

屏幕显示“安全警告”对话框，这时可能会考虑先不做任何选择，而是先把安全级别修改为“中”。这样做就可以启用宏，而不会把该宏的创建者添加到可靠发行商中。

◆勾选“总是相信来自此发布者的宏”复选框，将此程序发行商加入到可靠发行商中，但是之后单击“禁用宏”按钮而不是“启用宏”按钮，禁用该文件中的宏。一般不会这么做。

注意：一旦指定发行商为可靠发行商，所有的VBA宿主软件都会认定来自该发行商的程序可靠。

删除一个可靠发行商

要从可靠发行商的列表上删除某个发行商，可以遵循以下步骤：

1. 选择“工具”>“宏”>“安全性”以显示“安全性”对话框。
2. 单击“可靠发行商”或“可靠来源”选项卡显示“可靠发行商”或“可靠来源”页面。
3. 在列表框中，选择需要删除的可靠发行商。
4. 单击“删除”按钮，从可靠发行商中删除该项目。

锁定代码

为了防止其他人看到工程的内容，可以用密码将其锁定。一般会在把工程分发给同事时采取这样的措施。如果工作室特别不保险，有时也会锁定正在开发的工程。有人反对这样锁定工程，因为必须首先输入密码才能进入工程的模块和窗体，感觉十分麻烦——但是，如果从安全性考虑，这样稍微花点精力是很值得的。

通过以下的步骤可以锁定文档或模板工程：

1. 打开相应的文档或模板，然后调用Visual Basic编辑器。
2. 在“工程资源管理器”中，右击需要锁定的工程，然后从菜单中选择“Project属性”，显示“Project—工程属性”对话框。也可使用另一种方法，在“工程资源管理器”中选择相应的工程，然后选择“工具”>“Project属性”。
3. 单击“保护”选项卡，显示“保护”页面（如图19.15所示）。

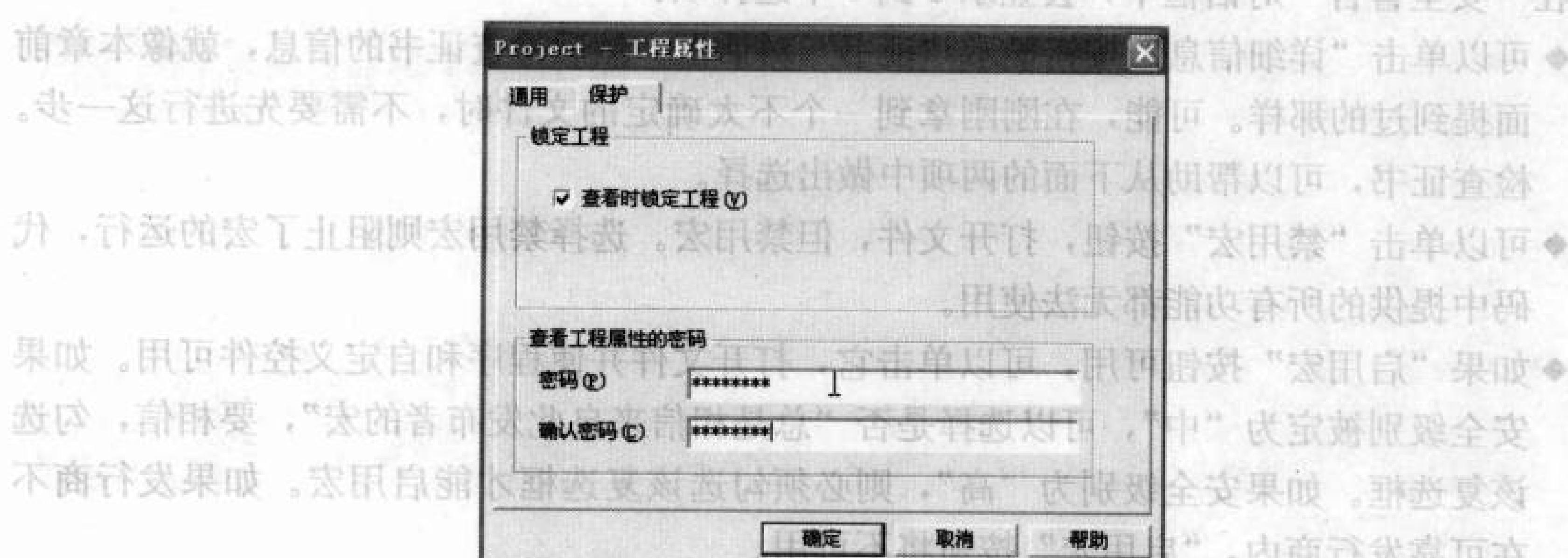


图19.15 使用“Project—工程属性”对话框中的“保护”页面锁定工程

4. 勾选“锁定工程”组合框中的“查看时锁定工程”复选框。
5. 在“查看工程属性的密码”组合框的“密码”文本框中输入密码，并在“确认密码”文本框中输入相同的密码。设定密码是必需的：不设定密码就不能锁定工程。
6. 单击“确定”按钮，锁定工程。Visual Basic 编辑器将关闭“Project—工程属性”对话框，但不关闭工程，以便继续查看或修改。
7. 回到软件，保存项目并关闭。

完成了上面的操作就完成了锁定，必须输入密码才能查看和修改工程。（除非有人破译了密码，相关内容可参见后面的补充内容“破译保护密码有多难”。）当要编辑该工程中的程序，或想要在 Visual Basic 编辑器的“工程资源管理器”中展开该工程的内容时，就会弹出“Project 密码”对话框（如图 19.16 所示），在“密码”文本框中输入密码，单击“确定”按钮，显示该工程的内容。（如果输入了错误的密码，软件或 Visual Basic 编辑器将显示“工程锁定”信息框，之后再次显示“Project 密码”对话框，要求再次输入密码。）

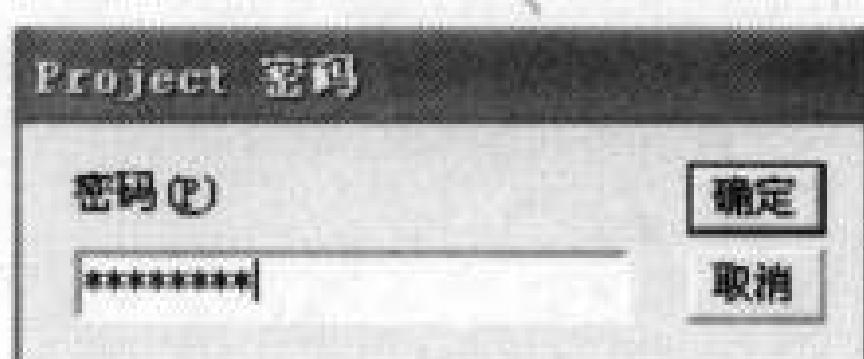


图 19.16 当打开一个锁定的工程时，
需要在“Project 密码”对话
框中输入正确的密码

选择更安全的密码锁定代码

锁定密码越长越复杂，要破译它的难度就越大。在实际情况中，最好使用 8 到 15 个字符长的密码；长于 15 个字符的密码比较难记，输入起来也很麻烦。

通常不要使用现实存在的单词做密码，即便是另一种语言的实际单词也不要使用：破译者（恶意的黑客）可以使用外文字典来破解密码，就像使用母语字典一样。把单词或词组连接在一起，在其中加入数字和符号（&、!、#，等等），这样组成的密码更加复杂。另外，一定要牢记自己的密码。

要解除对一个工程的锁定，可以在 Visual Basic 编辑器中打开它（这时需要输入密码），调用“Project—工程属性”对话框，选择“保护”页面，勾除“查看时锁定工程”复选框。单击“确定”按钮，保存包含该工程的文件。

破译保护密码有多难

任何密码都是可以被破译的。可以用足够长的字符，加入数字和符号，避免使用任何语言中的实际单词，组成一个难以猜测的密码，但是任何的密码保护最终都能够打开，可以用强硬的手段打开，也可以通过解密打开。如果有无限多的猴子和无限多的键盘供它们敲击来破译 VBA 的保护密码，他们迟早会成功。

没有人抓来这么一群猴子，当然也不需要，因为人们已经编写了很多密码破解程序来破译密码。有些破解程序使用强硬的手段，不断输入不同的密码，直到找到有用的那个，但是有更加精明的做法：为了找到文件中某个 VBA 工程的密码，他们阅读这个文件，然后就

破译了密码。

要找到这样的密码破解程序，可以搜索“Microsoft Office 密码修复”或类似的关键字。

假定任何人都可以免费或花小钱获得密码破解程序，那应当将密码作为第一防线。要保护好自己的秘密，就需要看好自己的文件，不能让任何人对它们运行密码破解程序。

如果真的有人想破解你的 Word 文档，首先他必须知道“打开或修改”对话框的密码（如果设置了）。如果他不知道密码，但知道文档中包含“密码”字样，那么他就可以通过“插入”→“文本”→“对象”→“Microsoft Project”来打开该文件。这样，他就可以看到“项目”→“项目信息”→“常规”选项卡下的“打开或修改”对话框，从而知道正确的密码。



如果 Word 文档没有设置密码，那么直接单击“OK”按钮即可。

如果“忘记密码”按钮不可用，

则说明该文档未设置密码。

如果 Word 文档设置了密码，但忘记了密码，那么可以尝试以下方法。首先，将 Word 文档另存为纯文本文件，然后将其转换为记事本格式。接着，将记事本文件打开，然后在其中输入一些乱字符，如“ $\#$ ”、“\$”、“%”等。之后，将光标移动到文件末尾，按住 Shift 键不放，同时按住 Alt 键，然后按字母键“P”（如果忘记了密码，那么可以尝试按字母键“O”或“F”）。这样，Word 就会提示你输入密码。如果输入正确的密码，那么 Word 就会显示该文件的内容。

如果忘记了密码，可以在“打开或修改”对话框中输入任意字符串，然后单击“确定”按钮。这样，Word 就会提示你输入密码。如果输入正确的密码，那么 Word 就会显示该文件的内容。

如果忘记了密码，可以在“打开或修改”对话框中输入任意字符串，然后单击“确定”按钮。这样，Word 就会提示你输入密码。如果输入正确的密码，那么 Word 就会显示该文件的内容。

第六部分 办公软件编程

- ◆ 第 20 章 了解 Word 对象模型和重要对象
- ◆ 第 21 章 使用 Word 中广泛使用的对象进行工作
- ◆ 第 22 章 了解 Excel 对象模型和重要对象
- ◆ 第 23 章 使用 Excel 中广泛使用的对象进行工作
- ◆ 第 24 章 了解 PowerPoint 对象模型和重要对象
- ◆ 第 25 章 使用 Shapes 进行工作和运行幻灯片放映
- ◆ 第 26 章 了解 Outlook 对象模型和重要对象
- ◆ 第 27 章 使用 Outlook 中的事件进行工作
- ◆ 第 28 章 了解 Access 对象模型和重要对象
- ◆ 第 29 章 使用 VBA 处理 Access 数据库中的数据
- ◆ 第 30 章 实现不同软件间的访问

第 20 章 了解 Word 对象模型和重要对象

- ◆ 考查 Word 对象模型
- ◆ 考了解 Word 的可创建对象
- ◆ 使用 Documents 集合和 Document 对象进行工作
- ◆ 使用 Selection 对象进行工作
- ◆ 创建和使用 Range 对象
- ◆ 设置 Options 对象

本章将介绍如何从作为 Word 的基础性理论体系结构的 Word 对象模型开始，使用一些最为立即可用的 Word 对象来实施常见的行动。这些对象包括 Documents 集合和 Document 对象、Selection 对象、Range 对象以及 Options 对象。

考查 Word 对象模型

为了在 Word 中使用 VBA 进行工作，不一定要懂得 Word 对象模型是如何构建起来的，但是很多人发现，具备一些对象模型的一般知识是很有帮助的。为了考查对象模型，可遵循如下步骤：

1. 启动或激活 Word，然后按下 Alt+F11 键。以启动或激活 Visual Basic 编辑器。
2. 选择“帮助”>“Microsoft Visual Basic 帮助”以显示“Microsoft Visual Basic 帮助”任务窗格。
3. 在“搜索”文本框中键入“Word 对象模型”，然后按下 Enter 键，或单击“开始搜索”按钮。
4. 在“搜索结果”窗格中，单击“Word 对象模型”条目，就可以看到帮助屏幕（如图 20.1 所示）。

在屏幕上观看“Word 对象模型”条目时，可以看到有些框的底色是黄色的，而另外一些框的底色是蓝色的。黄框是组织成集合的对象，而蓝框里是没有集合的对象。单击一个框，就可以看到对应于该对象的帮助屏幕，包括图解和对象模型中的有关对象。图 20.2 显示出对应于“RecentFiles 集合对象”的帮助屏幕的一部分，它包括“最近使用过的文件”列表中各文件的详细情况（按照默认方式，该列表出现在“文件”菜单的底部）。

了解 Word 的可创建对象

和大多数 VBA 能使的应用程序一样，Word 有很多可创建对象。Word 允许用户去访问这些常用对象，并且可以不必通过 Application 对象去访问。下面列出一些最常用的可创建对象：

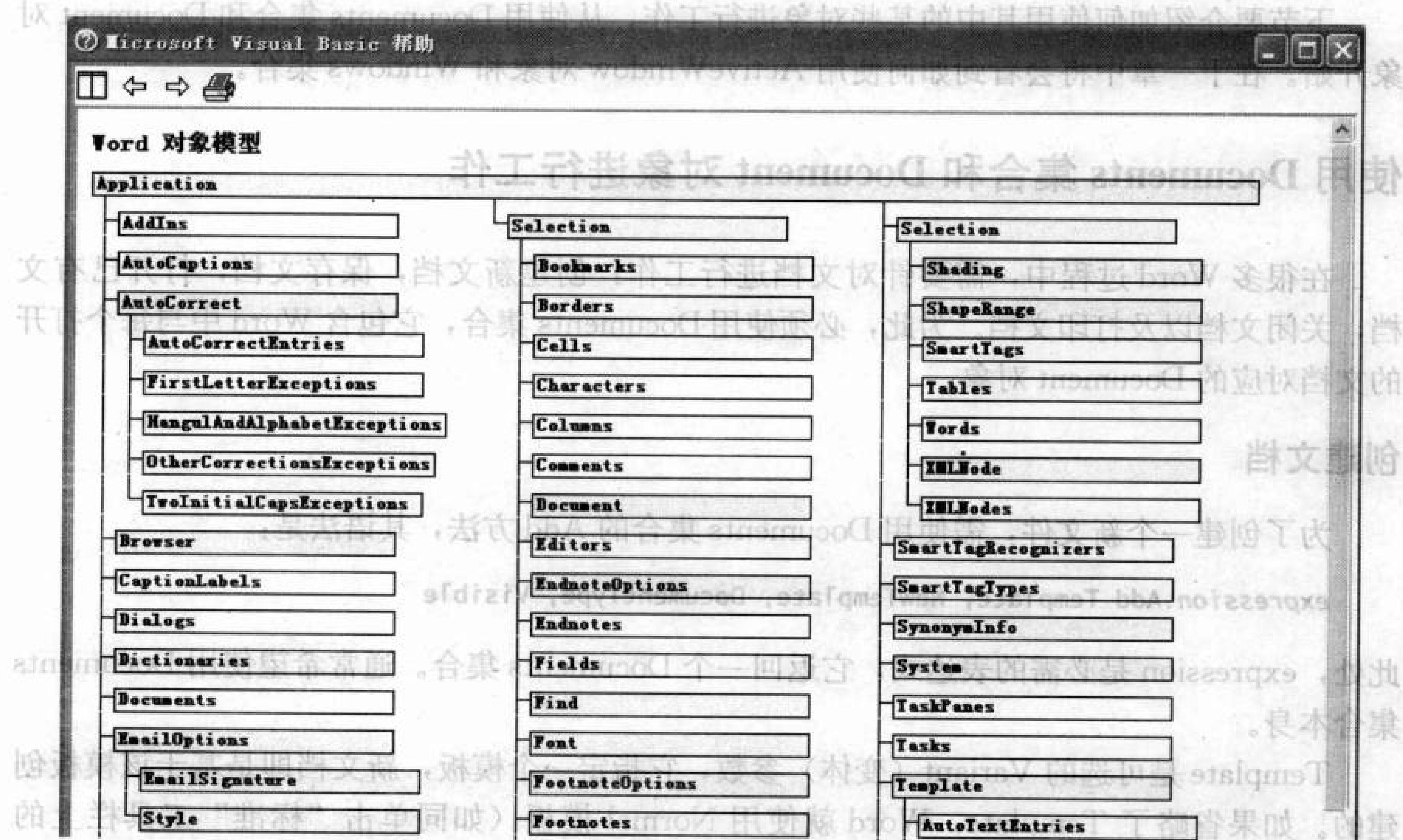


图 20.1 “Microsoft Visual Basic 帮助”任务窗格提供了 Word 对象模型的图解（这里显示的是对应于 Word 2003 的对象模型），可单击某一对象以查看其详情

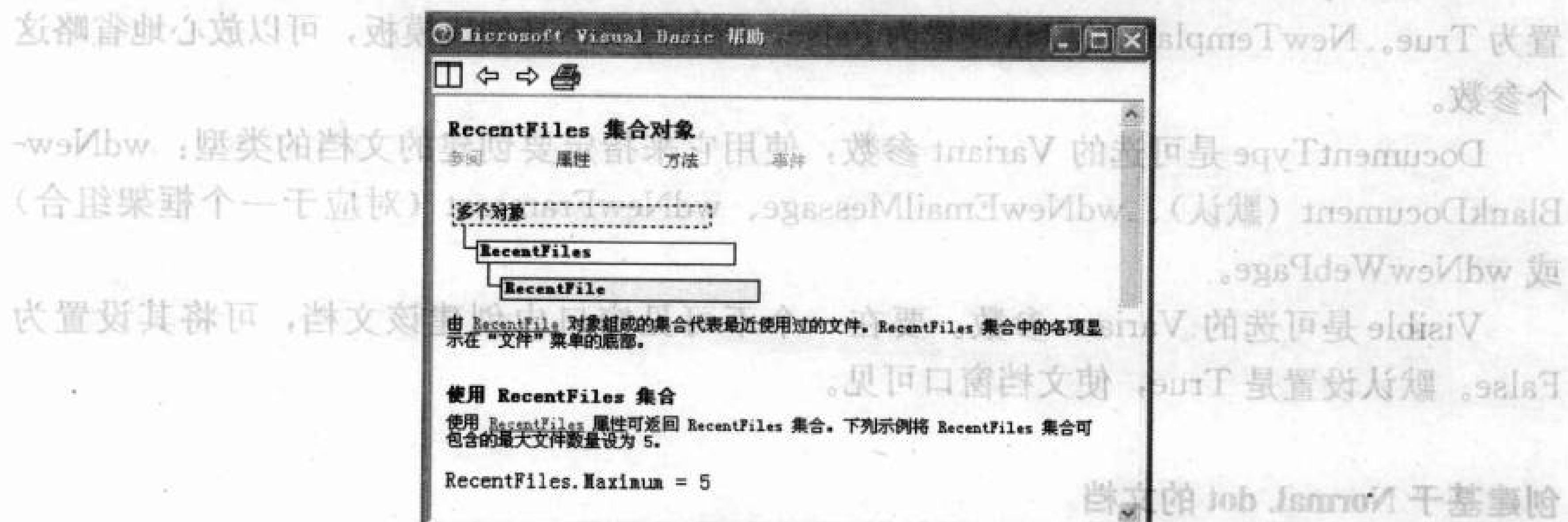


图 20.2 与单个对象或集合对应的帮助屏幕，包含一个显示与该对象或集合有关的对象的略图

- ◆ ActiveDocument 对象，它返回一个表示活动文档的 Document 对象。
- ◆ ActiveWindow 对象，它返回一个表示活动窗口的 Window 对象。
- ◆ Documents 集合，它包含若干 Document 对象，每个对象都表示一个打开的文档。
- ◆ Options 对象，它表示 Word 选项和文档选项，包括在“选项”对话框中出现的大部分选项。
- ◆ Selection 对象，它表示活动文档中的所选内容。Selection 表示文档中的选择内容（包含文本或其他对象）或文档中已折叠的选择内容（仅包含一个插入点）。
- ◆ Windows 集合，它包含若干 Window 对象，表示所有打开的窗口。

下节要介绍如何使用其中的某些对象进行工作，从使用 Documents 集合和 Document 对象开始。在下一章中将会看到如何使用 ActiveWindow 对象和 Windows 集合。

使用 Documents 集合和 Document 对象进行工作

在很多 Word 过程中，需要针对文档进行工作：创建新文档，保存文档，打开已有文档，关闭文档以及打印文档。为此，必须使用 Documents 集合，它包含 Word 中与每个打开的文档对应的 Document 对象。

创建文档

为了创建一个新文件，需使用 Documents 集合的 Add 方法，其语法是：

```
expression.Add Template, NewTemplate, DocumentType, Visible
```

此处，expression 是必需的表达式，它返回一个 Documents 集合。通常希望使用 Documents 集合本身。

Template 是可选的 Variant (变体) 参数，它指定一个模板，新文档即是基于该模板创建的。如果省略了 Template，Word 就使用 Normal 模板（如同单击“标准”工具栏上的“新建空白文档”按钮）。所以，只有需要基于 Normal 之外的另一模板来创建新文档时，才有必要指定一个 Template 参数。

NewTemplate 是可选的 Variant 参数，如果要创建的是一个模板而不是文档，要将它设置为 True。NewTemplate 的默认设置为 False，所以只要不是创建模板，可以放心地省略这个参数。

DocumentType 是可选的 Variant 参数，使用它来指定要创建的文档的类型：wdNewBlankDocument (默认)、wdNewEmailMessage、wdNewFrameset (对应于一个框架组合) 或 wdNewWebPage。

Visible 是可选的 Variant 参数。要在不可见窗口内创建该文档，可将其设置为 False。默认设置是 True，使文档窗口可见。

创建基于 Normal.dot 的文档

下述语句创建一个基于 Normal.dot 全局模板的新文档：

```
Documents.Add
```

创建基于模板的文档

下述语句创建一个基于模板的新文档。模板名称为 Company Report.dot，该模板保存在名为 \\server\public\templates 的网络文件夹中：

```
Documents.Add Template:= "\\server\public\templates\Company Report.dot"
```

创建模板

下述语句声明一个新的对象变量，以 Template 类中的 myTemplate 作为变量的名称，

创建一个基于保存在默认模板文件夹之中的名为 Overhead.dot 的模板的新模板，将它指定给 myTemplate，并隐藏窗口：

```
Dim myTemplate As Template
Set myTemplate = Documents.Add(Template:="Overhead.dot",
    NewTemplate:=True, Visible:=False)
```

在本例中，通往模板的路径未加指定，因为该模板在一个模板文件夹之中。

注意：Word 有两个模板文件夹：用户模板文件夹和工作组模板文件夹。通过选择“工具”>“选项”，并在“选项”对话框中的“文件位置”标签上设置，可以改变这些文件夹的位置。

保存文档

和以交互方式工作时一样，当使用 VBA 工作时，要第一次保存某个文档时，需指定文件名和路径。在这之后，可以在相同名称之下保存该文档或指定另一个不同的名称或格式。

第一次保存某一文件或另存为另一文件

为了第一次保存某一文件，或者以另一个不同名称保存某一文件，或者以另一种不同格式保存某一文件，需使用 SaveAs 方法，其语法是：

```
expression.SaveAs(FileName, FileFormat, LockComments, Password, AddToRecentFiles,
WritePassword, ReadOnlyRecommended, EmbedTrueTypeFonts, SaveNativePictureFormat,
SaveFormsData, SaveAsAOCELetter, Encoding, InsertLineBreaks, AllowSubstitutions,
LineEnding, AddBiDiMarks)
```

此处，expression 是一个表达式，它返回一个 Document 对象。例如，可以使用 ActiveDocument 对象或者 Documents 集合中的一个对象。

FileName 是可选的 Variant 参数，它为该文档指定名称。如果省略 FileName，VBA 就使用当前文件夹、对应于一个文档的默认文件名 Docn.doc 以及对应于一个模板的默认文件名 Dotn.dot，这里的 n 是下一个可用序号（例如，对应于一个文档的 Doc5.dot，或对应于一个模板的 Dot2.dot）。

警告：保存某一文档时，应检查是否已经有具备这个名称和位置的文档存在。如果有的话，VBA 会覆盖它且不加警告，这将引起数据丢失。

FileFormat 是可选的 Variant 参数，它指定以哪种格式来保存该文档。表 20.1 列出了与指定最常用的格式对应的 wdSaveFormat 常量。

表 20.1 wdSaveFormat 常量

常量	将文档保存为
wdFormatDocument	一个 Word 文档
wdFormatDOSText	一个 DOS 文本文件
wdFormatDOSTextLineBreaks	一个使用版式的 DOS 文本文件
wdFormatEncodedText	一个使用编码的文本文件
wdFormatFilteredText	一个已筛选的 HTML 文件（仅 Word 2003 和 XP 有）

常用的常量，如表 10-1 所示。（续表）

常量	将文档保存为
wdFormatHTML	一个 HTML 文件
wdFormatRTF	一个 Rich Text Format (RTF) 文件
wdFormatTemplate	一个 Word 模板
wdFormatText	一个文本文件 (明文 ASCII)
wdFormatTextLineBreaks	一个使用分行符的文本文件
wdFormatUnicodeText	一个使用 Unicode 字符的文本文件
wdFormatWebArchive	一个 Web 档案文件
wdFormatXML	一个 XML 文件 (仅 Word 2003 有)

例如，下述语句是在当前文件夹中在名称 Example.html 之下，以已筛选的 HTML 文件的格式来保存活动文档：

```
ActiveDocument.SaveAs FileName:="Example.html", _  
FileFormat:=wdFormatFilteredHTML
```

提示：除了上述常量以外，还可以以其他格式来保存文档。方法是：为 FileConverter 对象的 SaveFormat 属性指定一个适当的值，使对应于所需格式的文件转换器被安装——例如，使用 ActiveDocument. SaveAs FileName: = FileConverters(15). SaveFormat。详情请查阅“VBA 帮助”文件中的 FileConverters 条目。

AddToRecentFiles 是可选的 Variant 参数，当设置为 True 时，使得 Word 将该文档添加到“文件”菜单上的“最近使用过的文件”列表中去。（当在过程中使用一些文档进行工作时，为使用户原先的“最近使用过的文件”列表不受打扰，常常希望避免在“最近使用过的文件”列表中列出这些文档。）

为了在保存文档时对其进行保护，可以使用四种不同的保护特性：

- ◆ LockComments 是可选的 Variant 参数，将其设置为 True，可以锁定该文档，这样，审阅者可以输入批注，但不能改变文档的文本。
- ◆ Password 是可选的 Variant 参数，可用来设置打开文档所需要的密码。
- ◆ WritePassword 是可选的 Variant 参数，可用来设置保存对文档的更改所需要的密码。
- ◆ ReadOnlyRecommended 是可选的 Variant 参数，将其设置为 True，是让 Word 向用户建议应以只读方式打开该文档。

最后，还有如下一些不常使用的可选参数：

- ◆ EmbedTrueTypeFonts 是可选的 Variant 参数，将它设置为 True，是要将 TrueType 字体与文档一起保存（当向某人分配文档时，如果知道某人并未安装可正确查看文档的 TrueType 字体，此时使用这一参数是个好办法）。
- ◆ SaveNativePictureFormat 是可选的 Variant 参数，将它设置为 True，是把从另一平台导入的图形作为 Windows 图形来保存。
- ◆ SaveFormsData 是可选的 Variant 参数，将它设置为 True，是把在窗体中输入的数据保存为数据记录（与保存整个窗体（包括它的静态文本）不同）。

- ◆ SaveAsAOCELetter 是可选的 Variant 参数，将它设置为 True，是把文档保存为 AOCE 信函（这是发送文档的一种邮件发送格式）。
- ◆ Encoding 是可选的 Variant 参数，设置此参数是为了使用不同于系统代码页的另一种代码页。例如，可能需要使用 Cyrillic 代码页来保存文档。
- ◆ InsertLineBreaks 是可选的 Variant 参数，当要把文档保存为文本文件，让 Word 在文本的每个行末都插入一个分行符时，将其设置为 True。
- ◆ AllowSubstitutions 是可选的 Variant 参数，当要把文档保存为文本文件，让 Word 以类似文本来代替某些符号字符时，将其设置为 True，例如，Word 以 (TM) 来代替商标符号 (™)。
- ◆ LineEnding 是可选的 Variant 参数，当要把文档保存为文本以控制 Word 如何标明分行符和分段符时，使用此参数。
- ◆ AddBiDiMarks 是可选的 Variant 参数，将它设置为 True，可以使 Word 把一些控制字符添加到文件，从而保持双向版式。

通常，如果是第一次保存某一个文件，必须指定它的名称和路径；如果想要以不同于 Word 文档的另一格式来保存文件，也需要指定其名称和路径。下述语句是在文件夹 \\server\Products\Field\之中，在名称 Beehives.doc 之下保存活动文档：

```
ActiveDocument.SaveAs  
    "\\server\Products\Field\Beehives.doc"
```

保存已经保存过的文档

如果某一个文档已经保存过了，可以使用 Save 方法在相同名称下再保存它。对于 Document 对象，Save 方法无需参数。例如，下述语句保存名为 Recipe01.doc 的文档：

```
Documents("Recipe01.doc").Save
```

保存所有打开的文档

为了保存所有打开的文档，需使用针对 Documents 集合的 Save 方法，其语法是：

```
expression.Save(NoPrompt, OriginalFormat)
```

此处，expression 是一个表达式，它返回一个 Documents 集合，常常是使用 Documents 集合本身。

NoPrompt 是可选的 Variant 参数，将它设置为 True，是让 Word 保存包含有未保存的更改的所有打开的文档，以及包含有未保存的更改的任何附着模板，而不向用户做任何提示。默认设置是 False，它使得 Word 向用户提示，是否要保存每个文档和模板。即使将 NoPrompt 设置为 True，但如果“选项”对话框的“保存”标签上的“提示保存 Normal 模板”复选框被选中，Word 仍将提示用户，是否要保存对 Normal.dot 的更改。

OriginalFormat 是可选的 Variant 参数，可以设置为 wdOriginalDocumentFormat，以便按各文档的初始格式来保存各文档；设置为 wdWordDocument，以迫使每个文档保存为 Word 文档；或设置为 wdPromptUserX 以提示用户使用哪一种格式。

例如，下述语句保存所有打开的文档和模板，不向用户做任何提示：

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

文档操作文档是，**WT** 文档对象，通过 **insisV** 对象和 **set**。通过 **Set** 和 **Set**。

检查文档中是否有未保存的更改

要找出文档中是否有未保存的更改，需检查它的 Saved 属性。Saved 是一个读/写布尔型属性，如果文档包含有未保存的更改，它返回 False；如果没有，则返回 True。一个新建文档不会含有未保存的更改，因为它从未被保存过。

打开文档

为了打开文档，需使用针对适当的 Document 对象的 Open 方法。对应于 Open 方法的语法是：

```
expression.Open FileName, ConfirmConversions, ReadOnly, AddToRecentFiles,
PasswordDocument, PasswordTemplate, Revert, WritePasswordDocument,
WritePasswordTemplate, Format, Encoding, Visible
```

各参数说明如下：

- ◆ expression 是必需的表达式，它返回一个 Documents 集合。通常，希望使用 Documents 集合本身。
- ◆ FileName 是必需的 Variant 参数，它指定要打开的文档的名称（如有必要，还要加上路径）。
- ◆ ConfirmConversions 是可选的 Variant 参数，将它设置为 True 时，如果文件格式不是 Word 文件格式，使得 Word 显示出“文件转换”对话框。
- ◆ ReadOnly 是可选的 Variant 参数，将它设置为 True，是要以只读方式打开该文档。
- ◆ AddToRecentFiles 是可选的 Variant 参数，将它设置为 True，使得 Word 将文件添加到“文件”菜单底部处的“最近使用过的文件”列表中去。
- ◆ PasswordDocument 是可选的 Variant 参数，可以使用它来设置打开文档的密码。
- ◆ PasswordTemplate 是可选的 Variant 参数，可以使用它来设置打开模板的密码。
- ◆ Revert 是可选的 Variant 参数，它指定当 FileName 与已经打开的某一文件同名时，Word 应做什么。按照默认方式（即不包含 Revert 参数），Revert 设置为 False，意味着 Word 激活已打开的文档，但不打开已保存的文档。如果将 Revert 设置为 True，则 Word 打开已保存的文档，并放弃对已打开文档的任何更改。
- ◆ WritePasswordDocument 是可选的 Variant 参数，它指明保存对文档的更改所需要的密码。
- ◆ WritePasswordTemplate 是可选的 Variant 参数，它指明保存对模板的更改所需要的密码。
- ◆ Format 是可选的 Variant 参数，可以使用它来指定以哪种文件转换器来打开文档。表 20.2 列出了可以用来指定文件转换器的 wdOpenFormat 常量。

表 20.2 与打开文档对应的 wdOpenFormat 常量

常量	效果
wdOpenFormatAllWord	Word 把具有任何已识别的 Word 格式的文档作为 Word 文档来打开
wdOpenFormatAuto	Word 自动选择一个转换器，这是默认设置
wdOpenFormatDocument	Word 把文档作为一个 Word 文档来打开

(续表)

常量	效果
wdOpenFormatEncodedText	Word 把文档作为一个使用编码的文本文件来打开
wdOpenFormatRTF	Word 把文档作为一个 Rich Text Format (RTF) 文件来打开
wdOpenFormatTemplate	Word 把文档作为一个模板来打开
wdOpenFormatText	Word 把文档作为一个文本文件来打开
wdOpenFormatUnicodeText	Word 把文档作为一个 Unicode 文本文件来打开
wdOpenFormatWebPages	Word 把文档作为一个 Web 页来打开

◆ Encoding 是可选的 Variant 参数，它指定打开文档时 Word 使用的文档编码（代码页或字符集）。

◆ Visible 是可选的 Variant 参数，将它设置为 False 使得 Word 在一个不可见窗口内打开文档（默认设置为 True，此时指定一个可见窗口）。

下述语句打开 C:\My Documents\文件夹中的文档 Times.doc：

```
Documents.Open "C:\My Documents\Times.doc"
```

下述语句以只读方式打开 D:\Temp\文件夹中的文件 Statistics.doc，并将它加至“最近使用过的文件”列表上。

```
Documents.Open "D:\Temp\Statistics.doc", ReadOnly:=True, _  
AddToRecentFiles:=True
```

关闭文档

为了关闭文档，可使用针对应用程序的 Document 对象的 Close 方法。其语法是：

```
expression.Close(SaveChanges, OriginalFormat, RouteDocument)
```

此处，expression 是必需的表达式，它返回一个 Document 对象或 Documents 集合。

SaveChanges 是可选的 Variant 参数，可以用它来指定如何处理未保存的更改。使用 wdDoNotSaveChanges 以放弃更改；使用 wdPromptToSaveChanges 让 Word 提示用户保存更改；或使用 wdSaveChanges 以保存更改而不加提示。

OriginalFormat 是可选的 Variant 参数，可以使用它来指定文档的保存格式。使用 wdOriginalDocumentFormat，让 Word 使用初始文档格式；使用 wdPromptUser，让 Word 提示用户选择一种格式；或使用 wdWordDocument，以采用 Word 文档格式。

RouteDocument 是可选的 Variant 参数，将它设置为 True 以便传送具有附加的传送名单的文档。

例如，下述语句将关闭活动文档且不保存更改：

```
ActiveDocument.Close SaveChanges:=wdDoNotSaveChanges
```

下述语句关闭所有打开的文档（但不关闭 Word 应用程序自身），并自动地保存更改：

```
Documents.Close SaveChanges:=wdSaveChanges
```

(未翻) 更改文档的模板

为了更改附着某一文档的模板，需将希望受其影响的 Document 对象的 AttachedTemplate 属性设定为适当模板的路径和名称。例如，下述语句使名为 SalesMarket02.doc 的模板附着活动文档。由于已假定该模板在某一个 Word 模板文件夹之内，所以路径就不指定了。

```
ActiveDocument.AttachedTemplate = "SalesMarket02.dot"
```

打印文档

为了打印文档，可使用对应于适当的 Document 对象的 PrintOut 方法。与 PrintOut 方法对应的语法是：

```
expression.PrintOut(Background, Append, Range, OutputFileName, From, To, Item,
Copies, Pages, PageType, PrintToFile, Collate, FileName, ActivePrinterMacGX,
ManualDuplexPrint, PrintZoomColumn, PrintZoomRow, PrintZoomPaperWidth,
PrintZoomPaperHeight)
```

以下是 PrintOut 方法的各组成部分：

- ◆ expression 是必需的表达式，它指定一个 Application 对象、 Document 对象或 Window 对象。通常要打印的是 Document 对象。
- ◆ Background 是可选的 Variant 参数，将它设置为 True 时，允许过程继续运行，而使 Word 在后台打印文档。
- ◆ Append 是可选的 Variant 参数，将它设置为 True，是把被打印文档添加到指定的打印文件中去。
- ◆ Range 是可选的 Variant 参数，它指定要打印的选择内容或页面范围： wdPrintAllDocument (0)，(默认值)、 wdPrintCurrentPage (2)、 wdPrintFromTo (3)；(使用 From 和 To 参数来指定页面)、 wdPrintRangeOfPages (4) 或者 wdPrintSelection (1)。
- ◆ OutputFileName 是可选的 Variant 参数，在打印某一个文件时，指定输出文件的名称。
- ◆ From 是可选的 Variant 参数，用来指定打印页面范围的起始页码。
- ◆ To 是可选的 Variant 参数，用来指定打印页面范围的结束页码。
- ◆ Item 是可选的 Variant 参数，用来指定要打印的项目： wdPrintAutoTextEntries (4)、 wdPrintComments (2)、 wdPrintDocumentContent (0) (默认值)、 wdPrintKeyAssignments (5) 指定文档或其模板的快捷键、 wdPrintProperties (1) 或者 wdPrintStyles (3)。
- ◆ Copies 是可选的 Variant 参数，用来指定打印的份数 (如省略 Copies, Word 只打印一份)。
- ◆ Pages 是可选的 Variant 参数，用来指定要打印的页面——例如，1, 11~21, 31。
- ◆ PageType 是可选的 Variant 参数，用来指定是打印所有页面 (wdPrintAllPages, 0, 默认值)、奇数页面 (wdPrintOddPagesOnly, 1) 或者偶数页面 (wdPrintEvenPagesOnly, 2)。

- ◆ PrintToFile 是可选的 Variant 参数，将它设置为 True，是把打印操作的输出发送到某一个文件。
- ◆ Collate 是可选的 Variant 参数，在打印一个文档的多份副本时，用来确定是否在打印好一份副本后，再打印一份副本。设置为 True 则是，为 False 则不是。
- ◆ FileName 是可选的 Variant 参数，用来指定要打印的文档的文件名和路径（如果省略了 FileName，VBA 就打印活动文档）。
- ◆ ActivePrinterMacGX 是可选的 Variant 参数，使用 Macintosh 计算机时，如果安装了 QuickDrawGX，用此参数来指定打印机。
- ◆ ManualDuplexPrint 是可选的 Variant 参数，将它设置为 True，是要在无双面打印组件的打印机上进行双面打印。当它为 True 时，可以使用 Options 对象的 PrintOddPagesInAscendingOrder 属性或 PrintEvenPagesInAscendingOrder 属性，以递增顺序打印奇数页面或偶数页面，以便产生出人工双面打印效果（换一个方向将印好的奇数页面纸重新装入打印机，以便打印出偶数页面）。ManualDuplexPrint 参数只能在某些语言中使用。
- ◆ PrintZoomColumn 和 PrintZoomRow 是可选的 Variant 参数，在单张纸上打印多页文档时，用它们来指定一张纸上水平方向（PrintZoomColumn）和垂直方向（PrintZoomRow）要印出的页数。每个属性可取值为 1、2 或 4。
- ◆ PrintZoomPaperWidth 是可选的 Variant 参数，用来指定要把待打印页面缩放到多大宽度（以 twips 计）。
- ◆ PrintZoomPaperHeight 是可选的 Variant 参数，用来指定要把待打印页面缩放到多大高度（以 twips 计）。

例如，下述语句在后台相继打印出活动文档的三份副本：

```
ActiveDocument.PrintOut Background:=True, Copies:=3, Collate:=True
```

下述语句打印出活动文档的第 2 页至第 5 页：

```
ActiveDocument.PrintOut Range:=wdPrintFromTo, From:=2, To:=5
```

下述语句在每张纸上打印出活动文档的两个页面：

```
ActiveDocument.PrintOut PrintZoomColumn:=2, PrintZoomRow:=1
```

使用 ActiveDocument 对象进行工作

ActiveDocument 对象返回一个表示活动文档的 Document 对象——活动文档即是在 Word 窗口内具有焦点的文档。ActiveDocument 对象的行为与 Document 对象相像，但使用它工作时，有两个可能出现的问题要加以注意。

第一个问题是，如果 Word 中没有文档是打开的，那就没有 ActiveDocument 对象，试图使用 ActiveDocument 对象来工作的任何代码都会返回一个错误。可以检查 Documents 集合的 Count 属性来确认是否有文档是打开的。例如：

```
If Documents.Count = 0 Then  
    If MsgBox("No document is open." & vbCrLf & vbCrLf & _  
        "Do you want to create a new blank document?", _  
        vbYesNo + vbExclamation, "No Document Is Open") = vbYes Then  
            Documents.Add
```

```

    Else
        End If
    End If

```

第二个问题是，代码设想某一文档是活动文档，实际上可能是另一个不同文档是活动文档。这个问题在下面的情况下有可能发生：如果一个过程是从使用活动文档开始，然后创建了一个新文档并在其中工作；新文档成为了活动文档，而从此处起，可能会产生混乱。如果知道某一文档的名称并知道它应该是活动文档，可以检查实际上是活动的那个文档的名称是否与之相同，以便确定它是不是正确的文档。

如果对正在使用哪个文档进行工作存在疑问，那么应该声明一个 Document 对象变量，并使用这个对象变量来工作，而不是使用 ActiveDocument 对象来工作。例如，下述语句声明一个 Document 对象，并将 ActiveDocument 对象指派给它，所以以后的代码可以针对这个 Document 对象来工作：

```

Dim myDocument As Document
Set myDocument = ActiveDocument
With myDocument
    'actions here
End With

```

使用 Selection 对象进行工作

为了使用 VBA 在一个文档的某部分上实施行动，可以按以下几种方式中的任一种方式来工作：使用 Selection 对象，或者直接访问想要施加影响的对象，或者定义一个把该对象包括进去的区域。使用 Selection 对象，类似于以交互方式使用 Word 进行工作。某些过程要求用户选择插入点所指的某一个对象或位置以指示过程应对谁施加影响，对这些过程来说，使用 Selection 对象是很好的。

注意：如果正在学习针对 Word 来使用 VBA，使用 Selection 对象也是很好的，因为使用宏录制器录制的很多行动就使用了 Selection 对象。

Selection 对象表示 Word 中活动文档内的当前所选内容。所选内容可以折叠回缩为一个插入点，在这种情况下没有什么被选择；或者是，所选内容可以包含一个或多个对象——一个或多个字符、一个或多个字、一个或多个段落、一个图形、一个表格，等等。

即使所选内容折叠成了一个插入点，仍可使用它来引用所选内容之外的对象。例如，Selection.Paragraphs(1).Range.Words(10).Text 返回插入点所定位的那个段落里的第十个字，或者被选择的一个或多个段落中的第一个段落里的第十个字。

检查所选内容的类型

当工作在活动文档中时，常常需要检查哪种所选内容类型是活动的，这样就能知道正在处理的是一个被选文本块，还是没有选择内容，还是一个特殊的选择内容类型，如一个表格或一个图形。根据当前选择内容的情况，可能在某一过程中无法采取确切的行动，也可能不愿意采取其他行动。

表 20.3 列出了 Word 可区分的当前所选内容的类型。

表 20.3 Word 中所选内容的类型

wdSelectionType 常量	值	意义
wdNoSelection	0	无选择内容（这一状态似乎是不可能达到的，可以设想为：此时没有文档打开，但随后 Selection 语句返回运行时错误 91）
wdSelectionIP	1	选择内容折叠为一个插入点——没有什么被选择
wdSelectionNormal	2	一个“普通”的选择内容，如被选择的字或句子
wdSelectionFrame	3	一个框架被选择
wdSelectionColumn	4	一列或一列中的某部分被选择（如一列中的两个或多个单元格被选择，或者在两列或多列中，每列有一个单元格被选择）
wdSelectionRow	5	一个表格中的整行被选择
wdSelectionBlock	6	一个块被选择（例如，保持 Alt 键按下并用鼠标来拖曳，或使用列扩展模式，从而使一个或多个段落的垂直部分被选择）
wdSelectionInlineShape	7	一个嵌入式形状或图形被选择（形状或图形是在文本层之中，而不是浮移其上）
wdSelectionShape	8	一个 Shape 对象被选择（文本框视为 Shape 对象）

为了找出当前所选内容的类型，需返回 Selection 对象的 Type 属性。下述语句检查当前所选内容是否是一个插入点，如果是插入点，随后需插入一个文本字符串 strMyString：

```
If Selection.Type = wdSelectionIP Then
    Selection.TypeText strMyString
End If
```

检查所选内容的文字部分类型

除了所选内容的类型外，常常需要找出所选内容是在哪一种文字部分之中——是在主文本文字部分，或是批注文字部分，或是文本框文本部分，等等。检查文字部分可以帮助用户回避问题，例如用户试图在页眉或页脚中实施行动，但 Word 只支持在主文本文字部分中实施行动。

表 20.4 列出了 wdStoryType 常量以及它们对应的文字部分。

表 20.4 Word 文字部分类型

wdStoryType 常量	值	意义
wdMainTextStory	1	文档的主文本
wdCommentsStory	4	批注部分
wdEndnotesStory	3	尾注部分
wdFootnotesStory	2	脚注部分
wdTextFrameStory	5	框架中的文本
wdPrimaryFooterStory	9	基本页脚
wdEvenPageFooterStory	8	偶数页页脚
wdFirstPageFooterStory	11	第一页页脚
wdPrimaryHeaderStory	7	基本页眉
wdEvenPageHeaderStory	6	偶数页页眉
wdFirstPageHeaderStory	10	第一页页眉

例如，下述语句说明如果所选内容不在文档中的主文本之内，就显示一个消息框：

```
If Selection.StoryType <> wdMainTextStory Then
```

```
    MsgBox "This range is not in the main text."
```

```
End If
```

获得关于当前所选内容的其他信息

为了高效地使用当前所选内容进行工作，常常需要知道的选内容包含些什么以及位于何处。要找到这些信息，需使用 **Information** 属性以返回所需要知道的细节情况。表 20.5 列出了 **Information** 属性中的可用信息。

表 20.5 Information 属性中的可用信息

wdInformation 常量	返回如下信息
环境信息	
wdCapsLock	如 Caps Lock 接通（大写字母锁定有效），则返回 True
wdNumLock	如 NumLock 接通（数字锁定有效），则返回 True
wdOverType	如果改写模式有效，则返回 True（改变 Overtype 属性，可使改写模式有效或无效）
wdRevisionMarking	如果跟踪更改有效，则返回 True
wdSelectionMode	返回一个值，该值表明当前选定模式：0 表明正常选定，1 表明扩展选定（扩展模式有效），2 表明列选定
wdZoomPercentage	返回当前的放大百分比
所选内容和插入点信息	
wdActiveEndAdjustedPageNumber	返回页码，在该项中包含所选内容或区域的活动结尾。这个页码反映了对起始页码的更改情况；wdActiveEndPageNumber 则不考虑对页码的更改
wdActiveEndPageNumber	返回页码，在该项中包含所选内容或区域的活动结尾
wdActiveEndSectionNumber	返回节号，在该项中包含所选内容或区域的活动结尾
wdFirstCharacterColumnNumber	返回所选内容或区域中第一个字符的位置。如果所选内容或区域折叠成一个插入点，这个常量返回紧贴着插入点右边的字符的编号（注意：这个编号就是状态栏中“列”字后面的数字。这个“列”是相对于当前活动页面的左边界，且不必在表格里面）
wdFirstCharacterLineNumber	在“页面”视图和“打印预览”中，这个常量返回所选内容中第一个字符的行号。在非页面视图中（如“普通”视图），它返回 -1
wdFrameIsSelected	如果所选内容或区域是整个框架或文本框，则返回 True
wdHeaderRooterType	返回一个值，该值表明了包含所选内容或区域的页眉或页脚的类型：-1 表明所选内容或区域不在一个页眉或页脚中；0 表明一个偶数页页眉；1 表明具有奇数和偶数页页眉的文档中的一个奇数页页眉，或表明没有奇数和偶数页页眉的文档中的页眉；2 表明一个偶数页页脚；3 表明具有奇数和偶数页页脚的文档中的一个奇数页页脚，或表明没有奇数和偶数页页脚的文档中的页脚；4 表明第一页页眉；5 表明第一页页脚
wdHorizontalPositionRelativeToPage	返回所选内容或区域的水平位置——所选内容或区域的左边与页面左边之间的距离，以 twips 计
wdHorizontalPositionRelativeToTextBoundary	返回所选内容或区域的水平位置——所选内容或区域的左边与围住它的文本边界之间的距离，以 twips 计
wdInCommentPane	如果所选内容或区域在批注窗格内，则返回 True

(续表)

wdInformation 常量	返回如下信息
wdInEndnote	如果所选内容或区域出现在“普通”视图的尾注窗格中或“页面”视图的尾注区中，则返回 True
wdInFootnote	如果所选内容或区域出现在“普通”视图的脚注窗格中或“页面”视图的脚注区中，则返回 True
wdInFootnoteEndnotePane	如果所选内容或区域出现在脚注或尾注中，则返回 True
wdInHeaderFooter	如果所选内容或区域出现在“普通”视图的页眉或页脚窗格中或“页面”视图的页眉或页脚区中，则返回 True
wdInMasterDocument	如果所选内容或区域在一个主控文档之中，则返回 True（主控文档至少包含一个子文档）
wdInWordMail	返回一个值，该值表明所选内容或区域的 WordMail 位置：0 表明所选内容或区域不在一条 WordMail 消息中；1 表明位于正在发送的一条 WordMail 消息中；2 表明位于已收到的一条 WordMail 中
wdNumberOfPagesInDocument	返回某一文档的页数，所选内容或区域在该文档中
wdReferenceOfType	返回一个值，该值表明所选内容相对于脚注、尾注或批注引用的位置：-1 表明所选内容或区域包含一个引用；0 表明所选内容或区域不在一个引用之前；1 表明所选内容或区域在一个脚注引用之前；2 表明在一个尾注引用之前；3 表明在一个批注引用之前
wdVerticalPositionRelativeToPage	返回所选内容或区域的垂直位置——即所选内容或区域的上边与页面的上边之间的距离，以 twips 计
wdVerticalPositionRelativeToTextBoundary	返回所选内容或区域的垂直位置——即所选内容或区域的上边与围住它的文本边界之间的距离，以 twips 计
表格信息	
wdWithInTable	如果所选内容在一个表格中，则返回 True
wdStartOfRangeColumnNumber	返回包含所选内容或区域的起点的表格列号
wdEndOfRangeColumnNumber	返回包含所选内容或区域的结尾的表格列号
wdStartOfRangeRowNumber	返回包含所选内容或区域的起点的表格行号
wdEndOfRangeRowNumber	返回包含所选内容或区域的结尾的表格行号
wdAtEndOfRowMarker	如果所选内容或区域位于表格的行结尾标记处（不是单元格结尾标记），则返回 True
wdMaximumNumberOfColumns	返回所选内容或区域中任意行的最大表格列数
wdMaximumNumberOfRows	返回所选内容或区域中表格的最大行数

在所选内容处、所选内容之后或之前插入文本

使用 Selection 对象的 TypeText 方法，可以在所选内容处插入文本；使用 InsertAfter 方法，可在所选内容之后插入文本；使用 InsertBefore 方法，可在所选内容之前插入文本。其语法是：

```
Selection.TypeText string
Selection.InsertAfter string
Selection.InsertBefore string
```

此处 string 是必需的字符串表达式，它包含希望在双引号中插入的文本。例如：

```

Selection.TypeText "Please come to the meeting next Friday at 9:00 A.M."
Selection.InsertBefore "Dr. "
Selection.InsertAfter vbCr & Address

```

使用 InsertAfter 方法或 InsertBefore 方法时, VBA 扩展所选内容以包括进插入的文本。

注意: 当选择了整个段落时, 所选内容包括段尾处的段落标记。所以, 添加到所选内容末尾的文本, 出现在下一段落的起始处, 而不是出现在被选段落的末尾。

在所选内容中插入段落

为了适当地安排文本, 可以插入一个段落:

- ◆要在当前所选内容处插入一个段落, 需使用 InsertParagraph 方法。
- ◆要在当前所选内容之后插入一个段落, 需使用 InsertParagraphAfter 方法。
- ◆要在当前所选内容之前插入一个段落, 需使用 InsertParagraphBefore 方法。

使用 Selection.TypeParagraph 命令, 也可以使 VBA 键入一个段落。

应用某种样式

要将某种样式应用于一个段落, 需设置 Paragraph 对象的 Style 属性:

```
Selection.Style = "Heading 3"
```

与此相似, 可以将字符样式应用于当前所选内容, 或者(如下例所示)应用于一些字或一些字符的特定区域:

```
myDocument.Paragraphs(1).Range.Words(3).Style = "Emphasis"
```

警告: 字符样式必须总是应用于某一个区域, 而不是直接应用于某一个段落。

扩展所选内容

要扩展所选内容, 需使用 Range 或 Selection 对象的 EndOf 方法。与 EndOf 方法对应的语法如下:

```
expression.EndOf(Unit, Extend)
```

此处, expression 是必需的表达式, 它返回一个 Range 或 Selection 对象, 例如: Characters、Words、Sentences 或 Paragraphs 集合中的一个对象。Unit 是可选的 Variant 参数, 它指定移动单位(如表 20.6 所示)。

表 20.6 EndOf 方法的移动单位

Unit	意义
wdCharacter	一个字符
wdWord	一个字(如省略该参数, 这是默认设置)
wdSentence	一个句子
wdLine	一行(这个单位只用于 Selection 对象, 不用于区域)
wdParagraph	一个段落

第 20 章 了解 Word 对象模型和重要对象 (续表)

Unit	意义
wdSection	文档的一节
wdStory	当前文字部分——例如文档的文字部分或页眉和页脚的文字部分
wdCell	表格中的一个单元格
wdColumn	表格中的一列
wdRow	表格中的一行
wdTable	整个表格

Extend 是可选的 Variant 参数，它指定是要移动还是要扩展所选内容或区域。wdMove 是移动所选内容或区域，这是默认设置；wdExtend 是扩展所选内容或区域。

例如，下述语句将当前所选内容扩展至段落结束处：

```
Selection.EndOf Unit:=wdParagraph, Extend:=wdExtend
```

下述语句将所选内容移动到段落结束处：

```
Selection.EndOf Unit:=wdParagraph, Extend:=wdMove
```

下述语句将当前所选内容扩展到当前 Word 文字部分结束处：

```
Selection.EndOf Unit:=wdStory, Extend:=wdExtend
```

要选择整个活动文档，需使用 ActiveDocument.Content.Select。这个命令与以交互方式工作时选择“编辑”>“全选”有相同的效果。

折叠所选内容

当完成了针对某个所选内容的工作之后，所选内容并不折叠成一个插入点，可能需要撤销对它的选择，让所选内容在过程结束时被折叠。将所选内容折叠到它的起始位置或结束位置的最容易办法是使用 Selection 对象的 Collapse 方法：

```
Selection.Collapse Direction:=wdCollapseStart  
Selection.Collapse Direction:=wdCollapseEnd
```

把所选内容的结束位置设置成等于它的起始位置（将所选内容折叠到它的起始位置），或者把所选内容的起始位置设置成等于它的结束位置（将所选内容折叠到它的结束位置），同样也可以把所选内容缩到仅为一个点：

```
Selection.End = Selection.Start  
Selection.Start = Selection.End
```

创建和使用 Range

在 Word 中，区域 (range) 是指文档中的连续区域，有确定的起始点和结束点。例如，如果要定义一个区域，它由某一个指定文档的前两段组成，则该区域的起始点在第一段的起始处，它的结束点在第二段的结束处（段落标记之后）。

Word VBA 中区域的典型应用与以交互方式使用 Word 工作时对书签的使用相类似：如

果希望能够很快地访问或很容易地操控文档中的某一处，则把该处的位置做上标记。同书签一样，一个区域可以包含文档中的任意多文本，从单个字符到文档全部内容。一个区域甚至可以有相同的起始点和结束点，这就说明区域内无内容，在效果上把该区域变成了文档中的一个不可见标志，用户可以使用它来插入文本。一旦创建了一个区域，便可以引用它，访问其内容，或在其中插入新内容，或为它指定格式等，所有这些都可以使用它的属性和应用于它的方法来达到。

提示：区域和书签的主要区别在于：区域的寿命仅限于定义它的 VBA 过程，而书签可以与包含它的文件或模块一起保存，所以可以在任何时间被访问（不管过程是否在运行）。

定义带名称的 Range

要创建一个 Range 对象，除了 Set 语句外，还要使用 Document 对象上的 Range 方法或者与某一个对象对应的 Range 属性——对象可以是 Selection 对象、Paragraphs 集合或 Paragraph 对象等。与使用 Range 方法对应的语法是：

```
Set RangeName = Document.Range(Start, End)
```

此处，RangeName 是指派给这个区域的名称，Start 和 End 是指定该区域起始点和结束点的可选参数。

与使用某一对象上的 Range 属性对应的语法是：

```
Set RangeName = object.Range
```

例如，下述语句使用 Paragraphs 集合的 Range 属性来定义名为 FirstPara、由活动文档的第一个段落组成的区域。这一语句没有使用 Start 和 End 参数，因为该段落的起始点和结束点已明确定义过了：

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

下述语句使文档起始处的前三个字变为由大写字母组成的字：

```
Dim InitialCaps As Range
Set InitialCaps = ActiveDocument.Range(Start:=ActiveDocument.Words(1).Start, _
End:=ActiveDocument.Words(3).End)
InitialCaps.Case = wdUpperCase
```

第一条语句定义一个名为 InitialCaps 的 Range 对象。第二条语句将 InitialCaps 指定到活动文档中的一个区域，从第一个字的起始处至第三个字的结束处。第三条语句使 InitialCaps Range 对象的内容更改为大写字母组成的字。

因为 InitialCaps 现在已定义为声明它的过程的持续期内的一个 Range 对象，所以如果愿意的话，以后在过程中可以返回到 InitialCaps 并操控它。

重定义 Range

要重新定义一个区域，使之引用到文档的另一部分，需使用 SetRange 方法。其语法是：

```
expression.SetRange(Start, End)
```

此处，expression 是必需的表达式，它返回一个 Range 对象或 Selection 对象，Start 和 End 是指定该区域的起始点和结束点的可选参数。

例如，下述语句重新定义区域 InitialCaps，使之引用文档的前两个字符：

```
InitialCaps.SetRange Start:=0, End:=2
```

也可以再次使用 Set 方法重新定义一个区域，或从头做起再创建一个区域来达到重新定义一个区域的目的。

使用 Duplicate 属性存储或复制格式设置

可以使用 Duplicate 属性来存储或复制某一区域，从而使它可以应用于另一区域。例如，下述语句先声明两个区域 Range1 和 Range2，将当前选定区域的复制副本存入 Range1，把活动文档中第一个书签的 Range 指定给 Range2，然后把 Range1 的内容应用于 Range2：

```
Dim Range1 As Range, Range2 As Range  
Set Range1 = Selection.Range.Duplicate  
Set Range2 = ActiveDocument.Bookmarks(1).Range  
Range2.Paragraphs(1).Range = Range2
```

设置 Options 对象

在过程中，常常有必要检查 Word 应用程序或某一特定文档中各选项的状态。在 VBA 中，很多选项都是受 Options 对象控制的，该对象有许多属性，但只有一个方法（即 SetW-PHelpOptions，它设置与 Word 的 WordPerfect 帮助特性对应的各选项）。

本节介绍四个设置选项的简单例子，三个例子使用 Options 对象，一个例子使用 Document 对象的一个属性。要看到与 Options 对象对应的可用属性的完整列表，请在 Word VBA 帮助文件中对“Options 对象”进行搜索。

确认超链接需用 Ctrl+单击

Word 文档中的超链接已被证明既有优点也有不足——尤其是 Microsoft 改变了 Word 处理超链接的方式，可能弄得用户不敢肯定，要跟随超链接，是单击它，还是按住 Ctrl 键并单击它。但是，如果将 Options 对象的 CtrlClickHyperlinkToOpen 属性设置为 True，就可以确信超链接需用 Ctrl+单击：

```
Options.CtrlClickHyperlinkToOpen = True
```

关断改写模式

为了确认过程的行为符合期望，可能有必要检查 Word 正在使用插入模式还是改写模式。在插入模式中，Word 在插入点处把键的字符插进来，插入点后的文本向右顺移。在改写模式中，键入的每个字符都取代插入点右边的字符。

改写模式受 Options 对象的 Overtype 属性控制。当 Overtype 属性设置为 True 时，改写模式有效；当 Overtype 属性设置为 False 时，插入模式有效。下述语句先将 Overtype 设置存入名为 blnOvertypeOn 的布尔型变量之中，再将 Overtype 设置为 False，实施其行动，

然后恢复用户的 Overtype 设置:

```
Dim blnOvertypeOn As Boolean
blnOvertypeOn = Options.Overtype
Options.Overtype = False
'take actions here
Options.Overtype = blnOvertypeOn
```

设置默认的文件路径

在计算机上配置 Word 时, 可能需要确认它的默认文件路径是否已被设置到正确的文件夹。使用 Options 对象的 DefaultFilePath 属性, 就可以做到这一点。其语法是:

expression.DefaultFilePath(Path)

此处, expression 是必需的表达式, 它返回一个 Options 对象。通常, 最容易的是使用 Options 对象本身。Path 是一种自解释列举型常量, 如下所示:

wdAutoRecoverPath	wdStyleGalleryPath
wdBorderArtPath	wdTempFilePath
wdCurrentFolderPath	wdTextConvertersPath
wdDocumentsPath	wdToolsPath
wdGraphicsFiltersPath	wdTutorialPath
wdPicturesPath	wdUserOptionsPath
wdProgramPath	wdUserTemplatesPath
wdProofingToolsPath	wdWorkgroupTemplatesPath
wdStartupPath	

例如, 下述语句设置用户模板路径和工作组模板路径:

```
Options.DefaultFilePath(wdUserTemplatesPath) = "c:\user\templates"
Options.DefaultFilePath(wdWorkgroupTemplatesPath) = "\\\server\users\templates"
```

关闭跟踪更改

如果一个过程要添加或删除文件, 或给文本指定格式, 在运行该过程之前, 可能有必要关闭“跟踪更改”特性, 从而使过程做出的更改不留痕迹。如果用户曾经让“跟踪更改”有效, 在过程结束处应将其转回到有效, 这样用户做出的更改就能再次被跟踪。下述例子将对应于 ActiveDocument 对象的 TrackRevisions 设置存入名为 blnTrackChangesOn 的布尔型变量之中, 再把 TrackRevisions 设置为 False, 实施其行动, 然后恢复用户的 TrackRevisions 设置:

```
Dim blnTrackChangesOn As Boolean
blnTrackChangesOn = ActiveDocument.TrackRevisions
ActiveDocument.TrackRevisions = False
'take actions here
ActiveDocument.TrackRevisions = blnTrackChangesOn
```

通过本章的学习，读者将能够掌握如何使用 Word 对象模型来完成各种操作。例如，可以使用 Find 对象进行查找和替换操作，使用 Headers 和 Footers 对象插入页眉和页脚，使用 PageNumbers 对象插入页码，使用 Sections 对象插入节，使用 PageSetup 对象设置页面格式，使用 Windows 对象显示或隐藏窗口，使用 Views 对象切换视图，以及使用 Tables 对象插入表格。

第 21 章 使用 Word 中广泛使用的对象进行工作

- ◆ 在 VBA 中使用 Find 对象和 Replacement 对象

- ◆ 使用 Headers、Footers 和 PageNumbers 进行工作

- ◆ 使用 Sections、PageSetup、Windows 和 Views 进行工作

- ◆ 使用 Tables 进行工作

在上一章中学习了如何使用 Word 对象模型中某些主要的对象（如 Document 对象、Selection 对象、Range 对象和 Options 对象）进行工作。本章将进一步学习如何在 Word 中使用 VBA 进行工作，包括使用 Find 和 Replacement、HeadersFooters 和 PageNumbers、Sections、PageSetup、Windows 和 Views，以及 Tables。

在 VBA 中使用 Find 对象和 Replacement 对象

Word 的“查找”和“替换”特性在过程中是非常有用的。要通过 VBA 来访问这些特性，需使用 Find 和 Replacement 对象。本节介绍使用 Find 对象的最容易的方法，即通过使用它的 Execute 方法来使用它。这样做时，通常要把与查找操作对应的参数设置为 Execute 语句中的参数，但是也可以在使用各属性之前设置参数。

表 21.1 列出了 Find 对象的属性，从中可以找出用于常见的搜索操作的最有用的属性。

表 21.1 Find 对象的属性

Find 属性	意义
Font	正在搜索（在指定的文本上或在一个空字符串上）的字体格式设置
Forward	是布尔型参数，它指定在文档中向前搜索（True），还是向后搜索（False）
Found	是布尔型属性。如搜索时找到了匹配项则为 True，没有找到则为 False
Highlight	是长整型参数，它控制在替换文本的格式设置中，要包括突出显示（True）还是不包括突出显示（False）
MatchAllWordForms	是布尔型属性——True 或 False——对应于“查找单词的各种形式”复选框
MatchCase	是布尔型属性，对应于“区分大小写”复选框
MatchSoundsLike	是布尔型属性，对应于“同音”复选框
MatchWholeWord	是布尔型属性，对应于“全文匹配”复选框
MatchWildcards	是布尔型属性，对应于“使用通配符”复选框
ParagraphFormat	正在搜索（在指定的文本上或在一个空字符串上）的段落格式设置
Replacement	返回包含替换操作所需条件的 Replacement 对象
Style	搜索文本的样式。通常希望使用当前模板中某一样式的名称，但也可以使用某个内置 Word 常量样式名称，例如 wdStyleHeading1
Text	正在搜索的文本（在“查找和替换”对话框中“查找内容”框内输入的内容）。使用空字符串则仅搜索格式设置
Wrap	是长整型属性，它控制：当从文档的非开头处搜索到文档末尾时，或在一个区域内搜索到所选范围的末尾时（以上为向前搜索），或者当从文档的非末尾处搜索到文档开头时，或在一个区域内搜索到所选范围的开头时（以上为向后搜索），搜索是否继续进行

使用 Replacement 对象来指定替换操作中的替换条件。Replacement 对象具有如下属性：Font、Highlight、ParagraphFormat、Style 和 Text，它们与 Find 对象的属性相对应。

了解对应于 Execute 方法的语法

对应于 Execute 方法的语法如下：

```
expression.Execute(FindText, MatchCase, MatchWholeWord, MatchWildcards,
  MatchSoundsLike, MatchAllWordForms, Forward, Wrap, ReplaceWith, Replace)
```

这一语句的各部分说明如下：

- ◆ expression 是必需的表达式，它返回一个 Find 对象。通常，使用 Find 对象本身是最容易的。

- ◆ FindText 是可选的 Variant 参数，它指定要搜索的文本。尽管这个参数是可选的，但几乎总是要指定它，即使只是指定一个空字符串（" "）来搜索格式设置。（如果不指定 FindText，会有偶而搜索以前的项目的危险。）使用以交互方式工作时用的特殊字符（例如^P 用做段落标记，^a 用做注释），可以对特殊字符进行搜索。使用常规通配符，可以对通配符进行搜索。为了使用通配符工作，必须将 MatchWildcards 设置为 True。输入~和 0，后随字符代码，可以搜索一个符号。例如，要搜索双引号，可以指定~0148，因为它的字符代码是 148。

- ◆ MatchCase 是可选的 Variant 参数，要使搜索区分大小写，可将它设置为 True。

- ◆ MatchWholeWord 是可选的 Variant 参数，要限制搜索是查找完整的单词而不是包含在其他单词之中的单词，可将它设置为 True。

- ◆ MatchWildcards 是可选的 Variant 参数，要在搜索中使用通配符，可将它设置为 True。

- ◆ MatchSoundsLike 是可选的 Variant 参数，要使 Word 查找发音与指定的查找项目（单词）相近的单词，可将它设置为 True。

- ◆ MatchAllWordForms 是可选的 Variant 参数，要使 Word 查找指定的查找项目（单词）的各种形式（例如同一动词或名词的各种不同形式），可将它设置为 True。

- ◆ Forward 是可选的 Variant 参数，要使 Word 向前搜索（从文档的开头搜索至末尾），可将它设置为 True；向后搜索则将它设置为 False。

- ◆ Wrap 是可选的 Variant 参数，它控制：当从文档的非开头处搜索至文档末尾时，或在一个区域内搜索到达所选范围的末尾时（以上为向前搜索），或者，当从文档的非末尾处搜索到文档开头时，或在一个区域内搜索到达所选范围的开头时（以上为向后搜索），搜索是否继续进行。Word 为 Wrap 提供了如下选项：

常量	值	意义
wdFindAsk	2	Word 搜索所选内容或区域——或从插入点搜索到文档的末尾或开头——然后显示消息框，提示用户确定是否要搜索文档的其余部分
wdFindContinue	1	在到达搜索范围的末尾或开头之后，或到达文档的末尾或开头之后，Word 继续搜索
wdFindStop	0	到达搜索范围的末尾或开头时，或到达文档的末尾或开头时，Word 使查找操作停止

- ◆ Format 是可选的 Variant 参数，要使搜索操作在指定的文本的同时，可查找其格式设置，或仅查找格式设置，可将它设置为 True。
- ◆ ReplaceWith 是可选的 Variant 参数，它指定替换文本。可以使用空字符串做替换，以便简单地去掉 FindText 文本；也可以使用特殊字符做替换以对应 FindText 参数。要使用图形对象，则要将它复制到剪贴板，然后指定 ^c（剪贴板的内容）。

注意：如前所述，要使用图形对象，需在文本层之中使用（不是浮移在文本上方）。

如果图形浮移在文本上方，^c 会粘贴进前次的文本内容中。

- ◆ Replace 是可选的 Variant 参数，它控制查找操作要做出多少替换：一个（wdReplaceOne）、全部（wdReplaceAll）或不替换（wdReplaceNone）。

使用 ClearFormatting 方法

使用 Find 对象和 Replacement 对象时，常常需要使用 ClearFormatting 方法，它清除“查找内容”框或“替换为”框内指定的格式设置。使用 ClearFormatting 方法，与焦点在“查找内容”框或“替换为”框时单击“不限定格式”按钮有相同的效果。下述语句（出现在 With 结构之内）分别从 Find 和 Replacement 对象中清除格式设置：

```
With ActiveDocument.Content.Find
    .ClearFormatting
    .Replacement.ClearFormatting
End With
```

使查找和替换投入工作

使用查找和替换的最简单的方法是指定 Execute 语句中必要的参数，而不必指定的可选参数则予以忽略。例如，要把活动文档中所有段落标记对替换掉，可以使用如下语句，搜索 ^p^p，并以 ^p 来替换 ^p^p：

```
ActiveDocument.Content.Find.Execute FindText:="^p^p", ReplaceWith:="^p",
    Replace:=wdReplaceAll
```

在循环中运行这来语句，可以把文档中所有多余的段落标记对替换掉。

也可以使用 With 语句来指定对应于查找和替换操作的各属性，程序清单 21.1 就是这种例子，它将名为 Example.doc 的已打开文档中所有粗体字格式设置更改为斜体字格式设置。

程序清单 21.1

```
1. With Documents("Example.doc").Content.Find
2.     .ClearFormatting
3.     .Font.Bold = True
4.     With .Replacement
5.         .ClearFormatting
6.         .Font.Bold = False
7.         .Font.Italic = True
8.     End With
9.     .Execute FindText:"", ReplaceWith:"", _
        Format:=True, Replace:=wdReplaceAll
10.    End With
```

此处，行 1 以带有 Find 对象的 With 语句开头，指明要对其进行工作的 Document 对象（是 Documents 集合中的 Example. doc）。行 2 使用 ClearFormatting 方法从 Find 对象中清除格式设置，行 3 将它的 Font 对象的 Bold 属性设置为 True。

行 4 至行 8 包括与 Replacement 对象对应的嵌套 With 语句。行 5 使用 ClearFormatting 方法从 Replacement 对象中清除格式设置；行 6 将它的 Bold 属性设置为 False；行 7 将它的 Italic 属性设置为 True。

然后，行 9 使用 Execute 方法来执行替换操作。这里的 FindText 和 ReplaceWith 都被指定为空字符串，以便使 Word 仅工作于格式设置；将 Format 设置为 True 以激活 Find 和 Replacement 对象中的格式设置；将 Replace 设置为 wdReplaceAll，以便以斜体字格式设置来替换粗体字格式设置的所有实例。

行 10 结束外层的 With 语句。

使用 Headers、Footers 和 PageNumbers 进行工作

本节将说明如何在 Word 文档中针对页眉和页脚来进行工作，也要学习如何使用 VBA 来操控页码。很多页眉和页脚都需要包括页码。

了解 VBA 如何构成页眉和页脚

一个 Word 文档可以包含六种能创建的页眉和页脚——基本的页眉和页脚，不同的第一页页眉和页脚，不同的偶数页页眉和页脚——如果需要的话，还有与文档中每一节对应的页眉和页脚的不同组合。创建的每一个文档都得到一个基本页眉和一个基本页脚，即使不在它们里面放任何东西。然后，改变与节对应的“页面设置”选项，可以创建不同的第一页和偶数页页眉。

VBA 使用如下的与页眉和页脚对应的对象：

- ◆ 在 HeaderFooter 对象中既有页眉又有页脚，通过 Headers 属性来访问页眉，通过 Footers 属性来访问页脚。
- ◆ HeadersFooters 集合包含文档任一给定节内所有的 HeaderFooter 对象。因为文档中每一节都可以有与其他各节不同的页眉和页脚，所以在节内可以接触到任何给定的页眉或页脚。
- ◆ 为了返回 HeadersFooters 集合，需使用相应的 Document 对象中的适当 Section 对象的 Headers 属性和 Footers 属性。也可以使用 Selection 对象的 HeaderFooter 属性来返回单个 HeaderFooter 对象，但是这种方法在使用中局限性更大。
- ◆ 通过 HeaderFooter 对象可以访问到 Range 对象、Shapes 集合以及 PageNumbers 集合。

接触到页眉和页脚

在文档中适当的节内，可以访问到页眉或页脚。例如，下述语句显示一个消息框，其中有已打开文档 Transfer. doc 的第二节中第一页页脚的文本：

```
MsgBox Documents("Transfer.doc").Sections(2).  
Footers(wdHeaderFooterFirstPage).Range.Text
```

下述语句声明 HeaderFooter 对象变量 myHeader，并把活动文档中第一节内的基本页眉指定给它：

```
Dim myHeader As HeaderFooter
Set myHeader = ActiveDocument.Sections(1).Headers(wdHeaderFooterPrimary)
```

检查页眉或页脚是否存在

Word 在创建的每个文档中生成一个基本页眉和一个基本页脚，所以这些对象总是存在的。要查找其他页眉或页脚是否存在，需检查应用程序的 HeaderFooter 对象的 Exists 属性。下述语句在活动文档中挨个检查每一节里是否有偶数页页脚存在，并生成一个典型的页眉（含有节号和文档全名），使用名为 Even Footer 的样式（在大多数 Word 文档中，它是默认存在的）来设置其格式。

```
Dim cSection As Section
With ActiveDocument
    For Each cSection In .Sections
        cHeader = cSection.Headers(wdHeaderFooterEvenPages)
        If Not cSection.Headers(wdHeaderFooterEvenPages).Exists Then
            cSection.PageSetup.OddAndEvenPagesHeaderFooter = True
            cSection.Headers(wdHeaderFooterEvenPages).Range.Text =
                "Section " & cSection.Index & " of " & .FullName
            cSection.Headers(wdHeaderFooterEvenPages).Range.Style =
                "Even Footer"
        End If
    Next cSection
End With
```

与前一节的页眉或页脚相链接

按照默认方式，Word 使第一节之后的每一节的页眉和页脚与前一节的页眉和页脚相链接。要排除此链接，需将页眉或页脚的 LinkToPrevious 属性设置为 False；要建立此链接，需将这一属性设置为 True。下述语句使得活动文档第三节中的基本页脚与第二节中的相应页脚失去链接：

```
ActiveDocument.Sections(3).Footers(wdHeaderFooterPrimary).LinkToPrevious = False
```

生成不同的第一一页页眉

要使某一节第一页上的页眉与其他页不同，需将与该节对应的 PageSetup 对象的 DifferentFirstPageHeaderFooter 属性设置为 True。下述语句检查活动文档的第十节上是否有这种第一页页眉，如果没有，就生成一个：

```
With ActiveDocument.Sections(10)
    If .Headers(wdHeaderFooterFirstPage).Exists = False Then
        .PageSetup.DifferentFirstPageHeaderFooter = True
End With
```

生成不同的奇数页页眉和偶数页页眉

为了使文档的奇数页和偶数页有不同的页眉（但不同于第一页），可以生成一个偶数页页眉。按照默认方式，在生成偶数页页眉之前，奇数页上和偶数页上出现的都是基本页眉。在偶数页页眉生成时，基本页眉便成了奇数页页眉。

和生成第一页页眉一样，要使用 PageSetup 对象来生成不同的偶数页页眉，即如下述语句所示，把该对象的 OddAndEvenPagesHeaderFooter 属性设置为 True：

```
ActiveDocument.Sections(1).PageSetup.OddAndEvenPagesHeaderFooter = True
```

提示：如果编写程序来为文档设置格式，常常需要检查或更改文档中所有页眉和页脚。要做到这一点，最容易的办法是使用两个 For Each...Next 循环，外层循环在 Sections 集合中的每个 Section 对象内工作，而内层循环在节内的 HeadersFooters 集合中的每个 HeaderFooter 对象之内工作。

将页码添加到页眉和页脚

文档的页眉或页脚常常含有页码：以简洁格式表示的简单号码（1, 2, 3, 等等），或者复杂一点的号码，在号码内指明章和页，以分隔符隔开。

VBA 在 PageNumbers 集合内构成页码，使用文档相应节内适当的 HeaderFooter 对象的 PageNumbers 属性便可返回 PageNumbers 集合。

将页码添加到文档的一节或若干节

要将页码添加到文档中，可使用与文档的相应节对应的 PageNumbers 集合所具有的 Add 方法。

对应于 Add 方法的语法表示如下：

```
expression.Add PageNumberAlignment, FirstPage
```

此处，expression 是必需的表达式，它返回一个 PageNumbers 集合。通常使用 PageNumbers 集合本身。

PageNumberAlignment 是可选的 Variant 参数，它指定被添加的页码的对齐方式。表 21.2 列出了可用的常量和值。

表 21.2 PageNumberAlignment 常量和值

常量	值	产生的对齐方式
wdAlignPageNumberLeft	0	左对齐
wdAlignPageNumberCenter	1	居中
wdAlignPageNumberRight	2	右对齐（默认）
wdAlignPageNumberInside	3	内部边距（左页右对齐，右页左对齐）
wdAlignPageNumberOutside	4	外部边距（左页左对齐，右页右对齐）

FirstPage 是可选的 Variant 参数，将它设置为 False，可以使第一页上的页眉和页脚不含页码。如果省略了 FirstPage 参数，需由 PageSetup 对象的 DifferentFirstPageHeader-

Footer 属性来控制第一页上的页眉和页脚与节内其他页是否相同。

PageNumberAlignment 参数和 FirstPage 参数都是可选参数，但是，通常希望至少要指定 PageNumberAlignment 参数。

下面的子过程把页码添加到文档每一节内的所有页眉上，此时需使用两个 For Each... Next 循环：

```
Sub AddPageNumbersToAllHeadersAndSections()
    Dim cHeader As HeaderFooter, cSection As Section
    With Documents("Headers and Footers.doc")
        For Each cSection In .Sections
            For Each cHeader In cSection.Headers
                cSection.Headers(wdHeaderFooterPrimary).PageNumbers _ 
                    .Add.PageNumberAlignment:= _
                    wdAlignPageNumberRight, FirstPage:=True
            Next cHeader
        Next cSection
    End With
End Sub
```

从文档的一节或若干节中除去页码

要从页面上除去页码，需指定 PageNumber 对象，并使用 Delete 方法。下面的子过程从活动文档的当前节中除去每个 PageNumber 对象：

```
Sub RemovePageNumbersFromCurrentSection()
    Dim ThisHeader As HeaderFooter
    Dim ThisPageNumber As PageNumber
    With Selection.Sections(1)
        For Each ThisHeader In .Headers
            For Each ThisPageNumber In ThisHeader.PageNumbers
                ThisPageNumber.Delete
            Next ThisPageNumber
        Next ThisHeader
    End With
End Sub
```

找出文档中某一节是否有页码

要找出任何给定的页码是否存在，最容易的方法是检查与相应节对应的 PageNumbers 集合的 Count 属性。例如，如果页眉上还没有页码的话，下述语句将居中的页码添加到当前节中的偶数页页眉上：

```
If Selection.Sections(1).Headers(wdHeaderFooterEvenPages) _ 
    .PageNumbers.Count = 0 Then Selection.Sections(1) _ 
    .Headers(wdHeaderFooterEvenPages).PageNumbers.Add _ 
    PageNumberAlignment:=wdAlignPageNumberCenter
```

更改某一节的页码编排

要更改某一节的页码编排，需针对 StartingNumber 属性进行工作，并按照需要使用 RestartNumberingAtSection 属性、IncludeChapterNumber 属性和 ChapterPageSeparator 属性。

StartingNumber 属性是长整型属性，当 RestartNumberingAtSection 属性设置为 True 时，它包含该节的起始页码。当 RestartNumberingAtSection 属性设置为 False 时，StartingNumber 返回 0。下述语句设置流动文档第四节中与基本页眉对应的页码编排，如果当前并未指定起始页码的话，则页码从 55 开始：

```
With ActiveDocument.Sections(4).Headers(wdHeaderFooterPrimary)
    If .PageNumbers.StartingNumber = 0 Then
        .PageNumbers.RestartNumberingAtSection = True
        .PageNumbers.StartingNumber = 55
    End If
End With
```

要把章号添加到页码，需在文档中使用标题号码编排，此时需将 IncludeChapterNumber 属性设置为 True，然后指定要使用的分隔符（例如，wdSeparatorEnDash 对应于一个短破折号）：

```
With ActiveDocument.Sections(4).Headers(wdHeaderFooterPrimary)
    .PageNumbers
    .IncludeChapterNumber = True
    .ChapterPageSeparator = wdSeparatorEnDash
End With
```

使第一页不显示页码

为使第一页不显示页码，需将相应节内适当的 HeaderFooter 对象的 ShowFirstPageNumber 属性设置为 False：

```
ActiveDocument.Sections(1).Footers(wdHeaderFooterPrimary).PageNumbers_
    .ShowFirstPageNumber = False
```

为页码设置格式

为页码设置格式有两个方面：设置页码显示的格式（例如，以常规的阿拉伯数字来显示或以小写的罗马数字来显示）和设置以此格式显示时使用的字体。

为了选择页码以何种格式来显示，要设置所涉及的 PageNumbers 集合的 NumberStyle 属性。例如，下述语句设置流动文档第四节中基本页眉上的页码以小写字母来显示：

```
ActiveDocument.Sections(4).Headers(wdHeaderFooterPrimary)_
    .PageNumbers.NumberStyle = wdPageNumberStyleLowercaseLetter
```

一旦页码已在页眉或页脚中，就可以用任一种方法来为页码设置格式。为给定页码设置格式时选用字体的一种简易方法是使用 Select 方法来选择 PageNumber 对象，然后将格式设置应用于它，就像用在其他选择内容上一样，见下述语句：

```
ActiveDocument.Sections(4).Headers(wdHeaderFooterPrimary)_
    .PageNumbers(1).Select
    With Selection.Font
        .Name = "Impact"
        .Size = 22
        .Bold = True
    End With
```

生成“Page X of Y”页码

也可以在页眉或页脚中使用 Word 的域代码进行页码编排。当希望以“X of Y”的编号方法——如“Page 168 of 192”(192 页中的第 168 页)之类——编排页码时，这种技术特别有用。下述语句为活动文档最后一节选用基本页眉，使用居中对齐方式，然后输入文本和字段，以便生成这一类型的页码编排：

```
ActiveDocument.Sections(ActiveDocument.Sections.Count)
    .Headers(wdHeaderFooterPrimary).Range.Select
With Selection
    .Paragraphs(1).Alignment = wdAlignParagraphCenter
    .TypeText Text:="Page"
    .Fields.Add Range:=Selection.Range, Type:=wdFieldEmpty, Text:="PAGE", PreserveFormatting:=True
    .TypeText Text:=" of "
    .Fields.Add Range:=Selection.Range, Type:=wdFieldEmpty, Text:="NUMPAGES", PreserveFormatting:=True
End With
```

如果以此方法使用一个字段来插入一个页码，仍然可以使用适当的 PageNumber 对象来访问页码。(在本例中，PageNumber 对象包括 PAGE 字段，而不包括 NUMPAGES 字段。)

使用 Sections、PageSetup、Windows 和 Views 进行工作

在默认情况下，每个 Word 文档至少有一节，而当文档的内容和版式有需要时，可以包含多节。文档的节控制了页面的版式，如果必要的话，一个文档的不同节里可以使用不同的页面版式。

给文档添加一节

要给某一个文档添加一节，可以使用针对 Sections 集合的 Add 方法，或者使用针对 Range 或 Selection 对象的 InsertBreak 方法。

Add 方法使用如下语法：

```
expression.Add Range, Start
```

此处，expression 是必需的表达式，它返回一个 Sections 集合。Range 是可选的 Variant 参数，它指定在其之前插入分节符的区域。(如果省略 Range，VBA 将分节符插至文档末尾处。) Start 是可选的 Variant 参数，用来指定要插入的分节符的类型：

- ◆ wdSectionContinuous (0)，对应于一个连续分节符。
- ◆ wdSectionEvenPage (3)，对应于一个偶数页分节符。
- ◆ wdSectionOddPage (4)，对应于一个奇数页分节符。
- ◆ wdSectionNewColumn (1)，对应于一个新列分节符。
- ◆ wdSectionNewPage (2)，默认值，对应于一个新页分节符。

下述语句将一个新页添加至活动文档，并把它放在第二段之前：

```
ActiveDocument.Sections.Add Range:=.Range(Start:=.Paragraphs(2).Range.Start,
End:=.Paragraphs(2).Range.Start), Start:=wdSectionNewPage
```

InsertBreak 方法的语法如下：

expression.InsertBreak Type

此处，expression 是必需的表达式，它返回一个 Selection 或 Range 对象。Type 是可选的 Variant 参数，它指定要插入的分节符的类型：

- ◆ wdSectionBreakNextPage (2)，对应于一个新页分节符。
- ◆ wdSectionBreakContinuous (3)，对应于一个连续分节符。
- ◆ wdSectionBreakEvenPage (4)，对应于一个偶数页分节符。
- ◆ wdSectionBreakOddPage (5)，对应于一个奇数页分节符。
- ◆ wdColumnBreak (8)，对应于一个新列分节符。

下述语句在活动文档的第二段之前插入一个连续分节符：

```
ActiveDocument.Paragraphs(2).Range.InsertBreak Type:=wdSectionBreakContinuous
```

更改页面设置

为了更改一个文档或一节的页面设置，可以使用应用程序的 Document 对象或 Section 对象的 PageSetup 对象进行工作。例如，下述语句是针对名为 Planning. doc 的文档的 PageSetup 对象进行工作的，设置纸张大小、文本方向、边距（以点计）等：

```
With Documents("Planning.doc").PageSetup
    .PaperSize = wdPaperLetter
    .Orientation = wdOrientPortrait
    .TopMargin = 1
    .BottomMargin = 1
    .LeftMargin = 1
    .RightMargin = 1.5
    .MirrorMargins = True
End With
```

打开含有打开文档的新窗口

要打开含有打开文档的新窗口，需使用 Add 方法，它的语法很简洁：

expression.Add window

此处，expression 是一个表达式，它返回一个 Windows 集合；而 window 是可选的 Variant 参数，它指定含有某个文档的窗口，正是为了这个文档，才想要打开一个新窗口。如果省略 window，VBA 会为活动文档打开一个新窗口。

注意：有两个 Windows 集合：一个与应用程序相对应，一个与显示工作时用到的文档的窗口相对应。如果需要针对同一文档打开多个窗口（如同使用“窗口”>“新建窗口”命令所能做到的那样），对于 Document 对象的 Windows 集合会是很有用的，但是通常希望使用与 Application 对象对应的 Windows 集合。Windows 是可创建的对象，所以不必指定 Application 对象。

例如，下述语句将一个新窗口指定给变量 myWindow，这个新窗口是为第一个窗口打开的，而第一个窗口是为活动文档打开的：

```
Dim myWindow As Window
Set myWindow = Windows.Add(Window:=ActiveDocument.Windows(1))
```

关闭除第一个窗口之外的所有与活动文档对应的窗口

有些时候，为某一个文档打开一个或几个新窗口是有用的。如果这样做了，为了给自己提供更多的机动空间，用户迟早都得关闭所有的次级窗口。下述语句关闭除第一个窗口之外的所有与活动文档对应的窗口：

```
Dim myWin As Window, myDoc As String
myDoc = ActiveDocument.Name
For Each myWin In Windows
    If myWin.Document = myDoc Then
        If myWin.WindowNumber <> 1 Then myWin.Close
    Next myWin
```

拆分窗口

要沿水平方向将一个窗口拆分为两部分，需将其 Split 属性设置为 True。要指定拆分百分比（它控制，按垂直方向量度，拆分者置于窗口下多远），需设置 SplitVertical 属性。下述语句拆分活动窗口，使顶部窗格高度占拆分前窗口高度的 70%：

```
With ActiveWindow
    .Split = True
    .SplitVertical = 70
End With
```

要从窗口中移去拆分者，需将 Split 属性设置为 False：

```
ActiveWindow.Split = False
```

显示与窗口对应的文档结构图

为了以文档结构图原先宽度（占文档窗口）的某个百分比来显示与窗口对应的文档结构图，可将 DocumentMap 属性设置为 True：

```
ActiveWindow.DocumentMap = True
```

要以另一个不同宽度来显示文档结构图，或者要改变文档结构图的宽度，需将 DocumentMapPercentWidth 属性设置为窗口宽度的一个适当百分比：

```
ActiveWindow.DocumentMapPercentWidth = 25
```

要重新隐藏文档结构图，需将 DocumentMap 属性设置为 False，或将 DocumentMapPercentWidth 属性设置为 0。

滚动窗口

要使一个窗口上滚、下滚、左滚或右滚，需使用 LargeScroll 方法或 SmallScroll 方法。

LargeScroll 方法类似于在窗口的水平或垂直滚动条上的滚动块前后单击，它使窗口的

内容在指定方向上滚动一“屏”或多“屏”。SmallScroll 方法类似于在滚动条上的滚动按钮上单击，它使窗口的内容上滚或下滚一行，及左滚或右滚一个很小的滚动增量的距离。

与 LargeScroll 方法对应的语法是：

```
expression.LargeScroll(Down, Up, ToRight, ToLeft)
```

与 SmallScroll 方法对应的语法差不多是一样的：

```
expression.SmallScroll(Down, Up, ToRight, ToLeft)
```

此处，expression 是必需的表达式，它返回一个 Window 对象。Down、Up、ToRight 和 ToLeft 是可选的 Variant 参数，它们指定在它们的名称所指示的方向上滚动窗口内容的“屏”数（对于 LargeScroll）、行数或水平移动单位数（对于 SmallScroll）。

下述语句使活动窗口上滚两“屏”：

```
ActiveWindow.LargeScroll Up:=2
```

排列多个窗口

为了排列多个窗口，需使用 Arrange 方法。与 Arrange 方法对应的语法是：

```
expression.Arrange ArrangeStyle
```

此处，expression 是一个表达式，它返回一个 Windows 集合；而 ArrangeStyle 是可选的 Variant 参数，它指定如何排列窗口：按图标排列各窗口（wdIcons，1）或平铺各窗口（wdTiled，0）。默认设置为 wdTiled。

例如，下述语句平铺所有打开的窗口：

```
Windows.Arrange ArrangeStyle:=wdTiled
```

定位窗口和调整窗口的大小

要定位一个窗口，需设置它的 Left 和 Top 属性。例如：

```
ActiveWindow.Left = 100
```

```
ActiveWindow.Top = 200
```

要调整窗口的大小，需设置它的 Height 和 Width 属性：

```
With ActiveWindow
```

```
    .Height = 300
```

```
    .Width = 400
```

```
End With
```

要最大化、最小化或“恢复”一个窗口，需将它的 WindowState 属性分别设置为 wdWindowStateMaximize、wdWindowStateMinimize 或 wdWindowStateNormal。如果某一窗口已被最小化，下述语句使包含名为 Example.doc 的文档的该窗口最大化：

```
With Documents("Example.doc").Windows(1)
    If .WindowState = wdWindowStateMinimize Then _
        .WindowState = wdWindowStateMaximize
End With
```

确认项目已在窗口内显示

在打开或排列了窗口之后，常常需要确认希望用户看到的项目——一个区域，某个文本，一个图形或其他形状，或者一个字段——是否已在窗口内显示。要做到这一点，最容易的方法是使用 Window 对象的 ScrollIntoView 方法。这一方法移动视图，但不移动所选内容，所以如果需要将所选内容移到某一区域，必须单独移动它。

ScrollIntoView 方法使用如下语法：

```
expression.ScrollIntoView(Obj, Start)
```

此处，expression 是必需的表达式，它返回一个 Window 对象。Obj 是必需的参数，它指定一个 Range 或 Shape 对象。Start 是可选的布尔型参数，将它设置为 True（默认值），就使区域或形状的左上角被显示；将它设置为 False，则使右下角被显示。当需要确认可能会大于窗口的某一区域或形状的结束处被显示时，可将 Start 设置为 False。

下述语句将所选内容定位于活动文档中第一个列表最后一段的结束处，准备把一个新段落加至该表：

```
Dim rngFirstList As Range  
Set rngFirstList = ActiveDocument.Lists(1).Range  
  
ActiveDocument.Windows(1).ScrollIntoView Obj:=rngFirstList, Start:=False  
rngFirstList.Select  
Selection.Collapse Direction:=wdCollapseEnd  
Selection.MoveLeft Unit:=wdCharacter, Count:=-1, Extend:=wdMove
```

改变文档的视图

为了改变文档的视图，需将对应于适当窗口的 View 对象的 Type 属性设置为 wdMasterView、wdNormalView、wdOutlineView、wdPrintPreview、wdPrintView、wdReadingView 或 wdWebView。例如，下述语句将对应于 Sample. doc 的视图改变到页面视图：

```
Documents("Sample.doc").Windows(1).View.Type = wdPrintView
```

放大视图以显示多个页面

为了放大“页面”视图或“打印预览”以显示多个页面，需设置适当的 View 对象的 PageColumns 和 PageRows 属性。（如有必要，首先要改变该视图。）下述语句在“页面”视图中显示 Sample. doc，使六个页面被显示（横向三个纵向两个）：

```
With Documents("Sample.doc").Windows(1).View  
    .Type = wdPrintView  
    With .Zoom  
        .PageColumns = 3  
        .PageRows = 2  
    End With  
End With
```

使用表格进行工作

示显内口窗态与目取好部

常常需要在 Word 文档中使用表格进行工作，可以从头开始创建表格，或者利用已有的表格来操作。VBA 使用 Table 对象来表示一个表格，若干 Table 对象汇集在一起就组成了 Tables 集合。要使用表格来工作，需使用 Tables 属性以返回与所涉及的 Document、Range 或 Selection 对象相对应的 Tables 集合。

Tables 集合和 Table 对象又包含如下集合和对象：

- ◆ Rows 集合，它包含表格中的各行。每行又以一个 Row 对象来表示。
- ◆ Columns 集合，它包含表格中的各列。每列又以一个 Column 对象来表示。
- ◆ Cell 对象，它提供对出自 Table 对象的指定单元格的直接访问。通过浏览各单元格驻留的行或列，也可以到达表格中的各单元格。
- ◆ Range 对象，它提供对表格内各区域的访问。
- ◆ Borders 集合，它包括与表格对应的所有边框。
- ◆ Shading 对象，它包括与表格对应的所有底纹格式。

创建表格

要从头开始创建一个新表格（不是将已有文本转换为表格），需使用 Tables 集合的 Add 方法。Tables 集合的 Add 方法的语法如下：

```
expression.Add(Range, NumRows, NumColumns, DefaultTableBehavior, AutoFitBehavior)
```

各参数说明如下：

- ◆ expression 是必需的表达式，它返回一个 Tables 集合。特别希望使用与适当文档对应的 Tables 集合。
- ◆ Range 是必需的参数，它指定要插入表格的区域。如果区域是一个选定内容（不是被折叠），该表格就取代此区域。
- ◆ NumRows 是必需的长整型参数，它指定表格将要有的行数。
- ◆ NumColumns 是必需的长整型参数，它指定表格将要有的列数。
- ◆ DefaultTableBehavior 是可选的 Variant 参数，它指明：当改变列中内容或窗口宽度时，表格是否能够自动调整它的各列的大小，以便与列中内容或窗口相适应。使用 wdWord9TableBehavior 可使表格自动调整各列，而使用 wdWord8TableBehavior（默认值）则使得各列保持其宽度。
- ◆ AutoFitBehavior 是可选的 Variant 参数，它指定与表格对应的自动调整规则。这个参数只是在 DefaultTableBehavior 为 wdWord9TableBehavior 时使用。使用 wdAutoFitContent 会调整各列以适应其内容，使用 wdAutoFitWindow 会调整各列以适应窗口宽度，而 wdAutoFitFixed 则是使用固定列宽。

例如，下述语句在活动文档中插入点的当前位置处，插入一个 10 行 5 列的新的、空白的、无自动调整能力的表格：

```
ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=10, _  
NumColumns:=5, DefaultTableBehavior:=wdWord8TableBehavior
```

选择表格

为了选择某一表格，先要指定所涉及的 Document、Range 或 Selection 对象，然后指明 Table 对象，并使用 Select 方法。这个方法不用参数。

下述语句选择活动文档中的第一个表格：

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

下述语句先声明变量 tempTable，然后选择名为 Log.doc 的文档中的第一个表格，并将它的 Range 对象指派给 tempTable：

```
Dim tempTable  
Documents("Log.doc").Tables(1).Select  
Set tempTable = Selection.Tables(1).Range
```

下述语句选择名为 tempRange 的区域中的第二个表格：

```
tempRange.Tables(2).Select
```

下述语句选择当前所选内容中的第一个表格：

```
Selection.Tables(1).Select
```

将文本转换为表格

为了将文本转换为一个表格（与从头开始创建一个新表格不同），可以使用针对适当的 Range 或 Selection 对象的 ConvertToTable 方法。ConvertToTable 方法使用如下语法：

```
expression.ConvertToTable(Separator, NumRows, NumColumns, InitialColumnWidth,  
Format, ApplyBorders, ApplyShading, ApplyFont, ApplyColor, ApplyHeadingRows,  
ApplyLastRow, ApplyFirstColumn, ApplyLastColumn, AutoFit, AutoFitBehavior,  
DefaultTableBehavior)
```

各参数说明如下：

- ◆ expression 是必需的参数，它指定一个表达式，该表达式返回一个 Range 对象或 Selection 对象。
- ◆ Separator 是可选的 Variant 参数，它指定用来标记列信息在何处分隔的分隔符。可以使用与分隔符对应的以下各值：
 - ◆ wdSeparateByCommas，在各逗号处分隔列信息。
 - ◆ wdSeparateByDefaultListSeparator，在当前指定的其他分隔符（即“将文字转换成表格”对话框中“其他字符”右侧文本框内显示的字符）处分隔列信息。
 - ◆ wdSeparateByParagraphs，在段落标记处分隔列信息。
 - ◆ wdSeparateByTabs（如果未指定分隔符，这就是默认分隔符），在各制表符处分隔列信息。
- ◆ 也可以指定一个字符串或双引号作为自己选择的分隔符。例如，输入 Separator: = " | " 以使用一个竖条（|）作为分隔符。
- ◆ NumRows 是可选的 Variant 参数，它指定表格应有的行数。如果省略了 NumRows 参数，Word 将根据它所找到的指定的列数和所选分隔符的数目来决定表格中的

行数。

- ◆ NumColumns 是可选的 Variant 参数，它指定表格应有的列数。与 NumRows 参数的情况一样，如果省略了 NumColumns 参数，Word 将根据它所找到的指定的行数和所选分隔符的数目来决定表格中的列数。
- ◆ InitialColumnWidth 是可选的 Variant 参数，用来指定表格中每列的初始宽度（以点计）。如果省略了 InitialColumnWidth 参数，Word 将使用整个页面的宽度——从一个边界到另一边界——并把同等宽度分配至每一列，而不考虑各列内容的相对宽度。InitialColumnWidth 参数对于限制自动使用整个页面宽度是很有用的。在很多情况下，自动调整各列大小提供了更好的解决办法。
- ◆ Format 是可选的 Variant 参数，用来指定一种 Word 为表格内置的自动套用格式。要使用 Format 参数，需指定适当的 wdTableFormat 常量（例如使用 wdTableFormatElegant 来指定典雅型格式）。如果选择了要应用的格式，可以指定要将该格式的哪些属性应用于表格，这就要使用如下的可选 Variant 参数：
 - ◆ 设置 ApplyBorders 为 True 以应用指定格式的边框属性，设置为 False 则不用。
 - ◆ 设置 ApplyShading 为 True 以应用指定格式的底纹属性，设置为 False 则不用。
 - ◆ 设置 ApplyFont 为 True 以应用指定格式的字体属性，设置为 False 则不用。
 - ◆ 设置 ApplyColor 为 True 以应用指定格式的颜色属性，设置为 False 则不用。
 - ◆ 设置 ApplyHeadingRows 为 True 以应用指定格式的标题行属性，设置为 False 则不用。
 - ◆ 设置 ApplyLastRow 为 True 以应用指定格式的最后一行属性，设置为 False 则不用。
 - ◆ 设置 ApplyFirstColumn 为 True 以应用指定格式的第一列属性，设置为 False 则不用。
 - ◆ 设置 ApplyLastColumn 为 True 以应用指定格式的最后一列属性，设置为 False 则不用。
- ◆ AutoFit 是可选的 Variant 参数，将它设置为 True，可以使 Word 调整列宽以最好地适应各单元格的内容。在自动调整时，Word 并不增加表格的总宽度——它缩减该表格，或使它保持相同宽度。
- ◆ AutoFitBehavior 和 DefaultTableBehavior 在本章前面“创建表格”一节中已有说明。

下述语句将当前所选内容转换成一个 5 列的表格，信息以逗号分隔。它可以根据单元格内容来自动调整该表格，并使各单元格可以自动调整大小：

```
Set myTable = Selection.ConvertToTable(wdSeparateByCommas, _
    Selection.Paragraphs.Count, 5, , , , , , , , True, _ "表格"
    wdAutoFitContent, wdWord9TableBehavior)
```

确认所选内容在表格之内

在运行任何过程之前，如果过程期望在一个表格之内发生，那么确认当前所选内容是否适当是一个好想法。要检查所选内容当前是否在表格之内，需使用与该选择内容对应的 Information 属性的 wdWithInTable 参数。这是一个布尔型参数，如果所选内容是在表格之内，它返回 True，否则返回 False。例如：

```
If Selection.Information(wdWithInTable) = True Then
    'take actions here
End If
```

找出所选内容在表格中什么地方

除了确定所选内容是否在一个表格之内以外，还可以使用 Information 属性找到其他信息，当通过 Range 对象或 Selection 对象对表格进行工作时，这些信息会是很有用的。

一旦确认了所选内容是在一个表格之内（很可能是使用 wdWithInTable 参数得以确认的），就需检查所选内容是在行尾标记处，还是在单元格之中。如果所选内容在行尾标记处，某些行动就会失败。例如，既然所选内容在单元格或列之外，试图选择当前单元格或列就会失败，但是试图选择当前行则可以成功。

要检查所选内容是否在行尾标记处，使用与 Information 属性对应的 wdAtEndOfRowMarker 参数。如果所选内容是在行尾标记处，下述语句将所选内容左移一个字符（进到同一行中最后一个单元格）：

```
If Selection.Information(wdAtEndOfRowMarker) = True Then
    Selection.MoveLeft Unit:=wdCharacter, Count:=-1
```

如果所选内容包含行尾标记，而不是该标记之前的已折叠选择内容（一个插入点），wdAtEndOfRowMarker 参数返回 False。为了避免被选择的行尾标记在过程中引起问题，在检查所选内容是否在行尾标记处之前，如果所选内容没有折叠的话，应该使其折叠。下述语句是做这件事的，在语句中使用名为 curSel 的变量，如果折叠所选内容后所选内容并未在行尾标记处，就可以使用这个变量来恢复折叠的所选内容：

```
Dim curSel
With Documents("Communications.doc")
    If Selection.Type <> wdSelectionIP Then
        Set curSel = Selection.Range
        Selection.Collapse Direction:=wdCollapseStart
    End If
    If Selection.Information(wdAtEndOfRowMarker) = True Then
        Selection.MoveLeft Unit:=wdCharacter, Count:=-1, Extend:=wdMove
    Else
        If curSel <> "" Then curSel.Select
        Set curSel = Nothing
    End If
End With
```

放心地确认所选内容在表格之内以后，可以取回有关该表格的六个有用的信息片段：

- ◆ wdStartOfRangeColumnNumber 返回某一列的列号，所选内容或区域的起始就在这—列中。下述语句选择当前所选内容开始的那个列：

```
Selection.Tables(1).Columns(Selection.Information(wdStartOfRangeColumnNumber)).Select
```

- ◆ wdEndOfRangeColumnNumber 返回某一列的列号，所选内容或区域的结尾就在这—列中。下面的例子是如果名为 testRange 的区域不止一列宽的话，下述语句将删除该区域的结尾所在的那个列：

```

With testRange
    If .Information(wdStartOfRangeColumnNumber) <> _
        .Information(wdEndOfRangeColumnNumber) Then _
            .Tables(1).Columns(.Information _
                (wdEndOfRangeColumnNumber)).Delete
End With

```

- ◆ wdStartOfRangeRowNumber 返回某一行的行号，所选内容或区域的起始就在这一行中。
- ◆ wdEndOfRangeRowNumber 返回某一行的行号，所选内容或区域的结尾就在这一行中。
- ◆ wdMaximumNumberOfColumns 返回所选内容或区域中任意行内的最高列数。
- ◆ wdMaximumNumberOfRow 返回表格中所选内容或区域内的最高行数。

对表格排序

为了对某一个表格排序，可以指明该表格并使用 Sort 方法。Sort 使用针对 Table 对象的如下语法：

```

expression.Sort(ExcludeHeader, FieldNumber, SortFieldType, SortOrder, FieldNumber2,
SortFieldType2, SortOrder2, FieldNumber3, SortFieldType3, SortOrder3, CaseSensitive,
BidiSort, IgnoreThe, IgnoreKashida, IgnoreDiacritics, IgnoreHe, LanguageID)

```

各参数说明如下：

- ◆ expression 是表达式，它返回一个 Table 对象。
- ◆ ExcludeHeader 是可选的 Variant 参数，将它设置为 True 时，不对表格中的第一行进行排序（第一行往往是表格标题行）；设置为 False 时，要把表格中第一行包括进来。
- ◆ FieldNumber、FieldNumber2 和 FieldNumber3 是可选的 Variant 参数，分别指定用于排序的第一域、第二域和第三域。通常希望至少要指定 FieldNumber；否则，Word 实施对表格的按字母数字排序。
- ◆ SortFieldType、SortFieldType2 和 SortFieldType3 是可选的 Variant 参数，分别指定要使用的与 FieldNumber、FieldNumber2 和 FieldNumber3 对应的排序类型。对于美式英语，选项是按字母数字排序 (wdSortFieldAlphanumeric, 默认值)、按数字排序 (wdSortFieldNumeric) 或按日期排序 (wdSortFieldDate)。
- ◆ SortOrder、SortOrder2 和 SortOrder3 是可选的 Variant 参数，它们指定与 FieldNumber、FieldNumber2 和 FieldNumber3 对应的排序顺序。使用 wdSortOrderAscending 是指定按递增顺序排序（默认值），使用 wdSortOrderDescending 是指定按递减顺序排序。
- ◆ CaseSensitive 是可选的 Variant 参数，将它设置为 True 时，排序时要区分字母的大写。默认设置是 False。
- ◆ 后面的五个参数 (BidiSort、IgnoreThe、IgnoreDiacritics、IgnoreKashida 和 IgnoreHe) 用于特殊排序（例如从右向左的语言、阿拉伯文和希伯来文）。
- ◆ LanguageID 是可选的 Variant 参数，可使用它来指定排序语言。例如，排序语言为 Lithuanian 时，需指定与 LanguageID 对应的 wdLithuanian。排序语言为默认语言时，此参数可省略。

给表格添加一列

要给表格添加一列，需使用与适当的 Table 对象对应的 Columns 集合的 Add 方法。Columns 集合的 Add 方法的语法是：

```
expression.Add [BeforeColumn]
```

此处，expression 是必需的表达式，它返回一个 Columns 集合。BeforeColumn 是可选的 Variant 参数，它指定某个列，想要在这一列的左边插入一个新列。

下述语句使用 Count 属性来检查活动文档中第一个表格的列数，如果该表格的列数不到 5 列，就添加一列或几列，使其列数达到 5 列。每个新列都在表格中已有的最后一列之前加入：

```
With ActiveDocument.Tables(1)
    .Select
    If .Columns.Count < 5 Then
        Do Until .Columns.Count = 5
            .Columns.Add BeforeColumn:=.Columns(.Columns.Count)
        Loop
    End If
End With
```

从表格中删除一列

要删除一列，需指明该列，并使用 Delete 方法。Delete 方法不需要参数。下述语句删除以对象变量 myTable 引用的表格中的第一列：

```
myTable.Columns(1).Delete
```

设置列的宽度

使用 AutoFit 方法或者 SetWidth 方法，或者为某一列指定 Width 属性，都可以设置列的宽度。

AutoFit 方法自动地调整每列的大小，使其宽度与它的内容相适应。AutoFit 不需要参数。下述语句以 AutoFit 方法来调整活动文档中第一个表格的每列的大小：

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

SetWidth 方法可用来设置一列或几列的宽度，并指明表格中的其他列应随设置结果做何改变。与 SetWidth 方法对应的语法是：

```
expression.SetWidth ColumnWidth, RulerStyle
```

此处，expression 是表达式，它返回想要为之设置宽度的 Columns 集合或 Column 对象。ColumnWidth 是必需的单精度浮点型参数，它指定一列或几列的宽度，以点计。RulerStyle 是必需的长整型参数，它指定 Word 应如何调整各列的宽度：

- ◆ 默认值为 wdAdjustNone，将所有指定列设置为指定的宽度，必要时使其他各列左移或右移。这个参数与以交互方式工作时使用 Shift 键并拖曳列边界相类似。
- ◆ wdAdjustFirstColumn，将指定宽度应用于第一个被指定列，必要时只使该列右边某

些列进行调整。例如，使表格中的第一列稍微变宽，引起 Word 让第二列变窄，但第三列和后面各列不变；使第一列变宽甚多，引起 Word 让第二列和第三列都变窄，但第四列和后面各列不变。这个参数与以交互方式工作时拖曳列边界相类似。

- ◆ `wdAdjustProportional`，将指定宽度应用于第一个被指定列，保持表格的右边界仍在原先位置上，按比例调整所有未被指定的各列以适应这种变化。
- ◆ `wdAdjustSameWidth`，将指定宽度应用于第一个被指定列，保持表格的右边界仍在原先位置上，所有其他各列调整到相同宽度以适应这种变化。这个参数与以交互方式工作时使用 Ctrl 键并拖曳列边界相类似。

下述语句将活动文档中第一个表格的第二列的宽度设置为 50 点，使第二列右边的各列按比例进行调整：

```
ActiveDocument.Tables(1).Columns(2).SetWidth ColumnWidth:=50,  
RulerStyle:=wdAdjustProportional
```

`Width` 属性允许改变某一列的宽度，而不去担心对其他各列的影响。例如下述语句以点数来设置某一行的宽度：

```
ActiveDocument.Tables(11).Columns(44).Width = 100
```

选择某一列

要选择某一列，需使用针对适当的 `Column` 对象的 `Select` 方法。`Select` 不要参数。下述语句选择名为 `Originals.doc` 的文档中第三个表格的第二列：

```
Documents("Originals.doc").Tables(3).Columns(2).Select
```

给表格添加一行

要添加一行，需使用与该表格对应的 `Rows` 集合的 `Add` 方法。`Rows` 集合的 `Add` 方法的语法是：

```
expression.Add [BeforeRow]
```

此处，`expression` 是必需的表达式，它返回一个 `Rows` 对象。`BeforeRow` 是可选的 Variant 参数，它指定某行，想要在这一行的前面添加新行。如果省略了 `BeforeRow`，VBA 将新行添加到表格中已有的最后一行之后。

下述语句将新的第一行添加到对象变量 `myTable` 引用的表格中：

```
myTable.Rows.Add BeforeRow:=1
```

从表格中删除一行

要删除一行，需使用适当的 `Row` 对象的 `Delete` 方法。`Delete` 方法不要参数。下述语句删除对象变量 `myTable` 引用的表格的第一行：

```
myTable.Rows(1).Delete
```

设置一行或多行的高度

本文档内部分单回

要设置行的高度，既可以让 Word 自动设置，也可以使用 SetHeight 方法来指定确切的高度或最小高度，或者直接设置某一行或几行的 Height 属性。

要使 Word 自动设置某一行的高度，需将该行的 HeightRule 属性设置为 wdRowHeightAuto。然后，Word 会调整该行的高度，使之与内容最多的那个单元格相适应。下述语句将活动文档中第四个表格的第二行所对应的 HeightRule 属性设置为 wdRowHeightAuto：

```
ActiveDocument.Tables(4).Rows(2).HeightRule = wdRowHeightAuto
```

要为某一行或几行指定确切的高度或最小高度，需使用针对这一行或这几行的 SetHeight 方法。SetHeight 方法的语法是：

```
expression.SetHeight RowHeight, [HeightRule]
```

此处，expression 是表达式，它返回一个 Row 对象或 Rows 集合。HeightRule 是必需的 Variant 参数，它指定设置行高度的规则：使用 wdRowHeightAtLeast 来设置最小高度，使用 wdRowHeightExactly 来设置确切高度。HeightRule 的第三种设置是 wdRowHeightAuto，它指定自动设置的行高度，这里没有使用。

要是不用 SetHeight 方法，也可以通过设置所涉及的一行或几行的 Height 属性（即以点数）来指定高度：

```
Documents("Tables.doc").Tables(3).Rows(3).Height = 33
```

选择某一行

要选择某一行，需使用适当的 Row 对象的 Select 方法。Select 方法不要参数。下述语句选择名为 Tables.doc 的文档中最后一个表格的最后一行：

```
Documents("Tables.doc").Tables(.Tables.Count).Rows.Last.Select
```

插入单元格

要插入一个单元格，需使用 Cells 集合 Add 方法。Cells 集合的 Add 方法的语法是：

```
expression.Add [BeforeCell]
```

此处，expression 是表达式，它返回一个 Cells 集合。BeforeCell 是可选的 Variant 参数，它指定某一个单元格，新单元格应插入到该单元格的左边。（如果省略了 BeforeCell 参数，在使用 Columns 集合的 Cells 集合时，VBA 将新的一行单元格添加到表格末尾；在使用 Rows 集合的 Cells 集合时，VBA 将一个新单元格添加到表格中的第一行。）

下述语句在名为 Tables.doc 的文档中的第一个表格第一行内第二个单元格之前，插入一个单元格：

```
Documents("Tables.doc").Tables(1).Rows(1).Cells.Add _  
BeforeCell:=Documents("Tables.doc").Tables(1).Rows(1).Cells(2)
```

返回单元格内的文本

最高阶字符串置

为了返回某一个单元格的内容，需使用与该单元格对应的 Range 对象的 Text 属性。下述语句返回活动文档中第三个表格第二行内第一个单元格的文本，并将它指派给变量 strCellText：

```
strCellText = ActiveDocument.Tables(3).Rows(2).Cells(1).Range.Text
```

因为 Text 属性包含单元格结束标记（它占两个字符），所以在将 Text 属性指派给一个字符串时，通常希望去掉后两个字符：

```
strCellText = ActiveDocument.Tables(3).Rows(2).Cells(1).Range.Text  
strCellText = Left(strCellText, Len(strCellText) - 2)
```

在整个 Range 对象之内，可以使用它包含的任何对象和集合进行工作。例如，要针对某一个单元格内的若干段落进行工作，可以使用 Paragraphs 集合。

将文本输入单元格

要把文本输入单元格，需将该文本指派给对应此单元格的 Range 对象的 Text 属性。下述语句将文本输入当前所选内容的第一行的前三个单元格：

```
With Selection.Tables(1).Rows(1)  
.Cells(1).Range.Text = "Sample text in first cell."  
.Cells(2).Range.Text = "Sample text in second cell."  
.Cells(3).Range.Text = "Sample text in third cell."  
End With
```

删除单元格

行某对数

为了删除单元格，需使用适当的 Cell 对象或 Cells 集合的 Delete 方法。当删除一个或多个单元格时，必须指明表格中其余部分会发生什么事——被删除的那些单元格右边的各单元格是否要左移，或者被删除的单元格下面的各单元格是否要上移。

与 Cells 集合和 Cell 对象对应的 Delete 方法的语法是：

行元单人计

```
expression.Delete [ShiftCells]
```

此处，expression 是表达式，它返回一个 Cells 集合或 Cell 对象。ShiftCells 是可选的 Variant 参数，它指定被删除单元格下面或右边的各单元格应如何移动。它可以是如下各值：

- ◆ wdDeleteCellsEntireColumn，删除指定的单元格所在列的整个列。
- ◆ wdDeleteCellsEntireRow，删除整个行。
- ◆ wdDeleteCellsShiftLeft，横向左移各单元格以填满空隙。
- ◆ wdDeleteCellsShiftUp，上移各单元格以填满空隙。

下述语句删除活动文档中第一个表格第一行的第一个单元格，并使第一行的其他单元格左移以填满空隙：

```
ActiveDocument.Tables(1).Rows(1).Cells(1).Delete [ShiftCells:  
wdDeleteCellsShiftLeft]
```

对于那些要依赖用户在一个表格之内做出选择的过程，在决定如何移动单元格之前，可能想要确定所选内容中有多少行或多少列。下面的例子就是检查所选内容中的行数和列数。如果所选内容仅为一个单元格，或者所选内容是一列中的所有单元格，代码就删除这个或这些单元格，并将行中其他单元格左移。如果所选内容是一列中的多个单元格，代码就删除这些单元格，并将列中其他单元格上移。如果所选内容跨越多列多行，代码就显示一个消息框，要求用户只在一行或只在一列中做出选择：

```

With Selection
    If .Columns.Count > 1 And .Rows.Count > 1 Then
        MsgBox "Please select cells in only one row " _
            & "or only one column."
    End If
    Else
        If .Cells.Count > 1 Then
            If .Columns.Count > 1 Then
                .Cells.Delete ShiftCells:=wdDeleteCellsShiftUp
            Else
                .Cells.Delete ShiftCells:=wdDeleteCellsShiftLeft
            End If
        Else
            .Cells.Delete ShiftCells:=wdDeleteCellsShiftLeft
        End If
    End If
End With

```

选择单元格区域

要在表格之内选择一个单元格区域，需声明一个 Range 变量，把想要选择的单元格指派给它，然后选择该区域。下面的例子声明 Range 变量 myCells，把活动文档中第一个表格的前四个单元格指派给它，然后选择该区域：

```

Dim myCells As Range
With ActiveDocument
    Set myCells = .Range(Start:=-.Tables(1).Cell(1, 1).Range.Start, _
        End:=-.Tables(1).Cell(1, 4).Range.End)
    myCells.Select
End With

```

将表格或行转换为文本

为了把一个表格、一行或多行转换为文本，需要先指定该表格、这一行或这几行，再使用 ConvertToText 方法。ConvertToText 方法使用如下语法：

expression.ConvertToText(Separator, Nested Tables)

此处，*expression* 是必需的表达式，它返回一个 Table 对象、一个 Row 对象或 Rows 集合。*Separator* 是可选的 Variant 参数，它指定用来标记列信息在何处分隔的分隔符。可能使用的值如下：

- ◆ *wdSeparateByCommas*，用逗号来分隔列信息。
- ◆ *wdSeparateByDefaultListSeparator*，用当前指定的其他分隔符（即“将表格转换成文

“文本”对话框中“其他字符”右侧文本框内显示的字符) 来分隔列信息。

◆ wdSeparateByParagraphs，用段落标记来分隔列信息。

◆ wdSeparateByTabs (如未指定分隔符, 这就是默认分隔符), 用制表符来分隔列信息。

◆ 也可以指定一个字符串或双引号作为自己选择的分隔符。例如, 输入 Separator:=“|” 以使用一个竖条 (|) 作为分隔符。

下述语句使用星号 (*) 作为分隔符, 将当前所选内容中的第一个表格转换为文本:

```
Selection.Tables(1).ConvertToText Separator:="*"
```

可以使用一个 Table 对象、一个 Row 对象或一个 Rows 集合的 ConvertToText 方法。

下述语句仅将被选表格的第一行转换为以制表符来分隔的文本:

```
Selection.Tables(1).Rows(1).ConvertToText Separator:=wdSeparateByTabs
```

如果需要在完成转换之后, 仍使用表格的内容继续进行工作, 应在转换表格时将一个区域指定给该表格。以后可以使用这个 Range 对象来工作以进行信息的操控。例如, 下述语句先将名为 Cleveland Report. doc 的文档中的第一个表格转换为以段落来分隔的文本, 并将区域 exTable 指定到已转换信息, 然后复制该区域, 创建一个新文档, 并把它粘贴进信息中去:

```
Dim exTable As Range
Set exTable = Documents("Cleveland Report.doc").Tables(1). _
    ConvertToText(Separator:=wdSeparateByParagraphs)
exTable.Copy
```

```
Documents.Add
```

```
Selection.Paste
```

本文档对操作方法的讲解主要集中在如何使用 VBA 代码来实现各种功能上, 而不是对每一种操作的具体步骤进行详细说明。因此, 在阅读本章时, 可能会遇到一些不太熟悉的概念和术语, 请根据自己的需求查阅相关的书籍或网上资源, 以便更好地理解相关内容。

```
Dim myCells As Range
With ActiveDocument
    Set myCells = .Range(SelStart:=Tables(1).Cells(1, 1).Range.Start, _
        End:=.Tables(1).Cells(1, 4).Range.End)
    myCells.Select
End With
```

本文档对操作方法的讲解主要集中在如何使用 VBA 代码来实现各种功能上, 而不是对每一种操作的具体步骤进行详细说明。因此, 在阅读本章时, 可能会遇到一些不太熟悉的概念和术语, 请根据自己的需求查阅相关的书籍或网上资源, 以便更好地理解相关内容。

```
expression.ConvertToText(Separator, NestedTables)
expression.ConvertToText(Separator, NestedTables)
```

本节将介绍如何使用 VBA 代码来实现对表格的操作, 包括插入、删除、修改等操作。通过这些操作, 可以更灵活地处理和分析数据。

在学习本节内容之前, 需要了解一些基本的 VBA 语法知识, 包括变量、常量、语句等。建议读者在学习本节之前, 先阅读《VBA 基础教程》一书, 以便更好地理解相关内容。

第 22 章 了解 Excel 对象模型和重要对象

- ◆ 获得 Excel 对象模型的概括性知识
- ◆ 了解 Excel 的可创建对象
- ◆ 使用 Workbooks 进行工作
- ◆ 使用 Worksheets 进行工作
- ◆ 使用 ActiveCell 或 Selection 进行工作
- ◆ 使用 Range 进行工作
- ◆ 设置 Options 对象

本章介绍，如何从使用作为 Excel 的基础性理论体系结构的 Excel 对象模型开始进行工作，也介绍如何使用一些最为立即可用的 Excel 对象来实施常见的行动。这些对象包括 Workbooks 集合和 Workbook 对象，ActiveCell 对象以及 Range 对象。也要学习如何在 Excel 中设置选项。

获得 Excel 对象模型的概括性知识

为了在 Excel 中使用 VBA 进行工作，需要懂得 Excel 对象模型是如何构成的，这一点并不特别重要。但是，大多数人发现，了解对象模型中的重要对象还是有帮助的。要查看 Excel 对象模型，可遵循如下步骤：

1. 启动或激活 Excel，然后按下 Alt+F11 键以启动或激活“Visual Basic 编辑器”。
2. 选择“帮助”>“Microsoft Visual Basic 帮助”以显示“Microsoft Visual Basic 帮助”任务窗格。
3. 在“搜索”文本框中键入“Microsoft Excel 对象模型”，然后按下 Enter 键，或单击“开始搜索”按钮。
4. 在“搜索结果”窗格中，单击“Microsoft Excel 对象模型”条目，就可以看到“帮助”屏幕（如图 22.1 所示）。

在屏幕上查看 Excel 对象模型时，可以看到有些框的底色是黄的，而另有些框的底色是蓝的。黄框是组织成集合的对象，而蓝框里是没有集合的对象。单击一个框，就可以看到对应于该对象的帮助屏幕，包括图解和对象模型中的有关对象。图 22.2 显示出对应于 Workbooks 集合的帮助屏幕的一部分，它包括 Excel 中打开的工作簿的详细情况。

了解 Excel 的可创建对象

Excel 有很多可创建对象，不必明显地通过 Application 对象就可以接触到对象模型中大多数令人感兴趣的对象。在大部分场合下，以下对象都是重要的对象：

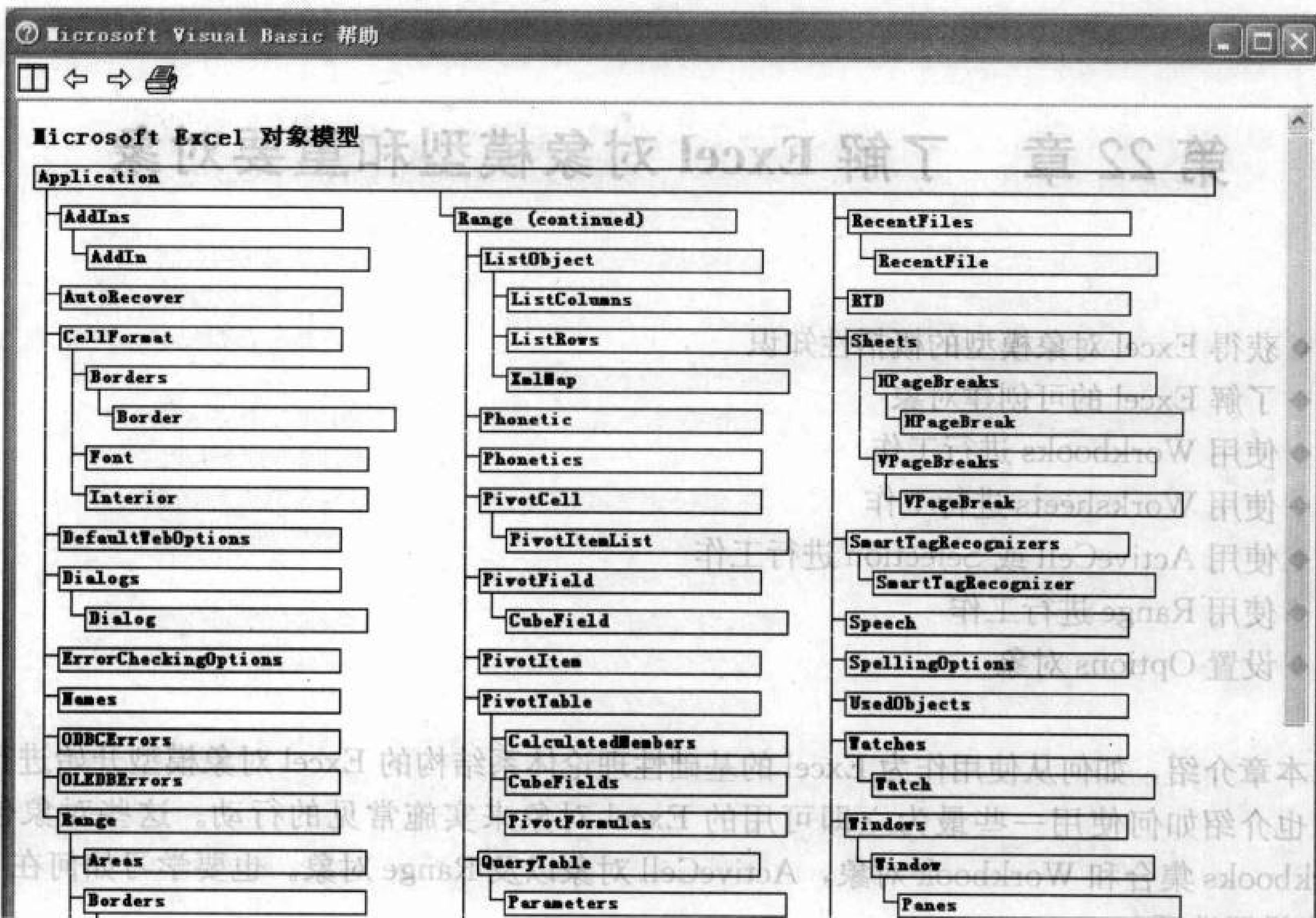


图 22.1 “Microsoft Visual Basic 帮助”任务窗格提供了 Excel 对象模型的图解

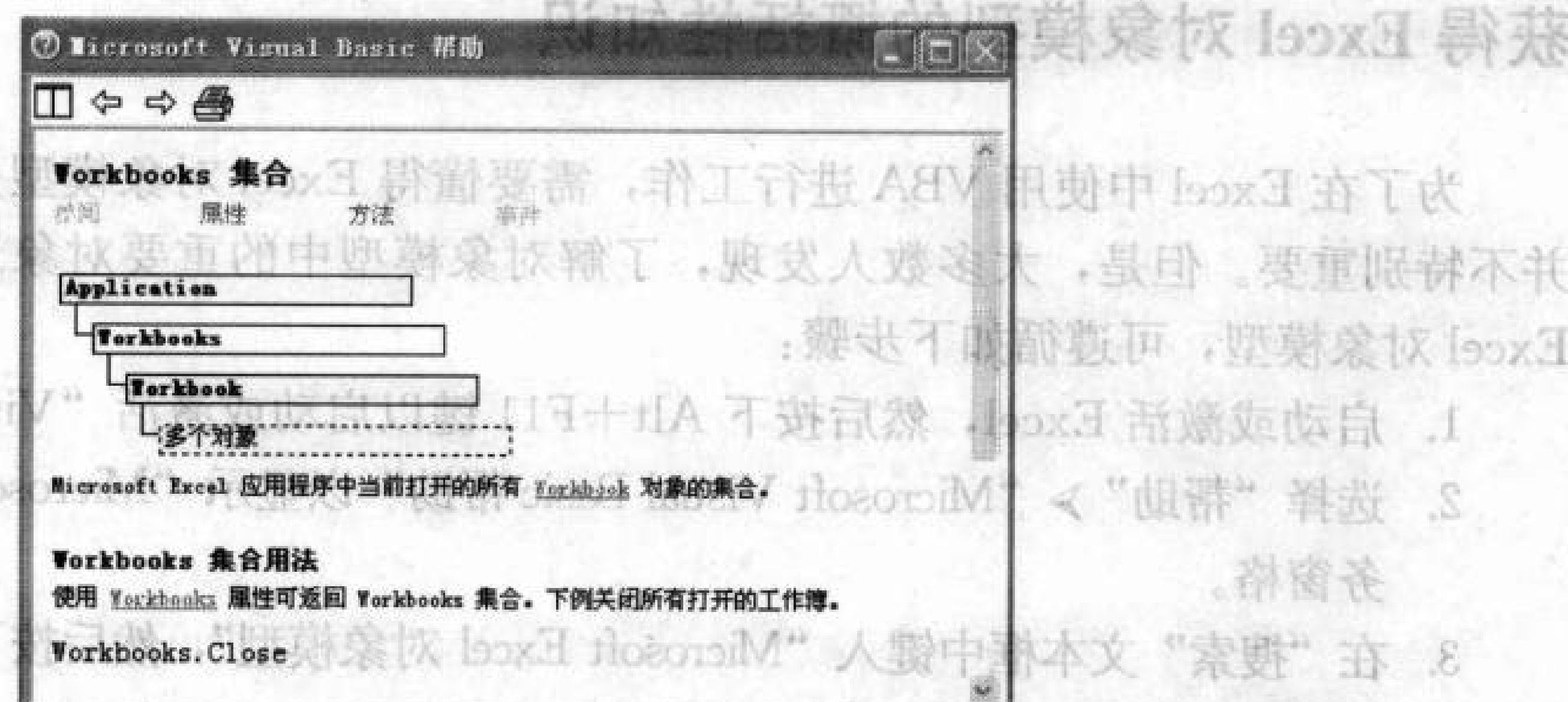


图 22.2 与单个对象或集合对应的帮助屏幕，这里是对应于 Excel 2003 的对象模型。可单击某一对象以查看其详情，其中包括一个显示与该对象或集合有关的略图

- ◆ **Workbooks 集合**，它包含多个 Workbook 对象，表示所有打开的工作簿。在一个工作簿内，Sheets 集合包含若干个表示工作表的 Worksheet 对象，以及若干个表示图表工作表的 Chart 对象。在一个工作表上，Range 对象可使用户访问若干种区域，从单个单元格直到完整的工作表。
- ◆ **ActiveWorkbook 对象**，它代表活动工作簿。
- ◆ **ActiveSheet 对象**，它代表活动工作表。
- ◆ **Windows 集合**，它包含若干 Window 对象，表示所有打开的窗口。

- ◆ ActiveWindow 对象，它表示活动窗口。使用此对象时，必须检查它所表示的窗口是不是想要操控的那种类型的窗口，因为该对象总是返回当前具有焦点的那个窗口。
- ◆ ActiveCell 对象，它表示活动单元格。这个对象对于在用户选择的单元格上进行工作的简单过程（例如，计算各种值或校正格式设置）特别有价值。

使用 Workbooks 进行工作

在很多 Excel 过程中，需要针对工作簿进行工作：创建新工作簿，在各种位置上及以各种格式保存工作簿，打开已有的工作簿，关闭工作簿以及打印工作簿。为此，需要使用 Workbooks 集合来工作，它包含与 Excel 中每个打开的工作簿对应的 Workbooks 对象。

创建工作簿

要创建一个工作簿，需使用 Workbooks 集合的 Add 方法。其语法是：

```
Workbooks.Add(Template)
```

此处，Template 是可选的 Variant 参数，它指明如何创建工作簿。以下各小节将讨论各种选择。

创建新的空白工作簿

要创建一个空白工作簿（如同以交互方式工作时，单击“标准”工具栏的“新建”按钮），需略去 Template 参数：

```
Workbooks.Add
```

新工作簿收到若干工作表，其数量设置在“选项”对话框的“常规”标签上的“新工作簿内的工作表数”文本框内。可以在 VBA 中使用 Application 对象的 SheetsInNewWorkbook 属性来得到或设置这一数值。例如，下面的宏先声明一个名为 mySiNW 的整型变量，将当前的 SheetsInNewWorkbook 属性存入其中。将这个属性设置为 12，创建工作簿（有 12 个工作表），然后将 SheetsInNewWorkbook 恢复为它原先的数值：

```
Sub MVBA_New_Workbook_with_12_Sheets()
    Dim mySiNW As Integer
    mySiNW = Application.SheetsInNewWorkbook
    Application.SheetsInNewWorkbook = 12
    Workbooks.Add
    Application.SheetsInNewWorkbook = mySiNW
End Sub
```

创建基于模板的新工作簿

要创建基于某个模板的新工作簿，需指定该模板文件的完整路径和名称。例如，下述语句创建工作簿（基于网络文件夹 \\server\\template\\excel 之中的模板 Balance Sheet.xlt）：

```
Workbooks.Add Template:="\\server\\template\\excel\\Balance Sheet.xlt"
```

是窗口的示例代码示例，如果执行成功，则窗口将显示“成功”，失败则显示“失败”。

创建基于已有工作簿的新工作簿

要在创建基于一个已有工作簿的新工作簿，需指定该已有工作簿文件的完整名称和路径。例如，下述语句创建一个基于 C:\Business 文件夹之中名为 Personnel.xls 的已有工作簿的新工作簿：

```
Workbooks.Add Template:="C:\Business\Personnel.xls"
```

创建图表工作簿、宏工作表或工作表

使用表 22.1 所示的针对 Template 参数的各常量，也可以创建包含单个图表、宏工作表或工作表的工作簿。

表 22.1 与创建图表工作簿、宏工作表或工作表对应的常量

常量	创建工作簿包含有
xlWBATChart	一个图表工作表
xlWBATExcel4IntlMacroSheet	一个国际性的宏工作表
xlWBATExcel4MacroSheet	一个宏工作表
xlWBATWorksheet	一个工作表

例如，下述语句创建包含单个图表工作表的工作簿：

```
Workbooks.Add Template:=xlWBATChart
```

保存工作簿

如果是第一次保存某个工作簿，必须指定要使用的路径和文件名。在这之后，便可以在相同名称之下保存那个工作簿，也可以另行指定路径、名称或格式，或者另行指定这三者来保存那个工作簿。

第一次保存某个工作簿或保存为另一文件

如果是第一次保存某个工作簿，或使用不同的路径、名称或格式来保存某个工作簿，需使用 SaveAs 方法。其语法是：

```
expression.SaveAs(FileName, FileFormat, Password, WriteResPassword,  
ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru,  
TextCodePage, TextVisualLayout, Local)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Workbook 对象。
- ◆ FileName 是可选的 Variant 参数，它为工作簿指定名称。如果省略 FileName，VBA 为该工作簿使用当前文件夹和 Bookn.xls 的默认文件名，此处的 n 是下一个可用序号（例如，Book5.xls）。VBA 使用默认的文件格式，它被设置在“选项”对话框（选择“工具”>“选项”）“转换”标签上的“将 Excel 文件保存为”下拉列表中。使用 Application 对象的 DefaultSaveFormat 属性，可以得到和设置默认保存格式。例如，下述语句将默认保存格式设置为 xlNormal，即“Microsoft Office Excel Workbook”或

“Microsoft Excel Workbook” 格式 (取决于所用的 Excel 版本):

```
Application.DefaultSaveFormat = xlNormal
```

警告: 保存某一工作簿时, 应检查那个文件夹中是否已经有使用这一名称的工作簿存在。如果有, VBA 会覆盖它且不加警告, 这将引起数据丢失。参阅第 9 章中的“使用 Dir 函数判断一个文件是否存在”一节, 以便获得对于检查使用特定名称的文件是否已经存在的指导。

- ◆ FileFormat 是可选的 Variant 参数, 它指定以什么格式来保存工作簿。表 22.2 列出了与指定常用格式对应的 xlFormat 常量。

表 22.2 与广泛使用的格式对应的 xlFormat 常量

常量	将文档保存为
xlNormal	一个普通工作簿
xlXMLSpreadsheet	一个 XML 数据表
xlWebArchive	一个单文件 Web 页
xlHtml	一个 Web 页
xlTemplate	一个模板
xlExcel9795	一个与 Excel 95 版和以后版本对应的 Excel 工作簿

- ◆ Password 是可选的 Variant 参数, 使用它来提供打开工作簿所需要的密码。Password 与字母大小写有关。如果用户不能提供密码, Excel 将不打开工作簿。
 - ◆ WriteResPassword 是可选的 Variant 参数, 使用它来提供以可写方式打开工作簿所需要的密码。WriteResPassword 与字母大小写有关, 如果用户不能提供密码, Excel 将以只读方式打开工作簿。
 - ◆ ReadOnlyRecommended 是可选的 Variant 参数, 设置为 True 时, 是让 Excel 推荐用户使用只读方式打开工作簿。通常这种推荐是不带强制性的, 最好使用以可写方式打开工作簿的密码来保护工作簿。
 - ◆ CreateBackup 是可选的 Variant 参数, 将它设置为 True, 是使 Excel 自动生成该工作簿的备份。默认设置为 False。
 - ◆ AccessMode 是可选参数, 用它指定工作簿是共享的还是处于排它方式之中。指定为 xlExclusive, 是与排它方式相对应; 指定为 xlShared, 是与共享方式相对应; 而 xlNoChange 则不更改访问方式 (这是默认设置)。
 - ◆ ConflictResolution 是可选参数, 用它来指定如何解决对工作簿的任何有冲突的更改。使用 xlLocalSessionChanges, 表示自动接受本地用户的修改, 使用 xlOtherSessionChanges 表示接受本地用户之外的其他用户的更改, 而使用 xlUserResolution 表示显示“冲突解决方案”对话框, 让用户来选择如何解决冲突。
 - ◆ AddToMru 是可选的 Variant 参数, 将它设置为 True, 是要把工作簿添加到“文件”菜单底部的“最近使用过的文件”列表中。默认设置为 False。
 - ◆ TextCodePage 和 TextVisualLayout 是可选的 Variant 参数, 用于国际版的 Excel (不是美式英语 Excel)。Local 是可选的 Variant 参数, 它控制使用的语言是 Excel 的语言 (True), 还是 VBA 的语言 (False)。(很少用到 Local。)
- 例如, 下述语句在当前文件夹中名称 Salaries.xls 之下, 使用默认保存格式来保存活动

工作簿：

```
ActiveWorkbook.SaveAs FileName:="Salaries.xls"
```

下述语句在名为\\server2\Public 的文件夹中，名称 Building Schedule.xls 之下，使用“Microsoft Excel 97—2003&5.0/95”格式（出自 Excel 2003）来保存打开的工作簿：

```
ActiveWorkbook.SaveAs Filename:="\\server2\Public\Building Schedule.xls", _  
FileFormat:=xlExcel9795
```

注意：xlExcel9795 格式在 Excel 用户界面中有另一个不同的名称，这取决于所用 Excel 的版本。Excel 2003 把它称为“Microsoft Excel 97—2003&5.0/95”格式。Excel XP 称它为“Microsoft Excel 97—2002&5.0/95”格式，而 Excel 2000 称它为“Microsoft Excel 97—2000&5.0/95 Workbook”格式。不管使用什么名称，此格式是与 Excel 95 及以后版本兼容的。

保存已经保存过的工作簿

如果某个工作簿已经保存过了，使用 Save 方法可以在相同名称下再保存它。对于 Workbook 对象，Save 方法无需参数。例如，下述语句保存名为 Data Book.xls 的工作簿：

```
Workbooks("Data Book.xls").Save
```

保存所有打开的工作簿

Workbooks 集合没有 Save 方法，但是使用与如下子过程类似的一个循环可以保存所有打开的工作簿：

```
Sub Save_All_Workbooks()  
    Dim myWorkbook As Workbook  
    For Each myWorkbook In Workbooks  
        myWorkbook.Save  
    Next myWorkbook  
End Sub
```

打开工作簿

要打开一个工作簿，需使用 Workbooks 集合的 Open 方法。其语法是：

```
expression.Open(Filename, UpdateLinks, ReadOnly, Format, Password, WriteResPassword,  
IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMru,  
Local, CorruptLoad)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Workbooks 集合。通常希望使用 Workbooks 集合本身。
- ◆ FileName 是必需的字符串参数，它提供要打开的工作簿的路径和名称。
- ◆ UpdateLinks 是可选的 Variant 参数，它控制 Excel 如何更新工作簿中的链接，表 22.3 列出各个值及其效果。

表 22.3 与 UpdateLinks 参数对应的值

值	效果
(省略)	Excel 提示用户决定如何更新链接
0	Excel 不更新链接
1	Excel 更新外部链接, 但不更新远程链接
2	Excel 更新远程链接, 但不更新外部链接
3	Excel 更新所有链接

- ◆ ReadOnly 是可选的 Variant 参数, 将它设置为 True, 是以只读方式来打开工作簿。默认设置为 False。
- ◆ Format 是可选的 Variant 参数, 用来指定打开文本文件时的分隔符。1 对应于制表符, 2 对应于逗号, 3 对应于空格, 4 对应于分号, 5 对应于没有分隔符, 6 对应于使用 Delimiter 参数指定的分隔符。
- ◆ Password 是可选的 Variant 参数, 使用它来提供打开工作簿所要求的密码。Password 是与字母大小写有关的。如果省略了 Password, 而又要求有密码的话, Excel 提示用户注意这一点。

警告: 只要可能, 避免把密码放在代码中, 因为有可能被其他人读到。

- ◆ WriteResPassword 是可选的 Variant 参数, 使用它来提供以可写方式打开工作簿所要求的密码。WriteResPassword 是与字母大小写有关的。如果省略了 WriteResPassword, 而又要求有密码的话, Excel 提示用户注意这一点。
- ◆ IgnoreReadOnlyRecommended 是可选的 Variant 参数, 将它设置为 True 时, 是让 Excel 忽略对以只读方式打开工作簿的推荐。
- ◆ Origin 是可选的 Variant 参数, 在打开文本文件时, 使用它来指定用于文件编码的操作系统, 因而也就指定了如何处理回车/换行字符以及字符编码。使用 xlWindows 以指明是 Windows, 使用 xlMacintosh 指明是 Mac, 使用 xlMSDOS 指明是 DOS。
- ◆ Delimiter 是可选的 Variant 参数, 把它与 Format 的值为 6 一起使用, 可指定打开文本文件时要使用的分隔符。
- ◆ Editable 是可选的 Variant 参数, 如果将它设置为 True, 此时 FileName 指定一个模板, 是为了打开这个模板本身, 而不是启动基于该模板的工作簿(启动该工作簿则应设置为 False)。Editable 也应用于 Excel 4.0 加载宏: True 对应于在可见窗口中打开加载宏, False 对应于以隐藏方式打开加载宏。
- ◆ Notify 是可选的 Variant 参数, 将它设置为 True, 是让 Excel 把工作簿添加到文件通知列表。当该工作簿可用时, Excel 便通知用户。如果将 Notify 指定为 False, 要是有人想把该工作簿打开, 这种行动便会失败。
- ◆ Converter 是可选的 Variant 参数, 使用它来指定打开文件时要使用的第一文件转换器。
- ◆ AddToMru 是可选的 Variant 参数, 将它设置为 True, 是要把工作簿添加到“文件”菜单底部的“最近使用过的文件”列表中。默认设置为 False。
- ◆ Local 是可选的 Variant 参数, 它控制使用的语言是 Excel 的语言 (True) 还是 VBA 的语言 (False)。(很少用到 Local。)
- ◆ CorruptLoad 是可选的 Variant 参数, 它控制 Excel 如何处理在打开工作簿时遭遇到的破坏。使用 xlNormalLoad 是采取常规举动——首先, 像往常一样打开工作簿; 第

二，如果文件有问题则进行修理；第三，从工作簿中恢复数据。使用 `xlRepairFile` 是直接进入修理阶段，使用 `xlExtractData` 则是直接进入恢复阶段。

例如，下述语句打开保存在 C:\Business 文件夹中名为 Expenses.xls 的工作簿，但不更新链接：

```
Workbooks.Open Filename:="C:\Business\Expenses.xls", UpdateLinks:=0
```

下述语句打开保存在 D:\Planning 文件夹中名为 Plan.xls 的工作簿，同时提供打开工作簿要用的密码：

```
Workbooks.Open Filename:="D:\Planning\Plan.xls", Password:="s@cur1ng!"
```

下述语句打开文件夹 z:\transfer 中名为 Data13.txt 的文本文件，使用感叹号（!）作为分隔符：

```
Workbooks.Open Filename:="z:\transfer\Data13.txt", Format:=6, Delimiter:="!"
```

关闭工作簿

要关闭一个工作簿，需使用针对适当的 `Workbook` 对象的 `Close` 方法。其语法是：

```
expression.Close(SaveChanges, FileName, RouteWorkbook)
```

该语法的各组成部分如下：

- ◆ `expression` 是必需的表达式，它返回一个 `Workbook` 对象或 `Workbooks` 集合。

- ◆ `SaveChanges` 是可选的 Variant 参数，它指定要保存 (True) 还是不保存 (False) 工作簿中任何未保存的更改。如果省略了 `SaveChanges`，Excel 提示用户要保存包含有未保存的更改的工作簿。

- ◆ `FileName` 是可选的 Variant 参数，用来指定在该文件名之下保存包含有更改的工作簿。在大多数场合下，在使用 `Close` 方法关闭工作簿之前，最好先使用 `SaveAs` 方法，在另一个不同名称之下保存该工作簿。

- ◆ `RouteWorkbook` 是可选的 Variant 参数，将它设置为 True，是把工作簿传送给传送名单上的下一个收件人。将它设置为 False，则是阻止传送工作簿。如果工作簿没有附加的传送名单，`RouteWorkbook` 就没有效果。

例如，下述语句关闭活动工作簿，但不保存更改：

```
ActiveWorkbook.Close SaveChanges:=False
```

关闭所有打开的工作簿

为了关闭所有打开的工作簿，可以使用 `Workbooks` 集合的 `Close` 方法：

```
Workbooks.Close
```

`Close` 方法不用参数。Excel 提示要保存包含有未保存的更改的任何工作簿。如果在过程中不便做出这种提示，可以使用一个循环（例如，针对 `Workbooks` 集合的 `For Each...Next` 循环）来逐个关闭每个打开的工作簿，同时使用 `SaveChanges` 参数来控制让 Excel 保存还是放弃任何未保存的更改。

共享工作簿

要确定某一个工作簿是否可以共享，需检查它的 MultiUserEditing 属性，这是一个只读的布尔型属性。

要共享某一个工作簿，需使用 SaveAs 方法（在本章前面“第一次保存某个工作簿或保存为另一文件”中有讨论），以便使用与 AccessMode 参数对应的 xlShared 值来保存该文件。

例如，如果名为 Brainstorming.xls 的工作簿是不能共享的话，下述语句使它变为可以共享：

```
With Workbooks("Brainstorming.xls")
    If MultiUserEditing = False Then
        .SaveAs Filename:=-.FullName, AccessMode:=xlShared
    End If
End With
```

保护工作簿

要保护某一个工作簿，需使用针对适当的 Workbook 对象的 Protect 方法。其语法是：

```
expression.Protect(Password, Structure, Windows)
```

该语法的各组成部分如下：

◆ expression 是必需的表达式，它返回一个 Workbook 对象。

◆ Password 是可选的 Variant 参数，它指定使工作表失去保护所对应的密码。Password 是与字母大小写有关的。几乎总是希望提供 Password——如果没有的话，能够打开该工作簿的任何人都会使工作簿失去保护。

◆ Structure 是可选的 Variant 参数，将它设置为 True，可以保护工作簿的结构（工作簿结构是指各工作表是如何相对定位的）；也可以将它留在默认设置 False。

◆ Windows 是可选的 Variant 参数，将它设置为 True，可以保护工作簿的各个窗口；如果省略此参数，就会使各窗口失去保护。

例如，下述语句使用密码 011securd 来保护活动工作簿的结构和各个窗口：

```
ActiveWorkbook.Protect Password:="011securd", Structure:=True, Windows:=True
```

注意：除了保护工作簿以防被人修改之外，还可以保护工作簿以防被人打开。详情

请见本章后面的“设置针对工作簿的密码和只读建议”一节。

使用 ActiveWorkbook 对象进行工作

ActiveWorkbook 对象返回一个表示活动工作簿的 Workbook 对象——活动工作簿是在 Excel 窗口中具有焦点的工作簿。ActiveWorkbook 对象的行为与 Workbook 对象类似。用户在打开想对其施加影响的工作簿之后，需要调用某些过程时，ActiveWorkbook 对象在这些过程中是很有用的。

如果没有工作簿是打开的，也就没有 ActiveWorkbook 对象，试图使用 ActiveWorkbook 对象的任何代码都将返回一个错误。如果 Excel 中没有工作簿是打开的，那么用户可

以运行宏。所以，在尝试执行假定存在一个活动工作簿的代码之前，证实至少一个工作簿已打开是一个好主意。一种选择是，在运行代码之前，检查一下 ActiveWorkbook 是否是 Nothing，如下例所示：

```
If ActiveWorkbook Is Nothing Then
    MsgBox "Please open a workbook and click in it before running this macro."
    & vbCr & vbCr & "This macro will now end.", _
    vbOKOnly + vbExclamation, "No Workbook Is Open"
End If
```

检查一下，代码中假设为活动工作簿的工作簿是否确实是活动工作簿，也是一个好主意。这个问题在下述情况很容易发生：当某一个过程从使用活动工作簿开始，然后创建了一个新工作簿并在其中工作；新工作簿变成了活动工作簿，而从此处起，代码可能会在错误的工作簿上开始运行。

如果对于哪一个工作簿正在进行工作心有疑虑，可以声明一个 Workbook 对象变量，并针对这个对象变量进行工作，而不用 ActiveWorkbook 对象。例如，下述语句先声明一个 Workbook 对象变量，将 ActiveWorkbook 对象指派给它，这样后面的代码就可以针对这个对象变量来工作了：

```
Dim myWorkbook As Workbooks
Set myWorkbook = ActiveWorkbook
With myWorkbook
    'actions here
End With
```

使用 Worksheets 进行工作

需要通过 VBA 来操控的大多数工作簿都包含一个或多个工作表，所以大多数过程都需要针对工作表来进行工作——插入工作表，删除工作表，复制或移动工作表，或者只是打印工作表中的适当区域。

每个工作表都用一个 Sheet 对象来表示。多个 Sheet 对象汇集成 Sheets 集合。

插入工作表

要将一个工作表插进工作簿，需使用 Sheets 集合的 Add 方法，其语法是：

```
expression.Add(Before, After, Count, Type)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Sheets 集合。通常想要用的是 Sheets 集合本身。
- ◆ Before 是可选的 Variant 参数，它指定要在它前面添加新工作表的那个工作表。After 是可选的 Variant 参数，它指定要在它后面添加新工作表的那个工作表。典型情况是，希望指定 Before 或 After，而不是二者均指定。也可以省略这两个参数，以便 Excel 将新工作表插入在活动工作表之前。
- ◆ Count 是可选的 Variant 参数，它指定要添加多少个工作表。如果省略了 Count，

VBA 使用默认值 1。

- Type 是可选的 Variant 参数，它指定要插入的工作表的类型。默认设置为 xlWorksheet，即标准工作表。也可以插入图表工作表 (xlChart)、Excel 4 宏工作表 (xlExcel4MacroSheet) 或者 Excel 4 国际宏工作表 (xlExcel4IntlMacroSheet)。

例如，下述语句声明一个名为 mySheet 的 Worksheet 对象变量，在第一个打开的工作簿中的第一个工作表之前，插入一个工作表，并将新工作表指派给 mySheet，然后将 mySheet 的 Name 属性设置为 Summary。(Name 属性控制出现在工作表的标签上的文本。)

```
Dim mySheet As Worksheet
Set mySheet = Workbooks(1).Sheets.Add(before:=Sheets(1))
mySheet.Name = "Summary"
```

下述语句在活动工作簿中最后一个工作表之后插入两个图表工作表，这两个图表工作表使用默认名称（如 Chart1 和 Chart2）

```
ActiveWorkbook.Sheets.Add After:=Sheets(Sheets.Count), Count:=2, Type:=xlChart
```

删除工作表

要删除一个工作表，需使用适当的 Sheet 对象的 Delete 方法。Delete 方法不用参数。例如，下述语句从以 myWorkbook 对象变量来引用的工作簿中删除名为 Summary 的工作表：

```
myWorkbook.Sheets("Summary").Delete
```

删除一个工作表会丧失存储在该工作表上的数据，所以按照默认方式，Excel 要求用户确认这种删除（如图 22.3 所示）。如果需要回避这一用户互动——例如，在某一个过程中，在用户不知道的情况下添加一个工作表，使用它来操控数据，然后又删除它——可以把 Excel 中的警告信息关掉，这就需要在删除工作表之前，将 Application 对象的 DisplayAlerts 属性设置为 False，然后把警告信息重新接通：

```
Application.DisplayAlerts = False
myWorkbook.Sheets("Summary").Delete
Application.DisplayAlerts = True
```

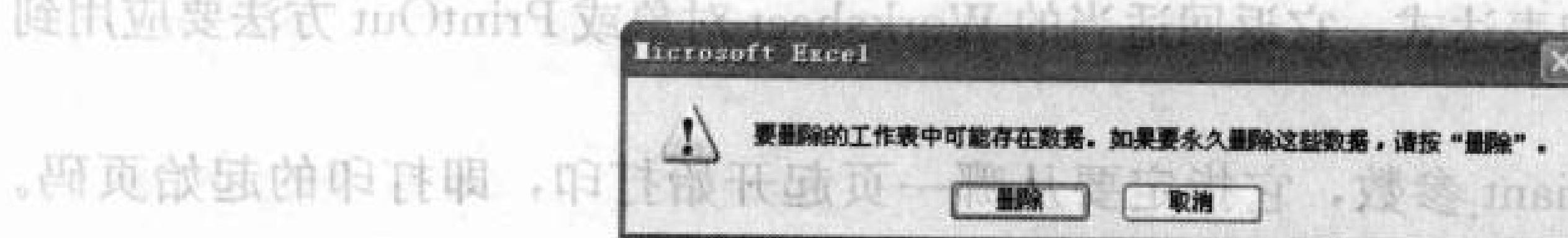


图 22.3 当删除工作表时，必须将 Excel 中的警告关掉，

或者让用户在这个对话框里确认这一删除

复制或移动工作表

要复制一个工作表，需使用适当的 Sheet 对象的 Copy 方法。要移动一个工作表，需使用 Move 方法。它们的语法是：

```
expression.Copy(Before, After)
expression.Move(Before, After)
```

此处，expression 是必需的表达式，它返回一个 Worksheet 对象。Before 是可选的 Variant 参数，它指定要在它前面放置复制副本或被移动的工作表的那个工作表。After 是可选的 Variant 参数，它指定要在它后面放置复制副本或被移动的工作表的那个工作表。

- ◆ 典型情况是，希望指定 Before 或 After，但不要二者均指定。
- ◆ 可以用名称来指定另一个工作簿，以便将工作表复制到或移动到另一个工作簿。
- ◆ 还可以把这两个参数都省略掉，让 Excel 创建一个包含被复制或被移动的工作表的新工作簿。新工作簿变成了活动工作簿，所以可以使用 ActiveWorkbook 对象开始对新工作簿进行工作，或将新工作簿指派给一个对象变量。

例如，下述语句复制名为 Building Schedule.xls 的工作簿中的名为 Costs- Materials 的工作表，并将复制副本放在该工作簿中最后一个当前工作表的后面：

```
Workbooks("Building Schedule.xls").Sheets("Costs - Materials").Copy,
After:=Sheets(Sheets.Count)
```

下述语句将名为 Planning.xls 的工作簿中的名为 Homes 的工作表，移动到名为 Building Schedule.xls 的工作簿中，并将该工作表插入该工作簿中第一个已有工作表的前面：

```
Workbooks("Planning.xls").Sheets("Homes").Move ,
Before:=Workbooks("Building Schedule.xls").Sheets(1)
```

打印工作表

要打印一个工作表，需使用适当的 Worksheet 对象的 PrintOut 方法。

提示：PrintOut 方法也应用于其他对象，包括 Worksheets 集合 Chart 对象和 Charts 集合，Workbook 对象，Window 对象以及 Range 对象。

与 PrintOut 方法对应的语法是：

```
expression.PrintOut(From, To, Copies, Preview, ActivePrinter, PrintToFile, Collate,
PrToFileName)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回适当的 Worksheet 对象或 PrintOut 方法要应用到的其他对象。
- ◆ From 是可选的 Variant 参数，它指定要从哪一页起开始打印，即打印的起始页码。省略 From，则是从该对象的起始处开始打印。

注意：From 和 To 指的是要打印的页码（要打印哪些页），不是指该对象总共多少页。

- ◆ To 是可选的 Variant 参数，它指定在哪一页上停止打印，即打印的结束页码。省略 To，则是打印到该对象结束处。
- ◆ Copies 是可选的 Variant 参数，它指定要打印的份数。如果省略了 Copies，Excel 只打印一份。
- ◆ Preview 是可选的 Variant 参数，如果将它设置为 True，在打印某一个对象之前，可以在“打印预览”中显示该对象。如果将 Preview 设置为 False 或简单地将这个参数

省略掉，那就只是打印该对象而不能预览它。

提示：使用 PrintPreview 方法，可以不打印某一个对象，而只是在“打印预览”中显示该对象。

◆ ActivePrinter 是可选的 Variant 参数，用来指定在哪一个打印机上进行打印。

◆ PrintToFile 是可选的 Variant 参数，将它设置为 True，是使 Excel 打印到某一个打印文件，而不是到某一个打印机。当打印到某一个文件时，可以使用 PrToFileName 属性来指定文件名，如果省略了该指定，Excel 会提示用户键入文件名。

◆ Collate 是可选的 Variant 参数，将它设置为 True，是让 Excel 逐份地打印出多份副本。而不是先打印出某一页的所有副本，再打印出下一页的所有副本，等等。

◆ PrToFileName 是可选的 Variant 参数，把它与 PrintToFile:=True 一起使用以指定打印文件的文件名。

例如，下述语句为活动工作簿中第一个工作表的每一页打印两份副本，需逐份印出：

```
ActiveWorkbook.Sheets(1).Printout Copies:=2, Collate:=True
```

下述语句打印名为 Planning.xls 的工作簿中的名为 Summary 的工作表的头两页，使它们打印到网络文件夹 \\server\\to-print 中名为 Planning Summary.prn 的文件：

```
Workbooks("Planning.xls").Sheets("Summary").PrintOut From:=1, To:=2, _  
PrintToFile:=True, PrToFileName:="\\server\\to_print\\Planning Summary.prn"
```

保护工作表

要保护一个工作表，需使用适当的 Worksheet 对象的 Protect 方法，其语法是：

```
expression.Protect(Password, DrawingObjects, Contents, Scenarios,  
UserInterfaceOnly, AllowFormattingCells, AllowFormattingColumns,  
AllowFormattingRows, AllowInsertingColumns, AllowInsertingRows,  
AllowInsertingHyperlinks, AllowDeletingColumns, AllowDeletingRows, AllowSorting,  
AllowFiltering, AllowUsingPivotTables)
```

该语法的各组成部分如下：

◆ expression 是必需的表达式，它返回一个 Worksheet 对象。

◆ Password 是可选的 Variant 参数，它指定要使工作表失去保护所对应的密码。 Password 是与字母大小写有关的。几乎总是要求提供 Password，以避免无特权人员使工作表失去保护。

◆ DrawingObjects 是可选的 Variant 参数，将它设置为 True 可以保护工作表中的各形状。默认设置是 False。

◆ Contents 是可选的 Variant 参数，将它设置为 True，可以保护锁定的单元格，这是它的默认值。将 Contents 设置为 False，则使锁定的单元格失去保护。

◆ Scenarios 是可选的 Variant 参数，将它设置为 True 可以保护方案，这是它的默认值。

◆ UserInterfaceOnly 是可选的 Variant 参数，将它设置为 True 时，可以在保护用户界面的同时取消对宏的保护。默认设置是 False。

◆ AllowFormattingCells、AllowFormattingColumns 和 AllowFormattingRows 都是可

选的 Variant 参数，将它们设置为 True，可以允许分别为单元格、列和行设置格式。每个参数对应的默认值都是 False。

- ◆ AllowInsertingColumns、AllowInsertingRows 和 AllowInsertingHyperlinks 都是可选的 Variant 参数，将它们设置为 True，可以允许用户分别插入列、行或超链接。每个参数对应的默认值都是 False。
- ◆ AllowDeletingColumns 和 AllowDeletingRows 都是可选的 Variant 参数，将它们设置为 True，可以允许用户分别删除列或行，要删除的列或行里的每个单元格都是解除锁定的。默认设置是 False。
- ◆ AllowSorting 是可选的 Variant 参数，将它设置为 True，可以允许用户对受保护的工作表上的未锁定单元格进行排序。默认设置是 False。
- ◆ AllowFiltering 是可选的 Variant 参数，将它设置为 True，可以允许用户在受保护的工作表上设置筛选条件或改变筛选条件（但是不能使自动筛选有效或无效）。默认设置是 False。
- ◆ AllowUsingPivotTables 是可选的 Variant 参数，将它设置为 True，可以允许用户在受保护的工作表上使用“数据透视表”进行工作。默认设置是 False。

例如，下述语句使用密码 no1get\$1n 来保护以对象变量 myWorksheet 引用的工作表：

```
myWorksheet.Protect Password:="no1get$1n"
```

下述语句使用与上面相同的密码来保护 myWorksheet 工作表，但允许为单元格设置格式，并允许对未锁定的单元格排序：

```
myWorksheet.Protect Password:="no1get$1n", AllowFormattingCells:=True, _  
    AllowSorting:=True
```

使用 ActiveSheet 对象进行工作

ActiveSheet 对象返回指定工作簿中的活动工作表，或者（如未指定工作簿的话）返回 Excel 中的活动工作表。

如果没有工作表是活动的，ActiveSheet 就返回 Nothing。在执行依赖于要有活动工作表的代码之前，对此进行检查是一个好主意，例如：

```
If ActiveSheet Is Nothing Then End
```

使用 ActiveCell 或 Selection 进行工作

在对用户选定的内容进行操作的过程里，通常要针对活动单元格或所选内容进行工作。活动单元格总是单个单元格，但所选内容可以包含多个单元格或其他对象。

使用 ActiveCell 进行工作

Application 对象或 Window 对象的 ActiveCell 属性返回表示 Excel 应用程序或指定窗口中的活动单元格的一个 Range 对象。如果使用 ActiveCell 但不指定窗口，VBA 便返回活动窗口中的活动单元格。

例如，下述语句返回活动工作簿中活动单元格的地址：

```
ActiveCell.Address
```

下述语句返回在名为 Planning.xls 的工作簿上打开的第一个窗口中的活动单元格内的文本：

```
MsgBox Workbooks("Planning.xls").Windows(1).ActiveCell.Text
```

如果没有工作表是活动的，或者是如果有一个图表工作表是活动的，那么就没有活动单元格。如果试图访问 ActiveCell，VBA 便返回一个错误。所以，在使用假定有一个活动单元格的代码之前，需检查 ActiveCell 是否是 Nothing：

```
If ActiveCell Is Nothing Then End
```

```
    If ActiveCell Is Nothing Then
        ActiveCell.Font.Name = "Times New Roman"
        ActiveCell.Font.Size = 12
        ActiveCell.Font.Color = RGB(0, 0, 255)
    End If
```

获得和设置活动单元格的值

要返回活动单元格的值，需使用 Value 属性。例如，下述语句将活动单元格的值设置为 25：

```
ActiveCell.Value = 25
```

移动单元格至另一地址

使用活动单元格来工作往往很方便，所以有时希望另一个单元格变成活动单元格以便通过 ActiveCell 对象来使用它进行工作。要使一个单元格成为活动单元格，需使用针对适当的 Range 对象的 Activate 方法。例如，下述语句使单元格 B5 成为以对象变量 myWorksheet 来标识的工作表中的活动单元格：

```
myWorksheet.Range("B5").Activate
```

通常，需要把活动单元格偏移指定的行数或列数，即移至另一个不同区域。要做到这一点，需使用活动单元格对象的 Offset 属性，以 RowOffset 参数来指定需偏移的行数，以 ColumnOffset 参数来指定需偏移的列数。将活动单元格右移或下移是正偏移，将活动单元格左移或上移则是负偏移。例如，下述语句将活动单元格上移两行 (RowOffset: = -2) 并右移四列 (ColumnOffset: = 4)：

```
ActiveCell.Offset(RowOffset:=-2, ColumnOffset:=4).Activate
```

在用户触发的过程里，往往希望使活动单元格返回到用户启动过程时该单元格所在的地方。为此，可以把活动单元格的位置先存储起来，然后让它返回到这个被存储的位置。例如：

```
Set myActiveCell = ActiveCell
Set myActiveWorksheet = ActiveSheet
Set myActiveWorkbook = ActiveWorkbook
```

```
'take actions here
```

```
myActiveWorkbook.Activate
myActiveWorksheet.Activate
myActiveCell.Activate
```

警告：要经常使用各种数据类型仔细测试各个过程。如果移动了含有使用相对单元格地址的等式的单元格，有时会产生错误。

使用环绕活动单元格的区域进行工作

使用 CurrentRegion 属性，可以针对环绕活动单元格的单元格区域进行工作，而返回 CurrentRegion 对象。当前区域从活动单元格开始，向上和向下各扩展到第一个空白行，向左和向右各扩展到第一个空白列。例如，下述语句使用 CurrentRegion 对象的 Font 属性，将当前区域的字体设置为不带粗体或斜体的 12 点 Times New Roman 字体：

```
With ActiveCell.CurrentRegion.Font
    .Name = "Times New Roman"
    .Size = 12
    .Bold = False
    .Italic = False
End With
```

使用 Selection 进行工作

在设计成由用户来运行的宏里，常常需要对用户已选定的单元格进行工作。例如，用户可能会选择一个单元格区域，然后运行一个宏来操控该区域的内容。

要对用户选定区域进行工作，需使用适当的 Window 对象的 RangeSelection 属性。例如，可以把 RangeSelection 属性指派给某一个区域，这样就能在宏里面使用它进行工作，然后在该宏的结束处再次选定它，让用户准备再次对这个选定内容进行工作：

```
Dim myMacroRange As Range
Set myMacroRange = ActiveWindow.RangeSelection
With myMacroRange
    'take actions on the range here
End With
myMacroRangeActivate
```

使用 Range 进行工作

在工作表之内，常常需要对单元格区域进行操作。可以针对绝对区域（要为这种区域指定想对其施加影响的单元格的绝对地址）来工作，也可以针对相对于活动单元格的区域来工作。

可以使用 Range 属性来指定一个区域，也可以使用 Names 集合来创建一个带名称的区域。Excel 还提供了 UsedRange 属性，以便针对工作表上已被使用的区域进行工作，并提供了 Range 对象的 SpecialCells 方法，以便针对满足特定条件的单元格进行工作。

使用单元格区域进行工作

要针对单元格区域进行工作，需使用适当的 Worksheet 对象的 Range 属性以指定这些单元格。例如，下述语句将活动工作表上单元格 C12 的值设置为 44：

```
ActiveSheet.Range("C12").Value = "44"
```

创建带名称的区域

要创建带名称的区域，需使用针对 Names 集合的 Add 方法，其语法是：

```
expression.Add(Name, RefersTo, Visible, MacroType, ShortcutKey, Category,  
NameLocal, RefersToLocal, CategoryLocal, RefersToR1C1, RefersToR1C1Local)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Names 对象。
- ◆ Name 是可选的 Variant 参数，它指定分派给带名称的区域的名称。如果没有指定 NameLocal 参数（见下述），就必须要有 Name 参数。名称不能是单元格引用，也不能包含空格。
- ◆ RefersTo 是可选的 Variant 参数，它指定与带名称区域相对应的区域。如果没有使用 RefersToLocal 参数、RefersToR1C1 参数或 RefersToR1C1Local 参数，就必须指定 RefersTo。
- ◆ Visible 是可选的 Variant 参数，省略此参数或将其设置为 True，Excel 会使用户界面中（在“定义名称”对话框中、“粘贴名称”对话框中以及其他位置上）能够看到名称；将它设置为 False，则使名称被隐藏。
- ◆ MacroType 是可选的 Variant 参数，用来把宏的类型指派给区域：1 对应于用户自定义的 Function 过程，2 对应于 Sub 过程，3 或省略对应于没有宏。
- ◆ ShortcutKey 是可选的 Variant 参数，它为指派给带名称区域的命令宏指定快捷键。
- ◆ Category 是可选的 Variant 参数，它指定由 MacroType 指定的宏或函数的种类。可以指定由 Function Wizard 使用的一个种类或指定另一个名称，使 Excel 创建一个使用这个名称的新种类。
- ◆ NameLocal 是可选的 Variant 参数，它以本地语言来为区域指定名称。未用 Name 时，必须使用 NameLocal。
- ◆ RefersToLocal 是可选的 Variant 参数，它为带名称的区域指定区域。未用 RefersTo、RefersToR1C1 和 RefersToR1C1Local 时，必须使用 RefersToLocal。
- ◆ CategoryLocal 是可选的 Variant 参数，用来指定由 MacroType 指定的宏或函数的种类。未用 Category 时，必须使用 CategoryLocal。
- ◆ RefersToR1C1 是可选的 Variant 参数，它使用 R1C1 样式的记号指定与带名称区域相对应的区域。未用 RefersTo、RefersToLocal 和 RefersToR1C1Local 时，必须使用 RefersToR1C1。
- ◆ RefersToR1C1Local 是可选的 Variant 参数，它使用 R1C1 样式的记号，以本地语言指定与带名称区域相对应的区域。未用 RefersTo、RefersToLocal 和 RefersToR1C1 时，必须使用 RefersToR1C1Local。

例如，下述语句定义一个名为 myRange 的区域，它引用名为 Building Schedule.xls 的工作簿中名为 Materials 的工作表上的区域 A1: G22：

```
Workbooks("Building Schedule.xls").Names.Add Name:="myRange", _  
RefersTo:="=Materials!$A$1:$G$22"
```

删除带名称的区域

删除带名称的区域

要删除一个带名称的区域，需使用针对适当的 Name 对象的 Delete 方法。例如，下述语句删除名为 Building Schedule.xls 的工作簿中名为 myRange 的区域：

```
Workbooks("Building Schedule.xls").Names("myRange").Delete
```

使用带名称的区域进行工作

要使用带名称的区域进行工作，需指定与 Range 对象相对应的名称。例如，下述语句将带名称区域 myRange 中各行的行高度设置为 20 点，并将 16 点 Arial 字体应用到各单元格：

```
With Range("myRange")
    .RowHeight = 20
    .Font.Name = "Arial"
    .Font.Size = 16
End With
```

使用已用区域进行工作

如果必须对工作表上的所有单元格但不包括工作表的任何未占用区域进行工作，需使用 UsedRange 属性。例如，下述语句自动调整活动工作表中已用区域内所有列的大小：

```
ActiveSheet.UsedRange.Columns.AutoFit
```

使用特定单元格进行工作

如果仅需要针对工作表上或某一个区域中某些类型的单元格进行工作，需使用 Range 对象的 SpecialCells 方法以返回所需的单元格。其语法是：

expression.SpecialCells(Type, Value)

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Range 对象。
- ◆ Type 是必需的参数，它指定需用哪些单元格。表 22.4 列出了可以使用的常量。

表 22.4 与 SpecialCells 方法中 Type 参数对应的常量

常量	返回单元格的类型
xlCellTypeAllFormatConditions	任意格式单元格
xlCellTypeAllValidation	含有验证条件的单元格
xlCellTypeBlanks	空单元格
xlCellTypeComments	含有注释的单元格
xlCellTypeConstants	含有常量的单元格
xlCellTypeFormulas	含有公式的单元格
xlCellTypeLastCell	已用区域中最后的单元格
xlCellTypeSameFormatConditions	含有相同格式的单元格
xlCellTypeSameValidation	含有相同验证条件的单元格
xlCellTypeVisible	所有可见单元格

- ◆ Value 是可选的 Variant 参数，当 Type 为 xlCellTypeConstants 或 xlCellTypeFormulas 时，使用该参数可以控制 Excel 包括哪些单元格。表 22.5 列出了各常量及它们返回的内容。

表 22.5 与 SpecialCells 方法中 Value 参数对应的常量

常量	返回单元格包含的内容
xlErrors	错误
xlLogical	逻辑值
xlNumbers	数字
xlTextValues	文本公式

例如，下述语句激活以对象变量 myWorksheet 引用的工作表中的最后单元格：

```
myWorksheet.Cells.SpecialCells(Type:=xlCellTypeLastCell).Activate
```

下述语句识别活动工作表中包含导致错误的公式的所有单元格：

```
ActiveSheet.Cells.SpecialCells(Type:=xlCellTypeFormulas, _  
Value:=xlErrors).Activate
```

将公式输入单元格

要把公式输入单元格，需设置适当的 Cell 对象的 Formula 属性。例如，下述语句将公式 = SUM (\$G\$12: \$G\$22) 输入活动单元格：

```
ActiveCell.Formula = "=SUM($G$12:$G$22)"
```

设置 Options 对象

在 Word 中，通过 Options 对象，在“选项”对话框中能找到的大部分选项都是可用的。与 Word 不同，Excel 把出自“选项”对话框的大部分选项都保持在 Application 对象中。通过适当的 Workbook 对象，可以访问“选项”对话框中出现的、为工作簿所特有的各个属性。

在 Application 对象中设置选项

本节将介绍三个例子，用于介绍在 Application 对象中设置广泛使用的选项。

控制 Excel 的计算

在需要进行很多计算的复杂工作表中，可能有必要把自动计算功能关闭，这样就可以在不产生计算的情况下，使过程能够迅速输入数据。为此，先将 Application 对象的 Calculation 属性设置为 xlCalculationManual，输入数据，然后将 Calculation 属性设回到它原先的值。例如：

```
Dim varAutoCalculation As Variant  
varAutoCalculation = Application.Calculation
```

```
Application.Calculation = xlCalculationManual
'enter the data here
Application.Calculation = xlCalculationAutomatic
```

清除最近使用过的文件列表

有时候，从 Excel 中“文件”菜单底部的“最近使用过的文件”列表中清除所有条目是很有用的。要做到这一点，需将 RecentFiles 对象的 Maximum 属性设置为 0。这样做过之后，可能仍需要恢复原先的设置，如下例所示：

```
Dim myMax As Long
With Application.RecentFiles
    myMax = .Maximum
    .Maximum = 0
    .Maximum = myMax
End With
```

设置默认的文件位置

要为保存和打开文件设置默认位置，需使用 Application 对象的 DefaultFilePath 属性。例如：

```
Application.DefaultFilePath = "\\server3\users\mjones\files"
```

在工作簿中设置选项

以下是工作簿所特有的选项：

- ◆ 安全性选项（例如下面两小节中介绍的选项）；
- ◆ 是否要更新工作簿中的远程引用（布尔型属性 UpdateRemoteReferences），以及是否要保存外部链接值（布尔型属性 SaveLinkValues）；
- ◆ 是否要使用“自动恢复”（布尔型属性 EnableAutoRecover）；
- ◆ 是否接受公式中的标签（布尔型属性 AcceptLabelsInFormulas），以及是否使用 1904 日期系统（布尔型属性 Date1904）。

设置 Excel 以便从文件属性中移出个人信息

要使 Excel 从工作簿的属性中移出个人信息以便保存，需将该工作簿的 RemovePersonalInformation 属性设置为 True：

```
ActiveWorkbook.RemovePersonalInformation = True
```

设置针对工作簿的密码和只读建议

为了保护某一个工作簿，防止无特权人员打开它或修改它，可以在该工作簿上设置打开工作簿所需要的密码或修改工作簿所需要的密码。Office 的保护功能相对较弱，但是在典型的办公环境下还是管用的。也可以这样指定，即当有人打开工作簿时，Excel 推荐他们以只读方式而不是以读/写方式打开工作簿。这种保护几乎没什么用处，但是可以让善于思考的

同事停下来想一想。

要设置打开工作簿所需要的密码，需设置 Workbook 对象的 Password 属性。例如，下述语句将用于打开活动工作簿的密码设置为 1mPass4：

```
ActiveWorkbook.Password = "1mPass4"
```

要设置修改工作簿所需要的密码，需设置 Workbook 对象的 WritePassword 属性。例如，下述语句将用于修改活动工作簿的密码设置为 n0mods：

```
ActiveWorkbook.WritePassword = "n0mods"
```

要把只读建议应用于某一个工作簿，需将它的 ReadOnlyRecommended 属性设置为 True。例如：

```
Workbooks("Strategy.xls").ReadOnlyRecommended = True
```

第 23 章 使用 Excel 中广泛使用的对象进行工作

- ◆ 使用 Charts 进行工作

- ◆ 使用 Windows 进行工作

- ◆ 使用 Find 和 Replace 进行工作

在前一章中，我们学习了如何针对 Excel 对象模型中的一些主要对象（例如，Workbook 对象、ActiveCell 对象、Range 对象以及 Options 对象）进行工作。本章介绍如何针对图表、窗口以及查找和替换来进行工作，以便在 Excel 中使用 VBA 来采取更多的行动。

使用 Charts 进行工作

本节介绍如何使用 VBA 来创建图表，以及如何为图表设置格式，这些图表可以作为工作簿中的图表工作表，也可以作为已有工作表上的对象。

创建图表

既可以在新图表工作表上创建新图表，也可以把新图表作为已有工作表中的对象来创建。VBA 使用 Chart 对象来表示图表工作表上的一个图表，使用 ChartObject 对象来表示工作表上的嵌入图表。ChartObject 对象包含一个 Chart 对象，通过在 ChartObject 之内访问 Chart 对象的方式，可以对 Chart 对象进行操控。

不管使用哪一种方法，都可以使用与交互方式工作时不同的顺序来创建图表或图表对象：

- ◆ 首先，添加图表或图表对象。
- ◆ 第二，为它的数据指定源区域。
- ◆ 第三，指定图表类型。
- ◆ 第四，指定想要添加的任何其他项目。

在新图表工作表上创建图表

要在新图表工作表上创建一个图表，需使用与 Charts 集合相对应的 Add 方法，其语法是：

expression.Add(Before, After, Count)

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Charts 集合。
- ◆ Before 是可选的 Variant 参数，用来指定在它前面要添加新的图表工作表的那个工作表。After 是可选的 Variant 参数，用来指定在它后面要添加新工作表的那个工作表。典型情况是指定 Before 或者 After。如果两个参数都省略，VBA 将在活动工作表之前添加新的图表工作表。

◆ Count 是可选的 Variant 参数，用来指定要添加多少个图表工作表。默认值为 1。例如，下述语句先将一个名为 myChartSheet 的对象变量声明为 Chart 类型（图表工作表），然后把添加到活动工作簿中最后一个已有工作表之后的新图表工作表指派给 myChartSheet：

```
Dim myChartSheet As Chart
Set myChartSheet = ActiveWorkbook.Sheets.Add _
(After:=ActiveWorkbook.Sheets(ActiveWorkbook.Sheets.Count), _
Type:=xlChart)
```

在已有工作表上创建图表

要在已有工作表上创建一个图表，需使用与 ChartObjects 集合相对应的 Add 方法，其语法是：

```
expression.Add(Left, Top, Width, Height)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 ChartObjects 集合。
- ◆ Left 是必需的双精度浮点型参数，它指定相对于单元格 A1 的左边缘，该图表左上角的位置，以点计。
- ◆ Top 是必需的双精度浮点型参数，它指定相对于单元格 A1 的顶边，该图表的左上角的位置，以点计。
- ◆ Width 是必需的双精度浮点型参数，它指定图表的宽度，以点计。
- ◆ Height 是必需的双精度浮点型参数，它指定图表的高度，以点计。

例如，下述语句声明一个名为 myChartObject 的新的 ChartObject 对象，并把宽为 400 点、高为 300 点、距离工作表左边缘 200 点、距离工作表顶边 200 点的一个新图表对象指派给它：

```
Dim myChartObject As ChartObject
Set myChartObject = ActiveSheet.ChartObjects.Add(Left:=200, Top:=200, _
Width:=400, Height:=300)
```

要对 ChartObject 内部的图表进行工作，需返回 ChartObject 对象的 Chart 属性。

指定图表的源数据区域

到目前为止，图表（图表工作表上的或是图表对象中的）都还是空白的。要给它赋予内容，需使用 Chart 对象的 SetSourceData 方法来指定图表的源数据区域。例如，下述语句指定活动工作簿中名为 Chart Data 的工作表上的区域 A1: E5，作为名为 myChartObject 的 ChartObject 对象中的 Chart 对象的数据源区域：

```
myChartObject.Chart.SetSourceData Source:= _
ActiveWorkbook.Sheets("Chart Data").Range("A1:E5")
```

指定图表类型

要指定图表类型，需设置 Chart 对象的 ChartType 属性。Excel 提供了非常丰富的图表

类型，在这里就不一一列出了，但是从它们的常量中能够很容易识别出图表类型。例如，常量 xl3DArea 表示三维面积图，xlColumnStacked 表示堆积柱形图，xlDoughnutExploded 表示分离型圆环图，等等。

下述语句将对象变量 myChart 所代表的图表的类型设置为堆积柱形图的型式：

```
myChart.ChartType = xlColumnStacked
```

使用图表中的系列进行工作

要对图表中的系列进行工作，需使用 SeriesCollection 集合，它包含指定图表中的全部系列。

其一：**添加新序列**

要给 SeriesCollection 集合添加一个新系列，需使用与适当的 SeriesCollection 对象相对应的 Add 方法，其语法是：

```
expression.Add(Source, Rowcol, SeriesLabels, CategoryLabels, Replace)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 SeriesCollection 集合。
- ◆ Source 是必需的 Variant 参数，它指定与新序列对应的数据源。此数据可用一个区域来提供，或用一个数据点数组来提供。
- ◆ Rowcol 是可选参数，设置为 xlRows 时，是指明新值在指定区域的行中。使用默认设置，即设置为 xlColumns 时，是指明新值在指定区域的列中。如果省略这个参数，Excel 使用 xlColumns。
- ◆ SeriesLabels 是可选的 Variant 参数，将它设置为 True 时，指定源区域中的第一行或第一列含有系列标签；将它设置为 False 时，指定源区域中的第一行或第一列包含的是该系列的第一个数据点。如果省略这个参数，需由 Excel 去搞清第一行或第一列是否包含有系列标签。最好指明此参数以避免混乱。如果 Source 是一个数组，那么 VBA 会忽略此参数。
- ◆ CategoryLabels 是可选的 Variant 参数，将它设置为 True 时，指定第一行或第一列含有分类标签的名称；设置为 False 时，不包含分类标签。如果省略这个参数，需由 Excel 去搞清第一行或第一列是否包含有分类标签。最好设置这个参数以避免混乱。如果 Source 是一个数组，那么 VBA 会忽略此参数。
- ◆ Replace 是可选的 Variant 参数，在 CategoryLabels 为 True 时将它设置为 True，是要以指定的与该系列对应的分类来取代该系列已有的分类。如果将它设置为 False（默认值），那么已有的分类不被取代。

例如，下述语句在以对象变量 myChart 为标识的图表中添加一个新系列，使用行从活动工作簿中活动工作表上的区域 C4:K4 里取出数据：

```
myChart.SeriesCollection.Add Source:=ActiveSheet.Range("C4:K4"), Rowcol:=xlRows
```

扩展已有系列

要扩展一个已有系列，需使用与适当的 SeriesCollection 对象相对应的 Extend 方法，其语法是：

```
expression.Extend(Source, Rowcol, CategoryLabels)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 SeriesCollection 对象。
- ◆ Source 是必需的 Variant 参数，它指定与新系列对应的数据源，此数据可用一个区域或一个数据点数组来提供。
- ◆ Rowcol 是可选参数。设置为 xlRows 时，是指明新值在指定区域的行中。使用默认设置，即设置为 xlColumns 时，是指明新值在指定区域的列中。如果省略这个参数，Excel 使用 xlColumns。
- ◆ CategoryLabels 是可选的 Variant 参数，将它设置为 True 时，指定第一行或第一列含有分类标签的名称；设置为 False 时，不包含分类标签。如果省略这个参数，需由 Excel 来搞清第一行或第一列是否包含有分类标签。最好设置这个参数以避免混乱。如果 Source 是一个数组，那么 VBA 会忽略此参数。

例如，下述语句使用 Chart Data 工作表上单元格 P3：P8 之内的数据，对以对象变量 myChart 为标识的图表中的系列进行扩展：

```
myChart.SeriesCollection.Extend Source:=Worksheets("Chart Data").Range("P3:P8")
```

创建新系列

要创建一个新系列，需使用与 SeriesCollection 集合相对应的 NewSeries 方法。例如，下述语句将一个新系列添加到对象变量 myChart 所表示的图表中：

```
myChart.SeriesCollection.NewSeries
```

给图表添加图例

要给图表添加一个图例，需将它的 HasLegend 属性设置为 True。要对图例进行操作，需对 Legend 对象的属性进行工作。其重要属性如下：

- ◆ Position 属性，它控制图例出现在何处：xlLegendPositionBottom、xlLegendPositionCorner、xlLegendPositionLeft、xlLegendPositionRight 或者 xlLegendPositionTop。
- ◆ Height 属性和 Width 属性分别控制图例的高度和宽度，以点计。
- ◆ Font 属性，它返回 Font 对象，可以设置其属性以指定字体大小、名称和效果。

例如，下述语句将图例添加到对象变量 myChart 所表示的图表中，并将 16 点 Arial 字体应用于该图例：

```
With myChart.Legend
    .HasLegend = True
    .Font.Size = 16
    .Font.Name = "Arial"
End With
```

给图表添加标题

要给图表添加一个标题，需将它的 HasTitle 属性设置为 True，例如：

```
myChart.HasTitle = True
```

Excel 以默认文本 Chart Title 来添加标题。要更改此文本，需设置表示图表标题的 ChartTitle 对象的 Text 属性。例如：

```
myChart.ChartTitle.Text = "Industrial Disease in North Dakota"
```

为了给标题定位，需设置它的 Top 属性（指定距离工作表顶边的点数）和它的 Left 属性（指定距离工作表左边缘的点数）。例如：

```
With myChart.ChartTitle
    .Top = 100
    .Left = 150
End With
```

为了给标题的文本设置格式，需使用它的 Font 对象，例如：

```
myChart.ChartTitle.Font.Name = "Elephant"
```

使用图表的坐标轴进行工作

要使用图表的坐标轴进行工作，需使用 Axes 方法来访问适当的坐标轴。其语法是：

```
expression.Axes(Type, Group)
```

此处，expression 是必需的表达式，它返回一个 Chart 对象。Type 是可选的 Variant 参数，它指定要返回的坐标轴。使用 xlValue 以返回数值坐标轴，使用 xlCategory 以返回分类坐标轴，或使用 xlSeriesAxis 以返回系列坐标轴（只用于三维图表）。Group 是可选参数，把它设置为 xlSecondary 是指定第二坐标轴组；xlPrimary 是默认设置，用于指定第一坐标轴组（主坐标轴组）。

例如，下述语句对图表的主坐标轴组中的分类坐标轴进行工作，应用标题，添加文本，设置字体和字体大小，使主要网格线显示而使次要网格线不显示：

```
With .Axes(Type:=xlCategory, AxisGroup:=xlPrimary)
    .HasTitle = True
    .AxisTitle.Text = "Years"
    .AxisTitle.Font.Name = "Times New Roman"
    .AxisTitle.Font.Size = 12
    .HasMajorGridlines = True
    .HasMinorGridlines = False
End With
```

使用 Windows 进行工作

Windows 集合包含 Excel 应用程序中与每一个打开的窗口相对应的 Window 对象。正常情况下，当打开一个工作簿时，Excel 就打开一个窗口，所以能看到那个工作簿。必要时还

可以进一步打开一些窗口——例如在用户界面中选择“窗口”>“新建窗口”。

在大多数场合下，要通过 VBA 来访问数据，窗口并不是很有用，这里因为使用诸如 ActiveSheet 或 ActiveCell 这些对象来访问数据要容易一些。但是，有可能希望以程序方式打开、关闭、激活或排列若干窗口，以便用特定的方式向用户显示数据。

在工作簿上打开新窗口

要在工作簿上打开一个新窗口，需使用适当的 Window 对象的 NewWindow 方法。这个方法不用参数。例如，下述语句打开一个新窗口，它显示在以对象变量 myWorkbook 为标识的工作簿上打开的第一个窗口的内容：

```
myWorkbook.Windows(1).NewWindow
```

关闭窗口

要关闭一个窗口，需使用针对适当的 Window 对象的 Close 方法，其语法是：

```
expression.Close(SaveChanges, Filename, RouteWorkbook)
```

此处，expression 是必需的表达式，它返回一个 Window 对象。这一语法与对应于关闭一个工作簿的方法相同（见上一章的“关闭工作簿”一节）。区别只在于，如果在同一工作簿上打开了两个或多个窗口，关闭第二个或以后各个窗口时不会关闭该工作簿，所以各参数是不相干的。（但是，如果正在关闭的是工作簿的最后一个窗口，就需要指定窗口——否则，Excel 会提示用户保存任何未保存的更改。）例如，下述语句关闭在以对象变量 myWorkbook 引用的工作簿上打开的所有窗口，但留下一个窗口不关闭：

```
Do While myWorkbook.Windows.Count > 1  
    myWorkbook.Windows(myWorkbook.Windows.Count).Close  
Loop
```

激活窗口

要激活一个窗口，需使用适当的 Window 对象的 Activate 方法。例如，下述语句激活在工作簿 Planning.xls 上打开的第一个窗口：

```
Workbooks("Planning.xls").Windows(1).Activate
```

与此类似，可以使用 ActivatePrevious 方法激活上一个窗口，或使用 ActivateNext 方法激活下一个窗口。

排列窗口和调整窗口的大小

要排列一些窗口，需使用与适当的 Windows 集合对应的 Arrange 方法，其语法是：

```
expression.Arrange(ArrangeStyle, ActiveWorkbook, SyncHorizontal, SyncVertical)
```

这一语法的各组成部分如下：

◆ expression 是必需的表达式，它返回一个 Windows 集合。

◆ ArrangeStyle 是可选的参数，将它设置为设置为 xlArrangeStyleTiled，是平铺各窗口

（默认设置），设置为 `xlArrangeStyleHorizontal` 是水平排列所有窗口，设置为 `xlArrangeStyleVertical` 是垂直排列所有窗口，设置为 `xlArrangeStyleCascade` 是对各窗口进行层叠，让人看到每个窗口的标题栏，但只能看到最前面窗口的内容。

- ◆ `ActiveWorkbook` 是可选的 Variant 参数，将它设置为 `True` 是使 VBA 只对活动工作簿的各窗口进行排列。默认设置为 `False`，则是对所有打开的窗口进行排列。
- ◆ `SyncHorizontal` 和 `SyncVertical` 是可选的 Variant 参数，在使用 `ActiveWorkbook` 为 `True` 时，将它们设置为 `True` 是使活动工作簿的各窗口同步地水平滚动或垂直滚动（当滚动一个窗口时，其他各窗口也朝同一方向做等量滚动）。默认设置为 `False`。

例如，下述语句垂直排列工作簿 `Budget.xls` 内的各窗口，并使它们可以同步垂直滚动：

```
Workbooks("Budget.xls").Windows.Arrange ArrangeStyle:=xlArrangeStyleVertical, _
    ActiveWorkbook:=True, SyncVertical:=True
```

将 `Application` 对象的 `WindowState` 属性设置为 `xlMaximized`、`xlMinimized` 或 `xlNormal`，可以最大化、最小化或恢复应用程序窗口。类似地，在应用程序窗口之中，通过设置某一文档的 `WindowState` 属性，可以最大化、最小化或恢复该文档。

当某一个窗口处于“正常”状态时（`xlNormal`；既不是最大化，也不是最小化），可以对它进行定位（即使用 `Top` 和 `Left` 属性来指定该窗口的左上角的位置），还可以设置它的 `Height` 和 `Width` 属性以确定它的大小。检查 `Application` 对象的 `UsableWidth` 和 `UsableHeight` 属性，可以找出 `Application` 窗口的可用空间有多大。（同样，检查 `Window` 对象的 `UsableWidth` 属性和 `UsableHeight` 属性，也可以看到窗口中有多大可用空间——这样便于正确地确定某一个对象的大小和位置。）

下面的例子先声明两个 `Window` 对象变量 `myWindow1` 和 `myWindow2`，将 `myWindow1` 指派为活动窗口，将 `myWindow2` 指派为新窗口，它与 `myWindow1` 一样，显示相同的工作表。然后为这两个窗口确定大小和位置，使得每个窗口都占满应用程序窗口的整个可用高度，但是 `myWindow1` 只占可用宽度的四分之一，而 `myWindow2` 占可用宽度的四分之三：

```
Dim myWindow1 As Window, myWindow2 As Window
Set myWindow1 = ActiveWindow
Set myWindow2 = myWindow1.NewWindow
With myWindow1
    .WindowState = xlNormal
    .Top = 0
    .Left = 0
    .Height = Application.UsableHeight
    .Width = Application.UsableWidth * 0.25
End With
With myWindow2
    .WindowState = xlNormal
    .Top = 0
    .Left = (Application.UsableWidth * 0.25) + 1
    .Height = Application.UsableHeight
    .Width = Application.UsableWidth * 0.75
End With
```

缩放窗口和设置显示选项

要改变缩放比，需设置适当的 `Window` 对象的 `Zoom` 属性。例如，下述语句将活动窗口

放大到 150%：

```
ActiveWindow.Zoom = 150
```

在某些应用中，可能需要改变 Excel 窗口的显示，以便确保某些特性能（或不能）为用户所用。使用布尔型属性 DisplayScrollBars、DisplayStatusBar 和 DisplayFormulaBar 来控制 Excel 是否显示滚动条、状态栏和公式栏。使用 DisplayFullScreen 属性来切换整屏显示的通和断。

例如，下述语句确保滚动条和状态栏被隐藏，而公式栏被显示：

```
With Application  
    .DisplayScrollBars = False  
    .DisplayStatusBar = False  
    .DisplayFormulaBar = True  
End With
```

使用 Find 和 Replace 进行工作

Excel 的查找和替换特性对于在过程中定位数据是很有用的。在 Excel 中，查找和替换是通过方法来实现的，而不像 Word 那样通过 Find 对象来实现。

Find 方法和 Replace 方法应用于（使用不同的语法）Range 对象和 WorksheetFunction 对象。对于大多数查找和替换操作，希望针对 Range 对象来使用这些方法——例如，替换某一个工作表上一些指定单元格的内容。

使用 Find 方法进行搜索

对应于 Range 对象的 Find 方法的语法是：

```
expression.Find(What, After, LookIn, LookAt, SearchOrder, SearchDirection,  
MatchCase, MatchByte, SearchFormat)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Range 对象。
- ◆ What 是必需的 Variant 参数，它指定要查找的数据。该数据可以是文本的一个字符串或者任何 Excel 数据。
- ◆ After 是可选的 Variant 参数，用来指定单元格，搜索即从该单元格之后开始。After 必须是被搜索区域中的一个单元格。如果省略 After，Excel 从区域的左上角单元格处开始搜索。
- ◆ LookIn 是可选的 Variant 参数，使用它来指定是在公式中 (xlFormulas)、数值中 (xlValues) 还是注释中 (xlComments) 搜索。
- ◆ LookAt 是可选的 Variant 参数，将它指定为 xlWhole，是搜索单元格的全部内容；将它指定为 xlPart，是搜索单元格内容之中的匹配部分。（这个设置与“查找和替换”对话框中“单元格匹配”复选框是对应的。）
- ◆ SearchOrder 是可选的 Variant 参数，将它设置为 xlByRows，是按各行的顺序进行搜索；设置为 xlByColumns，是按各列的顺序进行搜索。

- ◆ SearchDirection 是可选的 Variant 参数，将它设置为 xlNext，是向下搜索；设置为 xlPrevious，是向上搜索。
- ◆ MatchCase 是可选的 Variant 参数，将它设置为 True，是进行区分大小写的搜索。默认设置为 False。
- ◆ MatchByte 是可选的 Variant 参数，仅在已安装了双字节语言支持时使用。
- ◆ SearchFormat 是可选的 Variant 参数，它控制 Excel 要 (True) 还是不要 (False) 搜索指定的格式设置。

警告：LookIn、LookAt、SearchOrder 和 MatchByte 这些参数是“有黏性的”——Excel 将这些参数的设置从一次搜索保留到下一次搜索。所以，如果没有搞清楚，最近一次用过的设置是否适合自己的需要，就应该在每次搜索时明显地设置这些参数，以避免出现不希望的结果。

例如，下述语句在活动单元格之后的各单元格内的公式中查找 2008，不查找格式设置：

```
Cells.Find(What:="2008", After:=ActiveCell, LookIn:=xlFormulas, LookAt _  
:=xlWhole, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _  
True, SearchFormat:=False).Activate
```

使用 FindNext 方法和 FindPrevious 方法继续搜索

已经使用 Find 方法执行了一次搜索之后，可以使用 FindNext 方法来查找下一个匹配相同条件的实例，或使用 FindPrevious 方法来查找匹配相同条件的前一个实例。它们的语法是：

```
expression.FindNext(After)  
expression.FindPrevious(After)
```

此处，expression 是必需的表达式，它返回一个 Range 对象。After 是可选的 Variant 参数，它指定一个单元格，对应于 FindNext 方法，是要从该单元格之后搜索；对应于 FindPrevious 方法，是要从该单元格之前搜索。After 必须是一个单个的单元格。

例如，下述语句查找下一个匹配相同条件的实例：

```
Cells.FindNext
```

使用 Replace 方法进行替换

要使用 VBA 进行替换，需使用与 Range 对象相对应的 Replace 方法，其语法是：

```
expression.Replace(What, Replacement, LookAt, SearchOrder, MatchCase, MatchByte,  
SearchFormat, ReplaceFormat)
```

该语法的大部分组成部分与 Find 方法对应的语法的组成部分相同，但有两个是特有的：

- ◆ Replacement 是必需的 Variant 参数，它指定与查找相对应的替换字符串。
- ◆ ReplaceFormat 是可选的 Variant 参数，它控制 Excel 要 (True) 还是不要 (False) 替换格式设置。

例如，下述语句把活动工作表 B 列中的字 Sales，用 Sales & Marketing 来替换，并且使用区分大小写匹配：

```
ActiveSheet.Columns("B").Replace What:="Sales", _  
    Replacement:="Sales & Marketing", SearchOrder:=xlByColumns, _  
    MatchCase:=True
```

搜索和替换格式设置

为了搜索格式设置，先使用 Application 对象的 FindFormat 属性以定义格式设置，然后将 Find 方法的 SearchFormat 参数设置为 True。类似地，先使用 Application 对象的 ReplaceFormat 属性以定义替换格式设置，然后将 Replace 方法的 ReplaceFormat 属性设置为 True。

例如，下述语句使用一个 With 结构来设置要进行搜索的 Application. FindFormat. Font 属性，使用另一个 With 结构来设置要用做替换的 Application. ReplaceFormat. Font 属性，并使用 Cells 集合的 Replace 方法来实现替换：

```
With Application.FindFormat.Font  
    .Name = "Arial"  
    .Size = 12  
    .Bold = True  
End With  
With Application.ReplaceFormat.Font  
    .Name = "Arial Black"  
    .Bold = False  
End With  
Cells.Replace What:="5", Replacement:="5", LookAt:=xlPart, SearchOrder _  
:=xlByColumns, MatchCase:=False, SearchFormat:=True, ReplaceFormat:=True
```

图 23.1 显示了其后的显示结果。

图 23.1 显示了其后的显示结果。

PowerPoint 的对象

在 PowerPoint 中，可以使用 ActivePresentation 和 ActiveWindow 对象来访问当前的幻灯片。如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。

如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。

如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。

如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。如果希望在所有幻灯片中应用某种格式，则可以在 ActivePresentation 对象上应用该格式。

第 24 章 了解 PowerPoint 对象模型和重要对象

- ◆ 获得 PowerPoint 对象模型的概括性知识
- ◆ 了解 PowerPoint 的可创建对象
- ◆ 使用 Presentations 进行工作
- ◆ 使用 Windows 和 View 进行工作
- ◆ 使用 Slides 进行工作
- ◆ 使用 Master 进行工作

本章介绍如何使用 PowerPoint 对象模型进行工作。PowerPoint 对象模型是 PowerPoint 的基础性的理论体系结构。本章描述如何使用一些最有用的 PowerPoint 对象来实施常见的行动。这些对象包括 Presentations 集合和 Presentation 对象，ActivePresentation 对象，Slides 集合和 Slide 对象，Window 对象以及 Master 对象。

获得 PowerPoint 对象模型的概括性知识

要研究 PowerPoint 对象模型，可从启动“Microsoft Visual Basic 帮助”应用程序及显示“PowerPoint 对象模型”主题着手。如果“Visual Basic 编辑器”尚未启动，则启动它，然后选择“帮助”>“Microsoft Visual Basic 帮助”，键入“PowerPoint 对象”并按下 Enter 键，再单击“PowerPoint 对象模型”主题。

图 24.1 显示“PowerPoint 对象模型”主题。由于内容太长，屏幕不便于显示，所以左图显示该主题的顶部，右图显示出该主题的其余部分。

了解 PowerPoint 的可创建对象

在 PowerPoint 中，通过 Application 对象可以访问到 PowerPoint 应用程序中的所有对象。但是，对于很多操作，可以直接使用 PowerPoint 拥有的可创建对象。五种最有用的可创建对象如下：

- ◆ ActivePresentation 对象，它表示活动的演示文稿。
- ◆ Presentations 集合，它包含若干个 Presentation 对象，每个对象表示一个打开的演示文稿。
- ◆ ActiveWindow 对象，它表示应用程序中的活动窗口。
- ◆ CommandBars 集合，它包含若干个 CommandBar 对象，每个对象表示 PowerPoint 应用程序中的一个命令栏（工具栏、菜单栏及上下文菜单）。通过对各种 CommandBar 对象进行操作，能够以程序方式改变 PowerPoint 的界面。
- ◆ SlideShowWindows 集合，它包含若干个 SlideShowWindow 对象，每个对象表示一个

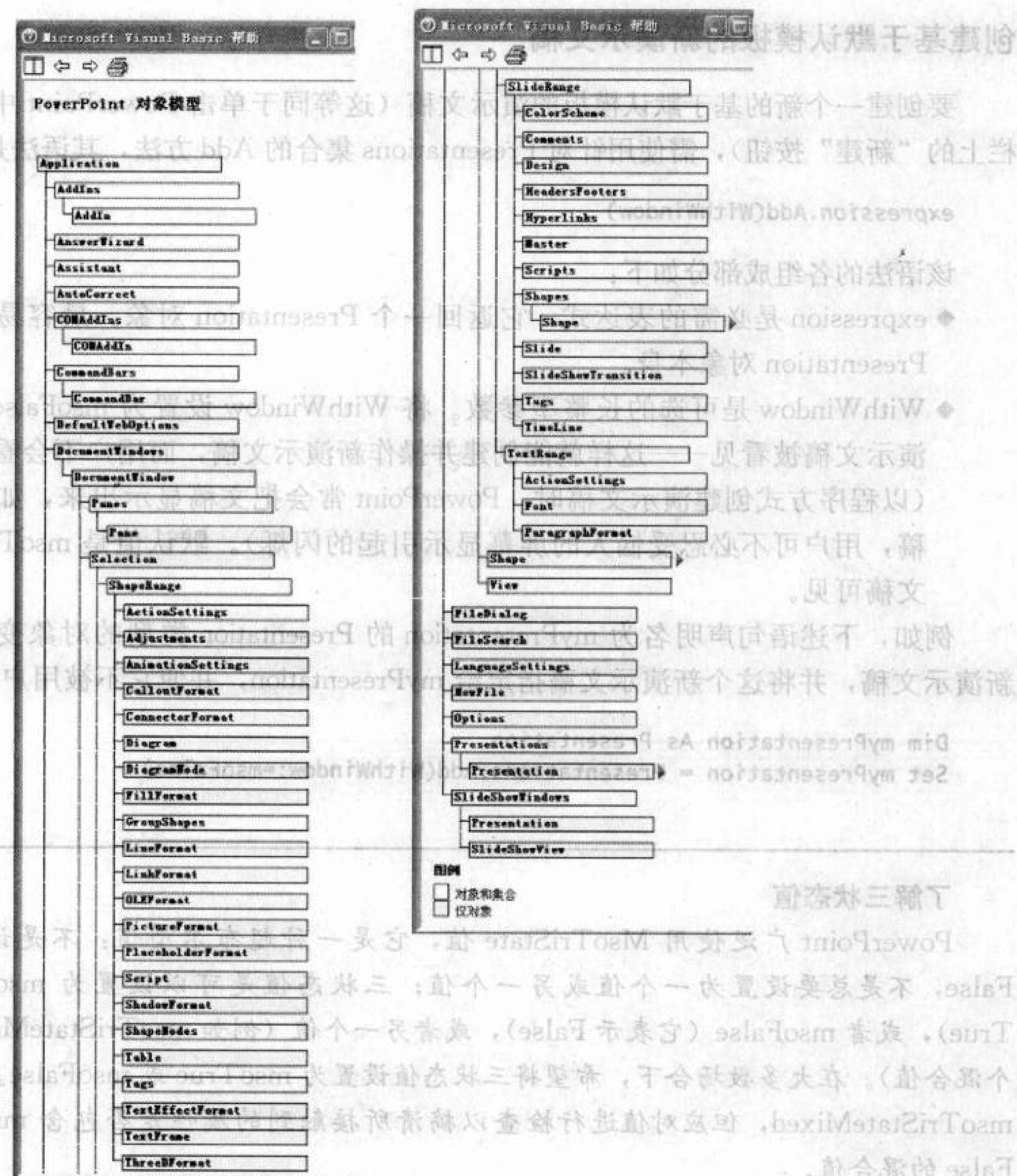


图 24.1 VBA 帮助中的“PowerPoint 对象模型”屏幕，便于用户研究 PowerPoint 对象模型

打开的幻灯片放映窗口。这个集合对于操作控制当前显示的幻灯片放映是很有用的。

在一个演示文稿中，往往要对 Slides 集合进行工作，这个集合包含表示各张幻灯片的 Slide 对象。在一张幻灯片中，大多数项目是由 Shape 对象来表示的，这些 Shape 对象汇集成为 Shapes 集合。例如，一个典型的占位符文本被包含在 TextFrame 对象中 TextRange 对象的 Text 属性之内，而这个 TextFrame 对象则在一张幻灯片上的某一个 Shape 对象之中。

使用 Presentations 进行工作

要完成在 PowerPoint 中的工作，常常需要针对一个或多个演示文稿进行工作。VBA 使用 Presentation 对象来表示一个演示文稿，并把若干个打开的 Presentation 对象组织成 Presentations 集合。

创建基于默认模板的新演示文稿

要创建一个新的基于默认模板的演示文稿（这等同于单击 PowerPoint 中“标准”工具栏上的“新建”按钮），需使用针对 Presentations 集合的 Add 方法，其语法是：

```
expression.Add(WithWindow)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Presentation 对象。最容易的往往是使用 Presentation 对象本身。
- ◆ WithWindow 是可选的长整型参数。将 WithWindow 设置为 msoFalse，可以免得新演示文稿被看见——这样就能创建并操作新演示文稿，而用户不会看到显示的详情（以程序方式创建演示文稿时，PowerPoint 常会把文稿显示出来，如果隐藏演示文稿，用户可不必忍受恼人的屏幕显示引起的闪烁）。默认值是 msoTrue，使新演示文稿可见。

例如，下述语句声明名为 myPresentation 的 Presentation 类型的对象变量，创建一个新演示文稿，并将这个新演示文稿指定给 myPresentation，并使它不被用户看见：

```
Dim myPresentation As Presentation  
Set myPresentation = Presentations.Add(WithWindow:=msoFalse)
```

了解三状态值

PowerPoint 广泛使用 MsoTriState 值，它是一种超布尔型值：不是设置为 True 或 False，不是总要设置为一个值或另一个值；三状态值是可以设置为 msoTrue（它表示 True），或者 msoFalse（它表示 False），或者另一个值（例如 msoTriStateMixed，它表示一个混合值）。在大多数场合下，希望将三状态值设置为 msoTrue 或 msoFalse。可以不设置为 msoTriStateMixed，但应对值进行检查以搞清所接触到的属性是否包含 msoTrue 和 msoFalse 的混合值。

创建基于模板的新演示文稿

要创建一个基于模板的新演示文稿，需使用 Presentations 集合的 Open 方法，其语法是：

```
expression.Open(fileName, ReadOnly, Untitled, WithWindow, OpenConflictDocument)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Presentation 对象。最容易的往往是使用 Presentation 对象本身。
- ◆ FileName 是必需的字符串参数，它指定用于与新演示文稿对应的模板文件的路径和名称。这个文件可以是常规意义下的一个模板，或者是打算用做模板的一个演示文稿。

- ◆ `ReadOnly` 是可选的参数，它指定文件是以只读方式打开 (msoTrue)，还是以读/写方式打开 (msoFalse)。基于一个模板来创建新演示文稿时，不必指定 `ReadOnly`。
- ◆ `Untitled` 是可选的参数，它指定是把文件打开为它本身 (msoFalse)，还是打开为一个复制副本 (msoTrue)。基于一个模板来创建新演示文稿时，将 `Untitled` 设置为 msoTrue。
- ◆ `WithWindow` 是可选的参数，将它设置为 msoFalse，可以避免新演示文稿被看见。默认值是 msoTrue，使新演示文稿可见。
- ◆ `OpenConflictDocument` 是可选的参数，它控制：PowerPoint 是打开一个与具有脱机冲突的演示文稿对应的任何现有的冲突文件 (msoTrue)，还是忽视该冲突文件 (msoFalse)。（冲突文件是当同一基文件的两个版本包含有未被同步的不同修改时所创建的文件。）基于模板来创建一个新演示文稿时，这个参数是不相干的，它只在打开一个现有文件时才有关系（参见下一小节）。

例如，下述语句基于名为 `Capsules.pot`、在 `C:\Program Files\Microsoft Office\Templates\Presentation Designs` 文件夹中的模板来创建一个新演示文稿：

```
Presentations.Open FileName:="C:\Program Files\Microsoft  
Office\Templates\Presentation Designs\Capsules.pot", Untitled:=msoTrue
```

打开演示文稿

要打开一个现有的演示文稿，需使用 `Presentations` 集合的 `Open` 方法，其语法如前一节所示。差别在于，这里是使用 `FileName` 参数来指定要打开的演示文稿（这与上一节的情况不同，那里是指定要用做与新演示文稿对应的模板文件），同时可以省略 `Untitled` 参数，或者将它设置为 `msoFalse`。也可能需要使用 `OpenConflictDocument` 参数来指定如何处理与正要打开的演示文稿相对应的冲突文件。

例如，下述语句打开文件夹 `Z:\Public` 之中的名为 `Train Time.ppt` 的现有演示文稿，打开该演示文稿是为了对其进行编辑，因此不是以只读方式来打开它：

```
Presentations.Open FileName:="Z:\Public\Train Time.ppt", ReadOnly:=msoFalse
```

保存演示文稿

如果是第一次保存某个演示文稿，就必须指定要使用的路径和文件名。此后，就可以在相同名称下保存该演示文稿，或指定另一路径、名称、格式，或指定这三者来保存该演示文稿。

第一次保存或在另一名称下保存演示文稿

第一次保存某个演示文稿，或使用另一不同路径、名称或格式来保存某个演示文稿，需使用 `SaveAs` 方法，其语法是：

```
expression.SaveAs(Filename, FileFormat, EmbedFonts)
```

该语法的各组成部分如下：

- ◆ `expression` 是必需的表达式，它返回一个 `Presentation` 对象。
- ◆ `FileName` 是必需的字符串参数，它指定文件名以便在该名称下保存演示文稿。正常

情况下，FileName 中包括有路径；如果省略路径，PowerPoint 就使用当前文件夹。

◆ FileFormat 是可选的参数，它指定要使用的文件格式。表 24.1 列出了应用最广泛的格式。

表 24.1 与保存 PowerPoint 文件对应的 FileFormat 常量

格式名称	常量
PowerPoint 格式	ppSaveAsPresentation
默认格式（在“选项”对话框的“保存”标签上设置）	ppSaveAsDefault
单个文件 Web 页	ppSaveAsWebArchive
Web 页	ppSaveAsHTML
PowerPoint 95	ppSaveAsPowerPoint7
用于审阅的演示文稿	ppSaveAsPresForReview
设计模板	ppSaveAsTemplate
PowerPoint 放映	ppSaveAsShow

◆ EmbedFonts 是可选的参数，将它设置为 True，是要将 TrueType 字体嵌入要保存的演示文稿；将它设置为 False（默认）则不嵌入。

例如，下述语句在文件夹 Z:\Shared\Presentations 之中名称 HR.ppt 之下，保存以对象变量 myPresentation 来标识的演示文稿，使用 PowerPoint 95 格式，不嵌入字体：

```
myPresentation.SaveAs FileName:="Z:\Shared\Presentations\HR.ppt", _  
FileFormat:=ppSaveAsPowerPoint7, EmbedTrueTypeFonts:=msoFalse
```

在其现有名称下保存演示文稿

要在其现有名称下保存某一个演示文稿，需使用 Save 方法。这个方法不要参数。例如，下述语句保存活动的演示文稿：

```
ActivePresentation.Save
```

如果对其使用 Save 方法的那个演示文稿从未被保存过，PowerPoint 不会提示用户去指定文件名和位置。PowerPoint 使用另外的方法，即在当前文件夹中以指派给该演示文稿的窗口的默认名称来保存演示文稿（例如，如果某个演示文稿的窗口称为 Presentation11，那么这个文稿就会被保存为 Presentation11.ppt）。在使用 Save 方法之前，如果需要确定演示文稿是否被保存过，可以检查 Presentation 对象的 Path 属性，这样也避免了使用默认名称和位置，例如：

```
If ActivePresentation.Path = "" Then  
    ActivePresentation.SaveAs FileName:="z:\public\presentations\Corporate.ppt"  
Else  
    ActivePresentation.Save  
End If
```

保存演示文稿的副本

如果不使用 SaveAs 方法，而在另一个不同名称之下保存一个打开的演示文稿，那么就可以使用 SaveCopyAs 方法来保存打开的演示文稿的副本，同时又不影响打开的演示文稿（该演示文稿仍为打开，而且任何未保存的更改仍未保存）。与 SaveCopyAs 方法对应的语法和参数与 SaveAs 方法的相同：

```
expression.SaveCopyAs(Filename, FileFormat, EmbedFonts)
```

例如，下述语句在文件夹 Z:\Public\Presentations 之中名称 Copy 1.ppt 之下，保存活动演示文稿的副本，并且它使用的文件格式与该演示文稿当前使用的相同：

```
ActivePresentation.SaveCopyAs FileName:="Z:\Public\Presentations\Copy 1.ppt"
```

保存所有打开的演示文稿

Presentations 集合没有 Save 方法，要保存所有打开的演示文稿，可以使用诸如下述子过程中的循环。这个子过程使那些文件名尚未被指定的演示文稿不被保存：

```
Sub Save_All_Presentations()
    Dim myPresentation As Presentation
    For Each myPresentation In Presentations
        If myPresentation.Path <> "" Then myPresentation.Save
    Next myPresentation
End Sub
```

关闭演示文稿

要关闭某一个演示文稿，需使用相应的 Presentation 对象的 Close 方法。Close 方法不用参数。例如，下述语句关闭活动的演示文稿：

```
ActivePresentation.Close
```

如果要关闭的演示文稿包含有未保存的更改，那么 PowerPoint 会提示用户保存它们。如果不提示用户，可在使用 Close 方法之前，将 Presentation 对象的 Saved 属性设置为 True。例如：

```
With Presentations("Karelia Industry.ppt")
    .Saved = True
    .Close
End With
```

将演示文稿或幻灯片导出到图形文件

使用 Presentation 对象、Slide 对象或 SlideRange 对象的 Export 方法，可以将整个演示文稿、单张幻灯片或一个幻灯片子集导出。Presentation 对象的 Export 方法的语法是：

```
expression.Export(Path, FilterName, ScaleWidth, ScaleHeight)
```

Slide 对象或 SlideRange 对象的 Export 方法的语法几乎与上述语法一样：

```
expression.Export(FileName, FilterName, ScaleWidth, ScaleHeight)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，根据情况，它返回一个 Presentation 对象，或一个 Slide 对象，或一个 SlideRange 对象。
- ◆ Path（与 Presentation 对象相对应）是必需的字符串参数，它指定文件夹的路径，各

幻灯片的图形文件要保存在该文件夹中。

- ◆ `FileName`（与 `Slide` 对象或 `SlideRange` 对象相对应）是必需的字符串参数，它指定要用于被导出图形的文件名。如果不希望 PowerPoint 使用当前文件夹，`FileName` 中就必须包含路径。
- ◆ `FilterName` 是必需的字符串参数，它指定要使用的图形筛选器。对于 `FilterName`，需使用已注册的文件扩展名（JPG、TIF、BMP 或 PNG）。
- ◆ `ScaleWidth` 是可选的长整型参数，可以用来指定被导出图形的宽度，以像素计。
- ◆ `ScaleHeight` 是可选的长整型参数，可以用来指定被导出图形的高度，以像素计。

例如，下述语句将活动演示文稿中所有的幻灯片导出到 Z:\Public\Presentations 文件夹内的 800×600 JPG 图形文件中。PowerPoint 将图形取名为 Slider1、Slide2，等等。

```
ActivePresentation.Export Path:="Z:\Public\Presentations",
    FilterName:="JPG", ScaleWidth:=800, ScaleHeight:=600
```

下述语句将活动演示文稿中的第六张幻灯片导出到 Z:\Public\Presentations 文件夹内名为 Slide6.png 的文件中，使用 PNG 格式：

```
ActivePresentation.Slides(6).Export _
    FileName:="Z:\Public\Presentations\Slide6.png", FilterName:="PNG"
```

打印演示文稿

要打印一个演示文稿，需使用适当的 `Presentation` 对象的 `PrintOut` 方法，其语法是：

```
expression.PrintOut(From, To, PrintToFile, Copies, Collate)
```

该语法的各组成部分如下：

- ◆ `expression` 是必需的表达式，它返回一个 `Presentation` 对象。
- ◆ `From` 和 `To` 是可选的整型参数，它们指定要打印的第一张幻灯片和最后一张幻灯片。如果省略 `From`，PowerPoint 从第一张幻灯片开始打印；如果省略 `To`，PowerPoint 打印持续到最后一张幻灯片。
- ◆ `PrintToFile` 是可选的字符串参数，使用它来让 PowerPoint 将指定的要打印的文件传送至文件而不是打印机。
- ◆ `Copies` 是可选的整型参数，它指定要打印的演示文稿或幻灯片的副本的份数。如果省略 `Copies`，那么使用默认值 1。
- ◆ `Collate` 是可选参数，将它设置为 `msoFalse`，可以避免 PowerPoint 用逐份打印的方式打印多份副本（这是默认设置）。

例如，下述语句打印活动演示文稿中所有的幻灯片：

```
ActivePresentation.PrintOut
```

下述语句打印以对象变量 `myPresentation` 来标识的演示文稿中的第五张至第十二张幻灯片：

```
myPresentation.PrintOut From:=5, To:=12
```

将模板应用于演示文稿、幻灯片或幻灯片子集

要将某一个设计模板应用于演示文稿、演示文稿中的单张幻灯片或幻灯片子集，需使用 Presentation 对象、Slide 对象或 SlideRange 对象的 ApplyTemplate 方法，其语法是：

```
expression.ApplyTemplate(FileName)
```

此处，expression 是必需的表达式，它返回一个 Presentation 对象、Slide 对象或 SlideRange 对象。FileName 是必需的字符串参数，它指定设计模板的路径和名称。

例如，下述语句将名为 Clouds.pot、保存在 C:\Program Files\Microsoft Office\Templates\Presentation Designs 文件夹中的设计模板，应用于活动演示文稿：

```
ActivePresentation.Slides(1).ApplyTemplate FileName:=  
"C:\Program Files\Microsoft Office\Templates\Presentation Designs\Clouds.pot"
```

下述语句将名为 Mountain Top.pot、保存在 Z:\Public\Template 文件夹中的设计模板应用于名为 Success.ppt 演示文稿中的第一张幻灯片：

```
Presentations("Success.ppt").Slides(1).ApplyTemplate FileName:=  
"Z:\Public\Template\Mountain Top.pot"
```

下述语句将名为 Disaster.pot、保存在 Z:\Public\Template 文件夹中的设计模板应用于一个幻灯片子集，该子集由活动演示文稿中的第一张、第四张和第六张幻灯片组成：

```
ActivePresentation.Slides.Range(Array(1, 4, 6)).ApplyTemplate _  
FileName:="Z:\Public\Template\Disaster.pot"
```

使用 ActivePresentation 进行工作

Application 对象的 ActivePresentation 属性返回一个表示活动演示文稿（活动窗口中的演示文稿）的 Presentation 对象。ActivePresentation 对象对于用户启动的某些过程是很有用的。

如果没有窗口是打开的，那么试图使用 ActivePresentation 对象便返回一个错误。所以，要是不能肯定有一个活动的演示文稿的话，在访问 ActivePresentation 对象之前检查是否有窗口已打开是一个好办法。例如：

```
If Windows.Count = 0 Then  
    MsgBox "Please open a presentation before running this macro."  
End If
```

使用 Windows 和 Views 进行工作

要使 PowerPoint 窗口进入所需要的状态，常常有必要对窗口和视图进行工作。PowerPoint 使用两种类型的窗口：文档窗口和幻灯片放映窗口。

- ◆ 文档窗口是容纳文档（演示文稿文件）而不是幻灯片放映的窗口。VBA 把文档窗口视为 DocumentWindow 对象，若干个 DocumentWindow 对象（用 Window 对象来表

示) 组织成 DocumentWindows 集合(用 Windows 集合来表示)。

◆ 幻灯片放映窗口是容纳打开的幻灯片放映的窗口。VBA 使用 SlideShowWindow 对象和 SlideShowWindows 集合来表示幻灯片放映窗口。

本节介绍如何对文档窗口进行工作。在第 25 章的“设置和运行幻灯片放映”一节中，我们将学习如何针对幻灯片放映窗口进行工作。

Windows 集合包含与 PowerPoint 应用程序中每一个打开的窗口对应的 Window 对象。当以交互方式工作时，如果打开一个演示文稿，PowerPoint 便打开一个窗口，使人能看到这个演示文稿。当通过 VBA 打开一个演示文稿时，可以把 Add 方法的 WithWindow 参数设置为 msoFalse，以避免 PowerPoint 显示与该演示文稿相对应的窗口。当然，必要时也可以打开更多的窗口——例如，通过在用户界面中选择“窗口”>“新建窗口”。

使用 ActiveWindow 进行工作

PowerPoint 使用 ActiveWindow 对象来表示活动窗口(即具有焦点的窗口)。某一个时间只能有一个窗口是活动窗口。活动窗口总是 Windows 集合中的第一个 Window 对象——Windows(1)。

如果没有窗口是打开的，或者所有打开的窗口都被隐藏，那么就没有活动窗口，此时使用 ActiveWindow 对象就会使 VBA 返回一个错误。要确认是否有窗口已打开，需检查 Windows 集合的 Count 属性是否为 0——例如：

```
If Windows.Count = 0 Then MsgBox "There is no active window.", vbOkOnly + _  
vbExclamation, "No Window Is Open"
```

当使用 VBA 对演示文稿进行工作时，有时可能会发现 ActiveWindow 对象是访问演示文稿的便利方式，尤其是用户选择了想对其施加影响的演示文稿、幻灯片或其他对象之后，再运行一个宏的时候。在其他场合，可能会发现，使用 ActivePresentation 对象来访问需对其进行工作的演示文稿是比较方便的，但也可能宁愿通过 Presentation 集合来访问演示文稿。

在演示文稿上打开新窗口

要打开一个新窗口，需使用适当的 Window 对象的 NewWindow 方法。这一方法不用参数。例如，下述语句打开一个显示活动窗口内容的新窗口：

```
ActiveWindow.NewWindow
```

关闭窗口

要关闭一个窗口，需使用适当的 Window 对象的 Close 方法。在 PowerPoint 中，Close 方法不用参数。

警告：如果正在关闭的窗口是为演示文稿打开的最后一个窗口，那么 PowerPoint 只是简单地将窗口关闭，而不提示用户保存任何未保存的更改。因此，关闭窗口应多加小心。

例如，可以关闭所有窗口，但保留演示文稿上的一个窗口不关闭：

```
Do While ActivePresentation.Windows.Count > 1
    ActivePresentation.Windows(ActivePresentation.Windows.Count).Close
```

Loop

另外，还可以在关闭演示文稿的最后一个窗口之前，使用 Save 方法来保存该演示文稿，如下例所示。（更简单的是，在保存演示文稿之后，使用 Close 方法来关闭该演示文稿本身。）

```
With ActivePresentation
    If .Path = "" Then
        MsgBox "Please save this presentation.", vbOKOnly
    Else
        .Save
    End If
    For Each myWindow In Windows
        .Close
    Next myWindow
End With
```

激活窗口

要激活一个窗口或者窗口的一个窗格，需使用适当的 Window 对象的 Activate 方法。例如，下述语句激活在演示文稿 Benefits.ppt 上打开的第一个窗口：

```
Presentations("Benefits.ppt").Windows(1).Activate
```

排列各窗口和调整窗口的大小

要排列若干窗口，需使用与适当的 Windows 集合相对应的 Arrange 方法，其语法是：

```
expression.Arrange(ArrangeStyle)
```

此处，expression 是必需的表达式，它返回一个 Windows 集合。ArrangeStyle 是必需的参数，它指定如何排列各窗口：ppArrangeCascade（以叠加方式排列各窗口，让人能看到每个窗口的标题栏，但只能看到最前面窗口的内容）或者 ppArrangeTitled（平铺各窗口，这是默认设置）。

将 Application 对象的WindowState 属性设置为 ppWindowMaximized、ppWindowMinimized 或 ppWindowNormal，可以最大化、最小化或恢复应用程序窗口。与此类似，在应用程序窗口之内，通过设置其WindowState 属性，可以最大化、最小化或恢复该文档。

当窗口处于“常规”状态时（ppWindowNormal，既非最大化又非最小化），可以使用 Top 和 Left 属性来指定窗口左上角的位置，以此来给该窗口定位；同时可以设置窗口的 Height 和 Width 属性来调整它的大小。

下述语句使应用程序窗口最大化，并在应用程序窗口内层叠各个文档窗口：

```
Application.WindowState = ppWindowMaximized
Windows.Arrange ArrangeStyle:=ppArrangeCascade
```

改变视图

要改变一个窗口中的视图，需将相应的 Window 对象的 ViewType 属性设置为适当的常量：ppViewHandoutMaster、ppViewMasterThumbnails、ppViewNormal、ppViewNotesMaster、ppViewNotesPage、ppViewOutline、ppViewPrintPreview、ppViewSlide、ppViewSlideMaster、ppViewSlideSorter、ppViewThumbnails 或者 ppViewTitleMaster。例如，下述语句使活动窗口转换为普通视图：

```
ActiveWindow.ViewType=ppViewSlideSorter
```

要放大视图，可以为对应于适当窗口的 View 对象的 Zoom 属性指定 10 至 400 之间的一个值。该值表示放大百分比，但不用加上百分比符号。例如，下述语句将活动窗口放大至 150%：

```
ActiveWindow.View.Zoom = 150
```

使用窗格进行工作

Pane 对象表示幻灯片视图中 PowerPoint 窗口的一个窗格。大纲窗格用索引号 1 来表示，幻灯片窗格用索引号 2 来表示，而备注窗格用索引号 3 来表示。使用与适当的 Pane 对象相对应的 Activate 方法，可以激活一个窗格。下述例子将活动窗口中的视图转换为幻灯片视图，并激活大纲窗格：

```
With ActiveWindow
    .ViewType = ppViewSlide
    .Panes(1).Activate
End With
```

要改变幻灯片视图中 PowerPoint 窗口的设置，需使用 Window 对象的 SplitHorizontal 属性和 SplitVertical 属性。SplitHorizontal 属性控制大纲窗格在文档窗口的宽度中所占的百分比，而 SplitVertical 属性控制幻灯片窗格在文档窗口的高度中所占的百分比。下述例子将大纲窗格设置为占文档窗口的宽度的 25%（75% 留给幻灯片窗格），幻灯片窗格则占窗口高度的 75%（25% 留给备注窗格）：

```
With ActiveWindow
    .SplitHorizontal = 25
    .SplitVertical = 75
End With
```

使用 Slides 进行工作

一旦创建或打开了想对其进行工作的演示文稿，使用 Slides 集合就可以访问它所包含的幻灯片。Slides 集合包含了该演示文稿中与每一张幻灯片对应的 Slide 对象。每张幻灯片都以它的索引号来标识，也可以使用对象变量来引用幻灯片或为幻灯片指派名称。在向演示文稿中添加幻灯片或从演示文稿中删除幻灯片时，这些技术是很有用的；当进行上述操作时，各幻灯片的索引号会发生变化。

添加幻灯片到演示文稿

要将一张幻灯片添加到某一个演示文稿，需使用与 Slides 集合对应的 Add 方法，其语法是：

```
expression.Add(Index, Layout)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Slides 集合。在很多情况下，最方便的是使用 Slides 集合本身。
- ◆ Index 是必需的长整型参数，它指定为幻灯片在演示文稿中定位所对应的索引号。例如，指定索引号为 2 的新幻灯片为演示文稿中的第二张幻灯片。
- ◆ Layout 是必需的长整型参数，它为新幻灯片指定版式。版式的名称，与“插入幻灯片”对话框或“幻灯片版式”任务窗格中看到的名称相吻合。例如，ppLayoutBlank 指定一张空白幻灯片，ppLayoutTitleOnly 指定一张只有标题的幻灯片，而 ppLayoutChartAndText 指定一张图表与文本的幻灯片。

例如，下述语句声明一个名为 mySlide 的对象变量，并将它指派给添加在活动演示文稿起始处的新标题幻灯片：

```
Dim mySlide As Slide  
Set mySlide = ActivePresentation.Slides.Add(Index:=1,  
Layout:=ppLayoutTitle)
```

了解“混合”常量

观看对应于 Layout 属性的常量列表时，会注意到一个名为 ppLayoutMixed 的常量。在幻灯片版式的列表中是没有“混合”版式的，如果试图将 ppLayoutMixed 应用于一张幻灯片，VBA 将返回一个错误。这是因为，ppLayoutMixed 是 VBA 对一个幻灯片子集的 Layout 属性返回的值，这个子集包含使用不同设计得到的多张幻灯片。

还有些属性以类似于 Mixed 的值来指明各对象使用不同的值，例如 ppTransitionSpeed-Mixed 意味着各幻灯片或形状使用不同的切换速度。别去尝试把某一个属性设置为 Mixed 值，因为这样做总会造成一个错误。

从现有演示文稿中插入幻灯片

在自动创建演示文稿时，从现有演示文稿中插入幻灯片是很有用的。为此，需使用 Slides 集合的 InsertFromFile 方法，其语法是：

```
expression.InsertFromFile(FileName, Index, SlideStart, SlideEnd)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Slides 集合。常常希望使用 Slides 集合本身。
- ◆ FileName 是必需的字符串参数，它指定包含有要插入的幻灯片的文件。

- ◆ Index 是必需的长整型参数，它指定打开的演示文稿中的幻灯片位置，就是要在这个位置插入幻灯片。
- ◆ SlideStart 是可选的长整型参数，它指定要插入的第一张幻灯片。如果省略 SlideStart 参数，PowerPoint 就从第一张幻灯片开始取用。
- ◆ SlideEnd 是可选的长整型参数，它指定要插入的最后一张幻灯片。如果省略 SlideEnd 参数，PowerPoint 就一直用到最后一张幻灯片。

例如，下述语句将取自保存在文件夹 Z:\Transfer\Presentations 之内、名为 Handbook.ppt 的演示文稿中的第二张至第八张幻灯片，插入打开的演示文稿 Corporate.ppt 中的第五张幻灯片处：

```
Presentations("Corporate.ppt").Slides.InsertFromFile _
    FileName:="Z:\Transfer\Presentations\Handbook.ppt", Index:=5, _
    SlideStart:=2, SlideEnd:=8
```

使用标识符查找幻灯片

当以程序方式针对某个演示文稿进行工作时，要跟踪各幻灯片，搞清哪张是哪张是很困难的，尤其是添加、删除、插入、复制或移动了幻灯片的时候。为了解决这个问题，PowerPoint 为已创建的每张幻灯片指派一个幻灯片标识符。如果把某一张幻灯片移动到演示文稿中另一个不同的位置处，该幻灯片的标识符不会改变，这和索引号不同，索引号总是反映出幻灯片在演示文稿中的位置。返回相应的 Slide 对象的 SlideID 属性，便可以检查出某一张幻灯片的标识符。

为了使用标识符来查找幻灯片，需使用 Slides 集合的 FindBySlideID 方法，其语法是：

```
expression.FindBySlideID(SlideID)
```

此处，expression 是必需的表达式，它返回一个 Slides 集合。SlideID 是必需的长整型参数，它指定希望返回的幻灯片标识符。

下述例子声明一个名为 TargetSlide 的长整型变量，将添加在活动演示文稿中索引号为 5 的位置处的一张新幻灯片指派给该变量，在索引号为 3 的位置处插入一个完整的演示文稿，然后使用 FindBySlideID 方法以返回 TargetSlide 所标识的幻灯片，并将另一个不同的设计模板应用于它：

```
Dim TargetSlide As Long
TargetSlide = ActivePresentation.Slides.Add(Index:=5, _
    Layout:=ppLayoutFourObjects).SlideID
Presentations("Corporate.ppt").Slides.InsertFromFile _
    FileName:="Z:\Transfer\Presentations\Handbook.ppt", Index:=3
ActivePresentation.Slides.FindBySlideID(TargetSlide).ApplyTemplate _
    FileName:="C:\Program Files\Microsoft Office\Templates\Presentation
    ↪Designs\Brain Blitz.pot"
```

改变现有幻灯片的版式

要改变某一张现有幻灯片的版式，需设置其 Layout 属性。例如，下述语句将活动演示文稿中第一张幻灯片的版式改变为剪贴画和竖排文字版式：

```
ActivePresentation.Slides(1).Layout = ppLayoutClipArtAndVerticalText
```

注意：当改变某一张幻灯片的版式时，PowerPoint 移动该幻灯片的现有内容，以允许所需的新对象添加到该幻灯片。

删除现有的幻灯片

要删除一张现有的幻灯片，需使用与适当的 Slide 对象相对应的 Delete 方法。例如，下述语句删除活动文档中的第一张幻灯片：

```
ActivePresentation.Slides(1).Delete
```

警告：PowerPoint 不对通过 VBA 删除幻灯片的操作进行确认。

复制和粘贴幻灯片

要复制一张幻灯片，需使用相应的 Slide 对象的 Copy 方法。Copy 方法不用参数。（也可以使用 Cut 方法来剪切幻灯片，此方法也不用参数。）

要粘贴一张幻灯片，需使用 Slides 集合的 Paste 方法。Paste 方法有一个 Index 参数，它指定在幻灯片的哪个位置粘贴幻灯片。

例如，下述语句复制活动演示文稿中的第一张幻灯片，并粘贴它，使它成为第五张幻灯片：

```
ActivePresentation.Slides(1).Copy  
ActivePresentation.Slides.Paste Index:=5
```

创建幻灯片副本

不进行复制和粘贴，使用 Slide 对象的 Duplicate 方法可以得到一张幻灯片副本。这一方法不用参数，它把幻灯片的副本紧挨着放在原本之后。例如，下述语句创建活动演示文稿中的第四张幻灯片的副本，将它放在索引号为 5 的位置处：

```
ActivePresentation.Slides(4).Duplicate
```

移动幻灯片

不使用剪切和粘贴，使用与适当的 Slide 对象相对应的 MoveTo 方法可以移动一张幻灯片。移动幻灯片与剪切并粘贴有相同的效果，但是它的好处是不改变剪贴板的内容（出于用户要求或其他目的，可能需要保护此内容）。MoveTo 方法的语法是：

```
expression.MoveTo(ToPos)
```

此处，expression 是必需的表达式，它返回一个 Slide 对象。ToPos 是必需的长整型参数，它指定索引号位置，以便把幻灯片移至此位置。

例如，下述语句将以对象变量 myPresentation 来标识的演示文稿中的第三张幻灯片，移至该演示文稿的起始处：

```
myPresentation.Slides(3).MoveTo Topos:=1
```

借助名称来访问幻灯片

与借助索引号来访问某一张幻灯片不同，通过使用 Slide 对象的 Name 属性，可以将一个名称指派给幻灯片。例如，下述语句将名称 Chairman's Introduction 指派给活动演示文稿中的第一张幻灯片，然后使用 Slide 对象的 Select 方法借助于名称来选择该幻灯片：

```
ActivePresentation.Slides(1).Name = "Chairman's Introduction"
ActivePresentation.Slides("Chairman's Introduction").Select
```

对幻灯片子集进行工作

要对一个幻灯片子集进行工作，需使用 Slides 集合的 Range 方法以返回表示这些幻灯片的一个 SlideRange 对象。可以用 SlideRange 对象来表示单张幻灯片，但是使用它来表示一个幻灯片子集往往更好。（通过索引号或指派给单张幻灯片的名称来访问单张幻灯片，比通过 SlideRange 对象来访问更容易。）

要返回包含两张或多张幻灯片的一个 SlideRange 对象，需使用后随以逗号来分隔的幻灯片列表的 Array 函数。该列表可以使用各幻灯片的索引号或名称。例如，下述语句声明 SlideRange 对象变量 mySlideRange，并将名为 HR.ppt 的打开的演示文稿中的前五张幻灯片指派给它：

```
Dim mySlideRange As SlideRange
Set mySlideRange = Presentations("HR.ppt").Slides.Range(Array(1, 2, 3, 4, 5))
```

下述语句将活动文档中名为 Intro 和 Outro 的两张幻灯片指派给 SlideRange 对象变量 mySlideRange：

```
Set mySlideRange = ActivePresentation.Slides.Range(Array("Intro", "Outro"))
```

为幻灯片设置格式

使用 ApplyTemplate 方法，可以把设计模板应用于幻灯片，这在本章前面“将模板应用于演示文稿、幻灯片或幻灯片子集”中已有讨论。本节要讨论的是，也可以把背景或配色方案应用于幻灯片。

将背景应用于一张或多张幻灯片

要将背景应用于一张或多张幻灯片，需使用适当的 Slide 对象或 SlideRange 对象以返回表示幻灯片背景的 ShapeRange 对象。然后可以使用 Fill 对象来设置背景中的颜色、填充、过渡或图片。

下述例子将出自文件夹 C:\Sample Pictures 的图片 Winter.jpg 应用于名为 Corporate.ppt 的演示文稿的第四张幻灯片。本例将 FollowMasterBackground 属性设置为 msoFalse，这就使得该幻灯片使用不同于幻灯片母版的背景，还将 DisplayMasterShapes 属性设置为 msoFalse，使得该幻灯片不显示幻灯片母版上的各形状：

```
With Presentations("Corporate.ppt").Slides(4)
    .FollowMasterBackground = msoFalse
    .DisplayMasterShapes = msoFalse
    With .Background
        .Fill.ForeColor.RGB = RGB(255, 255, 255)
        .Fill.BackColor.SchemeColor = ppAccent1
        .Fill.UserPicture "C:\Sample Pictures\Winter.jpg"
    End With
End With
```

将配色方案应用于幻灯片

配色方案是八种颜色的组合，用以形成标题、背景、和幻灯片、讲义或备注页中其他元素的外观。每种颜色由 VBA 使用的一个 RGBColor 对象来表示，每个配色方案由 VBA 使用的一个 ColorScheme 对象来表示。各个 ColorScheme 对象汇集成与整个演示文稿对应的 ColorSchemes 集合。

要改变一张或几张幻灯片的配色方案，需使用相应的 Slide 对象或 SlideRange 对象的 ColorScheme 属性，以返回 ColorScheme 对象，然后使用 Colors 方法来指定颜色。其语法是：

```
expression.Colors(SchemeColor)
```

此处，expression 是必需的表达式，它返回一个 ColorScheme 对象。SchemeColor 是必需的参数，它指定配色方案中要设置的颜色——例如，ppAccent1（对应于配色方案中的第一种强调色），ppBackground（对应于背景色）或 ppTitle（对应于标题色）。

下述语句将与活动演示文稿中前三张幻灯片对应的配色方案的背景色设置为黑色（RGB (0, 0, 0)）：

```
ActivePresentation.Slides.Range(Array(1, 2, 3))_
    .ColorScheme.Colors(ppBackground).RGB = RGB(0, 0, 0)
```

为幻灯片、幻灯片子集或母版设置切换方式

为了设置幻灯片、幻灯片子集或母版所对应的切换方式，需使用 Slide 对象、SlideRange 对象或 Master 对象的 SlideShowTransition 属性以返回 SlideShowTransition 对象。

要指定所用的效果，需将 EntryEffect 属性设置成与效果对应的常量。这些常量太多了，在此不能一一列出，但从它们的名称中可以推测出含义。例如，ppEffectBlindsHorizontal 常量表示横向淡出切换，ppEffectDissolve 常量表示消溶效果，而 ppEffectNone 常量表示没有切换效果。

要指定切换以什么速度进行，需将 Speed 属性设置为 ppTransitionSpeedFast、ppTransitionSpeedMedium 或者 ppTransitionSpeedSlow。

要控制如何换片，可将 AdvanceOnTime 属性设置为 msoTrue（对应于自动换片）或 msoFalse（对应于手动换片）。如果使用自动换片，要用 AdvanceTime 属性来指定秒数。如果希望在用户单击时换片，要将 AdvanceOnClick 属性设置为 msoTrue。（可以把 Advan-