

ceOnTime 和 AdvanceOnClick 均设置为 msoTrue，这样如果用户在 AdvanceTime 时间间隔到达之前单击，可以实现手动换片。)

要在切换时放送出预置的声音效果，需使用 SlideShowTransition 对象的 SoundEffect 属性以返回 SoundEffect 对象，使用 Name 属性来指定声音效果的名称，然后使用 Play 方法来发出声音效果。也可以使用 SoundEffect 对象的 ImportFromFile 方法来运行一个单独的声音文件，以及使用 FileName 参数来指定声音文件的路径和文件名。

如果希望某声音循环放送，持续到下一声音发出之时，需将 SlideShowTransition 对象的 LoopSoundUntilNext 属性设置为 msoTrue。默认设置为 msoFalse。

下述语句设置如何切换到活动演示文稿中的第二张幻灯片。切换时使用中速运行的消溶效果，既可单击时换片，又可在延迟 30 秒钟之后换片，并放送不带循环的源于外部的声音文件：

```
With ActivePresentation.Slides(2)
    With .SlideShowTransition
        .EntryEffect = ppEffectDissolve
        .Speed = ppTransitionSpeedMedium
        .AdvanceOnClick = msoTrue
        .AdvanceOnTime = msoTrue
        .AdvanceTime = 30
        .SoundEffect.ImportFromFile _
            FileName:="d:\Sounds\Crescendo.wav"
        .LoopSoundUntilNext = msoFalse
    End With
End With
```

使用 Masters 进行工作

VBA 使用 Master 对象来表示 PowerPoint 所用的多种母版：幻灯片母版，标题母版，讲义母版以及备注母版。

使用幻灯片母版进行工作

要使用与某一个演示文稿对应的幻灯片母版进行工作，需使用 Presentation 对象的 SlideMaster 属性。

要返回与某一张幻灯片对应的幻灯片母版，需使用适当的 Slider 对象的 Master 属性。例如，下述语句将标题添加到与活动演示文稿相对应的幻灯片母版。（如果该幻灯片母版已有标题，VBA 将返回一个错误。）

```
ActivePresentation.SlideMaster.Shapes.AddTitle.TextFrame.TextRange.Text = _
    "Orientation"
```

使用标题母版进行工作

要找出某一个演示文稿是否有标题母版，需检查 HasTitleMaster 属性。如果没有标题母版，可以使用 Presentation 对象的 AddTitleMaster 方法来添加一个标题母版，如下例所示。如果该演示文稿已经有标题母版，VBA 就会在有人试图添加标题母版时返回一个错误。

```
If Not ActivePresentation.HasTitleMaster Then ActivePresentation.AddTitleMaster
```

要返回与演示文稿对应的标题母版，需使用 Presentation 对象的 TitleMaster 属性。下面的例子检查标题母版是否存在，如果有的话，设置日期和时间为可见，并使用带有自动更新功能的 dMMMyy 格式：

```
With myPresentation
If .HasTitleMaster Then
    With .TitleMaster.HeadersFooters.DateAndTime
        .Visible = msoTrue
        .Format = ppDateTimedMMMyy
        .UseFormat = msoTrue
    End With
End If
End With
```

使用讲义母版进行工作

要使用讲义母版进行工作，需使用 Presentation 对象的 HandoutMaster 属性以返回 Master 对象。下述例子使用 ActivePresentation 对象的 HandoutMaster 属性，以便用一个图片来填充讲义母版的背景：

```
With ActivePresentation.HandoutMaster.Background
    .Fill.ForeColor.RGB = RGB(255, 255, 255)
    .Fill.BackColor.SchemeColor = ppAccent1
    .Fill.UserPicture "d:\igrafx\dawn.jpg"
End With
```

使用备注母版进行工作

要使用备注母版进行工作，需使用 Presentation 对象的 NotesMaster 属性以返回 Master 对象。下述语句清除第一个打开的演示文稿中的备注母版内的 HeaderFooter 对象。

```
Presentations(1).NotesMaster.HeadersFooters.Clear
```

删除母版

可以删除标题母版或讲义母版，但不能删除幻灯片母版或备注母版。要删除标题母版或讲义母版，需使用 Master 对象的 Delete 方法。下述语句检查活动演示文稿是否有标题母版，如有则删除它：

```
If ActivePresentation.HasTitleMaster Then ActivePresentation.TitleMaster.Delete
```

- ◆ 使用 Shapes 进行工作
- ◆ 使用 HeadersFooters 进行工作
- ◆ 设置和运行幻灯片放映

前一章学习了如何针对 Presentation 对象、Slide 对象和 Master 对象进行工作。本章学习使用 Shapes 对象进行工作，以便对幻灯片的内容进行操作，还要学习使用 HeaderFooter 对象进行工作，以便控制页眉和页脚的内容。也要学习如何使用 VBA 来设置和运行幻灯片放映。

使用 Shapes 进行工作

典型的 PowerPoint 幻灯片上的大多数对象都是 Shape 对象。例如，标题框是 Shape 对象，粘贴进来的图片和 Word 表格也是 Shape 对象。通过 Slide 对象、SlideRange 对象或 Master 对象的 Shapes 集合，可以访问各个 Shape 对象。

将 Shapes 添加到幻灯片

Shapes 集合使用多种不同的方法来添加不同类型的形状。表 25.1 列出了可供添加的 Shape 对象，以及添加它们时所用的方法和参数。本节解释这些参数。

表 25.1 各种形状和添加它们到幻灯片的方法

添加的形状	使用的方法和参数
标注	AddCallout (Type, Left, Top, Width, Height)
批注	AddComment (Left, Top, Width, Height)
连接符	AddConnector (Type, BeginX, BeginY, EndX, EndY)
曲线	AddCurve (SafeArrayOfPoints)
图示	AddDiagram (Type, Left, Top, Width, Height)
标签	AddLabel (Orientation, Left, Top, Width, Height)
线条	AddLine (BeginX, BeginY, EndX, EndY)
多媒体对象	AddMediaObject (FileName, Left, Top, Width, Height)
OLE 对象	AddOLEObject (Left, Top, Width, Height, ClassName, FileName, DisplayAsIcon, IconFileName, IconIndex, IconLabel, Link)
图片	AddPicture (FileName, LinkToFile, SaveWithDocument, Left, Top, Width, Height)
占位符	AddPlaceholder (Type, Left, Top, Width, Height)
连续线段或多边形	AddPolyline (SafeArrayOfPoints)
自选图形	AddShape (Type, Left, Top, Width, Height)
表格	AddTable (NumRows, NumColumns, Left, Top, Width, Height)
文本框	AddTextbox (Orientation, Left, Top, Width, Height)
标题	AddTitle
艺术字对象	AddTextEffect (PresetTextEffect, Text, FontName, FontSize, FontBold, FontItalic, Left, Top)

与添加形状相对应的共用参数

下面的参数是各种添加形状方法共用的：

- ◆ BeginX 和 EndX 是必需的单精度浮点型参数，它们指定连接符或线条在水平方向上相对于幻灯片左边缘的起点位置和终点位置，以点计。
- ◆ BeginY 和 EndY 是必需的单精度浮点型参数，它们指定连接符或线条在垂直方向上相对于幻灯片上边缘的起点位置和终点位置，以点计。
- ◆ FileName 是必需的字符串参数，用来指定创建对象所依据的文件（例如，创建多媒体对象所依据的多媒体文件）。
- ◆ Left 是必需的单精度浮点型参数，它指定形状的左边缘相对于幻灯片左边缘的位置，以点计。Top 是必需的单精度浮点型参数，它指定形状的上边缘相对于幻灯片上边缘的位置，以点计。
- ◆ Height 是必需的单精度浮点型参数，它指定形状的高度，以点计。Width 是必需的单精度浮点型参数，它指定形状的宽度，以点计。
- ◆ LinkToFile 是可选参数，将此参数设置为 msoTrue，是要把图片链接到它的源文件。
- ◆ NumColumns 和 NumRows 是必需的长整型参数，它们指定要添加的表格的列数和行数。
- ◆ Orientation 是必需的参数，它指定文本方向：msoTextOrientationHorizontal（水平）或 msoTextOrientationVerticalFarEast（垂直）。
- ◆ SafeArrayOfPoints 是必需的 Variant 参数，它提供一个坐标对数组，该数组给出曲线或多边形的各个顶点和控制点。第一个坐标对处是线条起点，最后一个坐标对处是线条终点。
- ◆ SaveWithDocument 是必需的参数，它控制要（msoTrue）还是不要（msoFalse）让 PowerPoint 在演示文稿中保存已链接的图片。如果设置了 LinkToFile: = msoFalse，那么就必须设置 SaveWithDocument: = msoTrue。

与添加形状相对应的 Type 参数

Type 参数因所使用的方法不同而各有不同：

- ◆ 对于 AddPlaceholder 方法的 Type 参数是必需的参数，它指定要添加的占位符的类型。名称是自解释型的：ppPlaceholderBitmap、ppPlaceholderBody、ppPlaceholderCenterTitle、ppPlaceholderChart、ppPlaceholderDate、ppPlaceholderFooter、ppPlaceholderHeader、ppPlaceholderMediaClip、ppPlaceholderObject、ppPlaceholderOrgChart、ppPlaceholderSlideNumber、ppPlaceholderSubtitle、ppPlaceholderTable、ppPlaceholderTitle、ppPlaceholderVerticalBody 或者 ppPlaceholderVerticalTitle。

警告：ppPlaceholderVerticalBody 和 ppPlaceholderVerticalTitle 占位符只能用于使用垂直文本的幻灯片，即版式为 ppLayoutVerticalText、ppLayoutClipArtAndVerticalText、ppLayoutVerticalTitleAndText 和 ppLayoutVerticalTitleAndTextOverChart 的幻灯片。

- ◆ 对于 AddCallout 方法的 Type 参数是必需的参数，它指定要添加的标注线的类型：

msoCalloutOne（单段标注线，可为水平线或垂直线）、msoCalloutTwo（可自由旋转的单段标注线）、msoCalloutThree（两段标注线）或者 msoCalloutFour（三段标注线）。

◆ 对于 AddShape 方法的 Type 参数是必需的参数，它指定要添加的自选图形的类型。

可用常量太多，此处无法一一列出，但是大多数都可以从其名称中轻易认出。例如，msoShapeHeart 为心脏形状，msoShapeLightningBolt 为闪电形状。要看到这些常量的列表，需在帮助文件中搜索 AddShape 方法，然后单击 Type 条目中的链接。

◆ 对于 AddDiagram 方法的 Type 参数是必需的参数，它指定图示的类型：msoDiagramCycle（循环图）、msoDiagramOrgChart（层次图）、msoDiagramPyramid（棱锥图）、msoDiagramRadial（辐射图）、msoDiagramTarget（靶形图）或者 msoDiagramVenn（维恩图）。

AddTextEffect 方法特有的参数

下述参数仅应用于 AddTextEffect 方法：

◆ PresetTextEffect 是必需的参数，它指定要使用的预设的文本效果。这些预设的文本效果是从 msoTextEffect1 到 msoTextEffect30 这些参数来标识的。它们与“艺术字库”对话框中出现的样本的顺序相对应（1 到 6 对应于第一行，7 到 12 对应于第二行，等等）。

◆ Text 是必需的字符串参数，它指定艺术字对象中要使用的文本。

◆ FontBold 是必需的参数，设置为 msoTrue，则使字体为粗体，设置为 msoFalse 则不为粗体。

◆ FontItalic 是必需的参数，设置为 msoTrue，则使字体为斜体，设置为 msoFalse 则不为斜体。

◆ FontName 是必需的字符串参数，它指定要使用的字体的名称。

◆ FontSize 是必需的字符串参数，它指定要使用的字体的大小。

AddOLEObject 方法特有的参数

下述参数仅应用于 AddOLEObject 方法：

◆ ClassName 是可选的字符串参数，它指定对应于所创建对象的 ProgID 或 OLE 长类名。必须使用 ClassName 或 FileName，但不能二者都用。在大多数场合下，使用 FileName 更方便。

◆ DisplayAsIcon 是可选的参数，将它设置为 msoTrue，是要将 OLE 对象显示为一个图标，而不是显示为对象本身（默认值）。

◆ IconFileName 是可选的字符串参数，把它和 DisplayAsIcon := True 一起使用，以指定包含要针对 OLE 对象显示的图标文件的文件名。

◆ IconIndex 是可选的整型参数，它指明图标文件（由 IconFileName 指定）之中要使用的图标的索引。如果省略 IconIndex 参数，VBA 使用图标文件中的第二个图标，即位置 1 处的图标（因为文件中的第一个图标在位置 0 处）。

◆ IconLabel 是可选的字符串参数，用来指定要在图标之下显示的标题（或标签）。

◆ Link 是可选的参数，当使用 FileName 参数时，将 Link 设置为 msoTrue，是要使 OLE 对

象链接到它的源文件。当使用 ClassName 以指定类名时，Link 必须是 msoFalse。

使用 AddShape 方法的一个例子

下述语句使用 AddShape 方法，将一个弯曲向上的箭头添加到活动演示文稿中倒数第二张幻灯片的右上角：

```
ActivePresentation.Slides(ActivePresentation.Slides.Count - 1)_
    .Shapes.AddShape Type:=msoShapeBentUpArrow, Left:=575, Top:=10, _
    Width:=150, Height:=75
```

使用 AddTextEffect 方法的一个例子

下述例子先将一张空白幻灯片插入到演示文稿中最后一张幻灯片之后，然后使用 AddTextEffect 方法在该幻灯片上添加文本为 Questions and Answers 的艺术字（分三行）。艺术字选用 54 点粗体 ITC Avant Garde Gothic 字体。

```
Dim QASlide As Slide
With myPresentation
    Set QASlide = .Slides.Add(Index:=.Slides.Count + 1, Layout:=ppLayoutBlank)
    QASlide.Shapes.AddTextEffect PresetTextEffect:=msoTextEffect28, _
        Text:="Questions" + Chr$(CharCode:=13) + _
        "and" + Chr$(CharCode:=13) + "Answers", _
        FontName:="ITC Avant Garde Gothic", FontSize:=54, FontBold:=msoTrue, _
        FontItalic:=msoFalse, Left:=230, Top:=125
End With
```

使用 AddTextbox 方法的一个例子

下述例子将一个文本框添加到以对象变量 myPresentation 标识的演示文稿中的第八张幻灯片中，并为文本框指定文本：

```
Dim myTextBox As Shape
With myPresentation.Slides(8)
    Set myTextBox = .Shapes.AddTextbox _
        (Orientation:=msoTextOrientationHorizontal, Left:=100, Top:=50, _
        Width:=400, Height:=100)
    myTextBox.TextFrame.TextRange.Text = "Corrective Lenses"
End With
```

删除形状

要删除某一个形状，需使用对应于适当的 Shape 对象的 Delete 方法。例如，下述语句删除活动演示文稿中第二张幻灯片上的第一个 Shape 对象：

```
ActivePresentation.Slides(2).Shapes(1).Delete
```

选择所有形状

要选择某一张幻灯片上的所有形状，需使用相应的 Shapes 集合的 SelectAll 方法。例如，下述语句选择以对象变量 myPresentation 来标识的演示文稿中的最后一张幻灯片上的

所有 Shape 对象：`myPresentation.Slides(myPresentation.Slides.Count).Shapes.SelectAll`

重定位形状和调整形状的大小

要重定位某一个形状，需设置它的 `Left` 属性（以点数指定形状的左边缘离幻灯片左边距的距离）和 `Top` 属性（以点数指定形状的上边缘离幻灯片上边缘的距离）。

要改变某一个形状的大小，需将它的 `Width` 和 `Height` 属性设置为适当的点数。

例如，下述语句将活动演示文稿中第一张幻灯片上的第一个形状定位为：离幻灯片左边距 200 点，离幻灯片上边缘 100 点；形状宽度为 300 点，高为 200 点：

```
With ActivePresentation.Slides(1).Shapes(1)
    .Left = 200
    .Top = 100
    .Width = 300
    .Height = 200
End With
```

也可以使用 `IncrementLeft` 方法和 `IncrementTop` 方法来移动某一个形状，以及使用 `IncrementRotation` 方法来旋转某一个形状。上述各方法都要使用 `Increment` 参数：

- ◆ 对于 `IncrementLeft` 和 `IncrementTop` 方法，`Increment` 参数指定形状要移动的点数。点数的数值为负时，形状向左或向上移动；而点数的数值为正时，形状向右或向下移动。
- ◆ 对于 `IncrementRotation` 方法，`Increment` 参数指定形状要旋转的角度。此数值为正时，形状顺时针旋转；此数值为负时，形状逆时针旋转。

下述例子对以对象变量 `myPresentation` 来标识的演示文稿的第三张幻灯片上的第一个形状进行工作，将它左移 100 点，下移 200 点，并逆时针旋转 90°：

```
With myPresentation.Slides(3).Shapes(1)
    .IncrementLeft Increment:=-100
    .IncrementTop Increment:=-200
    .IncrementRotation Increment:=-90
End With
```

复制形状的格式设置

能将相同的格式设置应用于多个形状常常是有好处的。如果某一个形状已经有了所需要的格式设置，那么可以使用 `Shape` 对象的 `PickUp` 方法来复制该形状的格式设置，然后使用 `Apply` 方法将这一格式设置应用于其他的形状。

`PickUp` 方法和 `Apply` 方法均不使用任何参数。下述例子复制活动演示文稿中第二张幻灯片中的第一个形状的格式设置，将它应用于第四张幻灯片上的第三个形状：

```
With ActivePresentation
    .Slides(2).Shapes(1).PickUp
    .Slides(4).Shapes(3).Apply
End With
```

使用形状中的文本进行工作

形状中的文本包含在 TextRange 之中，而 TextRange 对象本身又包含在 TextFrame 对象之中。要使用形状中的文本进行工作，需使用 Shape 对象的 TextFrame 属性以返回 TextFrame 对象，然后使用 TextFrame 对象的 TextRange 属性以返回 TextRange 对象。在 TextRange 对象之内，Text 属性包含文本，Font 对象包含字体格式设置，ParagraphFormat 对象包含段落格式设置，而 ActionSettings 集合包含与文本范围对应的动作设置。

查找形状是否有文本框

并非每一个形状都有文本框，所以在打算对某一个形状中的文本进行操作时，检查该形状是否有文本框是一个好主意。为此，需检查 Shape 对象的 HasTextFrame 属性是否为 msoTrue。例如：

```
If ActivePresentation.Slides(1).Shapes(1).HasTextFrame = msoTrue Then  
    MsgBox "The shape contains a text frame."  
End If
```

还需要检查文本框内是否有文本。为此，需检查 TextFrame 对象的 HasText 属性是否为 msoTrue。例如：

```
With ActivePresentation.Slides(1).Shapes(1).TextFrame  
    If .HasText = msoTrue Then MsgBox .TextRange.Text  
End With
```

返回和设置文本范围中的文本

要返回或设置某一个文本范围中的文本，可以简单地使用 TextRange 对象的 Text 属性。例如，下述语句将以对象变量 myPresentation 标识的演示文稿中第一张幻灯片的第一个形状中的文本设置成为 Strategic Planning Meeting：

```
myPresentation.Slides(1).Shapes(1).TextFrame.TextRange.Text _  
= "Strategic Planning Meeting"
```

也可以使用 Paragraphs 方法、Sentences 方法、Lines 方法、Words 方法、Characters 方法或者 Runs 方法来返回文本的某些部分。这些方法的语法如下所示，这里使用 Paragraphs 方法作为例子：

```
expression.Paragraphs(Start, Length)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 TextRange 对象。
- ◆ Start 是可选的长整型参数，它指定要返回的第一个项目（段落、句子、行、单词、字符或文本域）。
- ◆ Length 是可选的长整型参数，它指定有多少个项目要返回——例如，两个段落、三个句子或者四个单词。

注意：一个文本域由具有相同字体格式设置的一批字符组成。要把以某种特定方式

设置好格式的文本范围的某些部分鉴别出来，文本域是很有用的。

例如，下述语句从活动演示文稿中第一张幻灯片上的第一个形状之中返回第二个至第五个单词（从第二个单词开始，共返回四个单词）：

```
MsgBox ActivePresentation.Slides(1).Shapes(1).TextFrame.TextRange.Words(Start:=2, Length:=4)
```

下述语句将以对象变量 myPresentation 来标识的演示文稿中第六张幻灯片上的第二个形状内的第二段文本设置为 VP of Business Development：

```
myPresentation.Slides(6).Shapes(2).TextFrame.TextRange.Paragraphs(Start:=2, Length:=1).Text = "VP of Business Development"
```

为文本范围中的文本设置格式

要为某一文本范围中的文本设置格式，需使用 ParagraphFormat 对象来控制段落的格式设置（包括对齐方式和前后间距），以及使用 Font 对象来控制字体的格式设置。

ParagraphFormat 对象的最常用的属性如下：

- ◆ Alignment 属性，它控制对齐方式。ppAlignLeft 对应于左对齐，ppAlignCenter 对应于居中，ppAlignJustify 对应于两端对齐，ppAlignDistribute 对应于分散对齐，ppAlignRight 对应于右对齐。
- ◆ Bullet 属性，它返回 BulletFormat 对象，表示项目符号的格式设置，详情见下一节。
- ◆ LineRuleBefore 属性、LineRuleAfter 属性和 LineRuleWithin 属性确定，由 SpaceBefore 属性、SpaceAfter 属性和 SpaceWithin 属性所设置的量度使用的是行数 (msoTrue)，还是点数 (msoFalse)。
- ◆ SpaceBefore 属性控制每个段落首行前的间距大小。SpaceAfter 属性控制每段末行后的间距大小。SpaceWithin 属性控制段落中各基准行之间的间距大小。所有量度都以点数计算。

下述语句对以对象变量 mySlide 来标识的幻灯片上的第二个形状设置左对齐方式，设置与段前和段后的段间距均为 18 点，行间距为 12 点：

```
With mySlide.Shapes(2).TextFrame.TextRange.ParagraphFormat
    .Alignment = ppAlignLeft
    .LineRuleAfter = msoFalse
    .SpaceAfter = 18
    .LineRuleBefore = msoFalse
    .SpaceBefore = 18
    .LineRuleWithin = msoFalse
    .SpaceWithin = 12
End With
```

为文本范围中的项目符号设置格式

项目符号和编号对很多 PowerPoint 幻灯片是至关重要的。要控制项目符号和编号如何出现，需使用 TextRange 对象的 Bullet 属性以返回 BulletFormat 对象，然后使用 BulletFormat 对象的属性和方法进行工作。

要使项目符号和编号可见，需将 BulletFormat 对象的 Visible 属性设置为 msoTrue；要隐藏项目符号和编号，需将 Visible 属性设置为 msoFalse。

要指定使用哪种类型的项目符号和编号，需将 BulletFormat 对象的 Type 属性设置为 ppBulletUnnumbered（对应于项目符号）、ppBulletNumbered（对应于编号）、ppBulletPicture（对应于图片）或者 ppBulletNone（不使用项目符号）。

注意：当所选内容包括多种类型的项目符号时，BulletFormat 对象的 Type 属性返回的值是 ppBulletMixed。但是不能将 Type 设置为 ppBulletMixed。

要指定项目符号字符，需使用 Character 属性和字符号（字符号可从“符号”对话框或字符映射表中查到。选择“开始”>“所有程序”>“附件”>“系统工具”>“字符映射表”，即可运行字符映射表）。使用 Font 属性指定字体的名称、大小和颜色。下述例子将以对象变量 mySlide 标识的幻灯片上的第一个形状所用的符号设置为 Wingdings 字符 254，即一个复选框，其颜色为白色 (RGB(255, 255, 255))，大小为 44 点：

```
With mySlide.Shapes(1).TextFrame.TextRange.ParagraphFormat.Bullet
    .Type = ppBulletUnnumbered
    .Character = 254
    With .Font
        .Name = "Wingdings"
        .Size = 44
        .Color = RGB(255, 255, 255)
    End With
End With
```

要使用图片作为项目符号，需将 BulletFormat 对象的 Type 属性设置为 ppBulletPicture，然后使用带有 Picture 参数的 Picture 方法。Picture 参数是必需的字符串参数，它指定要作为项目符号的文件的路径和文件名。大部分类型的图形文件都可供使用，包括 BMP、EPS、GIF、JPG、PCX、PNG、TIFF 以及 WMF 文件。下述例子把保存在文件夹 z:\Public\ Pictures 之中的 Face1.jpg 作为以对象变量 mySlide 标识的幻灯片上的第一个形状所使用的项目符号：

```
With mySlide.Shapes(1).TextFrame.TextRange.ParagraphFormat.Bullet
    .Type = ppBulletPicture
    .Picture Picture:="z:\Public\Pictures\Face1.jpg"
End With
```

为形状或形状范围设置动画

要为形状或形状范围设置动画效果，需使用 Shape 对象或 ShapeRange 对象的 AnimationSettings 属性，以便返回 AnimationSettings 对象。

要指定打算使用的动画效果，需将 EntryEffect 属性设置为与效果对应的常量。这些常量很多，不能在此一一列出，但是，从 PowerPoint 的自定义动画窗格中的名称里，很容易搞清这些常量的名称的含义。例如，ppEffectFlyFromRight 常量表示从右侧飞入的动画效果。

要产生动画，需将 Animate 属性设置为 msoTrue（要关闭动画，则将 Animate 属性设置为 msoFalse）。

要控制某一个形状中的文本具有怎样的动画效果，需将 TextLevelEffect 属性设置为

`ppAnimateLevelNone`（无动画）、`ppAnimateByFirstLevel`、`ppAnimateBySecondLevel`、`ppAnimateByThirdLevel`、`ppAnimateByFourthLevel`、`ppAnimateByFifthLevel` 或者 `ppAnimateByAllLevels`。

如果将 `TextLevelEffect` 设置为 `ppAnimateByAllLevels` 或 `ppAnimateLevelNone` 之外的任何值，就可以使用 `TextUnitEffect` 属性来指定如何使文本具有动画效果。使用 `ppAnimateByParagraph` 是按段落产生动画，`ppAnimateByWord` 是按字（单词）产生动画，`ppAnimateByCharacter` 是按字符（字母）产生动画。

要按相反顺序显示动画，需将 `AnimateTextInReverse` 属性设置为 `msoTrue`。（默认值是 `msoFalse`。）

要控制动画的换片方式，需将 `AdvanceMode` 属性设置为 `ppAdvanceOnTime`（定时自动换片）或者 `ppAdvanceOnClick`（手动换片）。如果使用自动换片，则可使用 `AdvanceTime` 来指定在换片之前要等待多少秒钟。

要随着幻灯片的切换播放出预设的声音效果，需使用 `AnimationSettings` 对象的 `SoundEffect` 属性，以返回 `SoundEffect` 对象；使用 `Name` 属性来指定声音效果的名称，再使用 `Play` 方法来播放声音效果。也可以使用 `SoundEffect` 对象的 `ImportFromFile` 来播放一个单独的声音文件，以及使用 `FileName` 参数来指定声音文件的路径和文件名。

要控制如何播放媒体剪辑，需使用 `AnimationSettings` 对象的 `PlaySettings` 属性，以返回 `PlaySettings` 对象。例如：如果希望某声音循环播放直至下一声音出现，需将 `AnimationSettings` 对象之内的 `PlaySettings` 对象的 `LoopSoundUntilNext` 属性设置为 `msoTrue`。默认值是 `msoFalse`。

下述例子将自定义动画应用于以对象变量 `mySlide` 来标识的幻灯片上的第一个形状。该动画使用从右侧飞入的进入效果，播放来自一个文件的声音，使文本按第一级段落和全段进行动画显示，并在用户单击时换片：

```
With mySlide.Shapes(1).AnimationSettings
    .EntryEffect = ppEffectFlyFromRight
    .AdvanceMode = ppAdvanceOnClick
    .SoundEffect.ImportFromFile FileName:="D:\Media\Whistle4.wav"
    .TextLevelEffect = ppAnimateByFirstLevel
    .TextUnitEffect = ppAnimateByParagraph
End With
```

使用 HeadersFooters 进行工作

PowerPoint 使用 `HeaderFooter` 对象来表示幻灯片上的页眉、页脚、幻灯片编号（页码）以及日期和时间。各个 `HeaderFooter` 对象组成 `HeadersFooters` 集合，通过 `Master` 对象、`Slide` 对象或 `SlideRange` 对象的 `HeadersFooters` 属性，可以访问到 `HeaderFooter` 对象。

注意：备注页没有 `HeaderFooter` 对象。备注母版和讲义母版有页眉，但是其他对象没有。

返回所需要的页眉或页脚对象

要返回所需要的对象，需使用 `HeaderFooter` 对象的适当属性。

◆ 使用 DateAndTime 属性以返回日期和时间。

◆ 使用 Footer 属性以返回页脚。

◆ 使用 Header 属性以返回备注页或讲义上的页眉。

◆ 使用 SlideNumber 属性以返回幻灯片上的幻灯片编号，或备注页或讲义上的页码。

下述例子使用 Footer 属性来设置 SlideMaster 对象的 HeaderFooter 对象的文本，使得对幻灯片母版的改动，引起使用该母版的所有幻灯片有相应的改变：

```
myPresentation.SlideMaster.HeadersFooters.Footer.Text = "Sentience 102"
```

显示或隐藏页眉或页脚对象

要显示 HeaderFooter 对象，需将它的 Visible 属性设置为 msoTrue。要隐藏 HeaderFooter 对象，需将它的 Visible 属性设置为 msoFalse。例如，下述语句隐藏活动演示文稿中第五张幻灯片上的页脚：

```
ActivePresentation.Slides(5).HeadersFooters.Footer.Visible = msoFalse
```

设置页眉或页脚中的文本

要设置 HeaderFooter 对象中所希望的文本，需将包含此文本的字符串指派给该对象的 Text 属性。例如，下述语句将活动演示文稿中第五张幻灯片的页脚的文本设置为 Confidential：

```
ActivePresentation.Slides(5).HeadersFooters.Footer.Text = "Confidential"
```

为使用日期和时间的页眉和页脚设置格式

如果幻灯片、备注页或讲义要在它们的页脚或页眉中使用日期和时间，那就要使用 Format 属性来指定日期和时间应该如何出现。表 25.2 列出了可以使用的常量。

表 25.2 与使用日期和时间的页眉和页脚相对应的格式属性常量

格式	例子
ppDateTimeddddMMMMddyyyy	Thursday, October 05, 2006
ppDateTimedMMMMyyyy	5 October 2006
ppDateTimedMMMyy	5-Oct-06
ppDateTimeHm	10: 17
ppDateTimehmmAMPM	10: 17 AM
ppDateTimeHmss	10: 17: 16
ppDateTimehmmssAMPM	10: 17: 16 AM
ppDateTimeMdyy	10/5/2006
ppDateTimeMMddyyHm	10/5/2006 10: 17AM
ppDateTimeMMddyyhmmAMPM	10/5/2006 10: 17: 16 AM
ppDateTimeMMMMMyyyy	October 5, 2006
ppDateTimeMMMMMyy	October 06
ppDateTimeMMyy	Oct-06

如果希望日期和时间会自动更新的话，需将 HeaderFooter 的 UseFormat 属性设置为 msoTrue。如果希望日期和时间保持不变，需将 UseFormat 属性设置为 msoFalse。

下节介绍一个使用 HeaderFooter 对象的 UseFormat 属性和 Format 属性的例子。

使演示文稿中所有页眉和页脚标准化

如果要编写过程，把来自各个现有演示文稿的若干幻灯片汇集成演示文稿，或者如果不同人员在他们的演示文稿中使用了不一致的页眉和页脚，这时，有可能需要使演示文稿中的所有页眉和页脚实现标准化。下述例子先清除原有的页眉和页脚，确保所有幻灯片均显示幻灯片母版上的各个形状，然后将一种页眉和页脚应用到演示文稿中的所有幻灯片：

```
Sub Standardize_Headers_and_Footers()

    Dim myPresentation As Presentation, mySlide As Slide
    Set myPresentation = ActivePresentation

    For Each mySlide In myPresentation.Slides
        mySlide.HeadersFooters.Clear
        mySlide.DisplayMasterShapes = msoTrue
    Next mySlide

    With myPresentation.SlideMaster.HeadersFooters
        With .Footer
            .Visible
            .Text = "Company Confidential"
        End With
        With .DateAndTime
            .Visible = True
            .UseFormat = True
            .Format = ppDateTimeMMMdyy
        End With
    End With
End Sub
```

设置和运行幻灯片放映

不仅可以使用 VBA 来汇集幻灯片放映和为其设置格式，还可以使用 VBA 来运行幻灯片放映。要设置幻灯片放映，需使用 Presentation 对象的 SlideShowSettings 属性以返回 SlideShowSettings 对象。在运行幻灯片放映时，VBA 会创建一个 SlideShowWindow 对象，然后就可以对该对象进行操作以控制幻灯片放映。

控制放映类型

要指定放映类型，需将 SlideShowSettings 对象的 ShowType 属性设置为 ppShowTypeSpeaker，以对应于演讲者放映（由演讲者来放映标准的全屏幕演示文稿）；或设置为 ppShowTypeKiosk，以对应于展台浏览（在展台放映全屏幕演示文稿）；或设置为 ppShowTypeWindow，以对应于观众自行浏览（观众在窗口内自行浏览各个单独的演示文稿）。在窗口内放映时，可以使用 Left 和 Top 属性来指定窗口左上角的位置，使用 Height 和 Width 属性来指定其大小。

要控制放映时是否加动画和旁白，需将 SlideShowSettings 对象的 ShowWithAnimation 属性和 ShowWithNarration 属性设置为 msoTrue 或者 msoFalse。

要控制演示文稿是否持续循环放映，直到用户按 Esc 键时才终止，需将 SlideShowSettings 对象的 LoopUntilStopped 属性设置为 msoTrue 或者 msoFalse。

要控制演示文稿如何换片，需将 AdvanceMode 属性设置为 ppSlideShowManualAdvance（对应于手动换片），或者 ppSlideShowUseSlideTimings（对应于使用已设置好的定时进行自动换片），或者 ppSlideShowRehearseNewTimings（对应于放映时排练新的定时）：

```
With ActivePresentation.SlideShowSettings
    .LoopUntilStopped = msoCTrue
    .AdvanceMode = ppSlideShowUseSlideTimings
    .ShowType = ppShowTypeKiosk
    .Run
End With
```

下述例子将名为 Corporate.ppt 的演示文稿设置为演讲者放映（全屏幕）方式，画面大小为 800×600 像素，将画面定位于屏幕的左上角，放映时使用手动换片：

```
With Presentations("Corporate.ppt").SlideShowSettings
    .LoopUntilStopped = msoFalse
    .ShowType = ppShowTypeSpeaker
    .AdvanceMode = ppSlideShowManualAdvance
    With .Run
        .Height = 600
        .Width = 800
        .Left = 0
        .Top = 0
    End With
End With
```

创建自定义放映

演示文稿之内的自定义放映使用 SlideShowSettings 对象之内的 NamedSlideShows 集合来表示。使用 SlideShowSettings 对象的 NamedSlideShows 属性以返回 NamedSlideShows 集合。

要创建一个自定义放映，需使用 NamedSlideShows 集合的 Add 方法，其语法是：

```
expression.Add(Name, SafeArrayOfSlideIDs)
```

此处，expression 是必需的表达式，它返回一个 NamesSlideShows 对象。Name 是必需的字符串参数，它指定要指派给新的自定义放映的名称。SafeArrayOfSlideIDs 是必需的 Variant 参数，它指定要在自定义放映中包括进去的各幻灯片的编号或名称。

例如，下述语句声明一个长整型数据类型的数组，将出自名为 Corporate.ppt 的演示文稿中的第 2、4、5 和 10 张幻灯片指派给它，使用该数组来创建一个名为 Short Show 的新自定义放映：

```
Dim myArray(4) As Long
With Presentations("Corporate.ppt")
    myArray(1) = .Slides(2).SlideID
    myArray(2) = .Slides(4).SlideID
    myArray(3) = .Slides(5).SlideID
    myArray(4) = .Slides(10).SlideID
    .SlideShowSettings.NamedSlideShows.Add Name: "Short Show",
        safeArrayOfSlideIDs:=myArray
End With
```

删除自定义放映

要删除一个自定义放映，需使用与适当的 NamedSlideShow 对象对应的 Delete 方法。例如，下述语句删除出自活动演示文稿的名为 Overview 的自定义放映：

```
ActivePresentation.SlideShowSettings.NamedSlideShows("Overview").Delete
```

开始幻灯片放映

要开始进行使用整个演示文稿的幻灯片放映，需使用 SlideShowSettings 对象的 Run 方法。例如，下述语句在以对象变量 myPresentation 来标识的演示文稿中开始运行幻灯片放映：

```
myPresentation.SlideShowSettings.Run
```

如果只放映出自某一个演示文稿的一个幻灯片子集，需将 SlideShowSettings 对象的 RangeType 属性设置为 ppShowSlideRange，使用 SlideShowSettings 对象的 StartingSlide 属性来指定第一张幻灯片；使用 EndingSlide 属性来指定最后一张幻灯片，然后使用 Run 方法来运行该演示文稿。下述例子将放映名为 Corporate.ppt 的演示文稿中的第四张至第八张幻灯片：

```
With Presentations("Corporate.ppt").SlideShowSettings
    .RangeType = ppShowSlideRange
    .StartingSlide = 4
    .EndingSlide = 8
    .Run
End With
```

为了开始运行一个自定义放映，先将 SlideShowSettings 对象的 RangeType 属性设置为 ppShowNamedSlideShow，使用 SlideShowName 属性来指定自定义放映的名称，然后使用 Run 方法来运行该自定义放映。下述例子进行活动演示文稿中名为 Short Show 的自定义放映：

```
With ActivePresentation.SlideShowSettings
    .RangeType = ppShowNamedSlideShow
    .SlideShowName = "Short Show"
    .Run
End With
```

开始幻灯片放映时，VBA 创建一个表示对象的 SlideShowWindow 对象，通过 SlideShowWindows 集合（一个可创建对象，它包含与每一个打开的幻灯片放映相对应的 SlideShowWindow 对象）或者 Presentation 对象的 SlideShowWindow 属性，可以访问 SlideShowWindow 对象。如果知道哪一个演示文稿正在运行，那么通过相应的 Presentation 对象来访问是比较方便的。

改变幻灯片放映的大小和位置

要搞清楚某一个幻灯片放映是全屏幕显示还是在一个窗口内显示，需检查 SlideShowWindow 对象的 IsFullScreen 属性。如果 IsFullScreen 属性返回 -1，那么演示文稿是全屏幕

显示；如果该属性返回 0，那么演示文稿是在一个窗口内显示。

要以像素数来指定幻灯片放映窗口的高度和宽度，需使用 Height 属性和 Width 属性。要设置其位置，需使用 Top 属性以像素数来指定演示文稿的上边缘离窗口或屏幕的上边缘的距离，以及使用 Left 属性以像素数来指定演示文稿的左边缘离窗口或屏幕的左边缘的距离。

在各幻灯片之间移动

除了控制幻灯片放映的大小和位置之外，对它采取的大多数行动都要涉及 View 对象。要搞清楚哪张幻灯片被显示，则需返回 CurrentShowPosition 属性：

```
MsgBox ActivePresentation.SlideShowWindow.View.CurrentShowPosition
```

要显示演示文稿中的第一张幻灯片，需使用 First 方法。要显示最后一张幻灯片，需使用 Last 方法：

```
ActivePresentation.SlideShowWindow.View.First  
ActivePresentation.SlideShowWindow.View.Last
```

要显示下一张幻灯片，需使用 Next 方法。要显示前一张幻灯片，需使用 Previous 方法，例如：

```
ActivePresentation.SlideShowWindow.View.Previous
```

要显示幻灯片放映中某一张特定的幻灯片，需使用 View 对象的 GotoSlide 方法，以 Index 参数来指明该幻灯片的编号。例如，下述语句显示第一个打开的幻灯片放映窗口中的幻灯片 5：

```
Application.SlideShowWindows(1).View.GotoSlide Index:=5
```

暂停放映和使用白屏与黑屏

要显示黑屏，需将 View 对象的 State 属性设置为 ppSlideShowBlackScreen。要显示白屏，需将 State 属性设置为 ppSlideShowWhiteScreen：

```
ActivePresentation.SlideShowWindow.View.State = ppSlideShowBlackScreen  
ActivePresentation.SlideShowWindow.View.State = ppSlideShowWhiteScreen
```

要关掉黑屏或白屏并再次开始放映，需将 State 属性设置为 ppSlideShowRunning。

要暂停演示文稿的放映，需将 View 对象的 State 属性设置为 ppSlideShowPaused。要再次开始放映，需将 State 属性设置为 ppSlideShowRunning。例如：

```
With ActivePresentation.SlideShowWindow.View  
    .State = ppSlideShowPaused  
    .State = ppSlideShowRunning  
End With
```

切换到自定义放映和结束自定义放映

要切换到自定义放映，需使用 GotoNamedShow 方法，并使用 SlideShowName 参数来

指定自定义放映的名称。例如，下述语句切换到名为 New Show 的自定义放映：

```
SlideShowWindows(1).GoToNamedShow SlideShowName:="New Show"
```

要结束自定义放映，需使用 EndNamedShow 方法，然后使用 Next 方法以切换到演示文稿。PowerPoint 随后便显示整个演示文稿的第一张幻灯片。例如：

```
With ActivePresentation.SlideShowWindow.View
    .EndNamedShow
    .Next
```

退出幻灯片放映

要退出幻灯片放映，需使用 SlideShowWindow 对象的 View 属性的 Exit 方法。例如，下述语句退出活动演示文稿中的幻灯片放映：

```
ActivePresentation.SlideShowWindow.View.Exit
```

第 26 章 了解 Outlook 对象模型和重要对象

- ◆ 获得 Outlook 对象模型的概括性知识
- ◆ 了解 Outlook 将 VBA 项目存储在何处
- ◆ 了解 Outlook 的可创建对象和主要的用户界面项目
- ◆ 使用 Application 对象进行工作
- ◆ 了解使用 Outlook 对象进行工作的一般方法
- ◆ 针对邮件进行工作
- ◆ 针对日历项目进行工作
- ◆ 针对任务和任务请求进行工作
- ◆ 搜索项目

在本章中，首先要了解 Outlook 对象模型以及如何使用 VBA 来操作 Outlook。学习 Outlook 将 VBA 项目存储在何处，了解与 Outlook 的可创建对象相对应的 VBA 对象以及主要的用户界面项目，要学习如何使用一些主要的 Outlook 对象进行工作，从使用表示整个 Outlook 应用程序的 Application 对象，直到使用表示邮件、日历和任务的各种对象来进行工作。还要学习如何以程序方式来搜索项目。

获得 Outlook 对象模型的概括性知识

很多人发现，要在使用 Outlook 时把握住 VBA 的使用，比使用其他应用程序时要困难一些，所以先看看 Outlook 对象模型，了解一下 Outlook 使用哪些对象以及对象之间有何关系是很有帮助的。要查看 Outlook 对象模型，需遵循如下步骤：

1. 启动或激活 Outlook，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“Outlook 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“Outlook 对象模型”条目，就可以看到帮助屏幕（如图 26.1 所示）。

在屏幕上（不是在书本的图上）观看这一条目时，会看到有些框的底色是黄色的，而另外有一些框的底色是蓝色的。黄框是组织成集合的对象，蓝框是没有集合的对象。

单击某一个框，就能看到与某一对象对应的帮助屏幕，包括对象模型中相关对象的图解。图 26.2 显示出与 Inspectors 集合对应的帮助屏幕，它包含 Outlook 中可用的 Inspector 对象的详细情况。

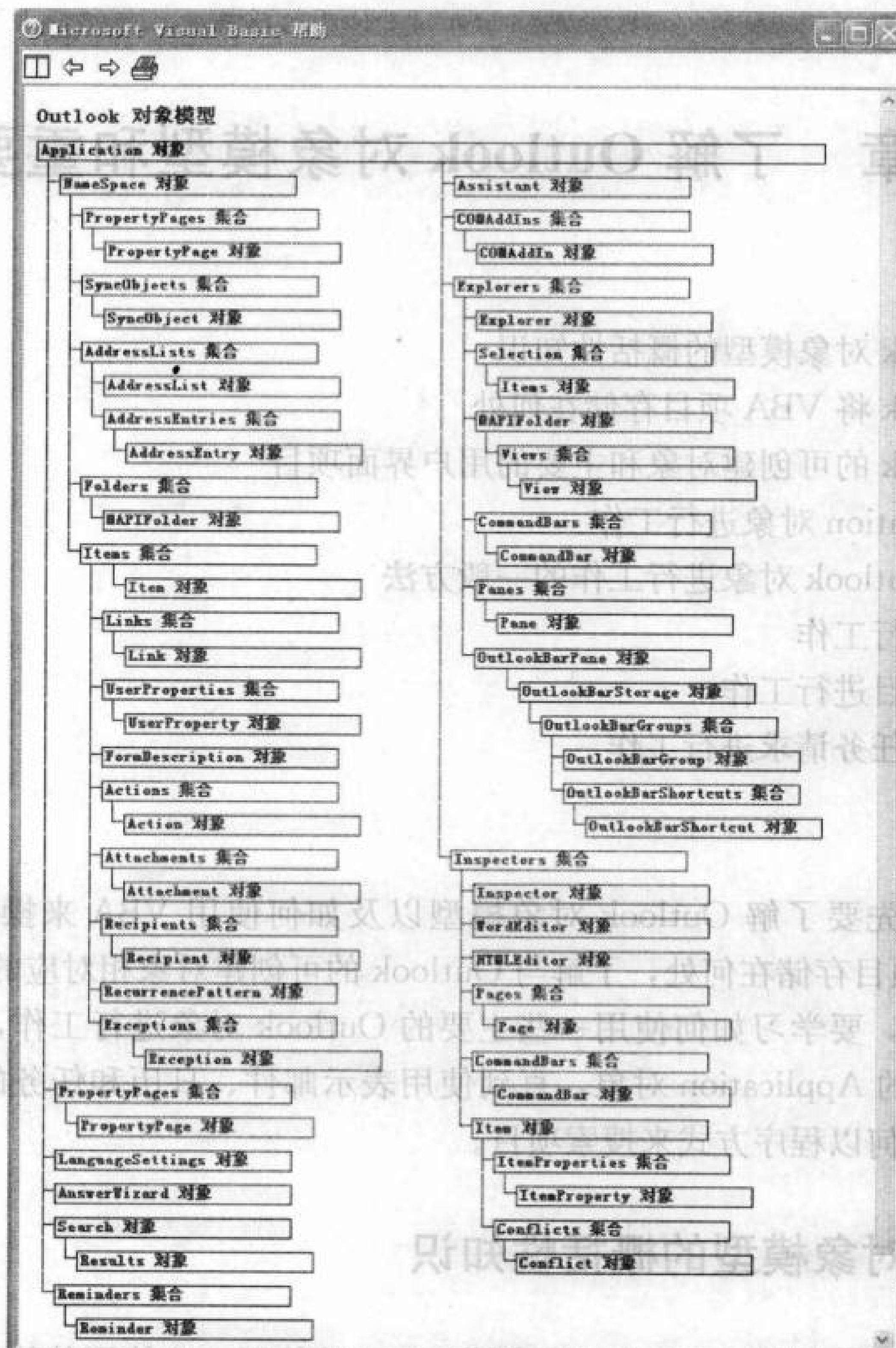


图 26.1 帮助文件中的 Outlook 对象模型，显示出 Outlook 各对象是如何相互关联的

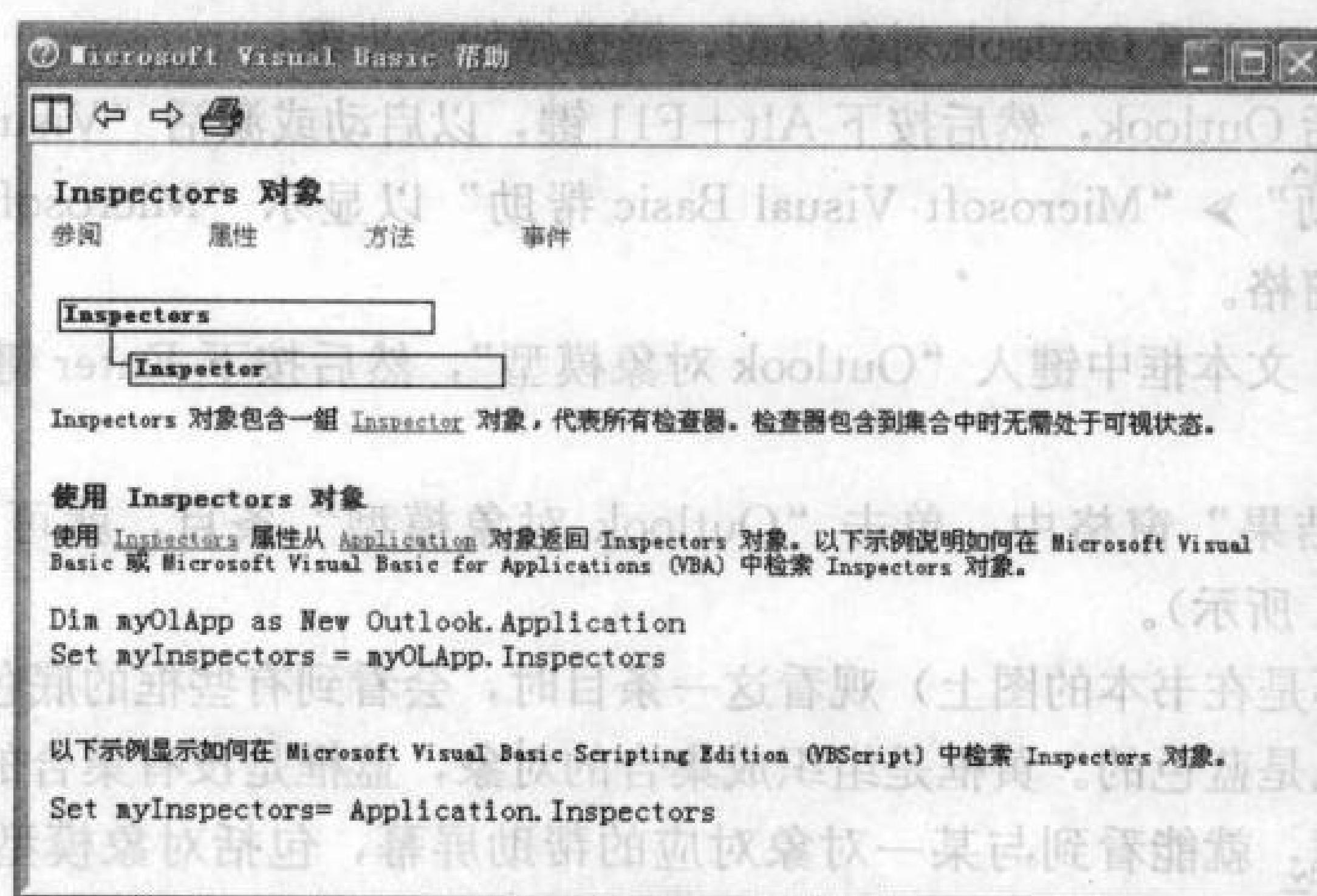


图 26.2 与各个对象或集合相对应的帮助屏幕，显示与其有关的各对象的图解

了解 Outlook 将 VBA 项目存储在何处

从本书前面的内容中可以知道，Word 和 Excel 把 VBA 工程存储在一个全局存储位置之内（Word 中的 Normal.dot 模板，或 Excel 中的个人宏工作簿），或存储在单独的模板或文档文件之内，PowerPoint 则是把 VBA 工程存储在演示文稿文件和模板之内。Outlook 与它们不同，它不把 VBA 工程存储在单独的项目（例如电子邮件或联系人）之内。Outlook 使用名为 VbaProject.OTM 的单个 VBA 工程，把它存储在 %userprofile%\Application Data\Microsoft\Outlook 文件夹之中。

注意：%userprofile% 是一个 Windows 环境变量，它存储到达某个文件夹的路径，而该文件夹包含有若干文件夹和文件，它们附有关于用户对 Windows 和自己的应用程序的优先和设置的详情。

了解 Outlook 的可创建对象和主要的用户界面项目

Application 对象表示整个 Outlook 应用程序，所以可以通过 Application 对象来访问任何 Outlook 对象。但是，Outlook 也具有很多可创建对象，即允许用户不必明显地通过 Application 对象就可以接触到对象模型中的一些对象。下面列举出主要的可创建对象，在本章和下一章中会对它们之中的大多数有更详细的介绍：

- ◆ Explorers 集合，它包含与显示文件夹内容的每个窗口相对应的 Explorer 对象。
- ◆ Inspectors 集合，它包含与显示 Outlook 项目的每个打开的窗口相对应的 Inspector 对象。
- ◆ COMAddIns 集合，它包含与每个加载进 Outlook 中的 COM 加载项相对应的 COMAddIn 对象。
- ◆ Reminders 集合，它包含与每个提示相对应的 Reminder 对象。

Outlook 用户界面中最突出的一些对象在 VBA 中用一些项目来表示，这些项目的名称反映出对象的类型。例如：

- ◆ MailItem 对象表示邮件项目。
- ◆ ContactItem 对象表示联系人。
- ◆ TaskItem 对象表示任务。
- ◆ AppointmentItem 对象表示约会。
- ◆ JournalItem 对象表示日记条目。
- ◆ NoteItem 对象表示便笺。

本章后面及下一章将要学习如何对这些对象进行工作。

使用 Application 对象进行工作

在某一时刻只能有一个 Outlook 实例在运行。（与此不一样的是，在同一时刻 Word 或 Excel 可以有多个实例在运行。）在 Outlook 之内工作时，可能觉察不到这种限制，但是，如果创建了一个从其他应用程序中自动启动 Outlook 的过程，那么就有必要检查，在以程序方式创建出这个 Outlook 实例之前是否已经存在一个 Outlook 实例。（参见第 30 章中的有关内容，那里

对从另一个应用程序中以程序方式来访问某一个应用程序的问题做了一些指导。)

使用 NameSpace 对象进行工作

为了以程序方式针对 Outlook 项目（例如电子邮件、任务或联系人等）实施很多行动，需使用 Application 对象的 GetNameSpace 方法以返回表示数据源的根对象的 NameSpace 对象。其语法是：

```
expression.GetNameSpace(Type)
```

此处，expression 是必需的表达式，它返回一个 Application 对象。Type 是必需的字符串参数，它指定需要返回的命名域的名称。Outlook 仅支持 MAPI 数据源，所以在使用 GetNameSpace 方法时，总是要使用 Type := "MAPI"。例如，下述语句返回 NameSpace，并使用 CurrentUser 属性在消息框中显示当前登录的用户的名称：

```
MsgBox Application.GetNameSpace("MAPI").CurrentUser
```

访问 NameSpace 对象之内的默认文件夹

NameSpace 对象包含 Outlook 使用的各个文件夹——既有保存各个项目（诸如电子邮件、任务和联系人等）的默认文件夹，又有用户创建或自定义过程创建的其他文件夹。在 VBA 中，这些文件夹都以组织成 Folders 集合的各个 MAPIFolder 对象来表示。

要访问 NameSpace 对象之内的默认文件夹，需使用 NameSpace 对象的 GetDefaultFolder 方法，其语法是：

```
expression.GetDefaultFolder(FolderType)
```

此处，expression 是必需的表达式，它返回一个 NameSpace 对象。FolderType 是必需的参数，它指定需要返回哪一个默认文件夹。各个常量是自解释型的：olFolderCalendar、olFolderContacts、olFolderDeletedItems、olFolderDrafts、olFolderInbox、olFolderJournal、olFolderNotes、olFolderOutbox、olFolderSentMail、olFolderTasks 或者 olPublicFoldersAllPublicFolders。

下面的例子创建对象变量 myCal，并将默认的日历文件夹指派给它：

```
Dim myCal As MAPIFolder
Set myCal = Application.GetNameSpace("MAPI") _ .GetDefaultFolder(FolderType:=olFolderCalendar)
```

访问 NameSpace 对象之内的其他文件夹

通过 GetDefaultFolder 方法来访问 NameSpace 对象之内的默认文件夹是容易的，但是常常还需要访问其他文件夹。为此，可以使用 Folders 集合来访问文件夹。

下面的例子显示一个消息框，内有命名域中包含的所有文件夹的列表（如图 26.3 所示）：

```
Sub List_All_NameSpace_Folders()
    Dim myNS As NameSpace
    Dim myFolder As MAPIFolder
    Dim mySubfolder As MAPIFolder
    Dim strFolderList As String

```

```

strFolderList = "Your Outlook NameSpace contains these folders:" & vbCrLf
& vbCrLf

Set myNS = Application.GetNamespace("MAPI")
With myNS
    For Each myFolder In myNS.Folders
        strFolderList = strFolderList & myFolder.Name & vbCrLf
        For Each mySubfolder In myFolder.Folders
            strFolderList = strFolderList & "*" & mySubfolder.Name & vbCrLf
        Next mySubfolder
    Next myFolder
End With
MsgBox strFolderList, vbOKOnly + vbInformation, "Folders in NameSpace"
End Sub

```



图 26.3 包含在 NameSpace 对象中的文件夹列表

创建新文件夹

要创建一个新文件夹，需使用与 Folders 集合相对应的 Add 方法，其语法是：

expression.Add(Name, Type)

此处，expression 是必需的表达式，它返回一个 Folders 集合。Name 是必需的字符串参数，用来指定指派给新文件夹的名称。Type 是可选的长整型参数，用来指定要创建的文件夹的类型：olFolderCalendar、olFolderContacts、olFolderDrafts、olFolderInbox、olFolderJournal、olFolderNotes 或者 olFolderTasks。如果省略 Type 参数，Outlook 就指定新文件夹与它的父文件夹（在其中创建新文件夹的那个文件夹）是相同的类型。

下述语句在 Tasks 文件夹中创建一个名为 Personal Tasks 的新文件夹，并明显地指定新文件夹为 olFolderTasks 类型：

```

Application.GetNamespace("MAPI").GetDefaultFolder.olFolderTasks)
.Folders.Add Name:="Personal Tasks", Type:=olFolderTasks

```

删除文件夹

要删除一个文件夹，需使用与适当的 MAPIFolder 对象相对应的 Delete 方法。这个方法不用参数。下面的例子删除 Tasks 文件夹中名为 Personal Tasks 的文件夹：

```

Application.GetNamespace("MAPI").GetDefaultFolder.olFolderTasks)
.Folders("Personal Tasks").Delete

```

警告：删除与 Outlook 相对应的对象时应加小心。首先，在删除对象之前，Outlook 不请求确认。其次，删除是永久性的。

使用检查器和浏览器进行工作

VBA 使用两个大多数用户都未能从使用 Qutlook 用户界面进行的工作中辨认出来的重要 的 Outlook 对象，它们是：

- ◆ Inspector 对象，它表示显示 Outlook 项目的窗口，例如显示电子邮件的窗口或显示约会的窗口。
- ◆ Explorer 对象，它表示显示文件夹内容的窗口。

注意：与很多集合不同，Explorer 对象包括在 Explorers 集合之中，即使它是不可见的。

打开检查器窗口

为了打开与某一个对象相对应的检查器窗口，需使用 Inspector 对象的 Display 方法。例如，下述语句打开与一个对象相对应的检查器窗口，该对象是以对象变量 myItem 来引用的：

```
myItem.Display
```

返回与项目相关联的检查器

要返回与某一个项目相关联的检查器，需使用适当的对象的 GetInspector 属性。下述例子返回一个与以对象变量 myItem 标识的项目相对应的检查器：

```
myItem.GetInspector
```

返回活动窗口、检查器或浏览器

与 Word、Excel 和 PowerPoint 不同，Outlook 没有表示活动窗口的 ActiveWindow 对象。但是，Outlook 的 Application 对象具有 ActiveWindow 方法，它返回最顶层的 Outlook 窗口。（如果没有窗口，ActiveWindow 返回 Nothing。）这个窗口是 Inspector 对象或者 Explorer 对象。同样地，Application 对象的 ActiveExplorer 方法返回活动的浏览器，而 Application 对象的 ActiveInspector 方法返回活动的检查器。

可以使用 TypeName 函数来确定哪种类型的窗口是活动的。下述例子在存在活动窗口的情况下显示一个消息框，以便指明活动窗口是哪种类型的：

```
If Not TypeName(ActiveWindow) = "Nothing" Then
    MsgBox "An " & TypeName(ActiveWindow) & " window is active."
End If
```

使用活动检查器进行工作

在很多过程中都需要确定，哪一个检查器是 Outlook 应用程序中最顶层的检查器，以便能够针对这个检查器进行工作，或者在使用其他检查器的过程结束时，能够使这个检查器恢复到最顶层位置。要返回最顶层的检查器，可以使用 Application 对象的 ActiveInspector 方

法。例如，下述语句最大化最顶层的检查器窗口：

```
Application.ActiveInspector.WindowState = olMaximized
```

Outlook 并不总是有活动检查器的，所以在试图对活动检查器进行操作之前，确认有没有活动检查器是一个好主意。为此，需检查，当运行 Application 对象的 ActiveInspector 方法时，TypeName 函数是不是返回 Nothing：

```
If TypeName(Application.ActiveInspector) = "Nothing" Then
    MsgBox "No item is open."
End If
```

创建项目

要创建 Outlook 中的新项目，需使用 Application 对象的 CreateItem 方法或 CreateItemFromTemplate 方法。CreateItem 方法创建默认项目，而 CreateItemFromTemplate 方法创建基于指定模板的项目。

注意：也可以使用自定义窗体来创建新对象。为此，需使用与 Items 集合相对应的 Add 方法。

使用 CreateItem 方法创建默认项目

与 CreateItem 方法相对应的语法是：

```
expression.CreateItem(ItemType)
```

此处，expression 是必需的表达式，它返回一个 Application 对象。ItemType 是必需的参数，它指定要创建的项目的类型：olAppointmentItem、olContactItem、olDistributionListItem、olJournalItem、olMailItem、olNoteItem、olPostItem 或者 olTaskItem。

下述例子创建一个新的电子邮件，指定一个收件人（通过设置 To 属性）、一个主题（通过设置 Subject 属性）和正文（通过设置 Body 属性），然后显示这个邮件窗口：

```
Dim myMessage As MailItem
Set myMessage = Application.CreateItem(ItemType:=olMailItem)
With myMessage
    .To = "test@example.com"
    .Subject = "Test message"
    .Body = "This is a test message."
    .Display
End With
```

使用 CreateItemFromTemplate 方法创建基于模板的项目

与使用 CreateItem 方法创建默认项目不同，可以使用 Application 对象的 CreateItemFromTemplate 方法创建基于模板的新项目。与 CreateItemFromTemplate 方法相对应的语法是：

```
expression.CreateItemFromTemplate(TemplatePath, InFolder)
```

此处，expression 是必需的表达式，它返回一个 Application 对象。TemplatePath 是必需的字符串参数，它指定基于模板创建新项目时所用模板的路径和文件名。InFolder 是可选的 Variant 参数，用来指定要在其中创建项目的文件夹。如果省略 InFolder 参数，Outlook 就在与该项目的类型相对应的默认文件夹中创建该项目。

下述例子基于自定义模板 Quick Note.oft，该模板存储在某个用户专用的 \ Application Data \ Microsoft \ Templates 文件夹之中，来创建一个新的便笺项目，然后显示这个新的便笺项目：

```
Dim myNoteItem As NoteItem
Set myNoteItem = Application.CreateItemFromTemplate _
    ("C:\Documents and Settings\Jana Cook\Application Data\" & _
    "Microsoft\Templates\Quick Note.oft")
myNoteItem.Display
```

关闭 Outlook

要关闭 Outlook，需使用 Application 对象的 Quit 方法，这个方法不用参数：

```
Application.Quit
```

注意：可能希望使用 Application 对象可用的一些事件来进行工作。请参阅第 27 章，那一章对如何使用应用程序级事件和项目级事件进行工作做了讨论。

了解使用 Outlook 对象进行工作的一般方法

Outlook 中的很多对象都要使用下面叙述的一些方法，为了节省篇幅，本节只讨论具有共性的一些方法以及使用这些方法的简明的例子。本章后面及下一章要分若干节来讨论各种对象类型——电子邮件、约会、联系人、任务，等等，其中可以看到更多的进一步的例子。

使用 Display 方法

要打开某一个检查器窗口之内的一个项目，需使用 Display 方法，其语法是：

```
expression.Display(Modal)
```

此处，expression 是必需的表达式，它返回想要显示的对象的类型，例如 ContactItem 对象或 MailItem 对象。Modal 是可选的 Variant 参数，将它设置为 True，是使该窗口为模式窗口；如果使用默认方式或将 Modal 设置为 False，是使该窗口为无模式窗口。使某个窗口为模式窗口，意味着用户必须关闭该窗口，然后才能使用另一个窗口进行工作。

注意：Modal 参数不能应用于 Explorer 和 MAPIFolder 对象。

例如，下述语句使用 Display 方法来显示收件箱：

```
Application.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Display
```

使用 Close 方法

要关闭一个窗口，需使用 Close 方法，其语法是：

```
expression.Close(SaveMode)
```

此处，`expression` 是必需的表达式，它返回想要关闭的对象。`SaveMode` 是必需的参数，它指明要保存更改（`olSave`），要放弃更改（`olDiscard`），还是提示用户去决定是否保存更改（`olPromptForSave`）。

下述例子关闭活动检查器，并保存对该检查器内容的更改：

```
ActiveInspector.Close SaveMode:=olSave
```

使用 Delete 方法

要删除一个项目，需使用 `Delete` 方法。这个方法不用参数。下述例子删除联系人文件夹中索引号为 1 的项目：

```
Application.GetNamespace("MAPI").GetDefaultFolder.olFolderContacts _  
.Items(1).Delete
```

使用 PrintOut 方法

要打印一个项目，可以使用 `PrintOut` 方法。这个方法不用参数。下述例子打印 Inbox 中索引号为 1 的项目：

```
Application.GetNamespace("MAPI").GetDefaultFolder.olFolderInbox _  
.Items(1).PrintOut
```

使用 Save 方法

要保存一个项目，需使用 `Save` 方法。这个方法不用参数。下述例子创建一个新任务，为其指定主题、开始日期和截止日期，关断对这个任务的提醒，然后保存它：

```
Dim myTask As TaskItem  
Set myTask = Application.CreateItem(ItemType:=olTaskItem)  
With myTask  
    .Subject = "Arrange Review Meeting"  
    .StartDate = Date  
    .DueDate = Date + 7  
    .ReminderSet = False  
    .Save  
End With
```

使用 SaveAs 方法

要把某一个项目另存为另一个单独的文件，需使用 `SaveAs` 方法，其语法是：

```
expression.SaveAs(Path, Type)
```

此处，`expression` 是必需的表达式，它返回要保存的对象。`Path` 是必需的字符串参数，它指定路径和文件名，就是在此路径和文件名之下来保存该文件。`Type` 是可选的 Variant 参数，用它来控制该文件所使用的文件类型，如下表所示。

参数	文件类型
<code>olHTML</code>	HTML 文件
<code>olMSG</code>	Outlook 邮件格式（.msg 文件）

olRTF	Rich 文本格式 (RTF 文件)
olTemplate	模板
olDoc	Word 文档格式 (使用 WordMail 的电子邮件)
olTXT	文本文件
olVCal	vCal 文件
olVCard	vCard 文件
olICal	iCal 文件
olMSGUnicode	Outlook Unicode 邮件格式 (.msg 文件)

下述例子保存在活动检查器中打开的邮件。如果 ActiveInspector 对象的 IsWordMail 属性返回 True，这个例子就把该邮件保存为一个 .doc 文件；如果 IsWordMail 属性返回 False，就把该邮件保存为一个 .rtf 文件。如果没有检查器窗口是活动的，就显示一个消息框，把问题告诉用户：

```
If TypeName(ActiveInspector) = "Nothing" Then
    MsgBox "This macro cannot run because " & _
        "there is no active window.", vbOKOnly, "Macro Cannot Run"
End If
Else
    If ActiveInspector.IsWordMail Then
        ActiveInspector.CurrentItem.SaveAs "c:\keep\message.doc"
    Else
        ActiveInspector.CurrentItem.SaveAs "c:\keep\message.rtf"
    End If
End If
```

针对邮件进行工作

如果同事之间广泛地使用 Outlook 的电子邮件功能，那么，对 Outlook 编程以便自动地创建或处理邮件，可能会节省时间。本节介绍如何创建一个新邮件，对其内容进行工作，添加一个附件，然后发送该邮件。

创建新邮件

要创建一个新邮件，需使用 Application 对象的 CreateItem 方法，并指定与 ItemType 参数对应的 olMailItem 常量。下述例子创建一个名为 myMessage 的 MailItem 对象变量，并将一个新邮件指派给它：

```
Dim myMessage As MailItem
Set myMessage = Application.CreateItem(ItemType:=olMailItem)
```

针对邮件内容进行工作

要针对邮件内容进行工作，需设置或获得适当的属性。下面是使用最广泛的一些属性：

属性

说明

To

邮件的一个或多个收件人

CC

邮件抄送副本的一个或多个收件人

BCC	邮件的密件抄送副本的一个或多个收件人
Subject	邮件的主题行
Body	邮件的正文文本
BodyFormat	邮件的格式设置类型：olFormatPlain 对应于纯文本，olFormatRichText 对应于 Rich 文本格式，olFormatHTML 对应于 HTML 格式
Importance	邮件的重要性级别，设置为 olImportanceHigh、olImportanceNormal 或 olImportanceLow

下述例子创建一个新的邮件项目，并把它指派给对象变量 myMessage，然后添加一个附件、一个主题以及正文文本，应用 HTML 格式，将重要性级别设置为高，再发送该邮件：

```

Dim myMessage As MailItem
Set myMessage = Application.CreateItem(ItemType:=olMailItem)
With myMessage
    .To = "petra_smith@ourbigcompany.com"
    .Subject = "Preparation for Review"
    .Body = "Please drop by tomorrow and spend a few minutes" _
        & " discussing the materials we need for the review."
    .BodyFormat = olFormatHTML
    .Importance = olImportanceHigh
    .Send
End With

```

给邮件添加附件

要给邮件添加一个附件，需使用与 Attachments 集合相对应的 Add 方法。使用 MailItem 对象的 Attachments 属性即返回 Attachments 集合。与所用的 Add 方法相对应的语法是：

expression.Add(Source, Type, Position, DisplayName)

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Attachments 集合。
- ◆ Source 是必需的字符串参数，它指定附件的路径和文件名。
- ◆ Type 是可选的字符串参数，它用来指定附件的类型。
- ◆ Position 是可选的字符串参数，把它同 Rich 文本邮件一起使用，以便指定附件应定位在文本中何字符处；使用字符 0 是隐藏附件；使用字符 1 是将附件定位在邮件的起始处；指定一个更高的数值时，即将附件定位在指定的字符位置处。要把附件定位在邮件末尾处，应使用比邮件中字符数目更高的一个数字。
- ◆ DisplayName 是可选的字符串参数，用来指定邮件中附件显示的名称。

下述例子把存储在文件夹 Y:\Sample Documents 之中的文件 Corporate Downsizing.ppt 作为以对象变量 myMessage 来引用的邮件的附件，将该附件定位在邮件的起始处，并设置它的显示名称为 Downsizing Presentation：

```

myMessage.Attachments.Add _
    Source:="Y:\Sample Documents\Corporate Downsizing.ppt", _
    Position:=1, DisplayName:="Downsizing Presentation"

```

发送邮件

要发送一封邮件，需使用 Send 方法。这个方法不用参数。下述例子发送以对象变量 myMessage 来引用的邮件：

```
myMessage.Send
```

注意：Send 方法应用于 Appointment Item 对象、Meeting Item 对象 TaskItem 对象及 MailItem 对象。要检查是否发送了一封邮件，需查看其 Sent 属性。如果发送了邮件，这个布尔型属性返回 True，否则返回 False。

针对日历项目进行工作

如果创建或接收很多日历项目，那么使用 VBA 可能能够节省用户的时间或安排好用户的日程。本节介绍如何创建日历项目以及针对它的内容进行工作。

创建新的日历项目

要创建一个新的日历项目，需使用 Application 对象的 CreateItem 方法，并指定与 ItemType 参数对应的 olAppointmentItem 常量。下述例子创建一个名为 myAppointment 的 AppointmentItem 对象变量，并将一个新约会指派给它：

```
Dim myAppointment As AppointmentItem
Set myAppointment = Application.CreateItem(ItemType:=olAppointmentItem)
```

针对日历项目内容进行工作

要针对日历项目内容进行工作，需设置或获得适当的属性。下面是使用最广泛的一些属性：

属性	说明
Subject	约会的主题
Body	约会的正文文本
Start	约会的开始时间
End	约会的结束时间
BusyStatus	约会期间用户的状态：olBusy、olFree、olOutOfOffice 或 olTentative
Categories	指派给该项目的类别
ReminderSet	为约会设置了提醒（True）或没有设置提醒（False）
ReminderMinutesBeforeStart	应在事件（约会）开始之前多少分钟提醒

下述例子创建一个新的 AppointmentItem 对象，并把它指派给对象变量 myAppointment。然后设置主题、正文、开始日期（当前日期之后七天，那天下午 2：30）、结束日期（开始之后一个小时），标明该时间为忙碌时间，指定为 Personal 类别，设定应在约会之前 30 分钟做提醒，然后保存该约会：

```

Dim myAppointment As AppointmentItem
Set myAppointment = Application.CreateItem(ItemType:=olAppointmentItem)
With myAppointment
    .Subject = "Dentist"
    .Body = "Dr. Schmitt" & vbCrLf & "4436 Acacia Blvd."
    .Start = Str(Date + 7) & " 2:30 PM"
    .End = Str(Date + 7) & " 3:30 PM"
    .BusyStatus = olBusy
    .Categories = "Personal"
    .ReminderMinutesBeforeStart = 30
    .ReminderSet = True
    .Save
End With

```

提示：以程序方式为某一个项目指定类别是有困难的，尤其是因为很多用户创建自定义类别或者按个人癖好来指定类别。在很多情况下，更好的办法是在过程中适当的时刻显示出“类别”对话框，让用户以手动方式指定类别。使用项目的 ShowCategoriesDialog 方法可以做到这一点——例如，对应于以对象变量 myAppointment 来引用的某一个项目，可以使用 myAppointment. ShowCategoriesDialog。

针对任务和任务请求进行工作

使用 VBA 来自动设置和保存任务与任务请求，也可以节省大量时间和精力。本节介绍如何创建任务，针对任务内容进行工作以及发送任务请求。

创建任务

要创建一个新的任务项目，需使用 Application 对象的 CreateItem 方法，并指定与 ItemType 参数对应的 olTaskItem 常量。下述例子创建一个名为 myTask 的 TaskItem 对象变量，并将一个新任务项目指派给它：

```

Dim myTask As TaskItem
Set myTask = Application.CreateItem(ItemType:=olTaskItem)

```

针对任务项目内容进行工作

要针对任务项目内容进行工作，需设置或获得适当的属性。下面是使用最广泛的一些属性：

属性	说明
Subject	任务的主题
Body	任务的正文文本
Start	任务的开始时间
DueDate	任务的截止日期
Importance	任务的重要性级别，设置为 olImportanceHigh、olImportanceNormal 或 olImportanceLow
Status	任务的状态：olTaskNotStarted、olTaskWaiting、olTaskDeferred、

olTaskInProgress 或 olTaskComplete

PercentComplete 已完成任务的百分比

Companies 与任务相关的公司

BillingInformation 为该任务付账的公司或部门

下述例子创建一个名为 myTask 的 TaskItem 对象变量，并把一个新的任务项目指派给它。然后设置任务的主题和正文，指定截止日期，将任务状态设置为 olTaskInProgress，而且任务已完成的百分比为 10%，指定所涉及的公司和谁为任务付账，将重要性级别设置为高，再保存该任务：

```
Dim myTask As TaskItem
Set myTask = Application.CreateItem(ItemType:=olTaskItem)
With myTask
    .Subject = "Create a business plan"
    .Body = "The business plan must cover the next four years." & _
        vbCr & vbCr & "It must provide a detailed budget, " & _
        "staffing projections, and a cost/benefit analysis."
    .DueDate = Str(Date + 28)
    .Status = olTaskInProgress
    .PercentComplete = 10
    .Companies = "Acme Polyglot Industrialists"
    .BillingInformation = "Sales & Marketing"
    .Importance = olImportanceHigh
    .Save
End With
```

将任务指派给同事

要将任务指派给同事，需使用 TaskItem 对象的 Assign 方法，再使用 Recipients 集合的 Add 方法来添加一个或多个收件人。然后使用 Send 方法，把任务发送给同事。

下述例子创建一个任务，使用 Assign 方法以表明要进行指派，然后指定收件人，并发送该任务：

```
Dim myTaskAssignment As TaskItem
Set myTaskAssignment = Application.CreateItem(ItemType:=olTaskItem)
With myTaskAssignment
    .Assign
    .Recipients.Add Name:="Peter Nagel"
    .Subject = "Buy Bagels for Dress-Down/Eat-Up Day"
    .Body = "It's your turn to get the bagels on Friday."
    .Body = .Body & vbCr & vbCr & "Remember: No donuts AT ALL."
    .DueDate = Str(Date + 3)
    .Send
End With
```

搜索项目

要搜索项目，需使用 Application 对象的 AdvancedSearch 方法，其语法是：

expression.AdvancedSearch(Scope, Filter, SearchSubFolders, Tag)

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，它返回一个 Application 对象。
- ◆ Scope 是必需的字符串参数，它指定搜索的范围（要搜索哪些项目），通常是搜索特定的文件夹。例如，可能要搜索匹配某些条件的邮件的收件箱，或者搜索对应于某些特定任务的 Tasks 文件夹。
- ◆ Filter 是可选的 Variant 参数，它指定搜索的筛选条件。尽管这个参数是可选的，但是，除非想要返回已指定的搜索范围内的所有项目，否则，还是有必要使用它。
- ◆ SearchSubFolders 是可选的 Variant 参数，将它设置为 True 时，在 Scope 参数所指定的文件夹的各个子文件夹中都要进行搜索；将它设置为 False 时，则仅搜索指定的文件夹。默认设置为 False。
- ◆ Tag 是可选的 Variant 参数，它指定作为搜索标识符的名称。如果创建了一个名称，就可以再次调用它。

注意：DASL 是 DAV Search and Locating”的缩写。DAV 是 Distributed Authoring and Versioning 的首字母简略词。详情请查 <http://www.webdav.org/dasl/>。

实施高级搜索的较复杂部分是创建搜索筛选器，它需要在 DASL 格式中进行。如果有 Outlook 2003 或更新版本，可以使 Outlook 包含筛选。为此，首先要把“筛选”按钮放到工具栏上去：

1. 在 Outlook 中，选择“工具”>“自定义”以显示“自定义”对话框。单击“命令”标签，如果它尚未显示的话。
2. 在“类别”列表框内，单击“视图”项目。
3. 在“命令”列表框内，向下滚动至“筛选”命令，然后将该命令拖曳到工具栏。

注意：如果喜欢的话，可以把“筛选”命令放在菜单上，而不是放在工具栏上。

4. 单击“关闭”按钮以关闭“自定义”对话框。

现在可以显示“筛选”对话框，并把筛选包含进去：

1. 单击工具栏上的“筛选”按钮以显示“筛选 (Filter)”对话框（如图 26.4 所示）。
2. 使用对“邮件 (Message)”页、“其他选择 (More Choices)”页和“高级 (Advanced)”页的控制以指定想要的筛选。
3. 单击“SQL”标签以显示“SQL”页，它包含已创建的筛选。
4. 选中“直接编辑这些条件，所有其他标签不使用 (Edit these criteria directly. All other tabs will be unavailable)”复选框，以便“查找与这些条件匹配的项目 (Find items that match these criteria)”文本框可以使用。
5. 拖曳鼠标以选择筛选，然后按下 Ctrl+C 键，将它复制到剪贴板。
6. 单击“取消 (Cancel)”按钮以关闭“筛选”对话框。
7. 按下 Alt+F11 键以接通“Visual Basic 编辑器”，然后将筛选粘贴进代码。

下述例子在收件箱内 (Scope: = "Inbox") 对带有主题行 Dam Project 的邮件进行搜索 (Filter: = "urn:schemas:mailheader:subject='Dam Project'")。如果有任何邮件被找到，此过程就产生一个发送人名单，将它指定给字符串变量 strMessages。如果没有邮件被找到，此过程将结果指定给 strMessages 文本，让用户知道这个结果。然后，过程在消息

框内带上标题 Search Results 来显示 StrMessages：

```

Sub Sample_Advanced_Search()
    Dim mySearch As Search
    Dim myResults As Results
    Dim intCounter As Integer
    Dim strMessages As String

    Set mySearch = AdvancedSearch(Scope:="Inbox", _
        Filter:="urn:schemas:mailheader:subject = 'Dam Project'")
    Set myResults = mySearch.Results
    If myResults.Count > 0 Then
        strMessages = "The Inbox contains messages that match " & _
            "the search criteria from these senders:" & vbCrLf & vbCrLf
        For intCounter = 1 To myResults.Count
            strMessages = strMessages & _
                myResults.Item(intCounter).SenderName & vbCrLf
        Next intCounter
    Else
        strMessages = "The Inbox contains no messages " & _
            "that match the search criteria."
    End If
    MsgBox strMessages, vbOKOnly, "Search Results"
End Sub

```

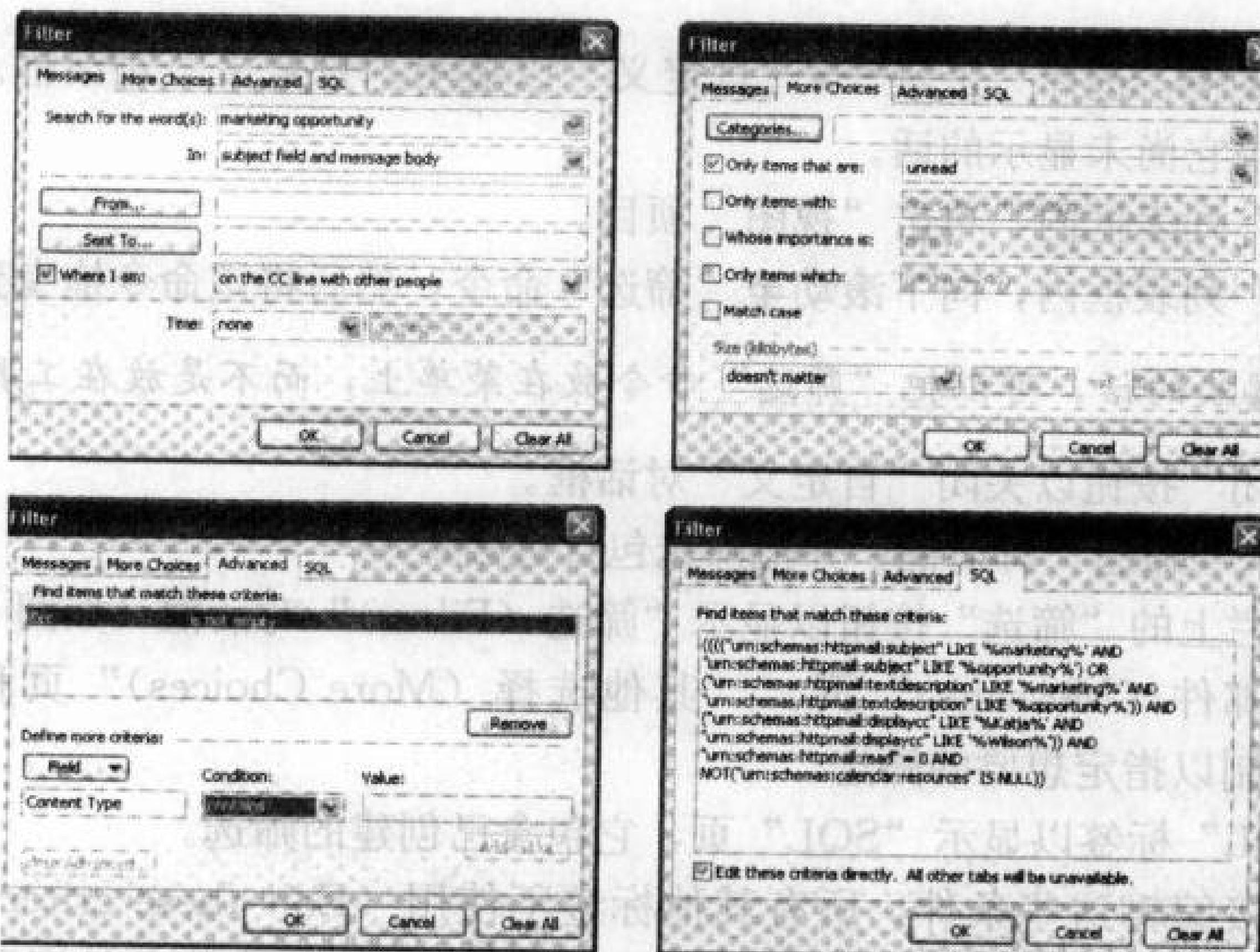


图 26.4 使用“筛选”对话框的“邮件”页、“其他选择”页和“高级”页，把高级搜索所需要的 DASL 筛选包含进去。在“SQL”页上，选中“直接编辑这些条件，所有其他标签不使用”复选框，选择筛选，然后按下 Ctrl+C 键，将它复制到剪贴板，以便粘贴进“Visual Basic 编辑器”。

注意：如果有必要，可以在同一时间运行两个或多个搜索。为此，在连续几个代码行中使用 AdvancedSearch 方法。但是，如果在同一时间运行多至 100 个搜索，这就会给计算机增加相当大的负担，可能会使计算机运行变慢或者使计算机停止响应。

如果要使 Outlook 的工作方式自动化，需要编写响应 Outlook 各事件的代码。Outlook 有两类事件，即应用程序级事件和项目级事件，这些事件便于用户编写对出现在 Outlook 中的大多数情况产生响应的代码。在本章中，将要学习如何使用这两类事件进行工作，并能看到有关事件的若干例子。

- ◆ 使用应用程序级事件进行工作
- ◆ 使用项目级事件进行工作

如果要使 Outlook 的工作方式自动化，需要编写响应 Outlook 各事件的代码。Outlook 有两类事件，即应用程序级事件和项目级事件，这些事件便于用户编写对出现在 Outlook 中的大多数情况产生响应的代码。在本章中，将要学习如何使用这两类事件进行工作，并能看到有关事件的若干例子。

注意：除了本章中讨论的事件之外，Outlook 还支持第 15 章“使用事件来控制窗体”一节中讨论过的事件。

使用应用程序级事件进行工作

应用程序级事件是与 Outlook 应用程序相关联，而不是仅与应用程序之中的某一个特定项目相关联的事件。例如，Startup 事件是 Outlook 启动时发生的一个应用程序级事件，而 Quit 事件是 Outlook 关闭时发生的一个应用程序级事件。与此不同的是，项目级事件表示与特定项目（例如，打开一个电子邮件或联系人记录，或者用户从一个文件夹转到另一个文件夹）相关联的情况。

应用程序级事件比项目级事件更易于访问，因为 Application 对象是最顶层对象，只要 Outlook 在运行，它总是可用的。这意味着，无需使用事件管理程序来创建 Application 对象，但是必须针对项目级事件创建对象。

要访问应用程序级事件，需使用 ThisOutlookSession 类模块。在“Visual Basic 编辑器”中，先扩展表示 Outlook VBA 工程的 Project1 项目，扩展 Microsoft Office Outlook 项目（在 Office 2003 中）或者 Microsoft Outlook Objects 项目（在 Office XP 或 Office 2000 中），然后双击 ThisOutlookSession 项目，以打开显示其内容的代码窗口。（如果是第一次打开 ThisOutlookSession 类模块，它还没有内容。）

Outlook 2003 支持以下各小节中讨论的应用程序级事件。Outlook 2000 仅支持 Startup、Quit、Reminder、ItemSend、NewMail 以及 OptionsPagesAdd 这些应用程序级事件。Outlook XP 支持除 NewMailEx 事件之外的所有应用程序级事件。

注意：每个事件都要依靠 Application 对象来工作。为了简单起见，下述例子大多数是使用 Outlook Application 对象本身。也可以使用对象变量来返回 Application 对象。

使用 Startup 事件

当 Outlook 启动时，Startup 事件发生，这个事件不用参数。要确认 Outlook 已经正确

配置以便用户开始工作，Startup 方法是很有用的。下述例子创建一个新的 NoteItem 对象（一个便笺），把文本指定给它的 Body 属性，再使用 Display 项目来显示它：

```
Private Sub Application_Startup()
    Dim myNoteItem As NoteItem
    Set myNoteItem = Application.CreateItem(ItemType:=olNoteItem)
    myNoteItem.Body = "Please start a new time card for the day."
    myNoteItem.Display
End Sub
```

使用 Quit 事件

当用户关闭 Outlook 时（例如，用户选择“文件”>“退出”来关闭 Outlook），或使用 VBA 中 Application 对象的 Quit 方法来关闭 Outlook 时，Quit 事件发生。这个事件发生时，所有窗口都被关闭，所有全局变量都被释放，所以，这个事件几乎没让什么东西留下来。一件还可能会做的事情是向用户显示一个临别邮件，如下例所示，这个例子是在公休假日的前一个工作日显示出一个邮件，提醒用户假日就要到了：

```
Private Sub Application_Quit()
    Dim strMessage As String
    Select Case Format(Date, "MM/DD/YYYY")
        Case "01/13/2006"
            strMessage = "Next Monday is Martin Luther King Day."
        Case "02/17/2006"
            strMessage = "Next Monday is Presidents Day."
        Case "05/26/2006"
            strMessage = "Next Monday is Memorial Day."
        Case "06/30/2006"
            strMessage = "Next Tuesday is Independence Day." & _
                         " Monday is a company holiday."
        Case "09/01/2006"
            strMessage = "Next Monday is Labor Day."
            'other National Holidays here
    End Select
    MsgBox strMessage, vbOKCancel + vbExclamation, "Don't Forget..."
End Sub
```

使用 ItemSend 事件

当用户发出一个 Send 命令（例如，单击邮件窗口中的“发送”按钮）使某一个项目被发送时，或使用 VBA 中的 Send 方法使项目被发送时，ItemSend 事件发生。与 ItemSend 事件对应的语法是：

```
Sub expression_ItemSend(ByVal Item As Object, Cancel As Boolean)
```

此处，expression 是必需的表达式，它返回一个 Application 对象。Item 是必需的参数，它指定正要发送的项目。Cancel 是可选的布尔型参数，将它设置为 True，则不发送该项目。

下述例子比较正要发送的 Item 对象的 Subject 属性。如果 Subject 属性是一个空字符串，消息框就提示用户添加一个主题行，再使用 Cancel = True 语句取消对项目的发送。

```

Private Sub Application_ItemSend(ByVal Item As Object, Cancel As Boolean)
    If Item.Subject = "" Then
        MsgBox "Please add a subject line to this message."
        Cancel = True
    End If
End Sub

```

使用 NewMail 事件和 NewMailEx 事件

当有一个或多个新邮件到达收件箱时，NewMail 事件发生。NewMail 事件对于自动分拣邮件是很有用的。（也可以使用自动分拣邮件的规则。）NewMail 事件不用参数。

当有新邮件到达以及 NewMail 事件被触发时，下述例子显示一个消息框，它提出是否要查看收件箱：

```

Private Sub Application_NewMail()
    If MsgBox("You have new mail. Do you want to see your Inbox?", _
              vbYesNo + vbInformation, "New Mail Alert") = vbYes Then
        Application.GetNamespace("MAPI").GetDefaultFolder(olFolderInbox).Display
    End If
End Sub

```

NewMailEx 事件是更复杂版本的 NewMail 事件，它传递自从最近一次这种事件发生以来收件箱中收到的各项目的列表。NewMailEx 事件仅为交换服务器提供已收到邮件通知的其他信箱传递这一列表。其语法是：

```
Sub expression.NewMailEx(EntryIDCollection As String)
```

此处，expression 是必需的表达式，它返回一个 Application 对象。EntryIDCollection 是一个字符串，它包含已经接收到的邮件的条目标识符。每个条目标识符以一个逗号来与下一个条目标识符分隔开；如果只有一个条目标识符，EntryIDCollection 字符串中就没有逗号。

下面是一个 NewMailEx 事件过程的例子，它使用一个 Do While...Loop 循环来分隔各个邮件标识符（依次使用 InStr 函数来鉴别 EntryIDCollection 字符串的各节，直至达到下一个逗号），再构建一个字符串，它包含有介绍性文本，后随每个邮件的主题行，一个邮件对应一行。然后，此过程在一个消息框内显示该字符串，这样，当 Outlook 接收到新邮件时，用户就收到各主题行的概要汇集：

```
Private Sub Application_NewMailEx(ByVal EntryIDCollection As String)
```

```

Dim myMailItem As Object
Dim intMsgIDStart As Integer, intMsgIDEnd As Integer
Dim intCutPoint As String, strMailItemID As String, strMailList As String

intMsgIDStart = 1
intCutPoint = Len(EntryIDCollection)

intMsgIDEnd = InStr(intMsgIDStart, EntryIDCollection, ",")息部分
strMailList = "You have the following messages:"个一显面不

Do While intMsgIDEnd > 0
    strMailItemID = Strings.Mid(EntryIDCollection, intMsgIDStart, _量变串齐字即
    intMsgIDStart = intMsgIDEnd + 1
    strMailList = strMailList & strMailItemID & vbCrLf
    intMsgIDEnd = InStr(intMsgIDStart, EntryIDCollection, ",")息部分
Loop
End Sub

```

```

        (intMsgIDEnd - intMsgIDStart))
Set myMailItem = Application.Session.GetItemFromID(strMailItemID)
strMailList = strMailList & vbCrLf & myMailItem.Subject
intMsgIDStart = intMsgIDEnd + 1
intMsgIDEnd = InStr(intMsgIDStart, EntryIDCollection, ",")  

Loop

MsgBox strMailList, vbOKOnly + vbInformation, "Mail Alert"
End Sub

```

提示：如果不使用 NewMail 事件和 NewMailEx 事件，还可以使用针对收件箱中各项目的 ItemAdd 事件来处理到达的每个新邮件。

使用 AdvancedSearchComplete 事件和 AdvancedSearchStopped 事件

为了对以 AdvancedSearch 方法创建的高级搜索进行工作，Outlook 提供了两种事件。当 AdvancedSearch 方法通过 VBA 运行并完成了搜索时，AdvancedSearchComplete 事件发生。当 AdvancedSearch 方法通过 VBA 运行，并因使用了该搜索的 Stop 方法而停止时，AdvancedSearchStopped 事件发生。

与 AdvancedSearchComplete 事件相对应的语法是：

```
Private Sub expression_AdvancedSearchComplete(ByVal SearchObject As Object)
```

此处，expression 是必需的表达式，它返回一个在类模块中以事件形式声明的 Application 类型的对象变量。SearchObject 是 AdvancedSearch 方法返回的 Search 对象。

下述例子使用 AdvancedSearchComplete 事件来返回 AdvancedSearch 方法所找到的搜索结果的数目：

```
Private Sub Application_AdvancedSearchComplete(ByVal SearchObject As Search)
    MsgBox "The search has finished running and found " & _
        SearchObject.Results.Count & " results.", vbOKOnly + vbInformation, _  

        "Advanced Search Complete Event"
End Sub
```

下述例子使用 AdvancedSearchStopped 事件来告知用户该搜索已经停止：

```
Private Sub Application_AdvancedSearchStopped(ByVal SearchObject As Search)
    MsgBox "The search was stopped by a Stop command.", vbOKOnly
End Sub
```

使用 MAPILogonComplete 事件

当用户已经成功地登录到 Outlook 时，MAPILogonComplete 事件发生。可以使用 MAPILogonComplete 事件来确认 Outlook 已经为用户做好了正确配置，或者简单地显示一个消息。MAPILogonComplete 事件不用参数。

下面是一个 MAPILogonComplete 过程的例子。当用户成功地登录到 Outlook 时，该过程显示出关于当前交易情况的消息。代码中含有一条注释，它指明在什么地方声明字符串变量 strPubDowBegin 和 strPubForecast，并为它们指定在整个实施过程中要使

用的数据：

```
Private Sub Application_MAPILogonComplete()
    Dim strMsg As String
    'strPubDowBegin and strPubForecast declared and assigned strings here
    strMsg = "Welcome to the UltraBroker Trading System!" & vbCrLf & vbCrLf
    strMsg = strMsg & "Today's starting value is " & strPubDowBegin & "."
    & vbCrLf & vbCrLf
    strMsg = strMsg & "Today's trading forecast is " & strPubForecast & "."
    MsgBox strMsg, vbOKOnly + vbInformation, _
        "UltraBroker Trading System Logon Greeting"
End Sub
```

使用 Reminder 事件

在显示会议、任务或约会的提醒之前，Reminder 事件立即发生。可以使用 Reminder 事件，来采取与该提醒相关的行动。对于引起用户对有关的会议、任务或约会的注意，这种提醒本身通常是胜任的，所以，在以程序方式访问 Outlook 时，Reminder 事件往往比以交互方式使用 Outlook 工作时更有用。与该事件相对应的语法是：

```
Sub expression_Reminder(ByVal Item As Object)
```

此处，expression 是必需的表达式，它返回一个 Application 对象。Item 是与该提醒相关的 AppointmentItem 对象、MailItem 对象、ContactItem 对象或者 TaskItem 对象。

使用 OptionsPagesAdd 事件

当打开“选项”对话框时（选择“工具”>“选项”），或者打开与某一个文件夹对应的“属性”对话框时（要打开对应于某一个文件夹的“属性”对话框，需双击该文件夹，然后从上下文菜单中选择“属性”），OptionsPagesAdd 事件发生。可以使用这一事件，将包含在已创建的 COM 加载项中的自定义页添加到“选项”对话框或“属性”对话框。与 OptionsPagesAdd 事件相对应的语法是：

```
Sub expression_OptionsPagesAdd(ByVal Pages As PropertyPages, _
    ByVal Folder As MAPIFolder)
```

此处，expression 是必需的表达式，它返回一个 Application 对象或 NameSpace 对象。Pages 是必需的参数，它给出添加到对话框的各个自定义属性页的集合。Folder 是 expression 返回 MAPIFolder 对象时必须使用的参数。Folder 返回为其打开“属性”对话框的 MAPIFolder 对象。

使用项目级事件进行工作

除了应用程序级事件之外，Outlook 还支持大量的项目级事件——与应用程序级事件是使应用程序整体受到影响不同，项目级事件是在项目被操作控制时发生的事件。

在 Outlook 中可以使用两种方式来处理项目级事件。

- ◆ 在类模块中声明事件并运行初始化过程，从而使得 VBA 在该事件发生时捕获该事件。

本章即采取这一方法。

- ◆ 创建 Visual Basic Script (VBScript) 代码，并将它放入项目所使用的窗体之中。这一方法对于自定义窗体特别有用，但是，它比上一个方法受到更多限制，因为 Outlook 支持的某些事件在 VBScript 中不可用。

声明对象变量并初始化事件

遵循如下步骤来声明对象变量并对事件进行初始化。

1. 确定要使用哪一个类模块来进行这种声明，为此可使用下述三种方法中的一种：
 - ◆ 要使用 ThisOutlookSession 模块，需双击表示 Outlook VBA 工程的 Project1 项目，双击 Microsoft Office Outlook 项目（在 Office 2003 中）或 Microsoft Outlook Objects 项目（在 Office XP 或 Office 2000 中），然后双击 ThisOutlookSession 项目。
 - ◆ 在“工程资源管理器”中右击，并从快捷菜单中选择“插入”>“类模块”，从而创建一个类模块。“Visual Basic 编辑器”自动为该类模块打开一个代码窗口。
 - ◆ 在“工程资源管理器”中双击一个已有的类模块，从而打开该类模块。
2. 在类模块起始处的声明区内，声明一个表示对象的变量，事件即应用于该变量。使用 WithEvents 关键字来指明该对象有若干事件。下述例子创建一个名为 myPublicContactItem 的公有变量：

```
Public WithEvents myPublicContactItem As ContactItem
```

3. 将这个对象变量设置为表示适当的对象，从而对该对象变量进行初始化。下述例子将 myPublicContactItem 变量设置为表示默认联系人文件夹中的第一个项目：

```
Set myPublicContactItem = Application.GetNamespace("MAPI")_
    .GetDefaultFolder(olFolderContacts).Items(1)
```

初始化对象变量后，过程便在事件发生之后运行。

在必要时，可以以手动方式来初始化对象变量。当编写代码来处理事件时，可能会发现这样做是很方便的。但是，如果每次 Outlook 运行时都需要处理事件，最好运行代码以自动初始化对象变量。例如，可以使用 Application 对象的 Startup 事件（在本章前面“使用 Startup 事件”一节中讨论过），在每次 Outlook 启动时自动地运行代码。

了解应用于所有消息项目的事件

很多项目级事件可以应用于所有的消息项目，这些项目是：AppointmentItem、ContactItem、DistListItem、DocumentItem、Explorer、Inspector、JournalItem、MailItem、MeetingItem、PostItem、RemoteItem、ReportItem、TaskItem、TaskRequestAcceptItem、TaskRequestDeclineItem、TaskRequestItem 以及 TaskRequestUpdateItem。表 27.1 列出了可以应用于这些项目的所有事件。

注意：Close 事件除了应用于上述各对象之外，还应用于 Inspector 对象和 Explorer 对象。

表 27.1 应用于所有消息项目的项目级事件

事件	事件发生于	是否在 VBScript 中可用
AttachmentAdd	将某一个附件添加到项目之后	是
AttachmentRead	用户打开某一个电子邮件的附件以便阅读时	是
BeforeAttachmentSave	用户选择了要保存某一个附件，但在该命令执行之前时	是
BeforeCheckNames	在 Outlook 检查要发送的项目的各收件人的名称之前	是
*BeforeDelete	删除某一个项目之前	是
Close	检查器要被关闭，但这种关闭发生之前	是
CustomAction	执行某一个项目的自定义行动时	是
CustomPropertyChange	更改某一个项目的一个自定义属性时	是
Forward	当用户转发某一个项目时	是
Open	某一个项目在检查器中被打开时	是
PropertyChange	更改项目中的一个标准属性时（标准属性与自定义属性不同）	是
Read	某一个项目被打开以便在检查器窗口中进行编辑时，或某一个项目被选定以便在单元格内编辑时	是
Reply	用户发出针对某一个项目的“答复”命令时	是
ReplyAll	用户发出“答复所有人”命令时	是
Send	一个“发送”命令已经发出，但项目被发送之前	是
Write	某一个项目被用户明显地保存，或被 Outlook 隐含地保存时	是

如果事件发生时某一个行动尚未发出，那么允许将这个行动取消，使其不发出。与这些事件对应的语法使用一个名为 Cancel 的布尔型参数。当把这个参数设置为 True 时，就可以避免此行动发出。例如，与 BeforeDelete 事件相对应的语法是：

```
Sub expression_BeforeDelete(ByVal Item As Object, Cancel As Boolean)
```

此处，expression 是必需的表达式，它返回一个消息项目，该事件即应用于这一项目（例如，一个 TaskItem 对象）。下述例子使用 BeforeDelete 事件来检查在用户试图删除某一个 TaskItem 对象时，这个在检查器中打开的对象是不是标明为完成。如果这个任务并未标明为完成，就用一个消息框来提示用户去完成此任务，然后本例将 Cancel 参数设置为 True，以免发生删除行动：

```
Private Sub myTaskItem_BeforeDelete(ByVal Item As Object, Cancel As Boolean)
    If myTaskItem.Complete = False Then
        MsgBox "Please complete the task before deleting it.", _
            vbOKOnly + vbExclamation, "Task Is Incomplete"
        Cancel = True
    End If
End Sub
```

注意：当用户打开一个已有项目以便编辑时，Read 事件和 Open 事件均会发生。

两种事件的区别在于，Open 事件是项目在检查器窗口中被打开时发生的，而 Read 事件是项目在检查器窗口中被打开时，以及项目被选定在单元格内编辑时发生的。

了解应用于浏览器、检查器和视图的事件

表 27.2 列出了应用于浏览器、检查器和视图的各种事件。某些事件既应用于浏览器，又应用于检查器。

表 27.2 应用于浏览器、检查器或视图的事件

事件	应用于	事件发生于	是否在 VBScript 中可用
BeforeFolderSwitch	浏览器	浏览器显示新文件夹之前	否
BeforeItemCopy	浏览器	用户发出了复制命令，但复制操作发生之前	是
BeforeItemCut	浏览器	从文件夹中剪切某一个项目时	是
BeforeItemPaste	浏览器	粘贴某一个项目之前	是
BeforeViewSwitch	浏览器	在 Outlook 窗口中视图更改之前	
FolderSwitch	浏览器	浏览器显示新文件夹之后	否
SelectionChange	浏览器	焦点移至文件夹中另一不同项目时，或用户选择某一文件夹而 Outlook 选择该文件夹中的第一个项目时	否
ViewSwitch	浏览器	在浏览器窗口中视图更改时	否
Activate	浏览器，检查器	浏览器窗口或检查器窗口被激活时（成为活动窗口）	否
Deactivate	浏览器	浏览器窗口或检查器窗口激活时（不再是活动窗口）	否
BeforeMaximize	浏览器	用户要最大化浏览器或检查器，但在检查器最大化发生之前	是
BeforeMinimize	浏览器，检查器	用户要最小化浏览器或检查器，但在最小化发生之前	是
BeforeMove	浏览器，检查器	用户要移动浏览器窗口或检查器窗口，但在该行动发生之前	是
BeforeSize	浏览器，检查器	用户要调整浏览器或检查器的大小，但在该行动发生之前	是
NewExplorer	浏览器	打开一个新浏览器窗口时	否
NewInspector	检查器	打开一个新检查器窗口时	否
ViewAdd	视图	将一个视图添加到 Views 集合时	是
ViewRemove	视图	将一个视图从 Views 集合中删除时	是

如果是在一个较小的屏幕上工作（例如便携式电脑的屏幕），可能宁愿使用 NewInspector 事件以最大化打开每个检查器窗口，而隐藏任何不需要的工具栏。下述例子（包括声明）之中的第一个过程，使用 NewInspector 事件来确认显示标准工具栏，隐藏格式工具栏，将表示新检查器的 Inspector 对象指派给 Public 对象变量 myInspector。第二个过程使用 myInspector 对象的 Activate 事件，通过将WindowState 属性设置为 olMaximized，使该对象的窗口最大化。

这两个事件过程的实际效果是按上面所述的方式来配置工具栏以及使检查器窗口最大化。Activate 事件过程是必需的，因为 NewInspector 事件是在检查器窗口显示之前运行的，这意味着 NewInspector 事件过程不能够使检查器窗口最大化。

```

Public WithEvents myInspectors As Inspectors
Public WithEvents myInspector As Inspector

Private Sub myInspectors_NewInspector(ByVal Inspector As Outlook.Inspector)
    With Inspector
        With .CommandBars
            .Item("Standard").Visible = True
            .Item("Formatting").Visible = False
        End With
        Set myInspector = Inspector
    End With
End Sub

Private Sub myInspector_Activate()
    myInspector.WindowState = olMaximized
End Sub

```

了解应用于文件夹的事件

Outlook 提供三种应用于文件夹的事件（如表 27.3 所示）。

表 27.3 应用于文件夹的事件

事件	事件发生于	是否在 VBScript 中可用
FolderAdd	将一个文件夹添加到指定的 Folders 集合时	否
FolderChange	当更改指定的 Folders 集合中的一个文件夹时	否
FolderRemove	当从指定的 Folders 集合中删除一个文件夹时	否

了解应用于项目和结果的事件

表 27.4 列出了应用于项目和结果的事件。

表 27.4 应用于项目和结果的事件

事件	事件发生于	是否在 VBScript 中可用
ItemAdd	将一个或多个项目添加到指定的集合时，但不是很多项目突然一起添加时	否
ItemChange	更改 Items 集合或 Results 集合中的一个项目时	否
ItemRemove	从 Items 集合或 Results 集合中删除一个项目时，但不是 16 个或更多个项目一起从个人文件夹文件、交换邮箱，或交换公有文件夹中删除时，也不是删除个人文件夹文件中的最后一个项目时	否

下述例子使用 ItemChange 事件来监视 Contacts 文件夹中什么时候有联系人更改的情况发生。如有更改，该事件过程就显示一个消息框，询问用户是否要把联系人更改之事告知其同事。如果用户单击“是”按钮，本例即创建一个包含对应于联系人更改的 vCard 的邮件，并将邮件发送出去。这个邮件送往买卖合伙人分布表，并有主题行 Contact details change: 和联系人名称。

```
Public WithEvents myContacts As Items
```

```

Public Sub Initialize_myContacts
    Set myContacts = Application.GetNamespace("MAPI") _
        .GetDefaultFolder(olFolderContacts).Items

```

```

End Sub

Private Sub myContacts_ItemChange(ByVal Item As Object)
    Dim ContactUpdateMessage As MailItem

    If MsgBox("Notify your colleagues of the contact change?", _
              vbYesNo + vbQuestion, "Contact Data Changed") = vbYes Then
        Set ContactUpdateMessage = Application.CreateItem(ItemType:=olMailItem)
        With ContactUpdateMessage
            .To = "Marketing Associates"
            .Subject = "Contact details change: " & Item.Subject
            .Body = "The following contact has changed: " & vbCrLf & vbCrLf _
                    & Item.Subject
            .Attachments.Add Source:=Item, Position:=50
            .Send
        End With
    End If
End Sub

```

了解应用于 Outlook 面板的事件

Outlook 支持应用于 Outlook 面板的各种事件（如表 27.5 所示）。如果需要自定义 Outlook 面板，或者要限制用户的导航能力，这些事件是很有用的。例如，可以使用 BeforeNavigate 事件来检查用户试图导航到哪个快捷方式。如果用户未被允许访问与该快捷方式相关联的文件夹，可将此导航取消。

表 27.5 应用于 Outlook 面板对象的事件

事件	应用于	事件发生于	是否在 VBScript 中可用
GroupAdd	OutlookBarGroup	将新组添加到快捷方式窗格之后	否
BeforeGroupAdd	OutlookBarGroups	将新组添加到快捷方式窗格之前	否
BeforeGroupRemove	OutlookBarGroups	将新组从快捷方式窗格中删除之前	否
BeforeGroupSwitch	OutlookBarPane	在 Outlook 面板中打开新组时发生，但在 Outlook 2003 中不发生，因为 Outlook 有不同的导航窗格和快捷方式窗格	否
BeforeNavigate	OutlookBarPane	用户单击快捷方式窗格中某一个快捷方式之后，但在 Outlook 显示该快捷方式表示的文件夹之前	否
BeforeShortcutAdd	OutlookBarShortcuts	将新快捷方式添加到快捷方式窗格中的一个组之前	否
BeforeShortcutRemove	OutlookBarShortcuts	将一个快捷方式从快捷方式窗格中的一个组内删除之前	否
ShortcutAdd	OutlookBarShortcuts	将新快捷方式添加到一个快捷方式窗格组之后	否

了解应用于提醒的事件

表 27.6 说明 Outlook 提供的应用于提醒的事件。当提醒发生时，或者“提醒”对话框

出现之前，或者用户单击“暂停”按钮以消除提醒时，或者添加、更改或删除提醒时，都可以使用这些事件来采取相应的行动。

表 27.6 应用于提醒的事件

事件	事件发生于	是否在 VBScript 中可用
BeforeReminderShow	Outlook 显示“提醒”对话框之前	是
ReminderAdd	添加一个提醒时	是
ReminderChange	更改过某一个提醒之后	是
ReminderFire	执行一个提醒之前	是
ReminderRemove	从 Reminders 集合中删除一个提醒时	是
Snooze	用户单击“暂停”按钮以消除某一个提醒时	是

了解应用于同步处理的事件

如果要针对 Outlook 中的同步处理编写过程，可能需要使用下述三种事件，它们应用于表示某一个用户的发送/接收组的 SyncObject 对象。（使用 NameSpace 对象的 SyncObjects 属性访问 SyncObject 对象，以返回 SyncObjects 集合。）表 27.7 说明应用于 SyncObject 对象的事件。

表 27.7 应用于 SyncObject 对象的事件

事件	事件发生于	是否在 VBScript 中可用
SyncStart	Outlook 开始同步处理用户的文件夹时	否
SyncEnd	同步处理完成之后	否
OnError	同步处理期间有错误产生时	否

下述例子使用与对象变量 mySyncObject 相对应的 OnError 事件。如果在 mySyncObject 表示的 SyncObject 的同步处理期间有错误产生时，此过程会显示出错消息，给出错误代码和说明：

```
Private Sub mySyncObject_OnError(ByVal Code As Long, _
    ByVal Description As String)

    Dim strMessage As String
    strMessage = "An error occurred during synchronization:" & vbCrLf & vbCrLf
    strMessage = strMessage & "Error code: " & Code & vbCrLf
    strMessage = strMessage & "Error description: " & Description
    MsgBox strMessage, vbOKOnly + vbExclamation, "Synchronization Error"

End Sub
```

本章将简要地概述如何使用 Access 对象模型。首先将介绍 Access 对象模型的组成，然后将通过一个示例来说明如何使用 Access 对象模型。

第 28 章 了解 Access 对象模型和重要对象

屏幕上显示了 Access 对象模型的层次结构。

子对象

对象

- ◆ 开始使用 Access 中的 VBA
- ◆ 纵览 Access 对象模型
- ◆ 了解 Access 中的可创建对象
- ◆ 打开和关闭数据库
- ◆ 应用 Screen 对象
- ◆ 为数据库设定启动选项
- ◆ 用 DoCmd 对象执行命令

在开发 Access 数据库、窗体或报表时，经常需要利用 VBA 定制 Access 来优化自己和同事的工作。根据使用 Access 的目的，可以通过编程自动获得需要的数据或者定期生成自定义报表。

即使利用 Access 仅仅是为了输入数据和检查数据的准确性，仍然可以编写 VBA 程序，使烦琐的工作简单化。例如，可以利用 VBA 简化数据输入的过程或处理用户输入的数据，以避免在之后的工作中可能出现的问题。

本章首先将介绍如何开始使用 Access 中的 VBA，因为 Access 以与本书中讨论的其他软件不同的方法集成 VBA。然后介绍 Access 对象模型并重点讲解一些重要的可创建对象。之后，将介绍如何打开和关闭数据库，为数据库设定启动选项，应用 Screen 对象，以及用 DoCmd 对象执行命令。

下一章将讨论使用 VBA 处理 Access 数据库中的数据。

开始使用 Access 中的 VBA

Access 集成 VBA 的方法与大部分的 VBA 宿主软件有所不同，以下是主要的不同点：

- ◆ Access 中的集合以 0 为起点——集合中第一项的编号为 0，而不是 1。例如，Forms(0).Name 返回 Forms 集合中第一个 Form 对象的 Name 属性。
- ◆ Access 中的宏与其他 Office 软件中的宏有所不同。宏是指一系列需要执行的动作。Access 把宏单独存放，而其他软件把它们和 VBA 放在一起。在 Access 中通过选择数据库窗口中的宏对象创建宏，而不需要进入 Visual Basic 编码器。
- ◆ 可以从 Access 用户界面直接调用模块，而不需要先调用 Visual Basic 编码器。当然也可以选择通过 Visual Basic 编码器调用模块，就像本书讨论的其他软件中所做的那样。
- ◆ 要在 Access 中使用 VBA 完成大多数任务，应编写函数（函数过程），而不是编写像其他 VBA 宿主软件（如 Word 或 Excel）那样的子过程。
- ◆ 要执行函数，可以使用 RunCode 动作调用该函数的宏。

◆ 要执行子过程，需要先编写一个调用该子过程的函数。在 Visual Basic 编码器中，可以使用 Visual Basic 编码器中的常用命令（比如，按 F5 键运行子过程）来调试和运行子过程，然而在 Access 中，不能从 Access 用户界面直接运行子过程。下面给出了一个直接的例子，示范如何使用 Access 中的 VBA。

创建模块

要创建一个模块，需要打开数据库并执行以下步骤：

1. 打开数据库窗口，例如，打开“窗口”菜单，然后单击该数据库窗口的条目。
2. 在“对象”列上单击“模块”按钮，显示模块的列表（如果已有模块），如图 28.1 所示。
3. 单击“新建”按钮。Access 激活 Visual Basic 编码器并添加一个新模块，名为“模块 1”（或者下一个未使用的名字——例如，模块 2）。Access 自动输入 Option Compare Database 语句。可在本章较后部分的“了解 Option Compare Database 语句”一节查阅该语句的解释，以及其他可以实现比较的方法。

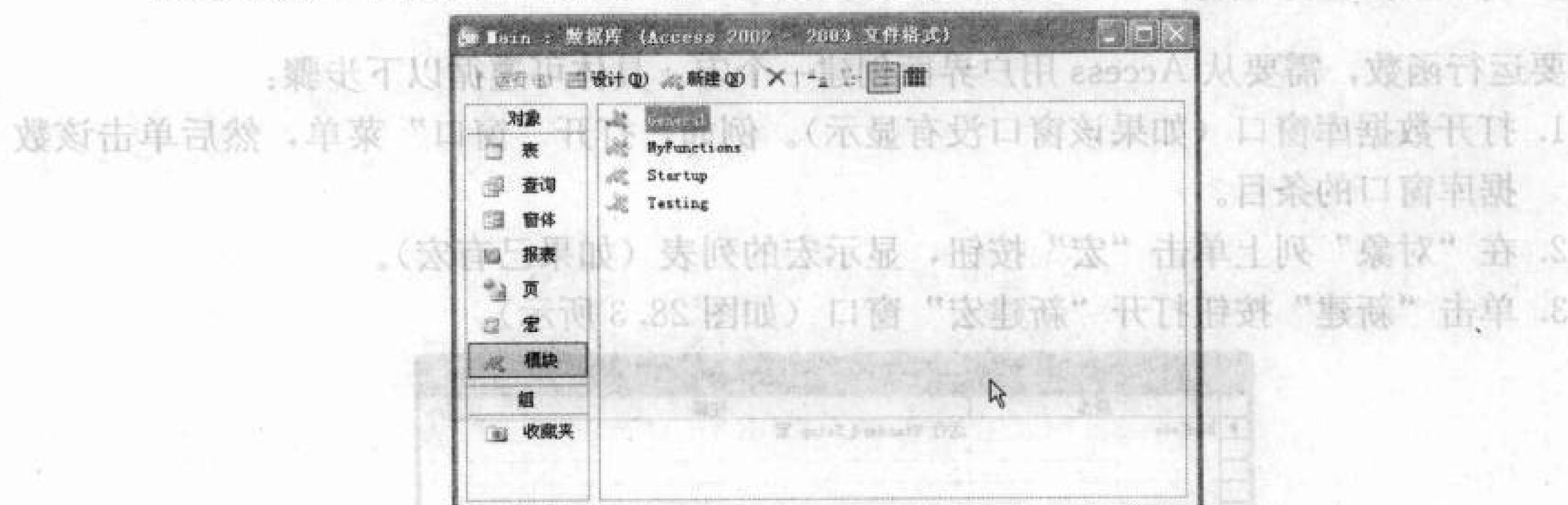


图 28.1 在 Access 的数据库窗口中为宏创建一个模块

注意：也可以通过其他方法新建模块。可以在 VBA 中单击“插入”>“模块”命令，或者在“工程资源管理器”窗口中右击工程条目，在弹出的快捷菜单中选择“插入”>“模块”。

4. 按 Ctrl + S 键或选择“文件”>保存（“保存”命令的条目中也显示了数据库的名称），Access 显示“另存为”对话框（如图 28.2 所示）。

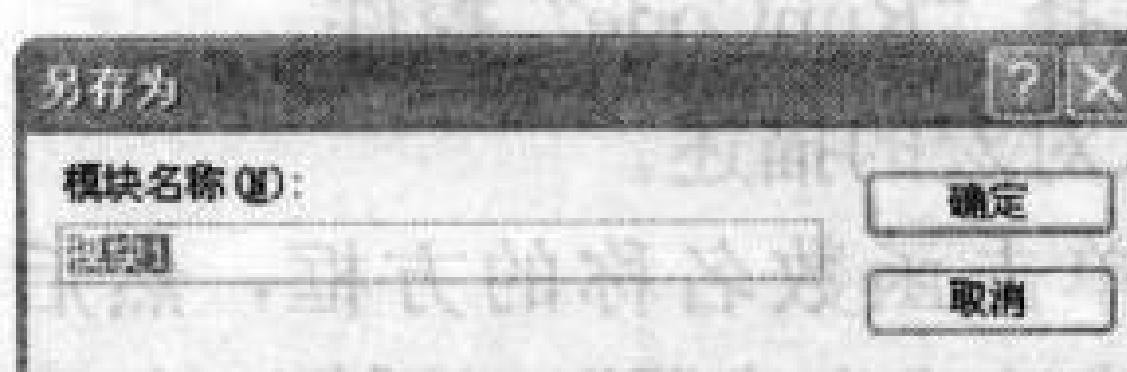


图 28.2 在“另存为”对话框中键入新模块的名称

5. 键入模块的名称，然后按 Enter 键或者单击“确定”按钮应用该名称。Access 保存了模块。

注意：也可以按 F4 键激活“属性”窗口，键入新模块的名称，然后回车。这时在显示“另存为”对话框时，输入框中已经填上了新的名称，不需要再做更改。

创建函数

创建完模块之后，就可以在其中创建函数，就像本书前面的章节所提到的那样。下面的例子创建了一个名为 Standard_Setup 的函数，让计算机在函数运行时弹出信息框，说明函数开始执行。下一节会使用该宏作为实例。

```
Public Function Standard_Setup()
    'put your choice of commands here
    MsgBox "The Standard_Setup macro is running."
End Function
```

创建完函数后，按 Alt + F11 键或者单击 Visual Basic 编辑器的“标准”工具栏上的“视图 Microsoft Access”按钮返回到 Access。（也可以单击 Access 窗口或者它的任务栏返回到 Access。）

创建一个宏来运行函数

要运行函数，需要从 Access 用户界面创建一个宏。具体可遵循以下步骤：

1. 打开数据库窗口（如果该窗口没有显示）。例如，打开“窗口”菜单，然后单击该数据库窗口的条目。
2. 在“对象”列上单击“宏”按钮，显示宏的列表（如果已有宏）。
3. 单击“新建”按钮打开“新建宏”窗口（如图 28.3 所示）。

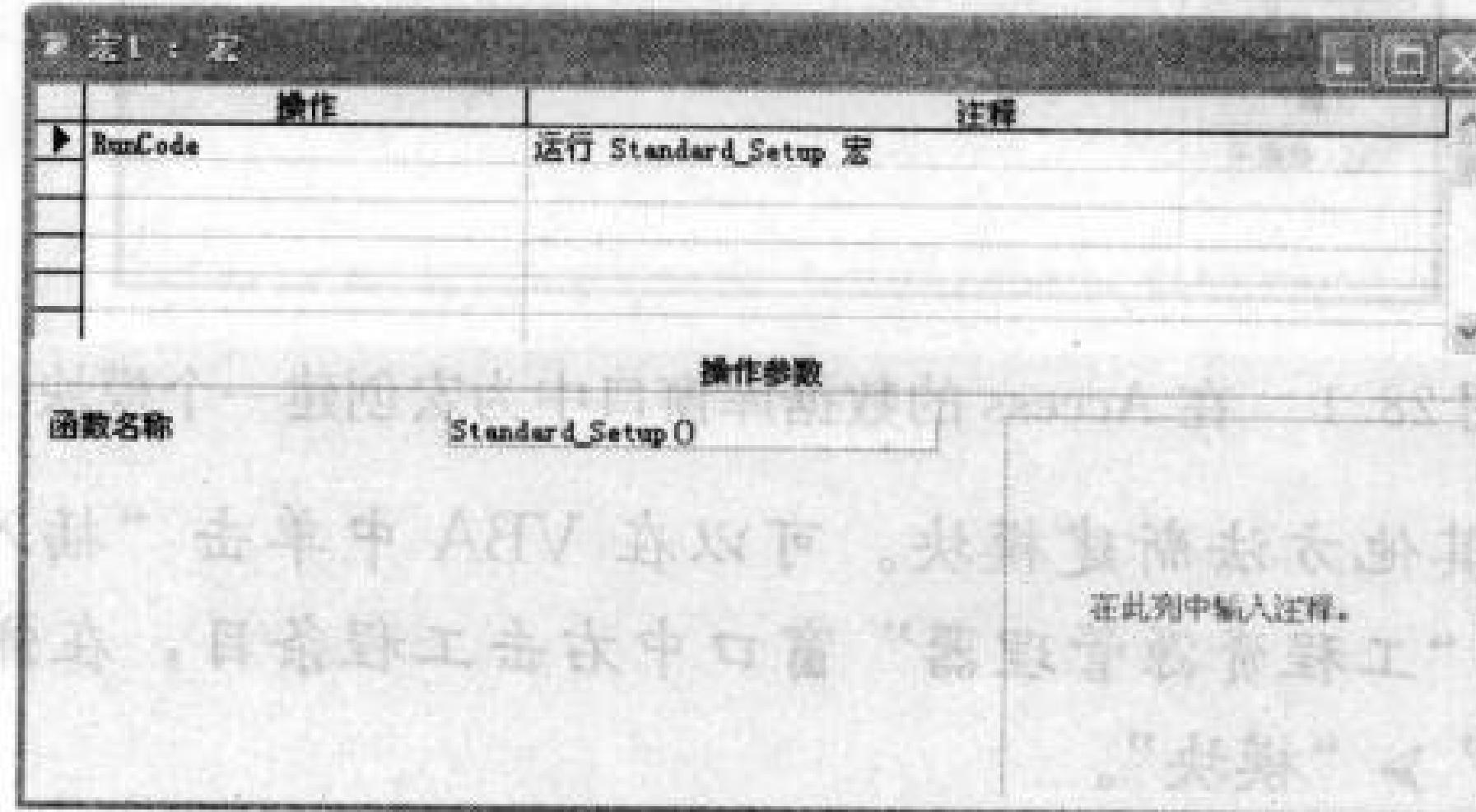


图 28.3 使用“宏”窗口在 Access 中创建宏

4. 在“操作”下拉列表中选择“RunCode”控件。
5. 在“注释”文本框中键入对宏的描述。
6. 在“操作参数”区域，单击函数名称的方框，然后单击右边有省略号（三个点“...”）的按钮，显示“表达式生成器”对话框（如图 28.4 所示）。
7. 双击函数条目使其展开，然后选择数据库（本例中名为 Main）的条目、模块，然后是函数。单击“粘贴”按钮将函数的名称粘贴到“表达式生成器”对话框顶部的文本框中。
8. 单击“确定”按钮关闭“表达式生成器”对话框。Access 在“函数名称”文本框中输入了函数的名称。
9. 按 Ctrl + S 键或选择“文件”>“保存”显示“另存为”对话框。
10. 键入宏的名称，然后回车或单击“确定”按钮应用该名称。
11. 单击“关闭”按钮关闭宏窗口。

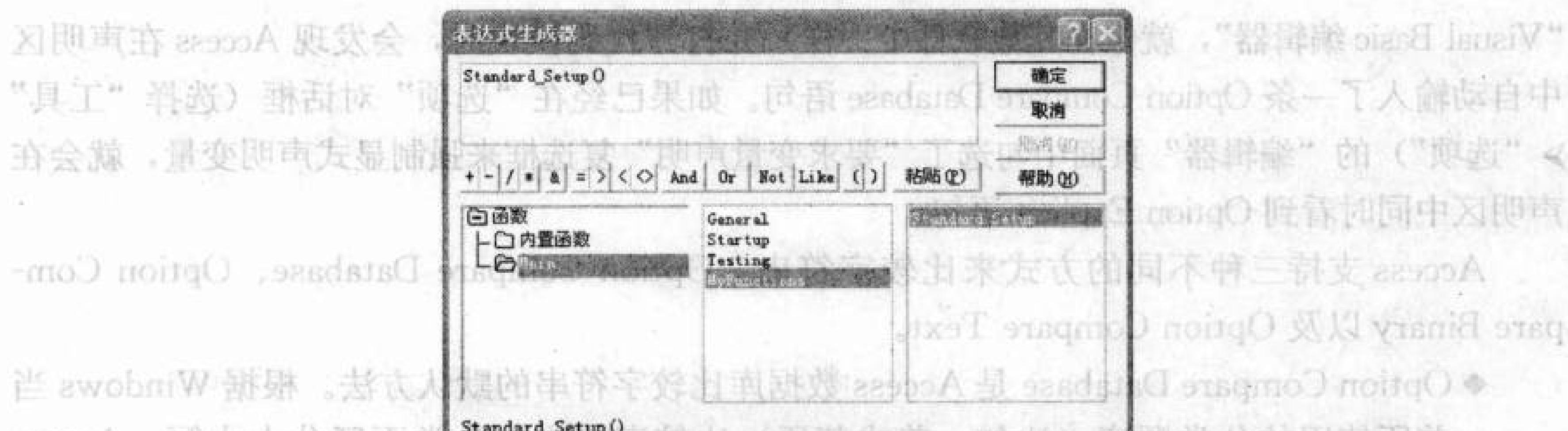


图 28.4 使用“表达式生成器”对话框指定需要在宏中运行的函数

12. 如果想要测试一下宏，可以双击数据库窗口的“宏”列表中的相应条目。

使用 AutoExec 宏设置 Access 的启动项

要设置 Access 的启动项，可以使用 AutoExec 宏。AutoExec 是一个专有名称，用于那些在 Access 打开时自动运行的宏。例如，可以最大化软件窗口，打开一个特定的控件（比如，一个表格）或者显示某一条特定的记录。

提示：如果不想在打开数据库时运行 AutoExec 宏，可以在打开时按住 Shift 键。

要创建 AutoExec 宏，可以像前面所说的那样，新建一个宏，为宏添加想要运行的指令，然后用 AutoExec 名称保存。这样在下次打开数据库时就会运行这个宏。

注意：在 Access 中也可以用宏实现很多操作。例如，可以创建一个宏来打开一个窗体并执行一系列的操作，或者可以创建一个宏来打印报表。

运行子过程

在 Access 中，没有必要在子过程中编写 VBA 代码，只要编写函数即可。如果在子过程中编写了代码，想要运行该子过程时，仍然需要再编写一个程序调用该子过程。这样做显得很多余，不过程序仍然可以运行。下面是一个样本实例。

1. 在 Visual Basic 编码器中，编写一段子过程来完成想要执行的操作：

```
Sub SampleProcedure()
    MsgBox "The subprocedure named Sample Procedure is running."
End Sub
```

2. 仍然是在 Visual Basic 编码器中，创建一个函数来调用这个子过程：

```
Public Function Run_SampleProcedure()
    Call SampleProcedure
End Function
```

3. 回到 Access，创建一个宏，使用 RunCode 操作运行上面的函数，调用子过程。（子过程的具体内容见上文。）

了解 Option Compare Database 语句

当在 Access 中打开 Visual Basic 编码器（通过按 Alt + F11 键或者选择“工具”>“宏”>

“Visual Basic 编辑器”，就像在其他软件中一样）并打开代码模块时，会发现 Access 在声明区中自动输入了一条 Option Compare Database 语句。如果已经在“选项”对话框（选择“工具”>“选项”的“编辑器”页面中勾选了“要求变量声明”复选框来强制显式声明变量，就会在声明区中同时看到 Option Explicit 语句。

Access 支持三种不同的方式来比较字符串：Option Compare Database、Option Compare Binary 以及 Option Compare Text。

- ◆ Option Compare Database 是 Access 数据库比较字符串的默认方法。根据 Windows 当前所使用的分类顺序（比如，美式英语）比较字符串。分类不区分大小写。Access 会在每一个模块的声明部分自动添加这条语句。可以删除 Option Compare Database 语句，这样 Access 将会采用 Option Compare Binary 方式比较字符串。
- ◆ Option Compare Binary 是区分大小写的分类方式。要使用 Option Compare Binary 方式，可以删除声明部分的 Option Compare Database 语句，或将其替换为 Option Compare Binary 语句。
- ◆ Option Compare Text 是区分大小写的分类方式。要使用 Option Compare Text 方式，可将 Option Compare Database 语句或者 Option Compare Binary 语句替换为 Option Compare Text 语句。

纵览 Access 对象模型

要浏览 Access 对象模型，可以打开“Microsoft Visual Basic 帮助”窗口，调用“Microsoft Access 对象模型”主题。如果还没有打开 Visual Basic 编辑器，那么首先需要打开 Visual Basic 编辑器，然后选择“帮助”>“Microsoft Visual Basic 帮助”，键入“Access 对象模型”并回车，然后单击“Microsoft Access 对象模型”主题。图 28.5 显示了“Microsoft Access 对象模型”主题下的主表格。

了解 Access 的可创建对象

可以通过 Application 对象访问 Access 软件中的所有对象。但也可以直接通过 Access 公布的可创建对象完成许多操作。Access 中主要的可创建对象有：

- ◆ Forms 集合包括所有的 Form 对象，代表数据库中当前打开的表格。
- ◆ Reports 集合包括了所有的 Report 对象，代表数据库中所有当前打开的报表。
- ◆ DataAccessPages 集合包括了所有 DataAccessPage 对象，代表工程或数据库中所有打开的数据访问页。

注意：Access 工程是指一个与 SQL Server 数据库相关联的文件。

- ◆ CurrentProject 对象代表 Access 中的当前工程或数据库。
- ◆ CurrentData 对象代表当前数据库中储存的对象。
- ◆ CodeProject 对象代表包含代码数据库的项目。代码数据库既可以属于某个 Access 工程，也可以属于一个数据库。
- ◆ CodeData 对象代表储存在代码数据库中的对象。

- ◆ Screen 对象代表当前拥有焦点的屏幕对象（接受输入或者准备接受输入的对象），该对象可以是窗体、报表或控件。
- ◆ 通过 DoCmd 对象可在 VBA 中运行 Access 命令。
- ◆ Modules 集合包含 Module 对象，代表 Microsoft Access 数据库中所有打开的标准模块和类模块。
- ◆ References 集合包含 Reference 对象，代表 Access 中当前所设的引用。

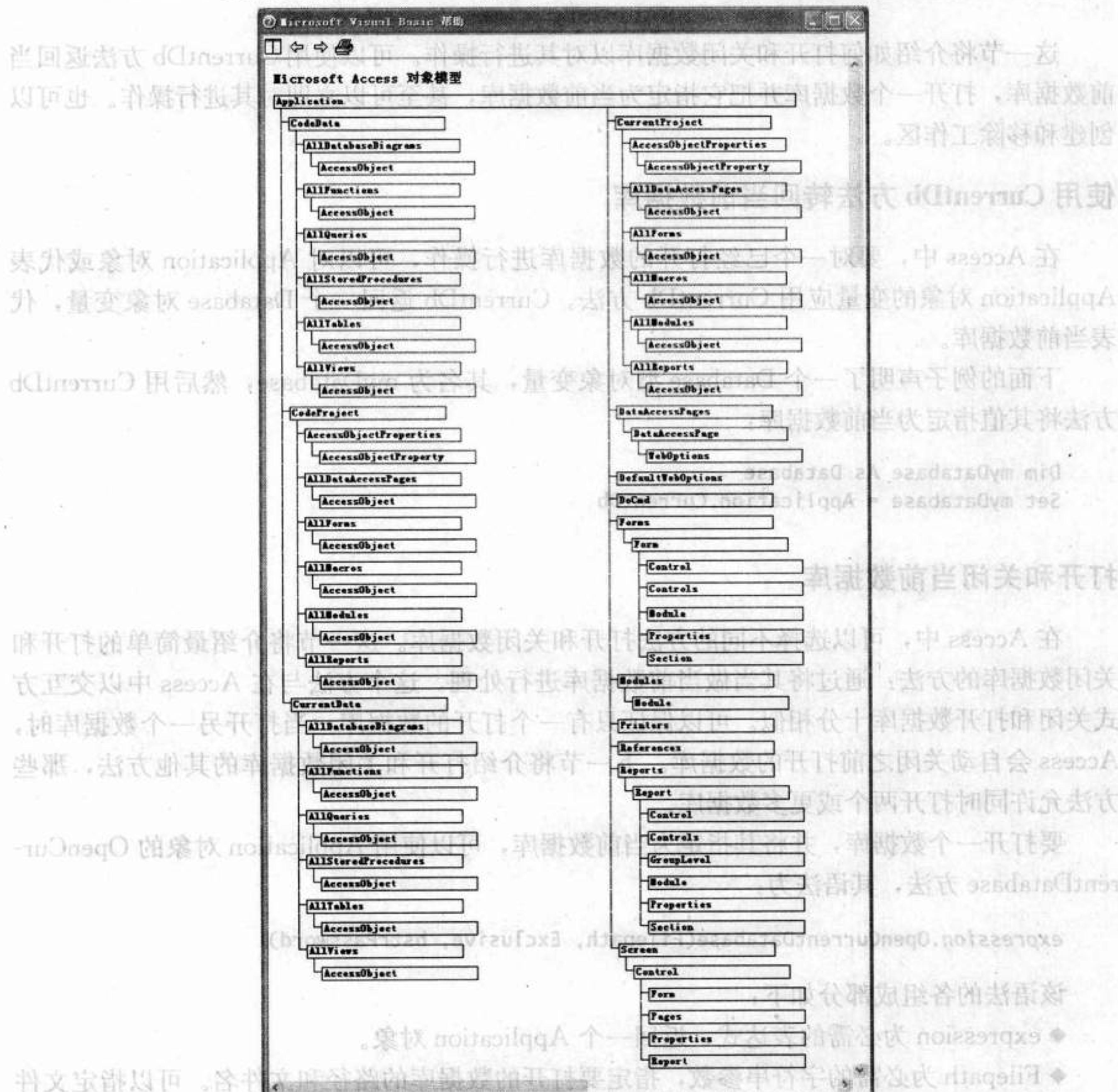


图 28.5 在 VBA 的“Microsoft Access 对象模型”页面中，可以浏览 Access 对象模型。页面中还包含一些没有显示在本图中的其他内容

- ◆ DBEngine 对象代表 Microsoft Jet 数据库引擎是 Data Access 对象 (DAO) 层次的最高一级的对象。DBEngine 对象提供了连接到 Workspaces 集合、Errors 集合的途径。Workspaces 集合包括了所有 Access 中可以应用的 Workspace 对象，Errors 集合包括了所有与 DAO 操作相关的 Error 对象。

- ◆ Workspace 对象为用户提供了一个命名的流程。当打开一个数据库时，Access 会以默认方式创建一个工作区，并将其指定为当前数据库。可以在当前工作区完成工作或者根据需要创建工作区。
- ◆ Error 对象包括了与 DAO 操作中发生的错误有关的信息。

打开和关闭数据库

这一节将介绍如何打开和关闭数据库以对其进行操作。可以使用 CurrentDb 方法返回当前数据库，打开一个数据库并把它指定为当前数据库，甚至可以立即对其进行操作。也可以创建和移除工作区。

使用 CurrentDb 方法转回当前数据库

在 Access 中，要对一个已经打开的数据库进行操作，可以对 Application 对象或代表 Application 对象的变量应用 CurrentDb 方法。CurrentDb 返回一个 Database 对象变量，代表当前数据库。

下面的例子声明了一个 Database 型对象变量，其名为 myDatabase，然后用 CurrentDb 方法将其值指定为当前数据库：

```
Dim myDatabase As Database  
Set myDatabase = Application.CurrentDb
```

打开和关闭当前数据库

在 Access 中，可以选择不同的方法打开和关闭数据库。这一节将介绍最简单的打开和关闭数据库的方法：通过将其当做当前数据库进行处理。这个方法与在 Access 中以交互方式关闭和打开数据库十分相似。可以保证只有一个打开的数据库，当打开另一个数据库时，Access 会自动关闭之前打开的数据库。下一节将介绍打开和关闭数据库的其他方法，那些方法允许同时打开两个或更多数据库。

要打开一个数据库，并将其指定为当前数据库，可以使用 Application 对象的 OpenCurrentDatabase 方法，其语法为：

expression.OpenCurrentDatabase(*Filepath*, *Exclusive*, *bstrPassword*)

该语法的各组成部分如下：

- ◆ *expression* 为必需的表达式，返回一个 Application 对象。
- ◆ *Filepath* 为必需的字符串参数，指定要打开的数据库的路径和文件名。可以指定文件的扩展名，如果不指定，Access 将扩展名默认为 .mdb。
- ◆ *Exclusive* 是可选的逻辑参数，将其设为 True，则以独占的模式打开数据库而不是以共享的模式打开（共享模式为默认模式，显式声明 Exclusive 值为 False 也可以设定打开模式为共享）。
- ◆ *bstrPassword* 是可选的字符串参数，指定打开数据库的密码。

要关闭当前数据库，可以对 Application 对象应用 CloseCurrentDatabase 方法。该方法没有相关的参数。

可以从当前的数据库中使用 CloseCurrentDatabase 方法，但在此之后就不能执行其他操作了，因为在 VBA 执行 CloseCurrentDatabase 方法后，当前数据库就关闭了。要关闭当前数据库，并用 OpenCurrentDatabase 方法打开另一个数据库，必须在数据库之外运行代码——例如，通过自动方式从其他应用程序打开数据库。

下面的实例从另一个 VBA 宿主软件（比如从 Excel 或 Word）运行了上面提到的两个方法。例子中声明了对象变量 myAccess 为 Access. Application 类型，声明对象变量 myDatabase 为 Object 类型。例子中使用 GetObject 方法将 myAccess 指定为正在运行的 Access 的副本。使用 CloseCurrentDatabase 方法关闭打开的数据库，然后用 OpenCurrentDatabase 方法以独占模式打开另一个数据库。最后一条语句使用 CurrentDb 方法指定 myDatabase 对象变量值为当前数据库。

```
Dim myAccess As Access.Application  
Dim myDatabase As Object  
  
Set myAccess = GetObject(, "Access.Application")  
myAccess.CloseCurrentDatabase  
myAccess.OpenCurrentDatabase _  
    filePath:="Z:\Database\Current\Region1.mdb", Exclusive:=True  
Set myDatabase = myAccess.CurrentDb
```

在 Access 中创建新数据库、窗体和报表

在介绍其他软件的章节里，主要侧重于创建和存储新文件——例如，在 Word 中创建新文档或在 Excel 中创建新工作簿；并将其保存为适当的文件名和文件格式。

Access 包含了创建新数据库、窗体、报表以及其他对象的 VBA 程序。例如：

- ◆ 要创建一个新数据库，可使用 Application 对象的 NewCurrentDatabase 方法。
- ◆ 要创建一个新的窗体，可使用 CreateForm 方法。要对窗体进行操作，可使用 CreateControl 方法。
- ◆ 要创建一个新的报表，可使用 CreateReport 方法。要对报表进行操作，可使用 CreateReportControl 方法。

尽管用程序来创建一个新的数据库是可行的，但这样做不仅烦琐，而且并不常用。在大多数的情况下，在 Access 中应用 VBA 编写程序是为了操控那些数据库和对象，而创建数据库和对象的操作则大部分是手动完成的。

同时打开多个数据库

除了可以使用 OpenCurrentDatabase 方法打开数据库并将其指定为当前数据库以外，还可以使用 Workspace 对象的 OpenDatabase 方法来打开另一个数据库，并返回一个代表 Database 对象的地址。OpenDatabase 方法的语法如下：

```
Set database = workspace.OpenDatabase (Name, Options, ReadOnly, Connect)
```

该语法的各组成部分如下：

- ◆ database 是一个对象变量，代表所打开的数据库。

- ◆ workspace 是可选的对象变量，指定工作区，用于在其中打开数据库。如果不指定，Access 将会在默认工作区中打开数据库。尽管在默认工作区中打开数据库没有任何问题，但是为数据库单独创建一个工作区的做法可以使工作更加顺利。本章后面的“创建和移除工作区”一节将详细介绍相关的内容。
- ◆ Name 是必选的字符串参数，指定了要打开的数据库的名称。如果该数据库不存在或不可用，或者另一个用户已经以独占模式打开了该数据库，将发生错误。
- ◆ Options 是可选的不定型参数，指定了需要为数据库设定的选项。对于 Access 数据库，可以将其值设为 True 来以独占模式打开数据库，或设定为 False（默认值）以共享模式打开。对于 ODBC Direct 工作区，可以使用其他的选项；可以查阅“Access Visual Basic 帮助”文件获取详细信息。
- ◆ ReadOnly 是可选的不定型参数，可以将其值设定为 True 以便以只读模式打开数据库。默认值为 False，即以读/写模式打开。
- ◆ Connect 是可选的不定型参数，可以使用它来传递必要的连接信息，例如打开数据库的密码。

下面的例子声明了一个 Workspace 对象变量（其名为 myWorkspace）以及 Database 对象变量（其名为 myDatabase）。变量 myWorkspace 被指定为 Workspaces 集合中的第一个 Workspace 对象（默认工作区），变量 myDatabase 被指定为数据库 Testing.mdb 以独占和读/写模式打开：

```
Dim myWorkspace As Workspace
Dim myDatabase As Database

Set myWorkspace = DBEngine.Workspaces(0)
Set myDatabase = myWorkspace.OpenDatabase _
    (Name:="\\server\database\Testing.mdb", _
     Options:=True, ReadOnly:=False)
```

关闭数据库

要关闭之前通过 OpenDatabase 方法打开的数据库，可以对代表该数据库的对象变量应用 Close 方法。例如，下面的语句关闭了 myDatabase 对象变量所代表的数据库：

```
myDatabase.Close
```

创建和移除工作区

要对不同的数据库分开进行操作，可以为它们分别创建工作区，并在操作完之后关闭相应的工作区。

创建新的工作区

要创建一个新的工作区，可以使用 DBEngine 对象的 CreateWorkspace 方法，其语法如下：

```
Set workspace = CreateWorkspace(Name, UserName, Password, UseType)
```

该语法的各组成部分如下：

- ◆ workspace 是代表新建的工作区的对象变量。

- ◆ Name 是必选的字符串参数，指定新工作区的名称。
- ◆ UserName 是必选的字符串参数，指定新工作区的所有者。
- ◆ Password 是必选的字符串参数，指定新工作区的密码。密码最长为 14 个字符。如果不设密码，就使用空字符串。
- ◆ UseType 是可选参数，表示将要创建的工作区的类型。可以使用 dbUseJet 创建 Microsoft Jet 工作区，使用 dbUseODBC 创建 ODBC Direct 工作区。省略此参数，DBEngine 对象的 DefaultType 属性将自行决定与工作区相连的数据源类型。

下面的例子声明了一个名为 myWorkspace 的 Workspace 类型的对象变量，并将其值指定为新建的名为 Workspace2 的 Jet 工作区。本例中将 admin 账户指定为新工作区的所有者。

```
Dim myWorkspace As Workspace
Set myWorkspace = CreateWorkspace(Name:="Workspace2",
    UserName:="admin", Password:="", UseType:=dbUseJet)
```

创建新工作区之后，可以用它来打开一个新的数据库。参见前面提到的例子。

移除工作区

在从 Workspace 集合中移除一个工作区之前，必须关闭所有的连接和数据库。可以使用 Close 方法关闭 Workspace 对象。例如，下面的语句关闭了对象变量 myWorkspace 所代表的 Workspace 对象：

```
myWorkspace.Close
```

为数据库设定启动选项

可以使用 Access 的启动选项来控制打开数据库的方式和显示的用户界面。例如，可以打开数据库中的某个特定窗体，这样用户就可以在打开数据库之后立刻对其进行操作。

当通过 VBA 设定启动选项时，必须小心不能出错。如果数据库是新建的，那么在用户修改默认的启动设定（例如，可以选择“工具”>“启动”，然后使用“启动”对话框中的选项修改启动设定，见图 28.6）之前，数据库中并不包含启动选项，因为这一选项可能并不存在。必须编写代码来处理由于使用了不存在的选项而产生的错误，并在必要时创建该选项。

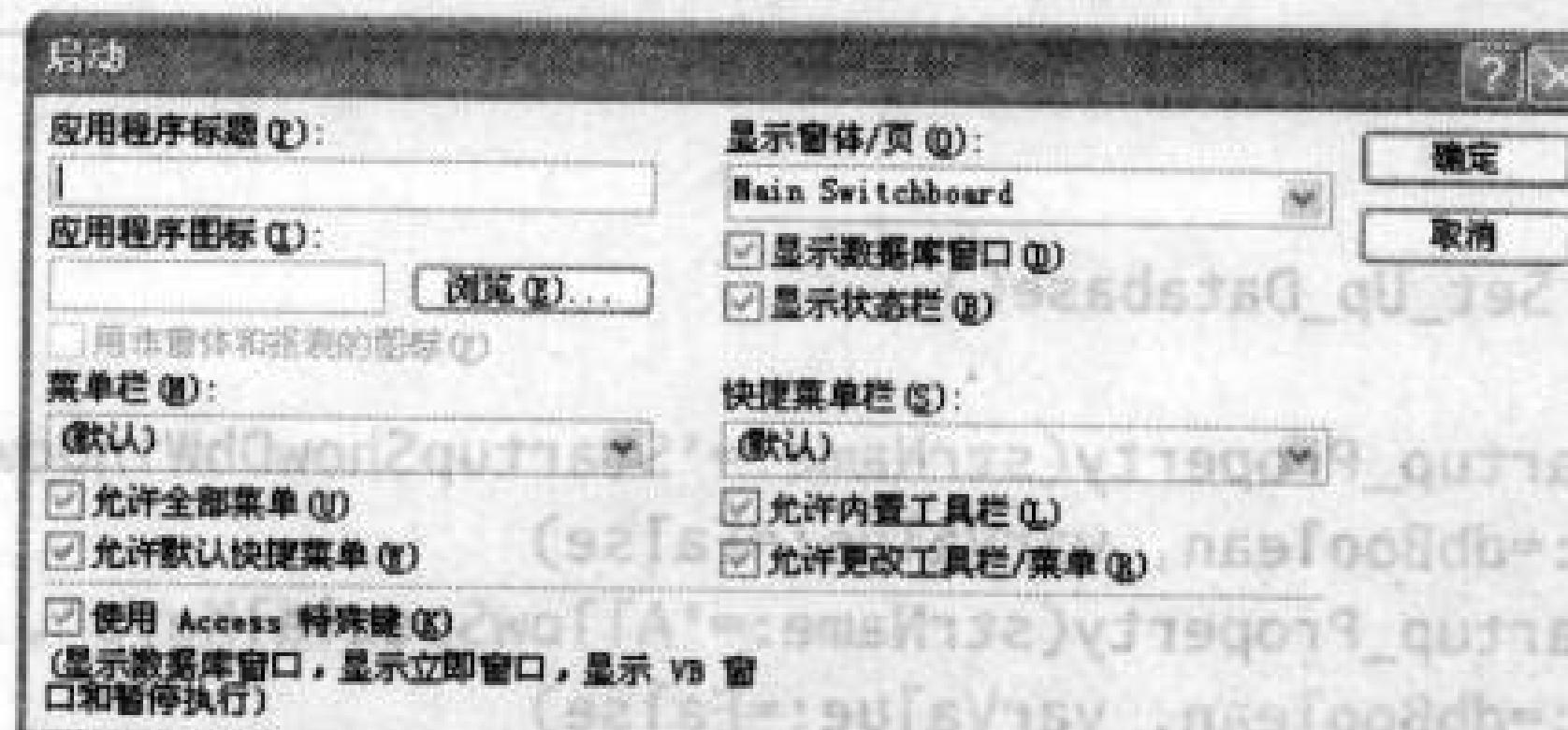


图 28.6 可以通过“启动”对话框为 Access 数据库或工程人工设定启动选项。当通过 VBA 处理启动选项时，必须确保该选项存在，并避免错误

要通过 VBA 设定启动选项，可使用表 28.1 中列出的选项，在表中同时也列出了各选项在“启动”对话框中对应的条目。

表 28.1 Access 数据库的启动选项

选项名称	在“启动”对话框中所对应的条目	控制的操作
AllowBreakIntoCode	允许进入代码	决定用户是否能在发生运行错误之后在 Visual Basic 编辑器中浏览代码
AllowBuiltInToolbars	允许内置工具栏	决定是否显示 Access 内置工具栏
AllowFullMenus	允许全部菜单	决定是否显示 Access 全部的内置菜单，显示则设定为 True 或 -1，不显示则设定为 False 或 0；使用 False 可以显示自定义的缩减内置菜单（例如，禁止用户进行某种特定的操作）
AllowShortcutMenus	允许默认快捷菜单	决定是否允许用户使用快捷菜单
AllowSpecialKeys	使用 Access 特殊键	决定是否允许用户使用特殊键
AllowToolbarChanges	允许更改工具栏/菜单	决定是否允许用户修改工具栏或菜单的项目
AppIcon	应用程序图标	该图标可取代默认的 Access 图标，可使用位图文件 (.bmp) 或图标文件 (.ico)
AppTitle	应用程序标题	该标题将显示在软件的标题栏中
StartupForm	显示窗体/页	决定数据库打开时显示哪一个窗口
StartupMenuBar	菜单栏	决定用哪一个自定义窗口（如果有）代替常规的 Access 菜单栏
StartupShortcutMenuBar	快捷菜单栏	决定用哪一个自定义快捷菜单栏（如果有）代替常规的 Access 快捷菜单
StartupShowDBWindow	显示数据库窗口	决定在打开数据库时，是否显示数据库窗口，要显示则设定为 True 或 -1，不显示则设定为 False 或 0
StartupShowStatusBar	显示状态栏	决定是否在打开数据库时显示状态栏。显示则设定为 True 或 -1，不显示则设定为 False 或 0

注意：启动选项在 Access 工程 (.adp 文件) 和 Access 数据库 (.mdb 文件) 中的执行方式不同。在工程中，启动选项通过 CurrentProject 对象的 Access-ObjectProperties 集合实现，而在数据库中则需通过 CurrentProject 对象的 Properties 集合实现。

程序清单 28.1 为当前数据库设定了启动选项，这里通过使用 CurrentDb 对象实现。

程序清单 28.1

```

1. Public Function Set_Up_Database()
2.
3.     Call Set_Startup_Property(strName:="StartupShowDbWindow", _
        varType:=dbBoolean, varValue:=False)
4.     Call Set_Startup_Property(strName:="AllowSpecialKeys", _
        varType:=dbBoolean, varValue:=False)
5.     Call Set_Startup_Property(strName:="StartupShowStatusBar", _
        varType:=dbBoolean, varValue:=False)
6.
7. End Function
8.
```

```

9.
10. Function Set_Startup_Property(strName As String, varType As Variant, _
      varValue As Variant)
11.
12.     Dim myProperty As Property
13.     Const conPropertyNotFound = 3270
14.
15.     On Error GoTo ErrorHandler
16.     CurrentDb.Properties(Item:=strName) = varValue
17.     Exit Function
18. ErrorHandler:
19.     If Err = conPropertyNotFound Then
20.         Set myProperty = CurrentDb.CreateProperty _
          (Name:=strName, Type:=varType, Value:=varValue)
21.         CurrentDb.Properties.Append myProperty
22.     Else
23.         MsgBox "An error occurred: " & vbCrLf & vbCrLf & _
          "Error number: " & Err.Number & vbCrLf & _
          "Error description: " & Err.Description, vbOKOnly + _
          vbCritical, "Set_Up_Database Function"
24.     End If
25. End Function

```

程序清单 28.1 中列出的两个函数以如下方式运行：

- ◆ 第 1 行到第 7 行包含了一个 Public 函数，其名为 Set _ Up _ Database，用于启动数据库。第 10 行到第 25 行为 Set _ Startup _ Property 函数，执行启动选项的实际设定，并在选项不存在时创建它们。
- ◆ 第 1 行声明了 Set _ Up _ Database 函数，第 7 行结束该函数。其中，第 2 行和第 6 行为空行，第 3、4 和 5 行分别调用 Set _ Startup _ Property 函数，传递不同的字符串：第 3 行传递 StartupShowDbWindow，第 4 行传递 AllowSpecialKeys，第 5 行传递 StartupShowStatusBar。这 3 个字符串分别代表了 3 个需要设定的选项名称，且都被指定为 dbBoolean 类型（一个数据库的逻辑型），并指定它们的值为 False。
- ◆ 第 10 行声明函数 Set _ Startup _ Property，并声明其参数为字符型参数 strName、不定型参数 varType 以及不定型参数 varValue。第 11 行为空行。
- ◆ 第 12 行声明变量 myProperty 为 Property 类型。第 13 行声明常量 conPropertyNotFound，并指定其值为数字 3270。这是所设定的选项不存在时 VBA 返回的错误编号。第 14 行为空行。
- ◆ 第 15 行使用 On Error GoTo 语句，以便在发生错误时将程序引至第 18 行的 ErrorHandler 标签处。
- ◆ 第 16 行尝试设定选项，该选项名由 strName 传递，方式为 varValue。如果选项存在，该语句将顺利执行。第 17 行使用 Exit Function 语句退出函数。如果选项不存在，程序将出现错误，并转到纠错程序部分（第 18 行的标签处）。
- ◆ 第 18 行开始纠错程序，其中包含了一个 If …Then …Else 语句块。第 19 行检查错误是否为 conPropertyNotFound，即选项不存在。如果是，第 20 行创建选项并将其指定给 myProperty 对象变量。如果不是，程序将转到第 22 行的 Else 语句，第 23 行显示信息框，其中给出错误的详细信息。第 24 行结束 If …Then …Else 语句块，第 25

行结束函数。

应用 Screen 对象

如果经常在其他的 Office 软件中使用 VBA，那么一定编写过程序来操控当前对象。例如，在 Word 中，会使用 ActiveDocument 对象操控当前文档，或使用 Selection 对象来操控当前选区；在 PowerPoint 中，会使用 ActivePresentation 对象来操控当前演示文稿。

在 Access 中，可以使用 Screen 对象来控制当前窗体、报表或控件。Screen 对象有多种属性，其中包括以下几点：

- ◆ ActiveForm 属性返回当前窗体，如果没有当前窗体，则在使用 ActiveForm 属性时，返回错误代码 2475。
- ◆ ActiveDatasheet 属性返回当前数据表。如果没有当前数据表，则在使用 ActiveDatasheet 属性时返回错误代码 2484。
- ◆ ActiveReport 属性返回当前报表。如果没有当前报表，则在使用 ActiveReport 属性时返回错误代码 2476。
- ◆ ActiveDataAccessPage 属性返回当前数据访问页。如果没有当前数据访问页，则在使用 ActiveDataAccessPage 属性时返回错误代码 2022。
- ◆ ActiveControl 属性返回当前控件。如果没有当前控件，则在使用 ActiveControl 属性时返回错误代码 2474。
- ◆ PreviousControl 属性可以让你调用上一个当前控件。

为了避免错误，必须先检查哪一个对象是当前对象，然后再使用 Screen 对象对其进行操作。下面的例子用上面列出的错误来决定当前对象是窗体、报表、数据表还是数据访问页，然后显示信息框，其中给出对象的种类和名称：

```

On Error Resume Next
Dim strName As String
Dim strType As String
strType = "Form"
strName = Screen.ActiveForm.Name
If Err = 2475 Then
    Err = 0
    strType = "Report"
    strName = Screen.ActiveReport.Name
    If Err = 2476 Then
        Err = 0
        strType = "Data access page"
        strName = Screen.ActiveDataAccessPage.Name
        If Err = 2022 Then
            Err = 0
            strType = "Datasheet"
            strName = Screen.ActiveDatasheet.Name
        End If
    End If
End If
MsgBox "The current Screen object is a " & strType & vbCrLf

```

```
(继续) & vbCrLf & "Screen object name: " & strName, _
vbOKOnly + vbInformation, "Current Screen Object"
```

使用 DoCmd 对象执行命令

可以通过 DoCmd 对象在 VBA 中执行 Access 命令。可以使用 Application 对象的 DoCmd 属性返回 DoCmd 对象，但既然 DoCmd 是一个可创建对象，就不需要通过 Application 对象来使用 DoCmd 对象了。要执行一个命令，可以使用 DoCmd 对象中该命令所对应的方法。表 28.2 列出了这些方法并简要解释了它们的功能。

表 28.2 DoCmd 对象的方法

方法	解释
AddMenu	向全局菜单栏或自定义菜单栏添加菜单
ApplyFilter	应用筛选，只显示符合特定条件的记录
Beep	使计算机通过扬声器发出嘟嘟声——例如，在发生错误时吸引用户的注意力
CancelEvent	取消已经发生的事件
Close	关闭指定的对象——例如一个窗体或报表
CopyDatabaseFile	将与当前项目连接的数据库复制到 SQL Server 文件中
CopyObject	将指定对象（例如，一个查询或表）复制到指定的数据库中（或复制到数据库的一个新表中）
DeleteObject	从数据库中删除特定的对象
DoMenuItem	执行菜单上或工具栏上的某个命令。这是一个旧的命令，现在已经被 RunCommand 方法取代（见下文）
Echo	为早期版本 VBA 中的 Echo 操作提供兼容性支持。现在最好使用 Application.Echo
FindNext	查找符合 FindRecord 方法中所指定条件的下一条记录
FindRecord	查找符合指定条件的记录
GoToControl	将焦点移动到窗体或数据表中的指定控件或字段
GoToPage	将焦点移动到窗体的指定页
GoToRecord	将特定记录指定为当前记录
Hourglass	将鼠标指针转换为沙漏（等待指针）或普通指针
Maximize	最大化当前窗口
Minimize	最小化当前窗口
MoveSize	移动或改变当前窗口大小（也可以两个操作一同执行）
OpenDataAccessPage	在指定视图中打开指定的数据访问页
OpenDiagram	打开指定的数据库图标
OpenForm	打开指定的窗体，可以选择应用筛选
OpenFunction	在指定视图（例如，数据库视图）中以指定模式（例如，为数据输入打开函数）打开用户定义的函数
OpenModule	打开指定过程中的指定 VBA 模块
OpenQuery	在指定视图中以指定模式打开指定查询
OpenStoredProcedure	在指定视图中以指定模式打开指定存储过程
OpenTable	在指定视图中以指定模式打开指定表
OpenView	在指定视图中以指定模式打开指定视图
OutputTo	将数据以指定格式输出到指定对象（例如，一个报表或一个数据访问页）
PrintOut	打印指定对象
Quit	为 Access 95 提供兼容性支持。在较新版本的 Access 中，使用 Application.Quit 取代该方法

(续表)

方法	解释
Rename	用给定的名称重命名指定对象
RepaintObject	完成指定数据库对象变量被挂起的所有屏幕更新
Requery	通过根据指定条件重新对数据源完成查询来更新数据
Restore	将当前窗口恢复到一般的大小(既不是最大化也不是最小化)
RunCommand	运行指定的内置菜单命令或者工具栏命令
RunMacro	运行指定宏
RunSQL	用指定的 SQL 语句执行 Access 的当前查询
Save	保存指定对象或者(在没有指定对象时)保存当前对象
SelectObject	在数据库窗口或打开的对象中选择指定的对象
SendObject	将指定对象(例如一个窗体或一张报表)用 E-mail 发出
SetMenuItem	设置菜单项的状态——例如,启用或禁用一个菜单项
SetWarnings	打开或关闭系统信息
ShowAllRecords	从当前的窗体、查询或表中移除所有的筛选
ShowToolbar	显示或隐藏指定的工具栏
TransferDatabase	向当前的数据库或工程导入数据或从中导出数据
TransferSQLDatabase	将一个指定的 SQL Server 数据库传送到另一个 SQL Server 数据库中
TransferSpreadsheet	从电子表格导入数据或向其导出数据
TransferText	从文本文件导入数据或向其导出数据

下面的内容给出了使用表 28.2 中的方法的实例。

使用 OpenForm 方法打开窗体

要打开一个窗体, 可以使用 DoCmd 对象的 OpenForm 方法, 其语法如下:

```
expression.OpenForm(FormName, View, FilterName, WhereCondition, DataMode,  
WindowMode, OpenArgs)
```

该语法的各组成部分如下:

- ◆ expression 是必选的表达式, 返回一个 DoCmd 对象。在大部分情况下, 最好使用 DoCmd 对象本身。
- ◆ FormName 是必选的不定型参数, 指定所要打开的窗体的名称。该窗体必须属于当前数据库。
- ◆ View 是可选参数, 指定所要使用的视图: acNormal(默认值)、acDesign、acFormDS、acFormPivotChart、acFormPivotTable 或 acPreview。
- ◆ FilterName 是可选的不定型变量, 可以用来指定查询名称。该查询必须存储在当前数据库中。
- ◆ WhereCondition 是可选的不定型参数, 用来指定 SQL WHERE 子句, 省略子句中的单词 Where。
- ◆ DataMode 是可选参数, 用来指定打开窗体的模式: acFormPropertySettings、acFormAdd、acFormEdit 或 acFormReadOnly。acFormPropertySettings 是默认设定, 使用窗体中设定的模式打开窗体。

- ◆ WindowMode 是可选参数，用来指定如何打开窗体。默认模式为 acWindowNormal 即正常的窗口模式。也可以用对话模式 (acDialog) 或图标模式 (acIcon) 打开窗体，或者让它隐藏 (acHidden)。
- ◆ OpenArgs 是可选的不定型参数，用来指定当前窗口的参数——例如，将焦点移动到指定记录。

下面的例子使用 DoCmd 对象打开一个窗体，并显示在 Country 一栏中的内容为 France 的第一个记录：

```
DoCmd.OpenForm FormName:="Customers", View:=acNormal, _  
    WhereCondition:="Country = 'France'"
```

使用 PrintOut 方法打印对象

要打印一个对象，可使用 PrintOut 方法，其语法如下：

```
expression.PrintOut(PrintRange, PageFrom, PageTo, PrintQuality, Copies  
    CollateCopies)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，返回一个 DoCmd 对象。
- ◆ PrintRange 是可选参数，指定打印的内容：对象的所有内容 (acPrintAll，为默认值)、指定页 (acPages) 或者选区 (acSelection)。
- ◆ PageFrom 和 PageTo 是可选的不定型参数，与 PrintRange: = acPages 一起使用，指定了打印范围的起始页和终止页。
- ◆ PrintQuality 是可选参数，用于指定打印质量。默认设定为 acHigh，但也可以指定为 acLow、acMedium 或 acDraft (草图质量，用于需要节约墨水和时间的情况)。
- ◆ Copies 是可选的不定型参数，用来指定要打印的份数，默认值为 1。
- ◆ CollateCopies 是可选的不定型参数，可设定为 True，即采用自动分页模式；也可设定为 False，即不采用自动分页模式。默认设定为 True。

下面的例子打印了当前对象的所有页面，打印质量为高，份数为 3，不采用自动分页模式：

```
DoCmd.PrintOut PrintRange:=acPrintAll, Copies:=3, CollateCopies:=False
```

使用 RunMacro 方法运行宏

要运行一个宏，可使用 RunMacro 方法，其语法如下：

```
expression.RunMacro(MacroName, RepeatCount, RepeatExpression)
```

该语法的各组成部分如下：

- ◆ expression 是必需的表达式，返回一个 DoCmd 对象。
- ◆ MacroName 是必需的不定型参数，指定宏的名称。
- ◆ RepeatCount 是可选的不定型参数，用于指定一个表达式来控制宏运行的次数。默认值为 1。

◆ RepeatExpression 是可选的不定型参数，其中包括一个算术表达式，每次宏运行时都要对其进行运算。当表达式运算结果为 0 (False) 时，宏结束运行。

下面的例子运行名为 RemoveDuplicates 的宏：

```
DoCmd.RunMacro "RemoveDuplicates"
```

试看内部中的一段显示代码，将这个一段代码插入到 DoCmd.RunMacro 语句前面，就可以实现去重功能。

```
With Selection
    .Copy
    DoCmd.RunMacro "RemoveDuplicates"
    .Paste
End With
```

使用 RunMacro 方法

不取去其，去实 RunMacro 语句，这个一语要

```
Selection.Copy
DoCmd.RunMacro "RemoveDuplicates"
Selection.Paste
```

不取去其，去实 RunMacro 语句，这个一语要

去其，去实 RunMacro 语句，这个一语要

```
DoCmd.RunMacro "RemoveDuplicates", , , , False
```

宏 RunMacro 方法

不取去其，去实 RunMacro 语句，这个一语要

```
expression.RunMacro , RedoCount, RedoException
```

不取去其，去实 RunMacro 语句，这个一语要

去其，去实 RunMacro 语句，这个一语要

去其，去实 RunMacro 语句，这个一语要

去其。

。遨游 DAO 立数据集像 DAO 对象类 DAO 立数据集像 DAO 已——遨游 DAO

第 29 章 使用 VBA 处理 Access 数据库中的数据

- ◆ 了解如何进行操作
- ◆ 准备访问数据库中的数据
- ◆ 打开记录集
- ◆ 访问记录集中的特定记录
- ◆ 查找记录
- ◆ 返回记录的字段
- ◆ 编辑记录
- ◆ 插入和删除记录
- ◆ 关闭记录集

本章将介绍如何开始处理 Access 数据库中的数据。这样的处理可以在 Access 中，也可以在其他 VBA 宿主软件中——例如，在 Excel 或 Word 中。本章将讲解的就是如何在其他软件中进行相关的操作。

访问 Access 数据库中的数据，主要有两种方式：通过 Data Access 对象（Data Access Object，简称 DAO）或通过 ActiveX Data 对象（ActiveX Data Object，简称 ADO）。DAO 是较早时期使用的方法，既可用于 Microsoft Jet 数据库（Microsoft Jet 是 Access 数据库引擎），也可以用于 ODBC 兼容数据源（ODBC，即开放式数据库互连，是存在已久的访问数据库的标准方法。ODBC 也被用于访问开放源代码解决方案，例如 MySQL）。ADO 是高端编程界面，可以访问多种数据源。

如果软件和数据库允许选择访问方式，那么最好选择 ADO，因为它比 DAO 要简便。如果无法使用 ADO，则使用 DAO。

了解如何进行操作

一旦在 ADO 和 DAO 中间做出了选择，就可以遵循以下步骤操控数据库中的数据。

1. 在正在使用的对象库中添加一个引用。
2. 与存储数据的数据库建立连接。
3. 创建一个记录集，存储所需要的记录。
4. 对记录集中的数据进行操作。
5. 关闭记录集。

这些步骤在 ADO 和 DAO 中基本相同，只有创建数据组的方式有所差别。下面的内容将根据上面的步骤进行演示，对于 ADO 和 DAO 的不同部分，将分别进行介绍。

准备访问数据库中的数据

在开始访问数据库中的数据之前，必须先引用相应的对象库。之后，必须建立到数据源

的连接——与 ADO 对象库建立 ADO 连接或与 DAO 对象库建立 DAO 连接。

引用相应的对象库

要引用所需要的对象库，可遵循以下步骤。

1. 离开访问数据的窗口，打开或转到所需要的软件。例如，打开 Excel 或 Word。
2. 打开或激活 Visual Basic 编辑器（按 Alt + F11 键，或选择“工具”>“宏”>“Visual Basic 编辑器”）。
3. 选择“工具”>“引用”，显示“引用”对话框。
4. 向下滚动“可使用的引用”列表框以便勾选需要使用的对象库条目，然后选择其复选框：
 - ◆ 要创建 ADO 连接，勾选“Microsoft ActiveX Data Objects Library”条目。
 - ◆ 要创建 DAO 连接，勾选“Microsoft DAO Object Library”条目。

注意：“Microsoft ActiveX Data Objects Library”或“Microsoft DAO Object Library”条目的版本号决定于所使用的 Access 的版本号。如果有多个版本可选，选择最新的。

5. 单击“确定”按钮，关闭“引用”对话框。

与数据库建立连接

在访问数据库的数据之前，还需要与数据库建立连接。最容易的方法是使用“控制面板”中“数据源”程序创建一个数据源。

注意：也可以编写程序来建立连接，本章没有介绍这部分内容。

要使用“数据源”程序建立到数据库的连接，可遵循以下步骤。

1. 选择“开始”>“控制面板”，打开“控制面板”窗口：
 - ◆ 如果控制面板为分类视图，单击“性能和维护”链接，再单击“管理工具”链接，然后双击“数据源（ODBC）”图标。
 - ◆ 如果控制面板为经典视图，双击“数据源（ODBC）”图标。
2. 在“ODBC 数据源管理器”对话框中，单击“系统 DSN”选项卡，显示其页面（如图 29.1 所示）。
3. 单击“添加”按钮以显示“创建新数据源”对话框（如图 29.2 所示）。
4. 在列表框中选择“Microsoft Access Driver (*.mdb)”条目。
5. 单击“完成”按钮，显示“ODBC Microsoft Access 安装”对话框（如图 29.3 所示，图中已设定相关内容）。
6. 在“数据源名”文本框中，键入所要创建的数据源的名称。该名称不一定要使用数据源文件的实际名称，只是用于代称该数据源的名称，可根据需要进行指定。
7. 在“说明”文本框中，键入对连接的说明。这是可选的，但如果要创建多个名称相似的连接，“说明”框中的文本就可以帮助区分它们。（当其他人需要使用由自己创建的连接时，也可以帮助他们了解每个连接的用途。）
8. 单击“选择”按钮，显示“选择数据库”对话框。转到并选择数据库文件。
9. 如果要以只读模式打开数据库，可勾选“只读”复选框。

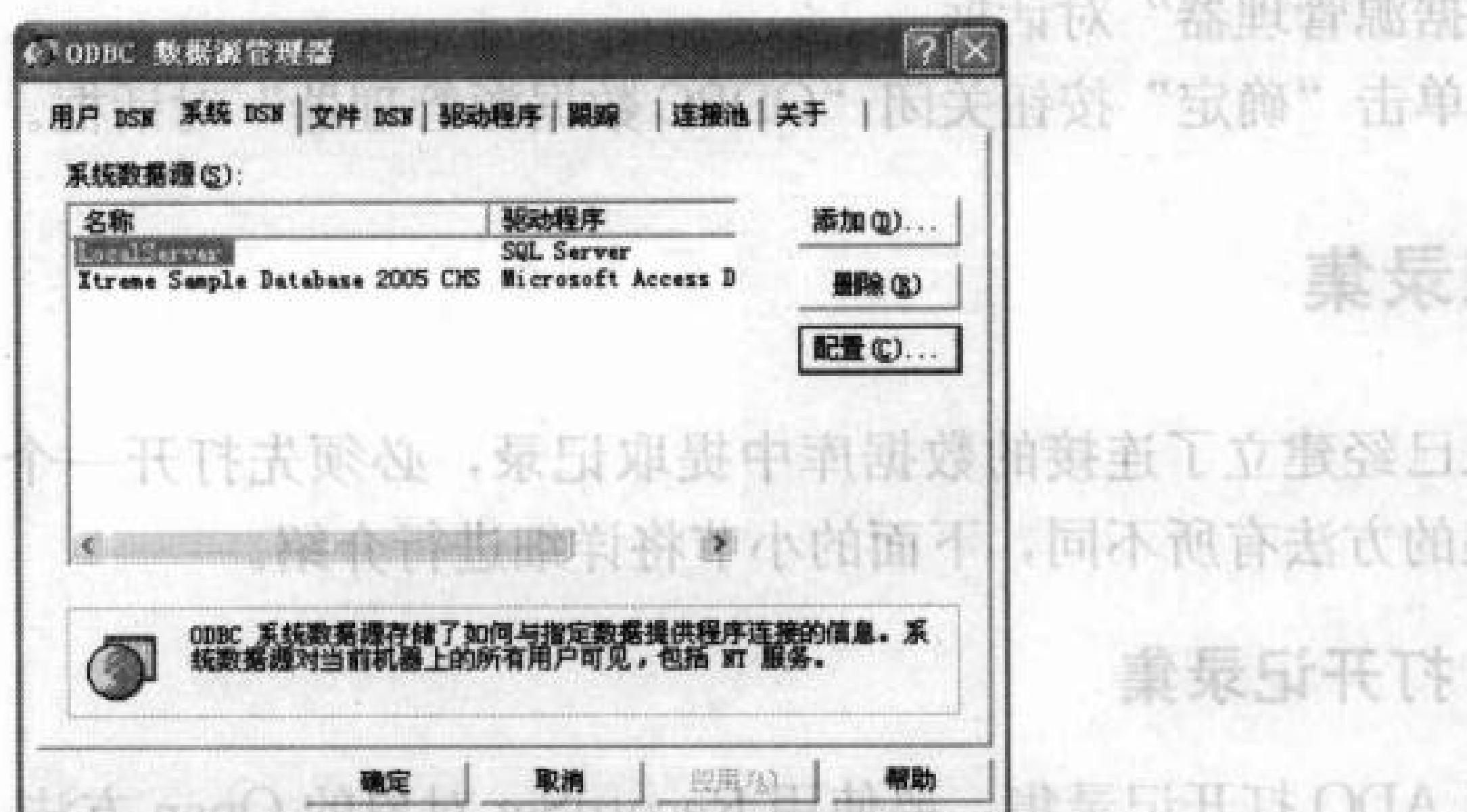


图 29.1 使用“ODBC 数据源管理器”对话框建立到数据库的连接

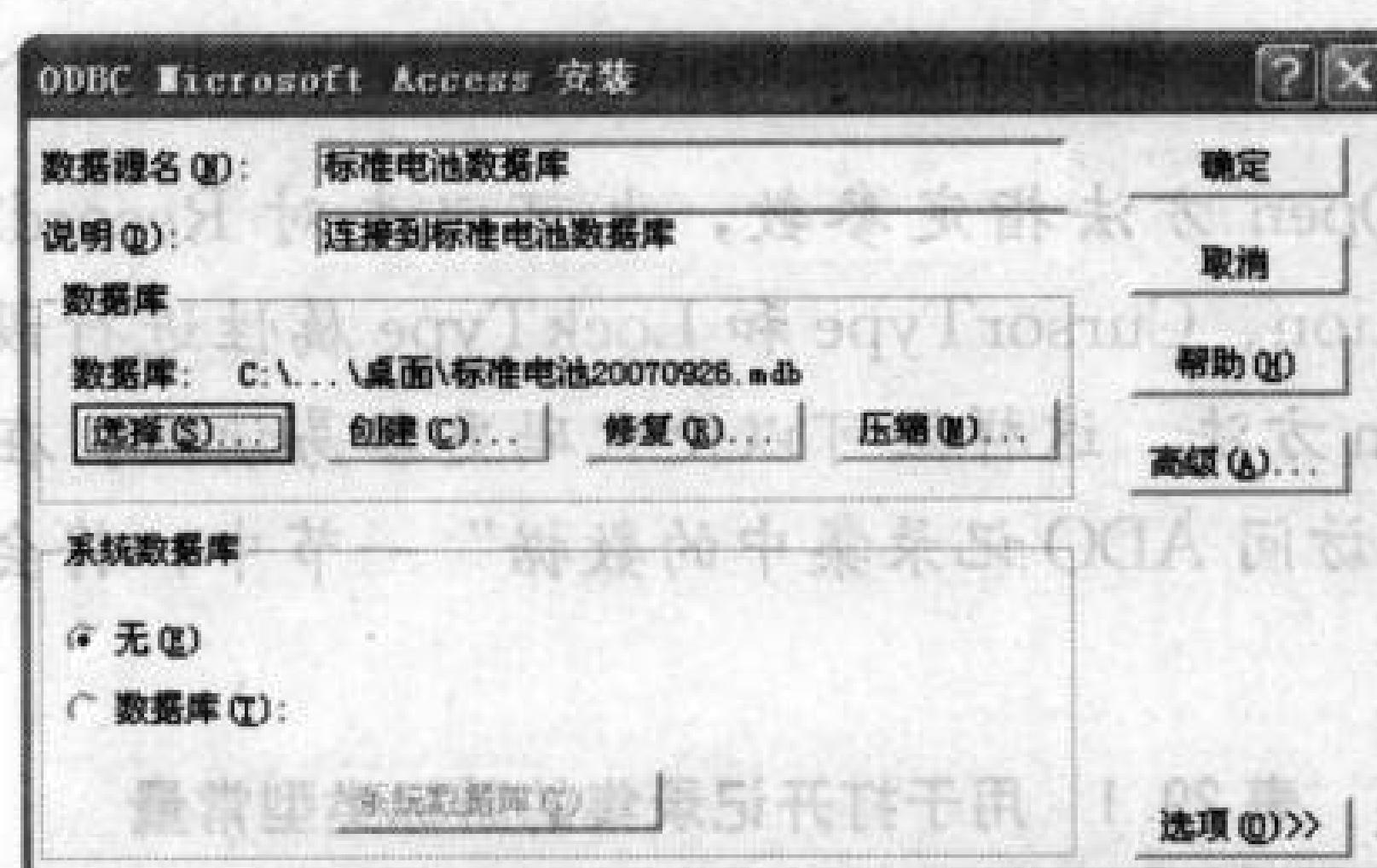
图 29.2 要创建到 Access 数据库的连接，可在“创建新数据源”对话框中条目选择“Microsoft Access Driver (*.mdb)”。
图 29.3 使用“ODBC Microsoft Access 安装”对话框设定连接的详细内容

图 29.3 使用“ODBC Microsoft Access 安装”对话框设定连接的详细内容

10. 如果要以独占模式而不是共享模式打开数据库，可勾选“独占”复选框。

提示：也可以在“ODBC Microsoft Access 安装”对话框中设定独占模式或只读模式。单击“选项”按钮，然后勾选“独占”复选框或“只读”复选框。

11. 单击“确定”按钮关闭“选择数据库”对话框，回到“ODBC Microsoft Access 安装”对话框。

12. 单击“确定”按钮关闭“ODBC Microsoft Access 安装”对话框，显示“ODBC 数

据源管理器”对话框。

13. 单击“确定”按钮关闭“ODBC 数据源管理器”对话框。

打开记录集

要从已经建立了连接的数据库中提取记录，必须先打开一个记录集。ADO 和 DAO 打开记录集的方法有所不同，下面的小节将详细进行介绍。

用 ADO 打开记录集

要用 ADO 打开记录集，要使用 RecordSet 对象的 Open 方法，其语法如下：

```
recordset.Open Source, ActiveConnection, CursorType, LockType, Options
```

该语法的各组成部分如下：

- ◆ recordset 是需要打开的 RecordSet 对象。通常会使用一个对象变量来引用 RecordSet 对象。
- ◆ Source 是可选的不定型参数，指定包含该记录集的表格、命令、SQL 语句或文件。
- ◆ ActiveConnection 是可选的不定型参数，既可以是一个 Connection 类别的对象变量，也可以是一个具有用于连接的参数的不定型或字符串变量。
- ◆ CursorType 是可选参数，用于指定在记录集中使用的游标的类型。表 29.1 解释了游标的各种类型。
- ◆ LockType 是可选参数，用于指定如何在记录集打开时锁定它。表 29.2 解释了锁定选项。
- ◆ Options 是可选的长整型参数，用于控制 Source 值的运算方式，在 Source 值不是 Command 对象时使用。表 29.3 解释了可用的常量，这些常量被分为两种：一种是命令类型选项，一种是执行选项。软件允许在 Options 参数中使用两个或多个常量。

提示：如果不想用 Open 方法指定参数，也可以先对 RecordSet 对象的 Source、ActiveConnection、CursorType 和 LockType 属性进行设定，然后再使用没有参数的 Open 方法。这样做可以使代码更容易读懂。在后面的“使用 SQL SELECT 语句访问 ADO 记录集中的数据”一节中，将会给出一个使用这种方法的例子。

表 29.1 用于打开记录集的游标类型常量

常量	游标类型及解释
adOpenForwardOnly	仅向前游标。只允许在记录集中向前滚动。这是默认的游标类型。在只需要检查一遍记录时，可以选择这个类型。
adOpenDynamic	动态游标。允许在记录集中自由移动，查看其他用户对记录做出的修改。
adOpenKeyset	键集游标，允许在记录集中自由移动，查看其他用户对记录做出的修改。但禁止查看其他用户添加的记录，并且禁止访问其他用户删除的记录。
adOpenStatic	静态游标，不能看到其他用户做出的修改。当只需要查找某个数据，或者依据已有数据生成报表时，可以使用静态游标。

表 29.2 用于通过 ADO 打开记录集的锁定选项

常量	打开记录集的方式
adLockReadOnly	数据为只读模式，不允许修改。如果只需要查找或分析数据而不需要对其进行任何操作，就可以使用这个常量
adLockOptimistic	仅在运行 Update 方法对记录进行显式更新时锁定它
adLockBatchOptimistic	允许对多条记录同时进行更新
adLockPessimistic	在修改过记录之后立即锁定该记录

表 29.3 打开记录集时可用的 Options 参数

常量	解释
命令类型选项	
adCmdText	按指定命令或存储过程调用的文本计算 Source 值
adCmdTable	按表名计算 Source 值，该表的列全部由内部生成的 SQL 查询返回
adCmdStoredProc	按储存的过程名计算 Source 值
adCmdFile	按存储记录集的文件名计算 Source 值
adCmdTableDirect	按表名计算 Source 值，并返回表的所有列。不能与 adAsyncExecute 同时使用
执行选项	
adAsyncExecute	异步执行命令。不能与 adCmdTableDirect 同时使用
adAsyncFetch	同步找回 CacheSize 属性所指定的行，并异步执行剩下的行
adAsyncFetchNonBlocking	防止在获取数据时阻挡其他的数据访问
adExecuteRecord	将 Source 返回的数据指定为单一的行并转换成一条记录

本章后面的内容将给出打开记录集的实例。首先要决定的是如何访问记录集中的数据，最简单的方法是使用一个已经存在的表或使用 SQL SELECT 语句。

选择如何访问 ADO 记录集中的数据

访问记录集的方法取决于需要表中所有的数据还是部分数据。如果需要调用表中所有的数据，可以用一个表来访问数据。如果只想调用一部分特定的数据，则可以使用 SQL SELECT 语句。

用表来访问 ADO 记录集中的数据

要用表来打开记录集的数据库中的数据，可在 Open 语句中将表的名称指定给 Source 参数。下面的例子中声明了一个 RecordSet 对象变量，用 CreateObject 语句将其赋值为相应的记录集，然后使用 Open 方法打开 Main 数据库中的 Customers 表：

```
Dim myRecordset As RecordSet
Set myRecordset = CreateObject("ADODB.Recordset")
myRecordset.Open Source:="Customers", ActiveConnection:="Main"
```

使用 SQL SELECT 语句访问 ADO 记录集中的数据

如果只想将符合指定条件的数据添加到记录集中，可以使用 SQL SELECT 语句。SELECT 语句可能很复杂，但稍加练习就能够熟练使用，其语法如下：

```
SELECT [DISTINCT] fields FROM table WHERE criteria ORDER BY fields [DESC]
```

大写的单词为 SQL 关键词，小写的斜体单词部分，可填入用户提供的信息。以下是详细介绍：

- ◆ SELECT 关键词表示生成一条语句选择记录（相应地，也可删除记录）。
- ◆ 可以加入可选关键词 DISTINCT（方括号表示其为可选关键词）让程序只返回唯一的记录，而排除重复的记录，如果省略 DISTINCT，程序也会返回重复的记录。
- ◆ fields 列出了需要在记录集中显示的字段。如果有两个或更多的字段名，用逗号将它们隔开——例如，contact, company, address。要返回所有字段名，可输入通配符（*）。
- ◆ FROM table 指定要从中抽取数据的表名。
- ◆ WHERE criteria 指定筛选记录的条件。输入字段名、等号、左单引号、需要查找的值以及右单引号。例如：WHERE City = 'Taos'，返回所有 City 字段内的值为 Taos 的记录。
- ◆ ORDER BY fields 指定用于排序的字段。如果有两个或更多字段，将它们按顺序排列（主要关键词放在第一个位置，次要关键词放在第二个位置，依次类推）并用逗号隔开。默认的排序方式为升序，但也可以添加一个 DESC 关键字，强制降序排列。例如，ORDER BY ZIP DESC 根据 ZIP 字段降序排列，而 ORDER BY State, City 则根据 State 字段升序排列，State 字段相同时再根据 City 字段升序排列。

由于 SQL SELECT 语句包含了太多的元素，将 SELECT 语句作为参数添加到 Open 语句中，会使代码行过长。可以像平常一样将同一行代码分成几行，但更简单的方法是使用 Recordset 对象的属性，而不是用 Open 参数来指定记录集的详细信息，就像下面的例子中那样。（也可以将 SELECT 语句赋值给一个字符变量，然后用它来提供参数。）下面的例子使用 SELECT 语句创建一个记录集，其中包含了 Country 字段的值为 Mexico、Brazil 或 Argentina 的记录。例子中先将记录集中的记录按 Country 字段排列，再按 City 字段排列。

```

With myRecordset
    .Source = "SELECT * FROM Suppliers WHERE Country='Mexico' & _
    "OR Country='Brazil' OR Country='Argentina'" & _
    "ORDER BY Country, City;"
    .ActiveConnection = "Main"
    .CursorType = adOpenStatic
    .Open
    If myRecordset.RecordCount = 0 Then
        MsgBox "There are no records that match the specified criteria."
    Else
        MsgBox myRecordset.Fields("CompanyName") & ":" & _
        myRecordset.Fields("Country")
    End If
End With

```

因为有可能出现没有任何记录符合条件的情况，所以在例子中加入了 RecordSet 对象的 RecordCount 属性，以便记录符合要求的记录的数量。如果没有 (myRecordset. RecordCount = 0)，则显示信息框以提示读者。如果有，则显示信息框提示第一条符合条件的记录的公司名和国家。

用 DAO 打开记录集

在使用 DAO 时，可以使用 Database 对象的 OpenRecordset 方法创建新的记录集并将其

加入到 Recordsets 集合中。

OpenRecordset 方法的语法如下：

```
Set recordset = object.OpenRecordset (Name, Type, Options, LockEdit)
```

该语法的各组成部分如下：

- ◆ recordset 为对象变量，代表需要打开的 RecordSet 对象。
- ◆ object 为对象变量，代表创建新的 RecordSet 对象所依据的数据库。
- ◆ Name 为字符串参数，指定为记录集提供记录的表、查询或 SQL 语句。如果使用 Jet 数据库，并返回了一个表型记录集，那么在 Name 参数中只能使用一个表名。
- ◆ Type 是可选参数，可用于指定需要打开的记录集的类型。表 29.4 解释了 Type 参数可使用的常量。
- ◆ Options 是可选参数，用于指定 Access 如何打开记录集的常量。表 29.5 解释了 Options 参数可使用的常量。
- ◆ LockEdit 是可选常量，用于指定锁定记录集的方式。表 29.6 解释了 LockEdit 参数可使用的常量。

表 29.4 OpenRecordSet 方法的 Type 参数中可用的常量

常量	打开该类型的数据
dbOpenTable	表型，只用于 Microsoft Jet 工作区。如果在 Jet 工作区中打开记录集，此为默认设定
dbOpenDynamic	动态型，只用于 ODBC Direct 工作区。类似于 ODBC 动态游标，允许用户添加、删除和修改数据库表中的行
dbOpenDynaset	动态集型，类似于键集游标，允许添加、删除和修改数据库表中的行，也允许用户在动态集内任意浏览
dbOpenSnapshot	快照型，类似于 ODBC 静态游标。打开了记录的快照，但在其他用户修改记录时，不会自动更新。想要更新快照，必须关闭记录集，并重新打开
dbOpenForwardOnly	仅向前型，只允许向前滚动记录集

表 29.5 参数 Options 可使用的常量

常量	解释	限制
dbAppendOnly	允许用户添加新记录但不能编辑或删除已有的记录	只用于 Jet 动态集型记录
dbSQLPassThrough	向利用 Jet 连接的 ODBC 数据源传递一条 SQL 语句	只用于快照型记录
dbSeeChanges	如果用户试图修改其他用户正在编辑的数据，就会产生运行错误	只用于 Jet 动态集型记录
dbDenyWrite	禁止其他用户添加或修改记录	只用于记录集
dbDenyRead	禁止其他用户读取记录数据	只用于表型记录
dbForwardOnly	强制使用仅向前记录集。该选项用于为较早版本提供兼容性。现在则使用 Type: =dbOpenForwardOnly 代替	只用于 Jet 快照型记录集
dbReadOnly	禁止用户修改记录集。该选项用于为较早版本提供兼容性。现在则使用 LockEdits: = dbReadOnly 代替。如果必须要使用 Options: = dbReadOnly，则不能同时使用 LockEdits 参数	只用于 Jet 记录集

(续表)

常量	解释	限制
dbRunAsync	(相对) 异步执行查询(以便能够在其他进程执行时返回一部分结果)	只用于 ODBCDirect 工作区
dbExecDirect	通过调用 SQLExecDirect 执行查询	只用于 ODBCDirect 工作区
dbInconsistent	允许不同步更新。可以只更新一个表格中的某个字段，而不同步更新其他字段。可以单独使用该常量，或者 dbConsistent 常量，但两者不能同时使用	只用于 Jet 动态集型和快照型记录集
dbConsistent	只允许同步更新，即属于不同记录集的表中的共享字段，必须同时更新。可以单独使用该常量，或 dbInconsistent 常量，但两者不能同时使用	只用于 Jet 动态集型和快照型记录集

表 29.6 参数 LockEdits 所使用的常量

常量	解释	默认值或限制
dbReadOnly	禁止用户修改记录集。可以代替 Options:=dbReadOnly；两者不能同时使用。	ODBCDirect 工作区的默认设定
dbPessimistic	在修改完一条记录之后立刻锁定它	Jet 工作区的默认设定
dbOptimistic	仅在运行 Update 方法对记录进行显式更新时锁定它	只用于 ODBCDirect 工作区
dbOptimisticValue	比较新旧记录中的值，在记录被访问后检查是否被修改。该同步基于记录的行值	只用于 ODBCDirect 工作区
dbOptimisticBatch	允许同时更新多条被修改过的记录	只用于 ODBCDirect 工作区

用表打开 DAO 记录集

最简单的打开 DAO 记录集的方法是打开一张完整的表。可以为 Name 参数指定表的名称，并使用 Type:=dbOpenTable 显式声明你正在打开表。下面的例子声明对象变量 myRecordset 为 DAO. Recordset 对象，然后将其赋值为 myDatabase 数据库中的 Customers 表：

```
Dim myRecordset As DAO.Recordset
Set myRecordset = myDatabase.OpenRecordset(Name:="Customers", _
```

```
Type:=dbOpenTable)
```

用 SQL SELECT 语句打开 DAO 记录集

如果只要返回一部分特定的数据，可以使用 SQL SELECT 语句打开 DAO 记录集。(要了解 SQL SELECT 语句的详细内容，可参阅本章前面的“使用 SQL SELECT 语句访问 ADO 记录集中的数据”小节。) 指定该语句为 OpenRecordset 方法的 Name 参数，如下例所示，声明了 Recordset 对象变量，然后将 SELECT 语句运行数据库的结果指定给该对象变量：

```
Dim myDatabase As DAO.Database
Set myDatabase = OpenDatabase("Z:\Database\Main.mdb")
```

```
Dim myRecordset As DAO.Recordset
Set myRecordset = myDatabase.OpenRecordset _
(Name:="SELECT * FROM Customers WHERE Country='Germany'", _
```

```
Type:=dbOpenDynaset)
```

访问记录集中的特定记录

要访问记录集中的特定记录，可以浏览记录集来查找所需的记录。RecordSet 对象包含了以下方法以便完成在记录集中的滚动。

方法	移动到记录
MoveFirst	移动到第一条记录
MoveNext	移动到下一条记录
MovePrevious	移动到前一条记录
MoveLast	移动到最后一条记录
Move	移动到指定的记录

使用 MoveFirst、MoveNext、MovePrevious 和 MoveLast 方法

MoveFirst 和 MoveLast 在任何情况下都可以使用，因为记录集都包含了至少一条记录，所以总是存在第一条记录和最后一条记录。（如果一个记录集中只包含了一条记录，那么它既是第一条记录也是最后一条记录。）但是如果在第一条记录处使用 MovePrevious 方法，或在最后一条记录处使用 MoveLast 方法，就会移动到记录集之外的被称为“幽灵”的记录中，事实上这条记录并不存在。当试图访问该记录的内容时，VBA 会给出运行错误 3021（“BOF 或 EOF 中有一个是‘真’，或者当前的记录已被删除，所需的操作要求一个当前的记录”）。参见图 29.4 中的错误提示。

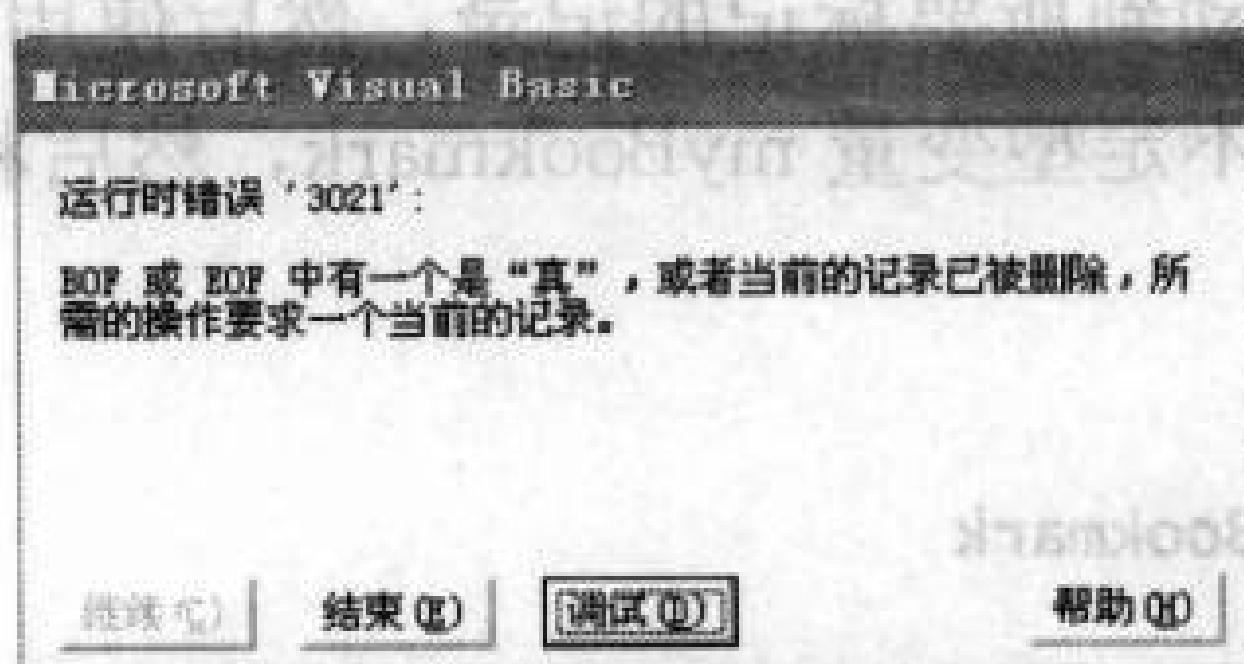


图 29.4 运行时错误“BOF 或 EOF 中有一个是‘真’，……”

通常表示由于在第一条记录处使用了 MovePrevious 方法，或在最后一条记录处使用了 MoveNext 方法，导致的移出记录集错误

BOF 是指文件的开头，EOF 是指文件的结尾。要检查你现在是否在记录集的开头或结尾，可使用 RecordSet 对象的 BOF 属性或 EOF 属性。当前记录在文件开头处时，BOF 属性返回 True；当前记录在文件结尾处时，EOF 属性返回 True。为了避免错误，可以在使用 MovePrevious 方法之后检查是否已经到达文件的开头处，如下例所示：

```
With myRecordset
    .MovePrevious
    If .BOF = True Then .MoveNext
End With
```

类似地，也可以在使用 MoveNext 方法之后检查是否已经到达了文件的结尾，如下例所示：

```
myRecordset.MoveNext  
If myRecordset.EOF Then myRecordset.MovePrevious
```

使用 Move 方法移动到指定记录

要移动到指定记录，而不是记录集的第一条或者最后一条记录，可使用 Move 方法。其语法对于 ADO 和 DAO 有所不同。

用于 ADO 的 Move 方法的语法如下：

```
recordset.Move NumRecords, Start
```

用于 DAO 的 Move 方法的语法如下：

```
recordset.Move Rows, StartBookmark
```

这里的 recordset 是需要使用的 RecordSet 对象，NumRecords 或 Rows 是要移动到的记录数（正数表示向下移动，负数表示向上移动），Start 或 StartBookmark 是可选参数，用于指定移动的书签。如果省略 Start 或 StartBookmark 参数，那么移动从当前记录开始。

例如，下面的语句从 ADO 记录集的当前记录向下移动了 10 条记录：

```
myRecordset.Move NumRecords:=10
```

下面的语句从 DAO 记录集的当前记录向上移动了 5 条记录：

```
myRecordset.Move Rows:=-5
```

要创建一个书签，可以移动到所要标记的记录，然后使用 RecordSet 对象的 Bookmark 属性。下面的例子声明了一个不定型变量 myBookmark，然后将其指定为代表当 ADO 记录集的当前记录的书签：

```
Dim myBookmark As Variant  
myBookmark = myRecordset.Bookmark
```

设定完书签后，可以将其作为移动的起始点来使用。例如，下面的语句将其移动到 ADO 记录集的书签下的第 8 条记录：

```
myRecordset.Move NumRecords:=8, Start:=myBookmark
```

查找记录

在 ADO 记录集和 DAO 记录集中查找记录的方法有所不同。下面将详细介绍如何查找记录。

提示：ADO 记录集和 DAO 记录集都包括了 Seek 方法。比起本文中提到的 ADO 的 Find 方法和 DAO 的四个 Find 方法，Seek 方法更加复杂，功能也更加强大。可以参考“Access VBA 帮助”文件中的 Seek 方法了解详情。

在 ADO 记录集中查找记录

要在 ADO 记录集中查找记录，可以使用 RecordSet 对象的 Find 方法，其语法如下：

```
recordset.Find Criteria, SkipRows, SearchDirection, Start
```

该语法的各组成部分如下：

- ◆ recordset 是需要在其中执行查找的记录集。

- ◆ Criteria 是必选的字符串参数，指定列的名称、比较的类型以及要搜索的值。例如，可以指定 State 列的值等于 (=) CA。

- ◆ SkipRows 是可选的长整型值，用于指定从当前行迁移的行数（或者从 Start 参数指定的书签处），查找将从迁移后的行开始，而不是从当前行。例如，如果设定迁移为 3 行，那么查找将从当前行向下的第三行开始。

- ◆ SearchDirection 是可选参数，用于指定查找的方向是向下还是向上。默认值为 adSearchForward；将值改为 adSearchBackward，可指定为向上查找。

- ◆ Start 是可选的不定型参数，指定开始查找或迁移起点的书签。如果省略 Start 参数，查找从当前行开始。

在执行查找的过程中，计算机会在找到第一个符合条件的记录时停止查找。如果没有符合条件的记录。且查找的方向为向下，则会停止在记录集的末尾；如果查找的方向为向上，将在记录集的开头停止。如果查找停止在记录集的开头或结尾，那么就说明在查找范围内没有符合条件的记录。

下面的例子首先移动到记录集的第一个记录，该记录集由对象变量 myRecordset 表示；然后查找符合条件 " State= 'CA' " 的记录。该例子检查 EOF 属性来确认是否已经到达了记录集的末尾。如果没有，显示一个信息框，提示该记录中的公司名字段；如果到达了结尾，显示信息框，说明没有符合条件的记录。

```
With myRecordset
```

```
.MoveFirst
.Find Criteria:="State='CA'"
If Not .EOF Then
    MsgBox .Fields("CompanyName")
Else
    MsgBox "No matching record was found."
End If
End With
```

要用相同的条件再进行一次查找，可使用 SkipRows 参数指定一个迁移值，这样就不会再将当前的记录计入查找范围。例如：

```
myRecordset.Find Criteria="State='CA'", SkipRows:=1
```

在 DAO 记录集中查找记录

要在 DAO 记录集中查找记录，可以使用以下的 4 种方法：

- ◆ FindFirst 方法，从记录集的开头开始向下查找。

- ◆ FindNext 方法，从当前记录开始向下查找。

- ◆ FindPrevious 方法，从当前记录开始向上查找。

- ◆ FindLast 方法，从记录集的末尾开始向上查找。

这 4 种方法的语法如下：

```
recordset.FindFirst Criteria
recordset.FindNext Criteria
recordset.FindPrevious Criteria
recordset.FindLast Criteria
```

recordset 是必需的对象变量，代表需要在其中执行查找的记录集。Criteria 是必需的字符串型参数，指定查找的条件。Criteria 的作用与 SQL 语句中的 WHERE 分句相同，只是没有使用 WHERE。

下面的例子使用 FindFirst 方法从记录集的开头开始查找第一个符合条件 Country = 'Mexico' 的记录：

```
myRecordset.FindFirst "Country = 'Mexico'"
```

当在 DAO 记录集中用上面的方法执行查找时，RecordSet 对象的 NoMatch 属性被指定为 True。如果找到了符合条件的记录，NoMatch 属性将被指定为 False。因此，可以通过 NoMatch 属性了解是否找到了符合条件的记录。例如：

```
If .NoMatch = False Then
```

```
MsgBox myRecordset.Fields("CompanyName").Value
```

```
End If
```

返回记录中的字段

如果移动到了某个记录，就可以通过使用 Fields 集合中相应的 Field 变量返回字段中的值。Field 属性是 RecordSet 对象的默认属性，因此可以在语句中省略。例如，下面两条语句都返回了当前记录中的 WorkPhone 字段：

```
myRecordset.Fields("WorkPhone")
myRecordset("WorkPhone")
```

编辑记录

要修改记录中的数据，可以给需要修改的字段指定相应的值，然后使用 RecordSet 对象的 Update 方法来更新表中的数据。下面的例子将 ContactName 字段中的值修改为 Pieter Schmidt，然后使用 Update 方法进行了更新：

```
With myRecordset
    .Fields("ContactName").Value = "Pieter Schmidt"
    .Update
End With
```

插入和删除记录

要插入一条记录，可以使用 RecordSet 对象的 AddNew 方法。然后为记录中的字段指定数据。最后用 Update 方法将该记录添加到数据库的表中。下面的例子用 With 语句完成

这些操作：

```
With myRecordset
    .AddNew
    .Fields("CustomerID").Value = "MURP04"
    .Fields("ContactName").Value = "Andrea Murphy"
    .Fields("CompanyName").Value = "Murphy's Chipping Services"
    .Fields("City").Value = "City of Industry"
    'add data for the other fields here
    .Update
End With
```

要删除记录，需要首先移动到该记录或者查找到它，然后先使用 Delete 方法删除记录，再使用 Update 方法更新它。下面的例子删除了当前记录并更新了表中的数据：

```
myRecordset.Delete
myRecordset.Update
```

关闭记录集

在操作完一个对象之后，必须关闭它。要关闭记录集，可以对相应的 RecordSet 对象或代表该 RecordSet 对象的对象变量使用 Close 方法。下面的例子关闭了一个记录集，该记录集由变量 myRecordset 表示：

```
myRecordset.Close
```

在关闭了记录集之后，将对象变量赋值为 Nothing 以释放其占有的内存：

```
Set myRecordset = Nothing
```

如果不再使用该对象，应将其关闭。关闭对象后，将不再从内存中读取数据。如果不再使用该对象，应将其关闭。关闭对象后，将不再从内存中读取数据。

感谢阅读本书！

如果您有任何问题或建议，请随时与我们联系。我们非常欢迎您的反馈和意见。感谢您选择我们的产品，祝您学习愉快！

第30章

第30章 实现不同软件间的访问

- ◆ 不同软件相互交流的工具
- ◆ 使用自动方式传输信息
- ◆ 使用 Shell 函数运行软件
- ◆ 使用数据对象存储和获取信息
- ◆ 通过 DDE 进行数据交换
- ◆ 通过 SendKeys 语句进行数据交换

至此，本书已经介绍了如何利用 VBA 在 VBA 宿主软件中完成所需的操作。然而，有时（也可能很经常）也需要在某个软件中对另一个软件进行操作。本章将介绍一些工具来实现对这样的操作，这些工具分别为：自动方式、数据对象、动态数据交换（DDE）以及 SendKeys 语句。

不同软件相互交流的工具

大多数的 VBA 宿主软件（比如，本章用做例子的 Office 软件）都提供了若干个工具以便与另一个软件进行交流。主要有以下几种：

自动方式 正式的名称为对象链接和嵌入（OLE），这是最新也是最有效的在 Windows 软件之间传输信息的方法。如果所使用的软件支持 OLE，那么最好优先使用这种方法，其次再考虑 DDE 和 SendKeys。

动态数据交换（DDE） 这是一种较早时期使用的传输数据的方法，当 OLE 不可用时能够起到很好的作用。DDE 只在某些软件中可用。

SendKeys 这是一种较早时期使用的软件交流的方法。SendKeys 通过传送另一软件的键击动作来实现对接，而不像 OLE 和 DDE 那样用更加成熟的方法进行操作。尽管它与 OLE 和 DDE 方法相比，显得十分原始，但仍然很有效。

除了这 3 种连接途径外，本章还将介绍 DataObject 对象，用来存储信息，通过 Windows 剪贴板传送并获取信息。

提示：如果软件没有提供任何本章中提到的方法，也可以通过命令行来实现操作。例如可以使用 /p 命令行在各个软件中切换，打印一个文件，而不需要通过任何用户界面。可以在网上搜索“命令行”、VBA 以及软件的名称获得相关的信息。

使用自动方式传输信息

在与其他软件交流时，自动方式是功能最强大且最有效的方法。每一种支持自动方式的软件都会提供一个或多个组件对象模型（COM）对象，它可以通过编写程序来访问，通常

一个对象代表一种软件（即该软件生成的各种文件），多个对象代表主要的组件，等等。

在自动方式传输中，包括一个提供信息的“服务器软件”和一个接受信息的“客户端软件”。（也有其他的术语来表示这样一对软件：“服务器软件”也被称为“对象软件”，“客户端软件”也被称为“控制软件”）。

自动方式允许客户端软件控制服务器软件的功能。例如，Excel 比 Word 具有更强大的计算功能，可以创建图表、数据分布图，等等。通过使用自动方式，Word 可以调用 Excel 的计算引擎来完成计算，并把结果插入到 Word 文档中。也可以使用 Excel 来创建图表，并插入到文档中。Word 可以完成更多被限制的操作，例如让 Excel 打开工作簿，复制其中的一些单元格，然后将它们复制并链接到文档中。

要通过 VBA 使用自动方式，可以在 VBA 中创建一个对象，用来引用需要操作的软件。可以用 CreateObject 函数在另一个软件中创建新的对象，用 GetObject 函数获取在另一个软件中已经存在的对象。

在使用自动方式时，可以选择显示服务器软件，也可以使其对用户隐藏。在某些过程中，需要显示服务器软件——例如，当需要用户选择文件或文件夹以及需要完成其他交互操作时。在其他过程中，最好对用户隐藏服务器软件，避免自行弹出的软件窗口使用户不知所措。

即使决定了要在运行过程中隐藏服务器软件，但在大多数情况下，在编写程序的过程中，最好选择显示服务器软件。这样做便于在代码运行不正确时查找错误。

前期绑定和后期绑定

当使用自动方式访问另一软件时，可以选择绑定的方式，即如何建立服务器软件和客户端软件之间的引用。

前期绑定需要将引用添加到软件的对象库中，即在设计阶段调用“引用”对话框（选择“工具”>“引用”），然后在代码的开头使用 Dim 语句声明对象为指定的对象类，而不是先声明一个对象（Object）。例如，下面的语句通过使用前期绑定调用了一个 PowerPoint 演示文稿中的一张幻灯片：

```
Dim myPowerPoint As PowerPoint.Application  
Dim myPresentation As Presentation  
Dim mySlide As Slide  
Set myPowerPoint = CreateObject("PowerPoint.Application")  
Set myPresentation = myPowerPoint.Presentations.Add  
Set mySlide = myPresentation.Slides.Add(Index:=1, Layout:=ppLayoutTitleOnly)
```

后期绑定是指在运行代码的过程中创建对象，并与另一软件建立关联。如果显式声明一个对象，就是将其声明为一个对象（As Object），而不是将其声明为特定的对象类。

例如，下面的语句声明了一个对象变量 myOutlook，然后将其指定为 Outlook. Application 对象的引用：

```
Dim myOutlook As Object  
Set myOutlook = CreateObject("Outlook.Application")
```

注意：不是所有支持自动方式的软件都支持前期绑定。一些软件没有提供预先设置功能的途径，而只能在运行过程中进行引用。这些软件只能使用后期绑定。

如果服务器软件支持前期绑定，那么最好优先使用前期绑定而非后期绑定。这样做有 3 个优点：

◆一旦向软件的对象库添加了相应的引用，就可以通过宿主软件中的 Visual Basic 编辑器访问该软件的对象、属性和方法。这样做更容易找到所需的对象、属性和方法，而且可以避免一些错误，例如打字错误和丢失参数的错误。

◆由于在声明变量时就已经指定了对象类型，所以就减少了错误地获取不当对象的可能性。

◆因为在使用前期绑定时 VBA 所获取的关于对象的信息更加全面，所以引用对象的方法和属性将会更快捷。

用 CreateObject 函数创建对象

CreateObject 函数能够创建并返回一个对已有的 OLE 对象的引用，其语法如下：

```
CreateObject(class [,servername])
```

这里的 class 是必选参数，指定要创建的对象所属的类。class 参数包括库的名称，该库能够提供对象以及对象的类型，举例如下：

```
applicationname.objecttype
```

例如，要将 Excel Application 对象指定为类，应使用 Excel Application 对象的 class 参数。这里的 Excel 是提供对象的软件名称，Application 是 Excel 所提供的对象的类型。类似地，可以使用 Excel.Sheet 来表示 Excel 中的工作表对象。

servername 是字符串子类型下的可选不定型参数，用于指定新建的对象所在的网络服务器的名称。如果要使用本地计算机，可以省略 servername 或指定一个空字符串。要使用远程计算机，则必须安装 DCOM（分布式对象组件模型），并且服务器计算机上的对象必须设定为允许远程创建操作。

通常，CreateObject 函数会与 Set 语句同时使用。Set 语句用于将所创建的对象赋值给一个对象变量。例如，下面的语句声明了一个名为 myNewSheet 的对象变量，并将其赋值为一个 Excel 工作表对象：

```
Dim myNewSheet As Object  
Set myNewSheet = CreateObject("Excel.Sheet")
```

提示：也可以对计算机系统中的 COM 对象使用 CreateObject 函数，并非只能对 application 对象使用该函数。

用 GetObject 函数返回一个对象

GetObject 函数可以返回一个到已有的自动方式对象的引用，其语法如下：

```
GetObject([pathname] [, class])
```

这里， pathname 是字符串子类型的可选不定型参数，指定需要获取的对象所属文件的文件名和完整路径。 pathname 是可选的，但如果指定了 pathname，就必须指定 class 参数。如果不指定 class 参数，则必须指定 pathname。类是字符串子类型的一个不定型参数，指定需要返回的对象所属的类。

就像和 CreateObject 同时使用一样，通常 GetObject 函数会与 Set 语句同时使用，以便将一个对象变量指定为用 GetObject 函数所获取的对象。例如，在下面第二行的语句中，GetObject 函数返回了一个对象，该对象中包括工作簿“Z:\Finance\Revenue.xls”。Set

语句将该对象指定给名为 Revenue 的对象变量的值。第一条语句声明了 Revenue 变量：

```
Dim Revenue As Object  
Set Revenue = GetObject("Z:\Finance\Revenue.xls")
```

这里的工作簿是与 Excel 链接的。当运行代码时，如果 Excel 不处在运行状态下，那么 VBA 会启动 Excel 并激活该工作簿。然后就可以通过相应的对象变量来引用对象。本例中，可以通过对 Revenue 对象的操作来影响工作簿 Z:\Finance\Revenue.xls。

在 Office 软件中使用自动方式的实例

本节给出针对 Office 软件使用自动方式的 3 个例子。

从 Excel 工作表向 Word 文档传输数据

本例从一个 Excel 工作表向一个 Word 文档传输数据。

首先，在 Word 工程（包括访问 Excel 代码的工程）中建立一个到 Excel 对象库的引用。例如：

1. 打开或激活 Word，然后打开 Visual Basic 编辑器（按 Alt + F11 键或选择“工具”>“宏”>“Visual Basic 编辑器”）。
2. 在“工程资源管理器”窗口中单击需要添加引用的工程。例如，如果需要将过程添加到 Normal.dot 模板中，就在添加引用之前在“工程资源管理器”中选择“Normal”工程。
3. 选择“工具”>“引用”，显示“引用”对话框。
4. 勾选与 Microsoft Excel Object Library 对应的复选框。该选项的具体名称与所使用的 Excel 的版本有关，例如如果使用的是 Excel 2003，那么选项的名称就是 Microsoft Excel 11.0 Object Library；使用 Excel XP 时，选项的名称为 Microsoft Excel 10.0 Object Library；使用 Excel 2000 时，选项为名称为 Microsoft Excel 9.0 Object Library。
5. 单击“确定”按钮，关闭“引用”对话框。

一旦添加了引用，就可以使用“对象浏览器”浏览 Excel 对象。打开“对象浏览器”（通常可以按 F2 键或“选择视图”>“对象浏览器”）然后在“工程/库”下拉列表中选择“Excel”。对象浏览器将显示 Excel 对象库中的内容，如图 30.1 所示。可以选择一个 Excel 对象，然后单击对象浏览器的“帮助”按钮，打开所选对象的帮助文件。

使用 Visual Basic 编辑器的帮助和自动填写代码的功能，创建如程序清单 30.1 所示的过程。该过程使用 GetObject 函数从 Excel 工作表的单元格中获取数据，并将其插入到当前 Word 文档的当前选区中。

程序清单 30.1

```
1. Sub Return_a_Value_from_Excel()  
2.  
3.     Dim mySpreadsheet As Excel.Workbook  
4.     Dim strSalesTotal As String  
5.  
6.     Set mySpreadsheet = _  
         GetObject("Y:\Users\Corporate\Sales Forecast.xls")  
7.     strSalesTotal = mySpreadsheet.Application.Range("SalesTotal").Value  
8.     Set mySpreadsheet = Nothing  
9.
```

```

10. Selection.TypeText "Current sales total: $" & strSalesTotal & "."
11. Selection.TypeParagraph
12.
13. End Sub

```



图 30.1 一旦加载了 Excel 对象库，就可以从宿主软件中调用 Visual Basic 编辑器，然后在“对象浏览器”中查看它的内容（本例中为 Microsoft Word）

该子过程从一个已经打开的 Excel 工作表中获取一条信息。这是一个简化了的例子，用来示范如何访问另一个软件中的数据。下面是对该子过程的解释。

- ◆ 第 3 行声明对象变量 mySpreadsheet 为 Excel.Workbook 类型。第 4 行声明字符串变量 strSalesTotal。
- ◆ 第 6 行使用 Set 语句和 GetObject 函数让 mySpreadsheet 引用工作表 Y:\Users\Corporate\Sales Forecast.xls。
- ◆ 第 7 行将字符串变量 strSalesTotal 指定为 Excel Application 对象中 Range 对象 Sales Total 的 Value 属性。Sales Total 是一个单元格，而 strSalesTotal 获得了该单元格中的值。
- ◆ 第 8 行指定 mySpreadsheet 对象的值为特殊值 Nothing，用于释放其占用的内存。（由于该过程很快就会结束，所以这条语句在这里并不是十分必要——但是，当不再需要一个对象变量时，释放它所占用的内存是很好的做法。）
- ◆ 第 10 行使用 Word 软件的 Selection 对象中的 TypeText 方法，在当前的选区中输入了文本串和 strSalesTotal。第 11 行使用 TypeParagraph 方法在文本后插入一个段落。第 13 行结束程序。

从 Word 文档向 Excel 工作簿传输信息

前面的例子假定 Excel 已经处于运行状态，如果事实上不是这样，GetObject 函数会返回一个错误，子过程也会停止（除非已经编写了错误处理程序）。要避免这样的错误，需要在访问目标软件之前检查它是否处于运行状态。

程序清单 30.2 进行了这样的检查。同时从 Word 中生成了 Excel 工作簿，输入信息，并且保存结果。

就像前面提到的那样，如果已经在Word工程中添加了对Excel对象库的引用，那么创建这样的程序就会更加简便。（创建引用的方法参见前文。）

程序清单30.2

```
1. Sub List_Page_Counts_in_Excel_Spreadsheet()
2.
3.     Dim intCounter As Integer
4.     Dim strPath As String
5.     Dim strFile As String
6.     Dim docCurDoc As Document,
7.         Dim myXL As Excel.Application
8.         Dim myXLS As Excel.Workbook
9.         Const errExcelNotRunning = 429
10.        Const errDocNotAvailable = 5174
11.
12.    On Error GoTo Handle
13.
14.    Set myXL = GetObject(, "Excel.application")
15.    myXL.Visible = True
16.
17.    Set myXLS = myXL.Workbooks.Add
18.    myXL.ActiveCell = "Current Page Counts"
19.    strPath = "Y:\Sample Documents\Users\Projects\8001"
20.
21.    For intCounter = 1 To 20
22.        strFile = "8001c" & Format(intCounter, "00") & ".doc"
23.        Set docCurDoc = Documents.Open(strPath & "\" & strFile, AddToRecentFiles:=False)
24.        myXL.ActiveCell.Offset(1, 0).Range("A1").Select
25.        myXL.ActiveCell = docCurDoc.Name
26.        myXL.ActiveCell.Offset(0, 1).Range("A1").Select
27.        myXL.ActiveCell = docCurDoc _
28.            .BuiltInDocumentProperties(wdPropertyPages)
29.        myXL.ActiveCell.Offset(0, -1).Range("A1").Select
30.        docCurDoc.Close SaveChanges:=wdDoNotSaveChanges
31.    SkipLoop:
32.    Next intCounter
33.    myXLS.SaveAs strPath & "\" & "8001stats.xls"
34.    myXLS.Close
35.    myXL.Quit
36.    Set myXL = Nothing
37.    Set myXLS = Nothing
38.
39. Handle:
40.     If Err.Number = errExcelNotRunning Then
41.         Set myXL = CreateObject("Excel.Application")
42.         Err.Clear
43.         Resume Next
44.     ElseIf Err.Number = errDocNotAvailable Then
45.         myXL.ActiveCell.Offset(1, 0).Range("A1").Select
46.         myXL.ActiveCell = strFile
47.         myXL.ActiveCell.Offset(0, 1).Range("A1").Select
```

```
48.     myXL.ActiveCell = "Not available"
49.     myXL.ActiveCell.Offset(0, -1).Range("A1").Select
50.     Err.Clear
51.     GoTo SkipLoop
52. Else
53.     Resume Next
54. End If
55.
56. End Sub
```

程序清单 30.2 的解释如下：

- ◆ 第 2 行为空行。第 3 行声明了整型变量 intCounter。第 4 行声明了字符串变量 strPath。第 5 行声明字符串变量 strFile。第 6 行声明了 Document 变量 docCurDoc。第 7 行声明了 Excel. Application 对象变量 myXL。第 8 行声明了 Excel. Workbook 对象变量 myXLS。
- ◆ 第 9 行声明了常量 errExcelNotRunning，将其值指定为 429。第 10 行声明了常量 errDocNotAvailable，将其值指定为 5174。第 11 行为空行。
- ◆ 第 12 行开始错误处理语句，在发生错误时将程序引导至 Handle 标签处。第 13 行为空行。
- ◆ 第 14 行指定 myXL 的值为当前的 Excel 对象，即返回 GetObject 函数的值。如果这时 Excel 不处在运行状态，就发生了错误 429（ActiveX 组件无法创建对象），于是第 40 行的错误处理程序使用 errExcelNotRunning 常量检查错误。如果错误相符，第 41 行用 CreateObject 函数创建一个新对象，并将其指定为 myXL 值。第 42 行用 Err. Clear 语句清除错误，第 43 行为 Resume Next 语句，让 VBA 在出现错误之后继续执行程序。
- ◆ 无论如何，当执行到第 15 行时，myXL 都引用了一个正在运行的 Excel 对象。第 15 行将 myXL 的 Visible 属性设为 True，让它显示在屏幕上。第 16 行是空行。
- ◆ 第 17 行将 myXLS 指定为一个新的工作簿。该工作簿通过使用 myXL 中的 Workbooks 对象的 Add 方法创建。
- ◆ 第 18 行将 myXL 中的当前单元格赋值为文本“Current Page Counts”。第 19 行指定 strPath 的值为包含文档的文件夹路径。第 20 行为空行。
- ◆ 第 21 行到第 31 行是 For…Next 循环，从 1 运行到 20，按排列顺序打开文档文件，将文件名和页数插入到工作表的前两列。以下是具体的过程。
 - ◆ 第 22 行指定 strFile 的值为代表文档名称的字符串。该字符串中包括了文本“8001c”、一个用两位数字表示的 intCounter 值（通过使用 Format 函数返回）以及“. doc”，也就是说，文件名依次为 8001c01. doc 到 8001c20. doc。
 - ◆ 第 23 行打开文档，其路径为 strPath、反斜杠和 strFile，并将其赋值给 docCurDoc 变量。如果文档不可用，则发生错误。后文中会解释如何处理这个错误。
 - ◆ 第 24 行在 Excel 工作表中选择了另一个单元格，根据单元格向下偏移一行，列保持不变（Offset (1, 0)）。
 - ◆ 第 25 行将 docCurDoc 的 Name 属性指定给工作表的当前单元格。
 - ◆ 第 26 行再次使用 Offset 属性，根据当前单元格向右偏移一列。

- ◆ 第 27 行在当前单元格中输入 docCurDoc 的 BuiltInDocumentProperties 集合的 wd-PropertyPages 属性的值。
- ◆ 第 28 行第三次使用 Offset 属性，根据当前单元格向左偏移一列，这个位置将成为下一次循环的起始位置。
- ◆ 第 29 行关闭但不保存 docCurDoc。
- ◆ 第 30 行为 SkipLoop 标签。如果第 23 行中需要打开的文档不可用，就导致了一个错误，程序就转到该标签处。如果第 23 行发生了错误，第 44 行错误处理程序中的 ElseIf Err. Number = errDocNotAvailable 语句接手该错误。第 45 行到第 49 行模仿第 24 行到第 28 行的方法，不同的是在左列中输入 strFile 的值，而不是 docCurDoc 的 Name 属性；在右列中输入“Not available”而不是页数。第 50 行用 Err. Clear 语句清理错误，第 51 行使用 GoTo 语句引导程序回到 SkipLoop 标签处。
- ◆ 第 31 行结束 For …Next 循环。
- ◆ 第 32 行在 strPath 指定的文件夹中用 8001stats.xls 文件名保存 myXLS。第 33 行关闭 myXLS。第 34 行使用 Quit 方法关闭 myXL。
- ◆ 第 36 行和第 37 行将 myXL 和 myXLS 赋值为 Nothing，释放内存。
- ◆ 第 37 行使用 Exit Sub 语句退出过程，避免程序进入错误处理语句。

在 Outlook 信息中添加 PowerPoint 幻灯片

下面给出了一个例子，示范如何在 PowerPoint 和 Outlook 之间进行交流。程序从 Power-Point 运行，返回了一个已存在的 Outlook 进程或者（如果 Outlook 没有处在运行状态）创建一个新的进程。然后程序使用 PowerPoint 送出一条信息，其中给出从演示文稿中提取的详细内容。

程序清单 30.3 给出了相应的代码。有一点需要注意：由于 PowerPoint 并没有包含一个中心宏存储工程，例如 Word 中的 Normal.dot 或者 Excel 中的个人宏工作簿，所以代码必须存储在一个打开的演示文稿中。这个演示文稿可以直接使用电子邮件所涉及的演示文稿，但是更方便的方法是保存一个只包含代码的演示文稿，并在需要在 PowerPoint 中使用代码时打开它。

首先，向目标 PowerPoint 工程（需要向其中添加代码并访问 Outlook 的工程）添加到 Outlook 对象库的引用。例如：

1. 打开或激活 PowerPoint，然后调用 Visual Basic 编辑器（按 Alt+F11 键或选择“工具”>“宏”>“Visual Basic 编辑器”）。
2. 在“工程资源管理器”中单击需要添加引用的工程。例如，如果已经指定了一个代码演示文稿，就在“工程资源管理器”中单击它。
3. 选择“工具”>“引用”，显示“引用”对话框。
4. 勾选与 Microsoft Outlook Object Library 对应的复选框。该选项的具体名称与所使用的 Outlook 的版本有关，例如如果使用的是 Outlook 2003，那么选项的名称就是 Microsoft Outlook 11.0 Object Library；如果使用的是 Outlook XP，名称为 Microsoft Outlook 10.0 Object Library；如果使用的是 Outlook 2000，名称为 Microsoft Outlook 9.0 Object Library。
5. 单击“确定”按钮，关闭“引用”对话框。

程序清单 30.3

```
1. Sub Notify_of_New_Presentation()
2.
3.     Dim myPresentation As Presentation
4.     Dim strPresentationFilename As String
5.     Dim strPresentationTitle As String
6.     Dim strPresentationPresenter As String
7.     Dim myOutlook As Outlook.Application
8.     Dim myMessage As Outlook.MailItem
9.     Const errOutlookNotRunning = 429
10.
11.    On Error GoTo ErrorHandler
12.
13.    Set myPresentation = ActivePresentation
14.    With myPresentation
15.        strPresentationFilename = .FullName
16.        strPresentationTitle = _
17.            .Slides(1).Shapes(1).TextFrame.TextRange.Text
18.        strPresentationPresenter = _
19.            .Slides(1).Shapes(2).TextFrame.TextRange.Text
20.    End With
21.    Set myOutlook = GetObject(, "Outlook.Application")
22.    Set myMessage = myOutlook.CreateItem(ItemType:=olMailItem)
23.    With myMessage
24.        .To = "Review Group"
25.        .CC = "Presentation Archive"
26.        .Subject = "Presentation for review: " & strPresentationTitle
27.        .BodyFormat = olFormatHTML
28.        .Body = "Please review the following presentation:" & _
29.            vbCr & vbCr & "Title: " & strPresentationTitle & vbCr & _
30.            "Presenter: " & strPresentationPresenter & vbCr & vbCr & _
31.            "The presentation is in the file: " & _
32.            strPresentationFilename
33.        .Send
34.    End With
35.    myOutlook.Quit
36.    Exit Sub
37. ErrorHandler:
38.    If Err.Number = errOutlookNotRunning Then
39.        Set myOutlook = CreateObject("Outlook.Application")
40.        Err.Clear
41.        Resume Next
42.    Else
43.        MsgBox Err.Number & vbCrLf & Err.Description, vbOKOnly + _
44.            Critical, "An Error Has Occurred"
45.    End If
46. End Sub
```

下面是对程序清单 30.3 的解释：

- ◆ 第 2 行是空行。第 3 行声明了一个 Presentation 对象变量，名为 myPresentation。第 4 行声明了一个字符串变量，名为 strPresentationFilename，该变量用于存储演示文稿的路径和文件名。第 5 行声明了一个字符串变量，其名为 strPresentationTitle，用于存储演示文稿的标题。第 6 行声明了一个字符串变量，其名为 strPresentationPresenter，用于存储演示文稿的演示者的名字。
- ◆ 第 7 行声明了一个 Outlook. Application 对象变量，其名为 myOutlook，用于代表 Outlook 软件。第 8 行声明了一个 Outlook. MailItem 对象变量，其名为 myMessage，用于代表程序所创建的信息。第 9 行声明了一个常量，其名为 errOutlookNotRunning，并将其赋值为 429，这是一个错误代号，代表在使用 GetObject 函数访问 Outlook 时，Outlook 不处于运行状态。第 10 行是空行。
- ◆ 第 11 行开始程序的错误处理部分，在发生错误时将程序引导至 ErrorHandler 标签处（第 36 行）。第 12 行是空行。
- ◆ 第 13 行将当前演示文稿指定给 myPresentation 对象变量。第 14 行到第 18 行是一个 With 语句块，用于处理 myPresentation。第 15 行将 myPresentation 变量的 Full-Name 属性的值指定给 strPresentationFilename。第 16 行将 strPresentationTitle 赋值 TextRange 对象的 Text 属性，该对象属于第一个 Slide 对象的第一个 Shape 对象中的 TextFrame 对象，换句话说，该文本属于演示文稿中第一张幻灯片上的第一个占位符。类似地，第 17 行指定 strPresentationPresenter 为第一张幻灯片中的第二个占位符。（这个演示文稿中的第一页包括了两个占位符，第一个是演示文稿的标题，第二个是演示者的名字。）
- ◆ 第 20 行指定 myOutlook 为 GetObject 函数返回的 Outlook 进程。如果 Outlook 不处于运行状态，返回错误 429（ActiveX 组件无法创建对象），于是第 37 行的错误处理程序使用 errOutlookNotRunning 常量检查错误。如果错误相符，第 38 行用 CreateObject 函数创建一个新对象，并将其指定为 myOutlook 的值。第 39 行用 Err. Clear 语句清除错误，第 40 行为 Resume Next 语句，让 VBA 从出错语句后继续执行程序。
- ◆ 第 21 行使用 Outlook. Application 对象（由 myOutlook 表示）的 CreateItem 方法创建一个新邮件，并将其指定为 myMessage 的值。第 22 行到第 29 行包括了一个 With 语句块，处理 myMessage。第 23 行设定 To 属性，指定收件人。第 24 行通过设定 CC 属性添加一个 CC 收件人。第 25 行输入 Subject 属性的文本。第 26 行指定信息使用 HTML 格式 (.BodyFormat = olFormatHTML)。第 27 行使用 Body 属性输入信息的正文。第 28 行使用 Send 方法发送信息。第 30 行为空行。
- ◆ 第 31 行使用 Quit 方法关闭 myOutlook。第 32 行为空行。
- ◆ 第 33 行将 myMessage 设为 Nothing，释放其占用的内存。类似地，第 34 行将 myOutlook 设为 Nothing。第 35 行退出过程。
- ◆ 就像在前面讨论的那样，错误处理程序的主要功能是在 Outlook 程序没有处于运行状态下打开一个 Outlook 进程。如果发生了错误 429 以外的错误，程序运行第 41 行的 Else 语句，第 42 行显示信息框，给出错误的代码和描述。

使用 Shell 函数运行软件

不是用 CreateObject 函数打开一个软件并向其建立引用而是使用 Shell 函数运行另一个软件。Shell 可以运行任何可以执行的程序，其语法很直接，如下所示：

```
Shell(pathname[,windowstyle])
```

这里 pathname 是需要用 Shell 函数运行的程序的名称，包括了路径和所有必要的命令行或参数。如果程序保存在计算机中，可以不用为其指定完整的路径，通常这样做是最安全的。

注意：Shell 也可以运行扩展名已知的文件。例如，Shell " testfile.txt" 通常会调用记事本，因为通常 .txt 文件与记事本之间建立了连接。如果 Shell 无法找到指定的软件或文件，则返回运行错误。

windowstyle 是整型子类型中的可选不定型参数，用于指定运行程序的窗口的类型。表 30.1 列出了 windowstyle 可用的常量和值。

表 30.1 windowstyle 参数中使用的常量和值

常量	值	窗口风格
vbHide	0	最小化并隐藏，有焦点
vbNormalFocus	1	正常（“还原”），有焦点
vbMinimizedFocus	2	最小化，但有焦点（默认值）
vbMaximizedFocus	3	最大化，有焦点
vbNormalNoFocus	4	正常（“还原”），但无焦点
vbMinimizedNoFocus	6	最小化，无焦点

使用 Sleep 函数以避免 Shell 函数的不同步问题

Shell 函数与其他的程序不同步运行，而不是同步运行。因此当 VBA 执行 Shell 语句时，可以将该语句视为将要执行的命令，但完成这一命令并不是执行下一条语句的必要条件。

而如果紧随其后的语句依赖于 Shell 语句执行后的结果，这样的不同步性就很可能导致程序出错。想要解决这类问题，比较原始但十分有效的方法是在下一个动作执行之前给 Shell 函数额外的时间来完成。例如，可以将 Shell 函数的执行过程提前，而不是刚好放在依赖于它的命令之前。更好的解决方法是使用一个 API 指令（比如 Sleep）来延迟后续语句的执行，这样 Shell 函数的操作就可以完成了。将下面的声明添加到代码页最开始的声明部分：

```
Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
```

然后在适当的时候调用 Sleep 函数，指定让代码等待的毫秒数。下面的语句用 Sleep 将程序延迟了 2 秒：

```
Sleep (2000)
```

返回运行软件的任务 ID

Shell 函数会返回一个独一无二的任务确认号码（任务 ID），用来识别刚刚启动的软件。可以使用这个任务 ID 来快速访问软件，而不用列出所有正在运行的软件。

要返回软件的任务 ID，可以在使用 Shell 语句时将任务 ID 赋值给一个变量。下面的例子运行了软件 Lotus 1-2-3，并将任务 ID 赋值给 myTaskID 变量：

```
Dim myTaskID As Long  
myTaskID = Shell("c:\lotus\123\programs\123w.exe")
```

激活软件窗口

要激活一个软件窗口，可以使用 AppActivate 语句。AppActivate 能够激活一个软件的窗口，而不会将其最大化或还原——因此，如果软件被最小化，焦点会切换到它的任务栏上，但是不会显示软件的窗口。（要最大化、最小化或还原一个软件的窗口，可以使用 Shell 语句，就像本章前面所提到的那样。）

AppActivate 的语法如下：

```
AppActivate title[, wait]
```

这里，title 是必选字符串，指明要激活的软件窗口在任务栏上所包含的标题。例如，要激活 Excel，就要将 title 指定为 Microsoft Excel，因为 Excel 在任务栏上显示的是“Microsoft Excel”：

```
AppActivate "Microsoft Excel"
```

提示：如果有两个或更多的 Excel 进程处于运行中，VBA 会任意挑选一个。要避免这样随机选择的情况发生，可以将 title 指定为标题栏上的完整标题——例如：“Microsoft Excel - Book2”。

也可以使用 Shell 函数返回的软件的任务 ID 来激活软件。使用任务 ID 就可以完全排除混淆同一个软件的多个进程的可能性。

wait 是可选逻辑值，用来指定是否要在软件调用其他软件之前获得焦点。wait 的默认设置为 False，即不要在软件调用其他软件之前获得焦点。也可以将 wait 设为 True，让软件持续等待，直到获得焦点时再调用。为了避免中断当前拥有焦点的重要程序，最好将 wait 设为 True。

例如，下面的语句激活了 PowerPoint：

```
AppActivate "Microsoft PowerPoint"
```

下面的语句运行了 Lotus 1-2-3 软件，将其任务 ID 指定给一个变量，然后使用该变量激活 1-2-3：

```
Dim myTaskID As Long  
myTaskID = Shell("c:\lotus\123\programs\123w.exe")  
AppActivate MyTaskID
```

使用数据对象存储和获取信息

数据对象在逻辑上附着于 Microsoft 窗体对象模型中的 UserForm 对象，但可以单独使用数据对象，而不使用窗体。在 VBA 中，数据对象用 DataObject 对象表示，用于存储数据。每一个数据对象都可以包含多条文本信息，而每一条信息的格式必须各不相同。可以创建并使用多个数据变量存储多条相同格式的信息，或者采取某些手段让 VBA 认为数据的格式各不相同，而事实上是相同的。

根据本书到目前为止所介绍的，可以使用 VBA 将信息存储在其他地方，但数据变量的作用在于向剪贴板写入数据和从中获取数据。

剪贴板可容纳一个文本对象和一个其他格式的对象，比如一个图形对象。如果向其中复制了另一个文本对象，那么剪贴板就会覆盖先前的文本对象，而其中的图形对象则不会受到影响。类似地，如果复制了一个图形对象，剪贴板会覆盖先前存储的图形对象（或者说覆盖任何非文本格式的对象），但是存储在其中的任何文本对象将不受影响。

数据对象的工作方式和剪贴板十分相似，但有所不同，它不能存储图形信息，但是可以存储多条文本信息，每条文本信息所定义的格式必须不同。（对于数据信息，不要求格式必须不同。）

创建数据对象

要创建数据对象，可以声明一个 DataObject 类型的对象，然后使用 Set 语句为其指定一个新的 DataObject 对象。例如下面的语句声明了一个 DataObject 变量，其名为 myDObj，并为其指定了一个新的 DataObject 对象：

```
Dim myDObj As DataObject  
Set myDObj = New DataObject
```

在数据对象中存储信息

要在数据对象中存储数据，可以使用 SetText 方法，该方法的语法如下：

```
object.SetText(StoreData [, format])
```

该语法的各组成部分如下：

- ◆ Object 是必选参数，指定一个有效的对象。
- ◆ StoreData 是必选参数，指定要存储在数据对象中的对象。
- ◆ format 是可选参数，可以为字符型或整型，指定 StoreData 中信息的格式。值 1 表示文本格式，其他数值或者字符串表示用户自定义的格式。

例如，下面的语句在名为 myDObj 的数据对象中存储了字符串“Sample text string”：

```
myDObj.SetText "Sample text string"
```

下面的语句在名为 myDObj 的数据对象中储存了字符串“Sample formatted text string”，其格式由用户自定义，名为 myFormat：

```
myDObj.SetText "Sample formatted text string", "myFormat"
```

一旦定义了用户格式并把它存储在数据对象中，就可以通过指定格式来访问该格式的变量。在本例中，事实上并没有涉及某个格式——代码只是简单地使用了 format 参数在数据对象中创建并定义了一个不同的数据存区，这样新的字符串就不会覆盖之前已有的字符串。

从数据对象中返回信息

要从数据对象中返回信息，可以使用 DataObject 对象的 GetText 方法。GetText 方法的语法如下：

```
object.GetText([format])
```

该语法的各组成部分如下：

- ◆ object 是必选参数，指定有效的对象。
- ◆ format 是可选参数，可以为字符型或整型，指定需要获取的数据的类型。

例如，下面的语句显示了一个信息框，其包括名为 myDObj 的数据对象中所储存的普通文本字符串：

```
MsgBox myDObj.GetText
```

下面的语句将字符串变量 strTemp 的值指定为名为 myObj 的数据对象中所储存的 myFormat 格式的数据：

```
strTemp = myDObj.GetText("myFormat")
```

向剪贴板传送信息

要将数据对象中的文本信息传送到剪贴版中，可以对相应的 DataObject 对象使用 PutInClipboard 方法。例如，下面语句创建了一个新的数据对象 myDO，将其赋值为文本“Atlanta Industrial Pharmaceuticals”，然后将该数据传送到剪贴板：

```
Dim myDO As DataObject  
Set myDO = New DataObject  
myDO.SetText "Atlanta Industrial Pharmaceuticals"  
myDO.PutInClipboard
```

从剪贴板中返回信息到数据对象

要从剪贴板中返回文本并存储在数据对象中，可以使用 DataObject 对象的 GetFromClipboard 方法。例如下面的语句创建了一个数据对象 aDO，并将其指定为剪贴板中的信息：

```
Dim aDO As DataObject  
Set aDO = New DataObject  
aDO.GetFromClipboard
```

要从剪贴板中获取特定格式的信息，并将其存储在数据对象中，可以使用 DataObject 对象的 GetFormat 方法。

确认数据对象中是否包含某特定格式

要确认数据对象中是否包含某个特定的格式，可以使用 DataObject 对象的 GetFormat

方法。GetFormat 方法的语法如下：

```
object.GetFormat(format)
```

该语法的各组成部分如下：

- ◆ object 是必选参数，返回有效的 DataObject 对象。

- ◆ format 可以为整型或字符型，指定所要查找的格式。如果 DataObject 包括了指定的格式，GetFormat 返回 True；如果不包括，GetFormat 返回 False。

例如，下面的语句检查名为 myDO 的 DataObject 对象中是否存在 myHTML 格式，如果存在，则在该格式的存储区中存储字符串 strHTMLText。

```
If myDO.GetFormat("myHTML") = True Then_
    strHTMLText = myDO.GetText(Format:="myHTML")
```

通过 DDE 进行数据交换

如果需要与之进行数据交换的软件不支持自动方式，那么可以尝试动态数据交换（DDE）。DDE 是一个协议，能够在两个软件之间建立通道，通过这个通道，它们可以自动交换信息。使用 DDE 需要很多技巧，但通常十分可靠。

注意：并不是所有的软件都支持 DDE，在 Office 软件中，Word、Excel 和 Access 支持 DDE，但 PowerPoint 和 Outlook 并不支持。

典型的 DDE 数据交换可以包括下面的操作：

- ◆ 使用 DDEInitiate 方法打开 DDE 连接，并建立连接运行的通道。

- ◆ 使用 DDERequest 方法从另一个软件中返回文本，或使用 DDEPoke 方法向另一软件传送文本。

- ◆ 使用 DDEExecute 方法在另一个软件中执行命令。

- ◆ 使用 DDETernate 方法关闭当前的 DDE 通道或使用 DDETernateAll 方法关闭所有的 DDE 通道。

使用 DDEInitiate 方法打开 DDE 连接

要打开 DDE 连接，可以使用 DDEInitiate 方法。DDEInitiate 方法的语法如下：

```
expression.DDEInitiate(App, Topic)
```

该语法的各组成部分如下：

- ◆ expression 是可选表达式，返回一个 Application 对象。

- ◆ App 是必选的字符串参数，指定将要连接到的软件的名称。

- ◆ Topic 是必选的字符串参数，指定软件中的 DDE 主题（比如一个打开的文件）。要了解软件中有哪些可用的主题，可以向软件的 System 对象发送一个 DDE 请求（使用 DDERequest 方法，这在后面的章节中将会介绍）。

DDEInitiate 返回已建立的 DDE 通道的号码，这样就可以在随后的 DDE 调用中使用这个号码。

例如，下面的语句声明了长整型变量 lngDDEChannel1，并将其指定为一个连接到 Excel 工作簿“Sales Results.xls”的通道：

```
Dim lngDDEChannel1 As Long
lngDDEChannel1 = DDEInitiate("Excel", "Sales Results.xls")
```

使用 DDERequest 方法返回另一个软件中的文本

要从另一个软件返回字符串文本，可以使用 DDERequest 方法。DDERequest 方法的语法如下：

```
expression.DDERequest(Channel, Item)
```

该语法的各组成部分如下：

- ◆ expression 是可选表达式，返回一个 Application 对象。
- ◆ Channel 是必选的长整型参数，指定所要使用的 DDE 通道。
- ◆ Item 是必选的字符串参数，指定需要进行查询的项。

要获得 DDE 中可用的主题列表，可以通过查询 System 主题下的 Topic 项得到。例如，下面的语句建立了到 FrontPage 的 DDE 通道（使用 DDEInitiate 方法实现），然后返回 DDE 主题的列表，并将该列表赋值给字符串变量 strDDETopics：

```
Dim lngDDE1 As Long
Dim strDDETopics As String
lngDDE1 = DDEInitiate(App:="FrontPage", Topic:="System")
strDDETopics = DDERequest(Channel:=lngDDE1, Item:="Topics")
```

下面的语句建立了到 Excel 工作簿“Sales Results.xls”的 DDE 通道，并将单元格 C7 (R7C3) 中的值赋值给字符串变量 strResult：

```
Dim lngDDEChannel1 As Long, strResult As String
lngDDEChannel1 = DDEInitiate("Excel", "Sales Results.xls")
strResult = DDERequest(lngDDEChannel1, "R7C3")
```

使用 DDEPoke 方法向另一个软件发送文本

要向另一个软件发送文本，可以使用 DDEPoke 方法。DDEPoke 方法的语法如下：

```
expression.DDEPoke(Channel, Item, Data)
```

该语法的各组成部分如下：

- ◆ expression 是可选的表达式，返回一个 Application 对象。
- ◆ Channel 是必选的长整型参数，指定所要使用的 DDE 通道。
- ◆ Item 是必选的字符串参数，指定接收数据的项。
- ◆ Data 是必选的字符串参数，指定所要发送的数据。

继续前面的例子，下面的语句使用了 DDEPoke 方法。如果工作表中 C7 单元格中的值小于 2000，则将数据“2000”写入该单元格。

```
Dim lngDDEChannel1 As Long, strResult As String
lngDDEChannel1 = DDEInitiate("Excel", "Sales Results.xls")
strResult = DDERequest(lngDDEChannel1, "R7C3")
```

```
If Val(strResult) < 2000 Then
    DDEPoke Channel:=1ngDDEChannel1, Item:="R7C3", Data:="2000"
End If
```

使用 DDEExecute 方法在另一个软件中执行命令

要在另一个软件中执行命令，可以使用 DDEExecute 方法。DDEExecute 方法的语法如下：

```
expression.DDEExecute(Channel, Command)
```

该语法的各组成部分如下：

- ◆ expression 是可选表达式，返回一个 Application 对象。
- ◆ Channel 是必选的长整型参数，指定所要使用的 DDE 通道。
- ◆ Command 是必选的字符串参数，指定要执行的命令。

例如，下面的语句建立了到 Excel 的 DDE 通道，并执行了 Close 命令以关闭当前工作簿：

```
Dim 1ngMyChannel
1ngMyChannel = DDEInitiate(App:="Excel", Topic:="System")
DDEExecute 1ngMyChannel, Command:=" [Close]"
```

使用 DDETerninate 方法关闭 DDE 通道

当使用完 DDE 连接时，可以使用 DDETerninate 方法关闭之前打开的 DDE 通道。DDETerninate 方法的语法如下：

```
expression.DDETerninate(Channel)
```

该语法的各组成部分如下：

- ◆ expression 是可选表达式，返回一个 Application 对象。
- ◆ Channel 是必选的长整型参数，指定所要关闭的 DDE 通道。

下面的语句继续前面的例子，关闭之前打开的 DDE 通道：

```
Dim 1ngMyChannel
1ngMyChannel = DDEInitiate(App:="Excel", Topic:="System")
DDEExecute 1ngMyChannel, Command:=" [Close]"
DDETerninate 1ngMyChannel
```

使用 DDETerninateAll 方法关闭所有打开的 DDE 通道

要关闭所有打开的 DDE 通道，可以使用 DDETerninateAll 方法：

```
DDETerninateAll
```

由于 VBA 不能在程序结束时自动关闭 DDE 通道，因此最好使用 DDETerninateAll 方法确保所有通道都被关闭。

通过 SendKeys 语句进行数据交换

通过 SendKeys 语句与另一个软件建立连接，是一种较为基本且有一定限制的方法。当目标软件既不支持自动方式也不支持 DDE 时，就可以使用 SendKeys 语句。

SendKeys 语句能够向目标软件发送键击。例如，要在记事本中创建一个新文件，可以使用 SendKeys 发送键击 Alt + F 和 N（执行命令“文件”>“新建”），于是记事本会做出相应的反应，这与人工操作的结果是相同的。

SendKeys 语句只用于当前正在运行的 Windows 软件：不能使用 SendKeys 语句打开另一个软件（因此，必须使用 Shell 函数，具体的方法可以参考前面的章节），也不能使用 SendKeys 语句与在 Windows 的 DOS 虚拟机中运行的 DOS 程序交换数据。

SendKeys 语句的语法如下：

SendKeys string[, wait]

这里，string 是必选的字符表达式，指定要向目标软件发送的键击。wait 是可选的逻辑值，决定等待另一个软件完成键击所要求的操作（True）还是立刻回到发送键击的程序（False，默认设置）。

通常，string 会包括一系列键击（而不是单个的键击）。键盘上的所有字母键都由这个字母本身代表：要发送字母 H，可以将 string 指定为 “H”，要发送单词 “Hello”，可以将 string 指定为 “Hello”。表示动作和修改操作的按键，SendKeys 语句会用大括号括起的关键词表示，参见表 30.2。

表 30.2 SendKeys 中表示动作和修改操作的关键词

按键	代码
,	{DOWN}
F1	{LEFT}
>	{RIGHT}
.	{UP}
Backspace	{BACKSPACE}, {BS} 或 {BKSP}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Delete	{DELETE} 或 {DEL}
End	{END}
Enter	{ENTER}
Esc	{ESC}
F1, F2, 等等	{F1}, {F2}, 等等（直到 {F16}）
Help	{HELP}
Home	{HOME}
Insert	{INSERT} 或 {INS}
Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Print Screen	{PRTSC}
Scroll Lock	{SCROLLLOCK}
Tab	{TAB}

要发送功能键，可以使用表 30.3 中的符号。

表 30.3 SendKeys 中表示功能键的符号

按键	代码
Shift	+
Ctrl	-
Alt	%

SendKeys 语句能够自动将功能键后的键击附加在功能键上。例如，要发送 Ctrl + O 的键击，可以指定键击为~O，于是 SendKeys 会自动把 O 附加在 Ctrl 的键击上；O 之后的键击会被认为是独立的键击。如果要将多个键击附加在功能键上，可以用括号将功能键之后的键击括起。例如要发送 Alt+F、Alt+I、Alt+I，可以使用代码% (FII) 而不是%FII。

在 SendKeys 中，加号 (+)、插入号 (^)、百分号 (%) 以及括号 () 有着特殊的含义；鼻音化符号 (~) 也有特殊的处理方式。要这些字符本身来代表键击，可以将它们放在大括号中：使用 {+} 发送一个普通的加号，使用 {^} 发送普通的插入号，使用 {%)} 发送百分号，使用 {~} 发送鼻音化符号，使用 {(0)} 发送圆括号。类似地，方括号（在某些软件的 DDE 中，方括号有特殊的含义）也必须放在大括号里；大括号本身也必须放在大括号内。

使用 SendKeys 语句并不是像上面这些细节所表现的那样复杂——因此不用担心。还有一个提示必须了解：要重复一个键，可以在大括号中输入这个键和需要重复的次数。例如，要发送 5 个↑的键击，可以将代码写成 {Up 5}；要发送 10 个 0，可以写成 {0 10}。

程序清单 30.4 给出了一个例子，其中程序会先打开记事本，然后使用 SendKeys 语句向其中添加日志信息。

警告：由于 SendKeys 语句需要激活目标软件，因此不能进入 Visual Basic 编码器的代码中——Visual Basic 编码器会在不适当的时候获取焦点，从而使键击被发送给 Visual Basic 编码器自身，而不是发送给目标软件。为了避免这样的情况发生，必须从 Visual Basic 编码器或宿主软件中直接运行过程。

程序清单 30.4

```

1. Sub Send_to_NotePad()
2.     Dim strLogDate As String
3.     Dim strSaveLog As String
4.     Dim strMsg As String
5.     Dim appNotePad As Variant
6.     strMsg = "Sample log text here."
7.     strLogDate = Month(Now) & "-" & Day(Now) & "-" & Year(Now)
8.     strSaveLog = "Log file for " & strLogDate & ".txt"
9.     appNotePad = Shell("notePad.exe", vbNormalFocus)
10.    AppActivate appNotePad
11.    SendKeys strMsg & "%FS" & strSaveLog & "{Enter}" & "%{F4}", True
12. End Sub

```

下面是对程序清单 30.4 的解释：

- ◆ Send_to_NotePad 过程首先声明了三个字符串变量（第 2 行、第 3 行和第 4

行)——strLogDate、strSaveLog 和 strMsg——以及一个不定型变量(第 5 行)appNotepad。

- ◆ 第 6 行将 strMsg 赋值为一个样本字符串文本。
- ◆ 第 7 行用 Day、Month 和 Year 的 Now 值(能够返回当前的日期和时间)指定当前的日期，并将其赋值给 strLogDate。例如，如果当天的日期为 2006 年 7 月 11 日，那么 Month(Now) 返回 7，Day(Now) 返回 11，Year(Now) 返回 2006，strLogDate 的值为 7-11-2006。
- ◆ 第 8 行将 strSaveLog 字符串变量(用于提供日志文件的文件名)赋值为描述文件的文本、strLogDate 字符串变量和.txt 扩展名(接着上面的例子，日志文件为 7-11-2006.txt)。
- ◆ 在第 9 行，程序进入正题，使用 Shell 语句在“正常”(不是最大化也不是最小化)窗口中运行记事本(窗口有焦点)，将记事本进程的任务 ID 保存在变量 appNotepad 中。
- ◆ 第 10 行使用 AppActivate 语句激活记事本。
- ◆ 第 11 行使用 SendKeys 语句向记事本发送下面的键击。
 - ◆ 字符串变量 strMsg 中所包含的信息。
 - ◆ Alt+F 键击(打开“文件”菜单)，接下来是选择菜单上的“保存”项目的按键，通过这个按键，可以显示“另存为”对话框，并且选定了“文件名”文本框。
 - ◆ strSaveLog 字符串变量中所保存的内容，用于在“文件名”文本框中输入信息。
 - ◆ Enter 键，用来选择“另存为”对话框中的“保存”按钮。
 - ◆ Alt+F4 键击，退出记事本。
- ◆ 第 12 行结束过程。

当运行这个过程时(再次提醒，直接运行过程，而不要进入过程的代码)会产生下面的效果：

1. “记事本”窗口弹出。
2. Msg 字符串中的内容出现在“记事本”窗口中。
3. “另存为”对话框自动打开，在“文件名”文本框中输入文件名，然后“另存为”对话框自动关闭。
4. “记事本”窗口关闭。