Microsoft Word 对象

赢政天下论坛

http://bbs.winzheng.com

三戒 制作

qinyangdl@163.com

目录

序言

Visual Basic 支持一个对象集合，该集合中的对象直接对应于 Microsoft Word 97 中的元素，并且通过用户界面，用户熟悉这些元素中的绝大多数。例如，Document 对象代表了一个打开的文档，Bookmark 对象代表了一个文档中的书签，而 Selection 对象则代表了在一个文档窗口窗格中的选定内容。在 Word 中，每一类元素－文档、表格、段落、书签、域等等－都可以用 Visual Basic 的对象来表示。要在 Word 中自动执行任务，可以使用这些对象的方法和属性。

关于理解和使用 Microsoft Office 97 对象模型的一般性内容，请参阅本书的第二章“理解对象模型”。在 Microsoft Office 97 中的对象模型相当丰富，其中包含了大约 180 个对象。要查看 Word 对象模型的层次关系图，请参阅“帮助”中的“Microsoft Word 对象”。要获得对某种特定对象的详细描述，可以在此图中单击该对象的名字，或是在“帮助”的索引中对特定对象进行搜索。

怎样显示 Word Visual Basic 帮助

要使用 Word Visual Basic 帮助，必须在安装过程中选择“自定义”的安装方式，并且为 Word 选中“Visual Basic 在线帮助”复选框。否则，安装程序不会安装 Visual Basic 帮助。如果用户已经安装好了 Word，那么可以再次运行 Setup 程序来安装 Visual Basic 帮助。

要查看 Word Visual Basic 帮助，可以在“Visual Basic 编辑器里的“帮助”菜单中点击“目录和索引”一项。在“帮助主题”对话框的“目录”一表中，双击“Microsoft Word Visual Basic 参考”，然后再双击“Visual Basic 参考”。“帮助主题”对话框会重新出现，显示出 Microsoft Word Visual Basic 的目录和索引。

1 运用 Application 对象

在用户启动一个 Word 时段的同时，也创建了一个 Application 对象。用户可以使用 Application 对象的属性或方法来控制或返回应用程序范围内的特性、控制应用程序窗口的外观或者调整 Word 对象模型的其他方面。可以使用 Application 属性来返回 Word Application 对象。以下的示例从视图状态切换到打印预览状态。

```
Application. PrintPreview = True
```

Application 对象的一些属性控制着应用程序的外观。例如，如果 DisplayStatusBar 属性为 True(真)，那么状态栏是可见的，如果 WindowState 属性值为 wdWindowStateMaximize，那么应用程序窗口处于最大化状态。以下的示例设置了屏幕上应用程序窗口的大小。

```
With Application
.WindowState = wdWindowStateNormal
.Height = 450
.Width = 600
End With
```

Application 对象的属性也可以访问处于对象层次中较低级的对象，比如 Windows 集合(代表了当前所有打开的窗口)和 Documents 集合(代表了当前所有打开的文档)。用户可以使用属性，有时候也叫做访问者，从对象层次中最高一级的 Application 对象向下访问到较低级的对象(Document 对象、Window 对象、Selection 对象等等)。可以使用以下两个示例之一来打开 MyDoc.doc。

```
Application. Documents. Open FileName :="C:\DOCS\MYDOC.DOC"
Documents. Open FileName :="C:\DOCS\MYDOC.DOC"
```

因为 Document 属性是共用的，所以 Application 属性是可选的。共用属性和方法无需 Application 对象限定符。要在“对象浏览器”中浏览共用属性和方法的列表，可以在“类别”栏中单击“<共用>”。共用项就会列在“成员”栏中。

注释 Option 对象包含了控制 Word 共用行为的一些属性。许多 Option 对象的属性对应于 Option 对话框中的各项(Tools 菜单)。使用 Application 对象的 Option 属性可以返回 Option 对象。以下的示例设置了三个应用程序范围内的选项(因为 Option 属性是共用的，所以在本示例中无需 Application 属性)。

```
With Application.Options
.AllowDragAndDrop = True
.ConfirmConversions = False
.MeasurementUnit = wdPoints
End With
```

2 运用 Document 对象

当用户在 Word 中打开或创建一个文件的同时，也创建了一个 Document 对象。用户可以使用 Document 对象或 Documents 集合的属性或方法来打开、创建、保存、激活或者关闭文件。

2.1 返回 Document 对象

用户可以使用语法 Documents (index)来返回作为一个 Document 对象的任何打开的文档，在这里 index 是该文档的名字或索引号。在以下的示例中，变量 myDoc 包含一个 Document 对象，该对象代表名为“Report.doc”的打开的文档。

Set myDoc = Documents ("Report.doc")

索引号代表了文档在 Documents 集合中的位置。在以下的示例中，变量 myDoc 包含了一个 Document 对象，该对象代表 Documents 集合中的第一个文档。

Set myDoc = Documents(1)

注释 因为当用户添加或关闭多个文档时，某个特定文档的索引号会发生改变，所以最好使用文档的名字来对 Documents 集合中的 Document 对象进行索引。

除了通过文档的名字或索引号来引用文档，还可以使用 ActiveDocument 属性来返回一个引用活动文档(当前着眼的文档)的 Document 对象。以下的示例显示产活动文档的名字；如果没有打开的文档，则该示例显示一条消息。

```
If Documents. Count >= 1 Then
MsgBox ActiveDocument. Name
Else
MsgBox "No documents are open"
End If
```

2.2 打开文档

要打开一个已经存在的文档，可以使用 Open 方法。Open 方法应用于通过 Document 属性返回的 Documents 集合。以下的示例打开了文件 Test.doc(从当前文件夹)并且启动了更改跟踪。

```
Set myDoc = Documents. Open (FileName :="TEST.DOC")
myDoc. TrackRevisions = True
```

注意到在上例中 Open 方法的返回值是一个 Document 对象，该对象表示刚被打开的文档。在此示例中的文件名不包含路径；因此，是假定了该文件在当前的文件夹中。但这会导致出现一个运行时错误，因为一旦用户创建了与当前文件夹不同的文件夹，Visual Basic 就再也找不到该文件了。但是，用户可以通过指明完整的路径，就如同下表所示的那样，来确保打开正确的文件。

操作系统	FileName 参数
Windows	FileName :="C:\Documents\Temporary File.doc"
Macintosh	FileName :="Hard Drive:Documents:Temporary File"

如果用户的宏只用于一种文件系统，那么可以在 FileName 参数中指定路径分隔符(“\”或者“:”)，就如同上表所示的那样。以下示例显示了与文件系统无关的代码，可以用来打开 Sales.doc，并且假定 Sales.doc 已经保存在 Word 的程序文件夹中。

```
programPath = Options. DefaultFilePath (wdProgramPath)
```

Documents. Open FileName :=programPath & Application. PathSeparator & "SALES.DOC"

PathSeparator 属性返回当前文件系统(例如，“\”用于 MS-DOS/Windows 的文件分配表，或者“:”用于 Macintosh)的正确的文件分隔符。DefaultFilePath 属性返回文件的位置，比如文档所在文件夹的路径、程序文件夹或者当前文件夹。

如果所指定的文件名既没有在当前文件夹(如果没有指定路径)出现也没有在指定的文件夹(如果指定了路径)中出现，那么就会发生错误。以下的示例使用 FileSearch 对象的属性和方法来判断名为“Test.doc”的文件是否存在于用户的默认文档文件夹中。如果找到了该文件(FoundFiles. Count = 1)，那么就打开它；否则，显示一条消息。

```
defaultDir = Options. DefaultFilePath (wdDocumentsPath)
With Application. FileSearch
.FileName = "Test.doc"
.LookIn = defaultDir
.Execute
If .FoundFiles. Count = 1 Then
Documents. Open FileName :=defaultDir & Application. PathSeparator & "TEST.DOC"
Else
MsgBox "Test.doc file was not found"
End If
End With
```

也可以允许用户选择要打开的文件，而不是定死 Open 方法的 FileName 参数值。如同以下示例所示的那样，可以使用带有 wdDialogFileOpen 常量的 Dialogs 属性来返回一个 Dialog 对象，该对象表示“打开”对话框(在“文件”菜单中)。Show 方法可以显示并且执行在“打开”对话框中完成的动作。

```
Dialogs (wdDialogFileOpen). Show
```

Display 方法只是用来显示特定的对话框而不作任何更多的操作。以下的示例检查 Display 方法的返回值。如果用户单击“确定”来关闭对话框，返回值 -1 并且打开所选择的文件，该文件的名字保存在变量 fSelected 中。

```
Set dlg = Dialogs (wdDialogFileOpen)
aButton = dlg. Display
fSelected = dlg. Name
If aButton = -1 Then
Documents. Open FileName :=fSelected
End If
```

要对如何显示 Word 对话框作进一步了解，请参阅“帮助”中的“显示内置 Word 对话框”一节的内容。

要判断某个特殊的文档是否打开，可以使用一条 For Each…Next 语句来对 Documents 集合进行列举。如果名为“Sample.doc”的文档已经被打开了，以下的示例就激活它；如果它尚未被打开，则该示例会打开它。

```
docFound = True
For Each aDoc In Documents
If InStr (1, aDoc. Name, "sample.doc", 1) Then
```

```
aDoc. Activate
Exit For
Else
docFound = False
End If
Next aDoc
If docFound = False Then Documents. Open _
FileName :="C:\Documents\Sample.doc"
```

可以使用 Count 属性来确定当前打开文档的数目。Count 属性应用于 Documents 集合，可以使用 Document 属性返回该集合。如果没有已打开的文档，以下的示例会显示一条消息。

```
If Documents. Count = 0 Then MsgBox "No documents are open"
```

2.3 创建和保存文档

要创建一个新文档，可以对 Documents 集合应用 Add 方法。以下的示例创建了一个新文档。

```
Documents. Add
```

Add 方法返回了仅作为一个 Document 对象而创建的文档。当用户添加一个文档时，可以设置 Add 方法的返回值是一个对象变量，以便用户可以在自己的代码中引用该新文档。以下的示例创建了一个新文档，并且设置它的上边距为 1.25 英寸。

```
Dim myDoc As Document
Set myDoc = Documents. Add
myDoc. PageSetup. TopMargin = InchesToPoints(1.25)
```

第一次保存新文档时，可以随 Document 对象使用 SaveAs 方法。以下的示例将名为“Temp.doc”的活动文档保存在当前文件夹中。

```
ActiveDocument. SaveAs FileName :="Temp.doc"
```

在文档被保存之后，用户可以使用它的文档名来调整 Document 对象。以下的示例创建了一个新文档并且立刻把它保存为“1996 Sales.doc”。然后，该示例使用新名字来作为 Documents 集合中的该文档的索引，并且向该文档添加一张表格。

```
Documents. Add.SaveAs FileName :="1996 Sales.doc"
Documents ("1996 Sales.doc").Tables.Add _
Range :=Selection. Range, NumRows :=2, NumColumns :=4
```

要保存对一个已经存在的文档所进行的更改，可以随同 Document 对象使用 Save 方法。以下的指令保存了名为“Sales.doc”的文档。

```
Documents ("Sales.doc"). Save
```

如果用户随同一个尚未保存的文档或模板使用 Save 方法，将会显示“另存为”对话框来提示用户为该文件取名。要保存所有打开的文档，可以对 Documents 集合应用 Save 方法。以下的示例保存了所有打开的文档，但没有提示用户为文件取名。

```
Documents. Save NoPrompt :=True
```

2.4 激活一个文档

要使另一个文档成为活动文档，可以对 Document 对象应用 Activate 方法。以下的示例激活了打开的文档(MyDocument.doc)。

```
Documents ("MyDocument.doc"). Activate
```

以下的示例打开了两个文档，然后激活其中的第一个文档(Sample.doc)。

```
Set Doc1 = Documents. Open (FileName :="C:\Documents\Sample.doc")
Set Doc2 = Documents.Open (FileName :="C:\Documents\Other.doc")
Doc1. Activate
```

2.5 打印一个文档

要打印一个文档，可以对 Document 对象应用 PrintOut 方法，如以下示例所示。

```
ActiveDocument. PrintOut
```

要通过程序来设置那些也可以在“打印”对话框(在“文件”菜单中)里进行设置的打印选项，可以使用 PrintOut 方法的参数来实现。用户可以使用 Options 对象的属性来设置打印选项，这些选项也可以在“选项”对话框(在“工具”菜单中)“打印”一表里进行设置。以下示例设置活动文档打印隐藏的文字，并且只打印出前三页。

```
Options. PrintHiddenText = True
ActiveDocument. PrintOut Range :=wdPrintFromTo, From :="1", To :="3"
```

2.6 关闭文档

要关闭一个文档，可以对 Document 对象应用 Close 方法。以下的示例关闭了名为“Sales.doc”的文档。

```
Documents ("Sales.doc"). Close
```

如果对文档进行过修改，Word 会显示出一条消息来询问用户是否要保存所进行的修改。用户可以随 SaveChanges 参数使用 wdDoNotSaveChanges 或 wdSaveChanges 常量来使该提示不再出现。以下的示例保存并且关闭了 Sales.doc。

```
Documents ("Sales.doc"). Close SaveChanges :=wdSaveChanges
```

要关闭所有已打开的文档，可以对 Documents 集合应用 Close 方法。以下的示例没有保存修改就关闭了所有的文档。

```
Documents. Close SaveChanges :=wdDoNotSaveChanges
```

2.7 访问文档中的对象

用户可以从 Document 对象来访问返回对象的一系列属性和方法。要查看在 Document 对象中所能使用的对象的层次关系图，可以参阅“帮助”中的“Microsoft Word 对象(Documents)”。例如，Table 属性，它能够返回一个 Table 对象的集合，可以在 Document 对象中使用。随着一个集合对象使用的 Count 属性能够判断在该集合中所包含的项数。以下的示例显示一条消息来指出在活动文档中有多少表格。

```
MsgBox ActiveDocument. Tables. Count & " table(s) in this document"
```

使用 Table(index)可以返回一个单张表格对象，这里的 index 是索引号。在以下的示例中，myTable 代表“Sales.doc”文档里的第一张表格。

Set myTable = Documents ("Sales.doc"). Tables(1)

关于返回一个特殊对象的信息可以从“帮助”中的对象主题本身 (例如，“Table 对象”)和相应的集合对象主题(例如，“Table 集合对象”)中获得。

2.8 向文档添加对象

可以使用随着能在 Document 对象中访问的集合对象的 Add 方法来向文档添加诸如脚注、备注或表格这样的对象。例如，以下命令在 myRange 变量(myRange 是一个包含了一个 Range 对象的对象变量)所指定的位置上添加了一张 3x3 的表格。

ActiveDocument. Tables. Add Range :=myRange, NumRows :=3, NumColumns :=3

以下的示例在 myRange 变量所指定的位置上添加了一个脚注。

ActiveDocument. Footnotes. Add Range :=myRange, Text :="The Willow Tree"A

要获得支持 Add 方法的集合对象列表，请参阅“帮助”中的“Add 方法”。

3 运用 Range 对象

当使用 Visual Basic 时一个常见的任务就是在文档中指定一块区域，然后对它进行处理，比如插入文字或应用格式等。例如，用户可能想编写一个宏，来对文档中某个部分里的某个单词或短语进行定位。那就可以使用一个 Range 对象来表示想在其中搜索特定单词或短语的部分文档。在确定 Range 对象后，用户能够应用该对象的方法和属性来修改相应范围中的内容。

一个 Range 对象代表了文档中的一块连续的区域。每一个 Range 对象都由一个起始字符位置和一个终止字符位置来定义。与用户在文档中使用书签的方法类似，可以在 Visual Basic 中使用 Range 对象来识别一个文档的特定部分。一个 Range 对象既可以和插入点一样小，也可以和整个文档一样大。但是，与书签不同，Range 对象仅仅在定义它的过程正在运行时才存在。

Range 对象同选定内容相互独立；也就是说，可以定义和修改范围而不会改变选定内容。用户也可以在文档中定义多个范围，而此时每个文档窗格中仅有一个选定内容。

Start、End 和 StoryType 属性唯一地确定了一个 Range 对象。Start 和 End 属性分别返回或者设置 Range 对象的起始和结束字符的位置。每个文档构成部分起始处的字符位置是 0 (zero)，而第一个字符之后的位置是 1，依此类推。StoryType 属性的 WdStoryType 常量可以表示十一种不同的文档构成部分类型。例如，如果在注脚区域中有一个 Range 对象，那么 StoryType 属性返回 wdFootnotesStory。要对文档构成部分作进一步了解，请参阅本节后面的“运用文档构成部分”一节的有关内容。

3.1 使用 Range 对象来代替 Selection 对象

宏录制器会经常创建一个使用 Selection 属性来控制操纵 Selection 对象的宏。但是，用户通常可以用一个或几个 Range 对象来以很少的命令完成相同的

任务。以下的示例是用宏录制器创建的。该宏对文档中的前两个单词进行加粗。

Selection. HomeKey Unit :=wdStory

Selection. MoveRight Unit :=wdWord, Count:=2, Extend :=wdExtend

Selection. Font.Bold = wdToggle

以下的示例没有使用 Selection 对象而完成了相同的任务。

ActiveDocument. Range (Start:=0, End :=ActiveDocument. Words(2). End) .Bold = True

以下的示例对文档中的前两个单词进行加粗，然后插入一个新的段落。

Selection. HomeKey Unit :=wdStory

Selection. MoveRight Unit :=wdWord, Count :=2, Extend :=wdExtend

Selection. Font.Bold = wdToggle

Selection. MoveRight Unit :=wdCharacter, Count:=1

Selection. TypeParagraph

以下的示例没有使用 Selection 对象就完成了上述示例中的同样任务。

Set myRange = ActiveDocument. Range(Start:=0, End :=ActiveDocument. Words(2). End)

myRange. Bold = True

myRange. InsertParagraphAfter

前面的两个示例改变了活动文档中的格式但没有改变选定内容。在大多数的场合下，Range 对象比 Selection 对象更可取，原因如下:

用户可以定义和使用多个 Range 对象，而在每个文档窗口中只能有一个 Selection 对象。

控制管理 Range 对象不会改变所选择的文字内容。

控制管理 Range 对象比运用 Selection 对象速度要快。

3.2 使用 Range 方法返回 Range 对象

可以使用 Range 方法在特定文档中创建一个 Range 对象。Range 方法(可以从 Document 对象使用)返回一个 Range 对象，该对象定位于主文档构成部分中，有给定的起始点和结束点。以下示例创建了一个 Range 对象，并且赋给 myRange 变量。

Set myRange = ActiveDocument. Range (Start :=0, End :=10)

在上述示例中，myRange 表示活动文档中的前十个字符。当对保存在 myRange 变量中的 Range 对象应用一种属性或方法时，就可以看到已经创建的 Range 对象。以下的示例对活动文档中的前十个字符进行加粗。

Set myRange = ActiveDocument. Range(Start :=0, End :=10)

myRange. Bold = True

当用户需要对一个 Range 对象进行多次引用时，可以使用 Set 语句来设置一个等价于 Range 对象的变量。但是，如果用户需要在一个对象上执行一次操作，那么就没有必要将对象保存到变量中。用户可以使用一条确定范围并且改变 Bold 属性的命令来取得同样的结果；如以下示例所示。

ActiveDocument. Range(Start :=0, End :=10). Bold = True

和书签类似，一个范围能够在文档中横跨一组字符，也可以标记其中的一个位置。在以下的示例中，Range 对象的起始和终止点相同，并且该范围内不包含任何文字。该示例在活动文档的起始处插入文字。

ActiveDocument. Range(Start :=0, End :=0). InsertBefore Text :="Hello "

用户可以通过使用如上述示例所示的字符位置数字，或者随同 Selection 对象、Bookmark 对象或 Range 对象使用 Start 属性和 End 属性，来定义一个范围的起始和终止位置。以下示例创建了一个 Range 对象，来表示活动文档中的第三和第四个句子。

```
Set myDoc = ActiveDocument
Set myRange = myDoc. Range (Start :=myDoc. Sentences(3). Start, _
End :=myDoc. Sentences (4). End)
```

技巧 Range 对象在文档中的表示不可见。但是，用户可以使用 Select 方法来选定一个 Range 对象，以保证 Range 对象表示正确的文字范围。在以下示例中的 Range 对象表示活动文档中的前三个段落。在该宏运行完毕之后，选定内容是指包含在 aRange 变量中的文字范围。

```
Set aRange = ActiveDocument.Range (Start :=0, _
End :=ActiveDocument. Paragraphs (3). Range. End)
aRange. Select
```

3.3 使用 Range 属性来返回一个 Range 对象

在许多对象都可以使用 Range 属性－例如，Paragraph 对象、Bookmark 对象、Endnote 对象以及 Cell 对象－Range 属性用来返回一个 Range 对象。以下的示例返回了一个 Range 对象，该对象表示活动文档的第一段。

```
Set myRange = ActiveDocument.Paragraphs (1). Range
```

在创建了对 Range 对象的引用之后，用户可以使用它的任何属性或方法来修改该范围。以下的示例复制了活动文档中的第一段。

```
Set myRange = ActiveDocument. Paragraphs (1). Range
myRange. Copy
```

以下的示例复制了活动文档中第一个表格的第一行。

```
ActiveDocument. Tables(1). Rows(1). Range. Copy
```

以下的示例显示了活动文档中由第一个书签标记的文字。Range 属性可以在 Bookmark 对象中使用。

```
MsgBox ActiveDocument. Bookmarks (1). Range. Text
```

如果用户需要对同一个 Range 对象应用多种属性或方法，那么可以使用 With…End With 语句。以下的示例为活动文档的第一段设定了文字的格式。

```
Set myRange = ActiveDocument. Paragraphs (1). Range
With myRange
.Bold = True
.ParagraphFormat. Alignment = wdAlignParagraphCenter
.Font. Name = "Arial"
End With
```

要获得关于返回 Range 对象的其他示例，请参阅“帮助”中的“Range 属性”。

3.4 修改文档的一部分

Visual Basic 包含了一些对象，可以使用它们来修改以下类型的文档元素：

字符、单词、句子、段落以及节。下面的表格包含了对应于这些文档元素的属性和属性所返回的对象。

表 达 式	返 回 的 对 象
Word (index)	Range 对象
Characters (index)	Range 对象
Sentences (index)	Range 对象
Paragraphs (index)	Paragraph 对象
Sections (index)	Section 对象

当用户不带 index 来使用这些属性时，就返回一个同名的集合－例如，Paragraphs 属性返回 Paragraphs 集合。但是，如果用户使用 index 来确定在某个集合中的一项，就返回上述表格第二列中的对象－例如，Words(1)返回一个 Range 对象。用户可以使用任何范围属性或方法来修改 Range 对象，如以下示例所示，该示例将选定内容中的第一个单词复制到剪贴板中。

Selection. Words (1). Copy

在 Paragraphs 集合以及 Sections 集合中的集合项分别是 Paragraph 对象和 Section 对象，而不是 Range 对象。但是，在 Paragraph 对象和 Section 对象中都可以使用 Range 属性(它返回一个 Range 对象)。以下的示例将获得文档中的第一段复制到剪贴板中。

ActiveDocument. Paragraphs(1). Range. Copy

上述表格中所有的文档元素属性都可以在 Document 对象、Selection 对象以及 Range 对象中使用，如以下三个示例所示。

本例设置活动文档中第一个单词的大小写。

ActiveDocument. Words(1). Case = wdUpperCase

本例设置选中的第一节的下边距为 0.5 英寸。

Selection.Sections(1). PageSetup. BottomMargin = InchesToPoints(0.5)

本例设置活动文档中的文字为两倍行距(Content 属性返回一个 Range 对象，该对象表示主文档构成部分)。

ActiveDocument. Content. ParagraphFormat. Space2

3.5 修改一组文档元素

要修改由一组文本元素(字符、单词、句子、段落或节)构成的范围，可以创建一个包含文档元素的 Range 对象。随同 Range 对象使用 Start 和 End 属性，用户可以新建一个 Range 对象，该对象引用了一组文档元素。以下的示例创建了一个 Range 对象(myRange)，该对象引用了活动文档中的前三个单词，然后将这些单词的字体改为 Arial。

Set Doc = ActiveDocument

Set myRange = Doc.Range (Start :=Doc. Words(1). Start, End :=Doc. Words(3). End)

myRange. Font. Name = "Arial"

以下的示例创建了一个 Range 对象，该对象起始于第二段的开头，到第四段之后结束。

Set myDoc = ActiveDocument

Set myRange = myDoc. Range (Start :=myDoc. Paragraphs(2). Range.

Start, End :=myDoc. Paragraphs(4). Range. End)

以下的示例创建了一个 Range 对象(aRange),该对象起始于第二段的开头,到第三段之后结束。ParagraphFormat 属性用来访问诸如 SpaceBefore 和 SpaceAfter 这样的设置段落格式的属性。

```
Set Doc = ActiveDocument
Set aRange = Doc. Range (Start :=Doc. Paragraphs(2). Range. Start, _
End :=Doc. Paragraphs(3). Range. End)
With aRange. ParagraphFormat
.Space1
.SpaceAfter = 6
.SpaceBefore = 6
End With
```

3.6 返回或设置范围中的文字

可以使用 Text 属性来返回或设置一个 Range 对象中的内容。以下的示例返回了活动文档中的第一个单词。

```
strText = ActiveDocument. Words(1). Text
```

以下示例将活动文档中的第一个单词改为“Hello”。

```
ActiveDocument. Words(1). Text = "Hello"
```

可以使用 InsertAfter 方法或者 InsertBefore 方法在一个范围的前面或后面插入文字。以下的示例在活动文档的第二段之前插入文字。

```
ActiveDocument. Paragraphs(2). Range. InsertBefore Text :="In the beginning "
```

在使用 InsertAfter 方法或者 InsertBefore 方法之后,范围随之扩大,包含新的文字。但是,也可以使用 Collapse 方法将范围折叠到起始位置或终止位置。以下的示例在现有文档之前插入单词“Hello”,然后将范围折叠到它的起始位置(在单词“Hello”之前)。

```
With ActiveDocument.Paragraphs(2).Range
.InsertBefore Text:="Hello "
.Collapse Direction:=wdCollapseStart
End With
```

设置范围中文字的格式

可以使用 Font 属性来取得设置字符格式的属性和方法,而使用 ParagraphFormat 属性来取得设置段落格式的属性和方法。以下的示例设置了活动文档中第一段的字符格式和段落格式。

```
With ActiveDocument. Paragraphs(1). Range. Font
.Name = "Times New Roman"
.Size = 14
.AllCaps = True
End With
With ActiveDocument. Paragraphs(1). Range. ParagraphFormat
.LeftIndent = InchesToPoints(0.5)
.Space1
End With
```

3.7 重新定义 Range 对象

可以使用 SetRange 方法来重新定义一个已经存在的 Range 对象。以下的示例定义 myRange 为当前的选定内容。SetRange 方法重新定义 myRange，使它表示当前的选定内容加上随后的十个字符。

```
Set myRange = Selection. Range
myRange. SetRange Start :=myRange. Start, End :=myRange. End + 10
```

要获得重新定义 Range 对象的其他信息和示例，请参阅“帮助”中“SetRange 方法”。

用户也可以通过改变 Start 属性和 End 属性的值，或者使用 MoveStart 方法或 MoveEnd 方法来重新定义一个 Range 对象。以下的示例重新定义 myRange 对象，使它表示当前的选定内容加上随后的十个字符。

```
Set myRange = Selection. Range
myRange. End = myRange. End + 10
```

以下的示例使用 MoveEnd 方法扩展了 myRange，使它包含下一段。

```
Set myRange = ActiveDocument. Paragraphs(2)
myRange. MoveEnd Unit :=wdParagraph, Count :=1
```

3.8 在范围内的段落中循环

可以通过几种不同的方法在范围内的段落中进行循环。本节包含使用 For Each...Next 语句和 Next 属性及方法在范围内的段落中进行循环的内容。也可以使用同样的技术在范围内的字符、单词或句子中进行循环。

使用 For Each...Next 语句

建议在范围内进行段落循环时使用 For Each...Next 语句，另外建议在集合内进行循环时也同样使用该语句。以下的示例在当前文档的前五段中进行循环，在每段之前添加文字。

```
Set myDoc = ActiveDocument
Set myRange = myDoc.Range (Start:=myDoc. Paragraphs(1). Range. Start, End :=myDoc. Paragraphs(5). Range. End)
For Each para In myRange. Paragraphs
para.Range. InsertBefore "Question:" & vbTab
Next para
```

假定用户想修改上述的代码，在范围内对用户选定的段落进行循环。可以使用 Selection 属性来表示选定内容中的各个段落。以下的示例在选定内容的段落中进行循环，去除加粗格式。

```
For Each para In Selection. Paragraphs
para.Range. Bold = False
Next para
```

3.9 使用 Next 属性或方法

用户也可以使用 Next 属性和方法在范围的段落中进行循环。以下的示例说明了怎样在范围的单词中进行循环，将每个单词增大一点字号。

```
Set myRange = ActiveDocument.Words(1)
```

```
For i = 1 To 5
myRange. Font. Size = myRange. Font. Size + i
Set myRange = myRange. Next(Unit :=wdWord, Count :=1)
Next i
```

以下的示例在范围的段落中进行循环，将范围的对齐方式由居中改为左对齐。该示例也使用 Next 属性重新定义了 myRange，使它表示下一段。

```
Set myRange = ActiveDocument. Paragraphs(1). Range
For i = 1 To 5
If myRange. Paragraphs(1). Alignment = wdAlignParagraphCenter Then
myRange. Paragraphs(1). Alignment = wdAlignParagraphLeft
End If
Set myRange = myRange. Paragraphs(1). Next. Range
Next i
```

3.10将 Range 对象赋值给变量

可以通过几种途径来把一个已经存在的 Range 对象赋给一个变量。在以下示例中，变量 Range1 和变量 Range2 都表示 Range 对象。在该示例中命令把活动文档中第一和第二个单词分别赋值给变量 Range1 和 Range2。

```
Set Range1 = ActiveDocument. Words(1)
Set Range2 = ActiveDocument. Words(2)
```

设置一个 Range 对象变量等价于另一个 Range 对象变量

以下的示例创建变量 Range2，并且与 Range1 相同。

```
Set Range2 = Range1
```

现在就有两个变量表示着同样的范围。当用户调整 Range2 的起始位置、终止位置或者文字时，所做的更改也同样会影响到 Range1，反之亦然。

以下的示例将 Range1 默认属性 (Text 属性)的值赋给 Range2 的默认属性。在此示例中的代码等价于 Range2. Text = Range1. Text，它没有改变 Range1 对象实际所表示的内容，它仅仅改变 Range2 的 contents (text)。

```
Range2 = Range1
```

这两个范围(Range2 和 Range1)包含有相同的内容，但是它们可能指向文档中的不同位置，或者干脆是不同的文档。

3.11使用 Duplicate 属性

以下的示例创建了一个新复制的 Range 对象——Range2，该对象有着和 Range1 一样的起始位置、终止位置以及文字内容。

```
Set Range2 = Range1. Duplicate
```

如果改变了 Range1 的起始位置或是终止位置，这种改变并不会影响到 Range2，反之亦然。但是，因为这两个范围指向文档的相同位置，改变一个范围中的文字内容也会同时改变另一个范围的文字内容。

3.12运用文档构成部分

一个文档构成部分是文档中的一个区域，该区域中的文字区别于文档中的其他区域。例如，如果一个文档包含了正文文字、脚注和页眉，则该文档就包含了

文档正文部分、脚注部分和页眉部分。

可以使用 StoryType 属性来返回指定范围、选定内容或书签的文档构成部分。如果在脚注部分中包含了选定内容，那么下例将关闭活动窗口中的脚注窗格。

```
ActiveWindow. View. Type = wdNormalView
If Selection.StoryType = wdFootnotesStory Then _
ActiveWindow.ActivePane.Close
```

StoryRanges 集合包含了一个文档中每种有效的文档构成部分类型的第一个过程部分范围。可以使用 NextStoryRange 方法来返回以后的文档构成部分。以下的示例搜索活动文档中的每个文档过程部分来找出文字“Microsoft Word”。该示例也将它每次找到的文字全部设置为斜体。

```
For Each myStoryRange In ActiveDocument. StoryRanges
myStoryRange. Find. Execute FindText :="Microsoft Word", Forward :=True
While myStoryRange. Find. Found
myStoryRange. Italic = True
myStoryRange. Find.Execute FindText :="Microsoft Word", _
Forward :=True, Format :=True
Wend
While Not (myStoryRange. NextStoryRange Is Nothing)
Set myStoryRange = myStoryRange. NextStoryRange
myStoryRange. Find.Execute FindText :="Microsoft Word", Forward :=True
While myStoryRange. Find. Found
myStoryRange. Italic = True
myStoryRange. Find.Execute FindText :="Microsoft Word", _
Forward :=True, Format :=True
Wend
Wend
Next myStoryRange
```

4 运用 Selection 对象

当用户使用 Word 中的某个文档时，通常会先选择文字，然后再执行一项操作，比如设定以有文字的格式，或是键入新文字。在 Visual Basic 中，通常没有必要在修改文字之前先选定该文字；而是创建一个表示文档特定部分的 Range 对象并对其进行操作。但是，当用户想要自己的代码对选定内容做出响应或是改变选定内容，就可以通过 Selection 对象来完成任务。

可以使用 Selection 属性来返回 Selection 对象。在文档窗口的每个窗格中只能有一个 Selection 对象，在任何时刻，也只能有一个活动的 Selection 对象。相当于 Excel 中的 ActiveCell 对象。选定内容可以包含文档的一块区域，也可以被折叠到一个插入点。以下的示例改变了选定内容中各个段的段落格式。

```
Selection. Paragraphs. SpaceBefore = InchesToPoints (0.25)
```

Selection 属性可以在 Application 对象、Window 对象和 Pane 对象中使用。如果随同 Application 对象使用 Selection 属性，则该 Selection 对象表示活动的选定内容。

Selection. InsertAfter Text :="Next Text"

也可以随同一个 Window 对象或 Pane 对象使用 Selection 属性来返回一个在特定窗口或窗口窗格中 Selection 对象。以下的示例随同 Window 对象使用 Selection 属性，在文档窗口“Document2”中插入文字。

Windows ("Document2"). Selection.InsertAfter Text :="New Text"

以下的示例随同 Pane 对象使用 Selection 属性，在主页眉窗格中插入文字。

With ActiveWindow
.View. Type = wdPageView
.View. SeekView = wdSeekPrimaryHeader
.ActivePane. Selection. InsertAfter Text :="Header"
End With

在使用 InsertAfter 方法或 InsertBefore 方法之后，选定内容随之扩大，包括了新的文字。但是，用户也可以使用 Collapse 方法将选定内容折叠到它的起始或终止位置。以下的示例在选定内容的文字之后插入单词“Hello”，任何将选定内容折叠到单词“Hello”之后的插入点。

Selection. InsertAfter Text :="Hello"
Selection. Collapse Direction :=wdCollapseEnd

4.1 移动和扩展选定内容

用户可以通过几种方法来移动或扩展由 Selection 对象所表示的选定内容(例如，Move 和 MoveEnd)。以下的示例将选定内容移动到下一段的开头。

Selection. MoveDown Unit :=wdParagraph, Count :=1, Extend :=wdMove

用户也可以通过改变 Selection 对象的 Start 属性及 End 属性的值，或者通过 MoveStart 和 MoveEnd 方法来移动或扩展选定内容。以下的示例通过把终止位置移动到段尾来对选定内容进行扩展。

Selection. MoveEnd Unit :=wdParagraph, Count :=1

因为在一个文档窗口或窗格中只能有一个选定内容，用户也可以通过选定另一个对象来移动选定内容。可以使用 Select 方法，选择文档的一项。在使用 Select 方法之后，就可以使用 Selection 属性返回一个 Selection 对象。以下的示例选定了活动文档中的第一个单词，然后把该单词变为“Hello”。

ActiveDocument. Words(1). Select
Selection. Text = "Hello "

用户也可以通过 GoToNext 方法、GoToPrevious 方法或 GoTo 方法来移动选定内容。以下的示例将选定内容移动到文档中的第四行。

Selection. GoTo What :=wdGoToLine, Which :=wdGoToAbsolute, Count :=4

以下的示例将选定内容恰好移动到活动文档中的下一个域之前。

Selection. GoToNext What :=wdGoToField

4.2 在 Selection 对象使用的对象

可以在 Range 对象和 Document 对象中使用的许多其他对象也能在 Selection 对象中使用，这使得用户可以在一个选定内容中控制操纵对象。要获得能在 Selection 对象使用的对象的完整列表，请参阅“帮助”中的“Microsoft

Word 对象(Selection)”，或者 “Selection 对象”。

以下的示例更新了选定内容中各个域的结果。

```
If Selection. Fields. Count >= 1 Then Selection. Fields. Update
```

以下的示例将一节中的各个段落缩进了 0.5 英寸。

```
Selection. Paragraphs. LeftIndent = InchesToPoints (0.5)
```

用户可以使用 For Each...Next 语句在选定内容中的单个对象上进行循环操作，来代替对选定内容中的所有对象逐个进行控制操纵。以下的示例在选定内容的每个段落上进行循环操作，将找到的任何居中的段落变为左对齐。

```
For Each para In Selection. Paragraphs
If para. Alignment = wdAlignParagraphCenter Then para. Alignment = _
wdAlignParagraphLeft
Next para
```

以下的示例显示出选定内容中每个书签的名字。

```
For Each aBook In Selection. Bookmarks
MsgBox aBook. Name
Next aBook
```

4.3 Selection 对象的属性和方法

本节突出介绍 Selection 对象的一些常用属性和方法。

4.3.1 返回或设置选定内容中的文字

可以使用 Text 属性来返回或设置一个 Selection 对象的内容。以下的示例返回所选的文字。

```
strText = Selection. Text
```

以下的示例将所选文字改为 “Hello World”。

```
Selection. Text = "Hello World"
```

可以 InsertBefore 方法或 InsertAfter 方法，在选定内容之前或之后插入文字。以下的示例在选定内容之前插入了文字。

```
Selection. InsertBefore Text :="And furthermore "
```

4.3.2 为许多文字设定格式

可以使用 Font 属性来访问设置字符格式的属性和方法，也可以使用 ParagraphFormat 属性来访问设置段落格式的属性和方法。以下的示例设置了选定内容的字符和段落格式。

```
With Selection. Font
.Name = "Times New Roman"
.Size = 14
End With
Selection. ParagraphFormat. LeftIndent = InchesToPoints (0.5)
```

4.3.3 返回一个 Range 对象

如果一种方法或属性是在 Range 对象而不是 Selection 对象中使用的(例如，CheckSpelling 方法)，那么可以使用 Range 属性来从 Selection 对象返回一个 Range 对象。以下的示例对所选单词进行拼写检查。

Selection. Range. CheckSpelling

4.3.4 返回关于选定内容的信息

可以使用 Information 属性来返回关于选定内容的信息。例如，用户可以判断当前页的页码、文档的总页数，或是判断选定内容是否在一个页眉或注脚中。Information 属性可取三十五种不同的常量(wdActiveEndPageNumber, wdNumberOfPagesInDocument 和 wdInHeaderFooter 等等)，用户可以使用它们来返回关于选定内容的不同信息。举个例子，如果选定内容处于一张表格中，那么以下的示例将显示表格中的行号或列号。

```
If Selection.Information(wdWithInTable) = True Then
MsgBox "Columns = " _
& Selection.Information(wdMaximumNumberOfColumns) _
& vbCr & "Rows = " _
& Selection.Information(wdMaximumNumberOfRows)
End If
```

要获得可以随 Information 属性一起使用的常量的完整列表和说明，请参阅“帮助”中的“Information 属性”。

4.3.5 判断文字是否被选定

可以使用 Type 属性来设置或返回选定内容在文档中被指定的方式。例如，可以使用 wdSelectionBlock 常量来判断一个文字块是否被选定。如果选定内容是一个插入点，那么以下的示例选定包含该插入点的段落。

```
If Selection. Type = wdSelectionIP Then
Selection. Paragraphs(1). Range. Select
End If
```

5 运用 Find 和 Replacement 对象

可以使用 Find 和 Replacement 对象来查找并且替换文档中文字的特定范围。Find 对象可以在 Selection 对象或 Range 对象中使用(根据 Find 对象是从 Selection 对象还是从 Range 对象返回的，查找操作也略有不同)。

5.1 使用 Selection.Find

如果是在 Selection 对象中使用 Find 对象，那么当找到符合选择条件的文本后选定内容将会改变。以下的示例选定下一次出现的单词“Hello”。如果在找到单词“Hello”之前已经到达了文档的末尾，那么停止搜索。

```
With Selection. Find
.Forward = True
.Wrap = wdFindStop
.Text = "Hello"
.Execute
End With
```

Find 对象包含与“查找”和“替换”对话框(在“编辑”菜单中)里的选项有关的属性。用户既可对 Find 对象的属性逐一进行设置，也可以随同 Execute 方

法使用参数来进行设置，如果以下示例所示。

Selection. Find. Execute FindText :="Hello", Forward :=True, Wrap :=wdFindStop

5.2 使用 Range.Find

如果是在 Range 对象中使用 Find 对象，选定内容不会改变，但是当找到符合选择条件的文本时范围会被重新定义。以下的示例确定活动文档中第一次出现的单词“blue”的位置。如果查找操作成功，将重新定义范围并且设置单词“blue”的格式为粗体。

```
With ActiveDocument. Content. Find
.Text = "blue"
.Forward = True
.Execute
If .Found = True Then .Parent.Bold = True
End With
```

以下的示例执行了和上述示例相同的操作，只是使用了 Execute 方法的参数。

```
Set myRange = ActiveDocument. Content
myRange. Find. Execute FindText :="blue", Forward :=True
If myRange. Find. Found = True Then myRange. Bold = True
```

5.3 使用 Replacement 对象

Replacement 对象表示查找-替换操作的替换条件。Replacement 对象的属性和方法对应于“查找”和“替换”对话框(在“编辑”菜单中)里的选项。

可以在 Find 对象中使用 Replacement 对象。以下的示例将所有出现单词“hi”的地方替换为“hello”。当找到符合选择条件的文本时，选定内容将会改变，因为代码从 Selection 对象返回 Find 对象。

```
With Selection.Find
.ClearFormatting
.Text = "hi"
.Replacement.ClearFormatting
.Replacement.Text = "hello"
.Execute Replace:=wdReplaceAll, Forward:=True, _
Wrap:=wdFindContinue
End With
```

以下的示例去除了活动文档中所有的粗体格式。Bold 属性对 Find 对象为 True(真)，对 Replacement 属性为 False(假)。要查找和替换格式，需设置查找和替换文本为空字符串(“”)，并且设置 Execute 方法的 Format 参数为 True(真)。选定内容保持不变，因为代码在 Range 对象中返回 Find 对象(Content 属性返回一个 Range 对象)。

```
With ActiveDocument.Content.Find
.ClearFormatting
.Font.Bold = True
With .Replacement
```

```
.ClearFormatting
.Font.Bold = False
End With
.Execute FindText:="", ReplaceWith:="", Format:=True, _
Replace:=wdReplaceAll
End With
```

6 运用 Table、Column、Row 和 Cell 对象

Word 对象模型包含了表格对象，也包含了表格中各种不同元素的对象。可以随同 Document 对象、Range 对象、Selection 对象使用 Table 属性来返回 Table 集合。Table(index)返回了一个单独的 Table 对象，在这里 index 是表格的索引号。索引号代表在选定内容、范围或文档中表格的位置。以下的示例将选定内容中的第一个表格转换为文本。

```
If Selection. Tables. Count >= 1 Then
Selection. Tables(1). ConvertToText Separator :=wdSeparateByTabs
End If
```

可以随同 Column 对象、 Range 对象、Row 对象或 Selection 对象使用 Cells 属性来返回 Cells 集合。用户可以通过使用 Table 对象的 Cell 方法或是索引 Cells 集合来获得一个 Cell 对象。以下的两条语句都能够设置 myCell 为一个 Cell 对象，该 Cell 对象代表活动文档中表格一的第一个单元格。

```
Set myCell = ActiveDocument. Tables(1). Cell(Row :=1, Column :=1)
Set myCell = ActiveDocument. Tables(1). Columns(1). Cells(1)
```

注释 要在一个表格的一个单元格中插入文字，可以使用 Text 属性、InsertAfter 方法或者随 Range 对象使用 InsertBefore 方法。可以随 Cell 对象使用 Range 属性来返回一个 Range 对象。以下的示例在当前文档第一个表格的每一个单元格中插入连续的单元格序号。

```
i = 1
For Each c In ActiveDocument. Tables(1). Range. Cells
c. Range. InsertBefore Text :="Cell " & i
i = i + 1
Next c
```

可以随同 Table 对象、Range 对象或 Selection 对象使用 Column 属性来返回 Columns 集合。Columns(index)返回了一个单独的 Column 对象，在这里 index 是索引号。以下的示例选择了当前文档第一个表格的第一列。

```
ActiveDocument. Tables(1). Columns(1). Select
```

可以随同 Table 对象、Range 对象或 Selection 对象使用 Row 属性来返回 Rows 集合。Rows(index)返回了一个单独的 Row 对象，在这里 index 是索引号。以下的示例给当前文档第一个表格的第一行加底纹。

```
ActiveDocument. Tables(1). Rows(1). Shading. Texture = wdTexture10Percent
```

6.1 修改图形表格的行与列

当用户试图使用在一个图形表格(或者任何表格，其中有两个以上相邻的单

元格被合并，但是行与列没有统一)中某一单独的行或列时，就可能出现一个运行时错误。如果活动文档中的第一个表格的每列含有不一致的行号，以下的示例就会出错。

```
ActiveDocument. Tables(1). Rows(1). Borders.Enable = False
```

用户可以首先使用 SelectColumn 或 SelectRow 方法来选定某个特定行或列的单元格，以此避免这样的错误。当用户选好了行或列，再随 Selection 对象使用 Cells 属性。以下的示例选定了活动文档中表格一的第一行。该示例使用 Cells 属性返回所选的单元格(在第一行中所有单元格)，以便可以删除边框。

```
If ActiveDocument. Tables(1). Uniform = True Then
ActiveDocument. Tables(1). Cell(1, 1). Select
With Selection
.SelectRow
.Cells. Borders. Enable = False
End With
End If
```

以下的示例选定了表格一的第一列。该示例使用了一个 For Each...Next 循环来向选定内容(第一列的所有单元格)中的每个单元格添加文字。

```
If ActiveDocument. Tables(1). Uniform = True Then
ActiveDocument. Tables(1). Cell(1, 1). Select
Selection. SelectColumn
i = 1
For Each oCell In Selection. Cells
oCell.Range. Text = "Cell " & i
i = i + 1
Next oCell
End If
```

7 运用其他普通对象

本节提供了关于运用一些普通 Word 对象的信息和技巧。

7.1 使用 HeaderFooter 对象

HeaderFooter 对象既可以表示一个页眉也可以表示一个注脚。HeaderFooter 对象是 HeaderFooter 集合的一个成员，它可以在 Section 对象中使用。Headers(index)或 Footer(index)属性返回了一个单独的 HeaderFooter 对象，在这里 index 是 WdHeaderFooterIndex 常量的一个值。

以下的示例创建了一个 Range 对象(aRange)，该对象表示活动文档第一节的主要注脚。在该示例设置 Range 对象之后，它删除了已有的注脚文本。它还向注脚添加了 AUTHOR 域以及两个表和 FILENAME 域。

```
Set oRange = ActiveDocument. Sections(1). Footers
(wdHeaderFooterPrimary). Range
With oRange
.Delete
.Fields. Add Range :=oRange, Type :=wdFieldFileName, Text :="\p"
```

```
.InsertAfter Text :=vbTab
.InsertAfter Text :=vbTab
.Collapse Direction :=wdCollapseStart
.Fields.Add Range :=oRange, Type :=wdFieldAuthor
End With
```

注释 PageNumbers 集合仅仅能在 HeaderFooter 对象中使用。可以对 PageNumbers 集合应用 Add 方法来向一个页眉或注脚添加页码。

7.2 使用 Styles 集合

Styles 集合可以在 Document 对象中使用。以下的示例改变了活动文档中“标题 1”样式的格式。

```
ActiveDocument. Styles (wdStyleHeading1). Font. Name = "Arial"
```

Styles 集合不能在 Template 对象中使用。如果想修改一个模板中的样式，可以使用 OpenAsDocument 方法按文档来打开模板，这样用户就可以对样式进行修改。以下的示例改变了活动文档所附的模板中“标题 1”样式的格式。

```
Set aDoc = ActiveDocument. AttachedTemplate. OpenAsDocument
With aDoc
.Styles(wdStyleHeading1). Font. Name = "Arial"
.Close SaveChanges :=wdSaveChanges
End With
```

7.3 指定 CommandBars

在使用 CommandBars 集合(表示菜单栏和工具栏)之前，可以使用 CustomizationContext 属性来设置保存着对菜单栏和工具栏的更改的 Template 或 Document 对象。以下的示例给“格式”工具栏添加了“双下划线”命令。因为自定义更改保存在 Normal 模板中，所有的文档都会受到影响。

```
CustomizationContext = NormalTemplate
CommandBars ("Formatting"). Controls. Add Type :=msoControlButton, _
ID :=60, Before :=7
```

要了解关于菜单栏和工具栏的更改范围的更多信息，请参阅第八章，“菜单栏和工具栏”。

7.4 使用 Dialogs 集合

可以使用 Dialogs 属性返回 Dialogs 集合，该集合表示内置 Word 对话框(例如，“打开”和“保存”对话框等)。用户不能新建内置对话框，也不能向 Dialogs 集合添加内置对话框。要了解关于创建用 ActiveX 控件创建自定义对话框的内容，请参阅第十二章，“ActiveX 控件和对话框”。

7.5 返回 MailMerge 和 MailMerge 对象

可以使用 Document 对象的 MailMerge 属性来返回一个 MailMerge 对象。无论所指定的文档是否是一个邮件合并文档，都可以使用 MailMerge 属性。在用户使用 Execute 方法执行合并之前，可以使用 State 属性来确定邮件合并操作的状态。如果活动文档是附加了数据源的主文档，那么以下的示例将执行邮件合

并。

```
Set myMerge = ActiveDocument.MailMerge
If myMerge. State = wdMainAndDataSource Then myMerge. Execute
```

使用 Document 对象的 Envelope 属性可以返回一个 Envelope 对象。无论用户是否向所指定的文档添加了一个信封，都可以使用 Envelope 对象。但是，如果用户使用以下属性而没有向文档添加信封，那么会出现错误：Address 属性，AddressFromLeft 属性，AddressFromTop 属性，FeedSource 属性，ReturnAddress 属性，ReturnAddressFromLeft 属性，ReturnAddressFromTop 属性或 UpdateDocument 属性。

以下的示例使用 On Error GoTo 语句来捕获用户没有向活动文档添加信封的错误。但是，如果用户已经向文档添加了信封，将显示收件人地址。

```
On Error GoTo ErrorHandler
MsgBox ActiveDocument. Envelope. Address
ErrorHandler:
If Err = 5852 Then MsgBox "Envelope is not in the specified document"
```

7.6 在文档中添加和编辑域

用户可以通过对 Field 集合应用 Add 方法来给文档添加域。以下的示例添加了一个 DATE 域来替换选定内容。

```
ActiveDocument. Fields. Add Range:=Selection. Range, _
Type :=wdFieldDate
```

在添加一个域之后，用户可以通过使用 Result 或 Code 属性来返回或设置域结果和域代码，这两种属性都可以返回一个 Range 对象。以下的示例更改选定内容的第一个域代码，更新该域，然后显示域结果。

```
If Selection. Fields. Count >= 1 Then
With Selection. Fields (1)
.Code. Text = "CREATEDATE \*MERGEFORMAT"
.Update
MsgBox . Result. Text
End With
End If
```

7.7 InlineShape 对象同 Shape 对象的对比

一个 Shape 对象代表图形层的一个对象，诸如自选图形、任意多边形、OLE 对象、ActiveX 控件、图片等。Shape 对象锁定于文本范围内，但是能够任意移动，使用户可以将它们定位于页面的任何位置。要了解运用 Shape 对象的信息，请参阅第十章，“形状和图形层”,也可以参阅“帮助”中的“Shape 对象”。

一个 InlineShape 对象代表文档文字层的一个对象。一个嵌入式形状可能是图片、OLE 对象或 ActiveX 控件。可以将 InlineShape 对象视为字符，并将其象字符一样放在一行文本中。要了解关于 InlineShape 对象的信息，请参阅“帮助”中的“InlineShape 集合对象”或“InlineShape 对象”。

7.8 在 Word 窗体中使用 FormField 对象

用户可以创建一个 Word 在线窗体，该窗体包含复选框、文本文本框和下拉式列表框。使用“窗体”工具栏可以插入这些窗体元素。对应的 Visual Basic 对象是 CheckBox, TextInput, 和 DropDown。所有这些对象都可以在 FormFields 集合的任何一个 FormField 对象中使用；但是，用户应当使用按照窗体域的类型使用相应的对象。例如，以下命令从活动文档的“Check1”域中选择复选框。

ActiveDocument. FormFields ("Check1").CheckBox. Value = True

除了“窗体”工具栏中的窗体元素之外，用户还可以向一个在线窗体添加 ActiveX 控件。使用“控件工具箱”可以插入 ActiveX 控件。用户可以把控件插入文字层或图形层；控件将分别由一个 InlineShape 对象或一个 Shape 对象来表示。要了解关于运用 ActiveX 控件的更多内容，请参阅第十二章，“ActiveX 控件和对话框”。

8 判断对象是否有效

可以通过在用户代码中包含判断语句来判断由一个表达式返回的特定对象或者一个变量所引用的对象是否有效，以此避免代码在运行时的错误。本节讨论了一些检查表达式返回值或变量保存值有效性的技术。

用户可以随同一个变量或表达式使用 TypeName 函数来确定对象的类型。如果 Selection. NextField 返回了一个 Field 对象，那么以下的示例将在状态栏中显示一条消息。

If TypeName (Selection. NextField) = "Field" Then StatusBar = "A field was found"

以下示例的功能等价于上述示例；不同之处仅在于它使用一个对象变量(myField)来保存 NextField 方法的返回值。

Set myField = Selection. NextField

If TypeName (myField) = "Field" Then StatusBar = "A field was found"

如果指定的变量或表达式不代表一个对象，它对 Nothing 求值。如果 NextField 方法没有返回 Nothing(也就是说，如果 NextField 方法返回了一个 Field 对象，它只是其他的可能返回值)，那么以下的示例将对 myField 应用 Update 方法。

Set myField = Selection. NextField

If Not (myField Is Nothing) Then myField. Update

Word 包含共用的 IsObjectValid 属性。可以使用该属性来判断一个被某个变量所引用的对象是否有效。如果变量所引用的对象已经被删除了，那么该属性返回 False。以下的示例向活动文档添加一个表格，并且把它赋值给变量 aTable。该示例从文档中删除了第一个表格。如果 aTable 引用的表格不是文档中的第一个表格(也就是说，如果 aTable 仍然是一个有效的对象)，那么该示例删除表格的边框。

Set aTable = ActiveDocument. Tables. Add(Range :=Selection. Range, NumRows :=2, NumColumns:=3)

ActiveDocument. Tables(1).Delete

If IsObjectValid (aTable) = True Then aTable. Borders. Enable = False

9 修改 Word 命令

用户可以通过把命令转变为宏来对绝大多数 Word 命令进行修改。例如，可以“文件”菜单中的“打开”命令，使 Word 显示当前文件夹中的每一个文件，而不是只显示 Word 文档文件列表(在 Windows 中，控制名为.doc 的文件)。

要在“宏”对话框(在“工具”菜单中)里显示内置 Word 命令的列表，可以在该对话框中“宏的位置”一栏里选中“Word 命令”。在该对话框中就会列出每一个可用的菜单、工具栏或快捷键命令。每个菜单命令用与该命令关联的菜单名开头。例如，在“文件”菜单中的“保存”命令显示为“FileSave”。

用户可以通过给宏取和 Word 命令相同的名字来用一个宏替换一条 Word 命令。例如，如果用户创建了一个名为“FileSave”的宏，当用户：在“文件”菜单中单击“保存”、在“常用”工具栏中单击“保存”按钮或按下给 FileSave 指定的快捷键时，Word 将运行该宏。

9.1 修改 Word 命令

在“工具”菜单中，用鼠标指向“宏”，然后单击“宏的位置”。

在“宏的位置”一栏中，单击“Word 命令”。

在“宏名”一栏中，选中要修改的 Word 命令(例如，FileSave)。

在“宏的位置”一栏中，选择要保存宏的模板或文档的位置。例如，单击 Normal.dot(共用模板)来创建一个共用宏(FileSave 命令将会自动地对所有文档进行修改)。

单击“创建”。

Visual Basic 编辑器被打开，其中显示出一个模块，该模块包含有一个与刚才被选中的命令同名的新过程。如果选中了 FileSave 命令，那么就出现 FileSave 宏，如以下示例所示。

```
Sub FileSave()
FileSave Macro
Saves the active document or template
保存活动文档或模板
ActiveDocument.Save
End Sub
```

用户可以添加附加的命令或删除已有的 ActiveDocument. Save 命令。FileSave 命令每次运行时，FileSave 宏都将 Word 命令来执行。要恢复原来的 FileSave 命令，需要更改 FileSave 宏的名字或删除它。

注释 用户还可以通过创建与 Word 命令(例如，FileSave)同名的代码模块来用一个名为“Main”的子例程替换一条 Word 命令。

10 运用事件

一个事件既可以是一个被对象识别的操作(比如，打开一个文档或者从应用程序退出)，也可以是一个能够编写代码来响应的操作。一个用户的操作或一段程序代码都可以导致事件的发生，事件也可以由系统引起。Word 所支持的事件均列在下表中，该表还列出了 ActiveX 控件事件，它们将在第十二章“ActiveX 控件和对话框”中进行讨论。

要了解关于运用 Word 事件的信息，请参阅下列“帮助”主题：“使用 Document 对象的事件”、“使用 ActiveX 控件的事件”以及“使用 Application 对象的事件”。

10.1Document 事件

当用户打开或关闭一个已经存在的文档时，或者用户在新建一个文档时，就发生了文档事件，如以下示例所示。

事件描述

当关闭一个文档时发生 Close 事件。

当创建一个基于模板的新文档时发生 New 事件。

当打开一个文档时发生 Open 事件。

一个文档事件过程的范围取决于它保存的位置。如果在一个文档中保存 Open 或 Close 事件过程，那么该过程仅当用户关闭或打开该文档时才发生；如果在一个模板中保存 Open 或 Close 事件，那么当打开或关闭基于该模板的文档或该模板本身时，该过程才发生。一个新事件过程必须保存在模板中；一个保存在文档中新事件过程决不会运行，因为文档只能基于模板来创建。

以下的示例在打开文档时将 Word 应用程序的窗口扩至最大状态。

```
Private Sub Document_Open()
Application. WindowState = wdWindowStateMaximize
End Sub
```

10.2ActiveX 控件事件

Word 为 Word 文档中的 ActiveX 控件执行 LostFocus 和 GotFocus 事件。

事件描述

在焦点移开嵌入式 ActiveX 控制时发生 LostFocus 事件。

当焦点移至内嵌式 ActiveX 控制时发生 GotFocus 事件。

以下的示例保持 CommandButton1 处于禁用状态，直到用户在 TextBox1 中键入一个值。

```
Private Sub TextBox1_LostFocus()
If TextBox1. Value = "" Then
CommandButton1. Enabled = False
Else
CommandButton1. Enabled = True
End If
End Sub
```

其他 ActiveX 控件事件的文档在“Microsoft 窗体帮助”中。有关在自定义对话框和文档中使用 ActiveX 控件的内容，请参阅第十二章“ActiveX 控件和对话框”。

10.3Application 事件

当用户退出应用程序或焦点移至另一个文档时发生 Application 事件。但是，与 Document 和 ActiveX 控件事件不同，Application 事件默认事件处于禁用状态。在用户随同 Application 对象使用事件之前，必须新建一个类模块并声明一个包

含事件的 Application 类型对象。用户可以在 Visual Basic 编辑器中使用类模块命令(插入菜单)的形式来新建一个类模块。

启用 Application 对象的事件，最好向类模块中添加以下声明。

Public WithEvents App As Application

定义了包含事件的新对象后，它将出现在类模块的“对象”下拉列表框中，然后可为新对象编写事件过程。(在“对象”框中选定新对象后，其有效事件将出现在“过程”下拉列表框中。)

但是在过程运行之前，必须将类模块中的已声名对象和 Application 对象相连接。要完成上述任务，可以在任何模块中使用下列声明(在这里“EventClass”是用户创建的用来启用事件的类模块的名字)。

Public X As New EventClass

当用户已经创建了 X 对象变量后(EventClass 类的一个实例)，可以设置 EventClass 类的 App 对象于 Word Application 对象等价。

```
Sub InitializeApp()
Set X. App = Application
End Sub
```

在用户运行 InitializeApp 过程之后，EventClass 类模块中的 App 对象指向了 Word Application 对象，而且只要事件发生，类模块中的事件过程也将会运行。

在用户禁用 Application 对象的事件之后，可以为下表中的事件创建事件过程。

10.4事件描述

当新建一个文档、打开一个已有文档或使另一个文档成为活动文档时 DocumentChange 事件发生。

当用户退出 Word 时 Quit 事件发生。

以下的实例确保了在用户退出 Word 之前，“常用”工具栏和“格式”工具栏均为可见。结果，当重新启动 Word 时，这些工具栏将不会出现。

```
Private Sub App_Quit()
CommandBars ("Standard"). Visible = True
CommandBars ("Formatting"). Visible = True
End Sub
```

11 使用自动宏

通过给一个宏赋予一个特殊的名字，用户可以在执行诸如启动 Word 或打开一个文档这样的操作时自动运行它。Word 将下列名称识别为自动宏，或者“自动”宏。

宏 名	运 行 时 刻
AutoExec	每次启动 Word 或装载一个全局模板时
AutoNew	每次新建一个文档时
AutoOpen	每次打开一个已有文档时
AutoClose	每次关闭一个文档时
AutoExit	每次退出 Word 或卸载一个全局模板时

关于使用自动宏的详细信息，请参阅“帮助”中的“自动宏”。

12 使用自动化

在运用Word数据之外,用户还可能想让自己的应用程序同其他的应用程序,比如 Microsoft Excel, Microsoft PowerPoint, 或 Microsoft Access 交换数据。用户可以通过使用自动化(以前的 OLE 自动化)来同其他应用程序进行通信。

12.1从另一个应用程序中对 Word 进行自动化

自动化运行用户通过引用另一个应用程序的对象、属性和方法来返回、编辑和输出数据。用户能够在另一个应用程序中进行引用的 Application 对象称为自动化对象。使 Word 能够被另一个应用程序所使用的第一步，就是创建对 Word 类型库的一个引用。要创建对 Word 类型库的引用，可以在 “Visual Basic 编辑器”里,单击 “工具”菜单中的“引用”,然后在“Microsoft Word 8.0 Object Library”旁边的复选框内打上钩。

下一步，声明一个引用 Word Application 对象的对象变量，如下例所示。

```
Dim appWD As Word. Application. 8
```

使用带有 Word OLE 程序标识符的 Visual Basic CreateObject 函数或 GetObject 函数，如下例所示。如果想看到 Word 时段，设置 Visible 属性为 True(真)。

```
Dim appWD As Word. Application. 8
Set appWD = CreateObject ("Word. Application.8")
appWd. Visible = True
```

CreateObject 函数返回一个 Word Application 对象,并且将它赋给 appWD。通过使用 Word Application 对象的对象、属性和方法,用户可以用该变量对 Word 进行控制。以下的实例新建一个 Word 文档。

```
appWd. Documents. Add
```

CreateObject 函数启动一个 Word 时段，当引用 Application 对象的对象变量失效时，该时段内的自动化不会关闭。设置该对象引用 Nothing 关键字也不会关闭 Word。而是使用 Quit 方法来关闭 Word。以下的 Microsoft Excel 示例把“工作表 1” 中 A1: B10 单元格里的数据插入到一个 Word 新文档中，然后调整表格中的数据。如果使用了 CreateObject 函数，则该示例使用 Quit 方法来关闭 Word 的新实例。如果 GetObject 函数返回错误 429，那么该示例使用 CreateObject 来启动一个 Word 新实例。

```
Dim appWD As Word. Application
Err.Number = 0
On Error GoTo notloaded
Set appWD = GetObject (, "Word. Application.8")
notloaded:
If Err. Number = 429 Then
Set appWD = CreateObject ("Word. Application.8")
theError = Err. Number
End If
appWD. Visible = True
With appWD
Set myDoc = .Documents. Add
```

```
With .Selection
For Each c In Worksheets ("Sheet1"). Range ("A1:B10")
.InsertAfter Text :=c.Value
Count = Count + 1
If Count Mod 2 = 0 Then
.InsertAfter Text :=vbCr
Else
.InsertAfter Text :=vbTab
End If
Next c
.Range. ConvertToTable Separator :=wdSeparateByTabs
.Tables(1). AutoFormat Format :=wdTableFormatClassic1
End With
myDoc. SaveAs FileName :="C:\Temp.doc"
End With
If theError = 429 Then appWD. Quit
Set appWD = Nothing
```

12.2从 Word 对另一个应用程序进行自动化

要通过从 Word 使用自动化来和另一个应用程序交换数据，必须首先在“引用”对话框(在“工具”菜单中)里设置对另一个应用程序的类类型库的一个引用。在这之后，另一个应用程序的对象、属性和方法都将出现在“对象浏览器”中，并且会在编译时自动对语法进行检查。用户也可以获得这些对象、属性和方法的内容敏感型帮助。

下一步，声明一个引用另一个应用程序中的特定类型对象的对象变量。以下示例声明了一个变量，该变量指向 Microsoft Excel Application 对象。

```
Dim xlObj As Excel.Application.8
```

用户可以通过使用 CreateObject 或 GetObject 函数来获得 Automation 对象。然后，通过使用另一个应用程序的对象、属性和方法，用户可以添加、更改或删除信息。当用户完成更改后，关闭应用程序。以下 Word 的示例判断 Microsoft Excel 是否正在运行。如果指定的 Microsoft Excel 任务存在，该示例使用 GetObject 函数；否则，它使用 CreateObject 函数。然后该示例将选定的文字传送给活动的 Microsoft Excel 工作簿里“工作表 1”的 A1 单元格中。在任务完成以后，使用带有 Nothing 关键字的 Set 语句来清除 Automation 对象变量。

```
Dim xlObj As Excel.Application
If Tasks.Exists("Microsoft Excel") = True Then
Set xlObj = GetObject(, "Excel.Application")
Else
Set xlObj = CreateObject("Excel.Application")
End If
xlObj.Visible = True
If xlObj.Workbooks.Count = 0 Then xlObj.Workbooks.Add
xlObj.Worksheets("Sheet1").Range("A1").Value = Selection.Text
Set xlObj = Nothing
```

以下 Word 的示例判断 PowerPoint 是否正在运行。如果存在 PowerPoint 任务，那么该示例使用 GetObject 函数；否则，它使用 CreateObject 函数。然后该示例创建一个新演示文稿，文稿的第一个文本框包含活动 Word 文档的名称，第二个文本框包含活动文档第一段的文字。在任务完成后，使用带有 Nothing 关键字的 Set 语句来清除 Automation 对象变量。

```
Dim pptObj As PowerPoint.Application
If Tasks.Exists("Microsoft PowerPoint") = True Then
Set pptObj = GetObject(, "PowerPoint.Application")
Else
Set pptObj = CreateObject("PowerPoint.Application")
End If
pptObj.Visible = True
Set pptPres = pptObj.presentations.Add
Set aSlide = pptPres.Slides.Add(Index:=1, Layout:=ppLayoutText)
aSlide.Shapes(1).TextFrame.TextRange.Text = ActiveDocument.Name
aSlide.Shapes(2).TextFrame.TextRange.Text = _
ActiveDocument.Paragraphs(1).Range.Text
Set pptObj = Nothing
```

要了解关于对 Microsoft Access 进行自动化的信息，请参阅第三章“Microsoft Access 对象”。要了解关于 Word 中 Data Access 对象(DAO)的使用，请参阅“帮助”中的“在 Microsoft Word 中使用 DAO”。

12.3同内嵌 Word 对象进行通信

以后可以使用任何 Word 对象的 Application 属性来返回 Word Application 对象。这有助于从内嵌在另一个应用程序中的 Word 文档里方法 Word Application 对象。以下的在 Microsoft Excel 中运行的示例，将一个对象变量设置为 Word Application 对象。(为了使该示例能够运行，在活动工作表中的形状一必须是一个内嵌 Word 文档。)该示例中最后一条命令在内嵌 Word 文档的开头添加文字。

```
Dim appWRD As Word. Application
Set embeddedDoc = ActiveSheet. Shapes(1)
Set appWRD = embeddedDoc. OLEFormat. Object. Application
appWRD. ActiveDocument. Range(Start :=0, End :=0). InsertBefore
Text :="New text "
```

以下在 PowerPoint 中运行的示例，将一个对象变量设置为 Word Application 对象。(为了使该示例能够运行，在演示文稿中的幻灯片一必须是一个内嵌 Word 文档。)该示例的最后一条命令显示了在内嵌 Word 文档中的文字。

```
Dim appWRD As Word. Application
Set embeddedDoc = Presentations(1). Slides(1). Shapes(1)
embeddedDoc. OLEFormat. Activate
Set appWRD = embeddedDoc.OLEFormat. Object. Application
MsgBox appWRD. ActiveDocument. Content.Text
```