- 更小，更轻便的应用程序。VB3 应用程序比功能相同的 VBA 应用程序趋于更小且运行时需更少的内存。
- 独立的 EXEs。VB3 能创建可以直接从 Windows 中运行的编译的 EXE 程序，而不需在其他应用程序，如 Excel 中运行。
- 用户自定义控制。VB3 应用程序可以与一些被称为用户自定义控制的特殊工具一同工作。在 VBA 中用户自定义控制则无效。
- 内置的数据访问。你可将 VB3 控制直接与数据库表和区域相连。

在另一方面，VBA 具有 VB3 所没有的以下一些优点：

- 内置的对象。Excel 的对象可以被 VBA 自动使用。为了在 VB3 中使用这些对象，必须经过几个特殊步骤(在后面部分描述)。
- 快速访问对象。在 VBA 中访问对象的属性和方法要比在 VB3 中快 4～10 倍。然而，VB3 对于这些缺点的补偿是它对一些没有对象的操作要比 VBA 快。
- 新的语言结构。VBA 有一些 VB3 至今还没有的一些语句，如 For Each 和 With... End with。

### 20.7.1 从 VB3 启动 Excel

VB3 不是 Excel 中固有的，因此必须采用特殊步骤来访问 Excel 对象。要从 VB3 中得到 Excel 对象，用 CreateObject 和 GetObject 函数。

下列代码从 VB3 中启动 Excel 并且返回 Excel 的 Application 对象：

```
' VB3 Code.
' declarations.
Global Application As Object
Sub ConnectToExcel()
        ' Start a copy of Excel.
        Set Application=CreateObject("Excel. Application")
        ' Make the copy visible (it starts invisibly otherwise).
        Application. Visible=True
End Sub
```

在与 Excel 取得联系后，你可以用 Application 对象获取和使用其他对象。下列代码创建并存贮一个工作簿：

```
' VB3 Code.
Sub CreateWorkbook()
        Dim objWorkbook As Object
        Set objWorkbook=Application. Workbooks. Add
        objWorkbook. SaveAs "VB3WRK. XLS"
End Sub
```

### 20.7.2 在 VB3 中使用 VBA 代码

或许学习如何用 VB3 编写 Excel 程序的最简单的方法是看在 VB3 中运行 VBA 代码所

必须做的改动。

要在 VB3 中使用 VBA 代码，需按以下步骤操作：

1. 将 VBA 模块保存成正文，并在 VB3 项目中装载它。

2. 声明对象变量，并且用 CreateObject 和 GetObject 函数给应用程序的对象建立引用。

3. 修改对象引用使它们更清楚。

4. 删除命名的参数的命名部分。

5. 在 VB3 项目中声明内置的常量(xlNone，xlOn 等)(有关内置的常量及其对应数值的列表，见附录 C“固有常量列表”)。

6. 用清晰的对象引用替换 With...End With 语句。

下面例子显示了要在 VB3 上运行的经修改的 VBA 代码，粗体表示增加到代码的各项。划掉的字体表示要从 VBA 中删除的代码。

```
' VB3 Declare object variables.
Dim Application As Object
Dim chrtObj As Object

Sub AddEmbeddedChart ()
     ' VB3 Get the running copy of Excel
     ' (use CreateObject to start new copy.)
     Set Application = GetObject(, "Excel.Application")
     ' Create a new workbook.
     Application.Workbooks.Add
     ' Add some random values to cells.
     For i = 1 To 10
          Application.ActiveSheet.Cells(i, 1).Value = 20 * Rnd
     Next
     ' Add an embedded chart. VB3 replace With with an object
     ' variable, remove named arguments.
     With Set chrtObj = Application.ActiveSheet.
          ChartObjects.Add(Left:=100, Top:=100, Width:=200, _
               Height:=200)
     ' Plot some data on the chart. VB3 adds object variables
     ' and removes named arguments.
     chrtObj.Chart.SeriesCollection.Add Source:=Application _
         .Range("A1:A10")
     End With
End Sub
```

### 20.7.3　将 VBA 代码移到 VB3 中

在你将代码从 VBA 移到 VB3 之前，先从 Application 对象中改变所有对象引用以使之更清楚。然后检查 VBA 中的代码。VBA 不需被告知就知道 Application 对象是什么，你必须告诉 VB3 那个 Application 是一个 Excel Application 对象。

在将代码装入 VB3 后，建立一个表示 Excel Application 对象的名为 Application 的对象变量。将它设成模块中的全局变量以使它能被容易地包含在一个项目中。

你也许想用其他公共的已有的对象名来做同样的事情，见下例：

```
' Commonly used object variables.
Global Application As Object
Global ActiveWorkbook As Object
Global ActiveSheet As Object

' Initialize objects.
Sub Main()
     ' Start Excel running.
     Set Application = CreateObject("Excel.Application")
     ' Make Excel visible.
     Application.Visible = True
     ' Microsoft Excel starts with no loaded workbook, so create _
       one.
     Application.Workbooks.Add
     Set ActiveWorkbook = Application.ActiveWorkbook
     Set ActiveSheet = Application.ActiveSheet
End Sub
```

### 20.7.4 从 VB3 中调用 VBA 过程

用 VB3 编写的程序可通过使用 Application 对象的 Run 方法来调用 VBA 程序，见下例：

```
' VB3.
Dim Application As Object
Sub Main()
     Set Application = CreateObject("Excel.Application")
     Application.Workbook.Open "BOOK1.XLS"
     Application.Run "SubWithNoArgs"
End Sub

' VBA BOOK1.XLS
Sub SubWithNoArgs ()
     MsgBox "I been to the desert on a Sub with no args."
End Sub
```

如果 VBA 程序使用参数，则将它们放在程序名之后。用逗号在各参数间以及参数与程序名间进行分隔。下行代码运行一个有 2 个参数的 VBA 程序：

Application. Run "SubWith2Args", 1234,"String Arg"

如果传给 VBA 程序的参数不能被强制转换成所期待的类型，则 Run 失败且不引起一个错误。然而，通过检查其返回值，可以检查它是否成功。

下面代码运行一个 VBA 程序并检查其是否成功：

```
' VB3
Sub TestVBAProc()
     vRet=Application. Run("BOOK1. XLS! SubWith2Args",_
       1234,"String Arg")
     If IsNull(vRet)=False Then MsgBox "Procedure ran OK."
End If
```

根据调用程序的类型，Run 返回不同的值：

| 返回值 | 如果… |
| --- | --- |
| Null | VBA 子程序或函数失败 |
| True | VBA 子程序成功 |
| 函数的返回值 | VBA 函数成功 |

### 20.7.5 VB3 的限制

当从 VB3 中调用 VBA 时，必须遵守以下限制：

- 不能传递数组。然而，通过将数组转换为字符串然后在被调用的程序中分析该字符串可避开这一限制。
- 不能传递非 OLE 对象。VB3 的控制和对象与 Excel 中的 OLE 对象不同。VB3 为 OLE 对象使用 Object 数据类型。
- 不能回调。不能从 VBA 中调用由 VB3 产生的程序。
- 不能通过引用传递。即使 VBA 程序的参数被声明为 ByRef，改变这些参数也不能影响 VB3 的变量。

## 20.8 使用动态链接库

动态链接库(DLLs)在整个 Windows 中被使用。Windows 自己也包含供所有应用程序用来执行其功能(如显示窗口，显示图形和管理内存)的几个动态链接库。Visual Basic 也为执行这些工作提供了许多方法，例如 Visual Basic 的 Shell 函数与 Windows 的 WinExec 函数相同。

一直以来，有些事情只用 Visual Basic 来做是困难或不可能的。例如，Windows 提供了获取和修改应用程序的 INI 文件中的字符串的函数。只用 Visual Basic 修改 EXCEL5.INI 需要许多条语句。

DLLs 使你能明显地扩展 Visual Basic，但使用它们需要一定代价。你必须正确声明和调用 DLL 函数。任何错误将使你的程序停止运行使系统崩溃。

### 20.8.1 声明和调用函数

使用一个 DLL 函数需要 2 个步骤：

1. 声明该函数。
2. 调用该函数。

下行程序声明一个将设置写到应用程序的初始化文件(*.INI)中的 DLL 函数。

```
Declare Function WritePrivateProfileString Lib "KERNEL"_
        (ByVal lpApplicationName As String,lpKeyName As Any,_
        lpString As Any,ByVal lplFileName As String) As Integer
```

下行代码用 WritePrivateProfileString()给 Excel 的 INI 文件中增加一行：

iWorked = WritePrivateProfileString("Microsoft Excel",ByVal
"OPEN",_
ByVal "C:\EXCEL\LIBRARY\DEMO. XLA","C:\WINDOWS\EXCEL5.
INI")

该行使得当 Excel 开始运行时装载一个 add-in。

大多数 DLL 函数在它们运行成功时返回非 0 值而失败时返回 0。当返回一个数据，如一个字符串或一个句柄时，一个 DLL 函数经常将数据返回成一个自变量，该自变量是通过引用(ByRef)传给该函数的。

当返回一个包含字符串的自变量时，你必须给该字符串留出足够的空间，因此你必须产生一个至少与返回值一样多的字符串。使用太小的字符串会引起一个整体保护错——一个致命错误。

GetWindowsDirectory()函数将 Windows 路径当做它的一个自变量返回。你必须产生一个至少能装 144 个字符的字符串来保存该信息。下面代码返回 Windows 目录并在一个消息框中显示该目录。

```
' Declarations.
Declare Function GetWindowsDirectory Lib "Kernel" (ByVal lpBuffer
     As String,ByVal nSize As Integer) As Integer
Sub ShowWindowsDirectory()
     ' Create a string large enough to hold any possible path.
     Dim sWinDir As String * 144
     ' GetWindowsDirectory returns the length of the string,
     ' if successful.
     iLength = GetWindowsDirectory(sWinDir, 144)
     ' If successful, trim the extra space and display the path.
     If iLength <> 0 Then
          MsgBox Mid$(sWinDir, 1, iLength)
     End If
End Sub
```

### 20.8.2　将 C 声明转换成 Visual Basic 声明

DLL 函数通常是由 C 语言语法所构成的。为了正确地从 Visual Basic 中调用，必须将它们转换成有效的 Declare 语句。表 20-4 中显示了如何对一些声明进行转换

**表 20-4　使 C 和 Visual Basic 声明相同**

| C 语言声明 | 等价的 Visual Basic 声明 | 与…一起调用 |
|---|---|---|
| 指向字符串的指针(LPSTR)<br>String ptr(MAC) | ByVal S As String | 任何字符串或变体型变量 |
| 指向整数的指针(LPINT)Short<br>*(MAC) | I As Integer | 任何整数或变体型变量 |
| 指向一个长整数的指针<br>(LPDWORD)Long *(MAC) | L As Long | 任何长整数或变体型变量 |

（续）

| C 语言声明 | 等价的 Visual Basic 声明 | 与…一起调用 |
|---|---|---|
| 指向结构的指针（如 LPRECT）Rect *（MAC） | S As Rect | 属于用户定义的类型的所有变量 |
| 整数（INT，UINT，WORD 和 BOOL）Short（MAC） | ByVal I As Integer | 任何整数或变体型变量 |
| 句柄（hWnd，hDC，hMenu，等） | ByVal h As Integer | 任何整数或变体型变量 |
| 长整型（DWORD 和 LONG） | ByVal L As Long | 任何长整数或变体型变量 |
| 指向一个数组的指针 | I As Intger | 数组中的第一个元素，如 I(0) |
| 指向一个 Void 的指针 Void *）( | As Any | 任何变量（当传送一个字符串时使用 ByVal） |
| Void（函数返回值）Null | Sub procedure As Any | 不要参数 ByVal 0& |

### 20.8.3 调用其他的 DLLS

在 Declare 语句中的 Lib libname 告诉 Visual Basic 到哪里查找 DLL。对于 Windows 的 DLL，libname 是 User，GDI，Kernal 或一个其他系统的 DLL，如 MMSystem。对于其他 DLL，libname 是一个可包括路径的文件说明。

下行代码在 DEMO. DLL 文件中声明 Splatter()函数

```
Declare Function Splatter Lib "c:\windows\demo. dll"_
        (ByVal in,ByVal out)
```

### 20.8.4 传送参数

缺省情况下，Visual Basic 通过引用（32 位远地址）来传递参数。然而，许多 DLL 函数都期望一个参数通过值来传递。如果用引用把一个参数传递给期待参数通过值来传递的函数，则该函数将得到错误的数据并且不能正确运行。

要用值来传递参数，则要在 Declare 语句中将 ByVal 关键字放在参数声明之前。这样做可保证每次调用该函数时，该参数由值来传递。

有些 DLL 程序可以给同一个参数接收多种类型的数据。为了传递多于一种类型的数据，用 As Any 声明参数可取消类型限制。

当使用 As Any 时，Visual Basic 猜想该参数由引用传递。在实际调用该函数时用 ByVal 从而用值来传递参数。当传递字符串时，用 ByVal 可将一个 Visual Basic 字符串转换成以 Null 结束的字符串。

### 20.8.5　使用 Alias 关键字

DLL 函数可以具有 Visual Basic 不允许使用的名字。例如，有些窗口函数以下划线字符开头，但在 Visual Basic 中的标识符不能以下划线开头。要改正这个问题，则用 Alias 关键字。

下行代码声明一个_lopen()函数并创建一个别名 LOpen()，它是你在 Visual Basic 中可以使用的：

```
Declare Function LOpen Lib "kernel" Alias "_lopen"_
        (ByVal fn As String,ByVal f As Integer) As Integer
```

### 20.8.6　使用特殊的 DLL 所要考虑的事情

大多数函数期待以 Null 字符结束字符串。用 ByVal 传递一个字符串则给该字符串后边加上一个 Null 字符。这样，用 Visual Basic 的 InStr 函数，通过查找 Chr $ (0)，你可以找到所返回的字符串的结尾。

通常你只能给 DLL 函数传递数字数组。要给一个 DLL 函数传递一个数字数组，应将该数组的第一个元素置为 ByRef。用这种方法传递的数组不能超过 64K。

为了给一个 DLL 函数传递一个空指针，可将参数声明成 As Any，并且传递表达式 ByVal 0&。

将句柄(hWnd，hDC 等)当作 ByVal Integer 传递给函数。返回句柄的参数应简单地定义成 As Integer。句柄在整个 Windows 过程中被用作标识诸如窗口和设备之类的对象。

除非 DLL 用 OLE 2.0 写成，否则你不能给 DLL 函数传递 Excel 对象。OLE 2.0 库定义了获取有关 Excel 对象的信息的方法。

### 20.8.7　利用公共 DLL 函数声明

下面代码显示了在 Visual Basic 中普遍使用的一些 Windows DLL 函数。这些函数是 Windows 提供的许多函数的一小部分。

```
' Gets a string from a section in WIN.INI.
Declare Function GetProfileString Lib "Kernel" _
     (ByVal lpAppName As String, lpKeyName As Any,ByVal _
     lpDefault As String, ByVal lpReturnedString As String, _
     ByVal nSize As Integer) As Integer

' Writes a string to a section in WIN.INI.
Declare Function WriteProfileString Lib "Kernel" _
     (ByVal lpApplicationName As String, lpKeyName As Any, _
     lpString As Any) As Integer

' Gets an integer value from an entry in a section of WIN.INI.
Declare Function GetProfileInt Lib "Kernel" _
     (ByVal lpAppName As String, ByVal lpKeyName As String, _
     ByVal nDefault As Integer) As Integer

' Gets an integer value from an entry
' in a section in an application's .INI file.
Declare Function GetPrivateProfileInt Lib "Kernel" _
     (ByVal lpApplicationName As String, ByVal lpKeyName _
     As String, ByVal nDefault As Integer, ByVal lpFileName _
     As String) As Integer
```

```
' Gets a string from a section in an application's .INI file.
Declare Function GetPrivateProfileString Lib "Kernel" _
     (ByVal lpApplicationName As String, lpKeyName As Any, _
     ByVal lpDefault As String, ByVal lpReturnedString As String, _
     ByVal nSize As Integer, ByVal lpFileName As String) As Integer

' Writes a string to a section in an application's .INI file.
Declare Function WritePrivateProfileString Lib "Kernel" _
     (ByVal lpApplicationName As String, lpKeyName As Any, _
     lpString As Any, ByVal lplFileName As String) As Integer

' Gets the Windows directory. For example, "\WINDOWS"
Declare Function GetWindowsDirectory Lib "Kernel" _
     (ByVal lpBuffer As String, ByVal nSize As Integer) As Integer

' Gets the Windows system directory. For example, "\WINDOWS\SYSTEM"
Declare Function GetSystemDirectory Lib "Kernel" _
     (ByVal lpBuffer As String, ByVal nSize As Integer) As Integer

' Clears data from the Clipboard.
Declare Function EmptyClipboard Lib "User" () As Integer

' Switches the focus to a specific window.
Declare Function SetFocusAPI Lib "User" Alias "SetFocus" _
     (ByVal hWnd As Integer) As Integer

' Gets the handle of the window that has focus.
Declare Function GetFocus Lib "User" () As Integer

' Gets the handle of the currently active window.
Declare Function GetActiveWindow Lib "User" () As Integer
```

## 20.9 总 结

编程只受你的想象力的限制。从现在起，对你来说有多个发展方向。做为一名合格的VBA 程序员，你也许会对以下出版物感兴趣。

- Microsoft Visual Basic，Professional Edition，这本书是对 Visual Basic 的最全面的说明，它包括许多用户自定义的控制，Help 编译器，Windows API Help，和许多代码实例。
- Microsoft Office Development Kit，在 CD－ROM 上包含了所有 Microsoft Office 产品和编程文档，还有用 Excel，Visual Basic 和 Microsoft Word 写的许多专业的例子。
- Programming Windows，by Charles Petzold，解释了许多编写 Windows 程序的底层概念。虽然这本书是为 C 程序员写的，你也可以在 Visual Basic 中使用所有的概念和大多数 Windows API 调用。
- Inside OLE 2.0，by kraig Brockschmidt，和 OLE 2 Programmer′s Reference，第 1、2 卷，解释了如何用 OLE 库创建和访问应用程序。这些是关于 OLE 2.0 的权威性技术丛书。

# 附录 B　键码表

SendKeys 和 OnKey 方法用码来识别键盘上的特殊键。这些码列在表 B. 1 中。字母数字键用其正常字符识别，如“a”。

Chr 和 Asc 函数用字符的数值码，表 B. 2 列出了 ANSI 字符集中的字符的数值码。

**表 B-1　特殊键的键码**

| Key name | Key code |
|---|---|
| BACKSPACE | "{BACKSPACE}" or "{BS}" |
| BREAK | "{BREAK}" |
| CAPS LOCK | "{CAPSLOCK}" |
| CLEAR | "{CLEAR}" |
| DELETE or DEL | "{DELETE}" or "{DEL}" |
| DOWN arrow | "{DOWN}" |
| END | "{END}" |
| ENTER (numeric keypad) | "{ENTER}" |
| ENTER | "~" |
| ESC | "{ESCAPE} or {ESC}" |
| HELP | "{HELP}" |
| HOME | "{HOME}" |
| INS | "{INSERT}" |
| LEFT arrow | "{LEFT}" |

(continues)

| Key name | Key code |
|---|---|
| NUM LOCK | "{NUMLOCK}" |
| PAGE DOWN | "{PGDN}" |
| PAGE UP | "{PGUP}" |
| RETURN | "{RETURN}" |
| RIGHT arrow | "{RIGHT}" |
| SCROLL LOCK | "{SCROLLLOCK}" |
| TAB | "{TAB}" |
| UP arrow | "{UP}" |
| F1 | "{F1}" |
| F2 | "{F2}" |
| F3 | "{F3}" |
| F4 | "{F4}" |
| F5 | "{F5}" |
| F6 | "{F6}" |
| F7 | "{F7}" |
| F8 | "{F8}" |
| F9 | "{F9}" |
| F10 | "{F10}" |
| F11 | "{F11}" |
| F12 | "{F12}" |
| F13 | "{F13}" |
| F14 | "{F14}" |
| F15 | "{F15}" |
| SHIFT | "+" |
| CTRL | "^" |
| ALT or OPTION | "%" |
| COMMAND | "*" |

| Number | Character |
|---|---|
| 1 | Not supported |
| 2 | Not supported |
| 3 | Not supported |
| 4 | Not supported |
| 5 | Not supported |
| 6 | Not supported |
| 7 | Not supported |
| 8 | Not supported |
| 9 | Tab |
| 10 | Line feed |
| 11 | Not supported |
| 12 | Not supported |
| 13 | Carriage return |
| 14 | Not supported |
| 15 | Not supported |
| 16 | Not supported |
| 17 | Not supported |
| 18 | Not supported |
| 19 | Not supported |
| 20 | Not supported |
| 21 | Not supported |
| 22 | Not supported |
| 23 | Not supported |
| 24 | Not supported |
| 25 | Not supported |

(continues)

| Number | Character |
|---|---|
| 26 | Not supported |
| 27 | Not supported |
| 28 | Not supported |
| 29 | Not supported |
| 30 | |
| 31 | |
| 32 | Space |
| 33 | ! |
| 34 | " |
| 35 | # |
| 36 | $ |
| 37 | % |
| 38 | & |
| 39 | ' |
| 40 | ( |
| 41 | ) |
| 42 | * |
| 43 | + |
| 44 | , |
| 45 | — |
| 46 | . |
| 47 | / |
| 48 | 0 |
| 49 | 1 |
| 50 | 2 |
| 51 | 3 |

| Number | Character |
|---|---|
| 52 | 4 |
| 53 | 5 |
| 54 | 6 |
| 55 | 7 |
| 56 | 8 |
| 57 | 9 |
| 58 | : |
| 59 | ; |
| 60 | < |
| 61 | = |
| 62 | > |
| 63 | ? |
| 64 | @ |
| 65 | A |
| 66 | B |
| 67 | C |
| 68 | D |
| 69 | E |
| 70 | F |
| 71 | G |
| 72 | H |
| 73 | I |
| 74 | J |
| 75 | K |
| 76 | L |
| 77 | M |

(continues)

| Number | Character |
|---|---|
| 78 | N |
| 79 | O |
| 80 | P |
| 81 | Q |
| 82 | R |
| 83 | S |
| 84 | T |
| 85 | U |
| 86 | V |
| 87 | W |
| 88 | X |
| 89 | Y |
| 90 | Z |
| 91 | [ |
| 92 | \ |
| 93 | ] |
| 94 | ^ |
| 95 | _ |
| 96 | ` |
| 97 | a |
| 98 | b |
| 99 | c |
| 100 | d |
| 101 | e |
| 102 | f |
| 103 | g |

| Number | Character |
|---|---|
| 104 | h |
| 105 | i |
| 106 | j |
| 107 | k |
| 108 | l |
| 109 | m |
| 110 | n |
| 111 | o |
| 112 | p |
| 113 | q |
| 114 | r |
| 115 | s |
| 116 | t |
| 117 | u |
| 118 | v |
| 119 | w |
| 120 | x |
| 121 | y |
| 122 | z |
| 123 | { |
| 124 | \| |
| 125 | } |
| 126 | ~ |
| 127 | Not supported |
| 128 | Not supported |
| 129 | Not supported |

(continues)

| Number | Character |
|---|---|
| 130 | ‚ |
| 131 | ƒ |
| 132 | „ |
| 133 | … |
| 134 | † |
| 135 | ‡ |
| 136 | ˆ |
| 137 | ‰ |
| 138 | Š |
| 139 | ‹ |
| 140 | Œ |
| 141 | |
| 142 | |
| 143 | |
| 144 | |
| 145 | |
| 146 | ‘ |
| 147 | ’ |
| 148 | “ |
| 149 | • |
| 150 | – |
| 151 | — |
| 152 | ˜ |
| 153 | ™ |
| 154 | š |
| 155 | › |

| Number | Character |
|---|---|
| 156 | œ |
| 157 | Not supported |
| 158 | Not supported |
| 159 | Ÿ |
| 160 | |
| 161 | ¡ |
| 162 | ¢ |
| 163 | £ |
| 164 | ¤ |
| 165 | ¥ |
| 166 | ¦ |
| 167 | § |
| 168 | ¨ |
| 169 | © |
| 170 | ª |
| 171 | « |
| 172 | ¬ |
| 173 | - |
| 174 | ® |
| 175 | ¯ |
| 176 | ° |
| 177 | ± |
| 178 | ² |
| 179 | ³ |
| 180 | ´ |
| 181 | µ |

(continues)

| Number | Character |
|---|---|
| 182 | ¶ |
| 183 | · |
| 184 | ¸ |
| 185 | ¹ |
| 186 | º |
| 187 | » |
| 188 | ¼ |
| 189 | ½ |
| 190 | ¾ |
| 191 | ¿ |
| 192 | À |
| 193 | Á |
| 194 | Â |
| 195 | Ã |
| 196 | Ä |
| 197 | Å |
| 198 | Æ |
| 199 | Ç |
| 200 | È |
| 201 | É |
| 202 | Ê |
| 203 | Ë |
| 204 | Ì |
| 205 | Í |
| 206 | Î |
| 207 | Ï |

| Number | Character |
|---|---|
| 208 | Ð |
| 209 | Ñ |
| 210 | Ò |
| 211 | Ó |
| 212 | Ô |
| 213 | Õ |
| 214 | Ö |
| 215 | × |
| 216 | Ø |
| 217 | Ù |
| 218 | Ú |
| 219 | Û |
| 220 | Ü |
| 221 | Ý |
| 222 | Þ |
| 223 | ß |
| 224 | à |
| 225 | á |
| 226 | â |
| 227 | ã |
| 228 | ä |
| 229 | å |
| 230 | æ |
| 231 | ç |
| 232 | è |
| 233 | é |

(continues)

| Number | Character |
|---|---|
| 234 | ê |
| 235 | ë |
| 236 | ì |
| 237 | í |
| 238 | î |
| 239 | ï |
| 240 | ð |
| 241 | ñ |
| 242 | ò |
| 243 | ó |
| 244 | ô |
| 245 | õ |
| 246 | ö |
| 247 | ÷ |
| 248 | Ø |
| 249 | ù |
| 250 | ú |
| 251 | û |
| 252 | ü |
| 253 | ý |
| 254 | þ |
| 255 | ÿ |

# 附录 D　专业水平代码的标准风格

为你所编写的程序中的变量名、程序名、常量名以及其他用户自定义项提供一个命名标准是个好主意。给这些项用一致的方法命名可帮助你将 Visual Basic 中的关键字和代码中定义的项区分开来。这对于编写大型程序以及几个人使用、编写、维护一个程序尤其重要。

起初，遵守一系列标准看起来是件多余的工作。然而，你很快就能感觉出它的优点。风格标准不仅使别人更容易读你编写的程序，它也帮助你记住你所写的代码，它们也使你的工作看起来很专业。当别的程序员看到你能编写出无任何缺陷的软件时，一定会忌妒你的。

这里公布的标准是个好的起点并且惯穿在本书的简单例子中。你也可选择性的开发你的程序或遵守其他的标准，只有少量的规则是要绝对遵守的：

- 用前缀标识数据类型和作用域
- 尽可能使用描述性的名字
- 缩进相关的代码块
- 在一块代码前用描述性的注释
- 在程序开头的注释中写上所有设想
- 当然还要给复杂的代码行加注释性解释

## D.1　标识作用域

作用域是一个变量，常量或程序在代码中可以使用的区域。在 Visual Basic 中，有三级作用域，如下表所示：

| 作用域 | 描述 | 前缀 |
|---|---|---|
| Local | 变量或常量只在定义它们的程序内部有效 | none |
| Module | 变量或常量只在定义它们的模块内部有效 | m |
| Global | 变量或常量在项目中的所有模块中有效 | g |

### D.1.1　局部作用域

缺省情况下，任何在一个程序中定义的变量或常量都有一个局部作用域。例如：

```
Sub LocalScope()
    ' Define a local variable of data type integer.
    Dim iCount As Integer
    For iCount = 0 to 200
        ' Do something...
    Next iCount
```

End Sub

如果另一个过程使用变量 iCount，它将是一个新的变量并且对 LocalScope 过程中的没有影响。在定义它的过程外，局部变量不能被修改，除了常量值永远不变外，常量的其他特性和变量一样。例如：

```
Sub LocalScope()
    ' Local constants for RGB colors,
    Const RED=&HFF,GREEN=&HFF00,BLUE=&HFF0000
    ' Following code would cause an error, so it' scommented out.
    ' RED=32
End Sub
```

再重复一遍，别的程序不能看到 LocalScope 程序中定义的 RED 值。

**注意**：常量的名字通常用大写字母表示，该规则的例外情况是一个用功能前缀定义的一个功能常量；例如：xlMaximize。

当使用局部变量时，Dim 语句是可选的。当一个词不能识别时，Visual Basic 自动将它定义成一个变量。例如：

```
Sub AutomaticVariable()
    ' Create a local variable that contains a string.
    vAutoString="this variable was not explicitly declared"
    ' Create a local variable that contains a number.
    vAutoNumber=10
End Sub
```

当 Visual Basic 自动创建一个变量时，它使用缺省的数据类型。Visual Basic 使用 Variant 数据类型作为缺省值，除非你用 Deftype 语句改变它。

**注意**：自动创建的变量使编程者不容易检查变量名的拼写错误，通过 Option Explicit 语句可关闭该特性从而更安全。

### D.1.2　模块作用域

缺省情况下，任何在一个程序外部定义的变量或常量都具有模块作用域，例如：

```
' Define a module-level variable of data type Boolean (true/false).
Dim mbFlag As Boolean

' Run this procedure to see how module-level variables work.
Sub ModuleVariable()
    ' Assign a value to the module-level variable.
    mbFlag = True
    ' Call another procedure to display the value of the variable.
    DisplayVariable
    ' Displays False in a message box.
    MsgBox bFlag
End Sub

' This procedure is called by ModuleVariable.
Sub DisplayFlag()
```

ModuleVariable 和 DisplayFlag 程序可以看见和修改 mbFlag 变量。然而，如果你将 Dis-

playFlag 程序移到另一个模块，则它不再能看见和修改该变量——mbFlag 变量只有在定义它的模块内可见。

### D.1.3 全局作用域

在一个程序之外定义，使用 Public 关键字的变量具有全局作用域。例如：

```
' Define a global variable of data type Boolean (true/false).
Public gbFlag As Boolean

' Define a global constant for the RGB color red.
Public Const gRED=&HFF
```

变量 gbFlag 和常量 gRED 对所有模块中的所有程序都有效。注意：在一个程序内部不能用 Public；这使得全局和模块作用域一起在每个模块开头进行定义。

## D.2 标识数据类型

数据类型指示一个变量可以包含的数据的类别。在 Visual Basic 中有几个固有的数据类型，如下表所示：

| 数据类型 | 描述 | 前缀 |
|---|---|---|
| Boolean | True 或 False | b |
| Currency | 一个介于+/-9.22E14 之间的金融值，货币值精确到小数点后面 4 位 | C |
| Date/time | 可用任何无符号数表示的一个日期或时间 | dt |
| Double | 一个介于+/-1.798E308 之间的小数 | d |
| Error | 一个 Visual Basic 或用户自定义的错误 | err |
| Integer | 介于-32768 和 32767 间的整数 | i |
| Long | 介于-2147483648 和 2147483647 之间的整数 | l |
| Object | 对一个应用程序中的对象的引用 | obj |
| Single | 一个介于+/-3.462823E38 之间的小数 | sn |
| String | 正文字符串可以达到 20 亿字符长 | S |
| 自户自定义类型 | 一个用 Type 语句描述的由以上一种或多种数据类型组合成的数据类型 | U |
| Variant | 除了用户自定义类型外的上面任何类型 | V |

除了固有的数据类型，Excel 中的 Visual Basic 为对象和集合定义了许多类型。变量通常引用 Excel 中的对象，因此为对象类型使用一组前缀非常重要。下表给出了 Excel 对象的对象和集合前缀。

| 对象类型 | 对象前缀 | 集合前缀 |
|---|---|---|
| AddIn | add | adds |
| Application | app | N/A |
| Arc | arc | arcs |
| Areas | N/A | areas |
| Axis | ax | axs |
| AxisTitle | axt | N/A |
| Border | brd | brds |
| Button | but | buts |
| Characters | N/A | chars |
| Chart | chrt | chrts |
| ChartArea | chrta | N/A |
| ChartGroup | chrtg | chrtgs |
| ChartObject | chrtobj | chrtobjs |
| ChartTitle | chrtt | N/A |
| ChectBox | chk | chks |
| Corners | N/A | cors |
| DialogLabel | dtl | dtls |
| Dialog | dlg | dlgs |
| DialogFrame | dlgf | N/A |
| DialogSheet | dlgsht | dlgshts |

（续）

| 对象类型 | 对象前缀 | 集合前缀 |
|---|---|---|
| DownBars | N/A | dbars |
| Drawing | drw | drws |
| DrawingObjects | N/A | drwobjs |
| DropDown | ddwn | ddwns |
| DropLines | N/A | drplns |
| EditBox | edt | edts |
| ErrorBars | N/A | errbars |
| Floor | flr | N/A |
| Font | fnt | N/A |
| Gridlines | N/A | grds |
| GroupBox | gbox | gboxs |
| GroupObject | grp | grps |
| HiLoLines(Legend) | N/A | hls |
| Interior | interior | N/A |
| Label | lbl | lbls |
| Legent | lgd | N/A |
| LengentEntry | lgden | lgdens |
| LengentKey | lgdkey | N/A |
| Line | ln | lns |
| ListBox | lbox | lboxs |

（续）

| 对象类型 | 对象前缀 | 集合前缀 |
|---|---|---|
| Mailer | mail | N/A |
| Menu | mnu | mnus |
| MenuBar | mnubr | mnubrs |
| MenuItem | mnuit | mnuits |
| Module | mod | mods |
| Name | name | names |
| OLEObject | oobj | oobjs |
| OptionButton | opt | opts |
| Outline | outline | N/A |
| Oval | oval | ovals |
| PageSetup | pgsetup | N/A |
| Pane | pane | panes |
| Picture | pict | picts |
| PivotField | pvtfld | pvtflds |
| PivotItem | pvtitm | pvtitms |
| PivotTable | pvt | pvts |
| PlotArea | plot | N/A |
| Point | point | points |
| Range | rng | N/A |
| Rectangle | rect | rects |

（续）

| 对象类型 | 对象前缀 | 集合前缀 |
|---|---|---|
| RoutingSlip | rslip | N/A |
| Scenario | scen | scens |
| ScrollBar | sbar | sbars |
| Series | series | seriesc |
| SeriesLines | N/A | serieslns |
| Sheets | N/A | shts |
| SoundNote | sound | N/A |
| Spinner | spin | spins |
| Style | style | styles |
| TextBox | tbox | tboxs |
| TickLabels | N/A | tlbls |
| Toolbar | tbar | tbars |
| ToolbarButton | tbarb | tbarbs |
| Trendline | trend | trends |
| UpBars | N/A | ubars |
| Walls | N/A | walls |
| Window | win | wins |
| Workbook | wb | wbs |
| Worksheet | wsht | wshts |

虽然有些前缀看起来很长，但它们对大多数使用的对象已是尽可能短了。

## D.3 选择有意义的名字

变量,常量和程序的名字将告诉你关于它们的一些情况。对于对象以及有全局和模块作用域的程序或变量,这一点特别重要。

例如,一个涉及标题为 OK 的按钮的变量应命名为 butOK。一个对数组排序的程序应命名为 SortArray。

对于经常使用的事物使用简短的名字很方便。例如,iCount 对于一个 For... Next 循环中的一个计数器是个合适的名字。许多人喜欢用 i 或 j 来表示 For... Next 循环中的计数器。

如果一个名字看起来太长,最好减少描述而用一串缩写来表示,你可在该项被声明处随时加一条注释来给该项进行完全的描述。

难于给一个程序命名有时提示我们程序太综合或太复杂,如果你不能顺利地给一个程序赋予有意义的名字,你应考虑将它分成几个小块。下表给出了在通常情况下建议使用或避免使用的名字。

| 使用 | 不使用 | 原因 |
|---|---|---|
| iCount,i,j,Index | vCounter,LoopCounter | 用在 For... Next 和 Do... Loop 语句中的 Counter 应该易键入和易识别。使用整型变量可加快循环的执行。 |
| SortArray, DisplayResult, AdjustMargins | DoThings, ShowIt, FormatOutputForPrinting | 过程名应该具有描述性。使用 It 和 Thing 太一般化。 |
| butOK,chrtInventory | Botton1,chrtInv,rMyData,rngSourceData | 对象变量应该具有一致的名字并表达的很清楚。 |

## D.4 格式代码

用制表可表示几块代码间的相互联系。对于有开始和结尾的命令,如循环,对每一级循环,其指令内容应缩进一级。

```
' Takes a string and reverses it.
Sub ReverseString(vInput)
     ' Begin body of Sub..End Sub, so indent once.
     If TypeName(vInput) = "String" Then
          ' Begin body of If...Then, so indent again.
          Dim sOutput As String, iCount As Integer
          For iCount = Len(vInput) To 1 Step -1
               ' Begin body of For...Next loop, so indent again.
               sOutput = sOutput & Mid$(vInput, iCount, 1)
          Next iCount
          vInput = sOutput
     End If
End Sub
```

在这个例子中,很容易看到循环和条件语句从哪开始和结束,对于大段的条件代码,如很长的 Select Case 语句或一系列 If... Then 语句,这一点更重要。

## D.5 注释代码

好的注释对于编写可以让别人明白和维护的代码来说是最重要的步骤。如果编程中偶尔远行，给代码加注释也至关重要。你可以在编写程序同时给它加注释，以下为通用格式：

1. 程序描述——这部分告诉人们该程序做什么，提及它使用的全局或模块级变量，并描述参数及返回值(如果有的话)，你也可考虑让该部分包括作者姓名和版本号。
2. 变量声明——这部分描述程序中使用的关键变量。
3. 有意义的注释——这部分叙述代码做什么，不需要用英文对每行代码所做工作均进行解释，只需简要地指明所执行的动作及代码中做的决定。
4. 设想及不能完成的标志——这些是指示你还没有完成的工作的标志，你也许想在一个程序被确认无错之前先将它注释掉并且运行程序，如果你想这么做，就给你的设想加上标志(你也许将它们用大写字母表示)，这使你在给老板演示软件之后很容易地返回来继续工作。

下面代码给出了以上 4 条规则如何实现的例子，其中粗体行涉及了每个规则。

```
' (1) Procedure Description
  Adds a button to the Standard toolbar when this workbook is opened.
  Use this commandline to install button:
      EXCEL.EXE VISIO.XLS
  Make sure "InsertVisioDrawing" macro is available (see VISIO.XLA).
  Written by: A. Wombat
  Revisions: None.
  The bitmap picture embedded on Sheet 1 determines the picture _
  that appears on the button
Sub Auto_Open()
' (2) Variable Declarations
Dim tbutCount As ToolbarButton, tbutVisio As ToolbarButton
Application.ScreenUpdating = False
' (3) Descriptive Comments
' Check if toolbar button already exists
For Each tbutCount In Toolbars("Standard").ToolbarButtons
    If tbutCount.Name = "Insert Visio drawing" Then Exit Sub
Next tbutCount
' Add a blank toolbar button to the Standard toolbar.
Toolbars("Standard").ToolbarButtons.Add Button:=231
' Create an object variable for the button (it is the last _
  button on the toolbar).
Set tbutVisio = Toolbars("Standard").ToolbarButtons _
(Toolbars("Standard").ToolbarButtons.Count)
' Set the macro the toolbar button will run.
tbutVisio.OnAction = "InsertVisioDrawing"
' Give the button a name to display in the Tool Tip balloon.
tbutVisio.Name = "Insert Visio drawing"
' Copy the bitmap.
Sheets("Sheet1").DrawingObjects _
("Visio Toolbar Button").CopyPicture
' Paste it into the button.
tbutVisio.PasteFace
Application.ScreenUpdating = True
' Close this workbook.
```

```
    (4) Assumptions and Undones.
    ' UNDONE: Uncomment this line to cause workbook to _
      automatically close.
    ' Makes it hard to edit this macro, though!
    'ActiveWorkbook.Close
End Sub
```

## D.6 变量声明

Visual Basic 让你以非常自由的格式工作，你甚至不必关心变量可用的空间或指定正确数据类型就可编写程序。这使得开始学习 Visual Basic 非常容易，并且意味着你几乎可以立即编写有用的程序。然而，这也意味着很难找出错误的变量名。

例如，下面代码不显示消息框，因为在 If 语句中，bAnswer 拼写错了。Visual Basic 认为“bAnwser”是一个新的变量并给它初始化为空：

```
Sub SpellingError()
    bAnswer=True
    If bAnwser=True Then MsgBox("True") ' Oops! Message box_
    never displays.
End Sub
```

如果你在模块的开头加一条 Option Explicit 语句，那么 Visual Basic 进行检查以保证你使用的每个变量已被定义。这是对你编写的所有代码中的变量进行拼写检查的有效方法。

这有一段带有 Option Explicit 语句的与上段代码相同的代码：

```
Option Explicit

Sub SpellingError()
    Dim bAnwser As Boolean ' This line is now required.
    bAnswer=True
    If bAnswer=True Then MsgBox("True") ' This line causes an_
    error when run!
End Sub
```

上述代码还有错误，但当你试图运行程序时，Visual Basic 警告你使用了未经定义的变量。

使用 Option Explicit 是更好的习惯，在需编写 Visual Basic 软件时，合同订立者经常需要它。

## D.7 总结

保持对变量和子程序命名的一致性。在适合用有意义的名字的地方，用前缀来识别作用域和数据类型，根据前缀，很容易知道 iCount 是一个整数型变量，但单独一个 X 则不能推断出其含义。

缩进所有相关的代码块。这一简单规则使用户容易看出哪几行是一起的。经常对一块程序，包括子程序提供注释，也对所有复杂逻辑注释。记住，你可能一周、一月或一年后会再看这段程序。

在一个模块中使用 Option Explicit 语句有助于消灭键入错误及其相关的逻辑错误。