

前 言

面向对象的程序设计以其快捷、易懂、功能强大、界面友好等一系列优点而深得程序员们和程序设计爱好者的衷心喜爱。如果稍加留意，你会注意到：面向对象程序设计的概念已经不再是天方夜谭，它正以无法形容的速度发展，它已经触及到人们生活的每一个角落，商店、银行、学校、政府机关、商业公司、家庭、个人，几乎无所不在。因此，计算机的实用程序设计者和程序设计的爱好者如果不能掌握一门面向对象的程序设计工具，那么他是落伍的，更是不能适应这个飞速发展的时代。在众多的面向对象程序设计工具中，Microsoft 公司的 Visual Basic 6.0 以其友好的界面、全面强大的功能、简单易懂的使用方法为广大使用者所称道。

针对当前程序设计的形式和计算机书籍市场良莠不齐的现象，我们出版了这套《实用程序设计开发丛书》，希望以我们的实战经验和心得体会，用浅显易懂的语言、循序渐进的方法向大家展示程序设计的基础知识和编程技巧。

为了帮助大家能够很快地了解这种开发工具的使用方法，更多地了解程序开发的知识，本书由易到难阐述了 Visual Basic 6.0 的使用方法和开发技巧。本书内容涉及到键盘和鼠标输入、菜单、各种对话框、各种控件、多文档，以及程序优化方法等，并着重介绍了 Visual Basic 6.0 在实际应用中的各种方法和技巧。

本书文字简洁、流畅，内容由浅入深，既可供初次接触面向对象程序设计的读者作为了解 Visual Basic 6.0 程序设计知识的入门书籍，也可供那些想自己实战体会，希望进一步提高水平的读者参考。

由于时间和水平有限，疏漏之处在所难免，恳请广大读者不吝赐教。

编 者

2000 年 10 月

EJS/2/01

目 录

第一章 Visual Basic 简介	1
1.1 Visual Basic 6.0 概述	1
1.2 集成开发环境	2
1.3 Internet 开发	5
1.4 数据库开发	6
1.5 语言中的新特性	8
1.6 Visual Basic 的向导	9
第二章 使用 Visual Basic 建立第一个程序	11
2.1 启动 Visual Basic	11
2.2 建立第一个 Visual Basic 程序	12
2.2.1 第一个 Visual Basic 程序	12
2.2.2 执行和终止	16
2.2.3 生成 EXE 文件	16
2.2.4 保存文件	16
2.3 退出 Visual Basic	17
第三章 Visual Basic 语言	18
3.1 代码编写机制	18
3.1.1 模块	18
3.1.2 编码基础	21
3.2 变量与常量	23
3.2.1 声明变量	23
3.2.2 变量类型	24
3.2.3 用户定义的数据类型	24
3.2.4 变量的范围	26
3.2.5 常量	27
3.3 数组	29
3.4 过程	31
3.4.1 子程序	31
3.4.2 函数	32
3.4.3 变元	33
3.5 控制结构	36
3.5.1 流程控制语句	36
3.5.2 循环控制语句	39

3.6 类与对象	42
3.6.1 面向对象程序设计语言	42
3.6.2 对象	43
第四章 Visual Basic 的标准控件	46
4.1 Visual Basic 控件简介	46
4.1.1 控件分类	46
4.1.2 使用控件值	48
4.2 ADO Data 控件	50
4.3 CheckBox 控件	54
4.4 ComboBox 控件	55
4.5 CommandButton 控件	58
4.6 CommonDialog 控件	60
4.6.1 显示“打开”和“保存”对话框	60
4.6.2 使用“颜色”对话框	61
4.6.3 使用“字体”对话框	62
4.6.4 使用“打印”对话框	64
4.6.5 使用 ShowHelp 方法显示帮助文件	65
4.7 Data 控件	66
4.8 DataCombo 控件和 DataList 控件	67
4.9 DataGrid 控件	69
4.10 文件系统控件	80
4.11 Frame 控件	85
4.12 HScrollBar 控件和 VScrollBar 控件	86
4.13 Image 控件	89
4.14 Label 控件	90
4.15 Line 控件	91
4.16 ListBox 控件	92
4.17 OLE 容器控件	95
4.18 OptionButton 控件	96
4.19 PictureBox 控件	97
4.20 Shape 控件	98
4.21 TextBox 控件	99
4.22 Timer 控件	101
4.23 使用控件数组	102
第五章 键盘和鼠标的事件	105
5.1 事件和事件过程	105
5.1.1 事件驱动	105

5.1.2 Visual Basic 对象识别事件	105
5.1.3 工作过程	106
5.2 响应鼠标事件	107
5.2.1 MouseDown 事件	108
5.2.2 MouseMove 事件	110
5.2.3 MouseUp 事件	113
5.3 检测鼠标按钮	114
5.4 检测 Shift、Ctrl 和 Alt 键状态	115
5.5 拖放	116
5.5.1 自动拖放模式	116
5.5.2 拖动时的图标	116
5.5.3 放下对象时的响应	117
5.5.4 对拖动的控制	117
5.5.5 改变控件的位置	119
5.6 OLE 拖放	120
5.6.1 自动 OLE 拖放	121
5.6.2 拖放操作	125
5.6.3 拖动文件	134
5.7 自定义鼠标指针	136
5.8 响应键盘事件	137
5.8.1 低级键盘处理程序	138
5.8.2 KeyPress 事件	138
5.8.3 KeyDown 事件和 KeyUp 事件	139
第六章 工程管理	144
6.1 工程文件	144
6.2 创建、打开和保存工程	146
6.3 添加、删除和保存文件	147
6.4 添加模块和控件	148
6.5 制作、运行可执行文件	150
6.6 工程选项设置	151
6.7 使用向导和外接程序	152
第七章 界面设计	155
7.1 窗体	155
7.1.1 Visual Basic 的窗体	155
7.1.2 在窗体上添加控件	155
7.1.3 启动设置	160
7.1.4 窗体的使用	162

7.1.5 窗体设计的基本原则	163
7.2 对话框	166
7.2.1 消息框	166
7.2.2 输入框	166
7.3 菜单	167
7.3.1 创建菜单	167
7.3.2 弹出式菜单	174
7.4 工具栏	175
7.4.1 创建工具栏	175
7.4.2 编写代码	179
第八章 图形开发	180
8.1 坐标系统	180
8.2 图形方法	181
8.3 使用颜色	183
8.4 图形控件	186
8.5 图片应用	187
8.6 创建从蓝逐渐变黑的背景	189
第九章 文本处理	191
9.1 字体	191
9.2 输出格式	196
9.3 剪贴板应用	198
9.4 文件的读写	201
第十章 多媒体应用	204
10.1 多媒体基础	204
10.2 多媒体控件与函数	204
10.3 多媒体应用	207
10.3.1 播放声音文件	207
10.3.2 播放视频文件	208
第十一章 API 函数	210
11.1 在 Visual Basic 中使用 API	210
11.1.1 API 简介	210
11.1.2 API 文本浏览器	215
11.2 应用实例	215
11.2.1 确定 CPU 类型	215
11.2.2 确定操作系统的版本	216
11.2.3 确定驱动器类型	217
11.2.4 确定消逝时间	219

11.3 屏幕保护程序	220
第十二章 用部件编程	225
12.1 部件	225
12.2 创建对部件对象的引用	226
12.2.1 创建引用	226
12.2.2 声明对象变量	227
12.2.3 变量赋值及使用	228
12.3 对象的方法、属性和事件	230
12.4 释放部件	232
第十三章 数据库开发	234
13.1 了解数据库	234
13.2 用 Data 控件可以做什么	235
13.3 怎样使用 Data 控件	236
13.4 记录集	238
13.5 使用数据绑定控件	238
13.6 添加新记录	240
13.7 使用事务处理维护数据库的完整性	243
13.7.1 开始一个事务	244
13.7.2 保存修改的结果或者撤销修改	244
13.7.3 使用多人事务	245
13.8 使用可视化数据管理器	245
第十四章 Internet 开发	247
14.1 Internet 控件	247
14.2 创建自己的浏览器	248
14.3 超文本标记语言	252
14.4 HTML 和 VBScript	253
14.5 HTML 和 Active 文档的调用	255
第十五章 帮助文件	257
15.1 创建帮助文件	257
15.1.1 帮助主题基础	258
15.1.2 建立项目文件	258
15.1.3 创建帮助文件	259
15.2 在项目中添加 Help 文件	262
15.3 在项目中添加 F1 帮助	263
15.4 如何利用 CommandDialog 控件为项目添加帮助文件	264
15.5 如何为窗体添加 WhatsThisHelp 功能	265
第十六章 优化	267

16.1 优化	267
16.2 优化速度	269
16.2.1 优化代码.....	269
16.2.2 优化显示速度.....	273
16.2.3 优化感觉速度.....	275
16.3 优化大小	277
16.3.1 减小代码大小.....	277
16.3.2 修剪图形.....	280
16.3.3 分段应用程序.....	282

第一章 Visual Basic 简介

众所周知, Basic 是世界上影响最广泛的计算机语言之一。随着计算机技术的迅猛发展, 计算机的性能价格比有了显著的提高, 这在很大程度上为 Windows 的发展提供了有利的物质基础。在软件的应用方面, 独领风骚的恐怕要数美国的微软公司了, 从 1995 年推出 Windows 95, 1998 年推出 Windows 98 后, 2000 年又相继推出 Windows 2000 和 Windows Me, 真可谓大出风头。

Windows 简洁、友善的图形界面, 确实给人留下了深刻的印象。那么如何在 Windows 环境下开发应用程序呢?是不是只有那些资深的计算机专业开发人员才能胜任呢?

要回答这个问题, 我们就必须要谈到在 Windows 环境下使用的开发语言, 这就是我们这本书所要讲的 Microsoft Visual Basic 6.0, 它与以前的 Basic 基本兼容, 但功能更强, 使用更容易、更方便。正如微软公司总裁比尔·盖茨所言, 别的语言能做到的, Visual Basic 也能做到。可以说 Visual Basic 的出现让世人重新认识了 Basic。

纵观计算机的发展, 软件的需求刺激了硬件的发展, 反过来硬件的发展又为软件的发展提供了强有力的物质基础。Visual Basic 顺应了时代的发展, 依靠现代计算机技术的支持, 从 1991 年的 Visual Basic 1.0 发展到今天的 Visual Basic 6.0, 它不但是专业人员的得力工具, 而且也成为许多非专业人员的好伴侣。据不完全统计, 目前, 全世界有数以百万计的程序员使用 Visual Basic 开发各种类型的软件。

好了, 现在就让我们一起打开 Visual Basic 的大门吧, 结识这位“老朋友”。



1.1 Visual Basic 6.0 概述

让我们看看微软公司的 Visual Basic 版本情况:

1991 年推出 Visual Basic 1.0 版;

1992 年推出 Visual Basic 2.0 版;

1993 年推出 Visual Basic 3.0 版;

1995 年推出 Visual Basic 4.0 版;

1997 年推出 Visual Basic 5.0 版;

1998 年推出 Visual Basic 6.0 版。

在这里我们简要概述 Visual Basic 6.0 (中文版) 的基本情况。

Visual Basic 6.0 是在 Visual Basic 5.0 的基础上发展起来的, 它保留了 Visual Basic 5.0 的优点, 同时增加了许多新的功能, 使之更完善、功能更强大。

Visual Basic 6.0 包括三个版本, 即标准版、专业版和企业版。由于这些版本都是在相同的基础上建立起来的, 因此, 大多数 Visual Basic 编写的应用程序可在三种版本中通用。标准版实际上是基础版, 可开发一般应用程序; 专业版包括标准版的全部功能, 它为专业编程人员提供了一套用于软件开发的功能完备的工具; 企业版包括专业版的全部功能, 可供专业编程人员开发功能更强大的应用程序。

Visual Basic 6.0 的新增特性, 基本上也是按照这三个版本进行划分的。总的来说, Visual Basic 6.0 重点加强了专业版和企业版的功能, 标准版中的功能也比 Visual Basic 5.0 有了较大增强。

Visual Basic 6.0 从表面上看与 Visual Basic 5.0 的区别不大, 但我们可以看出其主要发展方向是放在数据访问和因特网功能的完善和增强上。

在后面, 将详细介绍 Visual Basic 6.0 各个方面的新特性和增强的功能, 主要包括以下几方面:

- ①集成开发环境方面的内容;
- ②因特网特性 (Internet Feature) 方面的内容;
- ③数据访问 (Data Access) 方面的特性;
- ④语言 (Language) 方面的特性;
- ⑤向导 (Wizard) 方面的特性;
- ⑥Visual Basic 6.0 的帮助系统;

1.2 集成开发环境

Visual Basic 的集成开发环境, 简称 IDE (Integrated Development Environment)。IDE 给出了创建应用程序所需的一切条件, 可用它来编写代码、测试及细调程序, 最终生成可执行文件。这些文件独立于开发环境, 可以移植到没有 Visual Basic 的机器上执行。

· 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，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 1.1 所示。

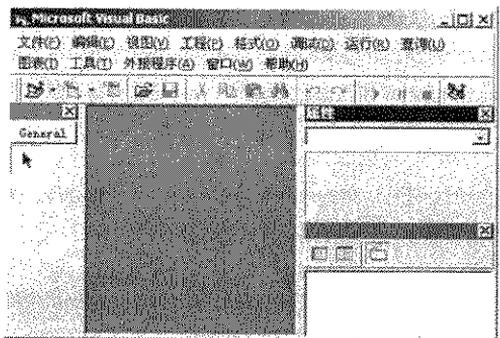


图 1.1 Visual Basic 集成开发环境

· 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:

(1) 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”。

(2) 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左键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。

(3) 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。

(4) 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。

(5) 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。

(6)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。

(7)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 中的对象和其他应用程序，查看这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。

(8)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。

(9)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。

(10)窗体布局窗口

Form Layout Window (如图 1.2 所示) 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 1.2 窗体布局窗口

(11)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 中运行应用程序时才有效。

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。

(12)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。

(13)SDI 或 MDI 界面

Visual Basic IDE 有两种不同的类型：单文档界面 (SDI) 和多文档界面 (MDI)。对 SDI 选项，所有 IDE 窗口可以在屏幕上任何地方自由移动；只要 Visual Basic 是当前应用程序，它们将位于其他应用程序之上。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。

④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，就可以通过在“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；

1.3 Internet 开发

使用 Visual Basic 6.0 因特网特性中的新内容，可以轻松高效地开发高质量的应用软件。例如运用 DHTML（动态 HTML）应用程序，可以编写 Visual Basic 代码，对 HTML 页上的操作作出响应，而不用将处理操作转至服务器。使用打包和展开向导，可以很容易地将应用程序包转至 Web 站点。

此外，在异步操作方面的功能也得到了增强，提供了更多的关于操作的进度和状态。

- IIS 应用程序(专业版和企业版)

用于编写服务器的因特网应用程序，它使用 Visual Basic 代码对来自浏览器的用户请求作出应答。

- DHTML 应用程序(专业版和企业版)

Internet Explorer 中的动态 HTML (DHTML) 技术可以显示 Web 中的每个元素的属性、方法和事件。运用 DHTML 应用程序，可以编写 Visual Basic 代码，对 HTML 页的操作优先作出响应，而不用将处理操作转至服务器。

- Web 出版向导 (标准版、专业版和企业版)

使用打包和展开向导 (Package and Deployment Wizard)，可以很容易地将应用程序包转至 Web 站点 (如图 1.3 所示)。

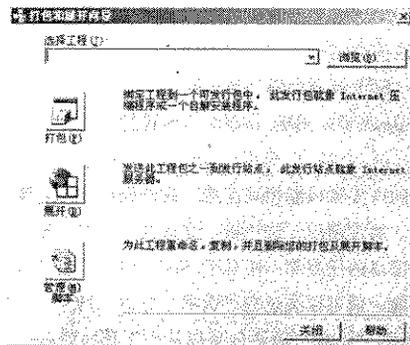


图 1.3 打包和展开向导

- AsyncRead 的增强 (标准版、专业版和企业版)

异步操作得到了增强, 提供了更多的关于操作的进度和状态, 如目前已经读了多少字节, 还有多少字节需要读取等。新语言成员包括: AsyncReadProgress 事件、BytesMax 属性、BytesRead 属性、Status 属性 (AsyncProperty 对象)、StatusCode 属性和 Target 属性。此外, AsyncRead 方法和 AsyncReadComplete 事件也得到了增强。

- 在 Internet Explorer 3.0 和 4.x 中跨页/框共享状态 (标准版、专业版和企业版)

ActiveX 部件的属性值可以驻留在 Internet Explorer 的全局属性包 (Property Bag) 中, 当浏览完包含用户控件或用户文档的 HTML 页时, 用户可以保存数据。

- Internet Explorer 4.x (专业版和企业版)

现在, 用 Internet Explorer 4.x 下载 ActiveX 文档与下载 ActiveX 控件的操作情况一样, 例如, 它可以支持相关的 URL。

1.4 数据库开发

在 Visual Basic 6.0 中, 可以明显看到其数据访问的功能得到了增强, 特别是与数据库的连接更加容易, 可以将任何数据源与任何数据用户连接起来, 操作也更加友好。其中的 ADO (ActiveX 数据对象) 和 ADO 控件, 在数据访问中扮演了重要的角色。这种新型的数据源控件允许用最少的代码来创建数据库应用程序。

此外, 在 Visual Basic 6.0 中增强了数据绑定功能。例如, 可以在运行中设置控件的数据源, 从而达到动态数据库的连接, 也可以创建属于数据源和数据用户的类或者创建属于数据源的用户控件。

- ADO (ActiveX 数据对象) (标准版、专业版和企业版)

这种新的访问技术配有更加简单的对象模型，与 Microsoft 及非 Microsoft 的技术实现了更好的集成，它是用于本地和远程数据访问的常用界面，是用户可以访问的数据连接界面，并且是分层记录集。

- 数据环境 (Data Environment) (专业版和企业版)

它允许直观地创建和操作 ADO 连接和命令。该数据环境提供了一个动态对象模型，模型封装了数据环境中定义的所有 ADO 记录组和连接。可以编写代码，对 Recordset 和 Connect 事件做出应答，将 ADO 命令作为方法来处理，并自动将 ADO Recordset 作为数据环境的属性来引用。可以将数据变成分层次的、分组的或汇总的记录集。

- ADO 数据控件 (标准版、专业版和企业版) (如图 1.4 所示)



图 1.4 ADO 数据控件

一种新数据源控件，其功能与内部数据和远程数据控件大致相同，它允许用最少的代码来创建数据库应用程序。

- 增强型数据绑定 (标准版、专业版和企业版)

在以前的 Visual Basic 版本中只能在窗体上将各个控件捆绑在一起。在 Visual Basic 6.0 中，可以将任何数据源与任何数据用户连接起来。现在可以在运行时设置控制的数据源属性 (Data Source)，以便动态地连接数据库。

还可以创建属于数据源和数据用户的类。通过新的 BindingsCollection (绑定集合) 对象将它捆绑在一起。可以创建属于数据源的用户控件，它类似于 ADO 数据控件。

此外，还可以创建属于复杂连接的用户控件 (也就是它们在整个记录集上运行)，它类似数据表格控件。

- 支持 OLE DB (标准版、专业版和企业版)

OLE DB 是一组 COM 界面，它配有许多应用程序，可以对存储在各种信息源中的数据 (关系型和非关系型数据) 进行统一访问。这些界面支持适合数据源的 DBMS 功能的数目，使它可以共享其数据。

ADO 是程序员访问 OLE DB 的一种方法和使用的界面，所有新数据的绑定控件、数据环境和数据报表设计都是 OLE DB 所能识别的。

- 可视化数据库工具集成 (适合企业版)

包括查询设计器 (Query Designer) 和数据库设计器 (Database Designer)。

可视化地创建和修改数据库方案和查询条件：创建 SQL 服务器和 Oracle 数据库表格，用拖放的方式创建视图，并自动改变列数据类型。

- 安装向导数据 (Setup Wizard Data) 的增强 (标准版、专业版和企业版)

打包和展开向导 (Package and Deployment Wizard)，原来是安装向导 (Setup Wizard)，可支持 ADO、OLE DB、ROD 和 DAO。

- 数据报表 (Data Report) (专业版和企业版) (如图 1.5 所示)

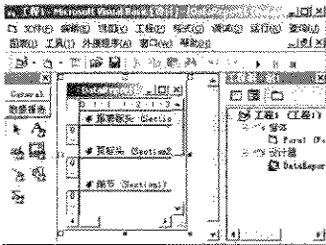


图 1.5 数据报表

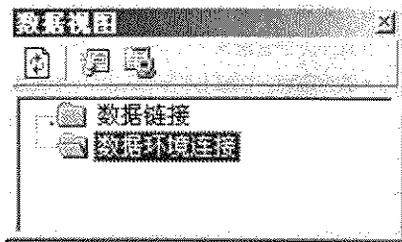


图 1.6 数据视图窗口

它允许使用拖放操作迅速用分层记录集来创建报表。

- 数据源 (Data Source) (专业版和企业版)

用于创建属于数据源的控件和类, 并且其他控件可以与它们连接。

- 数据视图窗口 (专业版和企业版) (如图 1.6 所示)

可以使用数据视图窗口来浏览其连接的所有数据库, 并可看到其中的表格、视图和存储过程等。

- SQL 编辑器 (专业版)

它允许将新的存储过程添加到现有的 SQL 服务器和 Oracle 数据库。可以用编辑器来编写触发程序。

- 分层 FlexGrid 控件 (标准版、专业版和企业版)

它是 FlexGrid 控件的更新版, 用于显示分层记录集 (使用不同表创建的记录集)。使用该控件的多种格式化功能, 可以用直观和便于理解的方式显示复杂的数据层次。

- 格式化对象 Format Objects (标准版、专业版和企业版)

提供了数据库和绑定控件的双向转换: 当从数据库读入数据时, Format 对象为绑定控件增加恰当的格式。

- 此外还有远程 ADO 记录集 (标准版、专业版和企业版)

1.5 语言中的新特性

对于 Visual Basic 6.0 中的各个版本 (标准版、专业版和企业版) 都不同程度较以前的版本有了增强。例如, 使用 Public 方法的用户自定义的类型; 函数可以返回数组; 大小可变的数组现在可以出现在赋值语句的左边; 遍历文件系统和创建文本文件和目录的文件系统对象; 增强了 CreateObject 函数和 StrCopy 函数的功能等。

除此之外, Visual Basic 6.0 还新增了很多字符串处理函数, 如 FormatCurrency 函数,

FormatDateTime 函数、FormatNumber 函数等。

- 使用 Public 方法的自定义的类型（适合标准版、专业版和企业版）

用户定义的类型可以是公共属性和方法的参数或返回类型。

```
[Private|Public] Type varname
    elementname([subscripts]) As type
    [elementname([subscripts]) As type]
```

...

End Type

- 函数可以返回数组（适合标准版、专业版和企业版）

函数和属性程序将能返回数组。

- 可以赋值可改变大小的数组（适合标准版、专业版和企业版）

大小可变的数组现在可以出现在赋值语句的左边。大小固定的数组不能出现在赋值语句的左边，不过它们可以出现在右边。

- 文件系统对象（适合标准版、专业版和企业版）

提供一串例程，以便遍历文件系统和创建文本文件和目录。

- CallByName（适合标准版、专业版和企业版）

通过设定一个包含属性或方法的字符串，而不是对名字进行显示编码，以便访问属性或方法。

- CreateObject 函数的增强（适合标准版、专业版和企业版）

使用 CreateObject 函数，现在可以将机器名设定为一个选项参数，可以用来创建远程机器上的对象。

- Strconv 函数的增强（适合标准版、专业版和企业版）

新的 LCID 参数能够为字符串设定一个不同于系统的 LocalID 的一个 LocalID。

- 新字符串函数（适合标准版、专业版和企业版）

Filter 函数、FormatCurrency 函数、FormatDateTime 函数、FormatNumber 函数、FormatPercent 函数、InstrRev 函数、Join 函数、MonthName 函数、Replace 函数、RoundFunction 函数、Split 函数、StrReverse 函数、WeekdayName 函数。

1.6 Visual Basic 的向导

向导（Wizard）的出现为编程人员提供了很大的方便，简化了操作，提高了效率，因

此其发展前途十分广泛。

Visual Basic 6.0 新增了几个新向导，同时进一步完善了某些向导的功能，并用新的向导代替原来的向导，例如使用“打包和展开向导”（Package and Deployment Wizard）代替原来的“安装向导”（Setup Wizard）。

在 Visual Basic 6.0 的向导中，数据处理的功能明显加强，如“数据对象生成器向导”（Data Object Generator Wizard）和“数据窗体向导”。

- 安装向导（Setup Wizard）的增强特性（标准版、专业版和企业版）

在 Visual Basic 6.0 中（包括标准版），使用 Package and Deployment Wizard（原来为 Setup Wizard），可以将.cab 文件（柜子文件）配置到 Web 服务器、网络共享器和其他文件柜中。这个新向导可支持对 ADO、OLE DB、RDO、ODBC 和 DAO 的数据访问。此外，该向导还可以更好地控制 Setup 程序使用的 Start 菜单组和图标，并可以在 Microsoft 的 Internet Explorer 4.x 中更好地显示用户文档。新向导可以作为一个附加的程序通过 Visual Basic 来运行，它可以作为进程的一部分，以静态方式来运行。

- 数据对象生成器向导（Data Object Generator Wizard）（专业版和企业版）

可使与数据环境及用户控件相绑定的多层对象的创建实现自动化。

- 数据窗体向导（Data Form Wizard Enhancements）（专业版和企业版）

现在可以建立只有代码的窗体，在这些窗体中，控件与数据控件不连接，它可以根据数据库类型来使用 DAO 或 RDO 代码。该向导与应用程序向导、Chart（图表）向导和 FlexGrid 向导相集成。

- 应用程序向导（Application Wizard）的增强（标准版、专业版和企业版）

可以把程序中的设置作为配置文件来保存，供以后使用，可以用相同格式创建多个应用程序。可以从应用程序向导内调用数据窗体向导（Data Form Wizard）和工具栏向导（Toolbar Wizard），以便创建数据窗体和工具栏。菜单现已可以全部定制。

- 附加程序设计器（Add-in Designer）（专业版和企业版）

如果要简单地设置附加程序的加载方法、名字、目标应用程序和版本，可以着手开发工作。

- 类建立实用程序（Class Builder Utility）的增强（专业版和企业版）

支持参数列表中的 ParamArray、Optional、ByVal 和 Default 等值，并支持枚举（enums）。

第二章 使用 Visual Basic 建立第一个程序

2.1 启动 Visual Basic

就像启动任何 Windows 可执行程序一样,可选择某种方式来启动 Visual Basic (见图 2.1)。

1. 可从开始菜单运用 Visual Basic 6.0 集成开发环境。确切的菜单命令路径取决于是否已经安装了 Visual Basic 6.0 作为 Microsoft Visual Studio 套件的一部分。

2. 可在桌面上为 Visual Basic 6.0 创建一个快捷方式,双击它即可运行 Visual Basic 6.0 集成开发环境。

3. 如果安装了 Visual Basic 6.0,就在操作系统中注册了.vbp、.frm、.bas 及其他一些扩展名。因此,可双击任何 Visual Basic 6.0 文件来运行集成开发环境。

4. 如果安装了 Microsoft Active Desktop (活动桌面),就可以在系统任务栏为 Visual Basic 6.0 的集成开发环境创建一个快捷键。这可能是运行 Visual Basic 6.0 最快捷的方式。



图 2.1 启动 Visual Basic

2.2 建立第一个 Visual Basic 程序

2.2.1 第一个 Visual Basic 程序

创建 Visual Basic 应用程序有四个主要步骤：创建应用程序界面、设置属性、编写代码、运行(测试)。为了说明这一实现过程，请按照以下步骤创建一个简单的应用程序，它由一个文本框和一个命令按钮组成。单击命令按钮，文本框中就会出现一条信息。

· 创建应用程序界面

窗体的创建是应用程序的基础，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见部分。例如，应用程序中可能有个作为图形容器的窗体，而这些图形是打算在其他窗体中显示的。

建立 Visual Basic 应用程序的第一步是创建窗体，然后在创建的窗体上绘制构成界面的对象。对于本节的例子，可使用工具箱中的两个控件：按钮控件、文本框。

1. 绘制控件

要用工具绘制控件：

- (1)单击要绘制的控件(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 2.2 所示。
- (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他位置。

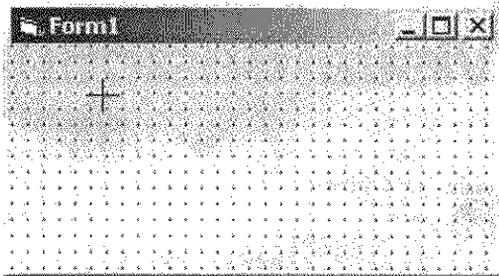


图 2.2 绘制控件

2. 调整大小、移动和锁定控件

注意出现的控件四周的小矩形框称作尺寸柄：下一步可用这些尺寸柄调节控件尺寸：

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。

要调整控件的尺寸：

(1)用鼠标单击要调整尺寸的控件。

(2)选定的控件中出现尺寸句柄。

(3)将鼠标指针定位到尺寸句柄上，拖动该尺寸句柄直到控件达到所希望的大小为止。

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

(5)释放鼠标按钮。

也可以用 Shift 键加箭头键调整选定控件的尺寸。

要移动控件，可以用鼠标把窗体中的控件拖动到新的位置，或用“属性”窗口改变 Top 和 Left 的属性。

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，可以在“格式”菜单中选取“锁定控件”，或在“窗体编辑器”工具栏中单击“锁定控件切换”按钮。这个操作将把窗体中所有的控件锁定在当前位置，防止已处于理想位置的控件因不小心而移动。本操作只锁住当前窗体中的全部控件，不影响其他窗体中的控件。这是一个切换命令，因此也可以用来解锁控件位置。

要调节锁定控件的位置，可按 Ctrl 键，再用合适的箭头键“微调”已获焦点的控件的位置，也可以在“属性”窗口中改变控件 Top 和 Left 的属性。现在已生成“Hello, World!”应用程序的界面，如图 2.3 所示。

· 设置属性

下一步是给创建的对象设置属性。属性窗口(如图 2.4 所示)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、单击工具栏中的“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。

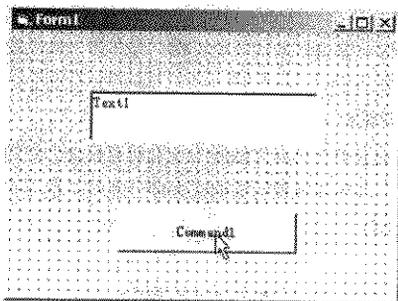


图 2.3 “Hello, World!”应用程序的界面



图 2.4 属性窗口

属性窗口包含如下的元素:

(1)对象框: 显示可设置属性的对象的名字。单击对象框右边的箭头, 可显示当前窗体的对象列表。

(2)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)分类的层次结构视图中进行选取。

(3)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。

1. 使用“属性窗口”

要在“属性窗口”中设置属性:

(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箭头, 可以显示这个清单, 或者双击列表项, 可以循环显示该清单。

2. 设置 Icon 属性

在 Visual Basic 中, 所有的窗体都有一个普通的缺省图标, 它们在窗体最小化时出现。如果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。32×32 像素的图标是 Windows 的 16 位版本的标准, 也可应用在 Windows 98 和 Windows NT 中, 另一种 16×16 像素的图标是用在 Windows 98 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, 可双击要编写代码的窗体或控体, 或从“工程管理器”窗口中选定窗体或模块的名称, 然后选取“查看代码”按钮。

如图 2.5 所示的是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。要在同一代码窗口中显示全部过程, 请按如下步骤执行:

(1)在“工具”菜单中, 选定“选项”对话框。

(2)在“选项”对话框的“编辑器”选项卡中, 选取“缺省为查看所有模块”左边的复选框。在“过程分隔符”左边的复选框中, 可在各过程间添加或去掉分隔线。也可在代码编辑器窗口的左下角单击“全模块查看”按钮。

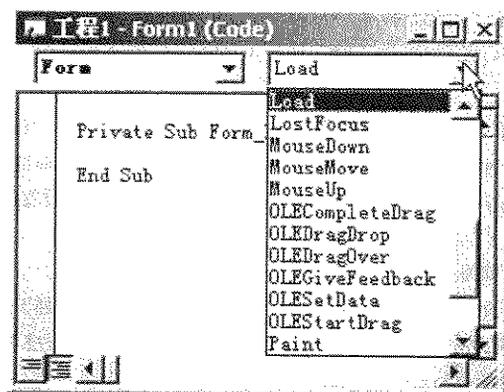


图 2.5 代码编辑器窗口

要使代码窗口每次只显示一个过程，请按如下步骤执行：

- (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，消除“缺省为查看所有模块”左边的复选框。也可在代码编辑器窗口的左下角单击“过程查看”按钮。

1. 代码窗口

代码窗口包含如下元素：

(1) 对象列表框：显示所选对象的名称。单击列表框右边的箭头，显示和该窗体有关的所有对象的清单。

(2) 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。

2. 事件过程的创建

Visual Basic 应用程序的代码被分为称为过程的小代码块。事件过程，正如正要创建的应用程序一样，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，请按如下步骤执行：

(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：

```
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 Text 的属性，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。

2.2.2 执行和终止

在应用程序代码编写完成以后，就可以运行此程序了，可以用如下几种方式开始程序的执行：从 Run 菜单中激活 Start 命令；单击工具栏中相应的图标或按 F5 键。在所有情况下，都将看到窗体设计器消失，并且由实际的窗体取代。

注意：通过激活 Run 菜单中的 End 命令，也可终止在环境中运行任何的 Visual Basic 程序。但通常情况下这不是一个好办法，因为它禁止一些与窗体有关的事件的启动，如 QueryUnload 和 Unload 事件。在有些情况下，这些事件过程包含所谓的清除代码，如关闭数据库或删除临时文件的语句。如果突然终止程序的执行，实际上就阻止了此代码的执行。通常只有在严格需要时才使用 End 命令。

2.2.3 生成 EXE 文件

在程序开发期间，需要创建可执行程序（EXE），因为编译程序通常比解释程序执行快，用户不需要安装 Visual Basic 就可运行程序，通常可防止其他人偷看程序源代码。Visual Basic 使得编译过程非常轻松，当程序编写完成后，只需要在 File 菜单中执行 Make projectname 命令。

Visual Basic 需要几秒钟来创建.exe 文件。这个执行文件独立于 Visual Basic 环境，并可像任何其他 Windows 应用程序一样被执行。例如，用 Start 菜单中的 Run 命令中来执行。但这并不意味着可将.exe 文件传给其他用户，并希望它工作。事实上，所有的 Visual Basic 程序都依赖于一些辅助文件，Visual Basic 运行时，除非所有这样的文件都正确安装到目标系统上，才会正确执行程序。

2.2.4 保存文件

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“第一个 Visual Basic 程序”。Windows 98 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 8 个字符的文件名外加 3 个字符的扩展名。

2.3 退出 Visual Basic

在程序编写、编译和保存完成以后，可以退出 Visual Basic 集成开发环境，方法有如下几种：

①从“文件”菜单中选取“退出”命令来结束本次工作，退出 Visual Basic 集成开发环境；

②直接点击 Visual Basic 集成开发环境窗口的关闭按钮。

第三章 Visual Basic 语言

3.1 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。

3.1.1 模块

· 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。

· 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他窗体或对象。

· 标准模块

标准模块（文件扩展名为.BAS）是应用程序内其他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。

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,这些模块应视为窗体模块。

• 使用“代码编辑器”

Visual Basic 的“代码编辑器”是一个窗口(见图 3.1),大多数代码都在此窗口中编写。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。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。

对于每个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。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。

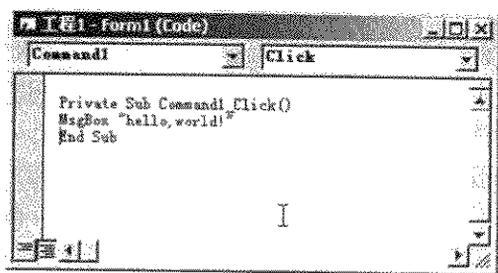


图 3.1 代码编辑窗口

模块通用段的过程列表只包含惟一一段——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。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”菜单中的“选项”命令访问“选项”对话框，在“选项”对话框的“编辑器”选项卡中可用这样的选项，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（如图 3.2 所示）。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。

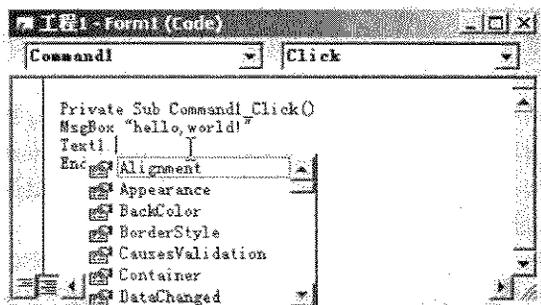


图 3.2 自动列出成员特性

“自动快速信息”功能显示语句和函数的语法（如图 3.3 所示）。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。

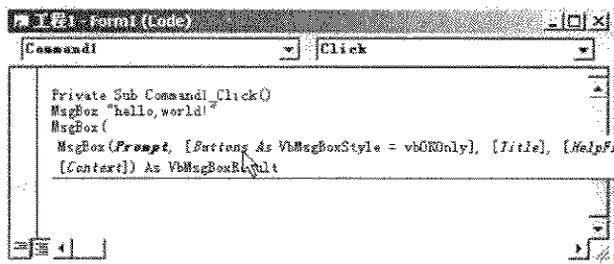


图 3.3 自动快速信息

· 书签

在代码编辑器中，书签用来标记代码的某些行，以便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。

3.1.2 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。

1. 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。

2. 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。在代码中添加注释时常会遇到注释符（'）。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做了示范。记住，不能在同一行上将注释接在续行符之后。

注意：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。

3. 理解数字系统

在本文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。表 3-1 说明同一个数的十进制、八进制和十六进制表示。

表 3-1 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	&O11	&H9
15	&O17	&HF
16	&O20	&H10
20	&O24	&H14
255	&O377	&HFF

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，有的数系就比其他数系更合适，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）。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：

- ① 它们必须以字母开头。
- ② 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）。
- ③ 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。
- ④ 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（如 If 和 Loop）、函数（如 Len 和 Abs）和操作符（如 Or 和 Mod）。

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(I)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 接受其他类型库提供的名称，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。

3.2 变量与常量

3.2.1 声明变量

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，用户就为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他数值，例如，文本数值、日期、各种数值类型，甚至对象也在此列。

声明变量就是事先将变量通知程序，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变故。

变量名：

- ① 必须以字母开头。
- ② 不能包含嵌入的（英文）句号或者嵌入的类型声明字符。
- ③ 不得超过 255 个字符。
- ④ 在同一个范围内必须是惟一的。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 或其他应用程序的对象。Object、Form1 和 TextBox 都是 Visual Basic 对象类型或类的实例。

3.2.2 变量类型

变量是用来存储值的空间，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，数据类型也用于其他场合。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。也可声明任何基本类型的数组。

3.2.3 用户定义的数据类型

用户自定义类型 (UDT) 是一种复合的数据结构，它可以包含多个较简单类型的变量。在使用自定义类型变量之前，必须首先在模块的说明段中用 Type 指令定义它的结构：

```
Private Type EmployeeUDT
    Name As String
    DepartmentID As Long
    Salary As Currency
End Type
```

UDT 可以说明为私有 (Private) 或公用 (Public) 的。在 Visual Basic 5.0 及其以前的版本中，只有在 BAS 模块中的自定义类型才可以说明为公用。在 Visual Basic 6.0 中，除了窗体以外，只要该工程不是标准 EXE 类型的并且类不是私有 (Private) 的，则所有的模块都可以包含自定义类型 (Public UDT) 的定义。

一旦定义了一种结构类型，就可以向创建任何一种 Visual Basic 基本数据类型的变量一样定义变量。然后就可以利用“.”符号来访问结构中的各单项项：

```
Dim Emp As EmployeeUDT
Emp.Name= "Roscoe Powell"
Emp.DepartmentID=123
```

UDT 可以包含变长及定长字符串。在第一种情况下，在内存中该结构只保存一个指针来指向实际的数据。而在后一种情况下，字符串中的字符与该结构中的其他项存储在同一个数据块中。这可通过 LenB 函数来体现，也可将该函数用于任意一个自定义变量来获知实际使用的字节数目：

```
Print LenB(Emp) 'Prints 16:4 for Name, regardless of its length +
                '4 for DepartmentID(Long)+8 for Salary (Currency)
```

结构类型中也可包括子结构，例如：

```
Private Type LocationUDT
    Address As String
    City As String
    Zip As String
    State As String*2
End Type
Private Type EmployeeUDT
    Name As String
    DepartmentID As Long
    Salary As Currency
    Location As LocationUDT
End Type
```

当访问这种嵌套的结构时，可利用 With...End With 语句以获得可读性更高的代码：

```
With Emp
    Print.Name
    Print.Salary
With Location
    Print.Address
    Print.City& " " & .Zip & "" & .State
End With
End Type
```

当使用复杂的 UDT 时，分别给它所有的组成项赋值通常是非常复杂的。由于 VBA 支持返回 UDT 的函数，故可以编写配套的过程来减少工作量：

```
Emp=InitEmployee("Roscoe Powell",123,80000)
...
Function InitEmployee(Name As String,DepartmentID As Long,
    Salary As Currency) As EmployeeUDT
    InitEmployee.Name=Name
    InitEmployee.DepartmentID=DepartmentID
    InitEmployee.Salary=Salary
End Function
```

Visual Basic 允许用普通的赋值方法将一个 UDT 复制为另一个同样结构的 UDT，例如：

```
Dim emp1 As EmployeeUDT, emp2 As EmployeeUDT
...
emp2=emp1
```

3.2.4 变量的范围

• 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(见表 3-2)。

表 3-2 变量的有效范围

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，但对其他模块的代码不可用。

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，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他模块中也有效，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。

3.2.5 常量

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：

① 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。其他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。

② 符号的或用户定义的常数是使用 `Const` 语句来声明的。

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`。

设计前缀时应尽量防止发生意外冲突，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。

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
```

一旦定义了常数，就可将其放置在代码中，使代码更易读懂。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：

① 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。

② 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。

③ 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。

由于常数可以用其他常数定义，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。

3.3 数 组

如果用过其他语言编程，就会熟悉数组的概念。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。在许多场合下，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）。可以声明任何基本数据类型的数组，包括用户定义的类型。

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：

- ① 建立公用数组，在模块的声明段用 Public 语句声明数组。
- ② 建立模块级数组，在模块的声明段用 Private 语句声明数组。
- ③ 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 ~ 2,147,483,647）。例如，下列数组声明可出现在模块的声明段：

```
Dim Counters (14) As Integer      ' 15 个元素。
```

```
Dim Sums (20) As Double          ' 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 0~14。第二个声明建立了一个有 21 个元素的数组，其索引号从 0~20，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 1~15，而 **Sums** 的索引值范围从 100~120。包含其他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 () ' 将其他数组移居到数组。
    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
```

3.4 过 程

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：

- ① 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。
- ② 一个程序中的过程，往往不必修改或只需稍做改动，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：

- ① Sub 过程不返回值。
- ② Function 过程返回值。
- ③ Property 过程返回并指定值，以及设置对象引用。

3.4.1 子程序

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。

3.4.2 函数

Visual Basic 包含内置的或内部的函数,如 Sqr、Cos 或 Chr。此外,还可用 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中。在 Sub 过程与 Function 过程之间有三种区别:

①一般说来,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。

②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 于句,缺省的数据类型为 Variant)。

③给 procedurename 自身赋一个值,就可以返回这个值。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)
```

3.4.3 变元

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 的数据类型。不过，也可以声明参数为其他数据类型。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
```

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他参数也必是可选的，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 int
Numsy = y + x
Next xintSum = 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：①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。②在语法中，命名参数是用粗体和斜体字表示的。所有其他参数只用斜体字表示。

注意：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。

3.5 控制结构

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。

3.5.1 流程控制语句

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：

- ① If...Then
- ② If...Then...Else
- ③ 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

```

```

Combo1.AddItem "BeiJing"
Combo1.AddItem "ChongQing"
Combo1.AddItem "ShangHai"

```

End Sub

运行时，只要加载窗体，而且用户单击向下箭头，则显示如图 4.8 所示的列表。

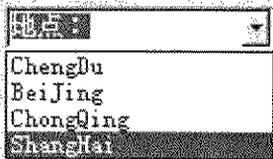


图 4.8 地点列表

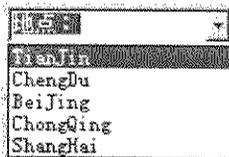


图 4.9 地点列表

• 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即可输入列表项目，然后按 Ctrl+Enter 组合键切换到新的一行。只能将项目添加到列表的末尾。所以，如果要列表按字母顺序排序，则应将 Sorted 属性设置为 True。

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“TianJin”插入到第一个位置并把其他项目的位置向下调整：

```
Combo1.AddItem "TianJin", 0
```

注意：指定列表中的第一个位置的是 0 而不是 1（如图 4.9 所示）。

• 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“tianJin”和“TianJin”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。

(1) 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
```

(2) 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“TianJin”，则通过下列代码显示有关 TianJin 的信息：

```

Private Sub Combo1_Click ()
    If Combo1.Text = "TianJin" Then
        Text1.Text = "TianJin is a big city."
    End If
End Sub

```

```

ElseIf Index = 1 Then      ' “复制” 命令。
    CopyActiveControl
ElseIf Index = 2 Then      ' “清除” 命令。
    ClearActiveControlElse ' “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。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。

• 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      ' “剪切” 命令。

```

```

CopyActiveControl      ' 调用通用过程。
ClearActiveControl
Case 1      ' “复制” 命令。
CopyActiveControl
Case 2      ' “清除” 命令。
ClearActiveControl
Case 3      ' “粘贴” 命令。
PasteActiveControl
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。

3.5.2 循环控制语句

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：

- ① Do...Loop
 - ② For...Next
 - ③ 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 0) 或 False (0) 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 (0)，则跳过所有语句。如果 condition 为 True (非 0)，则 Visual Basic 执行语句，然后返回到 Do While 语句再测试条件。

因此，只要 condition 为 True 或非 0，循环可以随意执行多少次。如果 condition 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
```

其他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。

• For...Next

在不知道循环内需要执行多少次语句时，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，

- (1) 设置 counter 等于 start。
- (2) 测试 counter 是否大于 end。若是，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 1, 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
            ElsefrmVCR.Controls (i).Visible = False
            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个元素, 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 element
```

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 TableDefSet 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 时的一些限制:

- ①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。
- ②对数组, element 只能是 Variant 变量。
- ③For 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。



3.6 类与对象

从 Visual Basic 4.0 开始, 类模块的概念被引入到 Visual Basic 编程之中。使用类主要有以下一些好处:

- ①类模块可以极大地提高程序功能, 可以帮助解决许多常见而错综复杂的编程问题, 甚至还允许执行一些特别困难的任务。
- ②可以使用类来更好地将程序组织成一个真正的可复用的模块, 并利用从 Object-Oriented Design 规则中派生出来的概念设计应用程序。
- ③对象是 Visual Basic 每一个特性的基础。没有对象, 就不能做严格的数据库编程, 不能输送 Internet 应用程序, 也不能为 COM、DCOM 或 MTS 编写组件。简言之, 如果不能牢固的掌握对象是什么以及如何使用它们, 就只能做很少或什么也做不了。

3.6.1 面向对象程序设计语言

· 封装性

对象是惟一拥有自己数据的所有者。所有的数据保存在一个内存区域内部, 不能被另一部分应用程序直接访问, 所有的赋值和获取操作都通过由对象本身提供的方法和属性执行。这个简单概念至少有如下两个作用:

- ①可以在所有赋给对象属性的值存储到内存中之前检查它们, 并立即剔除所有无效的值。
- ②可以自由的对保存在一个对象中的数据的内部实现进行修改, 而不需要改变程序的其他部分与对象相互联系的方式。即是说, 可以以后修改并改进一个类的内部工作, 而不需要改动应用程序中其他地方的任一行代码。

· 多态性

多态性是指不同的类对外界显现相似(或相同)的界面。在 Visual Basic 中最明显的多态性的形式是窗体和控件。TextBox 和 PictureBox 控件是完全不同的对象,但是它们具有一些相同的属性和方法,如 Left、BackColor 和 Move。这种相似性简化了编程工作。更重要的是,它可以利用一个单独的变量(Control、Variant 或 Object)管理一组控件,并创建作用于一个窗体上的所有控件的一般过程,因此显著的减少了编写的代码量。

• 继承性

继承性是许多面向对象语言提供的性能,能够从另一个类(基类)派生出一个新类(派生类或继承类)。派生类继承基类的属性和方法。例如,可以定义一个普通的 Shape 类,具有如 Color 和 Position 这样的属性,然后将之作为一个基类进一步制定新类(例如,Rectangle、Circle 类等),它们继承所有那些类属性。然后可以添加指定成员,如给 Rectangle 类添加 Width 和 Height,给 Circle 类添加 Radius。有趣的是,多态性往往会减少需要使用类的代码的数量,而继承性减少了自身内部的代码,因而简化了类创建者的工作。

3.6.2 对象

在 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,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。

①在 Visual Basic 的“工具箱”中,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。

②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。

把对象看作是类的复制品,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。

• 用对象能做什么

对象可提供现成代码,省去书写的麻烦。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,但也可大可不必。在此也可使用 Microsoft Project 提供的 Calendar、Resources 和 Task 对象。Visual Basic 可以组合来自其

他来源的对象, 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, 以开发工具和其他应用程序。

可把 Visual Basic 内部的控件结合在一起, 也可使用其他应用程序提供的对象。考虑把下列对象放入 Visual Basic 窗体:

- ①Microsoft Excel Chart 对象
- ②Microsoft Excel Worksheet 对象
- ③Microsoft Word Document 对象

· 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 0~10 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。一般来说, 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。

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
```

下而是设置属性的语句:

```

Text1.Top = 200          ' 设置 Top 属性为 200 缇 (twips)。
Text1.Visible = True    ' 显示文本框。
Text1.Text = "hello"    ' 在文本框中显示 "hello"。

```

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，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

```

注意：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。

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。

在代码中使用方法时如何书写语句，这取决于该方法要求多少个参数，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中圆的位置、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"。
```

第四章 Visual Basic 的标准控件

4.1 Visual Basic 控件简介

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他应用程序并处理数据，这时，那些远程应用程序就好像是代码的一部分。每个控件都有一组属性、方法和事件。

4.1.1 控件分类

Visual Basic 的控件有三种广义分类：

①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（见图 4.1），不像 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。

②ActiveX 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）和仅在专业版、企业版中提供的控件（例如 ListView、Toolbar、Animation 和 TabledDialog），另外还有许多第三方提供的 ActiveX 控件。

注意：具有文件扩展名 .vbx 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 .vbx 控件的工程时，在节省情况下用 .ocx 控件取代 .vbx 控件，当然，这只有在控件的 .ocx 版本存在时才可以。

③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。

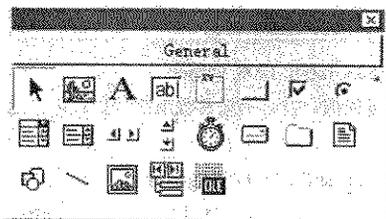


图 4.1 控件工具箱

· 内部控件

表 4-1 总结了 Visual Basic 工具箱中的内部控件。

表 4-1 工具箱中的内部控件

图 标	控件名	类 名	描 述
	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他控件的可视容器
	标签	Label	为用户显示用户不可交互操作或不可修改的文本
	文本框	TextBox	提供一个区域来输入文本、显示文本
	框架	Frame	为控件提供可视的功能化容器
	命令按钮	CommandButton	在用户选定命令或操作后执行它
	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
	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
	文件列表框	FileListBox	显示文件列表并允许用户从中进行选择
	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
	定时器	Timer	按指定时间间隔执行定时器事件
	水平和垂直滚动条	HScrollBar 和 VScrollBar	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)
	列表框	ListBox	显示项目列表, 用户可从中进行选择
	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
	形状	Shape	向窗体、框架或图片框添加矩形、正方形、椭圆或圆形
	线形	Line	在窗体上添加线段
	图像	Image	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
	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
	OLE 容器	OLE	将数据嵌入到 Visual Basic 应用程序中

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 学习版包含若干个 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。表 4-2 总结了 Visual Basic 学习版提供的标准 ActiveX 控件。

表 4-2 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，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

· 添加和删除 ActiveX 控件

按照下述步骤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，就像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误信息。

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，则将显示出错误信息。

· 更新旧版 Visual Basic 控件

文件扩展名为.vbx 的十六位旧版 Visual Basic 控件与此版 Visual Basic 不兼容。若试图加载包含.vbx 控件的旧工程，Visual Basic 将会警告控件不适用或不兼容，此时，可以继续加载无.vbx 控件的工程，当然，应用程序将无法正常运行。如果有包含第三方.vbx 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。

· 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 Commandn，其中 n 为 1, 2, 3 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，特别是在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：

- ①必须以字母开头。
- ②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。
- ③不能超过 40 个字符。

4.1.2 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，此属性称为控件的值，这是控件的最重要或最常用的属性。表 4-3 列出了每个控件属性，这些属性被看成是控件值。

表 4-3 控件值

控 件	值
CheckBox (复选框)	Value
ComboBox (组合框)	Text
CommandButton (命令按钮)	Value
CommonDialog (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
DirListBox (目录列表框)	Path
DriveListBox (驱动器列表框)	Drive
FileListBox (文件列表框)	FileName
FlexGridTextFrame (框架)	Caption
HScrollBar (水平滚动条)	Value
Image (图像)	Picture
Label (标签)	Caption
Line (线形)	Visible
ListBox (列表框)	Text
OptionButton (选项按钮)	Value
PictureBox (图片框)	Picture
Shape (形状)	Shape
TextBox (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，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，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，则第一个控件上的 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的。相反,通过使用 Validate 事件,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 MaskedEdit 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。这样能有效地防止用户单击其他控件。

- 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内增加;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控件剪切和粘贴到窗体上添加的引用控件。

- 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,如果窗体中有若干文本框,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。

4.2 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,可以用最少的代码创建数据库应用程序,如图 4.2 所示。



图 4.2 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，用法如下：

- ①连接一个本地数据库或远程数据库。
- ②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。
- ③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。
- ④添加新的记录，或根据对显示在绑定的控件中的数据的任何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 控件以及其他所需要的任何 Visual Basic 控件，可以根据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 (DataLink Name) (MDL) 必须是在机器上创建的。要创建一个简单的前端数据库应用程序，按如下步骤执行：

(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) 在窗体上再性“窗口”中，将 Text1 的“数据源”属性设为 ADO Data 控件的名称

(ADO-DC1)。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的名称。

(9) 对希望访问的其他每个字段重复第 (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。表 4-4 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。

表 4-4 ADO Data 控件事件

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
WillChangeRecord	当执行 Recordset.Update、Recordset.Delete、Recordset.Cancel Update、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

· 设置 ADO Data 控件的与数据库相关的属性

当创建连接时，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)，和一个 ODBC 数据源名称(DSN)。当使用 DSN 时，则无须更改控件的任何其他属性。

不过，如果对数据库技术比较了解，可以更改在 ADO Data 控件中出现的其他的一些属性。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、用户名以及密码等。

(2) **UserNam**——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, 是位于客户还是服务器上。这一决策将影响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晌。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、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。

4.3 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, 如图 4.3 和图 4.4 所示。



图 4.3 CheckBox 控件



图 4.4 CheckBox 的状态

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。

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)中的一种。选定时, Value 设置值为 1。

下表列出用于设置 Value 属性的数值和相应的 Visual Basic 常数。

表 4-5 设置 Value 属性的数值和常数

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 Backspace 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。如图 4.5 的例子。



图 4.5 响应鼠标和键盘

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 控件像 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。

➡ 4.4 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，用户可通过在组合框中输入文本来选定项目，也可从列表中选定项目。如果项目数超过了组合框能够显示的项目数，控件将自动出现滚动条，用户即可上下或左右滚动列表，如图 4.6 所示。



图 4.6 ComboBox 控件组合框

• 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到区域中。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。

• 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。

• 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（见表 4-6）。

表 4-6 组合框样式

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

· 下拉式组合框

在缺省设置 (Style = 0) 下，组合框为下拉式。用户可（像在文本框中一样）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。如图 4.7 所示。

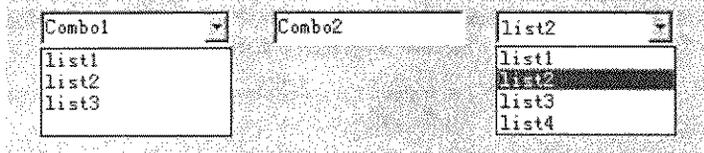


图 4.7 ComboBox 控件的三种样式

· 简单组合框

将组合框 Style 属性设置为 1 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。像下拉式组合框一样，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 (AddItem 参数见表 4-7)，其语法如下：

```
box.AddItem item[, index]
```

表 4-7 AddItem 参数

参 数	描 述
Box	列表框或组合框名称
Item	在列表中添加的字符串表达式。若 item 为文字常数，则用引号括起来
Index	指定新项目在列表中的插入位置。Index 为 0 表示第一个位置。若省略 index，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“ChengDu”、“BeiJing”、“ChongQing”和“ShangHai”放置到名为 Combo1、Style 属性为 0 (vbComboDropDown) 的组合框中：

```
Private Sub Form_Load ()
    Combo1.AddItem "ChengDu"
```

```

Combo1.AddItem "BeiJing"
Combo1.AddItem "ChongQing"
Combo1.AddItem "ShangHai"

```

End Sub

运行时，只要加载窗体，而且用户单击向下箭头，则显示如图 4.8 所示的列表。

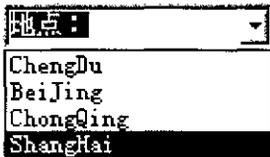


图 4.8 地点列表



图 4.9 地点列表

• 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 组合键切换到新的一行。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。

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“TianJin”插入到第一个位置并把其他项目的位置向下调整：

```
Combo1.AddItem "TianJin", 0
```

注意：指定列表中的第一个位置的是 0 而不是 1（如图 4.9 所示）。

• 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“tianJin”和“TianJin”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。

(1) 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
```

(2) 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“TianJin”，则通过下列代码显示有关 TianJin 的信息：

```

Private Sub Combo1_Click ()
    If Combo1.Text = "TianJin" Then
        Text1.Text = "TianJin is a big city."
    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“TianJin”，若是，则在文本框中显示信息。

(3)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)
```

(4)用 ListIndex 属性判断位置

欲知组合框列表中选定项目位置，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。

(5)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"
```



4.5 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。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，而且看起来就像是被按下和松开一样，因此有时称其为下压按钮，如图 4.10 所示。

图 4.10 CommandButton 控件

(1)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，就像在其他控件绘制按钮那样，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。

(2)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。

(3)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,或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。

(4)指定 Default 和 Cancel 属性

在每个窗体上部可选择一个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。

(5)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:

- ①用鼠标单击按钮。
- ②按 Tab 键,将焦点转移到按钮上,然后按 Backspace 或 Enter 键选定按钮。
- ③按命令按钮的访问键 (Alt+ 带有下划线的字母)。
- ④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他控件上,情况也是如此。
- ⑤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他控件上,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
```

(1)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。在 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。

(2)增强命令按钮的视觉效果

命令按钮像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,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。

4.6 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。CommonDialog 控件在 Visual Basic 和 Microsoft Windows 动态连接库 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，如图 4.11 所示。



图 4.11 CommonDialog 控件

CommonDialog 控件可以显示如下常用对话框：

- ① “打开”对话框
- ② “另存为”对话框
- ③ “颜色”对话框
- ④ “字体”对话框
- ⑤ “打印”对话框

(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。在标记对话框的“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。像 Timer 控件一样，CommonDialog 控件在运行时不可见。

(3) 运行时，请适当使用表 4-8 所列方法显示需要的对话框。

表 4-8 显示对话框

方 法	显示的对话框
ShowOpen	打开
ShowSave	另存为
ShowColor	颜色
ShowFont	字体
ShowPrinter	打印
ShowHelp	调用 Windows “帮助”

4.6.1 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，按如下步骤执行：

(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, 如图 4.12 所示。

(3) 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。

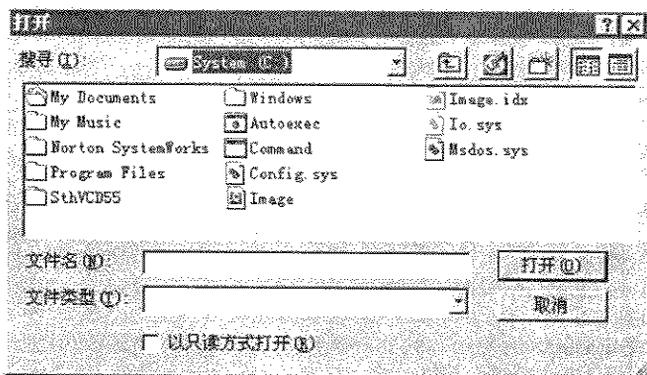


图 4.12 ShowOpen 方法显示对话框

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
```

4.6.2 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，如图 4.13 所示。要显示“颜色”对话框，按如下步骤执行：

- (1) 将 CommonDialog 控件的 Flags 属性设置成 Visual Basic 常数 cdCCRGBInit。
- (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 = cdCCRGBInit
    ' 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
```

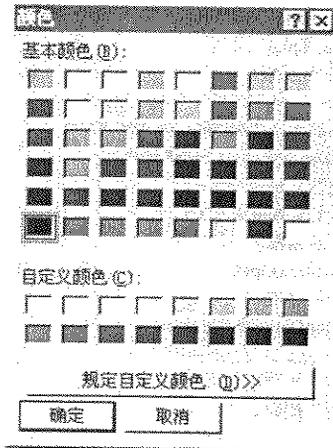


图 4.13 ShowColor 方法显示对话框

4.6.3 使用“字体”对话框

“字体”对话框根据大小、颜色、样式选择字体（如图 4.14 所示）。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（见表 4-9）。

表 4-9 字体属性

属 性	决 定
Color	选定的颜色。为使用此属性，必须先将 Flags 属性设置为 cdlCFEffects
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

(1)显示“字体”对话框

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）

注意：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
```

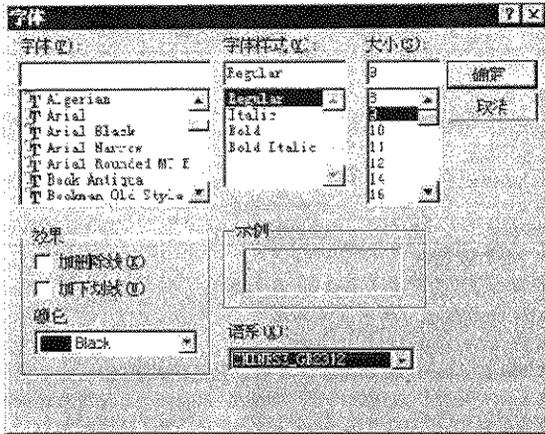


图 4.14 字体对话框

4.6.4 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。此对话框还显示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，但必须编写代码实现用选定格式打印数据。

运行时，当用户在“打印”对话框作出选择后，如表 4-10 所示，下述属性将包含用户选项的信息。

表 4-10 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
HDCOrientation	选定打印机的设备上下文页面定向（图像或风景）

要显示“打印”对话框，按如下步骤执行：

(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
    ' 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
ErrorHandler:
' 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。

4.6.5 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，按下如步骤执行：

(1) 设置 HelpCommand 和 HelpFile 属性。

(2) 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 ErrorHandler
' 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

```

4.7 Data 控件

内在的 Data 控件是通过使用 Microsoft 的 Jet 数据库引擎来实现数据访问的，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、末尾以及在记录集中向前或向后翻页。

· 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，按如下步骤执行：

- (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名称。
- (9) 对其他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) RType——DefaultType 属性指定所使用的是 JET 工作空间，还是 ODBCDirect 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——这个属性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）。



4.8 DataCombo 控件和 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，从而适合于创建“查找表”应用程序。

· 可能的用法

①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。

②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。

· 值得注意的控件属性

DataList 和 DataCombo 控件的一些重要属性，如表 4-11 所示。

表 4-11 DataList 和 DataCombo 控件的属性

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。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了该出版商，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 表中。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。

简而言之，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。表 4-12 概要地介绍这些属性及其使用方法。

表 4-12 部分属性及其使用方法

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 属性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。

4.9 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 来查看不同的表, 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

① 查看和编辑在远程或本地数据库中的数据。

② 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 控件来设计特性

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。下面概要地说明在实现 DataGrid 控件的典型应用时的一般步骤。

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。

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的方法是改变该 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，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 DataGrid 控件应用于一个远程数据库时，诸如 SQLServer 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

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。

• 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，按如下步骤执行：

- (1) 打开 Windows Explorer 或 Windows NT Explorer。
 - (2) 打开想要创建 OLE DB 数据源的目录。在该示例中，打开 ProgramFiles、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中的“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) 键单击文件并单击上下文菜单中的“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。
- 单击“测试连接”，检测连接。
- 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，按如下步骤执行：

- (1) 用 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，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，按如下步骤执行：

(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 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“MicrosoftDataGrid Control”、“Microsoft DataList Controls”以及“Microsoft ADOControl”。

(3) 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) 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) 将一个 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.mdb)。选择名为 adoProducts 的控件，并重复该操作。

(6) 置这两个 ADO Data 控件的 RecordSource 属性。选择 adoSuppliers 并在其“属性页”上单击“RecordSource”。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“RecordSource”，并输入 Select *From Products。这个查询将返回在 Products 表中的所有记录。

(7) 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) 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) 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.BoundText
    With adoProducts
        .RecordSource = strQuery
        .Refresh
    End With
    With DataGrid1
        .ClearFields.
        ReBind
    End With
End Sub
```

(12) 更新工程。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
    End With
End Sub
```

```

.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
                ' 隐藏其他所有的列。
                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 10 个字段组成的大表。在这种情况下，在控件中查看的记录集将有 10 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（不重新排列的顺序），可以对网格进行拆分。

(1) 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

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），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
```

(2) 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。

(3) 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：ControlNavigation、Column Navigation 和 Grid Navigation。当该属性设置为 ControlNavigation 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 键和箭头键的行为。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，按 Tab 键或箭头键，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第一列中的下一行。

(4) 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，需要将其他的每一列的 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(0).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。

(1) 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
```

通过在程序中添加到 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, 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 美元的所有记录, 则对记录进行过滤, 并将结果书签添加到 SelBookmarks 集合中。

```

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

```

(2) 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, 按如下步骤执行:

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一个数据源, 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 NewADODB.Recordset 的变量, 来创建一个 ADODB recordset 对象。在创建了这个 recordset 对象后, 再添加工段, 每个数据源中的每个字段都要加入, 然后使用合适的数字填充这个记录集。

注意: 也可以使用 OLE DB 示例提供者来创建一个数据源。

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，按如下步骤执行：

- (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) 该类模块的 Declarations 部分，创建一个 ADO DB Recordset 变量，如下所示：
Option Explicit Private WithEvents rsNames As ADO DB.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) 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
    ' 这使其他对象可以看见这些可用的
    DataMembersDataMembers.Add "Names"
    Set rsNames = New ADO DB.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（位于左上角），然后单击“rsNames”。在“过程/事件”框（位于右上角）中，下拉列表将显示这个 Recordset 对象的所有事件。

(1) 给类添加一个属性

类模块也可以进行修改，用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。当从其他对象调用该对象时，这一属性将返回该类的 RecordCount。

```
Public Property Get RecordCount() As Long
    RecordCount = rsNames.RecordCount
End Sub
```

(2) 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，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
```

4.10 文件系统控件

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。

• 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 4.15 文件系统

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。如图 4.15 所示的是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，这些文件的名字匹配一种模式，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"
```

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他基于 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。

• 驱动器列表框

驱动器列表框是下拉式列表框。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中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。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。

(1) 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，如果 Drive1.Path 赋以 "c:\payroll”，则目录 \Payroll 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中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。

(2) 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 属性并把它 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。

(3) 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 - 1Loop
' 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 并把其他属性设置为 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。缺省时代，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。

• 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。

· 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和目录并显示所有类型的文件。表 4-13 总结了 WinSeek 应用程序在 Seek.frm 窗体中使用的控件。

表 4-13 Seek.frm 中使用的控件

控 件	属 性	设置值
DriveListBox	Name	drvList
DirListBox	Name	dirList
FileListBox	Name	FillList Pattern **
第一个 CommandButton	Name	CmdSearch Caption &Search Default True
第二个 CommandButton	Name Caption	CmdExit&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。

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。当用户选定项目并改变窗体的焦点时也将触发 Change 事件。在目录列表框中, 需要 DblClick 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中的项目时, 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。

• 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) 变更 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。
fillList.Path = dirList.Path
End Sub

```

此事件过程将 dirList 列表框中的 Path 属性赋予 fillList 列表框中的 Path 属性。这将在 fillList 列表框中触发重新绘制的 PathChange 事件; 不必在 fillList_PathChange 过程中添加代码, 因为此应用程序中的事件在 fillList 列表框中已结束。

• 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)
    End 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 等属性的文件。这将把搜索限制在满足条件的文件中。

4.11 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。

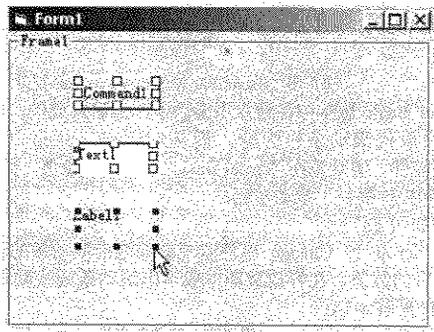


图 4.16 Frame 控件

在大多数的情况下，框架控件的用法是比较“消极的”，可以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。

• 在窗体中添加一个 Frame 控件

在使用框架控件分组其他选项时，首先绘出框架控件，然后再绘制它内部的其他控件。这样在移动框架时，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时使用双击方法，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，在移动时将不得不分别移动框架和控件。

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时 Ctrl 键。在释放鼠标时，位于框架之内的控件将被选定，如图 4.17 所示。



图 4.17 选择框架中的多个控件

4.12 HScrollBar 控件和 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，很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。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 窗体中的滚动条。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，如图 4.18 所示。



图 4.18 HScrollBar 控件和 VScrollBar 控件

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 98 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。

Scroll Bar 控件如何工作：

滚动条控件用 Scroll 和 Change 事件监视滚动框沿滚动条的移动。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。滚动条事件见表 4-14。

表 4-14 滚动条事件

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

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` 属性设置的数值来决定。要设置滚动条在运行时的位置，可将 `Value` 属性设为 0~32,767 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窗应用程序。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。

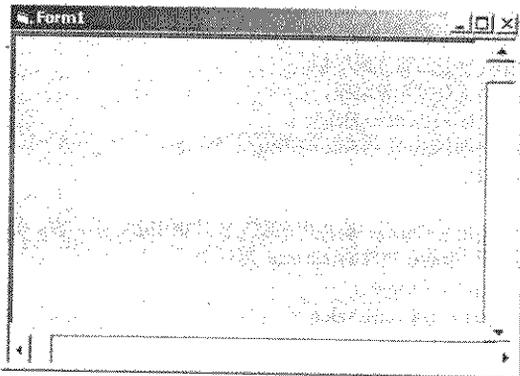


图 4.19 创建可滚动的图形视口

如图 4.19 所示，先创建一个新工程，然后在窗体中绘制两个图片框、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.HeightPicture2.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。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, 为了调整图形视窗应用程序的大小, 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

```

4.13 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，如图 4.20 所示。



图 4.20 Image 控件

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。

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。

· 支持的图形格式

Image 控件可显示下述标准格式的图片文件。如表 4-15。

表 4-15 Image 控件支持的文件格式

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 32×32，但在 Microsoft Windows 95 下，图标也可作为 16×16 像素。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。游标的扩展文件名为 .cur
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	是一种支持 8 位和 24 位颜色的压缩位图格式。它是 Internet 中流行的一种文件格式
GIF	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 中流行的一种文件格式

· 将图形加载到 Image 控件中

在设计时，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——不管在窗体中把 Image 控件画得有多大或有多少小。如果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，可粘贴其他应用程序中的图形，由此即可将图形加载到 Image 控件中。例如，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。

4.14 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，如图 4.21 所示。

A

图 4.21 Label 控件

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，所以被用来为其他控件创建访问键。

• 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。在设计时，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 LeftJustify(0, 缺省)、Center(1)或者 Right Justify(2)。

• AutoSize 和 WordWrap 属性

在缺省情况下,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。

· 用标签创建访问键

如果要将 Caption 属性中的字符定义为访问键,应将 UseMnemonic 属性设置为 True。定义了 Label 控件的访问键后,用户按 Alt+访问键指定的字符,就可将焦点按 Tab 键的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(&),就可为其他具有相同 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,可使用用标签。标签不接受焦点,因此焦点将按照 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) 先绘制标签,然后再绘制控件。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) 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,就会出现上述情况。为在 Label 控件中显示连字符,应将 UseMnemonic 属性设置为 False。

4.15 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,如图 4.22 所示。

图 4.22 Line 控件

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。

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他图形。要完成高级的功能,应使用 Line 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。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。

• 在窗体中画线

可用 Line 控件在窗体上绘制简单直线,按如下步骤执行:

- (1) 在工具箱中选择 Line 控件。指针移动到窗体中时将变成一个十字形。
- (2) 在窗体中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。



4.16 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,如图 4.23 所示。



图 4.23 ListBox 控件

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,但在缺省时将单列列表中垂直显示选项。如果项目的数目超过了列表框可显示的数目,控件将自动出现滚动条。这时用户可在列表上中、下、左、右滚动。

• 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。

• 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,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]
```

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。

表 4-16 Listbox 控件常用参数

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 item 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。index 为 0 表示第一个位置；若省略，则将项目插入在末尾（或按排序次序插入在适当的位置）

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 4.24 所示的列表。

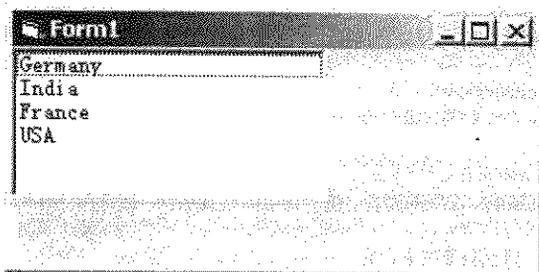


图 4.24 "Countries" 列表框

· 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他项目向下调整：

```
List1.AddItem "Japan", 0
```

注意：是 0 而不是 1 指定列表中的第一个位置（如图 4.25 所示）。

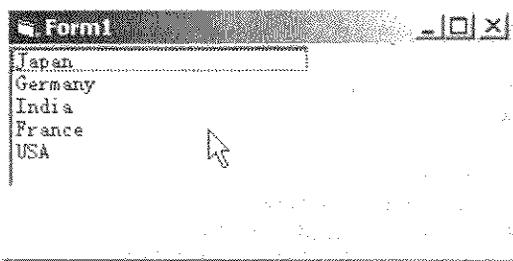


图 4.25 “Countries”列表框

· 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 组合键执行。

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。

· 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
```

要删除连续版或标准版的列表、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的方法是使用 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 属性可用于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"
```

• 创建多列和多选项列表框

可用 Columns 属性指定列表框中的列数目。此属性取值如表 4-17。

表 4-17 Columns 属性

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 属性取值如表 4-18 所示。

表 4-18 MultiSelect 属性取值

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 Backspace 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

➡ 4.17 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他基于 Windows 的应用程序（例如 Microsoft Excel 和 Microsoft Word for Windows）中的数据。

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 容器控件把其他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。

➡ 4.18 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，如图 4.26 所示。



图 4.26 OptionButton 控件

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，同组中的其他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。

• 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中的一个 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，应按按钮放置在框架或 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 Ctrl 键的同时用鼠标在这些控件周围绘制一个方框。

• 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。

- 禁止选项按钮

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。

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。

4.19 PictureBox 控件

PictureBox 控件被用来显示图形，作为其他控件的容器。显示图形方法的输出或显示 Print 方法输出的文本。

PictureBox 控件和 Image 控件相似，每个控件都用来显示应用程序中的图形——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，作为其他控件的容器的功能和支持图形方法的功能。

- 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。

- 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)
```

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 (.ico) 和光标 (.cur) 文件。LoadPicture 函数的设置允许从 .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: 从其他应用程序中复制图形后把它粘帖到 PictureBox 控件中。例如, 有时可能希望添加由 WindowsPaint 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“粘帖”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。

- 用 PictureBox 控件作容器

可用 PictureBox 控件作为其他控件的容器。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。

- 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, 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, 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。

- 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。



4.20 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。

可以设置绘制在窗体中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。

- 预定义的形状

Shape 控件的 Style 属性提供了六种预定义的形状。表 4-19 列出所有预定义形状、形状值和相应的 Visual Basic 常数。

表 4-19 Style 属性的预定义形状

形 状	样 式	常 数
矩形	0	vbShapeRectangle
正方形	1	vbShapeSquare
椭圆形	2	vbShapeOval
圆形	3	vbShapeCircle
圆角矩形	4	vbShapeRoundedRectangle
圆角正方形	5	vbShapeRoundedSquare

- Fill 和 Line 样式

可用 FillStyle 和 BorderStyle 属性对绘制在窗体中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。

- 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（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。



4.21 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。如图 4.27 所示。



图 4.27 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 32KB 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。

- 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。

- 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 4.28 所示,无论用户输入什么字符,文本框中都显示星号。

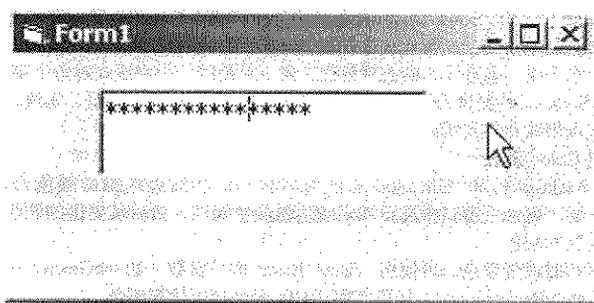


图 4.28 密码示例

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
```

- 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, 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, 就可在文本框中使用“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)
```

4.22 Timer 控件

使用 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。对于其他后台处理, Timer 控件也非常有用。

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:

① 如果应用程序或其他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。

② 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)。

③ 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。

④ 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他操作, 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。

· 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、跑表或其他定时设备的功能。

Digital Clock 应用程序包含一个定时器和一个有边框的标签。定时器在运行时不可见。表 4-20 列出了 Digital Clock 应用程序中的属性设置值。

表 4-20 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

下列应用程序中的惟一过程是定时器的过程：

```
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，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。若要用其他格式显示，可使用 Format 函数。

4.23 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之内增加；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。

• 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。

• 设计时创建控件数组

设计时有三种方法创建控件数组：

- ① 将相同名字赋予多个控件。
- ② 复制现有的控件并将其粘贴到窗体上。
- ③ 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。

要通过改变控件名称添加控件数组元素，按如下步骤执行：

(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; 其他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, 按如下步骤执行:

(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, 其参数设置见表 4-21, 然后在运行时使用如下语法:

Load object(index%)

Unload object(index%)

表 4-21 参数

参 数	描 述
object	在控件数组中添加或删除的控件名称
index	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。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, 无论它们是不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。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。

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
```

表 4-22 列出了应用程序中对象的属性设置值。

表 4-22 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 newcolor
Option1	Name Index	OptButton 0
Option2	Name Index	OptButton 0
第一个 CommandButton	Name Caption	CmdAdd &Add
第二个 CommandButton	Name Caption	CmdDelete &Delete
第三个 CommandButton	Name Caption	CmdClose &Close

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 10 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 + 400optButton(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
```

第五章 键盘和鼠标的事件

5.1 事件和事件过程

世界上的每一种生物都会对外界的刺激作出反应，但是不同的生物对同一种刺激作出的反应常常是大相径庭的，成语“对牛弹琴”从另一个侧面也说明了这个问题。在面向对象的程序设计这个世界里，同样存在着这样的问题，只不过在面向对象的程序设计中，为来自外部的刺激使用了一个术语——“事件”，而将对象对事件的反应称为“事件过程”。

5.1.1 事件驱动

“可视化”和“事件驱动”是使用 Visual Basic 进行 Windows 程序设计的精髓所在。“事件驱动”的意思是只有在事件发生时，程序才会运行。在没有事件的时候，整个程序是处于停滞状态的，这一点和 DOS 程序有极大的差别。在 Visual Basic 设计的程序中，流动的不是数据而是事件。如果说属性决定了对对象的外观，方法决定了对对象的行为，那么事件就决定了对对象之间联系的手段。

而事件本身也并没有什么神秘的，在 Visual Basic 中，事件就是能被对象所识别的动作，使用鼠标单击或者双击就是最常见的事件。此外，用户的键盘输入、鼠标的移动、窗体的载入，还有定时器产生的定时信号……，都是事件。

5.1.2 Visual Basic 对象识别事件

用户不必关心所使用的对象需要响应的事件类型，因为 Visual Basic 的每一个窗体有一个预定义的事件集，它们能够自动识别属于事件集中的事件。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，一个命令按钮和窗体都可以对(Click 单击)、Dbclick(双击)和 KeyPress(按键)这样的事件作出响应。而某些事件只可能发生在某些对象上，这就像领导可以训斥下级，却不能去训斥桌椅一样，“训斥”这样的事件决不会发生在桌椅上。

相同的事件发生在不同的对象上所得到的反应是不一样的，每个对象对每个可以识别的事件都有一个“事件过程”。当事件过程不同时，对事件所表现出来的反应自然也会不同。

事件过程的语法是这样的：

```
Sub 对象名_事件 ( )
```

处理事件的代码

```
End Sub
```

现在可以理解下面两个处理鼠标单击事件的事件过程（如图 5.1 所示）：

```
Private Sub Form_Click()
    Form.Caption="这是第一个应用程序！"
    Form.Circle(1920, 1300), 800
End Sub
```

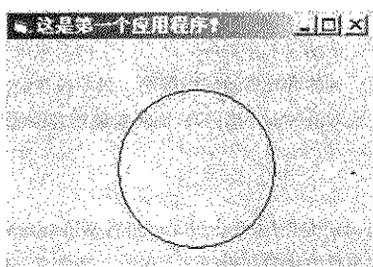


图 5.1 鼠标单击事件

Form_Click 事件过程可以完成这样的工作：将窗体的标题赋值为字符串，并且在窗体上画一个圆。而下面的 Command1_Click 事件过程则可以将窗体的外观恢复原状：

```
Private Sub Form_Click()
    Form1.Caption = "Form1"
    Form1.Cls
End Sub
```

5.1.3 工作过程

在 Visual Basic 程序设计中，基本的设计机制就是改变对象的属性、使用对象的方法和为对象事件编写事件过程。虽然对象可以自动识别事件，但是如果不必要，用户不必为所有的事件都编写事件过程，Windows 系统会以默认的方式来处理事件。当用户感觉在程序中不需要对某个事件进行额外处理时，可以不去管它。使用 Visual Basic 编程的妙处就在于：只有当用户要以某种特定的方式响应某个事件时，才需要编写针对这个事件的事件过程。

下面列出了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，产生 Form_Load 和 Form_Show 事件。
2. 窗体或窗体上的控件接受事件。事件可由用户引发（例如用键盘或鼠标操作），可由系统引发（例如定时器事件），也可由代码间接引发（例如当代码装载其他窗体时产生

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。

例如, 当应用程序中使用了一个窗体时, 在程序开始运行时窗体接受到 Load 消息, 从而触发窗体的 Load 事件, 任何窗体的默认的 Load 事件过程都与下面的例子大同小异(只是窗体的名字不同):

```
Sub Form_Load()
```

```
...
```

```
End Sub
```

在这个默认的事件过程中, 什么语句也没有, 这个过程仅仅是一调用就结束了, 而 Visual Basic 则执行了默认的行为——将窗体装入内存。如果必要, 程序员可以利用这个事件过程多做些事件, 例如要在程序运行前对某些属性进行初始化, 就可以针对 Load 事件编写事件过程, 在事件过程中初始化这些属性(如图 5.2 所示):

```
Sub Form_Load()
```

```
Form1.Caption = "你好!"
```

```
End Sub
```

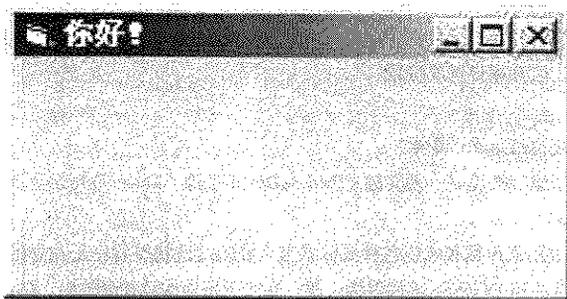


图 5.2 初始化 Caption 属性

重新编写了 Load 事件过程后, 不但窗体被调入内存, 而且还对窗体的 Caption 属性进行了赋值。

5.2 响应鼠标事件

鼠标和键盘一样, 都是用户与程序之间交互操作中的主要元素, 尤其是在 Windows 环境下, 鼠标的重要性甚至超过了键盘。Visual Basic 应用程序能够响应多种鼠标事件。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，还能响应鼠标按键与 Shift、Ctrl 或 Alt 键的各种组合。

在程序中，程序员要了解鼠标位置及状态的变化，可以使用 MouseDown、MouseDown、MouseMove 事件过程。在 Visual Basic 中的大多数控件能够识别这些鼠标事件。表 5-1 列出了这 3 种事件过程的触发条件。

表 5-1 三种鼠标事件的触发条件

事 件	描 述
MouseDown	按下任意鼠标按键时发生
MouseUp	释放任意鼠标按键时发生
MouseMove	每当鼠标指针移动到屏幕新位置时发生

三种鼠标事件均使用下列参数：

- ① Button 参数是一个位域参数，其中的 0、1、2 三位描述鼠标按键的状态。
- ② Shift 参数也是一个位域参数，其中的 0、1、2 三位描述 Shift、Ctrl 与 Alt 键的状态。
- ③ x,y 参数指明了鼠标的指针位置，这里用到了接受鼠标事件的对象的坐标系统描述的鼠标指针位置。

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。

如果按下鼠按钮不放，则对象将继续识别所有鼠标事件，即使此时指针已移离对象，情况也是如此，直到用户释放按钮。

5.2.1 MouseDown 事件

当用户按下鼠标按键时，就会触发 MouseDown 事件。MouseDown 是 3 种鼠标事件中最常使用的事件。

鼠标事件被用来识别和响应各种鼠标状态，并把这些状态看作是独立的事件，不应将鼠标与 Click 事件和 DblClick 事件混为一谈，在按下鼠标按键并释放时，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 DblClick 事件之处还在于：鼠标事件能够区分各鼠标按键与 Shift、Ctrl、Alt 键。

例如，在运行时将 MouseDown 事件与 Move 方法联合起来使用，就可以把控件移动到窗体的不同位置。在单击窗体的任意位置(不要在控件上)时，控件将移动到按下鼠标按键时鼠标指针所在的位置。下面以窗体上的一个命令按钮为例：

```
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。还可以修改此过程，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
```

如果能够在 MouseDown 事件中配合对鼠标按键的处理，就能够使应用程序具有更多的功能。例如在下面的例子中，只使用一个事件，就可以完成连续画直线的功能。用户只要按下左键就可以开始连续画直线，如果要停止连续画线，只须单击鼠标左键即可。下面是程序的代码，如图 5.3 所示的是程序运行的结果。

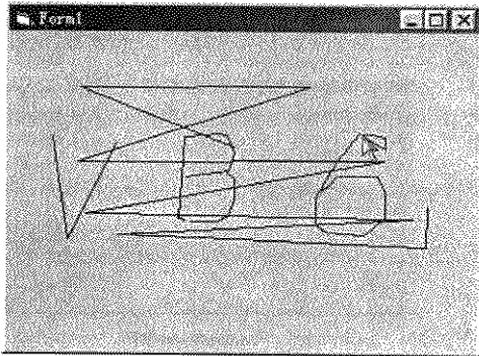


图 5.3 使用 MouseDown 事件配合不同的鼠标按键完成连续画线

```
Private Sub Form_MouseDown(Button As Integer,Shift As Integer,X As Single,Y As Single)
    Static blnDrawNow As Boolean
    If blnDrawNow Then
        Form1.Line - (X, Y)
        If Button = 2 Then blnDrawNow = False
    ElseIf Button= 1 Then
        Form1.CurrentX = X
        Form1.CurrentY = Y
        blnDrawNow = True
    End If
End Sub
```

在这段代码中需要注意的是：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。在默认状态下，绘图坐标是相对于上一次的绘图点的，在 Form_MouseDown 过程中重新设置窗体的 CurrentX 与 CurrentY 属性，就可以指定新的开始绘图起始位置。

在 MouseDown 事件过程中，使用 Button 参数可判断按下了哪个按钮，但是 MouseDown 事件无法检测是否同时按下了两个以上的按钮。如果按下多个按键，Visual Basic 就会将操作解释为两个或多个独立的 MouseDown 事件。例如在下面的代码中，如果用户同时按下左右键，不会一无所获，看到的反而是两条消息（如图 5.4 所示）：

```
Private Sub Form_MouseDown (Button As Integer, Shift As Integer, X As Single, Y As Single)
    If Button = 1 Then Print "您按下了鼠标的左键。"
    If Button = 2 Then Print "您按下了鼠标的右键。"
End Sub
```

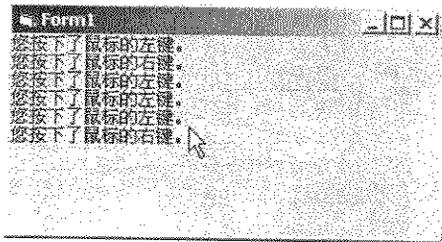


图 5.4 MouseDown 事件

5.2.2 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。

每当鼠标指针位置变更时，都要触发 MouseMove 事件。在下面的例子里，就使用 MouseMove 事件和 Line 方法绘制出鼠标移动的轨迹，创建一个涂鸦程序。为了控制是否绘制鼠标轨迹，在代码中要对 Button 参数进行判断，如果用户按下了左键，那么就on始绘制鼠标移动的轨迹(如图 5.5 所示)；否则，任由鼠标到处移动而不在窗体上留一点痕迹。示例的代码很简单，只有寥寥几行：

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
```

```

If Button = 1 Then Line -(X, Y)
End Sub

```

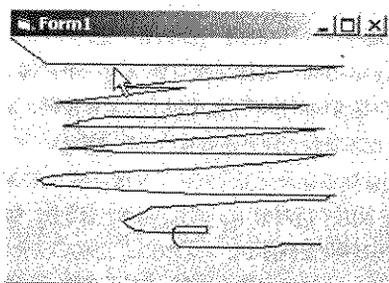


图 5.5 用 MouseMove 事件与 Line 方法创建的涂鸦程序

那么，系统是如何识别 MouseMove 事件的呢？当指针移过屏幕时要调用多少次 MouseMove 事件呢？

实际上，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，可用下述代码在每次识别 MouseMove 事件时，应用程序都绘制一个小圆圈。运行结果如图 5.6 所示。

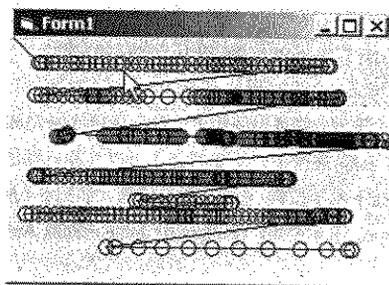


图 5.6 MouseMove 事件发生位置的演示

```

Private Sub Form_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As
Single)
    If Button = 1 Then
        Line -(X, Y)
        Circle(X, Y), 80
    End If
End Sub

```

如果将上面这一小段程序输入计算机中并运行，就会发现鼠标移动得越快，在任何两点之间能识别的 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的 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。

现在已经了解到，MouseDown 事件一次只能识别一个按键。与此形成对照的是：MouseMove 事件可以检测是否同时按下了两个以上的鼠标按键，MouseMove 还能检测是否按下某个特定的按键，而不管是否同时还有其他按键被按下。

如果在程序中要检测移动鼠标时是否只按住了左键，可以在 MouseMove 事件过程中查看 Button 参数是否等于 001(十进制为 1)。例如在下列代码中，如果还有其他按键与鼠标左键同时被按住，则不显示任何信息：

```
Private Sub Form_MouseMove( Button As Integer,Shift As Integer,X As Single,Y As Single)
    If Button = 1 Then Print "您按下了鼠标的左键"
End Sub
```

如果只是要检测是否按下某个特定的鼠标按键，应将 Button 参数和 And 操作符配合使用。例如，用户只要按下了鼠标的左键，下列代码就将显示提示信息，而不管是否还同时按下其他按键：

```
Private Sub Form_MouseMove(Button As Integer, Shift As Integer,X As Single,Y As Single)
    If Button And 1 Then Print "您按下了鼠标的左键"
End Sub
```

如果要同时检测多个按键，则可以使用 If...Then...Else 语句和 Select Case 语句判断左、右按键中的哪一个(或者全部)被按下。下面的例子可以检测鼠标的三种按键状态(按下左按键、按下右按键、同时按下两个按键)并显示相应信息(如图 5.7 所示)：

```
Private Sub Form_MouseMove(Button As Integer,Shift As Integer,X As Single,Y As Single)
    If Button = 1 Then
        Print "您按下了鼠标的左键"
    ElseIf Button = 2 Then
        Print "您按的是鼠标右键"
    ElseIf Button = 3 Then
        Print "您把两个按键都按下了"
    End If
End Sub
```

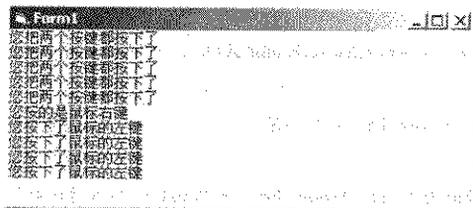


图 5.7 MouseMove 事件

5.2.3 MouseUp 事件

释放鼠标按键时，MouseUp 事件将会发生。MouseDown、MouseDown 和 MouseMove 事件搭配使用，往往相得益彰。

在如图 5.5 所示的涂鸦程序中，虽然可以使用鼠标到处乱画，也可以利用鼠标左键来控制是否画出鼠标移动的轨迹，但是这个程序致命的缺点就是画出的轨迹总是连续的，中间不能中断。如果能够做到在释放按钮时停止绘图，而在按住鼠标按键时从当前位置开始绘制轨迹，则这个程序将会更有用。在这里，就将利用到 MouseUp 事件。

MouseDown 与 MouseUP 将通知应用程序开始绘图与停止绘图，通过创建一个代表绘图状态的窗体级变量就可以达到这一目的。在窗体代码模块的声明部分输入如下代码：

```
Private blnDrawNow As Boolean
```

BlnDrawNow 变量代表的两种状态值：True 的意思是“绘制直线”；False 的意思是“停止绘制”。

因为变量的默认初始值为 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)
    blnDrawNow = True
    CurrentX = X
    CurrentY = Y
End Sub

Private Sub Form_MouseUp(Button As Integer,
    Shift As Integer, X As Single, Y As Single)
    BlnDrawNow = False
End Sub
```

而只有当 BlnDrawNow 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 btnDrawNow 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不断重复执行 MouseMove 事件过程。程序执行起来应该比前一个版本好用得多，如图 5.8 所示。

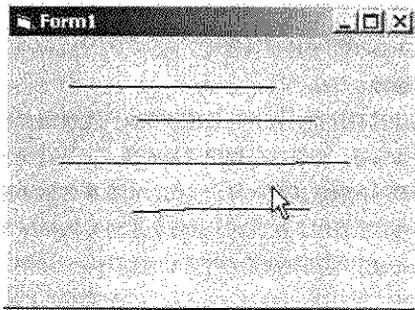


图 5.8 修改后的涂鸦程序

5.3 检测鼠标按钮

在编写代码时，往往要对鼠标事件作出不同的响应，同时还要考虑用户按下了哪个鼠标按钮或者是否按下了 Shift、Ctrl、Alt 键，这样才能使应用程序的功能更加完善。为提供这些选择，应使用 MouseDown、MouseDown 和 MouseMove 事件过程及 Button、Shift 参数。在以后的“KeyDown 和 KeyUp 事件”中，将会介绍如何利用 Shift 参数来判断 Shift、Ctrl 和 Alt 键的状态，这个参数对于鼠标事件和键盘事件都是一样的。

MouseDown、MouseDown 和 MouseMove 事件都使用 Button 参数判断按下的是哪个鼠标按钮或哪些组合按钮。Button 参数是一个位域参数，它的三个最低位分别表示鼠标的左按键、右按键和中按键。

每一位的默认值为 0(False)。如果未按下任何按钮，则三位的二进制值为 000。如果按下了鼠标的某个按钮，则相应的位也会由 0 变为 1。Button 参数用十进制数值或常数表示这些状态。表 5-2 列出了位的二进制值、相应的十进制值和 Visual Basic 常数。

表 5-2 Button 参数的不同数值以及相应的意义

二进制值	十进制值	常 数	意 义
001	1	vbLeftButton	按下左按键
010	2	vbRightButton	按下右按键
100	4	vbMiddleButton	按下中间按键

将十进制数值 4 分配给鼠标中间按键,同时按下鼠标的左、右按键就会产生单个数值 3(1+2)。对于三按键鼠标,同时按下三个按键将产生十进制数值 7(4+2+1)。表 5-3 中列出从可能的按键组合中导出的其他按键值。

表 5-3 Button 参数不同的取值组合

二进制值	十进制值	常 数	意 义
000	0		未按下任何按键
011	3	vbLeftButton+vbRightButton	按下左、右按键
101	5	vbLeftButton+vbMiddleButton	按下左、中按键
110	6	vbRightButton+vbMiddleButton	按下右、中按键
111	7	vbRightButton+vbMiddleButton+vbLeftButton	按下三个按键



5.4 检测 Shift、Ctrl 和 Alt 键状态

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 5.9 所示,Shift 中的三个最不明显位 S、C、A 对应 Shift、Ctrl 和 Alt 键的状态。根据 Shift、Ctrl 和 Alt 键的状态可在 Shift 中设置任一或设置所有位。表 5-4 列出这些值和常数。

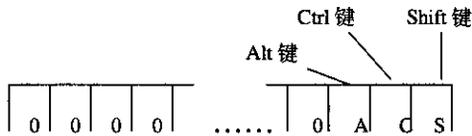


图 5.9 如何表示 Shift、Ctrl 和 Alt 键的状态

表 5-4 Shift、Ctrl 和 Alt 键状态值和常数

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

➡ 5.5 拖 放

在设计 Visual Basic 应用程序时，可能要经常在窗体中拖动控件。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。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。如表 5-5 所示。

表 5-5 拖放属性、事件和方法

类别	项 目	描 述
属性	DragMode DragIcon	启动自动拖动控件或手工拖动控件 指定拖动控件时显示的图标
事件	DragDrop DragOver	识别何时将控件拖动到对象上 识别何时在对对象上拖动控件
方法	Drag	启动或停止手工拖动

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，应将 TabStop 属性设置为 False。

5.5.1 自动拖动模式

为用户拖动控件，应将控件 DragMode 属性设置为 1-自动化。在将拖动设置为“自动化”后，拖动总是“打开”的。若要进一步控制拖动操作，应使用 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他鼠标事件。

5.5.2 拖动时的图标

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，就可用其他图像代替该轮廓。此属性包含对应图形图像的 Picture 对象。设置 DragIcon 属性的最简单的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，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("图片路径")
```

5.5.3 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。

在拖放操作时，有两个术语非常重要：源和目标。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。如表 5-6 所示。

表 5-6 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	其上是要放控件的对象。此对象可为窗体或控件，能识别 DragDrop 事件

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，所以可像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，它的 DragMode 属性已被设置为 1-Automatic，而其 DragIcon 属性被设置为另一个样本拖放图标文件。目标也是 Image 控件，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("图片路径")
End Sub
```

把 Image1 拖放到 Image2 后，Image1 就会消失，而 Image2 会将其图片改变。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 属性而导致运行时出现错误。可用 If...Then...Else 语句及 TypeOf 关键字判断放了哪种类型的控件。

5.5.4 对拖动的控制

对于 DragMode 属性，Visual Basic 的手动设置比自动设置提供更多的控制。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，就能拖动控件）。例如，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应使用 Drag 方法。如表 5-7 所示，用 Visual Basic 常数指定 Drag 参数的操作。

表 5-7 指定的 Drag 常数

常数	值	意义
vbCancel	0	取消拖动操作
vbBeginDrag	1	开始拖动操作
vbEndDrag	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("图片路径")
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，用户会看到相应的变化。

```
Private Sub Image2_DragOver(Source As Control,
    X As Single, Y As Single, State As Integer)
    Source.Drag vbEndDragSource.Visible = False
    Image2.Picture = LoadPicture("图片路径")
End Sub
```

将第三个 Image 控件添加到窗体中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

5.5.5 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，应对任何已被允许拖动的控件使用 Move 方法。在窗体中将控件拖放到任意不被其他控件占据的位置，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，在窗体中添加 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，因为它的功能是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按如下步骤执行：

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 1DragX = XDragY = Y
End Sub
```

Form_DragDrop 过程确实移动了控件。为简化此例，假设 Image1 是窗体中的惟一控件，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 和窗体的坐标系使用相同的单位。若不相同，则应进行单位转换。

5.6 OLE 拖放

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、在控件和其他 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，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中。例如，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 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，必须为拖放编程。

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，这样就可以将项目放在所希望的 ComboBox 中的位置。如表 5-8 所示。

表 5-8 OLE 拖放属性、事件和方法

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	指定控件如何响应拖放操作
事件	OLEDragDrop	识别源对象何时被拖放到控件上
	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拖放效果(复制、移动或拒绝数据)
	OLESetData	在拖放源对象时提供数据
	OLECompleteDrag	当把对象拖放到目标时通知被执行的操作的源
方法	OLEDrag	启动手工拖动

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：为判断其他 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。

5.6.1 自动 OLE 拖放

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。例如，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。当把一系列单元从 Excel 拖动到 Word 文档时，就已执行了自动拖放操作。

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个 OLE 拖放事件。为进一步控制拖放过程的每步操作、为用户自定义的视觉反馈、为创建自己的数据格式、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，手工实现是惟一的选择。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，如果该对象为源。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动作。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是在同一个控件中，这是很有用的。根据具体情况的不同，可能需要为源、目标或同时为两者编写代码。

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，将文本从文本框控件拖动到 Word for Windows 文档中，或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）文档中。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。

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，除非在拖动时同时按下 `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。表 5-9 列出了 `DataObject` 对象使用的属性及方法。

表 5-8 DataObject 对象的属性和方法

类别	项目	描述
属性	Files	保存出入于 Windows Explorer 的被拖动的文件名
	Clear	清除
方法	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 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。如果不使用缺省数据格式, 则用 Clear 方法。如果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, 这些文件能被拖放到拖放目标, 还可指定要传送数据的格式。如表 5-10 所示, 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。

表 5-10 SetData 和 GetData 使用的参数

参数	描述
Data	用来指定放到 DataObject 对象中的数据类型(如果设置了 format 参数, 此项是可选参数; 否则, 此项是必要参数)
Format	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(设置了 data 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他格式, 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。

表 5-11 Format 参数格式

常数	值	意义
vbCFText	1	文本
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，因此是传递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，使得位图与调色板正确地放入 DataObject 对象中。

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。如图 5-10 所示的是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。

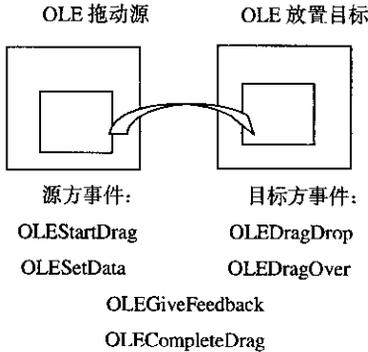


图 5.10 源端事件和目标端事件

例如，创建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他应用程序拖动来的数据。在这种情况下，直接将控件的 OLEDropMode 属性设置为“自动化”。如果希望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。

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。

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。

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）。

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。

5.6.2 拖放操作

• 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。

手工拖放操作的第一阶段是调用 `OLEDrag` 方法，设置允许的拖放效果，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` 方法手工启动拖放操作，并使用 `OLEStartDrag` 事件指定允许的拖放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。

• 指定拖放效果和数据格式

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。用此事件指定拖放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。

如表 5-12 所示，AllowedEffects 参数使用下面的数值指定拖放效果。

表 5-12 AllowedEffects 参数

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，用丰富的文本框控件作源，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。例如，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的文本格式。

• 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。如表 5-13 所示。

表 5-13 Format 应用

为了...	使用...
判断并响应源对象的位置	OLEDragOver 事件的 state 参数
响应鼠标按钮的状态	OLEDragDrop 和 OLEDragOver 事件的 Button 参数
响应 Shift、Ctrl 及 Alt 键的状态	OLEDragDrop 和 OLEDragOver 事件的 Shift 参数
修改缺省的鼠标视觉性能	OLEDragOver 事件的 effect 参数和 OLEGiveFeedback 事件的 Effect 参数

• 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，如表 5-14 所示。

表 5-14 OLEDragOver 事件常数

常 数	值	意 义
vbEnter	0	已将数据拖入目标的范围
vbLeave	1	已将数据拖出目标的范围
vbOver	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

那么如何向用户提供自定义的视觉反馈呢?

如果要改变 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, 此控件允许何种拖放效果。

• 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。

如表 5-15 所示, OLEDragOver 事件的 Effect 参数使用下列常数指示放下操作。

表 5-14 OLEDragOver 事件的 Effect 参数

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, 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，必须选择唯一的效果。`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` 参数来改变鼠标指针的缺省性能，需要用 `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` 参数像其他 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。如表 15-16 所示。

表 5-16 `OLEGiveFeedback` 事件的 `Effect` 参数

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>	&H80000000&	即将启动滚动或正在目标内滚动。除使用其他值外还使用次数值。

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。在大多数情况下无需指定自定义鼠标指针，因为 OLE 具有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，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(Effct As Long,
    DefaultCursors As Boolean)
    DefaultCursors = False
    If Effct = vbDropEffectNone Then
        Screen.MousePointer = vbNoDrop
    ElseIf Effct = vbDropEffectCopy Then
        Screen.MousePointer = vbCustom
        Screen.MouseIcon = LoadPicture("图片路径")
    ElseIf Effct = (vbDropEffectCopy Or
        vbDropEffectScroll) Then
        Screen.MousePointer = vbCustom
        Screen.MouseIcon = _LoadPicture("图片路径")
    ElseIf Effct = vbDropEffectMove Then
        Screen.MousePointer = vbCustom
        Screen.MouseIcon = LoadPicture("图片路径")
    ElseIf Effct = (vbDropEffectMove Or
        vbDropEffectScroll) Then
        Screen.MousePointer = vbCustom
        Screen.MouseIcon = LoadPicture("图片路径")
    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
```

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。

• 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
```

• 查询 `DataObject` 对象

有时需要向 `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, 以指示拖放操作。表 5-17 列出这些常数。

表 5-17 OLEDragDrop 事件的 Effect 参数

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	拖动源删除数据

下列示例设置 Effect 参数来指示拖放操作。

```

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他的 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
End Sub

```

```
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

· 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 lpszFormat$) As Integer
```

```
  Dim MyFormat As Integer
```

一经定义之后, 就可像使用其他 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。

注意: 用 GetData 方法获取自定义数据格式, 这可能会导致难以预料的结果。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。

5.6.3 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。

- ① 从 Windows 资源管理器中拖动文本文件到文本框控件。
- ② 在 Visual Basic 中启动新的工程。
- ③ 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。
- ④ 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

```

⑤ 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

```

⑥ 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

```

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。

5.7 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。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。表 5-18 描述了几种指针及其在应用程序中的可能作用。

表 5-18 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>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
```

· 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，在本质上像图标一样是位图。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。两者不同的是：.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

选定一个窗体或控件并将其 MousePointer 属性设置为 99-Custom。

将 .ico 文件加载到 MouseIcon 属性中。例如，对于窗体：

```
Form1.MouseIcon = LoadPicture("")
```

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。

5.8 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。

5.8.1 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。表 5-19 对这些事件都作了描述。

表 5-19 键盘事件

键盘事件	触发原因
KeyPress	按下对应某 ASCII 字符的键
KeyDown	按下键盘的任意键
KeyUp	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：

- ① Shift、Ctrl 和 Alt 键的特殊组合。
- ② 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。
- ③ PageUp 和 PageDown。
- ④ 区分数值小键盘的数字键与打字键盘的数值键。
- ⑤ 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)。
- ⑥ 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。

注意：Windows ANSI 字符集对应 256 个字符，包括标准的拉丁字母、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。这些字符由惟一的一字节数值 (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。

5.8.2 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 事件能够检测其他功能键、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。

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
```

5.8.3 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。如表 5-20 所示。

表 5-20 KeyDown 和 KeyUp 事件的参数

参 数	描 述
Keycode	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 keycode 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
Shift	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 键后都将显示消息框——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。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、PageUp、Delete 等等)、定位键 (Right、Left、Up 和 Down) 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
```

• 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（如图 5.11 所示）：

```
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
```

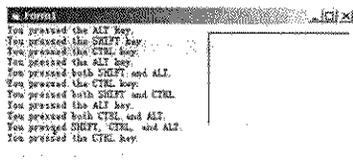


图 5.11 KeyDown 事件

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。

· 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 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了快捷键,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。

第六章 工程 管理

Visual Basic 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。本章首先介绍与工程有关的内容以及工程的结构；然后对如何创建、打开和保存工程作一总结；再介绍如何添加、删除控件，制作和运行可执行文件，设置工程选项以及如何使用向导和外接程序进行工作。

本章介绍的重点内容是：

- 工程中的文件
- 文件管理
- 向工程中添加文件的方法
- 删除工程中无用文件的方法
- 生成可执行文件的方法
- 向导的使用方法
- 外接程序的使用方法



6.1 工程文件

Visual Basic 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，如表 6-1 所示，构成工程的文件有如下几种主要类型。

表 6-1 工程中的文件

文件后缀	文件类型	文件后缀	文件类型
.Visual Basicp	跟踪所有的部件的工程文件	.ocx	包含 ActiveX 控件的二进制文件
.frm	每个窗体的窗体文件	.dll	是动态链接库的一种
.frx	窗体的二进制数据文件	.exe	可执行文件
.cls	类模块文件，用于类定义	.hlp	帮助文件
.bas	标准模块文件	.res	资源定义文件

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他工程共享。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，还可以创建其他类型的可执行文件，例

如.ocx 和.dll 文件。

• 工程资源管理器

当创建、添加或从一个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。

• Visual Basic 工程的结构

工程文件，是扩展名为.Visual Basicp 的文件，用来记录文件、对象、工程选项、环境选项、EXE 选项以及与工程相关联的引用。每个工程只有一个工程文件。Visual Basic 在每次保存工程时，都要更新工程文件。工程文件包含的文件列表就是出现在工程资源管理器窗口的文件列表。用户可以通过双击一个现存的工程文件图标，或从“文件”下拉菜单中选定“打开工程”命令，来打开一个现存的工程文件。如图 6.1 所示。

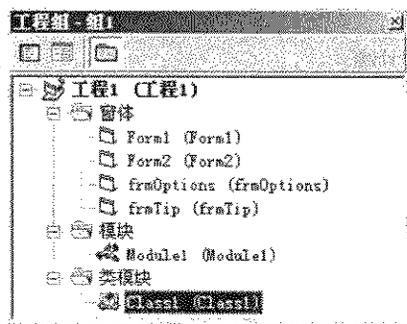


图 6.1 工程资源管理器

窗体模块，是扩展名为.frm 的文件，包含窗体及其控件的正文描述以及它们的属性设置，也包含具有窗体模块级的常量、变量等及事件过程、通用过程。工程中可以有多个窗体文件，对于没有用户界面的程序允许没有窗体文件。

类模块，是扩展名为.cls 的文件，功能与窗体模块类似，只是没有可见的用户界面。用户可以使用类模块创建含有方法和属性代码的对象。一个工程中可以有多个类模块，也可以没有类模块。

标准模块，是扩展名为.bas 的文件，可以包含类型、常量、变量、外部过程和通用过程的模块级说明。一个工程中可以有多个标准模块，也可以没有标准模块。

ActiveX 文档，是扩展名为.dob 的文件，它类似于窗体，与窗体不同的是 ActiveX 文档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。一个工程中可以有多个 ActiveX 文档，也可以没有 ActiveX 文档。

部件，除文件和模块以外，还有几个其他类型的部件可以添加到工程中。

1. “ActiveX 控件”，具有文件扩展名.ocx，是工程中可选的控件，它可以被添加到工具箱中并在窗体中使用。

2. “可插入对象”，一般运行在其他的应用程序中。通过部件对象模型接口与用户程序通信，完成指定工作。

3. “引用”，可以添加能被应用程序使用的外部 Active X 部件的引用。

4. “标准控件”，是位于工具箱中且包含于 Visual Basic.exe 文件中的控件。

资源文件，是扩展名为.res 的文件，包含着无需重新编辑代码便可以改变的位图、字符串和其他数据。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。

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。

• 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。从其他的源可取得附加的设计器。

6.2 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。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。工程管理见表 6-2。

表 6-2 工程管理

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。Visual Basic 打开一个现有工程，包括其工程文件(Visual Basic)中所列的窗体、模块和 ActiveX 控件
“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 提示用户保存所有修改过的窗体或模块

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。

• 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他部件的解决方案时，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”中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”。

6.3 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。如果想改变文件而不影响其他工程，应在“工程资源管理器”中选定该文件，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。

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. 此文件将从工程中被删除，但是仍存在于磁盘上。

如果从工程中删除了文件，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”中选定此文件。
2. 从“文件”菜单，选取“保存 filename”。

合并文本也能够从其他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



图 6.2 添加“filetype”

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. 在“工程”窗口中，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. “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：如果要使用 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 VISUAL_BASIC_PredeclaredId 的设置。如果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。

6.4 添加模块和控件

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，必须先位于工具箱中。在工程中添加 ActiveX 控件，只要将 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，请按照以下步骤执行：

1. 在“工程”菜单中选取“部件”。“部件”对话框如图 6.3 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. 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. 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，搜索其他目录寻找具有 .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。该文件中存储有高速缓存型库信息和该控件特有的其他数据。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)。

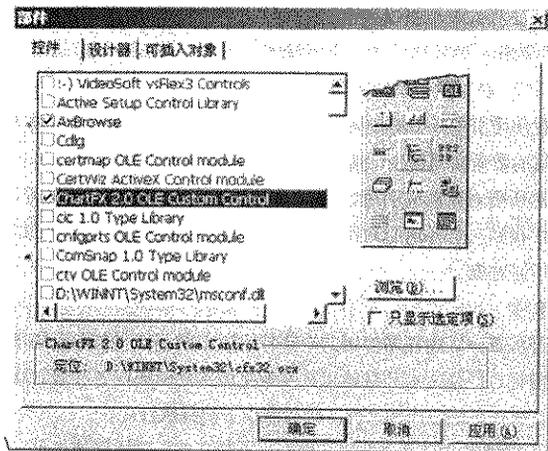


图 6.3 部件对话框

要从工程中删除控件，请按照以下步骤执行：

1. 从“工程”菜单中，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。

• 使用其他应用程序的对象

也可以使用来自其他应用程序的对象，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”。要使其他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。

要添加对其他应用程序的对象库的引用，请按照以下步骤执行：

1. 从“工程”菜单中，选取“引用”。“引用”对话框如图 6.4 所示。



图 6.4 “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，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”菜单中选取“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。

· 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、屏幕文本和其他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”菜单中，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。

6.5 制作、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”对话框中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，则从清单中选取一个。

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）。

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，像运行任何其他基于 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：Visual Basic6 /make projectname[.Visual Basicp] [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。

6.6 工程选项设置

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（见图 6.5），通过“工程”菜单中的“工程属性”命令可以设置。属性设置被保存在工程文件(.Visual Basicp) 中。

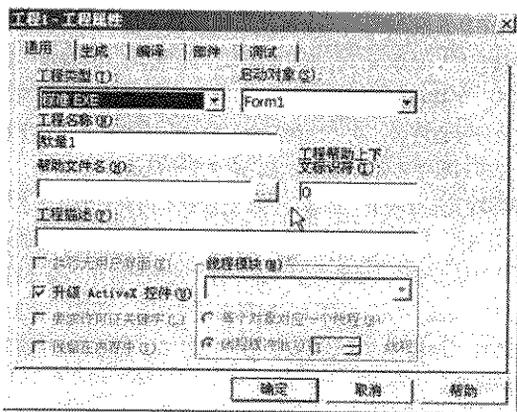


图 6.5 工程属性对话框

还有许多其他选项也是可使用的，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。

表 6-3 列出能够设置的某些选项。

表 6-3 工程属性设置

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”对话框里



6.7 使用向导和外接程序

在 Visual Basic 6.0 语言中, 允许用户在编写应用程序时选择和管理一些外接程序, 这是 Visual Basic 的扩充。通过使用这项扩充功能, 使 Visual Basic 6.0 的集成开发环境的能力得到了增强, 如源代码控制能力等。

Microsoft 和其他的一些软件开发商创建了许多可以用于 Visual Basic 的应用程序中的外接程序。向导就是一种外接程序, 它可以简化某些任务, 例如创建窗体等任务。

在开发程序时, 要使用外接程序, 必须先保证它们已经被正确地安装在机器的硬盘上, 并且在 Visual Basic 的“外接程序管理器”列表中显示出来。通过使用“外接程序管理器”来完成对工程进行添加或删除外接程序的操作。具体操作如下:

1. 单击“外接程序”菜单项, 打开下拉菜单, 从中选择“外接程序管理器”命令。
2. 屏幕中出现“外接程序管理器”对话框, 如图 6.6 所示。在“可用加载宏”的列表框中列出可以用的外接程序名。在“加载行为”的列表框中显示各外接程序的安装情况。单击可用的外接程序列表中要安装的程序名, 使它突出显示。
3. 在对话框的右下角有一个“加载行为”选项框, 内有三个复选选项, 如果选定“加载/卸载”选项的复选框, 则在“加载行为”列表框中显示“加载”, 如果选定“启动时加载”选项则显示“启动/加载”, 如果选定“命令行”选项则显示“命令行”, 如果三项全部选定则显示“全部”, 如果三项都未被选定则什么也不显示, 还可以继续组合下去。在“加载行为”列表中添加“加载”的项是要安装的外接程序, 在列表框中清除所显示的内容的项是要删除的外接程序。
4. 完成选定操作后, 单击对话框的“确定”按钮, Visual Basic 将连接被选定的外接程序, 断开与被清除的外接程序的连接, 同时将选定的外接程序作为菜单命令添加到“外接程序”菜单项中, 并从其下拉菜单中删除已被清除的外接程序。

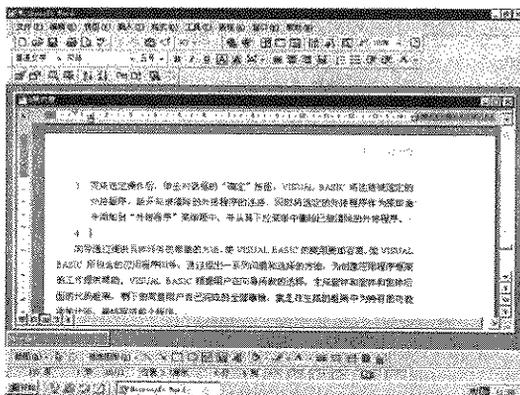


图 6.6 外接程序管理器

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。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。Visual Basic 根据用户在向导所作的选择, 生成窗体和其后面的代码框架, 剩下的需要用户自己完成的全部事情, 就是在生成的框架中为特有的功能添加代码, 最终完成整个程序。

用户可以使用外接程序管理器安装或删除向导。安装之后, 它们将作为“外接程序”菜单项上的菜单命令出现。此外, 某些向导也能够以图标的形式出现在有关的对话框中。如在“新建工程”对话框中的“应用程序向导”。

现在以使用“应用程序向导”为例, 介绍如何使用 Visual Basic 的向导。

1. 在“文件”下拉菜单中选择“新建工程”, 打开“新建工程”对话框。如图 6.7 所示。

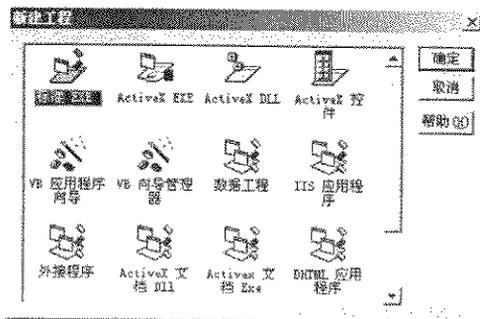


图 6.7 新建工程对话框

2. 选定“应用程序向导(VISUAL BASIC Application Wizard)”，进入它的第一步操作。如图 6.8 所示。

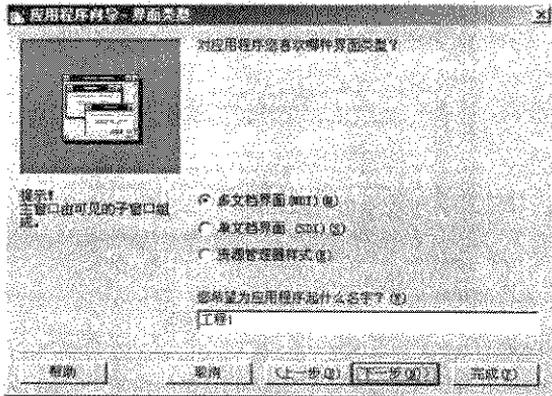


图 6.8 应用程序向导

3. 向导主要完成以下几方面设置：定义界面类型；设置菜单；设置各种资源；设置访问因特网；设置标准窗体；设置数据访问窗体；对新的应用程序进行存储。

第七章 界面设计

7.1 窗体

7.1.1 Visual Basic 的窗体

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他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。

7.1.2 在窗体上添加控件

一个光秃秃的窗体是完成不了什么工作的，只有在窗体上添加不同的控件，并且使用代码加以控制，才能创建出多姿多彩的应用程序来。向窗体上添加控件要用到控件工具箱和窗体编辑器。

· 使用控件工具箱

工具箱的名字起得很好，因为在编制 Visual Basic 程序时，使用控件的感觉和日常生活中从工具箱中拿锤子、钳子或者扳手差不多，都是在需要的时候取出来。只不过在 Visual Basic 中的“工具”不是用手去拿，而要使用鼠标点取。

在初次进入 Visual Basic 的集成开发环境中时，就可以看到默认的控件工具箱。同其他窗口一样，工具箱可以是连接状态，也可是浮动状态。

在工具箱中，除了“指针”按钮是用来选择控件的工具外，其余都是控件按钮。初次接触 Visual Basic 的用户看到这么多的控件，可能会觉得无所适从，但是随着对 Visual Basic 的不断深入了解和使用，以后只会觉得控件太少，而不够用。

添加控件要在窗体编辑器窗口中进行，程序员可以单击“工程资源管理器窗口”中的按钮，或者选择“视图/对象窗口”命令，还可以使用组合键 Shift+F7 来进入窗体编辑窗口。

使用工具箱向窗体上添加控件有两个途径：

① 在工具箱中的控件按钮上双击，则窗体的中央就会出现一个相应的控件。需要注意的是：如果多次双击同一个控件按钮，则窗体的中央就会重叠多个控件，程序员如果不注意就会认为只有一个控件。所以，在添加了一个控件后，要随时将它移动到其他位置上去。

② 在工具箱中的某个控件按钮上单击，则该按钮会凹下去，而且鼠标指针会变为“+”形状，程序员可以在窗体中认为合适的位置按下鼠标左键，然后拖动鼠标来设置控件的大小，当松开鼠标按钮后，控件就会出现在窗体上。

要关闭工具箱，只要单击工具条左上角的按钮即可。在关闭工具箱之后，如果要重新打开它，可以单击工具栏上的“工具箱”按钮，或者选择“视图/工具箱”命令。

· 使用窗体编辑器

当程序员把控件都添加到窗体上以后，对各个控件的布局工作就会都在窗体编辑器中进行。

· 调整窗体的大小

窗体编辑器的主体是一个空空的窗体，它的大小是默认的，所以每一个新窗体的大小总是不差毫厘。如果用户觉得窗体太小，无法容纳太多的控件，可以调整窗体的大小。首先，要在窗体中编辑，首先单击来选中窗体，窗体周围会出现 8 个小方块(控制柄)，有的是空心的，有的是实心的(如图 7.1 所示)。将鼠标指针移动到实心控制柄上，指针会变为类似的双箭头形状，这时按住鼠标左键并拖动，就可以改变窗体的大小。

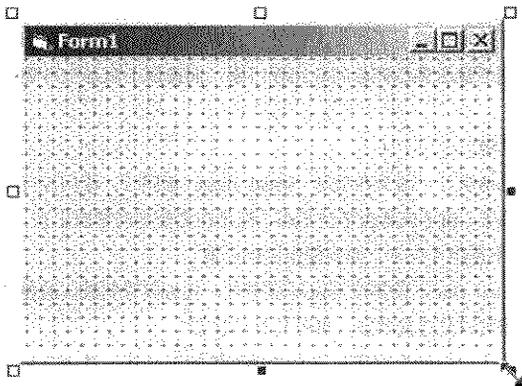


图 7.1 调整窗体的大小

如果要精确设置窗体的大小，可以到属性窗口中设置窗体对象的 Width 和 Height 属性的值。需要注意的是：这两个属性使用的计数单位都是“缇”，属性值往往都很大，一般都在几千左右。

“缇”是一种和屏幕分辨率无关的计量单位，不管在什么屏幕上，如果画一条 1440 缇的直线，它的长度打印出来都是 1 英寸(如果是 567 缇的话，就是 1cm)，因此采用缇作单位，就能保证在不同的屏幕上还能保持正确的相对位置和比例关系。

如果想使用其他的单位，只要重新设置 ScaleMode 属性即可，例如设置为 2，就可以使用“磅”作单位(每逻辑英寸为 72 磅)。

• 控件的选择

在窗体编辑器中，可以对窗体上的控件进行各种操作。但是在进行操作之前，都要首先选中控件。

选择一个控件的操作可能已经成为程序员的下意识动作——只要在这个控件上单击就可以了。被选中的控件周围会出现 8 个“控制柄”，而且在“标准”工具栏的左侧会显示这个控件的位置和大小，如图 7.2 所示。

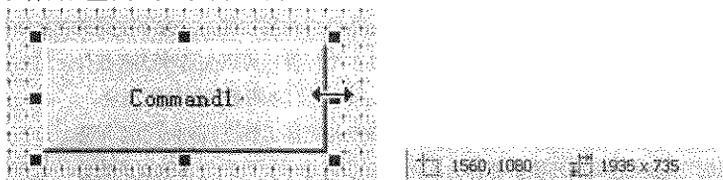


图 7.2 控件的选择

如果要同时选择多个控件，可以有两种方法：

① 在一个控件上单击，选中该控件，然后按住 Ctrl 或者 Shift 键，再到其他要选择的控件上单击，就可以同时选中多个控件。这种方法适用于选择多个不相邻的控件。

② 如果要选择一个区域内的所有控件，可以单击工具箱中的“指针”按钮，然后拖动鼠标在窗体上画一个框，包围所有要选择的控件，松开鼠标按钮后，就会选中框中所有的控件。

此时如果要撤销对控件的选择，可以在窗体上单击，这时选中状态就会转移到窗体上，如果要撤销对多个选中控件中某一个控件的选择，需要按住 Ctrl 或者 Shift 键，然后单击该控件就可以撤销对该控件的选择。

• 调整控件的大小

要调整控件的大小，应当首先选中这个控件。然后将鼠标指针移动到控制柄上，指针会变为双箭头形状，这时按住鼠标左键并拖动，可以改变控件的大小。

有的控件的大小不能改变，例如 Timer 控件。

同样，要设置控件的精确尺寸，应设定该控件的 Height 和 Width 属性(如果有的话)。

如果要设置多个控件的大小，应先选中这些控件，然后使用“格式”菜单下“按相同大小制作”子菜单下的命令。

在所选择的控件中，总有一个控件是被实心的控制柄包围着，通常这个控件是用户选

择的最后一个控件，为了与其他被空心控制柄包围的控件相区别，权且将它称为主控件，Visual Basic 将按照主控件的相应尺寸(例如它的宽或高)来设置其他控件的大小。掌握了这一原则，用户可以改变主控件来设置控件大小的基准。要改变主控件，只须在选中的控件中单击某个控件，就会看到该控件被实心的控制柄包围，成为主控件。

• 移动或删除控件

在设置窗体的布局时，需要经常将控件移来移去，遇到不合心意的时候，还要将控件删掉重来。

删除控件是名副其实的举手之劳，只要选中删除的控件(一个或者多个)，然后按 Del 键即可。

窗体的删除与删除控件不同，是不能使用 Del。要删除一个窗体，应首先在“工程资源管理器窗口”中选中该窗体模块，然后选择“工程”菜单中的“删除窗体名”命令。

移动控件也是很简单的操作，只要将鼠标指针移动到控件上，按下左键并移动鼠标就可以将控件拖动到别的位置。如果要同时移动多个控件，可以先选中这些控件，然后拖动其中的一个，就会使选中的控件都发生移动，并且它们之间相对的位置不变。

细心的用户也许很早就发现，控件的边缘是与窗体上规则排列的小点对齐。这些小点在 Visual Basic 的集成环境中被称为网格，有了网格用户可以轻易地对齐控件并且控制控件的大小。根据自己实际工作的需要，用户可以设置有关网络的选项，方法是选择“工具”菜单中的“选项”命令，会显示“选项”对话框，然后选择对话框中的“通用”选项卡，如图 7.3 所示。

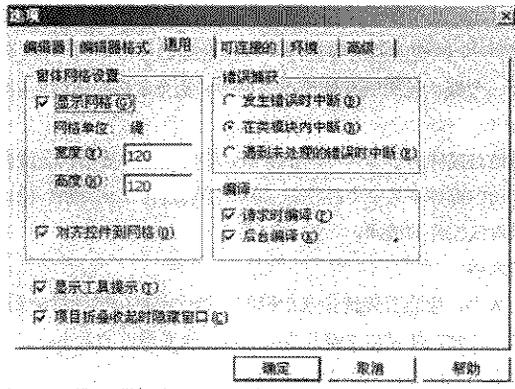


图 7.3 设置网络的有关选项

在这张选项卡中，用户可以自行设置网络的间距，还可以决定控件是否对齐到网格。

如果要对齐多个控件，可以同时选取这些控件，然后选择“格式”菜单中的“对齐”子菜单中的命令来进行不同方式的对齐，如图 7.4 所示。对齐的基准是选中控件中的主控

件(被实心控制柄包围的控件)。

此外,“格式”菜单中的“在窗体中居中对齐”命令也可以使所选的控件相对于整个窗体进行水平或者垂直居中。

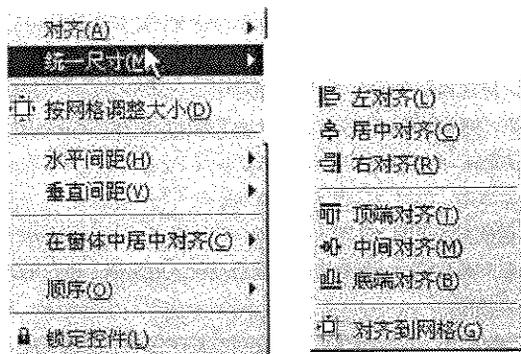


图 7.4 使用“对齐”命令迅速对齐选择的控件

当用户对控件在窗体上的位置觉得满意之后,应使用“格式”\“锁定控件”命令锁定当前窗体上的所有控件,这样当它们处于指定位置时就不会由于疏忽而被移动。因为“锁定控件”命令基于窗体面起作用,它仅锁定选定窗体上的控件,而不影响其他窗体上的控件。

• 在程序中引用其他窗体上的控件

在开发比较早的应用程序时,经常会用到多个窗体,那么怎样在程序中引用其他窗体中的控件呢?

如果引用其他窗体上的控件,只要在控件的名称前加上窗体的名称,中间利用“.”或者“!”分隔。例如,在下面的语句中,可以控制窗体 Form2 中的文本框 txtName 显示的内容。

```
Form2.txtName.Text = "Hello!"
```

• 使用窗体布局窗体设置窗体的位置

虽然设置窗体的 Top 和 Left 属性可以确定窗体在屏幕上出现的初始位置,但是使用窗体布局窗口可以进行更加直观地调整。例如,如图 7.5 所示的布局窗口中,有两个窗体。程序员可以用鼠标将 Form2 拖动到窗口中虚拟屏幕的中央,这样在程序运行的时候 Form2 就会出现在真实屏幕对应的位置上。

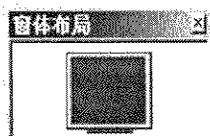


图 7.5 窗体布局窗口

7.1.3 启动设置

除了窗体的设置细节以外，程序员还需要考虑应用程序的开始与结束。每个应用程序都有自己的入口，即开始执行的地位，在这里可以使用一些技巧用于设计应用程序启动时的外观。

• 设置启动窗体

在默认情况下，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。如果想在应用程序启动时显示其他的窗体，那么就改变启动窗体。

如果要改变启动窗体，请按照以下步骤执行：

1. 从“工程”菜单中，选取“工程属性”命令。
2. 在显示的对话框中，选取“通用”选项卡。
3. 在“启动对象”列表框中，选取要作为新启动窗体的窗体，如图 7.6 所示。
4. 单击“确定”按钮。

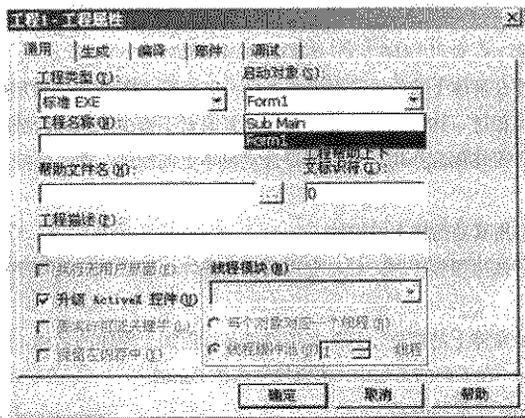


图 7.6 启动对象

• 不使用启动窗体开始程序的运行

有时候也许要应用程序启动时不加载任何窗体。例如，程序员想先装入数据文件，然后再加载窗体，或者先显示一个要求输入口令的对话框确认用户的身份。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
    IntStatus = GetUserStatus
    If intStatus = 1 Then
        FrmMain.Show
    End If
End Sub
```

```

Else
    FrmPassword.Show
End if
End Sub

```

Main 过程必须是一个位于标准模块中的子过程，不能在窗体模块内。要将 Sub Main 过程设为启动对象，可从“工程”菜单中，选取“工程属性”，再选择“通用”选项卡，最后从“启动对象”列表框中选定“Sub Main”（如图 7.6 所示）。

· 显示启动时的快速显示窗体

如果启动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，这时可以显示一个快速显示窗体，以吸引用户的注意，造成应用程序装载很快的错觉。

快速显示窗体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窗体。

要显示快速显示窗体，需用 Sub Main 过程作为启动对象，并用 Show 方法显示该窗体；VB5 提供了一些常用窗体的模板，快速显示窗体也在其中之列。要想在工程中添加快速显示窗体，只要选择“工程”菜单中的“添加窗体”命令，显示“添中窗体”对话框，在对话框中的“新建”选项卡中，选择其中的“Splash 屏幕”模板，然后单击“打开”按钮，如图 7.7 所示。

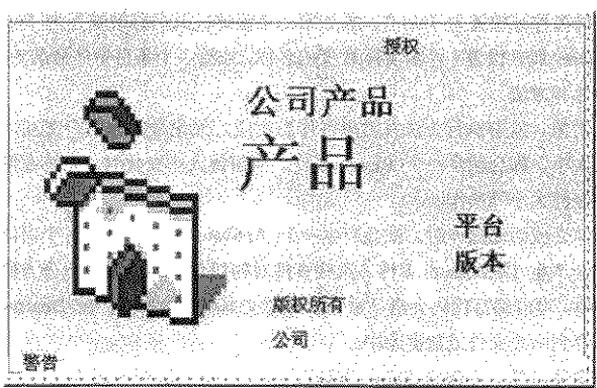


图 7.7 添加 Splash 屏幕

使用“Splash 屏幕”模板创建的快速显示窗体如图 7.7 所示。程序员可以在它的基础上进行改进，添加自己公司的名字、产品的名字、版本号以及版权信息等，还可以更改窗体上的图案。方法是在窗体的图案上单击，然后到属性窗口中更改它的 Picture 属性。

对于快速显示窗体的设置来说，应尽量简单。如果使用大量位图或者大量控件，则快

速显示的装入将会变慢。

7.1.4 窗体的使用

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。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，都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。

· 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，依此类推。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”。

· 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。当窗体大小变化时，允许在窗体上进行移动控件或调整控件大小等操作。

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。

7.1.5 窗体设计的基本原则



虽然利用 Visual Basic 创建用户界面非常容易，只简单地将控件拖动并放置到窗体上。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的改观。因为任何应用程序的可用性基本上由用户决定，而界面对用户第一感觉的影响是不可低估的。无论代码技术上多么卓越，或者优化得多么好，如果用户认为应用程序很难使用，那么就很难接受它。

值得一提的是，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，所以界面设计是须多次反复的过程，需要程序员有耐心。

· 什么是好的界面

在开始设计用户界面之前，最好先去看看 Microsoft 或其他公司一些销路不错的应用程序，从中借鉴一些东西。毕竟界面很差的应用程序不会卖得很好。从这些应用程序中，将会发现许多通用的东西，比如：工具栏、状态栏、工具提示、上下文菜单以及选项卡对话框。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。

程序员也可以凭自己使用软件的经验，想一想曾经使用过的一些应用程序，哪些好用、哪些不好用、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。

大多数成功的应用程序都提供选择来适应不同用户的偏爱。例如，Windows 98 的“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。提供选择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。

· 注意窗体的布局

窗体的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。布局包括诸如控件的位置、元素的一致性、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，以确保护越重要的元素越要尽快地显现给用户。重要的或者频繁访问的元素应当放在显著的位置上，而不太重要的元素应当降级到不太显著的位置上。

在大多数语言中一般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分，所以最重要的元素应当放在屏幕的左上部分。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很重要，应尽量把信息按功能或关系进行逻辑地分组。在许多情况下，可以使用 Frame 控件来帮助加强控件之间的联系。

· 界面元素的一致性

在用户面设计中，一致性是必须的。一致的外观与感觉可以在应用程序中创造一种和

谐。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，甚至可能引起用户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。

在 Visual Basic 中有大量的控件可供使用，这可能引起用户想使用所有的控件。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 ListBox、ComboBox、Grid 以及 Tree 等控件都可用来表示信息列表，最好还是尽可能地使用一种类型。

• 空白空间的使用

专业人员在设计名片时，非常讲究的一件事就是“留白”——也就是在名片上留出一些空白。如果一张小小的名片上有太多的内容、太多的头衔，只会导致别人一个也记不住。设计窗体也是这样，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。当然，空白空间不必一定是白色的——它指的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，使得寻找一个字或者控件非常困难。在设计中插入适当的空白可以突出设计元素。

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界面更可用。就像杂志中的文章那样，安排得行列整齐、行距一致，整齐的界面会使其容易阅读。

Visual Basic 在“格式”菜单中提出了几个命令，使得调整控件的间距、排列和尺寸非常容易，这些命令是“排列”、“按相同大小制作”、“水平间距”、“垂直间距”和“在窗体中央”。

• 保持界面的简单明了

界面设计最重要的原则就是简单化。对于应用程序而言，如果用户觉得界面很复杂，则可能会认为程序本身也很难，因此望而却步。所以与大而全的界面方案比较起来，整洁、简单明了的设计往往更可取。

在界面设计中，一个普通易犯的错误就是力图用界面来模仿真实世界的对象。例如，要求创建完整的人事管理的应用程序，很自然的反应就是在屏幕上设计一个完全依照个人履历表的界面。这样做会出现几个问题：履历表的形状与尺寸和屏幕上的有很大不同，要完全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。

最好是设计出一个相对简单的界面。程序员通过对原始履历表的分析，可以将有联系的字段为几个逻辑组，并使用选项卡对话框来装载这些输入字段，这样用户不滚动屏幕就可以查看所有的信息。程序员还可以使用附加的控件(例如列表框)或者默认值，以减少用户输入的工作量。

检验程序界面是否简洁，实现的最好方法就是观察用户对程序的使用。如果有代表性的用户没有提示就不能完成想要完成的任务，那么就需要重新考虑窗体的设计了。

• 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。随着硬件设备的更新，许多显示器已经能够显示几百万种颜色，这很容易使人产生全部使用它们的想法。

如果在开始设计时没有仔细地考虑，颜色的使用也会出现许多问题。

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。一般来说，最好保守传统，采用一些柔和、更中性化的颜色。

当然，预期的读者身份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可能会分散财务人员的注意力。

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。但是作为经验之谈，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，最好坚持标准的 16 种颜色。

· 图片和图标

在窗体中适当地使用图片与图标可以增加应用程序的视觉上的趣味，但是精心的设计同样是必不可少的。图片可以比文本更加形象地传达信息，但需要注意的是，不同的人对同一张图片的理解未必都是一样的。

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，应查看其他的应用程序以了解已经创建了什么样的标准。例如，许多应用程序用图标表示“打开文件”。也许程序员认为还有更好的比喻来表示这一功能，但改用其他的表示方法会引起用户的混淆。

· 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。程序员要选取在不同的分辨率和不同类型的显示器上都容易阅读的字体。最好能坚持使用标准的 Windows 字体，如 Arial、NewTimesRoman、宋体或者黑体。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。

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的字体。

· 不要背离 Windows 的界面准则

Windows 操作系统的主要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。如果说将“文件”菜单命名为“文档”菜单更好，或者将“帮助”菜单放在最左侧，这就值得讨论。没有任何事情阻止这样做，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。

同样，菜单命令的位置也很重要。用户也许会期望在“编辑”菜单下找到“复制”、“剪切”与“粘贴”等命令，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。



7.2 对话框

7.2.1 消息框

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对话框。以下代码显示如图 7.8 所示的消息框：

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

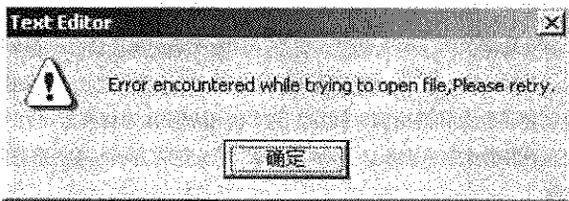


图 7.8 用 MsgBox 函数创建的错误消息对话框

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，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。

7.2.2 输入框

应用 InputBox 函数来请求提供数据。这个函数显示要求输入数据的模式对话框。如图 7.9 所示的文本输入框提示输入要打开的文件的名称。以下代码显示图 7.9 中所示的输入框。

```
FileName = InputBox(" Enter file to open:", "File Open")
```

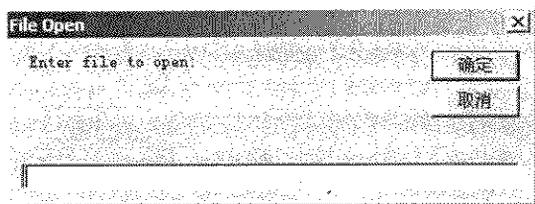


图 7.9 使用 InputBox 函数的对话框

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。

7.3 菜 单

7.3.1 创建菜单

菜单似乎是改善用户界面的一个主要手段，也是最早的手段之一。在 DSO 程序设计的年代，有了菜单选择，才使用户从冗长的命令行输入中摆脱出来。如果应用程序要为用户提供一组命令，菜单是一种给命令分组的很简单的方法，而且用户可以很容易地访问这些命令。

· 使用菜单编辑器设计菜单栏

菜单栏一般出现在窗体的标题栏下面，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(以下简称菜单项)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。

在 Visual Basic 中，有一个很好的工具可帮助用户创建自己的菜单系统，这个工具就是 Visual Basic 的菜单编辑器。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，程序员可以在只用很少编程的全交互方式中自定义菜单。

要使应用程序简单、好用，在使用菜单编辑器之前，应该将菜单项按其功能分组，以便于用户查找。例如，与文件有关的命令“新建”、“打开”和“另存为”等都列入“文件”菜单。

要显示菜单编辑器，可以从“工具”菜单上，选取“菜单编辑器”命令，还可以单击“标准”工具栏中的“菜单编辑器”按钮，如图 7.10 所示的就是一个菜单编辑器，其中的列表框显示了已经创建的菜单系统。用菜单编辑器可以创建新的菜单和菜单栏，还可以对已有的菜单系统进行增加、替换、修改和删除等操作。

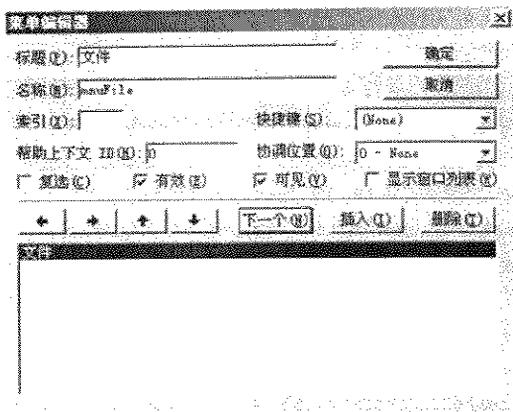


图 7.10 菜单编辑器

在 Visual Basic 中，菜单和菜单下的菜单项也都是控件，它们使用起来与命令按钮这样的控件没有什么区别。程序员可以在设计或运行时设置菜单控件的 Caption 属性、Enabled 和 Visible 属性、Checked 属性以及其他属性。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。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是一一对应的。

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。

在菜单编辑器中创建菜单控件，请按以下步骤执行：

1. 选取要添加菜单栏的窗体。
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，显示菜单编辑器。
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，即菜单控件的 Caption 属性。菜单标题文本会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，即菜单控件的“名称”属性。
5. 单击向左或向右的箭头按钮，可以改变该控件的缩进级。其中，单击向右按钮可以增加一级缩进，单击向左按钮可以删除一级缩进。在菜单编辑器中，每一缩进级都使用 4 个点“.”来表示。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；位于列表框中左侧齐平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。

6. 如果需要，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完成，也可以在以后的“属性”窗口中完成。

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，改变它们之间的相对位置。

8. 如果窗体所有的菜单控件都已创建，选取“确定”按钮可关闭菜单编辑器。

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。菜单编辑器下部的列表框列出了当前窗体的所有菜单控件，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。

如果某个菜单命令将激活一个对话框，则在其名称后面应当有一个省略号“…”。这种表示方法已经变为一种惯例，在设计自己的菜单时，不要随意修改。

· 分隔菜单项

一个以连字符“-”作为 Caption 属性的菜单控件，将作为菜单的分隔出现。分隔线可将菜单项划分成若干个逻辑组，如图 7.11 所示。

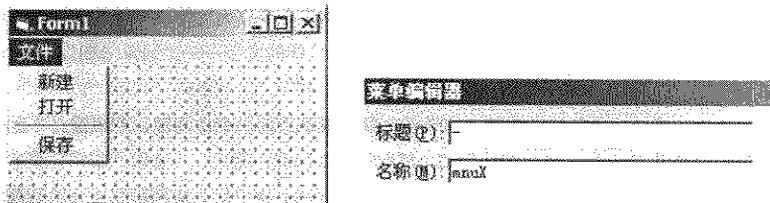


图 7.11 分隔符条

在菜单编辑器中创建分隔线，请按以下步骤执行：

1. 如果想在现有的菜单中增加一个分隔线，应选取“插入”按钮，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”按钮，关闭菜单编辑器。

虽然分隔线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。

· 定义菜单的访问键和快捷键

通过定义访问键和快捷键可改进键盘对菜单命令的访问。如果某一字符是该菜单项的访问键，在菜单中，这一字符下方会有一条下划线(如图 7.12 所示)，用户只要同时按下 Alt 键和该字符键，就可以激活该菜单。一旦菜单打开，通过按下所赋值的字符(访问键)可选取控件。

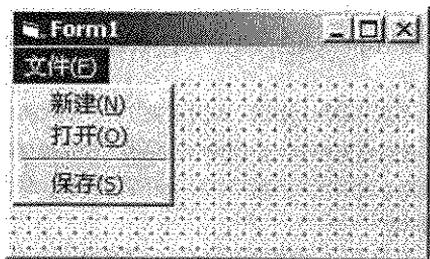


图 7.12 菜单的访问键

例如，Windows 的老用户一定会记得按下 Alt+E 键可打开“编辑”菜单，再按 P 键可执行“粘贴”命令。

如果要在菜单编辑器中给菜单控件赋值访问键，只须在菜单编辑器下方的列表框中，选取要赋值访问键的菜单项，然后在其“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，可以在“标题”框中键入这样的文字——“编辑(&E)”，就可以将 E 键作为“编辑”菜单的访问键。

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 键同时是“剪切”和“复制”的访问键，那么，当选取“编辑”菜单且键入 C 键时，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。

使用访问键虽然可以很方便地利用键盘来选取菜单命令，但是在敲击键盘多次选中某个菜单命令后，还需要再敲一下 Enter 键，菜单命令才会执行。

如果使用快捷键就可以大大提高选取命令的速度，快捷键按下时会立刻运行一个菜单项(所以才会有“快捷键”之名)。程序员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，例如大家都很熟悉的复制、粘贴、剪切系列快捷键——Ctrl+C、Ctrl+V 和 Ctrl+X。它们出现在菜单中相应菜单项的右边，如图 7.13 所示。

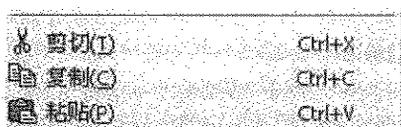


图 7.13 菜单项的快捷键

如果在对菜单项指定快捷键，首先打开“菜单编辑器”，然后从列表中选择该菜单项，再从“快捷键”组合框中选取功能键或者键的组合，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”框中键入诸如“Ctrl+C”这样的字眼。如果要删除快捷键值，应选取列表顶部的“(none)”选项。

需要注意的是，设置快捷键的关键在于精不在于多，如果指定了太多的快捷键，反而会加重用户的负担，快捷键也不会“快捷”起来。记住，最好要用在刀刃上。

· 改变菜单选项的状态

有些菜单命令的执行是需要一定条件的，例如“粘贴”命令，如果在剪贴板上没有任何内容，那么“粘贴”命令就不能执行，这时可以将菜单中的“粘贴”命令设置为失效，方法是设置控件的 Enabled 属性为 False。

这样，该菜单选项将变为灰色，不能接受用户的 Click 事件，如图 7.14 所示。

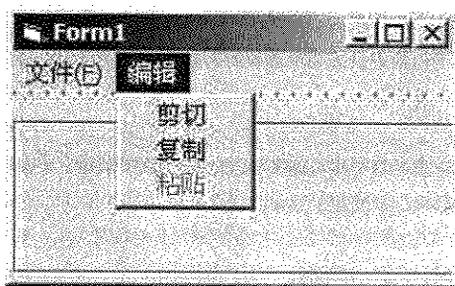


图 7.14 失效的“粘贴”菜单

如果要使菜单选项变为有效，只须设置 Enabled 属性为 True 即可。

在菜单编辑器中，只要消除对“有效”复选框的选择，也可以将当前的菜单项设置为无效。

· 在程序运行时增减菜单项

在程序运行时，用户还可以根据实际情况的需要动态地增减菜单项。这个功能听起来可能很玄，但是做起来却只须一条语句。如果要隐藏某个菜单选项，只须将该项目的 Visible 属性设置为 False，该项目就会从菜单中消失。如果要显示某个隐藏的菜单项，只要将 Visible 属性设置为 True 即可。

如果用户是在菜单编辑器中，只要消除“可见”复选框，就可以将选中的菜单项隐藏

起来。

· 在菜单中使用复选标记

有时，需要在菜单的选项前显示一个复选标记，以表示某种状态的 On 或者 Off，如图 7.15 所示。

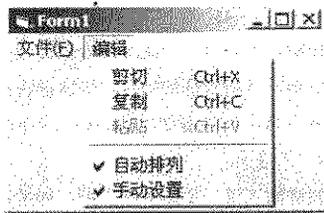


图 7.15 复选标记

要设置菜单选项的复选标记，只要设置控件的 Checked 属性即可。Checked 属性为 True 时，该选项之前就会出现一个复选标记；当 Checked 属性为 False 时，复选标记消失。例如：

```
Private Sub MenuOptions_Click()
    MenuOptionsToolbar.Checked = picToolbar.Visible
End Sub
```

如果希望在菜单编辑器中设置菜单选项的初始状态就是显示复选标记，只须选中该选项，然后选中“复选”复选框。

· 创建子菜单

使用菜单编辑器创建的每个菜单最多可以包含 5 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。须使用子菜单的场合有：

- ① 菜单栏已满；
- ② 某一特定菜单控件很少被用到；
- ③ 要突出某一菜单控件与另一个的关系。

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、分隔线和子菜单标题。

如果要创建子菜单，请按以下步骤执行：

1. 在菜单编辑器中，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，使它们的缩进级比子菜单标题的级别低一级。

限制使用子菜单也是一种好的编辑策略，它可以免去查找应用程序菜单界面的负责。大多数应用程序都只使用一级子菜单，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多个选择。

· 在菜单中添加最近使用过的文件列表

现在的许多 Windows 应用程序都具有这样一个功能——可以在文件菜单下列出最近访问过的文件列表，如图 7.16 所示。当然，这个功能还可以用在别处，例如可以列出最近与自己联系过的朋友等。

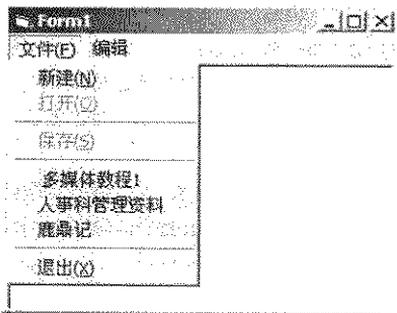


图 7.16 文件列表

显然，要实现这样的功能，就不能完全依靠菜单编辑器，因为这些菜单项目是在程序运行中动态改变的，必须有代码来配合控制。

菜单控件数组与普通的数组没有什么本质的区别，只是数组中的元素是菜单控件而不是其他的数据类型。每个菜单控件数组元素都由惟一的索引值来标识，该值在菜单编辑器上“索引”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。

在菜单编辑器中创建菜单控件数组，请按以下步骤执行：

1. 选取窗体。
2. 单击菜单编辑器按钮，显示菜单编辑器。
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，例如“文件”。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”来创建作为数组中第一个元素的菜单项。
6. 将数组中的第一个元素的“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。

菜单控件数组的各元素在菜单控件列表框中必须是连接的，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线也包括进去。

• 一种简单的方法

创建了菜单控件数组之后，最直接想到的方法就是在打开文件的过程中添加一些代码，把文件名添加到文件菜单下。例如，定义了一个 5 个元素的菜单控件数组，它们共同的名字叫做 menuFileMRU，其中的一个元素(也就是 Index 为 0 的元素)被定义为分隔线，所有的数组元素在没有打开文件之前，Visible 属性都是 False，用户无法看到。

当用户从“打开”对话框中选择一个文件名后，可以添加这样的代码，将文件名添加到文件列表中。

其中，变量 ItemCount 是一个窗体级的变量，它用来保存最新文件的位置。用户还可以继续向这段代码中添加内容，例如进行排序，将最近访问的文件放在第一个，当 5 个名额已满时，淘汰相隔时间最长的一个文件名。

7.3.2 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左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 Windows 98 中，一般是通过单击鼠标右键来激活上下文菜单。弹出式菜单通常用来提供一种访问与上下文有关命令的高效方法。

使用 PopupMenu 方法可显示弹出式菜单，只要菜单至少包含一个菜单项，就可以显示为弹出式菜单。PopupMenu 方法的语法是：

```
[object.]PopupMenu menuname [ , flags[,x[,y[,boldcommand]]]]
```

例如，当用户单击鼠标右键时，以下的代码显示一个名为 mnuFile 的菜单。程序员可用 MouseUp 或 MouseDown 事件来检测是否单击了鼠标右键，但是标准用法是使用 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 menuFile ' 把文件菜单显示为一个弹出式菜单
    End If
End Sub
```

直至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。

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。在一个菜单控件正在活动的任何时刻，调用 PopupMenu 方

法均不会被睬。

为创建一个不显示的菜单栏里的菜单，可在设计时使顶级菜单项目为不可见(在菜单编辑器中清除“可见”复选框)。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。

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。表 7-1 列出了可用手描述弹出式菜单位置的 flags 参数的取值。

表 7-1 用于描述弹出式菜单位置的 flags 参数

参数取值	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

表 7-2 列出了可用于定义弹出式菜单行为的参数取值。

表 7-2 描述弹出式菜单行为的 flags 参数取值

参数取值	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

如果要指定一个 flags 参数，可以从每组中选取一个常数，再用 Or 操作符将它们连起来。

下面的代码可以在用户按下鼠标右键时，显示一个上边框以鼠标当前位置为中心的弹出式菜单，而且这个弹出菜单还可以识别右键对菜单命令的选择。

```
Private Sub Command_Click()
    Dim xloc,yloc
    Xloc = ScaleWidth/2
    Yloc = ScaleHeight/2
    PopupMenu menuEdit,vbPopupMenuCenterAlign Or _
        vbPopupMenuRightButton,xloc,yloc
End Sub
```

在本例中，还可以使用 boldcommand 参数来指定在显示的弹出式菜单中以粗体字体出现的菜单控件的名称(本例中使用粗体显示 mnuFileOpen)，在弹出式菜单中只能有一个菜单控件被加粗。

7.4 工具栏

7.4.1 创建工具栏

工具栏已经成为 Windows 应用程序的一个标准配备，使用它可以进一步增加应用程序

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，许多工具栏按钮我们都已经非常熟悉，可以不假思索地进行操作，例如“打开”按钮、“保存”按钮以及“复制”按钮等。

如果用户现在使用的是 Visual Basic 6.0 的专业版或者企业版，那么恭喜你，因为这两个版本中包含了一个 ActiveX 控件——Toolbar，可以非常容易且很方便地创建工具栏。

Toolbar 控件是一组 ActiveX 控件的一部分，这组自定义控件可在文件 COMYL32.OCX 中找到。为在应用程序中使用 Toolbar 控件，必须将文件 COMCTL32.OCX 添加到工程中，在发行应用程序时，应将文件 COMCTL32.OCX 安装到用户的 Windows 98 的 SYSTEM 或 System32(Windows NT 平台上)的文件夹。

· 创建工具栏的一般步骤

有了 Toolbar 控件，就可以通过将 Button 对象添加到 Buttons 集合中来创建工具栏。每个 Button 对象可以用 Caption 属性显示文本，或者(并且)拥有相关联的 ImageList 控件提供的图像，或者二者兼而有之。

在创建了工具栏之后，还要给 Toolbar 编程，将代码添加到 ButtonClick 事件中，以便对已选定的按钮作出反应。总之，使用 Toolbar 控件创建工具栏的一般步骤是这样的：

1. 在与 Toolbar 控件相关联的 ImageList 控件中插入合适的图像。
2. 通过“属性页”对话框创建 Toolbar 控件的 Button 对象。
3. 在 Buttonclick 事件中用 SelectCase 语句确定按下了哪个按钮，并根据按钮进行相应的操作。

· 在 ImageList 控件中插入图像

ImageList 控件为其他 Windows 公共控件保管图像，这些控件通常包括 ListView、TreeView、TabStrip 和 Toolbar 控件。ImageList 控件还可用于那些需要将 Picture 对象赋予其 Picture 属性的控件上，如 PictureBox、Image 和 CommandButton 控件。

可以把 ImageList 控件视为一种图像仓库，它提供了单一的、一致的图像目录，这样就节省了开发时间。可以不编写装载位图或图标(使用 LoadPicture 函数)的代码，而是一次性将用到的所有图像充填到 ImageList 中，在需要的时候设置 Key 的值，在代码中可以使用 Key 或 Index 属性引用所需的图像。

ImageList 控件也包含在 COMCTL32.OCX 文件中，要使用这个控件必须在“部分”对话框“控件”选项卡上选中“Microsoft Windows Common Controls 5.0”选项。载入文件后，工具栏中会增加几个控件按钮，只要双击其中的 ImageList 按钮，窗体上就会出现一个 ImageList 控件。

ImageList 控件包含了 ListImage 对象的 ListImages 集合。每个 ListImage 对象可用集合的 Index 或 Key 属性值加以引用。在设计时和运行时均可在控件中添加和删除图像。要在设计时添加图像，可用 ImageList 控件的“属性页”对话框。

· 设计时添加 ListImage 对象

1. 用鼠标左键单击 ImageList 控件，然后单击“属性”。
2. 单击“图像”选项卡，显示出 ImageList 控件的“属性页”。
2. 单击“插入图片”，显示出“选定图片”对话框。
4. 用该对话框找到位图或图标文件，并单击“打开”。可以同时选中多个位图或图标文件。
5. 单击“关键字”框，并在其中键入一个字符串，为该图像赋予惟一的 Key 属性。
6. 可选的操作。单击“标记”框，并在其中键入一个字符串，为该图像赋予一个 Tag 属性。Tag 属性不必惟一，也可以不去理会它。
7. 重复 3~6 步，直到将需要的全部图像充填到该控件中，如图 7.17 所示。

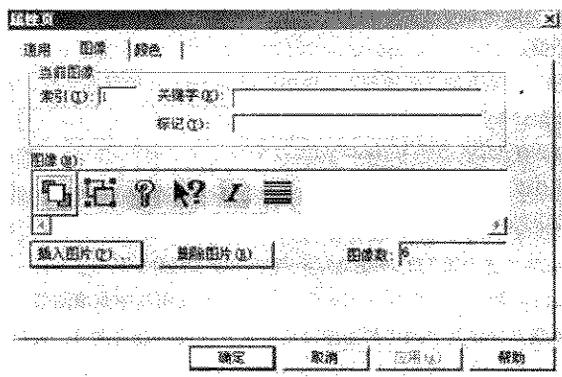


图 7.17 ImageList 控件

在将 ImageList 控件与某个 Windows 公共控件相关联后，就可以用图像的 Index 或 Key 属性指定特定的图像了。由于 ListImage 对象的 Key 属性必须是惟一的字符串，所以可用 Key 属性代替 Index 属性对图像进行引用，这可使代码易读。

由于 Key 必须是惟一的字符串，建议为每个 ListImage 对象起一个描述性的名字，使代码容易阅读和调试。

该控件可以包含任何大小的.bmp 或.ico 图像，虽然图像的显示大小相同，但它们的文件大小可以不同。通常，加入该控件的第一幅图像决定了随后加入的图像的显示大小。例如，如果第一次添加了大小为 32×32 像素的图标，那么在其他控件中显示其后添加图像时，也将显示出相同的大小。

在设计时，可以在 ImageList 控件“属性页”对话框的“通用”选项卡中设置图像的高度和宽度，以像素为单位。在确定图像的大小时，可以选择预定义的大小，或者单击“自定义”，并在“高度”和“宽度”框中键入需要的大小。需要注意的是，只能在 ImageList

中没有图像时设置图像的大小。如果控件已经存了图像，试图改变其大小将导致错误。

· 为工具栏添加按钮

在向工具栏中添加了 COMCTL32.OCX 文件后，双击工具栏按钮就可以向窗体上添加一个工具栏。

在默认情况下，工具栏会紧紧贴着窗体的标题栏。只要向工具栏中添加 Button 对象，并且赋给它图像或者文字，工具栏就大功告成了。

在属性窗口中，将工具栏的 Appearance 和 BorderStyle 属性都设置为 1，可以使工具栏更加突出、漂亮。

但是在能够将图像赋给 Button 对象之前，首先必须将 ImageList 与 Toolbar 控件相关联。

1. 用鼠标选中窗体上的 Toolbar 控件，然后选择“视图/属性页”命令以显示该控件的“属性页”对话框。
2. 在“通用”选项卡的“图像列表”框中选择 ImageList 控件的名称，如图 7.18 所示。

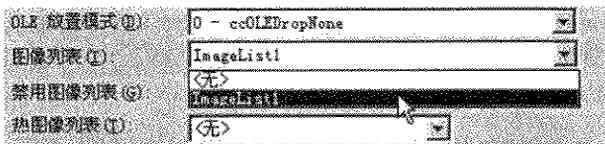


图 7.18 将 ImageList 与 Toolbar 控件相关联

在将 ImageList 控件与其他控件相关联之前，必须先向其中填充图像。一旦 ImageList 与其他控件相关联，并将其中的任何图像赋给了控件，就不能再向 ImageList 控件中添加图像了。

在设置了 Toolbar 和 ImageList 之间的关联后，就可以用“按钮”选项卡添加 Button 对象，并将图像赋给它。

将图像赋给 Button 对象，请按以下步骤执行：

1. 接着前面的步骤，单击“按钮”选项卡(在 Toolbar 控件的“属性页”对话框中)以显示“按钮”选项卡。
2. 单击“插入按钮”以添加新的 Button 对象，工具栏上出现一个空白的按钮。
3. 单击“图像”框，并键入 ListImage 对象的 Key 值。
4. 在“关键字”框中，键入按钮的 Key 属性值，这个值将用来区别按钮，所以不能重复，而且最好起一些描述性强的名字，避免混淆。
5. 用户还可以在“工具提示文本”框中键入简短的文字介绍按钮的功能，这样在程序运行时用户的鼠标停留在按钮上，这些文字就会显示出来。
6. 单击“应用”按钮，此时空白按钮上应出现相应的图案。
7. 步骤重复 2~3 以添加多重的按钮，并将图像赋给新添加的 Button 对象，如图 7.19

所示。

在工具栏中的每个 Button 对象还有 Style 属性，该属性决定了按钮的行为特点，并且与按钮相关联的功能可能受到按钮样式的影响。在属性页的“样式”框中，可以设定按钮的 Style 属性。

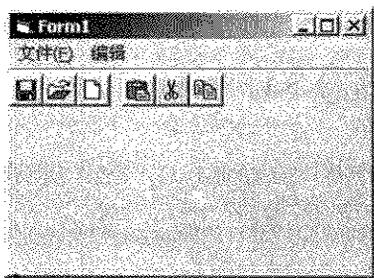


图 7.19 将图像赋给 Button 对象

7.4.2 编写代码

当用户单击按钮(占位符和分隔符样式的按钮除外)时，会发生 `ButtonClick` 事件。可以用按钮的 `Index` 属性或 `Key` 属性标识被单击的按钮。利用这些属性中的任意一个，可以用 `Select Case` 语句编写按钮的功能。

第八章 图形开发

8.1 坐标系统

坐标系统是一个二维网格,可定义屏幕上、窗体中或其他容器中(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。该坐标系如图 8.1 所示。

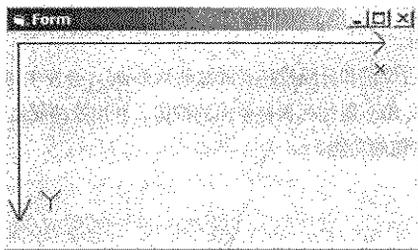


图 8.1 坐标系

以下规则用于 Visual Basic 坐标系:

① 当移动控件或调整控件的大小时,使用控件容器的坐标系。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。

② 所有的图形和 Print 方法,使用容器的坐标系。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。

③ 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。

④ 屏幕的左上角总是(0, 0)。任何容器的缺省坐标系,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,现在使用的是缺省系统。

· 缢(twips)的解释

所有 Visual Basic 的移动、调整和图形绘制语句，根据缺省规定，使用缢为单位。缢是打印机的一磅的 1/20(1440 缢等于一英寸；567 缢等于一厘米)。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

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。使用坐标系有以下三种不同的方法：

- ① 使用缺省的刻度
- ② 选择标准刻度
- ③ 创建自定义刻度

改变坐标系的刻度，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，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。

· 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。

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。缺省刻度单位为缢。若要返回缺省刻度，可使用无参数的 Scale 方法。

· 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，或取得有关坐标系当前刻度的详细信息。

8.2 图形方法

Visual Basic 提供有创建图形的一些方法（如图 8.2 所示）。总结在表 8-1 中的这些图形方法，适用于窗体和图片框。

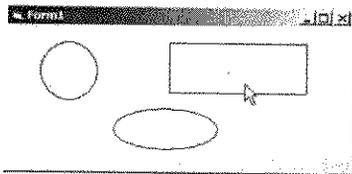


图 8.2 图形方法

表 8-1 图形方法

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, 并像 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)。

• 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, 在所有其他控件之下, 所以, 需要创建出现在应用程序中其他事物之下的图形时, 这种方法就很好。

• 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。

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 就认为是画在代码所连接的窗体上。例如, 下列语句画一个点, 在:

- ① 名为 MyForm 的窗体上
MyForm.PSet (500, 500)
- ② 名为 picPicture1 的图片框内
picPicture1.PSet (500, 500)
- ③ 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。

• 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。

• 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 True；否则，Form_Load 事件中的图形方法将被忽略。

8.3 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：

- ① 使用 RGB 函数。
- ② 使用 QBColor 函数，选择 16 种 Microsoft QuickBasic(r) 颜色中的一种。
- ③ 使用在“对象浏览器”中列出的内部常数之一。
- ④ 直接输入一种颜色值。

• 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 0~255 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)
```

• 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。表 8-2 描述了这些颜色属性。

表 8-2 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

• 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:

①使用定义的常数

②直接使用颜色设置值

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
```

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。如果知道 Visual Basic 是如何用数值来指定颜色, 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 0~16,777,215(&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 0~255 (&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~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~&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够精确地显示出来。

· 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：

&H8000002&

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：

- ① 窗体
- ② PictureBox
- ③ 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 VGA16 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他图像中。

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，那么窗体、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。

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。



8.4 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：

- ① Image 控件
- ② Line 控件
- ③ 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他 Visual Basic 控件少，这就提高了 Visual Basic 应用程序的性能。

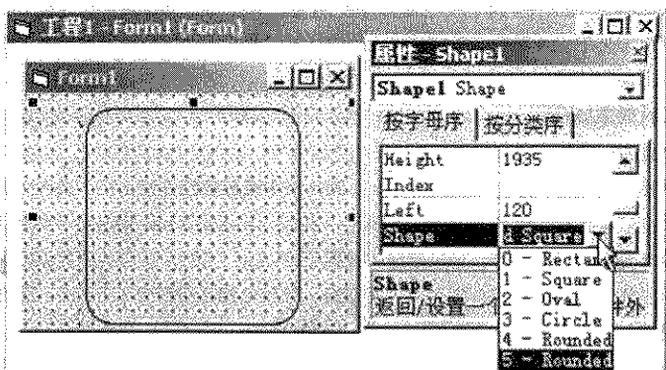


图 8.3 图形控件

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得要少。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 方法要求在运行时代码创建圆；而用形状控件，只需在设计时简单地把它拖到窗体上，并设置特定的属性。

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 控件共有的其他功能。图形控件的限制条件有以下几点：

- ① 不能出现在其他控件之上，除非它们是在一个容器里，而这个容器能出现在其他控件之上（如图片框）。
- ② 不能在运行时接收焦点。
- ③ 不能作为其他控件的容器。
- ④ 不具有 hWnd 属性。



8.5 图片应用

图片可显示在 Visual Basic 应用程序的三种位置：

- ① 窗体上
- ② 图片框内
- ③ 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，其他图形应用程序，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 也能像.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。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

在设计时,添加图片有两种方法:

①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。

②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

在运行时添加图片有四种方法:

①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。

②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)
```

③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
```

④从剪贴板对象复制图片。

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他对象。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，而无需用其他图片替换它。下列语句是从名为 imgDisplay 的图像控件中删除图片：

```
Set imgDisplay.Picture = LoadPicture("")
```



8.6 创建从蓝逐渐变黑的背景

下列代码绘制了窗体的背景，它使用变化的蓝色阴影框绘制而成。最复杂的问题是如何在 256 色模式以及增强色(16 位)和真彩色(24 位)模式下得到连续而平滑的渐变效果。

在 256 色模式下，从浅蓝色到黑色的平滑过渡需要颤动的颜色。Visual Basic 的 Line 方法对于值线来说不允许有颤动的颜色，但它允许一个框用颤动的“实”色来填充(如图 8.4 所示)。要完成这项工作，需要使用下面的过程把窗体的 DrawStyle 属性改为 vbInvisible，把 ScaleMode 属性改为 vbPixels。DrawStyle 属性确定线的样式，将其设为 vbInvisible，可以防止在每个蓝框的周围画一黑色边框。将 ScaleMode 属性设为 vbPixels 可以用像素计算每个框的尺寸，并且没有舍入错如。这样，能够防止框间的叠加成有空格出现。

```
Option Explicit
```

```
Private Sub Form_Paint ()
```

```

    Dim IY As Long
    Dim IScaleHeight As Long
    Dim IScaleWidth As Long
    ScaleMode = vbPixels
    IScaleHeight = ScaleHeight
    IScaleWidth = ScaleWidth
    DrawStyle = vbInvisible
    FillStyle = vbFSSolid

    For IY = 0 To IScaleHeight
```

```
FillColor = RGB (0,0,255 - (IY * 255) / ScaleHeight )  
Line (-1, IY - 1) - (ScaleWidth, IY + 1)  
Next IY  
End Sub
```

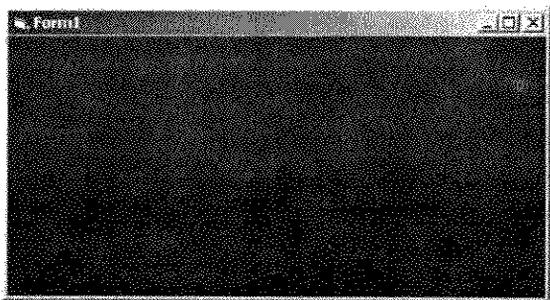


图 8.4 窗体从蓝色渐变到黑色

第九章 文本处理

9.1 字 体

文本是使用字体来显示的，字体是具有铅字字样的字符集，能够以特定的大小、风格和粗细来使用。不同的字体可以给人不同的感觉，例如楷体秀丽，魏碑体刚劲，而隶书则散发着古韵。不同内容的文字需要配合恰当的字体才会具有更强的感染力。如图 9.1 所示。

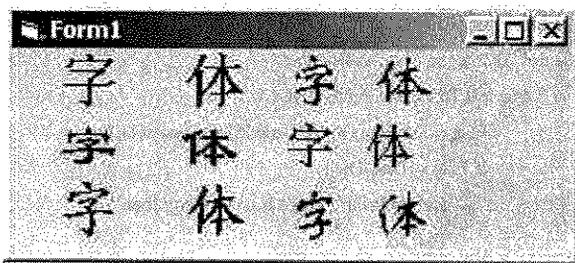


图 9.1 各种字体

在字体的使用上，还要注意不能走向另一个极端——那就是过分追求字体的多样性，恨不能把可以使用的字体全部都用上。要记住，字体的使用应是画龙点睛，而不是喧宾夺主，这其中要掌握好一个尺度。

Windows 98 和 Windows NT 操作系统提供了一套完整的基本字体。这些字体大部分是 TrueType 字体，可以随意缩放，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，它将根据选定的字号，变换成位图显示在屏幕上。

在打印时，选定的 TrueType 字体或其他字体，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，如果打印机中有一种可用的、能提高打印速度的、字体效果的字体时，打印字体将替换相应的 TrueType 字体。

在为应用程序选择字体时，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，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。例如，Windows 选择的字体可能会放大文本，以致屏幕上的标签有所重叠。

避免字体问题的一个方法是：与应用程序一起发布必要的各种字体。但是为了与应用程序一起发布，要从版权所有者那里获得字体的许可。

程序员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，可以让用户从所提供的字体列表中选择一种相近的字体。

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

程序能够容易地利用 Printer 和 Screen 对象的 Fonts 属性检测出用户系统和打印机中是否有可用的匹配字体。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，程序员可以在程序中循环地通过该属性数组查找相匹配的名称字符串。下面的代码示例，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 marching font."  
            Exit for  
        End If  
    Next I  
End Sub
```

· 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：

- ① 字体名(字样)；
- ② 字体大小(用磅表示)；
- ③ 特别特征(黑体、斜体、下划线或删除线)。

在设计时，通过单击“属性”窗口中的“Font”框中的按钮，可以进入“字体”对话框中进行字体属性的设定，如图 9.2 所示。

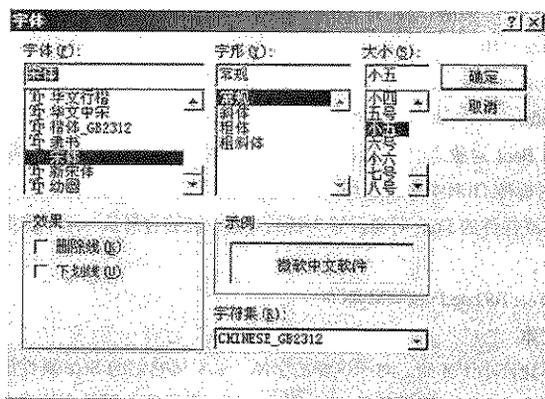


图 9.2 字体对话框

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。表 9-1 说明了 Font 对象的一些属性。

表 9-1 在运行时可用于设置字体样式的有关属性

属 性	类 型	描 述
Name	String	指定字体的名字，例如 Arial 或 Courier
Size	Single	以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"
    .Bold = True
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样式的变动。首先，应设置 Name 属性；然后，设置任何一个 Boolean 属性(Bold 和 Italic 属性)为 True 或 False。

也可以在 Font 对象中存储一组字体属性。使用 StdFont 类，像其他任何对象一样，可以声明 Font 对象：

```
Dim MyFont As New StdFont
With MyFont
```

```
.Name = "Arial"
.Size = 10
.Bold = True
```

```
End With
```

在创建新的 Font 对象之前，必须使用“引用”对话框(选择“工程”菜单中的“引用”命令)，先设置对标准 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，能够容易从一组字体属性切换到另一组：

```
Set lblYearToDate.Font = MyFont
```

· 使用小字体

某些字体不允许小于 8 磅。对于这样的字体，如果将它们的大小属性设置为小于 8 磅的尺寸，那么无论是 Name 属性还是 Size 属性，都会自动变为其他字体或尺寸。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不同而不同。如果文本是由一个属性指定(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，则字体属性改变之后对使用 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，一些词用黑体，而其他的用正常字体)时，则应创建图片框，使用 Print 方法来显示文本。

· 在窗体和图片框中显示文本

为了在窗体或图片框中显示文本，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 对象的 Print 方法。

Print 方法在本书中已经使用了很多次，但是还没有详细对它进行介绍。Print 方法的语法是：

```
[object.]Print[outputlist][[,],]
```

其中，object 参数是可选项；如果被省略，则 Print 方法将应用于当前窗体。

outputlist 参数是显示在窗体或图片框上的文本，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)。如果 outputlist 参数有多个项，它们必须用逗号或分号进行分隔。

在默认情况下，每个 Print 方法都是打印完文本后自动移到下一行。如果没有要打印的项，Print 只是简单地跳到下一行。但是如果把一个分号(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——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，使用下面两条语句在窗体上显示的语句将出现在一行上（如图 9.3 所示）。

```
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
```

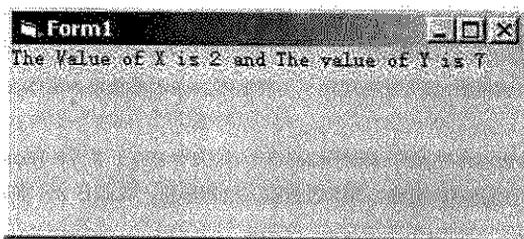


图 9.3 在窗口中打印文字

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。

如果没有特意指定，Print 方法将从绘图坐标的原点，即窗体或者图片框的左上角开始输出，程序员可以使用下面的方法指定绘图坐标，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。

对象中用 Print 和图形方法创建的所有文本和图形，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)，按照默认规定，原点是左上角。

使用 Current X 和 CurrentY 属性可以直接用来设置窗体和图片框的绘图坐标，Print 方法将从该坐标处开始输出。例如，下面的语句将在坐标(200, 300)处显示输出信息（如图 9.4 所示）。

```
CurrentX = 200
```

```
CurrentY = 300
```

```
Print "(200,300)"
```



图 9.4 在坐标(200, 300)处显示输出信息

此外, 在使用 Print 方法之前, 可使用 TextHeight 和 TextWidth 方法设置文本行的高度和宽度, 这两种方法都考虑了该对象的字体大小和风格。它们的语法是:

```
[object.]TextHeight(string)
```

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。

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 属性设置的特定的行。假设示例文本中没有回车符, 用以下语法可将 Current Y 设置到第 n 行的开始:

```
CurrentY = TextHeight("sample") * (n-1)
```

```
CurrentX = 0
```

例如, 下述语句将绘图坐标设置到第 5 行的开始:

```
CurrentY = TextHeight("sample") * 4
```

```
CurrentX = 0
```

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, 所以本方法将会很有用。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。

➡ 9.2 输出格式

在显示数字的格式上, 像日期和时间的模式一样, Visual Basic 也提供了巨大的灵活性。对于数字、日期和时间, 可以很容易用国际模式来显示。

• 数字的模式

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，以及货币模式。它的语法是：

`Format(expression[format])`

`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模式。表 9-2 列出了一些最常使用的一些符号。

表 9-2 Format 函数的常用模式符号

符 号	描 述
0	数字保留区，若恰当，在本位置打印尾部或前导
#	数字保留区，不打印尾部或前导
.	小数保留区
,	千位分隔符
- + \$ 0 space	字母字符和各种其他负号，都要按格式字符串中打入的原样，精确的显示出来

下面是一些使用 `Format` 函数的例子。

小数分隔符是句号“.”，千位分隔符是逗号“，”。然而，分隔符的正确显示要依 Windows “控制面板”中的“区域设置”的设置而定。

如果没有指定模式，则返回字符串。

• 日期和时间的格式

为了打印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 `Format` 函数和 `Now` 参数来标识和模式化当前的日期和时间。需要注意的是，对于日期分隔符(/)、时间分隔符(·)，以及 `AM/PM` 等文本而言，其真正的显示格式会因计算机上的“区域设置”不同而有所差异。

在开发阶段，日期与时间是以短日期的格式配合代码的国际标准来显示的。而在运行时，短日期则是根据系统的国际标准而定，而系统的国际标准和代码的国际标准可能并不相同。

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)”。

```
Format(8315.4, "00000.00") => 08315.40
```

```
Format(8315.4, "#####.##") => 8315.4
```

```
Format(8315.4, "##,##0.00") => 8315.40
```

```
Format(315.4, "$##0.00") => $315.40
```

• 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引进格式，而不用在 `format` 参数中指定符号。格式名总是用双引号(“”)括起来的。表 9-3 列出了可使用的格式名。

表 9-3 Format 函数可以使用的格式名

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。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	用标准的科学计数法
General Date	如果 expression 同时包含了日期和时间，则显示它们。如果 expression 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结构
Long Date	使用用户的系统设置所制定的 Long Date 格式
Medium Date	使用 dd-mmm=yy 格式，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制定的 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 hh:mm AM/PM 格式，显示小时、分钟和 AM 或 PM
Short Time	使用 hh:mm 格式，显示小时和分钟
Yes/No	1 为 Yes, 0 为 No
True/False	任何非零数字值为 True, 0 为 False
On/Off	任何非零数字值为 On, 0 为 Off

Format 函数还支持其他许多特殊自负，如百分比保留区和指数。

9.3 剪贴板应用

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。

· 使用剪贴板剪切、复制和粘贴文本

两个最有用的方法是 SetText 和 GetText。用这两个方法向剪贴板和从 Clipboard 传送字符串数据。

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 方法与选定文本属性结合起来使用，可容易的编写文本框的“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。然后，Copy 和 Cut 这两个过程，都用下面的语句将 Text1 中选择的文本复制到 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 (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 和其他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。

• 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，就能增加为其他应用程序提供可用格式的机会。其他的 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，在表 9-4 进行了描述。

表 9-4 文本外的数据格式

常 数	描 述
vbCFLink	动态数据交换链接
vbCFText	文本
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.ActiveControlSelText
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
...
End If
End Sub

Private Sub mnuCut_Click ()
MnuCopy_Click
If TypeOf Screen.ActiveControl Is TextBox Then
Screen.ActiveControlSelText = ""
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
```

```

...
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
        ...
    End If
End Sub

```



9.4 文件的读写

· 文件访问类型

文件本身除包括一系列定位在磁盘上的相关字节外，再也没有其他东西。当应用程序访问一个文件时，必须假定字节表示字符、数据记录、整数、字符串等。

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：

1. 顺序型：适用于读写在连续块中的文本文件。
2. 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。
3. 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，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数据。除了没有数据类型或者记录长度的含义外，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。

· 文件访问函数和语句

以下函数用于所有的三种类型的文件访问，如表 9-5 所示：

Dir	FileLen	LOF
EOF	FreeFile	Seek
FileCopy	GetAttr	SetAttr
FileDateTime	Loc	

表 9-5 可用于三种直接文件访问的语句和函数

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		

· 使用顺序文件访问

当以顺序型访问打开一个文件时，可执行如下操作：

1. 从文件输入字符
2. 向文件输出字符
3. 把字符加到文件

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，在为其他类型的操作重新打开它之前必须先使用 Close 语句关闭它。

· 使用随机文件访问

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意标准类型，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，下面定义的 Worker Type 创建由两个字段组成的 17 个字节的纪录：

```
Type Worker
    LastName As String *10
```

Title As String * 7

End Type

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他变量，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。

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纪录。

如果要编辑随机型访问的文件，要先将记录从文件读到程序变量，然后改变变量的值，最后把变量写回该文件。

• 使用二进制文件访问

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内容。当要使文件的尺寸近 2 英寸，应使用二进制型访问。

注意：当把二进制数据写入文件中时，使用的变量是 Byte 数据类型的数组，而不是 String 变量。String 包含的是字符，而二进制型数据可能无法正确地存储在 String 变量中。

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。

第十章 多媒体应用

10.1 多媒体基础

通常所说的多媒体(Multimedia)是指计算机的多媒体技术,它综合了文字、声音、图形图像、动画和视频等媒体,大大扩展了计算机的应用领域,丰富了人们的生活,可以相信,未来将会是图、文、声和感知技术共同发展的时代,未来的计算机也将更具人性化。

10.2 多媒体控件与函数

· 多媒体控件 Multimedia MCI(MMControl)

多媒体控件 MMControl 是外部 ActiveX 控件。在设计时,需要在菜单项“工程”中选择“部件”,在“部件”对话框的控件选项中选择“Microsoft Multimedia Control 6.0”,然后单击“确定”按钮,在工具箱中将多媒体控件 MMControl 加到一个窗体上。

正如前面所介绍的, Multimedia MCI 控件可用于管理媒体控制接口(MCI)设备上的多媒体文件的录制和播放。如用来向声卡、MIDI 序列发生器、CD-ROM 驱动器、视频 CD 播放器和视频磁带记录器及播放器等设备发出 MCI 命令。

MMControl 控件的按钮被分别定义为(如图 10.1 所示):

- Prve(前一个)
- Next(下一个)
- Play(播放)
- Pause(暂停)
- Back(向后进步)
- Step(向前进步)
- Stop(停止)
- Record(录制)
- Eject(开仓/弹出)



图 10.1 多媒体控件 MMControl

从外观上看，控件的按钮很像家用的录像机的按键。

MMControl 控件使用一套与设备无关的高层次 MCI 命令，控制多种多媒体设备。在按钮上的名称和对应的 MCI 命令基本一致。例如：

- 打开 MCI 设备的命令“Open”对应 MCI 的“MCI OPEN”；
- 关闭 MCI 设备的命令“Close”对应 MCI 的“MCI CLOSE”；
- 使用 MCI 设备进行播放的命令“Play”对应 MCI 的“MCI PLAY”；
- 暂停播放/录制的命令“Pause”对应 MCI 的“MCI-PAUSE”或“MCI-RESUME”；
- 停止 MCI 设备的命令“Stop”对应 MCI 的“MCI-STOP”；
- 录制 CMI 设备输入的命令“Record”对应 MCI 的“MCI-RECORD”；
- 从光驱中弹出 CD 盘的命令“Eject”对应 MCI 的“MCI-EJECT”。

此外，向后步进命令 Back 和向前步进命令 Step，对应 MCI 的“MCI-STEP”；查找命令 Seek(包括 Perve 和 Next)，对应 MCI 的“MCI-SEEK”；保存打开文件的“Save”命令，对应 MCI 的“MCI+SAVE”。

使用 MMControl 控件的 Command 属性的方法如下：

```
MMControl1.Command="Open"
```

```
MMControl1.Command="Play"
```

```
MMControl1.Command="Stop"
```

```
MMControl1.Command="Close"
```

在 FileName 属性中，可以选择一个 AVI 文件。此外，也可以设置控件的属性，方法是首先选中 MMControl 控件，然后单击鼠标左键，选择“属性”，如图 10.2 所示。

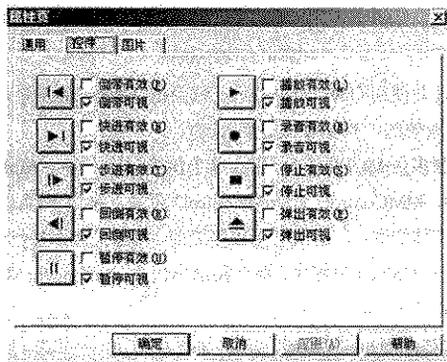


图 10.2 多媒体控件 MMControl 的属性设置

在大多数情况下,使用 Visual Basic 的这些命令基本能满足人们的需要,如果要满足某些特殊需要,可以考虑使用 Win32 的 API,使用其编程函数和技术。

· MMControl 控件编程方法

在 Visual Basic 中,通常应将 MCI Open 命令放到 Form Load 中。为了正确管理多媒体资源,在退出应用程序之前,应该关闭那些已经打开的 MCI 设备,将代码放到 Form Unload 过程中,那么在退出包含 Multimedia MCI 控件的窗体之前,就可以关闭那些已经打开的 MCI 设备。为了更好地说明多媒体控件的特性,在这里,主要介绍 MMControl 控件的属性设置和编程方法。

在运行时,控件可以是可见的或不可见的。如果使用 Multimedia MCI 控件中的按钮,需要将 Visible 和 enabled 属性都设置为 True。如果不使用控件中的按钮,而只是用 Multimedia MCI 控件的多媒体功能,可将 Visible 和 enabled 属性设置为 False,不管可见与否,无论有没有用户交互操作,应用程序均可控制 MCI 设备。

要设置这些属性,可以在窗体的 MMControl1 上单击鼠标左键,然后在弹出的菜单中选择“属性”,在“属性”对话框中选择“控件”标签,根据需要选择按钮在什么情况下可见或有效,如图 10.3 所示。

在用户从 Multimedia MCI 控件选取按钮之前,必须先把 MCI 设备打开,打开的命令写在 Form Load 中。

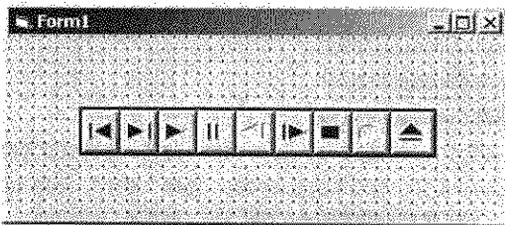


图 10.3 激活按钮

正确设置控件属性并使用 Open 命令之后,就会激活 MCI 设备所支持的 MMControl 控件的按钮(有些按钮是有效的,有些按钮是无效的)。

如果需要同时控制多台 MCI 设备,可以在单个窗体中加入多个 Multimedia MCI 控件(每台设备需要一个控件),MMControl1 和 MMControl2 可独立播放不同的 .AVI 文件。

为了正确管理多媒体资源,在退出应用程序之前,应该关闭那些已经打开的 MCI 设备。将下面的语句收到 Form Unload 过程中,那么在退出包含 Multimedia MCI 控件的窗体之前,就可以关闭那些已经打开的 MCI 设备。

```
MMControl1.Command="Close"
```

一般情况下,在使用 Multimedia MCI 控件记录音频信号之前,应打开一个新的文件。这

样就可以保证记录声音的数据文件格式与系统记录格式完全兼容。在关闭 MCI 设备之前，还应该发出 MCI Save 命令，把记录的数据保存到文件中去。

10.3 多媒体应用

10.3.1 播放声音文件

• mciExecute 函数

下列代码展示了如何声明 mciExecute API 函数并用它来播放 WAV 文件。要试用这个例子，可将代码加入到一个新项目的空窗体中，运行这个应用程序，并单击窗体的任何位置，如图 10.4 所示。

```
Option Explicit
Private Declare Function mciExecute Lib "winmm.dll" (ByVal lpstrCommand As String) As Long
Private Sub Form_Click()
    Dim x
    X = mciExecute("Play C:\Windows\Media\Tada.wav")
    'Change filename to name of your sample WAV file
End Sub
```

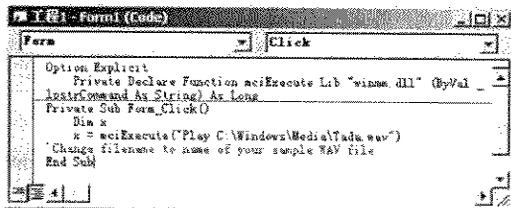


图 10.4 mciExecute 函数的使用

• Multimedia MCI 控件

Visual Basic 所含的 Multimedia MCI 控件是播放声音文件的极好的工具。稍微修改一下代码，还可以用这个控件播放面向 Windows 的视频 (AVI) 文件和多媒体电影 (MMM) 文件，并能控制多媒体硬件，如图 10.5 所示。

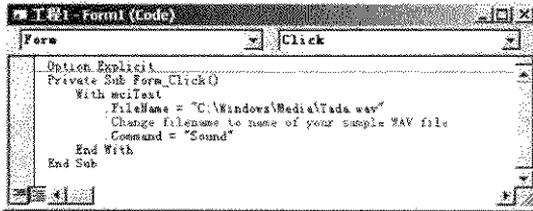


图 10.5 Multimedia MCI 控件的使用

对下面的例子而言，可把一个 Multimedia MCI 控件拖到一个新项目的空窗体上。

```
Option Explicit
Private Sub Form_Click()
    With mciTest
        .FileName = "C:\Windows\Media\Tada.wav"
        'Change filename to name of your sample WAV file
        .Command = "Sound"
    End With
End Sub
```

10.3.2 播放视频文件

• mciExecute 函数

下列代码展示了如何声明 mciExecute API 函数并用它播放的样本视频文件。要试用这个例子，可在一个新项目的空窗体中加入这些代码，然后运行应用程序，并单击窗体的任何位置。

```
Option Explicit
Private Declare Function mciExecute Lib "winmm.dll" ( ByVal lpstrCommand _
    As String) As Long
Private Sub Form_Click()
    Dim x
    X = mciExecute ("avi 文件的路径")
    'Change filename to name of your sample AVI file
End Sub
```

• Multimedia MCI 控件

Visual Basic 所包含的 Multimedia MCI 控件能较好的播放视频文件。

```
Option Explicit
Private Sub Form_Click()
```

```
With mciTest
    .FileName = " avi 文件的路径"
    .Command = "Open"
    .Command = "Play"
End With
End Sub
Private Sub mciTest_Done(NotifyCode As Integer)
    MciTest.command = "Close"
End Sub
```

第十一章 API 函数

Visual Basic 的一个强大的功能是可以调用动态链接库(DLL)文件中的过程,包括 Microsoft Windows 提供和使用的应用程序编程接口(API)函数。Windows API 函数和它们的语法在 Win32 SDK Help 文件中描述。由于能够访问数以千计的 Windows API 函数和其他 DLLs 中的函数,因而扩展了 Visual Basic 的功能,使其功能远远超过许多其他编程语言。

11.1 在 Visual Basic 中使用 API

11.1.1 API 简介

• 如何调用 API 函数

要使用 API 函数,只需在源代码中简单地声明它们,然后像调用 Visual Basic 的任何其他函数一样调用它们即可。

声明

由于 API 函数不是 Visual Basic 的内部函数,因此必须在使用前显式地声明它们,联机帮助给出了 Declare 语句的精确语法。

有些 API 函数的声明相当长。在过去,可以保留这种长的声明,也可以精简它以便放在一行内。例如,下列代码展示了 API 函数 GetTempFileName 的标准声明。在 Visual Basic 4.0 以前的版本中,在源代码文件中整个声明作为一行输入:

```
Private Declare Function GetTempFileName Lib "kernel32" Alias
    "GetTempFileNameA"(ByVal lpszPth As String,ByVal lpPrefixString
    As String,ByVal wUnique As Long,ByVal lpTempFileName As String)
    As Long
```

下面是该声明的精简形式,它也作为一行输入。这种声明更容易管理,但可读性被降低:

```
Private Declare Function GetTempFileName& Lib"kernel32" Alias
    "GetTempFileNameA"(ByVal Pth$,ByVal Ptr$,ByVal Unq$,ByVal Fnm$)
```

Visual Basic 允许用另一种方式格式化这些声明,即保持了较长的、易读的参数名,又使用了更短的行,以使它们在正常大小的编辑窗口内可视。对这种改进格式而言,行连续

符是关键。下列代码以容易阅读的风格展示了上面的声明(可自由修改这种布局以适合自己的风格)。

```
Private Declare Function GetTempFileName
Lib "kernel32" Alias "GetTempFileNameA"(
    ByVal lpszPath As String,
    ByVal lpPrefix String As String,
    ByVal wUnique As Long,
    ByVal lpTempFileName As String
) As Long
```

· 32 位函数声明

在前面的例子中, GetTempFileName 函数实际上化名为函数 GetTempFileNameA。在 Microsoft Windows 9X 中, 包含字符串参数的 32 位函数声明已经被改名, 以便与 16 位形式相区别, 因为它们已用 32 位代码规范进行了重建, 尽管它们的内容仍使用 ANSI 字符串。注意: 在 Visual Basic 的 32 位版本中, Windows API 函数名是区分大小写的。

注意: 已经为 32 位 Microsoft Windows NT 开发定义了三类函数, 其中字符串在系统内部作为 Unicode 字符串管理。在这种情况下, 源函数名带 W 后缀(代表 Wide), 但在 Windows 9X 中不支持 Unicode 函数版本。

要确保得到适当格式化的函数声明, 可以在 WIN32API.TXT 文件中见到 Windows API 函数声明, 该文件位于 Visual Basic 目录下的 WINAPI 目录。把这些函数声明从这一文件移到应用程序中有几种办法。可以把文件装载到 WordPad 中, 并用手工方式把需要的声明拷贝到 Visual Basic 应用程序中, 然后按上面提到的方法修改其格式。此外, 还可以使用 API Viewer 应用程序, 它是 Visual Basic 附带的, 可以使这一过程自动化。

API Viewer 允许装载文本 API 文件或数据库 API 文件, 并方便地浏览其内容。可以选择像函数声明这样的项, 把它拷贝到剪贴板上, 然后粘贴到 Visual Basic 中。可以在 Windows Start 菜单中从 Visual Basic 目录中选择 API Text Viewer 来启动 API Viewer, 也可以从 WINAPI 目录运行 APILOAD.EXT 程序。如图 11.1 所示。

· 字符串

当向 API 函数传递字符串参数时要注意几个问题。其中一个问题是 API 函数不能建立任何字符串空间。在向函数传递字符串之前, 必须创建足够长的字符串空间以便处理可能返回的最长的字符串。例如, 下列代码声明了 API 函数 GetWindowsDirectory, 它返回到 Windows 目录的路径。存放该路径的字符串缓冲区必须有足够大, 以容纳 GetWindowsDirectory 返回的数据。在下面的例子中, 在该函数被调用之前, 用 Space\$ 函数建立了长度为 144 字节的字符串变量 WinPath。如果字符串参数的长度不够, 将导致 API 函数无法

返回数据。

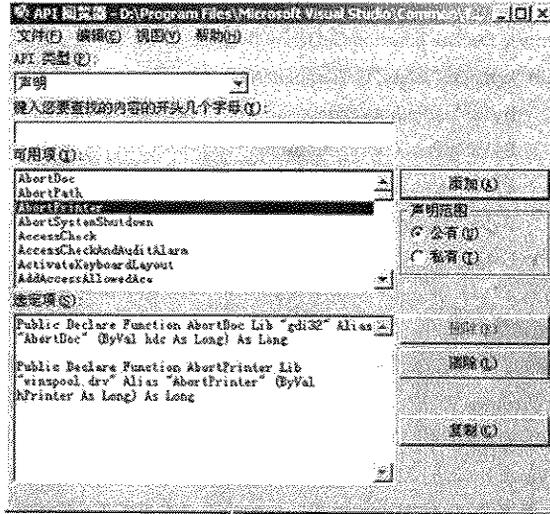


图 11.1 API Viewer

Option Explicit

Private Declare Function GetWindowsDirectory

```
Lib "kernel 32" Alias "GetWindowsDirectoryA"(  
ByVal lpBuffer As String, ByVal nSize As Long ) As Long
```

Private Sub Form_Click()

```
Dim WinPath As String  
Dim Rtn As Integer  
Const MAXWINPATH = 144  
WinPath=Space$(MAXWINPATH)  
Rtn=GetWindowsDirectory(WinPath,MAXWINPATH)  
WinPath=Left$(WinPath,Rtn)  
Print WinPath
```

End Sub

如果把这一代码段并入一程序中，当窗体被单击时，GetWindowsDirectory API 函数返回的 Windows 目录路径将被显示，结果如图 11.2 所示。

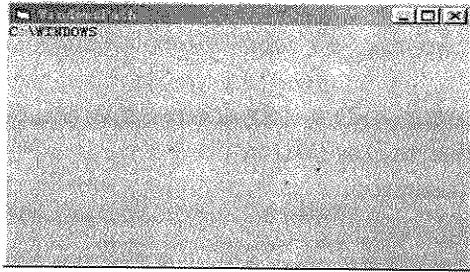


图 11.2 GetWindowsDirectory 返回的 Windows 目录路径

此外还要注意，返回的字符串以值为 0 的字节结束。在上面的例子中，函数返回了字符串的长度。但在许多情况下，当使用 API 函数时无法知道字符串数据的实际长度，除非查找 0 字节。如果 API 函数不返回字符串的长度，可以用下列方法删除字符串中的额外空间：

```
WinPath=Left$(WinPath,InStr(WinPath,Chr$(0))-1)
```

· 如何向 API 函数传递过程地址

向 API 函数传递 Visual Basic 过程地址是为实际代码保存该地址的一种技巧，从 Visual Basic 5 开始，增加了 AddressOf 操作符，它用于 C 编程技术。在 C 语言中，这一过程称为回调。回调是指在 API 函数被执行时调用 Visual Basic 过程。

调用的 Visual Basic 过程的参数由 API 函数确定。例如，在下列代码中，EnumChildWindows API 函数调用 Visual Basic ChildWindowsProc 函数。代码应该放在代码模块(BAS)中，Visual Basic 编程环境的窗口被显示在 Immediate 窗口中。

```
Option Explicit
```

```
Private Declare Function GetActiveWindow Lib "User32" ()As Long
```

```
Private Declare Function EnumChildWindowsLib"User32" (ByVal hWnd As Long,  
ByVal lpWndProc As Long, ByVal lp As Long )As Long
```

```
Sub Main()
```

```
Dim hWnd As Long
```

```
Dim x As Long
```

```
hWnd=GetActiveWindow()
```

```
If(hWnd)Then
```

```
    X=EnumChildWindows(hWnd,AddressOf ChildWindowProc,0)
```

```

End If
End Sub

Function ChildWindowProc ( ByVal hWnd As Long,ByVal lp As Long )As Long
    Debug.Print "Window:";hWnd
    ChildWindowProc=1
End Function

```

EnumChildWindows 函数向 ChildWindowProc 传递 hWnd 和 lp 参数。被调用过程的格式在关于 EnumChildWindows 的 Win32SDK Help 主题中描述。按照约定，被调用过程以后缀 Proc 结束。这一后缀告诉用户该过程不能被直接调用——相反，它是回调过程的目标。

回调在调试时可能产生一些混乱的结果。例如，前面的例子列出了当过程在 Visual Basic 中执行时，在 Visual Basic 编程环境中处理的所有窗口。该过程作为独立程序编译和执行时，能列出要处理的一套不同的窗口。

当使用 AddressOf 操作符时，要记住以下要点：

①AddressOf 操作符只能用在代码模块(BAS)中，并且只能作为过程变元的一部分。这样可以防止传递作为对象或窗体一部分的过程地址。

②通过 AddressOf 操作符引用的过程必须在同一个项目中。

• 如何理解函数声明中的 ByVal、ByRef 和 As Any

这一问题在 32 位 Visual Basic 编程中并不重要，因为大多数的 As Any 声明已被显式参数数据类型所取代。然而，这里介绍一种特殊情况——WinHelp API 函数的 16 位版本存在的问题。要注意这一函数的 32 位版本常有 As Any 参数声明。还要记住 Visual Basic 6.0 不支持 16 位 Windows；因此要试用这些功能需要有 Visual Basic 4.0 或更早的版本。

许多 16 位(和几个 32 位)API 函数声明中有一个或多个参数被声明为 As Any，而不是指定的数据类型(例如 Integer 或 String)。这就是这些参数能被用来传递各种数据类型的原因，传递何种类型取决于使用函数的目的。例如，在下列代码中使用 WinHelp API 函数的 16 位版本来显示 Visual Basic 帮助文件，考虑 WinHelp 函数的第四个参数，它被声明为 As Any。根据 wCommand 的值，最后一个参数可用来传递长整数或指向字符串的指针。

Option Explicit

```

Private Declare Function WinHelp Lib "User" ( ByVal hWnd As Integer,
    ByVal lpHelpFile As String, ByVal wCommand As Integer,
    ByRef dwDate As Any ) As Integer

```

```

Private Sub Form_Click()

```

```

Dim x%,y&
X%=WinHelp(Form1.hWnd, "VB.HLP",vbHelpContents,ByVal y&)
End Sub

```

按照约定，API 函数中的所有 As Any 参数都通过引用声明(笔者在声明中加入了 ByRef 关键字，以便显式声明它们，但当没有在参数声明中看到 ByVal 或 ByRef 时，ByRef 是缺省值)，通常还会发现 ByVal 关键字在为 As Any 参数传递的变量之前，As Any 参数位于函数被调用的位置。这意味着必须特别注意如何对待实际调用该函数的应用程序中的这些参数。实际上，不正确地使用 ByRef 和 ByVal 关键字将使应用程序崩溃。仔细查看代码中对 WinHelp 的调用。在这种情况下，长整数 y&作为第四个参数传递，它使用 ByVal 修改，这可保证传递长整数值 0。

11.1.2 API 文本浏览器

Visual Basic 含有方便的 API Viewer 实用程序(Apiloat)，它位于 WINAPI 目录，用来浏览和插入众多的 API 常量、相关的 Type 结构和过程声明。在 Visual Basic 简体中文版中又叫做 API 文本浏览器。

11.2 应用实例

以下是几个应用 API 编程的示例。

11.2.1 确定 CPU 类型

下面的代码使用 GetSystemInfo API 函数来确定系统的 CPU 类型。通过观察 SYSTEM INFO Type 结构，会发现这个函数还返回几个有用的有关系统的其他信息位。在不久的将来能够使用更高级的系统，这种数据结构甚至能返回当前系统中处理器的个数。

```

Option Explicit
Private Type SYSTEM_INFO
    dwOemID As Long
    dwPageSize As Long
    lpMinimumApplicationAddress As Long
    lpMaximumApplicationAddress As Long
    dwActiveProcessorMask As Long
    dwNumberOfProcessors As Long

```

```

        dwProcessorType As Long
        dwAllocationGranularity As Long
        dwReserved As Long
    End Type

    Private Declare Sub GetSystemInfo Lib"kernel32" ( LpSystemInfo As
SYSTEM_INFO )

    Private Sub Form_Click()
        Dim Sys As SYSTEM_INFO
        GetSystemInfo Sys
        Print"Processor type:";Sys.dwProcessorType
        Print"No. Processors:";Sys.dwNumberOfProcessors
    End Sub

```

如图 11.3 所示的是该窗体被单击时的输出实例。

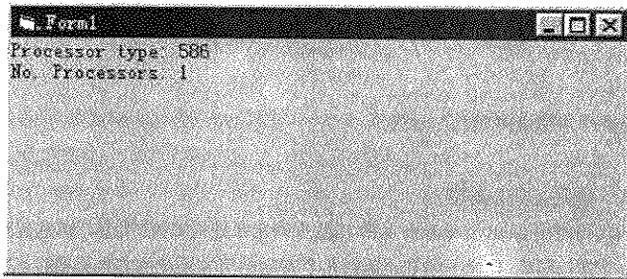


图 11.3 CPU 类型的确定

11.2.2 确定操作系统的版本

过去只能通过 Windows API 获取大量系统信息，现在可以通过 SysInfo 控件获取。这个控件可以提供系统版本、平台和系统事件信息。SysInfo 可以用于编写代码以处理 Windows 9X 和 Windows NT 之间的不同。它对于当 PCMCIA 卡插入便携型计算机时发生的即插即用事件也是非常有用的。

要使用 SysInfo 控件，可在 Components 对话框(该对话框可以在 Project 菜单中打开)中选择 Microsoft SysInfo Control 复选框。一旦设置了 SysInfo 控件的引用，该控件就会显示在 Toolbox 中，并且可以把它放到窗体中。下列代码展示了如何使用 SysInfo 控件显示操

作系统及其版本号:

```
Option Explicit
Private Sub Form_Click()
    Dim sMsg As String
    Select Case SysInfo1.OSPlatform
        Case 0
            sMsg = "Unidentified"
        Case 1
            If SysInfo1.OSVersion = 4# Then
                sMsg = "Windows 95,version" & CStr(SysInfo1.OSVersion)
            ElseIf SysInfo1.OSVersion = 4.1 Then
                sMsg = "Windows 98,version" & CStr(SysInfo1.OSVersion)
            ' Windows 95 版本为 4.0
            ' Windows 98 版本为 4.1
            End If
        Case 2
            sMsg = "Windows NT,version" & CStr(SysInfo1.OSVersion)
    End Select
    Print sMsg
End Sub
```

如图 11.4 所示的是这个窗体被单击后的输出实例。

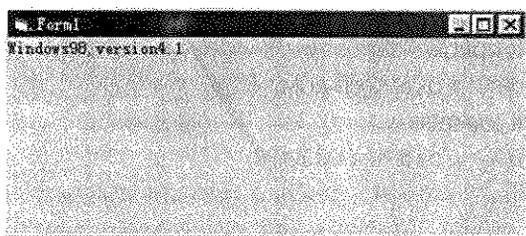


图 11.4 SysInfo 控件利用 SysInfo 控件显示操作系统及其版本号

11.2.3 确定驱动器类型

确定用户的计算机有几个软驱、几个硬驱或连接到几个远程驱动器是比较容易的，可以通过调用 GetDriveType Windows API 函数来完成。例如，在为特定数据文件搜索所有可用驱动器的程序中使用这一函数。在下面的代码中，GetDriveType 函数用来检测系统当前

所拥有的所有驱动器:

```
Option Explicit
Const DRIVE_REMOVABLE = 2
Const DRIVE_FIXED = 3
Const DRIVE_REMOTE = 4
Const DRIVE_CDROM = 5
Const DRIVE_RAMDISK = 6
```

```
Private Declare Function GetDriveType Lib "kernel32" Alias "GetDriveTypeA"
ByVal nDrive As String ) As Long
Private Sub form_click()
    Dim i, Drv, D$
    For i = 0 To 25
        D$ = Chr$(i + 65) & ":"
        Drv = GetDriveType(D$)
        Select Case Drv
            Case DRIVE_REMOVABLE
                Print "Drive" & D$ & "is removable."
            Case DRIVE_FIXED
                Print "Drive" & D$ & "is fixed."
            Case DRIVE_REMOTE
                Print "Drive" & D$ & "is remote."
            Case DRIVE_CDROM
                Print "Drive" & D$ & "is CD-ROM."
            Case DRIVE_RAMDISK
                Print "Drive" & D$ & "is RAM disk."
            Case Else
            End Select
    Next i
End Sub
```

当该窗体被单击时会显示可用驱动器列表, 如图 11.5 所示。

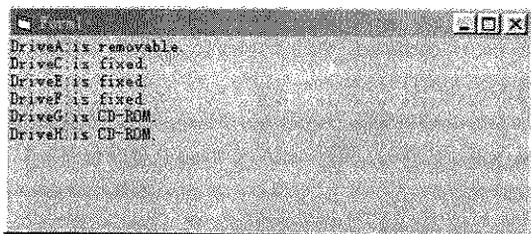


图 11.5 驱动器列表

11.2.4 确定消逝时间

GetTickCount Windows API 函数返回自 Windows 启动以来消逝的毫秒数。根据用户的系统，这个函数返回的值精度比 Visual Basic 的 Timer 函数高。Timer 函数返回自午夜以来的秒数，而且提供几乎是精确到毫秒的小数，但实际上 Visual Basic 每秒对 Timer 的返回值只更新 182 次。

和 Timer 函数相比，GetTickCount 的另一个优点是经过午夜边界也不会有问题。Timer 的返回值在午夜清 0，而 GetTickCount 只在计算机持续操作 497 天才返回 0。在下列代码中，当窗体被单击时，GetTickCount API 函数被调用，并且以毫秒方式在窗体上显示自 Windows 启动后消逝的时间。这一过程被重复 9 次以上，以展示 GetTickCount 返回值被更新的频度。

```

Option Explicit
Private Declare Function GetTickCount
Lib"kernel32"()As Long
Private Sub Form_Click()
    Dim i,j
    Print "Time elapsed since Windows was started:"
    For i=1 To 10
        j=GetTickCount
        Do While j=GetTickCount
            Loop
        Print j; "milliseconds"
    Next i
End Sub

```

如图 11.6 所示的是当该窗体被单击时的输出实例。

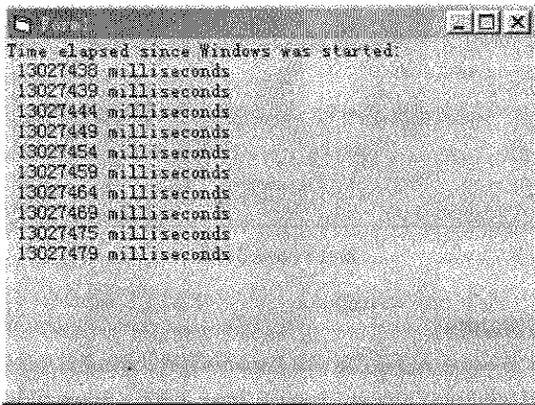


图 11.6 消逝的时间

11.3 屏幕保护程序

利用 Visual Basic 创建屏幕保护程序非常容易。

最初建立屏幕保护程序是为了保护屏幕，而现在屏幕保护程序可能主要用于娱乐而不是屏幕保护。但随时利用各种颜色切换所有像素位置这一基本原则仍是设计的出发点。为此，屏幕保护程序应该显示总在移动和变化的图形。

任何 Visual Basic 应用程序都可以作为屏幕保护程序来运行。当然，为了这个目的，有些程序将会比其他程序工作得更好。基本屏幕保护程序非常简单，包括一个占据整个屏幕的窗体、创建移动和变化图形的代码，以及当用户执行某个操作后终止程序的代码。这是创建简单屏幕保护程序所必需的。

要使用户的应用程序成为 Windows 环境下的屏幕保护程序，需要将它作为屏幕保护程序编译。要完成这项工作，需要从“文件”菜单中选择“生成...exe”，并作如下改动：不创建带有扩展名为.exe的可执行文件，而是在文件名文本框中键入扩展名为.scr来代替.exe。单击确定按钮，完成对可执行屏幕保护程序的编译。将生成的.scr文件复制到Windows目录以便能够找到它，然后将其设置为Windows屏幕保护程序。

注意：如果为屏幕保护程序提供一个以两个大写字母S开头的名字，Windows将会去掉这两个S，SSaver1.scr在屏幕保护程序对话框中显示为“aver1”，这是由于Windows早期版本处理屏幕保护程序的方式造成的。

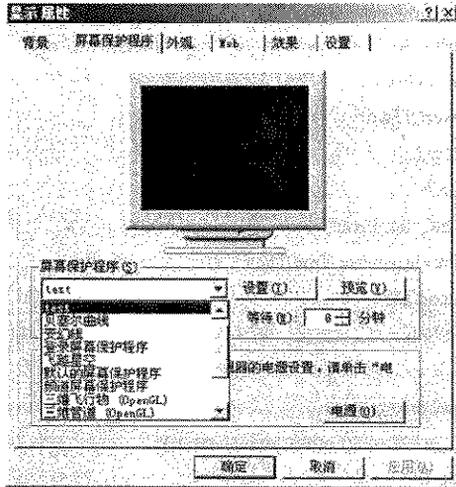


图 11.7 屏幕保护的设置

Option Explicit

```

Private Declare Function ShowCursor Lib "user32" (ByVal bShow As Long) As Long
Private Declare Function GetDesktopWindow Lib "user32" () As Long
Private Declare Function BitBlt Lib "gdi32" (ByVal hDestDC As Long, ByVal x As Long, ByVal Y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal XSrc As Long, ByVal YSrc As Long, ByVal dwRop As Long) As Long
Private Declare Function GetDC Lib "user32" (ByVal hwnd As Long) As Long
Private Declare Function ReleaseDC Lib "user32" (ByVal hwnd As Long, ByVal hdc As Long) As Long
Dim QuitFlag As Boolean
Private Sub Form_Click()
    QuitFlag = True
End Sub
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    QuitFlag = True
End Sub
Private Sub Form_Load()
    Dim x As Long, Y As Long

```

```
Dim XSrc As Long, YSrc As Long
Dim dwRop As Long, hwndSrc As Long, hSrcDC As Long
Dim Res As Long
Dim PowerOfTwo
Dim m1, m2
Dim n1, n2
Dim PixelColor, PixelCount
If App.PrevInstance = True Then
    Unload Me
    Exit Sub
End If
x = ShowCursor(False)
Select Case UCase$(Left$(Command$, 2))
    Case "/S"
        Randomize
        ScaleMode = vbPixels
        Move 0, 0, Screen.Width + 1, Screen.Height + 1
        dwRop = &HCC0020
        hwndSrc = GetDesktopWindow()
        hSrcDC = GetDC(hwndSrc)
        Res = BitBlt(hdc, 0, 0, ScaleWidth, ScaleHeight, hSrcDC, 0, 0, dwRop)
        Res = ReleaseDC(hwndSrc, hSrcDC)
        Show
        PowerOfTwo = 128
        Do
            Scale (0, 0)-(PowerOfTwo, PowerOfTwo)
            PixelColor = (PixelColor * 9 + 7) Mod 16
            PixelCount = 0
            m1 = Int(Rnd * (PowerOfTwo \ 4)) * 4 + 1
            m2 = Int(Rnd * (PowerOfTwo \ 4)) * 4 + 1
            n1 = Int(Rnd * (PowerOfTwo \ 2)) * 2 + 1
            n2 = Int(Rnd * (PowerOfTwo \ 2)) * 2 + 1
            Do
                x = (x * m1 + n1) Mod PowerOfTwo
```

```

        If x <> 0 Then
            Y = (Y * m2 + n2) Mod PowerOfTwo
        Else
            DoEvents
        End If
        Line (x, Y)-(x + 1, Y + 1), QBColor(PixelColor), BF
        PixelCount = PixelCount + 1
        If QuitFlag = True Then Exit Do
        Loop Until PixelCount = PowerOfTwo * PowerOfTwo
        PowerOfTwo = 2 ^ (Int(Rnd * 5) + 2)
    Loop Until QuitFlag = True
    tmrExitNotify.Enabled = True
Case Else
    Unload Me
Exit Sub
End Select
End Sub
Private Sub Form_MouseMove(Button As Integer, Shift As Integer, x As Single, Y As Single)
    Static Xlast, Ylast
    Dim Xnow As Single
    Dim Ynow As Single
    Xnow = x
    Ynow = Y
    If Xlast = 0 And Ylast = 0 Then
        Xlast = Xnow
        Ylast = Ynow
    Exit Sub
    End If
    If Xnow <> Xlast Or Ynow <> Ylast Then
        QuitFlag = True
    End If
End Sub
Private Sub Form_Unload(Cancel As Integer)

```

```
Dim x
x = ShowCursor(True)
End Sub
Private Sub tmrExitNotify_Timer()
Unload Me
End Sub
```

Form_Load 事件过程中的图形更新循环负责绘制实心彩色矩形块，它可使显示逐渐分解为指定的颜色。

第十二章 用部件编程

用户需要 Visual Basic 应用程序具有 Microsoft Excel 一样的分析与计算功能吗？或者，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。更理想的，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？

所有这些功能，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，就可以实现。ActiveX 部件是一段可重复使用的编程代码和数据，它是由 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，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。



12.1 部 件

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：

① 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。

② 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。与 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。

③ 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，如在窗体上显示日历牌，用特定的格式读取数据等。

④ ActiveX 文档可创建交互式的 Internet 应用程序，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 文档还可以有像代码部件那样的功能。

注意：除了使用 ActiveX 文档的 Internet 应用程序外，也可以用 Visual Basic 代码和 HTML 页面的组合创建基于客户和服务器的 Internet 应用程序。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。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。

12.2 创建对部件对象的引用

12.2.1 创建引用

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：

① 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。

② 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 从部件外而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。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。

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中。
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。

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。

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。

6. 用 Set 语句中的 New 关键字、CreateObject 或 GetObject 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。

2. 在 Set 语句中使用 CreateObject 或 GetObject 给变量赋值对象引用。

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。

12.2.2 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。

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,如表 12-1 所示,其格式为 component.class。

表 12-1 class 参数

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被赋值对象引用。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，应声明一般的 `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 用事前绑定的，这样能加速对象引用。

12.2.3 变量赋值及使用

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。可以用下面的方法赋值一个新的对象引用：

①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。

②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。

③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；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"
```

• 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，而且要访问的对象必须是外部可创建的。要用 CreateObject 赋值对象引用，应对 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")
```

• 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。表 12-2 描述了使用 GetObject 的结果。

表 12-2 GetObject 的结果

如果 ActiveX 部件正在运行	结果
Set X = GetObject("MySrvr.Application")X Set X = GetObject("", "MySrvr.Object")X	引用一个现存的 Application 对象 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") Set X = GetObject("", "MySrvr.Object")	返回一个错误 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 wrdApp As Word.Application
Set wrdApp = 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 wrdApp 将引用那个实例。

注意：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")
```



12.3 对象的方法、属性和事件

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：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，引用被其他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 Windows 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, 并使用 ActiveX 的应用程序之间共享该数据, 应使用 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。

• 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。
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.WorkbookVisual Basic
```

把对象变量名加到代码窗口的“对象”框中。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, 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

```



12.4 释放部件

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。
```

第十三章 数据库开发

Visual Basic 最大的优势在于它是一种通用性很强的程序设计语言，无论通信、数据库、多媒体、工具程序甚至 Internet，都可以使用 Visual Basic。

正所谓“成也萧何，败也萧何”，也正是这个通用性，为 Visual Basic 招来了不少诸如“博而不精”、“什么都能做，但是什么都做得不理想”这样的指责。但是争论归争论，无可争辩的是 Visual Basic 可以用最快的速度编写出质量很不错的应用程序，而且容易掌握。例如，使用 Visual Basic 可以让从未接触数据库的用户一行程序也不写，就创建一个可以浏览、修改数据库的应用程序来，仅此一点，Visual Basic 就足以傲视诸多的数据库开发工具的程序设计语言。

Visual Basic 提供了为数不少的数据库工具，例如 Data 控件、数据绑定控件、数据访问对象、远程数据对象和远程数据控件。但是，其中有些工具在专业版和企业版中才可以使用。

对于桌面应用程序，利用数据控件和数据绑定控件只要做很少的编程或根本不必编程就可以使用，还可以很方便地进行数据的定位、添加、删除和修改等工作。

13.1 了解数据库

在深入 Visual Basic 的数据库应用程序设计之前，首先要明确一些有关数据库的概念。由于 Visual Basic 所能识别的外部数据库多数在设计时是不相关的，所以 Visual Basic 须把外部数据库结构转换为关系模型，这些转换都是自动进行的。

关系数据库是 E.F.Cold 在 20 世纪 70 年代早期发明的一种数据库，从原理上讲，它是存储由列和行数据组成的表格的一种数据库。在 Visual Basic 中，把列称为字段，而行称为记录。

Microsoft Jet 数据引擎给予 Data 控件有将所有数据看作关系表集合的能力，而不管它们物理上的文件格式如何。这意味着从外部数据库(如 FoxPro、Parados、dBASE、Btrieve、Excel、Lotus 1-2-3、文本文件和 ODBC)调用数据时也能用相同的术语。下面对一些会经常用到的术语进行简单地解释。

· 表

表是一种按行与列排列的相关信息的逻辑组，类似于日常生活中常见的表格。例如，一张表可能包含有关一个人的一系列信息，诸如姓名、生日、地址和相片等。

· 字段

数据库表中的第一列称作一个字段。表是由其包含的各种字段定义的，每个字段描述了它所包含的数据。创建一个数据库时，为每个字段分配一个数据类型、最大长度和其他属性。字段可包含各种字符、数字甚至图形。

· 记录

有关的信息存放在表的行，被称为记录。一般来说，数据库表的记录创建时任意两行都不能相同。也就是说，不能有两个字段内容完全相同的记录。

· 索引

为更快地访问数据，大多数数据库都使用索引。数据库表的索引是表搜索的排序列表。每个索引输入项指向其相关的数据库行。如果数据库在寻找记录时先浏览索引，其工作将变得更容易且数据返回得更快。当与 Data 控件一起使用表类型的记录集时，表的主索引可用于加速检索操作。

· 结构化查询语言(SQL)

SQL 是一种数据库编程语言，它的起源与关系数据库紧密相联。现代的 SQL 已经发展成为关系数据库所广泛使用的标准，并且被 ANSI 标准所定义。一旦数据被存入数据库中，可用类似英语的语言进行查询，称为结构化查询。

数据库引擎通过返回满足查询要求的任何数据库行来“回答”提问。查询通常包含要搜索的各种表的名称，要返回的列的名称和其他设定此次搜索范围的信息。例如，一个 SQL 查询是这样的：

```
“Select Name,picture from Authors where Birth<#11/12/1974#”
```

这个 SQL 查询可返回所有在 Authors 表中，而且生日在 1974 年 11 月 12 日以前的作者的姓名和相片。对于返回的行，可用绑定控件显示它们的值。

现在的 SQL 的前身是 Sequel 语言，因此 SQL 的发音通常是 sequel，而不是 esscuel，虽然这两种发音都是可以接受的。

13.2 用 Data 控件可以做什么

Data 控件给数据库文件提供了一个关系界面，它可以用来显示、编程和更新各类已有的数据库中的信息，其中包括 Microsoft Access、Btrieve、dBASE、Microsoft FoxPro 和 Paradox。Data 控件还可以用来访问 Microsoft Excel、Lotus 1-2-3 和标准的文本文件，就像

它们是真的数据库，Data 控件还能够访问和操作完全的开放式数据库互连(ODBC)数据库，如 Microsoft SQL Server 和 Oracle。

Data 控件通过 Microsoft Jet 数据库引擎实现数据访问，同样的数据库引擎也给 Microsoft Access 以动力。该技术能天衣无缝地访问许多标准数据库格式，并可创建数据识别的应用而不必编写任何代码。

不用编写任何代码就能用 Data 控件创建简单的数据库应用程序，该程序将具有下面这些功能：

- ①与本地远程数据库连接。
- ②基于该数据库里各种表的 SQL 查询，打开指定的数据库表或定义记录集。
- ③传送数据字段到各种绑定控件，在其中可显示或改变数据字段的值。
- ④根据显示于绑定控件里的数据变化，添加新记录或更新数据库。
- ⑤捕获访问数据时出现的错误。
- ⑥关闭数据库。

与其他 Visual Basic 控件一样，要创建数据库应用程序，可以在窗体中添加 Data 控件。程序员可以根据需要在窗体中随意地创建多个 Data 控件，但是对每个要操作的数据库表使用一个 Data 控件为宜。



13.3 怎样使用 Data 控件

使用 Data 控件创建一个简单的数据库应用程序，要经过下面四个步骤：

1. 把 Data 控件添加到窗体中。
2. 设置其属性以指明要从哪个数据库和表中获取信息。
3. 添加各种绑定控件，如各种文本框、列表框和绑定到 Data 控件的其他控件。
4. 设置绑定控件的属性以指明要显示的数据源和数据字段。当运行应用程序时，这些数据绑定控件会自动地显示数据库当前记录的各个字段。

下面的过程将介绍如何在 Visual Basic 程序中使用 Data 控件，为了方便读者能够跟随创建应用程序的过程，将使用 Visual Basic 提供的 Biblio.mdb 示例数据库。这个数据库由一系列阐述关系数据库模型的图书列表组成，它存放在 Visual Basic 的安装目录下。

在应用程序里使用 Data 控件调用，要经过以下步骤：

1. 在“工具箱”中选择 Data 控件按钮，然后在窗体上绘制一个 Data 控件。

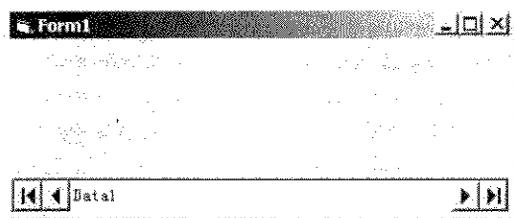


图 13.1 窗体中的 Data 控件

2. 在“属性”窗口中，设置 Connect 属性为要用的数据库类型。如果要连接 Access 数据库，则不必设置该属性。

3. 在“属性”窗口中，设置 Database Name 属性为要连接的数据库的文件名或目录名。若在设计时没有设置 DatabaseName 属性，则须在运行时写入 DatabaseName 和 RecordSource 属性。本例使用了 Bilio.mdb 示例数据库。

4. 设置 RecordSource 属性为要访问的数据库表的名称。

若数据库当前正有效，可从“属性”窗口中的下拉列表表中选择一个表。若找不到数据库，下拉列表将不会出现在 RecordSource 框里并显示一个错误信息，本例使用了 Titles 表。

5. 使用按钮在窗体里绘制文本框或显示数据库信息，控件将用于从数据库中显示和编辑一个选定的字段。也可用别的数据绑定控件，包括复选框、图形框、图像控件、标签、列表框和组合框等。

6. 使用按钮添加一个标签，把它的 Caption 属性设为该文本框将要显示的数据库字段的名称“书名：”。

7. 在“属性”窗口中，把文本框 Text1 的 DataSource 属性设置 Data 控件的名称“(Data1)”，这就使文本框绑定到 Data 控件。

8. 把 Text1 的 DataField 属性设为要修改的数据库表的字段名。此例用 Titles 表的 Title 字段。

9. 为每个要访问的另外的字段，重复步骤 5、6、7 和 8。在本例中，从 Titles 表选取了 Title、ISBN 和 Year3 个字段。

10. 单击工具栏中的“启动”按钮，开始运行应用程序。可用 Data 控件的四个方向按钮移动。

- 移动到第一条记录。
- 移动到前一条记录。
- 移动到下一条记录。
- 移动到最后一条记录。

通过改变显示在任何绑定控件中的值，能修改数据库的信息。当单击 Data 控件的箭头按钮向新记录移动时，Visual Basic 会自动地保存对数据库所做的任何改变。

程序员也能够运行时设置或改变 Data 控件及绑定控件的属性。只要设置正确的属性, 就可用 Data 控件的 refresh 方法来(在必要时)重新打开数据库及重新创建具备新属性设置值的记录集。

13.4 记录集

Visual Basic 使用 Microsoft Jet 数据库引擎提供的 Recordset 对象来检索和显示数据库记录, 一个 Recordset 对象代表一个库表里的记录, 或运行一次查询所得的记录的结果。表 13-1 列出了在 Data 控件中可用的 3 类 Recordset 对象。

表 13-1 Data 控件中可用的三类 Recordset 对象

对 象	描 述
表类型的 Recordset	(dbOpenTable)一个记录集合, 代表能用来添加、更新或删除记录的单个数据库表。
Dynaset 类型的 Recordset	(dbOpenDynaset)一个记录的动态集合, 代表了一个数据库表或包含一个或多个表取出的字段的查询结果。可从 dynaset 类型的 Recordset 中添加、更新或删除记录, 并且任何改变都将会反映在基本表上
快照类型 Recordset	(dbOpenSnapshot)一个记录的静态集合副本, 可用于寻找数据或生成报告。一个快照类型的 Recordset 能包含从一个或多个在同一数据库中的表中取出的字段, 但字段不能更改。

用 Data 控件的 Recordset Type 属性, 能选取要使用的 Recordset 对象的类型, 其默认值为 vbDynasetType。

动态类型和快照类型的记录集都存储于本地内存中。如果不需要应用程序从多个表中选择字段及操作一个非 ODBC 源, 那么表类型的 Recordset 在速度与内存占用方面及本地 TEMP 磁盘空间方面可能是最为有效的。

使用 Set 语句在代码中创建(在 Visual Basic 学习版中不能这么用)的 Recordset 对象能赋值给 Data 控件的 Recordset 属性, 例如:

```
Set Data1.Recordset=MyRecordset
```

同样, 一个 Data 控件创建的 Recordset 对象, 在运行时能赋值给另一个 Data 控件。Jet 数据库引擎提供了大量的数据库和记录集的属性和方法。通过引用 Data 控件的 Database 和 Recordset 属性, 可以直接与 Data 控件一起使用这些属性和方法。

13.5 使用数据绑定控件

所谓数据绑定控件就是数据识别控件, 在 Visual Basic 的数据库应用程序中要通过数据绑定控件来访问信息。当一个控件被绑定到 Data 控件时, Visual Basic 会把从当前数据

库记录取出的字段值应用于该控件，然后控件显示数据并接受更改。如果在绑定控件中改变数据，当移动到另一个记录时，这些改变会自动地写入数据库中。

Visual Basic 支持几种可绑定到 Data 控件的内部控件，来自第三方及在 Visual Basic 专业版与企业版的其他数据识别控件都是可用的。可以与 Data 控件一起使用的标准绑定控件包括以下几种：

- 复选框
- 图像
- 标签
- 图片框
- 文本框
- 列表框
- 组合框
- OLE 容器控件

除了内部的绑定控件外，Visual Basic 也提供下列可添加到工程中的 ActiveX 控件：

- DBListBox
- DBCComboBox
- MSFLexGrid
- ApexDBGrid
- MaskedEdit(在学习版中不可用)

对于复杂的界面，有一些 ActiveX 控件提供了附加功能显示整组的记录，如列表、表格或者整个记录集。它们包括列表框、组合框以及 MSFLexGrid 控件。

另外，有 3 种特殊的控件允许完成复杂数据库显示以及对多记录集和字段的更新任务，它们是数据绑定列表控件、组合框控件以及 DBGrid 控件，要在工具箱中添加这三个控件，应使用“工程”→“部件”命令，然后从“部件”对话框中选中“Microsoft Data Bound Grid Control”和“Microsoft Data Bound List Controls 5.0”两个选项。

在使用任何 ActiveX 控件之前，需要从“工程”菜单中使用“部件”命令来添加该控件，还必须使用“引用”对话框来注册“Microsoft DAO 3.5 Object Library”。

大多数绑定控件是以三种数据识别属性为特征的——DataChanged、DataField 和 DataSource。如表 13-2 所示。

表 13-2 三种数据识别属性

属 性	描 述
DataChanged	指示显示于绑定控件里的值是否已经改变
DataField	指定 Data 控件所建立的记录集里字段的名称
DataSource	指定 Data 控件所要绑定的控件名称

通常，把绑定控件添加到应用程序的步骤是这样的：

1. 在要绑定 Data 控件的同一窗体中绘制一个绑定控件。
2. 设置 DataSource 属性, 指定要绑定的 Data 控件。
3. 设置 DataField 属性为 Data 控件的记录集中的一个有效字段。

如果设计时数据库可用, 则有效字段的列表将显示在“属性”窗口里的“DataField”框中。如果在设计时数据库不可用, 则在数据值从数据库中发送给控件之前, 须在运行时提供一个有效的字段名称。

对特定字段可有多个绑定控件, 但不必为表中的每个字段都提供一个绑定控件。Data 控件和绑定控件不一定要设为可视的。因此, 能把数据访问的能力合并到设计的窗体中, 以使用 Visual Basic 代码在“幕后”操纵数据。

运行程序时, Data 控件与数据库一起工作, 以访问当前记录集或正在使用的记录集使用 Data 控件的箭头按钮可在记录间移动, 而用绑定控件可查看或编辑从每个字段中显示出来的数据。无论何时单击 Data 控件的按钮, Visual Basic 会自动地更新记录集所做的任何改变。

13.6 添加新记录

一旦确定数据库和记录集可被改变, 就可开始添加记录。把新记录添加到记录集中, 请按以下步骤执行:

1. 用 AddNew 方法创建一条新记录(空的)。当前记录指针已保存并移动到该新记录。
2. 在该新记录中给各字段指定新值。
3. 用 Update 方法保存新记录。当前记录指针恢复为原值(记录指针的值优先于使用 AddNew 方法)。

下面的代码将在 Biblio.mdb 数据库的 Titles 表格中添加一个新记录:

```
Data1.DatabaseName = "Biblio.mdb"  
Data1.RecordSource = "Titles"  
Data1.Refresh  
Data1.Recordset.AddNew  
Data1.Recordset("Title") = "The Data Control"  
Data1.Recordset("Year Published") = "1993"  
Data1.Recordset("AU_ID") = 37  
Data1.Recordset("ISBN") = "25444789873"  
Data1.Recordset("PubID") = 34  
Data1.Recordset.Update
```

在前述的代码示例中，PubID 指的是 Publishers 表格中的 PubID 字段。代码须核实正确值是为了维护数据库参考资料的完整性。

• 编辑当前记录

要改变数据库的数据，必须先把要编辑的记录设为当前记录，然后在绑定控件中完成任意的改变。要保存此改变，只须把当前记录指针移到其他记录上，或者使用在以下过程代码示例中所示的 Update 方法。编辑当前记录的字段值，按以下步骤执行：

1. 把当前记录定位到要编辑的记录上。
2. 给要改变的字段指定新的值。
3. 使用 Update、Move、Find 或 Seek 方法中的任何一种。

或者在 Data 控件上单击一个箭头按钮来保存改变并替换已有的各种字段值。

下面的代码展示了如何在第一条记录中编辑 PubID 字段值。

```
Data1.DatabaseName = "Biblio.mdb"
Data1.RecordSource = "Titles"
Data1.Refresh
Data1.Recordset("PubID") = "132456"
Data1.Recordset.Update
```

• 数据访问专用的 Update 方法

Update 方法最常应用在更新记录集中基于通过 Data 控件或者通过代码所做改变的信息。除 Update 方法以外，还有另外三种可以用于更新信息的方法：UpdateRecord、UpdateControls 和 Refresh。表 13-3 总结了这三种方法。

表 13-3 其他三种可以用于更新信息的方法

方 法	描 述
UpdateRecord	用绑定控件中的数据更新数据库(记录集)
UpdateControls	更新绑定控件的数据库(记录集)的变化
Refresh	基于 Data 控件属性创建一个新的记录集

• 确认数据库的变化

在向数据库中写入新信息以前，Data 控件的 Validate 事件可核对对记录集所做的改变，同时也可指定在 Validate 事件完成以后哪一条记录成为当前记录。除了由 UpdateRecord 方法改变的以外，当前记录行被改变时，Validate 将被引发。这意味着不论是否已改变了绑定控件中的数据，Validate 都可被引发。

在 Validate 事件期间，不能引用能引发其他 Validate 事件的任何方法。例如，不能调用 AddNew 或者任何重定位方法(Move、Find 与 Seek)。

仅当 Visual Basic 把改变从绑定控件中写入数据库以及把当前记录指针重新定位到数据库的其他记录行之前，Validate 事件被引发。

Validate 事件的 Save 和 Action 参数进行不同的设置，可以达到不同的目的：

• Save 参数

在 Validate 事件中, 可通过检查 Save 参数来确定是不是有绑定控件已经改变。Visual Basic 自动核查每一个绑定控件的 Changed 属性是否自从最后一个数据库动作设置以来它的值已经改变。如果有任何值已经改变, 则 Visual Basic 会把 Save 参数设置为 True。

如果 Save 参数设置为 True, 则 Visual Basic 将保存数据库的所有绑定控件。如果不保存改变, 可把 Save 参数设为 False。

• Action 参数

Action 参数说明是什么引发 Validate 事件, 并且在 Validate 事件处理完成以后重新定位当前记录的指针。在 Validate 事件中, Visual Basic 设置 Action 参数的值以表明哪个动作引发该事件, 可以在对象浏览器中找到 Action 参数的常数数据。表 13-7 总结了 Action 参数值以及引起 Validate 事件的相应动作。

表 13-4 Action 参数值、Validate 事件的相应动作

常 量	值	描 述
vbDataActionCancel	0	取消引起事件的动作
vbDataActionMoveFirst	1	MoveFirst 方法
vbDataActionMovePrevious	2	MovePrevious 方法
vbDataActionMoveNext	3	MoveNext 方法
vbDataActionMoveLast	4	MoveLast 方法
vbDataActionAddNew	5	AddNew 方法
vbDataActionUpdate	6	Update 方法
vbDataActionDelete	7	Delete 方法
vbDataActionFind	8	Find 方法
vbDataActionBookmark	9	已经设置了 BookMark 属性
vbDataActionClose	10	Close 方法
VbDataActionUnload	11	窗体被卸载

在有些情况下, 可以设置 Action 参数来指定完成事件以后 Visual Basic 将如何重新定位当前行的指针。如果 Validate 事件已经由 AddNew 或者一种 Move 方法引发, 就能进行这种指定。

例如, 假设由于单击 Data 控件上的 MoveNext 按钮, 引发了 Validate 事件。当 Visual Basic 进入 Validate 事件时, 它设置 Action 参数为 3, 这表明是 MoveNext。当有效的检查完成后, 希望把当前记录指针重新定位到前一条记录而不是下一条记录。要做到这一点, 应将 Action 参数设置为 2, 这表明是 MovePrevious; 然后, 重定位程序将使用所指定的 Action 参数以表明在读事物之后把哪一行设置为前行。

可指定任意一个 Move 或 AddNew 方法代替另一个 Move 或 AddNew 方法去执行。除了各种 Move 方法或 AddNew 方法外, 如果试图改变为其他任何动作, Visual Basic 都会忽略这种尝试, 并且执行原来要做的操作。

如果不希望 Validate 事件把当前记录指针重新定位到其他记录上, 可把 Action 参数设置为 0。把 Action 参数设置为 0, 不会影响数据是否保存到数据库——它只取消重定位操

作以及离开活动当前记录。如果没有发生重定位,则各种值依旧显示在绑定控件中,且当前记录指针保持不变。

• 删除记录

要删除一整条记录,应将当前记录指针定位到删除的记录上,使用 Delete 方法。删除多条记录时,每次删除以后必须用 MoveNext 方法来改变当前记录,因为已删除的记录不再包含有效的数据,继续访问这一数据会导致错误。

• 关闭记录集

Close 方法关闭记录集并释放分配给它的资源。试图对已关闭的记录集执行方法或元素访问,会导致一个可捕获的运行时错误。下面的代码关闭一个记录集:

```
Data1.Recordset.Close
```

在下列情况中,数据库和它们各自的记录集自动关闭:

- ①针对一指定的记录集使用 Close 方法。
- ②卸载包含 Data 控件的窗体。
- ③程序执行一个 End 语句。

当使用 Close 方法或者窗体卸载时, Validate 事件被引发。在 Validate 事件中可执行最终地清除操作。

13.7 使用事务处理维护数据库的完整性

事务是对一个记录集所做的可恢复的一系列改变。在向数据库提交新信息之前,须要确认所做的任何改变,应使用事务。例如,如果正在处理一系列的金融事务,如果最后合计超出平衡,应取消各种改动,这时如果用人工去恢复大段的处理,很难保证不会再出错。事务处理的作用就是:控制数据库要么把整个事务都做完,要么整个事务都不做。

在 Visual Basic 中,程序员可以用代码显式地开始一个事务。在打开事务的同时,对数据所做的任何改变都可撤销,或者被退回。决定完成工作时,可以保存或者提交对数据库的改变。

当第一次打开数据库并且没有事务挂起时,事务状态是自动提交的,这意味着对记录集做的所有改变将立刻改变基本表格,且不可逆转。如果不是所要的情况,在变化将发生时,可用事务来控制。

Visual Basic 有三种支持事务处理的语句: BeginTrans、CommitTrans 与 Rollback。但是,数据库本身也必须支持事务;否则,这些命令被忽略。而且在使用这些语句之前要确保数据库 Transactions 属性设置为 True。

事务跨越在各数据库上,也就是说,当使用任意一个事务语句时,它适用于工作区中

所有的数据库——即使数据库是在事务开始以后打开的。使用 CommitTrans 或者 Rollback 时，不管数据库如何，所有它前面的事务都被提交或者被回退。

在 Jet 数据库引擎的当前版本中，BeginTrans、CommitTrans 和 Rollback 语句都是 Workspace 对象的方法。Data 控件适时地结合这一功能，当用 Data 控件时，这些语句自动地映射到打开的默认工作区的 Workspace 方法中。除非分配给它的记录集是在其他 Workspace 中创建的，否则 Data 控件总是使用 Workspaces(o)。

为了将来最大的兼容性，建议专业版或企业版的 Visual Basic 用户显式地使用 Workspace 方法(如 Workspace.BeginTrans)。

13.7.1 开始一个事务

BeginTrans 语句标志着—一个事务的开始并且执行来自自动提交模式的下一步操作序列。一旦开始一个事务，就必须关闭数据库或应用程序之前使用 CommitTrans 或 Rollback。当应用程序结束时，没有提交的事务要自动回退。如果试图在事务仍在运行时关闭一个数据库，就会产生一个错误。

13.7.2 保存修改的结果或者撤销修改

用 BeginTrans 命令打开事务以后，CommitTrans 命令将保存对记录集所做的所有改变。执行 CommitTrans 时，做的所有改变成为永久性的改变，当前事务结束，事务返回到自动提交状态。

而使用 Rollback 命令会倒退或者撤销当前事务中所有已改变的数据。它同时终止事务并把事务返回到自动提交状态。下面示例开始一个事务，如果用户确认，则改变所有 PubID=5 的记录为 PubID=6。如果没有确认，则终止此事务并撤销所有的改变。

```
Dim MyTable As Recordset, MyWorkspace As Workspace
Set MyWorkspace = Workspaces(0)
Set MyTable = Data1.Recordset
MyWorkspace.BeginTrans
Do Until MyTable.EOF
    If MyTable![PubID] = 5 Then
        MyTable.Edit
        MyTable![PubID] = 6
        MyTable.Update
    End If
    MyTable.MoveNext
Loop
```

```

If MsgBox("Save all changes?", vbQuestion + vbYesNo, "Save Change") = vbYes Then
    MyWorkspace.CommitTrans
Else
    MyWorkspace.Rollback
End If

```

13.7.3 使用多人事务

打开一个事务后，程序员仍然可以继续打开另一个，这就是多重事务。多重事务就像嵌套的控制结构，在进行最外面语句(或者事务)之前，需要关闭最里面的语句(或者事务)。

通常没有必要使用如此复杂的结构，然而如果正在使用多个事务，则需要注意发出的事务命令的先后顺序。

13.8 使用可视化数据管理器

通过选择“外接程序”菜单中的“可视化数据管理器”命令，可以访问可视化数据管理器如图 13.2 所示。这个外接程序是使用 Visual Basic 创建的，在专业版和企业版中可以在示例目录中找到它的通用代码，它的工程名是 VisData.Vbp。在这个程序中，包含了很好的数据库程序设计的思想和技巧。

使用可视化数据管理可以对一个数据库进行彻底的“透视”，从而了解这个数据库中到底包括多少表(Table)、多少字段(Field)和多少索引(Index)，如图 13.3 所示。

要“透视”一个数据库，方法很简单：只要在“可视化数据管理器”中，使用“文件/打开”命令打开一个数据库文件，就可以很清楚地看到该数据库的内部结构。

“可视化数据管理器”除了可以对数据库结构进行“透视”以外，还可以生成数据窗体，方法是：打开一个数据库文件后，选择“实用程序”菜单下的“数据窗体设计器”命令，就会显示一个“数据窗体设计器”对话框。程序员只须选中记录源，并设置窗体的名称，然后单击按钮将须要显示的字段名旋转到右侧的框中，最后单击“生成窗体”按钮(如图 13.4 所示)，就可以生成一个非常不错的数据窗体，其中还包含许多事件过程的代码，可以完成数据库的一般维护工作。

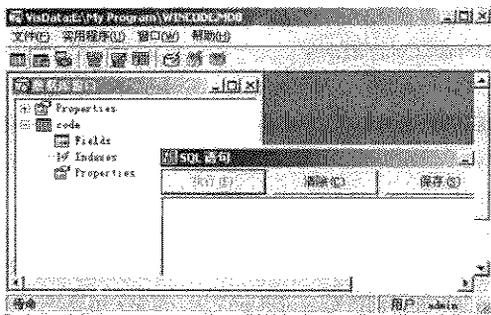


图 13.2 可视化数据管理器



图 13.3 数据库窗口

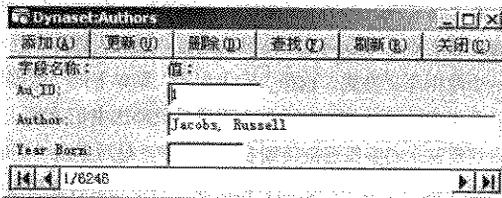


图 13.4 数据窗体

第十四章 Internet 开发

当人们谈及 Internet 应用程序时，由于类型太多，很难了解它们指的是什么。而 Visual Basic 为 Internet 的各个方面提供了专门的技术，因此在选择工具之前了解所需的应用程序类型时是很重要的。表 14-1 对 Internet 的各种用途进行了分类，并列出了完成各类任务所需的相应的 Visual Basic 工具。

表 14-1 Internet 各种用途的分类

Internet 任务	Visual Basic 工具
计算机间的直接通信	Winsock 控件 (MSWINSCK.OCX)
文件传输	Internet Transfer 控件 (MSINET.OCX)
Web 浏览	WebBrowser 控件 (SHDOCVW.DLL)
在 Microsoft Internet Explorer 中运行的应用程序的开发	ActiveX 文档
使用 Internet 安装程序发布应用程序	Setup Wizard
用于超文本标记语言(HTML)页面的组件开发	ActiveX 控件
数据库应用程序	ActiveX 文档，具有数据控件或使用 ADO 对象的 Active Server Pages
从服务器到客户机的事物/数据传输	Active Server Pages

14.1 Internet 控件

Visual Basic 专业版和企业版包含三个 ActiveX 控件，它们可以提供对每一层 Internet 通信的访问：

- WinSock 控件提供对用于 Internet 的传输控制协议(TCP)和用户数据报网络协议(UDP)地底层访问。WinSock 用于创建通话应用程序，并实现两台或多台联网计算机间的直接数据传输。

- Internet Transfer 控件允许从 WWW 和文件传输协议 (FTP) 服务器中拷贝文件，并允许以同步或异步方式浏览文件目录和检索数据。该控件过去常常用于创建 FTP 浏览和文件传输应用程序。

- WebBrowser 控件将所有的 Internet Explorer 功能打包到一个控件中。使用它可以将在 Internet 浏览器添加到一个现有的应用程序中，也可以从另一个应用程序中通过 Automation 控制 Internet Explorer。

注意：WebBrowser 控件在 Visual Basic Components 对话框中被称作 Microsoft Internet

Controls。它作为 Internet Explorer 安装程序的一部分来安装。如图 14.1 所示。



图 14.1 WebBrowser 控件和 Internet Transfer 控件

➡ 14.2 创建自己的浏览器

如果要创建一个简单的 FTP 浏览器，如图 14.2 所示，它使用两个文本框和一个 Internet Transfer 控件。可在 Address 文本框中输入 FTP 服务器的 URL(Uniform Resource Locator)，然后从内容文本框中选择一个文件或目录。如果选择目录，应用程序就会显示该目录。如果选择文件，浏览器就会将文件存于 Windows\Temp 目录中。

当用户按回车键时，Address 文本框会通过设置 Internet Transfer 控件的 URL 属性并调用 Execute 方法来执行请求。当请求一个特定文件时，用 OpenURL 方法实现相同的动作。然而，当使用 OpenURL 方法返回目录的内容时，会返回指明目录内容的 HTML 源代码。

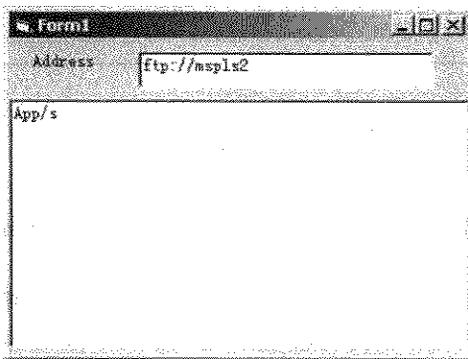


图 14.2 浏览器的界面

```
Private Sub txtAddress_KeyPress(KeyAscii As Integer)
    If KeyAscii = Asc(vbCr) Then
        'Eat keystroke
        KeyAscii = 0
        'Select text
        txtAddress.SelStart = 0
        txtAddress.SelLength = Len(txtAddress)
    End If
End Sub
```

```
On Error GoTo errOpenURL
'Set FTP address to view
inetBrowse.URL = txtAddress.Text
'Get directory
inetBrowse.Execute, "Dir"
'Display address
Caption = inetBrowse.URL

End If
Exit Sub
ErrOpenURL:
Select Case Err.Number
Case icBadUrl
    MsgBox "Bad address. Please reenter."
Case icConnectFailed,icConnectionAborted, icCannotConnect
    MsgBox "Unable to connect to network."
Case icExecuting
    'Cancel previous request
    inetBrowse.Cancel
    'Check whether cancel worked
    If inetBrowse.StillExecuting Then
        Caption = "Couldn't cancel request."
    'Resubmit current request
    Else
        Resume
    End If
Case Else
    Debug.Print Err.Number, Err.Description
End Select
End Sub
```

当提交一个请求时，设置错误陷阱很重要，尤其是 `icExecuting` 错误。Internet Transfer 控件采用异步方式处理所有请求。然而，在同一时间仅能处理一个请求。如果取消正在进行的请求，在重新开始前一定要检查 `StillExecuting` 属性，如代码中所示。有些请求不能被取消，并且简单地使用 `Resume` 语句会导致死循环。

下面的代码显示了 `txtContents` 文本框的 `DbClick` 事件过程，在其中显示了目录清单。

该段代码用来建立 URL 字符串。如果选择子目录，则执行 Dir 命令；如果选择文件，则执行 Get 命令。

```
Private Sub txtcontents_DblClick()  
    'Browse selected directory  
    If txtContents.SelLength Then  
        'If selection is a directory...  
        If Right(txtContents.SelText, 1) = "/" Then  
            'Add selected item to address  
            txtAddress = txtAddress & "/" & Left(txtContents.SelText, TxtContents.  
SelLength - 1)  
            'Trap errors (important!)  
            On Error GoTo errBrowse  
            'Show directory  
            msDir = Right(txtAddress, Len(txtAddress)  
                Len(inetBrowse.URL)  
            InetBrowse.Execute , "Dir" & msDir & "/" * "  
            'Otherwise, it's a file, so retrieve it  
        Else  
            'Build the path name of the file  
            msDir = Right(txtAddress, Len(txtAddress)  
                Len(inetBrowse.URL)) & "/" &  
                TxtContents.SelText  
            MsDir = Right(msDir, Len(msDir)-1)  
            'Retrieve file  
            inetBrowse.Execute , "Get" & msDir &  
                "" & msTempDir & txtContents.SelText  
        End If  
    End If  
Exit Sub  
ErrBrowse:  
    If Err = icExecuting Then  
        'Cancel previous request  
        inetBrowse.Cancel  
        'Check whether cancel worked
```

```

If inetBrowse.StillExecuting Then
    Caption = "Couldn't cancel request."
'Resubmit current request
Else
    Resume
End If
Else
'Display error
Debug.Print Err & "" & Err.Description
End If
End Sub

```

Execute 和 OpenURL 方法都激发 StateChanged 事件。一定要记住这一点, 因为 OpenURL 是一种同步方法, 激发 StateChanged 事件有时会引起重入问题。

下面的代码用于修改窗体标题以使用户能够跟上请求过程。如果执行的命令是 Dir, 则可使用 GetChunk 方法检索目录清单。

```

Private Sub inetBrowse_StateChanged (ByVal State As Integer)
Select Case State
Case icError
Debug.Print inetBrowse.ResponseCode & "" & InetBrowse.ResponseInfo
Case icResolvingHost,icRequesting,icRequestSent
Caption = "Searching..."
Case icHostresolved
Caption = "Found."
Case icReceivingResponse,icResponseReceived
Caption = "Receiving data."
Case icResponseCompleted
Dim sBuffer As String
'Get Data
sBuffer = inetBrowse.GetChunk(1024)
'If data is a directory,display it
If sBuffer <> "" Then
Caption = "Completed."
TxtContents = sBuffer
Else

```

```

Caption = "File saved in" & MsTempDir & "."
End If
Case icConnecting, icConnected
Caption = "Connecting."
Case icDisconnecting
Case icDisconnected
Case Else
Debug.Print State
End Select
End Sub

```

注意：当前文档表明 GetChunk 将处理 Execute 方法的 Get 命令，但情况似乎并非如此。Get 命令的语法指明了源文件和目录文件，因此，当从服务器上拷贝文件时并不需要 GetChunk。

14.3 超文本标记语言

超文本标记语言(HTML)是建立发表在线文档所采用的语言，是在 Internet 上显示信息的标准。所有 HTML 页面的总体组成了 World Wide Web。

HTML 文件是文本文件，扩展名为.htm，它包含要先给用户的信息。信息中有许多格式标记，如文本显示的格式、文本字体的修饰(粗体、斜体)等。除文字外，HTML 文件中还可以包括图像、声音以及与其他页面的超级链接。

```

<HTML>
<HEAD>
  <TITLE>
  这里为示例的标题
</TITLE>
  你好，朋友！
</HEAD>
<BODY>
  <H1>1 级标题</H1>
  <H2>2 级标题</H2>
  <HR>一个简单的 HTML 示例
</BODY>

```

</HTML>

上面是最简单的一个 HTML 的例子(如图 14.3 所示), 你可以用“记事本”, 或者使用 DOS 的 Edit 等录入, 注意存盘时后缀为“.html”。

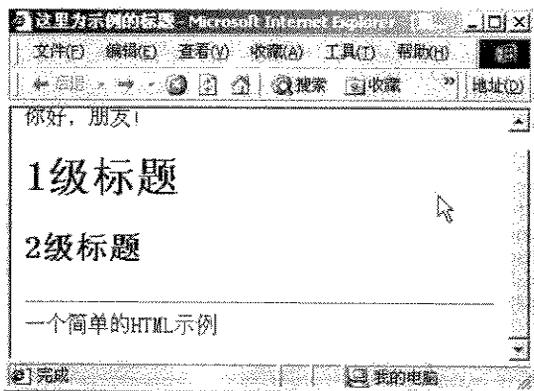


图 14.3 示例窗口

如果在示例中的<BODY>下面, 增加下面的内容指定字体大小和颜色, 则效果更佳。

```
<P><B><I><FONT COLOR = "#FF0080"><FONT SIZE = +3>
欢迎进入 Internet 世界
</FONT></FONT></I></B>
```

其中, 指定字体为黑体, <P>表示回车, <I>指定为斜体。

14.4 HTML 和 VBScript

在 HTML 实现超文本时有一定的局限性。在这种情况下, 就有许多用来加强超文本浏览能力的描述性语言, 目前比较流行的有 JavaScript 和 VBScript。

VBScript 提供用语言将 ActiveX 控件嵌入文档的能力, NetScape 浏览器和 Internet 浏览器都支持 VBScript 描述语言。

为了演示 VBScript, 可以打开 Windows 的“记事本”或 Winword 输入下面的文本, 如图 14.4 所示, 输入完毕后, 一定要将其保存为后缀为.htm 的文本文件。

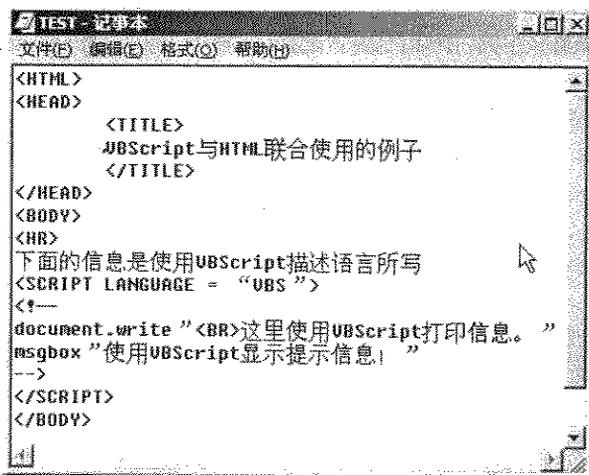


图 14.4 记事本中的示例

```

<HTML>
<HEAD>
    <TITLE>
    VBScript 与 HTML 联合使用的例子
    </TITLE>
</HEAD>
<BODY>
    <HR>
    下面的信息是使用 VBScript 描述语言所写
    <SCRIPT LANGUAGE = "VBS">
    <!--
    document.write "<BR>这里使用 VBScript 打印信息。"
    MsgBox "使用 VBScript 显示提示信息!"
    -->
    </SCRIPT>
</BODY>
</HTML>

```



图 14.5 VBScript 示例窗口

14.5 HTML 和 Active 文档的调用

创建基于 ActiveX 文档的工程，实际上相当于创建可以包含在 Internet Explorer 中的 Visual Basic “文档”。当编译 ActiveX 文档时，将创建 Visual Basic 文档文件，其扩展名为.VBD。设计 ActiveX 文档与设计 Visual Basic 程序大同小异，其中的每一个 ActiveX 文档页对应一个 Form 窗体，每个文档页可通过它所对应的.VBD 文件惟一确定，文档页之间可以互相调用。如果熟悉使用 Visual Basic 编程，则可以不必学习其他编程语言。

下面将介绍一个在 HTML 中可跳转 ActiveX 文档和其他.htm 文件的例子。首先单击“文件”菜单项，选择“新建工程”，然后在对话框中选择“ActiveX DocumentExe”，单击“确定”按钮。

接着，双击“工程资源管理器”中的“UserDocument”打开文档设计器。在工具箱中选择一个命令按钮(Command1)控件，将其放到文档设计器上，设置它的 Caption 属性为“测试”，并在 Command1_Click 中录入代码：

```
Msgbox “测试 ActiveX 文档”
```

单击“文件”菜单项，选择“另存为”命令，为 ActiveX 文档命名，然后单击“文件”菜单项，选择“生成.exe 文件”命令。

接下来可以打开 Windows 的“记事本”输入下面的文本，输入完毕以后，将其保存为后缀名为.htm 的文本文件。

```
<a href = "h2_vbs.htm">欢迎信息页面</a> 'h2_vbs.htm 是我们以前做的例子
<FROM NAME = "Login">
```

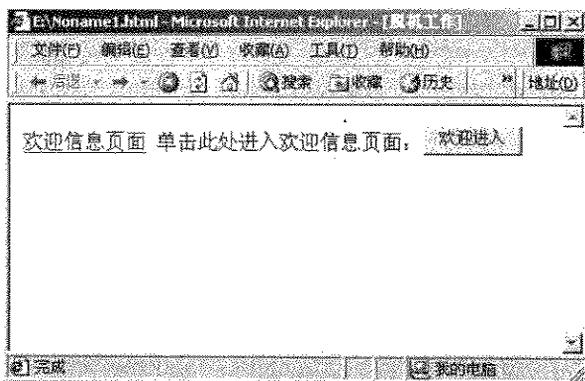


图 14.6 ActiveX 文档的调用

单击此处进入欢迎信息页面，如图 14.6 所示。

```

<INPUT NAME = "cmdlogin" TYPE = "Button" VALUE = "欢迎进入">
</FORM>
<OBJECT
    classid = "clsid:2F390484-1C7D-8908-00A0C90395F4"
    codebase = "userdocument.cab#version = 1,0,0,0">
</OBJECT>
<SCRIPTLANGUAGE = "VBScript">
Sub cmdLogin_onClick
Navigate "userdocument.vbd"
End Sub
</SCRIPT>

```

第十五章 帮助文件

15.1 创建帮助文件

如果应用程序拥有用户，必须知道怎样使用它并且最好的办法是采用联机帮助。市场上有一些很好的帮助软件制作工具，例如：RoboHelp、HelpBreeze 以及共享的 VBhelp。这些工具均是基于以前的 Visual Basic 版本产生的，那时 Microsoft 推出的惟一的帮助工具是 Help Compiler，近来事情多少有了些变化。

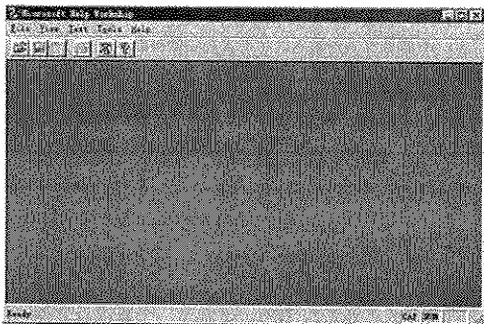


图 15.1 Help Workshop 界面

Visual Basic 包含一个改进的帮助工具如图 15.1 所示，如果它称为 Microsoft Help Workshop。这一工具单独放在 Visual Basic CD-ROM 的 COMMONVB\TOOLS\HCW 目录下，可以通过运行该路径下的安装程序来安装 Help Workshop。Help Workshop 最终提供一个图形用户界面，用它可以组织帮助项目以及编译、调试帮助文件。Help Workshop 并不是一个完全的制作环境，但它确实值得一看。

此外，值得一看的是由 32 位版本的 WinHelp 提供的帮助功能。帮助文件可以包括：目录表、全文本搜索功能、AVI 剪辑以及几个新的帮助宏。假如想选择一个附加的帮助工具，一定要选用最新版本的，这样就可以利用这些新的功能。

Help Workshop 建立一个帮助项目文件(*.HPJ)，它列出了用来建立帮助文件的源文档文件。源文档必须以多正文格式(.RTF)保存，并且能用 Microsoft Word 或其他支持所需格式的 Windows 字处理程序建立，Help Workshop 会启动编译器(HCRTF.EXE)，并在编译完

Help Workshop 项目窗口含有多个按钮，它们将显示对话框，并允许配置帮助文件，如果单击 Files 按钮，则可以将创建的任何主题文件添加到项目文件中。

15.1.3 创建帮助文件

• 编译和测试帮助文件

编译帮助文件是一个利用主题文件、图形和项目文件来创建帮助文件的过程，可以通过单击项目窗口中的“Save And Compile”按钮来编译帮助文件，它启动编译器(HCRTF.EXE)。编译后，将会显示错误或警告。要测试新的帮助文件，需单击“Help Workshop”中的“Run WiHelp”按钮，然后单击“View Help File”对话框中的“View Help”按钮。

• 映射主题 IDs

在帮助文件中，可以以字符串的形式输入主题 IDs。例如，可以在帮助主题的#脚注中键入“IDH_FileMenu”来解释 File 菜单。在应用程序中，可以以数字形式输入主题 ID。例如，可以在 HelpContextID 属性中为 File 菜单键入 1001。Help Workshop 会将名字与帮助项目文件的 MAP 部分中的数字相匹配，如下所示：

```
[MAP]
IDH_FILEMENU=1001
IDH_FILEOPENMENU=1002
IDH_FILECLOSEMENU=1003
```

让这个[MAP]部分与用户的应用程序同步是很必要的。有些工具，如 HelpBreeze，利用这些值作为常量来产生 BAS 模块。但利用 Enumsg 来做也非常容易：

```
Enum HelpContext
    IDH_FILEMENU=1001
    IDH_FILEOPENMENU=1002
    IDH_FILECLOSEMENU=1003
End Enum
```

• 使用全文搜索和目录表

出现在 Microsoft Windows 9X 和 Microsoft Windows NT 中的两个新帮助功能是：全文搜索和目录表。全文搜索功能包含在 32 位 Help Viewer(WINHLP32.EXE)中。当利用 WINHLP32.EXE 查看时，甚至 Microsoft Windows 3.1 版本编译的帮助文件也支持这一功能。

利用 Find Setup Wizard，Windows 将会在帮助文件中自动建立一个单词数据库，以便使用 Find 功能。

Help Contents 清单是通过 Help 项目文件的 Options 部分中所列出的文本文件建立的，

如图 15.3 所示。目录文本文件中每行有一个简单格式，如下面的文本所示：

- ； Form HCW.CNT
- 1 Getting started
- 2 What is Help Workshop?=BAS_THE_HELP_COMPILER
- 2 Help workshop components=SDK_FILES_DOCS
- 2 Notational Conventions=SDK_NOTE_CONVENTION
- 1 What's New in Version 4.0
- 2 WinHelp4.0
- 3 Redesigned User Interface
- 4 New Help Topics dialog box=NEW_WINHELP_INTERFACE
- 4 New context menu=NEW_WINHELP_CONTEXT_MENU
- 4 New Options menu=NEW_WINHELP_OPTIONS_MENU
- 4 New annotation capabilities=NEW_ANNOTATION
- 4 Improved copying=NEW_WINHELP_COPY
- 4 New printing options=NEW_WINHELP_PRINTING
- 4 New ESC key function=NEW_KEY
- 4 Background color override=NEW_COLOR

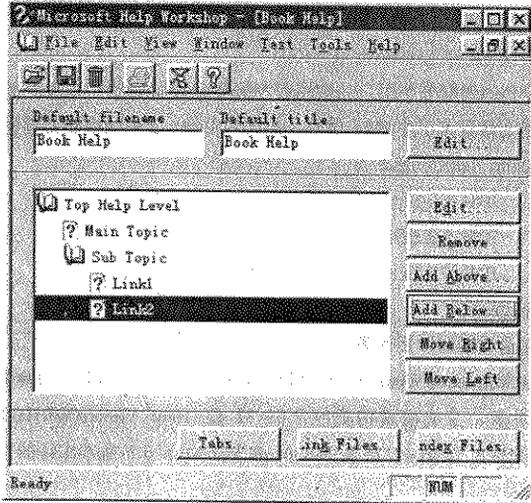


图 15.3 一个样本目录文本文件

可以通过打开 Help Workshop 中的源文件(CNT)来得到目录文件，编辑将出现在 Help 中的文件，如图 15.3 所示。

可以利用 File New 命令从 Help Workshop 中创建一个新的目录文件。

· 创建 Help 窗口

帮助文件可以在 Help Workshop 中定义的特定 Help 窗口中显示主题。要在 Help Workshop 中定义 Help 窗口，需要在帮助项目窗口中单击 Windows 按钮并完成如图 15.4 所示的对话框。

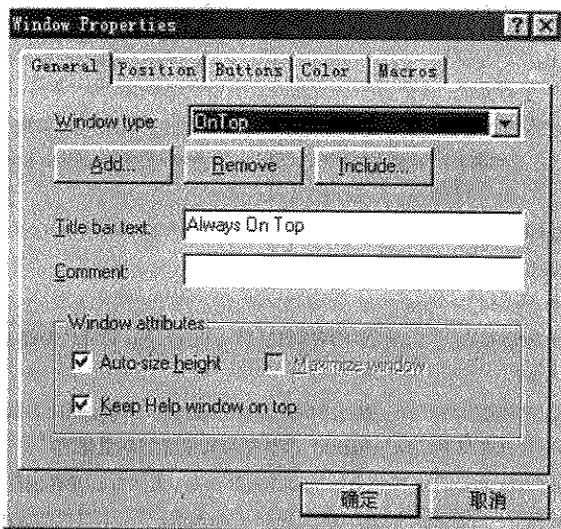


图 15.4 Windows Properties 对话框（它定义的 Help 窗口总是在上部）

要在这个窗口中显示一个主题，需在链接的隐藏文字的主题 ID 后键入 >OnTop。单击该链接，然后就会在 OnTop 窗口中显示主题。可以用同样的方式将窗口名添加到目录文件 (CNT) 的项中，只需在主题 ID 后键入 >OnTop 即可。为主题添加一个 >脚注，这样当用户从 Index 中选择主题时，会在定制窗口中显示主题。

· 使用宏

宏在 Help 系统中实现特殊的任务。可以在主题显示、用户单击按钮或用户单击链接时运行一个宏。要为链接添加一个宏，需在链接的隐藏文字的宏名前加上!脚注，例如，Related Information!KLink(API,WinHelp)。运行这个 KLink 宏，当用户单击 Related Information 时，会显示含有关键字 API 或 WinHelp 的主题清单。

用户能够使用的帮助宏清单可以在 Help Workshop Help 文件 HCW.HLP 中找到。

15.2 在项目中添加 Help 文件

用户可以直接调用 Windows API 来激活一个帮助文件，这是经实践证明可行的、老式的访问帮助文件的方法。该函数被命名为 WinHelp，在应用程序中需要为其添加一个声明，如下例所示。在这段样本代码中，当单击窗体时，Windows 帮助文件的 Contents 主题会弹出来。

```
Option Explicit
Private Declare Function WinHelp Lib"user32" Alias "WinHelpA"( ByVal hWnd As
    Long, ByVal lpHelpFile As String, ByVal wCommand As Long ,
    ByVal dwData As Long ) As Long
Const HELP_TAB =& HF
Private Sub Form_Click()
Dim x
x=WinHelp(hWnd," C:\WINDOWS\HELP\winhlp32.hlp"HELP_TAB,0)
End Sub
```

hWnd 参数被设置为用户的应用程序窗体的 hWnd 属性。lpHelpfile 参数是一个字符串，它含有要激活的帮助文件名。wCommand 参数是事先定义好的常量之一，它用于控制这个函数的行为(在 CommonDialog 控件被添加到 Toolbox 后，大多数常量都列在 Object Browser 的 MSComDlg 项下；在联机帮助下，列在 Help Constants 下；本节末的表中描述了这些常量)。dwData 参数可以采用几种类型的值，这取决于 wCommand 的值。尤其是，如果 wCommand 被设置为 cdlHelpContext，则 dwData 的值确定显示哪一个帮助文件。

要了解这些参数的工作方式，利用下面的 WinHelp 函数激活 Windows 帮助文件，然后显示一个特定的主题(需要将 Microsoft Common Dialog 控件添加到工具箱中，并需要编辑 winhlp32.hlp 的路径)。

```
Private Sub Form_Click()
Dim x
x=WinHelp(hWnd,"C:\WINDOWS\HELP\winhlp32.hlp",cdlHelpContext,700)
```

当启动帮助文件时，将显示 Keyword Not Found 主题，因为这里将 wCommand 参数设置为 cdlHelpContext 常量，dwData 参数设置为 Keyword Not Found 主题号(有些帮助创作工作，如 RoboHelp，会自动创建含有所有主题常量的 BAS 文件——它是一个很好的省时程序)。

用户会发现在函数调用中有一些有用的其他变化。当用 wCommand 参数传递表 15-2 中的每一个常量时，将由 WinHelp 函数实现一个特定的动作(大多数常量都是由 Visual Basic

为 CommonDialog 控件定义的, 因此, 用 cdl 作为前缀)。例如, 要显示有关如何使用 Help 系统本身的帮助信息, 需使用常量 cdlHelpOnHelp。

表 15-2 常量、值及其描述

常量	值	描述
cdlHelpContext	1	显示某一特定主题的帮助
cdlHelpQuit	2	关闭指定的帮助文件
cdlHelpIndex	3	显示指定的帮助文件的索引
HELP_FINDER	11	显示 Help Topics 对话框, 并在其中显示最后选择的标签。必须在应用程序中定义这个常量, 因为它不包含在内部常量中
HELP_TAB	15	显示 Help Topics 对话框, 它带有由 dwData 指定的标签索引 (Contents 标签是 0, Index 标签是 -2, Find 标签是 -1)。必须在应用程序中定义这个常量, 因为它不包含在内部常量中
cdlHelpContents	3	在当前帮助文件中显示 Contents 主题。该常量与帮助的早期版本兼容。新应用程序应该通过利用 HELP_FINDER 或 HELP_TAB 来显示 Help Topic 对话框
cdlHelpOnHelp	4	利用帮助应用程序自身显示帮助
cdlHelpSetIndex	5	为多索引帮助设置当前索引
cdlHelpSetContents	5	指定一个具体的主题作为 Contents 主题
cdlHelpContextPopup	8	显示一个由描述体号指定的主题
cdlHelpForceFile	9	创建一个仅以一种字体显示文本的帮助文件
cdlHelpKey	257	显示某一种特定关键字的帮助
cdlHelpCommandHelp	258	显示某一特定命令的帮助
cdlHelpPartialKey	251	调用 Windows Help 中的搜索引擎

系统已为 CommonDialog 文件预先定义了这些常量。可以利用 Object Browser 将其拷入代码中, 或在应用程序中使 CommonDialog 控件有效, 在这种情况下, 所有的 cdl 常量都将自动获得。要实现这一功能, 需从 Project 菜单中选择 Components, 并选中 Microsoft Common Dialog Control 复选框。这些常量在 Visual Basic 的联机帮助中的 HelpCommand 属性描述中也讨论过。

15.3 在项目中添加 F1 帮助

窗体、菜单项以及大多数控件都有一个 HelpContextID 属性。当按 F1 键时, 会提供到某一帮助主题的上下文相关跳转, App 对象的 HelpFile 属性为整个应用程序设置帮助文件的路径和文件名, 带有焦点的控件确定当按下 F1 键时激活哪一个帮助主题。如果控件的 HelpContextID 属性被设置为 0 (缺省值), 就会检查所含控件或窗体的非零 HelpContextID 值。最后, 如果窗体的 HelpContextID 为 0, 则帮助文件的 Contents 主题作为缺省主题被激活。

这种方案对于访问上下文相关的帮助很有效, 当按下 F1 键时帮助会被激活。但如果不按 F1 键, 如何激活帮助文件呢? 这里提供一种使 HelpContextID 属性更有效的技巧, 编写一个简单程序, 利用 SendKeys 命令将 F1 键发送到用户的应用程序中。SendKeys 语句告

诉 Windows 为窗口发送一个目前拥有焦点的按键，这样程序能够像 F1 键被按下那样进行响应，下列代码段展示了这一技术：

```
Option Explicit

Private Sub Form_Load()
    App.HelpFile="C:\WINDOWS\HELP\winhlp32.hlp"
End Sub

Private Sub cmdHelp_Click()
    cmdHelp.HelpContextID=700
    SendKeys "{F1}"
End Sub
```

要访问帮助文件在 Form Load 事件过程中的设置，但如果要访问多个帮助文件，可以在程序中改变路径和文件名。在 cmdHelp_Click 事件过程中，可以通过将 HelpContextID 属性设置为所需主题的上下文号，然后利用 SendKeys 命令发送 F1 键来激活帮助文件中指定的主题，这些 HelpFile 和 HelpContextID 属性可以在运行时设置(如上所示)，也可以在设计时设置。

菜单项还有一个 HelpContextID 属性，在帮助文件中指定的主题仅在菜单项高亮显示且由人工按下 F1 键时才被激活。如果单击菜单项，并在菜单事件过程中建立一个 SendKeys 命令，如下述代码所示，被激活的主题将不由菜单项的 HelpContextID 值确定，而是取决于当前拥有焦点的控件的 HelpContextID 值：

```
Private Sub munHelp_Click()
    SendKeys "{F1}"
End Sub
```

出现这种情况是因为菜单在单击之后立即消失，并且焦点在 SendKeys 命令有时间发送 F1 键之前返回给控件。使用菜单命令可以较好地获取有关控件的上下文相关的帮助信息，但这种行为在用户能识别事件序列之前可能会有些奇怪。

15.4 如何利用 CommonDialog 控件为项目添加帮助文件

CommonDialog 控件提供一种功能强大而又灵活的访问帮助文件的方法。可以为窗体添加一个 CommonDialog 控件，设置相关的帮助文件属性，并利用 ShowHelp 方法激活 Help 系统。这非常容易——甚至不必使用控件来显示一个对话框。例如，当单击 cmdHelpContents 命令按钮时，下列代码激活 Windows 自身的帮助：

```
Private Sub CmdHelpContents_Click()
```

```
dlgCommon.HelpCommand=cdlHelpOnHelp
dlgCommon.ShowHelp
```

```
End Sub
```

注意：如果 Windows 不能找到帮助目录文件，需要将 HelpCommand 属性设置为 cdlHelpContents，以显示帮助文件中的第一个主题。

下面是一个实例，它显示了当单击菜单项时激活的指定帮助文件主题：

```
Private Sub mnuHelp_Click()
    dlgCommon.HelpFile=" C:\WINDOWSHELP\winhlp32.hlp"
    dlgCommon.HelpCommand=cdlHelpContext
    dlgCommon.HelpContext=700
    dlgCommon.ShowHelp
End Sub
```

15.5 如何为窗体添加 WhatsThisHelp 功能

Windows 9X 为帮助文件提供了一些新方法。其中一个很好的功能是 WhatsThisHelp。具有这一功能的窗体的标题栏上会显示一个问号按钮，当单击时，将鼠标指针变为一个特殊的问号。单击带有这个特殊指针的窗体上的控件，会激发一个专属于该对象的弹出式主题。

为 Visual Basic 窗体及控件添加 WhatsThisHelp 很容易。首先，将窗体的 WhatsThisButton 和 WhatsThisHelp 属性设置为 True。在看到标题栏上的问号框之前，还必须改变窗体的 BorderStyle 属性。将 BorderStyle 设置为 1-FixedSingle 或 3-Fixed Dialog。如图 15.5 所示的是有问号标题栏按钮的窗体。

要确定激活哪一个帮助文件，必须将 App 对象的 HelpFile 属性设置为帮助文件的路径及其文件名，既可以在设计时，又可以在运行时实现。要在设计时设置帮助文件，需从 Project 菜单中选择项目的 Properties，然后选择 Project Properties 对话框的 General 标签，并在 Help File Name 文件框中输入相应的帮助文件名，包括文件的完整路径。要在运行时设置帮助文件，需将路径名和文件名赋给 App.HelpFile 属性。



图 15.5 有问号标题栏按钮的窗体

最后，要定义当单击给定控件时弹出的帮助文件主题，需将该控件的 `WhatsThisHelpID` 属性设置为主题的 ID 号。

• `WhatsThisMode`

使用 `WhatsThisHelp` 弹出式主题还有一种方法。可以利用窗体的 `WhatsThisMode` 方法激活特殊的问号鼠标指针。当为窗体添加 `WhatsThisHelp` 功能时，该项技术很有用，可把该窗体的 `Border/style` 设置为 `2-Sizable`。如图 15.6 所示的是单击 `Get Help` 命令按钮激活 `WhatsThisHelp` 的窗体。

当单击 `Get Help` 按钮时，鼠标指针变为特殊的问号，并且 `WhatsThisHelp` 状态操作方式几乎与在前例中单击标题栏问号按钮时完全相同。要使用这项 `WhatsThisHelp` 技术，需做的只是将窗体的 `WhatsThisHelp` 属性设置为 `True`，并将一行代码添加到命令按钮的 `Click` 事件中。代码如下：

```
Private Sub cmdGetHelp_Click()  
    whatsThisMode  
End Sub
```

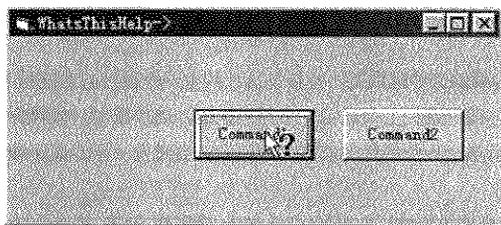


图 15.6 利用 `WhatsThisMode` 方法来激活 `WhatsThisHelp` 的窗体

第十六章 优化

在理想世界中,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。特别是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,从而优化所创建的应用程序。在设计 and 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 和其他一些应用程序中。VBS (VBScript) 是一种 Internet 的描述语言,也是 Visual Basic 语言的子集。如果用户是一个 VBA 或 VBS 的开发者,或许希望在这些语言间共享代码。

16.1 优化

优化具有科学和艺术两重含义。从科学的角度看,优化是指具体的优化技术;而从艺术的角度看,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:

- ①真实速度(应用程序实际计算或操作的速度)。
- ②显示速度(应用程序屏幕显示的速度)。
- ③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)。
- ④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 中时往往会产生其他影响)。

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他程序时会产生麻烦。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，无谓地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

绝大部分的开发者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。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，该方法与其他方法的效率一样，而且代码出奇地简单(只是比较难解)。

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他的应用程序，等等。

➡ 16.2 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。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）。

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。花费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。

➡ 16.2.1 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。其他典型的因素，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。例如，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。

- ①避免使用 Variant 变量。
- ②使用 Long integer 变量和整数运算。
- ③将常用的属性缓存在变量中。
- ④使用内嵌过程替代过程调用。
- ⑤尽可能使用常数。
- ⑥用 ByVal 传递参数，而不用 ByRef。
- ⑦使用类型确定的可选参数。
- ⑧利用集合的优点。

即使用户并不想优化代码的速度，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，如果要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 将转化为其他适当的数据类型，那么直接采用其他简单的数据

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
```

· 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 属性设置为缢或像素时，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。

表 16-1 把各种数值数据类型按运算速度顺序列出。

表 16-1 各种数值数据类型的运算速度

数值数据类型	速 度
Long	最快
Integer	
Byte	
Single	
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)，可以在循环体外将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 10~20 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，像这样的代码就非常慢：

```
For i = 0 To 10
    picIcon(i).Left = picPalette.Left
Next I
```

下面改写的代码就要快得多:

```
picLeft = picPalette.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))`。如上所述, 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, 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，将适当的值写入代码；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传递(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 6.0 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。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 intWeight 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：

- ① 使用 For Each...Next 替代 For...Next。
- ② 添加集合的对象时避免使用 Before 和 After 参数。
- ③ 使用对象集合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他对象后，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惟一的键，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high；当然，具体的数目还有赖于微处理器的速度和可用的内存。

16.2.2 优化显示速度

由于 Microsoft Windows 的图形特性，图形和其他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。下面的几种技术可用来提高应用程序的显示速度：

- ① 将容器的 ClipControls 属性设置为 False。
- ② 恰当地使用 AutoRedraw。
- ③ 使用 Image 控件替代 PictureBox 控件。
- ④ 设置属性时隐藏控件以避免多次重画。
- ⑤ 使用 Line 替代 PSet。
- ⑥ 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他控件之后绘图的图形方法，则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。在窗体包含大量控件时，会大大提高显示速度。

- 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，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。

- 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 控件。除非需要图片框提供的特殊功能时，如图形方法、包含其他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在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，就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。

16.2.3 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：

- ① 隐藏窗体而不是卸载。
- ② 预加载数据。
- ③ 使定时器在后台工作。
- ④ 使用进度指示器。
- ⑤ 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内容，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

在有些应用程序中，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他事情，这将进一步提高应用程序的速度。

· 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 98 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，这样在用户做其他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:

- ①在 Form_Load 事件中使用 Show。
- ②简化启动窗体。
- ③不要加载不需要的模块。
- ④启动时运行一个小的 Visual Basic 应用程序预加载运行时的(DLL)文件。

• 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他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), 你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, 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。实现这一目的的最好的方法是在启动窗体上显示进度栏, 并在加载每个其他窗体时予以更新。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他问题或过度的延迟。

• 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; 相反的, 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他模块的过程。

• 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的 DLL 文件。这

些 DLL 将用于 Visual Basic、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 的其他应用程序已经运行时，则你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 运行时的 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他部分。

16.3 优化大小

过去在设计应用程序时，经常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，在许多情况下，缩小应用程序仍然很重要。

有些应用程序需要从 Internet 网卸载或通过电子函件的附件形式传输，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，还可同时运行其他程序。通过优化应用程序所占的内存可提高程序性能。

16.3.1 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。

多数代码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。而下面元素的长度或数量是无须限制的：

- ①标识符名称
- ②注释
- ③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。其他元素，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：

- ①减少加载窗体的数目。
- ②减少控件数目。
- ③用标签代替文本框。
- ④保持数据在磁盘文件或资源中，并且只在需要时才加载。
- ⑤组织模块。
- ⑥考虑替换 Variant 数据类型。
- ⑦使用动态数组，并在删除时回收内存。
- ⑧回收被字符串或对象变量用过的空间。
- ⑨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

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就是：在设计时，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。

· 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，使用标签更为有效。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 事件中移动它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(像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。这对大的位图和字符串特别有价值。

• 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,但是比其他数据类型所占内存大。当要压缩应用程序多余的空间时,应考虑用其他数据类型替代 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 占用 2 个字节,Double 占用 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,共 5 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,使用其他数据类型替代 Variant,灵活性被降低,为弥补损失的灵活性,不得不增加更多的代码。结果没有真正的减小。

• 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,并回收数组所用的内存。例如,用下面的代码可回收动态数组所用的空间:

```
Erase MyArray
```

这里,Erase 用于完全删除数组,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的空间。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。要使应用程序尽量小，应尽可能地回收这些变量所用的空间。将零长度字符串赋给字符串变量时，可回收其空间：

```
SomeStringVar = ""
' 回收空间。
```

同样的，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，在 Debug.Print 语句中，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 语句时，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。若没出错，则该变量没有被使用。

16.3.2 修剪图形

图形(图片和图形方法)要消耗许多内存。从某种程度上来说，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：

- ①使用 Image 控件显示位图。
- ②需要时从文件中加载位图并共享图片。
- ③使用 PaintPicture 方法。
- ④释放图形所用内存。
- ⑤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。这样将浪费许多 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 控件是图形控件而不是一个窗口, 并且使用的资源不多。其实, 典型的情况是: 使用 5~10 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)、图形方法或包含其他控件的能力。

• 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。
```

与上而相比, 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")
```

同样的, 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, 而让其他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

不必总是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)`，但也可使用其他图形文件格式。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。

16.3.3 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：

- ①部件在需要时加载，不需要时卸载。
- ②在 Windows 95 或 Windows NT 下，即使应用程序的其他部分是 16 位部件，跨进程的部件也可以是 32 位的执行程序。
- ③远程部件可使用网络上其他机器的资源。

另外，部件可被单独调试并可在其他应用程序中重用。也许这样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：

- ①跨进程
- ②进程内
- ③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。

· 跨进程部件

跨进程部件是一个可执行程序，它可以向其他程序提供服务。像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。与其他类型相比，跨进程部件提供了一些很有价值的特点：

- ①异步操作（“线程”）。
- ②部件中不能捕获的错误不会导致调用应用程序崩溃。
- ③在 16 位和 32 位应用程序之间相互操作。

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。

即使应用程序在 32 位系统上运行，如果是惯用的 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。这样有助于充分利用 32 位特性，并保护在 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：

- ①启动速度
- ②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。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他程序提供服务。与跨进程部件相比，进程内部件有两个优点：

①改善加载时间

②无跨进程开销

对进程内部件来说，既无需创建新的进程，也不必加载运行时的 DLL 文件。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为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他地方获取，部件可在本地操作数据，之后只通过网络返回结果，这样是比较有效的。

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。