 事件并在源端操作 `OLEGiveFeedback` 事件。

在拖放操作期间，OLE 拖放提供自动视觉反馈。例如在开始拖动时，鼠标指针将发生变化，指出已开始拖动。当经过不支持 OLE 放下的对象时，鼠标指针将变成“不放下”光标。

当把数据放到某个控件时，改变鼠标指针以指出控件如何响应，这将涉及两个步骤：用 `GetFormat` 方法判断 `DataObject` 对象中的数据类型，然后设置 `OLEDragOver` 事件的 *effect* 参数以通知源，此控件允许何种放下效果。

OLEDragOver 事件

在把目标控件的 `OLEDropMode` 属性设置为“手工”时，拖动数据无论何时经过此控件即触发 `OLEDragOver` 事件。

如果放下了对象，则用 `OLEDragOver` 事件的 *effect* 参数指定执行的操作。设置此值时将触发源的 `OLEGiveFeedback` 事件。`OLEGiveFeedback` 事件包含其自身的 *effect* 参数，用此参数向用户提供视觉反馈，当拖动选定时，依据此反馈执行操作——也就是说，将改变鼠标指针以指示复制、移动，或“不放下”操作。

`OLEDragOver` 事件的 *effect* 参数使用下列常数指示放下操作：

常 数	值	描 述
vbDropEffectNone	0	放下目标无法接受数据
vbDropEffectCopy	1	放下结果为复制。拖动源未改变原始数据
vbDropEffectMove	2	拖动源删除了数据

注意 OLEDragOver 与 OLEGiveFeedback 事件的 *effect* 参数表示的放效果（复制、移动及不放下）与 OLEStartDrag 事件的 *allowedeffects* 参数表示的放效果相同。区别仅在于，OLEStartDrag 事件指定允许的效果，而 OLEDragOver 和 OLEGiveFeedback 用 *effect* 参数通知源将执行哪一个操作。

下列代码向 DataObject 对象查询与目标控件兼容的数据格式。如果数据兼容，则 *effect* 参数通知源，如果放下数据，则将执行移动操作。如果数据不兼容，则也将通知源并显示一个“不放下”鼠标指针。

```
Private Sub txtTarget_OLEDragOver(Data As _
    VB.DataObject, Effect As Long, Button As _
    Integer, Shift As Integer, X As Single, _
    Y As Single, State As Integer)
    If Data.GetFormat(vbCFText) Then
        Effect = vbDropEffectMove And Effect
    Else
        Effect = vbDropEffectNone
    End If
End Sub
```


在拖动源数据经过目标并触发 `OLEDragOver` 事件时，源将告诉目标，它允许什么效果（移动、复制及不放下）。此后，必须选择唯一效果。`OLEDragOver` 事件的 *effect* 参数通知源，它支持哪一个放下操作，然后，源又用 `OLEGiveFeedback` 事件改变鼠标指针，以此方式通知用户。

OLEGiveFeedback 事件

为了根据 `OLEDragOver` 事件的 *effect* 参数改变鼠标指针的缺省性能，需用 `OLEGiveFeedback` 事件手工指定新的鼠标指针值。设置 `OLEDragOver` 事件的 *effect* 参数时将会自动触发源的 `OLEGiveFeedback` 事件。

`OLEGiveFeedback` 事件包含两个参数 (*effect* 和 *defaultcursors*)，使用它们即可改变 OLE 拖放操作中的缺省鼠标指针。

Effect 参数象其它 OLE 拖放事件一样，指出是复制、移动还是拒绝数据。但是，在 `OLEGiveFeedback` 事件中，这个参数意在通过改变鼠标指针来指出操作类型，从而向用户提供自定义的视觉反馈。

常数	值	描述
<code>vbDropEffectNone</code>	0	放下目标无法接受数据
<code>vbDropEffectCopy</code>	1	放下的结果是复制操作。拖动源未改变原始数据
<code>vbDropEffectMove</code>	2	拖动源移动数据
<code>vbDropEffectScroll</code>	<code>&H80000000&</code>	即将启动滚动或正在目标内滚动。除使用其它值外还使用此数值

注意 某些应用程序或控件可用 `vbDropEffectScroll` 值指示用户把鼠标移动

到应用程序窗口边缘处，从而造成滚动。某些（而非全部）Visual Basic 标准控件自动支持滚动。如果将数据拖动到包含滚动条的程序中，例如拖动到 Word for Windows 中，则需编程控制滚动效果。

Defaultcursors 参数指示是否使用缺省的 OLE 光标集合。将此参数设置为 False 就可用 Screen 对象的 Screen.MousePointer 属性指定自己的光标。

对大多数情况无需指定自定义鼠标指针，因为 OLE 处理鼠标的缺省性能。如果决定用 OLEGiveFeedback 事件指定自定义鼠标指针，则需考虑每个可能的效果，包括滚动效果。创建选项，使当遇到未知效果时仍回过头来让 OLE 控制鼠标指针，由此可能会增添一些效果，对这些效果进行编程是非常可取的。

以下代码示例设置了 *effect* 和 *defaultcursors* 参数，并且通过设置 Screen 对象的 MousePointer 和 MouseIcon 属性为复制、移动和滚动效果指定了自定义光标（.ico 或 .cur 文件）。如果遇到未知效果，代码还会回过头来让 OLE 控制鼠标指针。

```
Private Sub TxtSource_OLEGiveFeedback(Eff As Long, _  
    DefaultCursors As Boolean)  
    DefaultCursors = False  
    If Eff = vbDropEffectNone Then  
        Screen.MousePointer = vbNoDrop  
    ElseIf Eff = vbDropEffectCopy Then  
        Screen.MousePointer = vbCustom  
        Screen.MouseIcon = LoadPicture("c:\copy.ico")
```

```

ElseIf Effect = (vbDropEffectCopy Or _
                vbDropEffectScroll) Then
    Screen.MousePointer = vbCustom
    Screen.MouseIcon = _
        LoadPicture("c:\copyscr1.ico")
ElseIf Effect = vbDropEffectMove Then
    Screen.MousePointer = vbCustom
    Screen.MouseIcon = LoadPicture("c:\move.ico")
ElseIf Effect = (vbDropEffectMove Or _
                vbDropEffectScroll) Then
    Screen.MousePointer = vbCustom
    Screen.MouseIcon = _
        LoadPicture("c:\movescr1.ico")
Else
    ' 如果添加了不了解的格式，则让 OLE 用正确的缺省值去处理。
    DefaultCursors = True
End If
End Sub

```

End Sub

注意 如果在 OLEGiveFeedback 事件中指定了自定义的鼠标指针，则总应在 OLECompleteDrag 事件中重置鼠标指针。关于当数据被放下时如何通知源的详细信息，请参阅“在 OLE 放下目标上放下 OLE 拖动源”。

详细信息 关于 `MousePointer` 和 `MouseIcon` 属性的详细信息，请参阅“自定义鼠标指针”。

将 OLE 拖动源放到 OLE 拖放目标上

如果目标支持手工 OLE 拖放操作，则可在目标中移动光标时控制发生的事情并指定目标所接受数据的种类。当用户将源对象放到目标控件时，可用 `OLEDragDrop` 事件向 `DataObject` 对象查询兼容的数据格式，然后再获取数据。

`OLEDragDrop` 事件还将通知放下操作源某些消息，例如，如果指定移动操作则允许源删除原始数据。

获取数据

将源放到目标时会触发 `OLEDragDrop` 事件。在启动拖动操作时，如果已将数据放到 `DataObject` 对象中，则可在触发 `OLEDragDrop` 事件时使用 `GetData` 方法获取数据。但是，如果启动拖动操作时仅仅声明了所支持的源的格式，则 `GetData` 方法将自动在源触发 `OLESetData` 事件，此事件将数据放到 `DataObject` 对象中，然后再从中获取数据。

由下例可看到，在启动拖动操作时将获取放在 `DataObject` 对象中的数据。拖动操作可能由手工启动（在源使用 `OLEDrag` 方法），也可能自动启动（通过将源的 `OLEDragMode` 属性设置为“自动化”）。该例中使用 `DataObject` 对

象的 GetData 方法获取拖动 的数据。GetData 方法提供了代表 DataObject 对象所支持的数据类型的常数。在这里，获取的数据是文本。

```
Private Sub txtTarget_OLEDragDrop(Data As _  
    VB.DataObject, Effect As Long, Button As _  
    Integer, Shift As Integer, X As Single, _  
    Y As Single)  
    txtTarget.Text = Data.GetData(vbCFText)  
End Sub
```

详细信息 关于 GetData 格式常数的完整列表，请参阅本章前面的“OLE 拖放 DataObject 对象”。

查询 DataObject 对象

有时可能需要向 DataObject 对象查询放到目标的数据的类型。可在 If... Then 语句中用 GetFormat 方法指定目标控件所能接受的数据类型。若 DataObject 对象中的数据兼容，则完成了放操作。

```
Private Sub txtTarget_OLEDragDrop(Data As _  
    VB.DataObject, Effect As Long, Button As _  
    Integer, Shift As Integer, X As Single, _  
    Y As Single)  
    If Data.GetFormat(vbCFText) Then  
        txtTarget.Text = Data.GetData(vbCFText)
```

```
End If  
End Sub
```

将数据放到 DataObject 对象中

当目标用 GetData 方法从源获取数据时，如果在启动拖动操作时未将数据放到源中，则只触发 OLESetData 事件。

在很多情况下，特别是如果源支持多种格式或者为创建数据会耗费大量时间，则可能希望仅仅在目标请求数据时才将数据放到 DataObject 对象中。OLESetData 事件使源对于一种指定的数据格式只响应一个请求。

例如，如果在启动拖动操作时用 OLEStartDrag 事件指定所支持的数据格式，但未将数据放到 DataObject 对象中，则将使用 OLESetData 事件把特定格式的数据放到 DataObject 对象中。

```
Private Sub txtSource_OLESetData(Data As _  
    VB.DataObject, DataFormat As Integer)  
    If DataFormat = vbCFText Then  
        Data.SetData txtSource.SelText, vbCFText  
    End If  
End Sub
```

放下数据时通知源

OLEDragDrop 事件的 *effect* 参数规定，在放数据时如何将数据并入目标。

设置此参数后，就用设置成该值的 *effect* 参数在源触发 `OLECompleteDrag` 事件。随后，源可执行适宜的操作：例如，若指定了移动，则源将删除数据。

`OLEDragDrop` 事件的 *effect* 参数与 `OLEDragOver` 事件的 *effect* 参数使用相同的参数，以指示放操作。下表列出这些常数：

常数	值	描述
<code>VbDropEffectNone</code>	0	放下目标无法接受数据
<code>VbDropEffectCopy</code>	1	放下结果为复制。拖动源未改变原始数据
<code>VbDropEffectMove</code>	2	拖动源删除数据

下列示例设置 *effect* 参数来指示放操作。

```
Private Sub txtTarget_OLEDragDrop(Data As _
    VB.DataObject, Effect As Long, Button As _
    Integer, Shift As Integer, X As Single, _
    Y As Single)
    If Data.GetFormat(vbCFText) Then
        txtTarget.Text = Data.GetData(vbCFText)
    End If
    Effect = vbDropEffectMove
End Sub
```

在源端，当把源放到目标上或者取消了 `OLE` 拖放操作时将会触发 `OLECompleteDrag` 事件。`OLECompleteDrag` 是拖放操作中的最后一个事件。

OLECompleteDrag 事件只包含一个参数 (*effect*)，这个参数被用来通知源，在将数据放到目标时执行的是什么操作。

Effect 参数返回的数值与其它 OLE 拖放事件 *effect* 参数使用的数值相同：vbDropEffectNone, vbDropEffectCopy, and vbDropEffectMove。

例如，在目标指定了一个移动操作而源被放到目标后设置此参数，源将删除控件中的原始数据。如果在 OLEGiveFeedback 事件中指定了自定义的鼠标指针，则还应该使用 OLECompleteDrag 事件重置鼠标指针。例如：

```
Private Sub txtSource_OLECompleteDrag(Effect As Long)
    If Effect = vbDropEffectMove Then
        txtSource.SelText = ""
    End If
    Screen.MousePointer = vbDefault
End Sub
```

用鼠标和键盘修改放下效果和用户反馈

可用 *button* 和 *shift* 参数响应鼠标按钮和 SHIFT、CTRL 和 ALT 键的状态来增强 OLEDragDrop 和 OLEDragOver 事件。例如，在控件中拖动数据时，用户可按 CTRL 键执行复制操作，按 SHIFT 键来执行移动操作。在下例中用 OLEDragDrop 事件的 *shift* 参数判断在放数据时是否按下了 SHIFT 键。若是，则执行移动操作，否则执行复制操作。

```
Private Sub txtTarget_OLEDragDrop(Data As _
```



```

        VB.DataObject, Effect As Long, Button As _
        Integer, Shift As Integer, X As Single, _
        Y As Single)
    If Shift And vbCtrlMask Then
        txtTarget.Text = Data.GetData(vbCFText)
        Effect = vbDropEffectCopy
    Else
        txtTarget.Text = Data.GetData(vbCFText)
        Effect = vbDropEffectMove
    End If
End Sub

```

可用 *button* 参数隔离并响应各种鼠标按钮状态。例如，可同时按下鼠标左右按钮，来移动数据。

当拖动源对象经过目标并按下鼠标按钮或 SHIFT、CTRL、ALT 键时，为了向用户指出即将执行的操作，可设置 OLEDragOver 事件的 *shift* 和 *button* 参数。例如，为了在拖动操作中按下 SHIFT 键时向用户指出将要执行的操作，可在 OLEDragOver 事件中添加下述代码：

```

Private Sub txtTarget_OLEDragOver(Data As _
    VB.DataObject, Effect As Long, Button As _
    Integer, Shift As Integer, X As Single, _
    Y As Single, State As Integer)
    If Shift And vbCtrlMask Then

```

```

        Effect = vbDropEffectCopy
    Else
        Effect = vbDropEffectMove
    End If
End Sub

```

详细信息 关于响应鼠标和键盘状态的详细信息，请参阅“检测鼠标按钮”和“检测 SHIFT、CTRL 和 ALT 键的状态”。

创建自定义数据格式

如果 Visual Basic 提供的格式不能满足某种特定需要，则可创建用于 OLE 拖放操作的自定义数据格式。例如，如果应用程序定义一个在两个应用程序实例之间或在应用程序内部拖动这个数据格式，自定义数据格式是非常有用的。

为了创建自定义数据格式，应调用 Windows API 中的 RegisterClipboardFormat 函数。例如：

```

Private Declare Function RegisterClipboardFormat Lib _
    "user32.dll" Alias "RegisterClipboardFormatA" _
    (ByVal lpstrFormat$) As Integer

Dim MyFormat As Integer

```

一经定义之后，就可象使用其它 DataObject 对象数据格式一样使用此格

式。例如：

```
Dim a() As Byte
```

```
a = Data.GetData(MyFormat)
```

为使用此功能，应将数据当作字节数组放入 `DataObject` 对象或从这个对象中获取数据。而后可将自定义数据格式赋予一个字符串变量，因为可对其自动进行转换。

警告 用 `GetData` 方法获取自定义数据格式，这可能会导致难以预料的后果。

由于 `Visual Basic` 不理解自定义的数据格式（因为这是程序员定义的），所以无法判断数据的大小。`Visual Basic` 可判断字节数组的大小，因为数组已由 `Windows` 分配，但操作系统分配的内存通常会多于实际需要。

因此，在获取自定义数据格式时，取回的字节数组所包含的字节数至少与由源实际放到 `DataObject` 对象中的字节个数一样多，很可能比此数多。在从 `DataObject` 对象中取回自定义数据格式时必须正确解释这一格式。例如，必须在简单字符串中搜索 `NULL` 字符并在此字符所在位置将字符串截断。

从 `Windows` “资源管理器” 中拖动文件

可在 `Windows` “资源管理器” 和合适的 `Visual Basic` 控件之间使用 `OLE` 拖放来拖动文件。例如，可在 `Windows` “资源管理器” 中选定一组文本文件，

然后将它们拖放到一个文本框控件中就可将文本全部打开。

为说明这一点，以下过程可用一个文本框控件以及 `OLEDragOver` 和 `OLEDragDrop` 事件，并用 `DataObject` 对象中的 `Files` 属性和 `vbCFFiles` 数据格式打开一组文本文件。

从 `Windows` 资源管理器中拖动文本文件到文本框控件

92- - 在 `Visual Basic` 中启动新的工程。

92- - 向窗体添加一个文本框控件并将其 `OLEDropMode` 属性设置为“手工”。将 `MultiLine` 属性设置为 `True` 并清除 `Text` 属性。

92- - 添加函数，选定一组文件。例如：

```
Sub DropFile(ByVal txt As TextBox, ByVal strFN$)
```

```
    Dim iFile As Integer
```

```
    iFile = FreeFile
```

```
    Open strFN For Input Access Read Lock Read _
```

```
Write As #iFile
```

```
    Dim Str$, strLine$
```

```
    While Not EOF(iFile) And Len(Str) <= 32000
```

```
        Line Input #iFile, strLine$
```

```
        If Str <> "" Then Str = Str & vbCrLf
```

```
        Str = Str & strLine
```

```
    Wend
```

```
    Close #iFile
```

```
txt.SelStart = Len(txt)
```

```
txt.SelLength = 0
```

```
txt.SelText = Str
```

```
End Sub
```

92- - 将以下过程添加到 OLEDragOver 事件中。用 GetFormat 方法检测兼容的数据格式 (vbCFFiles)。

```
Private Sub Text1_OLEDragOver(Data As _  
VB.DataObject, Effect As Long, Button As Integer, _  
Shift As Integer, X As Single, Y As Single, State _  
As Integer)
```

```
    If Data.GetFormat(vbCFFiles) Then
```

```
        ‘若数据格式正确，_
```

则将即将执行的操作通知源

```
        Effect = vbDropEffectCopy And Effect
```

```
        Exit Sub
```

```
    End If
```

```
    ‘若数据格式不合适，则不放下
```

```
    Effect = vbDropEffectNone
```

```
End Sub
```

92- - 最后将下列过程添加到 OLEDragDrop 事件中。

```
Private Sub Text1_OLEDragDrop(Data As _  
VB.DataObject, Effect As Long, Button As Integer, _  
Shift As Integer, X As Single, Y As Single)  
    If Data.GetFormat(vbCFFiles) Then  
        Dim vFN  
  
        For Each vFN In Data.Files  
            DropFile Text1, vFN  
        Next vFN  
    End If  
End Sub
```

92- - 运行应用程序，打开 Windows “资源管理器”，突出显示若干文本文件并将它们拖动到文本框控件中。在文本框中将打开每个文本文件。

自定义鼠标指针

可用 MousePointer 和 MouseIcon 属性显示自定义图标、光标或任意定义过的鼠标指针。鼠标指针的改变可以告知用户诸多信息，例如，正在进行长时间的后台任务，调整某个控件或窗口的大小，某控件不支持拖放操作等等。可用自定义图标或鼠标指针表达无穷多个有关应用程序状态和功能

的视觉信息。

可用 `MousePointer` 属性在十六个预定义指针中任选一个。这些指针表示各种系统事件和过程。下表描述了几种指针及其在应用程序中的可能作用。

鼠标指针	常数	描述
	<code>vbHourglass</code>	警告用户程序状态在改变。例如，显示沙漏是通知用户等待
	<code>VbSizePointer</code>	通知用户功能在改变。例如，双箭头型调整指针告知用户此时可以调整窗口的大小
	<code>VbNoDrop</code>	警告用户无法执行某操作。例如，不放下指针告知用户不能在此位置放下文件

每个指针选项均由一个整型设置值表示。缺省设置值为 `0-Default` 并显示成标准的 `Windows` 箭头指针。但是，此设置由操作系统控制，如果用户改变系统指针箭头，则会改变设置值。为在应用程序中控制鼠标指针，应将 `MousePointer` 属性设置为合适的数值。

选定控件或窗体的 `MousePointer` 属性并扫描下拉设置值列表，或者使用“对象浏览器”并搜索 `MousePointerConstants`，通过这两种方式就可使用鼠标指针的完全列表。

在设置控件的 `MousePointer` 属性而且鼠标经过此控件时，指针就会出现。在设置窗体的 `MousePointer` 属性而且鼠标经过窗体的空白区域或经过 `MousePointer` 属性为 `0-Default` 的控件时，选定的指针都会出现。运行时，可用整型数值或 Visual Basic 鼠标指针常数设置鼠标指针值。例如：

```
Form1.MousePointer = 11 '或 vbHourglass
```

详细信息 关于全部鼠标指针常数的列表，请参阅《Visual Basic 6.0 语言参考手册》的“`MousePointer` 常数”。

图标和光标

可设置鼠标指针来显示自定义图标或光标。用自定义图标或光标可进一步改变应用程序的外观和功能。图标就是 `.ico` 文件，与 Visual Basic 的文件相同。光标就是 `.cur` 文件，在本质上象图标一样是位图。但是创建光标主要是为了显示由鼠标引发的操作发生的位置——它们可表示鼠标的状态及当前的输入位置。

光标中还包含热点信息。热点是跟踪光标位置——`x` 和 `y` 坐标——的像素。热点通常位于光标的中央。在用 `MouseIcon` 属性将图标加载到 Visual Basic 后，Visual Basic 把它们转换成光标格式并将热点设置成中央像素。两者不同点是，`.cur` 文件的热点位置可以改变，而 `.ico` 文件的热点位置不能改变。可在 Windows SDK 提供的“Image Editor”中编辑光标文件。为使用自定义图标或光标，应设置 `MousePointer` 和 `MouseIcon` 属性。

使用 .ico 文件作鼠标指针

92- - 选定一个窗体或控件并将其 `MousePointer` 属性设置为 `99-Custom`。

92- - 将 .ico 文件加载到 `MouseIcon` 属性中。例如，对于窗体：

```
Form1.MouseIcon = LoadPicture("c:\Program _  
Files\Microsoft Visual _  
Basic\Icons\Computer\Disk04.ico")
```

为把图标显示成鼠标指针，必须正确设置这两个属性。在将 `MousePointer` 属性设置成 `99-Custom` 时，如果未在 `MouseIcon` 上加载图标，则使用缺省的鼠标指针。同样，如果未将 `MousePointer` 属性设置成 `99-Custom`，则将忽略 `MouseIcon` 的设置。

注意 Visual Basic 不支持动画光标 (.ani) 文件。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`MouseIcon` 属性”及“`MousePointer` 属性”。

响应键盘事件

键盘事件和鼠标事件都是用户与程序之间交互操作中的主要元素。单击鼠标和按下按键都可触发事件，而且还提供进行数据输入的手段以及在窗口和菜单中移动的基本形式。

尽管操作系统为这些操作提供了无懈可击的后端，但有时也需要修改或增

强它们的功能。而使用 `KeyPress`、`KeyUp` 和 `KeyDown` 事件就可做必要的修改并增强功能。

可以把编写响应击键事件的应用程序看作是编写键盘处理器。键盘处理器可在控件级和窗体级这两个层次上工作。有了控件级（低级）处理器就可对特定控件编程。例如，可能希望将 `Textbox` 这个控件中的输入文本都转换成大写字符。而有了窗体级处理器就可使窗体首先响应击键事件。于是就可将焦点转换成窗体的控件并重复或启动事件。

通过击键事件可在标准键盘编写代码来处理大多数按键。关于处理区域字符集和键盘的详细信息，请参阅第十六章“国际化”。

编写低级键盘处理程序

`Visual Basic` 提供三种事件，而且窗体和接受键盘输入的控件都识别这三种事件。下表对这些事件都作了描述。

键盘事件	触发原因
<code>KeyPress</code>	按下对应某 <code>ASCII</code> 字符的键
<code>KeyDown</code>	按下键盘的任意键
<code>KeyUp</code>	释放键盘的任意键

只有获得焦点的对象才能够接受键盘事件。对于键盘事件，只有当窗体为活动窗体且其上所有控件均未获得焦点时，窗体才获得焦点。这种情况只有在空窗体和窗体上的控件都无效时才发生。但是，如果将窗体上的

`KeyPreview` 属性设置为 `True`，则对每个控件在控件识别其所有键盘事件之前，窗体就会接受这些键盘事件。当您希望无论何时按下某按键都会执行同一个操作，而不管哪个控件在此时具有焦点时，这样设置 `KeyPreview` 属性极为有用。

`KeyDown` 和 `KeyUp` 事件提供了最低级的键盘响应。例如，可用这些事件检测 `KeyPress` 事件无法检测到的情况：

- `SHIFT`、`CTRL` 和 `ALT` 键的特殊组合。
- 箭头键。注意，某些控件（命令按钮、选项按钮和复选框）不接受箭头键事件：相反，按下箭头键后将使焦点移动到另一个控件。
- `PAGEUP` 和 `PAGEDOWN`。
- 区分数值小键盘的数字键与打字键盘的数值键。
- 不仅响应按键操作而且响应释放键的操作（`KeyPress` 只响应按键操作）。
- 与菜单命令无联系的功能键。

`Keyboard` 事件彼此并不相互排斥。按下一键时将生成 `KeyDown` 和 `KeyPress` 事件，而松开此键后生成 `KeyUp` 事件。当用户按下一个 `KeyPress` 不能检测的键时将触发 `KeyDown` 事件，而松开此键后时生成 `KeyUp` 事件。

使用 `KeyUp` 和 `KeyDown` 事件之前要确保 `KeyPress` 事件的功能不够使用。`KeyPress` 事件检测的键对应于所有标准 ASCII 字符：`ENTER`、`TAB`、`BACKSPACE` 键以及标准键盘的字母、数字和标点符号键。通常，编写 `KeyPress` 事件的代码比较容易。

还应考虑使用快捷键和访问键，第三章“窗体、控件和菜单”中的“菜单

基础”一节对此进行说明。快捷键必须与菜单命令相联系，但它们可包括功能键（包括某些功能键与 shift 键的组合）。分配快捷键时无需编写附加的代码。

注意 Windows ANSI 字符集对应 256 个字符，包括标准拉丁字母、出版符（例如版权标志、em 虚线和省略号）以及许多替换字符和重音符号字符。这些字符由唯一的一字节数值 (0-255) 表示。ASCII 本质上是 ANSI 字符集的一个子集 (0-127)，代表标准键盘上的标准字母、数字和标点符号。在提及这两种字符集时可将它们互换。

KeyPress 事件

在按下与 ASCII 字符对应的键时将触发 KeyPress 事件。ASCII 字符集不仅代表标准键盘的字母、数字和标点符号，而且也代表大多数控制键。但是 KeyPress 事件只识别 ENTER、TAB 和 BACKSPACE 键。KeyDown 和 KeyUp 事件能够检测其它功能键、编辑键和定位键。

无论何时要处理标准 ASCII 字符都应使用 KeyPress 事件。例如，如果希望将文本框中的所有字符都强制转换为大写字符，则可在输入时使用此事件转换大小写：

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
    KeyAscii = Asc(Ucase(Chr(KeyAscii)))
End Sub
```

keyascii 参数返回对应于 ASCII 字符代码的整型数值。上述过程用 *Chr* 将 ASCII 字符代码转换成对应的字符，然后用 *Ucase* 将字符转换为大写，并用 *Asc* 将结果转换回字符代码。

可用相同的 ASCII 字符代码检测，是否可通过按下 *KeyPress* 事件识别一个键。例如，下述事件过程使用 *KeyPress* 检测用户是否正在按 *BACKSPACE* 键：

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
    If KeyAscii = 8 Then MsgBox "You pressed the _
        BACKSPACE key."
End Sub
```

也可用 Visual Basic 的键代码常数代替字符代码。示例中的 *BACKSPACE* 键的 ASCII 值为 8，其常数值为 *vbKeyBack*。

详细信息 关于字符代码的完整列表，请参阅《Visual Basic 6.0 语言参考手册》的“字符集 (0-127)”和“字符集 (128-255)”。“键代码常数”提供了键代码常数与对应的 ASCII 值的完整列表，也可通过“对象浏览器”并搜索 *KeyCodeConstants* 获得此列表。

还可用 *KeyPress* 事件改变某些键的缺省行为。例如，当窗体上没有缺省按钮时，按 *ENTER* 键就会发出嘟嘟声。在 *KeyPress* 事件中截断 *ENTER* 键（字符代码 13）就可避免发声。

```
Private Sub Text1_KeyPress (KeyAscii As Integer)
    If KeyAscii = 13 Then KeyAscii = 0
End Sub
```

KeyDown 和 KeyUp 事件

KeyUp 和 KeyDown 事件报告键盘本身准确的物理状态：按下键(KeyDown)及松开键(KeyUp)。与此成对照的是，KeyPress 事件并不直接地报告键盘状态——它只提供键所代表的字符而不识别键的按下或松开状态。

进一步举例将有助于阐明这一差别。输入大写“A”时，KeyDown 事件获得“A”的 ASCII 码。在输入小写“a”时，KeyDown 事件获得相同的 ASCII 代码。为区分大小写，这些事件使用 *shift* 参数。与此相对照的是，KeyPress 事件将字母的大小写作为两个不同的 ASCII 字符处理。

KeyDown 和 KeyUp 事件通过提供下列两参数返回输入字符的信息。

参数	描述
<i>Keycode</i>	指示按下的物理键。这时将“A”与“a”作为同一个键返回。它们具有相同的 <i>keycode</i> 值。但是请注意，键盘上的“1”和数字小键盘的“1”被作为不同的键返回，尽管它们生成相同的字符
<i>Shift</i>	指示 SHIFT、CTRL 和 ALT 键的状态。只有检查此参数才能判断输入的是大写字母还是小写字母

Keycode 参数

keycode 参数通过 ASCII 值或键代码常数来识别键。字母键的键代码与此字母的大写字符的 ASCII 值相同。所以“A”和“a”的 *keycode* 都是由 Asc(“A”)返回的数值。在下例中用 KeyDown 事件判断是否按下了

“ A ” 键：

```
Private Sub Text1_KeyDown(KeyCode As Integer, _  
    Shift As Integer)  
    If KeyCode = vbKeyA Then MsgBox "You pressed _  
        the A key."  
End Sub
```

按下 SHIFT + “A” 或只按下 “A” 后都将显示消息框——也就是说，对每种情况，参数都是正确的。为判断按下的字母是大写形式还是小写形式需使用 *shift* 参数。请参阅本章后面的“Shift 参数”主题。

数字与标点符号键的键代码与键上数字的 ASCII 代码相同。因此“1”和“！”的 *keycode* 都是由 *Asc*(“1”) 返回的数值。同样，为检测“！”，需使用 *shift* 参数。

KeyDown 和 *KeyUp* 事件可识别标准键盘上的大多数控制键。其中包括功能键 (F1-F16)、编辑键 (HOME、PAGE UP、DELETE 等等)、定位键 (RIGHT、LEFT、UP 和 DOWN ARROW) 和数字小键盘上的键。可以通过键代码常数或相应的 ASCII 值检测这些键。例如：

```
Private Sub Text1_KeyDown(KeyCode As Integer, _  
    Shift As Integer)  
    If KeyCode = vbKeyHome Then MsgBox "You _  
        pressed the HOME key."  
End Sub
```

详细信息 关于字符代码的完整列表，请参阅《Visual Basic 6.0 语言参考手

册》的“字符集 (0-127)”和“字符集 (128-255)”。 “键代码常数”提供了键代码常数与对应的 ASCII 值的完整列表，也可通过“对象浏览器”并搜索 `KeyCodeConstants` 获得此列表。

Shift 参数

键盘事件使用 *shift* 参数的方式与鼠标事件所用方式相同——将它作为代表 `SHIFT`、`CTRL` 和 `ALT` 键的整数值或常数。可将 `KeyDown` 与 `KeyUp` 事件及 *shift* 参数一同使用以区分字符的大小写或检测多种鼠标状态。以上述示例为基础，可用 *shift* 参数判断是否按下了字母的大写形式。

```
Private Sub Text1_KeyDown(KeyCode As Integer, _  
    Shift As Integer)  
    If KeyCode = vbKeyA And Shift = 1 _  
        Then MsgBox "You pressed the uppercase A key."  
End Sub
```

与鼠标事件相似，`KeyUp` 和 `KeyDown` 事件可将 `SHIFT`、`CTRL` 和 `ALT` 键作为单个个体来检测，也可作为组合键检测。下列示例检测特定的 *shift* 键的状态。

打开一个新的工程并将变量 `ShiftKey` 添加到窗体的声明部分中：

```
Dim ShiftKey as Integer
```

将 `Textbox` 控件添加到窗体上并将此过程添加到 `KeyDown` 事件中：

```
Private Sub Text1_KeyDown(KeyCode As Integer, _
```



```
        Shift As Integer)
ShiftKey = Shift And 7
Select Case ShiftKey
    Case 1 '或 vbShiftMask
        Print "You pressed the SHIFT key."
    Case 2 '或 vbCtrlMask
        Print "You pressed the CTRL key."
    Case 4 '或 vbAltMask
        Print "You pressed the ALT key."
    Case 3
        Print "You pressed both SHIFT and CTRL."
    Case 5
        Print "You pressed both SHIFT and ALT."
    Case 6
        Print "You pressed both CTRL and ALT."
    Case 7
        Print "You pressed SHIFT, CTRL, and ALT."
End Select
End Sub
```

只要 Textbox 控件获得焦点，在按下每个键或键的每个组合时，键或键的组合都将在窗体上显示相应的信息。

详细信息 请参阅本章前面的“检测 SHIFT、CTRL 和 ALT 键的状态”。

编写窗体级键盘处理程序

每个 `KeyDown` 和 `KeyUp` 事件都附加在特定的对象上。为了编写应用于窗体上所有对象的键盘处理器，应将窗体的 `KeyPreview` 属性设置为 `True`。将 `KeyPreview` 属性设置为 `True` 时，对所有控件，在控件识别 `KeyPress`、`KeyUp`、`KeyDown` 事件之前，窗体会识别控件的这些事件。这就使它很容易对具体击键事件作出一般响应。

可在“属性”窗口中或通过 `Form_Load` 过程中的代码设置窗体的 `KeyPreview` 属性为 `True`：

```
Private Sub Form_Load
    Form1.KeyPreview = True
End Sub
```

通过声明 `ShiftKey` 变量并使用 `Select Case` 语句可检测窗体的多种击键状态。无论哪个控件获得焦点，以下过程都将在窗体上显示信息。

打开一个新工程并将变量 `ShiftKey` 添加到窗体的声明部分：

```
Dim ShiftKey as Integer
```

将 `Textbox` 控件和 `CommandButton` 控件添加到窗体中并将以下过程添加到窗体的 `KeyDown` 事件中：

```
Private Sub Form_KeyDown(KeyCode As Integer, _
    Shift As Integer)
    ShiftKey = Shift And 7
```

```
Select Case ShiftKey
    Case 1 '或 vbShiftMask
        Print "You pressed the SHIFT key."
    Case 2 '或 vbCtrlMask
        Print "You pressed the CTRL key."
    Case 4 '或 vbAltMask
        Print "You pressed the ALT key."
End Select
End Sub
```

如果已为菜单控件定义快捷键，那么，当按下该键时会自动触发菜单控件的 Click 事件而不是键事件。

同样，如果在窗体上有一个命令按钮，其 Default 属性被设置为 True，则 ENTER 键将触发此命令按钮的 Click 事件而不是键事件。如果将命令按钮的 Cancel 属性设置为 True，则 ESC 键将触发按钮的 Click 事件而不是键事件。

例如，如果将 Click 事件过程添加到 CommandButton，然后将 Default 或 Cancel 属性设置为 True，则按下 RETURN 或 ESC 键时，KeyDown 事件会遭到拒绝。以下过程关闭应用程序：

```
Private Sub Command1_Click()
    End
End Sub
```

注意，除非窗体上的每个控件都无效或其 TabStop 属性都为 False，否则，

TAB 键会将焦点从一个控件移动到另一个控件而不触发击键事件。当把窗体的 KeyPreview 属性设置为 True 时，窗体在控件之前识别键盘事件，但对控件来说事件仍然会发生。为防止这种情况，可将窗体键盘事件过程中的 *keyascii* 或 *keycode* 设置为 0。例如，如果在窗体上没有缺省按钮，则可用 ENTER 键将焦点从一个控件移动到另一个控件：

```
Private Sub Form_KeyPress (KeyAscii As Integer)
    Dim NextTabIndex As Integer, I As Integer
    If KeyAscii = 13 Then
        If Screen.ActiveControl.TabIndex = _
            Count - 1 Then
            NextTabIndex = 0
        Else
            NextTabIndex = Screen.ActiveControl._
                TabIndex + 1
        End If
        For I = 0 To Count - 1
            If Me.Controls(I).TabIndex = _
                NextTabIndex Then
                Me.Controls(I).SetFocus
            Exit For
        End If
    Next I
```

```
KeyAscii = 0
```

```
End If
```

```
End Sub
```

因为在 *keyascii* 等于 13 时，代码将其设置为 0，所以控件从不识别按下的 ENTER 键，而控件的键盘事件也从不被调用。

中断后台处理

应用程序为完成某些任务会进行长时间的后台处理。如果出现这种情况，向用户提供一种方法，使用户得以切换到其它应用程序或者中断或取消后台任务，这将是很有益的。Windows 操作系统环境提供了第一种选择：例如，可用 ALT+TAB 组合键切换到其它应用程序。还可编写程序，当用户单击取消按钮或 ESC 键时，程序作出响应，这样一来便提供了其它选择。在考虑如何在应用程序中实现这一点时，理解操作系统如何处理各种应用程序中的任务显得尤其重要。Windows 是抢先式多任务操作系统，这意味着后台任务可有效地共享空闲的处理器时间。这些后台任务可能来自正在使用的应用程序，也可能来自其它应用程序，或许还可能来自某些系统控制的事件。但是正在使用的应用程序总是处在优先的地位。这就保证鼠标和键盘总是直接作出响应。

后台处理可划分成两类：不变的和断续的。从服务器复制文件是不变任务的例子，而定期更新数值则是断续任务的例子。两种类型的任务都可被中

断或取消。但是，由于后台处理往往很复杂，因此，首先考虑如何启动这些任务是非常重要的。本章后面的主题“允许用户中断任务”描述了需要考虑的问题和实现技术。

允许用户中断任务

在长时间后台任务运行期间，应用程序无法响应用户的输入。因此应为鼠标事件或键盘事件编写代码，使用户能够中断或取消后台处理。例如，在长时间后台任务运行时可显示一个带“取消”按钮的对话框，用户可按下 ENTER 键（如果“取消”按钮获得焦点），也可用鼠标单击按钮来启动“取消”按钮。

注意 有时可能希望处理任务时提供某种视觉线索。例如，（用 Label 或 Gauge 控件）显示任务的进展状况，或者将鼠标指针变成沙漏来显示这种线索。

有几种技术而不是一种方法来编写操作后台处理的代码。一种中断任务的方法是显示“取消”按钮并处理其 Click 事件。为此，根据下述原则，可将后台任务代码放入 timer 事件中。

- 用静态变量存储 Timer 事件过程中必须一直保存的变量。
- 当 Timer 事件获得控制后使运行的时间稍长于为 Interval 属性指定的时间。这将确保后台任务能够利用系统分配的每一时刻的处理器时间。直到这个事件结束处理之前，下一个 Timer 事件都在消息队列中等待。
- 为 timer 的 Interval 属性设置较大的数值——5 到 10 秒，这将使处理

更为有效。抢先式多任务能够防止发生其它应用程序被阻塞，在取消长时间任务时，用户通常也容忍短暂的延迟。

- 用 Timer 控件的 Enabled 属性作为标志，防止启动已经运行的后台任务。

详细信息 请参阅第七章，“使用 Visual Basic 的标准控件”中的“使用 Timer 控件”。

使用 DoEvents

尽管 Timer 事件是后台处理的最好工具，对耗时极多的任务，情况更是如此，但是，DoEvents 函数还是提供了一种取消任务的简便方法。例如，下列代码将显示一个“Process”按钮，单击这个按钮时，它将变成“Cancel”按钮。再次单击按钮又将中断正在执行的任务。

此按钮最初的标题是“Process”。

```
Private Sub Command1_Click()  
    '过程的所有实例都  
    '共享静态变量。  
    Static blnProcessing As Boolean  
    Dim lngCt As Long  
    Dim intYieldCt As Integer  
    Dim dblDummy As Double  
    '按下按钮时，检测是否
```

‘已经在处理。

If blnProcessing Then

‘如果正在处理，则取消处理。

BlnProcessing = False

Else

Command1.Caption = “Cancel”

blnProcessing = True

lngCt = 0

‘执行一百万次浮点

‘乘法计算。每一千次后，

检测是否要取消。

Do While blnProcessing And (lngCt < 1000000)

For intYieldCt = 1 To 1000

lngCt = lngCt + 1

dblDummy = lngCt * 3.14159

Next intYieldCt

‘DoEvents 语句允许其它事件

发生，包括第二次

‘按此按钮。

DoEvents

Loop

blnProcessing = False


```
Command1.Caption = "Process"  
MsgBox lngCt & " multiplications were performed"
```

```
End If
```

```
End Sub
```

DoEvents 将控制切换到操作环境内核。只要此环境中的所有应用程序都有机会响应待处理事件，应用程序就又恢复控制。这不会使应用程序放弃焦点，但会使后台事件能够得到处理。

这种妥协的结果可能并不总是达到预期目标。例如，下述 **Click** 事件代码在单击按钮后要一直等候十秒钟，而后才显示一条信息。如果在按钮正在等待期间单击它，则将以相反顺序完成单击操作。

```
Private Sub Command2_Click()  
    Static intClick As Integer  
    Dim intClickNumber As Integer  
    Dim dblEndTime As Double  
        '每次单击按钮时  
        '赋予唯一数值。  
    IntClick = intClick + 1  
    intClickNumber = intClick  
        '等待十秒。  
    DblEndTime = Timer + 10#  
    Do While dblEndTime > Timer  
        '不做任何事情，仅仅允许
```

‘其它应用程序处理
它们的事件。

DoEvents

Loop

MsgBox “Click “ & intClickNumber & “ is finished”

End Sub

对于通过 DoEvents 放弃控制的事件过程，有时可能希望防止在 DoEvents 返回之前重新调用这一过程。否则将无穷无尽地调用该过程，直到系统资源消耗殆尽。可暂时禁止控件，或象上例一样，使用一个静态的“标志”变量防止此事发生。

在使用全局数据时避免 DoEvents

当一个函数已通过 DoEvents 放弃控制时，可相当安全地再次调用函数。例如，下一过程将检测质数并用 DoEvents 语句周期地启动其它应用程序处理事件：

Function PrimeStatus (TestVal As Long) As Integer

Dim Lim As Integer

PrimeStatus = True

Lim = Sqr(TestVal)

For I = 2 To Lim

If TestVal Mod I = 0 Then

```
        PrimeStatus = False
    Exit For
End If
If I Mod 200 = 0 Then DoEvents
Next I
End Function
```

该代码中每重复 200 次就调用一次 DoEvents 语句。这样一来，当该环境的其余部分对事件作出响应时，只要有必要，PrimeStatus 过程就可继续计算。

考虑在调用 DoEvents 期间发生的事情。在其它窗体和应用程序处理事件时将暂停执行应用程序代码。这些事件之一有可能是一个按钮单击操作，它将再次启动 PrimeStatus 过程。

这将导致重新进入 PrimeStatus 过程的，但是，因为在函数每次出现时，堆栈都为其参数和局部变量分配了空间，所以重入不会引发冲突。当然，如果过多调用 PrimeStatus，则可能出现“溢出堆栈空间”错误。

如果 PrimeStatus 使用或改变模块级变量或全局数据，情况就会完全不同。此时，在 DoEvents 能够返回之前执行 PrimeStatus 的另一个实例，这将导致模块数据或全局数据的值完全不同于它们在调用 DoEvents 之前的值。于是，PrimeStatus 的结果将会难以预料。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“DoEvents 函数”和“Refresh 方法”。

使用文本和图形

Visual Basic 为应用程序的使用，提供了复杂的文本和图形功能。如果能将文本当作可视化的元素，则大小、形状和颜色都可以看到，这就能增强所提供的信息。正如报纸要使用标题、专栏和要点来把文字分成互相穿插的一些块一样，文本的一些属性也可以用于强调重要的概念和感兴趣的细节。Visual Basic 在设计时还提供一些有很大灵活性的图形功能，包括添加通过显示一连串图像所实现的动画。

本章阐述了放置和操作文本图形的一些方法。包括有关格式设置、字体、调色板和打印的细节。通过良好的设计思路来运用这些功能，可以优化应用程序，使之有吸引力和易于使用。

目录

- 使用字体
- 在窗体和图片框中显示文本
- 格式化数字、日期和时间
- 使用选定的文本
- 用剪贴板对象传送文本和图形
- 坐标系统概述
- 使用图形控件

- 使用图形方法
- 使用颜色
- 使用图片对象
- 打印

示例应用程序：Blanker.vbp, Palettes.vbp

本章的一些代码示例取之于 **Blanker** (Blanker.vbp) 和 **Palettes** (Palettes.vbp) 示例。您可以在 `Samples!Alink("vbsamples")` 目录中找到这些示例应用程序。

使用字体

文本是使用字体来显示的——字体是具有相同铅字字样的字符集，能够以特定的大小、风格和粗细来使用。

Windows 95 和 **Windows NT** 操作系统提供了一套完整的基本字体。**TrueType®** 字体是可缩放的，也就是说它们能产生任何大小的字符。当选择了一种 **TrueType** 字体，它将变换成选定的磅数大小，并以位图显示在屏幕上。

在打印时，选定的 **TrueType** 字体或其它字体，都会变换成恰当的大小，然后发送给打印机。因此，不需要单独的屏幕和打印字体。然而，如果有一

种可用的、能提高打印速度的、等效的字体时，打印字体仍将被替换为 TrueType 字体。

为应用程序选择字体

要记住，应用程序的用户可能没有创建应用程序所使用的字体。如果选择了用户没有的 TrueType 字体，则 Windows 将会从用户系统中选择最接近匹配的字体。这样可能会给用户带来一些问题，要由应用程序的设计而定。例如，Windows 选择的字体可能会放大文本，以至屏幕上的标签有所重叠。避免字体问题的一个方法是：与应用程序一起分布必要的各种字体（为了与应用程序一起分布，可能要从版权所有者那里获得字体的许可）。

也可以设计应用程序去检查操作系统提供的字体中是否有所用到的字体。如果所需字体没有驻留在操作系统中，可以设计应用程序从所提供的字体列表中选择一种不同的字体。

避免字体问题的另一种方法是：使用大部分用户的操作系统都有的字体。如果使用了 Windows 特定版本的字体，要把该版本指定为应用程序的系统需求。

检查可用的字体

程序能够容易地检测出用户系统和打印机中是否有可用的匹配字体。Fonts 属性适用于 Printer 和 Screen 对象。被 Fonts 属性所返回的数组，是打印机或屏幕可用的所有字体的列表。可以循环地通过该属性数组，查找相匹

配的名称字符串。以下代码示例，将确定系统中是否有与所选定的窗体相匹配的打印机字体：

```
Private Sub Form_Click ()
    Dim I As Integer, Flag As Boolean
    For I = 0 To Printer.FontCount - 1
        Flag = StrComp (Font.Name,Printer.Fonts(I), 1)
        If Flag = True Then
            Debug.Print "There is a matching font."
            Exit For
        End If
    Next I
End Sub
```

详细信息 关于东亚系统字体的详细信息，请参阅第十六章“国际化”中的“DBCS 环境下字体、显示和打印的考虑因素”。有关设置字体属性的信息，请参阅本章后面的“设置字体特征”。

创建您自己的字体类型

如果您使用“引用”对话框设置一个对标准 OLE 类型的引用，那么就可以使用 StdFont 类来创建您自己的字体类型。如果浏览“对象浏览器”，您将会注意到有 StdFont 和 Font 类。Font 类是由 StdFont 基类推导出的，并且被所有的控件支持。

您可以使用以下语法：

```
Dim MyFont As Font
```

但是，您不能使用：

```
Dim MyFont As New Font
```

相反，要创建您自己的字体或图片类型，请使用下面的代码：

```
Dim MyFont As New StdFont
```

```
With MyFont
```

```
    .Bold = True
```

```
    .Name = "Arial"
```

```
End With
```

```
Set Text1.Font = MyFont
```

设置字体特征

显示文本（作为文本或标题）的窗体、控件，以及支持 **Font** 属性的 **Printer** 对象，它们将确定文本的可视特征，包括：

- 字体名（字样）
- 字体大小（用磅表示）
- 特别特征（黑体、斜体、下划线或删除线）

有关 **Printer** 对象的详细信息，请参阅本章后面的“从应用程序中打印”。

设置字体属性

在设计时，通过双击“属性”窗口中的“字体”，可以任意设置字体属性，然后在“字体”对话框中可进行属性的设定。

在运行时，通过设置各个窗体和控件的 **Font** 对象的属性，可以设定字体的特征。下表说明了 **Font** 对象的一些属性。

属性	类型	描述
Name	String	指定字体的名字，例如 Arial 或 Courier
Size	Single	用磅为单位来指定字体的大小（在打印时每英寸 72 磅）
Bold	Boolean	如果为 True，文本为黑体
Italic	Boolean	如果为 True，文本为斜体
StrikeThrough	Boolean	如果为 True，在文本中画一条删除线
Underline	Boolean	如果为 True，在文本中添加下划线
Weight	Integer	返回或设置字体的粗细。确定以上的粗细，Bold 属性将被强制为 True

例如，下列语句是给名称为 **lblYearToDate** 的标签设置各种字体属性：

With **lblYearToDate.Font**

 .Name = "Arial" ‘把字体改为 Arial。

 .Bold = True ‘把字体改为 bold。

End With

选择字体属性的顺序是很重要的，因为并不是所有的字体都支持所有字体的变动。首先应设置 **Name** 属性。然后设置任何一个 **Boolean** 属性，如 **Bold**

和 `Italic` 属性，为 `True` 或为 `False`。

也可以在 `Font` 对象中存储一组字体属性。使用 `StdFont` 类，象其它任何对象一样，可以声明 `Font` 对象：

```
Dim MyFont As New StdFont
```

```
With MyFont
```

```
    .Name = "Arial"
```

```
    .Size = 10
```

```
    .Bold = True
```

```
End With
```

注意 在创建新的 `Font` 对象之前，必须使用“引用”对话框（可从“工程”菜单中取得），先创建一个对标准 `OLE` 类型的引用。

通过将窗体或控件的 `Font` 对象设置为新的对象，能够容易地从一组字体属性切换到另一组：

```
Set lblYearToDate.Font = MyFont
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`Font` 对象”。

使用小字体

某些字体不允许小于 8 磅。对于这样的字体，如果将它们的 `Size` 属性设置为小于 8 磅的尺寸，那么无论是 `Name` 属性还是 `Size` 属性，都会自动变为其它字体或其它尺寸。为了避免不是预期的结果，每次设置字体尺寸小于 8 磅的 `Size` 属性时，在设置好以后，一定要再检查一次 `Name` 属性和 `Size` 属性。

把 Font 属性应用到特定的对象

设置字体属性产生的效果随着显示文本技术的不同而不同。如果文本是由一个属性指定（例如 `Text` 或 `Caption`），则一个字体属性的改变将适用于此控件的所有文本。标签、文本框、框架、按钮、复选框和所有文件系统的控件，都是用一个属性来指定文本的。

如果应用程序使用 `Print` 方法显示文本，则字体属性的改变，对属性改变之后使用 `Print` 的所有文本都有影响。而对属性改变之前所打印的文本，没有影响。只有窗体、图片框以及 `Debug` 和 `Printer` 对象支持 `Print` 方法。因为字体属性的改变适用于文本框和标签中的所有文本，所以在这些控件中不能使用混合字体。如果需要混合字体（例如，使一些词用黑体，而其它的用正常字体）时，则应创建图片框，使用 `Print` 方法来显示文本。“在窗体和图片框上显示文本”中说明了 `Print` 方法的使用问题。

FontTransparent 属性

窗体和图片框具有另一个字体属性，即 `FontTransparent`。当 `FontTransparent` 为 `True` 时，背景将透过显示在窗体或图片框上的文本而显示出来。图 12.1 显示出了 `FontTransparent` 属性的效果。



Font Transparent 设为 True



Font Transparent 设为 False

图 12.1 FontTransparent 属性的使用效果

在窗体和图片框上显示文本

为了在窗体或图片框上显示文本，应使用 `Print` 方法，将该窗体或图片框的名称加在它的前面即可。为了把输出文本发送到打印机上，应使用 `Printer` 对象上的 `Print` 方法。

使用 `Print` 方法

`Print` 方法的语法是：

```
[object.]Print [outputlist] [{ ; | , }]
```

object 参数是可选项；如果被省略，则 **Print** 方法将应用于当前窗体。

例如，以下语句将把消息打印到：

- 名为 **MyForm** 的窗体：

```
MyForm.Print "This is a form."
```

- 名为 **picMiniMsg** 的图片框：

```
picMiniMsg.Print "This is a picture box."
```

- 当前窗体：

```
Print "This is the current form."
```

- **Printer** 对象：

```
Printer.Print "This text is going to the printer."
```

Outputlist 参数是显示在窗体或图片框上的文本。如果 *outputlist* 参数有多个项，它们必须用逗号或分号，或逗号和分号同时使用来进行分隔，参见本章稍后“在单行上显示不同的项”一节中的说明。

文本的截断

如果窗体或图片框太小，以至不能显示所有的文本，则文本将被切断。在什么地方截断文本，取决于文本打印开始点的位置坐标。被截断的文本不能通过滚动窗体或图片框来显示。

分层

当将文本打印到窗体上时，文本将显示在窗体上已有的任一控件 `behind` 的那一层上。因此，最好的方法是特意创建一个用于显示文本的窗体，把文本打印到该窗体上。有关文本和图形如何显示在窗体的层次中的详细信息，请参阅本章后面的“用 `AutoRedraw` 和 `ClipControls` 使图形分层”。

在单行上显示不同的项

被显示或被打印的项，可包括属性值、常数和变量（字符串或数字）。“在窗体或图片框上显示文本”一节中讨论过的 `Print` 方法，可打印数字项的值。正数数值具有一个前导和一个尾部空格。负数数值用负数符号替代一个前导空格。

用分号 (;) 或逗号 (,) 来分隔相邻的项。如果用的是分号，`Visual Basic` 将一项接着一项地打印，中间没有空格。如果用的是逗号，在打印一项后，`Visual Basic` 将跳到下一个制表列打印下一项。

例如，下列语句打印到当前窗体：

```
Print "The value of X is "; X; "and the value of Y _  
      is "; Y
```

如果 `X` 的值为 2，`Y` 的值为 7，语句的输出如下：

```
The value of X is 2 and the value of Y is 7
```

在缺省情况下，每个 `Print` 方法都是打印完文本后自动移到下一行。如果没有项，`Print` 只是简单地跳到下一行。一系列的 `Print` 语句（在下述示例中，是用于名为 `picLineCount` 的图片框）自动地使用不同的行：

```
picLineCount.Print "This is line 1."
```

```
PicLineCount.Print "This is line 2."
```

如果把一个分号（或逗号）放在第一条语句的最后，第二条 `Print` 语句的输出将会和第一条在同一行上：

```
picLineCount.Print "This all appears ";
```

```
picLineCount.Print "on the same line."
```

在指定位置显示 `Print` 输出

用以下的一种或两种方法，指定绘图坐标，就可以控制 `Print` 输出的位置：

- 使用 `Cls`（清屏）方法来清除窗体或图片框，同时把绘图坐标恢复到原点（0，0）。
- 使用 `CurrentX` 和 `CurrentY` 属性，设置绘图坐标。

`Cls` 方法

对象中用 `Print` 和图形方法创建出的所有文本和图形，都可以用 `Cls` 方法来删除。同时，`Cls` 方法还把绘图坐标恢复到原点（0，0），按照缺省规定，原点是左上角。例如，这些语句将清除：

- 名为 `Picture1` 的图片框：

```
Picture1.Cls
```

- 当前窗体：

```
Cls
```

设置绘图坐标

可以直接用 `CurrentX` 和 `CurrentY` 属性来设置窗体和图片框的绘图坐标。例如，这些语句把 `Picture1` 和当前窗体的绘图坐标恢复到左上角：

- 名为 `Picture1` 的图片框：

```
Picture1.CurrentX = 0
```

```
Picture1.CurrentY = 0
```

- 当前窗体：

```
CurrentX = 0
```

```
CurrentY = 0
```

若该处已有文本和图形，则所打印的任何新文本将显示在它们的顶部。为了有选择地清除文本，应使用 `Line` 方法画一个框，然后用背景色对它填充。要注意的是，绘图坐标是由 `CurrentX` 和 `CurrentY` 指定的，当使用图形方法后，它通常会改变位置。

在缺省情况下，窗体和图片框使用每一单位对应一个缇的坐标系统（每英寸为 1，440 缇，近似地，每厘米为 567 缇）。因为文本的高度是用磅来度量的，有时需要把窗体、图片框和 `Printer` 对象的 `ScaleMode` 属性从缇

改为磅。对文本和要把文本打印其上的对象使用同一单位，使得对文本打印到位置的计算简单些。

详细信息 有关缇和绘图坐标的详细信息，请参阅本章后面的“坐标系统概述”。`TextHeight` 和 `TextWidth` 方法

在使用 `Print` 方法之前，可使用 `TextHeight` 和 `TextWidth` 方法确定 `CurrentX` 和 `CurrentY` 属性所指定的位置。`TextHeight` 所返回的文本行的高度，考虑了该对象的字体大小和风格。它的语法是：

```
[object.]TextHeight(string)
```

如果 `string` 参数包含嵌入的回车符 (`Chr(13)`)，则文本对应于多行，`TextHeight` 属性将返回字符串中所有行的文本的高度。如果没有嵌入的回车符，`TextHeight` 将返回一行文本的高度。

`TextHeight` 的使用方法是，把 `CurrentY` 属性设置到特定的行。例如，下述语句将绘图坐标设置到第五行的开始：

```
CurrentY = TextHeight("sample") * 4
```

```
CurrentX = 0
```

假设示例文本中没有回车符，用以下语法可将 `CurrentY` 设置到第 `n` 行：

```
CurrentY = [object.]TextHeight(string) * (n - 1)
```

如果 `object` 被省略了，该方法将应用于当前窗体。`Object` 参数可为窗体、图片框或 `Printer` 对象。

`TextWidth` 方法所返回的字符串宽度，考虑了该对象的字体大小和风格。因为许多字体具有均衡宽度的字符，本方法将会很有用。`TextWidth` 属性可用于确定字符串的宽度是否比窗体、图片框或 `Printer` 对象的宽度更大。

例如，以下语句通过 `CurrentX` 和 `CurrentY` 定位，用 `TextWidth` 和 `TextHeight` 把文本确定在方框的中心。此示例中该方框的名字是 `MealCard`。

```
CurrentX = (BoxWidth - TextWidth("MealCard")) / 2
```

```
CurrentY = (Boxheight - TextHeight("MealCard")) / 2
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`TextHeight` 方法”和“`TextWidth` 方法”。

格式化数字、日期和时间

在显示数字的格式上，象日期和时间的格式一样，`Visual Basic` 也提供了巨大的灵活性。对于数字、日期和时间，可以很容易用国际格式来显示。

`Format` 函数把数字值转换为文本字符串，从而能够对该字符串的外观进行控制。例如，可以指定小数的位数、前导和尾部零，以及货币格式。它的语法是：

```
Format(expression[, format[,firstdayofweek[,firstweekofyear]]])
```

expression 参数指定要转换的数值，*format* 参数是字符串，该串是由一些符号组成的，这些符号用于说明如何确定该数字的格式。下表列出了一些最常使用的一些符号。

符 号	描 述
0	数字保留区，如果恰当，在本位置打印尾部或前导零
#	数字保留区，不打印尾部或前导零
.	小数保留区
,	千位分隔符
- + \$ () space	字母字符，各种字符，都要按格式字符串中打入的原样，精确地显示出来

firstdayofweek 参数是指定本星期第一天的常数，而 *firstweekofyear* 参数是指定本年度第一天的常数。两者都是可选的。关于这些常数的更详细信息，请参阅《Visual Basic 6.0 语言参考手册》的“Format 函数”。

命名的格式

Visual Basic 提供了几种与 Format 函数一起使用的标准格式。在 Format 函数的 *format* 参数中，可使用名字来指定这些格式，而不用在 *format* 参数中指定符号。格式名总是用双引号 (“”) 括起来。

下面的表格列出了可使用的格式名。

命名的格式	描述
General Number	显示没有千位分隔符的数字
Currency	显示带千位分隔符的数字；在小数点的右边显示两位数字。而输出则依据用户的系统设置
Fixed	在小数点的左边至少显示一位数字，在小数点的右边显示两位数字
Standard	显示带千位分隔符的数字，在分隔符的左边至少显示一位数字，而在分隔符的右边至少显示两位
Percent	该值乘以 100，在后面加上一个百分号
Scientific	用标准的科学记数法
General Date	如果 <i>expression</i> 同时包含了日期和时间，则显示它们。如果 <i>expression</i> 只包含日期或只包含时间，则缺少的信息不显示。日期的显示取决于用户的系统设置
Long Date	使用用户的系统设置所指定的 Long Date 格式
Medium Date	使用 <i>dd-mmm-yy</i> 格式（例如，03-Apr-93）。日期的显示取决于用户的系统设置
Short Date	使用用户的系统设置所指定的 Short Date 格式
Long Time	用用户系统的长时间格式显示时间，包括时、分、秒
Medium Time	使用 <i>hh:mm AM/PM</i> 格式，显示小时、分钟和 AM 或 PM
Short Time	使用 <i>hh:mm</i> 格式，显示小时和分钟

续表

Yes/No	任何非零数字值（通常是 -1）为 Yes，零为 No
True/False	任何非零数字值（通常是 -1）为 True，零为 False
On/Off	任何非零数字值（通常是 -1）为 On，零为 Off

Format 函数还支持其它许多特殊字符，如百分比保留区和指数。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“Format 函数”。

数字的格式

以下数字转换的假设是，Windows “控制面板”中的“区域”，被设为“英语（美国）”。

格式语法	效果
Format(8315.4, “00000.00”)	08315.40
Format(8315.4, “#####.##”)	8315.4
Format(8315.4, “##,##0.00”)	8,315.40
Format(315.4, “\$##0.00”)	\$315.40

小数分隔符是句号 (.)，千位分隔符是逗号 (,)。然而，分隔符的正确显示得依 Windows “控制面板”中的“区域”的设置而定。

打印格式化的日期和时间

为了打印有格式化的日期和时间，应使用具有代表日期和时间符号的

Format 函数。以下示例用 Now 和 Format 函数来标识和格式化当前的日期和时间。下述示例的假设是，Windows “控制面板”中，“区域设置”对话框中的设置为“英语（美国）”。

格式语法	效果
Format(Now, “m/d/yy”)	1/27/93
Format(Now, “dddd, mmm dd, yyyy”)	Wednesday, January 27, 1993
Format(Now, “d-mmm”)	27-Jan
Format(Now, “mmmm-yy”)	January-93
Format(Now, “hh:mm AM/PM”)	07:18 AM
Format(Now, “h:mm:ss a/p”)	7:18:00 a
Format(Now, “d-mmmm h:mm”)	3-January 7:18

通过使用“dddd”和“tttt”格式的 Now 函数，可以把当前日期和时间，按照在 Windows “控制面板”中的“区域设置”对话框内所选择的相应格式进行打印。

国家/地区	格式语法	效果
Sweden	Format(Now, “dddddd tttt”)	1992-12-31 18.22.38
United Kingdom	Format(Now, “dddddd tttt”)	31/12/92 18:22:38
Canada (French)	Format(Now, “dddddd tttt”)	92-12-31 18:22:38
United States	Format(Now, “dddddd tttt”)	12/31/92 6:22:38 PM

详细信息 当使用 Format 函数时，有关国际问题的详细信息，请参阅第十六章“国际化”中的“区域识别函数”。有关基于系统本地日期的详细信息，

请参阅第十六章“国际化”中的“用 Visual Basic 编写国际代码”。

使用选定的文本

当使用剪贴板时，文本框和组合框有一系列对选择文本特别有用的属性。这些属性，引用控件内选定的文本块（突出显示），能够为用户创建剪切和粘贴功能。在运行时下列属性都可改变。

属 性	描 述
<code>SelStart</code>	<code>Long</code> 整数，用来指定选定文本块的起始位置。如果没有选定的文本，则该属性指定插入点的位置。若设置值为 0，所指示的位置，是在文本框或组合框中第一个字符之前。若设置值等于文本框或组合框中文本的长度，所指示的位置，是在控件中最后一个字符之后
<code>SelLength</code>	<code>Long</code> 整数，指定所选的字符个数
<code>SelText</code>	<code>String</code> ，包含选定的字符（如果没有字符被选定的话，就是空字符串）

通过设置 `SelStart` 和 `SelLength` 属性，可控制选定的文本。例如，下列语句可使文本框中的所有文本突出显示：

```
Text1.SetFocus
```

‘从第一个字符之前开始突出显示。

```
Text1.SelStart = 0
```

‘突出显示，一直到文本尾。

```
Text1.SelLength = Len(Text1.Text)
```

如果给 **SelText** 赋值新的字符串，该字符串将替换选定的文本，并且把插入点放在新插入文本尾之后。例如，下面这条语句将用字符串 “I’ve just been inserted!” 替换选定的文本：

```
Text1.SelText = “I’ve just been inserted!”
```

如果没有被选定的文本，则该字符串就粘贴到文本框中的插入点处。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的 “SelStart 属性”、“SelLength 属性” 和 “SelText 属性”。

使用剪贴板对象传送文本和图形

Clipboard 对象没有属性或事件，但它有几个可以与环境剪贴板往返传送数据的方法。**Clipboard** 的方法可分为三类。**GetText** 和 **SetText** 方法，用来传送文本。**GetData** 和 **SetData** 方法，用来传送图形。**GetFormat** 和 **Clear** 方法，可以处理文本和图形两种格式。

详细信息 关于在应用程序中或应用程序间传送数据的详细内容，请参阅第十一章 “响应鼠标和键盘事件” 中的 “OLE 拖放操作”。

使用剪贴板剪切、复制和粘贴文本

两个最有用的 Clipboard 方法是 `SetText` 和 `GetText`。如图 12.2 所示，用这两个方法向剪贴板和从 Clipboard 传送字符串数据。

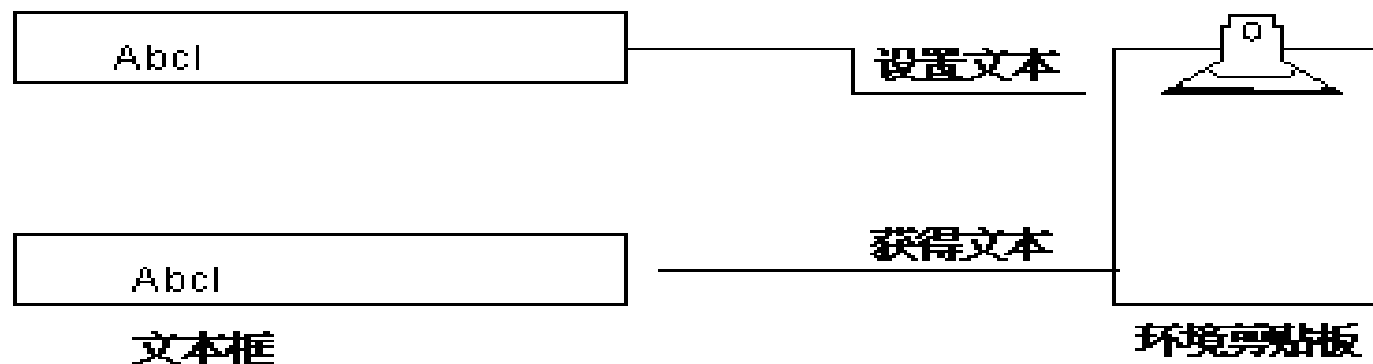


图 12.2 用 `SetText` 和 `GetText` 方法向剪贴板和从剪贴板传送数据

`SetText` 将文本复制到 Clipboard 上，替换先前存储在那里的文本。可将 `SetText` 作为一条语句使用。其语法如下：

```
Clipboard.SetText data[, format]
```

`GetText` 返回存储在 Clipboard 上的文本。也可将它作为函数使用：

```
destination = Clipboard.GetText()
```

将 `SetText` 和 `GetText` 方法，和“使用选定文本”中介绍的选择属性结合起来使用，可容易地编写文本框的“复制”，“剪切”和“粘贴”命令。下列一些事件过程，为以 `mnuCopy`，`mnuCut` 和 `mnuPaste` 命名的控件，实

现了这些命令：

```
Private Sub mnuCopy_Click ()  
    Clipboard.Clear  
    Clipboard.SetText Text1.SelText  
End Sub
```

```
Private Sub mnuCut_Click ()  
    Clipboard.Clear  
    Clipboard.SetText Text1.SelText  
    Text1.SelText = ""  
End Sub
```

```
Private Sub mnuPaste_Click ()  
    Text1.SelText = Clipboard.GetText()  
End Sub
```

注意 如果它们都是菜单控件时，示例会工作得最好，这是因为 Text1 有焦点时可使用菜单。

要注意的是 Copy 和 Cut 这两个过程，都要先用 Clear 方法将 Clipboard 清空（因为可能要以几种不同的格式在 Clipboard 上放置数据，所以 Clipboard 不应自动清空。这在本章后面的“在 Clipboard 上使用多种格式”中有详细描述）。然后，Copy 和 Cut 这两个过程，都用下面的语句将 Text1 中所选择的文本复制到 Clipboard 上：

```
Clipboard.SetText Text1.SelText
```

在“粘贴”命令中，`GetText`方法将返回 `Clipboard` 上当前的文本字符串。然后用一条赋值语句将该字符串复制到文本框的指定位置 (`Text1.SelText`)。如果当前没有被选定的文本，则 `Visual Basic` 将该文本放置在文本框中插入点处：

```
Text1.SelText = Clipboard.GetText()
```

该代码假定全部文本被传送到或传送出文本框 `Text1`，而用户可在 `Text1` 和其它窗体上的控件之间进行复制、剪切和粘贴。

由于 `Clipboard` 是被整个环境所共享的，所以在 `Text1` 和任何正在使用剪贴板的应用程序之间，也能传送文本。

使用 `ActiveControl` 属性

如果希望“复制”、“剪切”和“粘贴”命令能和任何有焦点的文本框一起工作，可使用 `Screen` 对象的 `ActiveControl` 属性。下列代码给任何有焦点的控件提供了一个引用：

```
Screen.ActiveControl
```

可将此代码段当作对控件的任何其它引用一样使用。如果知道该控件是一个文本框，就可引用支持文本框的任何属性，包括 `Text`、`SelText` 和 `SelLength`。下列代码假定活动的控件是一个文本框，且使用了 `SelText` 属性：

```
Private Sub mnuCopy_Click ()
```

```
        Clipboard.Clear
        Clipboard.SetText Screen.ActiveControl.SelText
End Sub

Private Sub mnuCut_Click ()
    Clipboard.Clear
    Clipboard.SetText Screen.ActiveControl.SelText
    Screen.ActiveControl.SelText = ""
End Sub

Private Sub mnuPaste_Click ()
    Screen.ActiveControl.SelText = Clipboard.GetText()
End Sub
```

在剪贴板上使用多种格式

在同一时刻，实际上可以把几块数据放置在 Clipboard 上，只要这几块数据的格式各不相同。这一点是很有用的，因为无法知道什么样的应用程序正在粘贴数据，所以用不同格式提供数据，就能增加为其它应用程序提供可用格式的机会。其它的 Clipboard 方法——GetData、SetData 和 GetFormat——允许通过提供指定格式的数字，处理文本外的数据格式。这些格式与相应的数字一起，在下表进行了描述。

常 数	描 述
VbCFLink	动态数据交换链
VbCFText	文本。本章前面的示例都用这一格式
VbCFBitmap	位图
VbCFMetafile	元文件
VbCFDIB	与设备无关的位图
VbCFPalette	调色板

从图片框控件中剪切和粘贴数据时，可使用后四种格式。下列代码为使用任何标准控件，提供了通用的“剪切”、“复制”和“粘贴”命令。

```
Private Sub mnuCopy_Click ()
    Clipboard.Clear
    If TypeOf Screen.ActiveControl Is TextBox Then
        Clipboard.SetText Screen.ActiveControl.SelText
    ElseIf TypeOf Screen.ActiveControl Is ComboBox Then
        Clipboard.SetText Screen.ActiveControl.Text
    ElseIf TypeOf Screen.ActiveControl Is PictureBox _
        Then
        Clipboard.SetData Screen.ActiveControl.Picture
    ElseIf TypeOf Screen.ActiveControl Is ListBox Then
        Clipboard.SetText Screen.ActiveControl.Text
    Else
        ‘对其它控件没有意义的动作。
```

End If

End Sub

Private Sub mnuCut_Click ()

‘首先要做的与复制相同。

MnuCopy_Click

‘现在清除活动控件的内容。

If TypeOf Screen.ActiveControl Is TextBox Then

Screen.ActiveControl.SelText = “”

ElseIf TypeOf Screen.ActiveControl Is ComboBox Then

Screen.ActiveControl.Text = “”

ElseIf TypeOf Screen.ActiveControl Is PictureBox _
Then

Screen.ActiveControl.Picture = LoadPicture()

ElseIf TypeOf Screen.ActiveControl Is ListBox Then

Screen.ActiveControl.RemoveItem Screen.ActiveControl.ListIndex

Else

‘无操作响应其它控件。

End If

End Sub

Private Sub mnuPaste_Click ()

```

If TypeOf Screen.ActiveControl Is TextBox Then
    Screen.ActiveControl.SelText = Clipboard.GetText()
ElseIf TypeOf Screen.ActiveControl Is ComboBox Then
    Screen.ActiveControl.Text = Clipboard.GetText()
ElseIf TypeOf Screen.ActiveControl Is PictureBox _
    Then
    Screen.ActiveControl.Picture = _
        Clipboard.GetData()
ElseIf TypeOf Screen.ActiveControl Is ListBox Then
    Screen.ActiveControl.AddItem Clipboard.GetText()
Else
    ‘对其它控件没有意义的动作。
End If
End Sub

```

检查剪贴板上的数据格式

可用 `GetFormat` 方法来确认剪贴板上的数据是否具有特定的格式。例如，可根据 `Clipboard` 上的数据是否与当前活动控件兼容，来决定使“粘贴”命令无效。

```
Private Sub mnuEdit_Click ()
```

‘单击事件，弹出“编辑”菜单。

```

MnuCut.Enabled = True
mnuCopy.Enabled = True
mnuPaste.Enabled = False
If TypeOf Screen.ActiveControl Is TextBox Then
    If Clipboard.GetFormat(vbCFText) Then mnuPaste.Enabled = True
ElseIf TypeOf Screen.ActiveControl Is ComboBox Then
    If Clipboard.GetFormat(vbCFText) Then mnuPaste.Enabled = True
ElseIf TypeOf Screen.ActiveControl Is ListBox Then
    If Clipboard.GetFormat(vbCFText) Then mnuPaste.Enabled = True
ElseIf TypeOf Screen.ActiveControl Is PictureBox _
    Then
    If Clipboard.GetFormat(vbCFBitmap) Then mnuPaste.Enabled = True
Else
    '不能从其它类型的控件剪切或复制。
    MnuCut.Enabled = False
    mnuCopy.Enabled = False
End If
End Sub

```

注意 也可能要检查使用常数 vbCFPalette、vbCFDIB 和 vbCFMetafile 的其它数据格式。如果要用 Clipboard 操作来替换图片的调色板，最好要求 Clipboard 是“vbCFBitmap”，而不是“vbCFDIB”。关于使用调色板的详细内容，请参阅本章后面的“使用 256 色”。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“剪贴板对象”。

坐标系统概述

本章描述的每一个图形操作（包括调整大小、移动和绘图），都要使用绘图区或容器的坐标系统。虽然用坐标系统能达到成功的图形效果，然而，如何用坐标系统定义窗体和控件在应用程序中的位置，也是很重要的。

坐标系统是一个二维网格，可定义屏幕上、窗体中或其它容器中（如：图片框或 Printer 对象）的位置。使用窗体中的坐标，可定义网格上的位置：

(x, y)

x 值是沿 x 轴点的位置，最左端是缺省位置 0。Y 值是沿 y 轴点的位置，最上端是缺省位置 0。该坐标系统的图示在图 12.3 中。

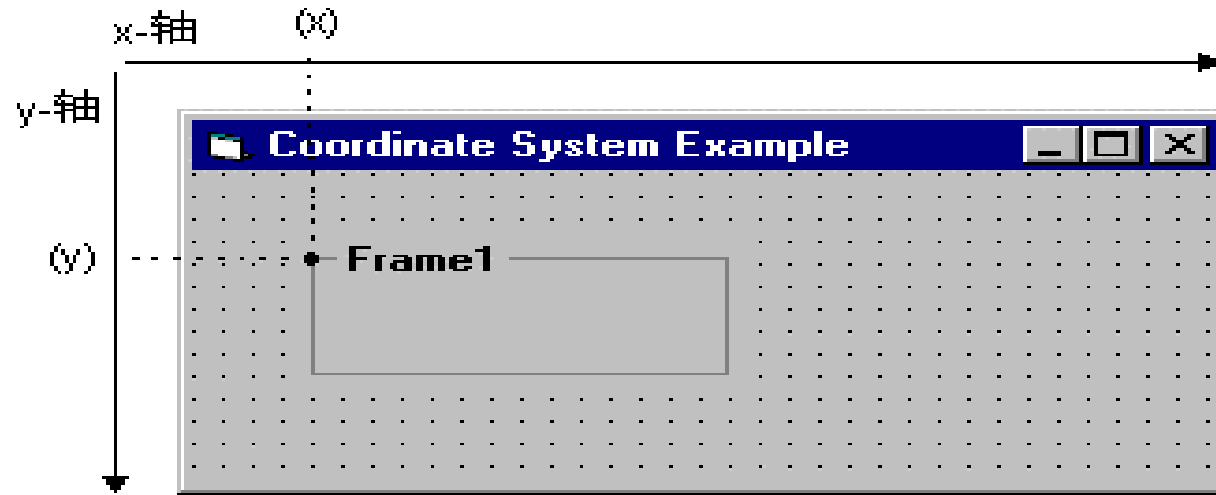


图 12.3 窗体的坐标系统

以下规则用于 Visual Basic 坐标系统：

- 当移动控件或调整控件的大小时，使用控件容器的坐标系统。如果直接在窗体上绘制对象时，窗体就是容器。如果在框架或图片框里绘制控件时，框架或控件是容器。
- 所有的图形和 `Print` 方法，使用容器的坐标系统。例如，那些在图片框里绘制控件的语句，使用的是控件的坐标系统。
- 一些用来调整窗体大小或移动窗体的语句，用缇（`twips`）来表示窗体的位置和大小。

当创建用来调整窗体大小或移动窗体的代码时，应先检查 `Screen` 对象的 `Height` 属性和 `Width` 属性，以确保窗体在屏幕上大小合适。

- 屏幕的左上角总是 `(0, 0)`。任何容器的缺省坐标系统，都是由容器的

左上角（0，0）坐标开始。

沿这些坐标轴定义位置的测量单位，统称为刻度。在 Visual Basic 中，坐标系统的每个轴都有自己的刻度。

坐标轴的方向、起点和坐标系统的刻度，都是可以改变的，但是，对现在来说，使用的是缺省系统。本章后面的“改变对象的坐标系统”中将讨论如何改动。

缇 (twips) 的解释

所有 Visual Basic 的移动、调整大小和图形绘制语句，根据缺省规定，使用缇为单位。缇是打印机的一磅的 1/20（1，440 缇等于一英寸；567 缇等于一厘米）。这些测量值指示对象打印后的大小。屏幕上的物理实际距离根据监视器的大小变化。“改变对象的坐标系统”中描述了如何选择缇以外的单位。

改变对象的坐标系统

可用对象的刻度属性和 Scale 方法，设置特定对象（窗体或控件）的坐标系统。使用坐标系统有以下三种不同的方法：

- 使用缺省的刻度
- 选择标准刻度
- 创建自定义刻度

改变坐标系统的刻度，使得在窗体上缩放图形和定位图形变得更容易。例如，一个在图片框里创建条形图的应用程序，能改变坐标系统，把控件分成四列，每列代表图表中的一条。以下部分，说明如何设置缺省、标准和自定义刻度，来改变坐标系统。

使用缺省刻度

每个窗体和图片框都有几个刻度属性（ScaleLeft、ScaleTop、ScaleWidth、ScaleHeight 和 ScaleMode）和一个方法（Scale），它们可用来定义坐标系统。对于 Visual Basic 中的对象，缺省刻度把坐标（0，0）放置在对象的左上角。缺省刻度单位为缇。
若要返回缺省刻度，可使用无参数的 Scale 方法。

选择标准刻度

若不直接定义单位，可通过设置 ScaleMode 属性，用标准刻度来定义它们。属性设置值见下表。

ScaleMode 设置值	描述
0	用户定义。若直接设置了 ScaleWidth、ScaleHeight、ScaleTop 或 ScaleLeft，则 ScaleMode 属性自动设为 0
1	缇。这是缺省刻度。1，440 缇等于一英寸
2	磅。72 磅等于一英寸

续表

3	像素。像素是监视器或打印机分辨率的最小单位。每英寸里像素的数目由设备的分辨率决定
4	字符。打印时，一个字符有 1/6 英寸高、1/12 英寸宽
5	英寸
6	毫米
7	厘米

除了 0 和 3，表中的所有模式都是打印长度。例如，一个对象长为两个单位，当 `ScaleMode` 设为 7 时，打印时就是两厘米长。

'设该窗体的刻度单位为英寸。

```
ScaleMode = 5
```

'设 `picPicture1` 的刻度单位为像素。

```
picPicture1.ScaleMode = 3
```

设置 `ScaleMode Screen` 的值后，Visual Basic 会重定义 `ScaleWidth` 和 `ScaleHeight`，使它们与新刻度保持一致。然后，`ScaleTop` 和 `ScaleLeft` 设置为 0。直接设置 `ScaleWidth`、`ScaleHeight`、`ScaleTop` 或 `ScaleLeft`，将自动设置 `ScaleMode` 为 0。

创建自定义刻度

可使用对象的 `ScaleLeft`、`ScaleTop`、`ScaleWidth` 和 `ScaleHeight` 这些属性，来创建自定义刻度。与 `Scale` 方法不同，这些属性能用来设定刻度，或取

得有关坐标系统当前刻度的详细信息。

使用 **ScaleLeft** 和 **ScaleTop** 属性

ScaleLeft 和 **ScaleTop** 属性，用来给对象指定左上角的数字值。例如，这些语句给当前窗体的左上角和名为 **picArena** 图片框的左上角设定数值。

```
ScaleLeft = 100
```

```
ScaleTop = 100
```

```
picArena.ScaleLeft = 100
```

```
picArena.ScaleTop = 100
```

这些刻度值显示在图 12.4 中。

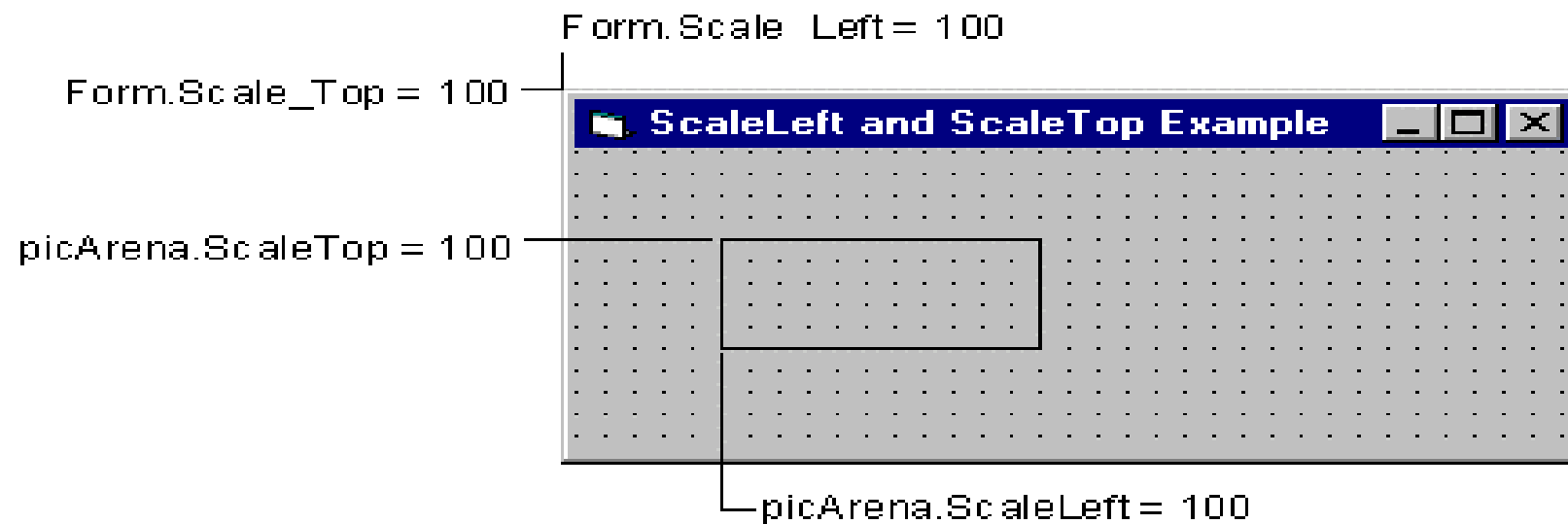


图 12.4 窗体和控件的 **ScaleLeft** 和 **ScaleTop** 属性

这些语句定义左上角为 (100, 100)。虽然这些语句不直接改变那些对象的大小或位置，但它们改变其后一些语句的作用。例如，其后的一条设置控件 Top 属性为 100 的语句，将把对象置于它的容器的最上端。

使用 **ScaleWidth** 和 **ScaleHeight** 属性

ScaleWidth 和 **ScaleHeight** 属性，将根据绘图区的当前宽度和高度定义单位。例如：

```
ScaleWidth = 1000
```

```
ScaleHeight = 500
```

这些语句定义的是，当前窗体内部宽度的 1/1,000 为水平单位；当前窗体内部高度的 1/500 为垂直单位。如果窗体的大小以后被调整，这些单位保持原状。

注意 **ScaleWidth** 和 **ScaleHeight** 是按照对象的内部尺寸来定义单位的，这些尺寸不包括边框厚度或菜单（或标题）的高度。因此，**ScaleWidth** 和 **ScaleHeight** 总是指对象内的可用空间的大小。内部尺寸和外部尺寸（尺寸由 **Width** 和 **Height** 指定）的区别，对于有宽厚边框的窗体特别重要。这些单位也可不同：**Width** 和 **Height** 总是按照容器的坐标系统来表示；**ScaleWidth** 和 **ScaleHeight** 决定了对象本身的坐标系统。

改变坐标系统的属性设置

所有的四个刻度属性，都可包括分数，也可是负数。对于 `ScaleWidth` 和 `ScaleHeight` 属性，设置值为负数则改变坐标系统的方向。

图 12.5 中所示的刻度，有 `ScaleLeft`、`ScaleTop`、`ScaleWidth` 和 `ScaleHeight`，都设置为 100。

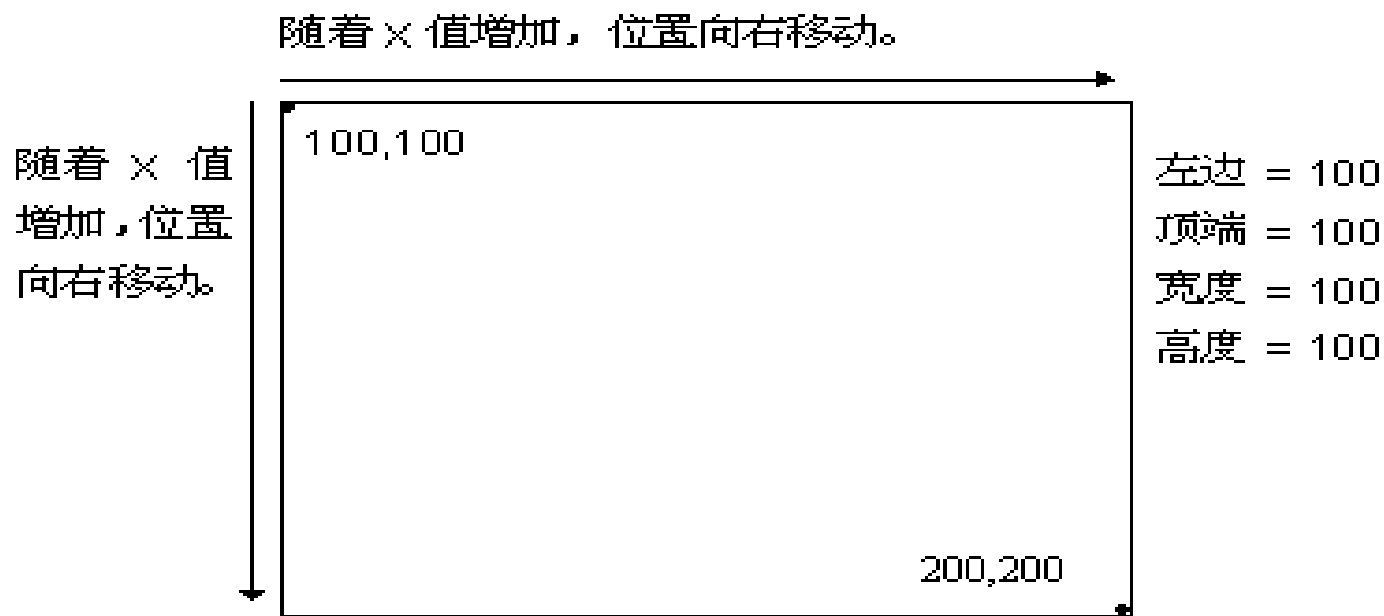


图 12.5 刻度从 (100, 100) 变化到 (200, 200)

使用刻度方法改变坐标系统

一个更有效的改变坐标系统的途径，不是设置个别属性，而是使用 `Scale` 方法。可使用下述的语法，指定自定义刻度：

`[object.]Scale (x1, y1) - (x2, y2)`

x1 和 y1 的值，决定了 ScaleLeft 和 ScaleTop 属性的设置值。两个 x-坐标之间的差值和两个 y-坐标之间的差值，分别决定了 ScaleWidth 和 ScaleHeight 属性的设置值。例如，假定要为一窗体设置坐标系统，而将两个端点设置为 (100, 100) 和 (200, 200)：

```
Scale (100, 100)-(200, 200)
```

该语句定义窗体为 100 单位宽和 100 单位高。用该刻度，下述语句将一个形状控件移动窗体宽度行程的五分之一：

```
shpMover.Left = shpMover.Left + 20
```

指定 $x1 > x2$ 或 $y1 > y2$ 的值，与设置 ScaleWidth 或 ScaleHeight 为负值的效果相同。

刻度的转换

使用 ScaleX 和 ScaleY 方法，用来将一种刻度模式转换为另一种刻度模式。这些方法的语法如下：

```
[object.]ScaleX (value [, fromScale [, toScale]])
```

```
[object.]ScaleY (value [, fromScale[, toScale]])
```

目标 *object* 是窗体、图片框或 Printer 对象。*value* 表示在由刻度模式 *fromScale* 指定的坐标系统中。返回值表示在由 *toScale* 指定的刻度模式中，或者，如果 *toScale* 省略时，表示在 *object* 的刻度模式中。如果 *fromScale* 被省略，则 *value* 的刻度模式是 HIMETRIC。

HIMETRIC 是一种指定了物理尺寸的刻度模式。例如，相应 10 厘米长的一条直线，按 HIMETRIC 单位的长度数是 10,000。无论视频显示区的大小如何，在屏幕上绘出的直线都是 10 厘米长。关于 HIMETRIC 刻度模式和物理尺寸的详细内容，请参阅 Microsoft Windows 的 SDK。

以下语句把 MyPic 图片框控件的内容宽度，拉伸到原来的两倍。MyPic.Picture.Width 返回包含在图片控件中的图片的宽度，这是 HIMETRIC 数值，需要转换为 Form1 的刻度模式。

```
Form1.PaintPicture MyPic.Picture, X, Y, _  
    Form1.ScaleX(MyPic.Picture.Width) * 2
```

下例展示了指定窗体宽度为 np 像素宽的两种相同途径。

'窗体的 ScaleMode 设为像素。

```
ScaleMode = vbPixels
```

'选项 1:

'临时地把窗体的 ScaleMode 设为缇。

```
ScaleMode = vbTwips
```

'ScaleX() 返回以缇为单位的值。

```
Width = Width - ScaleWidth + ScaleX(np, vbPixels)
```

'把窗体的 ScaleMode 设回到像素。

```
ScaleMode = vbPixels
```

'选项 2:

'转换像素为缇，但不改变窗体的 ScaleMode。

```
Width = Width + ScaleX(np - ScaleWidth, vbPixels, _  
vbTwips)
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“ScaleX 属性”或“ScaleY 属性”。

使用图形控件

Visual Basic 提供三种控件，是为了在应用程序中创作图形效果而设计的：

- image 控件
- Line 控件
- shape 控件

图形控件的优点

在设计时，图像控件、直线控件和形状控件，对于创建图形十分有用。图形控件的一个优点是，需要的系统资源比其它 Visual Basic 控件少，这就提高了 Visual Basic 应用程序的性能。

图形控件的另一个优点是，创建图形所用的代码比图形方法用的要少。例如，在窗体上放置一个圆，既可用 Circle 方法，也可用形状控件。Circle 方法要求在运行时用代码创建圆；而用形状控件的话，只需在设计时简单地把它拖到窗体上，并设置特定的属性。

图形控件的限制条件

当图形控件的设计目的是为了使其性能最大化，而对应用程序的要求又最小化时，要达到这一目标，就要限制 Visual Basic 控件共有的其它功能。图形控件：

- 不能出现在其它控件之上，除非它们是在一个容器里，而这个容器能出现在其它控件之上（如：图片框）。
- 不能在运行时接收焦点。
- 不能作为其它控件的容器。
- 不具有 `hWnd` 属性。

详细信息 关于图形方法的详细内容，请参阅本章后面的“使用图形方法”。关于图形控件的详细内容，请参阅第七章“使用 Visual Basic 的标准控件”中的“使用 Image 控件”、“使用 Line 控件”和“使用 Shape 控件”。关于图形对应用程序性能的影响，特别要参阅第十五章“性能和兼容性设计”的“修剪图形”。

给应用程序添加图片

图片可显示在 Visual Basic 应用程序的三种位置处：

- 窗体上
- 图片框内

- 图像控件内

图片可来自 Microsoft Windows 的各种绘图程序，例如，随同各种版本 Microsoft Windows 一起提供的那些绘图程序，其它图形应用程序，或剪切美术库等。Visual Basic 提供了一个大图标库，可在应用程序中作为图形使用。Visual Basic 也能象 .bmp、.dib、.ico、.cur、.wmf、.emf 等文件那样，把 .jpeg 和 .gif 文件添加到应用程序中。关于 Visual Basic 支持的图形格式，请参阅第七章“使用 Visual Basic 的标准控件”中的“使用 Image 控件”和“使用 PictureBox 控件”。

根据是在设计时还是运行时，可采用不同途径把图片添加到窗体、图片框或图像控件中。

设计时添加图片

设计时添加图片有两种方法：

- 从图片文件中将图片加载到窗体上、图片框中或图像控件里：
在“属性”窗口，从“属性”列表中选择“图片”，并单击“属性”按钮。Visual Basic 将显示一个对话框，从中可选择要加载的图片文件。
如果给窗体设置了 Picture 属性，选定的图片就会显示在窗体上，被放置在其上任何控件的后面。类似地，如果给图片框设置了 Picture 属性，则该图片就会显示在该框中，被放置在其上任何控件的后面。
- 把一个图片粘贴到窗体上，图片框中或图像控件里：
把图片从另一个应用程序（例如 Microsoft Paint）复制到剪贴板上。返

回 Visual Basic 环境中，选择该窗体，图片框或图像控件，然后从“编辑”菜单上，选择“粘贴”。

一旦为窗体，图片框或图像控件设置了 Picture 属性——无论是加载或粘贴图片——设置值框所显示的字，将是 "(Bitmap)", "(Icon)", 或 "(Metafile)"。为了改变此设置值，可加载或粘贴另一幅图片。双击设置值框内所显示的字，并按 DEL 键，可将 Picture 属性重新设置为 "(None)"。

运行时添加图片

运行时添加图片有四种方法：

- 使用 LoadPicture 函数，指定一文件名，并将该图片赋值给 Picture 属性。

下列语句将 Cars.bmp 文件加载到名为 picDisplay 的图片框内（通过设置其 Name 属性，可对一控件命名）：

```
picDisplay.Picture = LoadPicture("C:\Picts\Cars.bmp")
```

任何需要的时候，都可加载一新图片到窗体上，图片框中或图像控件里。尽管图片的源文件不会受到影响，但加载的新图片将会完全替代正显示的图片。

- 使用 LoadResPicture 函数，可把工程中 .res 文件的一图片，赋值给 Picture 属性。

下列语句将资源文件里资源标识号 ID 为 10 的位图，加载到名为 picResource 的图片框内：

```
Set picResource.Picture = LoadResPicture(10, _  
vbResBitmap)
```

- 对象间图片的相互复制。
如果图片一旦被加载或粘贴到窗体、图片框或图像控件以后，那么运行时就可把它赋值给另一窗体，图片框或图像控件。例如，下列语句将把名为 `picDisplay` 图片框中的图片，复制到名为 `imgDisplay` 的图像控件内：

```
Set imgDisplay.Picture = picDisplay.Picture
```

- 从剪贴板对象复制图片。

详细信息 关于从剪贴板复制图片的详细内容，请参阅“在剪贴板上使用多种格式”。

关于资源文件的内容，请参阅第八章“使用资源文件”中的“再论编程”。

注意 如果是在设计时从文件中加载或粘贴图片，则图片就和窗体一起被保存和加载，而应用程序可将图片从一个对象复制到其它对象。然后，在创建 `.exe` 文件时，就不必把此图片文件的备份提供给用户，因为 `.exe` 文件本身包含有它的图像。另外，可以考虑提供 `.res` 文件和使用 `LoadResPicture`。`.res` 文件将被编译到 `.exe` 文件中，而位图则以任何资源编辑器都可读的标准格式保存。如果要在运行时用 `LoadPicture` 函数加载图片，则必须把图片文件和应用程序一起提供给用户。

运行时删除图片

也可使用 `LoadPicture` 函数在运行时删除图片，而无需用其它图片替换它。下列语句是从名为 `imgDisplay` 的图像控件中删除图片：

```
Set imgDisplay.Picture = LoadPicture("")
```

移动图片和改变图片尺寸

如果窗体、图片框或图像控件被移动（设计时或运行时），则其上的图片也将自动地随它一起移动。如果窗体、图片框或图像控件调整大小后，容纳不下图片，则该图片将从右边和底部被裁剪。如果图片被加载或复制到那些因为过小而不能完整显示它的窗体、图片框或图像控件时，该图片也会被裁剪。

AutoSize 属性

如果想让图片框能自动扩展到可容纳新图片的大小，可将该图片框的 `AutoSize` 属性设置为 `True`。这样，在运行时当往图片框加载或复制图片时，`Visual Basic` 会自动扩展该控件到恰好能够显示整个图片。由于窗体不会改变大小，如果加载的图像大于窗体的边距，图像经过裁剪后被显示出来。也可以使用 `AutoSize` 属性使图片框自动收缩，以便对新图片的尺寸作出反应。

注意 图像控件没有 `AutoSize` 属性，但为了适应加载给它的图片的要求，

可以自动调整大小。窗体没有 `AutoSize` 属性，并且也不能自动扩大以显示整个图片。

图像控件的 **Stretch** 属性

如果要使图像控件中的图片能自动扩展以适应特定的尺寸，可使用 `Stretch` 属性。当 `Stretch` 属性为 `False` 时，图像控件自动调整其大小以适应加载图片的要求。为了调整图片大小以适应图像控件的要求，可将图像控件的 `Stretch` 属性设置为 `True`。

为图像控件选择图画

从哪儿得到图片文件？如果想要图标，可用 `Visual Basic` 所提供的图标库。在 `Visual Basic` 主目录下的子目录 (`\Vb\Graphics\Icons`) 中，可查找图标文件。可用 `Microsoft Paint` 创建 `.bmp` 文件，或购买包括位图、图标文件或元文件的剪贴美术集，还可以创建包含图片的资源 (`.res`) 文件。

详细信息 关于创建资源文件的详细内容，请参阅第八章“使用资源文件”中的“再论编程”。

窗体和控件的图形属性概述

窗体和各种控件都具有图形属性。下表列出这些属性。

类别	属性
显示处理	AutoRedraw, ClipControls
当前绘图位置	CurrentX, CurrentY
绘图技术	DrawMode, DrawStyle, DrawWidth, BorderStyle, BorderWidth
填充技术	FillColor, FillStyle
颜色	BackColor, ForeColor, BorderColor, FillColor

窗体和图片框具有的一些附加属性：

- **Scale** 属性，在本章前面的“改变对象的坐标系统”中进行了描述。
- **Font** 属性，在本章前面的“设置字体特征”中进行了描述。

窗体和图片框有两个属性很快就会用到：**BackColor** 和 **ForeColor**。**BackColor** 绘制绘画区的背景。若 **BackColor** 为亮蓝，则清除后整个区域为亮蓝色。尽管某些图形方法提供使用不同颜色的选项，仍由 **ForeColor**（前景）来决定对象上绘制的文本或图形的颜色。关于颜色的详细内容，请参阅本章后面的“使用颜色”。

用 AutoRedraw 创建持久的图形

每个窗体和图片框都具有 **AutoRedraw** 属性。**AutoRedraw** 是 **Boolean** 属性，当它设置为 **True** 时，会把图形输出保存在内存里。可用 **AutoRedraw** 属性创建持久的图形。

持久的图形

Microsoft Windows 操纵屏幕图像，来造成重叠窗口的影像。当一个窗口移到其它窗口上时，可暂时隐藏其它窗口，移过后，被覆盖的窗口和它的内容需要重新显示。Windows 管理和控制窗口与控件的重新显示。而用户的 Visual Basic 应用程序必须控制窗体和图片框内图形的重新显示。如果在窗体上用图形方法创建图形，通常希望它们重新显示在以前的位置 (persistent graphics)，可用 `AutoRedraw` 属性创建持久的图形。

AutoRedraw 和窗体

`AutoRedraw` 的缺省值是 `False`。当 `AutoRedraw` 设置为 `False`，窗体上显示的任何由图形方法创建的图形如果被另一个窗口暂时挡住，将会丢失。另外，如果扩大窗体，窗体边界外的图形将会丢失。设置 `AutoRedraw` 为 `False` 的效果如图 12.6 所示。



图 12.6 设置 AutoRedraw 为 False 的效果

当窗体的 `AutoRedraw` 属性设置为 `True` 时，Visual Basic 会将图形方法适用于内存中的一块“画布”上。应用程序复制此画布的内容，以便重新显示被其它窗口暂时隐藏起来的图形。在大多数情况下，该窗体画布的大小即屏幕的大小。如果窗体的 `MaxButton` 属性为 `False` 且窗体的边界不可调，那么该画布的大小即窗体的大小。

当窗体大小可调时，该画布也可使应用程序保存超出窗体边界之外的图形。设置 `AutoRedraw` 为 `True` 的结果显示在图 12.7 中。

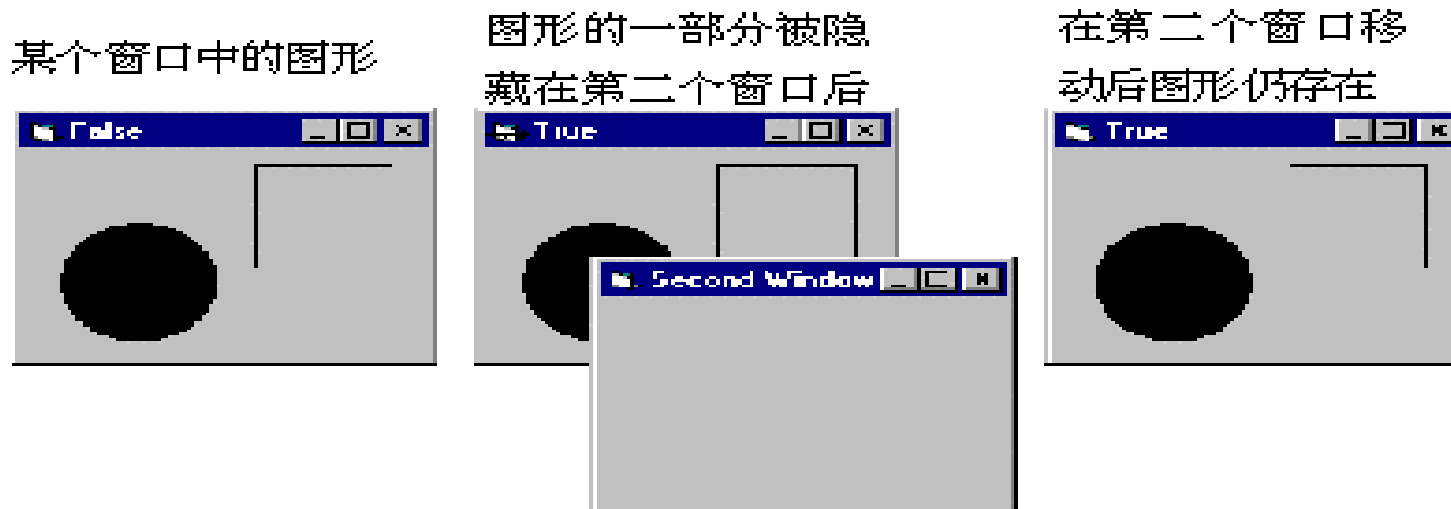


图 12.7 设置 AutoRedraw 为 True 的效果

AutoRedraw 和图片框

当图片框的 AutoRedraw 属性设置为 True 时，Visual Basic 仅在内存保存此图片框的可视内容。这是因为保存图片框内容的画布，与图片框大小是相同的。超出其外的图形被裁剪后，即使改变图片框的大小，也不能再显示。

使用非持久的图形

可将窗体及其所有图片框的 AutoRedraw 设置为 False 来节省内存。但那样的图形将不再自动成为持久性的。需要时必须用代码管理所有图形的重

画。

可将所有想要重画的线、圆、点的代码，包括在窗体或图片框的 `Paint` 事件中。这种方式最适用于有限个易于重建的图形。

无论何时需要重画窗体或图片框的一部分时，都可调用 `Paint` 事件过程，例如，当覆盖住对象的窗口移去后，或重新调整窗体大小使得图形恢复可视时。如果 `AutoRedraw` 设置为 `True`，除非应用程序显式地调用它，否则，该对象的 `Paint` 过程不会被调用执行。因为对象的可视内容保存在画布中，所以不需要 `Paint` 事件。

注意，使用非持久的图形的决定，会影响图形在窗体或容器上的绘制方法。

“用 `ClipControls` 裁剪区域”和“用 `AutoRedraw` 和 `ClipControls` 使图形分层”中讨论了决定是否需用非持久的图形的其它因素。

运行时改变 `AutoRedraw` 的设置

运行时可以改变 `AutoRedraw` 的设置。如果 `AutoRedraw` 的设置为 `False`，图形和 `Print` 方法的输出，只输出到屏幕，不到内存。当 `AutoRedraw` 设置为 `True` 时，如果用 `Cls` 方法清除对象，已有输出并不能被清除。这是因为输出保存在内存，必须再次设置 `AutoRedraw` 为 `True`，才能用 `Cls` 方法清除。

详细信息 为了得到有关 `AutoRedraw` 的性能含义，请参阅第十五章“性能和兼容性设计”的“优化显示速度”。

用 ClipControls 裁剪区域

每个窗体、图片框和框架控件，都具有 ClipControls 属性。ClipControls 是 Boolean 属性，当它设置为 True 时，导致容器定义一个裁剪区域，用以绘制该容器中几乎所有的控件，但不包括：

- Shape 控件
- line 控件
- image 控件
- Labels
- 任何 ActiveX 图形控件

将 ClipControls 属性设置为 False，则可提高窗体在屏幕上的绘制速度。对于具有许多不重叠控件的窗体，如对话框等，绘制速度的提高最为明显。

裁剪区域

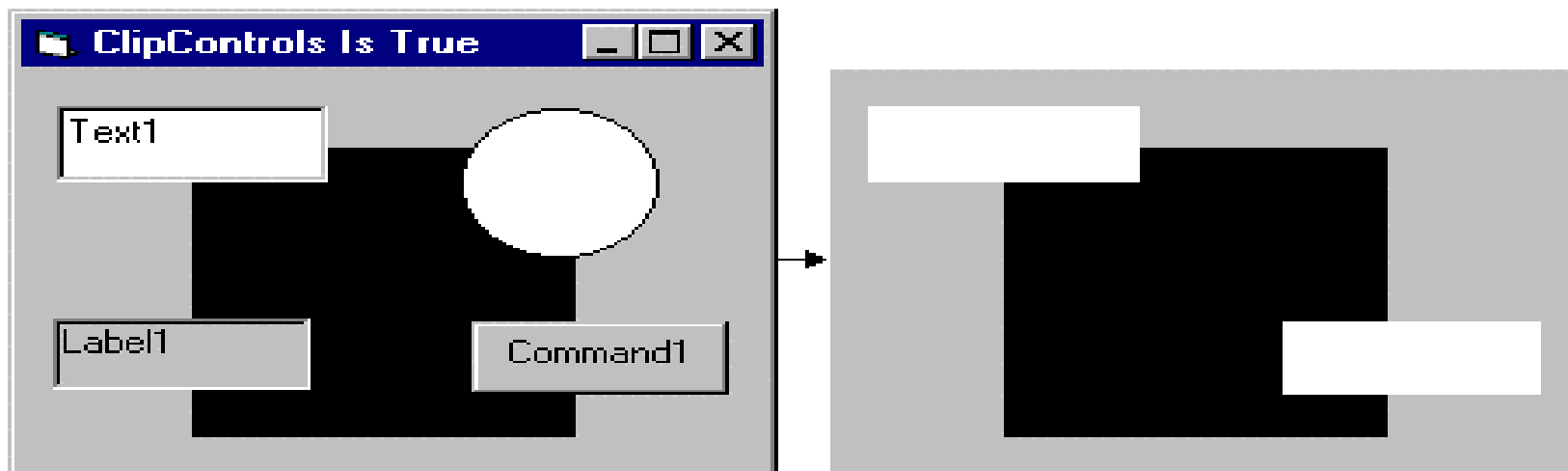
Clipping 是当显示窗体或容器时，决定窗体或容器的哪部分被绘制的过程。决定窗体或容器的哪部分被绘制所用的轮廓线，或称为“被裁剪”，就定义为 clipping region。当基于 Windows 的应用程序，需要保存部分显示内容，同时重画其余部分时，裁剪区域是十分有用的。

窗体和容器的裁剪

`ClipControls` 缺省设置为 `True`。当 `ClipControls` 属性为 `True` 时，`Windows` 在 `Paint` 事件之前，窗体或容器的背景定义一个裁剪区域。此裁剪区域将围绕所有非图形控件。当使用 `ClipControls` 时，标签的作用与图形控件类似。

在 `Paint` 事件期间，`Windows` 只重画裁剪区域的背景，而不重画非图形控件。图 12.8 显示出具有四个控件的一个窗体、一个用 `Line` 方法绘制的矩形、以及通过将 `ClipControls` 设置为 `True`，为窗体创建的裁剪区域。注意，该裁剪区域没有裁剪窗体上标签或形状控件的边线。背景上用 `Line` 方法所画的矩形只在裁剪区域显示。

当 `ClipControls` 为 `False` 时，`Windows` 不在 `Paint` 事件之前给窗体或容器的背景定义裁剪区域。而且，`Paint` 事件中图形方法的输出，只显示在窗体或控件必须重画的部分。因为计算和管理裁剪区域需要花费时间，那么设置 `ClipControls` 为 `False` 就可以使含有许多非重叠控件的窗体（如复杂对话框）显示得更快。



一个窗体，它带有两个非图形控件，一个 shape 控件，一个 label，以及一个用 Line 方法创建的方框。

以淡灰色显示的剪切区；shape 控件和 label 没有被剪切。仅在剪切区画方框。

图 12.8 当 ClipControls 为 True 时所创建的裁剪区域

注意 应避免把 ClipControls 为 True 的控件嵌入到 ClipControls 为 False 的控件中。这样做会导致嵌入控件不能正确重画。把容器和控件的 ClipControls 都设置为 True，就可避免这种情况发生。

详细信息 请参阅第十五章“性能和兼容性设计”中的“优化显示速度”。

用 AutoRedraw 和 ClipControls 使图形分层

AutoRedraw 和 ClipControls 的不同组合，对图形控件和图形方法绘制屏幕有不同影响。

创建图形时，应记住图形控件和标签、非图形控件、以及图形方法，将出现在一个容器的不同层次上。这些层次变化取决于三个因素：

- `AutoRedraw` 的设置值。
- `ClipControls` 的设置值。
- 图形方法出现在 `Paint` 事件之内还是之外。

正常的分层

一般来说，窗体或其它容器的层次是，从前到后，如下表所示：

层次	内容
上层	非图形控件，如命令按钮、复选框、文件控件
中层	图形控件和标签
下层	窗体或容器的绘图区。这是图形方法显示结果的地方

某层次中的任何对象，都会覆盖其下层的任何对象，所以用图形控件创建的图形，将出现在窗体上其它控件的后面；而用图形方法创建的图形，则出现在所有图形控件和非图形控件的下面。正常的层次排列在图 12.9 示出。

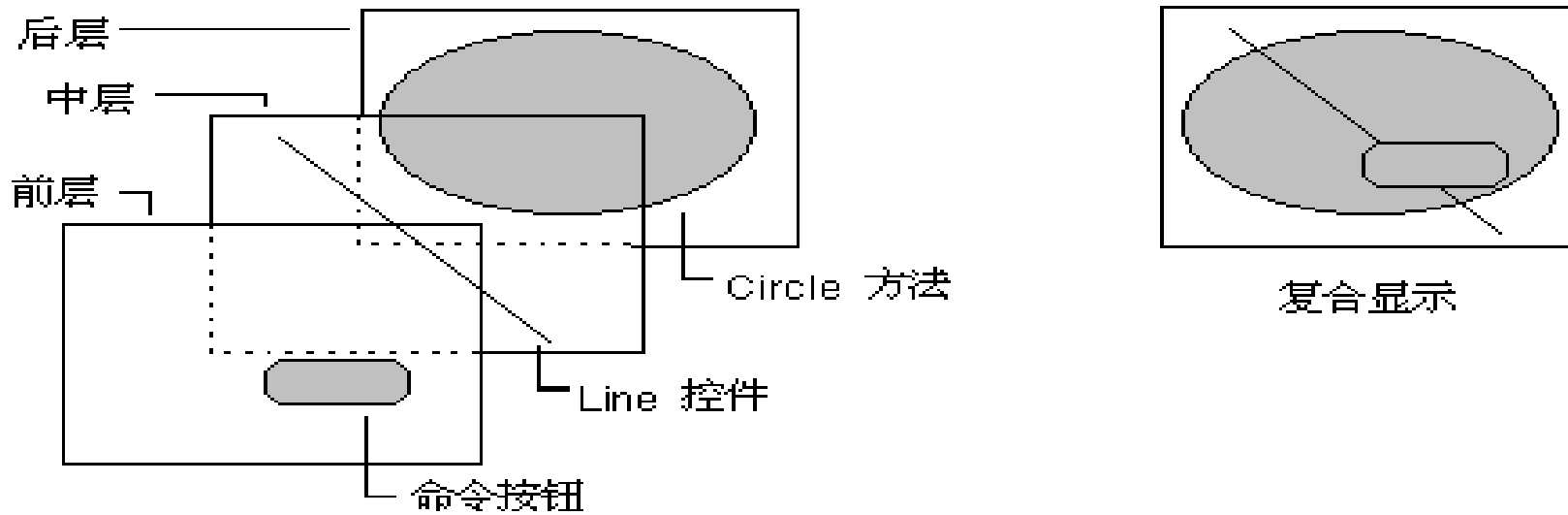


图 12.9 图形在窗体上的正常层次关系

影响分层的因素

可用几种不同的途径产生正常的分层。AutoRedraw 和 ClipControls 的不同设置组合，以及将图形方法置于事件内部或外部，都会影响层次排列和应用程序的执行速度。

下表列出 AutoRedraw 和 ClipControls 的不同组合，以及图形方法的放置方式所造成的各种效果。

AutoRedraw	ClipControls	图 形 方 法 在 Paint 事 件 之 内 / 外	分 层 的 行 为
True	True (缺省)	忽略 Paint 事件	正常的层次
True	False	忽略 Paint 事件	正常的层次。因为不需要计算或建立裁剪区域，具有许多非重叠控件的窗体会画的更快
False (缺省)	True (缺省)	内	正常的层次
False	True	外	非图形控件在前。图形方法和图形控件混合将出现在中层和下层。建议不要使用此种设定
False	False	内	正常的层次，只影响先前被覆盖或重新调整窗体大小才出现的像素
False	False	外	图形方法和所有控件混合将出现在三个层次中。建议不要使用此种设定

AutoRedraw 的效果

设置 AutoRedraw 为 True 时，总能产生正常的层次。虽然使用 AutoRedraw

是层叠图形的最简便方法，但由于 `AutoRedraw` 要占用内存空间，故含有较大窗体的应用程序的执行速度会降低。

ClipControls 的效果

当 `AutoRedraw` 为 `True` 时，`ClipControls` 的设置对图形在窗体上或容器内如何层叠没有影响。但 `ClipControls` 会影响窗体的显示速度。当 `ClipControls` 为 `False` 时，应用程序不创建裁剪区域。不必在裁剪区域计算或绘图以弥补漏洞，就可使窗体显示得更快些。

而且，当 `AutoRedraw` 和 `ClipControls` 都为 `False` 时，应用程序只重画在下述两种情况下被显露出来的窗体或容器的那些像素：

- 另一个覆盖了该窗体或容器的窗口被移去
- 重新调整窗体或容器的大小时

Paint 事件的影响

当 `AutoRedraw` 为 `False` 时，图形方法最好置于窗体或容器的 `Paint` 事件内。把图形方法局限在 `Paint` 事件内，能使这些方法按可预期的顺序来绘图。

当 `AutoRedraw` 为 `False` 时，在 `Paint` 事件之外使用图形方法，会产生不稳定的图形。每次图形方法的输出在窗体上或容器内出现时，可能会覆盖任何控件或图形方法（若 `ClipControls` 为 `False`）已经显示在那里的内容。当应用程序使用许多图形方法来产生视觉效果时，除非将这些图形方法局

限于 Paint 事件内，否则对输出结果的管理是非常困难。

动态地移动控件

使用 Visual Basic，在运行时移动控件是最容易取得的效果之一。可以直接改变控件位置定义的属性，也可使用 Move 方法。

使用 Left 和 Top 属性

Left 属性是控件左上角到窗体左边的距离。Top 属性是控件左上角到窗体上边的距离。图 12.10 显示控件的 Left 和 Top 属性。

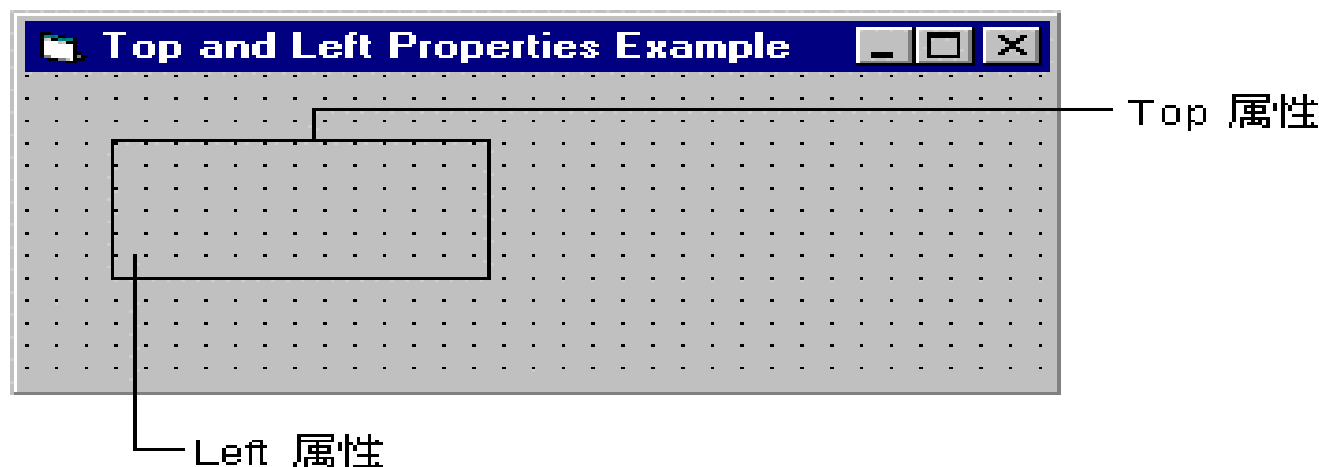


图 12.10 Left 和 Top 属性

通过改变 `Left` 和 `Top` 属性的设置值移动控件，可使用以下语句：

```
txtField1.Left = txtField1.Left + 200
```

```
txtField1.Top = txtField1.Top - 300
```

移动 `Line` 控件

如前面提到的，直线控件没有 `Left` 和 `Top` 属性。但是，可使用特殊的属性，对窗体上直线控件的位置进行控制。下表列出这些属性，以及用它们怎样决定直线控件的位置。

属性	描述
<code>X1</code>	直线起点的 <code>x</code> 坐标。该坐标用当前刻度单位给出。直线的起点是开始画之前创建的端点
<code>Y1</code>	直线起点的 <code>y</code> 坐标
<code>X2</code>	直线终点的 <code>x</code> 坐标。直线终点是停止画时最后创建的端点
<code>Y2</code>	直线终点的 <code>y</code> 坐标

`Blanker` 应用程序的跳跃直线演示程序，使用以下语句随机地改变 `DemoForm` 上直线控件的位置：

'为第一个直线端点，设置随机的 `X` 位置值。

```
linLineCtl.X1 = Int(DemoForm.Width * Rnd)
```

'为第一个直线端点，设置随机的 `Y` 位置值。

```
linLineCtl.Y1 = Int(DemoForm.Height * Rnd)
```

'为第二个直线端点，设置随机的 `X` 位置值。

```
linLineCtl.X2 = Int(DemoForm.Width * Rnd)
```

'为第二个直线端点，设置随机的 Y 位置值。

```
linLineCtl.Y2 = Int(DemoForm.Height * Rnd)
```

'清除移动直线遗留像素。

```
Cls
```

'在下次移动之前使显示暂停。

```
Delay
```

使用 Move 方法

改变 Left 和 Top 或 X 和 Y 属性，会使控件产生先水平移动然后再垂直移动的颠簸效果。Move 方法则能产生更平滑的对角线方向的移动。

Move 方法的语法如下：

```
[object.]Move left [, top[, width[, height]]]
```

object 是被移动的窗体或控件。如果省略了 *object*，则移动的是当前窗体。*left* 和 *top* 参数是 *object* 对象的 Left 和 Top 属性的新设置值，而 *width* 和 *height* 则是它的 Width 和 Height 属性的新设置值。其中，只有 *left* 是必需的，但是，若要指定其它参数时，就必须一并指定参数列表中出现在指定参数之前的所有参数。

绝对移动

当对象在其容器中移动到指定坐标处时，就是绝对移动。下述语句是用绝

对移动，将名为 `txtField1` 的控件，移动到坐标（100，200）处：

```
txtField1.Move 100, 200
```

相对移动

通过指定从当前位置应当移动的距离，使控件移动时，就是相对移动。下述语句是使用相对移动，将 `txtField1` 移动到它当前位置向右和向下 100 缇处：

```
txtField1.Move txtField1.Left + 100, txtField1.Top _  
               + 100
```

本节显示 **Blanker** 示例应用程序中控件的移动。回弹演示程序，可使图片框沿着窗体对角线移动，所以图片框看起来象是从窗体边“反弹”回来一样。因为图像控件移动时重画引起抖动，所以该演示程序使用图片框而不用图像控件。

图 12.11 显示了 **Blanker** 应用程序 (**DemoForm**) 的主窗体和此例中用到的图片框。

图片框的名称是 `picBall`。当从“选项”菜单中选定“**Rebound**”命令，然后单击“**Start Demo**”按钮后，该控件就开始沿着窗体移动。这个命令按钮的事件过程再调用 `CtlMoveDemo` 过程。

`CtlMoveDemo` 过程随机地从以下四个可能中选定一个开始方向：

- 左上方
- 右上方
- 左下方

- 右下方

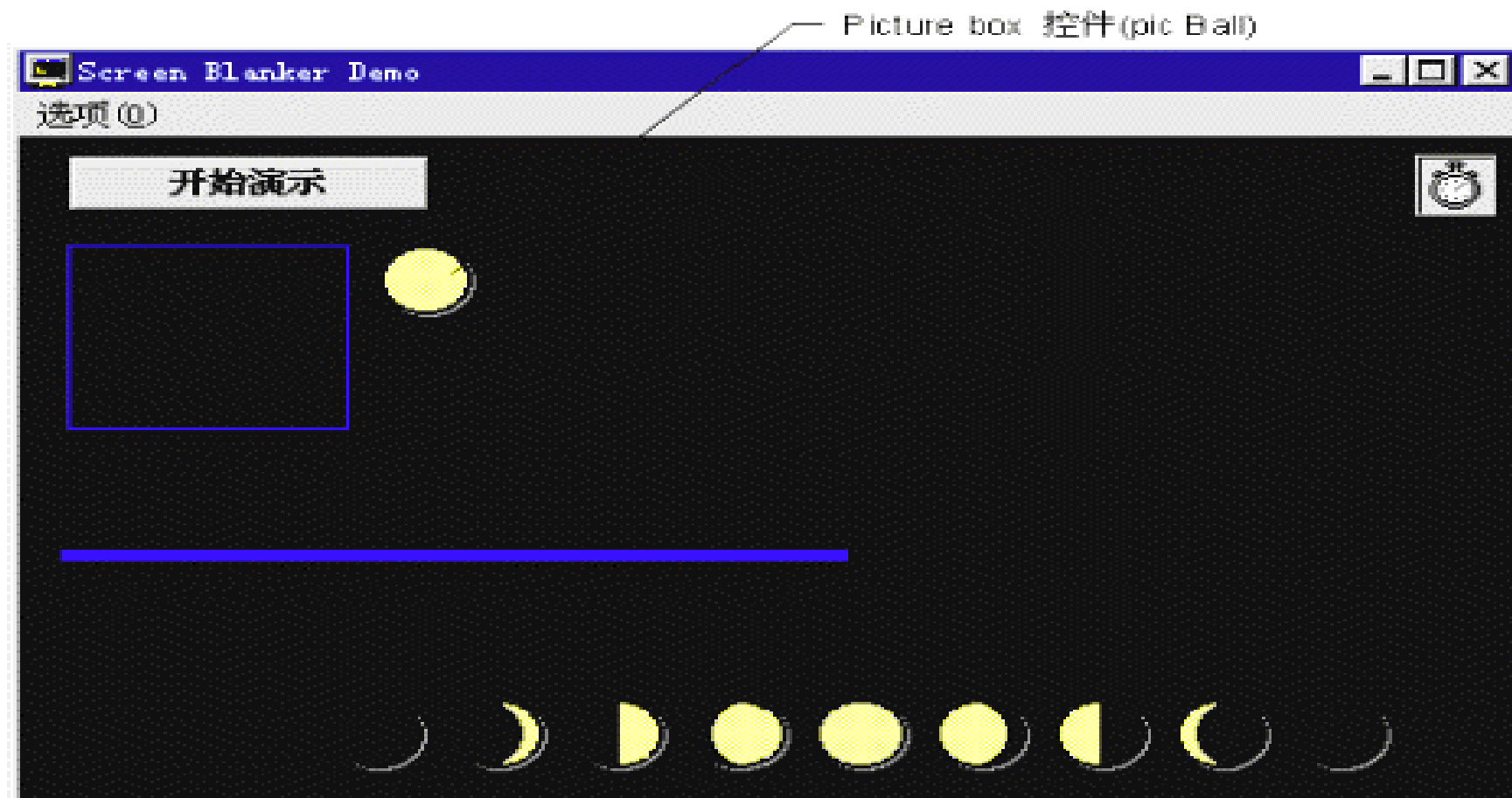


图 12.11 Blanker 应用程序中的图片框 (picBall)

picBall 图片框一直沿着选定方向移动，直到控件到达窗体四个边缘中的一个为止。然后图片框从它到达的边缘处改变方向；方向由变量 Motion 控制。例如，当图片框正在向左上方移动时，该过程的这一部分，将改变 Motion

的值，以指挥代码让 `picBall` 向另一个方向移动。

下述语句来自于 `Blanker` 应用程序中的 `CtlMoveDemo` 过程：

```
Select Case Motion
```

```
Case 1
```

```
    '若是向左上方运动，则使控件移动
```

```
    '    20 缇。
```

```
    picBall.Move picBall.Left - 20, picBall.Top - 20
```

```
    '若控件接触左边缘，则改变运动
```

```
    '    向右下方。
```

```
    If picBall.Left <= 0 Then
```

```
        Motion = 2
```

```
    '若控件接触上边缘，则改变运动
```

```
    '    向左下方。
```

```
    ElseIf picBall.Top <= 0 Then
```

```
        Motion = 4
```

```
    End If
```

应注意的是，使 `picBall` 移动的代码行，从它的 `Left` 和 `Top` 属性的当前值减 20 缇，以便建立控件的新位置。这可保证控件一直相对其当前位置移动。

控件移动的速度和光滑度，决定于 `Move` 方法中的缇（或其它单位）数。增加缇数，可提高移动的速度，却要降低移动的光滑度。降低缇数，要降低移动的速度，却可提高控件移动的光滑度。

详细信息 关于 Move 方法的补充内容，请参阅《Visual Basic 6.0 语言参考手册》的“Move 方法”。

动态地改变控件大小

在 Visual Basic 应用程序中，在运行时可以改变图片框、图像控件或窗体的大小和形状，就象改变其位置一样。

以下属性影响大小。

属性	应用于	描述
Align	PictureBox 和 Data 控件	如果设置图片框和窗体的顶部 (1) 或底部 (2) 对齐，该图片框的宽度等于窗体内部的宽度如果设置图片框和窗体的左边 (3) 或右边 (4) 对齐，该图片框的高度等于窗体内部的高度
Height	除 Timer、菜单和 Line 以外的所有窗体和控件	用窗体的刻度模式表示的对象高度（缺省是缇数）
Width	除 Timer、菜单和 Line 以外的所有窗体和控件	用窗体的刻度模式表示的对象宽度（缺省是缇数）
AutoSize	Label 和 PictureBox	如果为 True，则总是要使 Visual Basic 根据内容大小调节图片框的尺寸
Stretch	Image 控件	如果为 True，位图或元文件拉伸到适合图像控件的大小。如果为 False，图像控件的大小要与它包含的位图或元文件的大小相匹配

在该例中，每单击名为 `cmdGrow` 的命令按钮一次，该按钮就变大一些。

```
Private Sub cmdGrow_Click ()  
    cmdGrow.Height = cmdGrow.Height + 300  
    cmdGrow.Width = cmdGrow.Width + 300  
End Sub
```

创建简单动画

在运行时改变图片可以创建简单动画。最简单的方法是在两幅图片之间切换。也可通过几个框架使用一系列图片来创建动画。还可使图片动态地移动，能够创建出更精致的动画。

在两幅图片之间切换

一些图标可成对使用。例如，在 `\Icon` 子目录中有两个匹配的信封图标，一个信封未打开，一个已撕开，如图 12.12 中所示。通过在这两个图标之间进行转换，或称切换，可创建一个动画，以便显示邮件的状态。

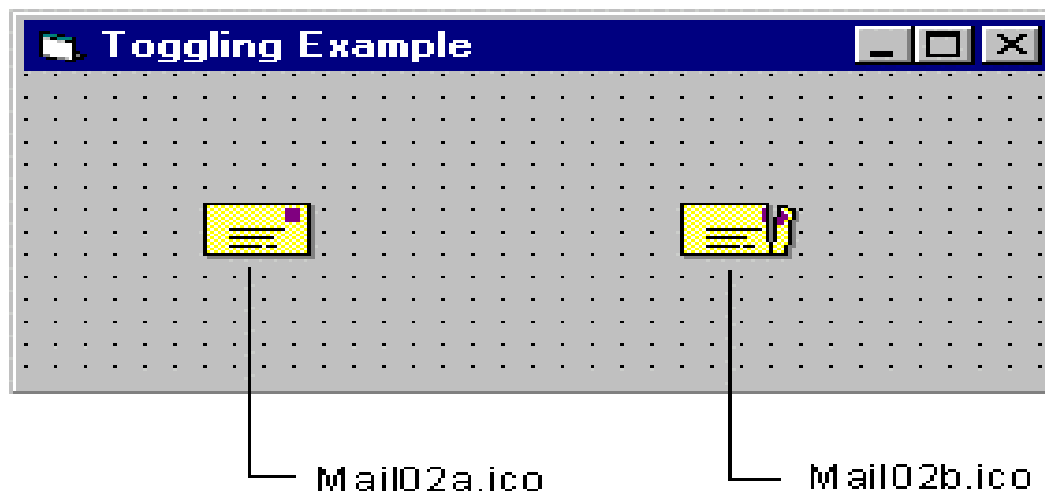


图 12.12 邮件图标

下列语句改变名为 `imgMailStatus` 的图标控件的 `Picture` 属性，以便将它的图片从一个未打开的信封切换到一个打开的信封。

```
imgMailStatus.Picture = imgMailOpen.Picture
```

在一些图片间轮转

通过在多个图片之间进行轮转，可以创建更长的动画。这项技术基本和在两幅图片之间切换一样，但是需要应用程序进行选择当前图像所用位图的操作。对动画中各个图片进行控制的一种方法，是使用控件数组。

详细信息 关于控件数组的详细信息，请参阅第九章“用对象编程”中的“创建对象数组”。

Blanker 示例应用程序包括有一个显示旋转月亮的动画。旋转月亮演示程

序，使用了包含有九个图像控件的数组创建该动画。为了查看控件数组中的图片之间是怎样工作的，运行时，可在“选项”菜单中，选择“Spinning Moon”，然后选择“启动演示程序”按钮，它将调用 ImageDemo 过程。

使用图形方法

除了图形控件之外，Visual Basic 还提供有创建图形的一些方法。总结在下表中的这些图形方法，适用于窗体和图片框。

方法	描述
Cls	清除所有图形和 Print 输出
Pset	设置各个像素的颜色
Point	返回指定点的颜色值
Line	画线、矩形或填充框
Circle	画圆、椭圆或圆弧
PaintPicture	在任意位置画出图形

注意 Print 方法也可认为是一种图形方法，因为它的输出也写在对象上，并象 PSet、Line 和 Circle 方法一样，也要以内存图像的方式进行保存（如果 AutoRedraw 是打开的）。关于 Print 方法的详细信息，请参阅本章前边的“在窗体和图片框上显示文本”。

图形方法的优点

在许多的工作需要使用图形控件的情况下，图形方法能工作得很好。例如，在图表中要创建网格线需要用到直线控件数组，而使用 `Line` 方法时却只需少量代码即可。当窗体改变大小时，跟踪数组中 `Line` 控件的位置，比起用 `Line` 方法简单地重新画线要麻烦得多。

当需要可视的效果直接显示在窗体上时，如，要显示出有关对话的色条，可以为这种临时性效果，编写几组代码行，而不是使用另一个控件。

图形方法提供了一些在图形控件无效的可视效果。例如，使用图形方法只能创建圆弧或画单个像素。用这些图形方法创建出的图形，显示在窗体上它们自己的那一层中。窗体中的这一层，在所有其它控件之下，所以，需要创建出现在应用程序中其它事物之下的图形时，这种方法就很好。

详细信息 请参阅本章前边的“用 `AutoRedraw` 和 `ClipControls` 使图形分层”。

图形方法的限制条件

用图形方法创建图形是在代码中进行的，这就意味着，必须运行应用程序才能看到图形方法的结果。因而，对于创建界面的简单设计元素来说，图形方法就不能代替图形控件的作用。设计时改变图形控件的外观，比修改并测试图形方法的代码容易。

详细信息 关于使用鼠标事件和 `Line` 或 `Move` 方法创建图形应用程序的

详细信息，请参阅第十一章“响应鼠标和键盘事件”中的“MouseDown 事件”、“MouseMove 事件”和“使用按钮增强图形鼠标应用程序”等部分。

图形方法画图的原理

每一种图形方法都是绘制输出到窗体上、图片框内或者是 Printer 对象。为了指示画出的位置，要给图形方法加上窗体或图片框控件的名字。如果省略了画出的对象，Visual Basic 就认为是要画在代码所连接的窗体上。例如，下列语句画一个点，在：

- 名为 MyForm 的窗体上
MyForm.PSet (500, 500)
- 名为 picPicture1 的图片框内
picPicture1.PSet (500, 500)
- 当前窗体上
PSet (500, 500)

每一个绘图区都有自己的坐标系，决定坐标使用的单位。另外，每一个绘图区都有自己的完整的一组图形属性。

详细信息 关于 Printer 对象的详细信息，请参阅本章稍后的“应用程序中的打印”部分。关于坐标的详细信息，请参阅“坐标系统概述”。

清除绘图区

在任何时候若想清除绘图区进行重画，应使用 `Cls` 方法。指定的绘图区以背景色 (`BackColor`) 重画：

`[object.]Cls`

使用没有指定 `object` 的 `Cls` 方法，将清除该代码所连接的窗体。

画点

单个像素的控制，是简单的图形操作。`PSet` 方法用来设置指定点处像素的色彩：

`[object.]PSet (x, y)[, color]`

`x` 和 `y` 参数是单精度参数，所以它们可以接受整数或分数的输入。输入可以是任何含有变量的数值表达式。

如果没有包括 `color` 参数，`PSet` 将像素设置为前景色 (`ForeColor`)。例如，下述语句在当前窗体（代码所连接的窗体），`MyForm` 和 `picPicture1` 上设置各种点：

`PSet (300, 100)`

`PSet (10.75, 50.33)`

`MyForm.PSet (230, 1000)`

`picPicture1.PSet (1.5, 3.2)`

添加 *color* 参数可提供更多控制：

'设置 50, 75 点为亮蓝色。

```
PSet (50, 75), RGB(0, 0, 255)
```

Blanker 应用程序是用随机选定的色彩画点，创建 Confetti 演示程序。

PSetDemo 过程创建 confetti:

```
Sub PSetDemo ()
```

```
    '设置红色为随机数。
```

```
    R = 255 * Rnd
```

```
    '设置绿色为随机数。
```

```
    G = 255 * Rnd
```

```
    '设置蓝色为随机数。
```

```
    B = 255 * Rnd
```

```
    '设置水平位置。
```

```
    XPos = Rnd * ScaleWidth
```

```
    '设置垂直位置。
```

```
    YPos = Rnd * ScaleHeight
```

```
    '用随机色画点。
```

```
    PSet (XPos, YPos), RGB(R, G, B)
```

```
End Sub
```

Confetti 的显示结果如图 12.13 所示：

为了“擦除”一点，只要把其颜色设置为背景色即可：

```
PSet (50, 75), BackColor
```

如同本章稍后的“画各种直线和形状”一节中描述的，可在 Step 之前加上 (x, y) 坐标，它可相对最后所画点的位置向下画点。

Point 方法和 PSet 方法是密切相关的，它只是返回指定位置处的颜色值：

```
PointColor = Point (500, 500)
```

详细信息 详细信息，请参阅《Visual Basic 6.0 语言参考手册》的“PSet 方法”和“Point 方法”。

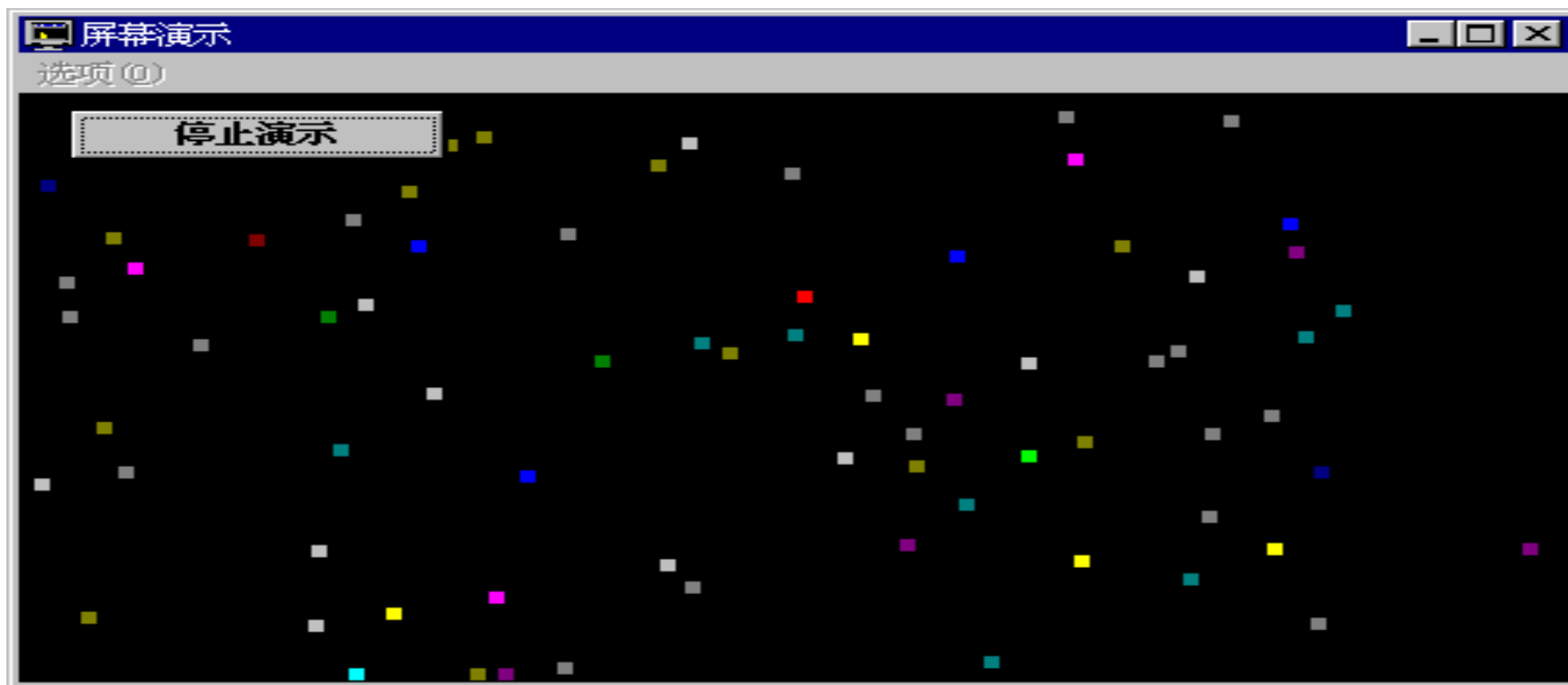


图 12.13 在 Blanker 应用程序中的 Confetti 显示

画各种直线和形状

尽管清除绘图区和画出单个的点是有用的，但最有趣的图形方法是画完整的直线和形状。

画直线

为了在两个坐标点之间画一条直线，可使用 **Line** 方法的简单形式，其语法如下：

```
[object.]Line [(x1, y1)-(x2, y2)[, color]
```

其中 *Object* 是可选的；若省略，此方法将在代码所连接的窗体（当前窗体）上画线。第一对坐标也是可选的。就所有的坐标值来说，*x* 和 *y* 参数都既可以是整数，也可以是分数。例如，下列语句可在窗体上画一条斜线。

```
Line (500, 500)-(2000, 2000)
```

Visual Basic 所画出的一条直线，要包括第一个端点，而不是最后一个端点。这种行为在从点到点画封闭曲线时是有用的。为了画最后一个端点，应使用这种语法：

```
PSet [Step] (0, 0)[, color]
```

第一对坐标 (*x1*, *y1*) 是可选的。如果省略这些坐标，将把该对象的 *x*, *y* 当前位置（画图坐标）作为端点。当前位置是由 **CurrentX** 和 **CurrentY** 属性指定的，然而在其它情况下会等于以前的图形方法或 **Print** 方法所画最后

点的位置。如果以前没有使用过图形方法或 `Print` 方法，或没有设置 `CurrentX` 和 `CurrentY` 属性，则缺省位置为对象的左上角。

例如，以下语句通过三点连接画出一个三角形。

'设置起点的 `x` 坐标。

```
CurrentX = 1500
```

'设置起点的 `y` 坐标。

```
CurrentY = 500
```

'向起点的右下方画一直线。

```
Line -(3000, 2000)
```

'向当前点的左方画一直线。

```
Line -(1500, 2000)
```

'向右上方画一直线到起点。

```
Line -(1500, 500)
```

图 12.14 显示运行的结果：

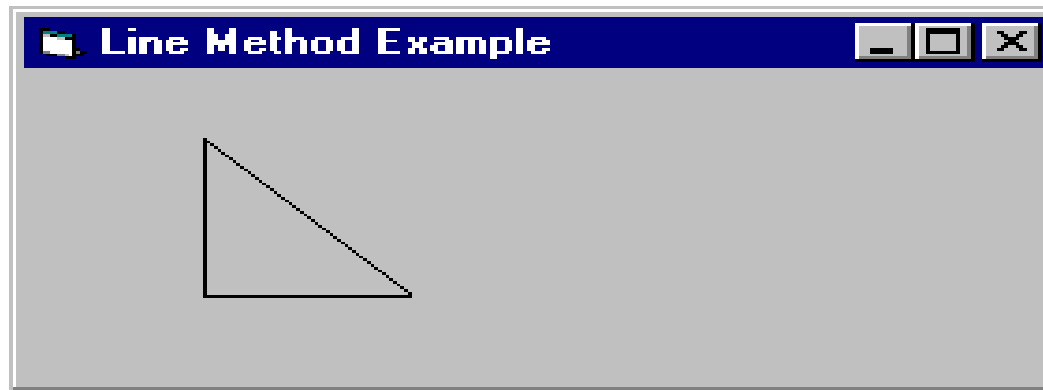


图 12.14 用 Line 方法画出的三角形

Blanker 应用程序使用 Line 方法创建出了一些有趣的图案样式。为了看到这些图案，可从“选项”菜单中，选定“Crossfire”，然后选定“Start Demo”按钮，即可看到。

Step 关键字

PSet、Line 和 Circle 方法使用以下语法可指定一个或多个点：

(x, y)

可在每个点之前加上 Step 关键字，用来指定要画出的点，是相对最后画出点的位置。Visual Basic 要将 x 和 y 的值加到最后所画的点上。例如，下边这条语句：

```
Line (100, 200)-(150, 250)
```

等价于：

```
Line (100, 200)-Step(50, 50)
```

在许多情况下，**Step** 关键字可免除持续不断地记录最后所画点位置的负担。经常最为关心的可能是两点的相对位置，而不是它们的绝对位置。

使用颜色参数

为了改变直线的颜色，应将可选的 *color* 参数与图形方法一起使用。例如，下述语句将画一条深蓝色的直线：

```
Line (500, 500)-(2000, 2000), RGB(0, 0, 255)
```

如果省略了 *color* 参数，将使用在其上画线的对象的 **ForeColor** 属性，来决定直线的颜色。

绘制方框

可使用 **Line** 方法绘制并填充方框。下列示例画出一个方框，其左上角为 (500, 500)，每边长为 1,000 缇：

```
Line (500, 500)-Step(1000, 0)
```

```
Line -Step(0, 1000)
```

```
Line -Step(-1000, 0)
```

```
Line -Step(0, -1000)
```

然而，**Visual Basic** 提供了一个更简单的画方框的方法。当在 **Line** 方法中

使用 **B** 选项时，**Visual Basic** 把指定点作为矩形的对角点，可画出一个矩形。所以，可用下列语句，代替上例中的四个语句：

```
Line (500, 500)-Step(1000, 1000), , B
```

注意，在 **B** 之前要有两个逗号，以表示色彩参数被省略了。**Line** 方法的语法包括在本章稍前的“画各种直线和形状”一节中。

FillStyle 和 FillColor

只要不改变 **FillStyle** 属性的设置值，所画的方框将是空的（该方框使用缺省的 **FillStyle** 和设置值来进行填充，不过 **FillStyle** 的缺省值是 1-透明）。可将 **FillStyle** 属性改变为下表列出的任意设置值。

设置值	描述
0	实心的。可用 FillColor 属性设置的色彩填充方框
1	透明（缺省值）。无论使用何种色彩，图形对象表现为空
2	水平线
3	垂直线
4	向上的对角线
5	向下的对角线
6	十字交叉线
7	对角交叉线

所以，将 **FillStyle** 设置为 0，就可以用 **FillColor** 属性的色彩设置，将方框填充为实心。

另一种填充方框的方法，是在 B 之后指定 F（注意，没有 B 就不能用 F）。当使用 F 选项时，Line 方法将忽略 FillColor 和 FillStyle 属性。使用 F 选项时，方框总是被填充为实心。下述语句使用具有 ForeColor 属性的一种实心图样，对方框进行填充。

```
Line (500, 500)-Step(1000, 1000), , BF
```

输出结果显示在图 12.15。

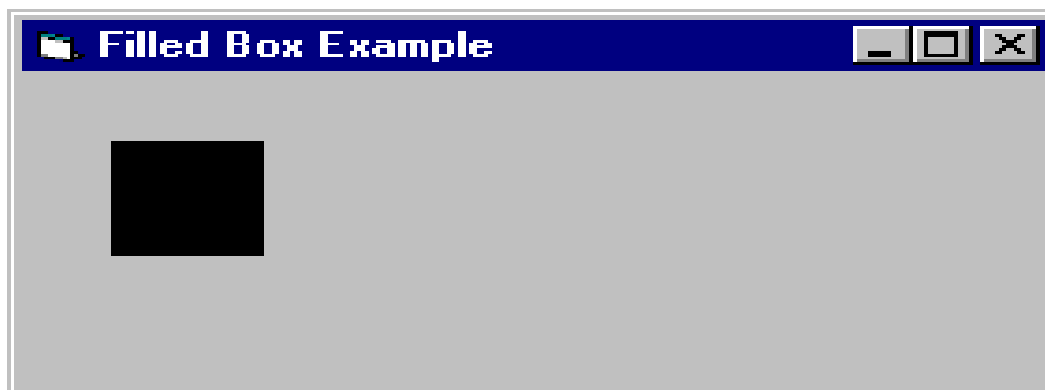


图 12.15 用实心图样填充的方框

画圆

Circle 方法可画出圆形和椭圆形的各种形状。另外，Circle 方法还可以画出圆弧（圆的一部分）和楔形饼块。使用变化的 Circle 方法，可画出多种曲线。

为了画圆，Visual Basic 需要给出该圆的圆心位置和它的半径。画一个理想圆的语法是：

```
[object.]Circle [Step](x, y), radius[, color]
```

方括号表明 *object* 和 **Step** 关键字是任选项。如果不指定 *object*，则认为指定的就是当前的窗体。*x* 参数和 *y* 参数是圆心的坐标，*radius* 参数是圆的半径。例如，下面语句将画出一个以 (1200,1000) 为圆心、750 为半径的圆：

```
Circle (1200, 1000), 750
```

这个语句产生的精确作用，还要依赖于窗体的大小和窗体的坐标系。如果不知道窗体的大小，也就不能知道这个圆是否可见。可使用绘图区的标尺属性，将该圆的圆心置于窗体的中心处：

```
Circle ((ScaleWidth + ScaleLeft) / 2, (ScaleHeight + _  
    ScaleTop) / 2), ScaleWidth / 4
```

此时，关于 ScaleWidth 和 ScaleHeight 所有应知道的是，它们能帮助将图形定位于窗体的中心。

详细信息 在本章前面“改变对象的坐标系统”中详细讨论了 ScaleWidth 和 ScaleHeight 属性。

注意 圆的半径通常是按照水平单位来指定的。如果坐标系使用的水平单位和垂直单位是相同的话（按照缺省规定，应当这样），可以不用考

虑单位问题。但如果使用的是自定义标尺，则水平单位和垂直单位可能对应不同的长度。上例中，半径是用水平单位来指定的，可以保证圆的实际高度和实际宽度相等。

Blanker 应用程序创建了一些圆圈，作为多彩地毯演示程序的一部分。该演示程序围绕窗体中心画出一系列虚线圆。这些圆看起来正好象一个编织成的圆形地毯。**CircleDemo** 过程使用下列语句，在多彩地毯演示程序中画圆：

```
Sub CircleDemo ()
```

```
    Dim Radius
```

```
    '将红色设置为随机数。
```

```
    R = 255 * Rnd
```

```
    '将绿色设置为随机数。
```

```
    G = 255 * Rnd
```

```
    '将蓝色设置为随机数。
```

```
    B = 255 * Rnd
```

```
    '将 x 坐标设置在窗体中间。
```

```
    XPos = ScaleWidth / 2
```

```
    '将 y 坐标设置在窗体中间。
```

```
    YPos = ScaleHeight / 2
```

```
    '将半径设置在窗体高度的 0 到 50% 之间。
```

```
    Radius = ((YPos * 0.9) + 1) * Rnd
```

```
    '用随机颜色画圆。
```

```
    Circle (XPos, YPos), Radius, RGB(R, G, B)
```

End Sub

多彩地毯演示程序的结果，显示在图 12.16 中。

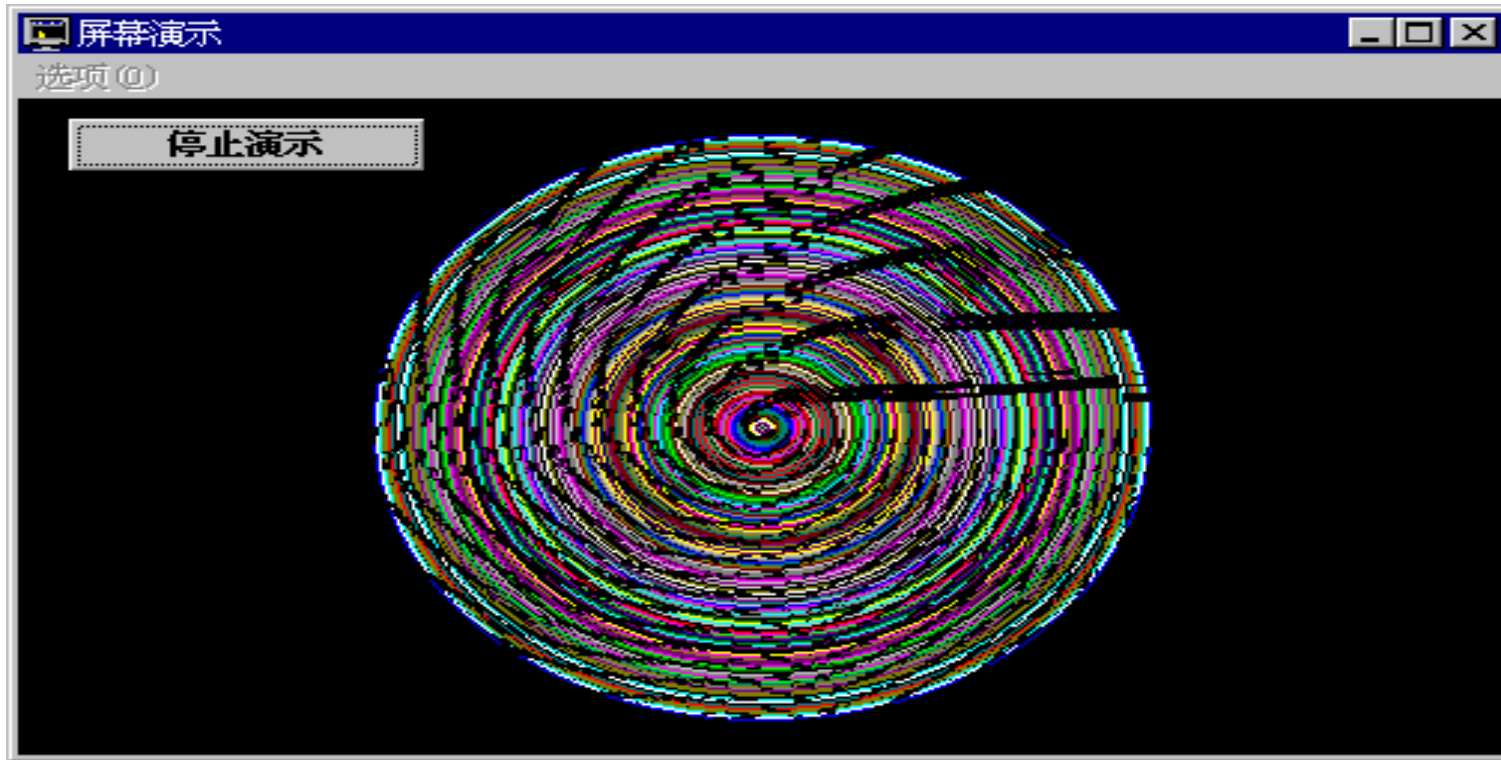


图 12.16 Blanker 应用程序里的多彩地毯演示程序

画圆弧

为了用 Circle 方法画出圆弧，应以弧度为单位，给出定义弧线 start 和 end

的角度参数。画圆弧的语法是：

`[object.]Circle [Step](x, y), radius, [color], start, end[, aspect]`

如果 `start` 参数或 `end` 参数是负数的话，Visual Basic 将画一条连接圆心到负端点的线。例如，下面的过程将用消除切片方式画出一个饼。

```
Private Sub Form_Click ()
```

```
    Const PI = 3.14159265
```

```
    Circle (3500, 1500), 1000, , -PI / 2, -PI / 3
```

```
End Sub
```

注意 将度换算成弧度的公式是将度数乘以 $\text{Pi}/180$ 。

画椭圆

圆的方位比，决定了圆是显得很圆（圆）还是被拉长（椭圆）。Circle 方法的完整语法是：

`[object.]Circle [Step](x, y), radius, [color], [start], [end] [, aspect]`

`start` 参数和 `end` 参数是可选的；但是，如果想省掉参数，逗号是不能省的。例如：如果含有 `radius` 参数和 `aspect` 参数，省掉了 `color`、`start`、`end` 三个参数，则必须加上四个连续的逗号，它表明这三个参数被省掉了。

```
Circle (1000, 1000), 500, , , , 2
```

`aspect` 参数指定了水平长度和垂直长度比。该 `aspect` 参数是正浮点数，这说明它既可以是整数表达式，也可以是小数表达式，但不能是负数。该 `aspect`

参数较大时，椭圆沿垂直轴线拉长；相反，该 *aspect* 参数较小时，则沿水平轴线拉长。由于椭圆有两个半径——水平方向的 *x* 半径和垂直方向 *y* 半径——Visual Basic 在 `Circle` 语句中只使用了一个参数 *radius* 对应椭圆的长轴。如果 *aspect* 参数小于 1 的话，该 *radius* 半径指的是水平方向的 *x* 半径，如果 *aspect* 参数大于或等于 1 的话，则 *radius* 指的是垂直方向的 *y* 半径。

注意 *aspect* 参数指定的总是水平长度和垂直长度的实际物理距离比，要保证这一点（即便使用的是自定义标尺），半径应以水平单位指定。

下列过程说明的是，不同的 *aspect* 参数值，`Circle` 方法是如何决定对 *radius* 参数的使用的；是将它作为椭圆的 *x* 半径，还是作为椭圆的 *y* 半径：

```
Private Sub Form_Click ()
```

```
'画一个实心椭圆。
```

```
    FillStyle = 0
```

```
    Circle (600, 1000), 800, , , , 3
```

```
'画一个空心椭圆。
```

```
    FillStyle = 1
```

```
    Circle (1800, 1000), 800, , , , 1 / 3
```

```
End Sub
```

结果显示在图 12.17 中。

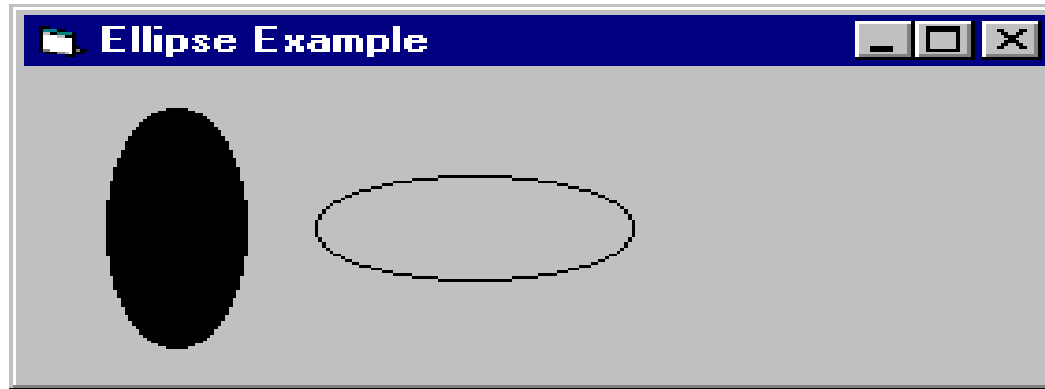


图 12.17 使用 Circle 方法画出的椭圆

详细信息 有关画圆和画圆弧的更多信息，请参见本章前面的“画圆”。

在任意位置绘制图形

使用 `PaintPicture` 方法，可以在窗体、图片框和 `Printer` 对象上的任何地方绘制图形。`PaintPicture` 方法的语法是：

```
[object.]PaintPicture pic, destX, destY[, destWidth[, destHeight[, srcX _  
[, srcY[, srcWidth[, srcHeight[, Op]]]]]]]
```

目标 *object* 指的是窗体、图片框或 `Printer` 对象，这些地方都是 *pic* 图片表现的处所。如果 *object* 被忽略了，则认为指定的就是当前的窗体。*pic* 参数必须是一个图片对象，它是由窗体或控件的 `Picture` 属性决定。

destX 和 *destY* 参数，是按照 *object* 的 `ScaleMode`，该图像将出现的水平和垂直位置。*destWidth* 和 *destHeight* 参数是可选项，用来设置在 *object* 目标

中该图像的宽度和高度。

srcX 和 *srcY* 参数是可选项，用来定义 *pic* 中裁剪区左上角的 *x* 和 *y* 坐标。可选的 *op* 参数用来定义当在目标 *object* 上绘图时，在图片上执行的光栅操作（例如，AND 和 XOR）。

PaintPicture 方法可代替 BitBlt Windows API 函数，在将矩形图形块从一个地方移到任意另一地方时，它可执行广泛的各种操作。

例如，可以使用 PaintPicture 方法生成同一位图的多份副本，并将它们平铺在窗体上。使用这种方法，比在窗体上移动图片控件要快。下列代码是用来平铺图片控件的 100 份拷贝，并且通过给 destWidth 设置一个负值，可以使每张图片进行水平翻转。

```
For i = 0 To 10
    For j = 0 To 10
        Form1.PaintPicture picF.Picture, j * _
            picF.Width, i * picF.Height, _
            picF.Width, -picF.Height
    Next j, i
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“PaintPicture 方法”。

指定线宽

DrawWidth 属性用来指定图形方法输出时线的宽度，BorderWidth 属性用来指定直线和形状控件轮廓线的粗细。

下列过程将画出几条不同宽度的线。

```
Private Sub Form_Click ()  
    DrawWidth = 1  
    Line (100, 1000)-(3000, 1000)  
    DrawWidth = 5  
    Line (100, 1500)-(3000, 1500)  
    DrawWidth = 8  
    Line (100, 2000)-(3000, 2000)  
End Sub
```

结果显示在图 12.18 中。

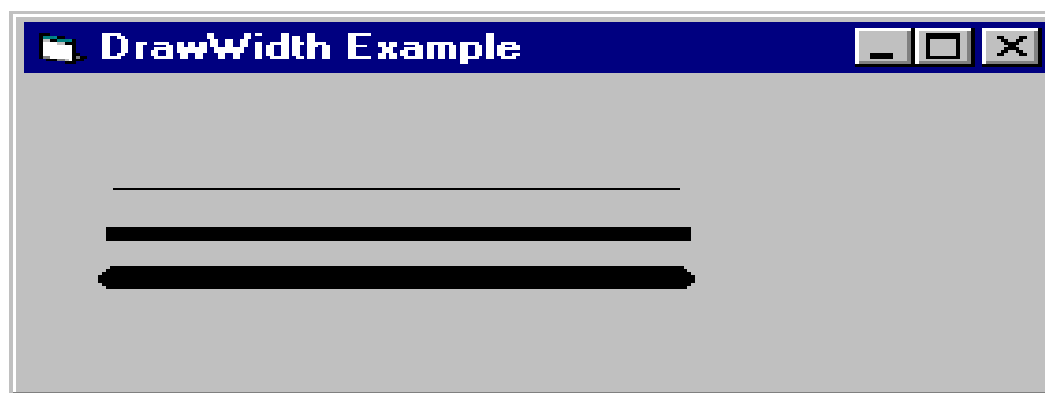


图 12.18 改变 DrawWidth 属性产生的效果

图 12.19 显示具有不同的 BorderWidth 值的三种形状控制。

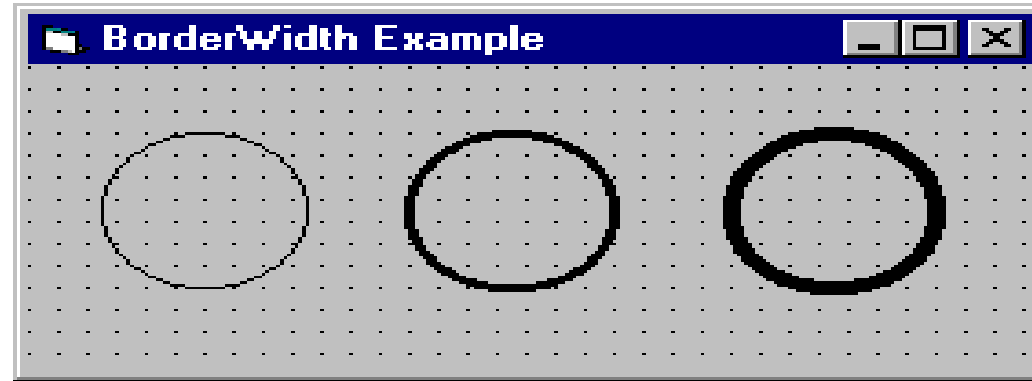


图 12.19 改变 BorderWidth 属性产生的效果

指定实线或虚线

DrawStyle 属性指定用图形方法创建的线，是实线还是虚线。形状控件的 BorderStyle 属性的作用与 DrawStyle 属性相同，但是仅应用于某些对象。

注意 形状控件的 BorderStyle 属性出于不同的使用目的，要使用来自其它的控件和窗体中 BorderStyle 属性的不同设置。形状控件或直线控件的 BorderStyle 属性出于不同的使用目的，要使用来自其它对象的 BorderStyle 属性的不同设置。对于形状控件和线控件来说，BorderStyle 属性的工作象本节所述的 DrawStyle 属性一样。对于窗体和其它控件来说，BorderStyle 属性取决于该窗体或控件是否有边界，如果有，将决定其边界是固定的还是可调整的。

实线和内侧实线方式

内侧实线方式（`DrawStyle` 或 `BorderStyle` 的设置等于 6）和实线方式非常接近。它们都是要画出一条实线。当用宽线来画方框或形状控件时，这两种设置的差别便可明显看出。在这些情况下，实线方式画的线一半在方框或形状的里边，一半在方框或形状的外边，而内侧实线方式画的线是全部在方框或形状的里边。关于如何画方框，请参见本章前面“画方框”。

下列过程演示了 `DrawStyle` 属性所支持的各种设置值，它建立了一个循环，设置值从 0 到 6，每次一个值。结果显示在图 12.20 中。

```
Private Sub Form_Click ()  
    Dim I As Integer, Y As Long  
    For I = 0 To 6  
        DrawStyle = I  
        Y = (200 * I) + 1000  
        Line (200, Y)-(2400, Y)  
    Next I  
End Sub
```

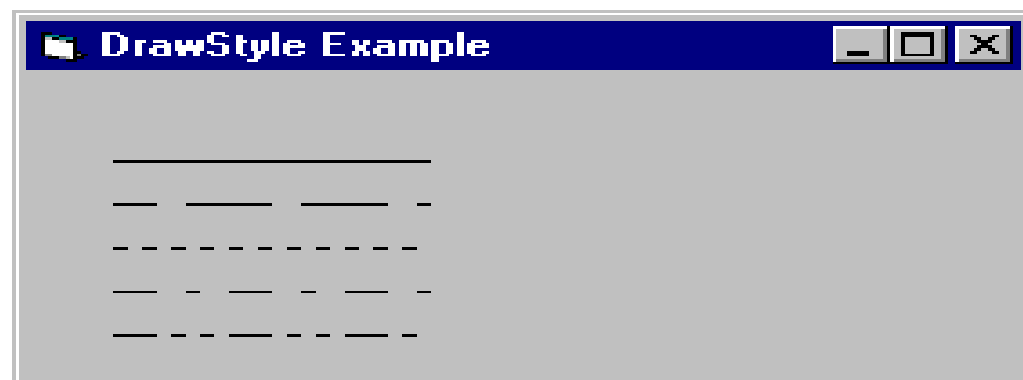


图 12.20 改变 DrawStyle 属性产生的结果

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“DrawStyle 属性”或“BorderStyle 属性”。

DrawMode 的显示控制作用

DrawMode 属性决定，当把一个样式画在另一个之上时，能够取得什么结果。尽管改变 DrawMode 属性，经常会带来一些影响（尤其是彩色系统），但是当在一个空白或是纯白色的背景上，或者是在未定义颜色的背景上绘制时，是没有必要使用该属性的。

可以将 DrawMode 设置为从 1 到 16 的值。下表中是常用的设置值。

设置值	描述
4	Not Copy 笔。画出相反的直线样式，不管该处原来的存在如何
7	Xor 笔。显示出直线样式和现存显示的区别，参阅本节后面的解释。用这种模式绘制对象两次，将精确地恢复该处原来的背景
11	无操作，效果相当于关掉了绘画
13	Copy 笔（缺省值）。使用直线的样式，不管该处原来的存在如何

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“DrawMode 属性”。

Xor 笔

对于动画来说，7 是 DrawMode 的一个有用的设置值。一条线画出两次，就可精确还原该线画出之前的显示内容。这样就可能创建一个在背景上移动却不会使其毁坏的对象，因为在移动时，可以还原背景。大多数模式不能保护原来的背景。

例如，下列代码的作用是：每单击一下鼠标就移动一个圆，无论圆底下有什么，它都能被还原。

```
Private Sub Form_Click ()
    ForeColor = 255 : DrawMode = 7
    Circle (CurrentX, CurrentY), 1000
    CurrentX = CurrentX + 220
    CurrentY = CurrentY + 220
    Circle (CurrentX, CurrentY), 1000
```

End Sub

Xor 笔的绘图模式（以及大多数其它的 **DrawMode** 设置值），是通过将绘图样式（被称作“笔”）中的各个像素与存在区（被称作“目标”）中的相应像素进行比较的方式工作的。在单色系统中，像素不是被打开就是被关闭，**Visual Basic** 进行一种简单的逻辑比较：如果笔和目标中有一个打开，而不是两个同时都打开时，则使像素打开。

在彩色系统中，每一个像素都赋给一个色值。对于 **DrawMode** 的 **Xor** 笔这样的设置值来说，**Visual Basic** 将对笔和目标里相应的每一对像素进行比较，执行二进制（位方式）的比较操作。结果决定生成像素的色值，如图 12.21 中所示。

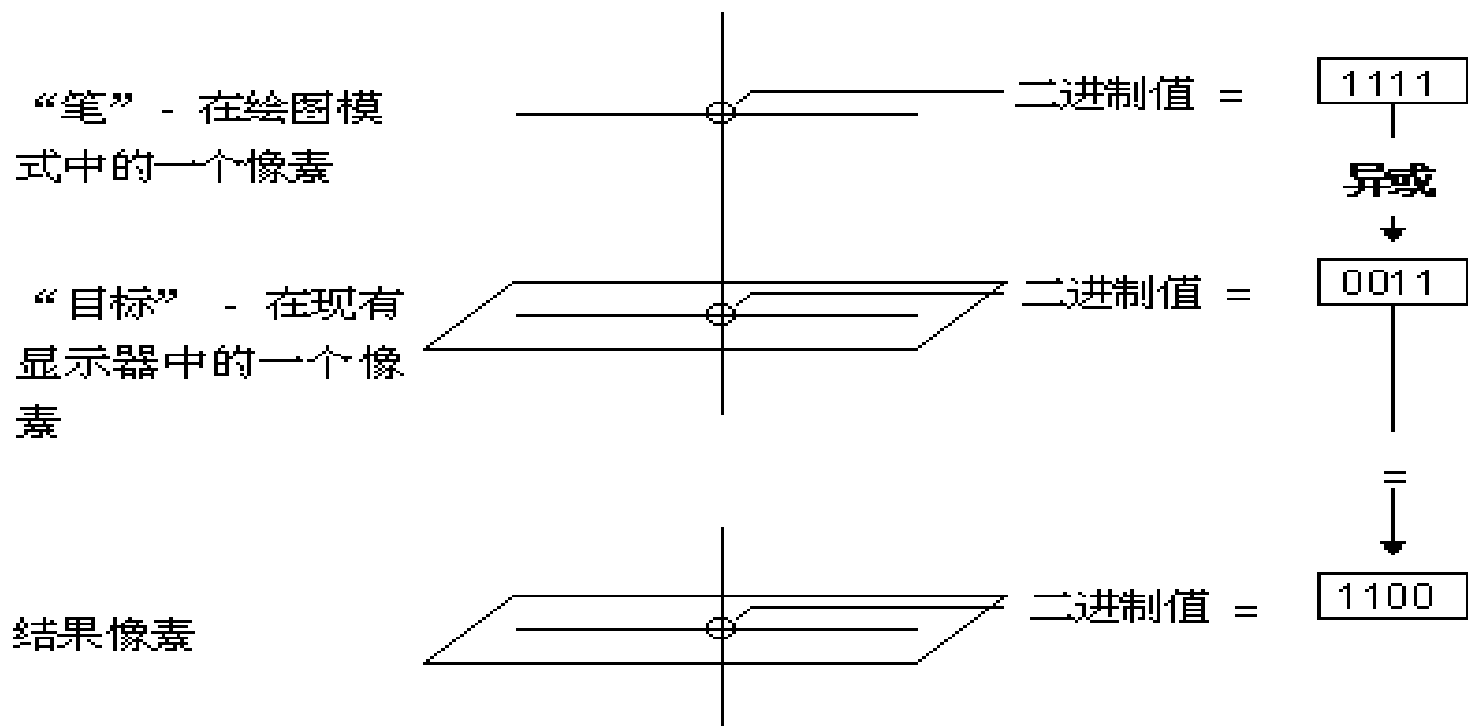


图 12.21 使用 Xor 笔给直线像素设置二进制值

窗体加载时创建图形

当窗体加载时为了创建其上显示的图形，应考虑将各种图形方法安放在 `Form_Paint` 事件中。`Form_Paint` 图形在每一个绘画事件中将自动地被重画。如果将图形放入 `Form_Load` 事件中，则应将窗体上的 `AutoRedraw` 属性设置为 `True`。在这种情况下，`Form_Load` 将显示窗体并画上图形。请记

住，在 `Form_Load` 事件中窗体是不可见的。因为 `Visual Basic` 对于不可见窗体上的图形方法不进行处理，除非 `AutoRedraw` 被设置为 `Ture`，否则 `Form_Load` 事件中的图形方法将被忽略。

使用颜色

对于所有的颜色属性和图形方法，`Visual Basic` 使用固定的系统。每种颜色都由一个 `Long` 整数表示，并且在指定颜色的所有上下文中，该数值的意义相同。

在运行时指定颜色

在运行时有四种方式可指定颜色值：

- 使用 `RGB` 函数。
- 使用 `QBColor` 函数，选择 16 种 `Microsoft QuickBasic(r)` 颜色中的一种。
- 使用在“对象浏览器”中列出的内部常数之一。
- 直接输入一种颜色值。

本节讨论如何使用 `RGB` 和 `QBColor` 函数，作为指定颜色的一种简单方法。关于使用常数定义颜色或直接输入颜色值的信息，请参阅本章稍后的“使用颜色属性”部分。

使用 RGB 函数

可以用 RGB 函数来指定任何颜色。

为了用 RGB 函数指定颜色

1. 要对三种主要颜色（红、绿、蓝）中的每种颜色，赋给从 0 到 255 中的数值，0 表示亮度最低，而 255 表示亮度最高。
2. 使用红-绿-蓝的排列方式，将三个数值输入给 RGB 函数。
3. 将结果赋给颜色属性或颜色参数。

每一种可视的颜色，都由这三种主要颜色组合产生。例如：

'设定背景为绿色。

```
Form1.BackColor = RGB(0, 128, 0)
```

'设定背景为黄色。

```
Form2.BackColor = RGB(255, 255, 0)
```

'设定背景为深蓝色。

```
PSet (100, 100), RGB(0, 0, 64)
```

详细信息 要想对 RGB 函数了解更多，请参阅《Visual Basic 6.0 语言参考手册》的“RGB 函数”。

使用颜色属性

Visual Basic 中的许多控件，有决定控件显示颜色的属性。请记住，这些属

性中有些也适用于不是图形的控件。下表描述了这些颜色属性。

属性	描述
BackColor	对用于绘画的窗体或控件设置背景颜色。如果在绘图方法进行绘图之后改变 BackColor 属性，则已有的图形将会被新的背景颜色所覆盖
ForeColor	设置绘图方法在窗体或控件中创建文本或图形的颜色。改变 ForeColor 属性不影响已创建的文本或图形
BorderColor	给形状控件边框设置颜色
FillColor	为用 Circle 方法创建的圆和用 Line 方法创建的方框，设置填充颜色

详细信息 想了解有关这些颜色属性的更多内容，请参阅《Visual Basic 6.0 语言参考手册》的“BackColor 属性”、“ForeColor 属性”、“BorderColor 属性”以及“FillColor 属性”等部分。

定义颜色

颜色属性可使用几种方法定义颜色值。在“使用颜色”一节所述的 RGB 函数，就是定义颜色的一种方法。这部分介绍的是另外两种定义颜色的方法：

- 使用定义的常数
- 直接使用颜色设置值

使用定义的常数

当使用“对象浏览器”中列出的内部常数时，没有必要去了解这些常数是如何产生的。另外，这些内部常数也无需声明。例如：无论什么时候想指

定红色，作为颜色参数或颜色属性的设置值，都可以使用常数 `vbRed`：

```
BackColor = vbRed
```

直接使用颜色设置值

使用 `RGB` 函数来指定颜色和用内部常数来指定颜色，都不是直接的，因为 `Visual Basic` 只是将它们解释为与它所代表的颜色较接近的一种颜色。如果自己清楚知道 `Visual Basic` 是如何用数值来指定颜色的话，就可以给颜色参数和属性指定一个值，这样能直接指定颜色。多数情况下，用十六进制数输入这些数值更简单。

正常的 `RGB` 颜色的有效范围，是从 0 到 16,777,215 (`&HFFFFFF&`)。每种颜色的设置值（属性或参数）都是一个四字节的整数。对于这个范围内的数，其高字节都是 0，而低三个字节，从最低字节到第三个字节，分别定义了红、绿、蓝三种颜色的值。红、绿、蓝三种成分都是用 0 到 255 (`&HFF`) 之间的数表示。

因此，可以用十六进制数按照下述语法来指定颜色：

```
&HBBGGRR&
```

`BB` 指定蓝颜色的值，`GG` 指定绿颜色的值，`RR` 指定红颜色的值。每个数段都是两位十六进制数，即从 00 到 FF。中间值是 80。因此，下面的数值是这三种颜色的中间值，指定了灰颜色：

```
&H808080&
```

将最高位设置为 1，就改变了颜色值的含义：颜色值不再代表一种 `RGB` 颜

色，而是一种从 Windows “控制面板”指定的环境范围颜色。这些数值对应的系统颜色范围是从 &H80000000 到 &H80000015。

注意 尽管可以指定 1,600 万种以上的不同颜色，但并不是所有的系统都能精确地显示出来。关于 Windows 如何指定颜色的详细内容，请参阅本章后面的“使用 256 种颜色”。

使用系统颜色

在应用程序中设置控件或窗体的颜色时，可以不指定颜色值，而用操作系统指定的颜色。如果指定了操作系统的颜色，当应用程序的用户改变计算机上的系统颜色值时，应用程序将自动地反映用户所指定的颜色值。

每一种系统颜色，既有所定义的常数也有直接的颜色设置值。对系统颜色来说，其直接颜色设置值的高位字节与普通 RGB 颜色的高位字节是不同的。对于 RGB 颜色来说，其高位字节为 0，而对于系统颜色来说，其高位字节为 80，剩下的数字则指的是某一特定的系统颜色。例如：

&H80000002& 这个十六进制数，是用来指定一个活动窗口的标题颜色。

设计时，通过属性窗口选择颜色属性时，选择“系统”选项卡，就能够选择系统设置值，可自动转换成十六进制值。也可在“对象浏览器”中寻找系统颜色的定义常数。

使用 256 种颜色

基于具有能处理 256 色或 256 色以上的视频适配器和显示驱动程序的系统，Visual Basic 可支持 256 种颜色。对于多媒体应用程序，或对于那些需要显示接近相片质量图像的应用程序来说，能同时显示 256 色的功能，是特别有价值的。

在以下场合，可显示 256 色的图像，也可为图形方法定义高达 256 色：

- 窗体
- PictureBox
- Image 控件（仅显示图像）

注意 Windows 的元文件，不支持 256 种色。Visual Basic 用缺省的 VGA 16 色的调色板来显示图元文件。

调色板

在 Visual Basic 应用程序中，调色板提供了支持 256 种颜色的基础。在关于调色板的讨论中，了解不同类型调色板之间的关系很重要。硬件调色板包含了 256 个记录项，它们定义了将在屏幕上显示的实际 RGB 值。系统中间色调的调色板是一套预定义的 256 种 RGB 值，对 Windows 本身有效。逻辑调色板是一套可到达 256 种 RGB 的值，它们包含在位图或其它图像中。

Windows 可使用硬件调色板中的 256 种颜色绘图。其中有 20 种颜色被称为静态颜色，是由系统保护的，应用程序不能将其改变。静态颜色包括缺省的 VGA 调色板中的 16 种颜色（与 Visual Basic QBColor 函数定义的

颜色一样)，外加四种不同深度的灰色。系统中间色调的调色板，通常包含这些静态颜色。

前景窗口（有焦点的窗口），将决定硬件调色板中剩下的 236 种非静态颜色。每当硬件调色板改变时，使用这些颜色的所有背景窗口都要重画。如果背景窗口的逻辑调色板的颜色与当前硬件调色板的颜色不完全匹配，Windows 会赋予一个最接近的匹配值。

256 色图像的显示

如果显示硬件和软件能支持 256 种颜色的话，窗体、图片框和图像控件能自动地用 256 色显示图像。如果用户系统支持的颜色少于图像所需的颜色，Visual Basic 尽可能地将所需颜色映射到最相近的颜色上。

对于真彩色（1,600 万种颜色）的显示，Visual Basic 总是使用正确的颜色。对于单色或 16 色的显示，Visual Basic 会使背景色和用 FillColor 属性设置的颜色抖动。抖动是用来模拟从视频适配器和显示驱动设备得不到的颜色的过程。

用调色板绘图

对于 256 色的视频驱动程序，通过图形方法可使用高达 256 种的颜色。按照缺省规定，Visual Basic 中有效的 256 色是系统中间色调的调色板中的颜色。尽管可用 RGB 函数来指定一种具体的颜色，但实际显示的颜色，是从系统中间色调的调色板中得到的最接近匹配的颜色，这一点显示在图

12.22 中。

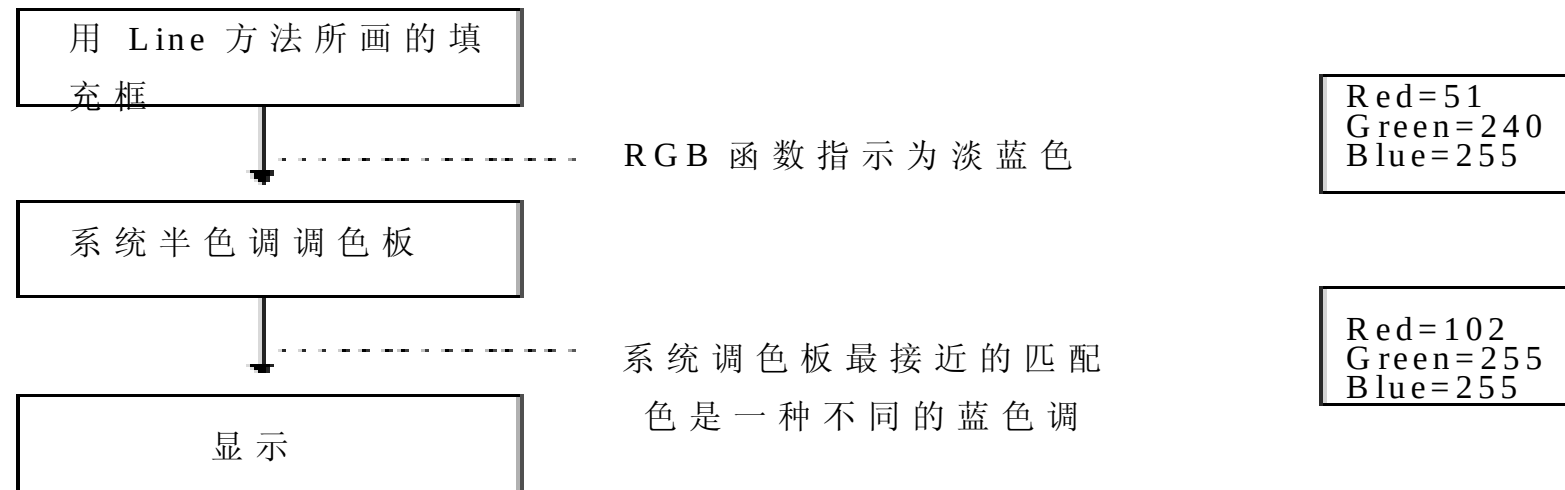


图 12.22 从指定色到显示的颜色匹配方式

尽管缺省调色板对于 Visual Basic 来说是系统中间色调的调色板，但使用窗体、用户控件和用户文档的 `PaletteMode` 属性和 `Palette` 属性，也可控制颜色的显示。在这种情况下，除了颜色要在硬件调色板中与最接近色匹配之外，颜色的匹配关系是一样的。

管理多个调色板

使用调色板时，应记住对很多显示来说，屏幕上同时最多只能显示 256 种颜色。

当应用程序中使用一种以上的彩色调色板时，此项限制就变得非常重要。例如，在单个窗体上，可在图像控件中显示一个 256 色位图，同时在图片框中显示第二个图像。如果这两个图像的逻辑调色板包含的 256 种颜色不完全相同，Windows 就必须决定先将哪一个逻辑调色板的颜色调到硬件调色板中。记住，硬件调色板决定了屏幕上实际显示的颜色。

当 Visual Basic 的应用程序有两个以上的窗体的逻辑调色板互不相同同时，也会发生类似的情况。哪个窗体接收到焦点，它的逻辑调色板就控制硬件调色板。这样常会导致显示效果比在 256 色系统下的最优显示要差一些。作为 Visual Basic 程序员，可通过使用 `PaletteMode` 属性来控制硬件调色板。

PaletteMode 属性

当设计可在 256 色系统下运行的应用程序时，通过设置窗体、用户控件或用户文档的 `PaletteMode` 属性，能控制 Windows 选择显示颜色的途径（用户控件和用户文档只在专业版和企业版中有效。）。窗体、用户控件或用户文档中含有的所有控件，都要根据 `PaletteMode` 的设置来显示。下表列出了 `PaletteMode` 的有效设置：

模 式	常 数	应 用 到
Halftone	vbPaletteModeHalftone	窗 体 ， 用 户 控 件 ， 用 户 文 档
UseZOrder	vbPaletteModeUseZOrder	窗 体 ， 用 户 控 件 ， 用 户 文 档
Custom	vbPaletteModeCustom	窗 体 ， 用 户 控 件 ， 用 户 文 档
Container	vbPaletteModeContainer	用 户 控 件
None	vbPaletteModeNone	用 户 控 件
Object	vbPaletteModeObject	含 有 调 色 板 的 ActiveX 设 计 器

PaletteMode 属性只应用于 256 色的显示。在增强色或真彩色显示中，可通过视频驱动程序分别使用 32,000 色或 16 兆色的调色板，来控制颜色的选择。即使在一个以增强色或真彩色显示的系统中设计程序时，还是要设置 PaletteMode，因为许多用户可能用的是 256 色显示。

PaletteMode 属性可在设计时通过“属性”窗口来设置，也可在运行时通过代码来改变。调色板示例应用程序使用几种不同的 PaletteMode 设置，显示出带有不同调色板图像的显示效果。

注意 对于 Visual Basic 的老版本，PaletteMode 相当于 UseZOrder。

中间色调的 **PaletteMode**

窗体和用户文档的缺省模式为中间色调。在这种模式下，包含在窗体中的任何控件、图像，或图形方法使用系统中间色调的调色板绘图。

在大多数情况下，中间色调模式是一种很好的选择，因为它在窗体中的图像与其它窗体或图像中使用的颜色之间，提供了折衷办法。当然，它也会导致图像质量的下降。例如，一个具有包含 256 种深度灰色调色板的图像，在显示时可能会丢失某个细节或显示出其它难以预料的颜色。

UseZOrder PaletteMode

零排序是一个相对的排序，它决定窗体中的控件，相互间如何重叠。对于带有焦点的窗体，当它的 **PaletteMode** 被设置为 **UseZOrder** 时，总是最顶层控件的调色板享有优先权。这就是说，每当一个新控件成为最顶层时（如：调一个新图像到图片框中），硬件调色板都将重新映射。这常常会引起称为调色板闪烁的副作用：当新颜色显示时，会出现闪烁，在当前窗体和其它任何可见的窗体或应用程序中都是如此。

尽管 **UseZOrder** 设置提供了最准确的颜色显示，但它速度较慢。而且，这种方法会引起窗体的背景色，或无图像控件的背景色出现抖动。当准确显示一个顶层图像比产生调色板闪烁更重要，或需要保持 **Visual Basic** 老版本的向下兼容时，将 **PaletteMode** 设置为 **UseZOrder** 是最好的选择。

自定义 **PaletteMode**

如果需要更精确地控制实际显示的颜色，可用 256 色映象定义一个自定义调色板。要实现这一点，需赋值给窗体的 `Palette` 属性一个 256 色映象（`.bmp`、`.cur`、`.ico`、`.dib` 或 `.gif`），并将 `PaletteMode` 属性设置为 `Custom`。位图不用很大，甚至一个单独的像素就可以为窗体或图片框定义 256 种颜色。这是因为位图的逻辑调色板能列出 256 种颜色，而不论那些颜色是否全部出现在位图中。

因为有缺省方法，所以使用 `RGB` 函数定义的颜色也必须存在于位图中。如果此颜色不匹配，它将被映射到逻辑调色板中最接近的匹配色，此逻辑调色板属于由 `Palette` 属性指定的位图。

要想在运行时设置自定义 `PaletteMode`，需在 `Form_Load` 事件中添加下列代码（假定具有所选调色板的图像已经被赋值给名为 `Image1` 的图像控件）：

'将调色板从 `Image1` 赋值给窗体。

```
Form1.Palette = Image1.Picture
```

'使用 `Custom` 模式。

```
Form1.PaletteMode = vbPaletteModeCustom
```

另外，也可使用图片对象来达到相同的效果，而无需另加图像控件：

```
Dim objPic As Picture
```

```
Set objPic = LoadPicture(App.Path & "\Pastel.bmp ")
```

'将图片对象的调色板赋值给窗体。

```
Form1.Palette = objPic
```

'使用 `Custom` 模式。

`Form1.PaletteMode = vbPaletteModeCustom`

当希望在应用程序中，自始至终保持同一个调色板时，自定义 `PaletteMode` 是最佳的选择。

注意 在不能使用任何 256 色图形的情况下，使用“Custom `PaletteMode`”也可增强应用程序的性能。如果将窗体的 `PaletteMode` 设置为自定义的，并将 `Palette` 属性设置为空，则该窗体就能快速装入，因为不会发生调色板匹配的问题。

详细信息 关于图片对象的详细信息，请参阅本章后面的“使用图片对象”。

其它调色板模式

创建用户控件时还有两种可用的附加 `PaletteMode` 设置：`Container` 和 `None`。`Container` 模式将用户控件和所包含其它控件的调色板，在运行时映射为容器（窗体或用户文档）的环境调色板。如果该容器没有提供环境调色板，则调用中间色调模式。因为事先不知道用户控件会在何处被调用，所以该模式可以避免控件与其它调色板处理方法发生冲突。

`None` 模式所做的正如所预料的那样：它清除所有的调色板处理。当创建一个不显示图像或图形的用户控件时，通过清除处理调色板信息的附加表头，将 `PaletteMode` 设为 `None`，可提高性能。

使用相关调色板颜色

当为 256 色显示设计时，有些颜色可能会出现抖动。这可能使文本或其他

的图形元素难于辨认。通过指定一个相关调色板颜色，Visual Basic 能够以更高的颜色深度精确地显示颜色的同时，显示 256 色中某种指定颜色的最近似的非抖动颜色。

要强制 Visual Basic 对一个给定的属性使用最近似的单色而不是抖动的颜色，可以在该颜色属性的高位字节放置一个 2，例如，要强制一个窗体的背景色为纯浅橙色，可以使用下面的代码：

```
Private Function PaletteRGB(RGB As Long) As Long
```

```
    PaletteRGB = &H02000000 Or RGB
```

```
End Function
```

如果在设计时设置下面的属性：

```
Form1.BackColor = &H00C0E0FF&    '抖动的浅橙色
```

并且添加下列代码到 Form_Click 事件中：

```
Private Sub Form_Click()
```

```
    Form1.BackColor = PaletteRGB(Form1.BackColor)
```

```
End Sub
```

在运行时单击该窗体，其背景色将改变为单色而不是抖动的阴影。现在，它所使用的是来自于 halftone 调色板的最近似颜色。如果系统运行的颜色深度高于 256 色，可能就看不到这种效果。

使用图片对象

在某些方面，Picture 对象与 Printer 对象很相似——它不可见，但很有用。可将图片对象想象为一个不可见的图片框，作为图像的一个显示区使用。例如，以下代码装入一个带位图的图片对象，使用该位图可设置图片框控件的 Picture 属性：

```
Private Sub Command1_Click()  
    Dim objPic As Picture  
    Set objPic = LoadPicture("Butterfly.bmp")  
    Set Picture1.Picture = objPic  
End Sub
```

Picture 对象支持位图，GIF 图像，JPEG 图像，元文件和图标。

使用图片对象数组

可用图片对象数组，把一系列图形保存在内存中，而不必用包含多个图形框或图像控件的窗体。当制作动画或其它应用程序时，在要求图像快速变换的地方，这是非常有用的。数组在模块级声明：

```
Dim objPics(1) As Picture
```

将下列代码添加到 Form_Load 事件中：

将位图加载到图片对象数组。

```
Set objPics(0) = LoadPicture("Butterfly1.bmp")
```

```
Set objPics(1) = LoadPicture("Butterfly2.bmp")
```

然后在 Timer 事件中循环显示这些图像：

```
Static intCount As Integer
```

```
If intCount = 0 Then
```

```
    intCount = 1
```

```
Else
```

```
    intCount = 0
```

```
End If
```

```
'用 PaintPicture 方法
```

```
'    在窗体上显示位图。
```

```
PaintPicture objPics(intCount), 0, 0
```

添加循环，同时增加 x 和 y 的坐标值，可以很容易地使位图“蝴蝶”穿越窗体“飞行”。

使用图片对象代替 Windows API

在 Windows API 中可使用位图、图标或元文件做许多事情，不过图片对象已做了其中大部分事情。这就是说，如果有可能的话，使用图片对象来代替 Windows API 比较好。而且，图片对象能使用 jpg 和 gif 文件，而 Windows API 则不行。

Picture.Handle 和 PictureBox.hDC 之间没有直接联系。hDC 属性是操作系

统提供给图片框控件的设备上下文的句柄。Picture 对象的 Handle 属性，实际上是图片对象所含 GDI 对象的句柄。

现在有两种完全不同的方法可在窗口（或 blit）中画图。可在对象的 hDC 上使用 BitBlt 或 StretchBlt，也可在图片对象或属性中使用 PaintPicture 方法。若有图像控件，则只能用 PaintPicture，因为图像控件没有 hDC。

详细信息 关于 Windows API 的详细信息，请参阅《部件工具指南》中的“访问 DLLs 和 Windows API”，只在专业版和企业版中有效。

打印

在基于 Windows 的应用程序操作中，打印是最复杂的任务之一。一个好的结果依赖于过程中各部分共同协调工作。如果应用程序出现问题，打印机驱动程序发生变化或打印机功能受到限制，都会发生劣质的打印结果。尽管使用常用的打印机和打印机驱动程序检查应用程序是个好办法，也不可能检查用户所有可能的组合。

从应用程序中进行打印涉及下列三种部件：

- 应用程序中开始打印过程的代码。
- 设计者的系统和应用程序用户的系统中安装的打印机驱动程序。
- 应用程序的用户可用的打印机功能。

应用程序中的代码决定应用程序打印输出的类型和质量。但用户的打印机驱动程序和打印机也会影响打印质量。这部分讨论 Visual Basic 应用程序

的有效打印问题。应用程序中的代码决定应用程序打印输出的类型和质量。但用户的打印机驱动程序和打印机也会影响打印质量。这部分讨论 Visual Basic 应用程序的有效打印问题。关于从 Visual Basic 开发环境打印的详细信息，请参阅第十三章“调试代码和处理错误”的“在立即窗口中打印信息”。

从应用程序中打印

Visual Basic 为打印正文和图形提供了三种技术。

- 可在窗体中建立所需要的输出，然后用 `PrintForm` 方法打印窗体。
- 通过设置 `Printers` 集合中的缺省打印机，可传送正文和图形到打印机。
- 可先传送正文和图形给 `Printer` 对象，再用 `NewPage` 和 `EndDoc` 方法打印。

这部分将探讨以上三种方法的优缺点。

使用 `PrintForm` 方法

`PrintForm` 方法将指定窗体的图像传送到打印机。要用 `PrintForm` 方法打印应用程序中的信息，需先将该信息显示在窗体中，然后再用 `PrintForm` 打印窗体。语法如下所示：

```
[form.]PrintForm
```

若省略窗体名称，则 Visual Basic 打印当前窗体。`PrintForm` 打印窗体的全

部内容，即使窗体的某部分在屏幕上见不到。若窗体中包含图形，只有当 `AutoRedraw` 属性设置为 `True` 时，才能将图形打印出来。打印结束后，`PrintForm` 调用 `EndDoc` 方法清空打印机。

例如，可通过将正文显示在屏幕上的方法，把正文传送到打印机，然后用下列语句调用 `PrintForm` 方法：

```
Print "Here is some text."
```

```
PrintForm
```

`PrintForm` 方法是应用程序打印的最简便方法。因为它可按用户屏幕的分辨率传送信息到打印机（每英寸打印 96 点），即使打印机有更高的分辨率（对于激光打印机，每英寸打印 300 点），结果也不会更好。结果根据窗体中的对象可以变化。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`PrintForm` 方法”。

使用打印机集合

打印机集合是一个包括操作系统中所有可用打印机的对象。打印机列表与打印安装对话框或 Windows “控制面板”中的那些有效内容相同。打印机集合中的每台打印机有唯一索引定义。索引从 0 开始，每台打印机都可根据索引编号来引用。

不论用何种打印方法，所有 Visual Basic 应用程序打印输出的内容，都直接指向 `Printer` 对象，它最初时代表 Windows “控制面板”中指定的缺省打印机。不过，也可把 `Printers` 集合中的任意一台设置为缺省打印机。

使用下列语法，可从打印机集合中选择打印机：

```
Set Printer = Printers(n)
```

下列语句打印操作系统中所有打印机的设备名称到“立即”窗口中：

```
Private Sub Command1_Click()
```

```
Dim x As Printer
```

```
    For Each x In Printers
```

```
        Debug.Print x.DeviceName
```

```
    Next
```

```
End Sub
```

注意 不能在代码中创建新的 Printer 对象实例，也不能在 Printers 集合中直接添加或删除打印机。要在系统中添加或删除打印机，应使用 Windows “控制面板”。

使用 Printer 对象

Printer 对象是一个与设备无关的图片空间，支持用 Print、PSet、Line、PaintPicture 和 Circle 方法来创建文本和图形。只有在窗体或图片框中，才能将这些方法用于 Printer 对象。Printer 对象也包括本章前面所描述的所有字体属性。当完成在 Printer 对象中放置信息后，用 EndDoc 方法将输出传送到打印机。每次应用程序结束时，它们会自动使用 EndDoc 方法，将打印机对象中任何未确定的信息送到打印机。

对于各种不同的打印机，Printer 对象提供最好的打印质量，因为 Windows

在 `Printer` 对象设备无关的图片空间中，将文本和图形与打印机的分辨率和功能进行了最佳的匹配。也可在打印机对象中使用 `NewPage` 方法来打印多页文档。

使用 `Printer` 对象的主要缺点，是取得最佳结果所需要的代码量较大。在 `Printer` 对象中打印位图也要花费时间，因此会降低应用程序的性能。

使用 `Printer` 对象打印

有几种方法可在 `Printer` 对象中放置文本和图形。为了使用 `Printer` 对象进行打印，应按以下任一方式进行：

- 如果要使用缺省打印机以外的打印机打印，需在 `Printers` 集合中指定打印机给 `Printer` 对象。
- 将文本和图形装入 `Printer` 对象。
- 用 `NewPage` 和 `EndDoc` 方法打印 `Printer` 对象的内容。

`Printer` 对象的属性

`Printer` 对象的属性，初始化时与 Windows “控制面板”中设置的缺省打印机的那些属性匹配。运行时，可设置任何 `Printer` 对象的属性，包括：`PaperSize`，`Height`，`Width`，`Orientation`，`ColorMode`，`Duplex`，`TrackDefault`，`Zoom`，`DriverName`，`DeviceName`，`Port`，`Copies`，`PaperBin` 和 `PrintQuality`。关于这些方法的细节和语法，请参阅《Visual Basic 6.0 语言参考手册》。

如果 `TrackDefault` 属性为 `True`，并且在 Windows “控制面板”中改变了缺省打印机，则 `Printer` 对象属性值将会反映新的缺省打印机的属性。在一页当中，一旦设置了某个属性，就不能在该页中改变某些属性。对这些属性的改变只能影响以后各页。下列语句给出了如何用不同的打印质量来打印各页：

```
For pageno = 1 To 4
    Printer.PrintQuality = -1 * pageno
    Printer.Print "The quality of this page is"; pageno
    Printer.NewPage
Next
```

打印质量值可从 $-4 \sim -1$ ，或是一个与打印机分辨率（每英寸的点数 DPI）对应的正整数。例如，下列代码把打印机分辨率设置为 300 DPI：

```
Printer.PrintQuality = 300
```

详细信息 关于 `Printer` 对象属性的详细信息，请参阅《Visual Basic 6.0 语言参考手册》的有关属性的内容。

注意 `Printer` 属性值的效果依赖于打印机生产厂家提供的驱动程序。有些属性设置值没有作用，或几个不同的属性设置值可能有相同作用。在允许值范围之外的设置，可能产生错误，也可能不会产生错误。关于指定驱动程序的详细信息，请参阅生产厂家的文档资料。

Scale 属性

Printer 对象有以下刻度属性：

- ScaleMode
- ScaleLeft 和 ScaleTop
- ScaleWidth 和 ScaleHeight
- Zoom

ScaleLeft 和 ScaleTop 属性分别定义打印页左上角的 x 坐标和 y 坐标。通过改变 ScaleLeft 和 ScaleTop 的值，可改变打印页的左边距和上边距。例如，通过下列语句，用 ScaleLeft 和 ScaleTop 可指定打印的窗体 (PFrm) 在页中心：

```
Printer.ScaleLeft = -((Printer.Width - PFrm.Width) / 2)
```

```
Printer.ScaleTop = -((Printer.Height - PFrm.Height) _  
    / 2)
```

许多打印机都支持 Zoom 属性。该属性定义按原来的百分之多少输出。缺省的 Zoom 属性值为 100，指定输出将按实际尺寸的百分之百来打印。可利用 Zoom 属性使打印页比实际纸页大一些或小一些。例如，将 Zoom 属性设置为 50，可使打印页看起来只是实际打印纸页长和宽的一半。下列语法将缺省 Printer 对象的 Zoom 属性设置为原来的一半：

```
Printer.Zoom = 50
```

定位文本和图形

就象为窗体和图形框设置属性一样，可为 Printer 对象设置 CurrentX 和 CurrentY 属性。这两个属性决定 Printer 对象当前页中的输出位置。下列

语句设置了当前页左上角的图片坐标：

```
Printer.CurrentX = 0
```

```
Printer.CurrentY = 0
```

也可用 `TextHeight` 和 `TextWidth` 方法在 `Printer` 对象中定位文本。关于使用这些文本方法的详细内容，请参阅本章前面的“在指定位置显示打印输出”一节。

在 `Printer` 对象上打印窗体

也许需要应用程序打印一个或多个窗体（连同窗体上的信息），尤其是当窗体的设计用于发票或时间表之类的打印资料时，更是如此。要做到这一点，最简单的途径是用 `PrintForm` 方法。在 `Printer` 对象中使用 `Print` 和图形方法，可在激光打印机上获得最佳效果。但要注意，使用 `Printer` 对象之前需要做较多准备，因为在打印之前要在 `Printer` 对象中重建窗体。

在 `Printer` 对象中重建窗体也可能需要重建下列内容：

- 窗体的轮廓，包括标题和菜单栏。
- 控件和它们的内容，包括文本和图形。
- 直接应用于窗体的图形输出方法，包括 `Print` 方法。

重建 `Printer` 对象中的元素的扩展工作，取决于应用程序和所需打印窗体内容的多少。

在窗体上重建文本和图形

当使用 Print, Line, Circle, PaintPicture 或 PSet 方法，在窗体中创建文本和图形时，可能需要在 Printer 对象中显示一个输出备份。实现此操作的最简便方法就是编写一个与设备无关的过程来重建文本和图形。

例如，下列过程就使用了 PaintPicture 方法，将窗体或控件的 Picture 属性打印到任意一个输出对象中，如，打印机或另一个窗体：

```
Sub PrintAnywhere (Src As Object, Dest As Object)
    Dest.PaintPicture Src.Picture, Dest.Width / 2, _
        Dest.Height / 2
    If Dest Is Printer Then
        Printer.EndDoc
    End If
End Sub
```

然后调用此过程，并传送原对象和目标对象：

```
PrintAnywhere MyForm, Printer
```

```
PrintAnywhere MyForm, YourForm
```

详细信息 详细信息请参阅《Visual Basic 6.0 语言参考手册》的“Print 方法”，“Line 方法”，“Circle 方法”，“Pset 方法”或“PaintPicture 方法”。

打印窗体上的控件

Printer 对象可接收 Print 方法和图形方法的输出（如：Line 方法或 Pset 方法）。但不能把控件直接放在 Printer 对象中。如果应用程序需要打印控件，

则必须编写过程来调用 `Printer` 对象要用到的每种控件，或使用 `PrintForm` 方法。

打印 `Printer` 对象的内容

一旦将文本和图形放到 `Printer` 对象中，就可用 `EndDoc` 方法打印该内容。`EndDoc` 方法要换页，并将悬置的所有输出都送给假脱机系统。`spooler` 在输出传送到打印机的过程中，截取打印作业，并将它送往硬盘或内存，将打印工作保存在那里，直到打印机准备好为止。例如：

```
Printer.Print "This is the first line of text in _
```

```
    a pair."
```

```
Printer.Print "This is the second line of text in _
```

```
    a pair."
```

```
Printer.EndDoc
```

注意 如果应用程序结尾没有明确调用 `EndDoc`，`Visual Basic` 会自动调用。

创建多页文档

打印较长的文档时，可用 `NewPage` 方法，在代码中指定新的一页。例如：

```
Printer.Print "This is page 1."
```

```
Printer.NewPage
```

```
Printer.Print "This is page 2."
```

```
Printer.EndDoc
```

取消打印作业

可用 `KillDoc` 方法结束当前的打印作业。例如，可用对话框询问是继续打印还是结束文档：

```
Sub PrintOrNot()  
    Printer.Print "This is the first line to _  
        illustrate KillDoc method"  
    Printer.Print "This is the second line to _  
        illustrate KillDoc method"  
    Printer.Print "This is the third line to _  
        illustrate KillDoc method"  
    If vbNo = MsgBox("Print this fine document?", _  
        vbYesNo) Then  
        Printer.KillDoc  
    Else  
        Printer.EndDoc  
    End If  
End Sub
```

如果操作系统的打印管理器正在处理打印工作，则 `KillDoc` 方法将删除送入打印机的所有作业。但如果打印管理器没有控制打印工作，可能有页已送入打印机并且不会受 `KillDoc` 的影响。送入打印机的数据量与打印机驱动程序中的数据量稍有不同。

注意 使用 KillDoc 方法不能结束使用 PrintForm 方法开始打印作业。

捕获打印机错误

打印时可能出现可捕获的运行错误。下表列出了一些错误示例：

错误号	错误信息
396	在页内不可设置属性 当同一页中同一属性设置为不同值时，将发生该错误
482	打印机错误 每当打印机驱动程序返回一个错误代码时，Visual Basic 将报告该错误
483	打印机驱动程序不支持该属性 当试图使用一个当前打印机驱动程序不支持的属性时，将发生该错误
484	打印机驱动程序无效 当 WIN.INI 中的打印机信息丢失或不完整时，将发生该错误

注意 打印机错误一般不会立即发生。如果一条语句引起一个打印机错误时，那么该错误可能直到下一条对打印机操作寻址的语句执行时，才会引发。

详细信息 关于运行时错误的详细讨论，请参阅第十三章“调试代码和处理错误”。

第十三章 调试代码和处理错误

无论多么仔细精巧地制作代码，都可能（而且很可能）会出现错误。就理想的情况而言，Visual Basic 过程根本不需要错误处理代码。遗憾的是，有时文件会被误删除、磁盘驱动器空间会溢出、网络驱动器会意外的分离。这些可能发生的事情都能在代码中引起运行时错误。为了处理这些错误，需要将错误处理代码添加到过程中。

有时，错误也可能出现在代码内部，通常称这类错误为缺陷。例如，象光标不听指挥之类的小缺陷就足以令人感到沮丧并深感不便。更严重的缺陷可能还会使应用程序不再对命令作出响应，这时，可能要重新启动应用程序，从而使已经完成但尚未存储的工作丢失。

在应用程序中查找并修改错误的进程称之为调试。为了分析应用程序的操作方式，Visual Basic 提供了几种工具。这些调试工具不但对查出错误根源特别有用，而且还能用来尝试着改变应用程序，或用来了解其它应用程序的工作方式。

本章说明如何使用 Visual Basic 中的调试工具，还要解释如何处理在代码运行时产生的运行时错误，这类错误是由于试图完成无效操作而造成的。

目 录

- 如何处理错误
- 设计错误处理程序
- 错误处理的分层结构
- 通过生成错误测试错误处理
- 联机错误处理
- 集中化的错误处理
- 关闭错误处理
- 含 ActiveX 部件的错误处理
- 调试方法
- 避免错误
- 设计时、运行时以及中断模式
- 使用调试窗口
- 使用中断方式
- 运行应用程序的选定部分
- 监视调用堆栈
- 用立即窗口测试数据和过程
- 特别的调试考虑
- 调试的提示

示例应用程序：Errors.vbp

在本章，许多代码示例取自 Errors.vbp 示例应用程序。你将会在 Samples 目录中找到这些示例应用程序。

如何处理错误

就理想的情况而言，Visual Basic 过程根本不需要错误处理代码。但实际上情况却是这样的，硬件出现的问题或用户出乎意料的操作都会造成运行时错误，这些错误会使代码终止，而且通常无法恢复应用程序的运行。其它错误也许不会中断代码，但是这些错误可能使代码产生意想不到的操作。例如，如果指定文件存在，则以下过程返回真，否则返回假，但该过程不包含错误处理代码：

```
Function FileExists(filename) As Boolean
```

```
    FileExists = (Dir(filename) <> "")
```

```
End Function
```

Dir 函数返回与指定的文件名（带有通配符或无通配符、有驱动器名或路径）相匹配的第一个文件，如果相匹配的文件不存在，则 Dir 函数返回零长度字符串。

代码似乎适用于两种 Dir 调用的可能结果的任何一种。但是，如果参数中

指定的驱动器号是无效驱动器，则产生“装置不可用”的错误。如果指定的驱动器是软盘驱动器，则只有在驱动器内有磁盘且驱动器门关闭时，Dir 函数才能正确工作。否则，Visual Basic 显示“磁盘未准备”的错误并终止代码的执行。

为避免这种情况，在 Visual Basic 中可用错误处理功能中断错误并执行正确操作。（截击错误亦称作捕获错误。）当错误发生时，Visual Basic 将设置错误对象 Err 的各种属性，如错误号、描述等。这样，在错误处理例程中就能使用 Err 对象及其属性，以便应用程序可智能化地对错误状况作出响应。

例如，可用以下代码处理无效驱动器或空软盘驱动器等诸如此类的装置问题：

```
Function FileExists(filename) As Boolean
```

```
    Dim Msg As String
```

```
    '如果检测到任何一个错误，则开始捕获错误，从而响应错误处理程序。
```

```
    On Error GoTo CheckError
```

```
        FileExists = (Dir(filename) <> "")
```

```
        '如果未出现错误，则避免使用错误处理程序。
```

```
        Exit Function
```

```
CheckError:                                '如果出现错误，则在此分枝。
```

```
    '定义常数以表示固定的 Visual Basic 错误代码。
```

```
    Const mnErrDiskNotReady = 71, _
```



```
mnErrDeviceUnavailable = 68
```

'vbExclamation、vbOK、vbCancel、vbCritical 和 vbOKCancel 是
'定义在 VBA 类型库中的常数。

```
If (Err.Number = MnErrDiskNotReady) Then
```

```
    Msg = "Put a floppy disk in the drive "
```

```
    Msg = Msg & "and close the door."
```

'显示具有感叹号图标以及“确定”和“取消”按钮的消息框。

```
If MsgBox(Msg, vbExclamation & vbOKCancel) = _
```

```
vbOK Then
```

```
    Resume
```

```
Else
```

```
    Resume Next
```

```
End If
```

```
ElseIf Err.Number = MnErrDeviceUnavailable Then
```

```
    Msg = "This drive or path does not exist: "
```

```
    Msg = Msg & filename
```

```
    MsgBox Msg, vbExclamation
```

```
    Resume Next
```

```
Else
```

```
    Msg = "Unexpected error #" & Str(Err.Number)
```

```
    Msg = Msg & " occurred: " & Err.Description
```

' 用“Stop”信号灯图标和“确定”按钮显示消息框。

```
        MsgBox Msg, vbCritical
        Stop
    End If
    Resume
End Function
```

在代码中，Err 对象的 Number 属性包含错误号，它与运行时错误相关联，Err 对象的 Description 属性包含错误的简短描述。

当 Visual Basic 生成“磁盘未准备”的错误时，代码显示一段信息，通知用户选择两个按钮之一——“确定”或“取消”。如果选择“确定”，则 Resume 语句将控件返回到出错的语句处，并重新执行该语句。如果错误已得到改正，则操作获得成功；否则程序返回到错误处理程序。

如果选择“取消”，则 Resume Next 语句将控件返回到出错语句（此时为 Exit Function 语句）之后的那条语句。

即使出现“装置不可用”的错误，代码也显示对问题进行描述的信息。于是，由于使用 Resume Next 语句，函数将在出错语句之后的那条语句处继续执行。

如果出现意想不到的错误，则将显示一段简短描述错误的说明并在 Stop 语句处终止代码。

已建立的应用程序可修正错误，或提示用户修改造成错误的条件。为此可使用象上例中展示的那些技术。下一节将详细讨论这些技术。

详细信息 关于如何使用 Stop 语句，请参阅本章后面“错误处理的分层结构”中的“复杂错误处理指南”所做的解释。

设计错误处理程序

错误处理程序是应用程序中捕获和响应错误的例程。对于预感可能会出错的任何过程（应该假定任何 **Basic** 语句都可能导致错误，除非确知情况并非如此），均要对这些过程添加错误处理程序。设计错误处理程序的进程包括三步：

1. 当错误发生时，通知应用程序在分支点（执行错误处理例程的地方）设置或激活错误捕获。

On Error 语句激活捕获并指引应用程序到标记着错误处理例程开始的标号处。

在 **Errors.vpb** 示例应用程序中，**FileExists** 函数包含错误处理例程，该例程名为 **CheckError**。

2. 编写错误处理例程，这对所有能预见的错误都作出响应。如果在某些点，控件实际上分支进入捕获，则说捕获是活动的。

CheckError 例程使用 **If...Then...Else** 语句处理错误，该语句响应 **Err** 对象 **Number** 属性中的值，该值为数值代码，对应于一个 **Visual Basic** 错误。在示例中，如果产生“磁盘未准备”错误，则信息提示用户关闭驱动器门。如果发生“设备不可用”错误，则会显示不同信息。如果发生任何其它错误，则显示恰如其分的描述并终止程序。

3. 退出错误处理例程。

在出现“磁盘未准备”错误的情况下，**Resume** 语句使代码后退到语句

出错的地方分枝。然后，Visual Basic 再次执行那条语句。如果状态还未变化，则发生另一条错误，而且，执行后退到错误处理例程处分支。

在出现“装置不可用”错误的情况下，Resume Next 语句使代码在出错语句之后的那条语句处分支。

这一主题余下的部分提供有关如何完成这些步骤的详细资料。阅读这些步骤时请参阅 FileExists 函数示例。

设置错误捕获

当 Visual Basic 执行 On Error 语句时激活错误捕获，On Error 语句指定错误处理程序。当包含错误捕获的过程是活动的时候，错误捕获始终是激活的，也就是说，直到该过程执行 Exit Sub、Exit 函数、Exit 属性、End Sub、End 函数或 End 属性语句时，错误捕获才停止。尽管在任一时刻任一过程中只能激活一个错误捕获，但可建立几个供选择的错误捕获并在不同的时刻激活不同的错误捕获。借助于 On Error 语句的特例——On Error GoTo 0 也能停用某一错误捕获。

为设置一个跳转到错误处理例程的错误捕获，可用 On Error GoTo line 语句，此处，line 指出识别错误处理代码的标签。在 FileExists 函数示例中，标签是 CheckError。（虽然冒号是标签的一部分，但在 On Error GoTo line 语句中未使用它。）

详细信息 有关停用错误处理的详细信息，请参阅本章后面的“关闭错误处理”中的内容。

编写错误处理例程

书写错误处理例程的第一步是添加行标签，标志着错误处理例程开始。行标签应该有一个具有描述性的名称，其后必须加冒号。有这样一个公共约定，即把错误处理代码放置在过程末端，该过程在紧靠行标签前方处具有 `Exit Sub`、`Exit` 函数或 `Exit` 属性语句。这样，如果未出现错误，则过程可避免执行错误处理代码。

错误处理例程体包含实际处理错误的代码，通常以 `Case` 或 `If...Then...Else` 语句的形式出现。需要确定可能会发生什么错误并对每种错误提供操作方法，例如，在“磁盘未准备”错误的情况下，要提示用户插盘。为了处理所有意想不到的错误，总应使用 `Else` 或 `Case Else` 语句提供选项，例如，在 `FileExists` 函数示例中，该选项先警告用户，然后结束应用程序。

`Err` 对象的 `Number` 属性包含数值代码，代码代表最新的运行时错误。借助 `Err` 对象与 `Select Case` 或 `If...Then...Else` 语句的组合，可对出现的任何错误采取针对性的操作。

注意 包含在 `Err` 对象 `Description` 属性中的字符串可以解释错误，该错误与当前的错误号相关联。在不同的 Microsoft Visual Basic 版本中，正确描述的措词也许会有变化。因此，请用 `Err.Number` 而不要用 `Err.Description` 识别已发生的具体错误。

退出错误处理例程

`FileExists` 函数示例使用错误处理程序内的 `Resume` 语句重新执行最初导致错误的那条语句，使用 `Resume Next` 语句把执行返回到出错语句之后的那条语句。还有其它退出错误处理例程的方法。根据具体情况，可用下表中的任一语句实现这一操作。

语句	描述
<code>Resume [0]</code>	对于出错的语句，或者对于最近曾执行过程调用的语句（这里，过程包括错误处理例程），程序执行会恢复这些语句。改正了产生错误的条件后，可用它重复操作
<code>Resume Next</code>	在紧接着出错语句之后的那条语句处恢复程序执行。如果错误发生在包含错误处理程序的过程之外，而且所调用的过程不具有激活的错误处理程序，则在调用了出现错误的过程之后的那条语句处，执行恢复
<code>Resume line</code>	在 <code>line</code> 指定的标签处恢复程序执行，此处， <code>line</code> 是行标签（或是非零的行号），必须与错误处理程序在同一过程中
<code>Err.Raise Number:= number</code>	触发运行时错误。在错误处理例程内执行这一语句时， <code>Visual Basic</code> 搜索另一个错误处理例程的调用列表。（调用列表是为到达当前执行点而调用的过程链。请参阅本章后面的“ 错误处理的分层结构 ”一节的内容。）

`Resume` 和 `Resume Next` 语句间的差别

Resume 和 Resume Next 之间的差别如图 13.1 所示。

一般来说，错误处理程序无论何时修正错误，都可使用 Resume，而在错误处理程序不能修正错误时使用 Resume Next。可编写这样一个错误处理程序，使用户看不到运行时存在错误的，又不为用户显示错误信息，也不允许用户进行修改。

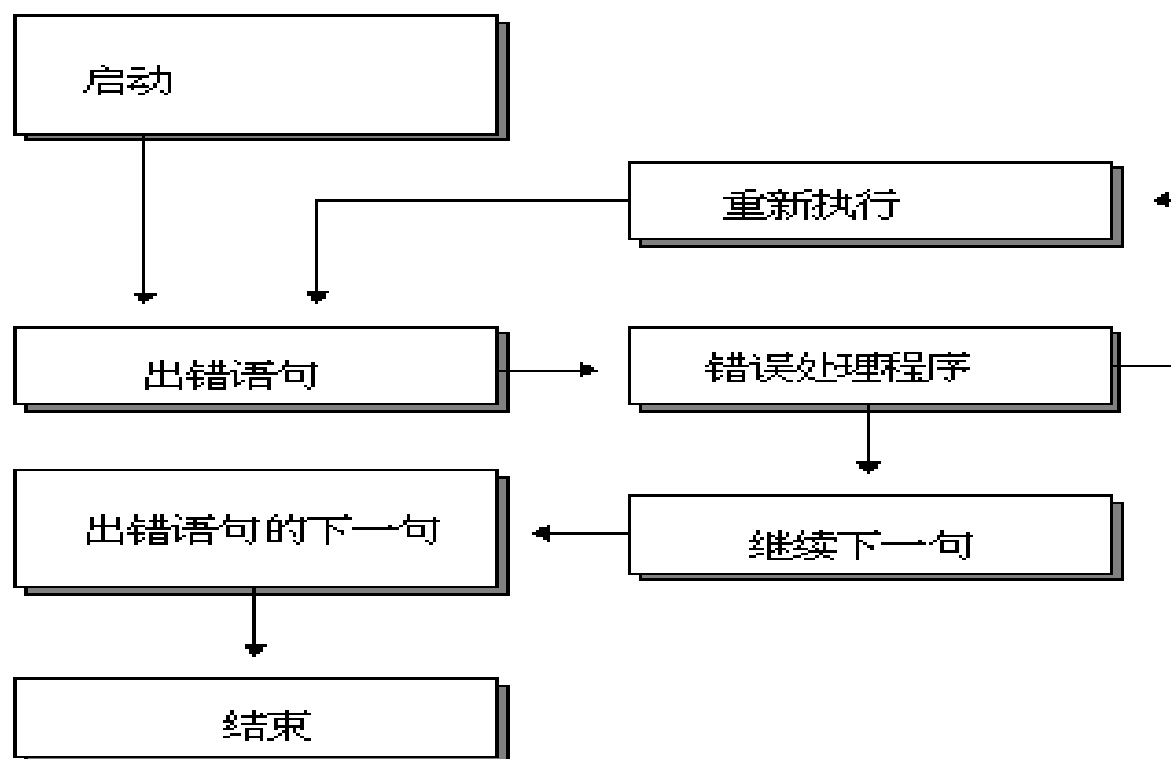


图 13.1 Resume 和 Resume Next 的程序流程

例如，在以下代码示例中，Function 过程用错误处理来对参数施行“安全”

除法，却不揭示可能发生的错误。当施行除法时，可能产生的错误有：

错误	原因
“被零除”	分子非零，但分母为零
“溢出”	分子和分母都为零（在浮点除法中）
“非法的过程调用”	分子或分母不是数值（或不能认为是数值）

在这三种情况中，以下 Function 过程捕获这些错误并返回 Null：

```
Function Divide (numer, denom) as Variant
```

```
    Dim Msg as String
```

```
    Const mnErrDivByZero = 11, mnErrOverflow = 6
```

```
    Const mnErrBadCall = 5
```

```
    On Error GoTo MathHandler
```

```
        Divide = numer / denom
```

```
    Exit Function
```

```
MathHandler:
```

```
    If Err.Number = mnErrDivByZero Or _
```

```
    Err.Number = ErrOverflow _
```

```
    Or Err = ErrBadCall Then
```

```
        Divide = Null ' 如果错误是被零除、溢出或非法过程调用，
```

```
        ' 则返回 NULL。
```

```
    Else
```

```
        '显示意想不到的错误信息。
```

```
        Msg = "Unanticipated error " & Err.Number
```



```

        Msg = Msg & ": " & Err.Description
        MsgBox Msg, vbExclamation
    End If          ' 不管什么情况， Resume Next
    Resume Next     ' 总在 Exit 函数语句处继续执行。
End Function

```

在指定行恢复执行

`Resume Next` 也可用于发生在循环内的错误，此时，需要重新启动操作。还可使用 `Resume line`，它将控件返回到指定的行标签。

以下示例说明了 `Resume line` 语句的用法。前面所举示例 `FileExists` 的变异，如果文件存在的话，该函数允许用户进入函数返回的文件规格说明。

```

Function VerifyFile As String
    Const mnErrBadFileName = 52, _
    mnErrDriveDoorOpen = 71
    Const mnErrDeviceUnavailable = 68, _
    mnErrInvalidFileName = 64
    Dim strPrompt As String, strMsg As String, _
    strFileSpec As String
    strPrompt = "Enter file specification to check:"
StartHere:
    strFileSpec = "*.*"      ' 用缺省的指定启动。
    strMsg = strMsg & vbCrLf & strPrompt

```

' 允许用户修改缺省。

```
strFileSpec = InputBox(strMsg, "File Search", _  
strFileSpec, 100, 100)
```

' 如果用户删除缺省则退出。

```
If strFileSpec = "" Then Exit Function
```

```
On Error GoTo Handler
```

```
VerifyFile = Dir(strFileSpec)
```

```
Exit Function
```

Handler:

```
Select Case Err.Number      ' 分析错误代码并加载消息。
```

```
Case ErrInvalidFileName, ErrBadFileName
```

```
    strMsg = "Your file specification was "
```

```
    strMsg = strMsg & "invalid; try another."
```

```
Case MnErrDriveDoorOpen
```

```
    strMsg = "Close the disk drive door and "
```

```
    strMsg = strMsg & "try again."
```

```
Case MnErrDeviceUnavailable
```

```
    strMsg = "The drive you specified was not "
```

```
    strMsg = strMsg & "found. Try again."
```

```
Case Else
```

```
    Dim intErrNum As Integer
```

```
    intErrNum = Err.Number
```

```
Err.Clear           ' 清除 Err 对象。
Err.Raise Number:= intErrNum    ' 重新产生该错误。
End Select
Resume StartHere    ' 跳回 StartHere 标号，用户可尝试另一文件名。
End Function
```

如果找到了与说明书匹配的文件，则函数返回文件名。如果未找到匹配文件，则函数返回零长度的字符串。如果发生一个预期的错误，则信息被赋给 `strMsg` 变量，而且将跳回到 `StartHere` 标签。这为用户提供了另外一个进入有效路径和文件规格说明的机会。

如果是意想不到的错误，则 `Case Else` 段重新生成错误，以便调用列表中的下一个错误处理程序能捕获该错误。这是非常必要的，因为，如果不重新生成错误，则代码将继续在 `Resume StartHere` 行处执行。重新生成错误，实际上就是再次产生错误；这样，在调用堆栈的下一层将捕获新的错误。

详细信息 请参阅本章后面“错误处理的分层结构”中的内容。

注意 虽然使用 `Resume line` 是写代码的合法方法，但是，增加向行标签的跳转可能使代码难以理解和调试。

错误处理的分层结构

激活的错误处理程序通过执行 `On Error` 语句而被激活，并且不会被 `On Error GoTo 0` 语句关闭，也不会通过退出激活它的过程而关闭。活动的错

误处理程序是当前正在执行的错误处理程序。为了使错误处理程序为活动的，首先应激活它，但并非所有已激活的错误处理程序都是活动的。例如，在 **Resume** 语句之后，处理程序失活但仍然是激活的。

当错误发生在没有激活的错误处理例程过程内部，或发生在活动的错误处理例程内部时，**Visual Basic** 对另一个激活的错误处理例程搜索调用列表。调用列表是调用的序列，该序列指向当前执行的过程；它被显示在 **Call Stack** 对话框中。只有在中断方式下（在暂停执行应用程序时），选定“视图”、“字体”菜单项或按 **CTRL+L** 键，才能显示“字体”对话框。

搜索调用列表

假设以下调用序列发生，如图 13.2 所示：

1. 一事件过程调用 **Procedure A**。
2. **Procedure A** 调用 **Procedure B**。
3. **Procedure B** 调用 **Procedure C**。

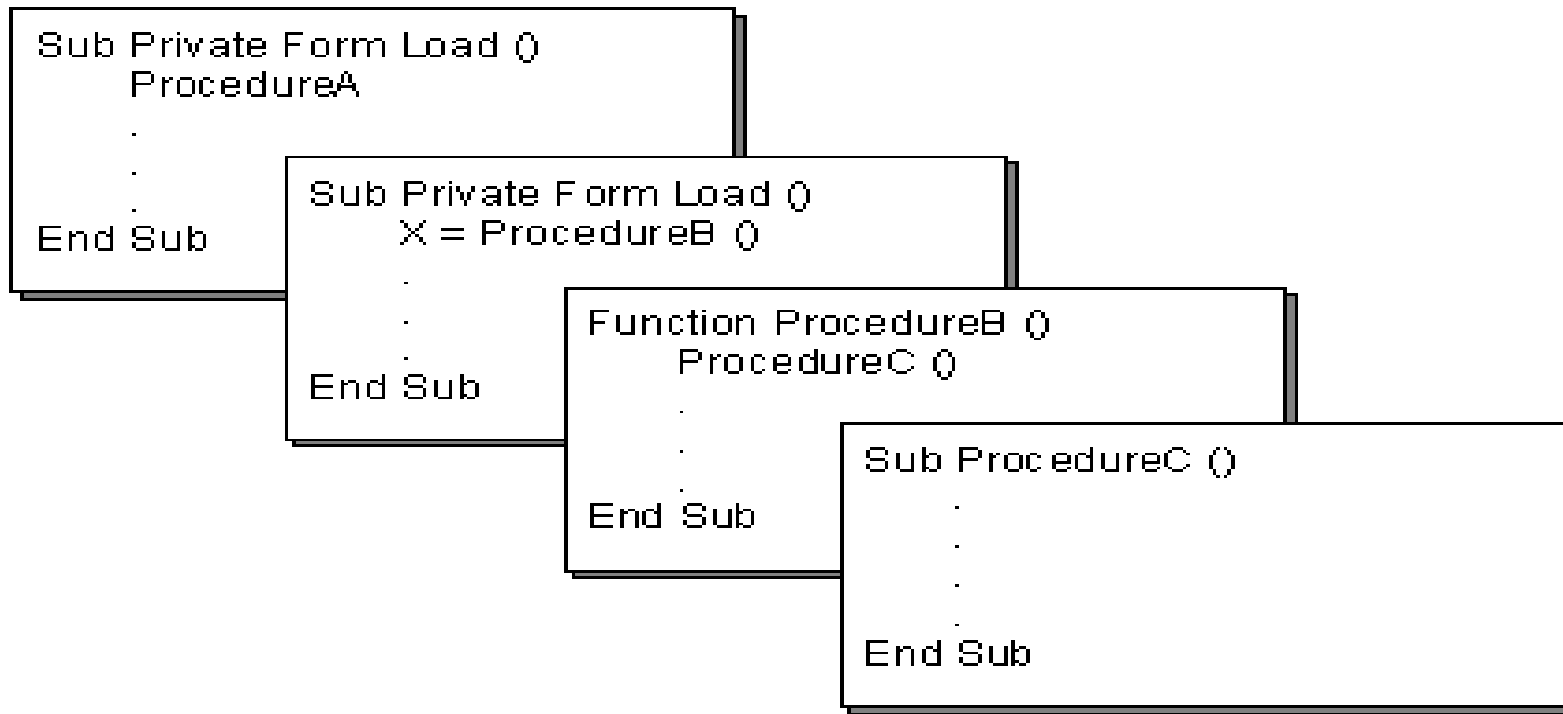


图 13.2 调用序列

当执行 Procedure C 时，其它过程挂起，如“字体”对话框中调用列表所示。

详细信息 请参阅本章后面“监视调用堆栈”中的内容。

图 13.3 显示了“字体”对话框中的调用列表。

如果 Procedure C 中出现错误，而且该过程没有激活的错误处理程序，则 Visual Basic 在调用列表中沿挂起的过程向后搜索，首先搜索 Procedure B、Procedure A，然后是初始事件过程（但非父过程），并执行查找到的第一个

激活的错误处理程序。如果在调用列表中的所有地方都没有遇到激活的错误处理程序，则显示默认的意外错误信息并终止执行。

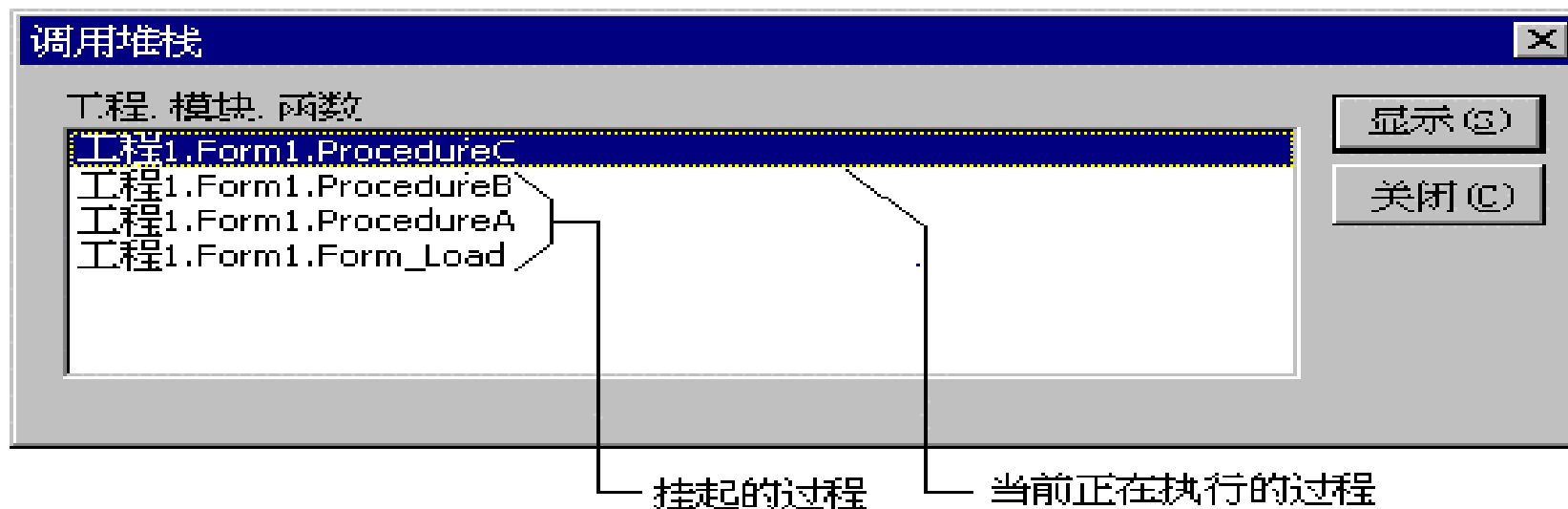


图 13.3 当过程挂起时的调用列表

如果 Visual Basic 查找到一个激活的错误处理例程，那么，就好象在包含错误处理程序的同一过程中出现了错误，执行将在该例程中继续下去。如果在错误处理例程中执行 Resume 或 Resume Next 语句，则如下表所示，执行也继续下去。

语 句	结 果
Resume	再次执行这样一个过程的调用，该过程是 Visual Basic 搜索的过程。在前述的调用列表中，如果 Procedure A 有激活的包含 Resume 语句的错误处理程序，则 Visual Basic 对 Procedure B 再执行调用
Resume Next	执行将返回到过程中已执行过的上一条语句之后的那条语句。这就是紧随着调用过程的语句，而过程又恰好是 Visual Basic 后退搜索过的。在前述调用列表中，如果 Procedure A 有激活的包含 Resume Next 语句的错误处理程序，则执行将返回到调用 Procedure B 之后的语句

注意，已执行的语句在找到错误处理过程的过程中，而不必在错误发生的过程中。如果不考虑这一点，则代码可能不尽如人意。为使代码更易于调试，无论何时出现错误都可直接进入中断方式，如同本章后面的“关闭错误处理”一节解释的那样。

如果错误处理程序的错误范围不包括实际产生的错误，则在具有激活的错误处理程序的过程内会产生意想不到的错误。此时，过程可能会无止境地执行下去，特别是如果错误处理程序还执行 Resume 语句的话，情况更是如此。为防止这种情况出现，在程序内的 Case Else 语句中使用 Err 对象的 Raise 方法。这实际上是在错误处理程序内生成一条错误，从而迫使 Visual Basic 通过调用列表搜索能够处理错误的处理程序。

在 Errors.vbp 示例应用程序的 VerifyFile 过程示例中，将原来包含在

`Err.Number` 中的数赋予变量 `intErrNum`，在 `Case Else` 语句中，该变量将随后作为参数传递到 `Err` 对象的 `Raise` 方法中，由此而生成错误。当这样的错误在活动的错误处理程序内产生时，应着手通过调用列表后退搜索。

将错误分配到不同的处理程序

通过调用列表后退搜索的效果很难预测，因为这要取决于在成功地处理错误的处理程序中是否执行了 `Resume` 或 `Resume Next`。`Resume` 将控件返回到刚执行过的过程调用，而该过程包含错误处理程序。`Resume Next` 则将控件返回到最近一次执行过的过程调用之后的任何语句之处，而且调用的过程包含错误处理程序。

例如，在图 13.3 所示的调用列表中，如果 `Procedure A` 含有激活的错误处理程序，而 `Procedure B` 和 `C` 未包含这样的程序，则在 `Procedure C` 中发生的错误将由 `Procedure A` 中的错误处理程序处理。如果一退出，该错误处理程序就使用 `Resume` 语句，则程序继续调用 `Procedure B`。但是，如果一退出，`Procedure A` 的错误处理程序就使用 `Resume Next` 语句，则在 `Procedure A` 中，不管 `Procedure B` 的调用之后是什么语句，程序都将继续执行该语句。在这两种情况下，错误处理程序都不直接返回到最初发生错误的过程或语句。

复杂错误处理指南

当用多个模块编写大型 `Visual Basic` 应用程序时，错误处理代码可能相当

复杂。请牢记这些方针：

- 在调试代码时，对于在程序中没有代码处理具体错误的情况，应在所有错误处理程序中使用 `Err` 对象的 `Raise` 方法重新生成错误。由此，应用程序就可按照调用列表，试着将其它错误处理例程中的错误加以改正。如果发生一个代码未处理的错误，则也要确保 `Visual Basic` 显示错误信息。在测试代码时，这一技术有助于揭示难以处理的错误。但是，对于独立方式的 `.exe` 文件，应该切记：如果执行 `Raise` 方法，而且无任何其它过程捕获错误，则应用程序将立即中断执行，且没有任何 `QueryUnload` 或 `Unload` 事件发生。
- 在处理错误后，如果需要明确地清除 `Err` 对象，则可用 `Clear` 方法。当使用具有 `On Error Resume Next` 的联机错误处理时，这种做法是非常必要的。`Visual Basic` 无论何时执行任何类型的 `Resume` 语句、`Exit Sub`、`Exit` 函数、`Exit` 属性或其它 `On Error` 语句，都可自动调用 `Clear` 方法。
- 如果不要另一个过程在调用列表中捕获错误，则可用 `Stop` 语句强制终止代码。在开发环境中精制代码时，可用 `Stop` 检查错误的上下文。
警告 在建立 `.exe` 文件之前要确保删除了所有 `Stop` 语句。如果独立状态的 `Visual Basic` 应用程序（`.exe`）碰到 `Stop` 语句，则该应用程序按 `End` 语句来处理 `Stop` 语句并立即中断执行，而且没有任何 `QueryUnload` 或 `Unload` 事件发生。
- 编写一个故障保险的错误处理过程，所有错误处理程序都可调用它作为最后手段，解决不能处理的错误。故障保险的过程能够通过卸载窗体和保存数据，按顺序中断应用程序。

详细信息 请参阅本章后面的“联机错误处理”、“设计时、运行时以及中断模式”及“通过生成错误测试错误处理”。

通过生成错误测试错误处理

当测试应用程序，或想要处理等价于 Visual Basic 运行时错误的具体条件时，对错误进行模拟是非常有用的。例如，用一个定义在外部应用程序中的对象编写模块并希望应用程序的其余部分把该对象返回的错误作为实际的 Visual Basic 错误来处理。

为测试所有可能发生的错误，可能需要在代码中生成一些错误。在代码中可用 **Raise** 方法生成错误：

object .**Raise** *argumentlist*

通常，*object* 参数是 **Err**，它是 Visual Basic 定义的全局错误对象。*argumentlist* 参数是命名的参数列表，这些参数可通过方法传递。在 **Errors.vbp** 示例应用程序中，**VerifyFile** 过程在错误处理程序中使用以下代码重新生成当前错误：

```
Err.Raise Number:=intErrNum
```

在这种情况下，**intErrNum** 是一变量，包含触发错误处理程序的错误号。当代码到达 **Resume** 语句时，要调用 **Err** 对象的 **Clear** 方法。在调用堆栈上，为了将代码后退传递到上一过程，有必要重新生成错误。

通过为错误提供错误代码也可以模拟任何 Visual Basic 的运行时错误。

`Err.Raise Number:=71` ' 模拟 “磁盘未准备” 错误。

定义自己的错误

有时，除了 Visual Basic 定义的错误，可能还想定义一些错误。例如，当载波信号失落时，依靠调制解调器连接的应用程序可能会产生错误。如果想要生成并捕获自己的错误，则应将错误号添加到 `vbObjectError` 常数中。`vbObjectError` 常数保留的错误号在这样一个范围内，该范围从其自身偏移量到其偏移量 + 512。使用高于这个范围的号数将确保错误号不会与 Visual Basic 将来的版本或其它 Microsoft Basic 产品相冲突。ActiveX 控件也可定义它们自己的错误号。为避免冲突，请参阅为应用程序所用控件提供的文档。

为定义自己的错误数，将常数添加到模块的声明部分。

' 错误常数

```
Const gLostCarrier = 1 + vbObjectError + 512
```

```
Const gNoDialTone = 2 + vbObjectError + 512
```

然后，当处理任何固定错误时可使用 `Raise` 方法。在这种情况下，`Err` 对象的描述属性将返回标准描述——“应用程序定义的错误或对象定义的错误”。为描述自己的错误，需将描述作为参数添加到 `Raise` 方法中。

详细信息 为了更多了解有关生成自己的错误，请参阅“`Raise` 方法”。

联机错误处理

您也许习惯于用语言编程，换言之，当错误发生时，产生例外情况也不会中断代码的执行，而是将错误记录下来以供以后查对。C 编程语言就是以这种方式工作的，有时会发现，在 Visual Basic 代码中遵循这种习惯是很方便的。

直接在每个可能导致错误的行之后检查错误时，要执行联机错误处理。该主题解释联机错误处理的不同方法，其中有：

- 编写函数和语句，当错误发生时，它们返回错误号。
- 在过程中产生 Visual Basic 错误，并在调用过程中处理联机错误处理程序中的错误。
- 编写一个返回 Variant 数据类型的函数，并用 Variant 指出发生错误的调用过程。

返回错误号

有许多返回错误号的方法。如果错误发生，最简单的方法就是创建返回错误号而不是值的函数和语句。下例说明在 FileExists 函数示例中如何使用这种方法，该实例说明具体文件是否存在。

```
Function FileExists (p As String) As Long
    If Dir (p) <> " " Then
        FileExists = conSuccess      ' 返回表示文件存在的常数。
    Else
```

FileExists = conFailure' 返回表示失败的常数。

End If

End Function

Dim ResultValue As Long

ResultValue = FileExists ("C:\Testfile.txt")

If ResultValue = conFailure Then

.
. '处 理 错 误 。

.

Else

.
. '执 行 程 序 。

.

End If

联机错误处理的关键是直接 在每一个语句或函数调用后测试错误。用这种方式，可以设计一个处理程序，准确预测可能产生的错误种类，从而解决它。这种方法不需要产生实际的运行时错误。这对以 API 和其它 DLL 过程工作的过程是有用的，它们不会使 Visual Basic 产生例外情况。相反，这些过程或者在返回值中，或者在过程传递的参数中指出错误条件；检查正在使用的过程的文档，以确定这些过程如何指示错误条件。

在调用过程中处理错误

另一种指出错误条件的方法是在过程本身引起 Visual Basic 的错误，并在调用过程中处理联机错误处理程序中的错误。下例所示为相同的 FileExists 过程，如果这个过程不成功，则产生错误号。在调用这个函数前，On Error Resume Next 语句在错误发生时设置 Err 对象属性的值，但并不执行错误处理例程。

错误处理代码在 On Error Resume Next 语句之后。该代码能检查 Err 对象的属性以观察是否出现错误。如果 Err.Number 不包含零，则错误已发生，而且，错误处理代码可根据 Err 对象属性的值采用适当的操作。

```
Function FileExists (p As String)
```

```
    If Dir (p) <> " " Then
```

```
        Err.Raise conSuccess ' 返回表示文件存在的常数。
```

```
    Else
```

```
        Err.Raise conFailure ' 产生错误号 conFailure。
```

```
    End If
```

```
End Function
```

```
Dim ResultValue As Long
```

```
On Error Resume Next
```

```
ResultValue = FileExists ("C:\Testfile.txt")
```

```
If Err.Number = conFailure Then
```

```

        .
        .      '处 理 错 误 。
        .
Else
    .
    .      '继 续 程 序 。
    .
End If

```

下例用返回值和其中的一个传递参数指出错误的条件是否是由函数的调用造成的。

```

Function Power (X As Long, P As Integer, _
ByRef Result As Integer)As Long
    On Error GoTo ErrorHandler
    Result = x^P
    Exit Function
ErrorHandler:
    Power = conFailure
End Function

```

'调用 Power 函数。

```

Dim lngReturnValue As Long, lngErrorMaybe As Long
lngErrorMaybe = Power (10, 2, lngReturnValue)

```

```
If lngErrorMaybe Then
```

```
    .  
    .      '处 理 错 误 。  
    .
```

```
Else
```

```
    .  
    .      '继 续 程 序 。  
    .
```

```
End If
```

如果所写的函数直接返回结果值或错误代码，则结果值可能在错误代码的范围内，而调用的过程也许无法区分它们。假如即使用返回值又使用其中的一个传递参数，则程序能确定函数调用失败的原因并采用适当的操作。

使用变量数据类型

另一个返回嵌入错误信息的方法是利用 Visual Basic Variant 数据类型和某些相关函数。Variant 有一个标记，指出包含在变量中的数据类型，并且能将其标记成 Visual Basic 错误代码。可编写一个返回 Variant 的函数，并用这个标记指示已出现错误的调用过程。

下例说明如何编写 Power 函数以返回 Variant。

```
Function Power (X As Long, P As Integer) As Variant
```

```
    On Error GoTo ErrorHandler
```



```
Power = x^P  
Exit Function
```

ErrorHandler:

```
Power = CVErr(Err.Number)'将错误代码转换成已标记的变体型。
```

```
End Function
```

'调用 Power 函数。

```
Dim varReturnValue As Variant  
varReturnValue = Power(10, 2)  
If IsError(varReturnValue) Then
```

```
.  
.    '处理错误。  
.
```

```
Else
```

```
.  
.    '继续程序。  
.
```

```
End If
```

集中化的错误处理

在应用程序中添加错误处理代码时，很快就会发现反复处理的是相同的错误。经过仔细筹划，编写几个可供错误处理代码调用来处理公共错误情况的过程，这样就可减少代码的数量。

下列 `FileErrors` 函数过程显示了适合于已发生的错误的信息，在可能的情况下还允许用户选择按钮，为程序指定下一步应该执行的操作。然后 `FileErrors` 函数将代码返回到调用它的过程。代码值指示程序应该执行哪个操作。注意，一定要在某个地方定义由用户定义的诸如 `MnErrDeviceUnavailable` 的常数，（或者全局定义，或者在包含过程的模块级别定义，或者在过程本身中定义）。由于在 `Visual Basic (VB)` 对象库中定义了常数 `vbExclamation`，所以不需对它作声明。

```
Function FileErrors ( ) As Integer
```

```
    Dim intMsgType As Integer, strMsg As String
```

```
    Dim intResponse As Integer
```

```
    '返回值          含义
```

```
    ' 0                继续
```

```
    ' 1                继续下一个
```

```
    ' 2                不可恢复的错误
```

```
    ' 3                无法辨认的错误
```

```
    intMsgType = vbExclamation
```

Select Case Err.Number

```
Case MnErrDeviceUnavailable      '错误 68。
    strMsg = "That device appears unavailable."
    intMsgType = vbExclamation + 4
Case MnErrDiskNotReady           ' 错误 71。
    strMsg = "Insert a disk in the drive "
    strMsg = strMsg & "and close the door."
    intMsgType = vbExclamation + 4
Case MnErrDeviceIO               ' 错误 57。
    strMsg = "Internal disk error."
    intMsgType = vbExclamation + 4
Case MnErrDiskFull               '错误 61。
    strMsg = "Disk is full. Continue?"
    intMsgType = vbExclamation + 3
                                '错误 64 和 52。
Case ErrBadFileName, ErrBadFileNameOrNumber
    strMsg = "That filename is illegal."
    intMsgType = vbExclamation + 4
Case ErrPathDoesNotExist         ' 错误 76。
    strMsg = "That path doesn't exist."
    intMsgType = vbExclamation + 4
Case ErrBadFileMode              ' 错误 54。
```

```

        strM sg = "Can't open your file for that "
        strM sg = strM sg & "type of access."
        intM sgType = vbExclamation + 4
    Case ErrFileAlreadyOpen ' 错误 55。
        strM sg = "This file is already open."
        intM sgType = vbExclamation + 4
    Case ErrInputPastEndOfFile ' 错误 62。
        strM sg = "This file has a nonstandard "
        strM sg = strM sg & "end-of-file marker, "
        strM sg = strM sg & "or an attempt was made "
        strM sg = strM sg & "to read beyond "
        strM sg = strM sg & "the end-of-file marker."
        intM sgType = vbExclamation + 4
    Case Else
        FileErrors = 3
        Exit Function
End Select
intResponse = MsgBox (strM sg, intM sgType, _
"Disk Error")
Select Case intResponse
    Case 1, 4 ' “确定”、“重试”按钮。
        FileErrors = 0

```

```
Case 5          , “忽略”按钮。  
    FileErrors = 1  
Case 2, 3       , “取消”、“结束”按钮。  
    FileErrors = 2  
Case Else  
    FileErrors = 3  
  
End Select
```

```
End Function
```

该过程处理公共文件和磁盘相关错误。如果错误与磁盘输入/输出无关，则返回数值 3。于是，调用该过程或者自己处理错误，用 **Raise** 方法重新生成错误，或者调用另一过程来处理。

注意 在编写较大的应用程序时就会发现，在各种窗体和模块的几个过程中使用着相同的常数。要使这些常数为公有的，并在单独的标准模块中声明它们，这样就可以更好地组织代码并避免重复输入相同的声明。

无论在何处执行读磁盘或写磁盘过程，都可直接通过 **FileErrors** 过程的调用简化错误处理。例如，可能已用过这样的应用程序，它对是否要更换现有磁盘文件发出警告。相反，当试着打开不存在的文件时，许多应用程序都将预先提示该文件不存在，并询问是否要建立该文件。在这两个例子中，当应用程序将文件名传递到操作系统时都可能产生错误。

以下的检查例程使用 **FileErrors** 过程返回的值，由此决定有关磁盘错误事件中采用的操作。

```
Function ConfirmFile (FName As String, _  
Operation As Integer) As Integer
```

'参数：

'Fname：检查并确认文件。

'操作：顺序文件访问方式的代码（输出、输入等）。

'注意：由于信息在 Operation 上被调整成有序模式，

 ' 而 Operation 是 <>，因此，过程以二进制

 ' 和随机访问方式工作。

'返回值：

 ' 1 确认操作不会引起问题。

 ' 0 决定不进行操作。

```
Const conSaveFile = 1, conLoadFile = 2
```

```
Const conReplaceFile = 1, conReadFile = 2
```

```
Const conAddToFile = 3, conRandomFile = 4
```

```
Const conBinaryFile = 5
```

```
Dim intConfirmation As Integer
```

```
Dim intAction As Integer
```

```
Dim intErrNum As Integer, varMsg As Variant
```

```
On Error GoTo ConfirmFileError ' 打开错误捕获。
```

```
FName = Dir(FName)              ' 检查文件是否存在。
```

```
On Error GoTo 0                  ' 关闭错误捕获。
```

' 如果用户正将文本保存到已存在的文件中...

If FName <> "" And Operation = conReplaceFile Then

varMsg = "The file " & FName &

varMsg = varMsg & "already exists on " & vbCRLF

varMsg = varMsg & "disk. Saving the text box "

varMsg = varMsg & & vbCRLF

varMsg = varMsg & "contents to that file will "

varMsg = varMsg & "destroy the file's current "

varMsg = varMsg & "contents, " & vbCRLF _

varMsg = varMsg & "replacing them with the "

varMsg = varMsg & "text from the text box."

varMsg = varMsg & vbCRLF & vbCRLF

varMsg = varMsg & "Choose OK to replace file, "

varMsg = varMsg & "Cancel to stop."

intConfirmation = MsgBox(varMsg, 65, _

"File Message")

' 如果用户想从不存在的文件中加载文本。

ElseIf FName = "" And Operation = conReadFile Then

varMsg = "The file " & FName

varMsg = varMsg & " doesn't exist." & vbCRLF

varMsg = varMsg & "Would you like to create and

varMsg = varMsg & "then edit it?" &

vbCRLF

```

varMsg = varMsg & vbCRLF & "Choose OK to "
varMsg = varMsg & "create file, Cancel to stop."
intConfirmation = MsgBox(varMsg, 65, _
    "File Message")
' 设置 intConfirmation = 2, 如果 Fname 不存在, 强迫过程返回 0。
ElseIf FName = "" Then
    If Operation = conRandomFile Or _
        Operation = conBinaryFile Then
        intConfirmation = 2
    End If
' 若文件存在而操作没有成功, 则 intConfirmation = 0, 过程返回 1。
End If
' 如果没有框显示, 则 intConfirmation = 0;
' 如果用户卸载“确定”, 则无论如何 intConfirmation = 1,
' 并且 ConfirmFile 返回 1, 确认期望的操作。
' 如果 intConfirmation > 1, 则 ConfirmFile 返回 0,
' 因为用户不想完成该操作...
If intConfirmation > 1 Then
    ConfirmFile = 0
Else
    ConfirmFile = 1
    If Confirmation = 1 Then

```



```

        ' 用户想创建文件。
    If Operation = conLoadFile Then
        ' 赋值 conReplaceFile,
        ' 使调用者明了将要发生的动作。
        Operation = conReplaceFile
    End If
    ' 返回代码，确认下一步是替换老文件还是创建新文件。
End If
End If
Exit Function
ConfirmFileError:
intAction = FileErrors
    Select Case intAction
        Case 0
            Resume
        Case 1
            Resume Next
        Case 2
            Exit Function
        Case Else
            intErrNum = Err.Number
            Err.Raise Number:=intErrNum

```

Err.Clear

End Select

End Function

ConfirmFile 过程接收对文件的指定，而该文件已被确认存在，该过程还接受附加信息，这些信息是有关实际打开一文件时使用的访问方式。如果要保存一个顺序文件（**conReplaceFile**），同时找到一个已经使用那个名称的文件（因此该文件要被覆盖），则提示用户确认将文件覆盖是可接受的。如果要打开一个顺序文件（**conReplaceFile**），但未找到该文件，则提示用户确认应建立一个新文件。如果以随机或二进制访问方式打开一文件，则文件的存在与否，要么被确定（返回数值 1），要么被拒绝（返回数值 0）。如果在调用 **Dir** 时产生错误，则调用 **FileErrors** 过程来分析错误并提示用户采用合理的操作进程。

关闭错误处理

如果在过程中激活了错误捕获，则当过程完成执行时，错误捕获会自动无效。但是，当过程中的代码一直在执行时，可能会想要关闭过程中的错误捕获。为了关闭激活的错误捕获，可用 **On Error GoTo 0** 语句。一旦 **Visual Basic** 执行了该语句，则在过程内检测错误而不捕获错误。在过程中到处都可用 **On Error GoTo 0** 关闭错误处理，甚至在错误处理例程自身内也是如此。

如下段所示，试着在整个过程中用 Step Into 单步调试：

```
Sub ErrDemoSub ()
    On Error GoTo SubHandler '激活错误捕获。
        '需要捕获错误并在此修正错误。
        'Kill 函数用于删除文件。
        Kill "Oldfile.xyz"
    On Error GoTo 0 '在此关闭错误处理。
        Kill "Oldfile.xyz"
    On Error GoTo SubHandler '再次激活错误捕获。
        Kill "Oldfile.xyz"
    Exit Sub
SubHandler: '错误处理例程在此进行。
    MsgBox "Caught error."
    Resume Next
End Sub
```

详细信息 为了学习如何使用 Step Into 的特点，请参阅本章后面的“运行应用程序的选定部分”。

用错误处理程序调试代码

在调试代码时可能会发现，当代码产生错误而且错误处理程序捕获了这个错误时，代码将搅乱对自身特点的分析。虽然在项目的每一个模块中都可

在 On Error 行外注释，但这也不胜其烦。

为此另辟蹊径，在调试时可以关闭错误处理程序，使得在每次有错误的时候都进入中断模式。

要在调试时禁止错误处理程序，请按照以下步骤执行：

1. 从“代码”窗口上下文菜单（在“代码”窗口上单击鼠标右键）选取“切换”命令。

2. 选择“发生错误时中断”项。

当错误发生在项目中任何一个地方时，通过选择这个选项就会进入中断模式，而且，“代码”窗口将在出现错误的地方显示代码。

如果没有选择这一选项，错误可能会显示错误信息，也可能不这样，这要取决于错误发生在什么地方。例如，应用程序引用的外部对象就可能产生错误。如果它确实显示了信息，则该信息可能是无意义的，这与错误最初出现的位置有关。

含 ActiveX 部件的错误处理

在使用一个或多个对象的应用程序中，确定错误发生在何处将会更困难，尤其是在错误发生在另一个应用程序的对象中时，情况更是如此。例如，图 13.4 所示为包含窗体模块的应用程序，窗体模块引用类模块，类模块轮流引用 Microsoft Excel Worksheet 的对象。

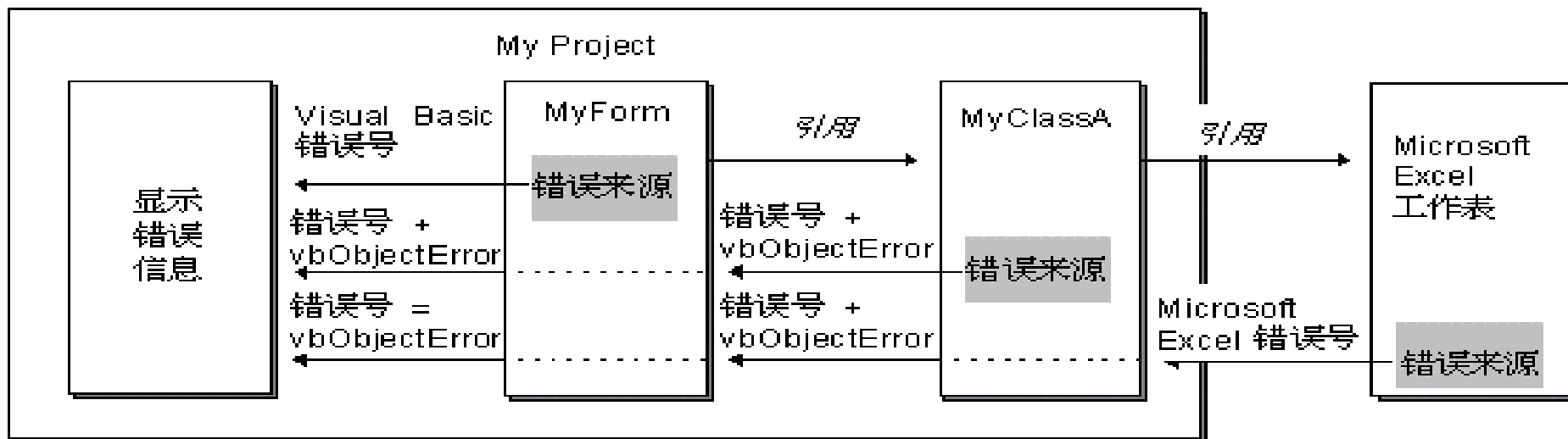


图 13.4 在窗体、类和 ActiveX 组件之间更新生成的错误

如果 Worksheet 对象不处理在 Worksheet 中出现的具体错误，却重新生成这个错误，则 Visual Basic 将错误传递到引用的对象 MyClassA。如果在外部对象中引发了错误，这个错误将在调用这个外部对象的过程中产生。

MyClassA 对象或者能处理错误（这当然最好），或者重新生成它。界面指出的是，对引用的对象所产生的错误，任何重新生成这个错误的对象不应该直接传播错误（传递该错误代码），而应该将错误号重新映射到有意义的东西上。在重新映射错误时，如果处理程序能确定该错误类似于 Visual Basic 定义的错误（例如，溢出或被零除），则该号可能是 Visual Basic 定义的号，它指示错误条件，否则可能是未定义的错误号。将新号添加到固定的 Visual Basic 常数 vbObjectError 中，以便通知其它处理程序，这个错误是由对象引起的。

只要可能，类模块应试着处理每一个在模块自身内出现的错误，也应试着处理出现在该模块所引用的一个对象中的错误，这些错误不被该对象处理。但是，有一些错误是类模块不能处理的，因为它不能预料到这些错误。也有这样的情况发生，这时类模块更适合于引用对象来处理错误，而不适合于被引用的对象。

当窗体模块中发生错误时，Visual Basic 会提出预定义的 Visual Basic 错误号中的一个。

注意 如果创建一个公共类，则应确保清楚地将所定义的每个非 Visual Basic 错误处理程序的意义编成文档。（不能在学习版中创建公共类。）引用该公共类的其它程序员需要知道如何处理由对象引起的错误。

当重新生成错误时，将不改变 Err 对象的其它属性。如果没有捕获所产生的错误，则可能显示 Source 和 Description 属性以帮助用户执行改正了的操作。

在对象中处理错误

类模块可能包括以下错误处理程序，以适应它可能捕获的任何错误，重新生成它不能解决的错误：

MyServerHandler:

```
Select Case ErrNum
```

```
Case 7           '处理内存溢出错误。
```

```

        .
        .
        .
Case 440          '处理外部对象错误。
    Err.Raise Number:=vbObjectError + 9999
'来自其它 Visual Basic 对象的错误。
Case Is > vbObjectError and Is < vbObjectError _
+ 65536
    ObjectError = ErrNum
Select Case ObjectError
    '根据对象的错误代码的文档，该对象处理错误。
    Case vbObjectError + 10
        .
        .
        .
    Case Else
        '当对象错误重新生成时，重新映射错误。
        Err.Raise Number:=vbObjectError + 9999
    End Select
Case Else
    '当对象错误重新生成时，重新映射错误。
    Err.Raise Number:=vbObjectError + 9999

```

End Select

Err.Clear

Resume Next

Case 440 语句捕获错误，该错误出现在 Visual Basic 应用程序之外的被引用对象中。在这个例子中，仅使用数值 9999 传播错误，因为用这种类型的集中化处理程序难于确定错误的原因。当错误发生时，通常是致命的自动错误的结果（一个错误是使部件结束执行），或者是由于对象不能正确地处理捕获的错误。不应传播错误 440，除非它是个致命的错误。如果为嵌入程序编写这样一种捕获，如同以前在“联机错误处理” (Inline Error Handling) 主题中讨论过的那样，则也许能够确定错误的原因并将其改正。

语句

Case Is > vbObjectError and Is < vbObjectError + 65536

捕获最初在 Visual Basic 应用程序内的对象中发生的错误，或在包含处理程序的相同对象中的错误。只有对象定义的错误才在 vbObjectError 偏移量的范围内。

为对象提供的错误代码文档应该定义可能的错误代码并确定它们的含义，以便所写的这部分处理程序可智能化地解决预期的错误。实际错误代码可能被归纳而没有 vbObjectError 偏移量，或者可能在添加偏移量后被归档，在这种情况下 Case Else 语句应该减去而不是增加 vbObjectError。另一方面，对象错误可以是显示在对象类型库中的常数，就象显示在 Object Browser 中一样。在那种情况下，在 Case Else 语句中使用错误常数而不使用错误代码。

任何未处理过的错误应该用新号重新生成，如 `Case Else` 语句中显示的那样。在应用程序中可以设计这样一种程序，它预先知道已定义的新号。如果这是一个公共类（在标准版中不可用），则在应用程序的文档中也应包含新的错误处理代码的解释。

最后一条 `Case Else` 语句捕获并重新生成任何其它错误，这些错误在处理程序的其它地方未被捕获。由于这部分捕获将抓住错误，这些错误也具有添加的 `vbObjectError` 常数，所以应该直接将这些错误重新映射到一个生成的“不可解决的错误”代码。应将代码添加到 `vbObjectError` 中，以便向任何处理程序指示这个错误源于被引用的对象。

用 ActiveX 部件调试错误处理程序

当调试一个应用程序时，如果该应用程序引用一个在 `Visual Basic` 中建立的对象或一个在类模块中定义的类，就可能发现，它将使确定哪一个对象产生了错误变得混乱。为使这个调试变得更轻松，可在“选项”对话框（可从“工具”菜单中得到）中的“通用”选项卡中在“添加类模块”选项中选择“中断”。利用已选择的选项，类模块中的错误或在另一个正在 `Visual Basic` 中运行的应用程序或工程中的对象，将使类进入调试程序的中断模式，从而进行错误分析。出现在编译对象中的错误在中断模式下将不显示立即窗口；这样的错误最好由对象的错误处理程序来处理，或者由引用的模块来捕获。

详细信息 关于类模块选项中的关于中断的详细讨论，请参阅“用部件编

程”中的“调试类模块”。

调试方法

本章介绍的调试技术使用 Visual Basic 提供的分析工具。Visual Basic 不能诊断或更正错误，但确实能提供工具来帮助分析运行是如何从过程的一部分流动到另一部分的，分析变量和属性是如何随着语句的执行而改变的。有了调试工具，就能深入到应用程序内部去观察，从而确定到底发生了什么以及为什么会发生。

Visual Basic 的调试支持包括：断点、中断表达式、监视表达式、通过代码一次经过一个语句或一个过程、显示变量和属性的值。Visual Basic 还包括专门的调试功能，比如可在运行过程中进行编辑、设置下一个执行语句以及在应用程序处于中断模式时进行过程测试等。

详细信息 关于 Visual Basic 调试的概述，请参阅本章后面部分“调试的提示”。

错误种类

为了更有效地使用调试手段，把可能遇到的错误分成三类：

- 编译错误
- 运行时错误

- 逻辑错误

编译错误

编译错误是由于不正确构造代码而产生的。如果错误地键入了关键字、遗漏了某些必需的标点符号或在设计时使用了一个 `Next` 语句而没有 `For` 语句与之对应，那么 `Visual Basic` 在编译应用程序时就会检测到这些错误。编译错误还包括语法错误。例如，可能会有象下面这样的语句：

`Left`

在 `Visual Basic` 语言中，`Left` 是一个有效的词，由于在它的前面没有 `object`，所以不符合该词的语法要求 (`object.Left`)。如果已在“选项”对话框的“编辑器”选项卡中选定“自动语法检测”选项，那么，只要在“代码”窗口中输入一个语法错误，`Visual Basic` 就会立即显示错误消息。

要设置“自动语法检测”选项，请执行：

1. 选定“工具”菜单中的“选项”，单击“选项”对话框的“编辑器”选项卡。
2. 选定“自动语法检测”。

详细信息 关于在代码中用来避免错误的其它技术，请参阅本章后面部分“避免错误”。

运行时错误

应用程序正在运行（而且被 Visual Basic 检测）期间，当一个语句力图执行一个不能执行的操作时，就会发生运行时错误。下面是一个除数为 0 的例子。假定有这样一个语句：

```
Speed = Miles / Hours
```

如果变量 `Hours` 的值为零，除法就是无效操作，尽管语句本身的语法是正确的。必须运行应用程序才能检测到这个错误。

详细信息 应用程序中可以包含处理运行时错误的代码，在这些错误发生时俘获并处理它们。关于处理运行时错误的详细信息，请参阅本章前面部分“如何处理错误”。

逻辑错误

当应用程序未按预期方式执行时就会产生逻辑错误。从语法角度来看，应用程序的代码可以是有效的，在运行时也未执行无效操作，但还是产生了不正确的结果。应用程序运行的正确与否，只有通过测试应用程序和分析产生的结果才能检验出来。

调试工具怎样提供帮助

调试工具的功能是帮助处理：

- 逻辑错误和运行时错误
- 观察无错代码的状况

例如，在结束一长串计算后。可能会得到一个不正确的结果。调试过程的任务就是确定导致错误结果的原因，以及错误发生的地方。很可能是忘记了初始化某个变量、用错了操作符或使用了不正确的公式。

调试的技巧并不神秘，每次工作也无成规可循。调试主要是有助于了解在应用程序运行时正在发生的事情。调试工具提供了应用程序当前状态的快照，包括：

- 用户界面 (UI) 的外观
- 变量、表达式和属性的值
- 活动的过程调用

越是透彻了解应用程序的运行，就越能迅速发现错误。

详细信息 关于查看和测试变量、表达式、属性以及活动的过程调用等详细信息，请参阅本章后面的“用‘立即’窗口测试数据和过程”以及“监视调用堆栈”。

调试工具栏

在众多调试工具中，Visual Basic 在可选的“调试”工具栏上提供了几个很有用的按钮。图 13.5 显示了这些工具。要显示“调试”工具栏，可在 Visual Basic 工具栏上单击鼠标右键并选定“调试”选项。

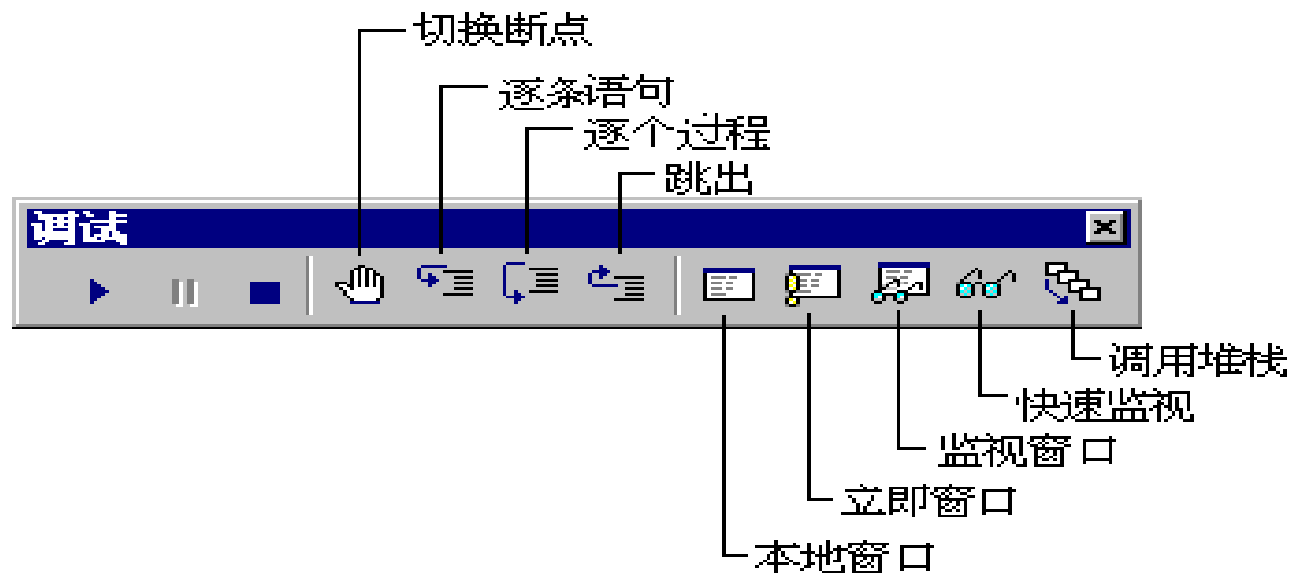


图 13.5 调试工具栏

下表简要叙述每个工具的用途。本章主题是讨论每个工具在什么情况下才能更有效地帮助调试或分析应用程序。

调试工具	目的
断点	在“代码”窗口中确定一行，Visual Basic 在该行终止应用程序的执行
跟踪	执行应用程序代码的下一个可执行行，并跟踪到过程中
单步	执行应用程序代码的下一个可执行行，但不跟踪到过程中

续表

单步出	执行当前过程的其它部分，并在调用过程的下一行处中断执行
“本地”窗口	显示局部变量的当前值
“立即”窗口	当应用程序处于中断模式时，允许执行代码或查询值
“监视”窗口	显示选定表达式的值
快速监视	当应用程序处于中断模式时，列出表达式的当前值
调用堆栈	当处于中断模式时，呈现一个对话框来显示所有已被调用但尚未完成运行的过程

详细信息 仅当应用程序有错误时才需要调试工具。请参阅本章后面的“避免错误”。

避免错误

可用几个方法避免在应用程序中产生错误：

- 写出相关事件以及代码响应每个事件的方法，精心设计应用程序。为每个事件过程和每个普通过程都指定一个特定、明确的目标。
- 多加注释。如果用注释说明每个过程的目的，那么，在回过头来分析代码时，就能更深入理解这些代码。
- 尽可能显式地引用对象。要象对象浏览器中所列举对象那样声明对象，而不用 Variant 或一般的 Object 数据类型。

- 在应用程序中对变量和对象提出一种前后一致的命名方案。更详细的信息，请参阅附录 B “编程约定”。
- 造成错误的一个最普通的原因就是键入了不正确的变量名，或把一个控件与另一个控件搞混了。可用 `Option Explicit` 避免变量名的拼写错误。关于要求显式变量声明的详细信息，请参阅第五章“编程基础”中的“变量、常量和数据类型概述”。

设计时、运行时以及中断模式

为测试和调试应用程序，在任何时刻都要知道应用程序正处在三种模式的哪种模式之下。在设计时用 `Visual Basic` 创建应用程序，而在运行时运行这个程序。本章介绍中断模式，它能够中断程序的执行，从而得以检查和改变数据。

标识当前模式

`Visual Basic` 的标题栏总显示当前模式。图 13.6 所示为设计时、运行时、和中断模式的标题栏。

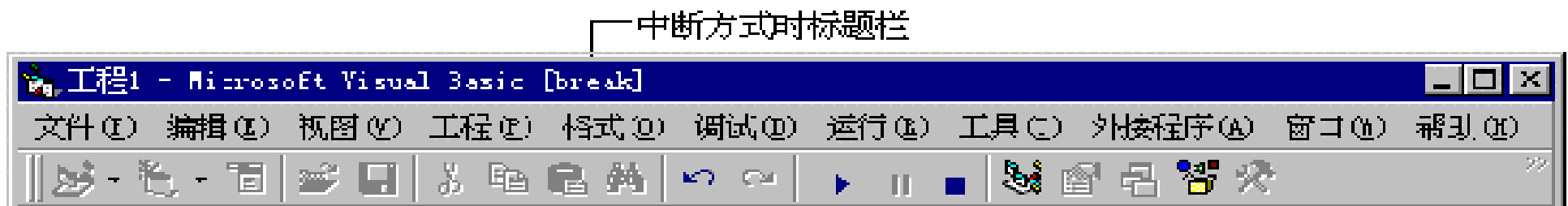
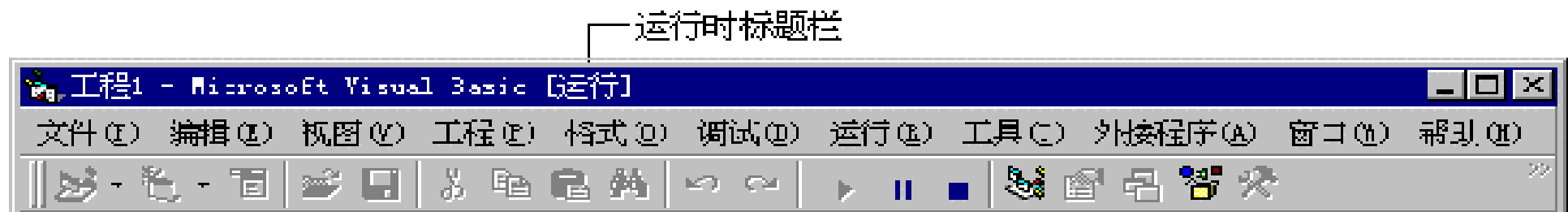


图 13.6 用 Visual Basic 标题栏标识当前模式

下表列出三种模式的特性。

模式	描述
设计时	创建应用程序的大多数工作都是在设计时完成的。在设计时，可以设计窗体、绘制控件、编写代码并使用“属性”窗口来设置或查看属性设置值。除了可以设置断点和创建监视表达式外，不能使用调试工具 在“运行”菜单中选择“启动”，或单击“运行”按钮切换到运行时 如果应用程序在启动时执行了代码，而且本身又包含这些代码，那么就选择“运行”菜单中的“单步”（或按下 F8），使应用程序在第一个可执行语句处被置为中断模式
运行时	当应用程序进行控制时，可象用户那样与应用程序交往。可查看代码，但不能改动它 选择“运行”菜单中的“结束”，或单击“结束”按钮切换回设计时
中断模式	选择“运行”菜单中的“中断”，单击“中断”按钮，或按下 CTRL+BREAK 切换到中断模式 运行应用程序时也可中止执行。可以查看并编辑代码（选择“视图”菜单中的“代码”，或按下 F7）、检查或修改数据、重新启动应用程序、结束执行或从中止处继续运行 可以在设计时设置断点和监视表达式，但其它调试工具只能在中断模式下使用 请参阅本章后面的“使用中断模式”。

用工具栏改变模式

工具栏提供了三个按钮从一种模式快速切换到另一种。在图 13.7 中显示了

这些按钮。

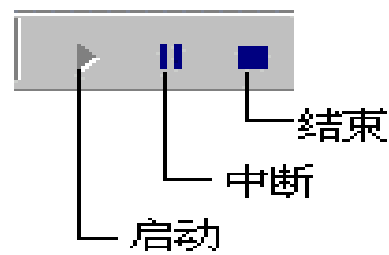


图 13.7 工具栏的“启动”、“中断”、和“结束”按钮

根据 Visual Basic 是处于运行时模式、设计时模式还是中断模式，来确定哪些工具是可用的。下表列出了不同模式下所用的按钮。

模式	可用的工具栏按钮
设计时	“启动”按钮
运行时	“中断”、“结束”按钮
中断	“继续”、“结束”（在中断模式下，“启动”按钮变为“继续”按钮）

使用调试窗口

有时可运行部分代码来查找问题产生的原因。但是，经常要做的往往还是分析数据到底发生了什么变化。可以在有关变量或属性的问题中将不正确的值放到一边，然后就需确定变量或属性是如何得到不正确的值的，为什么会得到这些值。

在逐步运行应用程序的语句时，可用调试窗口监视表达式和变量的值。有三个调试窗口，它们是：“立即”窗口、“监视”窗口、和“本地”窗口。

- “立即”窗口所示为代码中正在调试的语句所产生的信息，或直接往窗口中键入的命令所请求的信息。

详细信息 关于“立即”窗口的详细信息，请参阅本章后面的“用“立即”窗口测试数据和过程”。

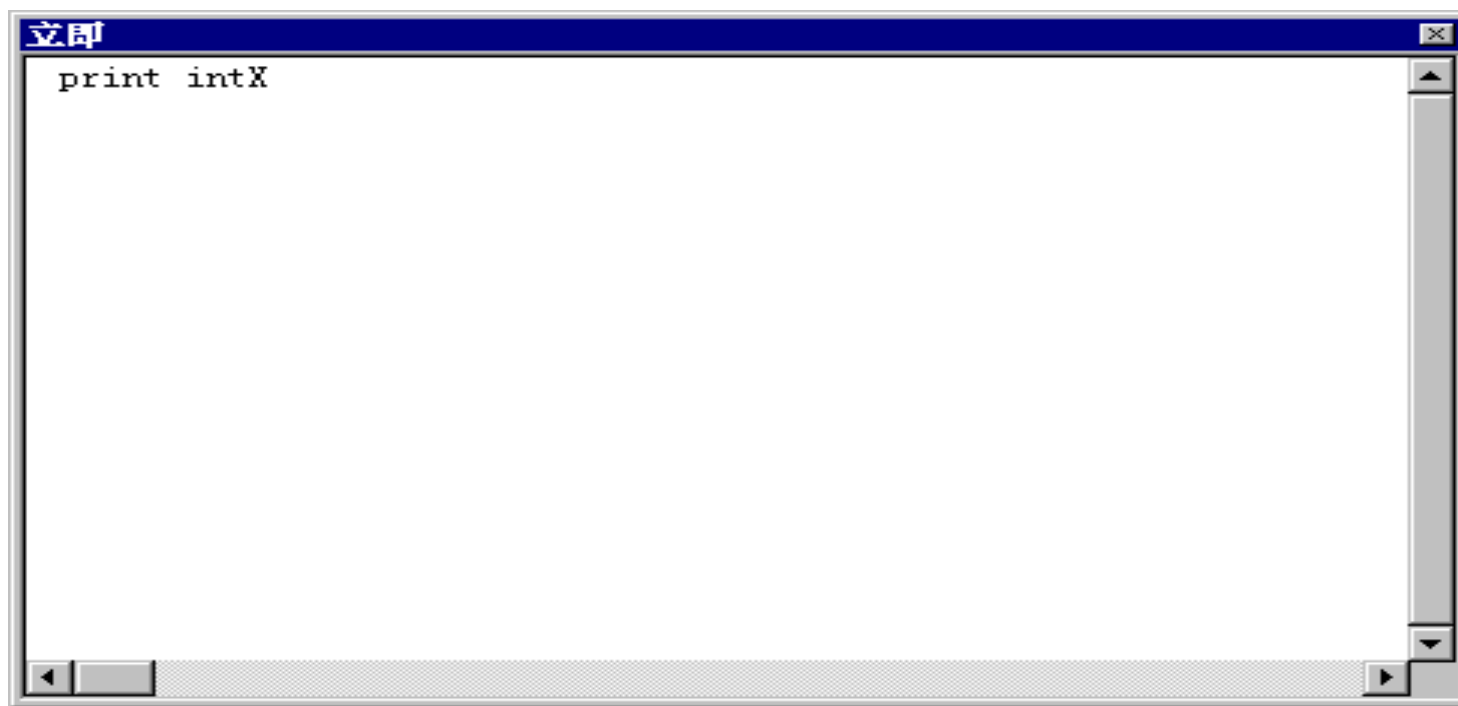


图 13.8 “立即”窗口

- “监视”窗口显示当前的监视表达式，在代码运行过程中可决定是否监控这些表达式的值。中断表达式是一个监视表达式，当定义的某个条件为真时，它将使 Visual Basic 进入中断模式。在“监视”窗口中，“上下文”列指出过程、模块，每个监视表达式都在这些过程或模块中进行计算。只有当前语句在指定的上下文中时，“监视”窗口才能显示监视表达式的值。否则，“Value”列只显示一条消息，指出语句不在上下文中。为访问“监视”窗口，应选定“视图”菜单中的“监视窗口”。图 13.9 显示了“监视”窗口。



图 13.9 “监视”窗口

详细信息 关于“监视”窗口的详细信息，请参阅本章后面的“用监视表达式监视数据”。

- “本地”窗口显示当前过程中所有的变量值。当程序的执行从一个过程切换到另一个过程时，“本地”窗口的内容会发生改变，它只反映当前过程中可用的变量。为了访问“本地”窗口，应选定“视图”菜单中的

“本地窗口”。图 13.10 显示了“本地”窗口。

根据第五章“编程基础”的“变量的范围”所介绍的范围规则，当前过程和窗体（或模块）可决定显示哪些变量。例如，假定“立即”窗口指出 Form1 是当前窗体。在这种情况下就可显示 Form1 中的任何窗体级变量。也可用 Debug.Print 检查“代码”窗口中所显示过程的局部变量。（公共变量的值总是可检查的。）关于在“立即”窗口中打印信息的详细信息，请参阅本章后面的“用‘立即’窗口测试数据和过程”。

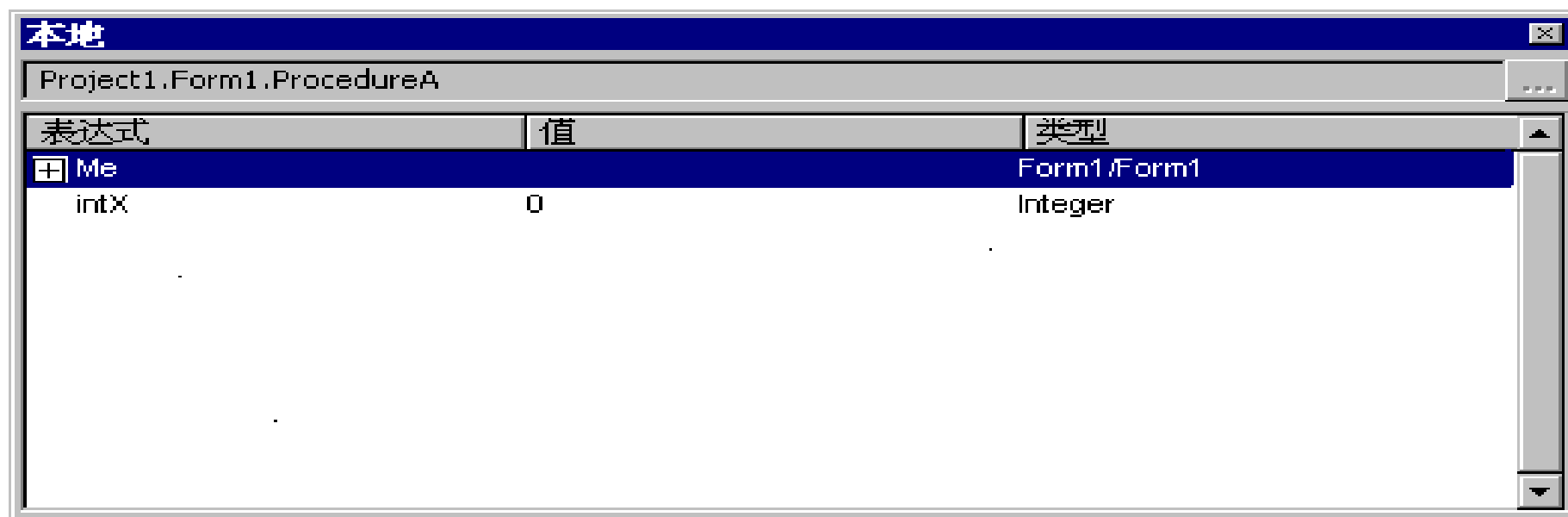


图 13.10 “本地”窗口

使用中断模式

在设计时可改变应用程序的设计或代码，但却看不到这些变更对应用程序的运行产生的影响。在运行时可观察应用程序的执行状况，但不能直接改变代码。

中断模式可随时中止应用程序的执行，并提供有关应用程序的情况快照。因为变量和属性设置值被保留下来，所以，可以分析应用程序的当前状态并输入修改内容，这些修改将影响应用程序的运行。当应用程序处于中断模式时，可以：

- 在应用程序中修改代码
- 观察应用程序界面的情况
- 确定哪个活动过程已被调用
- 监视变量值、属性和语句
- 改变变量值和属性设置值
- 查看或控制应用程序下一步运行的语句
- 立即运行 Visual Basic 语句
- 手工控制应用程序的操作

注意 可在设计时设置断点和监视表达式，但其它调试工具只能在中断模式中运行。

在有问题的语句处进入中断模式

在调试时，可能会希望应用程序能够在认为有问题的代码处中止执行。这就是 Visual Basic 提供断点和 Stop 语句的原因之一。断点定义了一条语句或一系列条件，使 Visual Basic 能够在此自动停止执行，并且不运行包含断点的语句，把应用程序置于中断模式下。关于 Stop 语句和断点的比较，请参阅本章后面的“使用 Stop 语句”。

在应用程序运行时，可用下列方法手动进入中断模式：

- 按下 CTRL+BREAK。
- 选择“运行”菜单中的“中断”。
- 单击工具栏上的“中断”按钮。

当应用程序处于空闲状态（当应用程序处于两事件的进程之间）时，可中断应用程序的执行。此时，应用程序不在特定行中止执行，但无论如何，Visual Basic 还是切换到中断模式。

以下任何情况发生时都会自动地进入中断模式：

- 语句产生了非俘获的运行时错误。
- 语句产生运行时错误，并且“发生错误时中断”错误捕获选项被选中。
- “添加监视”对话框中定义的中断表达式将发生变化或变成真，这取决于它的定义方式。
- 执行到一个设有断点的行。
- 执行到一个 Stop 语句。

更正运行时错误并继续

某些运行时错误是由于输入代码时的疏忽造成的；这些错误很容易更正。例如，把名字拼错，属性或方法与对象不匹配或对文本框使用了 `Clear` 方法，对文件列表框使用 `Text` 属性，这些都是常见错误。图 13.11 显示一个运行时错误消息。

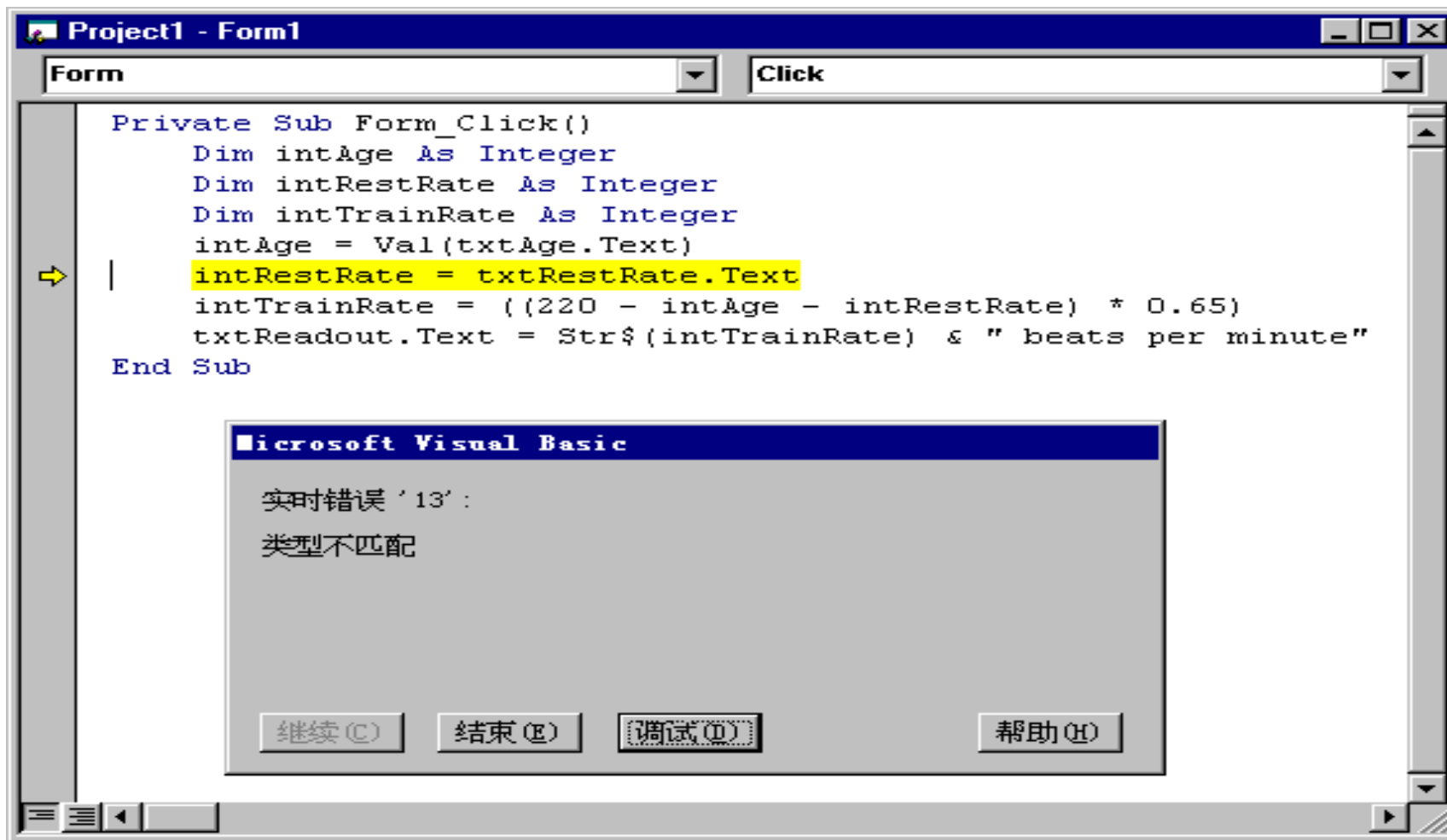


图 13.11 运行时错误中止执行

通常可在中止应用程序的同一行处输入更正并继续执行程序，即使已改变了某些代码，情况亦如此。直接选择“运行”菜单的“继续”，或单击工具栏的“继续”按钮，就可以继续执行程序。在继续运行应用程序时，可以

验证问题是否得到更正。

如果在“选项”对话框（可从“工具”菜单中获得）的“通用”选项卡的“缺省的错误捕获状态”选项组中，选择了“发生错误时中断”，那么 Visual Basic 将禁止代码中的错误处理程序，于是当一个语句产生运行时错误时，Visual Basic 将进入中断模式。如果未选定“发生错误时中断”选项，而且存在一个错误处理程序，那么，错误处理程序将中断代码并采取纠正措施。

注意 如果通过“选项”对话框改变了“缺省的错误捕获状态”选项，那么该设置将变成后续的 VB 会话期的缺省。如果只需改变当前会话期的错误处理，就从“代码窗口上下文”菜单中选择“切换”命令，打开子菜单，选择所需的中断模式。

作过某些变更（最常见的是改变变量声明、添加新的变量或过程）之后将要求重新启动应用程序。当这种情况发生时，Visual Basic 会弹出一条消息，询问是否想要重新启动应用程序。

用监视表达式监视数据

在调试应用程序时，一个计算可能未得到想要的结果，当某个变量或属性取特定的值或一个范围之内的值时，问题也可能会出现。许多调试问题不是由单个语句产生的，所以需要在整个过程中观察变量或表达式的情况。

Visual Basic 自动对监视表达式——定义的表达式——进行监视。当应用程序进入中断模式后，这些监视表达式会出现在“监视”窗口中，可在此处观察它们的值。

无论何时，只要表达式的值改变或等于一个特定的值，就可把应用程序设置为中断模式来直接观察表达式。例如，在循环计数器达到一个特定值时，可用一个监视表达式把应用程序设置为中断模式，而不是一次一条语句逐步执行数十次乃至数百次循环。在过程中一个标志的值每次改变时都可能会希望应用程序进入中断模式。

添加一个监视表达式

在设计时或中断模式下可添加监视表达式。用“添加监视”对话框（如图 13.12 所示）来添加监视表达式。



图 13.12 “添加监视”对话框

下表描述了“添加监视”对话框。

部件	描述
“表达式”框	在输入表达式的地方，监视表达式将计算表达式的值。表达式是一个变量、属性、函数调用或其它任何有效的表达式

续表

“上下文”选项组	用来设置表达式中要监视的变量范围。当有名字相同而范围不同的变量时使用该选项组。还可把监视表达式中的变量的范围限制为由特定过程、特定窗体或模块组成，也可通过选定“所有过程及所有模块”使范围可用于整个应用程序。Visual Basic 在小范围内可更快地计算变量
“监视类型”选项组	用来设置 Visual Basic 对监视表达式作响应的方式。Visual Basic 可在应用程序进入中断模式后对表达式进行监视并显示其值。在表达式的值为真（非零）语句时，或表达式的值每次发生改变时，就可使应用程序自动进入中断模式

要添加监视表达式，请按照以下步骤执行：

1. 选择“调试”菜单的“添加监视”命令。
2. 代码编辑器中的当前表达式（如果有的话）将出现在“添加监视”对话框的“表达式”框中。如果这不是要监视的表达式，则在“表达式”框中输入表达式来求值。
3. 如果有必要，则设置要监视的变量范围。
如果在“上下文”中选定“过程”或“添加模块”选项，则从相应的列表框中选定一个过程、窗体或模块名。
4. 如果有必要，则在“监视类型”组中选定选项按钮来确定 Visual Basic 对监视表达式进行响应的方式。
5. 选择“确定”。

注意 也可把表达式从代码编辑器中拖放到监视窗口来添加表达式。

编辑或删除监视表达式

图 13.13 中显示的“编辑监视”对话框列出所有当前监视表达式。可在“监视”窗口中编辑和删除所有已列出的监视表达式。

要编辑监视表达式，请按照以下步骤执行：

1. 在“监视”窗口中双击想要编辑的监视表达式。

-或-

选定想要编辑的监视表达式，然后选择“调试”菜单中的“编辑监视”。

2. 显示“编辑监视”对话框，除了标题栏不同以及增加了一个“删除”按钮外，这个对话框与“添加监视”对话框相同。
3. 改变表达式、变量的求值范围或监视类型。
4. 选择“确定”。

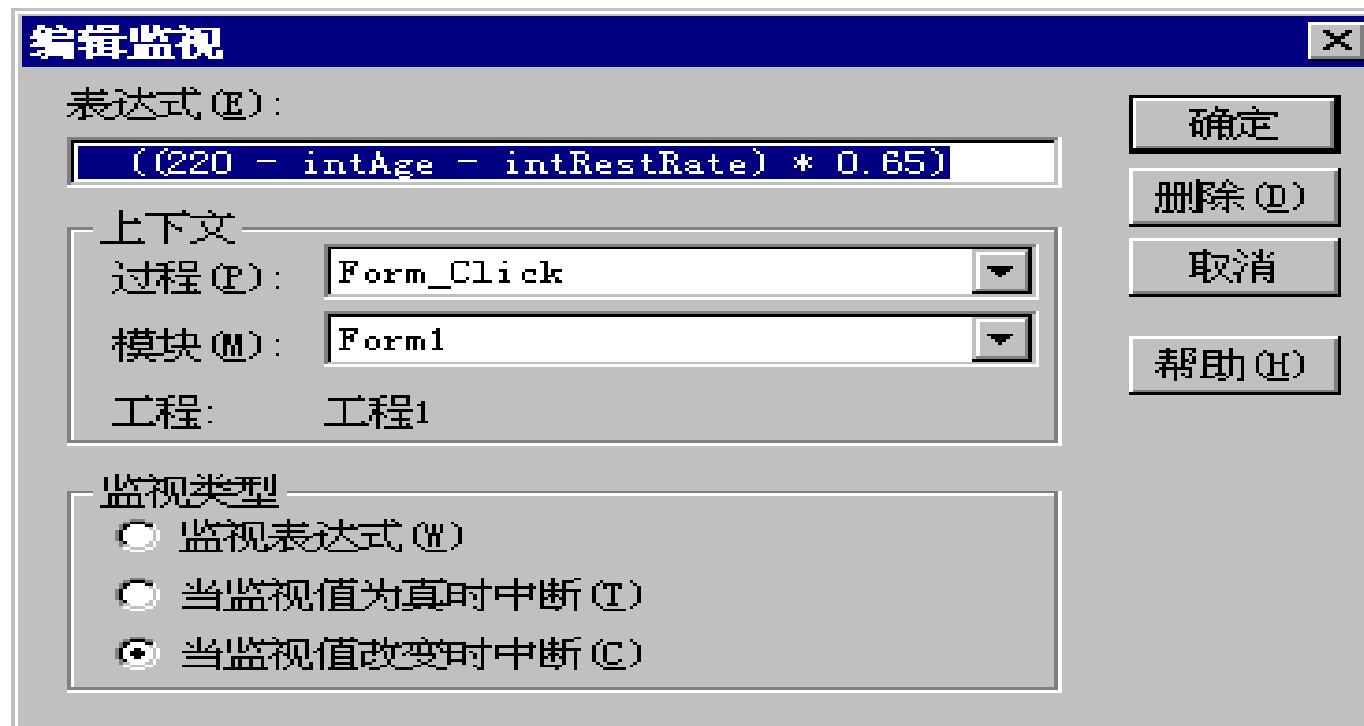


图 13.13 “编辑监视”对话框

要删除监视表达式的步骤，请在“监视”窗口中选定要删除的监视表达式。

- 按下 DELETE 键。

标识监视类型

在“监视”窗口中，每个监视表达式的左边都有一个图标来标识该表达式的监视类型。图 13.14 中对三种监视类型确定了图标。

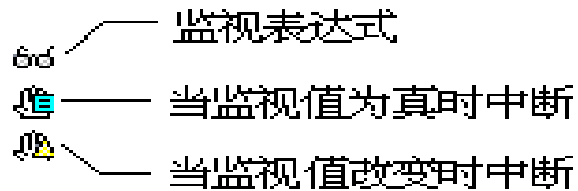


图 13.14 监视类型图标

使用快速监视

在中断模式下可检查那些未定义监视表达式的属性、变量或表达式的值。要检查这样的表达式，应使用“快速监视”对话框，如图 13.15 中所示。

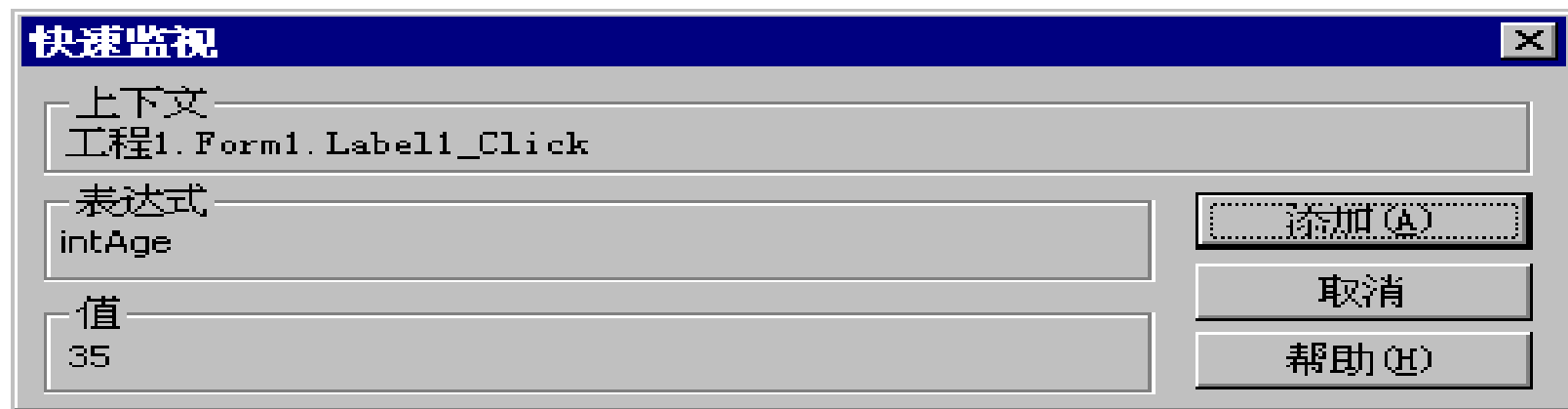


图 13.15 “快速监视”对话框

“快速监视”对话框显示从“代码”窗口中选定的表达式的值。为继续监

视这个表达式，单击“添加”按钮；然后会出现“监视”窗口，并在“监视”窗口中显示“快速监视”对话框中的相关信息。如果 Visual Basic 不能对当前表达式求值，就禁止使用“添加”按钮。

要显示“快速监视”对话框，请按照以下步骤执行：

1. 在“代码”窗口中选定监视表达式。
2. 单击“调试”工具栏的“快速监视”按钮。（为显示“调试”工具栏，在 Visual Basic 工具栏上右击鼠标并选定“调试”选项。）
 - 或 -
 - 按下 SHIFT+F9。
 - 或 -
 - 选择“调试”菜单中的“快速监视”。
3. 如果要把根据“快速监视”对话框的一个表达式定义出来的监视表达式添加到“监视”窗口中，则选择“添加”按钮。

用断点有选择地中断执行

在运行时，一个断点会在执行一行代码之前告诉 Visual Basic 中止运行。当 Visual Basic 正在运行一个过程并遇到一行具有断点的代码时，它就切换到中断模式。

可在中断模式下或设计时设置或删除断点，当应用程序处于空闲时，也可在运行时设置或删除断点。

要设置或删除断点，请按照以下步骤执行：

1. 在“代码”窗口中，把插入点移到要设置或删除断点的代码行。
-或-
在“代码”窗口要设置或删除断点的那一行代码的左边空白区单击鼠标。
2. 选择“调试”菜单的“切换断点”。
-或-
单击“调试”工具栏的“切换断点”按钮。（为显示“调试”工具栏，在 Visual Basic 工具栏上单击鼠标右键并选定“调试”选项。）
- 或 -
按下 F9 键。

设置了断点后，Visual Basic 将以粗体形式突出显示选定的行，所用颜色为“选项”对话框的“编辑器格式”选项卡上所指定的颜色，该对话框可从“工具”菜单中获得。

例如，图 13.16 显示了在第五行处有个断点的过程。在“代码”窗口中，Visual Basic 使用为断点指定的颜色以及粗体形式来显示那一行代码的文本，以指出这是一个断点。

标识当前语句

在图 13.16 中，代码的第七行被一个高亮显示的矩形围绕着。这个轮廓指出它是当前语句或下一个要执行的语句。如果当前行还包含一个断点，则只有矩形轮廓使该代码行突出显示。一旦当前语句移到另一行，具有断点的行就再次以粗体和那种颜色显示。



图 13.16 一个被断点中止的过程

要指定当前语句的文本颜色，请按照以下步骤执行：

1. 选择“工具”菜单中的“选项”，单击“选项”对话框的“编辑器格式”选项卡。
2. 选择“代码颜色”的“执行点文本”，设置“前景”、“背景”和“指示器”颜色。

在断点处检查应用程序

应用程序一旦运行到断点处并被中止执行，就可检查应用程序的当前状态。

检查应用程序的结果是很容易的，因为可以在应用程序的窗体和模块、“代码”窗口以及调试窗口之间移动焦点。

就在执行含有断点的行之前，这个断点中止执行应用程序。如果要观察设有断点的行在运行时发生了什么，就必须至少再运行一个语句。为此要使用跟踪或单步运行。

注意 尽管可以在 `MouseMove` 事件过程或在 `Timer` 事件中设置断点，但这样做可能引起未预期的结果。当输入中断模式时正常事件流被中断；借助来自这些过程内部代码的单步可能表现出不同于运行模式中将会发生的行为。

详细信息 请参阅本章后面的“运行应用程序的选定部分”。

当把一个问题隔离开时要记住，一条语句可能会间接产生错误，这是因为它为变量赋予了不正确的值。在处于中断模式时，可使用“本地”窗口、“快速监视”、监视表达式或“监视”窗口来检查变量和属性的值。

详细信息 关于如何使用“立即”窗口来测试属性和变量的值，请参阅本章后面的“用‘立即’窗口测试数据和过程”。关于监视表达式的详细信息，请参阅“用监视表达式监视数据”。

使用 Stop 语句

在过程中放置一条 `Stop` 语句是设置断点的一个变通方法。每当 Visual Basic 遇到 `Stop` 语句时，这条语句就中止执行并切换到中断模式。虽然 `Stop` 语句与断点的功能相像，但它们的设置或清除方法却是不同的。

警告 在创建 .exe 执行文件之前要确信删除了所有 Stop 语句。如果独立的 Visual Basic 应用程序 (.exe) 遇到了 Stop 语句，那么应用程序就把它当作 End 语句并立即中断执行，而且不发生任何 QueryUnload 或 Unload 事件。

注意 通常使用 Assert 方法要比使用 Stop 语句更好。Assert 方法只有当不符合指定的条件时才终止执行；不像 Stop 语句，当编译应用程序时，将自动删除对 Assert 方法的调用。详细信息请参阅本章后面的 "Verifying Your Code with Assertions"。

记住，Stop 语句除了暂时中止执行外，不做任何事情，而 End 语句则中止执行、重置变量并返回设计时。总可选择“运行”菜单中的“继续”来继续运行应用程序。

详细信息 请参阅“Stop 语句”。请参阅本章前面的“如何处理错误”中使用 Stop 语句的一个例子。

运行应用程序的选定部分

如果能够识别产生错误的语句，那么单个断点就有助于对问题定位。但更常见的情况是只知道产生错误代码的大体区域。断点有助于将问题区域进行隔离。然后用跟踪和单步执行来观察每个语句的效果。如果有必要，还可在一条新行上开始执行，从而跳过几条语句或倒退回去。

单步执行模式	描述
跟踪	执行当前语句并在下一行处中断执行，即使它处在另一个过程中
单步	执行当前行调用的整个过程，并在当前行之后的一行处中断
单步出	执行当前过程的其余部分，并在调用该过程的语句之后的一条语句处中断

注意 必须在中断模式下使用这些命令。在设计时或运行时它们都是不可用的。

使用跟踪

可用跟踪来一次一条语句地执行代码（这也被称为单步执行）。在单步通过每条语句之后，可以通过查看应用程序的窗体或调试窗口来看它的效果。

要单步执行代码，请

- 选择“调试”菜单的“单步”。

-或-

单击“调试”工具栏的“单步”按钮。（为显示“调试”工具栏，在 Visual Basic 工具栏上单击鼠标右键并选择“调试”选项。）

-或-

按下 F8。

当单步执行整个代码时，Visual Basic 暂时切换到运行时来执行当前语句，并前进到下一条语句。然后又回过头来切换到中断模式。

注意 Visual Basic 允许分别跟踪一些语句，即使它们处在同一行内。一行代码可包含两条或更多语句，其间用冒号 (:) 隔开。Visual Basic 用一个矩形轮廓将要执行的下一条语句醒目标出。断点只应用于多语句行的第一条语句。

使用单步

除了当前语句包含过程调用的情况外，在其它时刻，单步与跟踪是相同的。它们之间的差别是，跟踪将进入到被调用的过程里面，而单步则把被调用过程当成单元来执行，然后到达当前过程的下一条语句。例如，假定一条语句调用了过程 `SetAlarmTime 11, 30, 0`

如果选择跟踪，那么“代码”窗口将显示 `SetAlarmTime` 过程，并把过程的开始语句行设置为当前语句。只有在需要分析 `SetAlarmTime` 的内部代码时，这才是最佳选择。

如果使用单步，则“代码”窗口继续显示当前过程。调用 `SetAlarmTime` 后就立即执行语句，除非 `SetAlarmTime` 包含断点或 `Stop` 语句。如果要停留在同级别代码上，而且不需要分析 `SetAlarmTime` 过程，则选择单步。

可在跟踪和单步之间随意选择。使用什么命令，这取决于在任何已知时刻要分析代码的哪个部分。

要使用单步，请

- 选择“调试”菜单的“单步”。

-或-

单击“调试”工具栏的“单步”按钮。（显示“调试”工具栏，在 Visual Basic 工具栏上单击鼠标右键并选择“调试”选项。）

-或-

按下 **SHIFT+F8**。

使用单步出

除了要运行当前过程的其余代码外，单步出与单步大同小异。如果过程被另一个过程调用，则前者立即到达紧跟在调用该过程的语句之后的语句处。要使用单步出，请

- 选择“调试”菜单的“逐过程”。

-或-

单击“调试”工具栏的“逐过程”按钮。（显示“调试”工具栏，在 Visual Basic 工具栏上单击鼠标右键并选择“调试”选项。）

-或-

按下 **CTRL+SHIFT+F8**。

绕过部分代码

当应用程序处在中断模式时，可用“运行到光标处”命令在代码的后部选择想要停止运行的语句。因此可以略过不感兴趣的那部分代码，比如巨大的循环。

要运行到光标处，请按照以下步骤执行：

1. 把应用程序设置为中断模式。
2. 把光标设置在需要停止运行的地方。
3. 按下 **CTRL+F8**。
-或-
选择“调试”菜单的“运行到光标处”命令。

设置下一条要执行的语句

在调试或试验应用程序时，可以使用“设置下一条语句”命令跳过部分代码——例如，包含已知错误的某部分代码——从而可继续跟踪其它问题。也可返回到前面的语句处，用不同的属性或变量值测试应用程序的某部分。可用 **Visual Basic** 设置不同的代码行来执行下一行，只要这行代码也在同一过程中。此时达到的效果与使用跟踪时相同，只是跟踪仅执行过程的下一行代码。通过设置下一条执行语句，可以选择下一次将执行哪行代码。要设置下一条执行语句，请按照以下步骤执行：

1. 在中断模式下，把插入点（光标）移到下一次要执行的代码行处。
2. 选择“调试”菜单的“设置下一条语句”。
3. 选择“运行”菜单中的“继续”来恢复执行。
-或-
选择“调试”菜单的“运行到光标处”、“单步入”、“单步”或“逐过程”。

显示下一条要执行的语句

可用“显示下一条语句”把光标设置到下一个要执行的行。如果已经执行错误处理程序中的代码，但不能肯定要在哪儿恢复运行，则本功能极为方便。“显示下一条语句”只在中断模式下使用。

要显示下一个将要执行的语句，请按照以下步骤执行：

1. 当处于中断模式时，选择“调试”菜单的“显示下一条语句”。
2. 选择“运行”菜单的“继续”来恢复执行。

-或-

选择“调试”菜单的“运行到光标处”、“单步入”、“单步”或“逐过程”。

监视调用堆栈

在“字体”对话框中显示了所有活动过程调用的一个列表。活动过程调用是应用程序中已启动但尚未完成的过程。

在应用程序执行一系列嵌套过程时，“字体”对话框用有助于跟踪程序的操作。例如，一个事件过程可以调用第二个过程，而第二个过程又可以调用第三个过程——一切都发生在启动这条链的事件过程完成之前。这样的嵌套过程调用很难跟踪，并且还可能把调试过程弄复杂。图 13.17 显示了“调用堆栈”对话框。

注意 如果在空闲循环期间把应用程序置为中断模式，那么，在“调用堆

栈”对话框中将不会出现任何项。

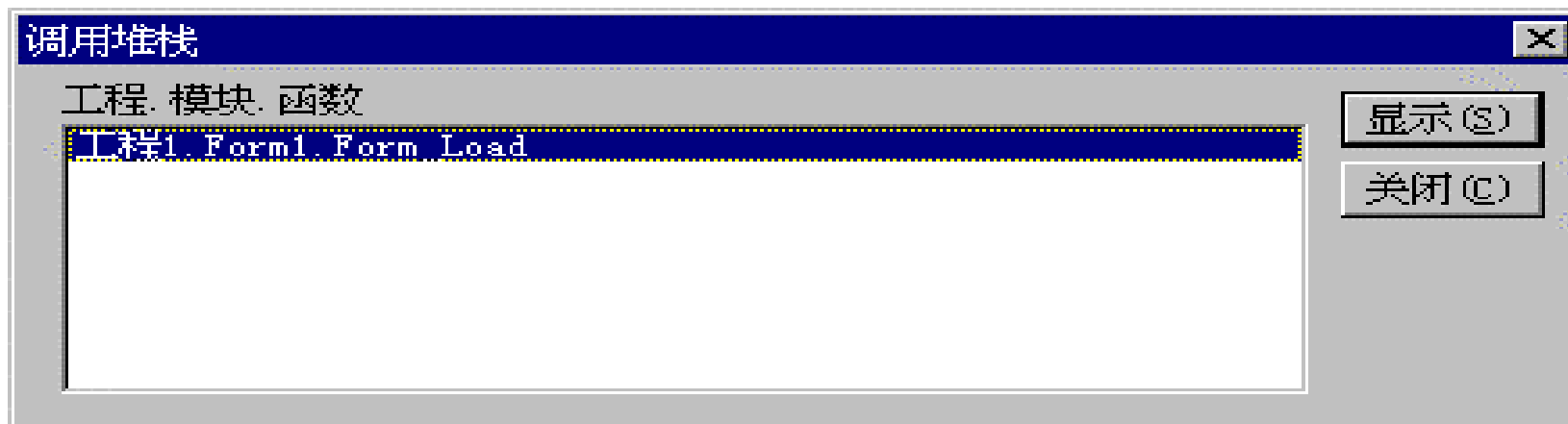


图 13.17 “调用堆栈”对话框

仅在应用程序处于中断模式下使用才可显示“调用堆栈”对话框。
要显示“字体”对话框，请

- 选择“视图”菜单的“调用堆栈”。

-或-

单击“调试”工具栏的“调用堆栈”按钮。（显示“调试”工具栏，在 Visual Basic 工具栏上单击鼠标右键并选择“调试”选项。）

-或-

按下 CTRL+L。

-或-

单击“本地”窗口中“过程”框旁的按钮。

跟踪嵌套过程

“调用堆栈”对话框列出了一系列嵌套调用中的所有活动过程调用。把最早的活动过程调用放在列表底部，而在列表顶部添加随后的过程调用。

每个过程的信息用模块或窗体的名字开头，随后是被调用过程的名字。因为“调用堆栈”对话框不指出分配给窗体实例的变量，所以不对窗体或类的多个实例进行区分。关于窗体的多个实例的详细信息，请参阅第九章“用对象编程”和第六章“设计用户界面”中的“多文档 (MDI) 应用程序”。

可用“字体”对话框显示过程中的语句，该过程将应用程序的控件传递给列表中的下一个过程。

要显示调用“调用堆栈”对话框中的另一个过程的语句，请按照以下步骤执行：

1. 在“调用堆栈”对话框中选择想要显示的过程调用。
2. 选择“显示”按钮。

关闭对话框后，该过程将在“代码”窗口中显示出来。

“代码”窗口中的光标所处位置指出调用“字体”对话框中的下一个过程的语句。如果在“调用堆栈”对话框中选择当前过程，则光标将出现在当前语句处。

检查递归过程

在确定“堆栈溢出”错误是否由递归所致时，“调用堆栈”对话框是很有用

的。递归是例程调用自身的能力。可通过将以下代码添加到一个新工程的窗体中来测试它：

```
Sub Main()  
    Static intX As Integer  
    intX = intX + 1  
    Main  
End Sub
```

```
Private Sub Form_Click()  
    Main  
End Sub
```

运行该应用程序，单击窗体，然后等待“堆栈溢出”的错误消息。选择“调试”按钮，然后选择“视图”菜单的“调用堆栈”。如图 13.18 所示，可看到许多 Main 过程调用。

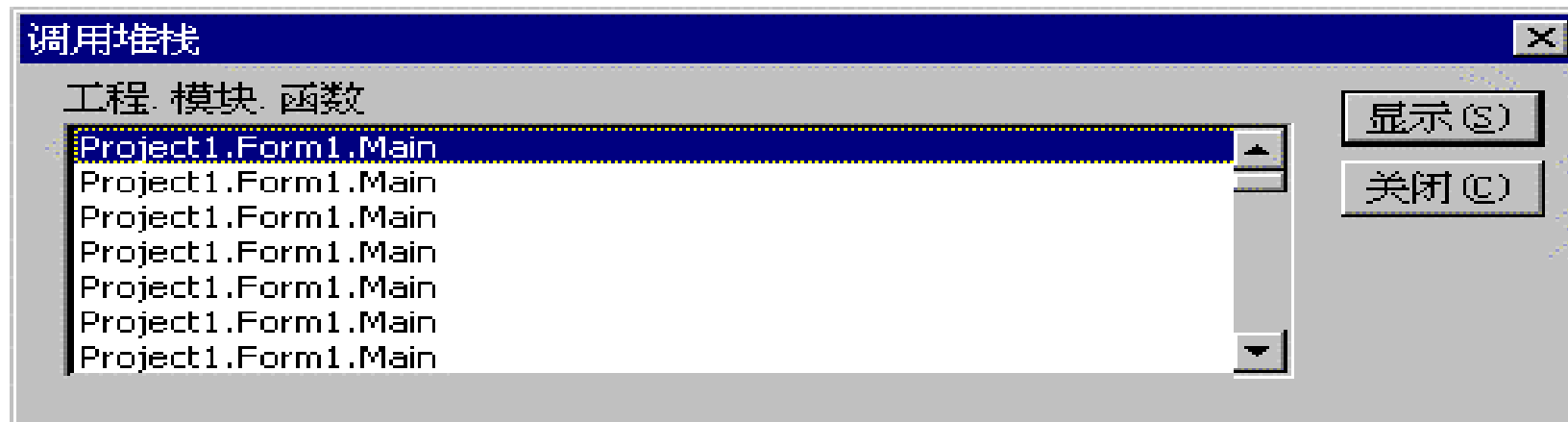


图 13.18 “调用堆栈”对话框列出递归过程

像进行复选那样在“代码”窗口中突出显示 `intX`，并选择“调试”菜单的“快速监视”。`intX` 的值表示的是中断前执行 `Main` 过程的次数。

用“立即”窗口测试数据和过程

有时，当调试或试验一个应用程序时，可能要执行单个过程、对表达式求值或为变量或属性赋予新的值。可用“立即”窗口完成这些任务。可以通过在“立即”窗口中显示表达式的值来计算表达式。

在“立即”窗口中打印信息

在“立即”窗口中有两种方法进行打印。

- 将 `Debug.Print` 语句写入应用程序代码。
- 直接在“立即”窗口中输入 `Print` 方法。

对于监视表达式，这些打印技术有多个优点：

- 为得到反馈信息以获知关于应用程序执行得如何，不必中断执行。在运行应用程序时，可看到显示出的数据或其它信息。
- 由于在一个单独的区域内（“立即”窗口）显示反馈，所以反馈不会影响用户看到的输出。
- 因为可以把这个代码作为窗体的一部分来保存，所以下次处理应用程序时不必重新定义这些语句。

通过应用程序代码来打印

无论何时，只要把 `Debug` 对象的前缀包括进去，`Print` 方法就把输出传送到“立即”窗口：

```
Debug.Print [items][;]
```

例如，下面的语句在每次执行时都把 `Salary` 的值打印到“立即”窗口中：

```
Debug.Print "Salary = "; Salary
```

当应用程序代码中有一个特殊位置，并且知道变量（在这个例子中为 `Salary`）在这个位置上会发生变化时，这种技术最为适用。例如，可以在循环中放置以前的语句来重复改变 `Salary`。

注意 当把应用程序编译到 `.exe` 执行文件时，将删掉 `Debug.Print` 语句。

于是，如果在使用 `Debug.Print` 语句时，应用程序仅仅把字符串或简单变量类型当作参数，那么它将没有任何 `Debug.Print` 语句。但是，Visual Basic 不能去掉在 `Debug.Print` 中作为参数的函数调用。所以，那些函数的副作用将继续在编译后的 `.exe` 执行文件中存在，即使不打印函数结果，情况也是如此。

详细信息 请参阅“调试对象”。

从“立即”窗口内打印

一旦在中断模式之下，就可把焦点移到“立即”窗口来检查数据。

要检查“立即”窗口中的数据，请按照以下步骤执行：

1. 单击“立即”窗口（如果可见的话）。

-或-

选择“视图”菜单的“立即窗口”。

一旦把焦点移到“立即”窗口，就可用不带 `Debug` 对象的 `Print` 方法。

2. 将语句键入或粘贴到“立即”窗口中，然后按下 `ENTER` 键。

“立即”窗口执行语句作出响应，如图 13.19 所示。

问号 (?) 是 `Print` 方法常用的速记形式。问号与 `Print` 具有相同的意义，并且可在使用 `Print` 的任何上下文中使用。例如，可按照图 13.20 所示形式输入图 13.19 中的语句。



图 13.19 使用 Print 方法来打印到“立即”窗口中

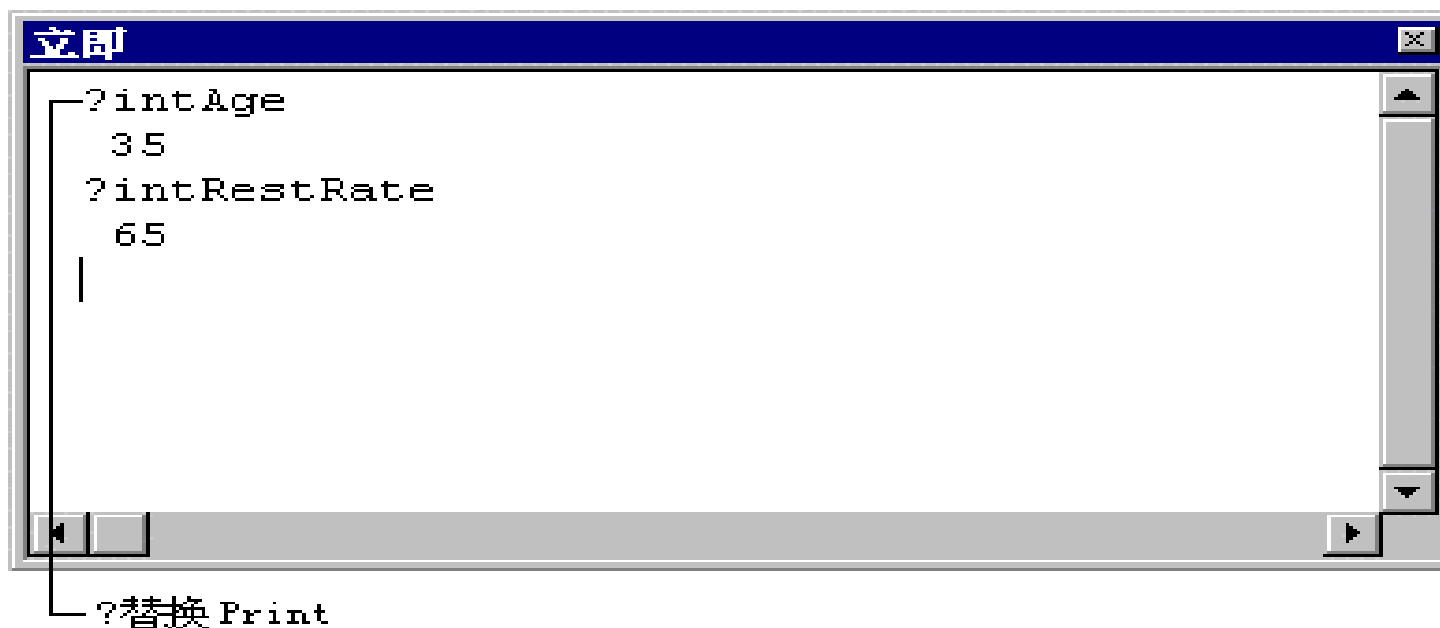


图 13.20 使用问号来代替 Print 方法

打印属性值

可在“立即”窗口中计算任何有效的表达式，包括调用了属性的表达式。当前活动窗体或模块决定了表达式的范围。如果在与窗体或类相关联的代码内中止了执行，则可引用那个窗体（或其控件之一）的属性，也可以像下面这样，用语句隐式地引用窗体：

? BackColor

? Text1.Height

假定 Text1 是当前活动窗体中的控件，第一个语句将在“立即”窗口中打

印当前窗体的背景颜色的数值。第二个语句打印 `Text1` 的高。
如果在模块或另一窗体中将执行挂起来，那么必须明确指定窗体名：

```
? Form1.BackColor
```

```
? Form1.Text1.Height
```

注意 在“立即”窗口中（或其它任何地方）引用卸载的窗体来装载那个窗体。

详细信息 关于在“立即”窗口中改变属性和值的详细信息，请参阅本章后面的“给变量和属性赋值”。

为变量和属性赋值

在开始隔离一个可能产生错误的原因时，或许要测试特定数据值的效果。当处于中断模式下时，可以像下面这样在“立即”窗口中用语句设置值：

```
BackColor = 255
```

```
VScroll1.Value = 100
```

```
MaxRows = 50
```

第一个语句改变当前活动窗体的属性，第二个语句改变 `VScroll1` 的一个属性，第三个语句则给变量赋值。

在设置了一个或多个属性和变量的值后，可以继续执行并观察结果。还可像下一个主题“用‘立即’窗口测试过程”描述的那样来测试过程的效果。

用“立即”窗口测试过程

“立即”窗口可对任何有效的 Visual Basic 可执行语句求值，却不接受数据声明。但是，可以输入 Sub 和 Function 过程调用，由此可用任何给定的一系列参数来测试过程的可能效果。简而言之，在“立即”窗口中输入一个语句（在中断模式下）与在“代码”窗口中是一样的。例如：

```
X = Quadratic(2, 8, 8)
```

```
DisplayGraph 50, Arr1
```

```
Form_MouseDown 1, 0, 100, 100
```

当按下 Enter 键时，Visual Basic 会切换到运行来执行语句，然后再返回到中断模式。此时可观察结果并测试变量或属性值的任何可能效果。

过程调用的范围与变量的范围一样。可在当前活动窗体内调用任何过程。除非把过程定义为 Private（在这种情况下，只有在模块中执行时才能调用这个过程），在模块中总可以调用过程。

详细信息 在第五章“编程基础”的“变量、常量和数据类型概述”中讨论了范围的问题。

查看和测试过程的多个实例

可用“立即”窗口重复运行一个过程以测试不同条件的效果。Visual Basic 把过程的每个独立的调用都作为独立的实例来维护。这就可在过程的每个实

例中独立测试变量和属性设置值。为看到这是如何工作的，打开一个新的工程并把下列代码添加到窗体模块中：

```
Private Sub Form_Click()
```

```
    AProcedure
```

```
End Sub
```

```
Sub AProcedure()
```

```
    Dim intX As Integer
```

```
    intX = 10
```

```
    BProcedure
```

```
End Sub
```

```
Sub BProcedure()
```

```
    Stop
```

```
End Sub
```

运行应用程序并单击窗体。Stop 语句把 Visual Basic 置为中断模式并显示“立即”窗口。在过程 "AProcedure" 中把 intX 的值改为 15，然后切换到“立即”窗口并键入：

```
AProcedure
```

这样做就会调用了过程 "AProcedure" 并重新启动应用程序。如果切换到“立即”窗口并再次运行 "AProcedure"，然后打开“字体”对话框，将看到与图 13.21 中很相像的列表。每个单独的程序运行实例都被列出，并被

[<Debug Window >] 列表分隔开。

Visual Basic 维护“立即”窗口的每个命令所执行的过程列表。较新的列表处在表的顶部。可用“字体”对话框选定过程的任何实例，然后在“立即”窗口中打印过程中的变量值。

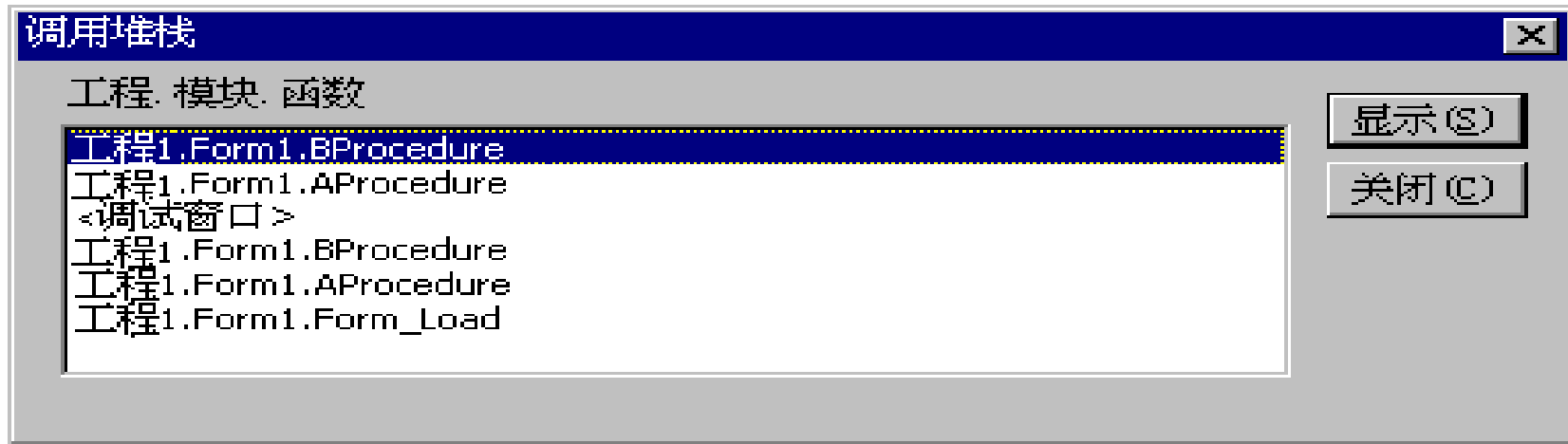


图 13.21 “调用堆栈”对话框显示过程的多个实例

例如，如果双击“AProcedure”的最早实例并使用“立即”窗口打印 intX 的值，则值返回 10，如图 13.22 所示。如果在第二次运行“AProcedure”时把 intX 的值变成 15，则将那个值与过程的第二个实例一起存储。

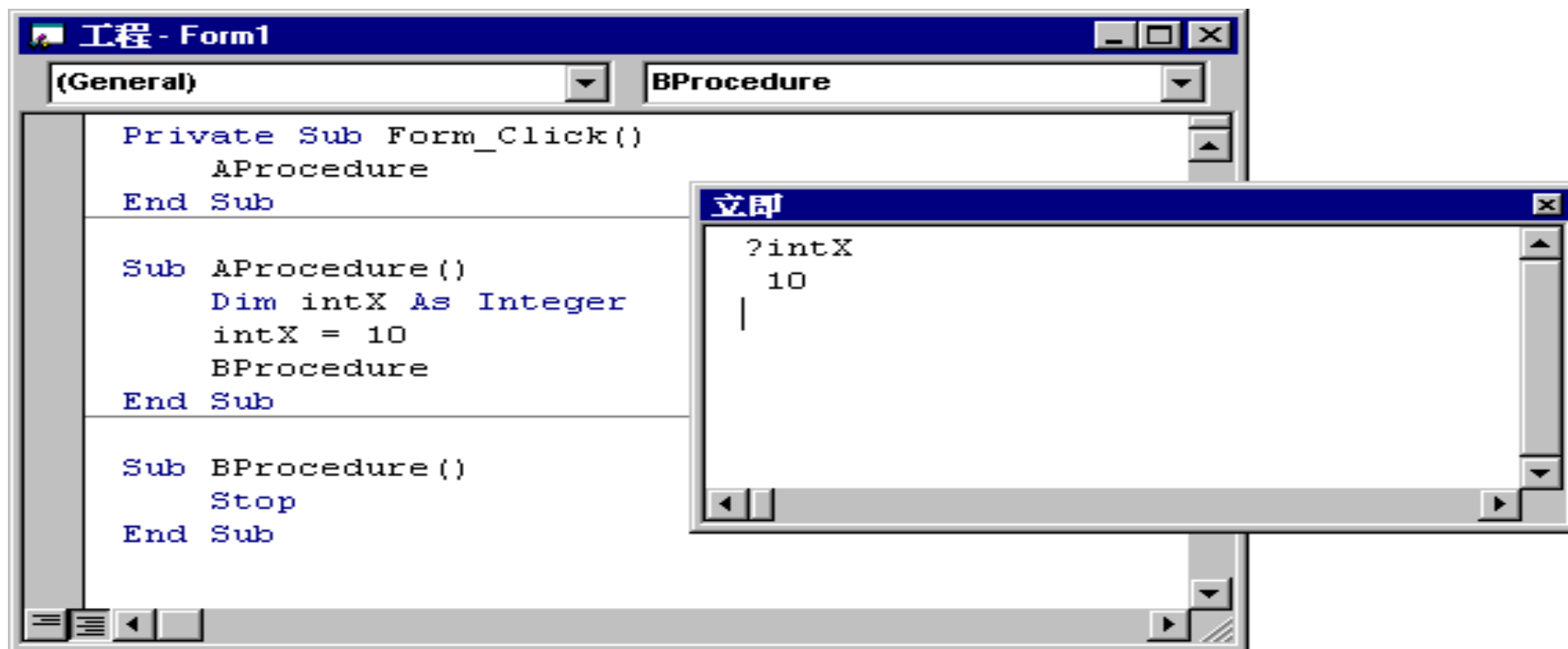


图 13.22 在“立即”窗口中打印变量的值

注意 虽然“立即”窗口支持大多数语句，但只有能够在一行代码内完全表示控件结构时，这个结构才是有效的；用冒号将组成控件结构的各个语句隔开。下面的 For 循环在“立即”窗口中是有效的：

```
For I = 1 To 20 : Print 2 * I : Next I
```

检查错误号

可用“立即”窗口显示与特定错误号相联系的消息。例如，在“立即”窗口中输入以下语句：

```
error 58
```

然后按下 **ENTER** 执行该语句。相应的错误信息出现在“立即”窗口中，如图 13.23 所示。

使用“立即”窗口的提示

可在“立即”窗口中使用下列快捷键：

- 可以使用 Data Tips 检查变量或对象的值。Data Tips 与 ToolTips 类似，当鼠标在 Break 模式中的代码窗口中的变量或对象属性上按下时，显示当前值。Data Tips 的显示受当前范围变量或对象的限制。
- Data Tips 在中断模式的立即窗口中也可以使用。与代码编辑器不同的是，如果提供了完整限定对象名，立即窗口将显示对象的属性，而不考虑范围。例如：Form1.Text1.Width 的 Data Tips 将总显示，而 Text1.Width 却不是，除非 Text1 在当前范围内。
- 一旦输入语句，就可把插入点移动到该语句并在那行的任何位置按下 **ENTER**，从而再次运行它。
- 在按下 **ENTER** 之前，可编辑当前语句来改变效果。
- 可用鼠标或方向键在“立即”窗口中到处移动。除非处在要执行的语句处，否则不要按下 **ENTER** 键。

- CTRL+HOME 将把插入点设置到“立即”窗口的顶部,而 CTRL+END 将把插入点设置到窗口的底部。
- HOME 和 END 键分别使插入点移动到当前行的起点和终点。



图 13.23 在“立即”窗口中显示错误信息

注意 虽然“立即”窗口主要用于中断模式，但在设计模式下使用“立即”窗口也是可能的。为了在设计环境中调试 ActiveX 部件而提供了这种能力。在设计时使用“立即窗口”，对于标准工程类型可能会引起不可预见的结果。

特别的调试考虑

某些事件是使用 `Microsoft Windows` 时的公共部分，它仍很可能会对调试应用程序造成潜在问题。重要的在于知道这些特殊问题，这样就不至于把调试过程搞得混乱不堪。

如果始终都知道中断模式到底如何行事，不按照应用程序的预期目标处置事件，则总可找到解决方案。在某些事件过程中，用 `Debug.Print` 语句而不用监视表达式或断点监视变量或属性的值。也有必要改变与事件序列有关的变量的值。下列主题将讨论到这个问题。

在 `MouseDown` 期间中断执行

如果在 `MouseDown` 事件过程期间中断执行，则可松开鼠标按钮或使用鼠标来做任何数量的任务。但在继续执行时，应用程序会认为鼠标按钮一直是被按下的。在再次按下鼠标按钮并接着松开它之前不可能获得 `MouseUp` 事件。

在运行时按下鼠标按钮将会认为在 `MouseDown` 事件过程中有一断点，从而在此处再次中断执行。在这种情况下将再也得不到 `MouseUp` 事件。解决的办法通常是删除 `MouseDown` 过程中的断点。

在 KeyDown 期间中断执行

如果在 KeyDown 期间中断执行，则可有大致相同的考虑。如果在 KeyDown 过程中保留一个断点，则可能再也得不到 KeyUp 事件。（在第十一章“响应鼠标和键盘事件”中描述了 KeyDown 和 KeyUp）。

在 GotFocus 或 LostFocus 期间中断执行

如果在 GotFocus 或 LostFocus 事件过程期间中断执行，则系统消息的计时可能会产生不一致的结果。应使用 Debug.Print 语句代替 GotFocus 或 LostFocus 事件过程中的断点。

模拟对话框和消息框禁止事件

由于调试器中潜在的冲突问题，当显示一个模拟窗体或消息框时，开发环境不能产生事件。这样一来，事件就被禁止，直到该模拟窗体或消息框被释放为止。

重点事件的禁止只发生在开发环境中。一旦项目编译通过后，即使显示模拟窗体或消息框时也仍旧会产生事件。

下面的例子是一些导致这种禁止的情形：

- 一个带有 Timer 控件的窗体正在开发环境中运行。选择“工具”菜单中的“选项”将打开“选项”对话框，此对话框是模拟对话框。Timer 控

件的 `Timer` 事件将被禁止，直到此对话框被释放。

- 设计时，一个带有 `Timer` 控件的 `UserControl` 的实例被放到一个窗体上。（这个计时器可能是用来动画显示此控件的；这种动画效果甚至在设计时也会产生，因为在设计时控件可以执行代码。）选择“工程”菜单的“添加类模块”将打开“添加类模块”对话框，此对话框是模拟对话框。`Timer` 控件的 `Timer` 事件将被禁止，直到此对话框被释放为止。
- 一个 `UserDocument` 包含一个 `Timer` 控件以及一个显示消息框的命令按钮。如果使用 `Internet Explorer` 调试此 `UserDocument`，按下命令按钮来显示消息框将导致 `Timer` 控件的 `Timer` 事件被禁止，直到此消息框被释放为止。

测试和使用命令行参数

可以做这样的选择，即让应用程序使用命令行参数，在启动时，这些参数将向应用程序提供数据。用户可选择操作环境的“运行”命令来输入它们，然后在应用程序名之后键入参数。在创建应用程序的图标时也可使用命令行参数。

例如，假定要创建闹钟应用程序。一种设置闹钟时间的方法就是让用户直接键入选定的时间。用户可在“运行”对话框中键入以下字符串：

```
Alarm 11:00:00
```

`Command` 函数返回所有参数，这些参数是在应用程序名（本例中为 `Alarm`）之后键入的。因为闹钟应用程序只有一个参数，所以在应用程序代码中可

直接把参数赋给存储了选定时间的字符串：

```
AlarmTime = Command
```

如果 `Command` 返回空字符串，则无命令行参数。应用程序必须直接请求信息或者选择默认操作。

为测试使用 `Command` 的代码，可在 `Visual Basic` 环境中指定示例的命令行参数。应用程序对示例的命令行输入进行计算的方式与用户键入参数时是一样的。

要设置示例的命令行参数，请按照以下步骤执行：

1. 选择“工程”菜单的“属性”。
2. 单击“工程属性”对话框的“生成”选项卡。
3. 在“命令”行参数域中键入示例的参数。（不要键入应用程序名本身。）
4. 选择“确定”。
5. 运行这个应用程序。

详细信息 请参阅“`Command` 函数”。

在编译之前删除调试信息

如果不想在发布给用户的应用程序中包含调试信息语句，则在使用“制作 EXE 文件”命令时，用条件编译来删除这些语句亦很方便。

例如：

```
Sub Assert(Expr As Boolean, Msg As String)
```

```

        If Not Expr Then
            MsgBox Msg
        End If
    End Sub

Sub AProcedure(intX As Integer)
    # If fDebug Then
        Assert intX < 10000 and intX > 0, _
            "Argument out of range"
    # End If
    ' 代码现在可取正确的值。
End Sub

```

因为对 `Assert` 过程的调用是条件编译的，所以，如果把 `fDebug` 设置为 `True`，则只有 `.exe` 文件才包含这个调用。在编译应用程序的发行版本时，把 `fDebug` 设置为 `False`。这样 `.exe` 文件就会尽可能作到小、快、灵。

注意 自从发行 Visual Basic 5.0 版开始就不再有必要创建自己的 `Assert` 过程。`Debug.Assert` 语句完成相同的功能并可从被编译的代码中自行去掉。详细信息请参阅本章后面的“用断言检验代码”。

用断言检验代码

断言是一种方便的方法，它可以测试应该在您的代码中特定点出现的条件。设想一个 `Assert` 语句如同做一个假设。如果您的假设是 `True`，将忽略该断言；如果您的假设是 `False`，VB 将会引起您的注意。

在 Visual Basic 中，断言总是以方法的形式出现：`Debug` 对象的 `Assert` 方法。`Assert` 方法需要一个 `Boolean` 类型参数，这个参数规定计算的条件。`Assert` 方法的语法如下：

Debug.Assert(*boolean expression*)

`Debug.Assert` 语句不会出现在编译后的应用程序中，但是，在设计环境中运行时，它将使应用程序进入中断模式，并突出显示包含这条语句的代码行（假定表达式的值为 `False`）。下例说明了 `Debug.Assert` 语句的用法：

```
Debug.Assert Trim(CustName) = "John Doe"
```

在上例中，如果 `CustName` 的值不是 `John Doe`，则应用程序将进入中断模式；否则将正常执行。使用 `Debug.Assert` 与选定“当值为 `True` 时中断”选项来设置监视大致相同，只是前者在表达式的值为假时将中断执行。

使用请求时编译

请求时编译和后台编译是使应用程序在开发环境中快速运行的相关功能。直到编译出整个工程的 `.exe` 文件之前，使用这些功能可能会隐藏代码中的编译错误。这两个功能被默认为是打开的，在“工具”菜单的“选项”对话框的“通用”选项卡中可打开或关闭这些功能。

在开发环境中，请求时编译只允许应用程序在必要时才编译代码。在打开请求时编译功能并选择“运行”菜单中的“启动”（或按 F5 键）时只编译运行应用程序所必须的代码。于是，在开发环境中运用应用程序的功能越多，需要编译的代码也就越多。

在开发环境中，如果不出现其它操作，则后台编译允许 Visual Basic 在运行时继续编译代码。

当打开这些功能并在开发环境中运行工程时，有些代码可能不会被编译。于是，当选择了“生成 EXE 文件”命令（或关闭请求时编译功能）时，重新编译代码后将会看到意想不到的新错误。

在开发过程的里程碑处或者其它任何时刻，有三种方法可用来消除请求时编译隐藏的误差。

- 关闭请求时编译功能并运行应用程序。这将强制 Visual Basic 检查整个应用程序以查出编译误差。
- 生成工程的可执行文件。这也将强制 Visual Basic 检查整个应用程序以查出编译误差。
- 在“运行”菜单中选择“全编译执行”。

调试的提示

简化调试有以下几种方式：

- 当应用程序产生的结果不正确时，请浏览代码并试图查找可能产生问题

的语句。在这些语句设置断点并重新启动应用程序。

- 当程序运行暂停时，测试重要变量和属性的值。使用“快速监视”命令或设置监视表达式来监视这些值。使用“立即”窗口来检查变量和表达式。
- 用“发生错误时中断”选项来确定错误发生之处。若要临时改变这个选项，从“代码窗口上下文”菜单中选择“切换”命令，然后在子菜单中切换选项。单步执行代码，使用监视表达式和“本地”窗口来监视在代码运行中值的变化方式。
- 如果错误发生在循环语句中，则定义一个中断表达式来确定问题出现的地方。在修改代码之后使用“立即”窗口和“设置下一条语句”命令来重新执行这段循环。
- 如果确知是变量或属性导致应用程序的错误，那么当变量或属性被赋予错值时，使用 `Debug.Assert` 语句来中断执行。
- 为了设置 Visual Basic 在任何调试会话期开始时的错误捕获状态的缺省值，打开“选项”对话框（可从“工具”菜单中获得），选择“通用”选项卡，设置“缺省的错误捕获状态”选项。在下次启动 Visual Basic 时将使用这个设置值，即使为另一个工程输入了该设置值。

有时候，可能会碰到一个很难跟踪的错误。请别紧张——可以试着做下面一些事情：

- 首先最重要的，应做一个备份。这是一个关键的地方，即使是有经验的程序员，也常常会损失好几个小时的工作。当进行调试时，很容易意外地覆盖或删除掉必要的代码部分。

- 使用 Visual Basic 的内置调试功能。试着找到出错的代码行或代码部分。隔离这些代码。如果能将问题隔离到一个代码块中，请试着用从程序其余部分分离出的代码块来重现问题。选择并复制这些代码，开始一个新工程，将这些代码粘贴到新工程中，运行新工程，然后看看是否还出现同样的错误。
- 创建一个日志文件。如果您不能分离出代码，或者问题变化不定，或者问题只在编译的时候出现，那么 Visual Basic 的调试功能就不太有效了。在这种情况下，可以创建一个日志文件来记录程序的活动。这将能够帮您逐步地找出可疑代码的位置。在程序中的不同位置调用如下的过程。在字符串中传递的信息应能够表明您的程序中正在执行的代码的当前位置。

```
Sub LogFile (Message As String)
    Dim LogFile As Integer
    LogFile = FreeFile
    Open "C:\VB\LogFile.Log" For Append As #LogFile
    Print #LogFile, Message
    Close #LogFile
End Sub
```

```
Sub Sub1 ()
    '...
    Call LogFile("Here I am in Sub1")
```

'...

End Sub

- 简化问题。如果可能的话，请移走工程中任何第三方控件和自定义控件。用 Visual Basic 的标准控件替换它们。排除任何看来和问题无关的代码。
- 缩小搜索空间。如果以上的任何方法都不能解决问题，那么现在应该从问题的搜索空间中排除所有其他的非 Visual Basic 因素。将 AUTOEXEC.BAT 和 CONFIG.SYS 文件复制到备份文件中。将这两个文件中所有在 Windows 中运行您的程序时可以不用了的驱动程序和应用程序全部注释掉。将 Windows 视频驱动程序设为标准的 Windows VGA 驱动。关闭 Windows 并重新启动计算机。这一步将消除某些其它驱动程序和应用程序干扰您的程序的可能性。
- 如果使用以上所有的方法，还不能找到解决办法，也不能分离或解决问题的话，那就只能寻求帮助了。请参阅技术支持文档。

详细信息 本章前面的“用断点有选择地中断执行”描述了有关断点的内容。关于监视表达式的详细信息，请参阅“用监视表达式监视数据”。“用‘立即’窗口测试数据和过程”中讨论了“立即”窗口的内容。关于 Debug 对象的 Assert 方法的详细信息，请参阅“用断言检验代码”。

第十四章 处理驱动器、文件夹和文件

在 Windows 中编程时，下面这些功能是非常重要的：包括添加、移动、改变、创建、或删除文件夹（目录）和文件，以及获得驱动器的信息和操作驱动器。

Visual Basic 允许用如下两种不同的方法来处理驱动器、文件夹和文件：使用传统的方法诸如 Open 语句、Write# 等等，以及使用一套新的工具 File System Object (FSO) 对象模型。

目录

- File System Object 模型介绍
- 用 FSO 对象模型编程
- 用传统的文件 I/O 语句和函数处理文件

File System Object 模型介绍

Visual Basic 的一个新功能是 File System Object (FSO) 对象模型，该模型提供了一个基于对象的工具来处理文件夹和文件。这使您除了使用传统的 Visual Basic 语句和命令之外，还可以使用您所熟悉的带有一整套属性、方

法和事件的 `object.method` 语法来处理文件夹和文件。

FSO 对象模型使应用程序能够创建、改变、移动和删除文件夹，或者检测是否存在指定的文件夹，如果存在，在哪。FSO 对象模型也能使您获取关于文件夹的信息，诸如名称、创建日期或最近修改日期等等。

FSO 对象模型也使得对文件的处理变得更加简单。在处理文件时，首要目标就是以一种可以有效利用空间和资源、并且易于存取的格式来存储数据。需要能够创建文件、插入和修改数据、以及输出（读）数据。虽然可以将数据存储诸如 Jet 或 SQL 这样的数据库中，但是这样做将在应用程序中加入相当数量的额外开支。出于多种原因，您不想有这样的额外开支，或者数据存取要求不需要用一个与全功能数据库关联的所有额外功能。在这种情况下，用二进制或文本文件来存储数据是最有效的解决方法。

包含在 Scripting 类型库 (Scriun.Dll) 中的 FSO 对象模型支持通过 TextStream 对象创建和操作文本文件。但是，它还不支持二进制文件的创建和操作。要操作二进制文件，请使用带 Binary 标志的 Open 命令。关于如何操作二进制文件的详细信息，请参阅本章的“使用二进制文件访问”部分。

文件系统对象 (File System Object)

FSO 对象模型包括这些对象：

对象	描述
Drive	允许收集关于系统所用的驱动器的信息，诸如驱动器有多少可用空间，其共享名称是什么，等等。请注意，一个“驱动器”并不一定是一个硬盘。它可以是 CD-ROM 驱动器、一个 RAM 盘等等。而且，驱动器不一定是和系统物理地连接；也可以通过一个 LAN 进行逻辑地连接
Folder	允许创建、删除或移动文件夹，并向系统查询文件夹的名称、路径等。
Files	允许创建、删除或移动文件，并向系统查询文件的名称、路径等
FileSystemObject	该组的主要对象，提供一整套用于创建、删除、收集相关信息、以及通常的操作驱动器、文件夹和文件的方法。与本对象相关联的很多方法复制了其它对象中的方法
TextStream	允许您读和写文本文件

关于 FSO 对象模型中各种属性、方法和事件的信息，请使用 Visual Basic 中的“对象浏览器”（按 F2 键）并查看 Scripting 类型库。

FSO 对象模型编程

FSO 对象模型编程包括三项主要任务：

- 使用 `CreateObject` 方法，或将一个变量声明为 `FileSystemObject` 对象类型来创建一个 `FileSystemObject` 对象。
- 对新创建的对象使用适当的方法。
- 访问该对象的属性。

FSO 对象模型包含在一个称为 `Scripting` 的类型库中，此类型库位于 `Sccrrun.Dll` 文件中。如果还没有引用此文件，请从“属性”菜单的“引用”对话框选择“`Microsoft Scripting Runtime`”项。然后就可以使用“对象浏览器”来查看其对象、集合、属性、方法、事件以及它的常数。

创建 `FileSystemObject` 对象

第一步是创建一个 `FileSystemObject` 对象以便进行处理。可以通过如下两种方法完成：

- 将一个变量声明为 `FileSystemObject` 对象类型：

```
Dim fso As New FileSystemObject
```

- 使用 `CreateObject` 方法来创建一个 `FileSystemObject` 对象：

```
Set fso = CreateObject("Scripting.FileSystemObject")
```

在上面的语法中，`Scripting` 是类型库的名称，而 `FileSystemObject` 则是想

要创建一个实例的对象的名称。

注意 第一种方法只在 Visual Basic 中有效，而第二种方法在 Visual Basic 或 VBScript 中都是可行的。

使用适当的方法

下一步就是使用该 FileSystemObject 对象的适当方法。例如，如果想要创建一个新对象，既可以使用 CreateFolder 方法也可以使用 CreateTextFile 方法（FSO 对象模型不支持创建或删除驱动器）。

如果想要删除对象，可以使用 FileSystemObject 对象的 DeleteFile 和 DeleteFolder 方法，或者 File 和 Folder 对象的 Delete 方法。

使用适当的方法，还可以复制、移动文件和文件夹。

请注意，FileSystemObject 对象模型中有一些功能是冗余的。例如，要复制一个文件，既可以使用 FileSystemObject 对象的 CopyFile 方法，也可以使用 File 对象的 Copy 方法。这两者的功能是完全相同的。两者同时提供最大的编程灵活性。

访问已有的驱动器、文件和文件夹

要访问一个已有的驱动器、文件或文件夹，请使用 FileSystemObject 对象中相应的 "get" 方法：

- GetDrive
- GetFolder

- **GetFile**

例如：

```
Dim fso As New FileSystemObject, fil As File
Set fil = fso.GetFile("c:\test.txt")
```

但是，请注意，并不需要对新创建的对象使用 "get" 方法，因为 "create" 函数已经返回了一个句柄到新创建的对象。例如，如果使用 **CreateFolder** 方法创建了一个新的文件夹，就没有必要使用 **GetFolder** 方法来访问该对象的诸如名称、路径、大小等属性。只要给 **CreateFolder** 函数设置一个变量来获取新建文件夹的句柄，然后就可以访问其属性、方法和事件：

```
Private Sub Create_Folder()
    Dim fso As New FileSystemObject, fldr As Folder
    Set fldr = fso.CreateFolder("C:\MyTest")
    MsgBox "Created folder: " & fldr.Name
End Sub
```

访问对象的属性

一旦有了对象的句柄，就能够访问其属性。例如，假定想要获得一个特定文件夹的名称。首先要创建该对象的一个实例，然后通过适当的方法（在本例中是 **GetFolder** 方法，因为该文件夹已经存在）得到其句柄：

```
Set fldr = fso.GetFolder("c:\")
```

现在有了一个 **Folder** 对象的句柄，可以查看其 **Name** 属性：

```
Debug.Print "Folder name is: "; fldr.Name
```

如果想要找出一个文件的最新修改时间，可以使用如下的语法：

```
Dim fso As New FileSystemObject, fil As File
```

```
Set fil = fso.GetFile("c:\detlog.txt") ' 获得要查询的 File 对象。
```

```
Debug.Print "File last modified: "; fil.DateLastModified ' 打印信息。
```

使用驱动器和文件夹

可以在程序中使用 FSO 对象模型来处理驱动器和文件夹，如同在“Windows 资源管理器”中能进行的交互方式一样。可以复制和移动文件夹，获得驱动器和文件夹的信息，等等。

获得驱动器的信息

Drive 对象允许获得一个系统的各个驱动器的信息，这些驱动器可以是物理的，也可以是位于网络上的。通过该对象的属性可以获得下列信息：

- 以字节表示的驱动器总空间 (TotalSize 属性)
- 以字节表示的驱动器可用空间 (AvailableSpace 或 FreeSpace 属性)
- 为驱动器指定的字母号 (DriveLetter 属性)
- 驱动器类型，诸如可移动的、固定的、网络、CD-ROM、或者 RAM 盘 (DriveType 属性)
- 驱动器序列号 (SerialNumber 属性)

- 驱动器使用的文件系统类型, 诸如 FAT、FAT32、NTFS 等等 (FileSystem 属性)
- 驱动器是否可用 (IsReady 属性)
- 共享和/或卷标的名称 (ShareName 和 VolumeName 属性)
- 驱动器的路径或根文件夹 (Path 和 RootFolder 属性)

Drive 对象用法示例

下列示例说明了如何使用 Drive 对象来收集有关驱动器的信息。请记住, 在下面的代码中并没有对一个实际 Drive 对象的引用; 而是使用 GetDrive 方法来获得一个对已有 Drive 对象 (本例中就是 drv) 的引用:

```
Private Sub Command3_Click()  
    Dim fso As New FileSystemObject, drv As Drive, s As String  
    Set drv = fso.GetDrive(fso.GetDriveName("c:"))  
    s = "Drive " & UCase("c:") & " - "  
    s = s & drv.VolumeName & vbCrLf  
    s = s & "Total Space: " & FormatNumber(drv.TotalSize / 1024, 0)  
    s = s & " Kb" & vbCrLf  
    s = s & "Free Space: " & FormatNumber(drv.FreeSpace / 1024, 0)  
    s = s & " Kb" & vbCrLf  
    MsgBox s  
End Sub
```

使用 **CurDir**、**ChDrive**、**ChDir** 或 **App.Path**

如果使用 **CurDir** 函数、**ChDrive** 和 **ChDir** 语句、或者 **Path** 属性 (**App.Path**)，请注意，它们返回的有可能是一个 UNC 路径 (即，形如 `\\Server\Share`) 而不是一个驱动器路径 (如 `E:\Folder`)，这取决于运行程序或工程的方式。

在如下的情况，**App.Path** 返回一个 UNC 路径：

- 当运行一个从共享网络加载的工程时，即使该网络共享已经被映射到一个驱动器号。
- 当从一个共享网络运行一个已编译的可执行文件时（但是仅当通过一个 UNC 路径运行该程序）。

ChDrive 不能处理 UNC 路径，因此当 **App.Path** 返回一个 UNC 路径给 **ChDrive** 时，就会产生一个错误。要避免这个错误，可以在 **ChDrive** 语句之前添加 **On Error Resume Next**，或者测试 **App.Path** 的头两个字符是否是反斜杠：

```
On Error Resume Next
```

```
ChDrive App.Path
```

```
ChDir App.Path
```

这个改动将处理所有使用 UNC 路径从 Windows 启动程序（例如，在“启动”菜单的“运行”对话框中）的情况，因为 Windows 将当前的目录设为一个 UNC 路径。**ChDir** 可以正确地处理 UNC 路径之间的转换（**ChDrive** 的失效可以被忽略，因为 UNC 路径当中没有驱动器号）。

然而，如果在 MS-DOS 的命令行提示符后键入一个 UNC 路径来运行程序的话，上面的代码不会生效。这是因为命令行提示符用的总是当前目录的驱动器路径，所以 CurDir 被设为一个驱动器路径。ChDir 不会产生错误，但是它不能将一个目录从驱动器路径变为 UNC 路径。在这种情况下，唯一的解决办法是定位一个已经映射为 UNC 路径中指定共享的本地驱动器，或者使用网络命令来创建这样一个映射。

如果工程是从一个网络共享——UNC 路径或映射的驱动器路径——加载到 Visual Basic 集成开发环境（IDE）中的话，那么当运行工程时 App.Path 将返回一个 UNC 路径，而 ChDrive 失败并产生一个错误。ChDir 虽然不会出错，但目录并未改变。唯一的解决办法是手工来设置驱动器和目录：

```
Const PROJECTSHARE = "E:\VBPROJ\MYPROJECT"
```

```
#Const Debug = True
```

```
#If Debug Then
```

```
    ChDrive PROJECTSHARE
```

```
    ChDir PROJECTSHARE
```

```
#Else
```

```
    On Error Resume Next
```

```
    ChDrive App.Path
```

```
    ChDir App.Path
```

```
#End If
```

如果不止一个人可能在共享网络上打开该工程，那么可以使用一个 DOS 环境变量来允许每个人都拥有自己对该共享的映射：

```
#Const Debug = True
#If Debug Then
    ChDrive Environ("MYPROJECTDIR")
    ChDir Environ("MYPROJECTDIR")
#Else
    On Error Resume Next
    ChDrive App.Path
    ChDir App.Path
#End If
```

MYPROJECTDIR 的值指定了被映射的驱动器号和路径，例如：

```
SET MYPROJECTDIR=M:\VBProj\MyProject
```

使用文件夹

下表显示了普通的文件夹任务以及完成这些任务的方法：

任务	方法
创建一个文件夹	FileSystemObject.CreateFolder
删除一个文件夹	Folder.Delete 或 FileSystemObject.DeleteFolder
移动一个文件夹	Folder.Move 或 FileSystemObject.MoveFolder

续表

复制一个文件夹

Folder.Copy 或

检索文件夹的名称

FileSystemObject.CopyFolder

查找一个文件夹是否在驱动器上

Folder.Name

获得已有 Folder 对象的一个实例

FileSystemObject.FolderExists

找出一个文件夹的父文件夹的名称

FileSystemObject.GetFolder

找出系统文件夹的路径

FileSystemObject.GetParentFolderName

FileSystemObject.GetSpecialFolder

示例

该示例演示如何使用 Folder 对象和 FileSystemObject 对象来操作文件夹并获得其信息：

```
Private Sub Command10_Click()
```

```
    ' 获得 FileSystemObject 的实例。
```

```
    Dim fso As New FileSystemObject, fldr As Folder, s As String
```

```
    ' 获取 Drive 对象。
```

```
    Set fldr = fso.GetFolder("c:")
```

```
    ' 打印父文件夹名。
```

```
    Debug.Print "Parent folder name is: " & fldr
```

```
    ' 打印驱动器名。
```

```
    Debug.Print "Contained on drive " & fldr.Drive
```



```
' 打印根文件名。
If fldr.IsRootFolder = True Then
    Debug.Print "This folder is a root folder."
Else
    Debug.Print "This folder isn't a root folder."
End If
' 用 FileSystemObject 对象创建一个新的文件夹。
fso.CreateFolder ("c:\Bogus")
Debug.Print "Created folder C:\Bogus"
' 打印文件夹的基本名称。
Debug.Print "Basename = " & fso.GetBaseName("c:\bogus")
' 删除新建的文件夹。
fso.DeleteFolder ("c:\Bogus")
Debug.Print "Deleted folder C:\Bogus"
End Sub
```

使用文件

通过使用新的面向对象的 FSO 对象，诸如 Copy、Delete、Move 以及 OpenAsTextStream，或者使用传统的函数，诸如 Open、Close、FileCopy、GetAttr 等等，可以使用 Visual Basic 中的文件。请注意，不用考虑其文件类型就可以移动、复制或删除文件。

关于传统函数用法的详细信息，请参阅本章的“用传统的文件 I/O 语句和函数处理文件”部分。本节的余下部分将描述如何使用新的 FSO 对象、方法和属性来使用文件。

文件操作主要有两类：

- 创建、添加、或移动数据，以及阅读文件
- 移动、复制、和删除文件

用 File System Objects 创建文件并添加数据

有三种不同方法来创建一个顺序文本文件（有时被称为“文本流”）。一种方法就是使用 CreateTextFile 方法。要创建一个空文本文件，可以：

```
Dim fso As New FileSystemObject, fil As File
```

```
Set fil = fso.CreateTextFile("c:\testfile.txt", True)
```

注意 FSO 对象模型尚不支持创建随机文件或二进制文件。要创建随机文件和二进制文件，请使用带 Random 或 Binary 标志的 Open 命令。关于如何操作随机文件和二进制文件的详细信息，请参阅本章的“使用随机文件访问”和“使用二进制文件访问”部分。

另一种方法是使用 FileSystemObject 对象的 OpenTextFile 方法，该对象的 ForWriting 标志设置为：

```
Dim fso As New FileSystemObject, ts As New TextStream
```

```
Set ts = fso.OpenTextFile("c:\test.txt", ForWriting)
```

或者使用带 ForWriting 标志设置的 OpenAsTextStream 方法：

```
Dim fso As New FileSystemObject, fil As File, ts As TextStream
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CreateTextFile ("test1.txt")
Set fil = fso.GetFile("test1.txt")
Set ts = fil.OpenAsTextStream(ForWriting)
```

添加数据到文件

文本文件一经创建，就可以分三步向其中加入数据：

1. 打开文本文件以备写入数据。
2. 写入数据。
3. 关闭文件。

要打开文件，可以使用下面两种方法中的任何一种：File 对象的 `OpenAsTextStream` 方法，或 `FileSystemObject` 对象的 `OpenTextFile` 方法。要向打开的文本文件中写入数据，可以使用 `TextStream` 对象的 `Write` 或 `WriteLine` 方法。它们之间的唯一差别是 `WriteLine` 在指定的字符串末尾添加换行符。

如果想要向文本文件中添加一个空行，请使用 `WriteBlankLines` 方法。

要关闭一个已打开的文件，请使用 `TextStream` 对象的 `Close` 方法。

下面这个例子示范了如何打开一个文件，使用上述三种写入方法向该文件中添加数据，然后关闭文件：

```
Sub Create_File()
    Dim fso, txtfile
```

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set txtfile = fso.CreateTextFile("c:\testfile.txt", True)
txtfile.Write ("这是一个测试例子。") ' 写入一行。
' 写入一行带有换行符的文本。
txtfile.WriteLine("Testing 1, 2, 3.")
' 向文件中写入三个换行符。
txtfile.WriteLine(3)
txtfile.Close
End Sub
```

使用 File System Objects 读取文件

要从一个文本文件中读取数据，请使用 `TextStream` 对象的 `Read`、`ReadLine` 或 `ReadAll` 方法：

任 务	方 法
从一个文件中读取指定数量的字符	<code>Read</code>
读取一整行（紧跟，但不包括，换行符）	<code>ReadLine</code>
读取一个文本文件的所有内容	<code>ReadAll</code>

如果使用 `Read` 或 `ReadLine` 方法并且想要跳过数据的某些部分，可以使用 `Skip` 或 `SkipLine` 方法。

这些读取方法产生的文本被存储在一个字符串中，而这个字符串可以在一个控件中显示，也可以被字符串操作符分解（诸如 `Left`、`Right` 和 `Mid`）、

合并等等。

注意 `vbNewLine` 常数包含一个或多个字符（取决于操作系统）使得光标移至下一行的开头（回车/换行）。请注意，有的字符串末尾可能有这样的非打印字符。

示例

```
Sub Read_Files()  
    Dim fso As New FileSystemObject, txtfile, _  
        fil1 As File, ts As TextStream  
    Set txtfile = fso.CreateTextFile("c:\testfile.txt", True)  
    MsgBox "正在写入文件"  
    ' 写入一行。  
    Set fil1 = fso.GetFile("c:\testfile.txt")  
    Set ts = fil1.OpenAsTextStream(ForWriting)  
    ts.Write "Hello World"  
    ts.Close  
    ' 读取文件的内容。  
    Set ts = fil1.OpenAsTextStream(ForReading)  
    s = ts.ReadLine  
    MsgBox s  
    ts.Close  
End Sub
```

移动、复制和删除文件

对文件的移动、复制和删除，FSO 对象模型都提供了两种方法：

任务	方法
移动一个文件	File.Move 或 FileSystemObject.MoveFile
复制一个文件	File.Copy 或 FileSystemObject.CopyFile
删除一个文件	File.Delete 或 FileSystemObject.DeleteFile

示例

本例先在 C 盘的根目录下创建了一个文本文件，并向其中写入一些信息，然后将该文件移至一个名叫 \tmp 的目录，并复制到一个叫 \temp 的目录下，最后删掉该文件在这两个子目录下的拷贝。

要运行本例，请先确认在 C 盘的根目录下已经存在名叫 \tmp 和 \temp 的目录。

```
Sub Manip_Files()  
    Dim fso as New FileSystemObject, txtfile, fil1, fil2  
    Set txtfile = fso.CreateTextFile("c:\testfile.txt", True)  
    MsgBox "写入文件"
```

' 写入一行。

txtfile.Write ("这是一个测试例子。")

' 关闭要写入的文件。

txtfile.Close

MsgBox "移动文件到 c:\tmp"

' 获得 C 盘根目录下的一个文件句柄。

Set fil1 = fso.GetFile("c:\testfile.txt")

' 将该文件移至 \tmp 目录下。

fil1.Move ("c:\tmp\testfile.txt")

MsgBox "复制文件到 c:\temp"

' 将该文件复制到 \temp 目录下。

fil1.Copy ("c:\temp\testfile.txt")

MsgBox "删除文件"

' 获得这些文件当前位置的句柄。

Set fil1 = fso.GetFile("c:\tmp\testfile.txt")

Set fil2 = fso.GetFile("c:\temp\testfile.txt")

' 删除这些文件。

fil1.Delete

fil2.Delete

MsgBox "全部完成！"

End Sub

用传统的文件 I/O 语句和函数处理文件

从 Visual Basic 的第一版直到如今，文件处理都是通过使用 Open 语句以及其它一些相关的语句和函数（在下面列出）来实现的。这些机制将逐步被淘汰而转向使用 FSO 对象模型，但是在 Visual Basic 6.0 中仍得到完全支持。

如果设计应用程序使用数据库文件，那么在应用程序中就不需提供直接的文件访问。Data 控件与绑定控件都可以向数据库写数据以及从数据库读数据，这比使用直接的文件访问技术容易得多。

然而，有时常会需要读写文件而不是数据库。这里给出的一组主题展示了如何处理文件，为了直接创建、操纵以及保存文本和其它数据。

文件访问类型

文件本身除了包括一系列定位在磁盘上的相关字节外，再也没有其它东西。当应用程序访问一个文件时，必须假定字节表示什么（字符、数据记录、整数、字符串等等）。

应根据文件包括什么类型的数据，使用合适的文件访问类型。在 Visual Basic 中，有三种文件访问的类型：

- 顺序型——适用于读写连续块中的文本文件。
- 随机型——适用于读写有固定长度记录结构的文本文件或者二进制文

件。

- 二进制型——适用于读写任意有结构的文件。

顺序的访问是为普通的文本文件的使用设计的。文件中每一个字符都被假设为代表一个文本字符或者文本格式序列，比如换行符 (NL)。数据被存储为 ANSI 字符。而为随机型访问打开的文件则认为是由相同长度的记录集合组成。可用用户定义的类型来创建由各种各样的字段组成的记录——每个字段可以有不同的数据类型。数据作为二进制信息存储。

二进制访问允许使用文件来存储所希望的数据。除了没有数据类型或者记录长度的含义以外，它与随机访问很相似。然而，为了能够正确地对它检索必须精确地知道数据是如何写到文件中的。

详细信息 有关文件访问类型的详细信息，请参阅本章中的“使用顺序文件访问”、“使用随机文件访问”和“使用二进制文件访问”。

文件访问函数和语句

以下函数用于所有三种类型的文件访问：

Dir	FileLen	LOF
EOF	FreeFile	Seek
FileCopy	GetAttr	SetAttr
FileDateTime	Loc	

下表列出可用于三种直接文件访问类型的各种文件访问的所有语句和函数。

语 句 与 功 能	顺 序 型	随 机 型	二 进 制 型
Close	X	X	X
Get		X	X
Input()	X		X
Input #	X		
Line Input #	X		
Open	X	X	X
Print #	X		
Put		X	X
Type...End Type		X	
Write #	X		

详细 信息 有关文件访问函数和语句的其它信息，请参阅索引中的函数或语句主题。

使用顺序文件访问

推荐使用 File System Objects 来创建文本文件，但该信息只在需要使用旧文本文件创建方法时才提供。

当要处理只包含文本的文件时，比如由典型文本编辑器所创建的文件——也就是说，其中的数据没有分成记录的文件，使用顺序型访问最好。顺序型访问不太适于存储很多数字，因为每个数字都要按字符串存储。一个四位数将需要 4 个字节的存储空间，而不是作为一个整数来存储时只需的 2

个字节。

打开顺序访问文件

当以顺序型访问打开一个文件时，可执行以下操作：

- 从文件输入字符 (Input)
- 向文件输出字符 (Output)
- 把字符加到文件 (Append)

要顺序型访问打开一个文件，Open 语句使用以下语法：

Open *pathname* **For** [Input | Output | Append] **As** *filenumber* [**Len** = *buffersize*]

当打开顺序文件作为 Input 时，该文件必须已经存在，否则，会产生一个错误。然而，当打开一个不存在的文件作为 Output 或 Append 时，Open 语句首先创建该文件，然后再打开它。

当在文件与程序之间拷贝数据时，选项 Len 参数指定缓冲区的字符数。

在打开一个文件 Input、Output 或 Append 以后，在为其它类型的操作重新打开它之前必须先使用 Close 语句关闭它。

编辑为顺序访问打开的文件

如果要编辑一个文件，先把它的内容读入到程序变量，然后改变这些变量，最后再把这些变量写回到该文件。以下各节讨论如何编辑以顺序型访问打开的各记录。

从文件中读取字符串

要检索文本文件的内容，应以顺序 `Input` 方式打开该文件。然后使用 `Line Input#`，`Input()`，或者 `Input#` 语句将文件拷贝到程序变量中。

`Visual Basic` 提供了一次读写顺序文件一个字符或一行的语句和功能。

例如，以下代码段逐行读取一个文件：

```
Dim LinesFromFile, NextLine As String
```

```
Do Until EOF(FileNum)
```

```
    Line Input #FileNum, NextLine
```

```
    LinesFromFile = LinesFromFile + NextLine + Chr(13) + Chr(10)
```

```
Loop
```

尽管 `Line Input #` 到达回车换行时它会识别行尾，但是，当它把该行读入变量时，不包括回车换行。如果要保留该回车换行，必须添加代码。

也可以使用 `Input #` 语句，它读取文件中所写的一列的数字和/或字符串表达式。例如，要从一个邮件列表文件中读取一行，可使用以下语句：

```
Input #FileNum, name, street, city, state, zip
```

可以使用 `Input` 函数来从文件向变量拷贝任意数量的字符，所给的变量大小应足够大。例如，以下的代码使用 `Input` 把指定数目的字符复制到变量。

```
LinesFromFile = Input(n, FileNum)
```

要将整个文件复制到变量，请使用 `InputB` 函数将字节从文件复制到变量。

因为 InputB 函数返回一个 ANSI 字符串，必须使用 StrConv 函数将 ANSI 字符串转换为一个如下的 UNICODE 字符串：

```
LinesFromFile = StrConv(InputB(LOF(FileNum), FileNum), vbUnicode)
```

把字符串写入文件

要在顺序文件中存储变量的内容，应以顺序 Output 或 Append 打开它，然后使用 Print # 语句。例如，文本编辑器可以使用以下代码行来把一个文本框的内容拷贝到文件。

```
Print #FileNum, TheBox.Text
```

Visual Basic 也支持 Write # 语句，它把一系列数字和/或字符串表达式写入文件。它自动地用逗号分开每个表达式，并且在字符串表达式端放置引号。

```
Dim AnyString As String, AnyNumber As Integer
```

```
AnyString = "AnyCharacters"
```

```
AnyNumber = 23445
```

```
Write #FileNum AnyString, AnyNumber
```

这个代码段把两个表达式写入由 FileNum 指定的文件。第一个表达式包含一个字符串而第二个包含数字 23445。因而，Visual Basic 把以下的字符（包括所有标点符号）写入到文件中：

```
"AnyCharacters",23445
```

注意 如果正在用顺序型访问 Write # 与 Input # 语句，可考虑换用随机型

或二进制型访问，因为它们更适合面向于记录的数据。

详细信息 有关顺序文件访问的其它信息，请参阅“Open 语句”。

使用随机文件访问

File System Object 模式不提供随机文件创建或访问方法。如果需要创建或读取随机文件，此信息将能帮助您。

随机型访问文件中的字节构成相同的一些记录，每个记录包含一个或多个字段。具有一个字段的记录对应于任一标准类型，比如整数或者定长字符串。具有多个字段的记录对应于用户定义类型。例如，下边所定义的 **Worker Type** 创建由三个字段组成的 19 个字节的记录。

```
Type Worker
```

```
LastNameAs String * 10
```

```
TitleAs String * 7
```

```
RankAs String * 2
```

```
End Type
```

变量声明

在应用程序打开以随机型访问的文件以前，应先声明所有用来处理该文件数据所需的变量。这包括用户定义类型的变量，它对应该文件中的记录，和标准类型的其它变量，这些变量保存为随机型访问而打开的文件与处理

相关的数据。

定义记录类型

在打开一个文件进行随机访问之前，应定义一个类型，该类型对应于该文件包含或将包含的记录。例如，一个雇员记录文件可定义一个称为 **Person** 的用户定义的数据类型，如下所述：

Type Person

ID	As Integer
MonthlySalary	As Currency
LastReviewDate	As Long
FirstName	As String * 15
LastName	As String * 15
Title	As String * 15
ReviewComments	As String * 150

End Type

在类型定义中声明字段变量

因为随机访问文件中的所有记录都必须有相同的长度，所以固定的长度对用户定义类型中的各字符串元素通常很有用，就像以上的 **Person** 类型说明中所示的一样，在此作为例子，**FirstName** 与 **LastName** 都具有 15 个字符的固定长度。

如果实际字符串包含的字符数比它写入的字符串元素的固定长度少，则 **Visual Basic** 会用空白（字符代码 32）来填充记录后面的空间。还有，如

果字符串比字段的尺寸长，则它就会被截断。如果使用 长度可变的字符串，则任何用 **Put** 存储的或用 **Get** 检索的记录总长度都不能超过 **Open** 语句的 **Len** 分句中所指定的记录长度。

声明其它变量

在定义与典型记录对应的类型以后，应接着声明程序需要的任何其它变量，用来处理作为随机访问而打开的文件。例如：

'记录变量。

```
Public Employee As Person
```

'跟踪当前记录。

```
Public Position As Long
```

'文件中最后一条记录的编号。

```
Public LastRecord As Long
```

打开随机访问的文件

要打开随机访问的文件，**Open** 语句使用以下语法：

```
Open pathname [For Random] As filename Len = reclength
```

因为 **Random** 是缺省的访问类型，所以 **For Random** 关键字是可选项。

表达式 **Len** = *reclength* 指定了每个记录的尺寸 in bytes. Note that every string variable in Visual Basic stores a Unicode string and that you must specify the byte length of that Unicode string。如果 *reclength* 比写文件记录的实际长度短，则会产生一个错误。如果 *reclength* 比记录的实际长度长，则记录可写入，

只是会浪费些磁盘空间。

可用以下代码打开文件：

```
Dim FileNum As Integer, RecLength As Long, Employee As Person
```

'计算每条记录的长度。

```
RecLength = LenB(Employee)
```

'取出下一个可用文件编号。

```
FileNum = FreeFile
```

'用 Open 语句打开新文件。

```
Open "MYFILE.FIL" For Random As FileNum Len = RecLength
```

编辑随机型访问打开的文件

如要编辑随机型访问的文件，请先把记录从文件读到程序变量，然后改变各变量的值，最后，把变量写回该文件。后面各节讨论如何编辑随机型访问打开的文件。

把记录读入变量

使用 Get 语句把记录复制到变量。例如，要把一个记录从雇员记录文件拷贝到 Employee 变量，可使用以下代码：

```
Get FileNum, Position, Employee
```

在这行代码中，FileNum 包含用于打开文件的 Open 语句的编号；Position

包含要拷贝的记录数；而 `Employee` 声明为用户定义类型 `Person`，它用来接收记录的内容。

把变量写入记录

使用 `Put` 语句把记录添加或者替换到随机型访问打开的文件。

替换记录

要替换记录，请使用 `Put` 语句，指定想要替换的记录位置，例如：

```
Put #FileNum, Position, Employee
```

这个代码将用 `Employee` 变量中的数据来替换由 `Position` 所指定的编号的记录。

添加记录

要向随机访问打开的文件的尾端添加新记录，应使用前述代码段中所示的 `Put` 语句。把 `Position` 变量的值设置为比文件中的记录数多 1。例如，要在一个包含五个记录的文件中添加一个记录，把 `Position` 设置为 6。

下述语句把一个记录添加到文件的末尾：

```
LastRecord = LastRecord + 1
```

```
Put #FileNum, LastRecord, Employee
```

删除记录

通过清除其字段可以删除一个记录，但是该记录仍在文件中存在。通常文件中不能有空记录，因为它们会浪费空间且会干扰顺序操作。最好把余下的记录拷贝到一个新文件，然后删除旧文件。

要清除随机访问文件中删除的记录，请按照以下步骤执行：

1. 创建一个新文件。
2. 把所有有用的记录从原文件复制到新文件。
3. 关闭原文件并用 `Kill` 语句删除它。
4. 使用 `Name` 语句把新文件以原文件的名字重新命名。

详细信息 有关随机文件访问的其它信息，请参阅“`Open` 语句”。

使用二进制文件访问

`File System Object` 模式不提供二进制文件创建或访问方法。如果需要创建或读取二进制文件，此信息将会帮助您做到。

二进制访问能提供对文件的完全控制，因为文件中的字节可以代表任何东西。例如，通过创建长度可变的记录可保存磁盘空间。当要保持文件的尺寸尽量小时，应使用二进制型访问。

注意 当把二进制数据写入文件中时，使用变量是 `Byte` 数据类型的数组，而不是 `String` 变量。`String` 被认为包含的是字符，而二进制型数据可能无法正确地存在 `String` 变量中。

打开二进制型访问的文件

要为二进制型访问打开文件，应使用以下 `Open` 语句的语法：

`Open` *pathname* **For Binary **As** *filenumber***

可以看到，二进制访问中的 `Open` 与随机存取的 `Open` 不同，它没有指定 `Len = reclength`。如果在二进制型访问的 `Open` 语句中包括了记录长度，则被忽略。

在长度可变的字段中保存信息

为了更好地了解二进制型访问，考虑有一个雇员记录文件。这个文件用长度固定的记录和字段来存储雇员的信息。

```
Type Person
```

ID	As Integer
MonthlySalary	As Currency
LastReviewDate	As Long
FirstName	As String * 15
LastName	As String * 15
Title	As String * 15
ReviewComments	As String * 150

```
End Type
```

不管字段的实际内容如何，该文件每条记录都占用 209 个字节。通过使用二进制型访问可使磁盘空间的使用降到最小。因为这不需要固定长度的字段，类型声明语句可以省略字符串长度参数。

```
Type Person
```

ID	As Integer

MonthlySalary	As Currency
LastReviewDate	As Long
FirstName	As String
LastName	As String
Title	As String
ReviewComments	As String

End Type

Public Empl As Person '定义记录。

因为各字段长度是可变的，所以雇员记录文件中每个雇员记录只存储在精确所需的字节中。用长度可变字段来进行二进制输入/输出的缺点，是不能随机地访问记录——必须顺序地访问记录以了解每一个记录的长度。可以直接查看文件中指定字节的位置，但是，如果记录的长度可变，则无法直接知道哪条记录在哪个字节处。

详细信息 有关二进制文件访问的其它信息，请参阅“Open 语句”。

第十五章 性能和兼容性设计

在理想世界里，用户的计算机拥有最快的处理器、大量的内存、无限的外存空间以及高速网络连接。而实际上对大多数用户而言，应用程序的性能将受到上述诸多因素的制约。特别当创建复杂的大型应用程序时，其运行时需要的内存和速度倍受关注。所以，有必要缩小应用程序的大小、提高计算和显示的速度，进而优化所创建的应用程序。

在设计和编写应用程序时，可以采用不同的方法来优化应用程序的性能。有的能加快应用程序的运行，有的能缩小应用程序的大小。本章将介绍应用程序优化的常用方法。

Visual Basic 和 VBA 有很多共性，其中 VBA 包含在 Microsoft Office 和其它一些应用程序中。VBS (VBScript) 是一种 Internet 的描述语言，也是 Visual Basic 语言的子集。如果您也是一个 VBA 或 VBS 的开发人员，或许希望在这些语言间共享代码。

本章将讨论 Visual Basic 语言三种不同版本间的区别，并给出创建可移植代码的提示。

目 录

- 优 化
- 优 化 速 度
- 优 化 大 小
- 优 化 对 象
- 编 译 型 和 解 释 型 应 用 程 序 的 比 较
- 与 其 它 Microsoft 应 用 程 序 的 兼 容 性

示 例 应 用 程 序：Optimize.vbp

列在目录中的示例应用程序 Optimize.vbp 列举了本章将要介绍的许多优化技术。

优 化

优化具有科学和艺术两重含义。从科学角度看，优化意指具体的优化技术；而从艺术角度看，优化指的是确定在什么地方、什么时候需要优化。由此，优化可定义为“通过选择和设计数据结构、算法以及指令序列，来提高程序效率（更小、更快）的过程”。

有关优化的一个普遍存在的误解是，认为只是在应用程序开发周期的最后

阶段才进行优化。而实际上，为了创建真正优化的应用程序，就必须在开发时实行优化。一般来说，优化的过程为：仔细选择算法，并在速度、大小等诸多限制因素间进行权衡，初步估计应用程序各个部分的速度和大小，再在以后的开发过程中检验上述假设。

优化的第一步是确定优化目标。优化可以从以下几个方面进行：

- 真实速度（应用程序实际计算或操作的速度）。
- 显示速度（应用程序屏幕显示的速度）。
- 感觉速度（应用程序运行时给人的感觉速度，它往往和显示速度有关，但并不总是和真实速度相关）。
- 占用内存的大小。
- 图形大小（这直接影响占用内存的大小，而且工作在 Microsoft Windows 中时往往会产生其它影响）。

一般来说，不可能在几个方面同时得到优化。一个经过大小优化的应用程序往往会降低其速度；相应的，经过速度优化的应用程序则会增加其大小。由此可见，实现不同目标的优化技术往往是相抵触的。

值得注意的是，优化并不总是完全有益的。加快或降低应用程序的速度，可能导致维护或调试方面的困难；还有些优化技术与结构化的程序设计相矛盾，这在将来扩充应用程序的功能或把它嵌入其它程序时会产生麻烦。

在确定应用程序优化策略时，有三方面的问题值得考虑：优化什么、在何处优化及何时结束优化。

优化什么：理解实际问题

如果没有明确的优化目标，就会因为方向错误而浪费大量的时间。优化的目标是为了满足用户的需求。例如，速度对于计算销售税的应用程序来说就至关重要，而对于可以从 Internet 上下载的应用程序，则其大小倍受关注。所以，正确理解优化所要解决的问题所在，是确定优化策略的关键。即使已经确定了某一优化目标，仍需在开发过程中全面考虑优化。在编写代码时，一步一步地浏览代码，仔细思考其实际发生的情况，则会帮助了解很多东西。例如，设置属性会产生许多事件，而恰好这些事件过程中有大量的代码，则无谓的设置属性语句会导致程序执行时的巨大延迟。有时优化即便是针对大小的，仍然可以在不增加大小的前提下实现速度的优化。

在何处优化：事半功倍

绝大部分的开发不可能对应用程序的所有地方都进行优化。增加时间等于增加开发成本，所以有必要进行“优化预算”。哪些地方能够花一些时间换取最大的投资回报呢？显然，那些速度慢、代码冗长的地方最需要优化，把精力花在这样的地方就会有事半功倍的效果。

例如，速度是主要目标，则循环体往往是开始优化的好地方。一旦循环体内的操作得到加快，则该优化会被放大，倍数是循环体执行的次数。对于有大量重复的循环，减少循环体内的一个字符串操作，结果就会有很大不同。同样的规律也可以应用于经常调用的子程序。

何时结束优化：效果的衡量

有时一些东西并不值得优化。例如，为了十几个项目的排序而编写一个精致的快速排序程序是没有意义的。一种排序的方法是把项目加到排序列表框中，再把它们安排好的顺序读出来。如果项目的数量巨大，则该方法效率极低；但对于少量的项目，该方法与其它方法的效率一样，而且代码出奇地简单（只是有点难解）。

在一些情况下，优化纯属浪费。如果应用程序的运行受到驱动器或网络速度的限制，则对代码的任何优化都无助于速度的提高。此时，就应设法减少因延迟而造成对用户的影响：如利用进度栏提示当前应用程序的运行状况，或利用高速缓存减小延迟，或放弃控制，这样用户在等待时可以运行其它的应用程序，等等。

详细信息 请参阅第十一章“响应鼠标和键盘事件”中的“中断后台处理”。

优化速度

在用户评测应用程序是否满意的诸多因素中，速度是决定性的因素。不幸的是，很多影响应用程序速度的因素不是编程者力所能及的，诸如微处理器的速度、内存空间的不足或数据连接的速度等等。所以，有必要优化创建的应用程序，加快其速度（至少看起来运行得快了）。

速度的优化有三种：真实速度（计算和执行代码的实际时间）、显示速度（图

形显示或屏幕显示的时间) 和感觉速度(应用程序运行时的感觉速度)。因为优化并不是在所有的情况下都是有益的, 所以选择优化的具体类型有赖于应用程序的性质及其目的。

无论是哪一种优化, 都需要权衡收益和代价。花数小时去优化一个很少调用的程序是没有意义的。所以要选择最受用户关注的地方进行优化, 例如应用程序的初始加载时间。

优化代码

除非应用程序是用来产生分形图形的, 否则应用程序看上去并不受限于代码的实际处理速度。其它典型的因素, 如显示速度、网络延迟或磁盘操作, 才是应用程序速度的限制因素。比如, 导致窗体加载慢的原因, 是窗体上控件和图形的个数太多, 而不是由于 `Form_Load` 事件中的代码太慢。尽管如此, 代码本身的速度也可以成为限制应用程序速度的瓶颈, 特别是经常调用到的例程。这种情况下, 可采用以下的几种技术来优化应用程序的真实速度。

- 避免使用 `Variant` 变量。
- 使用 `Long integer` 变量和整数运算。
- 将常用的属性缓存在变量中。
- 使用内嵌过程替代过程调用。
- 尽可能使用常数。
- 用 `ByVal` 传递参数, 而不用 `ByRef`。

- 使用类型确定的可选参数。
- 利用集合的优点。

即使您并不想优化代码的速度，这也有助于了解这些技术及其基本原则。而且，一旦养成选择高效率算法作为代码的习惯，就可以从总体上大大改善应用程序的速度。

避免使用 Variant 变量

Variant 变量是 Visual Basic 中的缺省变量。这对于初学者以及处理速度不成问题的应用程序来说，是非常方便的。然而，如果想优化应用程序的实际速度，就要避免使用 Variant 变量。因为，运行时 Variant 将转化为其它适当的数据类型，那么直接采用其它简单的数据类型，就会避免不必要的操作而加快应用程序的速度。

避免使用 Variant 变量的一种好办法是使用“Option Explicit”语句，此时所有的变量都必须声明。要使用“Option Explicit”，就要在“工具”菜单中启动“选项”对话框，选择“编辑器”选项卡，选定“要求变量声明”复选框。

在声明多个变量时要小心：如果没有用 As type 子句，它们实际上被声明为 Variant 变量。在下面的例子中，X 和 Y 是 Variant 变量：

```
Dim X, Y, Z As Long
```

重写上面的语句，则三个变量为 Long 变量：

```
Dim X As Long, Y As Long, Z As Long
```

详细信息 关于 Visual Basic 数据类型的详细信息，请参阅第五章“编程基础”中的“数据类型”。

使用 Long 整型变量和整数运算

算术运算中要避免使用 Currency、Single 和 Double 变量；并尽量使用 Long 整型变量，尤其在循环体中。因为 Long 整数是 32 位 CPU 的本机数据类型，所以其操作非常快；如果无法使用 Long 变量，就要尽量使用 Integer 或 Byte 数据类型。很多时候，即使在要求使用浮点数的情况下，也可以使用 Long 整数。例如，在窗体和图形控件的 ScaleMode 属性设置为缇或像素时，就可以在控件和图形方法中使用 Long 整型变量表示大小和位置。进行除法运算时，如果不需要小数部分，就可以使用整数除法运算符 (/)。由于浮点运算需要转移到协处理器上进行，而整数运算并不需要，所以整数运算总是比浮点运算快。如果确实需要做小数运算，则 Double 数据类型比 Currency 数据类型快。

下表把各种数值数据类型按运算速度顺序列出。

数值数据类型	速度
Long	最快
Integer	
Byte	
Single	

续表

Double

Currency

最慢

将常用的属性缓存在变量中

变量的访问和设置速度比属性快。如果经常用到某一属性的值（如在循环体中），可以在循环体外把该属性值赋给某一变量，以后用该变量替代该属性，这样就能够提高代码的速度。一般来说，变量的处理速度比同类型的属性处理速度快 10 到 20 倍。

除非知道属性已经改变，否则在过程中就无需再次读取该属性值。可以把属性值赋予某一变量，然后在以后的代码中使用该变量。例如，象这样的代码就非常慢：

```
For i = 0 To 10
    picIcon(i).Left = picPallette.Left
Next I
```

下面改写的代码就要快得多：

```
picLeft = picPallette.Left
For i = 0 To 10
    picIcon(i).Left = picLeft
Next I
```

同样地，像这样的代码 ...

```
Do Until EOF(F)
    Line Input #F, nextLine
    Text1.Text = Text1.Text + nextLine
Loop
```

... 比下面的代码慢得多：

```
Do Until EOF(F)
    Line Input #F, nextLine
    bufferVar = bufferVar& + nextLine&vbCrLf
Loop
Text1.Text = bufferVar
```

然而，下面的代码完成了相同的功能，而且还要快：

```
Text1.Text = Input(F, LOF(F))
```

如上述，几种方法都实现了同样的任务；同时，最好的算法也是最优的。同样的技术可用于处理函数的返回值。缓存函数的返回值，避免经常调用运行时的动态链接库 (DLL)，Msvbvm60.dll。

使用模块级变量代替静态变量

当声明静态变量用于存储多个过程执行引用的值时，它们比局部变量慢。如果在模块级变量中存储同样的值，执行会比较快。当然，必须确定是一个过程改变模块级变量，缺点是代码可读性较差，较难维护。

使用内嵌过程替代过程调用

采用过程调用使代码更具有模块化的风格，但模块调用总是增加额外的操作和处理时间。如果循环体中多次调用某一过程，就可以直接把该过程写到循环体中去，以消除过程调用时的额外负担。但另一方面，直接把某一过程写到好几个循环体中时，重复的代码无疑要增加应用程序的大小；同时，在更改程序时，有可能忘记更改这些重复的代码，这就增加了出错的机会。

同样，调用同一模块中的过程要比在独立的 `.BAS` 模块中调用同一模块快，如果要从不同模块调用同一个过程，情况就不同了。

尽可能使用常数

使用常数可以加快应用程序的运行，增强代码的可读性，而且易于维护。如果代码中的字符串或数字是不变的，则可把它们声明为常数。常数在编译时只处理一次，将适当的值写进代码；而变量在每次运行应用程序时都要读取当前值。

尽量使用对象浏览器中列举的内部常数，而不要自己去创建。不要担心应用程序中引用的模块包含多余的常数；多余的常数在形成 `.exe` 文件时被删除。

用 ByVal 传递参数，而不用 ByRef

编写含参数的 Sub 或 Function 过程时，按值 (ByVal) 传递参数比按地址 (ByRef) 快。尽管 Visual Basic 中参数传递的缺省方式是按地址的 (ByRef)，但实际上需要改变参数值的过程极少。如果过程中不需改变参数的值，就可以按值 (ByVal) 来传递，举例说明如下：

```
Private Sub DoSomething(ByVal strName As String, _  
    ByVal intAge As Integer)
```

使用类型确定的可选参数

使用 Visual Basic 5.0 中类型确定的可选参数，可以提高 Sub 或 Function 的调用速度。Visual Basic 以前版本中的可选参数只能是 Variant 的。如果过程是按值传递参数的，正如下面的例子，16 个字节的 Variant 变量保存在堆栈中。

```
Private Sub DoSomething(ByVal strName As String, _  
    Optional ByVal vntAge As Variant, _  
    Optional ByVal vntWeight As Variant)
```

使用类型确定的可选参数，每次调用时占用的堆栈空间较少，而且传递到内存中的数据也较少：

```
Private Sub DoSomething(ByVal strName As String, _  
    Optional ByVal intAge As Integer, _
```

Optional ByVal Integer As Integer)

类型确定的可选参数的访问速度比 `Variant` 快，而且一旦数据类型错误，编译时就显示错误信息。

利用集合的优点

可以定义和使用对象的集合是 `Visual Basic` 的强大功能之一。尽管集合是非常有用的，但还要正确使用才能获得最好的效果：

- 使用 `For Each...Next` 替代 `For...Next`。
- 添加集合的对象时避免使用 `Before` 和 `After` 参数。
- 使用键集而不用几组相同对象的数组。

集合可以用 `For...Next` 循环进行迭代。但采用 `For Each...Next` 可读性更好，而且多数情况下更快。`For Each...Next` 是由集合的生成器实现迭代的，所以实际的操作速度将随集合对象的不同而改变。由于 `For Each...Next` 的最简单的实现机理就是 `For...Next` 的线性迭代，因此 `For Each...Next` 不会比 `For...Next` 慢。但是，有些情况下采用了比线性迭代更复杂的实现机理，所以 `For Each...Next` 要快得多。

如果没有使用 `Before` 和 `After` 参数，则往集合中添加对象是非常快的。否则，`Visual Basic` 必须在集合中检测到其它对象后，才能添加新对象。

如果对象的类型都一样，集合或数组都可以用来管理这些对象（如果对象的类型不一样，则只能用集合）。从速度的观点看，选择何种方式取决于对象的访问方式。如果能够为每一对象分配唯一的键，则集合是访问对象的

最快方式。使用键从集合中检索对象比从数组中顺序遍历对象快。当然，如果没有键而要遍历对象时，则选择数组比较好。就顺序遍历方式而言，数组比集合快。

如果对象的个数少，则数组使用的内存小，并且搜索的速度快。当对象的个数在 100 左右时，集合比数组的效率高；当然，具体的数目还有赖于微处理器的速度和可用的内存。

详细信息 请参阅第八章“再论编程”中的“用集合替代数组”。

性能评测

针对某一具体情况所选的算法是否最优，有时并不好判定。为了验证假设，可以象下面的例子那样，创建一个简单的应用程序来评测其性能。示例应用程序 `Optimize.vbp` 包含了几个不同的测试方案。

创建性能测试应用程序

1. 打开一个新的 `.exe` 工程。
2. 创建一个窗体，该窗体有两个命令按钮：`Command1` 和 `Command2`。
3. 在 `Command1_Click` 事件中添加下列代码：

```
Private Sub Command1_Click()  
    Dim dblStart As Double  
    Dim dblEnd As Double  
    Dim i as Long
```

```

dblStart = Timer          '读取开始时间。

For i = 0 To 9999
    Routine to test      '插入测试例程。
Next

dblEnd = Timer            '读取结束时间

Debug.Print dblEnd - dblStart    '显示
                                '运行时。

End Sub

```

4. 在 `Command2_Click` 事件中添加相同的代码，把要测试的程序替换成第二个版本。

5. 运行这个应用程序，并在立即窗口中监视运行结果。

这个示例使用了 `Visual Basic` 的 `Timer` 类的缺省属性，计算循环体中代码的运行时间。针对每个命令按钮，在循环体中替换不同的程序，可以很快地比较出不同算法的性能。测试的代码可以直接写在循环体内，也可以是对其它过程的调用。

也许需要试验不同的循环上限值，尤其是评测运行快的程序。由于每次运行的结果可能不同，所以每一版本都要运行几次，然后求得其平均速度。

优化显示速度

由于 Microsoft Windows 的图形特性，图形和其它操作的显示速度在很大程度上决定了应用程序的感觉速度。窗体出现及画图的速度越快，应用程序就会显得越快。以下的几种技术可用于提高应用程序的显示速度：

- 将容器的 ClipControls 属性设置为 False。
- 恰当地使用 AutoRedraw。
- 使用 Image 控件替代 PictureBox 控件。
- 设置属性时隐藏控件以避免多次重画。
- 使用 Line 替代 PSet。

将容器的 ClipControls 属性设置为 False

除非正在使用图形方法（Line，PSet，Circle 和 Print），否则将窗体、框架及 PictureBox 控件的 ClipControls 设置为 False（如果代码包含了在其它控件之后绘图的图形方法，则这可能导致不可预测的结果）。当 ClipControls 设置为 False 时，在重画控件本身之前，Visual Basic 不会用背景覆盖控件。这在窗体包含大量控件时，会大大提高显示速度。

详细信息 请参阅第十二章“使用文本和图形”中的“使用 AutoRedraw 和 ClipControls 使图形分层”。

恰当地使用 AutoRedraw

当窗体或控件的 AutoRedraw 设置为 True 时，Visual Basic 会利用位图重画该窗体或控件。这种方法虽然提高了简单情况的重画速度（例如，在删

除覆盖在窗体或控件上的窗口后，窗体或控件重新显示)，但会降低图形方法的速度。此时，Visual Basic 就会在 AutoRedraw 位图上进行图形方法操作，再把整个位图复制到屏幕上。这个过程也占用了相当数量的内存。如果应用程序产生的图形复杂但是不常改变，AutoRedraw 设置为 True 较为合适。如果图形需要经常改变，则 AutoRedraw 设置为 False 的效果更好，并且在 Paint 事件中进行窗体或控件的图形方法操作。

详细信息 请参阅第十二章“使用文本和图形”中的“使用 AutoRedraw 和 ClipControls 使图形分层”。

用 Image 控件替代 PictureBox 控件

优化能够提高应用程序的速度，并减小其大小，所以要尽量应用优化技术。如果仅简单显示图片，并只对单击事件和鼠标操作作出响应，应使用 Image 控件替代 PictureBox。除非需要图片框提供的特殊功能时，如图形方法、包含其它控件的能力或动态数据交换 (DDE)，否则不要使用图片框。

设置属性时隐藏控件以避免多次重画

重画的代价是昂贵的。重画的操作越少，应用程序就显得越快。减少重画次数的一种方法，就是操作控件时使其不可见。例如，假设在窗体的 Resize 事件中调整数个列表框的大小：

```
Sub Form_Resize ()
```

```
Dim i As Integer, sHeight As Integer
```

```

    sHeight = ScaleHeight / 4
    For i = 0 To 3
        lstDisplay(i).Move 0, i * sHeight, _
            ScaleWidth, sHeight
    Next
End Sub

```

该示例产生四次独立的重画，每个列表框一次。把所有的列表框放在图片框中，并在移动或调整列表框大小之前隐藏图片框，就会减少重画的次数。当再次使图片框可见时，所有的列表框一次画出：

```

Sub Form_Resize ()
    Dim i As Integer, sHeight As Integer
    picContainer.Visible = False
    picContainer.Move 0, 0, ScaleWidth, ScaleHeight
    sHeight = ScaleHeight / 4
    For i = 0 To 3
        lstDisplay(i).Move 0, i * sHeight, _
            ScaleWidth, sHeight
    Next
    picContainer.Visible = True
End Sub

```

值得注意的是，该示例中使用了 **Move** 方法替代设置 **Top** 和 **Left** 属性。**Move** 方法只需一次操作就设置了这两个属性，所以节省了多余的重画。

使用 Line 替代 PSet

使用 Line 比使用一系列的 PSet 方法快。避免使用 Pset 方法，把一些点组成一条线由 Line 方法一次画出。一般来说，简单、不常改变的图形元素采用形状和直线控件来处理比较合适；而复杂的或经常改变的图形则最好采用图形方法处理。

优化感觉速度

通常，应用程序的感觉速度和代码的实际执行速度并无多大关系。对用户来说，启动快、绘画快并提供不间断的反馈信息的应用程序显得速度快；而在完成任务时似乎“悬挂”起来的应用程序则显得速度慢。许多技术都可以使应用程序显得速度快：

- 隐藏窗体而不是卸载。
- 预加载数据。
- 使定时器在后台工作。
- 使用进度指示器。
- 加快应用程序的启动速度。

隐藏窗体而不是卸载

把窗体隐藏起来而不是把它们卸载，是一项 Visual Basic 1.0 以来就采用的

技巧，并至今有效。这项技术的负效应是加载窗体时要占用一定的内存。但如果能够提供足够的内存，而且窗体能够快速出现被认为是最重要的，此时则可以采用这项技术。

预加载数据

预先读取数据也可以改善应用程序的感觉速度。例如，需要从磁盘上加载几个文件的第一个时，为什么不一次加载尽可能多的文件呢？除非文件特别小，否则用户还是能够感觉到延迟。加载额外的文件增加的时间也许并不会被觉察到，而且以后不会再次延迟了。

使定时器在后台工作

有些应用程序中，在等待用户响应时可以完成一些重要的工作。这可通过定时器控件来实现。使用静态（或模块级）变量来跟踪运行进程，每当定时器空闲时就进行一小部分工作。如果保证每次定时器事件完成的工作很少，用户就不会觉察应用程序的应答有什么影响。此时，可以预先读取数据或完成其它事情，这将进一步提高应用程序的速度。

详细信息 关于定时器控件的详细信息，请参阅第七章“使用 Visual Basic 的标准控件”中的“使用 Timer 控件”；关于后台处理的讨论，请参阅第十一章“响应鼠标和键盘事件”中的“中断后台处理”。

使用进度指示器

如果程序中存在不可避免的长时间延迟，则必须给用户以提示，说明应用程序并没有悬挂起来。Windows 95 使用了标准进度栏进行提示。可以使用 Microsoft Windows 公共控件中的 ProgressBar 控件，这些公共控件包含在 Visual Basic 的专业版和企业版中。在关键时刻使用 DoEvents，特别是每次更新 ProgressBar 的值时，这样在用户做其它事时让应用程序重画。至少，应设置窗体的 MousePointer 属性为 vbHourglass (11)，这样就会显示等待光标来表明延迟。

加快应用程序的启动

当应用程序启动时，其感觉速度非常重要。用户对应用程序速度的第一印象，是在“启动”菜单中单击程序名后看到的变化。由于 VBA 及 ActiveX 控件等需加载各种运行时所需的 DLL，所以一些延迟对应用程序来说是不可避免的。即使这样，还是可以尽快给用户以响应：

- 在 Form_Load 事件中使用 Show。
- 简化启动窗体。
- 不要加载不需要的模块。
- 启动时运行一个小的 Visual Basic 应用程序预加载运行时 DLL。

在 Form_Load 事件中使用 Show

首次加载窗体时，显示该窗体前首先执行 `Form_Load` 事件中的代码。在 `Form_Load` 代码中使用 `Show`，可以改变这种模式，并在执行事件中的其它代码时给用户一些可见的东西。在使用 `Show` 后，用 `DoEvents` 确保该窗体已经被画出。

```
Sub Form_Load()  
    Me.Show           '显示启动窗体。  
    DoEvents          '确保启动窗体已经被画出。  
    Load MainForm    '加载主应用程序窗体。  
    Unload Me         '卸载启动窗体。  
    MainForm.Show     '显示主窗体。  
End Sub
```

简化启动窗体

窗体越复杂，其加载的时间就越长，所以要使启动窗体尽量简单。正如大部分 `Microsoft Windows` 的应用程序启动时都显示一份简洁的版权屏幕（也称之为快速显示），您的应用程序也可以这么做。启动窗体上的控件越少，包含的代码越少，则其加载和出现的速度就越快。即使它立即加载另一个更复杂的窗体，用户已知道应用程序已经启动。

对于大的应用程序，也许要在启动时预加载一些最常用的窗体，这样需要时能够及时显示这些窗体。实现这一目的的满意的方法是在启动窗体上显示进度栏，并在加载每个其它窗体时予以更新。每次加载窗体后都调用 `DoEvents`，保证启动窗体重画。一旦所有重要的窗体加载完毕，启动窗体

就显示第一个加载的窗体并自行卸载。当然，每个预加载的窗体都会运行 `Form_Load` 事件中的代码，所以要小心不要引起其它问题或过度的延迟。

不要加载不需要的模块

Visual Basic 根据要求加载代码模块，而不是在启动时立即加载所有的模块。这意味着不调用模块中的过程，模块就不会被加载。相反地，如果启动窗体调用了数个模块中的过程，则应用程序启动时加载所有这些模块，这就会降低速度。所以应避免从启动窗体中调用其它模块的过程。

启动时运行一个小的 **Visual Basic** 应用程序预加载运行时 **DLLs**

启动 Visual Basic 的应用程序时，大部分时间用于加载各种运行时 DLL，这些 DLL 将用于 Visual Basic、ActiveX 及 ActiveX 控件。当然，一旦这些已经加载，则无需花费这些时间。所以当使用部分或全部这些 DLL 的其它应用程序已经运行时，则您的应用程序在启动时就显得快多了。

一种能够大大改善应用程序启动性能的方法，就是提供另一个经常使用的简单实用的应用程序。例如，编写一个日历程序，安装在 Windows 的启动组。这样，每次系统启动时它将自动加载，不仅本身非常有用，同时也保证加载了各种 Visual Basic 运行时 DLL。

最后，对于 Visual Basic 专业版和企业版，可以把应用程序划分成一个核心应用程序和数个可执行部件或 DLL。较小的核心应用程序加载时就较快，而且需要时它会加载其它部分。

优化大小

过去，在设计应用程序时，经常得受到可用内存和系统资源的限制。在 32 位操作系统下，如 Windows 95 和 Windows NT，大多数 Visual Basic 程序员则很少考虑这些因素。可是，许多情况下，缩小应用程序大小仍然很重要。

有些应用程序需要从 Internet 网卸载或通过电子邮件的附件形式传输，它们的大小则显得十分重要。有些连接上数据传输得不够快，传输 1 兆字节的文件要花 1 小时或更多的时间。另外，许多应用程序除了 .exe 文件外，还要有 .dll 或 .ocx 文件，这就进一步增加了卸载的大小（和时间）。在这些情况下，就要设法优化磁盘中应用程序的大小。

即使用户不想卸载应用程序，但编制尽可能紧凑的程序仍是一个好主意。小的应用程序加载快，而且，由于占用内存少，还可同时运行其它程序。通过优化应用程序所占的内存可提高程序性能。

减小代码大小

当减小应用程序的大小非常重要时，有许多技术可用于编制更紧凑的代码。除了缩小应用程序所占内存的大小外，大部分优化方法还缩小了 .exe 文件的大小。作为一种附属的优点，较小的应用程序加载得更快。

多数大小优化技术都包括从代码中删除不必要的元素。在编译应用程序时，

Visual Basic 自动删除某些元素。而以下元素的长度或数量是无须限制的：

- 标识符名称
- 注释
- 空行

当应用程序作为一个 .exe 文件运行时，以上这些元素都不会影响应用程序所占内存的大小。

其它元素，如变量、窗体和过程，确实要占据内存的一部分空间。最好将它们精简以使效率更高。当应用程序作为一个 .EXE 文件运行时，有几项技术可用于缩小应用程序所需内存。以下技术可缩小代码大小：

- 减少加载窗体的数目。
- 减少控件数目。
- 用标签代替文本框。
- 保持数据在磁盘文件或资源中，并且只在需要时才加载。
- 组织模块。
- 考虑替换 Variant 数据类型。
- 使用动态数组，并在删除时回收内存。
- 回收被字符串或对象变量用过的空间。
- 消除死代码和无用的变量。

减少加载窗体数目

每一个加载的窗体，无论可视与否，都要占据一定数量的内存（其数量随

窗体上控件的类型和数量，以及窗体上位图的大小等的不同而变化)。只在需要显示时才加载窗体，不再需要时，卸载窗体（而不是隐藏窗体）。记住，任何对窗体的属性、方法或控件的引用，或对用 `New` 声明的窗体变量的引用，都会导致 `Visual Basic` 加载该窗体。

当使用 `Unload` 方法卸载窗体时，只能释放部分窗体所占空间。要释放所有空间，可用关键字 `Nothing` 使窗体的引用无效：

```
Set Form = Nothing
```

减少控件数目

当设计应用程序时，窗体应尽量少用控件。实际的限制取决于控件的类型和系统，但实际上，含有大量控件的窗体将运行缓慢。一项与之相关的技术是：设计时，尽可能地使用控件数组，而不是在窗体上放置大量同类型的控件。

详细信息 关于控件数组的详细信息，请参阅第七章“使用 `Visual Basic` 的标准控件”中的“使用控件数组”。

用标签代替文本框

标签控件占用的 `Windows` 资源比文本框少，因此，在可能的情况下，应使用标签代替文本框。例如，当窗体上需要一个隐藏的控件保存文本时，使用标签更有效。

甚至需要大量文本域的数据输入窗体都可以用这种技术优化。可以针对每

一个域创建一个标签，并使用单文本框输入，在 `Lost Focus` 事件中移动到下一个标签的位置：

```
Private Sub Label1_LostFocus()  
    'Update Label1  
    Label1.Caption=Text1.Text  
    'Move the Textbox over the next Label  
    Text1.Move Label2.Left,Label2.Top  
    'Update Text1 contents  
    Text1.Text1 contents  
    Text1.Text=Label2.Caption  
End Sub
```

通过适当设置 `BackColor` 和 `BorderStyle` 属性，可以使标签看起来像文本框。虽然这一技术需要更多的代码，但对于包含大量文本域的窗体，它可以大大地减少资源的使用。

保持数据在磁盘文件或资源中，并且只在需要时才加载

在设计时，直接放入应用程序的数据（象属性或代码中的文字字符串和数值）将增加运行时应用程序占用的内存。运行时从磁盘文件或资源中加载数据可减少占用内存。这对大位图和字符串特别有价值。

详细信息 关于向应用程序添加资源的详细信息，请参阅第八章“再论编程”中“利用资源文件进行工作”。

组织模块

Visual Basic 只在需要时才加载模块-即当代码调用模块中的一个过程时，模块才被加载到内存。如果从未调用一特定模块中的过程，Visual Basic 决不加载该模块。因此，尽量把相关的过程放在同一模块中，让 Visual Basic 只在需要时才加载模块。

考虑替换 Variant 数据类型

Variant 数据类型使用极其灵活，但是比其它数据类型所占内存大。当要压缩应用程序多余的空间时，应考虑用其它数据类型替代 Variant 变量，特别是替代 Variant 变量数组。

每一个 Variant 占用 16 个字节，而 Integer 占 2 个字节，Double 占 8 个字节。变长字符串变量占用 4 个字节加上字符串中每一个字符占用 1 个字节，但是，每一个包含字符串的 Variant 都要占用 16 个字节加上字符串中每一个字符占用 1 个字节。因为它们太大，因此在用作局部变量或过程的参数时，Variant 变量是特别烦人的，这是因为它们消耗堆栈空间太快。但在有些情况下，使用其它数据类型替代 Variant，灵活性降低了，为弥补损失的灵活性，不得不增加更多的代码。结果是大小没有真正的减小。

使用动态数组，并在删除时回收内存

使用动态数组代替固定数组。当不再需要动态数组的数据时，用 `Erase` 或 `ReDim Preserve` 放弃不需要的数据，并回收数组所用内存。例如，用以下代码可回收动态数组所用空间：

```
Erase MyArray
```

这里，`Erase` 完全删除数组，`ReDim Preserve` 则只缩短数组而不丢失其内容：

```
ReDim Preserve MyArray(10, smallernum)
```

删除了固定大小数组，也不能回收该数组所占空间——只是简单地清除数组每一元素中的值。如果元素是字符串，或包含字符串或数组的 `Variant` 变量，那么删除数组可回收这些字符串或 `Variants` 所占内存，而不是数组本身所占内存。

回收被字符串或对象变量用过的空间

当过程结束时，可自动回收（非静态）局部字符串和数组变量所用空间。但是，全局和模块级的字符串和数组变量一直存活到整个程序结束。要想应用程序尽量小，就得尽可能回收这些变量所用空间。将零长度字符串赋给字符串变量，可回收其空间：

```
SomeStringVar = "" '回收空间。
```

同样地，将对象变量设置成 `Nothing` 可回收该对象所用的部分（而不是全

部) 空间。例如，删除一个 **Form** 对象变量：

```
Private rs As New RecordSet
```

```
...                '此处为初始化和使用记录集的代码  
rs.Close           '关闭记录集  
Set rs = Nothing   '设置对象引用为 Nothing
```

如果不把对象引用设置为 **Nothing**，对象引用将驻留在内存中，直到终止应用程序。大量使用对象的应用程序会快速消耗可用内存，使应用程序运行速度减慢。

即使没有使用显式窗体变量，也应注意将不再用的窗体卸载，而不是简单地隐藏。

消除死代码和无用的变量

在开发和修改应用程序时，可能遗留了死代码——代码中的一个完整过程，而它并没有被任何地方调用。也可能声明了一些不用的变量。虽然，在创建 **.exe** 文件中，**Visual Basic** 确实可删除无用的常数，但不能删除无用的变量和死代码。注意要复查代码，查找并删除无用的变量和死代码。例如，**Debug.Print** 语句，在运行 **.exe** 时被忽略，可它常常出现在 **.exe** 文件中。当创建 **.exe** 文件时，含有字符串和变量作为参数的 **Debug.Print** 语句不会被编译。但是，对于含有函数作为参数的 **Debug.Print** 语句，其本身被编

译器忽略，而函数则被编译。因此，在应用程序运行时，函数被调用，但返回值被忽略。因为，在 .exe 文件中，函数作为 Debug.Print 的参数出现时，将占用空间和 CPU 周期时间，所以在生成 .exe 文件前，最好删除这些语句。

在“编辑”菜单中使用“查找”命令搜索特定变量的引用。或者，当每个模块都含有 Option Explicit 语句时，通过删除或注释该变量的声明，并运行应用程序，可迅速发现变量是否被使用。若该变量被使用，则 Visual Basic 将出错。若不出错，则该变量没被使用。

详细信息 关于 Debug.Print 语句的详细信息，请参阅第十三章“调试代码和处理错误”中“在立即窗口中打印信息”一节。

修剪图形

图形（图片和图形方法）要消耗许多内存。从某种程度上说，这是不可避免的：图形包含很多信息，所以规模巨大。但在许多情况下，通过以下方法，可减少图形对应用程序大小的影响：

- 使用 Image 控件显示位图。
- 需要时从文件中加载位图并共享图片。
- 使用 PaintPicture 方法。
- 释放图形所用内存。
- 使用 RLE 格式位图或元文件。

使用 Image 控件显示位图

在许多 Visual Basic 应用程序中，Picture 控件仅仅被用于单击和拖放。如果您也是这样使用 Picture 控件的，这就浪费了许多 Windows 资源。为了避免浪费资源，Image 控件就比 Picture 控件好。每一个 Picture 控件是一个真正的窗口，使用了大量的系统资源。而 Image 控件是轻图形控件而不是一个窗口，并且使用的资源不多。其实，典型情况是：使用五到十个 Image 控件相当于使用一个 Picture 控件。另外，Image 控件重画的速度比 Picture 控件快。只有在需要仅 Picture 控件才有的性能时才使用它，如动态数据交换 (DDE)、图形方法或包含其它控件的能力。

需要时从文件中加载位图并共享图片

在设计时，设置 Picture 属性，图形就会被添加到窗体中，增加了窗体在运行时所用的内存。通过将图片存储资源文件，并在运行时使用 LoadResPicture 加载图片，可以减少内存消耗。如果并没有用到所有与窗体关联的图片，那么这一技术比将所有图片存储在窗体上的控件中节省内存。而且，由于在窗体显示前，并不需要加载所有的图片，因此加快了窗体的加载速度。

此外，可在多个 Picture 控件、Image 控件和窗体之间共享相同图片。如果使用下列代码，则只需维护一个图片副本：

```
Picture = LoadPicture("C:\Windows\Chess.bmp")
```

`Image1.Picture = Picture` '使用相同的图片。

`Picture1.Picture = Picture` '使用相同的图片。

与上面相比，下列代码加载了同一位图的三个副本，占用了更多的内存和时间：

```
Picture = LoadPicture("C:\Windows\Chess.bmp")
```

```
Image1.Picture = LoadPicture("C:\Windows\Chess.bmp")
```

```
Picture1.Picture = LoadPicture("C:\Windows\Chess.bmp")
```

同样地，如果在设计时将相同的图片加载到几个窗体或控件中，每一个窗体或控件都存储了该图片的一个副本。然而，应该将图片存于一个窗体中，而让其它窗体或控件用上述的方法共享。这将使应用程序既小（因为不包含冗余的图片拷贝）又快（因为不必从磁盘中多次加载）。

使用 `PaintPicture` 方法

不必总将位图直接放于控件上，也可以使用 `PaintPicture` 方法在窗体任何地方显示位图。当在窗体中重复平铺一个位图时，这种方法特别有用：只需加载位图一次，但可以使用 `PaintPicture` 绘制多次。

释放图形占用的内存

当不再使用窗体、图片框或图形控件中 `Picture` 属性的图片时，将 `Picture` 属性设置成 `Nothing`，释放空间：

```
Set Picture1.Picture = Nothing
```

如果使用图片框或窗体的 Image 属性，Visual Basic 创建一个 AutoRedraw 位图（即使 AutoRedraw 属性是 False）。在 Image 属性用完后，将 AutoRedraw 设置成 False 之前，用 Cls 方法可回收该位图所占内存。例如，下列代码回收 mypic 控件中 Image 属性所用内存：

```
mypic.AutoRedraw = True '打开 AutoRedraw 位图。
```

```
mypic.Cls '清除。
```

```
mypic.AutoRedraw = False '关闭位图。
```

使用 RLE 格式位图或图元文件

Visual Basic 的缺省图片格式是 bitmap (.bmp)，但也可使用其它图形文件格式。有几种绘图和图形程序允许将位图存为标准压缩位图格式，称作行程编码 (Run Length Encoded) (.rle)。RLE 位图比非压缩副本小几倍，特别是那些包含大片实心颜色的位图，而且它们在加载或显示时一点也不慢。使用元文件可节省更多的内存——有些情况下，可达 10 倍或者更多。当元文件被放大或缩小时，绘图速度很慢，所以最好以原图大小使用图元文件。从 Visual Basic 5.0 开始，也支持 .gif 和 .jpg 格式。这些格式一般比较小，但是，在选择格式时应权衡图形质量和加载速度。

分段应用程序

Visual Basic 提供一种以新方法设计应用程序的体系结构。代替单一的整体执行程序，可以写一个这样的应用程序，它包含一个核心的前端执行程序，并由许多 ActiveX 部件来支持。这种方法有以下几个显著的优点：

- 部件在需要时加载，不再需要时卸载。
- 在 Windows 95 或 Windows NT 下，即使应用程序的其它部分是 16 位部件，跨进程的部件也可以是 32 位的执行程序。
- 远程部件可使用网络上其它机器的资源。

另外，部件可被单独调试并可在其它应用程序中重用。也许，这不能提高创建应用程序的速度，但可提高创建下一个应用程序的速度。

在确定如何使用分段技术最优化应用程序之前，必须估计到能创建的部件类型和怎样适合应用程序。使用 Visual Basic 专业版和企业版，能创建以下三类部件：

- 跨进程
- 进程内
- 远程

三种类型并不互相排斥：可以在一个应用程序中同时使用这三类部件。但从优化应用程序的观点看，它们各有极不相同的特性。

详细信息 在 Visual Basic 专业版和企业版所提供的《部件工具指南》中深入讨论部件的创建。

跨进程部件

跨进程部件是一个可执行程序，它可以向其它程序提供服务。象所有的可执行程序一样，启动后，在自己的进程空间内有自己的堆栈；因此，当作为客户使用应用程序使用部件提供的对象时，操作将从客户的进程空间转到部件的进程空间——因此而得名。与其它类型相比，跨进程部件提供了一些很有价值的特点：

- 异步操作（“线程”）。
- 部件中不能捕获的错误不会导致调用应用程序崩溃。
- 在 16 位和 32 位应用程序之间互操作。

当然，从优化观点看，第一点和最后一点最有意义。

因为跨进程部件是一个分离的程序，可以与作为客户的部件异步操作。它有一个单独的“线程”，与客户程序构成多任务（从技术上讲，这不是线程，而是一个分离的进程；但从概念上看，二者是等价的）。两个程序可相互通讯并共享对象，但它们是独立运行的。当应用程序需执行一些费时的操作时，异步操作特别有用。客户可先调用部件执行该项操作，而后继续响应用户。

即使应用程序将在 32 位系统上运行，如果你惯用的 16 位应用程序或部件，可以不必立即将它们改成 32 位。若使用跨进程部件将应用程序分段时，则可将 16 位和 32 位部件混合在一起并相互适应。这将有助于充分利用 32 位特性，并保护了在 16 位部件上的投资。

考虑它们所有的能力，跨进程部件都有一个明显的缺点：性能。以下几点

可显现这些不足：

- 启动速度
- 跨进程调用的开销

跨进程部件是一个由 Visual Basic 创建的执行程序，因此，与应用程序的启动相关的启动问题也同样存在。好在，当从另一个 Visual Basic 程序中调用在 Visual Basic 中写的跨进程部件时，几乎所有的支持 DLL 已经被加载。这就大大缩短了启动部件的时间。许多部件比 Visual Basic 应用程序的平均规模还小，它们具有很少或根本没有要加载的窗体，这又进一步缩短了加载时间。然而，跨进程部件启动总比进程内部件慢。

一旦运行起来，跨进程部件就必须承受它的固有缺点：部件间每一次互操作就是一次跨进程的调用。跨越进程边界占用大量的 CPU 周期。因此，从跨进程部件引用对象要比从客户应用程序自身引用或从进程内部件中引用付出更多的代价。减少代码中跨进程的调用，就可以减少跨进程调用开销的影响。

进程内部件

进程内部件在其进程空间内向其它程序提供服务。与跨进程部件相比，进程内部件有两个优点：

- 改善加载时间
- 无跨进程开销

对进程内部件来说，既无需创建新的进程，也不必加载运行时 DLL。因此，

进程内部件比跨进程部件加载速度快得多。

因为它是在进程内的，在引用部件提供的对象的方法或属性时，就没有跨进程开销。因此，部件的对象操作起来与客户应用程序自身中的对象一样效率很高。

当然，对进程内部件也有一些限制。最重要的可能是：部件必须是 32 位，并且不能使用模态窗体。

远程部件

在 Visual Basic 企业版中，可创建远程部件，该部件能在网络任何地方的机器上单独运行。虽然，网络开销将不可避免地应用程序性能付出代价，但可通过使用别的 CPU 资源来弥补。当使用远程部件，并且部件操作的数据对包含部件的机器来说是本地数据时，这样做尤为正确。这是因为数据必须从网络中其它地方获取，部件可在本地操作数据，之后只通过网络返回结果，这可能是比较有效的。

例如，可在部件中编写一个对象，用于搜索本地硬盘中满足特定条件的文件。将该部件做成远程部件，并在网络的所有机器上放置一个副本，再编写一个分布式文件查找程序，就可使用所有这些 CPU 资源并行搜索所有网上部件。

优化对象

在 Visual Basic 应用程序中，对象的使用越来越多，因此优化对象的使用变得越来越重要。以下几种关键技术可以最有效地使用对象：

- 使用前连接。
- 尽量少用点操作。
- 使用 Set 和 With...End With。
- 尽量减少跨进程调用。

在 Visual Basic 中，在代码中（通过获取或设置对象的属性，或执行它的一个方法）引用另一个应用程序的对象将构成一个跨进程的调用。因为，跨进程调用的代价昂贵，所以，当须考虑优化应用程序时就应避免使用它们。

事前连接与事后连接的对比

如果对象可事前连接，Visual Basic 使用对象就更有效。若提供了对包含对象的类型库的引用，并声明了对象的类型，那么对象就可以事前连接：

```
Dim X As New MyObject
```

或者，相当于：

```
Dim X As MyObject
```

```
Set X = New MyObject
```

事前连接可使 Visual Basic 在编译时完成大多数处理对象的定义的工作，而不是在运行时处理。这也可使 Visual Basic 检查对象所用属性和方法的语法错误，并报告错误。

若 Visual Basic 不能事前连接对象，那么就必须是事后连接。事后连接对象的代价很大：在编译时不进行错误检查，在运行时每个引用需要至少由 Visual Basic 增加 50% 的工作量。

通常，应尽可能事前连接对象。如果没有针对问题的对象的类型库，或需要将任意类型的对象作为参数传给过程，只有这时才不得不将变量声明为 As Object。

详细信息 关于事前连接的详细信息，请参阅第十章“用部件编程”中“加速对象引用”。

尽量少用点操作

当从 Visual Basic 中引用其它应用程序的对象时，可使用点语法“.”对对象的集合、对象、属性和方法进行定位。创建特别长的定位字符串并不常见。例如：

'引用 Sheet1 中单元 A1，它在 Microsoft Excel 电子表格的第一部
'工作册上。

```
Application.Workbooks.Item(1).Worksheets.Item_  
("Sheet1").Cells.Item(1,1)
```

除了键入的字符串相当长以外，这行代码还相当难读——并且效率极其低。

当从 Visual Basic 中调用对象时，每一个“点”都需要 Visual Basic 产生多次调用。

为了写出最有效的应用程序，引用对象时尽量少使用点操作。

通过分析可用的对象和方法，常常可减少点操作的数量。例如，删除 Item 方法（对于集合，这是缺省的方法，因此在代码中很少用）并使用比较有效的 Range 方法，就可缩短上面的代码行：

'引用 Sheet1 中单元 A1，它在 Microsoft Excel 电子表格的第一部
'工作册上。

```
Application.Workbooks(1).Worksheets("Sheet1").  
.Range("A1")
```

重写代码，上面的代码行还可进一步缩短。引用活动工作册上的活动工作单，而不是指定工作册上的指定工作单：

'引用活动工作册上的活动工作单中的单元 A1。

```
Range("A1")
```

当然，上面的示例假设引用任何碰巧是活动的工作单上的单元 A1 都是可以的。

使用 Set 和 With...End With

用 Set 语句也可缩短定位字符串，并提供对代码更灵活的控制。下面的示例使用 Dim 和 Set 语句创建变量，使其引用经常使用的对象：

```
Dim xlRange As Object
```

```
Set xlRange = Application.ActiveSheet.Cells(1,1)
```

```
xlRange.Font.Bold = True
```

```
xlRange.Width = 40
```

Visual Basic 提供了 With...End With 语句，用来设置代码中隐含的对象：

```
With Application.ActiveSheet.Cells(1,1)
```

```
    .Font.Bold = True
```

```
    .Width = 40
```

```
End With
```

尽量减少跨进程调用

使用跨进程 ActiveX 部件，就不能完全避免产生跨进程调用。但是，有几种方法可减少跨进程的调用。如果可能，不要在 For...Next 循环中引用对象。将值缓存在变量中，然后在循环体中使用变量。若必须调用对象中大量的方法，通过将代码移入部件，可大大改进应用程序性能。例如，若部件是 Word 或 Microsoft Excel，可将循环宏放入 Word 的模板，或将循环过程放入 Microsoft Excel 模块。然后，从 Visual Basic 中调用宏或过程，这只是一次调用，它将在部件中引发一个循环操作。

在编写部件时，通过减少跨进程的调用可将部件中对象设计得更有效。例如，当有几个内关联的属性时，可使用带有多个参数的方法，其中，每个参数对应一个属性。无论方法带有多少个参数，调用方法只需一次跨进程调用，反之，设置每个属性则需一次单独的跨进程调用。同样地，如果预

先知道，作为客户的部件将循环调用该部件（如，对 List 属性的所有值求和或求平均），那么，可提供在对象中完成循环的方法，然后返回适当的结果，这样可以改善程序性能。

详细信息 在 Visual Basic 专业版和企业版所提供的《部件工具指南》中深入讨论了部件的创建。

编译型和解释型应用程序的比较

按照缺省规定，Visual Basic 将应用程序编译成可运行的解释型或 P-code 程序。运行时，动态链接库（DLL）将翻译或解释执行程序中的指令。Visual Basic 专业版和企业版还可将程序编译成 .exe 本机代码。在许多情况中，运行本机代码比运行解释型代码本质上要快得多。但事情并非总是这样。以下是一些有关本机代码编译的一般指南：

- 若代码做了大量固定类型的、非字符串变量的基本操作，则其产生的本机代码将与编译的 P-code 操作码产生最大的反差。然而，对于复杂的经济计算或生成分形图形，用本机代码有很多好处。
- 计算密集型程序，或在局部数据结构中处理大量的位和字节操作的程序，用本机代码就可获得明显的好处。
- 在许多程序中，特别是那些含有大量 Windows API 调用，COM 方法调用和字符串操作的程序，本机代码不比 P-code 快多少。
- 如果应用程序主要含有来自 Visual Basic for Applications 运行库的函

数，那么就看不到多少本机代码的好处，这是因为 Visual Basic for Applications 运行库已经高度优化了。

- 含有大量子例程调用而非内嵌过程的代码，用本机代码也似乎不太快。这是因为设置栈结构，初始化变量，返回时的清除等工作，用 P-code 引擎和用本机代码所花费的时间相同。

注意，调用一次对象、DLL 或 Visual Basic for Applications 运行函数，都会消弱本机代码性能上的优点。这是因为执行代码所花时间相对地少，而大量时间（常常达到 90-95%）花在窗体、数据对象、Windows DLL，或 Visual Basic for Applications 运行库，以及处理内部字符串和变量上。

实际测试中，典型情况是客户应用程序总的执行时间有 5% 用于执行 P-code。因此，如果本机代码是瞬时执行的，则在这些程序中用本机代码也只能改进性能最高达 5%。

本机代码的目的是可让程序员用 Basic 编写一小片代码或有大量计算的算法。这些，由于性能问题，在以前是决不可能的。使用这些运行非常快的“小片”，也能改善应用程序特定部分的响应，从而改进了整个应用程序的感觉性能。

详细信息 关于本机代码的编译的详细信息，请参阅第八章“再论编程”的“将工程编译成本地代码”。

与其它 Microsoft 应用程序的兼容性

Visual Basic 是 Visual Basic 产品家族中的高级成员。Visual Basic 产品有 Visual Basic for Applications、Visual Basic 和 Scripting 版 (VBScript)。用 Visual Basic 5.0 写的大部分代码可与 Visual Basic for Applications 或 VBScript 所写的应用程序共享。但也有一些例外。

与其它 Microsoft 应用程序的兼容性

Visual Basic for Applications 是单一的通用的应用程序脚本语言和环境，用户和开发者可在 Windows 桌面上扩展其功能。Visual Basic for Applications 包含在 Microsoft Office 和其它 Microsoft 应用程序中。也允许其他软件商使用，并包含在范围广泛的其它产品中。

Visual Basic for Applications 5.0 是 Visual Basic 5.0 基本语言引擎，包含在 Vba5.dll 中。这个库包含了所有可为 Visual Basic for Applications 和 Visual Basic 共享的语言元素。从“对象浏览器”的“Library”列表框中选择 VBA 可浏览这些语言元素。在 Visual Basic for Applications 下编写的代码可移植到 Visual Basic 中，但有以下限制：引用特定的应用程序元素（如 Microsoft Excel 工作单）的 Visual Basic for Applications 代码，只有该引用是完全合理的，而且所引用的应用程序在目标机器上，才可以移植。

Visual Basic 的特有元素，如窗体和固有控件，都包含在 Vb5.olb 类型库里

(也可用对象浏览器浏览它们)。一般地,只要不引用这些元素,在 Visual Basic 中编写的代码就可移植到 Visual Basic for Applications 中。

详细信息 关于 Visual Basic for Applications 的详细信息,请访问 Microsoft Web,地址为 <http://www.microsoft.com>。关于引用对象的详细信息,请参阅第十章“用部件编程”中“创建对对象的引用”。关于本地代码编译的详细信息,请参阅第八章“再论编程”中“将工程编译成本地代码”。

与 Visual Basic Scripting 版的兼容性

Visual Basic Scripting 版 (VBScript) 设计为宿主于 Internet 浏览器中的语言,如 Microsoft Internet Explorer,或其它第三方浏览器。对于特别像 Internet, intranets, 或 World Wide Web 这样的环境,VBScript 是一个轻量 and 极快的语言引擎。VBScript 扩展了 Visual Basic 的能力,并能让开发者使用 Visual Basic 的开发知识,为 Internet 和 World Wide Web 快速创建应用程序。

VBScript 支持 Visual Basic for Applications 语言语法的一个子集。但 VBA 没有像 Microsoft Visual Basic 中那样的 IDE。这是因为 Visual Basic Scripting 设计为轻量语言引擎,其可被不同的平台共享。可以在 Visual Basic 代码编辑器中编写 VBScript 代码,但不能在 Visual Basic IDE 下运行或测试该应用程序。

因为 VBScript 是一个跨平台开发语言,因此,Visual Basic for Applications 语言的某些元素并不包含在内。这包括文件输入/输出函数,固有常数,固

有数据类型等等。当将代码从 Visual Basic 移植到 VBScript 时，就必须注意这些不支持的元素。

详细信息 关于 Visual Basic Scripting 版所支持的语言元素清单的详细信息，请访问 Microsoft Web，地址为 <http://www.microsoft.com>。

第十六章 国 际 化

如果计划向国际市场发行应用程序，开发在国内市场和国际市场具有相同功能的应用程序，使用 Visual Basic 可以节省时间并简化代码。本章介绍了用 Visual Basic 开发国际应用程序的主要概念和定义：首先展示一个本地化模型，然后重点阐述设计国际化软件的优点。

本章还讨论了如何用 Visual Basic 编写灵活、便捷和真正的国际应用程序。其中一部分专门介绍编写 Visual Basic 代码处理东亚版 Windows 的双字节字符集 (DBCS)。

目 录

- 国际化软件的定义
- 设计国际化软件
- 使用本地化资源文件
- 设计区域识别的用户界面
- 编写国际化代码时的一般考虑
- 用 Visual Basic 编写国际化代码
- 双字节字符集 (DBCS) 的特殊问题

示例应用程序： `Atm.vbp`

本章中的部分列在 `Samples` 目录中的代码示例取自示例自动取款机 (`Atm.vbp`)。

国际化软件的定义

开发国际化软件前需要了解一些基本术语。

国际化软件

国际化软件是指在全世界都可以出售的软件。只有国内市场版本和国际市场版本功能相同的软件才能称为国际化软件。关于如何本地化应用程序的详细信息，请参阅本章后面的“设计国际化软件”。

区域

区域描述了用户的环境——用户地理区域内的惯例、文化和语言。区域是语言和国家/地区的唯一的组合。区域的两个例子是：英语 / 美国和法语 / 比利时。

一种语言可能在不只一个国家/地区中使用，比如：法国、比利时、加拿大

和一些非洲国家/地区都使用法语。虽然这些国家/地区都使用同一种语言，但各国都有自己独特的惯例（如货币的表示方法）。因此每个国家/地区代表一个唯一的区域。同样地，一个国家/地区可能有不止一种官方语言：比利时有三种语言：法语、荷兰语和德语。因此比利时有三个不同的区域。关于特定区域设定的详细信息，请参阅本章后面的“编写国际化代码时的一般考虑”。

本地化

本地化是指使应用程序适应区域的过程。不仅指在文字上逐字逐句地翻译这些资源，更重要的是要与用户交流。关于如何本地化应用程序的详细信息，请参阅本章后面的“设计国际化软件”。

字符串资源

字符串资源是指在应用程序的用户界面中出现的所有文本。其中包括菜单、对话框以及提示、警告和错误信息，但不仅限于此。如果应用程序要在不同于开发区域的其它区域中使用，就需要翻译或本地化这些资源。

详细信息 东亚术语的定义请参阅本章后面的“ANSI, DBCS 和 Unicode 的定义”。关于字符串资源和资源文件的详细信息，请参阅本章后面的“使用本地化资源文件”。

设计国际化软件

从设计开始就考虑应用程序的本地化，把字符串资源从代码中分离出来，这样做比在后面的开发过程中修订已完成的应用程序效率要高得多。

设计国际化软件的优点

设计和实现 Visual Basic 应用程序，以适应国际惯例、外国数据及格式处理，主要有以下四个优点：

- 使 Visual Basic 应用程序更快地面市。初始版本完成后不需要再进行国际化的工作。
- 更有效地使用资源。在已完成 Visual Basic 应用程序中增加国际化支持可能使稳定性降低。这样做与在原始开发过程就提供该项支持相比，在开发和测试上要花费更多的资源。
- 如果国际版本的 Visual Basic 应用程序使用了原始开发应用程序的相同源代码，只有一些独立的模块需要国际化，那么在进行国际化支持的过程中可以更方便、更节约地维护代码。请参阅本章后面的“使用本地化资源文件”。
- 开发国际应用程序更加简便。比如，可以开发英语版的应用程序在德语操作系统下使用而不必重新编写代码。需要做的只是定制用户界面。请参阅本章后面的“设计区域识别的用户界面”。

本地化模型

本地化的应用程序都是由两个概念块构成：代码块和数据块。如图 16.1 所示，数据块代表“用户界面部件”，代码块代表“应用程序部件”。

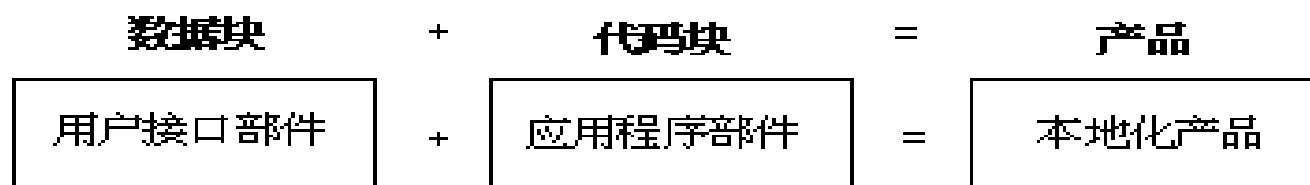


图 16.1 数据块和代码块构成一个本地化产品

数据块包含所有用户界面的字符串资源而不包含代码。相反，代码块只包含可运行于所有区域的应用程序代码。Visual Basic 代码处理字符串资源和区域的设定。在“用 Visual Basic 编写国际化代码”一节中将详细介绍如何编写 Visual Basic 代码来处理诸如日期、货币、数字和分隔符等与区域有关的设置。

理论上开发本地化版本的 Visual Basic 应用程序只需修改数据块。代码块对所有的区域都是相同的。把数据块和代码块结合在一起就构成了本地化版本的应用程序。成功地设计国际应用程序的关键在于分离数据块和代码块，并使 Visual Basic 应用程序能够准确地读取数据而不用考虑区域。

尽管要麻烦一些，但在开发 Visual Basic 应用程序时还是应该把所有用户界面的字符串资源放到单独的文件而不是代码中，该文件可以在运行时加

载。这就是资源文件 (.res)。它包含了所有本地化的字符串、位图和图标。关于资源文件的详细信息，请参阅本章后面的“使用本地化资源文件”。开发不同语言版本的应用程序时，本地化小组应把精力只花在资源文件上。这样做有以下优点：

- **高效：** 由于代码块是相同的，因此开发新的国际应用程序只需创建新的资源文件。这样可以很顺利地开发多国语言版本的 Visual Basic 应用程序。
- **安全：** 无论在内部或利用外部公司本地化应用程序，都不用修改源代码。这样做还可以减少花在国际版测试上的精力。
- **更好的本地化：** 由于所有的字符串资源都在一个文件中，因此可以提高本地化工作的效率并防止遗漏。

下表列出了在设计 Visual Basic 应用程序时要考虑的一些因素。

因素	项目
语言	用户界面（菜单、对话框及错误信息）中的字符串
	打印的文档和联机的文档
与区域有关的设置	日期 / 时间格式（分隔符、年月日的顺序）
	数字格式（小数点和千分位符）
	货币格式（符号和格式）
	排序顺序和字符串比较

作为第一因素的语言，在 Visual Basic 应用程序的设计阶段就应该首先指出。详细信息，请参阅“设计区域识别的用户界面”。与区域有关的设置，将在本章中的“用 Visual Basic 编写国际化代码”及“国际排序和字符串

比较”中介绍。

使用本地化资源文件

Visual Basic 通过使用资源文件有效地把本地化信息从代码中分离出来。

注意 工程中只能有一个资源文件，如果试图添加一个以上的资源文件 Visual Basic 将产生一个错误信息。

把字符串保存在资源文件中的优点

编写代码时，可以调用 LoadResString、LoadResPicture 和 LoadResData 函数代替对文字、图象和数据的引用。把这些元素存储在资源文件中有两大好处：

- 提高性能，这是因为字符串、位图、图标和数据可以不必在调用窗体或模块时就全部加载，而可以按需要从资源文件中加载。
- 把需要翻译的资源单独放在一个资源文件中，这样就没有必要访问源代码或重新编译应用程序。

要创建资源文件，请按照以下步骤执行：

1. 从“工程”菜单中选择“新资源文件”。

注意 该命令只有在资源编辑器外接程序加载的情况下才能使用。要加载资源管理器外接程序，请选择“外接程序”菜单中的“外接程

序管理器”。在“外接程序管理器”对话框中，选择“VB6 资源编辑器”并复选“加载/卸载”。

2. 在“打开一个资源文件”对话框中，为资源文件输入一个名称。资源文件名称将被添加到“工程资源管理器”的“相关文档”节点。

注意 您也可以将一个现存的资源文件添加到您的工程。然而请注意，您对现存资源文件所做的任何更改可能会影响其他使用该资源文件的应用程序。

Visual Basic 把具有 .res 扩展名的文件识别为资源文件。如果资源文件不具有正确的扩展名，Visual Basic 将不会加载它。相反地，任何具有 .res 扩展名的文件被加入到工程时都会被 Visual Basic 当作资源文件处理，如果该文件不具有标准的资源文件格式，Visual Basic 将在第一次使用资源文件支持函数（LoadResString、LoadResPicture 和 LoadResData）或试图编译成 .exe 文件时产生错误信息。试图添加 16 位的资源文件到工程中也会产生同样的错误信息。

资源文件被添加到工程后，.res 文件将出现在“工程”窗口中。与窗体或模块不同，在 Visual Basic 中不能查看 .res 文件。该文件是标准的资源文件，可由 Microsoft Visual C++ 和其它大部分的 Windows 开发工具创建或使用。一旦在“文件”菜单中选中“生成 projectname.exe”一项，Visual Basic 将把该文件中的所有资源编译到 .exe 文件中作为 Windows 的资源。

在编译 .exe 文件之前和之后，.res 文件都是标准的 Windows 资源文件，这意味着任何标准的 Windows 资源编辑器都可以加载该文件。

要编辑资源文件，请按照以下步骤执行：

1. 从“工具”菜单中选择“资源编辑器”。

注意 该命令仅在资源编辑器外接程序加载的情况下才能使用。要加载资源编辑器外接程序，请从“外接程序”菜单中选择“外接程序管理器”。在“外接程序管理器”对话框中，选择“VB6 资源编辑器”并复选“加载/卸载”框。

2. 从“资源编辑器工具栏”中选择一个按钮来编辑一个现存资源文件或添加一个新的。有关编辑资源文件的更详细信息，请参阅资源编辑器外接程序文档。

锁定资源文件

Visual Basic 可以对 .res 文件进行文件锁定，以防止同时有几个应用程序使用该文件。在下列条件下 .res 文件将被锁定：

- Visual Basic 处于运行模式或中断模式。
- 创建了一个 .exe 文件。

详细信息 关于如何使用资源文件开发多区域应用程序的示例，请参阅本章后面的“自动取款机应用程序示例”。关于使用资源文件编程的背景资料，请参阅第八章“再论编程”中的“利用资源文件进行工作”。

自动取款机示例应用程序

本示例应用程序演示如何在 Visual Basic 中使用资源文件。该应用程序包

含三个窗体、一个标准模块和一个资源文件。运行示例应用程序自动取款机 (Atm.vbp) 后，将显示一个打开的屏幕，允许使用多种语言，包括德语、法语、意大利语和西班牙语，进行银行业务操作。

以下代码摘自 FrmInput.frm，加载保存在 Atm32.res 文件中的资源，该文件包含所有语言的本地化字符串。

```
Sub Form_Load()  
    imgFlag = LoadResPicture(I, vbResBitmap)  
    Caption = LoadResString(I)  
    lblPINCode = LoadResString(1 + I)  
    fraAccount = LoadResString(2 + I)  
    optChecking.Caption = LoadResString(3 + I)  
    optSavings.Caption = LoadResString(4 + I)  
    lblAmount = LoadResString(5 + I)  
    cmdOK.Caption = LoadResString(6 + I)  
    SetCursor cmdOK  
End Sub
```

```
Sub cmdOK_click()  
    '显示一条操作信息。  
    MsgBox LoadResString(7 + I)  
    frmAmountWithdrawn.Show vbModal  
    Unload Me
```

End Sub

运行时，本段代码根据用户在打开的屏幕上选择的语种所决定的偏移量，从资源文件中读入相应的部分。偏移量是在标准模块中定义的公共变量，它表示从起点到特定条目的距离。在示例应用程序 ATM 中，偏移量为 I。资源文件保留资源标识符 16 至 47 用于英语，资源标识符 48 至 79 用于法语，资源标识符 80 至 111 用于德语，以此类推。每种语言都有相应的本地化条目构成示例应用程序的数据块，当前每种语言的数据块包含十一项资源。

本示例应用程序包含了几个数据块，这是一种只采用一个资源文件实现与多种语言相关的资源文件的变通方式。根据应用程序的特性，可能考虑为每种语言使用一个资源文件，或使用一个资源文件包含所有本地化的数据块。

示例应用程序自动取款机的设计展示了超出本章前面所述内容的几个优点：

- 通过提供更多的语种，扩大了应用程序的使用范围。若想添加新的语种支持，只需把相同的数据块添加到资源文件中，然后对其本地化，并在屏幕上增加一个按钮。
- 当应用程序需要扩展功能时，比如，允许 ATM 用户进行存款操作，只需将资源文件中每种语言的标识符范围增大（如 160）。当前的标识符范围是 16 到 47，48 到 79 等等。

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“LoadResString 函数”，“LoadResPicture 函数”和“LoadResData 函数”。关于资源文件的内

容，请参阅第八章“再论编程”中“利用资源文件进行工作”以及第十五章“性能和兼容性设计”。

设计区域识别的用户界面

对应用程序进行本地化时文本会加长，因此在设计以下用户界面时应特别注意：

- 信息
- 菜单和对话框
- 图标和位图
- 访问键和快捷键

信息

英语字符串一般比其它语言的字符串短。下表列出了在初始长度基础上字符串长度增加的平均值。数据取自以前的 Visual Basic 本地化工程的平均增加率。

英语长度（字符数）	本地化字符串的长度增加率
1 到 4	100%
5 到 10	80%
11 到 20	60%

(续表)

21 到 30	40%
31 到 50	20%
超过 50	10%

设计界面时，应考虑上述的增加率，当信息过长时应允许文本折行。

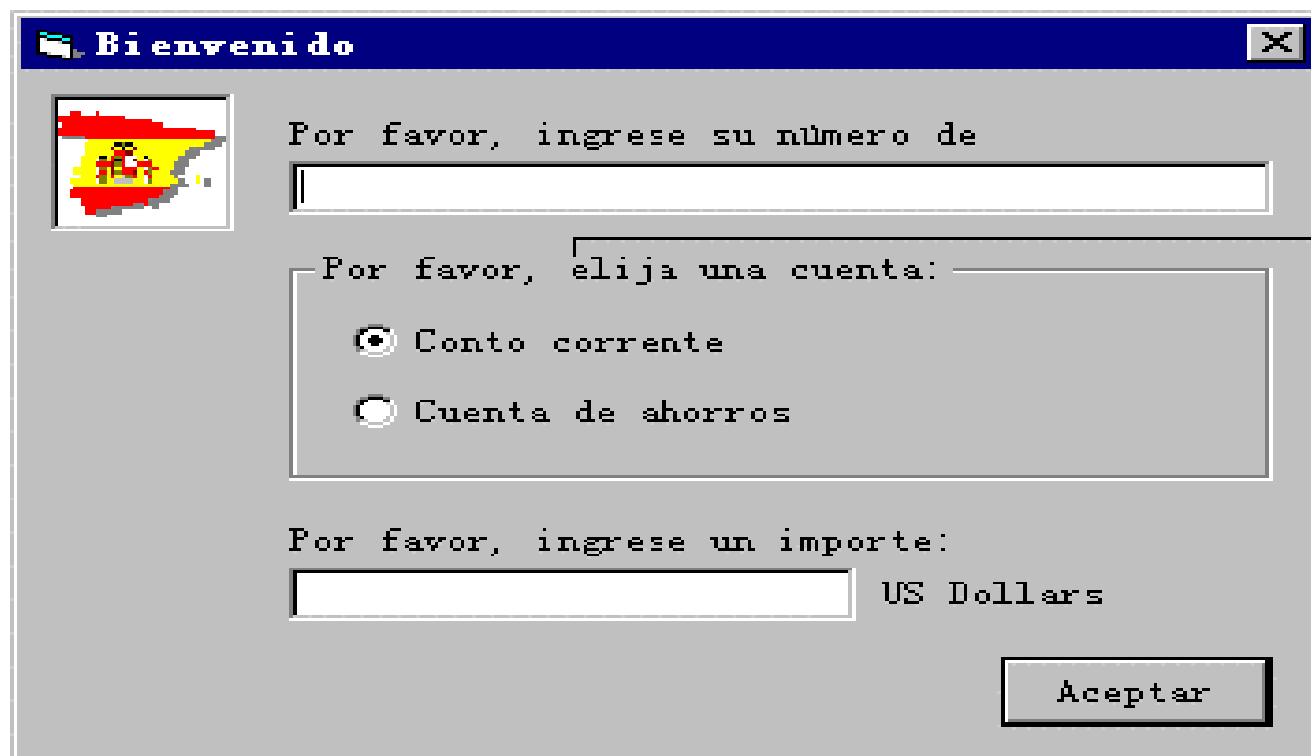
菜单和对话框

与信息类似，应用程序在进行本地化时，菜单和对话框的长度会增加。在以下自动取款机示例应用程序中两个内容相同的对话框中，为了允许文本变长增加的附加的空格。图 16.2 为英语对话框，图 16.3 为西班牙语对话框。预先考虑文本长度的增加，安排好界面，这样在本地化时就不用再改变控件的大小或重新设计其它的元素了。



图 16.2 在 ATM 示例应用程序中的英语输入对话框

由于在其它语言中即便使用缩写文本也可能太长或根本不存在缩写，因此在菜单和对话框中应避免使状态栏过于拥挤。



在 Spanish 版本中文
本更长一些

图 16.3 在 ATM 示例应用程序中的西班牙语输入对话框

图标和位图

图标和位图常用来代替文本表示某个特定功能。使用图标和位图时应考虑以下规则：

- 避免使用非国际标准的位图。以下位图在美国表示邮箱，但其它国家/地区的人可能不认识。



- 避免使用包含文本的位图。否则要花费时间重画位图，另外文本的长度也可能成为障碍，如以下位图所示。



- 位图或图标在文化上是敏感的。在一个区域可以接受的在另一个区域可能是不合适的或令人不快的。

访问键和快捷键

不同的区域有不同的键盘布局。并非所有的键盘布局都包含全部字符。开发 Visual Basic 应用程序时，应保证组合的访问键和快捷键可以在国际键盘上重建。使键盘设置满足本地化要求的一个简单的方法是从 Windows 的控制面板上选择合适的键盘布局，并通过键盘布局图（一些参考手册中有）找到适当的访问键和快捷键组合。

由于一些访问键和快捷键组合不能在某些区域下使用，或已被某版本 Windows 系统保留，因此在开发 Visual Basic 应用程序时应尽量避免使用。以下是一些应该避免使用的字符的示例：

@ \$ { } [] \ ~ | ^ ' < >

绕过该限制的一个方法是在快捷键组合中使用数字键和功能键（F1、F2 等）替代字母。这样做虽然不太直观但不需要本地化，因为实际上所有的键盘

布局都有数字键和功能键。

注意 DBCS 字符不能被用作访问键或快捷键。

编写国际化代码时的一般考虑

无论使用 Visual Basic 或是其它工具，开发将要本地化的应用程序时，都必须估计到语言间的差异。指明需要本地化的字符串，允许字符串变长，避免字符串接在一起。

硬编码的可本地化字符串

“设计国际化软件”中的本地化模型介绍了数据块和代码块的概念。在创建包含所有本地化字符串的资源文件时，应特别注意只提取需要本地化的字符串。对那些部分或全部都不需本地化的字符串可以是硬编码的。相反的，应该确保所有需要本地化的资源都放到了资源文件中。

缓冲区大小

在根据字符串或词的长度声明缓冲区大小时，应确保缓冲区可以适应更长的词和字符串。关于翻译字符串的平均增长率，请参阅“设计区域识别的用户界面”。在 Visual Basic 代码中声明的缓冲区必须适应这种增长。

例如，Visual Basic 为“确定”这个词声明一个 2 字节的缓冲区，但实际上西班牙语中，该词译作“Acceptar”，这将引起程序溢出。双字节字符也要考虑这些相同的问题。关于 DBCS 的详细信息，请参阅本章后面的“双字节字符集 (DBCS) 的特殊问题”一节。

字符串联接

可以通过字符串联接来缩短串的长度。该方法允许把一个资源分解到几个字符串中。但是这样做有一定的危险，请注意以下的示例：

英语	法语
字符串 1: one after the other	字符串 1: un après l'autre
字符串 2: The controls will be deleted.	字符串 2: Les contrôles seront supprimés.
字符串 3: The forms will be deleted.	字符串 3: Les feuilles seront supprimées.

单独看，字符串 1、2、3 很容易本地化。代码执行字符串 2 + 字符串 1 或字符串 3 + 字符串 1 的合并操作，结果的英语含义很正确。但本地化字符串的结果却可能是错误的。比如在法语中，字符串 3 + 字符串 1 是错误的，因为在法语语法中窗体 (feuilles) 是阴性词，因此字符串 1 应变为“une après l'autre”而不是“un après l'autre”。类似的情况在其它许多外语中都有。因此只有把字符串 2 和 字符串 1 及字符串 3 和 字符串 1 合在一起保存在资源文件中才是避免此类错误发生的唯一办法。

在上述示例中，组成句子的词序对英语和法语都是一样的。但通常对这两种语言或其它许多外语，词序并不总是相同的（例如，德语和日语总是把

动词放在句尾)。以下示例列举了这种情况：

英语	法语
字符串 1: DLL	字符串 1: DLL
字符串 2: Missing Data Access	字符串 2: Accès aux données manquante
字符串 3: Missing OLE	字符串 3: OLE manquante

当代码执行字符串 1 + 字符串 2 的操作时，本地化版本将出现错误因为两个字符串组成的信息构不成一个句子。一个解决方法是把字符串 1 分别与字符串 2 和字符串 3 组合保存在资源文件中并去掉字符串 1。

以下是另一种解决方法：

英语	法语
字符串 2: Missing Data Access '1'	字符串 2: '1' d'accès aux données manquant
字符串 3: Missing OLE '1'	字符串 3: '1' OLE manquant

此时，本地化版本可以识别占位符 '1' 并在资源文件中进行必要的调整以构成本地化语言的句子。

最后一点特别要注意的是在英语中相同的词或句在本地化时可能要翻译为不同的词或句。请注意以下的示例：

英语	法语
字符串 1: Setup program	字符串 1: Programme d'installation
字符串 2: String1 did not complete successfully.	字符串 2: String1 a échoué

在英语版中，字符串 1 可以作为安装程序标志，也可以在字符串 2 中作为错误信息的一部分。在法语版中，字符串 1 作为单独的标志是合适的，

但要变成 “`Le programme d'installation`” 才能作为字符串 2 的一部分。

用 Visual Basic 编写国际化代码

对一个产品进行本地化不仅仅是翻译文本信息，该产品还必须支持本国的惯例和数字格式。为了使用不同格式的日期、货币、及数值和分隔符，必须了解 Visual Basic 是如何区分系统区域和代码区域的。

系统区域和代码区域

系统区域是应用程序用户的区域——它是用户输入和输出的参照，是通过设置操作系统的控制面板实现的。在 Visual Basic 中 `code locale` 总是英语/美国而不考虑用的是哪一国际版本。代码区域决定编程语言和所有与区域有关的设置。

日期

在 Visual Basic 中不要把日期当成字符串输入。应以 `# 月/日/年 #` 的格式输入日期以确保能被任何系统区域识别。由于 Visual Basic 只允许英语/美国作为编程的区域，因此无论应用程序在哪里运行，日期都是一样的。例如，在输入对话框中输入 `8/2/97`，

CDate ("8/2/97")

根据不同的系统区域返回以下结果：

操作系统	输出
法语/法国	08/02/97 （= 1997 年 2 月 8 日）
英语/美国	8/2/97 （= 1997 年 8 月 2 日）

但如果在代码中输入 8/2/97，

CDate (#8/2/97#)

根据不同的系统区域返回以下结果：

操作系统	输出
法语/法国	02/08/97 （= 1997 年 8 月 2 日）
英语/美国.	8/2/97 （= 1997 年 8 月 2 日）

如果在法国输入 8/2/97，应用程序将解释为 1997 年 2 月 8 日，这是因为在法国日期格式为日/月/年。而在美国输入相同的字符串，应用程序就知道该日期为 1997 年 8 月 2 日，因为日期的格式为月/日/年。

货币

避免在代码中按字符串输入货币。以下代码只能运行于把美元符号 (\$) 当作货币符号的区域。

```
Money = "$1.22"
```

```
NewMoney = CCur(Money)
```

如果本代码示例运行于法语/法国区域，此时“F”代表货币符号，Visual Basic 将产生一条错误信息“类型不匹配”，这是因为 \$ 不被本国当成货币符号。此时应只简单地使用数字，如下例所示。由于 Visual Basic 的区域总是英语/美国，因此使用句号作为小数点。以下代码将不考虑区域因而能正确执行。

```
Money = 1.22
NewMoney = CCur(Money)
```

数值和分隔符

在美国，句号 (.) 被作为小数点。但在一些欧洲国家/地区，小数点是逗号 (,)。类似地，美国把逗号作为分隔小数点左边每三位的千分位符，而一些欧洲国家/地区使用空格实现此功能。下表列出了不同数字格式的一些例子。

国家/地区	数字格式		
美国	1,234,567.89	1,234.56	.123
法国	1 234 567,89	1 234,56	0,123
意大利	1.234.567,89	1.234,56	0,123

注意 在 Visual Basic 中函数 Str 和 Val 总是把句号当成小数点。但这在大多数区域中是不适用的。因此应该使用 CStr, CDBl, CSng, CInt 和 CLng 函数把其它数据类型转换为需要的数据类型。这些函数通过系统区域来决定小数点。

详细信息 关于 Print 和 Format 函数的详细信息请参阅本章后面的“区域

识别函数”。关于函数“CStr 函数”，“CSng 函数”，“CInt 函数”，请参阅《Visual Basic 6.0 语言参考手册》。也可参阅第六章“变量、常数和数据类型”。

区域识别函数

不同的区域在显示日期、时间、数字、货币和其它信息时有不同的惯例。但是，没有必要了解用户所在区域的所有惯例。这是因为在 Visual Basic 中的一些函数使用用户的系统区域运行时自动确定惯例，系统区域可以使用操作系统提供的控制面板来设置。这些函数被称为区域识别函数。

Print 方法

尽管 Print 方法为不同的输出格式提供了很少的灵活性，但它确实使用了用户的系统区域。在以下示例中，将用正确的短日期格式打印日期，将用正确的小数点打印数字，将用正确的符号打印货币。

```
MyDate = #11/24/1997#
```

```
MyNumber = 26.5
```

```
Money = 1636.32
```

```
MyMoney = Format(Money, "###,###.##")
```

```
Debug.Print MyDate, MyNumber, MyMoney
```

此代码运行在英语/美国的区域时，以下输出将出现在立即窗口中。

11/24/1997 26.5 1,636.32

此代码运行在德语/德国的区域时，以下输出将出现在立即窗口中。

24/11/1997 26,5 1.632,32

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“Print 方法”。

Format 函数

Format 函数可以接受格式代码，但格式代码总是产生相同的输出而不考虑用户的区域。例如，格式代码“mm-dd-yy”对于比利时的用户不适用，因为他们的日在月之前。

为了增加更多的灵活性，Format 函数还提供了命名格式可以在运行时自动决定使用何种转换，包括 General Date, Long Date, Short Date 和 Long Time。使用命名格式产生基于用户的系统区域的输出。命名格式甚至可以采用用户本国语言的输出，包括月名和星期名。如下例所示：

```
MyDate = #8/22/1997 5:22:20 PM#
```

```
NewDate1 = Format(MyDate, "Medium Date")
```

```
NewDate2 = Format(MyDate, "Short Date")
```

```
NewDate3 = Format(MyDate, "Long Date")
```

```
NewDate4 = Format(MyDate, "General Date")
```

```
Debug.Print NewDate1, NewDate2, NewDate3, NewDate4
```

代码运行在英语/美国区域时，下列输出显示在立即窗口中：

22-Aug-97 8/22/97

Monday, August 22, 1997

8/22/97 5:22:20 PM

代码运行在法语/法国区域时，下列输出显示在立即窗口中：

22-août-97 22/08/97

lundi 22 août 1997

22/08/97 17:22:20f

详细信息 请参阅《Visual Basic 6.0 语言参考手册》里的“Format 函数”。

国际排序顺序和字符串比较

字符串比较在 Visual Basic 中应用很广。然而使用此功能时，如果忽略某些编程要求，有可能导致错误的结果。

文本排序

文本排序的意思是根据语言惯例对文本排序。由于格式和字体只代表表示形式而不是内容，因此它们与排序过程无关。乍看起来，文本排序看似简单：a 优先于 b，b 优先于 c，等等。然而，很多语言具有很复杂的排序规则。正确的国际排序不是英文文本排序的简单扩展，因此需要对排序过程有不同的理解。

正确的国际排序的意思是上下文相关排序。字符收缩和扩展是上下文相关排序的两个重要方面。

- 当把两个字符组合作为单个的唯一字母对待时，出现字符收缩。例如，西班牙语中的两个字符组合 *ch* 是一个单独字母，排序时排在 *c* 和 *d* 之间。
- 代表一个字符的一个字母在排序时却看作是两个，在这种情况下出现字

符扩展。例如，在德语 / 德国和德语 / 瑞士区域中， β (eszett) 等同于 ss。

而在德语 / 奥地利区域中， β 等同于 sz。

在进行排序顺序之前，必须考虑代码页。代码页是一顺序字符集，由一数字索引（代码指针）指向其中的每一个字符。因为有各种各样的代码页，所以单个代码指针在不同代码页中代表不同的字符。然而很多代码页共享代码指针 32 至 127（ASCII 字符集），除此之外，它们是不同的。通常，代码页中其它额外字母不是按字母排序的。

详细信息 关于东亚语言排序顺序的详细信息，请参阅本章后面的“DBCS 排序顺序和字符串比较”。

Visual Basic 中的字符串比较

每个国家/地区的字符串比较规则不同。Visual Basic 提供了许多具有区域识别功能的字符串比较工具，例如 Like 和 StrComp。为了有效使用这些工具，首先必须清楚地了解 Option Compare 语句。

用 Option Compare 语句比较字符串

使用此语句时，对给定的模块必须指定字符串的比较方法：Binary 或 Text。如果指定 Binary，则根据代表字符的内部二进制码的排序顺序进行字符串比较。如果指定 Text，则根据由用户系统区域决定的不区分大小写的文本排序顺序完成字符串比较。缺省的文本比较方法是 Binary。

在下面的代码示例中，用户在两个输入框中输入信息。接着，按适当的字

母顺序进行信息的比较和排序。

```
Private Sub Form_Click ()
Dim name1 As String, name2 As String
    name1 = InputBox("Enter 1st hardware name here:")
    name2 = InputBox("Enter 2nd hardware name here:")
If name1 < name2 Then
    msg = " " & name1 & " 'comes before ' " & _
    name2 & " ' "
Else
    msg = " " & name2 & " 'comes before ' " & _
    name1 & " ' "
End If
    MsgBox msg
End Sub
```

如果该代码在区域为英语 / 美国的系统下运行，输入 `printer` 和 `Screen`，则消息框内的输出如下：

```
'Screen' comes before 'printer'
```

这是基于缺省的文本比较方法为 `Binary` 时的结果。因为大写 `S` 的内部二进制代码小于小写 `p` 的内部二进制代码，所以条件语句 `Screen < printer` 被确认。当在模块的声明区内增加 `Option Compare Text` 语句后，Visual Basic 按不区分大小写的方法比较这两个字符串，输出结果如下：

```
'printer' comes before 'Screen'
```

如果该代码在区域为法语 / 加拿大的系统下运行，而且输入 `imprimante` 和 `écran`，则消息框内的输出如下：

```
'i mp r i ma n t e' c o m e
```

同样，如果在代码中增加 `Option Compare Text` 语句，则这两项以正确的顺序出现——即 `écran` 先于 `imprimante`。此外，由于不区分大小写，比较时应考虑重音字符，例如法语中的 `é`，排序时将它紧排在其标准字符之后，此时是 `e`。

如果输入 `ecran` 和 `écran`，输出将是：

```
'ecran' comes before 'écran'
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`Option Compare` 语句”。

用 “Like” 操作符比较字符串

可以使用 “Like” 操作符比较两个字符串。也可以使用其模式匹配功能。当编写国际化软件时，必须考虑模式匹配功能。当字符范围使用 “Like” 时，指定的模式表示排序顺序范围。例如，用 `Binary` 方法比较字符串时（通过设置缺省值或通过代码中添加 `Option Compare Binary`），字符范围 `[A - C]` 将忽略大写重读 `a` 字符和所有小写字符。只有以 `A`、`B` 和 `C` 开头的字符串才能与之匹配。而在很多语言中却不是这样。例如德语中，上述范围将忽略所有以 `Ä` 开头的字符串。在法语中，则忽略所有以 `À` 开头的字符串。

使用 `Text` 方法比较字符串时，所有重音 `A` 和 `a` 字符将包含在内。然而在区域为法语 / 法国中，由于 `ç` 和 `Ç` 在排序顺序中出现在 `ç` 和 `Ç` 之后，所以以 `ç` 或 `Ç` 开头的字符串不包含在内。

在某些区域里，利用 `[A - Z]` 范围来选定所有以字母字符开头的字符串并不是一种有效的方法。如果在区域为丹麦语 / 丹麦的系统下运行应用程序，用 `Text` 方法比较字符串时，以 `Ø` 和 `ø` 开头的字符串不包含在此范围内。虽然这两个字符是丹麦语字符的一部分，但它们排在 `Z` 之后。因此，应在 `Z` 之后增加这些字母。例如，用 `Option Compare Text` 语句比较字符串时，`Print "øl" Like "[A-Z]*"` 将返回 `False`，但是 `Print "øl" Like "[A-Zø]*"` 将返回 `True`。

用 `StrComp` 函数比较字符串

用 `StrComp` 函数比较字符串很有用。它的返回值表明某一字符串是否小于、等于或大于另一字符串。返回值也与用 `Option Compare` 语句定义的字符串比较方法（`Binary` 或 `Text`）有关。在比较同一字符串时，根据定义的字符串比较方法不同，`StrComp` 的返回值也不同。

详细信息 关于东亚语言字符串比较的详细信息，请参阅本章后面的“`DBCS` 排序顺序和字符串比较”一节。也可参阅《`Visual Basic 6.0` 语言参考手册》的“`StrComp` 函数”。

国际文件的输入 / 输出

Visual Basic 中，当有文件的输入 / 输出操作时，区域也是考虑的重要因素。
`Print #` 和 `Write #` 语句都可用于数据文件输出，但它们有不同的用途。

`Print #`

`Print #` 语句以区域识别的格式将数据输出到文件里，就象数据显示到屏幕上一样。例如，日期输出使用系统 `Short Date` 格式，数字值使用系统小数点。

在 Visual Basic 中，当数据是用 `Print #` 语句写到文件里时，`Input #` 语句不能读取具有区域识别格式的数据。为了能使 Visual Basic 在任何区域里读取不依赖于区域的数据，可使用 `Write #` 语句，而不是 `Print #` 语句。

`Write #`

与 `Print #` 语句类似，`Write #` 语句也是以固定的格式将数据输出到文件，并保证这些数据在任何区域里都能用 `Input #` 语句从文件中读取。例如，使用通用日期格式将日期输出到文件里，使用句号作为小数点将数字值输出到文件里。在下面的代码示例中，使用 `Write #` 语句将日期和数字值写到文件里。然后再打开该文件，使用 `Input #` 语句读取其内容，同时将结果输出到立即窗口中。`Long Date` 信息从系统区域中获取：

```

Dim MyDate As Date, NewDate As Date
Dim MyNumber As Variant
    MyDate = #8/2/67#
    MyNumber = 123.45
Open "Testfile" for Output As #1
    Write #1, MyDate, MyNumber
Close #1

Open "Testfile" for Input As #1
    Input #1, MyDate, MyNumber
    NewDate = Format(Mydate, "Long Date")
Debug.Print NewDate, MyNumber
Close #1

```

在英语 / 美国的区域中运行该代码，立即窗口出现如下信息：

```
Wednesday, August 02, 1967      123.45
```

在法语 / 法国的区域中运行该代码，立即窗口出现如下信息：

```
mercredi 2 août 1967            123,45
```

在这两个区域中，输出都是准确的-通过使用 Write # 和 Input # 语句，信息得到了恰当的存储和获取。

详细信息 有关文件处理的背景信息，请参阅第十四章“处理驱动器、文件夹和文件”中的“使用文件”。也可参阅《Visual Basic 6.0 语言参考手册》的“Print # 语句”或“Write # 语句”。

基于日期的区域识别的 SQL 查询

正如“用 Visual Basic 编写国际化代码”中说明的那样，不同的国家/地区具有不同的日期格式。当应用程序执行比较两个日期的操作时，为了可靠，日期必须以唯一的格式存储，而忽略其区域。在 Visual Basic 中，数据库引擎存储一日期 / 时间值作为 DateSerial 值，它由 8 位浮点数表示，日期为整数部分，时间为分数部分。这种表示方法完全不依赖于区域，而且可使用国际日期 / 时间格式执行日期 / 时间比较操作。

结构化查询语言 (SQL) 是 ANSI 标准，Visual Basic 也遵从该标准。存入表和数据库中的日期，使用的是英语 / 美国格式(月 / 日 / 年)。Microsoft Jet 数据库引擎也采用这种格式。如果使用非美国日期格式，用这些字段查寻时可能返回错误的记录或没有记录返回。

该约束同样适用于 Filter 属性，Recordset 对象中的 FindFirst, FindNext, FindPrevious 和 FindLast 方法，以及 SQL 语句中的 WHERE 子句。

使用 DateSerial 和 DateValue

这里有两个函数可以解决 SQL 标准的限制。避免代码中使用日期 / 时间，代之以使用 DateValue 或 DateSerial 函数生成所需日期。DateValue 函数用系统的 Short Date 设置解释所提供的字符串，DateSerial 函数使用一系列可在任何区域中运行的参数。如果 SQL 查询或 Filter 属性中使用日期 / 时间，则日期和时间只能使用英语 / 美国格式，除此之外别无选择。

以下示例说明了如何执行基于日期的查询。第一个例子使用非美国日期格式。由于日期表达式中出现语法错误，返回的 **Recordset** 是空的：

```
Dim mydb As Database
```

```
Dim myds As Recordset
```

```
Set mydb = OpenDatabase("MyDatabase.mdb")
```

'包含日期 / 时间字段的表。

```
Set myds = mydb.OpenRecordset("MyTable,dbopenDynaset")
```

'日期格式是日/月/年。

```
myds.FindFirst "DateFiled > #30/03/97#"
```

'数据控件和 mydb 相联。

```
Data1.Recordset.Filter = "DateFiled = #30/03/97#"
```

```
mydb.Close
```

```
myds.Close
```

然而，下面的例子由于日期采用适当的格式，因此在任何区域里执行都正确：

```
Dim mydb As Database
```

```
Dim myds As Recordset
```

```
Set mydb = OpenDatabase("MyDatabase.mdb")
```

'包含日期 / 时间字段的表。

```
Set myds = mydb.OpenRecordset("MyTable, dbopenDynaset")
```

```
myds.FindFirst "DateFiled > #03/30/97#"      '日期格式是  
                                              '月/日/年。
```

'数据控件和 mydb 相联。

```
Data1.Recordset.Filter = "DateFiled = _  
DateValue(""" & DateString & """)"
```

```
mydb.Close
```

```
myds.Close
```

双字节字符集 (DBCS) 的特殊问题

建立双字节字符集 (DBCS) 是用来处理使用象形文字字符的东亚语言，它需要使用的字符超过 ANSI 支持的 256。DBCS 中字符是双字节、16 位符号。虽然为东亚语言定义的字符很少，但 16 位符号可代表 65,536 个字符。例如，现在定义的日语字符集大约有 12,000 个字符。

在使用 DBCS 的区域包括中国、日本、和朝鲜，字符集里都包含单字节和双字节字符。对每个国家/地区，使用的单字节字符都符合 8 位符号标准，并且与 ASCII 字符集相一致。在单字节字符集 (SBCS) 里，有些代码的范

围在 DBCS 字符里称为前导字节。由一前导字符和一后续字符组成的一对连续字符代表一双字节字符。使用前导字符的代码范围根据区域而定。

注意 DBCS 是有别于 Unicode 的另一类字符。由于 Visual Basic 内部表示所有字符的格式是 Unicode，所以需要时，ANSI 和 DBCS 字符会自动转换成 Unicode 字符或者 Unicode 字符自动转换成 ANSI 或 DBCS 字符。也可人工完成 Unicode 和 ANSI/DBCS 之间的转换。有关不同字符集间转换的详细信息，请参阅“DBCS 字符串操作函数”一节。

当用 Visual Basic 开发允许使用 DBCS 的应用程序时，应考虑：

- Unicode、ANSI 和 DBCS 之间的差别。
- DBCS 排序顺序和字符串比较。
- DBCS 字符串操作函数。
- DBCS 字符串转换。
- 在 DBCS 环境下如何正确显示和打印字体。
- 如何处理包括双字节字符的文件。
- DBCS 标识符。
- 允许 DBCS 的事件。
- 如何调用 Windows API。

提示 无论应用程序是否运行在使用 DBCS 的区域下，开发允许 DBCS 的应用程序都是很好的方法。该方法帮助用户开发灵活、便捷和真正的国际应用程序。在使用单一单字符集 (SBCS) 的环境下，允许 DBCS 功能并不干扰应用程序的运行，并且由于 DBCS 和 SBCS 内部使用 Unicode，所

以应用程序的大小不会增加。

详细信息 有关使用 DBCS 访问键和快捷键的限制，请参阅“设计国际识别的用户界面”。

ANSI、DBCS 和 Unicode 的定义

Visual Basic 使用 Unicode 存储和操作字符串。Unicode 是一种用两个字节表示一个字符的字符集。另外一些程序，如 Windows 95 API，使用 ANSI (American National Standards Institute) 或 DBCS 存储和操作字符串。当从 Visual Basic 移出字符串时，会遇到 Unicode 和 ANSI / DBCS 之间的差别。下表列出了不同环境下 ANSI、DBCS 和 Unicode 字符集。

环境	使用的字符集
Visual Basic	Unicode
32-bit 对象库	Unicode
16-bit 对象库	ANSI 和 DBCS
Windows NT API	Unicode
Automation in Windows NT	Unicode
Windows 95 API	ANSI 和 DBCS
Automation in Windows 95	Unicode

ANSI

ANSI 是个人计算机使用得最普遍的字符集。由于 ANSI 标准使用单一字节表示每个字符，因此最多只能有 256 个字符和标点符号代码。虽然对英语来说已经足够了，但不能完全支持其它语言。

DBCS

发行在亚洲大部分地区的 Microsoft Windows 系统使用 DBCS。它支持很多不同的东亚语言字母，如汉语、日语和朝鲜语。DBCS 使用数字 0-128 表示 ASCII 字符集。其它大于 128 的数字作为前导字节字符，它并不是真正的字符，只是简单的表明下一个字符属于非拉丁字符集。在 DBCS 中，ASCII 字符的长度是一个字节，而日语、朝鲜语和其它东亚字符的长度是 2 个字节。

Unicode

Unicode 是用两个字节表示每个字符的字符编码方案。国际标准组织 (ISO) 几乎为每种语言的每个字符和符号在 0 到 65,535 ($2^{16} - 1$) 范围内定义了一个数字（再加上为将来发展保留的一些空余空间）。在所有 32 位版本的 Windows 中，部件对象模型 (COM) 都使用 Unicode，它是 OLE 和 ActiveX 技术的基础。Windows NT 全部支持 Unicode。虽然 Unicode 和

DBCS 都是双字节字符，但它们的编码方案完全不同。

字符代码示例

图 16.4 显示了每一个字符集中的一个字符代码示例。注意双字节字符每一个字节的代码不同。

A	ANSI 字符"A"	&H41	
A	Unicode 字符"A"	&H41	&H00
Ａ	DBCS 字符，表示一个日本的全角"A"	&H82	&H60
Ａ	Unicode 全角"A"	&H21	&HFF

图 16.4 ANSI、Unicode 和 DBCS 中 "A" 的字符代码

DBCS 排序顺序和字符串比较

对 DBCS 字符串使用 “Option Compare Text” 语句时，由于该语句产生一些特殊动作，因此必须清楚 DBCS 文本排序和比较的内容。当使用

Option Compare Binary 语句时，根据字符内部二进制码的排序顺序进行比较。当使用 Option Compare Text 语句时，根据由用户的系统区域决定的、不区分大小写的文本排序顺序进行比较。

英语中“不区分大小写”的意思是忽略大写和小写的差别。在 DBCS 环境中，不区分大小写有另外的意思。例如，在有些 DBCS 字符集（包括日语、繁体汉语和朝鲜语）中，对同一字符有两种表示法，半角字符和全角字符。例如，单字节“A”和双字节“A”。虽然它们显示时字符宽度不同，但 Option Compare Text 将它们作为同一字符处理。每个 DBCS 字符集都有类似的规则。

比较字符串时应小心。即使是使用 Like 或 StrComp 比较时认为相等的两个字符串，该字符串中确切的字符可能是不同的，字符串长度也可能不同。

详细信息 有关使用 Option Compare 语句比较字符串的一般信息，请参阅“国际排序顺序和字符串比较”一节。

DBCS 字符串操作函数

虽然双字节字符由一个前导字节和一个后续字节组成，并存储在两个连续的字节里，但是在任何涉及到字符和字符串的操作时，它应作为一个单元处理。一些字符串操作函数可以处理基于字符的所有字符串，包括 DBCS 字符。

这些函数有 ANSI/DBCS 版本、二进制版本或者 Unicode 版本，如下表所示。根据操作字符串的目的，使用适当的函数。

下表中函数的“B”版本尤其适合于使用二进制数据的字符串，“W”版本适合于使用 Unicode 的字符串。

函数	描述
Asc	返回字符串第一个字符的 ANSI 或 DBCS 字符代码
AscB	返回给定包含二进制数据的字符串第一个字节的值
AscW	返回字符串第一个字符的 Unicode 字符代码
Chr	返回包含指定 ANSI 或 DBCS 字符代码的字符串
ChrB	返回包含指定字节的二进制字符串
ChrW	返回包含指定 Unicode 字符代码的字符串
Input	从文件中返回指定数目的 ANSI 或 DBCS 字符
InputB	从文件中返回指定数目的字节
InStr	返回一字符串在另一字符串中第一次出现的位置
InStrB	返回一字节在另一二进制字符串中第一次出现的位置
Left, Right	返回字符串左边或右边指定数目的字符
LeftB, RightB	返回二进制字符串左边或右边指定数目的字节
Len	返回以字符数为单位的字符串长度
LenB	返回以字节数为单位的字符串长度
Mid	从字符串中返回指定数目的字符
MidB	从二进制字符串中返回指定数目的字节

表中没有 "B" 或 "W" 的函数能正确处理 DBCS 或 ANSI 字符。除上述函数以外，String 函数也能处理 DBCS 字符。这就是说所有函数把 DBCS 字符作为一个字符处理，即使该字符由两个字节组成。

当处理 SBCS 和 DBCS 字符时，这些函数的过程是不同的。例如，在 Visual Basic 中使用 Mid 函数从字符串中返回指定数目的字符。在使用 DBCS 的区域中，字符数和字节数不一定相同，Mid 函数只返回字符数而不是字节数。

大多数情况下，由于这些函数能够正确处理 ANSI、DBCS 和 Unicode 字符串，因此处理字符串数据时，都使用基于字符的函数。

基于字节的字符串操作函数，例如 LenB 和 LeftB 用于处理二进制数据的字符串数据。当向 String 变量存储字符或从 String 变量读取字符时，Visual Basic 会自动完成 Unicode 和 ANSI 字符间的转换。当处理二进制数据时，应使用 Byte 数组而不是 String 变量和基于字节的字符串操作函数。

详细信息 关于相应的函数，请参阅《Visual Basic 6.0 语言参考手册》。

处理二进制数据的字符串时，使用下列代码将字符串中的字符映射到 Byte 数组中：

```
Dim MyByteString() As Byte
'将字符串映射到一 Byte 数组中。
MyByteString = "ABC"
'显示二进制数据。
For i = LBound(MyByteString) to UBound(MyByteString)
    Print Right(" " + Hex(MyByteString(i)),2) + " ,";
Next
Print
```

DBCS 字符串转换

Visual Basic 提供了一些对 DBCS 字符进行字符串转换的函数：StrConv、UCase 和 LCase。

StrConv 函数

StrConv 函数的全局选项是将大写转换成小写或将小写转换成大写。此外，该函数还有一些 DBCS 特定的选项。例如，通过指定该函数第二个参数中的 vbWide，可将半角字符转换成全角字符。也可用此函数将一种字符类型转换成另一种字符类型，如日语中的平假名转换成片假名。如果与系统的 LocaleID 不同，StrConv 允许您为该字符串指定一个 LocaleID。

也可使用 StrConv 函数将 Unicode 字符转换成 ANSI/DBCS 字符或将 ANSI/DBCS 字符转换成 Unicode 字符。通常，Visual Basic 中的字符串由 Unicode 字符组成。当需要处理 ANSI/DBCS 字符串时（例如，在将字符串写入文件之前，计算字符串的字节数），可使用 StrConv 函数的这一功能。

全角字符的大小写转换

可使用 StrConv 函数的 vbUpperCase 或 vbLowerCase 选项，或者使用 UCase 或 LCase 函数进行字母的大小写转换。使用这些函数完成 DBCS 包括 ANSI 字符的英语全角字符的大小写转换。

DBCS 环境下字体、显示和打印的考虑因素

使用为 SBCS 字符设计的字体时，Windows 的 DBCS 版本不会正确显示 DBCS 字符。使用 Visual Basic 英语版本或其它 SBCS 语言版本开发允许 DBCS 的应用程序时，需要改变 Font 对象的 Name 属性。Name 属性决定控件中显示文本的字体、运行时图片中的字体和打印操作中的字体。该属性的缺省值是 Visual Basic 英语版本中的 MS Sans Serif。为在 DBCS 环境中正确显示文本，将设置值改成应用程序在 DBCS 环境下运行时适当的字体。也可能需要通过改变 Font 对象的 Size 属性来改变字体大小。通常，在大字数东亚平台上，应用程序中的文本最好以 9 磅字体显示，而欧洲平台上典型的是 8 磅字体。

同样，通过应用程序打印 DBCS 字符时，也应考虑这些因素。

如何避免改变字体设定值

如果没有允许 DBCS 的字体，或者不知道那一种字体适合于目标平台，可用一些选项来解决这些字体问题。

在 Windows 的繁体汉语、简体汉语和朝鲜语版本中，有一称为字体关联的系统功能。例如，朝鲜语 Windows 中，字体关联自动地将应用程序中的英语字体映射成朝鲜语字体。因此，即使是应用程序使用英语字体，仍然能够显示朝鲜语字符。关联的字体由运行平台的系统注册表中的 \HKEY_LOCAL_MACHINE\System\

`CurrentControlSet\control\fontassoc\Associated DefaultFonts` 的设置决定。由于有系统支持的字体关联，可以不用改变任何字体设置，而在汉语或朝鲜语平台上运行英语应用程序。其它平台如日语 Windows 没有字体关联。另一选项是使用 `System` 字体或 `FixedSys` 字体。每个平台都可使用这些字体。注意 `System` 字体和 `FixedSys` 字体大小不能改变。如果在设计时设置的任何一种字体大小（用 `Font` 对象的 `Size` 属性）与使用机器上的字体大小不匹配，则忽略设置值、截断显示文本。

运行时改变字体的操作方法

虽然有上述选项，这些解决方法仍有限制。下面的示例是应用程序在运行时，使用全局解决方法改变其中的字体。下面的代码可在 Windows 的任何语言版本上运行，它决定驻留在系统中的字体，应用程序将在该系统下运行，将适当的字体应用到参数中指定的 `Font` 对象。

```
Private Const DEFAULT_CHARSET = 1
Private Const SYMBOL_CHARSET = 2
Private Const SHIFTJIS_CHARSET = 128
Private Const HANGEUL_CHARSET = 129
Private Const CHINESEBIG5_CHARSET = 136
Private Const CHINESESIMPLIFIED_CHARSET = 134
Private Declare Function GetUserDefaultLCID Lib "kernel32" () As Long
```



```
Public Sub SetProperFont(obj As Object)
    On Error GoTo ErrorSetProperFont
    Select Case GetUserDefaultLCID
    Case &H404 ' 繁体中文
        obj.Charset = CHINESEBIG5_CHARSET
        obj.Name = ChrW(&H65B0) + ChrW(&H7D30) + ChrW(&H660E) _
            + ChrW(&H9AD4) 'New Ming-Li
        obj.Size = 9
    Case &H411 ' 日语
        obj.Charset = SHIFTJIS_CHARSET
        obj.Name = ChrW(&HFF2D) + ChrW(&HFF33) + ChrW(&H20) + _
            ChrW(&HFF30) + ChrW(&H30B4) + ChrW(&H30B7) + ChrW(&H30C3) + _
            ChrW(&H30AF)
        obj.Size = 9
    Case &H412 '朝鲜 UserLCID
        obj.Charset = HANGEUL_CHARSET
        obj.Name = ChrW(&HAD74) + ChrW(&HB9BC)
        obj.Size = 9
    Case &H804 ' 简体中文
        obj.Charset = CHINESESIMPLIFIED_CHARSET
        obj.Name = ChrW(&H5B8B) + ChrW(&H4F53)
        obj.Size = 9
    End Select
End Sub
```

```

Case Else      ' 其他国家/地区
    obj.Charset = DEFAULT_CHARSET
    obj.Name = ""      ' 获得缺省 UI 字体。
    obj.Size = 8
End Select
Exit Sub
ErrorSetProperFont:
    Err.Number = Err
End Sub

```

也可更改这个简单的示例代码将字体应用到其它字体设置上，如打印选项。

处理使用双字节字符的文件

在使用 DBCS 的区域里，文件可能包含双字节字符和单字节字符。由于 DBCS 字符由两个字节表示，Visual Basic 代码必须避免把它们分开。下面的示例，假定 Testfile 是一个包含 DBCS 字符的文本文件。

'打开文件，准备输入。

```
Open "TESTFILE" For Input As #1
```

'读取文件中的所有字符。

```
Do While Not EOF(1)
```

MyChar = Input(1, #1) '读取一字符。

'使用 Mychar 执行操作。

Loop

Close #1 '关闭文件。

当从二进制文件读取固定长度的字节时，使用 Byte 数组而不是 String 变量，以阻止 Visual Basic 中 ANSI 到 Unicode 的转换。

Dim MyByteString(0 to 4) As Byte

Get #1,, MyByteString

使用 String 变量，用 Input 或 InputB 从二进制文件中读取字节时，将出现 Unicode 字符转换，结果是错误的。

记住文件名称和目录也可能包含 DBCS 字符。

详细信息 有关文件处理的背景信息，请参阅第十四章“处理驱动器、文件夹和文件”中的“使用文件”。有关 Byte 数据类型的信息，请参阅第五章“编程基础”中的“数据类型”。

DBCS 环境的标识符

在下面的标识符中不支持 DBCS 字符：

- 公共过程名
- 公共变量

- 公共常数
- 工程名（当在“工程属性”对话框中指定时）
- 类名（类模块的 `Name` 属性、用户控件、属性页或用户文档）

允许 DBCS 的 KeyPress 事件

`KeyPress` 事件能够将双字节字符代码作为事件处理。`keyascii` 参数的高位字节代表双字节字符的前导字节，低位字节代表后续字节。

在下面的示例中，可将 `KeyPress` 事件传递给文本框，而不管输入的字符是单字节还是双字节。

```
Sub Text1_KeyPress (KeyAscii As Integer)
```

```
    Mychar = Chr(KeyAscii)
```

```
    '使用 Mychar 执行操作。
```

```
End Sub
```

详细信息 请参阅《Visual Basic 6.0 语言参考手册》的“`KeyPress` 事件”。

调用 Windows API 函数

很多 Windows API 和 DLL 函数的返回值是字节大小。返回值代表返回字符串的长度。Visual Basic 将返回的字符串转换成 Unicode，即使是返回值仍然代表 ANSI 和 DBCS 字符串的长度。因此不能使用返回长度作为字

字符串长度。下面的代码得到正确的返回的字符串：

```
buffer = String(145, Chr(" "))  
ret = GetPrivateProfileString(section, _  
entry, default, buffer, Len(buffer)-1, filename)  
retstring = Left(buffer, Instr(buffer, Chr(0))-1))
```

详细信息 有关详细信息，请参阅专业版和企业版的《部件工具指南》里“访问 DLL 和 Windows API”中的“访问 Microsoft Windows API”一节。

Visual Basic 的双向特征

Visual Basic 是允许双向（也被称为 "BiDi"）的，“双向”是一个一般术语，是指一个软件产品支持阿拉伯语或其它的从右向左书写的语言。更确切地说，“双向”是软件产品的一种能力，既能操作和显示从左向右书写的语言的文本，也能操作和显示从右向左书写的语言的文本。例如，如果一句话中既包含有英文单词，又包含有阿拉伯文的单词，那么显示这句话就需要“双向”能力。

Microsoft Visual Basic 提供了创建和运行具有完全双向语言能力的 Windows 应用程序的功能。但是，仅当 Microsoft Visual Basic 被安装在象 Arabic Microsoft Windows 95 这样的双向 32-位 Microsoft Windows 环境中时，这些功能才可使用。对于其它的双向 32-位 Microsoft Windows 环境同样可

用。

RightToLeft 属性已经被添加到窗体、控件和其它的 Visual Basic 对象中，以便能够提供一种简单机制，来创建具有双向特征的对象。尽管在每一个 Microsoft Visual Basic 的安装程序中都包含有 RightToLeft，但是仅当 Microsoft Visual Basic 被安装在一个双向 32-位 Microsoft Windows 环境中时，它才可用。

有关双向特征的文档描述了所有有关 Microsoft Visual Basic 的双向特征的问题。您可以在 Visual Basic 的目录表中参考的附加信息中找到该信息。单击该主题标题下的“请参阅”直接到概述主题。

为与 Microsoft Visual Basic 4.0 保持兼容，在 Microsoft Visual Basic 中包含有两个版本的 32-位 Grid 控件（Grid32.ocx），但是没有安装它们。这两个版本都位于产品媒介安装路径的 \Tools 文件夹下。标准版位于 \Controls 子文件夹，双向版位于 \Controls\Bidi 子文件夹。

第十七章 发布应用程序

在创建 Visual Basic 应用程序后，您可能希望将该程序发布给其他人。您可以将使用 Visual Basic 创建的任何应用程序自由地发布给使用 Microsoft Windows 的任何人。可以通过磁盘、CD、网络、或者 intranet 以及 Internet 这些途径来发布应用程序。

在发布应用程序时，必须经过下述两个步骤：

- 打包——必须将应用程序文件打包为一个或多个可以部署到选定位置的 .cab 文件，对于某些类型的软件包，您还必须为其创建安装程序。.cab 文件是一种经过压缩的、很适合通过磁盘或 Internet 进行发布的文件。
- 部署——您必须将打好包的应用程序放置到适当的位置，以便用户可以从该位置安装应用程序。这意味着将软件包复制到软盘上或复制到本地或网络驱动器上，也可以将该软件包部署到一个 Web 站点。

您可以使用两个工具来打包和发布应用程序：打包和部署向导 (Package and Deployment Wizard) (以前的安装向导)，或者是随 Visual Basic 安装软件提供的安装工具包。打包和部署向导通过提供有关如何配置 .cab 文件的选项，使发布应用程序所包含的许多步骤得以自动进行。安装工具包则让您自定义一些在安装过程中发生的事情。

目 录

- 打包和部署向导
- 使用向导为应用程序打包
- 使用向导部署应用程序
- 管理向导脚本
- 安装工具包
- 手工编辑 Setup.lst 文件
- 手工创建发布媒体
- 与安装工具包一起使用打包和部署向导
- 测试安装程序
- 允许用户删除您的应用程序
- 部署本地化的 ActiveX 控件

打包和部署向导

Visual Basic 的打包和部署向导可以帮助您创建应用程序的 .cab 文件，将这些文件组合成一个包含安装所需的所有信息的单元或软件包，并将这些软件包交付给最终用户。您可以使用 Visual Basic 的打包和部署向导来创建要发布到软盘、CD、本地或网络驱动器、以及 Web 上的软件包。打包和部署向导使在创建和部署这些文件时所包含的很多工作得以自动进行。

打包和部署向导提供了三个选项

- “打包”选项帮助您将一个工程的文件打包为一个可以部署的 .cab 文件，而且在某些情况下还要创建一个安装程序来安装该 .cab 文件。向导确定需要打包的文件，并引导您作出所必需的选择，以便为您的工程创建一个或多个 .cab 文件。
- “部署”选项帮助您将打好包的应用程序传送到适当的发布媒体，例如软盘、网络共享、或 Web 站点。
- “管理脚本”选项让您可以查看和操作以前使用打包和部署向导时所保存的脚本。每次使用该向导时，都会保存一个包含您所做的所有选择的脚本。在以后的过程中，如果您想使用相似的设置值以及做出与原来相同的选择，就可以重新使用这些脚本。

详细信息 有关将工程打包的详细信息，请参阅本章稍后的“使用向导为应用程序打包”。有关将应用程序部署到 Web 或另外的目标的详细信息，请参阅本章稍后的“使用向导部署应用程序”。有关如何创建和使用脚本的详细信息，请参阅“管理向导脚本”。

打包和部署向导与安装工具包的比较

打包和部署向导能够帮助您为自己的 Visual Basic 应用程序创建和发布专业的安装程序。除了为应用程序创建 .cab 文件之外，该向导还通过编译随 Visual Basic 一起安装的安装工具包工程来为应用程序创建安装程序。这个安装程序被称为 `setup1.exe`。

在多数情形下，打包和部署向导是创建和发布应用程序的 `setup1.exe` 程序的最佳途径。然而，如果要在应用程序的安装程序中使用打包和部署向导中没有提供的特性，则可以通过修改安装工具包工程来实现。与其它的 Visual Basic 工程一样，该工程的窗体、代码、及其功能也可以被修改或增强。

注意 打包和部署向导和安装工具包只能为 Visual Basic 应用程序创建安装程序和发布媒体。如果要为其它基于 Windows 的应用程序创建安装程序，请使用与相应的开发产品一起提供的安装工具包或在 Windows SDK 中提供的安装工具包。

详细信息 有关使用该工具包工程将应用程序打包和部署的详细信息，请参阅本章稍后的“安装工具包”。

启动打包和部署向导

Visual Basic 的打包和部署向导使您能够轻松地应用程序创建必需的 .cab 文件以及安装程序。与其它向导一样，打包和部署向导将提示您输入相应信息，以便创建您所需要的准确配置。

有三种启动打包和部署向导的方式：

- 可以在 Visual Basic 中将该向导作为一个外接程序运行。如果要将该向导作为外接程序运行，则必须首先在“外接程序管理器”中设置必需的引用来加载该向导。在将该向导作为外接程序使用时，Visual Basic 认为您是要对当前已打开的工程进行操作。如果您希望对另一个工程进行

操作，则要么在启动该外接程序之前打开该工程，要么就应将该向导作为一个独立的部件来使用。

- 可以将该向导作为一个独立的部件在开发环境外部运行。当将该向导作为独立的部件运行时，将会提示您选择希望操作的工程。
- 可以通过命令提示符方式，以静态模式来启动该向导。详细信息请参阅本主题中的“以静态模式运行向导”。

启动向导后，将出现一系列的屏幕提示您输入关于您工程的信息，并让您选择软件包的选项。每个屏幕上都有如何使用该屏幕的解释，包括哪些信息是可选的，以及在您能够进入下一个屏幕之前哪些信息是必须输入的。如果需要关于任意一个屏幕的详细信息，请按下 F1 键或单击“帮助”按钮。

注意 在运行打包和部署向导之前，应保存和编译您的工程。

在大多数情形下，使用打包和部署向导就足以创建一个可供部署的软件包。然而，如果希望进一步自定义您的打包过程，或提供打包和部署向导所不支持的功能，可以修改安装工具包工程。

要从 Visual Basic 中启动打包和部署向导，请按照以下步骤执行：

1. 打开想要使用向导来打包或部署的工程。

注意 如果您正在使用一个工程组，或已加载了多个工程，则在启动该向导之前，请确保当前的工程就是您要打包或部署的工程。

2. 如果必要，请使用“外接程序管理器”来加载打包和部署向导：从“外接程序”菜单中选择“外接程序管理器”，从该列表中选择“打包和部署向导”，然后单击“确定”。

3. 从“外接程序”菜单中选择“打包和部署向导”来启动该向导。
4. 在主屏幕上，选择下述选项之一：
 - 如果想为该工程创建一个标准软件包、Internet 软件包、或从属文件，请单击“打包”。
 - 如果想部署该工程，请单击“部署”。
 - 如果想查看、编辑、或删除脚本，请单击“管理脚本”。

有关这些选项的介绍，请参阅“打包和部署向导”。

5. 请按照向导屏幕进行操作。

要将打包和部署向导作为一个独立的部件启动，请按照以下步骤执行：

1. 如果想要打包的工程已被打开，请保存该工程并关闭 Visual Basic。
2. 单击“启动”按钮，然后从 Visual Basic 子菜单中单击“打包和部署向导”。
3. 在初始屏幕的“工程”列表中，选择您想要打包的工程。
注意 如果您的工程不在此列表中，可以单击“浏览”。
4. 在主屏幕上，选择下述选项之一：
 - 如果想为该工程创建一个标准软件包、Internet 软件包、或从属文件，请单击“打包”。
 - 如果想部署该工程，请单击“部署”。
 - 如果想查看、编辑、或删除脚本，请单击“管理脚本”。
5. 按照向导屏幕进行操作。

以静态模式运行向导

您可以使用脚本以静态模式来打包和部署工程文件。在静态模式中，向导将在您无需进行选择及对各个屏幕进行操作的情况下运行。向导使用脚本中所包含的设置值来打包和部署您的工程。

如果您要将打包和部署作为批处理的一部分，则静态模式是很有用的。例如，在工程开发的早期，您可以使用打包和部署向导来对工程打包，并将其部署到一个测试位置上。然后您可以创建一个批处理文件，随着该工程的更新而定期执行相同的打包和部署步骤。

要以静态模式进行打包和部署，请按照以下步骤执行：

1. 打开一个 **MS-DOS** 方式。
2. 输入向导的可执行程序名称，`pdcmdln.exe`，后面加上您的 **Visual Basic** 工程的路径和文件名，以及适当的命令行参数，如下面的示例所示：

```
PDCmdLn.exe C:\Project1\Project1.vbp /p "Internet Package"
```

```
/d Deployment1 /l "C:\Project1\Silent Mode.log"
```

注意 通过同时指定 `/p` 和 `/d` 参数，可以在一个单一的静态过程中进行打包和部署两个步骤，如上面的示例所示。否则，请使用 `/p` 或 `/d` 之一。

参数	描述
<code>/p packagingscript</code>	键入 <code>/p</code> ，后面加上原来保存的打包脚本名称，以按照指定的脚本来对工程进行静态打包。

续表

<code>/d deploymentscript</code>	键入 <code>/d</code> ，后面加上原来保存的部署脚本名称，以按照指定的脚本来静态部署该工程。
<code>/l path</code>	指定向导应将所有来自向导的输出，例如错误消息以及成功报告等，存储到文件中，而不是显示在屏幕上。 键入 <code>/l</code> ，后面加上用于存储输出的文件路径和文件名称。如果该文件不存在，向导将创建一个文件。

注意 任何包含空格的文件名或脚本名均要用引号引起来，如上面的示例所示。

详细信息 有关如何使用向导对工程打包的指示，请参阅“使用向导为应用程序打包”。有关如何使用向导来部署工程的指示，请参阅“使用向导为应用程序打包”。有关自定义安装过程的详细信息，请参阅“修改安装工程”。

使用向导为应用程序打包

为应用程序打包是指创建一个软件包的操作，该软件包可以将您的应用程序安装到用户的计算机上。软件包由一个或多个 `.cab` 文件组成，文件中包含了用户安装和运行应用程序所需的被压缩的工程文件和任何其它必需的文件。这些文件可能包括安装程序、辅助的 `.cab` 文件或其它必需的文件。

这些附加的文件根据您创建的软件包类型不同而不同。

您可以创建两种软件包，标准软件包或 Internet 软件包。如果计划通过磁盘、软盘或网络共享来发布应用程序，则应创建一个标准软件包。如果计划通过 intranet 或 Internet 站点来发布，则应创建一个 Internet 软件包。在多数情形下，可以使用 Visual Basic 提供的打包和部署向导来为您的应用程序打包。您可以手工打包应用程序，不过向导提供了很有用的快捷方式，并能自动处理一些在手工打包过程中不得不自己完成的任务。

注意 另外，可以将安装工具包和打包和部署向导一起使用。您可以修改安装工具包工程来自定义自己的安装程序，并添加打包和部署向导所没有提供的特性，然后使用向导来打包和部署应用程序。

除了创建标准和 Internet 软件包之外，还可以使用打包和部署向导的打包部分来创建从属文件。从属文件列出了必须随应用程序的工程文件一起发布的运行时部件。

重点 无论何时创建软件包，您都应确保已在“工程属性”对话框的“生成”选项卡中设置了工程的版本号码。如果您要发布一个现有应用程序的新版本，那么这一点特别重要：如果没有对版本号进行适当的更改，最终用户的计算机可能会认为关键文件不需要更新。

详细信息 有关标准软件包及其内容的详细说明，请参阅本章稍后的“标准软件包”。有关 Internet 软件包及其内容的详细信息，请参阅“Internet 软件包”。有关将两个工具一起使用来创建自定义安装程序的详细信息，请参阅本章稍后的“与安装工具包一起使用打包和部署向导”。有关从属文件内容的说明，请参阅“从属文件”。

允许发布的文件

您可以自由地发布使用 Visual Basic 创建的任何应用程序或部件。除了可执行 (.exe) 文件之外，应用程序可能还需要其它文件，例如 DLL、ActiveX 控件 (.ocx 文件) 或位图 (.bmp 文件)。

您可以合法地发布示例应用程序文件，以及在您的系统上第一次安装 Visual Basic 时复制到 \Visual Studio\Common\Graphics 目录的 \Icons 子目录中的文件。Microsoft 对这些应用程序的商用性或实用性不作任何担保、表明或暗示，而且也不对其使用承担任何义务或责任。

如果您已购买 Visual Basic 的专业版或企业版，也可以发布预先复制到 \Visual Studio\Common\Graphics 和 \Program Files\Common Files\ODBC 子目录中的任何文件。

注意 您也可以发布已购买的其它 ActiveX 控件、.exe 文件以及 DLL。对每个计划发布的文件，请参考其厂商的许可证协议，以确定您是否具有随应用程序发布该文件的权限。

打包过程的全部步骤

不论创建的软件包是什么类型，也不论创建软件包使用的是什么工具，都必须采取几个步骤。

注意 打包和部署向导能自动执行很多步骤。

1. **确定要创建的软件包类型** 您可以为基于 Windows 的、要通过磁盘、CD

或在网络上发布的程序创建一个标准软件包；或者可以为要在 Web 上发布的程序创建一个 Internet 软件包。也可以选择只创建从属文件。

2. **确定需要发布的文件** 向导在创建软件包之前，必须确定应用程序的工程文件以及从属文件。工程文件是包含在工程本身中的文件，例如，.vbp 文件及其内容。从属文件是运行应用程序所需要的运行时文件或部件。从属信息保存在 vb6dep.ini 文件中，或工程中与部件相对应的各种 .dep 文件中。
3. **确定将文件安装到用户机器上的位置** 程序和安装文件通常被安装到 Program Files 目录的某个子目录中，而系统和从属文件通常被安装到 \Windows\System 或 \Winnt\System32 目录中。您的安装程序必须考虑这一点，并确定每个文件要被安装在何处。
4. **创建软件包** 向导引用所有必需的文件来创建软件包，并为该软件包创建安装程序 (setup1.exe)。这一步骤的最终结果是得到一个或多个 .cab 文件以及任何必需的安装文件。
5. **部署软件包** 部署过程包括创建发布媒体，并将所有必需的文件复制到用户可以访问到的位置。有关部署的详细信息，请参阅本章稍后的“使用向导部署应用程序”。

详细信息 有关通用运行时库、安装和从属文件的全部列表，请参阅本章稍后的“需要发布的文件”。有关创建 .dep 文件的详细信息，请参阅本章稍后的“从属文件”。

打包功能

使用打包和部署向导可以轻松地为应用程序创建一个专业的安装程序，或将 Internet 应用程序部署到 Web 上。向导在打包过程中要执行以下步骤：

- **自动包含应用程序的主安装程序 (setup1.exe)** 向导将安装工具包应用程序、Setup1.exe 添加到软件包中。该文件是应用程序的主安装程序。
- **自动创建应用程序的 .cab 文件** 打包和部署向导可以为应用程序创建单个 .cab 文件或多个 .cab 文件。
- **基于脚本的打包过程** 如果想在整个向导操作过程中使用相同的或相似的设置值，您可以从同一个工程的另一个打包过程选择一个脚本。这样可以节省许多时间。此外，您可以使用原来保存的脚本以静态模式对一个工程打包。当打包作为批处理编译过程中的一部分时，这种方式特别有用。
- **可选创建从属文件** 从属文件确定在发布应用程序时必须包含的运行时文件。
- **自动支持数据访问、远程自动化及 DCOM 特性** 向导自动确定您的工程是包含将更改安装过程的功能。例如，如果您包含了某种类型的数据访问、远程自动化或 DCOM 特性，则可能需要在软件包中包括驱动程序或其它文件。向导将检查工程，并显示屏幕，允许您在这些情况下指定适当的选项。
- **共享文件的功能。** 向导允许您将某些文件安装为共享文件 这意味着当卸载时，如果有其它应用程序使用它们，则这些文件不能从系统中被删除。
- **Internet 软件包的其他文件位置** 在 Internet 软件包中，向导允许您指

定是要将从属文件包括在安装程序中，还是从其他的 Web 站点中下载这些从属文件。

- **Internet 软件包的安全性设置** 如果在您的工程中没有使用 IObjectSafety 接口，则打包和部署向导可以让您将应用程序中的部件标记为安全的。
- **为工程中的每个文件自定义目标位置** 大多数文件都有一个缺省的安装位置，这取决于它们是工程文件还是系统文件。如果要把这些文件安装到不同的位置，您可以更改这些位置。

详细信息 有关打包和部署向导特性的详细信息，请参阅本章稍后的“部署功能”。

标准软件包

标准软件包是一种专门用 `setup.exe` 程序安装、而不是从 Web 站点下载 `.cab` 文件而设计的软件包。您可以为基于 Windows 的、要通过磁盘、CD 或网络共享来发布的应用程序创建标准软件包。

当创建标准软件包时，必须在创建软件包之前仔细考虑计划使用的发布方法。如果计划使用软盘来发布，那么通常需要创建多个可以分别放置在若干张软盘上的 `.cab` 文件，而不能创建一个大的 `.cab` 文件。打包和部署向导中有一个选项让您指定是要创建一个还是多个 `.cab` 文件，以及要使用的 `.cab` 大小（1.44 MB、1.2 MB 等等）。如果您选择要创建多个 `.cab` 文件，向导将把应用程序文件分成若干个不超过指定大小的文件组。

重点 即使计划要发布到软盘上的应用程序小到当打包为一个大的 .cab 文件时也足够放置到一张磁盘上，您也仍然应该选择“多个 .cab 文件”选项，以便您在随后的向导操作中可以访问软盘的部署过程。而在这里，只创建了一个 .cab 文件。

如果计划部署到网络或本地共享目录、CD 或 Web 站点上，则既可以创建一个大的 .cab 文件，也可以创建多个较小的 .cab 文件。

标准软件包的必要部分

有若干个文件将始终作为标准软件包的一部分。这些文件包括：

- **setup.exe 文件** Setup.exe 是一个预安装可执行程序。安装过程中第一个在用户机器上运行的程序就是 Setup.exe，该程序执行在主安装之前必须进行的必要处理。
- **setup1.exe 文件** Setup1.exe 是应用程序的主安装程序。
- **所有必需的支持文件** 支持文件存储在 \Support 子目录，位于创建该软件包的目录的下一层。除了 setup.exe 和 setup1.exe 文件之外，该目录还包含用于自定义应用程序的 .cab 文件所需的文件，以备用户的需要。
- **应用程序的 .cab 文件** Internet 应用程序和基于 Windows 的应用程序在发布前都将被打包到 .cab 文件之中。.cab 文件取代了 Visual Basic 原来版本中的一长列被压缩的应用程序文件。现在所有这些被压缩的文件都包含在 .cab 文件中。可以为应用程序创建一个单一的 .cab 文件，也可以为软盘发布方式创建多个 .cab 文件。

注意 如果要在双向的 (BiDi) 操作系统上运行应用程序，则需要手工在打包和部署向导所创建的 Setup.lst 中包括 vbame.dll 文件。要实现这个操作，可以在运行打包和部署向导时，在“包含文件”屏幕上通过直接编辑 Setup.lst 来添加该文件；或者在 vb6dep.ini 文件中为 vbame.dll 添加一项，以便不论何时运行打包和部署向导，该文件都将被自动添加到 Setup.lst 中。

数据访问特性

如果应用程序使用了 Visual Basic 的数据访问技术之一，例如数据访问对象 (DAO)、ActiveX 数据对象 (ADO) 或远程数据对象 (RDO)，则打包和部署向导打包过程中还将执行两个附加步骤：

- 如果应用程序使用了 ADO、OLEDB 或 ODBC 部件，向导将自动添加一个名为 mdac_type.exe 的文件到要包含软件包的文件列表中。Mdac_type.exe 是一个自解压的可执行程序，它将安装您的数据访问技术所需的所有必需部件。
- 如果应用程序包括了 DAO 特性，向导将提示您选择适当的数据访问选项。您可以选择适当的方法，基于 ISAM、ODBCDirect、经由 Jet 的 ODBC 等。

远程自动化及 DCOM 特性

如果应用程序使用了远程代码部件（以前称为 OLE 服务器），则需要为应

用程序创建两个软件包：一个客户安装程序，一个服务器安装程序。您可以使用打包和部署向导来对该应用程序打包，只需简单地对同一个工程组运行两次向导，一次在客户工程上，一次在服务器工程上。

不论是为客户软件包，还是为服务器软件包打包之前，都必须确保已经为该工程创建了必需的远程支持 (.vbr) 文件，并将这些文件放置在与该工程的 .vbp 文件所处的相同目录中。

要创建对远程自动化或 DCOM 的支持，请按照以下步骤执行：

1. 在 Visual Basic 中打开工程组，选择要作为服务器的工程。
2. 从“工程”菜单选择“工程属性”。选择“工程属性”对话框中的“部件”选项卡，并复选“远程服务器文件”选项。

当在选择了这些选项的情况下编译工程时，将自动创建 .vbr 文件。

注册表文件

如果工程引用了任何 .reg 或 .vbl 文件，您将在向导中看到一个附加的屏幕，在该屏幕中可以指定如何处理这些注册表信息。您可以选择简单地将这些注册表文件复制到最终用户的计算机中，或者让系统将信息存储到注册表中，并自动将其注册到最终用户的计算机上。

缺少文件或文件已过时

在使用向导的过程中，如果找不到应用程序所需的任何文件，或者有任何文件缺少从属信息或从属信息已过时，则可能会出现一系列的对话框。您

可以选择不要部件的从属信息继续进行，也可以给出所缺少文件的位置，或永久性地将一个文件标记为不需要任何从属信息。

Internet 软件包

Internet 软件包是基于 .cab 的安装程序，它是专为从 Web 站点下载而设计的。Internet Explorer 使用一个称为“Internet 部件下载”的过程来安装 Internet 应用程序。打包和部署向导自动包括其所创建的软件包所需的信息。

有多种类型的 Visual Basic 应用程序或部件可以被打包，以供 Internet 部署，其中包括：

- 在 Web 页上显示的 ActiveX 控件（.ocx 文件）。
- 用于在客户或 Web 服务器上运行的 ActiveX .exe 或 .dll 文件。
- 代替 Web 页显示的 ActiveX 文档。
- DHTML 应用程序，基于客户的、通过使用动态的 HTML 来链接 HTML 页与 Visual Basic 代码的应用程序。
- IIS 应用程序，基于服务器的、链接 HTML 页与 webclass 对象的应用程序。该 webclass 截取来自浏览器的服务器请求，并执行 Visual Basic 代码来响应。

详细信息 有关 Internet 部件下载过程的众多信息，请参阅《部件工具指南》中《建立 Internet 应用程序》中的“下载 ActiveX 部件”。

Internet 软件包的部分

有若干个文件将始终作为 Internet 软件包的一部分。这些文件包括：

- **应用程序的主要 .cab 文件** Internet 软件包的主要 .cab 文件用作应用程序的安装程序。主要 .cab 文件包括工程部件，例如应用程序的可执行文件或 DLL 或控件的 .ocx 文件、一个引用辅助 .cab 文件以及包含安全性和注册表信息的 .inf 文件，以及不属于辅助 .cab 文件的所有必需的从属文件。
- **所有必需的支持文件** Internet 应用程序的支持文件包括 HTML 文件、Active Server Pages (.asp) 文件、各种格式的图形文件或应用程序运行时必需访问的其它文件。
- **应用程序的任何辅助 .cab 文件** 除了工程文件之外，应用程序通常会引用若干个运行时部件，例如 Visual Basic 运行时 DLL、单个的 ActiveX 控件以及数据访问对象。如果这些部件可以联机从预包装的 .cab 文件中获得，则可以在主要 .cab 文件中引用这些 .cab 文件，而不必自己来传送这些文件。

辅助 .cab 文件提供了一个有效的途径，可以确保用户使用的是该部件的最新版本。如果辅助 .cab 文件中部件的较新版本已经可以从外部 Web 站点上获得，则下载您应用程序的用户将自动收到该部件的更新版本。

注意 如果不能或不愿意让应用程序的安装程序需要与 Internet 的连接，则可以将辅助 .cab 文件放置在您的 intranet 服务器上。intranet 服务器通常提供了更快的下载，并允许用户从安全的网络下载。

Internet 部件下载是如何工作的

在为供下载的 Internet 应用程序或部件打包之后，您就可以将其部署到 Web 服务器上的一个特定位置上了，用户可以从该位置访问这些应用程序或部件。通常，您的软件包被作为一个现存 Web 页面的一部分来引用，也就是说，您的控件或其它部件宿驻在一个 Web 页面上。

当用户访问宿驻您软件包的 Web 页面时，系统将该软件包下载到用户的计算机上。该软件包在确认安全性、打开、注册、安装之后被激活。所有这些都发生在后台，并由浏览器控制。

打包和部署向导在上面所描述的处理中扮演两个角色：

1. 向导将部件及其所关联的文件打包到一个压缩的 (.cab) 文件中，供浏览器下载部件时使用。打包和部署向导确定运行工程需要哪些文件，然后收集这些文件，将它们压缩为一个 .cab 文件，并生成指向您的部件的 HTML。
2. 向导将打好包的文件部署到您选择的 Web 服务器位置上。有关部署 Internet 部件下载软件包的详细信息，请参阅本章稍后的“使用向导部署应用程序”。

安全性问题

在准备好供下载的 Internet 应用程序和部件后，必须将它们打包为一个可通过浏览器发布给用户的文件。此外，还必须执行一些预防步骤，以确保

您的应用程序不会损害用户的计算机。这些步骤包括：

- 对部件进行数字署名，以便用户能验证部件的内容，并知道您是该软件的开发者。
- 设置安全性级别来保证您的部件不会损害用户的计算机或毁坏其数据。
- 为任何需要许可证的部件安排许可。当添加一个 **ActiveX** 控件到 **Web** 页面时，可以将其发布给从该页面下载该控件的任何用户。除非对该控件进行了许可，否则很难防止最终用户取走您的控件并在他们自己的应用程序中使用。许可证可以作为控件的一种版权，来防止未经授权的使用。

可以在打包和部署向导内部进行安全性设置。当使用 **Internet** 软件包时，向导中的某个屏幕将要求您确认安全性设置。该屏幕上只列出工程中没有实现称为 **IOObjectSafety** 的安全性接口的对象。

注意 署名和许可必须在打包过程之外进行。在为部件打包之前，应对任何部件安排许可。数字署名可以在打包应用程序之后进行，打包和部署向导在 **.cab** 文件中为有关数字署名的信息保留了空间。

详细信息 有关部件安全性、使用许可以及签名的详细说明，请参阅《部件工具指南》中，《建立 **Internet** 应用程序》中的第四章“下载 **ActiveX** 部件”的“准备部件下载的步骤”。有关 **IOObjectSafety** 接口的详细信息，请参阅《部件工具指南》中，《建立 **Internet** 应用程序》中的第四章“下载 **ActiveX** 部件”的“为 **ActiveX** 部件设置安全级别”。

从属文件

从属 (.dep) 文件包含了有关应用程序或部件在运行时需要的信息，例如，哪些文件是需要的，如何注册这些文件，以及应安装到用户计算机上的什么位置。您可以在 Visual Basic 的所有版本中为标准工程创建一个 .dep 文件。如果您有 Visual Basic 的专业版或企业版，则还可以为 ActiveX 控件、ActiveX 文档以及其它 ActiveX 部件创建 .dep 文件。

打包和部署向导在为应用程序打包时将使用 .dep 文件。它扫描该应用程序所有可用的从属信息，生成一个有关该应用程序所需的运行时文件的综合列表，然后为列表生成安装信息。对于标准软件包，.dep 文件中的信息将被写入一个 Setup.lst 文件，该文件存储在打好包的 .cab 文件之外。对于 Internet 软件包，.dep 文件中的信息被写入到一个 .inf 文件，该文件存储在打好包的 .cab 文件中。

当为部件打包时，您可以选择创建一个伴随该部件部署的 .dep 文件。如果已经创建了一个希望带从属信息一起部署的部件，则可以选择该选项。建议您在打包和部署从属文件之前，打包和部署部件，以便向导的打包部分知道从属文件要引用的部件的来源位置。

从属文件的类型

在 Visual Basic 中，从属信息存储在由打包和部署向导生成的文件或由您手工创建的文件中。有两种可以包含从属信息的文件类型：

- 部件的 .dep 文件——一个 .dep 文件，其中列出了某个特定的控件或部件所需的文件。当打包和部署向导创建安装程序时将使用这个文件。

此外，向导可以为您创建这种类型的 .dep 文件。

- **VB6dep.ini 文件**—— 一个对于整个 Visual Basic 开发环境的从属文件列表。

在运行打包和部署向导时，将在 .dep 文件和 .vb6dep 文件中查找从属信息。如果在两个位置都没有找到某个部件的从属信息，向导将通知您缺少从属信息。您可以忽略这种遗漏，也可以通过创建适当的从属文件来解决该问题。

注意 如果忽略这种遗漏，您的程序在安装后可能不能正常工作。不过，如果您肯定从属文件已经在用户的机器中，则可以忽略该警告，继续执行。

部件的从属文件

一个 .dep 文件列出一个特定部件所需要的所有文件。当从代理商处购买或使用部件时，将得到一个 .dep 文件。例如，Visual Basic 附带的所有 ActiveX 控件都有一个相应的 .dep 文件。这些 .dep 文件列出了由该控件所使用的所有从属文件，还包括版本和注册信息。

对于使用 Visual Basic 创建的任何部件，如果该部件可能在另一个工程中使用，则应该为其生成一个 .dep 文件。工程中每个部件的 .dep 文件信息将组成该工程的从属信息。如果不为您的部件创建 .dep 文件，则任何使用该部件的工程的从属信息都是不正确的。

VB6dep.ini 文件

VB6dep.ini 文件为打包和部署向导提供了一个由 Visual Basic 使用的从属和引用的通用列表。这个列表是在安装 Visual Basic 时创建的，位于 Visual Basic 主目录的 \Wizards\PDWizard 子目录中。

缺少从属信息

如果工程中的部件缺少从属信息，打包和部署向导将会通知您。您可以通过三种途径添加必要的从属信息：

- 编辑 vb6dep.ini 文件，以便为特定的部件手工添加一项。
- 用打包和部署向导为该部件创建一个 .dep 文件。
- 与部件的代理商联系，并索要一个 .dep 文件。

使用向导部署应用程序

部署应用程序是指将打好包的应用程序放置到选定的发布媒体上，或放置到可由此下载的 Web 站点上的操作。有两种部署 Visual Basic 应用程序的途径：

- 可以使用打包和部署向导的“部署”部分来将应用程序部署到软盘、本地或网络驱动器或 Web 站点上。
- 可以手工将文件复制到磁盘或共享目录，或手工将文件在适当的 Web 站点上发布。

打包和部署向导提供了快捷方式，并能自动执行在手工部署应用程序时不得不手工执行的一些相同的任务。

部署过程的全部步骤

不论是使用打包和部署向导，还是手工方式来部署软件包，都有一些必须采取的步骤。

1. 创建用于部署的软件包。可以是单个的 .cab 文件，也可以是一系列 .cab 文件，这取决于您计划如何发布应用程序。
2. 确定要部署的软件包。可以为选定的工程选择任何有效的软件包。
3. 选择部署方法。可以将应用程序部署到 Internet、软盘或本地或网络驱动器的目录上。
4. 选择要部署的文件。如果要部署到 Internet 上，您可以对要部署的文件列表添加或删除文件。
5. 为要部署的文件确定目标。对于 Internet 部署方式，本步骤将指定一个应部署该软件包的 Web 站点。对于目录部署方式，则本步骤指定部署该软件包的驱动器位置。对于软盘部署方式，则本步骤要选择适当的软盘驱动器。
6. 部署软件包。如果使用的是打包和部署向导，向导将为您处理这一过程。如果没有使用，则您必须将文件复制到共享或本地驱动器上的适当位置，或将文件发布到 Web 上。

部署功能

您可以使用打包和部署向导轻松地将打好包的应用程序复制到适当的位置。打包和部署向导在部署过程中根据输入信息执行这些步骤：

- 选择部署方法。您可以选择是要部署到软盘、本地或网络驱动器、还是 intranet 或 Internet 站点。
- 基于脚本的部署过程。如果想在整个向导操作过程中使用相同或相似的设置值，可以从同一个工程的另一个部署过程选择一个脚本。这样可以节省许多时间。
- 自动访问 Web 发布技术。使用 Web 发布技术可以轻松地将文件发布到 intranet 或 Internet 站点。

详细信息 有关 Web 发布的详细信息，请参阅 Internet Client CDK 中的“Internet Tools and Technologies”。有关向导的更多特性，请参阅本章稍后的“打包功能”。

部署应用程序

向导可以让您选择是使用 Web 发布技术部署到 Web 上，还是部署到软盘或本地或网络驱动器的目录上。

部署到软盘、目录以及 CD

只有在使用“多个压缩文件”选项创建了标准软件包后，才可以使用打包和部署向导部署到软盘上。该选项确保了软件包将由多个 .cab 文件组成，或由一个小于一张磁盘容量的单个 .cab 文件组成。在将 .cab 文件复制到软盘前，系统将为每个磁盘提供格式化选项。您无需格式化磁盘，但在部署过程中必须使用空磁盘。

如果选择部署到某个目录，则系统将提示您挑选一个用来复制文件的本地或网络目录。然后您可以引导用户从该位置来访问应用程序的安装程序，也可以将文件放置到 CD-ROM 上。

注意 如果您有一个可写的 CD 驱动器，则可以使用向导的部署部分直接将文件复制到该驱动器，而不必先部署到一个目录后再将文件复制到 CD 上。

部署到 Web

您可以将任何软件包，不论是标准软件包还是 Internet 软件包，部署到 Web 上。当您选择“Web 发布”作为部署方法时，系统会把工程文件夹视为部署的本地基础文件夹。本地基础文件夹用于确定应如何将文件何目录复制到所选择的 Web 站点上。处于本地基础目录中的文件和目录将被部署到同基础目录有相同的目录结构的 Web 服务器上。

注意 缺省情况下，向导不会在工程目录或 \Support 子目录中部署源文

件。向导的打包部分将创建 \Support 目录并将文件放置在该目录中，该目录可用来重新创建 .cab 文件。

管理向导脚本

如果使用打包和部署向导，可以创建并存储脚本。脚本是指在打包或部署过程中所做选择的记录。创建一个脚本就可以将这些选择保存起来，以便您在向导以后的过程中为同一个工程应用该脚本。使用脚本可以显著地节省打包和部署时间。此外，可以使用脚本以静态模式打包和部署应用程序。每次打包或部署工程时，Visual Basic 都会将有关过程的信息保存为一个脚本。工程的所有脚本都存储在应用程序工程目录的一个特别文件中。您可以使用打包和部署向导的“管理脚本”选项来查看当前工程所有脚本的列表。该选项让您可以：

- 查看所有打包或部署脚本的列表。
- 重命名一个脚本。
- 创建一个具有新名字的脚本副本。
- 删除不再需要的脚本。

警告 如果删除了打包脚本，则打包和部署向导的部署部分将不再把由该脚本创建的软件包视为一个可以部署的软件包。因此，如果要部署这些文件，必须重新对文件打包。只有当您肯定不再需要这些脚本时，才能将其删除。

要查看脚本列表，请按照以下步骤执行：

1. 启动向导，然后从主屏幕上选择“管理脚本”。

重点 如果已经将向导作为独立的应用程序启动，那么在选择“管理脚本”之前必须选择所需的 Visual Basic 工程。

2. 给要查看的脚本选择适当的面板。

安装工具包 (Setup Toolkit)

安装工具包是随 Visual Basic 一起安装的一个工程，当打包和部署向导创建安装程序时，将使用这个工程。安装工具包工程包含了供应用程序的安装程序用来将文件安装到用户计算机上的窗体和代码。当使用打包和部署向导时，向导将包括由安装工具包工程所创建的 `setup1.exe` 文件。这个文件将用作应用程序的主安装文件。

注意 在安装过程中包含两个安装程序——`setup.exe` 和 `setup1.exe`。

`setup.exe` 程序在用户计算机上执行预安装处理，包括安装 `setup1.exe` 程序以及运行主安装程序所需的任何其它文件。在安装工具包中，只有 `setup1.exe` 是可以自定义的。

安装工具包除了在创建安装程序的过程中扮演了支持角色之外，还可以用于修改安装过程中出现的屏幕，或直接创建一个安装程序。如果需要在安装顺序中添加向导所不支持的附加功能，则您可以创建一个自定义的安装程序。

安装工具包工程位于 Visual Basic 主目录的 \Wizards\PDWizard\Setup1 子目录中。

小心 该工程中的文件与打包和部署向导的输出结果所使用的文件是相同的。在没有在另一个目录中备份之前，不要对其进行修改。如果修改了 `setup1.exe`，打包和部署向导所创建的后续安装程序将使用这个修改后的版本。

可以使用安装工具包将 `Setup1.vbp` 文件加载到 Visual Basic，然后对工程的外观或功能进行修改。要这样做，您可能需要手工来完成本来将由打包和部署向导自动完成的步骤。后面的章节将描述这个过程的步骤，并说明如何决定安装程序中需要包括哪些文件，如何创建 `Setup.lst`，如何创建发布媒体，以及如何测试安装程序。

修改打包和部署向导的全部步骤

如果要修改安装工具包来更改由打包和部署向导所创建的输出时，可以按照这些步骤进行操作：

1. 修改安装工具包工程，使之包含所有新的提示、屏幕、功能、代码或其它要包括的信息。完成后，编译该工程，以创建 `setup1.exe`。
2. 运行打包和部署向导，按照每个屏幕的提示进行操作，来创建您的发布媒体。

创建自定义安装程序的全部步骤

如果想使用安装工具包而不是打包和部署向导来创建一个安装程序，则必须按照这些步骤进行操作：

1. 如果必要，请修改安装工具包，使之包含任何新的提示、屏幕、功能、代码或其它要包括的信息。
2. 决定要发布的文件，包括所有的运行时文件、安装文件以及从属文件。
3. 决定要把文件安装到用户计算机的什么位置。
4. 手工创建 `Setup.lst` 文件，以说明该工程必须包括的所有文件的名称和安装位置。
5. 决定如何发布文件。
6. 使用 `Makecab` 实用程序来创建工程的 `.cab` 文件。

提示 您可以使用打包和部署向导来创建 `.cab` 文件，然后手工修改 `.cab` 文件。当向导创建 `.cab` 文件时，它将在工程目录的 `\Support` 子目录中创建一个 `.ddf` 文件以及一个批处理文件。要修改这些 `.cab` 文件，请编辑 `.ddf` 文件，然后运行所提供的批处理文件。该批处理文件将依次运行 `Makecab.exe`，以重新创建 `.cab` 文件。

7. 通过编译修改过的安装工具包工程，为您的工程创建 `setup1.exe`。
8. 将文件复制到发布媒体上，或使用 Web 发布向导手工将文件发布到 Web 站点上。该向导可以在 `ActiveX SDK` 中获得。

详细信息 有关使用 Web 发布向导的详细信息，请参阅 `Internet Client SDK` 中的“`Internet Tools and Technologies`”。有关修改安装工具包工程的详细信

息，请参阅本章稍后的“修改安装工程”。有关如何将文件放置到用户计算机上的详细信息，请参阅“需要发布的文件”以及“将文件安装到用户机器的什么位置”。有关将文件复制到适当媒体的详细信息，请参阅“手工创建发布媒体”。

修改安装工程

如果希望给打包和部署向导所创建的安装过程添加新的屏幕、提示或事件，则可以修改 `Setup1.vbp` 工程。您可以在安装程序中编写代码，如同在其它的 Visual Basic 程序中一样。大量可用的函数调用在安装程序中都是特别有用的。

可能修改安装工具包工程的一些情形示例包括：

- 需要在安装过程中添加特殊的用户提示。
- 希望为安装程序创建一个自定义的外观。
- 希望在安装过程中显示布告板。布告板介绍了有关产品的特性、服务和支持、注册的信息，以及其它相关信息。
- 希望使用自己的压缩工具将应用程序的文件复制到发布媒体上。

重点 由于打包和部署向导要使用安装工具包中的文件，因此应当在进行任何更改之前都要备份该工程。此外，还应该备份 `Setup1` 目录中的所有内容。

要修改安装工具包工程，请按照以下步骤执行：

1. 在进行任何更改之前，创建一个 `\Wizards\PDWizard\setup1.exe` 和

\Wizards\PDWizard\Setup1 目录中所有内容的备份。

2. 从 \Wizards\PDWizard\Setup1 目录打开 setup1.vbp 工程。
3. 对该工程的代码、窗体或模块进行更改。
4. 保存该工程，并将其进行编译以创建 setup1.exe。
5. 如果您使用打包和部署向导来对应用程序打包，则启动打包和部署向导并为应用程序创建一个软件包。
6. 如果创建您自己的自定义安装软件包，请继续执行“安装工具提示”中概述的步骤。

重点 不论是使用打包和部署向导，还是使用安装工具包工程，任何时候创建软件包时，都应确保您工程的版本号码已在 Visual Basic 中的“工程属性”对话框的“生成”选项卡中设置。如果您要发布一个已有应用程序的新版本，这一点将是特别重要的，因为如果没有对版本号码进行适当的更改，那么最终用户的计算机可能会认为关键文件不需要更新。

需要发布的文件

创建一个自定义安装程序的第一步就是决定哪些文件要发布。所有的 Visual Basic 应用程序都需要一个最小的文件集合，称之为引导文件，即在安装您的应用程序之前所需的文件。此外，所有的 Visual Basic 应用程序都需要应用程序特有的文件，例如一个可执行文件 (.exe)、数据文件、ActiveX 控件或 .dll 文件。

在运行和发布应用程序时，需要三种主要的文件类型：

- 运行时文件
- 安装文件
- 应用程序特有的文件

运行时文件

运行时文件是您的应用程序为在安装后能正确运行而必备的文件。这些文件是所有的 Visual Basic 应用程序都需要的。下面是 Visual Basic 工程所用的运行时文件：

- Msvbvm60.dll
- Stdole2.tlb
- Oleaut32.dll
- Olepro32.dll
- Comcat.dll
- Asyncfilt.dll
- Ctl3d32.dll

尽管这些文件是所有 Visual Basic 应用程序所需要的，但对每种类型的安装软件包，可能不是必需的。例如，当创建 Internet 软件包时，打包和部署向导认为任何能够执行 Internet 下载的计算机都已经具备了除 Msvbvm60.dll 之外的所有这些文件。因此这个文件是向导要包含到 Internet 软件包中的唯一的运行时文件。

注意 运行时文件可以根据其安装位置进一步分类。详细信息请参阅“将

文件安装到用户机器的什么位置”。

标准软件包的安装文件

安装文件是在用户计算机上安装标准应用程序所需的所有文件。这些文件包括安装可执行程序（`setup.exe` 和 `setup1.exe`）、安装文件列表（`Setup.lst`）以及卸载程序（`st6unst.exe`）。

要在磁盘、CD 或从网络位置发布的 Visual Basic 应用程序使用相同的安装文件，而不管是使用打包和部署向导，还是使用安装工具包来创建安装程序。这些文件如下所列。

文件名	描述
<code>setup.exe</code>	用户运行预安装文件的程序，这些预安装程序是将您的应用程序安装到用户机器所必需的。例如， <code>setup.exe</code> 文件将安装 <code>setup1.exe</code> 文件，Visual Basic 运行时 DLL，以及其它运行安装过程剩余部分所必不可少的文件
<code>Setup1.exe</code>	您的 Visual Basic 应用程序的安装程序。这个可执行程序由安装工具包生成，并由打包和部署向导包括在软件包中。您可以将这个文件重命名，只要将新名称包括在 <code>Setup.lst</code> 文件中
<code>Setup.lst</code>	包含安装指令并列出所有要安装到用户机器上的文件的文本文件
<code>Vb6stkit.dll</code>	包含 <code>Setup1.exe</code> 所使用的各种函数的库
<code>St6unst.exe</code>	应用程序删除实用程序

注意 以 Internet 方式来发布的应用程序通常无需使用这些文件的任何一种。有关 Internet 发布方式所包含的文件的详细信息，请参阅本章前

面的 “Internet 软件包”。

应用程序的从属信息

要运行应用程序，最终用户除了一般的运行时文件和特殊的安装文件之外，可能还需要某些文件。很多文件都是很显然的：可执行程序、任何数据文件以及您所使用的任何 **ActiveX** 控件。不太明显的文件是工程的其它从属文件。例如，工程中使用的有些 **ActiveX** 控件可能还需要其它文件。打包和部署向导的任务之一就是确定这种必需的文件完整列表。

详细信息 有关使用打包和部署向导来为应用程序创建从属文件的详细信息，请参阅本章前面的“从属文件”。

将文件安装到用户机器的什么位置

在编写安装程序之前，必须决定所有必需的文件将安装到用户机器的什么位置。可以将这个信息记录在 **Setup.lst** 文件中。有关如何在文件中记录该信息的详细信息，请参阅本章稍后的“手工编辑 **Setup.lst** 文件”。

应用程序所需的文件可以分为若干类。

- 程序特有的文件——运行您的应用程序所需的文件，并且这些文件不被其它应用程序所使用。
- 共享的文件——您的应用程序所使用的，但同时也被用户计算机上的其它应用程序所访问的文件。

- 远程自动化服务器部件——远程自动化或 DCOM 功能所必需的文件。每种类型的文件最好安装到不同的位置。

程序文件

程序文件是指为了运行您的应用程序而必需具备的、且只是在您的应用程序上下文中是有用的文件，例如，该应用程序的 .exe 文件及其所需的数据文件。

程序文件应安装在安装过程中由用户指定的应用程序目录中。Setup1.vbp 中的代码举例说明了如何将文件写入这一位置。缺省情况下，安装工具包使用 \Program Files 目录作为根目录，将应用程序安装到 Windows 95 或更新版本以及 Windows NT 系统上。例如，Setup1 建议将 Project1 安装在 \Program Files\Project1 目录。

警告 将文件安装到用户机器上时，不应该复制文件的旧版本去覆盖新版本。Setup1.bas 中的 CopyFile 函数使用 VerInstallFile API 函数将文件复制到用户的机器上。VerInstallFile 不会用一个旧版本去覆盖现存的文件。

共享的应用程序文件

共享的应用程序文件是指可能被系统中的多个应用程序使用的文件。例如，可能有多个软件商提供的应用程序都使用了同一个 ActiveX 控件。如果您创建了一个使用该控件的应用程序，则应在安装程序中将该控件的 .ocx 文件指明为共享文件。

共享文件必须安装在一个其它应用程序可以访问到的位置。在多数情形下，对于 Windows 95 和 Windows NT 4.0 及更新版本，这个位置是 \Program Files\Common Files。

当最终用户卸载您的应用程序时，只有当其它应用程序没有使用这些共享文件时，系统才会将其删除。

远程自动化部件

可以将远程自动化服务器部件安装到 \Windows\System 或 \Winnt\System32 目录。这可以确保您的应用程序使用的是最新的远程自动化服务器部件。

提示 您可以使用 \$(WinSysPath) 安装宏来确保这些文件被安装到正确的目录中。

手工编辑 Setup.lst 文件

如果使用打包和部署向导，向导将自动创建 Setup.lst 文件。如果需要自定义该文件，您可以在该文件被创建后，手工进行编辑。

Setup.lst 文件描述了为您的应用程序必须安装到用户机器上的所有文件，此外还包含了有关安装过程的关键信息。例如，Setup.lst 文件告诉系统每个文件的名称、安装位置以及应如何进行注册等。Setup.lst 文件共有五个段：

- **Bootstrap** 段——列出了关于应用程序的核心信息，例如应用程序的主安装程序的名称、在安装过程中使用的临时目录以及在安装过程的起始窗口出现的文字。
- **Bootstrap Files** 段——列出了主安装文件所需的所有文件。通常这部分只包括 Visual Basic 运行时文件。
- **Setup1 Files** 段——列出应用程序所需的所有其它文件，例如 .exe 文件、数据以及文本。
- **Setup** 段——包含应用程序中的其它文件需要的信息。
- **Icon Groups** 段——包含了安装过程将创建的程序组的信息。这部分的每个成员都有一个与之相关的部分，包含了在这个程序组中要创建的图标。

详细信息 有关这几段语法的详细信息，请参阅“Bootstrap 和 Setup1 Files 段的格式”。

Bootstrap 段

Bootstrap 段包含 setup.exe 文件安装和启动应用程序的主安装程序所需的所有信息。

注意 请记住，在安装过程中要用到两个安装程序：一个是 setup.exe，这是一个预安装程序；另一个是 setup1.exe，这是由安装工具包编译生成的。**Bootstrap** 部分将为 setup.exe 文件提供指示。

Bootstrap 段包含下述成员：

成 员	描 述
SetupTitle	当 <code>setup.exe</code> 将文件复制到您的系统时所出现的对话框中显示的标题
SetupText	当 <code>setup.exe</code> 将文件复制到您的系统时所出现的对话框中显示的文字
CabFile	应用程序的 <code>.cab</code> 文件的名称，或者，如果软件包有多个 <code>.cab</code> 文件，则表示第一个 <code>.cab</code> 文件的名称
Spawn	当 <code>setup.exe</code> 完成处理后要启动的应用程序名称。在大多数实例中，这个名称将是 <code>setup1.exe</code> 文件
TmpDir	存放在安装过程中产生的临时文件的位置
Uninstall	用做卸载程序的应用程序名称。一般来说，该程序是 <code>st6unst.exe</code> ，这个文件是由向导创建所有的软件包时自动打包加入的

Bootstrap Files 段

Bootstrap Files 段列出了在加载您的应用程序和从属文件之前，必须加载入用户机器的所有文件。这些预安装的、或称为引导文件，包括运行任何 Visual Basic 应用程序所必需的核心文件，例如 Visual Basic 运行时 DLL (`Msvbvm60.dll`)。安装程序在安装和启动主安装程序之前将安装这些文件。下面的示例显示了一个典型的 Bootstrap Files 段中的条目：

[Bootstrap Files]

File1=@ Msvbvm60.dll,\$(WinSysPathSysFile),\$ (DLLSelfRegister),
1/23/98 9:43:25 AM,1457936,6.0.80.23

File2=@ OleAut32.dll,\$(WinSysPath),\$ (DLLSelfRegister),
1/21/98 11:08:26 PM,571152,2.30.4248.1

File3=@ OlePro32.dll,\$(WinSysPathSysFile),\$ (DLLSelfRegister),
1/21/9811:08:27 ,152336,5.0.4248.1

Setup1 Files 段

Setup1 Files 段包含了您的应用程序所必需的所有其它文件，例如 .exe 文件、数据、文本以及从属文件等。安装程序将在安装了 Bootstrap Files 段所列出的核心文件之后安装这些文件。

下面的示例显示了一个典型的 Setup1 Files 段中的条目：

[Setup1 Files]

File1=@ LotsAControls.exe,\$(AppPath),\$ (EXESelfRegister),,1/26/98 3:43:48
PM,7168,1.0.0.0

File2=@ mscomctl.ocx,\$(AppPath),\$ (DLLSelfRegister),,1/23/98 9:43:40
AM,1011472,6.0.80.23

Setup 段

Setup.lst 文件的 Setup 段是一个简单的列表，它列出了安装过程的其它部分所使用的信息。下表列出了 Setup 段所包含的信息。

成员	描述
Title	将出现在安装期间的快速显示屏幕、“启动”菜单的程序组以及程序项名称上的应用程序名称
DefaultDir	缺省的安装目录。用户可以在安装过程中指定一个不同的目录
ForceUseDefDir	如果为空，则会提示用户输入一个安装目录。如果设为 1，则应用程序将自动安装到 Setup.lst 的“DefaultDir”所指定的目录中
AppToUninstal	应用程序在“控制面板”中的“添加/删除程序”实用程序中出现的名称
AppExe	应用程序的可执行文件的名称，例如 Myapp.exe

IconGroups 段

IconGroups 段包含了关于安装过程所创建的“启动”菜单的程序组的信息。每个要创建的程序组首先在 IconGroups 部分列出，然后指定一个单独部分（Group0, Group1, Group2 等），在此部分中包含关于这个程序组的图标和标题的信息。程序组从 0 开始顺序编号。

下面的示例显示了一个典型的 IconGroups 段中的条目及其相关的子段：

```
[IconGroups]
Group0=MyTestEXE
Group1=Group1
```

```
[MyTestExe]
Icon1=my.exe
Title1=MyTestExe
```

```
[Group1]
Icon1=ReadMe.txt
Title1=ReadMe
Icon2=my.hlp
Title2=Help
```

Bootstrap 和 Setup1 Files 段的格式

Setup.lst 文件的 Bootstrap Files 和 Setup1 Files 段包含了安装程序（setup.exe 及 setup1.exe）需要安装到用户计算机上的文件的完整列表。每个文件都用一行单独列出，且必须使用下述格式：

Filex= file,install,path,register,shared,date,size[,version]

部 分	含 义
Filex	必须在每行的开始出现的关键字。X 是一个顺序编号，每个部分中都是从 1 开始，并按递增顺序编号。不能跳值编号
File	安装完成后将在用户的计算机上出现的文件名称。通常该名称与 Install 参数一样。如果该文件需要从 cab 文件解压缩得到，请在其名称之前加上一个 @ （例如 @my.exe）
Install	出现在任何发布媒体上的文件名称
Path	安装该文件的目录。可以是一个实际的目录路径，也可以是表示一个用户指定的路径的宏，或者这二者的组合。有关可用的宏的详细信息，请参阅本章稍后的“Setup.lst 文件中的路径参数宏”
Register	说明如何将该文件包括到用户系统注册表中的键。详细信息请参阅本章稍后的“Setup.lst 文件中的注册表键”
Shared	指定该文件应作为共享文件安装
Date	文件最后一次修改的日期，即在 Windows 资源管理器中出现的日期。该信息能帮助您确认安装盘上文件的版本是否正确
Size	在 Windows 资源管理器中显示的文件大小。安装程序使用该信息来计算您的应用程序在用户机器上需要多少磁盘空间
Version	该文件可选的内部版本号码。请注意，这个版本号与在查验文件属性时所看到的版本号不必是同一个号码

Setup.lst 文件中的路径参数宏

Setup.lst 文件中的 Path 参数表示文件将被安装到的位置。这个参数所使用的值可以是一个实际的路径，也可以是一个表示用户所指定的路径的宏。下表列出了在安装过程中可以使用的宏。

宏	在此目录安装文件	在此段有效
\$(WinSysPath)	\Windows\System (Windows 95 或更新版本)	Setup1 Files 及 Bootstrap Files
\$(WinSysPathSysFile)	\Winnt\System32 (Windows NT) 该文件将作为系统文件安装，当删除应用程序时这些文件不会被删除。	只在 Setup1 Files 段有效
\$(WinPath)	• \Windows (Windows 95 或更新版本) • \Winnt (Windows NT)	Setup1 Files 及 BootStrap Files
\$(AppPath)	由用户指定的应用程序目录，或在 Setup 部分指定的 DefaultDir 值。	只在 Setup1 Files 段有效
\$(AppPath)\Samples	\Samples，位于应用程序目录之下。	只在 Setup1 Files 段有效
\path (for example, c:\)	由 path 所确定的目录（建议不使用）。	只在 Setup1 Files 段有效

续表

<code>\$(CommonFiles)</code>	<code>\Program Files\Common Files\</code> 通常与子目录结合使用，如 <code>\$(CommonFiles)\My Company\My Application。</code>	只在 Setup1 Files 段有效
<code>\$(CommonFilesSys)</code>	<code>\$(CommonFiles)\System</code>	只在 Setup1 Files 段有效
<code>\$(ProgramFiles)</code>	<code>\Program Files</code>	只在 Setup1 Files 段有效

Setup.lst 文件中的注册键

Setup.lst 文件中的注册键指明该文件应如何在用户的计算机上注册。您可以指明该文件无需注册，或为需要注册的文件指定若干个注册选项之一。下表列出了可能的键。

注册键	含义
(没有键)	该文件没有包含链接或嵌入对象，且不需要注册
\$(DLLSelfRegister)	该文件是一个自注册的 .dll、.ocx 或任何其它具有自注册信息的 .dll 文件
\$(EXESelfRegister)	该文件是一个用 Visual Basic 创建的 ActiveX .exe 部件或任何其它支持 /RegServer 及 /UnRegServer 命令行开关的 .exe 文件
\$(TLBRegister)	该文件是一个类型库文件，应相应地进行注册
\$(Remote)	该文件是一个远程支持文件 (.vbr)，应相应地进行注册
Filename.reg	该文件是一个您发布的部件，该部件需要注册，但不提供自注册能力。这个键指出了 .reg 文件，其中包含了需要在系统注册表中更新的信息。该 .reg 文件也必须添加到 Setup.lst 文件中，并被安装

推荐您不要使用 filename.reg 键将注册信息加入到注册表的方法。以这种方式加入的注册表项不能用应用程序删除程序自动卸载。

指定 Setup.lst 文件中的远程服务器部件

如果在应用程序中使用了远程服务器部件，则必须在 Setup1 Files 部分中放置一个将文件标记为远程服务器部件、并指定连接信息的条目。该信息将被“客户注册”实用程序使用，来注册该远程自动化服务器。

下面的示例显示了如何指定远程服务器部件：

File1=@ LotsAControls.vbr,\$(WinSysPath),\$(Remote),,1/26/98 3:43:48 PM,1024,1.0.0.0

File2=@ Server2.vbr,\$(WinSysPath),\$(Remote),,1/23/98 9:43:40 AM,1024,6.0.80.23

Remote1="Schweizer","ncacn_ip_tcp",1,RA

Remote2=,,1,DCOM

Remotex 条目包括服务器地址、网络协议以及身份验证信息等，中间用逗号分隔。您还必须使用 **RA** 或 **DCOM** 参数来指定该部件是否被用于远程自动化或 **DCOM** 环境。

手工创建发布媒体

您可以使用多种方式来发布应用程序，发布到软盘或光盘上、通过网络发布或在 Web 站点上发布。作为部署过程的一个部分，您必须将打好包的文件复制到计划使用的发布媒体上。

注意 如果使用打包和部署向导来对文件打包和部署，这些步骤是不需要的，因为向导将自动为您创建磁盘、网络位置或 Internet 部件下载发布媒体。

在手工创建发布媒体时，您可以使用下述任何方法：

- 如果计划发布到软盘上，则必须将文件按特定顺序复制到磁盘上。
setup.exe 和 **Setup.lst** 文件必须位于第一张磁盘上，接下来是您工程的 **Setup.lst** 文件中的 **Bootstrap Files** 段的所有文件，然后将其它 **.cab** 文件复制到剩余的磁盘上。

- 如果要发布到网络共享目录上，只需简单地将打好包的文件复制到适当的位置，可以使用 Windows 资源管理器、命令提示符方式或一些其它方法。
- 如果要发布到 CD 上，只需简单地将文件复制到 CD 上即可。
- 如果要发布到 Internet 上，可以使用 Web 发布向导将 Internet 软件包复制到适当的 Web 站点，Web 发布向导可以在 ActiveX SDK 中获得。

详细信息 有关使用 Web 发布向导创建 Web 发布站点的详细信息，请参阅 Internet Client SDK 中的“Internet Tools and Technologies”。

创建发布媒体

在确定了安装程序中要包括的文件、创建了 Setup.lst 文件、压缩所有必需的文件以及决定在媒体上如何安排这些文件之后，就可以创建发布媒体了。

- 如果使用软盘作为发布媒体，则需要将这些文件复制到一张或多张磁盘上。
- 如果使用其他机制，例如 CD 或网络位置，则需要将文件复制到适当的位置或 CD 产品的存储数据区域。
- 对于 Internet 或 intranet 发布方式，则应在为 Internet 部件下载进行格式化之后，将文件发布到适当的 Web 站点。

创建发布磁盘

使用数字和磁盘名以及有关安装的指示来标记您的发布磁盘是一个良好的习惯。可以在每张发布磁盘上重复安装消息。在标记了磁盘后，就可以准备将文件复制到发布磁盘了。

要创建发布磁盘，请按照以下步骤执行：

1. 将下面所列的预安装文件复制到第一张磁盘上：
 - Setup.exe
 - Setup.lst
2. 将剩余的引导 .cab 文件复制到第一张磁盘的剩余空间以及需要的任何后续磁盘上。
3. 在复制完所有的引导 .cab 文件之后，将余下的 .cab 文件复制到必要的磁盘上。

手工部署到 Web 站点

如果希望手工将应用程序或部件部署到 Web 站点，则可以使用 Web 发布向导将打好包的文件部署到适当的位置。Web 发布向导可以从 Microsoft 位于 www.microsoft.com 的 Web 站点获得，也可以与 Internet Explorer 4.x 一起安装。

详细信息 有关 Web 发布向导的详细信息，请参阅 Internet Client SDK 中的“Internet Tools and Technologies”。有关决定如何将文件放置到磁盘上的

详细信息，请参阅本章前面的“手工创建发布媒体”。

与安装工具包一起使用打包和部署向导

除了使用安装工具包工程来创建您的自定义安装工程外，还可以将安装工具包工程与打包和部署向导一起使用。在这种情况下，您可以使用安装工具包工程来自定义安装过程的屏幕或其它部分，然后使用向导来创建和部署应用程序的软件包。

例如，您可以将安装工具包与打包和部署向导一起使用，以便为安装工程添加对话框，提示用户指定是否安装应用程序的可选功能。例如，您可以提供一个联机帮助，而用户也可以不安装。可以按照您的需要添加任意的安装选项。

要为安装程序添加一个安装选项，请按照以下步骤执行：

1. 在 `Setup1.vbp` 工程中，编辑 `setup1.frm` 窗体中的 `Form_Load` 事件的代码。要添加功能，可以在调用 `ShowBeginForm` 函数 (`Sub ShowBeginForm`) 的代码块之后添加代码。

下面的示例显示了如何添加一个对话框，来询问用户是否要安装可选文件：

```
Dim LoadHelp As Integer
LoadHelp = MsgBox ("Do you want to install Help? ", vbYesNo)
If LoadHelp = vbYes Then
```



```

    CalcDiskSpace "Help"
EndIf
' 包含 cIcons = CountIcons(strINI FILES) 的代码块
If LoadHelp = vbYes Then
    cIcons = CountIcons("Help")
EndIf
' 包含 CopySection strINI_FILES 的代码块。
If LoadHelp = vbYes Then
    CopySection "Help"
EndIf
' 包含 CreateIcons, strINI_FILES, strGroupName 的代码块

```

2. 关闭 **Setup1.frm**，保存该窗体以及安装工具包工程，并编译创建 **Setup1.exe** 文件。
3. 运行打包和部署向导，并从主屏幕中选择“打包”。
4. 根据向导的各个屏幕进行适当的选择。对上面的示例，您可以在“添加和删除”屏幕中列出的“自定义”对话框中确认用户可以选择安装的所有可选文件。
5. 在完成打包和部署向导的所有操作后，将生成发布媒体。
6. 对 **Setup.lst** 文件进行任何必要的更改。在上面的示例中，您可以在代码的 **CopySection** 部分所使用的部分中添加一个新的部分。在这种情况下，这个部分可能会象下面这样：

```
[Help]
```

```
File1=MyApp.HL1,MyApp.HLP,$(AppPath),,,10/12/96,2946967,0.0.0
```

7. 部署并测试您的软件包。

当用户运行本过程所示示例的安装程序时，安装程序将所有的 **BootStrap** 文件复制到用户机器，然后提示用户指明是否安装帮助文件。如果用户选择“是”，则 **CalcDiskSpace** 语句将确定在用户机器上是否有足够的磁盘空间来存放帮助文件。然后该程序安装 **Setup.lst** 中的 **Setup1 Files** 部分列出的所有文件。

接下来，程序将再测试一次 **LoadHelp** 标记。如果用户选择安装帮助文件，**Setup1.exe** 就开始为帮助文件执行 **CopySection** 语句，然后安装 **Setup.lst** 的 **[Help]** 部分所列的文件。

详细信息 请参阅本章前面的“打包和部署向导”，了解有关该向导特性的详细信息。

测试您的安装程序

在完成了应用程序的包装工程并产生了发布媒体后，必须对安装程序进行测试。请确保在一台没有 **Visual Basic** 以及应用程序所需的任何 **ActiveX** 控件的机器上测试您的安装程序。您还应该在所有可用的操作系统上测试该安装程序。

要测试基于软盘或基于 **CD** 的安装程序

1. 将第一张磁盘或 **CD** 插入适当的驱动器。

2. 在 Windows 95 和 Windows NT 4.0 或更新版本中，从“启动”菜单，选择“运行”并键入：

drive:\setup

-或-

从磁盘驱动器上双击 Setup.exe。

3. 安装完成后，运行安装好的程序，确信其运转如常。

要测试基于网络驱动器的安装程序

1. 从同一个网络中的要作为发布服务器的另一个计算机上，与包含发布文件的服务器及其目录建立连接。
2. 在发布目录中，双击 Setup.exe 文件。
3. 安装完成后，运行安装好的程序，确信其运转如常。

要测试基于 Web 的安装程序

1. 将软件包部署到一个 Web 服务器。
2. 访问一个 Web 页面，要求从该页面可以引用您的应用程序的 .cab 文件。下载操作会自动开始，且您将得到提示，询问您希望如何继续进行。
3. 安装完成后，运行安装好的程序，确信其运转如常。

允许用户删除应用程序

当用户安装应用程序时，安装程序将应用程序删除实用程序复制到 \Windows 或 \Winnt 目录。每次使用 Visual Basic 安装程序来安装应用程

序时，都会在应用程序的安装目录中生成一个删除日志文件(St6unst.log)。

该 .log 文件包含的项说明了：

- 在安装过程中创建的目录。
- 所安装的文件及其位置。该列表包含了安装程序中的所有文件，即使有些文件由于已经有一个更新的版本而没有安装到用户机器上。日志文件指明了该文件是否是共享文件，是否替代了已有文件。
- 创建或修改的注册表项。
- Windows 95 或 Windows NT 4.0 以及更新版本所创建的“链接”和“开始”菜单项。
- 自注册的 .dll、.exe 或 .ocx 文件。

在 Windows 95 以及 Windows NT 4.0 或更新版本中，安装程序将应用程序删除实用程序添加到“控制面板”的“添加/删除程序”部分所显示的已注册应用程序的列表中。最终用户应使用“添加/删除程序”来卸载应用程序。

警告 很重要的一点是，要确保已经正确地为所有被共享的文件设置了选项，不管是通过添加文件到打包和部署向导的“共享文件”屏幕，还是为该文件指明安装位置来作为共享文件目录。如果您意外地安装了一个没有正确设置值的共享文件，则当用户卸载该应用程序时，就可能删除该文件，这样有可能导致系统的其它应用程序出现问题。

在安装失败或被取消的事件中，应用程序删除实用程序将自动删除安装期间安装程序创建的所有的目录、文件以及注册表项。

在 Windows 95 或更新版本以及 Windows NT 中，在注册表中对共享文件

的引用进行计数。例如，一个由三个应用程序所使用的共享文件其引用计数为 3。当删除一个使用共享文件的应用程序时，该共享文件的引用计数减 1。如果该文件的引用计数变成了 0，则将提示用户是否真的删除该项。

应用程序删除实用程序什么情况下会失败

如果使用应用程序删除实用程序来完全卸载应用程序，则安装程序所创建的日志文件和注册表项必须准确，且从安装时刻以后没有被更改。

如果存在下述任何情形，则应用程序删除实用程序可能会失败或工作不正确：

- 最终用户手工复制共享文件。在这种情况下，注册表中的引用计数不会被更新，因此系统无法正确地判断何时删除该文件。
- 最终用户删除安装的文件或一个应用程序目录，而不是使用应用程序删除实用程序。这样可能会删除日志文件，因此就不能使用应用程序删除实用程序。此外，也使得无法删除系统注册表中有关应用程序目录中的程序文件、DLL 或 .ocx 文件的注册表项，因为这些文件必须按顺序运行来删除。
- 应用程序的安装程序与 Windows 95 中所安装的共享文件不兼容，该共享文件与您的应用程序是相同的。
- 共享文件被安装到一个不同的目录，而不是硬盘上已有的目录。
- 最终用户将同一个 Visual Basic 应用程序安装到两个不同的目录。不仅首先安装的应用程序不能运行，而且应用程序的删除信息将造成冲突。

最终用户在将应用程序安装到不同的目录之前，应先删除先前的安装。

- 最终用户删除了应用程序的安装日志 (St6unst.log)。没有应用程序的安装日志文件，应用程序删除实用程序会因缺少安装信息而失败。

有些情况可能会降低已安装文件的注册表的一致性，导致应用程序删除实用程序过早地得到一个引用计数为 0 的特殊文件，随后就会询问是否要删除该文件。如果过早地删除一个文件，就可能导致其它的应用程序由于缺少从属文件、缺少部件等等而停止运行或运行不正确。

部署本地化的 ActiveX 控件

在大多数方式下，发布本地化的 ActiveX 控件与发布原始版本的控件没有什么不同。不过，如果该控件发行了新版本，并随另一个应用程序在 Web 上发布，就可能会产生问题。

随 Visual Basic 一起提供的 ActiveX 控件已自动被翻译或本地化为与您的 Visual Basic 的语言版相对应的语言。例如，Visual Basic 的日语版中的控件用的是日语，并可用于以该语言发布。当发布这些控件时，打包和部署向导必须确定需要随这些文件一起包装的文件。

注意 本主题只适用于随 Visual Basic 一起发布的 ActiveX 控件。它不适用于使用 Visual Basic 创建的用户控件，除非这些控件包含了 Visual Basic ActiveX 控件。

Visual Basic 的本地化 ActiveX 控件包括两部分：

- .控件的 .ocx 文件，该文件是控件的代码块。Visual Basic 的各种语言版本使用的是同一个 .ocx 文件。
- 附属的 DLL 文件，该文件包含了该控件的本地化字符串，是控件的数据块。不同版本的附属 DLL 根据您所购买的 Visual Basic 语言版本而有所不同。

当包装一个包含本地化的 Visual Basic ActiveX 控件的应用程序时，打包和部署向导将自动包括正确的附属 DLL。当最终用户下载该应用程序时，这个正确的附属 DLL 就被安装到用户的计算机上。

如果在 Internet 发布了该控件的新版本，除非新的、本地化的附属 DLL 也能随第二个应用程序一起下载，该 .ocx 文件和附属 DLL 将不再同步。当出现这种情况时，最终用户会突然发现新控件的字符串要么是错的，要么是英文字符串，因为当找不到兼容的附属 DLL 时，.ocx 缺省设置是使用英文字符串（在 .ocx 文件中始终是可用的）。要解决这个问题，您可以告诉最终用户从 www.microsoft.com/vstudio/ 下载最新版的附属 DLL。

详细信息 请参阅第 16 章“国际化问题”中的“设计国际化软件”，以获得代码块和数据块的说明。

附录 A Visual Basic 的规范、限制和文件格式

本附录描述系统的需求、对 Visual Basic 工程的限制、包含在 Visual Basic 工程中的文件类型以及对窗体文件 (.frm) 与工程文件 (.vbp) 的描述。

注意 当许多限制规定为具体数字时，要注意在这些限制未起作用之前，可能会先受到其它一些条件（例如可用的内存和系统资源）的限制。

目录

- Visual Basic 应用程序的系统需求
- 工程限制
- 工程文件格式
- 窗体结构

Visual Basic 应用程序的系统需求

Visual Basic 应用程序需要的硬件和软件如下：

- Pentium(R) 90MHz 或更高的微处理器。

- V G A 640x480 或 Microsoft Windows 支持的更高分辨率的屏幕。
- Windows 95, 24MB RAM , 对于 Windows NT, 32MB。
- Microsoft Windows NT 3.51 或更新的版本, 或 Microsoft Windows 95 或更新的版本。
- Microsoft Internet Explorer 4.01 或更新的版本(4.01 版的 Service Pack 1 或对 DHTML 应用程序开发者的更高版本, 以及对这些应用程序的最终用户的 4.x 版本)。
- 磁盘控件要求:
 - 标准版本: 典型安装 48MB, 完全安装 80MB。
 - 专业版: 典型安装 48MB, 完全安装 80MB。
 - 企业版: 典型安装 128MB, 完全安装 147MB。
 - 附加部件 (如果需要的话): MSDN (用于文档): 67MB, Internet Explorer 4.x: 大约 66MB。
- CD-ROM (假定没有 MS-DOS 支持)。

工程限制

一个单项工程可以包含多达 32,000 个"标识符"(非保留字), 它们包括 (但不限于) 窗体、控件、模块、变量、常量、过程、函数和对象。注意, 实际的标识符数受可用内存的限制。

Visual Basic 中的变量名不多于 255 个字符, 而窗体、控件、模块和类名

不多于 40 个字符。Visual Basic 不限制工程中不同对象的实际数目。

控件限制

每个非图形的控件（除形状、直线、图像和标签之外的所有控件）都使用一个窗口。每个窗口都使用系统资源，所以可同时存在的窗口总数受到可用资源的限制。确切的限制取决于可用的系统资源和使用的控件类型。为了减低系统资源的消耗，在创建或显示图形时，可以使用形状、直线、标签和图像控件而不用图片框控件。

控件总数

单个窗体允许使用的最大控件数目取决于使用的控件类型和可用的系统资源。然而，有每个窗体只允许 254 个控件名的固定限制。因为数组中的所有控件共享一个控件名，所以控件数组仅对这一限制计数一次。在所有的版本中，对控件数组索引的限制是在 0 到 32,767 的范围之内。如果在每个控件的顶层上再安放控件，例如在框架中嵌套框架控件，Visual Basic 一般只接受不多于 25 层的嵌套控件。

对特殊控件的限制

下表列出了用于 Visual Basic 中特殊控件的属性限制。

属 性	应 用 到	限 制
List	和 列表框控件和组合框控件	项数最多为 32K；对每项大小的限制为 1K（1024 字节）
ListCount		
Text	文本框控件	限制为 64K
Caption	标签控件	限于 1024 字节
	命令按钮控件、复选框控件、框架控件和选项按钮控件	限于 255 个字符。任何超过这些限制的标题均被截尾。自定义控件属性的标题限于 32K
	菜单控件	限于 235 个字符
Tag	所有控件	仅限于可用的内存大小
Name	所有控件	限于 40 个字符

注意 在 Visual Basic 中，控件属性名限于 30 个字符。

代 码 限 制

可被加载到窗体、类或标准模块的代码总数限于 65,534 行。一行代码限于 1023 个字节。在一行中的实际文本之前最多只能有 256 个空格的前导，在一个逻辑行中最多只能有 25 个续行符（_）。

过程、类型和变量

对每个模块的过程数没有限制。每个过程可包含至多 64K 的代码。如果过程或模块超过这一限制，Visual Basic 便产生编译时间错误。如果遇到这种错误，可将特别大的过程分割成若干个较小的过程，或将模块级声明移到另一模块，来避免此类错误发生。

Visual Basic 用表来保存代码中的标识符名（变量、过程、常量等）。每个表限于 64K。

动态链接库声明表

每个窗体和代码模块使用一个描述动态链接库入口点的结构的表。每个结构约 40 个字节，表的大小限于 64K，形成每个模块大约有 1500 个声明。

工程名表

整个应用程序用一张包含所有名称的表。这些名称包括：

- 常量名
- 变量名
- 自定义的类型定义名
- 模块名
- DLL 过程声明名

对工程名表总大小没有限制，但是区分大小写的条目不得超过 32K。如果超过了 32K 的限制，可以在不同的模块中重新使用 `private` 标识符以限制区分大小写的条目数到达 32K。

输入表

在不同的模块中每引用一个标识符，便在输入表中创建一个条目。每一个这样的入口最小是 24 字节，但限于 64K，这样每个模块大约可以有 2000 个引用。

模块条目表

该表中每个模块最多达 125 个字节，但有 64K 的总限制，这样每个工程大约可以产生 400 个模块。

数据限制

应用于 Visual Basic 语言中变量的限制有下面各种。

窗体、标准和类模块数据

Visual Basic 中任何窗体或模块的 VBA 模块数据段（即在声明部分中定义

的数据) 有多达 64K 的数据段, 该数据段包括下列数据:

- 用 Static 声明的局部变量
- 不是数组和变长字符串的模块级变量
- 每个模块数组和变长字符串占 4 个字节

过程、类型和变量

如果过程或模块超过 64K 的代码限制, Visual Basic 便产生一个编译时错误。

如果一个过程中定义的局部变量超过 64K, 则产生“太多的局部非静态变量”错误。

如果一个模块中定义的模块级变量超过 64K, 或定义的自定义类型大于 64K, 则产生“固定或静态数据不能超过 64K”错误。

如果遇到这类错误, 可将一个太大的过程分割成若干较小的过程, 或将模块级声明移到另一模块中, 来避免这种错误的发生。

作为变量声明的数组不能使用全部数组空间; 只有数组描述符对 64K 的限制起作用。例如, 在过程中声明或作为模块级声明, `Dim x(1000000) As Byte`。然而, 如果在记录中声明了一个大的、固定大小的数组, 然后将这些记录的示例声明为变量时, 就会发生内存溢出。

自定义的类型

虽然许多自定义类型的变长字符串的总和可以超过 64K (每个自定义类型

的变长字符串仅占 4 个字节；而字符串的实际内容是分别存储的）但自定义类型的变量不能超过 64K。自定义类型可用其它自定义类型来定义，但这种类型的总大小不能超过 64K。

堆栈空间

在运行时过程中的参数和局部变量占用堆栈空间。而模块级变量和静态变量不占堆栈空间，因为它们被分配在窗体或模块的数据段中。任何被调用的 DLL 过程，在执行时都要使用堆栈。

Visual Basic 自己也使用堆栈，例如，在计算表达式时保存中间值。

Visual Basic 全部可用堆栈的大小是每线程一兆字节（1MB）。然而，如果有相邻的闲置内存，堆栈的大小可以超过这个限定。

详细信息 关于保留堆栈空间的提示，请参阅第十五章“性能和兼容性设计”。

系统资源限制

对于 Visual Basic 和用它建立的应用程序的某些限制，是受 Microsoft Windows 的影响。这些限制会因 Microsoft Windows 的版本不同而不同。

窗口资源

每个打开的窗口都要使用系统资源（Microsoft Windows 使用的数据区）。如果超出系统资源范围运行，就会产生运行时错误“Windows 运行的可用资源不足”。可以从 Windows NT 3.51 的“程序管理器”或“文件管理器”的“帮助”菜单中选取“关于”，或者从 Windows 95 和 Windows NT 4.0 的“Windows 资源管理器帮助”菜单中选取“关于”，来检测剩余的系统资源的百分比。应用程序也能调用 Windows 的 API `GetFreeSystemResources` 来回收系统资源、关闭窗口（例如打开的窗体和“代码”窗口，以及在其它应用程序中的窗口）并退出运行的应用程序。

工程文件格式

Microsoft Visual Basic 在设计和运行时都利用和创建一些文件。工程或应用程序需要哪些文件取决于它的范围和功能。

工程文件的扩展名

Visual Basic 在创建和编译工程时要产生许多文件。这些文件分类如下：设计时文件、杂项开发文件和运行时文件。

设计时文件是工程的建造块：例如基本模块（.bas）和窗体模块（.frm）。

杂项文件是由 Visual Basic 开发环境中的各种不同的进程和函数产生的：例如打包和部署向导从属文件 (.dep)。

设计时和杂项文件

开发应用程序时会产生各种设计时文件和其它杂项文件，如下表所列：

扩展名	描述
.bas	基本模块
.cls	类模块
.ctl	用户控件文件
.ctx	用户控件的二进制文件
.dca	活动的设计器的高速缓存
.ddf	打包和部署向导 CAB 信息文件
.dep	打包和部署向导从属文件
.dob	ActiveX 文档窗体文件
.dox	ActiveX 文档二进制窗体文件
.dsr	活动的设计器文件
.dsx	活动的设计器的二进制文件
.dws	部署向导脚本文件
.frm	窗体文件
.frx	二进制窗体文件
.log	加载错误的日志文件

续表

.oca	控件类型库缓存文件
.pag	属性页文件
.pgx	二进制属性页文件
.res	资源文件
.tlb	远程自动化类型库文件
.vbg	Visual Basic 组工程文件
.vbl	控件许可文件
.vbp	Visual Basic 工程文件
.vbr	远程自动化注册文件
.vbw	Visual Basic 工程工作空间文件
.vbx	向导发射文件
.wct	WebClass HTML 模板

运行时文件

编译应用程序时，所有必须的设计时文件都被包括在运行时可执行文件中，运行时文件在下表中列出：

扩展名	描述
.dll	运行中的 ActiveX 部件
.exe	可执行文件或 ActiveX 部件
.ocx	ActiveX 控件

续表

.vbd	ActiveX 文档状态文件
.wct	WebClass HTML 模板

窗体结构

在典型的 visual Basic 工程中的许多文件都是二进制格式的，并且只有 Visual Basic 或应用程序的指定进程和函数才能读取，这时窗体文件 (.frm) 与工程文件 (.vbp) 都是 ASCII 文本文件。这些文件可在文本查阅器（如 Notepad）中阅读。

下面几节描述在典型的 Visual Basic 工程中的设计时文件和运行时文件，以及窗体文件 (.frm) 和工程文件 (.vbp) 的格式。

Visual Basic 窗体文件 (.frm) 创建并保存为 ASCII 格式。窗体结构由以下部分组成：

- 文件格式的版本号
- 包括窗体描述的文本块
- 窗体属性集
- 窗体的 Basic 代码

窗体描述包括窗体属性的设置值。定义窗体上的控件属性的文本块是嵌套在窗体中的。包括在其它控件中的控件有它们的属性，这些属性被嵌套在容器的文本中。图 A.1 说明了窗体描述的结构。

打印窗体描述

要打印窗体描述，无需保存窗体即可打印。

要打印窗体描述，请按照以下步骤执行：

1. 从“文件”菜单中选取“打印”。
2. 选择“窗体为文本”复选框并选取“确定”打印窗体描述。

```

Version 6.00
Begin VB.Form formname
[
    Form Properties
    [
        Begin VB.controltype controlname
        [
            Control Properties
            [
                Begin VB.controltype controlname
                [
                    Control Properties
                ]
            ]
        ]
        End
        [
            Begin VB.controltype controlname
            [
                Control Properties
            ]
        ]
        End
    ]
    End
End
Attributes

```

这个窗体的基本代码从这里开始。

图 A.1 窗体描述的结构

版本号

Visual Basic for Windows 所创建的窗体的版本号是 6.00。如果窗体中遗漏了版本号，就会产生错误。当加载的 Visual Basic 应用程序的版本号小

于 6.00 时，会出现一个警告对话框，告诉您该文件将被保存为新的格式。

窗体描述

窗体描述由 **Begin** 语句开始，用 **End** 语句结束。**Begin** 语句的语法是：

```
Begin VB.{Form|MDIForm} formname
```

End 语句决定结束窗体描述和开始窗体属性设置的地方。没有 **End** 语句，**Visual Basic** 会把属性当作窗体的控件与属性的描述，因而产生了错误。

在 **Begin Form** 语句和 **End** 语句之间是窗体自身的属性，之后是关于窗体每一个控件的描述。图 A.2 详细说明了窗体描述的嵌套结构。

```

Version 6.00
Begin VB.Form MyForm1
    BackColor = &H00FF0000&
    Caption = "Form1"
    Height = 6684
    Left = 828
    .
    .
    .
    Begin VB.Frame Frame1
        Caption = "Frame1"
        Height = 1692
        .
        .
        .
        Begin VB.CommandButton MyButton
            Caption = "Start"
            Height = 372
            .
            .
            .
        End
    End
End

```

控件代码块

图 A.2 窗体描述的嵌套结构

控件块

控件块由窗体描述中的文本组成，它定义了单独控件的属性。和窗体描述一样，控件块也是由 `Begin` 语句开始，用 `End` 语句结束。控件块的 `Begin`

语句的语法如下：

Begin *controlclass.controltype controlname*

控件属性出现在 **Begin** 语句和 **End** 语句之间。

控件块顺序

控件块的顺序决定控件的 **z** 顺序。**z** 顺序是决定窗体上各控件如何互相覆盖的相对顺序。窗体描述中的第一个控件建立了 **z** 顺序的底层。控件在窗体描述中出现得越靠后，它在 **z** 顺序中的层次便越高，它能覆盖层次比它低的控件。

嵌入式控件块

有些控件可以包含其它控件。当一个控件被包含在另一个控件中时，它的控件块就被嵌入了容器的控件块之中。可将控件块嵌入：

- 框架
- 图片框
- 菜单
- 自定义控件，取决于它们的用途

嵌入的控件通常用于在框架内放置选项按钮。在添加任何被包含的控件之前，**Visual Basic** 必须得到容器的全部信息，因此，控件的属性必须先于被嵌入的控件块到达。**Visual Basic** 忽略被嵌套的控件块出现之后到达控件块中的全部属性。

菜单控件

所有菜单控件必须在窗体描述结束，属性段刚要开始之前一块儿出现。Visual Basic 在加载 ASCII 窗体中遇到菜单控件时，它希望同时找到全部菜单控件。一旦检测到非菜单控件跟在一个或多个菜单控件之后，Visual Basic 便认为在窗体上不会再有其它菜单控件，如果在加载窗体期间再遇到任何其它菜单控件，将忽略它。

快捷键

快捷键是用于激活菜单控件的键。ASCII 窗体使用与 SendKeys 语句相同的语法定义键组合：“+”= SHIFT, “^”= CTRL, “{Fn}”= 函数键（其中 n 是键号）。字母字符代表其原意。快捷键的语法是：

```
Shortcut = ^{F4}      ' <CTRL><F4>
```

注意 顶层菜单没有快捷键

详细信息 请参阅“SendKeys 语句”。

窗体描述中的注释

可以在窗体描述中添加注释。单引号（'）是注释的定界符。

警告 在 Visual Basic 中保存窗体时，不保留窗体描述中的注释和格式部分。而在窗体文件的代码段中的注释和缩进却被保存。

窗体描述属性

当 Visual Basic 保存窗体时，它就以缺省的次序排列这些属性。当然，在创建窗体时可以任意顺序列出这些属性。

加载时，未列于表中的所有属性被设置为它的缺省值。当 Visual Basic 保存窗体时，它仅包含不用缺省值作为它的设置值的属性。每个控件确定了是保存全部属性，还是仅保存那些设置值不是缺省值的属性。

语法

在窗体描述中用此语法定义属性：

```
property = value
```

Text 属性值必须出现在双引号内，Boolean 属性值为 -1 表示 True，为 0 表示 False。Visual Basic 将 -1 和 0 以外的任何值均解释为 True。带列出值的属性包含它们的数值，以及包含在注释内的对于该值的描述。例如 BorderStyle 属性的表示如下：

```
BorderStyle = 0           ' None
```

二进制属性值

一些控件具有以二进制数为其值的属性，比如图片框和图像控件的 Picture 属性或某些自定义控件的属性。Visual Basic 将窗体的全部二进制数据保存

到独立于窗体的二进制数据文件中。

Visual Basic 将二进制数据文件与窗体保存到相同的目录中。这个二进制文件和窗体同名，而以 .frx 为扩展名。当加载窗体时，Visual Basic 读此二进制数据文件。当 Visual Basic 加载它时，二进制数据文件 (.frx) 必须是对窗体可用的。当与其它使用二进制数据文件的应用程序共享窗体时，则必须确保同时提供二进制数据文件 (.frx) 和窗体文件 (.frm)。

有二进制数据的属性，当它们的值出现在窗体上时，就作为在二进制数据文件中的字节偏移量来引用。例如，在窗体描述中，Picture 属性的值如下表示：

```
Begin VB.Image imgDemo
    Picture = "Myform.frx":02EB
End
```

此属性意味着定义这个控件的 Picture 属性的二进制数据开始于文件 Myform.frx 中的 2EB（十六进制数）字节处。

Icon 属性

窗体中图标属性的值取决于窗体使用哪种图标。下表列出图标属性值以及这个属性如何出现在窗体中。

Icon 属性设置	ASCII 窗体目录
缺省图标	不引用 Icon 属性

续表

(None)

除缺省外的任意图标

`Icon = 0`

二进制数据文件的字节偏移量引用。例如：

`Icon = "Myform.frx":0000`

TabIndex 属性

如果没有规定 `TabIndex` 属性，一旦所有其它控件加载之后，`Visual Basic` 将 `Tab` 顺序中最早的可能位置赋予该控件。

度量单位

控件大小、`X` 和 `Y` 座标以及其它使用度量单位的属性值均用缇来表示。如果控件使用缇以外的刻度模式，`Visual Basic` 在 `ASCII` 窗体中加载窗体时将缇值转变为由控件的 `ScaleMode` 属性指定的度量单位。

颜色值

颜色值表示为 `RGB` 值。例如，`ForeColor` 属性表示为：

`ForeColor = &H00FF0000&`

`Visual Basic` 也可以读 `QBColor` 值，当加载窗体时，将它们变为 `RGB`。用 `QBColor` 值的 `ASCII` 窗体必须用这样的语法：

`ForeColor = QBColor(qbcolor)`

这里 *qbcolor* 是从 0 到 15 的值。

注意 *qbcolor* 参数相当于其它 Basic 版本（例如 Visual Basic for MS-DOS、Microsoft QuickBasic 和 Microsoft Basic Professional Development System）中图形语句使用的颜色值。

Property 对象

有些属性对象，如 Font 对象，以独立块的形式出现，表示该对象各种属性的全部设置值。这些块包含在如下形式的 **BeginProperty** 和 **EndProperty** 语句之间：

BeginProperty *propertyname*

property1 = value1

property2 = value2

EndProperty

Basic 代码

在窗体描述的最后一个 **End** 语句之后的是属性部分，而属性部分之后紧接着的是 Basic 编码。窗体声明部分的语句首先出现，随后是事件过程、普通过程和函数。

ASCII 窗体示例

图 A.3 示出了取自 Blanker 应用程序示例的 Blanker.frm 窗体。

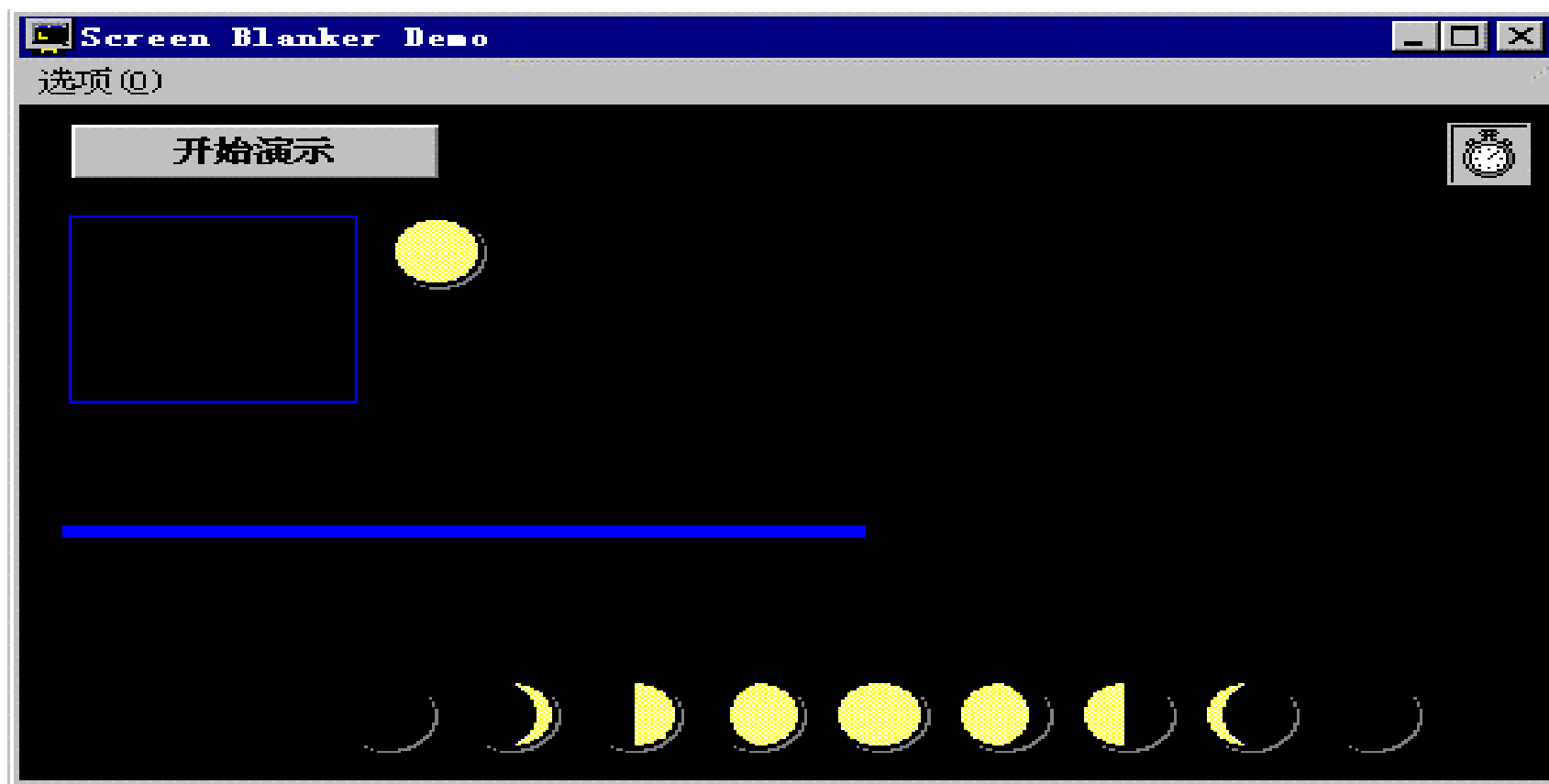


图 A.3 取自 Blanker.vbp 应用程序示例的窗体示例

这里是保存在 Visual Basic 中的 Blanker 窗体的部分。移动到保存空间的窗体部分用竖的省略号表明。

VERSION 6.00

Begin VB.Form DemoForm

```
BackColor = &H00000000&
Caption = "Screen Blanker Demo"
ClientHeight = 960
ClientLeft = 1965
ClientTop = 1965
ClientWidth = 7470
ForeColor = &H00000000&
Begin Property Font
    name = "MS Sans Serif"
    charset = 0
    .
    .
    .
End Property
Height = 5115
Icon = "Blanker.frx":0018
Left = 900
LinkMode = 1
LinkTopic = "Form1"
ScaleHeight = 4425
ScaleWidth = 7470
Top = 1335
```

'源

Width = 7590

Begin VB.Timer Timer1

Interval = 1

Left = 6960

Top = 120

End

Begin VB.CommandButton cmdStartStop

BackColor = &H00000000&

Caption = "Start Demo"

Default = -1

Height = 390

Left = 240

TabIndex = 0

Top = 120

Width = 1830

End

Begin VB.PictureBox picBall

AutoSize = -1 ' True

BackColor = &H00000000&

BorderStyle = 0 ' None

ForeColor = &H00FFFFFF&

Height = 465


```

    Left = 1800
    Picture = "Blanker.frx":0788
    ScaleHeight = 465
    ScaleWidth = 465
    TabIndex = 1
    Top = 720
    Visible = 0                                ' False
    Width = 465
End
.
.
.
Begin VB.Menu mnuOption
    Caption = "&Options"
    Begin VB.Menu mnuLineCtlDemo
        Caption = "&Jumpy Line"
        Checked = -1                            ' True
    End
    Begin VB.Menu mnuCtlMoveDemo
        Caption = "Re&bound"
    End
End
.

```

```

        .
        .
        Begin VB.Menu mnuExit
            Caption = "E&xit"
        End
    End
End

.
.
.
Attribute VB_Name = "DemoForm"
Attribute VB_Creatable = False
Attribute VB_Exposed = False
Dim Shared FrameNum
Dim Shared XPos
Dim Shared YPos
Dim Shared DoFlag
Dim Shared Motion

.
.
.
Sub CircleDemo ()

```

```
Dim Radius
R = 255 * Rnd
G = 255 * Rnd
B = 255 * Rnd

XPos = ScaleWidth / 2
YPos = ScaleHeight / 2
Radius = ((YPos * 0.9) + 1) * Rnd
Circle (XPos, YPos), Radius, RGB (R, G, B)
End Sub

.
.
.
Private Sub Timer1_Timer ()
.
.
.
End Sub
```

加载窗体文件时的错误

当 Visual Basic 将窗体加载到内存时，首先将窗体变为二进制格式。如果

对窗体做变更，在保存这些变更时，Visual Basic 就以 ASCII 格式重写该文件。

在加载窗体过程中遇到错误时，Visual Basic 会建立一个日志文件，并在此日志文件中报告这个错误。每当遇到窗体中的错误时，Visual Basic 就把错误消息添加到日志文件中。窗体加载完成时，Visual Basic 会显示一个信息，通知错误日志文件已经建立。

日志文件与窗体文件同名，带有 .log 文件扩展名。例如，当加载 Myform.frm 时发现错误，Visual Basic 将建立一个名为 Myform.log 的日志文件。如果以后加载 Myform.frm 并又发生错误，则 Visual Basic 用新的日志文件置换旧的文件。

窗体加载错误日志消息

下面是出现在错误日志文件上的错误消息。注意，这些错误消息仅涉及 Visual Basic 加载窗体描述时可能发生的问题。而不是在事件过程、普通过程或 Basic 编码的其它部分可能存在的问题。

不能加载菜单 **menuname**

此消息表明 Visual Basic 发现菜单控件的父菜单已被定义为菜单分隔符。菜单分隔符不能作为父菜单控件，Visual Basic 不能加载这样的菜单控件。此消息还表明 Visual Basic 发现某菜单控件，其父菜单的 Checked 属性已置为 True。父菜单控件不能是复选的，Visual Basic 不能加载这样的菜单控件。

在菜单 **menuname** 中不能设置 **Checked** 属性

此消息表明 Visual Basic 发现顶层菜单控件的 **Checked** 属性已被设置为 **True**。顶层菜单控件不能有复选标记。Visual Basic 加载此菜单控件，但不设置它的 **Checked** 属性。

在 **menuname** 中不能设置 **Shortcut** 属性

此消息表明 Visual Basic 发现顶层菜单控件有定义的快捷键。顶层菜单不能有快捷键。Visual Basic 加载此菜单控件，不设置 **Shortcut** 属性。

控件 **controlname** 中的类 **classname** 不是已加载的控件类

此消息表明 Visual Basic 发现不能辨认的类名。

不能加载控件 **controlname**

此消息表明 Visual Basic 在窗体描述中遇到了不认识的控件类型。Visual Basic 建立一个图片框来表示不认识的控件，从不认识的控件描述中赋予此图片框任何有效的属性。当此信息出现时，可能跟随一些无效属性错误。

控件 **controlname** 有一用引号括起来的字符串，此处本应是属性名

此消息表明 Visual Basic 发现属性名称被双引号内的文本取代了，属性名称是不能放在双引号内的。例如：

```
"Caption" = "Start Demo"
```

这里，属性名称 **Caption** 不能放在双引号内。Visual Basic 会跳过窗体描述中会产生错误的这一行。

控件名称 **controlname** 无效

此消息表明控件名称不是 Visual Basic 中的有效字符串。Visual Basic 将不加载此控件。

控件名太长；**controlname** 被截尾

此消息表明 Visual Basic 发现控件名称长度超过了 40 个字符。Visual Basic 将名称截短后加载控件。

没有发现 **index** 属性且控件 **controlname** 已存在。不能创建此控件

此消息表明 Visual Basic 发现一个没有索引的控件，它与先前已加载的控件同名。Visual Basic 不加载此控件。

窗体 **formname** 不能被加载

此消息表明 Visual Basic 遇到了不正确的文件尾或者第一个 **Begin** 语句已经丢失。

窗体或 MDIF 窗体名称 **formname** 不合法；不能加载此窗体

此消息表明窗体名称不是 Visual Basic 中有效的字符串。Visual Basic 不加载此窗体。

有效字符串必须以字母开头，只能包含字母、数字及下划线；最多只能 40 个字符。

控件 **controlname** 中的属性名称 **propertyname** 无效

此消息表明属性名称不是 Visual Basic 中的有效的字符串，或者多于 30 个字符。Visual Basic 将不设置此属性。

控件 **controlname** 中的属性 **propertyname** 不能被加载

此消息表明 Visual Basic 遇到了不认识的属性。加载窗体时，Visual Basic 跳过此属性。

控件 **controlname** 中的属性 **propertyname** 不能被设置

此消息表明 Visual Basic 不能按照窗体描述指示的那样设置指定控件的这

个属性。

控件 `controlname` 中属性 `propertyname` 的值无效

此消息表明 Visual Basic 遇到了无效的属性值。Visual Basic 将此属性值变为缺省值。

控件 `controlname` 中属性 `propertyname` 有无效的文件引用

此消息表明 Visual Basic 不能使用文件名引用。这发生在指定的目录中找不到这个引用的文件时（可能是窗体的二进制数据文件）。

控件 `controlname` 中的属性 `propertyname` 的属性索引无效

此消息表明 Visual Basic 发现属性名称的属性索引大于 255。例如：

```
Prop300 = 5436
```

Visual Basic 跳过窗体描述中会产生错误的这一行。

控件 `controlname` 中属性 `propertyname` 的值无效

此信息表明 Visual Basic 发现了控件的某个属性值不正确。例如：

```
Top = Cahr(22)      '实际上应是 Char(22).
```

Visual Basic 将此属性设置为缺省值

控件 `controlname` 中属性 `propertyname` 必须是双引号括起来的字符串

此时信息表明 Visual Basic 发现了没有双引号的属性值，而此值是应该在双引号内的。例如：

```
Caption = Start Demo
```

Visual Basic 跳过窗体描述中会产生错误的这一行。

语法错误：控件 `controlname` 中的属性 `propertyname` 缺少等号 '='

此信息表明 Visual Basic 发现在属性名称和值之间没有等号。例如：

Text "Start Demo"

Visual Basic 不加载此属性

详细信息 关于窗体加载期间遇到的错误的附加信息，请在“错误”对话框内单击“帮助”按钮，或者按 F1 键。

工程文件 (.vbp) 格式

Visual Basic 总是以 ASCII 格式保存工程文件 (.vbp)。工程文件包含了反映工程的设置的项目。这些项目包括工程中的窗体和模块、引用以及为控制编译而选取的各种各样的选项等等。

这是 .vbp 文件的示例，此工程包括用下表所示的类名和文件名保存的模块。

模块类型	类名	文件名
MDI 窗体	A form	A_Form.frm
窗体	B form	B_Form.frm
标准模块	C module	C_Module.bas
类模块	D class	D_Class.cls

Type=Exe

Form=B_Form.frm

Reference=*\G{00020430-0000-0000-C000-

000000000046}#2.0#0#..\..\WINDOWS\SYSTEM\STDOLE2.TLB#OLE Automation


```
Form=A_Form.frm
Module=CModule; C_Module.bas
Class=DClass; D_Class.cls
Startup="BForm"
Command32=""
Name="Project1"
HelpContextID="0"
CompatibleMode="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="Microsoft"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FltPointCheck=0
```

FDIVCheck=0

UnroundedFP=0

StartMode=0

Unattended=0

ThreadPerObject=0

MaxNumberOfThreads=1

当在工程中添加窗体、模块、部件等时，.vbp 文件中的项目也增加了。在工程中设置选项也会使项目增加。许多选项都是用“工程属性”对话框设置的。

详细信息 关于“工程属性”对话框中的选项的详细信息，请按 F1 键。

附录 B Visual Basic 编码约定

该附录给出一系列推荐使用的 Visual Basic 程序的编码约定。

编码约定是编程准则，它注重于程序的物理结构和外观，而不是程序的逻辑结构。它们使编码更加容易阅读、容易理解而且容易维护。编码约定可以包括：

- 对象、变量及过程的命名约定。
- 标记和注释编码的标准化格式。
- 空格、格式化及缩排的准则。

在后面的几节中将讨论这些方面，同时给出好的习惯用法的例子。

目录

- 为什么要进行编码约定
- 对象命名约定
- 常量和变量命名约定
- 结构化的编码约定

为什么要进行编码约定

使用统一编码约定集的主要原因，是使应用程序的结构和编码风格标准化，以便于阅读和理解这段编码。

好的编码约定可使源代码严谨、可读性强且意义清楚，与其它语言约定相一致，并且尽可能的直观。

最小编码约定

一组通用目的的编码约定应该定义完成上述目的所必需的、能让程序员自由地创建程序逻辑和功能流程的最小的要求。

最小编码约定的目的是使程序易于阅读和理解，而不是用过份的约束和绝对的限制来束缚程序员本身的创造性。

所以，该附录中提出的约定是简洁和建意性的。这些约定中没列出每一个可能的对象或控件，也没列出每种有用的信息注释。根据工程及机构的特殊要求，也许希望扩充这些准则，以包含附加的元素，如：

- 对于在家庭中开发的或从第三方厂商购买的特定对象及组件的约定。
- 描述机构的商业活动或设备的变量。
- 工程或企业认为对清楚性和可读性很重要的任何其它元素。

详细信息 关于对过程、变量及常量命名的限制的详细信息，请参阅第五章“编程基础”中的“编码基础”。

对象命名约定

应该用一致的前缀来命名对象，使人们容易识别对象的类型。下面列出了 Visual Basic 支持的一些推荐使用的对象约定。

推荐使用的控件前缀

控件类型	前缀	例子
3D Panel	pnl	pnlGroup
ADO Data	ado	adoBiblio
Animated button	ani	aniMailBox
Check box	chk	chkReadOnly
Combo box, drop-down list box	cbo	cboEnglish
Command button	cmd	cmdExit
Common dialog	dlg	dlgFileOpen
Communications	com	comFax
Control （当特定类型未知时，在过程中所使用的）	ctr	ctrCurrent
Data	dat	datBiblio
Data-bound combo box	dbcbo	dbcboLanguage

续表

Data-bound grid	dbgrd	dbgrdQueryResult
Data-bound list box	dblst	dblstJobType
Data combo	dbc	dbcAuthor
Data grid	dgd	dgdTitles
Data list	dbl	dblPublisher
Data repeater	drp	drpLocation
Date picker	dtp	dtpPublished
Directory list box	dir	dirSource
Drive list box	drv	drvTarget
File list box	fil	filSource
Flat scroll bar	fsb	fsbMove
Form	frm	frmEntry
Frame	fra	fraLanguage
Gauge	gau	gauStatus
Graph	gra	graRevenue
Grid	grd	grdPrices
Hierarchical flexgrid	flex	flexOrders
Horizontal scroll bar	hsb	hsbVolume
Image	img	imgIcon
Image combo	imgcbo	imgcboProduct

续表

ImageList	ils	ilsAllIcons
Label	lbl	lblHelpMessage
Lightweight check box	lwchk	lwchkArchive
Lightweight combo box	lwcbo	lwcboGerman
Lightweight command button	lwcmd	lwcmdRemove
Lightweight frame	lwfra	lwfraSaveOptions
Lightweight horizontal scroll bar	lwhsb	lwhsbVolume
Lightweight list box	lwlst	lwlstCostCenters
Lightweight option button	lwopt	lwoptIncomeLevel
Lightweight text box	lwtxt	lwoptStreet
Lightweight vertical scroll bar	lwvsb	lwvsbYear
Line	lin	linVertical
List box	lst	lstPolicyCodes
List View	lvw	lvwHeadings
M API message	mpm	mpmSentMessage
M API session	mps	mpsSession
MCI	mci	mciVideo
Menu	mnu	mnuFileOpen
Month view	mvw	mvwPeriod
MS Chart	ch	chSalesbyRegion

续 表

MS Flex grid	msg	msgClients
MS Tab	mst	mstFirst
OLE container	ole	oleWorksheet
Option button	opt	optGender
Picture box	pic	picVGA
Picture clip	clp	clpToolbar
ProgressBar	prg	prgLoadFile
Remote Data	rd	rdTitles
RichTextBox	rtf	rtfReport
Shape	shp	shpCircle
Slider	sld	sldScale
Spin	spn	spnPages
StatusBar	sta	staDateTime
SysInfo	sys	sysMonitor
TabStrip	tab	tabOptions
Text box	txt	txtLastName
Timer	tmr	tmrAlarm
Toolbar	tlb	tlbActions
TreeView	tre	treOrganization
UpDown	upd	updDirection

Vertical scroll bar

vsb

vsbRate

推荐使用的数据访问对象 (DAO) 的前缀

用下列前缀来指示数据访问对象。

数据库对象	前 缀	例 子
Container	con	conReports
Database	db	dbAccounts
DB Engine	dbe	dbeJet
Document	doc	docSalesReport
Field	fld	fldAddress
Group	grp	grpFinance
Index	ix	idxAge
Parameter	prm	prmJobCode
QueryDef	qry	qrySalesByRegion
Recordset	rec	recForecast
Relation	rel	relEmployeeDept
TableDef	tbd	tbdCustomers
User	usr	usrNew
Workspace	wsp	wspMine

一些例子：

```

Dim dbBiblio As Database
Dim recPubsInNY As Recordset, strSQLStmt As String
Const DB_READONLY = 4           ' 设置常量。
' 打开数据库。
Set dbBiblio = OpenDatabase("BIBLIO.MDB")
' 为 SQL 语句设置文本。
strSQLStmt = "SELECT * FROM Publishers WHERE _
    State = 'NY'"
' 创建新的记录集对象。
Set recPubsInNY = db.OpenRecordset(strSQLStmt, _
    dbReadOnly)

```

推荐使用的菜单前缀

应用程序频繁使用许多菜单控件，对于这些控件具备一组唯一的命名约定很实用。除了最前面 "mnu" 标记以外，菜单控件的前缀应该被扩展：对每一级嵌套增加一个附加前缀，将最终的菜单的标题放在名称字符串的最后。下表列出了一些例子。

菜单标题序列	菜单处理器名称
File Open	mnuFileOpen
File Send Email	mnuFileSendEmail

续表

File Send Fax	mnuFileSendFax
Format Character	mnuFormatCharacter
Help Contents	mnuHelpContents

当使用这种命名约定时，一个特定的菜单组的所有成员一个接一个地列在 Visual Basic 的“属性”窗口中。而且，菜单控件的名字清楚地表示出它们所属的菜单项。

为其它控件选择前缀

对于上面没有列出的控件，应该用唯一的由两个或三个字符组成的前缀使它们标准化，以保持一致性。只有当需要澄清时，才使用多于三个字符的前缀。

例如，对于派生的或修改的控件象上述那样扩展其前缀，使得在真正使用了哪一个控件的问题上避免混淆。对于第三方控件，应该把制造商的小写缩写名附加到前缀中。例如，从 Visual Basic Professional 3D 框架中创建一个控件实例可以用 fra3d 这样的前缀，以避免混淆所使用的控件。

常量和变量命名约定

除了对象之外，常量和变量也需要良好格式的命名约定。本节列出了 Visual

Basic 支持的常量和变量的推荐约定。并且讨论标识数据类型和范围的问题。

变量应该总是被定义在尽可能小的范围内。全局 (Public) 变量可以导致极其复杂的状态机构，并且使一个应用程序的逻辑非常难于理解。全局变量也使代码的重用和维护更加困难。

Visual Basic 中的变量可以有如下列范围：

范围	声明位置	可见位置
过程级	过程，子过程或函数过程中的 'Private'	在声明它的过程中
模块级	窗体或代码模块（.frm、.bas）的声明部分中的 'Private'	窗体或代码模块中的每一个过程
全局	代码模块（.bas）的声明部分中的 'Public'	应用程序中的每一处

在 Visual Basic 的应用程序中，只有当没有其它方便途径在窗体之间共享数据时才使用全局变量。当必须使用全局变量时，在一个单一模块中声明它们，并按功能分组。给这个模块取一个有意义的名称，以指明它的作用，如 Public.bas。

较好的编码习惯是尽可能写模块化的代码。例如，如果应用程序显示一个对话框，就把要完成这一对话任务所需要的所有控件和代码放在单一的窗体中。这有助于将应用程序的代码组织在有用的组件中，并减小它运行时的开销。

除了全局变量（应该是不被传递的），过程和函数应该仅对传递给它们的对象操作。在过程中使用的全局变量应该在过程起始处的声明部分中标识出

来。此外，应该用 `ByVal` 将参数传递给 `Sub` 过程及 `function` 过程，除非明显地需要改变已传递的参数值。

变量范围前缀

随着工程大小的增长，划分变量范围的工作也迅速增加。在类型前缀的前面放置单字母范围前缀标明了这种增长，但变量名的长度并没有增加很多。

范围	前缀	例子
全局	<code>g</code>	<code>gstrUserName</code>
模块级	<code>m</code>	<code>mblnCalcInProgress</code>
本地到过程	无	<code>dblVelocity</code>

如果一个变量在标准模块或窗体模块中被声明为 `Public`，那么该变量具有全局范围。如果一个变量在标准模块或窗体模块中被分别声明为 `Private`，那么该变量有模块级范围。

注意 一致性是卓有成效地使用这种技术的关键；Visual Basic 中的语法检查器不会捕捉以 `"p."` 开头的模块级变量。

常量

常量名的主体是大小写混合的，每个单词的首字母大写。尽管标准 Visual Basic 常量不包含数据类型和范围信息，但是象 `i`、`s`、`g` 和 `m` 这样的前缀对于理解一个常量的值和范围还是很有用的。对于常量名，应遵循与变量

相同的规则。例如：

<code>mintUserListMax</code>	'对用户列表的最大限制 '(整数值，本地到模块)
<code>gstrNewLine</code>	'新行字符 '(字符串，应用程序全局使用)

变量

声明所有的变量将会节省编程时间，因为键入操作引起的错误减少了（例如，究竟是 `aUserNameTmp`，还是 `sUserNameTmp`，还是 `sUserNameTemp`）。在“选项”对话框的“编辑器”标签中，复选“要求变量声明”选项。`Option Explicit` 语句要求在 Visual Basic 程序中声明所有的变量。应该给变量加前缀来指明它们的数据类型。而且前缀可以被扩展，用来指明变量范围，特别是对大型程序。

变量数据类型

用下列前缀来指明一个变量的数据类型。

数据类型	前缀	例子
Boolean	<code>bln</code>	<code>blnFound</code>

续表

Byte	byt	bytRasterData
Collection object	col	colWidgets
Currency	cur	curRevenue
Data(Time)	dtm	dtmStart
Double	dbl	dblTolerance
Error	err	errOrderNum
Integer	int	intQuantity
Long	lng	lngDistance
Object	obj	objCurrent
Single	sng	sngAverage
String	str	strFName
User-defing type	udt	udtEmployee
Variant	vnt	vntChecksum

描述变量和过程名

变量或过程名的主体应该使用大小写混合形式，并且应该足够长以描述它的作用。而且，函数名应该以一个动词起首，如 `InitNameArray` 或 `CloseDialog`。

对于频繁使用的或长的项，推荐使用标准缩略语以使名称的长度合理化。一般来说，超过 32 个字符的变量名在 VGA 显示器上读起来就困难了。

当使用缩略语时，要确保它们在整个应用程序中的一致性。在一个工程中，如果一会儿使用 `Cnt`，一会儿使用 `Count`，将导致不必要的混淆。

用户定义类型

在一项有许多用户定义类型的大工程中，常常有必要给每种类型一个它自己的三个字符的前缀。如果这些前缀是以 "u" 开始的，那么当用一个用户定义类型来工作时，快速识别这些类型是很容易的。例如，`ucli` 可以被用来作为一个用户定义的客户类型变量的前缀。

结构化编码约定

除了命名约定外，结构化编码约定，可以极大地改善代码的可读性，如代码注释和一致性缩进。

代码注释约定

所有的过程和函数都应该以描述这段过程的功能的一段简明注释开始（这段例程干什么）。这种描述不应该包括执行过程细节（它是怎么做），因为这常常是随时间而变的，而且这种描述会导致不必要的注释维护工作，甚至更糟——成为错误的注释。代码本身和必要的嵌入注释将描述实现方法。

当参数的功能不明显且当过程希望参数在一个特定的范围内时，也应描述传递给过程的参数。被过程改变的函数返回值和全局变量，特别是通过引用参数的那些，也必须在每个过程的起始处描述它们。

过程头注释块应该包括下列节标题。关于例子，请参阅下节“格式化代码”。

节标题	注释描述
目的	该过程完成什么（而不是怎么完成）
假设	列出每个外部变量、控件、打开文件或其它不明显元素
效果	列出每个被影响的外部变量、控件或文件及其作用（只有当它不明显时）
输入	每一个可能不明显的参数。参数分别在单独的行上，并嵌入注释
返回	函数返回值的说明

记住下列几点：

- 每一个重要变量的声明应该包括一个嵌入注释，来描述该变量的使用。
- 变量、控件及过程的命名应该足够清楚，使得只有复杂的执行细节才需要嵌入注释。
- .bas 模块包含工程的 Visual Basic 一般常量声明，在其起始处，应该包括描述应用程序的综述，列举主要数据对象、过程、算法、对话、数据库及系统需求。有时，一段描述算法的伪码可能会有所帮助。

格式化代码

因为许多程序员仍然使用 VGA 显示器，所以在允许代码格式来反映逻辑

结构和嵌套的同时，应尽可能地省屏幕空间。下面列出几点：

- 标准的、基于制表位的嵌套块应该被缩进四个空格（缺省情况下）。
- 过程的功能综述注释应该缩进一个空格。跟在综述注释后面的最高级的语句应该缩进一个制表位，而每一个嵌套的块再缩进一个制表位。例如：

```
'*****'
```

```
'目的：          在用户列表数组中找出  
'          一个指定用户的第一次出现位置。
```

```
'输入：
```

```
'  strUserList():      被搜索的用户列表。
```

```
'  strTargetUser:      要搜索的用户名。
```

```
' 返回：    在 rasUserList 数组中 rsTargetUser  
'          的第一次出现的索引。
```

```
'          如果目标用户没找到，返回 -1。
```

```
'*****'
```

```
Function intFindUser (strUserList() As String, strTargetUser As _
```

```
    String)As Integer
```

```
    Dim i As Integer          ' 循环计数器。
```

```
    Dim blnFound As Integer   ' 目标寻找标志。
```

```
    intFindUser = -1
```

```
    i = 0
```

```
    While i <= Ubound(strUserList) and Not blnFound
```

```
        If strUserList(i) = strTargetUser Then
```

```
        blnFound = True
        intFindUser = i
    End If
Wend
End Function
```

给常量分组

变量和定义的常量应该按功能分组，而不是分散到单独区域或特定文件中。Visual Basic 一般常量应该在单一模块中分组，以将它们与应用程序特定的声明分开。

“&”和“+”运算符

在连接字符串时总是使用“&”运算符，而当处理数值时常常用“+”运算符。当在两个变体上操作时，用“+”运算符来连接可能会导致问题。例如：

```
vntVar1 = "10.01"
vntVar2 = 11
vntResult = vntVar1 + vntVar2      'vntResult = 21.01
vntResult = vntVar1 & vntVar2      'vntResult = 10.0111
```

为 **MsgBox**、**InputBox** 及 **SQL** 查询创建字符串

当产生长字符串时，使用下划线连接字符产生多行代码，这样便于阅读或调试字符串。当显示一个消息框 (MsgBox) 或输入框 (InputBox)，或产生一个 SQL 字符串时，这一技术特别有用。例如：

```
Dim Msg As String
Msg = "This is a paragraph that will be " _
& "in a message box. The text is" _
& " broken into several lines of code" _
& " in the source code, making it easier" _
& " for the programmer to read and debug."
MsgBox Msg
```

```
Dim QRY As String
QRY = "SELECT *" _
& " FROM Titles" _
& " WHERE [Year Published] > 1988"
TitlesQry.SQL = QRY
```

附录 C 本地代码编译器开关

Microsoft Visual Basic 版本允许使用与 Microsoft Visual C++ 相同的优化后端编译器技术编译应用程序，产生快速、高效的本地代码。本地代码编译提供了若干优化和调试选项，而在伪代码编译中是没有这些选项的。习惯上将这些选项称为“开关”，因为每一个选项都可以被打开或关闭。本附录提供本地代码编译器选项的文档，这些选项处于“工程”菜单下“工程属性”对话框中的“编译”选项卡中。关于本地代码的详细信息，请参阅第八章，“再论编程”中的“将工程编译成本地代码”。

目录

- 优化快速代码
- 优化小代码
- 不优化
- 支持 Pentium Pro
- 创建符号调试信息
- 假设没有别名
- 取消数组边界检查

- 取消整数溢出检查
- 取消浮点数错误检查
- 取消对 Pentium FDIV 的安全检查
- 允许无四舍五入的浮点数操作

优化快速代码

通知编译器偏重速度甚于偏重大小，使编译后的可执行文件的速度达到最快。

当编译器将 Visual Basic 语句翻译成机器代码时，常常有许多不同的机器代码序列，它们都能正确地代表已知语句或结构。有时依据这些差异来在大小和速度之间进行权衡。选定本选项后就可确保当编译器辨认出是优化快速代码时，它就总会生成尽可能快的代码，即使这样做会增加编译后程序的大小。

优化小代码

通知编译器偏重大小甚于偏重速度，使编译后的可执行文件的大小达到最小。

当编译器将 Visual Basic 语句翻译成机器代码时，常常有许多不同的机器

代码序列，它们都能正确地代表已知语句或结构。有时依据这些差异来权衡大小和速度。选定本选项后就可确保，当编译器辨认出是优化小代码时，它就总会生成尽可能小的代码序列，即使这样做会降低编译后程序的执行速度。

不优化

关闭全部优化选项。

选定这个选项后，对编译器生成的代码来说，其速度要比选择了优化快速代码或优化小代码的情况明显减慢而大小都明显增加。

支持 Pentium Pro

为优化 Pentium Pro (P6) 处理器而优化代码生成。在本选项下生成的代码仍可在版本较早的处理器上运行，但效率较低。

有了 Pentium Pro 微处理器结构，就可采用某些显著提高效率的代码生成策略。但是在基于 80386、80486 或 Pentium 的计算机上，使用这种策略生成的代码的表现略有逊色。所以，只有当运行程序的机器全部或大部分都采用 Pentium Pro 时才应选择此选项。

创建符号调试信息

在编译后的可执行文件中生成符号调试信息。

使用本选项后，编译成本地代码的程序就可用 Visual C++(5.0 或以后版本)或其它兼容的调试器来调试。设置本选项后就会生成一个 .pdb 文件，其中含有可供兼容的符号调试器使用的符号信息。

假设没有别名

通知编译器，在程序中不使用别名引用。

别名是一个名字，它引用的内存位置已被其它名字引用。在使用 ByRef 参数以两种方式引用同一个变量时就会出现别名。例如：

```
Sub Foo(x as integer, y as integer)
    x = 5          '代码通过两个不同的名字 x 和 y
                  '引用同一个变量
    y = 6          '(Main 中的局部变量 z)。
End Sub
Sub Main
    Dim z as integer
    Foo z,z
```


End Sub

使用本选项后编译器就可以实现在其它情形下不能实现的优化，例如在寄存器中储存变量并执行循环的优化。但是，如果程序传递参数 **ByRef**，则应注意不能选定本选项，因为这种优化可能引起程序不能正确执行。

取消数组边界检查

关闭对数组有效索引和数组正确维数进行的错误检查。

在缺省情况下，**Visual Basic** 每次访问数组都进行检查，以确定索引是否在数组的范围之内。如果索引超出数组的边界，则返回一个错误。选定本选项后将会关闭错误检查并明显地加速对数组的处理。但是，如果程序通过一个超出边界的索引访问数组，则可能会在没有警告的情况下访问无效的内存位置。这将引起不可预料的操作或导致程序崩溃。

取消整数溢出检查

关闭错误检查，这种检查是用来确保赋给整数变量的数值在该数据类型的正确取值范围之内。

在缺省情况下，**Visual Basic** 每次对整数风格数据类型 (**Byte**, **Integer**, **Long**

和 `Currency`) 的变量求值时都进行检查，以确保结果数值在该数据类型的取值范围内。如果数值大小不合适，就会产生错误。选定本选项之后，就会关闭这一错误检查以加速整数运算。但是，如果超出了该数据类型的对应范围，将不返回错误，并有可能产生错误的结果。

取消浮点数错误检查

对赋给浮点数变量的数值是否处于该数据类型的正确取值范围，以及是否出现被零除的操作或其他无效操作不作错误检查。

在缺省情况下，`Visual Basic` 每次对浮点数据类型 (`Single` 和 `Double`) 的变量求值时都进行检查，以确保结果数值在该数据类型所对应的取值范围。如果数值大小不当就会产生错误。此外，错误检查也用来判定是否出现被零除的操作或其他无效操作。选定本选项后就会关闭这一错误检查以加速浮点数的运算。但是，如果超出了该数据类型的对应范围，将不返回错误，并有可能产生错误的结果。

取消对 `Pentium FDIV` 的安全检查

停止生成特殊代码，这些代码在带有浮点数除法 (`FDIV`) 错误的 `Pentium` 处理器上确保浮点数除法的安全。

本地代码编译器自动为浮点数运算添加额外代码，以在带有 **FDIV** 错误的 **Pentium** 处理器上保证运算的安全性。选定本选项后就可生成更小、更快的代码，但代码在带有 **FDIV** 错误的 **Pentium** 处理器上可能会偶尔出现小错。

允许无四舍五入的浮点操作

允许编译器比较浮点表达式的结果而不必将结果四舍五入并精确到一定程度。

对于浮点数计算，在进行比较前一般先要将数据四舍五入到准确的精度（单精度或双精度）。本选项允许编译器在进行四舍五入前进行浮点数比较，这样做可以提高效率。这将加快某些浮点操作的速度；但又使计算一直采用比实际需要更高的精度，可能使预期相等的比较结果不相等。

一般来说，如果直接用浮点计算的结果进行相等比较，则不使用本选项。

例如：

```
Dim Q As Single
```

```
Q = <floating-point computation>
```

```
...
```

```
If Q = <floating-point computation> then
```

```
...
```

End If

如果设置了选项，则使用浮点表达式的结果与 Q 进行比较，该结果的精度可能高于单精度，因此比较可能失败。如果没有设置选项，则在进行比较前将浮点表达式的结果四舍五入，并精确到适当程度（单精度），这时比较就会成功。

附录 D 为应用程序添加帮助

不论一个应用程序编制得如何完美，在某些情况下大多数用户还是会对如何使用它提出问题。除非打算亲临现场去回答他们的问题，处理这种情况最好的办法就是为该应用程序提供帮助文件。

Visual Basic 提供了对两种不同帮助系统的支持：传统的 Windows 帮助系统 (WinHelp) 和新的 HTML 帮助。本附录包括了为应用程序添加 WinHelp 和 HTML 帮助所必需的步骤，指出了可用的两种帮助之间的一些差别。所没有包括的是如何编写帮助文件，不过有许多编写帮助的工具，它们可以帮助您完成这项工作。

主题

- 添加对帮助的支持
- 添加对“这是什么”帮助的支持
- 与应用程序一同发布帮助

添加对帮助的支持

为 Visual Basic 应用程序添加帮助实际上是非常简单的。您所要做的仅仅是设置一个属性，`HelpFile`（并且，当然还要编写并编译帮助文件），使用户能够在按下 `F1` 键或从菜单中请求帮助时显示帮助。另一个属性是 `HelpContextID`，可以通过设置它为应用程序中的任何用户界面元素提供与上下文有关的帮助主题。挂接帮助的过程对 `WinHelp` 和 `HTML` 帮助来说本质上是相同的。

HelpFile 属性

`App` 对象的 `HelpFile` 属性用于指定应用程序使用的帮助文件的文件名。它必须是有效的 `WinHelp` (`.hlp`) 或 `HTML` 帮助 (`.chm`) 文件。如果该文件不存在，则会产生错误。

要设置 `HelpFile` 属性，请按照以下步骤执行：

1. 选择“工程”菜单中的“工程属性”，以打开“过程属性”对话框。
2. 在“通用”选项卡的“帮助文件名”字段中，输入应用程序帮助文件 (`.hlp` 或 `.chm`) 的路径和文件名。

也可以用编程方式设置 `HelpFile`。下面的代码将指定与应用程序的可执行文件在同一目录下的 `HTML` 帮助文件：

```
Private Sub Form_Load()  
    App.HelpFile = App.Path & "\foo.chm"  
End Sub
```

ErrObject 对象也有一个 HelpFile 属性，使您能够为错误消息指定不同的帮助文件。例如，如果有多个应用程序共享相同的错误消息，则可以将错误消息的帮助存入一个帮助文件中，这样就能够每个应用程序中通过 Err.Helpfile 调用了。

HelpContextID 属性

HelpContextID 属性用于将用户界面元素（例如控件、窗体或菜单）与帮助文件中的某个主题链接起来。HelpContextID 属性必须是与 WinHelp (.hlp) 或 HTML (.chm) 文件中的某个主题的 Context ID 匹配的长整数。

例如，可以在 TextBox 的 HelpContextID 属性中输入 10000。当用户选定 TextBox 并按下 F1 键时，Visual Basic 就会在由应用程序的 HelpFile 属性指定的帮助文件中查找 Context ID 为 10000 的主题。如果找到了相应的帮助主题，将打开“帮助”窗口并显示该主题；否则，将产生错误并显示帮助文件的缺省主题。

应当使用唯一的 HelpContextID 来匹配帮助文件中的每一个帮助主题。在某些情况下，如果某些对象共享一个通用的帮助主题，则可能需要为多个对象赋相同的 HelpContextID。

没有必要为窗体中的每个控件输入 HelpContextID。如果用户在 HelpContextID 为 0（缺省值）的控件上按下 F1 键，Visual Basic 将会为该控件的容器查找有效的 HelpContextID。

要为控件或窗体赋一个 HelpContextID，请按照以下步骤执行：

1. 选定要输入 HelpContextID 的控件或窗体。

2. 在“属性”窗口中双击 `HelpContextID`，并输入有效的长整数。
记录输入的值，以便为相关帮助主题的上下文标识符使用相同的值。

注意 对于 `CommonDialog` 控件，可能还有其它一些控件，该属性的名称是 `HelpContext` 而不是 `HelpContextID`。

要为菜单赋一个 `HelpContextID`，请按照以下步骤执行：

1. 在“工具”菜单中选择“菜单编辑器”。
2. 选择需要输入 `HelpContextID` 的菜单项。
3. 在“选择 `HelpContextID`”框中输入有效的长整数。

记录输入的值，以便能够为相关帮助主题的上下文标识符使用相同的值。

`HelpContextID` 还能够用如下方法以编程方式输入：

```
Private Sub Form_Load()  
    Command1.HelpContextID = 12345  
    MenuHelp.HelpContextID = 23456  
    Err.HelpContext = 34567  
End Sub
```

提示 如果有多个帮助主题，可能在输入 `HelpContextID` 之前确定一个编号计划会有所帮助。为应用程序中的每个窗体或主要的元素赋予不同的编号范围，例如，第一个窗体用 1000-1999,第二个用 2000 – 2999，依此类推。

添加对“这是什么”帮助的支持

Visual Basic 能够让您方便地为应用程序添加“这是什么”帮助。“这是什么”帮助提供了对弹出窗口中文本的快速访问，而不需要打开帮助查看程序。它一般用于为数据输入字段等用户界面元素提供简单的帮助。Visual Basic 在 WinHelp (.hlp) 和 HTML 帮助 (.chm) 文件中都支持“这是什么”帮助主题。

把窗体的 WhatsThisHelp 属性设置为 True 将启用“这是什么”帮助。启用“这是什么”帮助后，该窗体的上下文相关帮助将被禁用。

要启用窗体的“这是什么”帮助，请按照以下步骤执行：

- 1. 选定该窗体，在“属性”窗口中双击 WhatsThisHelp 属性，并将其设置为 True。
- 2. 设置如下属性，以在该窗体的标题栏添加“这是什么”按钮：

属性	设置值
BorderStyle	1 – 固定单线或 2 – 可变尺寸的
MaxButton	False
MinButton	False
WhatsThisButton	True

注意 如果窗体具有最小化和最大化按钮，则它就不能有“这是什么”按钮。还可以用另一种方法替代上述设置，即将 BorderStyle 属性

设置为 3 - 固定对话框，因为固定对话框没有最小化和最大化按钮。选定要提供“这是什么”帮助的每个控件，并为该控件的 `WhatsThisHelpID` 属性赋一个唯一的值。

记录输入的值，以便能够为相关帮助主题的上下文标识符使用相同的值。

重点 要在 HTML 文件中实现“这是什么”帮助，所有的“这是什么”帮助主题必须包含在被编译成 `.chm` 文件的 `Cshelp.txt` 文件中。详细信息请参阅编写 HTML 帮助工具的文档。

不使用“这是什么”按钮也能够启用“这是什么”帮助，只要将该窗体的 `WhatsThisHelp` 属性设置为 `True`，并调用窗体的 `WhatsThisMode` 方法或控件的 `ShowWhatsThis` 方法即可。

与应用程序一同发布帮助

为应用程序添加帮助的最后一步是保证它能送到最终用户的手中。与应用程序一同发布 WinHelp 帮助和 HTML 帮助所需要的步骤略有不同。

发布 WinHelp

由于每个 Windows 系统都已经安装了 Windows 帮助查看程序，所要发布的就只是帮助文件 (`.hlp`) 本身了。打包和展开向导 (Package and Deployment Wizard) 会自动为应用程序引用的帮助文件添加从属信息。如果是通过其它方法创建安装程序，就需要确保包含了该 `.hlp` 文件，并将其安装到正确的位置（通常在与应用程序相同的目录或在 `\Windows\Help` 目录中）。

发布 HTML 帮助

HTML 帮助是相对较新的技术，因此就不能假定每位最终用户都有查看 HTML 帮助所必需的文件。打包和展开向导会为被应用程序引用的 HTML 帮助文件 (.chm) 添加从属信息；但它可能不为 HTML 帮助查看程序文件添加所有的从属信息。您可能需要修改安装程序以包含这些文件。有关在给定情形下需要哪些文件的详细信息，请参考所使用的 HTML 帮助编写工具的文档。

