Excel_VBA 编程常用实例（150 例）

主要内容和特点

《ExcelVBA 编程入门范例》主要是以一些基础而简短的 VBA 实例来对 ExcelVBA 中的常用对象及其属性和方法进行讲解，包括应用程序对象、窗口、工作簿、工作表、单元格和单元格区域、图表、数据透视表、形状、控件、菜单和工具栏、帮助助手、格式化操作、文件操作、以及常用方法和函数及技巧等方面的应用示例。这些例子都比较基础，很容易理解，因而，很容易调试并得以实现，让您通过具体的实例来熟悉 ExcelVBA 编程。

■ 分 16 章共 14 个专题，以具体实例来对大多数常用的 ExcelVBA 对象进行讲解；

■ 一般而言，每个实例都很简短，用来说明使用 VBA 实现 Excel 某一功能的操作；

■ 各章内容主要是实例，即 VBA 代码，配以简短的说明，有些例子可能配以必要的图片，以便于理解；

■ 您可以对这些实例进行扩充或组合，以实现您想要的功能或更复杂的操作。

VBE 编辑器及 VBA 代码输入和调试的基本知识

在学习这些实例的过程中，最好自己动手将它们输入到 VBE 编辑器中调试运行，来查看它们的结果。当然，您可以偷赖，将它们复制/粘贴到代码编辑窗口后，进行调试运行。下面，对 VBE 编辑器界面进行介绍，并对 VBA 代码输入和调试的基本知识进行简单的讲解。

激活 VBE 编辑器

一般可以使用以下三种方式来打开 VBE 编辑器：

■ 使用工作表菜单“工具——宏——Visual Basic 编辑器”命令，如图 00-01 所示；

■ 在 Visual Basic 工具栏上，按“Visual Basic 编辑器”按钮，如图 00-02 所示；

■ 按 Alt+F11 组合键。

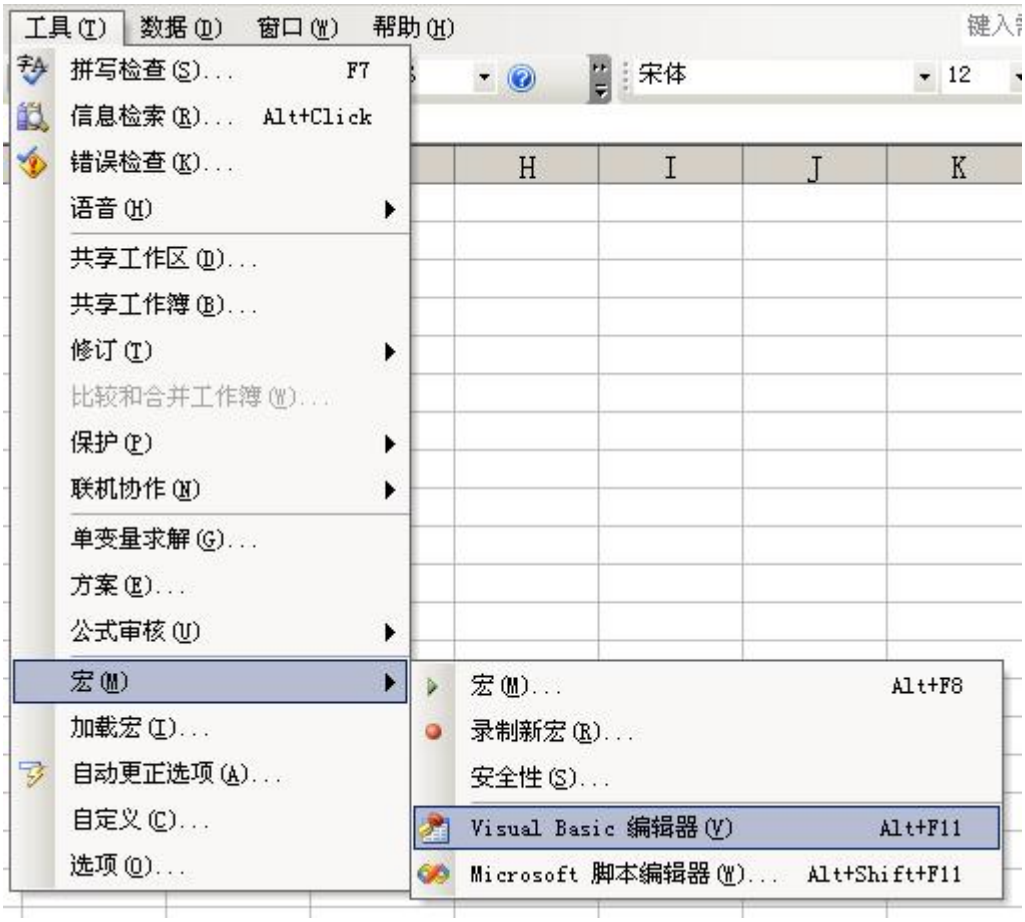


图 00-01：选择菜单“工具——宏——Visual Basic 编辑器”命令来打开 VBE 编辑器

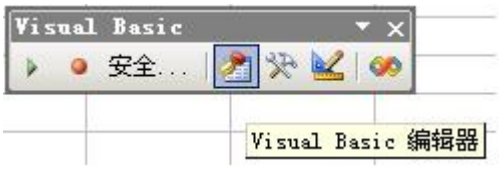


图 00-02：选择 Visual Basic 工具栏上的“Visual Basic 编辑器”命令按钮来打开 VBE 编辑器

此外，您也可以使用下面三种方式打开 VBE 编辑器：

■ 在任一工作表标签上单击鼠标右键，在弹出的菜单中选择“查看代码”，则可进入 VBE 编辑器访问该工作表的代码模块，如图 00-03 所示；

■ 在工作簿窗口左上角的 Excel 图标上单击鼠标右键，在弹出的菜单中选择“查看代码”，则可进入 VBE 编辑器访问活动工作簿的 ThisWorkbook 代码模块，如图 00-04 所示；

■ 选择菜单“工具——宏——宏”命令打开宏对话框，若该工作簿中有宏程序，则单击该对话框中的“编辑”按钮即可进行 VBE 编辑器代码模块，如图 00-05 所示。

图 00-03：右击工作表标签弹出菜单并选择“查看代码”打开 VBE 编辑器

图 00-04：右击 Excel 图标弹出菜单并选择“查看代码”打开 VBE 编辑器

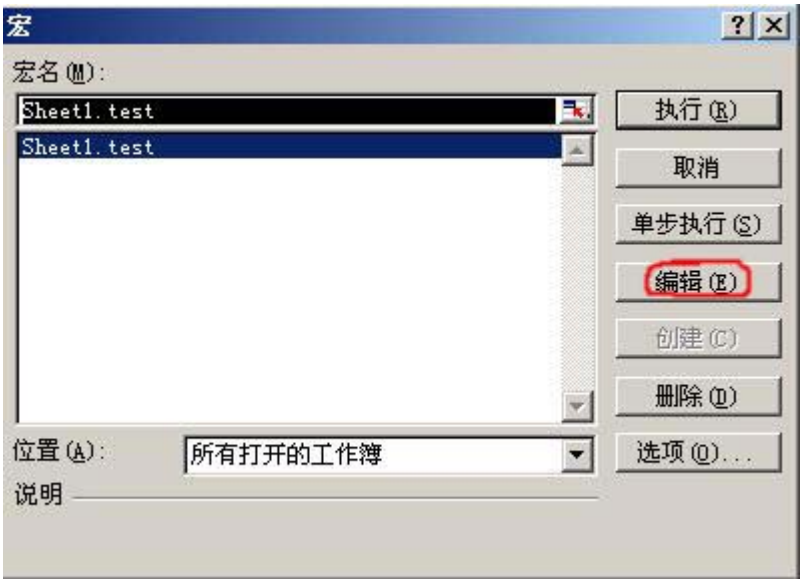


图 00-05：在宏对话框中单击“编辑”按钮打开 VBE 编辑器

VBE 编辑器窗口简介

刚打开 VBE 编辑器时，所显示的窗口如图 00-06 所示，其中没有代码模块窗口。

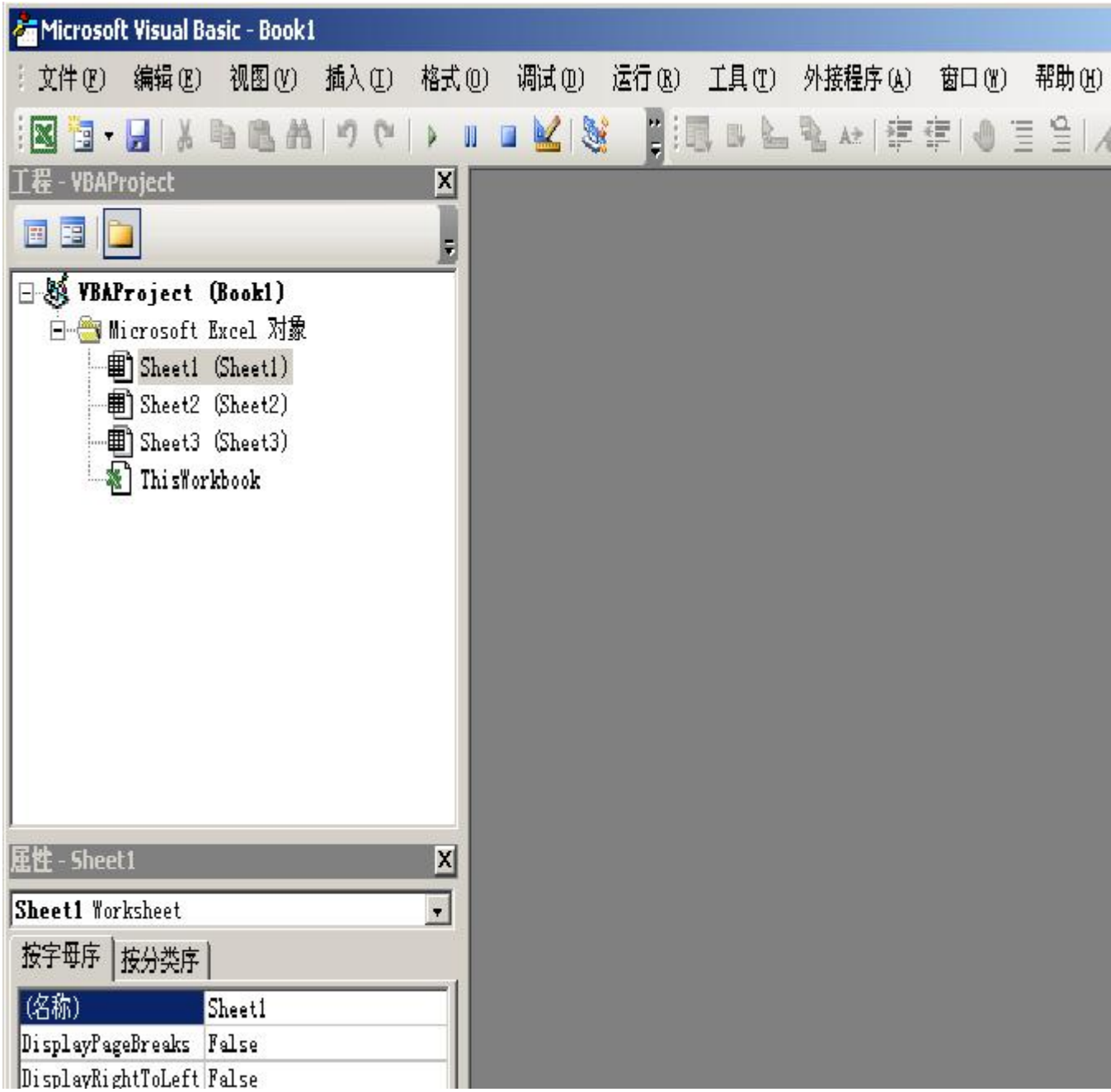


图 00-06：刚打开 VBE 编辑器时的窗口

可以在“工程资源管理器”中双击任一对象打开代码窗口，或者选择菜单“插入——模块”或“插入——类模块”来打开代码窗口。一般 VBE 编辑器窗口及各组成部件名称如图 00-07 所示，可以通过“视图”菜单中的菜单项选择所出现的窗口。同时，可以在“工程属性”窗口中设置或修改相应对象的属性。

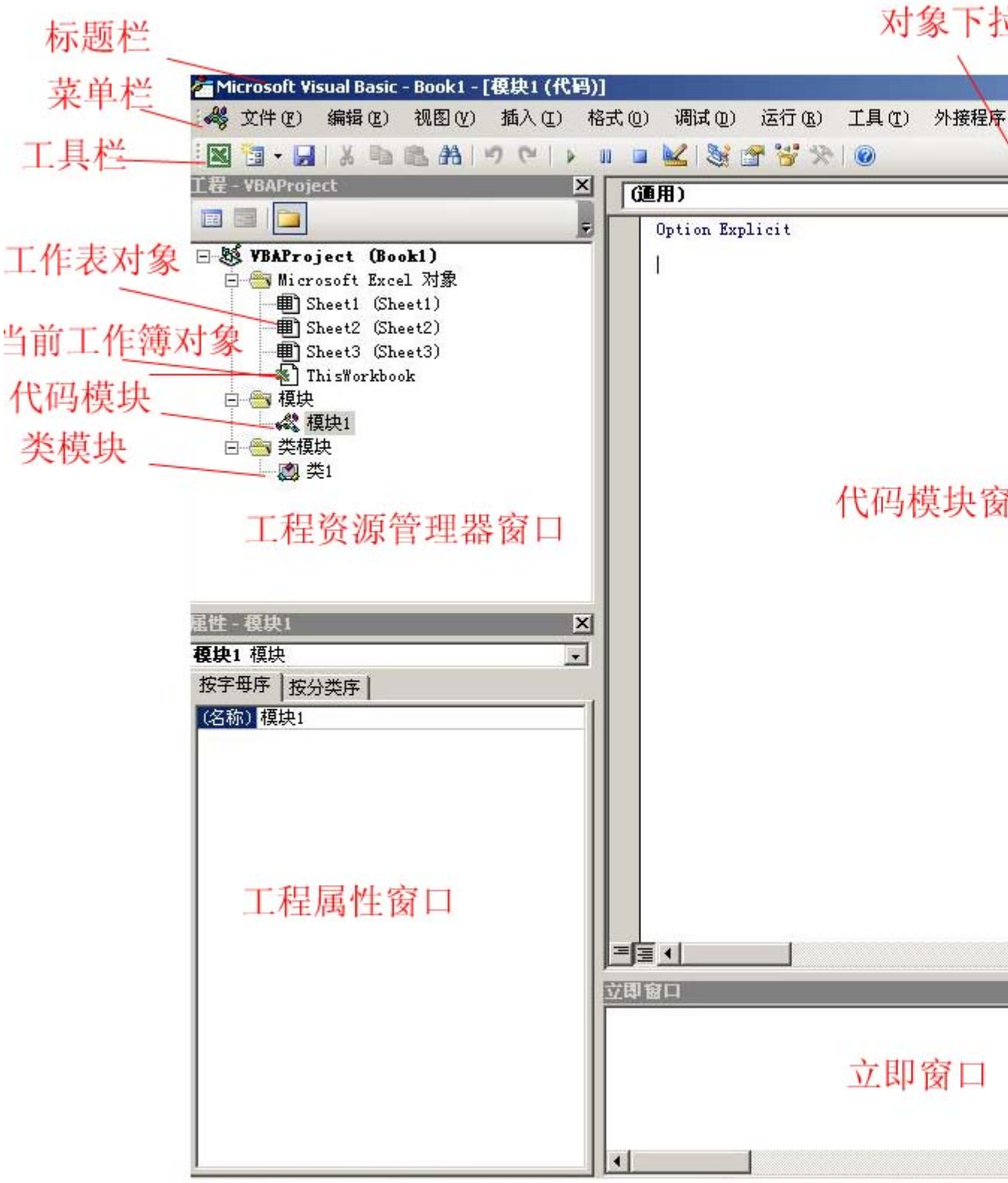


图 00-07：VBE 编辑器窗口

下面是带有用户窗体的 VBE 编辑器窗口，如图 00-08 所示。选择 VBE 菜单“插入——用户窗体”，即可插入一个用户窗体。当插入用户窗体后，在“工程资源管

理器”窗口中会出现一个用户窗体对象，“工程属性”窗口显示当前用户窗体的属性，可对相关属性进行设置或修改。同时，在用户窗体上用鼠标单击，会出现“控件工具箱”。在“工程资源管理器”窗口双击用户窗体图标，会出现相应的用户窗体；在用户窗体图标或者是在用户窗体上单击鼠标右键，然后在弹出的菜单中选择“查看代码”，则会出现用户窗体代码窗口。

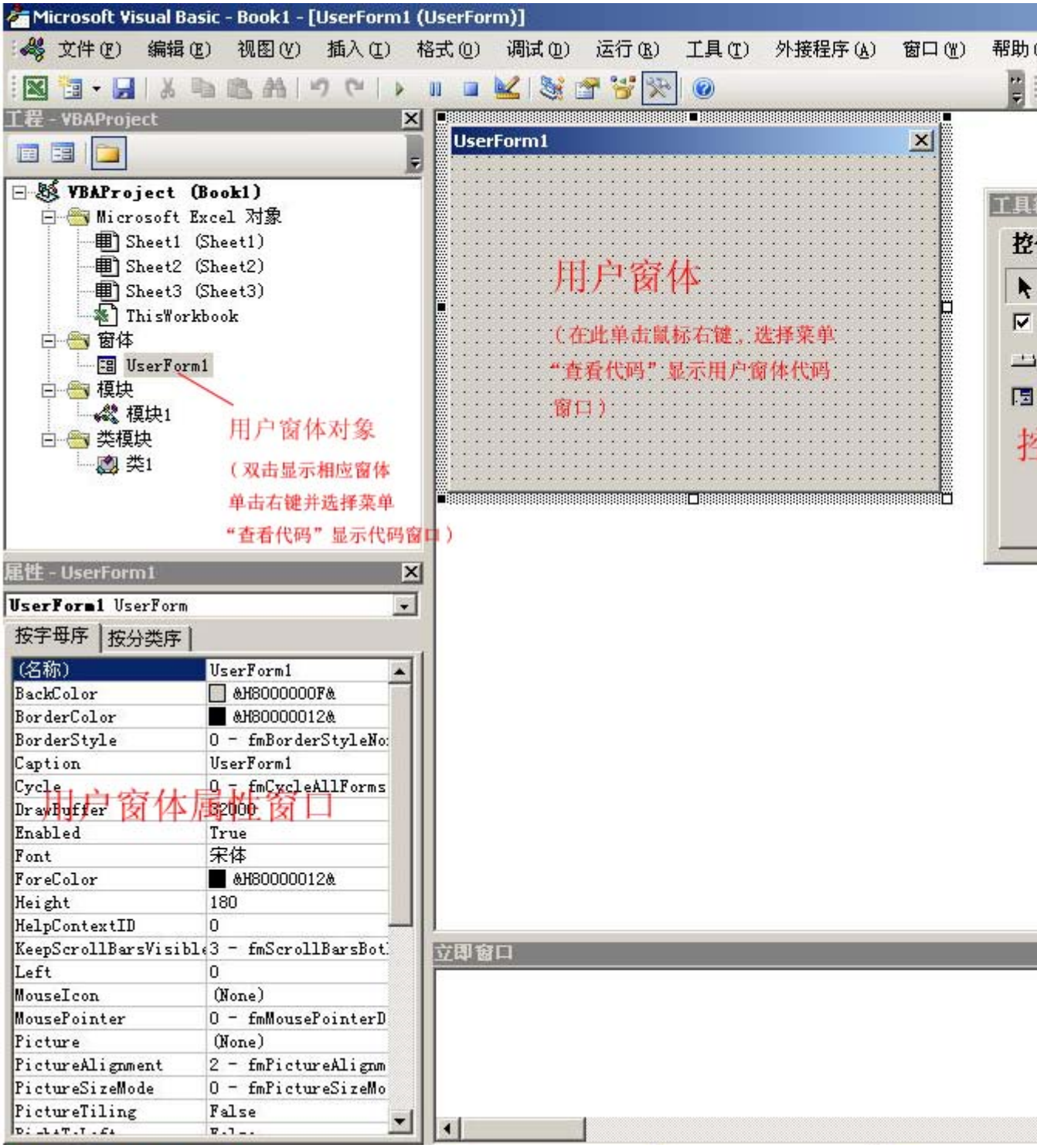


图 00-08：VBE 编辑器窗口(带有用户窗体)

在 VBE 编辑器中输入 VBA 代码

如前所述，您可以选择 VBE 菜单“插入——用户窗体/模块/类模块”来插入模块或用户窗体以及相应的代码窗口。此外，您也可以在“工程资源管理器”中单击鼠标右键，从弹出的菜单中选择“插入——用户窗体/模块/类模块”来实现上面的操作。在获取相应的代码模块窗口后，就可以输入 VBA 代码了。

在 VBE 编辑器的代码模块中输入 VBA 代码，通常有以下几种方法：

■ 手工键盘输入；

■ 使用宏录制器，即选择菜单“工具——宏——录制新宏”命令，将所进行的操作自动录制成宏代码；

■ 复制/粘贴代码，即将现有的代码复制后，粘贴到相应的代码模块中；

■ 导入代码模块，即在 VBE 编辑器中选择菜单“文件——导入文件”或在“工程资源管理器”的任一对象上右击鼠标选择菜单“导入文件”，选择相应的代码文件导入。

如果不想要某个模块了，可以选择菜单“文件——移除模块”，也可以在相应的模块上单击鼠标右键，从弹出的菜单中选择“移除模块”。此时，会弹出一个警告框，询问在移除模块前是否将其导出，可以根据需要进行选择。

也可以选择菜单“文件——导出文件”或在相应的模块上单击鼠标右键后，从弹出的菜单中选择“导出文件”，将移除的模块保存在相应的文件夹中。这样，以后可以对其进行导入，从而加以利用。

调试 VBA 代码

在 VBE 编辑器的菜单中，有两项与调试运行有关的菜单项，即“调试”菜单和“运行”菜单，它们提供了各种调试和运行的手段。在我现阶段进行代码调试时，常用到的有以下几个：

■ 逐语句。可以按 F8 键对代码按顺序一条一条语句运行，从而找出语句或逻辑错误。

■ 设置断点。在可能存在问题的语句处设置断点(可通过在相应代码前的空白部位单击，将会出现一个深红色的椭圆即断点)，当程序运行至断点处时，会中止运行。

■ 在语句的适当部位设置 Debug.Print 语句，运行后其结果会显示在“立即窗口”中，可以此测试或跟踪变量的值。

■ 在“立即窗口”中测试。对值的测试或跟踪，也可以以“？”开头，在“立即窗口”中输入需要测试值的语句，按 Enter 回车键后将立即出现结果；对执行语句的测试，可直接在“立即窗口”中输入，按 Enter 回车键后将执行。

■ 可以按 F5 键直接运行光标所在位置的子程序。

在执行程序后，必须在 Excel 工作表中查看所得到的结果。可以用鼠标单击 VBE 编辑器左上角的 Excel 图标或者是按 Alt+F11 组合键切换到 Excel 界面。

(当然，对程序代码的调试有很多方法和技巧，留待以后对 VBA 进一步研究和理解更透彻后一并讨论。)

利用 VBA 帮助系统

如果遇到疑问或错误，可以利用 Excel 自带的 VBA 帮助系统。

■ 可以在如图 00-09 所示的部位输入需要帮助的关键词，按 Enter 回车键后将会出现相关主题。用鼠标单击相应的主题即会出现详细的帮助信息。

图 00-09：帮助搜索窗口

■ 可以按 F2 键，调出“对象浏览器”窗口(如图 00-10 所示)，在搜索文本框中输入需要帮助的关键词，将会在“搜索结果”中出现一系列相关的对象及方法、属性

列表，单击相应的对象则会在“类”和“成员”列表框中显示相应的对象和方法、属性成员列表，在成员列表中相应的项目上按 F1 键即会出现详细的帮助信息。(“对象浏览器”是一个很好的帮助工具，值得好好研究)

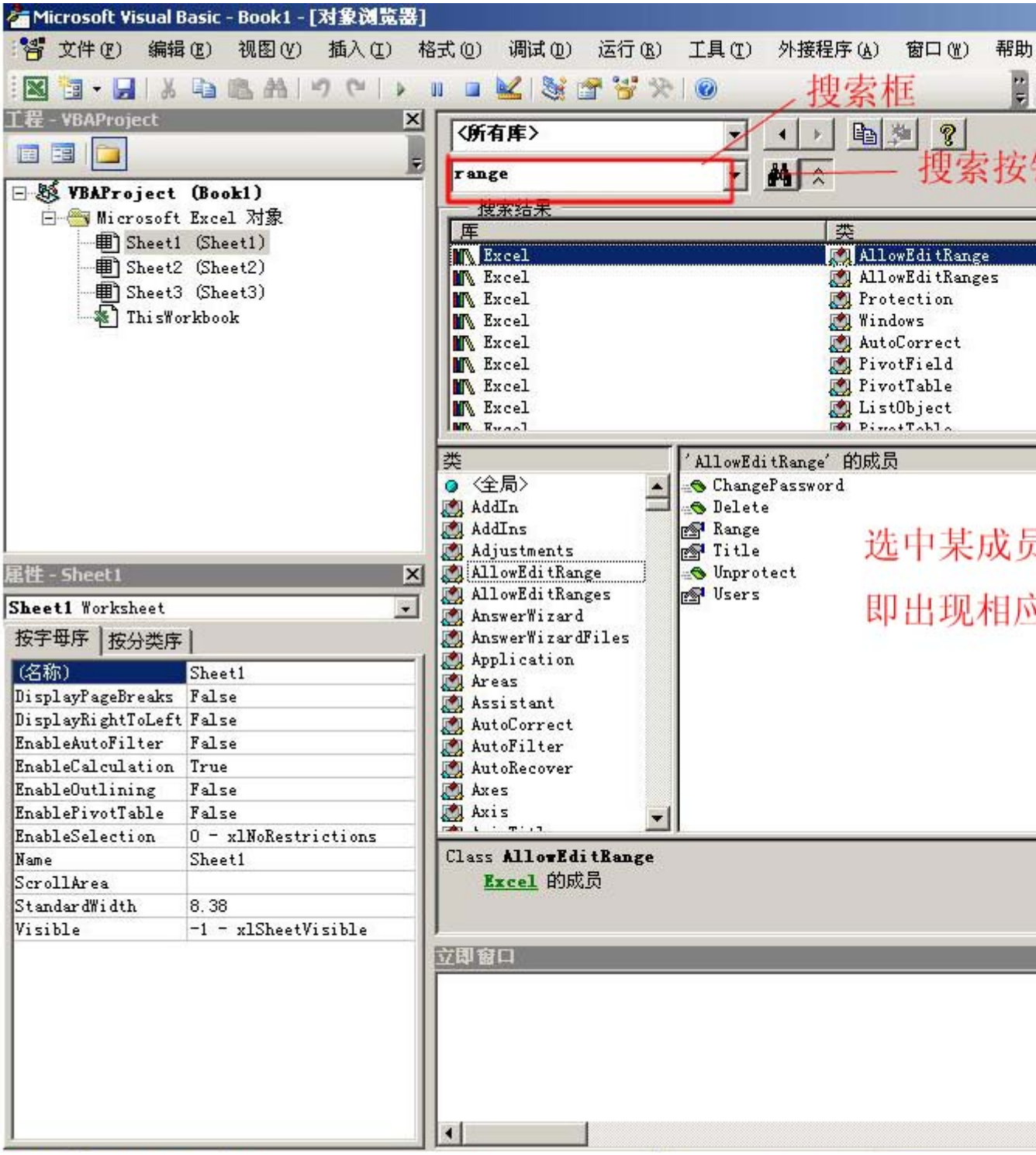


图 00-10：对象浏览器窗口

参考资料

《ExcelVBA 编程入门范例》参考或引用了以下书籍和资料：

(1)Excel 2003 高级 VBA 编程宝典

(2)Excel 2003 与 VBA 编程从入门到精通(中文版)
(3)巧学巧用 Excel 2003 VBA 与宏(中文版)
(4)ExcelVBA 应用程序专业设计实用指南
(5)ExcelVBA 应用开发与实例精讲
(6)一些网上资源

更多的信息

关于 ExcelVBA 的更多参考和学习资源，可以在 www.excelhome.net 上查找，有疑问也可以在 ExcelHome 论坛中提问。您也可以登录我的博客 http://fanjy.blog.excelhome.net，上面有很多 Excel 的学习资料。同时，欢迎与我联系交流，我的 e-mail 是：xhdsxfjy@163.com。

“学习 Excel，使用 VBA 对 Excel 进行控制操作是我很热衷的业余爱好之一。”
——fanjy

第一章 Excel 应用程序对象(Application 对象)及其常用方法

基本操作应用示例

分类:ExcelVBA>>ExcelVBA 编程入门范例

Application 对象代表整个 Microsoft Excel 应用程序，带有 175 个属性和 52 个方法，可以设置整个应用程序的环境或配置应用程序。

示例 01-01：体验开/关屏幕更新(ScreenUpdating 属性)

```
Sub 关闭屏幕更新()
  MsgBox "顺序切换工作表 Sheet1→Sheet2→Sheet3→Sheet2，先开启屏幕更新，然后关闭屏幕更新"
  Worksheets(1).Select
  MsgBox "目前屏幕中显示工作表 Sheet1"
  Application.ScreenUpdating = True
  Worksheets(2).Select
  MsgBox "显示 Sheet2 了吗？"
  Worksheets(3).Select
  MsgBox "显示 Sheet3 了吗？"
  Worksheets(2).Select
  MsgBox "下面与前面执行的程序代码相同,但关闭屏幕更新功能"
  Worksheets(1).Select
  MsgBox "目前屏幕中显示工作表 Sheet1" & Chr(10) & "关屏屏幕更新功能"
  Application.ScreenUpdating = False
  Worksheets(2).Select
```

```
  MsgBox "显示 Sheet2 了吗？"
  Worksheets(3).Select
  MsgBox "显示 Sheet3 了吗？"
  Worksheets(2).Select
  Application.ScreenUpdating = True
End Sub
```

示例说明：ScreenUpdating 属性用来控制屏幕更新。当运行一个宏程序处理涉及到多个工作表或单元格中的大量数据时，若没有关闭屏幕更新，则会占用 CPU 的处理时间，从而降低程序的运行速度，而关闭该属性则可显著提高程序运行速度。

示例 01-02：使用状态栏(StatusBar 属性)

```
Sub testStatusBar()
  Application.DisplayStatusBar = True '开启状态栏显示
  '赋值状态栏显示的文本
  Application.StatusBar = "http://fanjy.blog.excelhome.net"
End Sub
```

示例说明：StatusBar 属性用来指定显示在状态栏上的信息。若不想再显示状态栏文本，可使用 `Application.StatusBar = False` 语句关闭状态栏显示，也可以在程序开始将原先的状态栏设置存储，如使用语句 `oldStatusBar = Application.DisplayStatusBar` 将状态栏原来的信息存储在变量 oldStatusBar，在程序运行完成或退出时，将变量重新赋值给状态栏，如使用语句 `Application.DisplayStatusBar = oldStatusBar`，以恢复状态栏原状。

示例 01-03：处理光标(Cursor 属性)

```
Sub ViewCursors()
  Application.Cursor = xlNorthwestArrow
  MsgBox "您将使用箭头光标,切换到 Excel 界面查看光标形状"
  Application.Cursor = xlIBeam
  MsgBox "您将使用工形光标,切换到 Excel 界面查看光标形状"
  Application.Cursor = xlWait
  MsgBox "您将使用等待形光标,切换到 Excel 界面查看光标形状"
  Application.Cursor = xlDefault
  MsgBox "您已将光标恢复为缺省状态"
End Sub
```

示例 01-04：获取系统信息

```
Sub GetSystemInfo()
  MsgBox "Excel 版本信息为:" & Application.CalculationVersion
  MsgBox "Excel 当前允许使用的内存为:" & Application.MemoryFree
  MsgBox "Excel 当前已使用的内存为:" & Application.MemoryUsed
  MsgBox "Excel 可以使用的内存为:" & Application.MemoryTotal
  MsgBox "本机操作系统的名称和版本为:" & Application.OperatingSystem
  MsgBox "本产品所登记的组织名为:" & Application.OrganizationName
```

```
  MsgBox "当前用户名为:" & Application.UserName
  MsgBox "当前使用的 Excel 版本为:" & Application.Version
End Sub
```

示例说明：可以使用给 UserName 属性赋值以设置用户名称。

示例 01-05：退出复制/剪切模式(CutCopyMode 属性)

```
Sub exitCutCopyMode()
  Application.CutCopyMode = False
End Sub
```

示例说明：退出复制/剪切模式后，在程序运行时所进行的复制或剪切操作不会在原单元格区域留下流动的虚框线。需要提醒的是，在程序运行完后，应使用 Application.CutCopyMode = False 语句恢复该属性的默认设置。

示例 01-06：禁止弹出警告信息(DisplayAlerts 属性)

```
Sub testAlertsDisplay()
  Application.DisplayAlerts = False
End Sub
```

示例说明：在程序运行过程中，有时由于 Excel 本身设置的原因，会弹出对话框，从而中断程序的运行，您可以在程序之前加上 Application.DisplayAlerts = False 语句以禁止弹出这些对话框而不影响程序正常运行。需要注意的是，在程序运行结束前，应使 DisplayAlerts 属性恢复为缺省状态，即使用语句 Application.DisplayAlerts = True。该属性的默认设置为 True，当将该属性设置为 False 时，Excel 会使直接使用对话框中默认的选择，从而不会因为弹出对话框而影响程序运行。

示例 01-07：将 Excel 全屏幕显示

```
Sub testFullScreen()
  MsgBox "运行后将 Excel 的显示模式设置为全屏幕"
  Application.DisplayFullScreen = True
  MsgBox "恢复为原来的状态"
  Application.DisplayFullScreen = False
End Sub
```

示例 01-08：Excel 启动的文件夹路径

```
Sub ExcelStartfolder()
  MsgBox "Excel 启动的文件夹路径为：" & Chr(10) & Application.StartupPath
End Sub
```

示例 01-09：打开最近使用过的文档

```
Sub OpenRecentFiles()
  MsgBox "显示最近使用过的第三个文件名,并打开该文件"
  MsgBox "最近使用的第三个文件的名称为:" & Application.RecentFiles(3).Name
  Application.RecentFiles(3).Open
```

```
End Sub
```

示例 01-10：打开文件(FindFile 方法)

```
Sub FindFileOpen()
  On Error Resume Next
  MsgBox "请打开文件", vbOKOnly + vbInformation, "打开文件"
  If Not Application.FindFile Then
    MsgBox "文件未找到", vbOKOnly + vbInformation, "打开失败"
  End If
End Sub
```

示例说明：本示例将显示“打开”文件对话框，若用户未打开文件(即点击“取消”按钮)，则会显示“打开失败”信息框。示例中使用了 FindFile 方法，用来显示“打开”对话框并让用户打开一个文件。如果成功打开一个新文件，则该值为 True。如果用户取消了操作并退出该对话框，则该值为 False。

示例 01-11：文件对话框操作(FileDialog 属性)

```
Sub UseFileDialogOpen()
  Dim lngCount As Long
  '开启"打开文件"对话框
  With Application.FileDialog(msoFileDialogOpen)
    .AllowMultiSelect = True
    .Show
    '显示所选的每个文件的路径
    For lngCount = 1 To .SelectedItems.Count
      MsgBox .SelectedItems(lngCount)
    Next lngCount
  End With
End Sub
```

示例说明：本示例显示“打开文件”对话框，当用户在其中选择一个或多个文件后，将依次显示每个文件的路径。其中，FileDialog 属性返回打开和保存对话框中一系列对象的集合，您可以对该集合对象的属性进行进一步的设置，如上例中的 AllowMultiSelect 属性设置为 True 将允许用户选择多个文件。

示例 01-12：保存 Excel 的工作环境

```
Sub 保存 Excel 的工作环境()
  MsgBox "将 Excel 的工作环境保存到 D:\ExcelSample\中"
  Application.SaveWorkspace "D:\ExcelSample\Sample"
End Sub
```

示例说明：运行本程序后，将工作簿以带后缀名.xlw 保存到 D 盘的 ExcelSample 文件夹中，生成的文件全名为 Sample.xlw。当改变工作簿并保存时，Excel 会询问是覆盖原文件还是保存副本。

示例 01-13：改变 Excel 工作簿名字(Caption 属性)

```
Sub SetCaption()
```

```
  Application.Caption = "My ExcelBook"
End Sub
```
示例说明：运行本程序后，将工作簿左上角 Excel 图标右侧的“Microsoft Excel”改为“My ExcelBook”。

示例 01-14：使用 InputBox 方法

```
Sub SampleInputBox()
  Dim vInput
  vInput = InputBox("请输入用户名:", "获取用户名", Application.UserName)
  MsgBox "您好!" & vInput & ".很高兴能认识您.", vbOKOnly, "打招呼"
End Sub
```

示例 01-15：设置页边距(CentimetersToPoints 方法)

```
Sub SetLeftMargin()
  MsgBox "将工作表 Sheet1 的左页边距设为 5 厘米"
  Worksheets("Sheet1").PageSetup.LeftMargin = Application.CentimetersTo
Points(5)
End Sub
```

示例 01-16：使用 Windows 的计算器(ActivateMicrosoftApp 方法)

```
Sub CallCalculate()
  Application.ActivateMicrosoftApp Index:=0
End Sub
```
示例说明：运行本程序后，将调用 Windows 的计算器。

示例 01-17：在程序中运行另一个宏(Run 方法)

```
Sub runOtherMacro()
  MsgBox "本程序先选择 A1 至 C6 单元格区域后执行 DrawLine 宏"
  ActiveSheet.Range("A2:C6").Select
  Application.Run "DrawLine"
End Sub
```

示例 01-18：在指定的时间执行宏(OnTime 方法)

```
Sub AfterTimetoRun()
  MsgBox "从现在开始,10 秒后执行程序「testFullScreen」"
  Application.OnTime Now + TimeValue("00:00:10"), "testFullScreen"
End Sub
```
示例说明：运行本程序后，在 10 秒后将执行程序 testFullScreen。

示例 01-19：暂时停止宏运行(Wait 方法)

```
Sub Stop5sMacroRun()
  Dim SetTime As Date
  MsgBox "按下「确定」,5 秒后执行程序「testFullScreen」"
  SetTime = DateAdd("s", 5, Now())
```

```
  Application.Wait  SetTime
  Call  testFullScreen
End  Sub
```
示例说明：运行本程序后，按下弹出的提示框中的“确定”按钮，等待 5 秒后执行另一程序 testFullScreen。

示例 01-20：按下指定的按键后执行程序(OnKey 方法)

[示例 01-20-1]
```
Sub  PressKeytoRun()
  MsgBox  "按下 Ctrl+D 后将执行程序「testFullScreen」"
  Application.OnKey  "^{d}",  "testFullScreen"
End  Sub
```
[示例 01-20-2]
```
Sub  ResetKey()
  MsgBox  "恢复原来的按键状态"
  Application.OnKey  "^{d}"
End  Sub
```
示例说明：Onkey 方法的作用主要是指定特定的键，当按下指定的键时运行相应的宏程序，或者按下指定的键时，使 Excel 屏蔽特定的功能。

示例 01-21：重新计算工作簿

[示例 01-21-1]
```
Sub  CalculateAllWorkbook()
  Application.Calculate
End  Sub
```
示例说明：当工作簿的计算模式被设置为手动模式后，运用 Calculate 方法可以重新计算所有打开的工作簿、工作簿中特定的工作表或者工作表中指定的单元格区域。

[示例 01-21-2]
```
Sub  CalculateFullSample()
  If  Application.CalculationVersion  <>  Workbooks(1).CalculationVersion  The
n
    Application.CalculateFull
  End  If
End  Sub
```
示例说明：本示例先将当前 Microsoft Excel 的版本与上次计算该工作簿的 Excel 版本进行比较，如果两个版本不同，则对所有打开工作簿中的数据进行一次完整计算。其中，CalculationVersion 属性返回工作簿的版本信息。

示例 01-22：控制函数重新计算(Volatile 方法)
```
Function  NonStaticRand()
  '当工作表中任意单元格重新计算时本函数更新
  Application.Volatile  True
  NonStaticRand  =  Rnd()
```

```
End Function
```
示例说明：本示例摸仿 Excel 中的 Rand()函数，当工作表单元格发生变化时，都会重新计算该函数。在例子中，使用了 Volatile 方法，强制函数进行重新计算，即无论何时重新计算工作表，该函数都会重新计算。

示例 01-23：利用工作表函数(WorksheetFunction 属性)

```
Sub WorksheetFunctionSample()
  Dim myRange As Range, answer
  Set myRange = Worksheets("Sheet1").Range("A1:C10")
  answer = Application.WorksheetFunction.Min(myRange)
  MsgBox answer
End Sub
```
示例说明：本示例获取工作表 Sheet1 中单元格区域 A1:C10 中的最小值，使用了工作表函数 Min()。一般，使用 WorksheetFunction 属性引用工作表函数，但如果 VBA 自带有实现相同功能的函数，则直接使用该函数，否则会出现错误。

示例 01-24：获取重叠区域(Intersect 方法)

```
Sub IntersectRange()
  Dim rSect As Range
  Worksheets("Sheet1").Activate
  Set rSect = Application.Intersect(Range("rg1"), Range("rg2"))
  If rSect Is Nothing Then
    MsgBox "没有交叉区域"
  Else
    rSect.Select
  End If
End Sub
```
示例说明：本示例在工作表 Sheet1 中选定两个命名区域 rg1 和 rg2 的重叠区域，如果所选区域不重叠，则显示一条相应的信息。其中，Intersect 方法返回一个 Range 对象，代表两个或多个范围重叠的矩形区域。

示例 01-25：获取路径分隔符(PathSeparator 属性)

```
Sub GetPathSeparator()
 MsgBox "路径分隔符为" & Application.PathSeparator
End Sub
```
示例说明：本示例使用 PathSeparator 属性返回路径分隔符(“\”)。

示例 01-26：快速移至指定位置(Goto 方法)

```
Sub GotoSample()
  Application.Goto Reference:=Worksheets("Sheet1").Range("A154"), _
    scroll:=True
End Sub
```
示例说明：本示例运行后，将当前单元格移至工作表 Sheet1 中的单元格 A154。

示例 01-27：显示内置对话框(Dialogs 属性)

```
Sub DialogSample()
  Application.Dialogs(xlDialogOpen).Show
End Sub
```

示例说明：本示例显示 Excel 的“打开”文件对话框。其中，Dialogs 属性返回的集合代表所有的 Excel 内置对话框。

示例 01-28：退出 Excel(SendKeys 方法)

```
Sub SendKeysSample()
  Application.SendKeys ("%fx")
End Sub
```

示例说明：本示例使用 SendKeys 方法退出 Excel，若未保存，则会弹出提示对话框并让用户作出相应的选择。SendKeys 方法的作用是摸拟键盘输入，如例中的“%fx”表示在 Excel 中同时按下 Alt、F 和 X 三个键。

示例 01-29：关闭 Excel

```
Sub 关闭 Excel()
  MsgBox "Excel 将会关闭"
  Application.Quit
End Sub
```

示例说明：运行本程序后，若该工作簿未保存，则会弹出对话框询问是否保存。

==========

(by fanjy)

第二章　窗口(Window 对象)基本操作应用示例(一)

分类:ExcelVBA>>ExcelVBA 编程入门范例

Window 对象代表一个窗口，约有 48 个属性和 14 个方法，能对窗口特性进行设置和操作。Window 对象是 Windows 集合中的成员，对于 Application 对象来说，Windows 集合包含该应用程序中的所有窗口；对于 Workbook 对象来说，Windows 集合只包含指定工作簿中的窗口。下面介绍一些示例，以演示和说明 Window 对象及其属性和方法的运用。

示例 02-01：激活窗口(Activate 方法)

```
Sub SelectWindow()
  Dim iWin As Long, i As Long, bWin
  MsgBox "依次切换已打开的窗口"
  iWin = Windows.Count
```

```
  MsgBox “您已打开的窗口数量为：” & iWin
  For i = 1 To iWin
    Windows(i).Activate
    bWin = MsgBox("您激活了第 " & i & "个窗口，还要继续吗？", vbYesNo)
    If bWin = vbNo Then Exit Sub
  Next i
End Sub
```

示例 02-02：窗口状态(WindowState 属性)

[示例 02-02-01]

```
Sub WindowStateTest()
  MsgBox "当前活动工作簿窗口将最小化"
  Windows(1).WindowState = xlMinimized
  MsgBox "当前活动工作簿窗口将恢复正常"
  Windows(1).WindowState = xlNormal
  MsgBox "当前活动工作簿窗口将最大化"
  Windows(1).WindowState = xlMaximized
End Sub
```

示例说明：使用 WindowState 属性可以返回或者设置窗口的状态。示例中，常量 xlMinimized、xlNormal 和 xlMaximized 分别代表窗口不同状态值，Windows(1)表示当前活动窗口。可以使用 Windows(index)来返回单个的 Window 对象，其中的 index 为窗口的名称或编号，活动窗口总是 Windows(1)。

[示例 02-02-02]

```
Sub testWindow()
  '测试 Excel 应用程序窗口状态
  MsgBox "应用程序窗口将最大化"
  Application.WindowState = xlMaximized
  Call testWindowState
  MsgBox "应用程序窗口将恢复正常"
  Application.WindowState = xlNormal
  MsgBox "应用程序窗口已恢复正常"
    '测试活动工作簿窗口状态
    MsgBox "当前活动工作簿窗口将最小化"
    ActiveWindow.WindowState = xlMinimized
    Call testWindowState
    MsgBox "当前活动工作簿窗口将最大化"
    ActiveWindow.WindowState = xlMaximized
    Call testWindowState
    MsgBox "当前活动工作簿窗口将恢复正常"
    ActiveWindow.WindowState = xlNormal
    Call testWindowState
  MsgBox "应用程序窗口将最小化"
  Application.WindowState = xlMinimized
  Call testWindowState
```

```
End Sub
'*********************************************************
Sub testWindowState()
  Select Case Application.WindowState
    Case xlMaximized: MsgBox "应用程序窗口已最大化"
    Case xlMinimized: MsgBox "应用程序窗口已最小化"
    Case xlNormal:
      Select Case ActiveWindow.WindowState
        Case xlMaximized: MsgBox "当前活动工作簿窗口已最大化"
        Case xlMinimized: MsgBox "当前活动工作簿窗口已最小化"
        Case xlNormal: MsgBox "当前活动工作簿窗口已恢复正常"
      End Select
  End Select
End Sub
```

示例说明：本示例有两个程序，其中 testWindow()是主程序，调用子程序 textWindowState()，演示了应用程序窗口和工作簿窗口的不同状态。当前活动窗口一般代表当前活动工作簿窗口，读者可以在 VBE 编辑器中按 F8 键逐语句运行 testWindow()程序，观察 Excel 应用程序及工作簿窗口的不同状态。此外，在子程序中，还运用了嵌套的 Select Case 结构。

[示例 02-02-03]

```
Sub SheetGradualGrow()
  Dim x As Integer
  With ActiveWindow
    .WindowState = xlNormal
    .Top = 1
    .Left = 1
    .Height = 50
    .Width = 50
    For x = 50 To Application.UsableHeight
      .Height = x
    Next x
    For x = 50 To Application.UsableWidth
      .Width = x
    Next x
    .WindowState = xlMaximized
  End With
End Sub
```

示例说明：本示例将动态演示工作簿窗口由小到大直至最大化的变化过程。在运行程序时，您可以将 VBE 窗口缩小，从而在工作簿中查看动态效果，也可以在 Excel 中选择菜单中的宏命令执行以查看效果。

示例 02-03：切换显示工作表元素

[示例 02-03-01]

```
Sub testDisplayHeading()
```

```
  MsgBox “切换显示/隐藏行列标号”
  ActiveWindow.DisplayHeadings = Not ActiveWindow.DisplayHeadings
End Sub
```

示例说明：本示例切换是否显示工作表中的行列标号。运行后，工作表中的行标号和列标号将消失；再次运行后，行列标号重新出现，如此反复。您也可以将该属性设置为 False，以取消行列标号的显示，如 ActiveWindow.DisplayHeadings = False；而将该属性设置为 True，则显示行列标号。

[示例 02-03-02]

```
Sub testDisplayGridline()
  MsgBox “切换显示/隐藏网格线”
  ActiveWindow.DisplayGridlines = Not ActiveWindow.DisplayGridlines
End Sub
```

示例说明：本示例切换是否显示工作表中的网格线。运行后，工作表中的网格线消失，再次运行后，网格线重新出现，如此反复。您也可以将该属性设置为 False，以取消网格线显示，如 ActiveWindow.DisplayGridlines = False；而将该属性设置为 True，则显示网格线。

[示例 02-03-03]

```
Sub DisplayHorizontalScrollBar()
  MsgBox “切换显示/隐藏水平滚动条”
  ActiveWindow.DisplayHorizontalScrollBar = _
   Not ActiveWindow.DisplayHorizontalScrollBar
End Sub
```

示例说明：本示例切换是否显示工作表中的水平滚动条。运行后，工作表中的水平滚动条消失，再次运行后，水平滚动条重新出现，如此反复。您也可以将该属性设置为 False，以取消水平滚动条，如 ActiveWindow.DisplayHorizontalScrollBar = False；而将该属性设置为 True，则显示水平滚动条。

同理，DisplayVerticalScrollBar 属性将用来设置垂直滚动条。

[示例 02-03-04]

```
Sub DisplayScrollBar()
  MsgBox "切换显示/隐藏水平和垂直滚动条"
  Application.DisplayScrollBars = Not (Application.DisplayScrollBars)
End Sub
```

示例说明：本示例切换是否显示工作表中的水平和垂直滚动条。运行后，工作表中的水平和垂直滚动条同时消失，再次运行后，水平和垂直滚动条重新出现，如此反复。您也可以将该属性设置为 False，以取消水平和垂直滚动条显示，如 Application.DisplayScrollBars= False；而将该属性设置为 True，则显示水平和垂直滚动条。

示例 02-04：显示公式(DisplayFormulas 属性)

```
Sub DisplayFormula()
  MsgBox “显示工作表中包含公式的单元格中的公式”
  ActiveWindow.DisplayFormulas = True
End Sub
```

示例说明：本程序运行后，工作表中含有公式的单元格将显示公式而不是数值。

若要显示数值，则将该属性设置为 False，或者，如果工作表中的公式显示的是结果数值，则该属性为 False。

示例 02-05：显示/隐藏工作表标签(DisplayWorkbookTabs 属性)

```
Sub testDisplayWorkbookTab()
  MsgBox “隐藏工作表标签”
  ActiveWindow.DisplayWorkbookTabs = False
End Sub
```

示例说明：本程序运行后，工作表标签消失。将该属性设置为 True，重新显示工作表标签。

示例 02-06：命名活动窗口(Caption 属性)

```
Sub testCaption()
  MsgBox "当前活动工作簿窗口的名字是:" & ActiveWindow.Caption
  ActiveWorkbook.Windows(1).Caption = "我的工作簿"
  MsgBox "当前活动工作簿窗口的名字是:" & ActiveWindow.Caption
End Sub
```

示例说明：本程序运行后，显示当前活动工作簿窗口原先的名称(即工作簿窗口未处于最大化状态时，出现在窗口顶部标题栏中的文字)，然后设置当前活动工作簿窗口名称，即使用语句 ActiveWorkbook.Windows(1).Caption = "我的工作簿"，最后显示当前活动工作簿窗口的新名称。改变窗口的标题并不会改变工作簿的名称。

示例 02-07：移动窗口到指定位置(ScrollRow 属性和 ScrollColumn 属性)

```
Sub testScroll()
  MsgBox “将当前窗口工作表左上角单元格移至第 10 行第 3 列”
  ActiveWindow.ScrollRow = 10
  ActiveWindow.ScrollColumn = 3
End Sub
```

示例说明：本程序运行后，当前活动窗口左上角单元格为第 10 行第 3 列。可以通过设置这两个属性来移动窗口到指定的位置，也可以返回指定窗格或窗口最左上面的行号或列号。

示例 02-08：调整窗口(EnableResize 属性)

```
Sub testResize()
  MsgBox “设置窗口大小不可调整”
  ActiveWindow.EnableResize = False
End Sub
```

示例说明：测试本程序前，将当前工作簿窗口恢复为正常状态(即让工作簿标题可见)，运行程序后，当前工作簿窗口将不能调整其大小，右上角的最小化最大化按钮将消失(即隐藏最大化和最小化按钮)。该属性设置为 True，则能调整窗口大小。

示例 02-09：拆分窗格

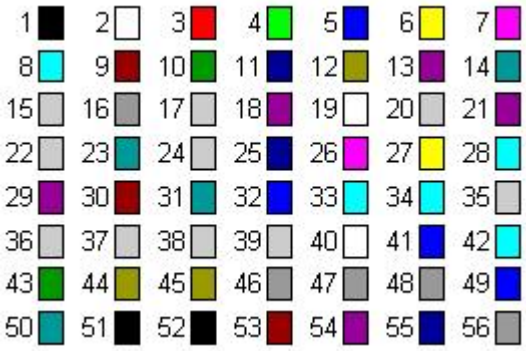
[示例 02-09-01]
```
Sub SplitWindow1()
  Dim iRow As Long, iColumn As Long
  MsgBox "以活动单元格为基准拆分窗格"
  iRow = ActiveCell.Row
  iColumn = ActiveCell.Column
  With ActiveWindow
    .SplitColumn = iColumn
    .SplitRow = iRow
  End With
  MsgBox "恢复原来的窗口状态"
  ActiveWindow.Split = False
End Sub
```
[示例 02-09-02]
```
Sub SplitWindow()
  Dim iRow As Long, iColumn As Long
  MsgBox "以活动单元格为基准拆分窗格"
  iRow = ActiveCell.Row
  iColumn = ActiveCell.Column
  With ActiveWindow
    .SplitColumn = iColumn
    .SplitRow = iRow
  End With
  MsgBox "恢复原来的窗口状态"
  ActiveWindow.SplitColumn = 0
  ActiveWindow.SplitRow = 0
End Sub
```
示例说明：本示例演示了以活动单元格为基准拆分窗格。如果指定窗口被拆分，则 Split 属性的值为 True；设置该属性的值为 False 则取消窗格拆分。也可以设置 SplitColumn 属性和 SplitRow 属性的值来取消窗格拆分。

示例 02-10：冻结窗格(FreezePanes 属性)
```
Sub testFreezePane()
  MsgBox “冻结窗格”
  ActiveWindow.FreezePanes = True
End Sub
```
示例说明：运行本程序后，将会冻结活动单元格所在位置上方和左侧的单元格区域。将该属性的值设置为 False，将取消冻结窗格。

示例 02-11：设置网格线颜色(GridlineColor 属性和 GridlineColorIndex 属性)
```
Sub setGridlineColor()
  Dim iColor As Long
  iColor=ActiveWindow.GridlineColor
  MsgBox "将活动窗口的网格线颜色设为红色"
```

```
  ActiveWindow.GridlineColor = RGB(255, 0, 0)
  MsgBox "将活动窗口的网格线颜色设为蓝色"
  ActiveWindow.GridlineColorIndex = 5
  MsgBox "恢复为原来的网格线颜色"
  ActiveWindow.GridlineColorIndex=iColor
End Sub
```

示例说明：运行程序后，当前工作表窗口网格线将被设置为红色。其中，GridlineColorIndex 属性可以用于返回或设置网格线的颜色，下面给出了默认调色板中颜色的编号值：



[小结]ActiveWindow 属性返回当前激活的工作簿窗口，可以用来设置工作表中的元素，也可以显示特定的单元格，或者用来调整窗口的显示比例，以及设置窗口。

第二章　窗口(Window 对象)基本操作应用示例(二)

分类:ExcelVBA>>ExcelVBA 编程入门范例

示例 02-12：设置工作表标签区域宽度和水平滚动条宽度比例(TabRatio 属性)

```
Sub test()
  MsgBox "设置工作表标签区域宽度为水平滚动条宽度的一半"
  ActiveWindow.TabRatio = 0.5
End Sub
```

示例说明：TabRatio 属性返回或设置工作簿中工作表标签区域的宽度与窗口水平滚动条的宽度比例(可为 0 到 1 之间的数字；默认值为 0.6)。您可以改变上面程序中的数值进行测试。

示例 02-13：设置激活窗口时运行的程序(OnWindow 属性)

```
Sub testRunProcedure()
  ThisWorkbook.Windows(1).OnWindow = "test"
End Sub
'*********************************************************
Sub test()
  MsgBox "您可以使用本窗口了!"
End Sub
```

示例说明：本示例包括两个程序，主程序为 testRunProcedure()，运行后，每当激活该窗口时，将会运行 test()程序。其中，OnWindow 属性返回或设置每当激

活一个窗口时要运行的过程的名称，如本例中的 test()程序。

示例 02-14：获取指定窗口单元格区域地址(RangeSelection 属性)

```
Sub testRangeSelection()
  MsgBox "显示所选单元格地址"
  MsgBox ActiveWindow.RangeSelection.Address
End Sub
```

示例说明：本示例返回当前窗口中所选单元格区域的地址。RangeSelection 属性返回指定窗口的工作表中的选定单元格(即使指定工作表中有图形对象处于活动状态，或者已选定图形对象，仍返回在图形对象被选定之前选定的单元格区域，这是该属性与 Selection 属性的区别)。

示例 02-15：返回指定窗口中所选择的工作表(SelectedSheets 属性)

```
Sub testSelectedSheet()
  Dim sh As Worksheet
  For Each sh In ActiveWorkbook.Windows(1).SelectedSheets
    MsgBox "工作表" & sh.Name & "被选择"
  Next
End Sub
```

示例说明：SelectedSheets 属性返回代表指定窗口中的所有选定工作表的集合。本示例中，如果您同时选择了活动工作簿中的工作表 Sheet1 和 Sheet2，那么运行程序后，将会显示相应工作表被选择的信息。

示例 02-16：排列窗口(Arrange 方法)

```
Sub testArrangeWindows()
  MsgBox "请确保应用程序至少含有两个工作簿,这样才能看出效果"
  MsgBox “窗口将平铺显示”
  Windows.Arrange ArrangeStyle:=xlArrangeStyleTiled
  MsgBox “窗口将层叠显示”
  Windows.Arrange ArrangeStyle:=xlArrangeStyleCascade
  MsgBox “窗口将水平排列显示”
  Windows.Arrange ArrangeStyle:=xlArrangeStyleHorizontal
  MsgBox “窗口将垂直并排排列显示”
  Windows.Arrange ArrangeStyle:=xlArrangeStyleVertical
End Sub
```

示例说明：运行本程序后，将平铺应用程序中的所有窗口。Arrange 方法用于对屏幕上的窗口进行排列，其语法为 expression.Arrange(ArrangeStyle, ActiveWorkbook, SyncHorizontal, SyncVertical)，所有的参数均为可选参数。其中，参数 ArrangeStyle 代表排列样式，可为以下常量：常量 xlArrangeStyleTiled 为缺省值，表示将平铺窗口；常量 xlArrangeStyleCascade 表示将窗口进行层叠；常量 xlArrangeStyleHorizontal 表示将水平排列所有窗口；常量 xlArrangeStyleVertical 表示将垂直并排排列所有窗口。您可以在上面的程序中测试这些常量，以体验效果。将参数 ActiveWorkbook 设置为 True，则只对当前工作簿的可见窗口进行排列。如果为 False，则对所有窗口进行排列。默认值为 False。设置参数 S

yncHorizontal 为 True，在水平滚动时同步滚动当前工作簿的所有窗口；如果为 False，则不同步滚动。设置参数 SyncVertical 为 True，则在垂直滚动时同步滚动当前工作簿的所有窗口；如果为 False，则不同步滚动，默认值为 False。如果参数 ActiveWorkbook 为 False 或者省略，则参数 SyncHorizontal 和 SyncVertical 被忽略。

示例 02-17：窗口尺寸(UsableHeight、UsableWidth、Height、Width 属性)

```
Sub testActiveWindowSize()
 MsgBox "当前窗口可用区域的高度为:" & ActiveWindow.UsableHeight
   MsgBox "当前窗口的高度为:" & ActiveWindow.Height
   MsgBox "当前窗口可用区域的宽度为:" & ActiveWindow.UsableWidth
   MsgBox "当前窗口的宽度为:" & ActiveWindow.Width
End Sub
```

示例 02-18：水平排列两个窗口

```
Sub testWindowArrange()
  Dim ah As Long, aw As Long
  Windows.Arrange xlArrangeStyleTiled
  ah = Windows(1).Height
  aw = Windows(1).Width + Windows(2).Width
  With Windows(1)
    .Width = aw
    .Height = ah / 2
    .Left = 0
  End With
  With Windows(2)
    .Width = aw
    .Height = ah / 2
    .Top = ah / 2
    .Left = 0
  End With
End Sub
```

示例说明：在运行本示例前，保证只打开了两个工作簿窗口。运行本示例后，将水平排列第一个窗口和第二个窗口，即每个窗口占用可使用的垂直空间的一半，占用所有水平空间。其中，Top 属性表示从窗口顶端到可用区域顶端的距离，无法对最大化的窗口设置本属性；Left 属性表示使用区域的左边界至窗口左边界的距离，如果窗口已最大化，则会返回一个负数；如果该属性被设置为 0，则窗口的主边框刚好在屏幕上可见。

示例 02-19：改变窗口的高度和宽度

```
Sub ChangeHeightAndWidth()
  Dim iWinHeight As Long, iWinWidth As Long
  ActiveWindow.WindowState = xlNormal
  MsgBox "将当前窗口的高度和宽度各减一半"
```

```
  iWinHeight = ActiveWindow.Height
  iWinWidth = ActiveWindow.Width
  ActiveWindow.Height = iWinHeight / 2
  ActiveWindow.Width = iWinWidth / 2
  MsgBox "恢复原窗口大小"
  ActiveWindow.Height = iWinHeight
  ActiveWindow.Width = iWinWidth
End Sub
```

示例说明：Height 属性和 Width 属性必须在窗口处于正常显示状态(即不是最大化或最小化状态)时使用，否则会出错。

示例 02-20：移动窗口

```
Sub SetWindowPosition()
  Dim iTop As Long, iLeft As Long
  MsgBox "将当前窗口向下移 60,向右移 90"
  ActiveWindow.WindowState = xlNormal
  iTop = ActiveWindow.Top
  iLeft = ActiveWindow.Left
  ActiveWindow.Top = iTop + 60
  ActiveWindow.Left = iLeft + 90
  MsgBox "恢复原来窗口的位置"
  ActiveWindow.Top = iTop
  ActiveWindow.Left = iLeft
End Sub
```

示例说明：Top 属性和 Left 属性必须在窗口处于正常显示状态(即不是最大化或最小化状态)时使用，否则会出错。

示例 02-21：并排比较窗口

```
Sub testCompare()
  MsgBox "与工作簿 Book2 进行并排比较"
  Windows.CompareSideBySideWith "Book2"
  MsgBox "启动窗口滚动功能,使两个窗口同时滚动"
  Windows.SyncScrollingSideBySide = True
  MsgBox "将工作簿 Book2 最小化"
  Windows("Book2").WindowState = xlMinimized
  MsgBox "重置并排比较显示,恢复并排比较"
  Windows.ResetPositionsSideBySide
  MsgBox "关闭并排比较"
  ActiveWorkbook.Windows.BreakSideBySide
End Sub
```

示例说明：在运行本示例前，确保在本窗口外还打开了一个名为 Book2 的窗口，或者您打开了一个其它命名的窗口，相应将上面程序中的 Book2 更换为您的窗口名。CompareSideBySideWith 方法将以并排模式打开两个窗口，其中一个是当前活动窗口，另一个就是该方法所指定的窗口，如本例中的 Book2。SyncScr

ollingSideBySide 属性设置是否将两个窗口的滚动保持同步，如果为 True，在对文档进行并排比较的同时启用窗口内容的滚动功能。若为 False，则在对文档进行并排比较的同时禁用窗口内容的滚动功能。ResetPositionsSideBySide 方法重置正在进行并排比较的两个工作表窗口的位置，例如，如果用户将正在进行比较的两个工作表窗口中的其中一个窗框最小化或最大化，就可以使用 ResetPositionsSideBySide 方法重置显示，以便这两个窗口再次并排显示。BreakSideBySide 方法用来关闭并排比较。

示例 02-22：返回或设置窗口中显示的视图(View 属性)

```
Sub testView()
  MsgBox "将视图切换为分页预览"
  ActiveWindow.View = xlPageBreakPreview
  MsgBox "窗口视图为:" & ActiveWindow.View
  MsgBox "将视图恢复正常"
  ActiveWindow.View = xlNormalView
  MsgBox "窗口视图为:" & ActiveWindow.View
End Sub
```

示例 02-23：返回窗口中可见单元格区域(VisibleRange 属性)

```
Sub testVisibleRange()
  MsgBox "当前窗口中共有" & Windows(1).VisibleRange.Cells.Count & "个单元格可见"
End Sub
```

示例说明：如果窗口中有部分行列的单元格可见，也包括在可见单元格区域中。

示例 02-24：创建窗口(NewWindow 方法)

```
Sub testNewWindow()
  MsgBox "为活动窗口创建一个副本"
  ActiveWindow.NewWindow
  MsgBox "所创建窗口的窗口号为" & ActiveWindow.WindowNumber
End Sub
```

示例说明：本示例中，NewWindow 方法为指定窗口(本例中为当前活动窗口)创建一个副本，然后显示该副本窗口的窗口号。注意，窗口号与窗口索引(Index 属性)的不同，例如名称为“Book1.xls:2”的窗口，其窗口号为 2，而窗口索引为该窗口在 Windows 集合中的位置，可以为窗口名称或编号。

示例 02-25：设置窗口大小(Zoom 属性)

```
Sub testWindowDisplaySize()
  MsgBox "将窗口大小设置为与选定区域相适应的大小"
  ActiveWindow.Zoom = True
  MsgBox "以双倍大小显示窗口"
  ActiveWindow.Zoom = 200
  MsgBox "以正常大小显示窗口"
  ActiveWindow.Zoom = 100
```

```
End Sub
```
示例说明：Zoom 属性将以百分数的形式(100 表示正常大小，200 表示双倍大小，以此类推)返回或设置窗口的显示大小。如果本属性为 True，则可将窗口大小设置成与当前选定区域相适应的大小。本功能仅对窗口中当前的活动工作表起作用，若要对其他工作表使用本属性，必须先激活该工作表。

示例 02-26：激活窗口(ActivateNext 方法和 ActivatePrevious 方法)

[示例 02-26-01]

```
Sub testActivateWindow1()
  MsgBox "若已打开 Book1.xls、Book2.xls 和 Book3.xls 三个工作簿且 Book1.xls 为当前窗口" & Chr(10) & "则按 Book3.xls-Book2.xls-Book1.xls 依次激活窗口"
    ActiveWindow.ActivateNext
    MsgBox "激活工作簿:" & Windows(1).Caption
    ActiveWindow.ActivateNext
    MsgBox "激活工作簿:" & Windows(1).Caption
    ActiveWindow.ActivateNext
    MsgBox "激活工作簿:" & Windows(1).Caption
End Sub
```

[示例 02-26-02]

```
Sub testActivateWindow2()
   MsgBox "若已打开 Book1.xls、Book2.xls 和 Book3.xls 三个工作簿且 Book1.xls 为当前窗口" & Chr(10) & "则按 Book2.xls-Book3.xls-Book1.xls 依次激活窗口"
    ActiveWindow.ActivatePrevious
    MsgBox "激活工作簿:" & Windows(1).Caption
    ActiveWindow.ActivatePrevious
    MsgBox "激活工作簿:" & Windows(1).Caption
    ActiveWindow.ActivatePrevious
    MsgBox "激活工作簿:" & Windows(1).Caption
End Sub
```

示例 02-27：滚动窗口(LargeScroll 方法和 SmallScroll 方法)

[示例 02-27-01]

```
Sub testScroll1()
  MsgBox "将当前窗口向下滚动 3 页并向右滚动 1 页"
  ActiveWindow.LargeScroll Down:=3, ToRight:=1
End Sub
```

示例说明：LargeScroll 方法将按页滚动窗口的内容，其语法为 expression.LargeScroll(Down, Up, ToRight, ToLeft)，带有 4 个可选的参数，其中参数 Down 表示将窗口内容向下滚动的页数；参数 Up 表示将窗口内容向上滚动的页数；参数 ToRight 表示将窗口内容向右滚动的页数；参数 ToLeft 表示将窗口内容向左滚动的页数。如果同时指定了 Down 和 Up，窗口内容上下方向滚动的页数由这两个参数的差决定，例如，如果 Down 为 3，Up 为 6，则窗口向上滚动三页。

如果同时指定了 ToLeft 和 ToRight，窗口内容左右方向滚动的页数由这两个参数的差决定，例如，如果 ToLeft 为 3，ToRight 为 6，则窗口向右滚动三页。所有这四个参数都可以使用负数。

[示例 02-27-02]

```
Sub testScroll2()
  MsgBox "将当前活动窗口向下滚动 3 行"
  ActiveWindow.SmallScroll down:=3
End Sub
```

示例说明：SmallScroll 方法按行或列滚动窗口，其语法为 expression.SmallScroll(Down, Up, ToRight, ToLeft)，带有 4 个可选的参数，其中参数 Down 表示将窗口内容向下滚动的行数；参数 Up 表示将窗口内容向上滚动的列数；参数 ToRight 表示将窗口内容向右滚动的列数；参数 ToLeft 表示将窗口内容向左滚动的列数。如果同时指定了 Down 和 Up，则窗口内容滚动的行数由这两个参数的差值决定，例如，如果 Down 为 3，Up 为 6，则窗口内容向上滚动三行。如果同时指定了 ToLeft 和 ToRight，则窗口内容滚动的列数由这两个参数的差值决定，例如，如果 ToLeft 为 3，ToRight 为 6，则窗口内容向右滚动三列。以上四个参数均可取负值。

示例 02-28：测试所选单元格宽度和高度

```
Sub testWidthOrHeight()
  Dim lWinWidth As Long, lWinHeight As Long
  With ActiveWindow
    lWinWidth = .PointsToScreenPixelsX(.Selection.Width)
    lWinHeight = .PointsToScreenPixelsY(.Selection.Height)
  End With
  MsgBox "当前选定单元格宽度为:" & lWinWidth & Chr(10) & _
    "当前选定单元格高度为:" & lWinHeight
End Sub
```

示例 02-29：关闭窗口(Close 方法)

```
Sub CloseWindow()
  MsgBox "关闭当前窗口"
  ActiveWindow.Close
End Sub
```

示例说明：本示例运行后，将关闭当前窗口。如果当前窗口未保存，则会弹出询问是否保存的消息框供选择。

By fanjy in 2006-9-23

第三章　工作簿(Workbook)基本操作应用示例(一)

分类:ExcelVBA>>ExcelVBA 编程入门范例

Workbook 对象代表工作簿，而 Workbooks 集合则包含了当前所有的工作簿。下面对 Workbook 对象的重要的方法和属性以及其它一些可能涉及到的方法和属性

进行示例介绍，同时，后面的示例也深入介绍了一些工作簿对象操作的方法和技巧。

示例 03-01：创建工作簿(Add 方法)

[示例 03-01-01]

```
Sub CreateNewWorkbook1()
   MsgBox "将创建一个新工作簿."
   Workbooks.Add
End Sub
```

[示例 03-01-02]

```
Sub CreateNewWorkbook2()
   Dim wb As Workbook
   Dim ws As Worksheet
   Dim i As Long
   MsgBox "将创建一个新工作簿,并预设工作表格式."
   Set wb = Workbooks.Add
   Set ws = wb.Sheets(1)
   ws.Name = "产品汇总表"
   ws.Cells(1, 1) = "序号"
   ws.Cells(1, 2) = "产品名称"
   ws.Cells(1, 3) = "产品数量"
   For i = 2 To 10
       ws.Cells(i, 1) = i - 1
   Next i
End Sub
```

示例 03-02：添加并保存新工作簿

```
Sub AddSaveAsNewWorkbook()
  Dim Wk As Workbook
  Set Wk = Workbooks.Add
  Application.DisplayAlerts = False
  Wk.SaveAs Filename:="D:/SalesData.xls"
End Sub
```

示例说明：本示例使用了 Add 方法和 SaveAs 方法，添加一个新工作簿并将该工作簿以文件名 SalesData.xls 保存在 D 盘中。其中，语句 Application.DisplayAlerts = False 表示禁止弹出警告对话框。

示例 03-03：打开工作簿(Open 方法)

[示例 03-03-01]

```
Sub openWorkbook1()
       Workbooks.Open "<需打开文件的路径>\<文件名>"
End Sub
```

示例说明：代码中的<>里的内容需用所需打开的文件的路径及文件名代替。Open 方法共有 15 个参数，其中参数 FileName 为必需的参数，其余参数可选。

[示例 03-03-02]
```
Sub openWorkbook2()
   Dim fname As String
   MsgBox "将 D 盘中的<测试.xls>工作簿以只读方式打开"
   fname = "D:\测试.xls"
   Workbooks.Open Filename:=fname, ReadOnly:=True
End Sub
```

示例 03-04：将文本文件导入工作簿中(OpenText 方法)
```
Sub TextToWorkbook()
   '本示例打开某文本文件并将制表符作为分隔符对此文件进行分列处理转换成为工作表
   Workbooks.OpenText Filename:="<文本文件所在的路径>/<文本文件名>",
_
            DataType:=xlDelimited, Tab:=True
End Sub
```
示例说明：代码中的<>里的内容需用所载入的文本文件所在路径及文件名代替。OpenText 方法的作用是导入一个文本文件，并将其作为包含单个工作表的工作簿进行分列处理，然后在此工作表中放入经过分列处理的文本文件数据。该方法共有 18 个参数，其中参数 FileName 为必需的参数，其余参数可选。

示例 03-05：保存工作簿(Save 方法)
[示例 03-05-01]
```
Sub SaveWorkbook()
   MsgBox "保存当前工作簿."
   ActiveWorkbook.Save
End Sub
```
[示例 03-05-02]
```
Sub SaveAllWorkbook1()
   Dim wb As Workbook
   MsgBox "保存所有打开的工作簿后退出 Excel."
   For Each wb In Application.Workbooks
       wb.Save
   Next wb
   Application.Quit
End Sub
```
[示例 03-05-03]
```
Sub SaveAllWorkbook2()
   Dim wb As Workbook
   For Each wb In Workbooks
       If wb.Path <> "" Then wb.Save
   Next wb
End Sub
```
示例说明：本示例保存原来已存在且已打开的工作簿。

示例 03-06：保存工作簿(SaveAs 方法)
[示例 03-06-01]
```
Sub SaveWorkbook1()
   MsgBox "将工作簿以指定名保存在默认文件夹中."
   ActiveWorkbook.SaveAs "<工作簿名>.xls"
End Sub
```
示例说明：SaveAs 方法相当于“另存为……”命令，以指定名称保存工作簿。该方法有 12 个参数，均为可选参数。如果未指定保存的路径，那么将在默认文件夹中保存该工作簿。如果文件夹中该工作簿名已存在，则提示是否替换原工作簿。
[示例 03-06-02]
```
Sub SaveWorkbook2()
   Dim oldName As String, newName As String
   Dim folderName As String, fname As String
   oldName = ActiveWorkbook.Name
   newName = "new" & oldName
   MsgBox "将<" & oldName & ">以<" & newName & ">的名称保存"
   folderName = Application.DefaultFilePath
   fname = folderName & "\" & newName
   ActiveWorkbook.SaveAs fname
End Sub
```
示例说明:本示例将当前工作簿以一个新名(即 new 加原名)保存在默认文件夹中。
[示例 03-06-03]
```
Sub CreateBak1()
   MsgBox "保存工作簿并建立备份工作簿"
   ActiveWorkbook.SaveAs CreateBackup:=True
End Sub
```
示例说明：本示例将在当前文件夹中建立工作簿的备份。
[示例 03-06-04]
```
Sub CreateBak2()
   MsgBox "保存工作簿时,若已建立了备份,则将出现包含 True 的信息框,否则出现 False."
   MsgBox ActiveWorkbook.CreateBackup
End Sub
```

示例 03-07：取得当前打开的工作簿数(Count 属性)
```
Sub WorkbookNum()
   MsgBox "当前已打开的工作簿数为:" & Chr(10) & Workbooks.Count
End Sub
```

示例 03-08：激活工作簿(Activate 方法)
[示例 03-08-01]
```
Sub ActivateWorkbook1()
   Workbooks("<工作簿名>").Activate
```

```
End Sub
```
示例说明：Activate 方法激活一个工作簿，使该工作簿为当前工作簿。
[示例 03-08-02]
```
Sub ActivateWorkbook2()
   Dim n As Long, i As Long
   Dim b As String
   MsgBox "依次激活已经打开的工作簿"
   n = Workbooks.Count
   For i = 1 To n
       Workbooks(i).Activate
       b = MsgBox("第 " & i & "个工作簿被激活，还要继续吗?", vbYesNo)
       If b = vbNo Then Exit Sub
       If i = n Then MsgBox "最后一个工作簿已被激活."
   Next i
End Sub
```

示例 03-09：保护工作簿(Protect 方法)
```
Sub ProtectWorkbook()
   MsgBox "保护工作簿结构,密码为 123"
   ActiveWorkbook.Protect Password:="123", Structure:=True
   MsgBox "保护工作簿窗口,密码为 123"
   ActiveWorkbook.Protect Password:="123", Windows:=True
   MsgBox "保护工作簿结构和窗口,密码为 123"
   ActiveWorkbook.Protect Password:="123", Structure:=True, Windows:=
True
End Sub
```
示例说明：使用 Protect 方法来保护工作簿，带有三个可选参数，参数 Password 指明保护工作簿密码，要解除工作簿保护应输入此密码；参数 Structure 设置为 True 则保护工作簿结构，此时不能对工作簿中的工作表进行插入、复制、删除等操作；参数 Windows 设置为 True 则保护工作簿窗口，此时该工作簿右上角的最小化、最大化和关闭按钮消失。

示例 03-10：解除工作簿保护(UnProtect 方法)
```
Sub UnprotectWorkbook()
   MsgBox "取消工作簿保护"
   ActiveWorkbook.Unprotect "123"
End Sub
```

示例 03-11：工作簿的一些通用属性示例
```
Sub testGeneralWorkbookInfo()
   MsgBox "本工作簿的名称为" & ActiveWorkbook.Name
   MsgBox "本工作簿带完整路径的名称为" & ActiveWorkbook.FullName
   MsgBox "本工作簿对象的代码名为" & ActiveWorkbook.CodeName
   MsgBox "本工作簿的路径为" & ActiveWorkbook.Path
```

```
    If ActiveWorkbook.ReadOnly Then
        MsgBox "本工作簿已经是以只读方式打开"
    Else
        MsgBox "本工作簿可读写."
    End If
    If ActiveWorkbook.Saved Then
        MsgBox "本工作簿已保存."
    Else
        MsgBox "本工作簿需要保存."
    End If
End Sub
```

示例 03-12：访问工作簿的内置属性(BuiltinDocumentProperties 属性)

[示例 03-12-01]

```
Sub ShowWorkbookProperties()
    Dim SaveTime As String
    On Error Resume Next
    SaveTime = ActiveWorkbook.BuiltinDocumentProperties("Last Save Tim
e").Value
    If SaveTime = "" Then
        MsgBox ActiveWorkbook.Name & "工作簿未保存."
    Else
        MsgBox "本工作簿已于" & SaveTime & "保存", , ActiveWorkbook.Na
me
    End If
End Sub
```

示例说明：在 Excel 中选择菜单“文件——属性”命令时将会显示一个“属性”对话框，该对话框中包含了当前工作簿的有关信息，可以在 VBA 中使用 Builtin DocumentProperties 属性访问工作簿的属性。上述示例代码将显示当前工作簿保存时的日期和时间。

[示例 03-12-02]

```
Sub listWorkbookProperties()
    On Error Resume Next
    '在名为"工作簿属性"的工作表中添加信息,若该工作表不存在,则新建一个
工作表
    Worksheets("工作簿属性").Activate
    If Err.Number <> 0 Then
        Worksheets.Add after:=Worksheets(Worksheets.Count)
        ActiveSheet.Name = "工作簿属性"
    Else
        ActiveSheet.Clear
    End If
    On Error GoTo 0
    ListProperties
```

```
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - -
Sub ListProperties()
   Dim i As Long
   Cells(1, 1) = "名称"
   Cells(1, 2) = "类型"
   Cells(1, 3) = "值"
   Range("A1:C1").Font.Bold = True
   With ActiveWorkbook
       For i = 1 To .BuiltinDocumentProperties.Count
           With .BuiltinDocumentProperties(i)
               Cells(i + 1, 1) = .Name
               Select Case .Type
                   Case msoPropertyTypeBoolean
                       Cells(i + 1, 2) = "Boolean"
                   Case msoPropertyTypeDate
                       Cells(i + 1, 2) = "Date"
                   Case msoPropertyTypeFloat
                       Cells(i + 1, 2) = "Float"
                   Case msoPropertyTypeNumber
                       Cells(i + 1, 2) = "Number"
                   Case msoPropertyTypeString
                       Cells(i + 1, 2) = "string"
               End Select
               On Error Resume Next
               Cells(i + 1, 3) = .Value
               On Error GoTo 0
           End With
       Next i
   End With
   Range("A:C").Columns.AutoFit
End Sub
```

示例说明：本示例代码在“工作簿属性”工作表中列出了当前工作簿中的所有内置属性。

示例 03-13：测试工作簿中是否包含指定工作表(Sheets 属性)

```
Sub testSheetExists()
   MsgBox "测试工作簿中是否存在指定名称的工作表"
   Dim b As Boolean
   b = SheetExists("<指定的工作表名>")
   If b = True Then
       MsgBox "该工作表存在于工作簿中."
   Else
       MsgBox "工作簿中没有这个工作表."
```

```
    End If
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - - -
Private Function SheetExists(sname) As Boolean
    Dim x As Object
    On Error Resume Next
    Set x = ActiveWorkbook.Sheets(sname)
    If Err = 0 Then
        SheetExists = True
    Else
        SheetExists = False
    End If
End Function
```

示例 03-14：对未打开的工作簿进行重命名(Name 方法)

```
Sub rename()
    Name "<工作簿路径>\<旧名称>.xls" As "<工作簿路径>\<新名称>.xls"
End Sub
```

示例说明：代码中<>中的内容为需要重命名的工作簿所在路径及新旧名称。该方法只是对未打开的文件进行重命名，如果该文件已经打开，使用该方法会提示错误。

示例 03-15：设置数字精度(PrecisionAsDisplayed 属性)

```
Sub SetPrecision()
    Dim pValue
    MsgBox "在当前单元格中输入 1/3，并将结果算至小数点后两位"
    ActiveCell.Value = 1 / 3
    ActiveCell.NumberFormatLocal = "0.00"
    pValue = ActiveCell.Value * 3
    MsgBox "当前单元格中的数字乘以 3 等于：" & pValue
    MsgBox "然后,将数值分类设置为[数值],即单元格中显示的精度"
    ActiveWorkbook.PrecisionAsDisplayed = True
    pValue = ActiveCell.Value * 3
    MsgBox "此时,当前单元格中的数字乘以 3 等于：" & pValue & "而不是 1"
    ActiveWorkbook.PrecisionAsDisplayed = False
End Sub
```

示例说明：PrecisionAsDisplayed 属性的值设置为 True，则表明采用单元格中所显示的数值进行计算。

示例 03-16：删除自定义数字格式(DeleteNumberFormat 方法)

```
Sub DeleteNumberFormat()
    MsgBox "从当前工作簿中删除 000-00-0000 的数字格式"
    ActiveWorkbook.DeleteNumberFormat ("000-00-0000")
End Sub
```

示例说明：DeleteNumberFormat 方法将从指定的工作簿中删除自定义的数字格式。

示例 03-17：控制工作簿中图形显示(DisplatyDrawingObjects 属性)

```
Sub testDraw()
   MsgBox "隐藏当前工作簿中的所有图形"
   ActiveWorkbook.DisplayDrawingObjects = xlHide
   MsgBox "仅显示当前工作簿中所有图形的占位符"
   ActiveWorkbook.DisplayDrawingObjects = xlPlaceholders
   MsgBox "显示当前工作簿中的所有图形"
   ActiveWorkbook.DisplayDrawingObjects = xlDisplayShapes
End Sub
```

示例说明：本属性作用的对象包括图表和形状。在应用本示例前，应保证工作簿中有图表或形状，以察看效果。

示例 03-18：指定名称(Names 属性)

```
Sub testNames()
   MsgBox "将当前工作簿中工作表 Sheet1 内单元格 A1 命名为 myName."
   ActiveWorkbook.Names.Add Name:="myName", RefersToR1C1:="=Sheet1!R1
C1"
End Sub
```

示例说明：对于 Workbook 对象而言，Names 属性返回的集合代表工作簿中的所有名称。

示例 03-19：检查工作簿的自动恢复功能(EnableAutoRecover 属性)

```
Sub UseAutoRecover()
   '检查是否工作簿自动恢复功能开启,如果没有则开启该功能
   If ActiveWorkbook.EnableAutoRecover = False Then
       ActiveWorkbook.EnableAutoRecover = True
       MsgBox "刚开启自动恢复功能."
   Else
       MsgBox "自动恢复功能已开启."
   End If
End Sub
```

示例 03-20：设置工作簿密码(Password 属性)

```
Sub UsePassword()
   Dim wb As Workbook
   Set wb = Application.ActiveWorkbook
   wb.Password = InputBox("请输入密码:")
   wb.Close
End Sub
```

示例说明：Password 属性返回或设置工作簿密码，在打开工作簿时必须输入密码。本示例代码运行后，提示设置密码，然后关闭工作簿；再次打开工作簿时，

要求输入密码。

示例 03-21：返回工作簿用户状态信息(UserStatus 属性)

```
Sub UsePassword()
   Dim Users As Variant
   Dim Row As Long
   Users = ActiveWorkbook.UserStatus
   Row = 1
   With Workbooks.Add.Sheets(1)
       .Cells(Row, 1) = "用户名"
       .Cells(Row, 2) = "日期和时间"
       .Cells(Row, 3) = "使用方式"
       For Row = 1 To UBound(Users, 1)
           .Cells(Row + 1, 1) = Users(Row, 1)
           .Cells(Row + 1, 2) = Users(Row, 2)
           Select Case Users(Row, 3)
               Case 1
                   .Cells(Row + 1, 3).Value = "个人工作簿"
               Case 2
                   .Cells(Row + 1, 3).Value = "共享工作簿"
           End Select
       Next
   End With
   Range("A:C").Columns.AutoFit
End Sub
```

示例说明：示例代码运行后，将创建一个新工作簿并带有用户使用当前工作簿的信息，即用户名、打开的日期和时间及工作簿使用方式。

示例 03-22：检查工作簿是否有密码保护(HasPassword 属性)

```
Sub IsPassword()
   If ActiveWorkbook.HasPassword = True Then
       MsgBox "本工作簿有密码保护,请在管理员处获取密码."
   Else
       MsgBox "本工作簿无密码保护,您可以自由编辑."
   End If
End Sub
```

示例 03-23：决定列表边框是否可见(InactiveListBorderVisible 属性)

```
Sub HideListBorders()
   MsgBox "隐藏当前工作簿中所有非活动列表的边框."
   ActiveWorkbook.InactiveListBorderVisible = False
End Sub
```

示例 03-24：关闭工作簿
[示例 03-24-01]
```
Sub CloseWorkbook1()
Msgbox “不保存所作的改变而关闭本工作簿”
ActiveWorkbook.Close False
 ‘或 ActiveWorkbook.Close SaveChanges:=False
 ‘或 ActiveWorkbook.Saved=True
End sub
```
[示例 03-24-02]
```
Sub CloseWorkbook2()
Msgbox “保存所作的改变并关闭本工作簿”
ActiveWorkbook.Close True
End sub
```
[示例 03-24-03]
```
Sub CloseWorkbook3()
Msgbox “关闭本工作簿。如果工作簿已发生变化，则弹出是否保存更改的对话框。”
ActiveWorkbook.Close True
End sub
```
[示例 03-24-04] 关闭并保存所有工作簿
```
Sub CloseAllWorkbooks()
   Dim Book As Workbook
   For Each Book In Workbooks
If Book.Name<>ThisWorkbook.Name Then
Book.Close savechanges:=True
End If
Next Book
ThisWorkbook.Close savechanges:=True
End Sub
```
[示例 03-24-05] 关闭工作簿并将它彻底删除
```
Sub KillMe()
With ThisWorkbook
.Saved = True
.ChangeFileAccess Mode:=xlReadOnly
Kill .FullName
.Close False
End With
End Sub
```
[示例 03-24-06]关闭所有工作簿，若工作簿已改变则弹出是否保存变化的对话框
```
Sub closeAllWorkbook()
   MsgBox "关闭当前所打开的所有工作簿"
   Workbooks.Close
End Sub
```

第三章　工作簿(Workbook)基本操作应用示例(二)

分类:ExcelVBA>>ExcelVBA 编程入门范例

<其它一些有关操作工作簿的示例>

示例 03-25：创建新的工作簿

```
Sub testNewWorkbook()
MsgBox "创建一个带有 10 个工作表的新工作簿"
Dim wb as Workbook
Set wb = NewWorkbook(10)
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - -
Function NewWorkbook(wsCount As Integer) As Workbook
'创建带有由变量 wsCount 提定数量工作表的工作簿，工作表数在 1 至 255 之间
Dim OriginalWorksheetCount As Long
   Set NewWorkbook = Nothing
   If wsCount < 1 Or wsCount > 255 Then Exit Function
   OriginalWorksheetCount = Application.SheetsInNewWorkbook
   Application.SheetsInNewWorkbook = wsCount
Set NewWorkbook = Workbooks.Add
   Application.SheetsInNewWorkbook = OriginalWorksheetCount
End Function
```

示例说明：自定义函数 NewWorkbook 可以创建最多带有 255 个工作表的工作簿。本测试示例创建一个带有 10 个工作表的新工作簿。

示例 03-26：判断工作簿是否存在

```
Sub testFileExists()
   MsgBox "如果文件不存在则用信息框说明,否则打开该文件."
   If Not FileExists("C:\文件夹\子文件夹\文件.xls") Then
       MsgBox "这个工作簿不存在!"
   Else
       Workbooks.Open "C:\文件夹\子文件夹\文件.xls"
   End If
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - -
Function FileExists(FullFileName As String) As Boolean
   '如果工作簿存在,则返回 True
   FileExists = Len(Dir(FullFileName)) > 0
End Function
```

示例说明：本示例使用自定义函数 FileExists 判断工作簿是否存在，若该工作簿已存在，则打开它。代码中，“C:\文件夹\子文件夹\文件.xls”代表工作簿所在的文件夹名、子文件夹名和工作簿文件名。

示例 03-27：判断工作簿是否已打开

[示例 03-27-01]

```
Sub testWorkbookOpen()
   MsgBox "如果工作簿未打开,则打开该工作簿."
   If Not WorkbookOpen("工作簿名.xls") Then
       Workbooks.Open "工作簿名.xls"
   End If
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - -
Function WorkbookOpen(WorkBookName As String) As Boolean
   '如果该工作簿已打开则返回真
   WorkbookOpen = False
   On Error GoTo WorkBookNotOpen
   If Len(Application.Workbooks(WorkBookName).Name) > 0 Then
       WorkbookOpen = True
       MsgBox "该工作簿已打开"
       Exit Function
   End If
WorkBookNotOpen:
End Function
```

示例说明：本示例中的函数 WorkbookOpen 用来判断工作簿是否打开。代码中，“工作簿名.xls”代表所要打开的工作簿名称。

[示例 03-27-02]

```
Sub testWookbookIFOpen()
   Dim wb As String
   Dim bwb As Boolean
   wb = "<要判断的工作簿名称>"
   bwb = WorkbookIsOpen(wb)
   If bwb = True Then
       MsgBox "工作簿" & wb & "已打开."
   Else
       MsgBox "工作簿" & wb & "未打开."
   End If
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - -
Private Function WorkbookIsOpen(wbname) As Boolean
   Dim x As Workbook
   On Error Resume Next
   Set x = Workbooks(wbname)
   If Err = 0 Then
       WorkbookIsOpen = True
   Else
       WorkbookIsOpen = False
   End If
End Function
```

示例 03-28：备份工作簿

[示例 03-28-01] 用与活动工作簿相同的名字但后缀名为.bak 备份工作簿

```
Sub SaveWorkbookBackup()
   Dim awb As Workbook, BackupFileName As String, i As Integer, OK As
 Boolean
   If TypeName(ActiveWorkbook) = "Nothing" Then Exit Sub
   Set awb = ActiveWorkbook
   If awb.Path = "" Then
       Application.Dialogs(xlDialogSaveAs).Show
   Else
       BackupFileName = awb.FullName
       i = 0
       While InStr(i + 1, BackupFileName, ".") > 0
           i = InStr(i + 1, BackupFileName, ".")
       Wend
       If i > 0 Then BackupFileName = Left(BackupFileName, i - 1)
       BackupFileName = BackupFileName & ".bak"
       OK = False
       On Error GoTo NotAbleToSave
       With awb
           Application.StatusBar = "正在保存工作簿..."
           .Save
           Application.StatusBar = "正在备份工作簿..."
           .SaveCopyAs BackupFileName
           OK = True
       End With
   End If
NotAbleToSave:
   Set awb = Nothing
   Application.StatusBar = False
   If Not OK Then
       MsgBox "备份工作簿未保存!", vbExclamation, ThisWorkbook.Name
   End If
End Sub
```

示例说明：在当前工作簿中运行本示例代码后，将以与工作簿相同的名称但后缀名为.bak 备份工作簿，且该备份与当前工作簿在同一文件夹中。其中，使用了工作簿的 FullName 属性和 SaveCopyAs 方法。

[示例 03-28-02] 保存当前工作簿的副本到其它位置备份工作簿

```
Sub SaveWorkbookBackupToFloppyD()
   Dim awb As Workbook, BackupFileName As String, i As Integer, OK As
 Boolean
   If TypeName(ActiveWorkbook) = "Nothing" Then Exit Sub
   Set awb = ActiveWorkbook
   If awb.Path = "" Then
```

```
        Application.Dialogs(xlDialogSaveAs).Show
    Else
        BackupFileName = awb.Name
        OK = False
        On Error GoTo NotAbleToSave
        If Dir("D:\" & BackupFileName) <> "" Then
            Kill "D:\" & BackupFileName
        End If
        With awb
            Application.StatusBar = "正在保存工作簿..."
            .Save
            Application.StatusBar = "正在备份工作簿..."
            .SaveCopyAs "D:\" & BackupFileName
            OK = True
        End With
    End If
NotAbleToSave:
    Set awb = Nothing
    Application.StatusBar = False
    If Not OK Then
        MsgBox "备份工作簿未保存!", vbExclamation, ThisWorkbook.Name
    End If
End Sub
```

示例说明：本程序将把当前工作簿进行复制并以与当前工作簿相同的名称保存在 D 盘中。其中，使用了 Kill 方法来删除已存在的工作簿。

示例 03-29：从已关闭的工作簿中取值

[示例 03-29-01]

```
Sub testGetValuesFromClosedWorkbook()
    GetValuesFromAClosedWorkbook "C:", "Book1.xls", "Sheet1", "A1:G20"
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - -
Sub GetValuesFromAClosedWorkbook(fPath As String, _
                        fName As String, sName, cellRange As String)
    With ActiveSheet.Range(cellRange)
        .FormulaArray = "='" & fPath & "\[" & fName & "]" _
                                    & sName & "'!" & cellRange
        .Value = .Value
    End With
End Sub
```

示例说明：本示例包含一个子过程 GetValuesFromAClosedWorkbook，用来从已关闭的工作簿中获取数据，主过程 testGetValuesFromClosedWorkbook 用来传递参数。本示例表示从 C 盘根目录下的 Book1.xls 工作簿的工作表 Sheet1 中的 A1:G20 单元格区域内获取数据，并将其复制到当前工作表相应单元格区域中。

[示例 03-29-02]

```
Sub ReadDataFromAllWorkbooksInFolder()
   Dim FolderName As String, wbName As String, r As Long, cValue As Variant
   Dim wbList() As String, wbCount As Integer, i As Integer
   FolderName = "C:\文件夹名"
   '创建文件夹中工作簿列表
   wbCount = 0
   wbName = Dir(FolderName & "\" & "*.xls")
   While wbName <> ""
       wbCount = wbCount + 1
       ReDim Preserve wbList(1 To wbCount)
       wbList(wbCount) = wbName
       wbName = Dir
   Wend
   If wbCount = 0 Then Exit Sub
   '从每个工作簿中获取数据
   r = 0
   Workbooks.Add
   For i = 1 To wbCount
       r = r + 1
       cValue = GetInfoFromClosedFile(FolderName, wbList(i), "Sheet1",
"A1")
       Cells(r, 1).Formula = wbList(i)
       Cells(r, 2).Formula = cValue
   Next i
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - - -
Private Function GetInfoFromClosedFile(ByVal wbPath As String, _
       wbName As String, wsName As String, cellRef As String) As Variant
   Dim arg As String
   GetInfoFromClosedFile = ""
   If Right(wbPath, 1) <> "\" Then wbPath = wbPath & "\"
   If Dir(wbPath & "\" & wbName) = "" Then Exit Function
   arg = "'" & wbPath & "[" & wbName & "]" & _
               wsName & "'!" & Range(cellRef).Address(True, True, xlR1C1)
   On Error Resume Next
   GetInfoFromClosedFile = ExecuteExcel4Macro(arg)
End Function
```

示例说明：本示例将读取一个文件夹内所有工作簿中工作表 Sheet1 上单元格 A1 中的值到一个新工作簿中。代码中，“C:\文件夹名”代表工作簿所在的文件夹名。

[示例 03-29-03]

```
Sub GetDataFromClosedWorkbook()
   Dim wb As Workbook
```

```
    Application.ScreenUpdating = False
    '以只读方式打开工作簿
    Set wb = Workbooks.Open("C:\文件夹名\文件.xls", True, True)
    With ThisWorkbook.Worksheets("工作表名")
    '从工作簿中读取数据
        .Range("A10").Formula  =  wb.Worksheets(" 源 工 作 表 名
").Range("A10").Formula
        .Range("A11").Formula  =  wb.Worksheets(" 源 工 作 表 名
").Range("A20").Formula
        .Range("A12").Formula  =  wb.Worksheets(" 源 工 作 表 名
").Range("A30").Formula
        .Range("A13").Formula  =  wb.Worksheets(" 源 工 作 表 名
").Range("A40").Formula
    End With
    wb.Close False '关闭打开的源数据工作簿且不保存任何变化
    Set wb = Nothing '释放内存
    Application.ScreenUpdating = True
End Sub
```

示例说明：在运行程序时，打开所要获取数据的工作簿，当取得数据后再关闭该工作簿。将屏幕更新属性值设置为 False，将看不出源数据工作簿是否被打开过。本程序代码中，“C:\文件夹名\文件.xls”、"源工作表名"代表工作簿所在的文件夹和工作簿文件名。

第四章 工作表(Worksheet)基本操作应用示例

分类:ExcelVBA>>ExcelVBA 编程入门范例

在编写代码时，经常要引用工作表的名字、知道工作表在工作簿中的位置、增加工作表、删除工作表、复制工作表、移动工作表、重命名工作表，等等。下面介绍与此有关及相关的一些属性和方法示例。

[示例 04-01]增加工作表(Add 方法)

```
Sub AddWorksheet()
    MsgBox "在当前工作簿中添加一个工作表"
    Worksheets.Add
    MsgBox "在当前工作簿中的工作表 sheet2 之前添加一个工作表"
    Worksheets.Add before:=Worksheets("sheet2")
    MsgBox "在当前工作簿中的工作表 sheet2 之后添加一个工作表"
    Worksheets.Add after:=Worksheets("sheet2")
    MsgBox "在当前工作簿中添加 3 个工作表"
    Worksheets.Add Count:=3
End Sub
```

示例说明：Add 方法带有 4 个可选的参数，其中参数 Before 和参数 After 指定

所增加的工作表的位置，但两个参数只能选一；参数 Count 用来指定增加的工作表数目。

[示例 04-02]复制工作表(Copy 方法)

```
Sub CopyWorksheet()
    MsgBox "在当前工作簿中复制工作表 sheet1 并将所复制的工作表放在工作表 sheet2 之前"
    Worksheets("sheet1").Copy Before:=Worksheets("sheet2")
    MsgBox "在当前工作簿中复制工作表 sheet2 并将所复制的工作表放在工作表 sheet3 之后"
    Worksheets("sheet2").Copy After:=Worksheets("sheet3")
End Sub
```

示例说明：Copy 方法带有 2 个可选的参数，即参数 Before 和参数 After，在使用时两个参数只参选一。

[示例 04-03]移动工作表(Move 方法)

```
Sub MoveWorksheet()
    MsgBox "在当前工作簿中将工作表 sheet3 移至工作表 sheet2 之前"
    Worksheets("sheet3").Move Before:=Worksheets("sheet2")
    MsgBox "在当前工作簿中将工作表 sheet1 移至最后"
    Worksheets("sheet1").Move After:=Worksheets(Worksheets.Count)
End Sub
```

示例说明：Move 方法与 Copy 方法的参数相同，作用也一样。

[示例 04-04]隐藏和显示工作表(Visible 属性)

[示例 04-04-01]

```
Sub testHide()
    MsgBox "第一次隐藏工作表 sheet1"
    Worksheets("sheet1").Visible = False
    MsgBox "显示工作表 sheet1"
    Worksheets("sheet1").Visible = True
    MsgBox "第二次隐藏工作表 sheet1"
    Worksheets("sheet1").Visible = xlSheetHidden
    MsgBox "显示工作表 sheet1"
    Worksheets("sheet1").Visible = True
    MsgBox "第三次隐藏工作表 sheet1"
    Worksheets("sheet1").Visible = xlSheetHidden
    MsgBox "显示工作表 sheet1"
    Worksheets("sheet1").Visible = xlSheetVisible
    MsgBox "第四隐藏工作表 sheet1"
    Worksheets("sheet1").Visible = xlSheetVeryHidden
    MsgBox "显示工作表 sheet1"
    Worksheets("sheet1").Visible = True
    MsgBox "第五隐藏工作表 sheet1"
```

```
    Worksheets("sheet1").Visible = xlSheetVeryHidden
    MsgBox "显示工作表 sheet1"
    Worksheets("sheet1").Visible = xlSheetVisible
End Sub
```

示例说明：本示例演示了隐藏和显示工作表的各种情形。其中，使用 xlSheetVeryHidden 常量来隐藏工作表，将不能通过选择工作表菜单栏中的“格式”——“工作表”——“取消隐藏”命令来取消隐藏。

```
[示例 04-04-02]
Sub ShowAllSheets()
    MsgBox "使当前工作簿中的所有工作表都显示(即将隐藏的工作表也显示)"
    Dim ws As Worksheet
    For Each ws In Sheets
        ws.Visible = True
    Next ws
End Sub
```

[示例 04-05]获取工作表数(Count 属性)

```
[示例 04-05-01]
Sub WorksheetNum()
    Dim i As Long
    i = Worksheets.Count
    MsgBox "当前工作簿的工作表数为：" & Chr(10) & i
End Sub
```

```
[示例 04-05-02]
Sub WorksheetNum()
    Dim i As Long
    i = Sheets.Count
    MsgBox "当前工作簿的工作表数为：" & Chr(10) & i
End Sub
```

示例说明：在一个包含图表工作表的工作簿中运行上述两段代码，将会得出不同的结果，原因是对于 Sheets 集合来讲，工作表包含图表工作表。应注意 Worksheets 集合与 Sheets 集合的区别，下同。

[示例 04-06]获取或设置工作表名称(Name 属性)

```
[示例 04-06-01]
Sub NameWorksheet()
    Dim sName As String, sChangeName As String
    sName = Worksheets(2).Name
    MsgBox "当前工作簿中第 2 个工作表的名字为：" & sName
    sChangeName = "我的工作表"
    MsgBox "将当前工作簿中的第 3 个工作表名改为：" & sChangeName
    Worksheets(3).Name = sChangeName
```

```
End Sub
```
示例说明：使用 Name 属性可以获取指定工作表的名称，也可以设置工作表的名称。

[示例 04-06-02]重命名工作表
```
Sub ReNameSheet()
     Dim xStr As String
Retry:
     Err.Clear
     xStr = InputBox("请输入工作表的新名称:" _
             , "重命名工作表", ActiveSheet.Name)
     If xStr = "" Then Exit Sub
     On Error Resume Next
     ActiveSheet.Name = xStr
     If Err.Number <> 0 Then
         MsgBox Err.Number & " " & Err.Description
         Err.Clear
         GoTo Retry
       End If
       On Error GoTo 0
        '.........
  End Sub
```

[示例 04-07]激活/选择工作表(Activate 方法和 Select 方法)
[示例 04-07-01]
```
Sub SelectWorksheet()
   MsgBox "激活当前工作簿中的工作表 sheet2"
   Worksheets("sheet2").Activate
   MsgBox "激活当前工作簿中的工作表 sheet3"
   Worksheets("sheet3").Select
   MsgBox "同时选择工作簿中的工作表 sheet2 和 sheet3"
   Worksheets(Array("sheet2", "sheet3")).Select
End Sub
```
示例说明：Activate 方法只能激活一个工作表，而 Select 方法可以同时选择多个工作表。

[示例 04-07-02]
```
Sub SelectManySheet()
   MsgBox "选取第一个和第三个工作表."
   Worksheets(1).Select
   Worksheets(3).Select False
End Sub
```

[示例 04-08]获取当前工作表的索引号(Index 属性)

```
Sub GetSheetIndex()
   Dim i As Long
   i = ActiveSheet.Index
   MsgBox "您正使用的工作表索引号为" & i
End Sub
```

[示例 04-09]选取前一个工作表(Previous 属性)

```
Sub PreviousSheet()
   If ActiveSheet.Index <> 1 Then
       MsgBox "选取当前工作簿中当前工作表的前一个工作表"
       ActiveSheet.Previous.Activate
   Else
       MsgBox "已到第一个工作表"
   End If
End Sub
```

示例说明：如果当前工作表是第一个工作表，则使用 Previous 属性会出错。

[示例 04-10]选取下一个工作表(Next 属性)

```
Sub NextSheet()
   If ActiveSheet.Index <> Worksheets.Count Then
       MsgBox "选取当前工作簿中当前工作表的下一个工作表"
       ActiveSheet.Next.Activate
   Else
       MsgBox “已到最后一个工作表”
   End If
End Sub
```

示例说明：如果当前工作表是最后一个工作表，则使用 Next 属性会出错。

[示例 04-11]工作表行和列的操作

[示例 04-11-01]隐藏行

```
Sub HideRow()
   Dim iRow As Long
   MsgBox "隐藏当前单元格所在的行"
   iRow = ActiveCell.Row
   ActiveSheet.Rows(iRow).Hidden = True
   MsgBox "取消隐藏"
   ActiveSheet.Rows(iRow).Hidden = False
End Sub
```

[示例 04-11-02]隐藏列

```
Sub HideColumn()
   Dim iColumn As Long
   MsgBox "隐藏当前单元格所在列"
   iColumn = ActiveCell.Column
```

```
    ActiveSheet.Columns(iColumn).Hidden = True
    MsgBox "取消隐藏"
    ActiveSheet.Columns(iColumn).Hidden = False
End Sub
```

[示例 04-11-03]插入行

```
Sub InsertRow()
    Dim rRow As Long
    MsgBox "在当前单元格上方插入一行"
    rRow = Selection.Row
    ActiveSheet.Rows(rRow).Insert
End Sub
```

[示例 04-11-04]插入列

```
Sub InsertColumn()
    Dim cColumn As Long
    MsgBox "在当前单元格所在行的左边插入一行"
    cColumn = Selection.Column
    ActiveSheet.Columns(cColumn).Insert
End Sub
```

[示例 04-11-05]插入多行

```
Sub InsertManyRow()
    MsgBox "在当前单元格所在行上方插入三行"
    Dim rRow As Long, i As Long
    For i = 1 To 3
        rRow = Selection.Row
        ActiveSheet.Rows(rRow).Insert
    Next i
End Sub
```

[示例 04-11-06]设置行高

```
Sub SetRowHeight()
    MsgBox "将当前单元格所在的行高设置为 25"
    Dim rRow As Long, iRow As Long
    rRow = ActiveCell.Row
    iRow = ActiveSheet.Rows(rRow).RowHeight
    ActiveSheet.Rows(rRow).RowHeight = 25
    MsgBox "恢复到原来的行高"
    ActiveSheet.Rows(rRow).RowHeight = iRow
End Sub
```

[示例 04-11-07]设置列宽

```
Sub SetColumnWidth()
    MsgBox "将当前单元格所在列的列宽设置为 20"
```

```
    Dim cColumn As Long, iColumn As Long
    cColumn = ActiveCell.Column
    iColumn = ActiveSheet.Columns(cColumn).ColumnWidth
    ActiveSheet.Columns(cColumn).ColumnWidth = 20
    MsgBox "恢复至原来的列宽"
    ActiveSheet.Columns(cColumn).ColumnWidth = iColumn
End Sub
```

[示例 04-11-08]恢复行高列宽至标准值

```
Sub ReSetRowHeightAndColumnWidth()
    MsgBox "将当前单元格所在的行高和列宽恢复为标准值"
    Selection.UseStandardHeight = True
    Selection.UseStandardWidth = True
End Sub
```

[示例 04-12]工作表标签

[示例 04-12-01] 设置工作表标签的颜色

```
Sub SetSheetTabColor()
    MsgBox "设置当前工作表标签的颜色"
    ActiveSheet.Tab.ColorIndex = 7
End Sub
```

[示例 04-12-01]恢复工作表标签颜色

```
Sub SetSheetTabColorDefault()
    MsgBox "将当前工作表标签颜色设置为默认值"
    ActiveSheet.Tab.ColorIndex = -4142
End Sub
```

[示例 04-12-03]交替隐藏或显示工作表标签

```
Sub HideOrShowSheetTab()
    MsgBox "隐藏/显示工作表标签"
    ActiveWindow.DisplayWorkbookTabs = Not ActiveWindow.DisplayWorkboo
kTabs
End Sub
```

[示例 04-13]确定打印的页数(HPageBreaks 属性与 VPageBreaks 属性)

```
Sub PageCount()
    Dim i As Long
    i = (ActiveSheet.HPageBreaks.Count + 1) * (ActiveSheet.VPageBreaks.
Count + 1)
    MsgBox "当前工作表共" & i & "页."
End Sub
```

[示例 04-14]保护/撤销保护工作表

[示例 04-14-01]
```
Sub ProtectSheet()
   MsgBox "保护当前工作表并设定密码"
   ActiveSheet.Protect Password:="fanjy"
End Sub
```
示例说明：运行代码后，当前工作表中将不允许编辑，除非撤销工作表保护。

[示例 04-14-02]
```
Sub UnprotectSheet()
   MsgBox "撤销当前工作表保护"
   ActiveSheet.Unprotect
End Sub
```
示例说明：运行代码后，如果原保护的工作表设置有密码，则要求输入密码。

[示例 04-14-03]保护当前工作簿中的所有工作表
```
Sub ProtectAllWorkSheets()
   On Error Resume Next
   Dim ws As Worksheet
   Dim myPassword As String
   myPassword = InputBox("请输入您的密码" & vbCrLf & _
     "(不输入表明无密码)" & vbCrLf & vbCrLf & _
     "确保您没有忘记密码！", "输入密码")
   For Each ws In ThisWorkbook.Worksheets
       ws.Protect (myPassword)
   Next ws
End Sub
```

[示例 04-14-04]撤销对当前工作簿中所有工作表的保护
```
Sub UnprotectAllWorkSheets()
   On Error Resume Next
   Dim ws As Worksheet
   Dim myPassword As String
   myPassword = InputBox("请输入您的密码" & vbCrLf & _
       "(不输入表示无密码)", "输入密码")
   For Each ws In ThisWorkbook.Worksheets
       ws.Unprotect (myPassword)
   Next ws
End Sub
```

[示例 04-14-05]仅能编辑未锁定的单元格
```
Sub OnlyEditUnlockedCells()
   Sheets("Sheet1").EnableSelection = xlUnlockedCells
   ActiveSheet.Protect DrawingObjects:=True, Contents:=True, Scenario
s:=True
```

```
End Sub
```
示例说明：运行本代码后，在当前工作表中将只能对未锁定的单元格进行编辑，而其它单元格将不能编辑。未锁定的单元格是指在选择菜单“格式——单元格”命令后所弹出的对话框中的“保护”选项卡中，未选中“锁定”复选框的单元格或单元格区域。

[示例 04-15]删除工作表(Delete 方法)

```
Sub DeleteWorksheet()
   MsgBox "删除当前工作簿中的工作表 sheet2"
   Application.DisplayAlerts = False
   Worksheets("sheet2").Delete
   Application.DisplayAlerts = True
End Sub
```
示例说明:本示例代码使用 Application.DisplayAlerts = False 来屏蔽弹出的警告框。

<一些编程方法和技巧>

[示例 04-16] 判断一个工作表(名)是否存在

[示例 04-16-01]

```
Sub testWorksheetExists1()
   Dim ws As Worksheet
   If Not WorksheetExists(ThisWorkbook, "sheet1") Then
       MsgBox "不能够找到该工作表", vbOKOnly
       Exit Sub
   End If
   MsgBox "已经找到工作表"
   Set ws = ThisWorkbook.Worksheets("sheet1")
End Sub
'- - - - - - - - - - - - - - - - - - -
Function WorksheetExists(wb As Workbook, sName As String) As Boolean
   Dim s As String
   On Error GoTo ErrHandle
   s = wb.Worksheets(sName).Name
   WorksheetExists = True
   Exit Function
ErrHandle:
   WorksheetExists = False
End Function
```
示例说明：在测试代码中，用相应的工作簿名和工作表名分别代替“ThisWorkbook”和“Sheet1”，来判断指定工作表是否在工作簿中存在。

[示例 04-16-02]

```
Sub testWorksheetExists2()
   If Not SheetExists("<工作表名>") Then
```

```
        MsgBox "<工作表名> 不存在!"
    Else
        Sheets("<工作表名>").Activate
    End If
End Sub
'- - - - - - - - - - - - - - - - - - -
Function SheetExists(SheetName As String) As Boolean
    SheetExists = False
    On Error GoTo NoSuchSheet
    If Len(Sheets(SheetName).Name) > 0 Then
        SheetExists = True
        Exit Function
    End If
NoSuchSheet:
End Function
```

示例说明：在代码中，用实际工作表名代替<>。

```
[示例 04-16-03]
Sub TestingFunction()
  '如果工作表存在则返回 True，否则为 False
    '测试 DoesWksExist1 函数
    Debug.Print DoesWksExist1("Sheet1")
    Debug.Print DoesWksExist1("Sheet100")
    Debug.Print "-----"
    '测试 DoesWksExist2 函数
    Debug.Print DoesWksExist2("Sheet1")
    Debug.Print DoesWksExist2("Sheet100")
End Sub
 '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Function DoesWksExist1(sWksName As String) As Boolean
    Dim i As Long
    For i = Worksheets.Count To 1 Step -1
        If Sheets(i).Name = sWksName Then
            Exit For
        End If
    Next
    If i = 0 Then
        DoesWksExist1 = False
    Else
        DoesWksExist1 = True
    End If
End Function
 '- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Function DoesWksExist2(sWksName As String) As Boolean
```

```
    Dim wkb As Worksheet
    On Error Resume Next
    Set wkb = Sheets(sWksName)
    On Error GoTo 0
    DoesWksExist2 = IIf(Not wkb Is Nothing, True, False)
End Function
```

[示例 04-17]排序工作表
[示例 04-17-01]

```
Sub SortWorksheets1()
    Dim bSorted As Boolean
    Dim nSortedSheets As Long
    Dim nSheets As Long
    Dim n As Long
    nSheets = Worksheets.Count
    nSortedSheets = 0
    Do While (nSortedSheets < nSheets) And Not bSorted
        bSorted = True
        nSortedSheets = nSortedSheets + 1
        For n = 1 To nSheets - nSortedSheets
            If StrComp(Worksheets(n).Name, Worksheets(n + 1).Name, vbT
extCompare) > 0 Then
                Worksheets(n + 1).Move Before:=Worksheets(n)
                bSorted = False
            End If
        Next n
      Loop
End Sub
```

示例说明：本示例代码采用了冒泡法排序。

[示例 04-17-02]

```
Sub SortWorksheets2()
    '根据字母对工作表排序
    Dim i As Long, j As Long
    For i = 1 To Sheets.Count
        For j = 1 To Sheets.Count - 1
            If UCase$(Sheets(j).Name) > UCase$(Sheets(j + 1).Name) The
n
                Sheets(j).Move After:=Sheets(j + 1)
            End If
        Next j
    Next i
End Sub
```

[示例 04-17-03]

```
Sub SortWorksheets3()
  '以升序排列工作表
    Dim sCount As Integer, i As Integer, j As Integer
    Application.ScreenUpdating = False
    sCount = Worksheets.Count
    If sCount = 1 Then Exit Sub
    For i = 1 To sCount - 1
        For j = i + 1 To sCount
            If Worksheets(j).Name < Worksheets(i).Name Then
                Worksheets(j).Move Before:=Worksheets(i)
            End If
        Next j
    Next i
End Sub
```

示例说明：若想排序所有工作表，将代码中的 Worksheets 替换为 Sheets。

[示例 04-18]删除当前工作簿中的空工作表

```
Sub Delete_EmptySheets()
    Dim sh As Worksheet
    For Each sh In ThisWorkbook.Worksheets
      If Application.WorksheetFunction.CountA(sh.Cells) = 0 Then
        Application.DisplayAlerts = False
        sh.Delete
        Application.DisplayAlerts = True
      End If
    Next
End Sub
```

Workbook 对象及其方法、属性解析与示例(1)

分类:eBook>>《使用 VBA 编写 Excel 宏》

在这篇文章中，我们来讨论 Workbook 对象和 Workbooks 集合。下图 1 显示了一部分直接与 Workbook 对象相关的 Excel 对象模型。

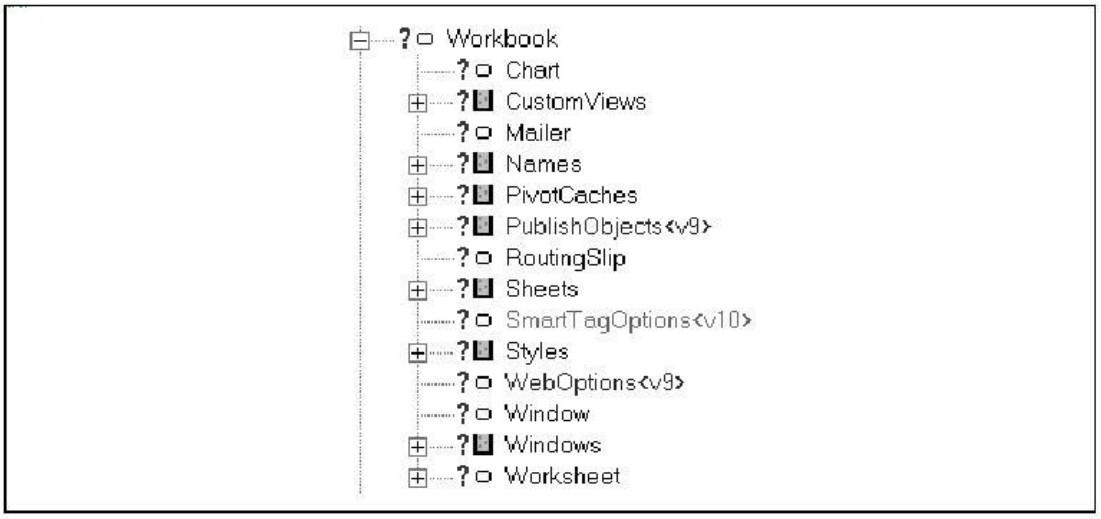


图 1　Workbook 对象

Workbooks 集合

Application 对象有一个 Workbooks 属性，可以返回一个 Workbooks 集合，该集合包含所有当前已打开的 Excel 中的 Workbook 对象。例如，下面的代码将显示所打开的工作簿的数量：

```
'************************************************************************
Sub testWorkbookCount()
  Dim wbs As Workbooks
  Set wbs=Application.Workbooks
  MsgBox wbs.Count
End Sub
'************************************************************************
```

下面让我们来看看 Workbooks 对象的一些属性和方法。

Add 方法

Add 方法创建一个新的工作簿，并添加到 Workbooks 集合中，新工作簿成为活动工作簿。其语法为：

工作簿对象.Add(Template)

在这里，可选参数 ***Template*** 决定如何创建新工作簿。如果该参数为一个指定已存在的 Excel 模板文件名称的字符串，那么新工作簿将以该文件作为模板创建。

正如您所知道的，一个模板是一个包含目录(例如行和列标签)、格式、宏和其它定制的内容(例如菜单和工具条)的 Excel 工作簿。当您以一个模板为基础创建一个新工作簿时，该工作簿将从模板中接受目录、格式和定制的内容。

该 ***Template*** 参数也可以是下面的常量之一：

```
Enum xlWBATemplate
    xlWBATWorksheet=-4167
    xlWBATChart=-4109
    xlWBATExcel4MacroSheet=3
    xlWBATExcel4IntlMacroSheet=4
End Enum
```

在这种情况下，新工作簿将包含有指定类型的单个工作表。如果省略 ***Template*** 参数，那么

Excel 将创建一个新工作簿，该工作簿带有由 Application 对象的 SheetsInNewWorkbook 属性所设置数量的空白工作表。

Close 方法

Close 方法关闭所有打开的工作簿。其语法为：

工作簿对象.Close

Count 属性

大多数集合对象都有一个 Count 属性，Workbooks 集合也不例外。该属性将返回当前已打开的工作簿的数量。

Item 属性

Item 属性返回 Workbooks 集合中特定的工作簿。例如：

Workbooks.Item(1)

返回 Workbooks 集合中的第一个工作簿。由于 Item 属性是缺省属性，因此我们也能简写为：

Workbooks(1)

注意，我们不能依赖特定的工作簿在集合中的索引号来指定工作簿(对所有的集合对象也如此)，您最好使用工作簿的名字指明特定的工作簿，如下所示：

Workbooks(“Book1.xls”)

注意，如果用户使用“文件”菜单中“新建”命令创建了一个名为 Book2 的新工作簿，我们应使用下面的代码指定该工作簿：

Workbooks(“Book2”)

但是，如果您在保存刚新建的工作簿 Book2 到您的硬盘中之前运行下面的代码：

Workbooks(“Book2.xls”)

将会产生一个错误(下标越界)。

Open 方法

该方法打开一个已存在的工作簿，其语法稍微有点复杂：

工作簿对象.Open(FileNaem,UpdateLinks,ReadOnly,Format,Password, _
WriteResPassWord,IgnoreReadOnlyRecommended,Origin,Delimiter, _
Editable,Notify,Converter,AddToMRU)

在这些参数中，大多数参数很少用到(例如，几个与打开文本文件有关的参数)。下面我们讨论一些常用的参数，并且在 VBA 帮助系统中有更多关于这些参数的信息。注意，所有的参数中，除了 *FileName* 外都是可选的。

参数 *FileName* 是所要打开的工作簿的文件名称。如果想要以只读的方式打开工作簿，则可将 *ReadOnly* 参数设置为 True。

如果需要用密码来打开工作簿，则 *PassWord* 参数应该设置为该密码。如果需要使用密码而您没有指定密码，Excel 将弹出对话框询问密码。

参数 *AddToMru* 指定将工作簿添加到最近使用的文件列表中，建议您将该参数值设置为 True 以添加该工作簿到最近使用的文件列表中，该参数的缺省值为 False。

OpenText 方法

这个方法将在一个新工作簿中装入文本文件。该方法分析文本数据并将其放入一个单独的工作表中。其语法稍微有点复杂：

工作簿对象.OpenText(Filename,Origin,StartRow,DataType,TextQualifier, _
ConsecutiveDelimiter,Tab,Semicolon,Comma,Space,Other,OtherChar,FieldInfo)

首先要注意的是，除了参数 *FileName* 外，该方法的所有参数都是可选的。

参数 *Filename* 指定要打开的文本文件的文件名。

参数 *Origin* 指定文本文件的来源，可以是下面的 XlPlatform 常量之一：

Enum XlPlatform
xlMacintosh=1
xlWindows=2
xlMSDOS=3
End Enum
注意，xlWindows 的值指定一个 ANSI 文本文件，而 xlMSDOS 常量指定一个 ASCⅡ文件。如果省略该参数，则需要使用文本导入向导中的文件源选项的当前设置。
参数 *StartRow* 指定从文本文件中开始进行分析的文本的行号，缺省值是 1。
可选参数*DataType*指定在字段中的文本格式，可以是下面的XlTextParsingType常量之一：
Enum XlTextParsingType
xlDelimited=1 ‘缺省值
xlFixedWidth=2
End Enum
参数 *TextQualifier* 是文本限定符，要以是下面的 XlTextQualifier 常量之一：
Enum XlTextQualifier
xlTextQualifierNone=-4142
xlTextQualifierDoubleQuote=1 ‘缺省值
xlTextQualifierSingleQuote=2
End Enum
参数 *ConsecutiveDelimiter* 应该设置为 True，以考虑用连续分隔符作为一个分隔符。缺省的值为 False。
有几个参数需要将参数 *DataType* 设置为 xlDelimited。当这些参数中的任何一个设置为 True 时，表示 Excel 应该使用与文本分隔符相应的字符，这些分隔符描述如下：(所有参数缺省值均为 False)
Tab 该参数设置为真时，将使用制表符作为分隔符
Semicolon 该参数设置为真时，将使用分号作为分隔符
Comma 该参数设置为真时，将使用逗号作为分隔符
Space 该参数设置为真时，将使用空格作为分隔符
Other 该参数设置为真时，将使用被参数 *OtherChar* 指定的字符作为分隔符
当参数 *Other* 设置为真时，通过 *OtherChar* 指定分隔符。如果参数 *OtherChar* 包含多于一个字符时，仅仅使用第一个字符。
参数 *FieldInfo* 是一个包含单一来源列信息的数组，该参数的解释依赖于参数 *DataType* 的值。
当参数 *DataType* 的值为 xlDelimited 时，参数 *FieldInfo* 应该是一个数组，该数组的大小应该与被转换数据的列的数量相同或更小。一个二维数组的第一维是列数(起始数为 1)，第二维是下面的数值之一，用来指明如何分析列：

数值	描述
1	通常的
2	文本
3	MDY 日期格式
4	DMY 日期格式
5	YMD 日期格式
6	MYD 日期格式
7	DYM 日期格式

8　　YDM 日期格式
9　　跳过列
如果提供给二维数组的列找不到了，那么该列将被分析为通常的设置。例如，下面参数 *FieldInfo* 的值导致第一列为文本，而第三列被跳过：
Array(Array(1,2),Array(3,9))
所有其它的列被视为通常的数据。
下面以一个实例证明，考虑有下面内容的文本文件：(将其存放在 D 盘的 excel 文件夹中，并命名为 temp.txt)
"John","Smith","Serial Record",1/2/98
"Fred","Gwynn","Serial Order Dept",2/2/98
"Mary","Davis","English Dept",3/5/98
"David","Johns","Chemistry Dept",4/4/98
下面的代码：

```
Sub test()
  Workbooks.OpenText Filename:="D:\excel\temp.txt", _
  Origin:=xlMSDOS, _
  StartRow:=1, _
  DataType:=xlDelimited, _
  TextQualifier:=xlTextQualifierDoubleQuote, _
  ConsecutiveDelimiter:=True, _
  Comma:=True, _
  FieldInfo:=Array(Array(1, 2), Array(2, 2), Array(3, 2), Array(4, 6))
End Sub
```

运行上面的代码后，将生成如下图 2 所示的工作表。注意，D 列中的单元格作为日期格式化。

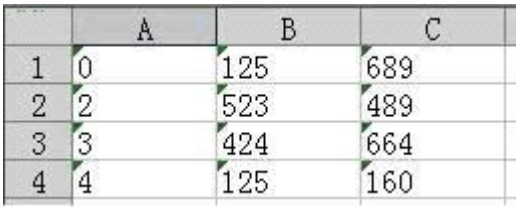
	A	B	C	D
1	John	Smith	Serial Record	1998-1-2
2	Fred	Gwynn	Serial Order Dept	1998-2-2
3	Mary	Davis	English Dept	1998-3-5
4	David	Johns	Chemistry Dept	1998-4-4

图 2　一个逗号分隔符文本文件在 Excel 中打开示例
另一方面，如果参数 *DataType* 的值是 xlFixedWidth，则每个二维数组的第一个元素指定字符在列中开始的位置(0 是第一个字符)，第二个元素指定分析选项(1-9)在列中的结果，各项数值所代表的结果描述如上。
为了证实上述描述，考虑有下面内容的文本文件：(将其存放在 D 盘的 excel 文件夹中，并命名为 temp.txt)
0-125-689
2-523-489
3-424-664
4-125-160
下面的代码：

```
Sub test()
  Workbooks.OpenText Filename:="D:\excel\temp.txt", _
  Origin:=xlMSDOS, _
  StartRow:=1, _
```

```
  DataType:=xlFixedWidth, _
  FieldInfo:=Array(Array(0, 2), Array(1, 9), Array(2, 2), Array(5, 9), Array(6, 2))
End Sub
```

运行上面的代码后，将生成如下图 3 所示的工作表。(注意，如何使用数组跳过这些连字符)



	A	B	C
1	0	125	689
2	2	523	489
3	3	424	664
4	4	125	160

图 3 一个固定宽度文本文件在 Excel 中被打开示例

最后，观察在 Excel 中打开的文本文件，并没有转换为 Excel 工作簿文件。为了保存为 Excel 工作簿文件，我们能使用 SaveAs 方法，如下所示：

```
Application.ActiveWorkbook.SaveAs _
 Filename:="D:\excel\temp.xls", FileFormat:=xlWorkbookNormal
```

将上面的代码加入上述两个子程序最后即可。

Workbook 对象及其方法、属性解析与示例(2)

分类:eBook>>《使用 VBA 编写 Excel 宏》

Workbook 对象

一个 Workbook 对象代表一个打开的 Excel 工作簿，正如我们已讨论过的，Workbook 对象存储在 Workbooks 集合中。

Workbook 对象共有 103 个属性和方法，如下表 1。

表 1　Workbook 对象成员列表

_CodeName　FullName　RefreshAll
_PrintOut<v9>　FullNameURLEncoded<v10>　RejectAllChanges
_Protect<v10>　HasMailer　ReloadAs<v9>
_ReadOnlyRecommended<v10>　HasPassword　RemovePersonalInformation<v10>
_SaveAs<v10>　HasRoutingSlip　RemoveUser
AcceptAllChanges　HighlightChangesOnScreen　Reply
AcceptLabelsInFormulas　HighlightChangesOptions　ReplyAll
Activate　HTMLProject<v9>　ReplyWithChanges<v10>
ActiveChart　IsAddin　ResetColors
ActiveSheet　IsInplace　RevisionNumber
AddToFavorites　KeepChangeHistory　Route
Application　Keywords　Routed

Author LinkInfo RoutingSlip
AutoUpdateFrequency LinkSources RunAutoMacros
AutoUpdateSaveChanges ListChangesOnNewSheet Save
BreakLink<v10> Mailer SaveAs
BuiltinDocumentProperties MergeWorkbook SaveCopyAs
CalculationVersion<v9> Modules Saved
CanCheckIn<v10> MultiUserEditing SaveLinkValues
ChangeFileAccess Name Sblt<v9>
ChangeHistoryDuration Names SendForReview<v10>
ChangeLink NewWindow SendMail
Charts OnSave SendMailer
CheckIn<v10> OnSheetActivate SetLinkOnData
Close OnSheetDeactivate SetPasswordEncryptionOptions<v10>
CodeName OpenLinks Sheets
Colors Parent ShowConflictHistory
CommandBars Password<v10> ShowPivotTableFieldList<v10>
Comments PasswordEncryptionAlgorithm<v10> SmartTagOptions<v10>
ConflictResolution PasswordEncryptionFileProperties<v10> Styles
Container PasswordEncryptionKeyLength<v10> Subject
CreateBackup PasswordEncryptionProvider<v10> TemplateRemove
ExtData
Creator Path Title
CustomDocumentProperties PersonalViewListSettings Unprotect
CustomViews PersonalViewPrintSettings UnprotectSharing
Date1904 PivotCaches UpdateFromFile
DeleteNumberFormat PivotTableWizard UpdateLink
DialogSheets Post UpdateLinks<v10>
DisplayDrawingObjects PrecisionAsDisplayed UpdateRemoteReferences
Dummy16<v10> PrintOut UserControl
Dummy17<v10> PrintPreview UserStatus
EnableAutoRecover<v10> Protect VBASigned<v9>
EndReview<v10> ProtectSharing VBProject
EnvelopeVisible<v9> ProtectStructure WebOptions<v9>
Excel4IntlMacroSheets ProtectWindows WebPagePreview<v9>
Excel4MacroSheets PublishObjects<v9> Windows
ExclusiveAccess PurgeChangeHistoryNow Worksheets
FileFormat ReadOnly WritePassword<v10>
FollowHyperlink ReadOnlyRecommended WriteReserved
ForwardMailer RecheckSmartTags<v10> WriteReservedBy

表 1 中所列的一些成员仅返回 Workbook 对象的子对象，这些子对象见图 4。表 2 给出了返回子对象的 Workbook 对象的成员。

图 4 Workbook 对象的子对象

表 2 返回子对象的 Workbook 对象成员

名称	返回类型
ActiveChart	Chart
Application	Application
Charts	Sheets
CustomViews	CustomViews
DialogSheets	Sheets
Excel4IntlMacroSheets	Sheets
Excel4MacroSheets	Sheets
Mailer	Mailer
Modules	Sheets
Names	Names
NewWindow	Window
PivotCaches	PivotCaches
PublishObjects	PublishObjects
RoutingSlip	RoutingSlip
Sheets	Sheets
SmartTagOptions	SmartTagOptions
Styles	Styles
WebOptions	WebOptions
Windows	Windows
Worksheets	Sheets

在表 2 中有一些项目值得注意。首先，ActiveSheet 属性可能返回 Chart 对象或者 Worksheet 对象，这取决于当前工作簿中所激活的是哪类对象。

第二，Charts、Sheets 和 Worksheets 属性都返回一个(不同的)Sheets 集合。特别地，Charts 对象返回在工作簿中包含所有图表工作表的 Sheets 集合(这不包括嵌入在工作表中的图表)。Worksheets 属性返回在工作簿中所有工作表的

Sheets 集合。最后，Sheets 属性返回所有工作表和图表工作表的 Sheets 集合。这是一个相当少的一个集合包含多于一类对象的例子。注意，在 Excel 对象模型中没有 Sheet 对象。

下面，让我们介绍表 1 中一些常用的成员。

Activate 方法

该方法激活工作簿，语法很简单：

Workbooks(“MyWorkBook”).Activate

注意，Workbooks 集合是全局的，不需要用 Application 进行限定。

Close 方法

Close 方法的作用是关闭工作簿。它的语法是：

工作簿对象.Close(SaveChange,FileName,RouteWorkbook)

注意，Workbook 对象的 Close 方法有三个参数，与 Workbooks 集合中的 Close 方法没有参数不一样。

可选参数 *SaveChanges* 用于在关闭工作簿前保存工作簿所发生的变化，特别地，如果工作簿没有变化，该参数将被忽略。如果工作簿显现在其它打开的窗口中，该参数也被忽略。另一方面，如果工作簿发生了变化且没有显现在任何其它打开的窗口中，该参数将生效。

在这种情况下，如果 *SaveChanges* 的值为 True，将存储该变化。如果仍没有一个文件名与工作簿相联系(也就是说，如果先前该工作簿没有保存)，那么将使用在参数 *FileName* 中所设置的名称。如果参数 *FileName* 也被忽略，Excel 将提示用户输入一个文件名。如果参数 *SaveChanges* 的值为 False，则工作簿所发生的变化不会被保存。最后，如果参数 *SaveChanges* 被忽略，Excel 将显示一个对话框询问是否保存工作簿所发生的变化。简而言之，该方法的行为正如你所希望的。

可选参数 *RouteWorkbook* 指出工作簿传送的问题，如果您对该参数感兴趣，可以在 ExcelVBA 帮助文件中获得更多的信息。

很重要的是，您要注意 Close 方法检查工作簿的 Saved 属性，以决定是否去提示用户保存工作簿所发生的变化。如果我们设置 Saved 属性的值为 True，那么 Close 方法将没有警告而直接关闭工作簿，不会保存工作簿所发生的任何变化。

DisplayDrawingObjects 属性

该属性返回或者设置一个值，表示如何显示形状，可以是下面的 XlDisplayShapes 常量之一：

```
Enum XlDisplayShapes
    XlDisplayShapes=-4101
    xlPlaceholders=2
    xlHide=3
End Enum
```

FileFormat 属性(只读)

该属性返回工作簿文件格式或者类型，可以是下面的 XlFileFormat 常量之一：

```
Enum XlFileFormat
xlAddIn = 18
xlCSV = 6
xlCSVMac = 22
xlCSVMSDOS = 24
```

xlCSVWindows = 23
xlCurrentPlatformText = -4158
xlDBF2 = 7
xlDBF3 = 8
xlDBF4 = 11
xlDIF = 9
xlExcel2 = 16
xlExcel2FarEast = 27
xlExcel3 = 29
xlExcel4 = 33
xlExcel4Workbook = 35
xlExcel5 = 39
xlExcel7 = 39
xlExcel9795 = 43
xlHtml = 44
xlIntlAddIn = 26
xlIntlMacro = 25
xlSYLK = 2
xlTemplate = 17
xlTextMac = 19
xlTextMSDOS = 21
xlTextPrinter = 36
xlTextWindows = 20
xlUnicodeText = 42
xlWebArchive = 45
xlWJ2WD1 = 14
This is the xlWJ3 = 40
xlWJ3FJ3 = 41
xlWK1 = 5
xlWK1ALL = 31
xlWK1FMT = 30
xlWK3 = 15
xlWK3FM3 = 32
xlWK4 = 38
xlWKS = 4
xlWorkbookNormal = -4143
xlWorks2FarEast = 28
xlWQ1 = 34
xlXMLSpreadsheet = 46
End Enum

Name，FullName，Path 属性

Name 属性返回工作簿的名字，Path 属性返回工作簿文件路径，FullName 属性返回工作簿文件完整的描述(路径和文件名)。所有这些属性都是只读的。

注意，使用如下代码：

Application.Path
将返回 Excel 应用程序路径(而不是工作簿路径)。

Workbook 对象及其方法、属性解析与示例(3)
分类:eBook>>《使用 VBA 编写 Excel 宏》

HasPassword 属性(只读/布尔值)
如果工作簿有密码保护，则该只读属性值为 True。注意，密码能作为 SaveAs 方法中的一个参数被指定。
PrecisionAsDisplayed 属性(可读写/布尔值)
当该属性的值为 True 时，在工作簿进行计算时，将仅使用工作表中所显示的数值进行计算，而不是它实际所存诸的值。该属性的缺省值为 False，工作表计算基于它们所存储的数值。
PrintOut 方法
PrintOut 方法打印完整的工作簿。(该方法也应用于一些其它的对象，诸如 Range、Worksheet、Chart)其语法为：
工作簿对象.PrintOut(*From,To,Copies,Preview,ActivePrinter,PrintToFile,Collate*)
注意，该方法的所有参数均是可选的。
参数 *From* 指定需要打印第一页的页码，参数 *To* 指定要打印的最后一页。如果忽略这些参数，将打印整个对象(如范围、工作表等)。
参数 *Copies* 指定要打印副本的数量，缺省值为 1。
将参数 *Preview* 设置为 True，将弹出打印预览而不是立即打印，缺省值为 False。
参数 *ActivePrinter* 设置活动打印机的名字。另一方面，设置参数 *PrintToFile* 的值为 True 将导致 Excel 打印到一个文件，Excel 将提示用户该输出文件的名字。(不巧的是，没有办法用代码指定所要输出文件的名字)。
应该设置参数 *Collate* 的值为 True，以逐份打印每份副本。
PrintPreview 方法
该方法产生 Excel 的打印预览功能，其语法是：
工作簿对象.PrintPreview
注意，PrintPreview 方法应用到与 PrintOut 方法相同的一组对象。
Protect 方法
该方法保护工作簿，以便它不能被修改。其语法为：
工作簿对象.Protect(*Password,Structure,Windows*)
该方法也应用于图表或者工作表，但语法略有不同。
可选参数 *Password* 指定一个密码(对大小写敏感)，如果忽略该参数，工作簿没有被保护，不需要密码。
设置可选参数 *Structure* 的值为 True，以保护工作簿的结构——也就是说，在工作簿中工作表的相关位置，缺省值为 False。
设置可选参数 *Windows* 的值为 True，以保护工作簿窗口，缺省值为 False。
ReadOnly 属性(只读/布尔值)
如果工作簿作为只读被打开，则该属性的值为 True。
RefreshAll 方法

该方法更新工作簿中所有外部数据区域和数据透视表，其语法为：
工作簿对象.RefreshAll

Save 方法

该方法保存工作簿中的任何变化。其语法为：
工作簿对象.Save

SaveAs 方法

该方法在指定的文件中保存工作簿所发生的变化。其语法为：
Expression.SaveAs(*Filename,FileFormat,Password,WriteResPassword, _*
ReadOnlyRecommended,CreateBackup,AccessMode,ConflictResolution, _
AddToMru,TextCodePage,TextVisualLayout)
参数 *Filename* 指定使用保存到磁盘中的文件名，如果没有包括路径，Excel 将使用当前文件夹。
参数 *FileFormat* 指定所使用保存文件时的文件格式，其值为我们在前面的 FileFormat 属性中所介绍过的 XlFileFormat 常量之一。
参数 *Password* 在保存文件时用来指定密码，能使用的任何字符，区分大小写但不得超过 15 个字符。
参数 *WriteResPassword* 是一个字符串，指定文件的写保护密码。如果一个文件带有写保护密码保存，当下一次没有提供密码打开该文件时，该文件将以只读方式打开。
我们可以设置参数 *ReadOnlyRecommended* 的值为 True，当文件打开时显示一个消息。建议文件以只读方式找开。
设置参数 *CreateBackup* 的值为真，以创建一个备份文件。
参数 *AccessMode* 和 *ConflictResolution* 指定共享问题，您能在 ExcelVBA 帮助系统中找到详细的介绍。
设置参数 *AddToMru* 的值为 True，以添加工作簿到最近使用的文件列表中，缺省值为 False。
其它参数不会在英文版的 Excel 中使用。

SaveCopyAs 方法

该方法保存工作簿的一份副本，但不会修改已打开的工作簿。其语法为：
工作簿对象.SaveCopyAs(*Filename*)
参数 *Filename* 指定原文件副本的文本名。

Saved 属性(可读写*I*布尔值)

如果工作簿自上次保存以来没有发生任何变化，则该属性值为 True。注意，该属性的值是可读写的，这意味着我们能设置该属性的值为 True，即使该工作簿在上次保存之后发生过变化。正如前面所介绍过的，我们能设置该属性的值为 True，关闭被修改过的工作簿，而不提示保存当前已发生的变化。

Workbook 对象的子对象

图 5 所显示的是 Workbook 对象的子对象(该图为图 4 的重复)。

图 5　Workbook 对象的子对象

让我们快速地介绍一下这些子对象中的一部分。(我们将在本书稍后部分介绍 Window 对象、Worksheet 对象和 WorkbookEvents 对象)

CustomView 对象

CustomViews 属性返回 CustomViews 集合，在该集合中的每一个 CustomView 对象代表工作簿的自定义预览。CustomView 对象是相当简单的，因此，我们将只介绍一个示例。如图 6 所示的工作表。

	A	B	C
1	Year	ItemCode	Quantity
2	1997	20	30
3	1997	50	60
4	1997	13	90
5	1998	15	56
6	1998	36	67
7	1998	44	78

图 6　CustomView 对象示例

如果我们使用 Autofilter 命令对年份进行筛选，则结果如图 7 所示。

	A	B	C
1	Year	ItemCod	Quantity
5	1998	15	56
6	1998	36	67
7	1998	44	78

图 7　筛选后的结果

下面的代码将这个自定义的视图命名为 View1998：

```
ThisWorkbook.CustomViews.Add  "View1998"
```

现在我们可以使用下面的代码在任何时间显示该视图：

```
ThisWorkbook.CustomViews!View1998.Show
```

或者：

```
strView  =  "View1998"
```

tiveWorkbook.CustomViews(strView).Show

Names 集合

与 Application 对象一样，Workbook 对象有一个 Name 属性，返回一个 Names 集合，该集合代表与工作簿相关的 Name 对象。要详细了解 Name 对象的信息，请参见第 16 章 Application 对象。

Sheets 集合

Sheets 属性返回一个 Sheets 集合，包含在工作簿中每个工作表中的 Worksheet 对象和每个图表工作表中的 Chart 对象。我们将在本书稍后的章节介绍 Worksheet 对象和 Chart 对象。

Styles 集合和 Style 对象

一个 Style 对象代表单元格区域一组格式选项。每个工作簿有一个 Styles 集合，包含该工作簿所有 Styles 对象。

为了对某单元格区域应用样式，我们可以编写下面的代码：

RangeObject.Style=styleName

在这里，StyleName 是某样式的名称。

为了创建一个 Style 对象，可以使用 Add 方法，其语法为：

工作簿对象.Add(*Name,BasedOn*)

注意，Add 方法返回新创建的 Style 对象。

参数 *Name* 指定样式的名称。可选参数 *BaseOn* 指定某 Range 对象，该对象指向所运用样式的单元格作为新样式的基础，如果忽略该参数，新创建的样式以 Normal(正常)样式为基础。

注意，根据文档资料，如果样式所指定的名称已存在，则 Add 方法将基于在参数 *BaseOn* 中所指定的单元格重新定义已存在的样式。(然而，在我的系统中，Excel 报出了一个错误信息，因此您应仔细检查。)

Style 对象的属性代表了不同的格式特征，诸如字体名称、字体大小、数字格式、对齐，等等。也有几个内置的样式，诸如 Normal、Currency 和 Percent，这些内置的样式能在 Style 对话框中的 Style 名称框找到(在格式菜单下)。

下面举一个例子。示例代码将创建一个样式，然后应用它到当前工作表中一个独立的范围中：

```
'*********************************************************************
Sub test3()
  Dim st As Style
'如果样式已存在则删除
  For Each st In ActiveWorkbook.Styles
    If st.Name = "Bordered" Then st.Delete
  Next
'创建样式
  With ActiveWorkbook.Styles.Add(Name:="Bordered")
    .Borders(xlTop).LineStyle = xlDouble
    .Borders(xlBottom).LineStyle = xlDouble
    .Borders(xlLeft).LineStyle = xlDouble
    .Borders(xlRight).LineStyle = xlDouble
    .Font.Bold = True
    .Font.Name = "Arial"
```

```
    .Font.Size = 36
  End With
  '应用样式
  Application.ActiveSheet.Range("A1:B3").Style = "Bordered"
End Sub
'*************************************************************************
```

示例：对工作簿中的工作表排序

让我们添加一个新的功能到我们的 SRXUtils 应用程序中。如果您要处理带有多个工作表(工作表和图表工作表)的工作簿，则您可能想按字母顺序排列工作表。对工作表进行排序的基本代码是 Move 方法，其语法是：

SheetsObject.Move(*Before,After*)

当然，为了有效地使用该方法，我们需要工作表名称的排序列表。

第一步是通过对新的功能添加一个新行来增加 DataSheet 工作表，如图 8 所示。(在 DataSheet 工作表中行的顺序是基于我们想显现在自定义菜单中的项目的顺序。)

	A	B	C	D	E	F	G	H
1	**Utility**	**OnAction Proc**	**Procedure**	**In Workbook**	**Menu Item**	**SubMenu Item**	**On Wks Menu**	**On Chart Menu**
2	Activate Sheet	RunUtility	ActivateSheet	ThisWorkbook	&Activate Sheet		TRUE	TRUE
3	Print Charts	RunUtility	PrintCharts	Print.utl	&Print	Embedded &Charts	TRUE	TRUE
4	Print Pivot Tables	RunUtility	PrintPivotTables	Print.utl		&Pivot Tables	TRUE	TRUE
5	Print Sheets	RunUtility	PrintSheets	Print.utl		&Sheets	TRUE	TRUE
6	Sort Sheets	RunUtility	SortSheets	ThisWorkbook	&Sort Sheets		TRUE	TRUE

图 8　增加的 DataSheet 工作表

下一步，我们插入一个名为 basSortSheets 的代码模块，它将包含实现这个功能的代码。

在 basSortSheets 模块中包括两个过程。第一个过程验证用户是否真的想排序工作表，如果想排序工作表的话，调用第二个过程去完成该项工作。第一个过程见示例 1，将在图 8 中显示对话框。

示例 1：SortSheets 过程

```
'*************************************************************************
Sub SortSheets()
  If MsgBox("Sort the sheets in this workbook?", _
    vbOKCancel + vbQuestion, "Sort Sheets") = vbOK Then
      SortAllsheets
  End If
End Sub
'*************************************************************************
```

产生动作的过程见示例 2。该过程首先在数组中收集工作表的名称，接着在新的工作表中放置该数组，然后使用 Sort 方法(应用于 Range 对象，将在第 19 章中介绍)对这些名称排序。接着，用排序好的数据重新填充数组。最后，使用 Move 方法重新排列这些工作表。

示例 2：SortAllSheets 过程

```
'*************************************************************************
```

```
Sub SortAllSheets()
'排序工作表
  Dim wb As Workbook
  Dim ws As Worksheet
  Dim rng As Range, i As Integer
  Dim cSheets As Integer
  Dim sSheets() As String

  Set wb = ActiveWorkbook

  '获取数组实际大小
  cSheets = wb.Sheets.Count
  ReDim sSheets(1 To cSheets)

  '用工作表名填充数组
  For i = 1 To cSheets
    sSheets(i) = wb.Sheets(i).Name
  Next

'创建新的工作表并在其第一列放置名称
  Set ws = wb.Worksheets.Add
  For i = 1 To cSheets
    ws.Cells(i, 1).Value = sSheets(i)
  Next

'对列排序
  ws.Columns(1).Sort Key1:=ws.Columns(1), Order1:=xlAscending

  '重新填充数组
  For i = 1 To cSheets
    sSheets(i) = ws.Cells(i, 1).Value
  Next

  '删除临时工作表
  Application.DisplayAlerts = False
  ws.Delete
  Application.DisplayAlerts = True

  '通过移动每个工作表到最后来重新排列工作表
  For i = 1 To cSheets
    wb.Sheets(sSheets(i)).Move after:=wb.Sheets(cSheets)
  Next

End Sub
```

```
'*********************************************************************
```

一旦代码插入完成后，您能保存 SRXUtils.xls 工作簿作为一个加载宏。不要忘记了首先要卸载该加载宏，否则 Excel 将会报错。

<完>

第五章 Range 对象基本操作应用示例(1)

Range 对象可能是 VBA 代码中最常用的对象，Range 对象可以是某一单元格、某一单元格区域、某一行、某一列、或者是多个连续或非连续的区域组成的区域。下面介绍 Range 对象的一些属性和方法。

- -

[示例 05-01] 赋值给某单元格

```
[示例 05-01-01]
Sub test1()
Worksheets("Sheet1").Range("A5").Value = 22
 MsgBox "工作表 Sheet1 内单元格 A5 中的值为" _
 & Worksheets("Sheet1").Range("A5").Value
End Sub
[示例 05-01-02]
Sub test2()
  Worksheets("Sheet1").Range("A1").Value = _
    Worksheets("Sheet1").Range("A5").Value
  MsgBox "现在 A1 单元格中的值也为" & _
    Worksheets("Sheet1").Range("A5").Value
End Sub
[示例 05-01-03]
Sub test3()
  MsgBox "用公式填充单元格,本例为随机数公式"
  Range("A1:H8").Formula = "=Rand()"
End Sub
[示例 05-01-04]
Sub test4()
  Worksheets(1).Cells(1, 1).Value = 24
  MsgBox "现在单元格 A1 的值为 24"
End Sub
[示例 05-01-05]
Sub test5()
  MsgBox "给单元格设置公式,求 B2 至 B5 单元格区域之和"
  ActiveSheet.Cells(2, 1).Formula = "=Sum(B1:B5)"
End Sub
[示例 05-01-06]
Sub test6()
  MsgBox "设置单元格 C5 中的公式."
```

```
  Worksheets(1).Range("C5:C10").Cells(1, 1).Formula = "=Rand()"
End Sub
```

- -

[示例 05-02] 引用单元格

```
Sub Random()
  Dim myRange As Range
  '设置对单元格区域的引用
  Set myRange = Worksheets("Sheet1").Range("A1:D5")
  '对 Range 对象进行操作
  myRange.Formula = "=RAND()"
  myRange.Font.Bold = True
End Sub
```

示例说明：可以设置 Range 对象变量来引用单元格区域，然后对该变量所代表的单元格区域进行操作。

- -

[示例 05-03] 清除单元格

[示例 05-03-01]清除单元格中的内容(ClearContents 方法)

```
Sub testClearContents()
  MsgBox "清除指定单元格区域中的内容"
  Worksheets(1).Range("A1:H8").ClearContents
End Sub
```

[示例 05-03-02]清除单元格中的格式(ClearFormats 方法)

```
Sub testClearFormats()
  MsgBox "清除指定单元格区域中的格式"
  Worksheets(1).Range("A1:H8").ClearFormats
End Sub
```

[示例 05-03-03]清除单元格中的批注(ClearComments 方法)

```
Sub testClearComments()
  MsgBox "清除指定单元格区域中的批注"
  Worksheets(1).Range("A1:H8").ClearComments
End Sub
```

[示例 05-03-04]清除单元格中的全部，包括内容、格式和批注(Clear 方法)

```
Sub testClear()
  MsgBox "彻底清除指定单元格区域"
  Worksheets(1).Range("A1:H8").Clear
End Sub
```

- -

[示例 05-04] Range 和 Cells

```
Sub test()
  '设置单元格区域 A1:J10 的边框线条样式
  With Worksheets(1)
    .Range(.Cells(1, 1), _
       .Cells(10, 10)).Borders.LineStyle = xlThick
  End With
```

```
End Sub
```
示例说明：可用 Range(cell1, cell2) 返回一个 Range 对象，其中 cell1 和 cell2 为指定起始和终止位置的 Range 对象。

- -

[示例 05-05] 选取单元格区域(Select 方法)

```
Sub testSelect()
  '选取单元格区域 A1:D5
  Worksheets("Sheet1").Range("A1:D5").Select
End Sub
```

- -

[示例 05-06] 基于所选区域偏离至另一区域(Offset 属性)

[示例 05-06-01]

```
Sub testOffset()
  Worksheets("Sheet1").Activate
  Selection.Offset(3, 1).Select
End Sub
```

示例说明：可用 Offset(row, column)(其中 row 和 column 为行偏移量和列偏移量)返回相对于另一区域在指定偏移量处的区域。如上例选定位于当前选定区域左上角单元格的向下三行且向右一列处单元格区域。

[示例 05-06-02] 选取距当前单元格指定行数和列数的单元格

```
Sub ActiveCellOffice()
  MsgBox "显示距当前单元格第 3 列、第 2 行的单元格中的值"
  MsgBox ActiveCell.Offset(3, 2).Value
End Sub
```

- -

[示例 05-07] 调整区域的大小(Resize 属性)

```
Sub ResizeRange()
  Dim numRows As Integer, numcolumns As Integer
  Worksheets("Sheet1").Activate
  numRows = Selection.Rows.Count
  numcolumns = Selection.Columns.Count
  Selection.Resize(numRows + 1, numcolumns + 1).Select
End Sub
```

示例说明：本示例调整所选区域的大小，使之增加一行一列。

- -

[示例 05-08] 选取多个区域(Union 方法)

```
Sub testUnion()
  Dim rng1 As Range, rng2 As Range, myMultiAreaRange As Range
  Worksheets("sheet1").Activate
  Set rng1 = Range("A1:B2")
  Set rng2 = Range("C3:D4")
  Set myMultiAreaRange = Union(rng1, rng2)
  myMultiAreaRange.Select
End Sub
```

示例说明：可用 Union(range1, range2, ...) 返回多块区域，即该区域由两个或多个连续的单元格区域所组成。如上例创建由单元格区域 A1:B2 和 C3:D4 组合定义的对象，然后选定该定义区域。

- -

[示例 05-09] 激活已选区域中的单元格

```
Sub ActivateRange()
  MsgBox "选取单元格区域 B2:D6 并将 C4 选中"
  ActiveSheet.Range("B3:D6").Select
  Range("C5").Activate
End Sub
```

- -

[示例 05-10] 选取指定条件的单元格(SpecialCells 方法)

```
Sub SelectSpecialCells()
  MsgBox "选择当前工作表中所有公式单元格"
  ActiveSheet.Cells.SpecialCells(xlCellTypeFormulas).Select
End Sub
```

- -

[示例 05-11] 选取矩形区域(CurrentRegion 属性)

```
'选取包含当前单元格的矩形区域
'该区域周边为空白行和空白列
Sub SelectCurrentRegion()
  MsgBox "选取包含当前单元格的矩形区域"
  ActiveCell.CurrentRegion.Select
End Sub
```

- -

[示例 05-12] 选取当前工作表中已用单元格(UsedRange 属性)

```
'选取当前工作表中已使用的单元格区域
Sub SelectUsedRange()
  MsgBox "选取当前工作表中已使用的单元格区域" _
    & vbCrLf & "并显示其地址"
  ActiveSheet.UsedRange.Select
  MsgBox ActiveSheet.UsedRange.Address
End Sub
```

- -

[示例 05-13] 选取最边缘单元格(End 属性)

```
'选取最下方的单元格
Sub SelectEndCell()
  MsgBox "选取当前单元格区域内最下方的单元格"
  ActiveCell.End(xlDown).Select
End Sub
```

示例说明：可以改变参数 xlDown 以选取最左边、最右边、最上方的单元格。

- -

[示例 05-14]设置当前单元格的前一个单元格和后一个单元格的值

```
Sub SetCellValue()
```

```
  MsgBox "将当前单元格中前面的单元格值设为""我前面的单元格""" & vbCrL
f _
   & "后面的单元格值设为""我后面的单元格"""
  ActiveCell.Previous.Value = "我前面的单元格"
  ActiveCell.Next.Value = "我后面的单元格"
End Sub
```

- -

[示例 05-15]确认所选单元格区域中是否有公式(HasFormula 属性)

```
Sub IfHasFormula()
  If Selection.HasFormula = True Then
    MsgBox "所选单元格中都有公式"
  Else
    MsgBox "所选单元格中，部分单元格没有公式"
  End If
End Sub
```

- -

[示例 05-16] 公式单元格操作

[示例 05-16-01]获取与运算结果单元格有直接关系的单元格

```
Sub CalRelationCell()
  MsgBox "选取与当前单元格的计算结果相关的单元格"
  ActiveCell.DirectPrecedents.Select
End Sub
```

[示例 05-16-02]追踪公式单元格

```
Sub Cal1()
  MsgBox "选取计算结果单元格相关的所有单元格"
  ActiveCell.Precedents.Select
End Sub
Sub TrackCell()
  MsgBox "追踪运算结果单元格"
  ActiveCell.ShowPrecedents
End Sub
Sub DelTrack()
  MsgBox "删除追踪线"
  ActiveCell.ShowPrecedents Remove:=True
End Sub
```

- -

[示例 05-17] 复制单元格(Copy 方法)

```
Sub CopyRange()
  MsgBox "在单元格 B7 中写入公式后,将 B7 的内容复制到 C7:D7 内"
  Range("B7").Formula = "=Sum(B3:B6)"
  Range("B7").Copy Destination:=Range("C7:D7")
End Sub
```

- -

[示例 05-18]获取单元格行列值(Row 属性和 Column 属性)

```
Sub RangePosition()
  MsgBox "显示所选单元格区域的行列值"
  MsgBox "第 " & Selection.Row & "行 " & Selection.Column & "列"
End Sub
```

- -

[示例 05-19]获取单元格区域的单元格数及行列数(Rows 属性、Columns 属性和 Count 属性)

```
Sub GetRowColumnNum()
  MsgBox "显示所选取单元格区域的单元格数、行数和列数"
  MsgBox "单元格区域中的单元格数为:" & Selection.Count
  MsgBox "单元格区域中的行数为:" & Selection.Rows.Count
  MsgBox "单元格区域中的列数为:" & Selection.Columns.Count
End Sub
```

- -

[示例 05-20]设置单元格中的文本格式

[示例 05-20-01] 对齐文本

```
Sub HorizontalAlign()
  MsgBox "将所选单元格区域中的文本左右对齐方式设为居中"
  Selection.HorizontalAlignment = xlHAlignCenter
End Sub
Sub VerticalAlign()
  MsgBox "将所选单元格区域中的文本上下对齐方式设为居中"
  Selection.RowHeight = 36
  Selection.VerticalAlignment = xlVAlignCenter
End Sub
```

[示例 05-20-02] 缩排文本(InsertIndent 方法)

```
Sub Indent()
  MsgBox "将所选单元格区域中的文本缩排值加 1"
  Selection.InsertIndent 1
  MsgBox "将缩排值恢复"
  Selection.InsertIndent -1
End Sub
```

[示例 05-20-03] 设置文本方向(Orientation 属性)

```
Sub ChangeOrientation()
  MsgBox "将所选单元格中的文本顺时针旋转 45 度"
  Selection.Orientation = 45
  MsgBox "将文本由横向改为纵向"
  Selection.Orientation = xlVertical
  MsgBox "将文本方向恢复原值"
  Selection.Orientation = xlHorizontal
End Sub
```

[示例 05-20-04]自动换行(WrapText 属性)

```
Sub ChangeRow()
```

```
  Dim i
  MsgBox "将所选单元格设置为自动换行"
  i = Selection.WrapText
  Selection.WrapText = True
  MsgBox "恢复原状"
  Selection.WrapText = i
End Sub
```

[示例 05-20-05]将比单元格列宽长的文本缩小到能容纳列宽大小(ShrinkToFit 属性)

```
Sub AutoFit()
  Dim i
  MsgBox "将长于列宽的文本缩到与列宽相同"
  i = Selection.ShrinkToFit
  Selection.ShrinkToFit = True
  MsgBox "恢复原状"
  Selection.ShrinkToFit = i
End Sub
```

- -

[示例 05-21]设置条件格式(FormatConditions 属性)

```
Sub FormatConditions()
  MsgBox "在所选单元格区域中将单元格值小于10的单元格中的文本变为红色
"
  Selection.FormatConditions.Add Type:=xlCellValue, _
 Operator:=xlLessEqual, Formula1:="10"
  Selection.FormatConditions(1).Font.ColorIndex = 3
  MsgBox "恢复原状"
  Selection.FormatConditions(1).Font.ColorIndex = xlAutomatic
End Sub
```

- -

[示例 05-22]插入批注(AddComment 方法)

```
Sub EnterComment()
  MsgBox "在当前单元格中输入批注"
  ActiveCell.AddComment ("Hello")
  ActiveCell.Comment.Visible = True
End Sub
```

- -

[示例 05-23]隐藏/显示单元格批注

```
Sub CellComment()
  MsgBox "切换当前单元格批注的显示和隐藏状态"
  ActiveCell.Comment.Visible = Not (ActiveCell.Comment.Visible)
End Sub
```

- -

[示例 05-24]改变所选单元格的颜色

```
Sub ChangeColor()
```

```
  Dim iro As Integer
  MsgBox "将所选单元格的颜色改为红色"
  iro = Selection.Interior.ColorIndex
  Selection.Interior.ColorIndex = 3
  MsgBox "将所选单元格的颜色改为蓝色"
  Selection.Interior.Color = RGB(0, 0, 255)
  MsgBox "恢复原状"
  Selection.Interior.ColorIndex = iro
End Sub
```

- -

[示例 05-25]改变单元格的图案

```
Sub ChangePattern()
  Dim p, pc, i
  MsgBox "依 Pattern 常数值的顺序改变所选单元格的图案"
  p = Selection.Interior.Pattern
  pc = Selection.Interior.PatternColorIndex
  For i = 9 To 16
    With Selection.Interior
      .Pattern = i
      .PatternColor = RGB(255, 0, 0)
    End With
    MsgBox "常数值 " & i
  Next i
  MsgBox "恢复原状"
  Selection.Interior.Pattern = p
  Selection.Interior.PatternColorIndex = pc
End Sub
```

- -

[示例 05-26]合并单元格

```
Sub MergeCells()
  MsgBox "合并单元格 A2:C2,并将文本设为居中对齐"
  Range("A2:C2").Select
  With Selection
    .MergeCells = True
    .HorizontalAlignment = xlCenter
  End With
End Sub
```

- -

[示例 05-27]限制单元格移动的范围

```
Sub ScrollArea1()
  MsgBox "将单元格的移动范围限制在单元格区域 B2:D6 中"
  ActiveSheet.ScrollArea = "B2:D6"
End Sub
Sub ScrollArea2()
```

```
  MsgBox "解除移动范围限制"
  ActiveSheet.ScrollArea = ""
End Sub
```

- -

[示例 05-28]获取单元格的位置(Address 属性)

```
Sub GetAddress()
  MsgBox "显示所选单元格区域的地址"
  MsgBox "绝对地址:" & Selection.Address
  MsgBox "行的绝对地址:" & Selection.Address(RowAbsolute:=False)
  MsgBox "列的绝对地址:" & Selection.Address(ColumnAbsolute:=False)
  MsgBox "以 R1C1 形式显示:" & Selection.Address(ReferenceStyle:=xlR1C
1)
  MsgBox "相对地址:" & Selection.Address(False, False)
End Sub
```

- -

[示例 05-29]删除单元格区域(Delete 方法)

```
Sub DeleteRange()
  MsgBox "删除单元格区域 C2:D6 后,右侧的单元格向左移动"
  ActiveSheet.Range("C2:D6").Delete (xlShiftToLeft)
End Sub
```

分类:ExcelVBA>>ExcelVBA 编程入门范例

第五章 Range 对象基本操作应用示例(2)

- -

小结

下面对 Range 对象的一些常用属性和方法进行简单的小结。

1、Activate 与 Select

试验下面的过程：

```
Sub SelectAndActivate()
  Range("B3:E10").Select
  Range("C5").Activate
End Sub
```

其结果如下图所示：

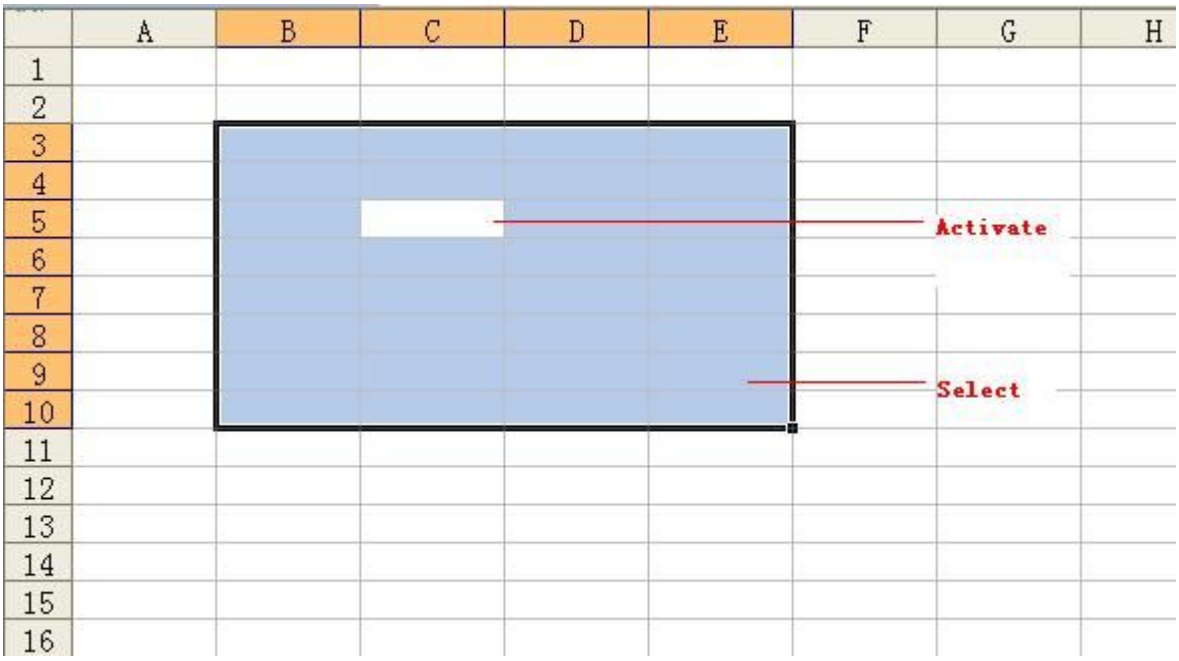


图 05-01：Select 与 Activate

Selection 指单元格区域 B3：E10，而 ActiveCell 则是单元格 C5；ActiveCell 代表单个的单元格，而 Selection 则可以代表单个单元格，也可以代表单元格区域。

2、Range 属性

可以使用 Application 对象的 Range 属性引用 Range 对象，如

Application.Range(“B2”) ‘代表当前工作表中的单元格 B2

若引用当前工作表中的单元格，也可以忽略前面的 Application 对象。

Range(“A1:D10”) ‘代表当前工作表中的单元格区域 A1:D10

Range(“A1:A10,C1:C10,E1:E10”) ‘代表当前工作表中非连续的三个区域组成的单元格区域

Range 属性也接受指向单元格区域对角的两个参数，如：

Range(“A1”,”D10”) ‘代表单元格区域 A1：D10

当然，Range 属性也接受单元格区域名称，如：

Range(“Data”) ‘代表名为 Data 的数据区域

Range 属性的参数可以是对象也可以是字符串，如：

Range(“A1”,Range(“LastCell”))

3、单元格引用的快捷方式

可以在引用区域两侧加上方括号来快速引用单元格区域，如：

[B2]

[A1:D10]

[A1:A10,C1:C10,E1:E10]

[Data]

但其引用的是绝对区域。

4、Cells 属性

可以使用 Cells 属性来引用 Range 对象。如：

ActiveSheet.Cells

Application.Cells ‘引用当前工作表中的所有单元格

Cell(2,2)
Cell(2,”B”) ‘引用单元格 B2
Range(Cells(1,1),Cells(10,5)) ‘引用单元格区域 A1:E10
若想在一个单元格区域中循环时，使用 Cells 属性是很方便的。
也可以使用 Cells 属性进行相对引用，如：
Range(“D10:G20”).Cells(2,3) ‘表示引用单元格区域 D10:G20 中第 2 行第 3 列的单元格，即单元格 F11
也可使用语句：Range(“D10”).Cells(2,3)达到同样的引用效果。

5、Offset 属性

Offset 属性基于当前单元格按所给参数进行偏移，与 Cells 属性不同的是，它基于 0 即基准单元格为 0，如：
Range(“A10”).Cells(1,1)和 Range(“A10”).Offset(0,0)都表示单元格 A10
当想引用于基准单元格区域同样大小的单元格区域时，则 Offset 属性是有用的。

6、Resize 属性

可使用 Resize 属性获取相对于原单元格区域左上角单元格指定大小的区域。

7、SpecialCells 方法

SpecialCells 方法对应于“定位条件”对话框，如图 05-02 所示：

图 05-02：“定位条件”对话框

8、CurrentRegion 属性

使用 CurrentRegion 属性可以选取当前单元格所在区域，即周围是空行和空列所围成的矩形区域，等价于“Ctrl+Shift+*”快捷键。

9、End 属性

End 属性所代表的操作等价于“Ctrl+方向箭”的操作，使用常量 xlUp、xlDown、xlToLeft 和 xlToRight 分别代表上、下、左、右箭。

10、Columns 属性和 Rows 属性

Columns 属性和 Rows 属性分别返回单元格区域中的所有列和所有行。

11、Areas 集合

在多个非连续的单元格区域中使用 Columns 属性和 Rows 属性时，只是返回第一个区域的行或列，如：
Range(“A1:B5,C6:D10,E11:F15”).Rows.Count

将返回 5。
此时应使用 Areas 集合来返回区域中每个块的地址，如：

```
For Each Rng In Range("A1:B5,C6:D10,E11:F15").Areas
  MsgBox Rng.Address
Next Rng
```

12、Union 方法和 Intersect 方法

当想从两个或多个单元格区域中生成一个单元格区域时，使用 Union 方法；当找到两个或多个单元格区域共同拥有的单元格区域时，使用 Intersect 方法。

操作单元格或单元格区域有很多有用的技巧，这需要在实践中总结和归纳。有关单元格区域的操作也可参见《在 VBA 代码中引用 Excel 工作表中单元格区域的方式小结》和《使用 VBA 代码选择单元格/区域》。

分类:ExcelVBA>>ExcelVBA 编程入门范例