

作为后一种技术的简单例子，考虑图 15.2 所示的对话框。该对话框显示时，只有上面部分能看到；单击 More 按钮时，底下一半也显示出来。程序清单 15.1 包含了该对话框幕后的代码。

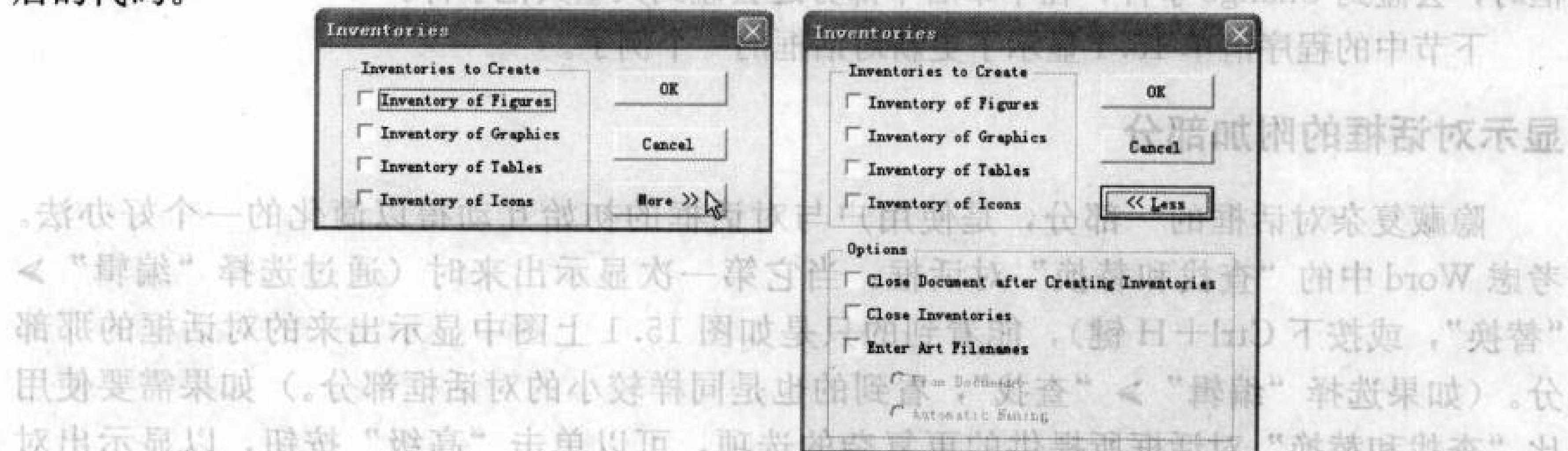


图 15.2 Inventories 对话框的上部（左图）提供了最常用的选项。单击 More 按钮，对话框其他部分（右图）也显示出来，它包含有使用较少的控件

程序清单 15.1

```

1. Private Sub UserForm_Initialize()
2.     frmInventories.Height = 120
3. End Sub
4.
5. Private Sub cmdMore_Click()
6.     If cmdMore.Caption = "More >>" Then
7.         cmdMore.Caption = "<< Less"
8.         cmdMore.Accelerator = "L"
9.         frmInventories.Height = 240
10.        fraOptions.Enabled = True
11.    Else
12.        frmInventories.Height = 120
13.        cmdMore.Caption = "More >>"
14.        cmdMore.Accelerator = "M"
15.    End If
16. End Sub
17.
18. Private Sub chkArtNames_Click()
19.     If chkArtNames = True Then
20.         optFromDocument.Enabled = True
21.         optFromDocument = True
22.         optAutoNames.Enabled = True
23.     Else
24.         optFromDocument.Enabled = False
25.         optFromDocument = False
26.         optAutoNames.Enabled = False
27.         optAutoNames = False
28.     End If
29. End Sub
30.
31. Private Sub cmdOK_Click()
32.     frmInventories.Hide
33.     Unload frmInventories
34.     'create inventories here
35. End Sub

```

```
36.    Private Sub cmdMore_Click()
37.      If cmdMore.Caption = "More" Then
38.        cmdMore.Caption = "Less"
39.      End Sub
```

程序清单 15.1 包含有五个短过程，它们控制对话框的行为：

UserForm _ Initialize 在对话框显示前，对其初始化。

cmdMore _ Click 在 cmdMore 按钮被选中时运行。这个按钮的标题是 More，此时对话框只有上一半显示出来。当对话框完全显示出来时，标题变为 Less。

chkArtNames _ Click 在 Enter Art Filenames 复选框被选中时运行。

cmdOK _ Click 在单击 OK（确定）按钮时运行。

cmdCancel _ Click 在单击 Cancel（取消）按钮时运行。

下面说明代码中有何事发生：

UserForm _ Initialize 过程将 frmInventories 用户窗体的 Height 属性设置为 120，它足以显示对话框的上半部。（为了确定对话框的合适高度，将它拖曳至看来合适的高度，再注意查看“属性”窗口中的 Height 属性。）这一过程只有在设计阶段把用户窗体设置到它的完全高度时才是必要的：如果设计阶段把用户窗体设置到 120 的高度，可以避免使用 UserForm _ Initialize 过程。但是，对于有三种或三种以上不同尺寸的用户窗体——或有两种不同尺寸的用户窗体，在运行阶段要选择哪一种，取决于环境条件——必须要使用 UserForm _ Initialize 过程。

cmdMore _ Click 过程一开始，先在行 6 里对 cmdMore 命令按钮的 Caption 属性是否为 More>进行检查。如果是，说明只显示了对话框的上半部。然后，行 7 把 cmdMore 命令按钮的 Caption 属性设置为 <<Less，这样，这个按钮将用于必要时再隐藏对话框的下半部分。行 8 将 cmdMore 命令按钮的 Accelerator 属性设置为 L（使 Less 中的 L 成为该按钮的加速键）。行 9 将 frmInventories 的 Height 属性设置为 240，这是显示对话框的全部内容所需要的高度。行 10 让 fraOptions 框架有效（该框架在对话框内被标识为 Options 选项，并且使它在用户窗体内无效，optFromDocument 选项按钮和 optAutoNames 选项按钮也是如此），使得该框架及其包含的各控件能为用户所用。

注意：检查 cmdMore 按钮的 Caption 属性是检查对话框当前状态的有效方法，但不能算是最巧妙的方法。更好的方法是保留一个内部状态变量，在其中存入对话框是以完全状态显示或是以部分状态显示的信息。使用这个内部状态变量还有额外的好处，那就是如果对话框要进行区域化，要适应另一个语言场合，不必为此去“打补丁”。

如果行 6 中的条件为 False，执行从行 6 转到行 11 中的 Else 语句。这说明 cmdMore 按钮的 Caption 属性已经设置成了 <<Less，即对话框已经扩展了，单击 <<Less 按钮可以使对话框回缩。行 12 将用户窗体的 Height 属性设回到 120，这样可以来隐藏对话框的下半部。行 13 将 cmdMore 命令按钮的 Caption 属性恢复为 More>。行 14 将 cmdMove 命令按钮的 Accelerator 属性设回为 M。行 16 结束 cmdMore _ Click 过程。

当 Enter Art Filenames 复选框被选中时，运行 chkArtNames _ Click 过程（行 18 至行 29）。这个过程根据情况使复选框下面的选项按钮有效或无效。行 19 查看 chkArtNames 复

选框是否被选中。如果被选中，运行行 20 至行 22 中的各语句。行 20 将 optFromDocument 选项按钮（在对话框中被标识为 From Document）的 Enabled 属性设置为 True，使其可用，行 21 选择这个选项按钮作为默认选择。行 22 使 optAutoNames 有效，该选项按钮在对话框中被标识为 Automatic Naming。

如果 chkArtNames 复选框未被选中，执行转到行 23 的 Else 语句，它指引执行行 24。行 24 将 optFromDocument 选项按钮的 Enabled 属性设置为 False，使其无效。行 25 使这个按钮脱离选择（不管它是否已被选中）。行 26 使 optAutoNames 选项按钮无效，行 27 使其脱离选择（同样，不管它是否已被选中）。行 28 的 End If 语句结束这个 If 语句，而行 29 结束这一过程。

行 31 是 cmdOK_Click 过程的开始，OK 按钮被单击时运行此过程。行 32 隐藏 Inventories 对话框，行 33 将该对话框从存储器中卸出。行 34 是注释行，它指出：生成 Inventories 的指令在此处出现。

cmdCancel_Click 过程只含有一条 End 语句，它在用户单击 Cancel 按钮时终止过程的执行。

在对话框中跟踪过程

对话框的复杂性的下一体现是使用它来跟踪一个过程的不同阶段，并告诉用户使过程如何继续下去。

看一看图 15.3 所示的 Create New Employee Web Page 对话框。这个对话框引导用户通过一个四阶段的过程来生成一个新雇员网页。第一步是鉴别一下值得有此荣幸的雇员，这就要使用 Step 1 框架中的下拉列表或者 Select Other Employee 命令按钮。第二步是输入对该雇员进行适当介绍和赞誉的文本。第三步是把该雇员那张最招人爱（也可能是最不招人爱）的照片放入网页中。第四步是把这个网页保存到公司内部网的一个文件夹中去。

当用户第一次显示 Create New Employee Web Page 对话框时，看到的是图 15.3 所示的对话框版本，其中 Step 2、Step 3 和 Step 4 均未生效，Step 1 的指令显示在顶部的 Instructions 框内。当用户遵循此指令，使用组合框下拉列表或 Select Other Employee 命令按钮来选择雇员时，附属于该组合框下拉列表或该命令按钮的代码使 Step 2 框架有效，使它的文本框可为用户使用，见图 15.4。此图后面是对于 cmbSelectEmployee 组合框的 Change 事件的代码；对于 cmdSelectOtherEmployee 命令按钮的 Click 事件的代码与此类似，但要复杂一些。

```
Private Sub cmbSelectEmployee_Change()
    lbEmployeeName = cmbSelectEmployee.Text
    fraStep2.Enabled = True
    lbInstructions = "Enter text in the Step 2 text box. " & _
        "For example, you might include brief biographical " & _
        "information on the employee, details of their position, " & _
        "or your hopes for their contribution to the company."
    cmdClearEmployeeName.Enabled = True
End Sub
```

注意：同某些其他命令按钮一样，Create New Employee Web Page 对话框中的 Select Other Employee 按钮上有一个省略号（...）。按照 Windows 的规定，省略号是指出：对话框中的选择（这里是命令按钮，但通常是菜单项）结果正在显示，但并不是正在立即采取某一行动。

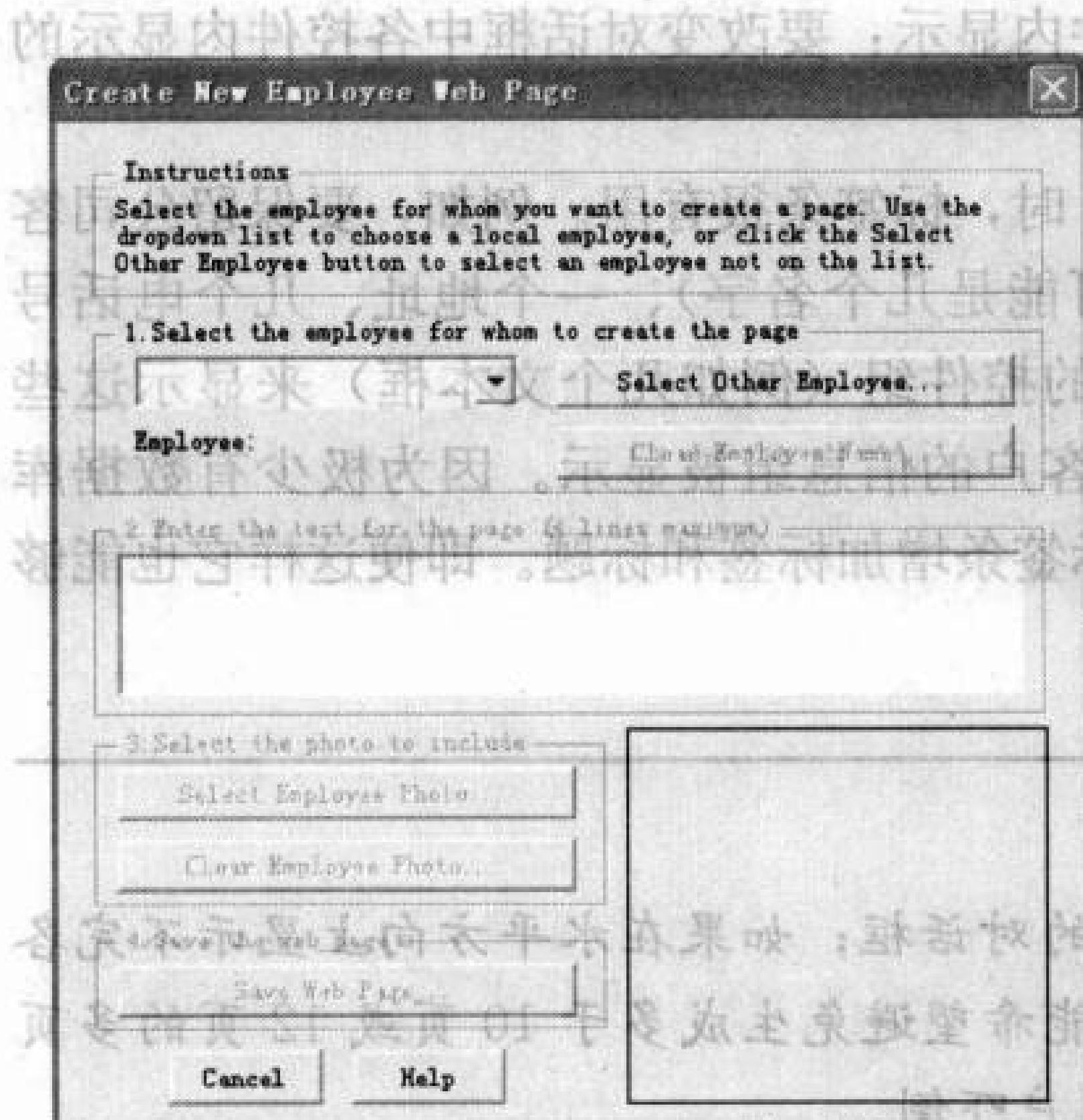


图 15.3 Create New Employee Web Page 对话框向用户提供指令。用户在

过程中工作时，对话框也在更新

用户完成对话框中的 Step 1 之后，有下述变化产生：

- ◆ 对话框顶部 Instructions 框内标签的文本有了变化，包含了有关该过程的 Step 2 的信息。
- ◆ 用户选出的雇员的姓名列在 Step 1 框架内 Employee 标签旁边。
- ◆ 用于 Step 2 的框架变为有效（该框架所包含的文本框同时有效）。

使用多页对话框和 TabStrip 控件

VBA 拥有 MultiPage 控件（可用来生成多页对话框）和 TabStrip 控件（可用来生成由标签条驱动的对话框）。很多人肯定都用过多页对话框（如有怀疑，在任一种 Office 应用程序中或在 Visual Basic 编辑器中选择“工具”>“选项”，就能看到例子）。通过单击页面顶部的标签，可以访问任一页面（一次访问一页）。每个页面含有不同的控件组合，并有适应于各控件的不同的布局。

注意：标签是突显于页面顶部的小东西，不是指整个页面。很多人拿标签来称呼页面，这是因为单击标签就可以访问页面。这里“标签”只用来指明页面上的标签部件，而用“页面”来指整个页。

多页对话框的优点是把很多信息汇集进一个对话框，但又不让这些信息占满整个屏幕而使人看不过来。必须把这些信息分成若干个分离的相关信息组，然后放入各页面。每个页面可以（也应该）有不同的控制各分离项目行为的控件的布局；各页面通常按主题来划分。同样，Office 应用程序和 Visual Basic 编辑器中的“选项”对话框提供了这种例子。

使用标签条的对话框和多页对话框不同，前者含有一个 TabStrip 控件，该控件又包含多个标签，而不是多个页面。不管选择标签条上的哪一个标签，标签条之外的对话框的其余部分保持相同。这意味着，这种对话框只有一种布局，所以各控件都不改变。标签条起一个

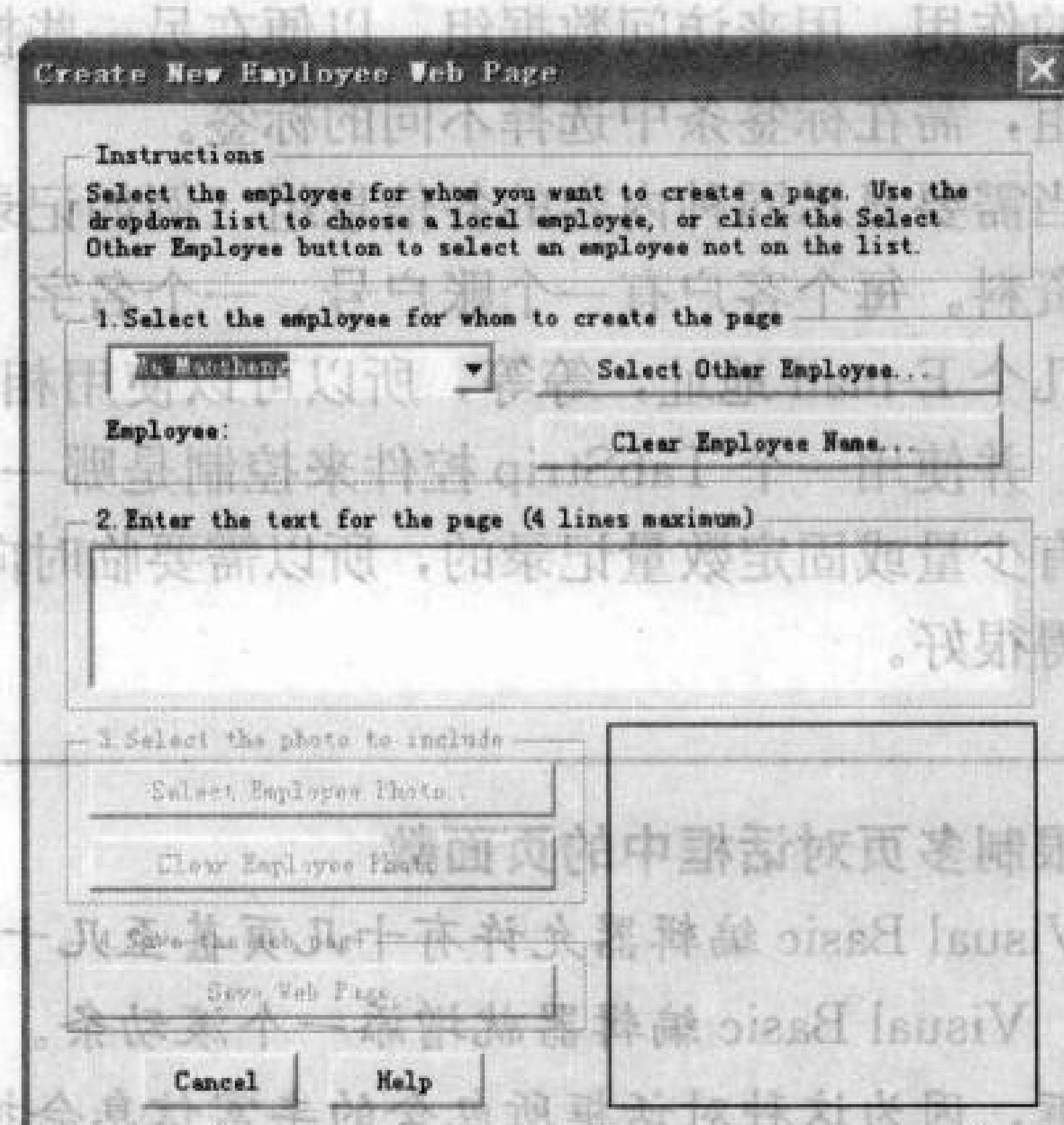


图 15.4 Create New Employee Web Page 对话框的第

二阶段，注意它与第一阶段时的不同之处： Instructions 框架中的指令有变化，Step 1 组合框下拉列表的使用使 Step 2 框架有效

控件的作用，用来访问数据组，以便在另一些控件内显示：要改变对话框中各控件内显示的数据组，需在标签条中选择不同的标签。

当需要显示同种内容的各信息组（例如记录）时，标签条很有用。例如，要保留公司客户的资料。每个客户有一个账户号、一个名字（可能是几个名字）、一个地址、几个电话号码、几个 E-mail 地址，等等，所以可以使用相同的控件组（例如几个文本框）来显示这些信息，并使用一个 TabStrip 控件来控制是哪一个客户的信息组被显示。因为极少有数据库内只有少量或固定数量记录的，所以需要临时向标签条增加标签和标题。即便这样它也能够工作得很好。

限制多页对话框中的页面数

Visual Basic 编辑器允许有十几页甚至几十页的对话框；如果在水平方向上显示不完各标签，Visual Basic 编辑器就增添一个滚动条。可能希望避免生成多于 10 页或 12 页的多页对话框，因为这种对话框所包含的丰富信息会把用户吓倒。

如果需要以十几个页面来组织一个对话框里的信息，很可能是在试图一下子向用户提供过多数据。考虑一个显示它的替代办法吧。

使用 TabStrip 则不同，因为它是使用标签条在一个记录组内的各记录之内移动，可能在一个给定的标签条中要用很多标签。但是，只要标签数不是大得不合理，通常不应该成为问题。

注意：第 14 章中的表 14.7 解释了 TabStrip 控件和多页控件独有的各种属性。

多页对话框

为了生成多页对话框，可以单击工具箱内的“多页”按钮，然后在用户窗体中想要该控件出现的地方单击。Visual Basic 编辑器把一个多页控件分两页放置，它们的标签为 Page1 和 Page2。然后照例给控件移动位置和调整大小。在典型应用中，希望生成的多页控件比它驻留的用户窗体小一些（就像在 Windows 应用程序中看到的大多数多页对话框那样）。

一旦生成了一个多页控件，通过右击它的标签和使用得到的上下文菜单，就可以在控件的一个页面上工作：

- ◆ 要添加一个页面，需右击标签，并从上下文菜单中选择“新建页”。VBA 将添加一个默认大小的新页，并把它命名为 Pagen，这个 n 是当前页面号之后的下一个序号（即使其他页面已经有了不同于 Page1、Page2 等的名称）。
- ◆ 要为多页控件内的一个页面重命名，需右击标签，并从上下文菜单中选择“重命名”。在“重命名”对话框中（见图 15.5），将对应于该页面的标题（标签文本）输进“题注”文本框，将加速键输进“加速键”文本框，将控件提示文本（用户使鼠标指针在该页标签上方游移时所看到的提示）输进“控件提示文字”文本框，然后单击“确定”按钮以关闭“重命名”对话框。
- ◆ 要从多页控件内删除一个页面，需右击标签，并从上下文菜单中选择“删除页”。Visual Basic 编辑器将除去该页，并且不做确认提示。
- ◆ 要将多页控件内的一个页面移到另一个不同的位置上，需右击标签，并从上下文菜单

中选择“移动”以显示“页面顺序”对话框（见图 15.6）。在“页面顺序”对话框中，选中想要移动的一页或几页（使用 Shift+单击以选择多个相邻页；使用 Ctrl+单击以选择多个不相邻页），并使用“上移”和“下移”按钮，按意愿重新排列这一页或这几页。完成后，单击“确定”按钮以关闭“页面顺序”对话框。

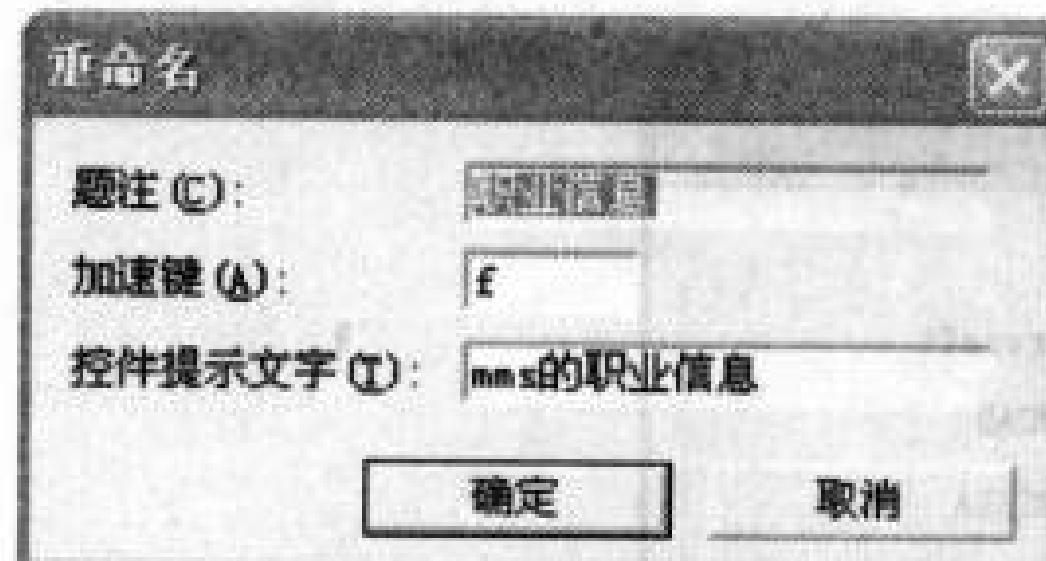


图 15.5 使用“重命名”对话框为一个页面设置标题、加速键和控件提示文本

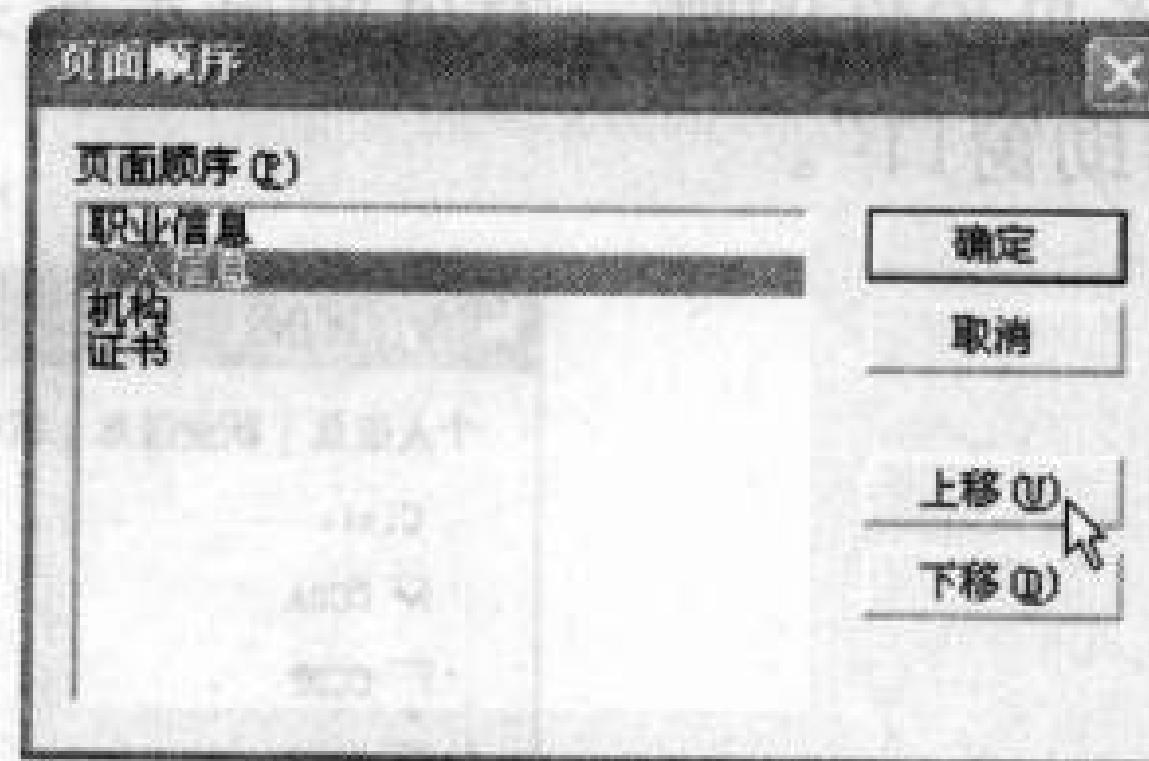


图 15.6 在“页面顺序”对话框中使用“上移”和“下移”按钮以改变多页控件中的页面顺序

- ◆为了指定多页对话框的哪一页为默认显示，需使用多页控件的 Value 属性。可以在设计时或运行时设置此属性。例如，可以使用如下所示的初始化过程，在运行时显示带有名为 MyMulti 的多页控件的对话框的第 3 页（以值 2 来标识，因为页面编号是从 0 开始的）：

```
Sub UserForm_Initialize()
    MyMulti.Value = 2
End Sub
```

生成多页对话框之后，可以使用第 14 章学习过的技术，用控件来丰富对话框的各页面。在对话框之内（不仅仅是控件出现的页面内），每个控件都必须有一个唯一的名称。

在设计多页对话框时，心中必须想着以下各点：

- ◆在对话框中区分信息或选项的最佳方式是什么？哪些信息或选项要放进哪一页？用户希望把哪些信息或选项组合在一起？
- ◆哪些控件应该出现在一页上？大多数对话框都需要每页上至少有两个命令按钮——如“确定”和“取消”，或“确定”和“关闭”——可供使用，以便允许用户从任一页面上退出对话框。在极少数情况下，也可能希望强迫用户返回到某一特定页面以关闭对话框。此时要确认，不包含有退出对话框的命令按钮的每个页面，要能向用户指明，他们在什么地方能找到这样的命令按钮。
- ◆关于设置：除了“确定”按钮，是否还需要使用“应用”（Apply）按钮，在不关闭对话框的情况下将各种变化作用于某个特定页面？

因为多页对话框中的每个控件都有一个唯一的名称，所以当从多页对话框返回信息时，只需要指定相关的对象——无需指定它在哪一页上。

图 15.7 是一个多页对话框的例子。第一页包含客户的个人交际信息，第二页是客户的职业信息，第三页是客户隶属的机构，第四页是客户所持有的证书。

多页控件的大多数属性是简洁的，但有几个特性需要做专门的论述：

- ◆ Style 属性提供了 fmStyleTabs（默认设置，显示标签，以便在页面间导引）、fmStyleButtons（给每页一个矩形按钮，对应于当前页面的按钮呈“推入”状）或 fm-

StyleNone (不为多页对话框内各页面之间的导引提供手段，也没有多页对话框的边框指示)。如果要生成的用户窗体有两种或多种可替换布局，但用户一次只需看其中一种，这时 fmStyleNone 是有用的。把一组控件放在多页对话框的一个页面上，再把另一组控件放在另一个页面上，就能提供出两个外观不同的对话框，这时只需要确定让多页控件的哪一个页面显示出来就可以了。例如，可以使用这种方法来生成一个“帮助窗口”。

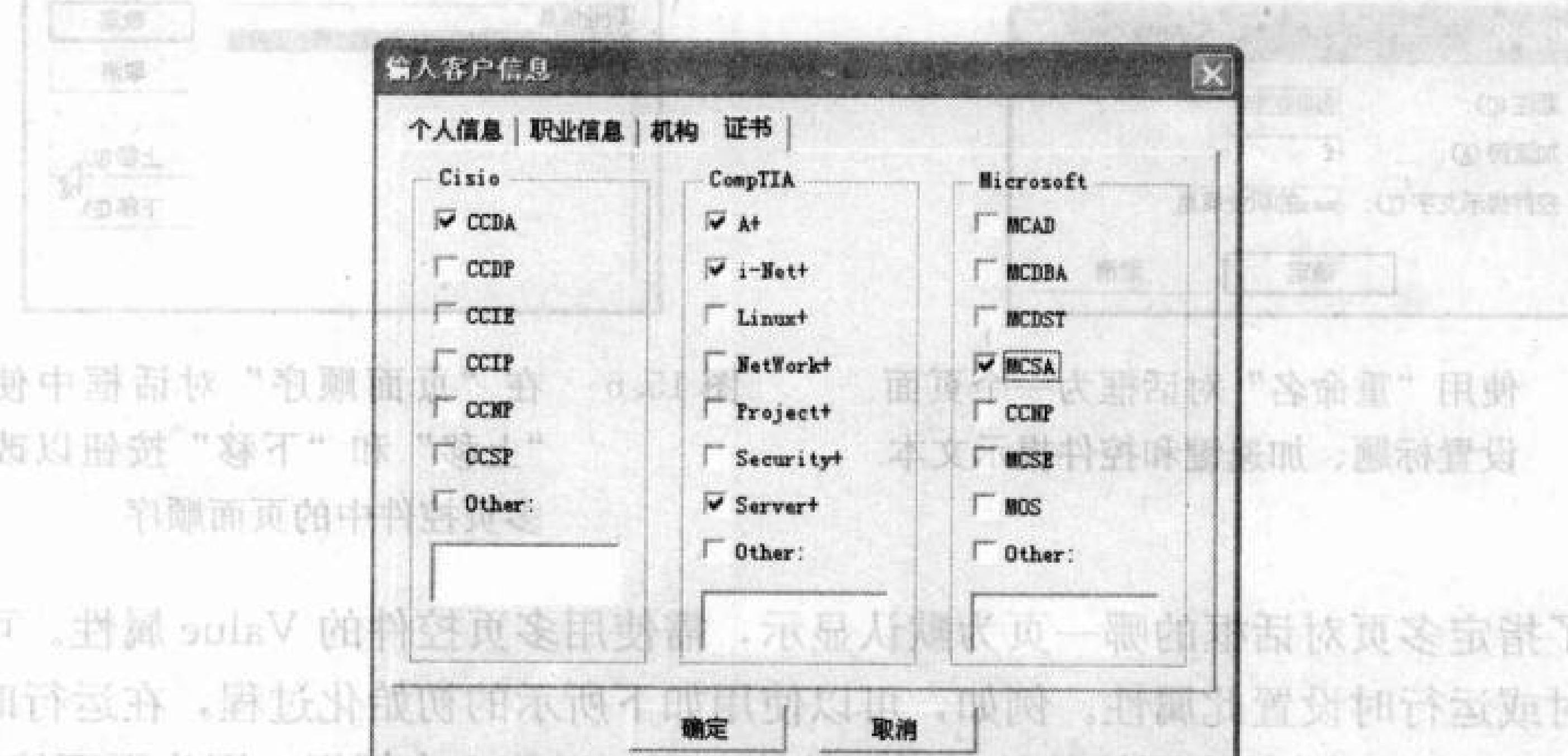


图 15.7 通过在对话框中使用多页，可以提供出清晰有序、用户容易寻找的信息

- ◆ TabOrientation 属性，它控制与页面对应的标签（或按钮）出现在控件上的什么地方。可以使用 fmTabOrientationTop（默认设置，将标签置于控件顶部），fmTabOrientationBottom，fmTabOrientationLeft 和 fmTabOrientationRight。但后三种设置如无特别优势，轻易不用。
- ◆ MultiRow 属性控制一个多页控件是只有一行对应于各页面的标签（False），还是有多行标签（True）。如果将 MultiRow 设置为 True，当第一行或当前行的空间已被占满时，Visual Basic 编辑器将添加第二个或更多个标签行。

多页控件不一定要占满整个对话框——实际上，大多数对话框把重要的命令按钮保持在多页区域外面，使得不论用户工作于哪一页，这些按钮都能为用户所用。据说，通常并不主张把多页控件作为对话框的次级主体部分来使用。在一个复杂而忙碌的对话框内，一个小小的多页控件出现得不比一个组框多多少，而且用户容易忽略标签，尤其是在用户草草扫视各个控件的时候。

使用 TabStrip 的对话框

使用 TabStrip 的对话框与多页对话框有本质的不同。TabStrip 控件不是用来安排其他控件的，而是用来控制对话框中应该出现何物的，这样用户就可以从一组数据移动到另一组数据。

例如，可以使用由 TabStrip 驱动的对话框来查看和更新数据源（如 Word 中的表格、Excel 中的数据表或 Access 中的数据库）中的记录。下例使用了 Excel 工作簿，在工作簿中，信息保存在很多张工作表上。图 15.8 显示了 DataSurfer 对话框，它是由一个 TabStrip 驱动的。

为了生成多标签对话框，需确定把一个 TabStrip 控件放在依靠它帮助才得以“繁衍”的其他一些控件的上方、下方，还是旁边。常规位置——也是默认位置，是上方。但是，近

来在 Word 应用程序上，竖向标签和置于下方的标签有增多之势。同多页控件一样，使用 TabStrip 控件的 TabOrientation 属性来指明标签条应该出现在它的控件的顶部、底部、左边还是右边。



图 15.8 使用 TabStrip 控件来生成多标签对话框。TabStrip 用来控制在对话框中的其他控件内显示哪一组信息

标签条可以包含零个、一个或多个标签。在大多数情况下，极少碰到一个标签条上只有一个标签的，完全没有标签的情况更少。但是，如果在过程中动态地增减标签条上的标签，为每个刚找到的记录生成一个标签，可能会陷入这种情况，即只有一个记录，因而对话框只有一个标签——甚至标签条上没有任何标签。

单击工具箱上的 TabStrip 按钮，在用户窗体内单击以给标签条定位，然后拖曳它至合适的大小。依据对话框其余部分的外观，可能希望使标签条足够大，以便在一维范围内（见图 15.8）或二维范围内包括受它影响的所有控件。要记住这只是给用户提供了视觉上的便利，而标签条和其他控件之间的逻辑联系是通过代码来建立的。此后，可以采用与多页控件相同的做法添加标签、为标签重命名、移动标签以及删除标签。

如果还没有向对话框放置其他控件，下面就可以放置了。

一切就位之后，就要编写代码了，这些代码可以让标签条生效以控制其他控件的内容。程序清单 15.2 显示了对应于 DataSurfer 对话框中标签条的代码。该标签条名为 tabSurfer，代码与它的 Change 事件一道工作——当用户从标签条中的一个标签移至另一标签时，此事件被触发。

程序清单 15.2

```

1. Private Sub tabSurfer_Change()
2.     If bInInitializing = False Then
3.         With ActiveWorkbook.Sheets(tabSurfer.Value + 1)
4.             'load the contents of the worksheet that corresponds to the tab chosen
5.             .Activate
6.             txtFirstName.Text = .Cells(1, 2).Text
7.             txtInitial.Text = .Cells(2, 2).Text
8.             txtLastName.Text = .Cells(3, 2).Text
9.             txtAddress1.Text = .Cells(4, 2).Text

```

```
10.     txtAddress2.Text = .Cells(5, 2).Text
11.     txtCity.Text = .Cells(6, 2).Text
12.     txtState.Text = .Cells(7, 2)
13.     txtZip.Text = .Cells(8, 2).Text
14.     txtHomeArea.Text = .Cells(9, 2).Text
15.     txtHomePhone.Text = .Cells(10, 2).Text
16.     txtWorkArea.Text = .Cells(11, 2).Text
17.     txtWorkPhone.Text = .Cells(12, 2).Text
18.     txtWorkExtension.Text = .Cells(13, 2).Text
19.     txtEmail.Text = .Cells(14, 2).Text
20. End With
21. End If
22. End Sub
```

指定了工作表之后，程序清单 15.2 中的代码基本上是在重复它自身，以便与 DataSurfer 对话框中的每一个文本框相对应。这个对话框使用组织成活动工作簿中的 Excel 数据表的数据源来工作。

工作簿中的每张工作表都是一个客户记录，客户的姓名出现在工作表的标签上，有关客户的数据出现在第二栏之中，包括姓名的全称、地址、电话号码、E-mail 地址等。所以要得到信息，必须知道所涉及的记录在哪张表上以及第二栏的那个方格内。

下面看看代码是如何工作的：

- ◆ 第 1 行声明私有过程 tabSurfer_Change，每当 tabSurfer 标签条的 Change 事件被触发，该过程自动运行。每次用户改变显示的标签时，都有 Change 事件发生，所以使用这一事件来控制显示在各文本框中的信息。

- ◆ 将一个标签添加到（或移出）标签条时，也有 Change 事件发生。因为 DataSurfer 用户窗体使用 Initialize 事件过程来增减标签条上的标签（工作簿中的每张工作表有一个标签），有必要停止那些不必要的 Change 事件的运行。所以，用户窗体声明了一个名为 blnInitializing 的私有布尔型变量。Initialize 过程在运行时，把 blnInitializing 设置为 True，在过程马上就要结束时把它设置为 False。Change 事件过程的第 2 行查看 blnInitializing 是否为 False。如果不是，说明 Initialize 过程已经触发了该事件，Change 过程不必将信息装载进各方格中——所以执行转到第 21 行，即马上就要结束该过程。但是，当 Initialize 过程已经结束运行时，blnInitializing 已经设置为 False，所以每次用户改变标签条中的标签时，Change 事件过程都将运行。

- ◆ 第 3 行开始一个 With 语句，它在活动工作簿中适当的工作表上工作：ActiveWorkbook. Sheets(tabSurfer.Value + 1)。tabSurfer 标签条的 Value 属性指明标签条中的哪一个标签被选中。因为标签条中第一个标签的编号为 0，而工作簿中第一张工作表的编号为 1，所以要把标签条的 Value 加上 1，使两个编号一致。

- ◆ 第 4 行是注释。第 5 行使用 Activate 方法来激活所涉及的工作表。

- ◆ 然后，第 6 行至第 19 行将用户窗体中每个文本框内的 Text 属性，设置为工作表上第二栏内相应方格的内容。例如，第 6 行将 txtFirstName 文本框（它出现在对话框的 FirstName 之下）的 Text 属性，设置为第二栏内第一个方格的内容：Cells(1, 2).Text。

- ◆ 第 20 行结束 End 语句，第 21 行结束 If 语句，第 22 行使过程终止。

在对话框中使用图片

使用 Image 控件可以将图片添加到对话框中。单击工具箱内的“图像”按钮，然后在用户窗体中希望该 Image 控件出现的地方单击即可。放置好 Image 控件之后，可以像对待任何其他控件那样，对图片进行尺寸调整及移位。

警告：确认所选择的图片是显示该对话框的所有计算机都可以使用的。如果图片不可用，它就不能在对话框内显示，就会破坏效果。

为了选择将要在 Image 控件中出现的图片，需在“属性”窗口中选择 Picture 属性，再单击随后出现在该条目右边的省略号按钮，Visual Basic 编辑器将显示“加载图片”对话框。选择图片文件，并选择“打开”按钮。“属性”窗口中的 Picture 属性记录的是被选图片的类型（如 Bitmap）而不是它的文件名，并且图片出现在 Image 控件内，从中可以看出它的大小是否适当。

提示：设置用户窗体的 Picture 属性，可以把图片作为用户窗体自身的背景。

通过程序将图片加载进 Image 控件

通过程序为 Image 控件指定图片时，需要使用 LoadPicture 语句，而不是简单地将图片指定给 Image 控件的 Picture 属性。LoadPicture 语句的语法如下：

```
LoadPicture filename, [WidthDesired], [HeightDesired]
```

filename 是一个字符串参数，它指定要加载进 Image 控件的图片文件的名称。WidthDesired 是可选的长整型参数，它指定图片的宽度，以 twips 计。而 HeightDesired 是可选的长整型参数，它指定图片的高度。

例如，下述语句将图片 Company Logo.jpg 加载进 f:\common\images\：

```
LoadPicture "f:\common\images\Company Logo.jpg"
```

选择好图片后，可以使用多个选项给它定位和定格式：

◆ 如有必要，可使用 PictureAlignment 属性来设置图片的对齐方式。（如果图片正好填满图像控件——既不是超覆盖，也未留下空白——就不必为它设置对齐效果。）表 15.1 列出与 PictureAlignment 属性对应的常量和值。

表 15.1 对应于 PictureAlignment 属性的常量和值

常量	值	图片在 Image 控件中的对齐方式
fmPictureAlignmentTopLeft	0	左上角
fmPictureAlignmentTopRight	1	右上角
fmPictureAlignmentCenter	2	中心
fmPictureAlignmentBottomLeft	3	左下角
fmPictureAlignmentBottomRight	4	右下角

◆ 如有必要，可使用 PictureSizeMode 属性来剪裁、扩展或放大图片：fmPictureSize-

ModeClip (0) 是剪裁图片使其适合于 Image 控件; fmPictureSizeModeStretch (1) 是扩展或压缩图片使其适合于 Image 控件; fmPictureSizeModeZoom (2) 是放大或缩小图片, 使图片的尺寸中最接近 Image 控件的那一维, 正好与 Image 控件的宽度或高度相适应, 但不改变图片的比例 (这一选项通常会使其他边上留下未填满的空隙)。

- ◆ 如果需要平铺图片以占据控件中的剩余空间, 可以将 PictureTiling 属性设置为 True。
- ◆ 如果需要调节图片相对于其标题的位置, 必须设置所涉及的复选框、命令按钮、标签、选项按钮或切换按钮等的 PicturePosition 属性。

表 15.2 列出与 PicturePosition 对应的常量和值。

表 15.2 对应于 PicturePosition 属性的常量和值

常量	值	图片位置	标题对齐
fmPicturePositionLeftTop	0	在标题左边	标题与图片顶端对齐
fmPicturePositionLeftCenter	1	在标题左边	标题与图片中间对齐
fmPicturePositionLeftBottom	2	在标题左边	标题与图片底边对齐
fmPicturePositionRightTop	3	在标题右边	标题与图片顶端对齐
fmPicturePositionRightCenter	4	在标题右边	标题与图片中间对齐
fmPicturePositionRightBottom	5	在标题右边	标题与图片底边对齐
fmPicturePositionAboveLeft	6	在标题上面	标题与图片左边对齐
fmPicturePositionAboveCenter	7	在标题上面	标题在图片下面中间位置 (这是默认设置)
fmPicturePositionAboveRight	8	在标题上面	标题与图片右边对齐
fmPicturePositionBelowLeft	9	在标题下面	标题与图片左边对齐
fmPicturePositionBelowCenter	10	在标题下面	标题在图片上面中间位置
fmPicturePositionBelowRight	11	在标题下面	标题与图片右边对齐
fmPicturePositionCenter	12	在控件中间	标题在图片上的水平和垂直方向的中心

把图片放置好、调整好大小、定好格式之后, 还可能有很多要做的事, 例如, 使用图片的 Click 事件来触发某一个行动; 例如, 如果让用户在对应于某一文档的两种格式中做出一个选择, 可以让用户单击适当的图片来做出选择, 而不必让用户先选择图片, 再去单击一个命令按钮。

生成无模型对话框

VBA 的版本 6 和更高版本有生成无模型对话框的内容——用户可以把这种对话框留在屏幕上不管, 自己继续做应用程序中的工作。从与 Office 的工作中, 用户可以肯定已熟悉了这种无模型对话框。例如, Word 和 Excel 中的“查找和替换”对话框以及 PowerPoint 中的“替换”对话框, 都是无模型对话框。

当显示一个无模型对话框时, 同任何模型对话框一样, 它要取得焦点, 而且它的标题栏取得活动标题栏的颜色, 但是用户可以在应用程序窗口内单击, 将焦点转回到该窗口。当无模型对话框失去焦点时, 它的标题栏取得不活动标题栏的颜色。要把焦点恢复到这个无模型对话框, 需再次单击它。

生成一个无模型对话框是很简单的, 只需将用户窗体的 ShowModal 属性从其默认设置 True 改变为 False。

不是生成模型对话框而是生成无模型对话框, 这有很多理由。作为一个简单的例子, 可以

在 Word 中生成一个过程和对话框，让它从用户中收集信息以产生出一个备忘录或报告。把对话框做成无模型对话框，就允许用户从一个打开的文档中获取信息（或打开其他一些文档从中收集信息），将得到的信息粘贴进对话框，如图 15.9 所示的那样——免得用户在调用对话框之前不得不去复制这些信息，还允许用户很容易地复制多个分离的项目。同样，也可以生成一个无模型用户窗体（可能其形状像个工具栏），用户可以把它保留在屏幕上，并使用它来自动地把文本输进三个或四个其他文档的预先确定的章节中，而不会失去用户自己在当前文档中的位置。

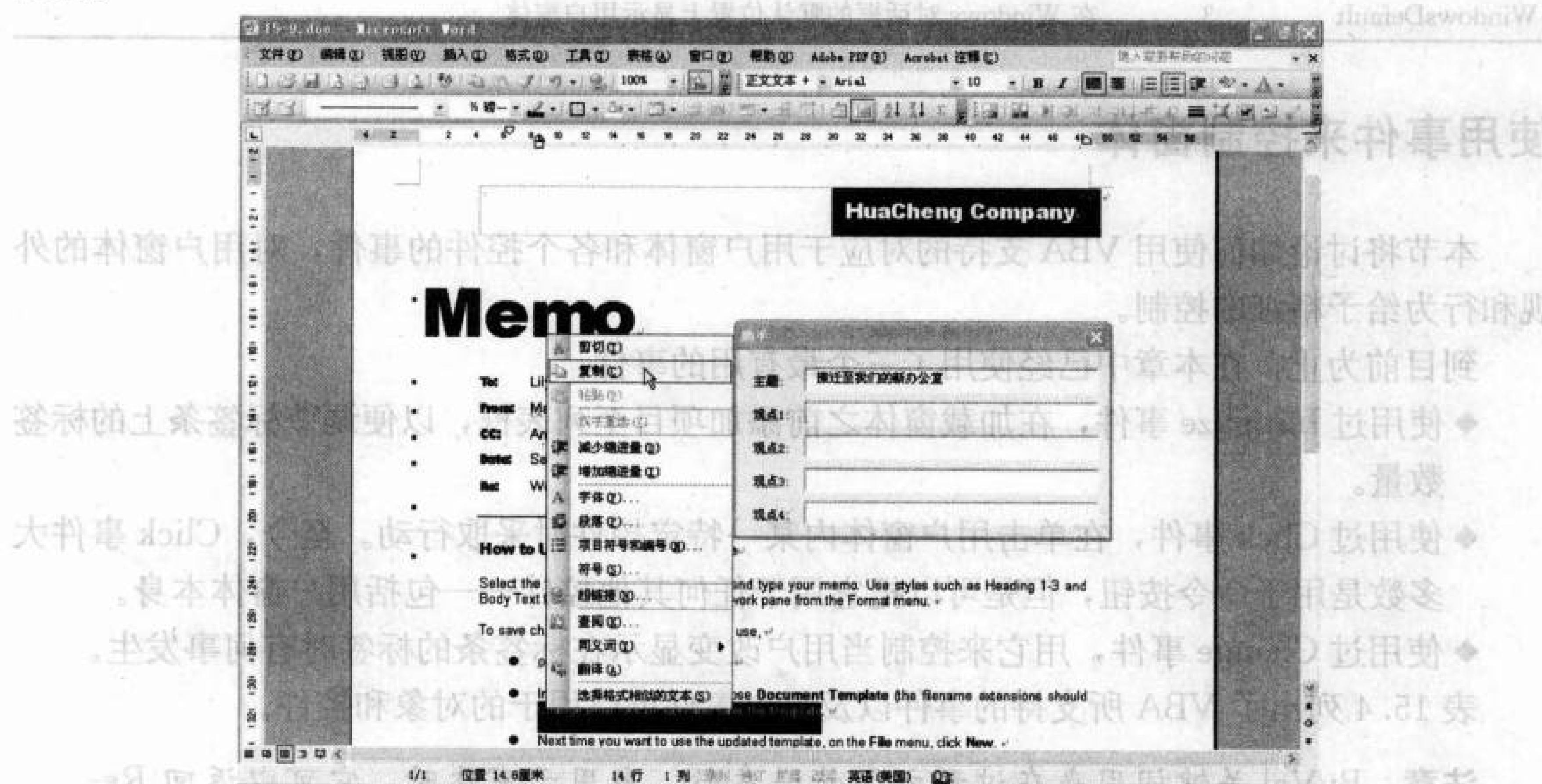


图 15.9 如果做一个无模型对话框而不是模型对话框，用户可以在对话框显示的同时继续在应用程序窗口中工作

也可以使用无模型对话框来显示复杂而又有相互关系的用户窗体组合。在这种组合中，用户可以很容易地把信息从一个用户窗体传到另一个用户窗体，或者至少可以在同一时间访问两个或多个已显示的用户窗体的不同区域。同时显示多个窗体会使用户忙乱，但有时可能是必要的。

在大部分时间里，可能都希望在 VBA 过程中使用模型对话框。使用模型对话框时，用户在继续其在应用程序中的工作之前，必须对对话框进行处理并且不会有下面这种危险：用户结束了工作却还有多个对话框散落在屏幕各处，且处于形形色色的弃用状态。

注意：可以在同一时间既使用模型用户窗体又使用无模型用户窗体，但是只可以从另一个模型对话框中显示一个模型对话框。当用户关闭第二个模型对话框时，VBA 按照默认方式使用户返回到第一个模型对话框。尽管如此，还是可以编写代码使第二个模型对话框在关闭自身之后关闭第一个对话框。

为对话框选择位置

按照默认方式，VBA 尽可能地将对话框放在应用程序窗口的正中间。对 Windows 应用程序来说，这是一种正常行为。如果需要使用一个不同的起始位置（例如，为了避免遮挡屏幕上的重要数据），可以为用户窗体设置 StartUpPosition 属性。表 15.3 说明可以使

用的设置。

表 15.3 StartUpPosition 属性的设置

StartUpPosition 属性	值	效果
Manual	0	在 Windows 屏幕的左上角显示用户窗体
CenterOwner	1	把用户窗体放在用户的应用程序中——即该用户窗体所隶属的应用程序的正中间
CenterScreen	2	把用户窗体放在整个屏幕的正中间。在多监视器配置中，放在包含活动窗口的那个监视器的正中间
WindowsDefault	3	在 Windows 对话框的默认位置上显示用户窗体

使用事件来控制窗体

本节将讨论如何使用 VBA 支持的对应于用户窗体和各个控件的事件，对用户窗体的外观和行为给予精细的控制。

到目前为止，在本章中已经使用了三个最有用的事件：

- ◆ 使用过 Initialize 事件，在加载窗体之前添加项目至列表框，以便调节标签条上的标签数量。
- ◆ 使用过 Click 事件，在单击用户窗体内某一特定控件时采取行动。至今，Click 事件大多数是用于命令按钮，但是可以把它用于任何其他控件——包括用户窗体本身。
- ◆ 使用过 Change 事件，用它来控制当用户改变显示在标签条的标签时有何事发生。

表 15.4 列出了 VBA 所支持的事件以及每个事件可以用于的对象和控件。

注意： ByVal 关键词用来在过程之间传递参数。当用于窗体时，它可以返回 ReturnBoolean、ReturnEffect、ReturnInteger 和 ReturnString 这些对象。

表 15.4 VBA 支持的事件以及与这些事件相关联的对象和控件

事件	发生时间	应用于这些控件和对象
Activate	当用户窗体成为活动窗口时	UserForm
Deactivate	当用户窗体不再是活动窗口时	UserForm
AddControl	当在运行时间内添加一个控件时	框架，多页，UserForm
AfterUpdate	当用户改变了一个控件内的数据之后	复选框，组合框，命令按钮，框架，图像，标签，列表框，多页，选项按钮，滚动条，微调按钮，TabStrip，文本框，切换按钮，UserForm
BeforeDragOver	当用户实施一次拖放操作时	复选框，组合框，命令按钮，框架，图像，标签，列表框，多页，选项按钮，滚动条，微调按钮，TabStrip，文本框，切换按钮，UserForm
BeforeDropOrPaste	当用户打算释放一个被拖曳的项目或打算粘贴一个项目时	复选框，组合框，命令按钮，框架，图像，标签，列表框，多页，选项按钮，滚动条，微调按钮，TabStrip，文本框，切换按钮，UserForm
BeforeUpdate	当用户改变了一个控件内的数据，但新数据还未在该控件内出现时	复选框，组合框，列表框，选项按钮，滚动条，微调按钮，文本框，切换按钮
Change	当一个控件的 Value 属性改变时	复选框，组合框，列表框，多页，选项按钮，滚动条，微调按钮，TabStrip，文本框，切换按钮

(表1)

(续表)

事件	发生时间	应用于这些控件和对象
Click	当用户使用主鼠标按钮单击一个控件或对象时	复选框, 组合框, 命令按钮, 框架, 图像, 标签, 列表框, 多页, 选项按钮, TabStrip, 切换按钮, UserForm
DblClick	当用户使用主鼠标按钮双击一个控件或对象时	复选框, 组合框, 命令按钮, 框架, 图像, 标签, 列表框, 多页, 选项按钮, TabStrip, 文本框, 切换按钮, UserForm
DropButtonClick	当用户显示或隐藏一个下拉列表时	组合框, 文本框
Enter	用户窗体上的一个控件即将接收到来自另一个控件的焦点之前	复选框, 组合框, 命令按钮, 框架, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮
Exit	当用户窗体上的一个控件即将失去焦点并把焦点传给另一控件之前	复选框, 组合框, 命令按钮, 框架, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮
Error	当一个控件或对象遭遇一个错误时	复选框, 组合框, 命令按钮, 框架, 图像, 标签, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮, UserForm
Initialize	加载一个用户窗体之后, 但显示该用户窗体之前	UserForm
KeyDown	当用户按下键盘上的一个键时	复选框, 组合框, 命令按钮, 框架, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮, UserForm
KeyUp	当用户释放在键盘上按下的一个键时	复选框, 组合框, 命令按钮, 框架, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮, UserForm
KeyPress	当用户按下键盘上的一个 ANSI 键时	复选框, 组合框, 命令按钮, 框架, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮, UserForm
Layout	当一个框架、多页或用户窗体的大小被改变时	框架, 多页, UserForm
MouseDown	当用户按下主鼠标按钮时	复选框, 组合框, 命令按钮, 框架, 图像, 标签, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮, UserForm
MouseUp	当用户释放主鼠标按钮(按下它之后)	复选框, 组合框, 命令按钮, 框架, 图像, 标签, 列表框, 多页, 选项按钮, 滚动条, 微调按钮, TabStrip, 文本框, 切换按钮, UserForm
MouseMove	当用户移动鼠标时	复选框, 组合框, 命令按钮, 框架, 图像, 标签, 列表框, 多页, 选项按钮, TabStrip, 文本框, 切换按钮, UserForm
QueryClose	当一个用户窗体将要关闭时	UserForm
RemoveControl	当一个控件被删除时	框架, 多页, UserForm
Resize	当一个用户窗体被调整大小时	UserForm
Scroll	当用户移动滚动条时	框架, 多页, 滚动条, UserForm

(表1)

(续表)

事件	发生于	应用于这些控件和对象
SpinDown	当用户单击一个微调按钮控件上的向下按钮时	微调按钮
SpinUp	当用户单击一个微调按钮控件上的向上按钮时	微调按钮
Terminate	当一个用户窗体已经从存储器中被卸出时	UserForm
Zoom	当控件或用户窗体的 Zoom 属性被改变时	框架, 多页, UserForm

正如你所看到的, VBA 事件可分为若干类。在以下各节中, 将以其用处大小为序对它们进行讨论:

- ◆ 仅仅应用于 UserForm 对象的事件;
- ◆ 应用于 UserForm 对象和其他容器对象(例如框架控件和多页控件)的事件;
- ◆ 应用于大多数控件, 有时也包括 UserForm 对象的事件。

提示: 为了最大限度地使用窗体, 需要了解各事件发生的顺序。如果不了解, 会因以事件相互触发或相互冲突的方式来使用事件, 而把自己搞糊涂。

仅仅应用于 UserForm 对象的事件

本节讨论仅仅应用于 UserForm 对象的事件, 这些事件是: Initialize、QueryClose、Activate、Deactivate、Resize 和 Terminate 事件。

Initialize 事件

当用户窗体被加载, 但还未在屏幕上出现时, Initialize 事件发生。

与 Initialize 事件对应的 VBA 的语法表示如下, 其中 userform 是一个有效的 userform 对象:

```
Private Sub userform_Initialize()
```

Initialize 事件的典型应用包括收回用户窗体或应用程序所需要的信息和将信息指定给用户窗体上的各控件(尤其是给列表框和组合框, 需要在运行时而不是设计时向它们添加信息)。

取决于用户窗体的风格和复杂性, 也可以使用 Initialize 事件调整用户窗体的大小、调整用户窗体上控件的大小、显示或隐藏特定的控件以及确认用户窗体尽可能做到最适合于用户的要求, 然后才显示它。

QueryClose 事件

QueryClose 事件只能应用于 UserForm 对象, 这一事件在用户窗体即将关闭时发生。

与 QueryClose 事件对应的语法为:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
```

此处, Cancel 的数据类型为整型, 典型值为零。非零值可以使 QueryClose 事件免于发生, 并使用户窗体(以及应用程序)停止关闭。

CloseMode 是一个值或常量，它给出 QueryClose 事件的原因。表 15.5 列出了对应于 CloseMode 的值和常量。

表 15.5 对应于 CloseMode 参数的值和常量

常量	值	QueryClose 事件的原因
vbFormControlMenu	0	通过单击用户窗体的“关闭”按钮，或者从用户窗体的控件菜单中调用 Close 命令（例如，右击用户窗体的标题栏，并从上下文菜单中选择“关闭”），用户将该用户窗体关闭
vbFormCode	1	代码中的一个 Unload 语句已将用户窗体关闭
vbAppWindows	2	Windows 正在关闭，因而该用户窗体正在关闭
vbAppTaskManager	3	TaskManager（任务管理器）正在关闭应用程序，因而该用户窗体正在关闭

初看起来，除了用来检查用户是否正在试图关闭确实想关闭的某一个用户窗体之外，QueryClose 似乎并无太多用处。例如，如果确定用户已经在他们打算关闭的用户窗体里输入了大量数据，可能想要检查一下，用户是否因失误而没有单击用户窗体的“关闭”按钮或“取消”按钮，就像下面的 Word 代码片段那样：

```

Private Sub UserForm_QueryClose(Cancel As Integer, _  
    CloseMode As Integer)  
    'make sure the user wants to close the user form  
    'if they have entered information in it  
    Select Case CloseMode  
        Case 0  
            'user has clicked the close button or invoked an Unload statement  
            'if text box contains more than 5 characters, ask to save it  
            If Len(txtDescription.Text) > 5 Then  
                If MsgBox("The Description text box contains " & _  
                    "a significant amount of text." & vbCrLf & _  
                    "Do you want to save this text?", vbYesNo + _  
                    vbQuestion, "Close Form") <> 0 Then  
                    Documents.Add  
                    Selection.TypeText txtDescription.Text  
                    ActiveDocument.SaveAs _  
                    "c:\temp\Temporary Description.doc"  
                    MsgBox "The contents of the Description text " & _  
                    "box have been saved in " & _  
                    "c:\temp\Temporary Description.doc.", _  
                    vbOKOnly + vbInformation,  
                    "Form Information Saved"  
                End If  
            End If

```

但是，当应用程序而不是用户窗体正在关闭时，QueryClose 还确实有它的用处。如果用户窗体是个无模型窗体，用户可能不会知道窗体仍开着，而且他们将要从中丢失数据。

有时可以使用 QueryClose 来保存来自某一个用户窗体的信息，此时应用程序已经停止响应，并正在被 Windows 或 TaskManager（任务管理器）关闭。要警惕的是，在这方面 QueryClose 的记录并不完美——代码有时候会不运行。

为了停止关闭一个应用程序，需将 QueryClose 事件的 Cancel 属性设置为 True。

于窗体已出线 3.21 第一回窗体的创建 CloseClose 由带进窗体，是带进窗体一个基 CloseClose

Activate 事件

当用户窗体成为活动窗口时，Activate 事件发生。尤其是，如果用户窗体被一个 Show 语句而不是 Load 语句加载，在 Initialize 事件之后紧接着该用户窗体被显示，这意味着有 Activate 事件发生。

注意：如果用户窗体在因使用 Show 语句而被显示之前，被一个 Load 语句加载的话，Initialize 事件在 Load 语句之后便发生，Activate 事件则是在 Show 语句之后发生。

但是，如果用户窗体曾经被激活，现在又重新激活时，Activate 事件也要发生。例如，如果使用 Activate 事件过程创建了一个无模型对话框，每当用户使该用户窗体在已被激活之后又重新激活时，代码便被执行。同时，如果从第一个用户窗体中显示第二个窗体，然后把焦点返回到第一个用户窗体，并将第二个用户窗体激活从而使其关闭时，Activate 事件也会发生。

对应于 Activate 事件的语法是：

```
Private Sub UserForm_Activate()
```

警告：因立即连续使用 Deactivate 和 Activate 引起的问题

VBA 不能总是刚执行了对应于一个用户窗体的 Deactivate 事件的事件过程，又立即去执行对应于另一个用户窗体的 Activate 事件的事件过程，有时候会是这样，但更经常的情况是不会如此。

例如，假设有两个用户窗体，名为 One 和 Two，每个都有 Activate 事件过程和 Deactivate 事件过程。如果从 One 来显示 Two，应该运行来自 One 的 Deactivate 事件代码，然后运行来自 Two 的 Activate 代码。但是，事情往往不是这样，常常碰到的是 One 的 Deactivate 代码要运行，但 Two 的 Activate 事件不运行。再使它运行时，可能是 Two 的 Activate 代码运行了，但是 One 的 Deactivate 代码又不运行了。不过，如果把 Deactivate 事件过程从 One 中除去或把它变成注释，然后再来尝试，结果是，Two 的 Activate 代码在每当 One 显示 Two 时正常运行，这说明，在 Deactivate 事件代码存在的情况下，Activate 事件发生，但 Activate 事件过程的代码没有执行。

Deactivate 事件

当用户窗体已经成为活动窗口之后又失去焦点时，Deactivate 事件发生。但是，当用户窗体被隐藏或被卸载时，该事件不发生。例如，如果显示一个包含有 Deactivate 事件过程的用户窗体，然后关闭该用户窗体，Deactivate 事件不发生。但是，如果从第一个用户窗体中显示第二个用户窗体，当焦点被传递到第二个用户窗体时，与第一个用户窗体对应的 Deactivate 事件发生。使用无模型用户窗体时，每当因单击另一窗体而离开一个窗体时，Deactivate 事件被触发。

对应于 Deactivate 事件的语法是：

```
Private Sub UserForm_Deactivate()
```

关于因立即连续使用 Deactivate 和 Activate 引起的问题，参见上面的内容。

Resize 事件

当以人工方法或程序方法调整一个用户窗体的大小时，Resize 事件发生。

对应于 Resize 事件的语法是：

```
Private Sub UserForm_Resize()
```

Resize 事件的主要应用是移动各控件、调整各控件的大小、显示或隐藏各控件以适应用户窗体的新尺寸。例如，使用程序清单 15.3 所示的代码，可以调整文本框的大小，使其占据它所在的用户窗体的大部分宽度（见图 15.10）。

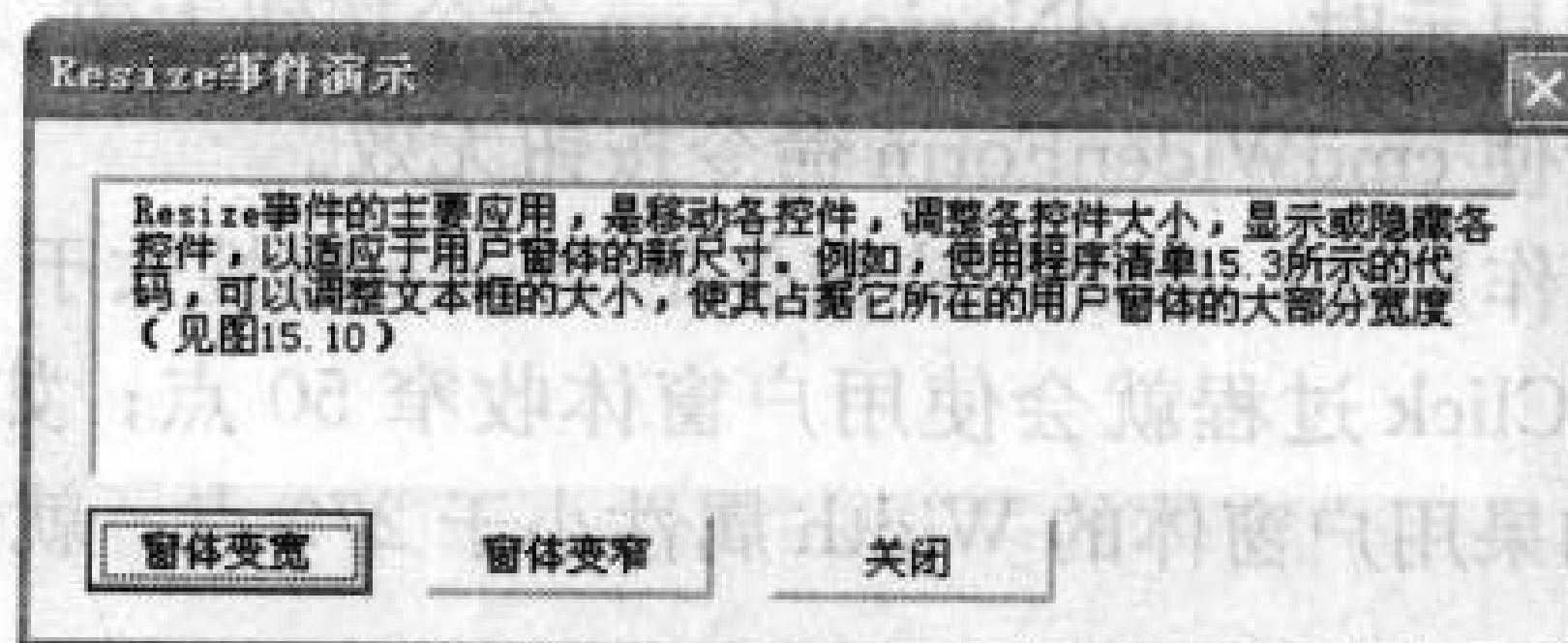


图 15.10 可以使用用户窗体的 Resize 事件调整用户窗体包含的各控件的大小和位置

程序清单 15.3

```
1. Private Sub cmdWidenForm_Click()
```

```
2.     With frmResize
```

```
3.         If .Width < 451 Then
```

```
4.             .Width = .Width + 50
```

```
5.             If cmdNarrowForm.Enabled = False Then _
```

```
6.                 cmdNarrowForm.Enabled = True
```

```
7.             If .Width > 451 Then _
```

```
8.                 cmdWidenForm.Enabled = False
```

```
9.         End If
```

```
10.        End With
```

```
11.    End Sub
```

```
12.
```

```
13.    Private Sub cmdNarrowForm_Click()
```

```
14.        With frmResize
```

```
15.            If .Width > 240 Then
```

```
16.                .Width = .Width - 50
```

```
17.                If cmdWidenForm.Enabled = False Then _
```

```
18.                    cmdWidenForm.Enabled = True
```

```
19.                If .Width < 270 Then _
```

```
20.                    cmdNarrowForm.Enabled = False
```

```
21.        End If
```

```
22.        End With
```

```
23.    End Sub
```

```
24.    Private Sub cmdClose_Click()
```

```
25.        Unload Me
```

```

23. End Sub
24.
25. Private Sub UserForm_Resize()
26.     txt1.Width = frmResize.Width - 30
27. End Sub

```

程序清单 15.3 含有四个短过程：一个是对应于 cmdWidenForm 命令按钮的 Click 事件的过程；一个是对应于 cmdNarrowForm 命令按钮的 Click 事件的过程；一个是对应于 cmdClose 命令按钮的 Click 事件的过程，而一个是对应于用户窗体的 Resize 事件的过程。

第 1 行至第 4 行的作用是：当用户单击“窗体变宽”按钮时，只要用户窗体的 Width 属性小于 451 点（1 点等于 1/72 英寸），cmdWidenForm_Click 过程就会使用户窗体的宽度增加 50 点。第 5 行使 cmdNarrowForm 命令按钮有效，如果它原来是无效的话。（当用户窗体按钮按它的初始的较窄宽度显示时，cmdNarrowForm 命令按钮无效。）如果用户窗体的 Width 属性大于 451 点，第 6 行使 cmdWidenForm 命令按钮无效。

第 11 行至第 19 行的作用是：只要用户窗体的 Width 属性大于 240 点（这是它的初始宽度），cmdNarrowForm_Click 过程就会使用户窗体收窄 50 点；如果 cmdWidenForm 已无效，就使它重新有效；如果用户窗体的 Width 属性小于 270 点，就使 cmdNarrowForm 按钮无效。

第 21 行至第 23 行所示的是 cmdClose_Click 过程只是用于关闭用户窗体（通过 Me 关键词指定）。第 25 行至第 27 行所示的是 UserForm_Resize 过程，将用户窗体中的文本框（即 txt1）的 Width 属性，设置成比用户窗体的 Width 小 30 点。如果单步执行对应于该用户窗体的代码，将会注意到，当用户窗体改变其大小时，Resize 事件发生。例如，当 cmdWidenForm_Click 过程中的第 4 行被执行时，执行便跳到第 25 行中的 Resize 事件过程，而这一过程就会在执行第 5 行中的代码之前被执行。

Terminate 事件

当用户窗体已经被卸出——或者更准确地说，当所有对该用户窗体的任何一个实例的引用都已从存储器中去除时，Terminate 事件发生。

对应于 Terminate 事件的语法是：

```
Private Sub UserForm_Terminate()
```

应用于 UserForm 对象和容器控件的事件

本节讨论应用于 UserForm 对象和容器控件的事件。容器控件是指多页控件和框架控件。（Scroll 事件应用于多页、框架和 UserForm 对象，也应用于滚动条控件。）这些事件有：Scroll、Zoom、Resize、Layout、AddControl 和 RemoveControl。

Scroll 事件

Scroll 事件应用于框架控件、多页控件、滚动条控件以及 UserForm 对象。当用户移动框架、多页控件、滚动条或用户窗体上滚动条上的滚动块时，这一事件发生。

对应于 Scroll 事件的语法在应用于上述三种控件和 UserForm 对象时各有不同。应用于

UserForm 对象时，对应于 Scroll 事件的语法是：

```
Private Sub UserForm_Scroll(ByVal ActionX As MSForms.fmScrollAction, ByVal ActionY
As MSForms.fmScrollAction, ByVal RequestDx As Single, ByVal RequestDy As Single,
ByVal ActualDx As MSForms.ReturnSingle, ByVal ActualDy As MSForms.ReturnSingle)
```

应用于滚动条控件时，对应于 Scroll 事件的语法是：

```
Private Sub scrollbar_Scroll()
```

应用于多页控件时，对应于 Scroll 事件的语法是：

```
Private Sub multipage_Scroll(index As Long, ActionX As fmScrollAction, ActionY As
fmScrollAction, ByVal RequestDx As Single, ByVal RequestDy As Single, ByVal
ActualDx As MSForms.ReturnSingle, ByVal ActualDy As MSForms.ReturnSingle)
```

应用于框架控件时，对应于 Scroll 事件的语法是：

```
Private Sub frame_Scroll(ActionX As fmScrollAction, ActionY As fmScrollAction,
ByVal RequestDx As Single, ByVal RequestDy As Single, ByVal ActualDx As
MSForms.ReturnSingle, ByVal ActualDy As MSForms.ReturnSingle)
```

在后三种语法的语句中，scrollbar 是一个有效的 ScrollBar 对象，multipage 是一个有效的 MultiPage 对象，而 frame 是一个有效的 Frame 对象。

下面是与 Scroll 事件对应的参数：

index 是指定与事件过程相关联的 MultiPage 的页面的必需参数。

ActionX 和 **ActionY** 必需参数，分别用于确定如表 15.6 所示的用户的水平方向行动和垂直方向行动。

表 15.6 对应于 Scroll 事件的 ActionX 和 ActionY 常量和值

常量	值	滚动块的移动
fmScrollActionNoChange	0	无变化或无移动
fmScrollActionLineUp	1	用户将垂直滚动条上的滚动块上移了一个很短的路程（等同于按下↑键），或将水平滚动条上的滚动块左移了一个很短的路程（等同于按下←键）
fmScrollActionLineDown	2	用户将垂直滚动条上的滚动块下移了一个很短的路程（等同于按下↓键），或将水平滚动条上的滚动块右移了一个很短的路程（等同于按下→键）
fmScrollActionPageUp	3	用户将垂直滚动条上的滚动块上移了一页（等同于按下 Page Up 键）或将水平滚动条上的滚动块左移了一页（也等同于按下了 Page Up）
fmScrollActionPageDown	4	用户将垂直滚动条上的滚动块下移了一页（等同于按下 Page Down 键），或将水平滚动条上的滚动块右移了一页（也等同于按下了 Page Down 键）
fmScrollActionBegin	5	用户将滚动块移到垂直滚动条顶端，或移到水平滚动条左端
fmScrollActionEnd	6	用于将滚动块移到垂直滚动条底端，或移到水平滚动条右端
fmScrollActionPropertyChange	8	用户改变 ScrollTop 属性或 ScrollLeft 属性的值，从而移动了滚动块
fmScrollActionControlRequest	9	滚动行动是所涉及的容器中的一个控件要求的
fmScrollActionFocusRequest	10	用户将焦点移至一个不同的控件。这一移动使用户窗体滚动，从而被选中的控件完整地显示在可用区中

RequestDx 使滚动块在水平方向上移动的距离，以点数指定。

RequestDy 使滚动块在垂直方向上移动的距离，以点数指定。

ActualDx 滚动块在水平方向上移动过的距离，以点数计量。

ActualDy 滚动块在垂直方向上移动过的距离，以点数计量。

Zoom 事件

在运行阶段改变对象的 Zoom 属性时，Zoom 事件发生。可以通过代码自动地改变 Zoom 属性，也可以通过用户对某一控件的操控，由控件通过代码来改变该属性，但用户不能用人工方法改变 Zoom 属性。

Zoom 属性使用如下的与控件和 UserForm 对象对应的语法：

```
Private Sub object_Zoom(Percent As Integer)
```

此处，object 是一个框架控件或 UserForm 对象。Percent 是一个整型参数，用于指定要把用户窗体放大到多少的百分率（以 10% 至 400%）。按默认方式，用户窗体和控件可以以 100% 的比率放大——即以全尺寸显示出来。

Zoom 属性使用如下的与多页控件对应的语法：

```
Private Sub multipage_Zoom(ByVal Index As Long, Percent As Integer)
```

此处，Index 是与 Zoom 事件过程相关联的多页控件中的 Page 对象的索引（名称或序号）。

将一个用户窗体放大，也就放大了它上面的所有控件。例如，假定一个名为 frmEventsDemo 的用户窗体包含有一个名为 cmbZoom 的组合框，它可提供放大百分率的选定范围。当用户从组合框中选中一项时，对应于 cmbZoom 的 Change 事件，把组合框的 Value 属性应用于用户窗体的 Zoom 属性，同时把用户窗体放大到选择的百分率。

放大用户窗体触发了 Zoom 事件，在下面的程序中 Zoom 事件过程将用户窗体的 Width 和 Height 设置成适合于新的放大百分率的值：

```
Private Sub cmbZoom_Change()
    frmEventsDemo.Zoom = cmbZoom.Value
End Sub

Private Sub UserForm_Zoom(Percent As Integer)
    frmEventsDemo.Width = 300 * cmbZoom.Value / 100
    frmEventsDemo.Height = 350 * cmbZoom.Value / 100
End Sub
```

Layout 事件

当框架、多页控件或用户窗体的大小被改变时，Layout 事件发生。多页控件或用户窗体的大小，既可以用程序方法改变（即自动让自动改变大小的控件成为被调整大小的控件），也可以由用户来改变。

按默认方式，Layout 事件自动为已经移动过的任何控件计算新位置，并相应地重画屏幕。但是，如果需要的话，也可以使用 Layout 事件来达到自己的目的。

用于框架控件或 UserForm 对象时，与 Layout 事件对应的语法是：

```
Private Sub object_Layout()
```

此处的 object 是框架控件或 UserForm 对象。

用于多页控件时，与 Layout 事件对应的语法是：

```
Private Sub multipage_Layout(index As Long)
```

此处的 multipage 是多页控件，index 是多页控件中的 Page 对象。

注意：当调整某一控件的大小时，在以 Height 和 Width 属性承接新的高度和宽度的同时，VBA 把控件原先的高度和宽度保存在 OldHeight 和 OldWidth 属性中。要使控件恢复到原先的大小，可使用这些 OldHeight 和 OldWidth 属性。

AddControl 事件

当某一控件以程序方法在运行阶段添加进框架、多页控件或用户窗体时，AddControl 事件被触发；但在设计阶段以人工方法添加控件时，该事件不被触发。用户窗体初始化时，该事件也不被触发，但 Initialize 事件将控件添加到用户窗体时例外。

对应于 AddControl 事件的语法，依据对象或控件而有变化。与 UserForm 对象和框架控件对应的语法是：

```
Private Sub object_AddControl(ByVal Control As MSForms.Control)
```

此处的 object 是 UserForm 对象或框架控件，而 Control 是正在被加入的控件。

与多页控件对应的语法是：

```
Private Sub multipage_AddControl(ByVal Index As Long, ByVal Control As MSForms.Control)
```

此处的 Index 是将要接收控件的 Page 对象的索引号或索引名。

例如，以下所示的 cmdAddControl_click 过程，把三个选项按钮（分别为 opt1、opt2 和 opt3）添加到框架 fraOptions，并为第一个选项按钮设置属性。（注释行指明了代码将继续为第二个和第三个选项按钮设置属性的地方。）fraOptions_AddControl 事件过程显示一个消息框，其中给出框架目前包含的控件的数目。因为 cmdAddControl_click 过程添加三个控件，所以 AddControl 事件发生三次，即 fraOptions_AddControl 过程也运行三次：

```
Private Sub cmdAddControl_click()
```

```
    Dim opt1 As OptionButton
```

```
    Dim opt2 As OptionButton
```

```
    Dim opt3 As OptionButton
```

```
    Set opt1 = fraOptions.Controls.Add("Forms.OptionButton.1")
```

```
    Set opt2 = fraOptions.Controls.Add("Forms.OptionButton.1")
```

```
    Set opt3 = fraOptions.Controls.Add("Forms.OptionButton.1")
```

```
    With opt1
```

```
        .Left = 10
```

```
        .Top = 10
```

```
        .Name = "optDomestic"
```

```
        .Caption = "Domestic"
```

```
        .AutoSize = True
```

```
        .Accelerator = "D"
```

```

End With
'set properties for opt2 and opt3 here
End Sub

Private Sub fraOptions_AddControl(ByVal Control As MSForms.Control)
    MsgBox "The frame now contains " & _
        fraOptions.Controls.Count & " controls."
End Sub

```

RemoveControl 事件

当某一控件以程序方法或人工方法在运行阶段从所涉及的框架、多页控件或用户窗体中被删除时，RemoveControl 事件发生。（为了用人工方法移出控件，用户通常使用内置于用户窗体的一个控件来达到目的。）

与 RemoveControl 事件对应的语法表示如下，它对多页控件之外的所有控件均适用：

```
Private Sub object_RemoveControl(ByVal Control As MSForms.Control)
```

此处的 object 是一个有效对象，而 Control 是一个有效控件。

用于多页控件的与 RemoveControl 事件对应的语法是：

```
Private Sub multipage_RemoveControl(ByVal Index As Long, ByVal Control As
MSForms.Control)
```

此处的 multipage 是一个有效的 MultiPage 对象。对于多页控件，Index 指明包含要删除的控件的多页控件中的 Page 对象。

应用于大多数控件的事件

本节讨论应用于大多数或所有控件的事件。在这些事件中，有一些也应用于 UserForm 对象。这些事件有：Click、Change、Enter 和 Exit；BeforeUpdate 和 AfterUpdate；KeyDown、KeyUp 和KeyPress；MouseDown、MouseUp 和MouseMove；BeforeDragOver；BeforeDropOrPaste；DblClick 以及 Error。

Click 事件

Click 事件应用于复选框、组合框、命令按钮、框架、图像、标签、列表框、多页、选项按钮、TabStrip 以及切换按钮控件。它不应用于文本框控件、滚动条控件或微调按钮控件，但是它应用于 UserForm 对象。

当用户以主鼠标按钮单击某一控件时，Click 事件发生。当用户为具有多个可能值的控件选择一个值时，也有 Click 事件发生。对于大多数控件来说，这意味着，每当用户单击控件时，该事件发生。但是也有几个例外：

- ◆ 单击无效控件时引起用户窗体的 Click 事件发生（好像用户是通过该控件来单击用户窗体）。
- ◆ 当用户单击选项按钮以选中它时，选项按钮控件的 Click 事件发生。如果该选项按钮已经被选中，单击它没有效果。（另一方面，每当用户单击复选框时——不管是选中它还是清除它，都有复选框选件的 Click 事件发生。）
- ◆ 当用户单击以便从列表中选择一个项目时，列表框控件或组合框控件的 Click 事件发

发生（当用户在下拉列表箭头上单击或在组合框的无下拉部分单击时例外）。如果用户单击一个已选择项目，Click 事件不会再发生一次。

- ◆ 每当切换按钮被单击时以及切换按钮的 Value 属性被改变时，切换按钮控件的 Click 事件发生。这意味着，使用切换按钮控件的 Click 事件来切换它的值，不是一个好方法。
- ◆ 当按下空格键时，被选择的命令按钮控件的 Click 事件发生。
- ◆ 当用户按下 Enter 键但没有其他命令按钮被选中时，默认命令按钮（其 Default 属性被设置为 True 的按钮）的 Click 事件发生。
- ◆ 当用户按下 Esc 键时，其 Cancel 属性被设置为 True 的命令按钮的 Click 事件发生。当用户按下某个加速键时，与使用该加速键的控件对应的 Click 事件也发生。

除了 TabStrip 控件和多页控件之外，对于其他所有控件来说，Click 事件都不需要参数，可表示如下：

```
Private Sub object_Click()
```

对于 TabStrip 控件或多页控件，用户会需要 Index 参数，这是一个必需的长整型参数。VBA 传递该参数，以便指明受影响的控件的标签或页面：

```
Private Sub object_Click(ByVal Index As Long)
```

此处的 object 是一个有效的多页控件或 TabStrip 控件。

事件的顺序：用户单击（和再次单击）时有何事发生

当用户单击某一命令按钮时，如果这次单击把焦点转换到该命令按钮上来，在发生 Click 事件之前，会有对应于该命令按钮的 Enter 事件发生。当此 Enter 事件发生时，它通常会使 Click 事件不发生。

当用户单击某一控件时，被触发的第一个事件是 MouseDown 事件，它在用户按下鼠标按钮时发生，然后 MouseUp 事件在用户释放鼠标按钮时发生，而 Click 事件发生在 MouseUp 事件之后。如果用户在 Windows 中的双击时间范围内再次单击，DblClick 事件会发生，随后便是另一个 MouseUp 事件。

Change 事件

Change 事件应用于复选框、组合框、列表框、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框以及切换按钮控件。当控件的 Value 属性改变时，该事件发生。这种改变可以通过用户的某个行动产生（如选择了一个选项按钮，选择或清除了一个复选框，单击了一个切换按钮，或改变了在多页控件上显示的页面），或通过运行时以程序方式采取的行动来产生。需要记住的是，当用户的某一行动使 Change 事件发生时，这个行动可能也会触发一个 Click 事件。（即使此事发生，较之 Click，Change 仍被看做是确定控件的新 Value 属性值的更佳方式——尽管在很多场合下，Click 也会工作得令人满意。）

注意：在设计时人工改变某一控件的 Value 属性，不会使 Change 事件发生。

对应于 Change 事件的语法是：

```
Private Sub object_Change()
```

在用户改变了某一控件之后，又需要更新其他控件时，Change 事件是有用的。例如，如果用户把一个新报告的名称输进了文本框（这里是 txtReportName），可以使用 Change 事件在另外的文本框里（这里是 txtFileName）构建保存该报告的文件名。

```
Private Sub txtReportName_Change()
    txtFileName.Text = txtReportName.Text & ".txt"
End Sub
```

Enter 和 Exit 事件

Enter 和 Exit 事件应用于复选框、组合框、命令按钮、框架、列表框、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框以及切换按钮控件。

当焦点从用户窗体上的一个控件移到另一个控件时，Enter 事件发生，该事件是在第二个控件即将接收焦点之前发生的。

与 Enter 事件相似，当焦点从用户窗体上的一个控件移到另一个控件时，Exit 事件发生，但是 Exit 事件是在第一个控件即将失去焦点时发生的。

对应于 Enter 事件的语法是：

```
Private Sub object_Enter()
```

对应于 Exit 事件的语法要复杂一些：

```
Private Sub object_Exit(ByVal Cancel As MSForms.ReturnBoolean)
```

此处的 Cancel 是指明事件状态的必需参数。默认设置为 False，它指明所涉及的控件应该处理该事件，而且焦点将传递到下一控件；如果设置为 True，则指明由应用程序来处理该事件，使焦点保持在当前控件上。

使用 Enter 和 Exit 事件，可以通过用户窗体上的控件追踪用户的进展。

Exit 事件，可以用来确认控件上的选择或者输入了一个合适的值。例如，可以检查控件的输入值，若不正确，则显示一个对话框指出错误，然后返回该控件，用户可以重新输入。

注意：在用户已经访问了某一控件之后，为了检查该控件的内容，可能要使用其他事件，包括 AfterUpdate 和 LostFocus。类似地，可能会使用 BeforeUpdate 和 GotFocus 事件，而不用 Enter 事件。Enter 与 GotFocus 之间——以及 Exit 与 LostFocus 之间——的重大区别在于：当用户窗体收到或失去焦点时，GotFocus 和 LostFocus 分别发生，但 Enter 和 Exit 不发生。

BeforeUpdate 事件

BeforeUpdate 事件应用于复选框、组合框、列表框、选项按钮、滚动条、微调按钮、文本框以及切换按钮控件。当所指定的控件的值或其内的数据有改变时，这个事件发生；可以使用该事件来判断这种变化，并决定是否要实现它。

对应于 BeforeUpdate 事件的语法是：

```
Private Sub object_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)
```

此处的 object 是一个有效对象，Cancel 是指明事件状态的必需参数。默认设置为 False，使得该控件来处理此事件；若为 True，则使这一更新免于执行，并让应用程序来处理此事件。

当要移到一个控件，对它更新然后继续前行时，各事件按如下顺序发生：

- ◆ 把焦点移到这个控件上时，对应于该控件的 Enter 事件发生。
- ◆ 在已经为更新输入了信息之后（例如，已经在文本框内按了一个键之后）但在执行这一更新之前，对应于该控件的 BeforeUpdate 事件发生。将 Cancel 设置为 True，可使这一更新免于发生。（如果没有将 Cancel 设置为 True，更新会发生。AfterUpdate 事件不能避免它的发生。）
- ◆ 在已经将信息输进了控件，更新已经执行之后，对应于该控件的 AfterUpdate 事件发生。如果已将对应于 BeforeUpdate 的 Cancel 参数设置为 True，AfterUpdate 事件不发生。
- ◆ 当从该控件移至另一控件时，对应于该控件的 Exit 事件发生。（与已离开的控件对应的 Exit 事件发生之后，与焦点已移至其上的控件对应的 Enter 事件发生。）

AfterUpdate 事件

AfterUpdate 事件应用于复选框、组合框、列表框、选项按钮、滚动条、微调按钮、文本框以及切换按钮控件。当用户改变了某控件中的信息且这种更新已经被执行之后，这一事件发生。

对于受到该事件作用的所有控件和对象，与 AfterUpdate 事件对应的语法都是相同的：

```
Private Sub object_AfterUpdate()
```

KeyDown 和 KeyUp 事件

KeyDown 和 KeyUp 事件应用于复选框、组合框、命令按钮、框架、列表框、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框以及切换按钮控件，还有 UserForm 对象。这些事件不应用于图像和标签控件。

当用户在键盘上按下一个键时，KeyDown 事件发生。当用户复原该键时，KeyUp 事件发生。通过使用 SendKeys 语句，以程序方法把一个键送到用户窗体或控件时，KeyDown 和 KeyUp 事件也发生。如果用户窗体中包含有 Default 属性被设置为 True 的命令按钮，当用户按下 Enter 键时，这些事件不发生。如果用户窗体中包含有 Cancel 属性被设置为 True 的命令按钮，当用户按下 Esc 键时，这些事件也不会发生。

当敲键以使焦点移至另一控件时，对应于原先控件的 KeyUp 事件发生，而对应于焦点移至其上的控件的KeyPress 和 KeyDown 事件也发生。

注意：KeyPress 事件发生于KeyDown 事件之后和KeyUp 事件之前。

与 KeyDown 事件对应的语法是：

```
Private Sub object_KeyDown(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
```

与 KeyUp 事件对应的语法是：

```
Private Sub object_KeyUp(ByVal KeyCode As MSForms.ReturnInteger, ByVal Shift As Integer)
```

此处的 object 是必需的对象名称。KeyCode 是必需的整型参数，它指明被按下的键代

码。例如，字母 t 的键代码是 84。键代码不是 ANSI 值——它是用来识别键盘上的键的一个特殊数。

Shift 是必需参数，它指明 Shift 键、Ctrl 键或 Alt 键是否被按下。可以使用的 Shift 常量和值如表 15.7 所示。

表 15.7 Shift 常量和值

常量	值	说明
fmShiftMask	1	Shift 键被按下
fmCtrlMask	2	Ctrl 键被按下
fmAltMask	4	Alt 键被按下

KeyPress 事件

KeyPress 事件应用于复选框、组合框、命令按钮、框架、列表框、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框以及切换按钮控件。另外，它也应用于 UserForm 对象，但不应用于标签控件。

当用户按下一个 ANSI 键——一个可印出字符，Ctrl 加上一个字母字符，Ctrl 加上一个专用字符，Esc 键或 Backspace 键——而所涉及的控件或对象又具有焦点时，KeyPress 事件发生。按下 Tab 键、Enter 键或箭头键，不会引起 KeyPress 事件发生。通过敲键使焦点从当前控件移至另一控件时，这种键的敲击也不会引起 KeyPress 事件发生。

Delete 键不是 ANSI 键，所以按下 Delete 键来删除某内容（例如删除文本框中的文本），不会使 KeyPress 事件发生。但是，使用 Backspace 键删除与此相同的文本框中的相同文本，则会使 KeyPress 事件发生，因为 Backspace 是一个 ANSI 键。

注意：KeyPress 事件在 KeyDown 事件之后和 KeyUp 事件之前发生。当使用

SendKeys 以程序方法将键敲击发送到一个用户窗体时，KeyPress 事件也发生。

对应于 KeyPress 事件的语法是：

```
Private Sub object_KeyPress(ByVal KeyAscii As MSForms.ReturnInteger)
```

此处的 object 是必需参数，它指定一个有效的对象。KeyAscii 是必需的整型参数，用于指定一个 ANSI 键代码。为了得到 ANSI 键代码，需使用 Asc 函数。例如，Asc ("t") 返回字母 t 的 ANSI 键代码（此代码为 116）。

按默认方式，KeyPress 事件处理与被按下键对应的代码——用一句通俗的话来说，按啥得啥。例如，按下 t 键，就得到一个 t；按下 Delete 键，就得到一个删除行动；等等。通过使用 KeyPress 事件过程，可以进行检查（例如，在用户必须输入一个数字键时，把所有非数字键滤除掉）。

MouseDown 和 MouseUp 事件

MouseDown 和 MouseUp 事件应用于复选框、组合框、命令按钮、框架、图像、标签、列表框、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框和切换按钮控件，以及 UserForm 对象。当用户按下鼠标上的一个按钮时，MouseDown 事件发生，而当用户释放

该按钮时，`MouseUp` 事件发生。`Click` 事件发生在 `MouseUp` 事件发生之后。

应用于多页控件和 TabStrip 控件之外的所有控件时，与 `MouseDown` 和 `MouseUp` 事件对应的语法是：

```
Private Sub object_MouseDown(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
```

```
Private Sub object_MouseUp(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
```

应用于多页控件和 TabStrip 控件时，与 `MouseDown` 和 `MouseUp` 事件对应的语法中要添加一个 `Index` 参数，以便指定所涉及的页面或标签的索引：

```
Private Sub object_MouseUp(ByVal Index As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
```

```
Private Sub object_MouseDown(ByVal Index As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
```

此处的 `object` 是一个有效对象。

如果用户在控件的页面或标签区之外，但仍属该控件之内（例如标签条顶行中最右边一个标签的右面）单击，`Index` 返回 -1。

`Button` 是一个必需的整型参数，它指明引起该事件的鼠标按钮。表 15.8 列出了 `Button` 可能的值。

表 15.8 Button 常量和值

常量	值	说明
<code>fmButtonLeft</code>	1	按下左按钮（主按钮）
<code>fmButtonRight</code>	2	按下右按钮（非主按钮）
<code>fmButtonMiddle</code>	4	按下中间按钮

`Shift` 是一个必需的参数，它指明 `Shift` 键、`Ctrl` 键或 `Alt` 键是否被按下。表 15.9 列出与 `Shift` 对应的值。

表 15.9 Shift 值

Shift 值	按下的键
1	Shift
2	Ctrl
3	Shift+Ctrl
4	Alt
5	Alt+Shift
6	Alt+Ctrl
7	Alt+Shift+Ctrl

也可以使用本章前面的表 15.7 来检测按下的单个 Shift 键、Ctrl 键或 Alt 键。X 是必需的单精度浮点型参数，它指明离用户窗体、框架或页面的左边缘的水平位置，以点计。Y 是必需的单精度浮点型参数，它指明离用户窗体、框架或页面的顶边的垂直位置，以点计。

MouseMove 事件

MouseMove 事件应用于复选框、组合框、命令按钮、框架、图像、标签、列表框、多页、选项按钮、TabStrip、文本框和切换按钮控件，以及 UserForm 对象。当用户使鼠标在所涉及的控件或对象上方移过时，这一事件发生。

与 MouseMove 事件对应的语法在用于多页控件和 TabStrip 控件时，和用于其他控件及 UserForm 对象时不相同。用于其他控件时的语法为：

```
Private Sub object_MouseMove(ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
```

用于多页控件和 TabStrip 控件时的语法为：

```
Private Sub object_MouseMove(ByVal Index As Long, ByVal Button As Integer, ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
```

此处 object 是一个必需的参数，它指定一个有效对象。

对于多页控件和 TabStrip 控件，Index 是一个必需的参数，它返回与该事件过程相关联的多页控件中的 Page 对象或 TabStrip 控件中的 Tab 对象。

Button 是一个必需的整型参数，它返回用户按下的鼠标按钮（如有的话）。表 15.10 列出了 Button 对应的值。

表 15.10 Button 的值

Button 值	按下的按键
0	无
1	左按钮
2	右按钮
3	左按钮和右按钮
4	中间按钮
5	左按钮和中间按钮
6	中间按钮和右按钮
7	左按钮、中间按钮和右按钮

Shift 是一个必需的整型参数，它返回一个值，指明用户是否按下了 Shift 键、Alt 键和 Ctrl 键，请参阅表 15.9 的 Shift 值列表。

X 是必需的单精度浮点型参数，它返回一个值，指明离用户窗体、框架或页面的左边缘的水平位置，以点计。Y 是必需的单精度浮点型参数，它指明离用户窗体、框架或页面的顶边的垂直位置，以点计。

同使用 MouseDown 和 MouseUp 事件一样，也可以使用本章前面的表 15.7 来检测按下的单个键。

同 Windows 一样，用户窗体总是无休止地同鼠标事件打交道。MouseMove 事件监视鼠标指针在屏幕上的位置以及哪一个控件捕获了它。即使使用键盘从鼠标指针之下来移动用户

窗体，也有 MouseMove 事件发生，这是因为，尽管鼠标指针没有做寻常意义上的移动，但是鼠标指针仍是停止在与用户窗体有关的一个不同位置上。

MouseMove 事件的一个应用是在用户正对准某一控件时，显示与该控件对应的适当的文本或一个图像。例如，假使用户窗体提供了一个可用产品的列表，其中每个产品的标题都出现在一个标签中，那么当用户使鼠标指针定位于该标签中某一个标题的上方时，可以使用 MouseMove 事件将该产品的一个图片加载进图像控件，将一个简短的说明加载进另一个标签。

注意：当鼠标指针在任何控件上方移动时，用户窗体捕获 MouseMove 事件。但是，如果用户使鼠标指针迅速地从一个控件移到另一个非常接近它的控件，那么用户窗体可能会捕获不到这种很短的介入空间之内的移动。

BeforeDragOver 事件

BeforeDragOver 事件应用于 UserForm 对象自身以及下述控件：复选框、组合框、命令按钮、框架、图像、标签、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框和切换按钮。当用户实施一个拖放动作时，BeforeDragOver 事件发生。

对应于 BeforeDragOver 事件的语法，取决于所涉及的对象或控件。用于 UserForm 对象、框架、TabStrip 与多页之外的所有控件的语法表示如下，其中的 object 是一个有效的 UserForm 对象或控件：

```
Private Sub object_BeforeDragOver(ByVal Cancel As MSForms.ReturnBoolean, ByVal Control As MSForms.Control, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal State As MSForms.fmDragState, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

用于框架控件的 BeforeDragOver 事件的语法表示如下，其中的 frame 是一个有效的框架控件：

```
Private Sub frame_BeforeDragOver(ByVal Cancel As MSForms.ReturnBoolean, ByVal Control As MSForms.Control, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal State As MSForms.fmDragState, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

用于多页控件的 BeforeDragOver 事件的语法表示如下，其中的 multipage 是一个有效的多页控件：

```
Private Sub multipage_BeforeDragOver(ByVal Index As Long, ByVal Cancel As MSForms.ReturnBoolean, ByVal Control As MSForms.Control, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal State As MSForms.fmDragState, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

用于 TabStrip 控件的 BeforeDragOver 事件的语法表示如下，其中的 tabstrip 是一个有效的 TabStrip 控件：

```
Private Sub tabstrip_BeforeDragOver(ByVal Index As Long, ByVal Cancel As MSForms.ReturnBoolean, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal DragState As MSForms.fmDragState, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

以下是各语句的不同部分：

- ◆ Index 是受拖放影响的多页控件中的 Page 对象的索引或 TabStrip 控件中的 Tab 对象的索引。
- ◆ Cancel 是一个必需的参数，它给出 BeforeDragOver 事件的状态。默认设置为 False，它让控件来处理此事件。若设置为 True，则由应用程序来处理此事件。
- ◆ Control 是一个必需的参数，它指明正在被拖曳的控件。
- ◆ Data 是一个必需的参数，它指明正在被拖曳的数据。
- ◆ X 是必需参数，它指定离控件左边缘的水平距离，以点计。Y 是必需参数，它指定离控件顶边的垂直距离，以点计。
- ◆ DragState 是必需参数，它指定鼠标指针相对于一个目标的位置（数据可放在此处）。表 15.11 列出了与 DragState 对应的常量和值。
- ◆ Effect 是必需参数，它指定被拖放的源所支持的操作，如表 15.12 所示。
- ◆ Shift 是必需参数，它指明在拖放操作期间，Shift 键、Ctrl 键或 Alt 键是否保持按下，如表 15.7 所示（本章前面）。

表 15.11 DragState 常量和值

常量	值	鼠标指针的位置
fmDragStateEnter	0	在一个目标的区域之内
fmDragStateLeave	1	在一个目标的区域之外
fmDragStateOver	2	在一个新位置上，但仍在同一目标的区域之内

表 15.12 Effect 常量和值

常量	值	拖放效果
fmDropEffectNone	0	既不复制也不移动
fmDropEffectCopy	1	将源复制到目标
fmDropEffectMove	2	将源移动到目标
fmDropEffectCopyOrMove	3	将源复制或移动到目标

使用 BeforeDragOver 事件来控制用户实施的拖放行动。使用 DragState 参数来确认鼠标指针是否在一个目标的区域之内。

BeforeDropOrPaste 事件

BeforeDropOrPaste 事件应用于复选框、组合框、命令按钮、框架、图像、标签、列表框、多页、选项按钮、滚动条、微调按钮、TabStrip、文本框和切换按钮，以及 UserForm 对象。

用户即将在一个对象上放下数据或粘贴数据之前，BeforeDropOrPaste 事件发生。

与 BeforeDropOrPaste 事件对应的语法在用于多页控件和 TabStrip 控件时，和用于 UserForm 对象及其他控件时不相同，其基本语法是：

```
Private Sub object_BeforeDropOrPaste(ByVal Cancel As MSForms.ReturnBoolean, ByVal Control As MSForms.Control, ByVal Action As MSForms.fmAction, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

用于多页控件时的语法如下，其中的 multipage 是一个有效的多页控件：

```
Private Sub multipage_BeforeDropOrPaste(ByVal Index As Long, ByVal Cancel As MSForms.ReturnBoolean, ByVal Control As MSForms.Control, ByVal Action As MSForms.fmAction, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

用于 TabStrip 控件的语法如下，其中的 tabstrip 是一个有效的 TabStrip 控件：

```
Private Sub tabstrip_BeforeDropOrPaste(ByVal Index As Long, ByVal Cancel As MSForms.ReturnBoolean, ByVal Action As MSForms.fmAction, ByVal Data As MSForms.DataObject, ByVal X As Single, ByVal Y As Single, ByVal Effect As MSForms.ReturnEffect, ByVal Shift As Integer)
```

以下是语法中的一些部分：

- ◆ object 是必需参数，指定一个有效对象。
- ◆ 对于多页控件，Index 是必需参数，它指定涉及的 Page 对象。
- ◆ Cancel 是必需参数，它给出该事件的状态。默认设置为 False，由控件来处理该事件；若设置为 True，则由应用程序来处理该事件。
- ◆ Control 是必需参数，它指明目标控件。
- ◆ Action 是必需参数，它指明拖放操作的结果。表 15.13 列出了对应于 Action 的常量和值。

表 15.13 Action 常量和值

Action 常量	值	采取的行动
fmActionPaste	2	将对象粘贴进目标
fmActionDragDrop	3	用户已将目标从它的源拖放到目标上

- ◆ Data 是必需参数，它指明正在拖放的数据（放在一个 DataObject 中）。
- ◆ X 是必需参数，它指定离要放下的控件的左边缘的水平距离，以点计。Y 是必需参数，它指定离该控件的顶边的垂直距离，以点计。
- ◆ Effect 是必需参数，它指明该拖放操作是复制数据还是移动数据，见表 15.12。
- Shift 是必需参数，它指明用户是否按下了 Shift 键、Ctrl 键和 Alt 键，见表 15.7。

当一个数据对象被传送到多页控件或 TabStrip 控件时，在即将于其他控件上实施放下或粘贴操作之前，BeforeDropOrPaste 事件发生。

DblClick 事件

DblClick 事件应用于复选框、组合框、命令按钮、框架、图像、标签、列表框、多页、选项按钮、TabStrip、文本框和切换按钮控件。也应用于 UserForm 对象。当用户以主鼠标按钮双击某一控件或对象时，DblClick 事件发生。双击必须足够快，以便作为一次双击而记入 Windows（这个速度可以通过在“鼠标属性”对话框中进行设置来控制），而且双击在 MouseDown 事件、MouseUp 事件和 Click 事件（对于支持 Click 事件的控件）之后发生。

与 DblClick 事件对应的语法在用于多页控件和 TabStrip 控件时，和用于其他控件和用户窗体时不相同。用于多页控件和 TabStrip 控件时，其语法是：

```
Private Sub object_DblClick(ByVal Index As Long, ByVal Cancel As MSForms.ReturnBoolean)
```

用于其他控件时，其语法是：

```
Private Sub object_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
```

此处，object 是必需参数，指定一个有效对象。对于多页控件和 TabStrip 控件，Index 是必需参数，它指明与该事件过程相关联的多页控件中的 Page 对象或 TabStrip 控件中的 Tab 对象。

Cancel 是必需参数，它指定该事件的状态。默认设置为 False，引起该控件来处理这一事件；若设置为 True，则由应用程序来处理此事件，并使该控件忽略第二次单击。

在既支持 Click 事件又支持 DblClick 事件的各控件中，Click 事件发生在 DblClick 事件之前。如果使用 Click 事件过程来采取一次界面行动（例如显示一个消息框），它会阻塞 DblClick 事件过程，使之不运行。在下例中，DblClick 事件过程不运行：

```
Private Sub CommandButton1_Click()
    MsgBox "Click event"
End Sub

Private Sub CommandButton1_DblClick _
    (ByVal Cancel As MSForms.ReturnBoolean)
    MsgBox "Double-click event"
End Sub
```

但是，在 Click 事件过程中执行非界面语句并不阻塞 DblClick 事件过程。下例在对应于用户窗体的代码表的声明区内，声明一个名为 strMessage 的私有字符串变量。对于 CommandButton1 命令按钮的 Click 事件过程，将文本指定给 strMess；DblClick 事件过程将更大些的文本指定给 strMessage，并显示包含 strMessage 的消息框，所以可以看到两个事件均已发生。不要逐语句执行这些代码——只是运行它，否则它会不工作：

```
Private strMess As String

Private Sub CommandButton1_Click()
    strMess = "Click event" & vbCrLf
End Sub

Private Sub CommandButton1_DblClick _
    (ByVal Cancel As MSForms.ReturnBoolean)
    strMessage = strMessage & "Double-click event"
    MsgBox strMessage
End Sub
```

对于大多数控件，不要希望既使用 Click 事件过程，又使用 DblClick 事件过程——视控件的需要，从中选择一个。

Error 事件

Error 事件应用于复选框、组合框、命令按钮、框架、图像、标签、列表框、多页、选

项按钮、滚动条、微调按钮、TabStrip、文本框和切换按钮控件，也应用于 UserForm 对象。当控件遇到一个错误而且不能将关于错误的信息返回到调用该控件的程序时，Error 事件发生。

用于 UserForm 对象和多页控件之外的其他所有控件时，与 Error 事件对应的语法是：

```
Private Sub object_Error(ByVal Number As Integer, ByVal Description As  
MSForms.ReturnString, ByVal SCode As Long, ByVal Source As String, ByVal HelpFile  
As String, ByVal HelpContext As Long, ByVal CancelDisplay As MSForms.ReturnBoolean)
```

用于多页控件时，与 Error 事件对应的语法如下，其中 multipage 是一个有效的多页控件：

```
Private Sub multipage_Error(ByVal Index As Long, ByVal Number As Integer, ByVal  
Description As MSForms.ReturnString, ByVal SCode As Long, ByVal Source As String,  
ByVal HelpFile As String, ByVal HelpContext As Long, ByVal CancelDisplay As  
MSForms.ReturnBoolean)
```

以下是语法的组成部分：

- ◆ object 是一个有效对象的名称。
- ◆ 对于多页控件，Index 是与该事件相关联的多页控件中 Page 对象的索引。
- ◆ Number 是必需参数，它返回控件用来识别此错误的值。
- ◆ Description 是必需的字符串参数，用来描述此错误。
- ◆ SCode 是必需参数，它给出对应于此错误的 OLE 状态代码。
- ◆ Source 是必需的字符串参数，它包含识别所涉及的控件的字符串。
- ◆ HelpFile 是必需的字符串参数，它包含放有 Description 的帮助文件的完整路径。
- ◆ HelpContext 是必需的长整型参数，它包含帮助文件内对应于 Description 的上下文标识符。
- ◆ CancelDisplay 是必需的布尔型参数，它控制 VBA 是否在消息框里显示出错消息。

仅仅应用于极少数控件的事件

本节将讨论三个事件，它们仅仅应用于一个或两个控件。第一个是 DropButtonClick 事件，它仅应用于组合框控件和文本框控件；第二个和第三个是 SpinUp 事件和 SpinDown 事件，它们只用于微调按钮控件。

DropButtonClick 事件

当用户单击下拉按钮或按下 F4 键来显示或隐藏一个组合框上的下拉列表时，DropButtonClick 事件发生。当用户选择了一个文本框控件，并按下 F4 键时，DropButtonClick 事件也发生。在 VBA 中执行 DropDown 方法以显示下拉列表时以及再次执行 DropDown 方法以隐藏此下拉列表时也都会有该事件发生。

对应于 DropButtonClick 事件的语法是：

```
Private Sub object_DropButtonClick( )
```

此处的 object 是一个有效的组合框或文本框控件。

DropButtonClick 事件的一个应用是向组合框控件添加项目，以便取代在加载时使用

Initialize 事件来添加这些项目。只在需要时才添加这些项目（假定用户完全可以不使用组合框，或者可以向它的文本框区域键入信息），便能够缩短对用户窗体加载的时间。还可以使用与用户在对话框中做出的选择有关的数据来加载组合框，这与以 Initialize 事件加载组合框相比，更能够提供针对性强的信息。

SpinDown 和 SpinUp 事件

SpinDown 和 SpinUp 事件只应用于微调按钮。SpinDown 和 SpinUp 用来控制当用户分别单击垂直微调按钮控件的向下箭头按钮和向上箭头按钮时，或者用户分别单击水平微调按钮控件的向右箭头按钮和向左箭头按钮时，会有什么事发生。当用户单击向下箭头或向右箭头按钮时，SpinDown 事件发生；而当用户单击向上箭头或向左箭头按钮时，SpinUp 事件发生。

对应于 SpinDown 和 SpinUp 事件的语法是：

```
Private Sub spinbutton_SpinDown()
Private Sub spinbutton_SpinUp()
```

此处的 spinbutton 是一个微调按钮控件。

按默认方式，使用 SmallChange 增量，让 SpinDown 事件来减小微调按钮的 Value 属性，让 SpinUp 事件来增大此属性。

第五部分 生成有效的代码

- ◆ 第 16 章 构建模块及使用类
- ◆ 第 17 章 调试代码和处理错误
- ◆ 第 18 章 构建出色的代码
- ◆ 第 19 章 用 VBA 的安全特性保护代码

第 16 章 构建模块及使用类

- ◆ 模块化代码：什么、何处、为何
- ◆ 模块中的代码布局
- ◆ 调用程序
- ◆ 代码的逻辑和布局改进
- ◆ 信息在程序之间传递
- ◆ 了解类及其作用
- ◆ 生成对象类
- ◆ 生成类的属性
- ◆ 生成类的方法
- ◆ 使用类

本章介绍如何构建模块化代码——代码各自独立而非一个整块。还将介绍怎样生成在其他程序中反复使用的代码。

本章第二节讨论怎样构建和使用自己的类以实施自己的对象、保存信息和返回信息。

构建模块化代码

到目前为止，本书的代码很有效，但大都不够简洁，不够完美。现在介绍怎样对代码进行改进。

注意：计算机代码称之为完美不仅要没有错误、结构完整、界面设计漂亮，而且不可以拖泥带水，应当以最简洁的方式实现功能。

什么是模块化代码

模块化代码指可以结合使用各种程序的代码。该名称并不是因为 VBA 的代码都放在模块中（尽管实际上仍然需要这样做）。

例如，使用 Word 软件。可以采用一次性方法，编写一个完整的程序，按选择的模板生成文档，针对文档进行处理（如插入文本并进行格式处理），按给定的名称存入特定的目录，在指定的打印机上打印出来，然后关闭文档。

另一种选择是采用模块化方法编写几个独立的程序（过程）：一个用来按选择的模板生成文档，一个用来处理文本与格式，一个用来保存文档，一个用来在指定的打印机上打印，一个用来关闭文档。然后编写一个程序调用这些独立的程序实现单个完整程序的同样功能。这些单独的过程可以以另一种方式组合在另外的程序中实现不同的功能。

模块化代码的优点

模块化代码和把所有内容组合在一起的代码项相比，它有几个优点。首先，模块化代码容易写，因为程序较短，每一个功能都很明确。调试起来也比较容易，因为程序短，发现、纠正错误都比较容易。程序还容易读懂，因为简单，容易理解其目的。

模块化代码可以提供更有效的编程方法，理由有四：

- ◆ 把程序分解成模块，不同部分的重复用单独的模块处理，不需要重复写代码。代码少，程序运行就快。
- ◆ 重复调用程序，可以大大减少编写代码的数量。编写数量少，产生错误的机会就小。
- ◆ 如果需要修改，只要修改相应的模块，而在长程序中需要多处修改，往往会丢三落四。模块化代码的修改对所有的调用均有效。
- ◆ 可以在其他程序中调用单独的过程，而不需要把内容写入。试想，VBA 的许多功能需要一个个生成而不能任意使用是多么乏味。自己的代码也同样如此。

怎样编写模块化代码

生成模块化代码因项目不同、程序不同，难处也不同。例如，录制一个宏以针对多个演示文稿完成一次性处理，就不需要担心分解录制宏并将其写成单独的过程。另一方面，如果需要编写程序自动生成工资的预算工作表，用一些单独的过程处理十分有好处。

可以有两种方法生成模块化代码：

- ◆ 按通常的方法录制（若程序支持此项操作）或编写程序，然后验证并分解为独立的模块。这是生成模块化代码的最好方法，但效率不够高。在生成模块时需要花大量的时间进行整理。
- ◆ 对不同的行为进行计划，然后把各个行为生成单独的模块。这种方法需要事先构思但通常行之有效。

模块中的代码布局

一旦生成了过程，就可以在同一个工程中放入不同的模块，甚至在不同的工程中。在模块中组织过程，可以很方便地把程序发给同事，不需要的程序不要放入。在工程中组织模块，可以很方便地发送模块和程序。另外，可以删去不需要使用的模块以避免计算机的速度变慢。

提示：模块的名称应当有意义，可以在组织者对话框或其他模块管理软件中进行识别。避免使用 Module1、Module2 这样的名称，容易产生混乱。

调用过程

在一个过程中调用另一个过程，可以采用第 9 章介绍的调用函数的方法。调用同一工程的过程，用需要调用的过程的名称作为语句，或者名称前加上 Call 的语句。

Call 语句的语法如同函数的语法：

[Call] 名称 [, 参数列表]

这里，“名称”是必选字符参数，指需要调用的过程名称。“参数列表”是可选参数，指

用逗号隔开的变量、数组或表达式。“参数列表”只针对需要使用参数的过程。

例如，下面的 CreateReceiptLetter 过程调用另一个过程：

```
Sub CreateReceiptLetter()
    'other actions here
    Call FormatDocument
    'other actions here
End Sub
```

可以不使用 Call 关键词：

```
Sub CreateReceiptLetter()
    'other actions here
    FormatDocument
    'other actions here
End Sub
```

在下面的例子中，Calling 过程调用 CallMe 过程，后者需要字符串型参数 strFeedMe。注意在调用时需要加上参数：

```
Sub Calling()
    Call CallMe("Hello")
End Sub
```



```
Sub CallMe(ByVal strFeedMe As String)
    MsgBox strFeedMe
End Sub
```

同样，Call 关键词可以省略：

```
Sub Calling2()
    CallMe "Hello"
End Sub
```

和使用函数一样，使用 Call 关键词可使代码更清楚，表明正在调用一个过程，查找调用也比较容易。

正如在同一个工程中调用过程一样，可以调用另一个打开的工程中的过程，但必须是同一个宿主软件（但是通常不在另一个软件中）。虽然不同软件的不同版本的方法不同，但一般来说，调用另一个工程中的过程的语法是：

Project.Module.Procedure

调用另一个工程中的过程，必须在对话框中加上对该工程的引用。选择“工具”>“引用”，再选择工程（如果需要还可使用浏览），然后单击“确定”按钮。如果该引用被选中，就可以调用该过程。

警告：不可以对当前工程引用自己的工程。如果选择这样的引用，在关闭引用对话框时，Visual Basic 编辑器将显示信息框指明不可以循环引用，而且不接受该引用（尽管引用对话框可正常关闭）。

除了可以编写模块化代码，还可以改进代码的逻辑和布局，使代码更加完善，运行得更快。

代码的逻辑改进

将过程分解为子过程可以改进代码的逻辑结构，因为不得不考虑每个子过程的指令，这意味着指令从整套指令中分解出来。改进代码的逻辑结构也可以使用显式变量声明的方法，简化录制的代码，以及用 With 语句减少对象引用的数量。

变量显式声明

所有的变量采取显式声明而不是隐式声明的方法，这就使 VBA 占用的内存按实际需要进行分配。如果指明对象的数据类型，VBA 就不需要每次花时间检查变量的数据类型，更重要的是，这可避免在变量中存储类型不相符的数据。因为变量采取显式声明，所以 VBA 就不会保存该数据或者改变变量的类型从而给出出错提示。

提示：也可采用隐式声明的方法，不过需使用类型声明字符，然而只有采用显式声明，代码才更容易看懂。

表 16.1 显示了不同数据类型需要的内存空间。

表 16.1 不同数据类型需要的内存

变量	需要的内存（字节）
逻辑型	2
字节型	1
货币型	8
日期型	8
不定型/十进制	12
双精度	8
整型	2
长整型	4
对象	4
单精度	4
字符串型	可变长度的字符串：10 字节加上保存该字符串需要的空间，字符数可达到 20 亿 固定长度的字符串：保存该字符串的字节数，字符数量可以从 1 到 64000 个
不定型	保存数字的不定型：16 字节 保存字符的不定型：22 字节加上保存字符需要的空间

指定数据类型需要多少内存，以及选择变量类型可使程序有多大差别取决于不同的处理要求。例如：在变量中保存一百万个字符，指定其为字符串变量而不是不定型变量，可节省 12 字节，但差别很小。然而，如果内存很小，变量很多，指定变量的数据类型就可能节省足够的内存空间使程序得以运行，否则也许不能运行，或者可以运行得更快。

显式声明而非隐式声明变量的另一个理由是：代码容易阅读，容易调试。

显式声明变量的第三个理由是：可以实施一些运行范围的检测。如果知道数据小于 32768，就可以声明为整型变量，如果使用了长整型变量，就会在运行时自动得到出错提示。

简化录制宏

宏录制器（微软公司使用它）提供了极好的方法学习如何生成代码，可以很快了解程序需要使用的对象以及需要使用的方法与属性。然而，正如前面的介绍，宏录制器的缺点是录制了很多程序中并不需要的代码，因为录制的命令和设定必须完整。例如，要

录制在“字体”对话框中改变设定的过程，宏录制器会记录下所有其他的设定，不仅包括字体还包括其他内容，假定全都需要。

一旦录制结束，往往需要进行修改，增加循环、决策、自定义内容（信息框、输入框、自定义窗体），甚至提取部分代码用于其他程序。进行这样处理时，必须审查宏录制器记录的代码，若有可能，应简化代码，只留下需要使用的语句。

以 Word 为例，比较下面两段程序：

```
Sub Recorded_Macro_Applying_Arial_Font()
```

```
    Recorded_Macro_Applying_Arial_Font
```

```
    Macro recorded 7/07/06 by Peter Beier
```

```
    With Selection.Font
```

```
        .Name = "Arial"
```

```
        .Size = 10
```

```
        .Bold = False
```

```
        .Italic = False
```

```
        .Underline = wdUnderlineNone
```

```
        .StrikeThrough = False
```

```
        .DoubleStrikeThrough = False
```

```
        .Outline = False
```

```
        .Emboss = False
```

```
        .Shadow = False
```

```
        .Hidden = False
```

```
        .SmallCaps = False
```

```
        .AllCaps = False
```

```
        .Color = wdColorAutomatic
```

```
        .Engrave = False
```

```
        .Superscript = False
```

```
        .Subscript = False
```

```
        .Spacing = 0
```

```
        .Scaling = 100
```

```
        .Position = 0
```

```
        .Kerning = 0
```

```
        .Animation = wdAnimationNone
```

```
    End With
```

```
End Sub
```

```
Sub Stripped_Down_Procedure_Applying_Arial_Font()
```

```
    Selection.Font.Name = "Arial"
```

```
End Sub
```

从中可以看出，程序 Stripped_Down_Procedure_Applying_Arial_Font 的效果和录制的程序完全相同，但它只有 3 行，而录制程序有 30 行。

用 With 语句简化程序

如果用对象执行多个命令，就可以使用 With 语句减少引用对象的次数。这样做可简化代码并且大大加快运行速度。如果要在单个对象中处理多个对象，可使用独立的 With 语句，或者针对共同需要使用的最低一级对象采用嵌套 With 语句。

例如，下面的语句包括多层引用，对象为 Word 软件中 ActiveDocument 对象的第一个 Paragraph 对象——Paragraphs(1)：

```

ActiveDocument.Paragraphs(1).Range.Font.Bold = True
ActiveDocument.Paragraphs(1).Range.Font.Name = "Times New Roman"
ActiveDocument.Paragraphs(1).LineSpacingRule = wdLineSpaceSingle
ActiveDocument.Paragraphs(1).Borders(1).LineStyle = wdLineStyleDouble
ActiveDocument.Paragraphs(1).Borders(1).ColorIndex = wdBlue

```

然而，可使用 With 语句引用 ActiveDocument 对象中的 Paragraphs (1) 对象来简化引用的数量：

```

With ActiveDocument.Paragraphs(1)
    .Range.Font.Bold = True
    .Range.Font.Name = "Times New Roman"
    .LineSpacingRule = wdLineSpaceSingle
    .Borders(1).LineStyle = wdLineStyleDouble
    .Borders(1).ColorIndex = wdBlue
End With

```

还可以用嵌套的 With 语句进一步简化 Range 对象中的 Font 对象和 Borders (1) 对象：

```

With ActiveDocument.Paragraphs(1)
    With .Range.Font
        .Bold = True
        .Name = "Times New Roman"
    End With
    .LineSpacingRule = wdLineSpaceSingle
    With .Borders(1)
        .LineStyle = wdLineStyleDouble
        .ColorIndex = wdBlue
    End With
End With

```

不可滥用 With 语句

With 语句简化对象引用很有效，且使代码容易阅读，但不可以能用就用。如果 With 语句中只有一条语句，例如下面的例子（同样以 Word 为例），那么会花很多时间输入过多的代码。

```

With ActiveDocument.Sections(1).Headers(wdHeaderFooterPrimary)
    .Range.Words(1)
    .Bold = True
End With

```

同理，不要随意嵌套 With 语句：

```

With ActiveDocument
    With .Sections(1)
        With .Headers(wdHeaderFooterPrimary)
            With .Range
                With .Words(1)
                    With .Font
                        .Italic = True
                        .Bold = False
                        .Color = wdColorBlack
                    End With
                End With
            End With
        End With
    End With
End With

```

上面这段语句最好这样写：

```
With ActiveDocument.Sections(1).Headers(wdHeaderFooterPrimary).Range
    Words(1).Font.Italic = True
    .Bold = False
    .Color = wdColorBlack
End With
```

优化 Select Case 语句

在使用 Select Case 语句时，应当把可能性最大的 Case 语句放在前面。这样做可以节省 VBA 的工作量和时间，因为 VBA 在 Case 语句中查找条件，一旦符合就停止，因此查找越快代码执行就越快。

不要滥用查验

如果每次在执行某个程序时都需要实施一项设定（特别是逻辑型），那么并不需要查验当前值。例如，需要确保 Excel 中 Active Workbook 对象的 EnableAutoRecover 属性（一个逻辑属性，用来设定或返回工作簿的 AutoRecover 属性）为真，可以先查验，如果为 False，就设定其为 True：

```
If ActiveWorkbook.EnableAutoRecover = False Then
    ActiveWorkbook.EnableAutoRecover = True
```

但是那样写会浪费代码，应当直接令其为 True：

```
ActiveWorkbook.EnableAutoRecover = True
```

去除代码中不使用的内容

要提高代码的效率，应去除所有不使用的内容。在使用多个内部相关的程序编写复杂的工程时，很容易产生一些程序几乎或完全无用。

如果在编写程序时做过详细注释，去除多余的程序就不困难，程序是否有用也就清清楚楚。如果不知道哪个程序调用了哪个程序，可使用 Call Stack 对话框（见图 16.1，选择“视图”>“调用堆栈”，或者按 Ctrl+L 键以打开该对话框）观察其结果。

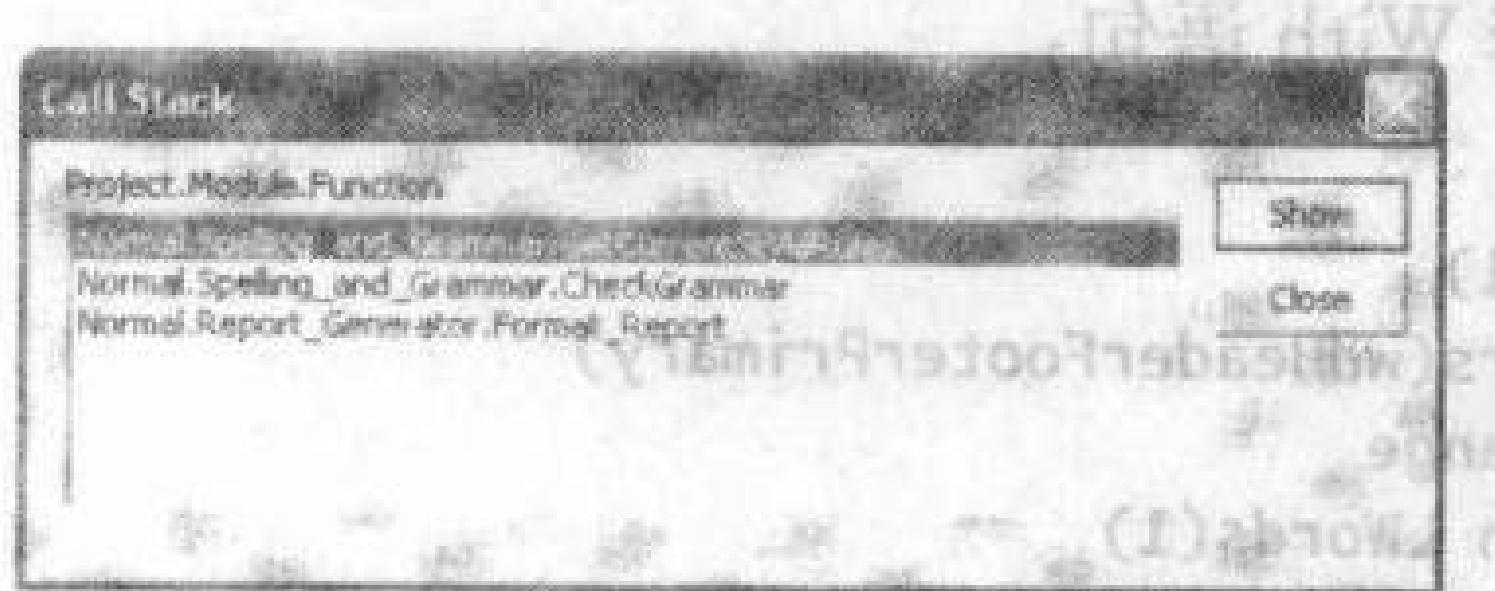


图 16.1 Call Stack 对话框可指出哪个程序调用哪个

还可以采用下面的技巧之一：

- ◆ 在有疑问的程序前设定断点，如果调用就会得到提示。
- ◆ 在代码中关键的结合点显示信息框。
- ◆ 使用调试。在适当的地方（也许是程序开头）打印语句，在“立即”窗口中进行临时记录。

在删除明显不用的程序之前，应确保该程序不仅在当前正在运行的程序中不需要，而且还应考虑情况改变后是否需要。如果可能有用，应转移到另一个工程中保存起来，而不要彻底删除。

一旦删除多余的程序，就应检查程序中的变量。即使使用了 Option Explicit 声明，而且每个变量都采取显式声明，也应检查没有声明最终也未使用的变量。对简单的工程，可使用“本地”窗口观察哪些没有得到赋值的变量，以此找出未使用的变量。对于复杂的工程，也许需要借助第三方的工具软件删除代码中不使用的内容。

提示：在删除整个模块之前，使用“文件”>“导出文件”功能，将发现有价值的模块复制成.BAS 文件保存在安全的地点。同样，可将自定义窗体导出成.FRM 文件，类导出成.CLS 文件。

如果有怀疑，可将认为多余的变量注释掉，在确信使用了 Option Explicit 后运行程序数次，并考虑不同的路径。如果未出现“变量未定义”的编译错误，应该可将该变量删除。

改进代码布局

改进的第二个方面是布局改进。这样的改进并非出于美学考虑，而是让代码更容易阅读、维护和修改。

按级别缩进

正如本书前面介绍的例子，把代码按其逻辑关系用退格或空格布局可使代码容易看懂。可使用“编辑”工具栏上的“缩进”和“凸出”按钮或按 Tab 键和 Shift+Tab 键使选择的代码模块进退移动，模块内部的相对缩进关系保持不变。

注意：不要对标记进行缩进处理。如果对标记缩进，一旦插入点离开该行，Visual Basic 编辑器会删除标记左面的空格。标记不用空格便于查找。

用行连接字符将长行分开

用行连接字符（空格加下划线）可以将长行代码分成多行。分行后长行语句分成短行便于在“代码”窗口中观察，分行也可以按逻辑分段。

用字符串连接字符将长字符串分开

不能使用行连接字符分开字符串，必须先把字符串分段，然后使用字符串连接字符(&)连接各字符串。各段字符串可使用行连接字符分行。例如：参阅下面的长字符串：

```
strMessageText = "The macro has finished running. Please check your presentation  
➥ to ensure that all blank slides have been removed."
```

上面的字符串可以按下面的方式分段并连接：

```
strMessageText = "The macro has finished running. " & _  
"Please check your presentation to ensure that " & _  
"all blank slides have been removed."
```

注意：也可使用加号(+)连接两个字符串，但它不可以用于字符串和数字变量之

且而，要需不间，否则 VBA 会使两者相加而不是连接。如果坚持用字符串连接字符，代码阅读起来很容易。

使用空行给代码分段

要使代码更容易阅读，可用空行把语句按逻辑分段。例如，在变量声明中进行分段，下面的例子可使代码更清晰：

```
Sub Create_Rejection_Letter
    Dim strApplicantFirst As String, strApplicantInitial As String,
        strApplicantLast As String, strApplicantTitle As String
    Dim strJobTitle As String
    Dim dteDateApplied As Date, dteDateInterviewed As Date
    Dim blnExperience As Boolean
```

next statements in the procedure

使用变量简化语句

可使用变量把复杂的语句变简单、变短。例如，对话框的语句可能很长：

```
If MsgBox("The document contains no text." & vbCrLf & vbCrLf
    & "Click the Yes button to continue formatting the document." & _
    " Click the No button to cancel the procedure.", _
    vbYesNo & vbQuestion, _
    "Error Selecting Document: Cancel Procedure?") Then
```

```
Dim strMsg As String
Dim strTBar As String
strMsg = "The document contains no text." & vbCrLf & vbCrLf
strMsg = strMsg & "Click the Yes button to continue formatting the document."
strMsg = strMsg & "Click the No button to cancel the procedure."
strTBar = "Error Selecting Document: Cancel Procedure?"
If MsgBox(strMsg, vbYesNo & vbQuestion, strTBar) Then
```

乍看起来，代码似乎比直接写要复杂，主要是因为变量声明增加了代码的数量。但长期使用就会觉得这样写更容易阅读和修改。

注意：在前面一段程序中，也可以将 MsgBox 语句的 vbYesNo & vbQuestion 用变量替换（最好用长整型而不用不定型）。但这样做，代码更难懂也不值得做。

用参数在程序之间传递信息

在调用另一个程序时，通常需要在调用时传递信息，在程序运行时，返回其他信息或修改过的信息。

在程序之间传递信息最好的方法是使用参数。可以在程序的声明行对参数进行声明。参数出现在程序名称后面的括号中。可以有一个参数或者多个参数，它们之间用逗号隔开：

```
Sub PassOneArgument(MyArg)
Sub PassTwoArguments(FirstArg, SecondArg)
```

与函数（见第 9 章）相同，传递参数可以根据地址或值。如果根据地址传递参数，接收程序进入原变量的内存位置可以更改原变量。然而，如果根据值传递参数，接收程序得到变量的值，不可以改变原变量。

根据地址传递参数很有用，变量在接收程序中处理然后返回到给出参数的程序中去。根据值传递用于在接收程序中使用保存在变量中的信息，并同时保证变量中的信息不会改变。

根据地址传递参数是默认的方法，但也可以使用 ByRef 关键词明示要根据地址传递。下面两例都采用根据地址传递的方法：

```
Sub PassByReference(MyArg)
Sub PassByReference(ByRef MyArg)
```

如果根据值传递，必须使用 ByVal 关键词。下面的语句根据值传递：

```
Sub PassByValue(ByVal ValArg)
```

若有必要，可以让有的参数根据地址传递，有的根据值传递。下面的例子采用了两种方法：

```
Sub PassBoth(ByRef MyArg, ByVal ValArg)
```

显式声明传递参数的类型可节省内存空间，而且明确程序传递信息按希望的方式。然而，根据地址传递，必须保证传递参数的类型符合接收程序的要求。例如，声明一个字符变量并传递参数，而接收程序需要不定型，VBA 将报错。

要声明参数的类型，应在参数列表中给出。以下语句同时给出参数类型和传递方式：

```
Sub PassBoth(ByRef MyArg As String, ByVal ValArg As Variant)
```

定义可选参数应使用 Optional 关键词。若使用 ByRef 或 ByVal，必须将关键词放在前面：

```
Sub PassBoth(ByRef MyArg As String, ByVal ValArg As Variant, _
Optional ByVal MyOptArg As Variant)
```

程序清单 16.1 给出用参数传递信息的例子。

程序清单 16.1

```
1. Sub GetCustomerInfo()
2.     Dim strCustName As String, strCustCity As String, _
       strCustPhone As String
3.     'Get strCustName, strCustCity, strCustPhone from sources
4.     CreateCustomer strCustName, strCustCity, strCustPhone
5. End Sub
6.
7. Sub CreateCustomer(ByRef strCName As String, _
                     ByRef strCCity As String, ByVal strCPhone As String)
8.     Dim strCustomer As String
9.     strCustomer = strCName & vbTab & strCCity &_
                     & vbTab & strCPhone
10.    'take action with strCustomer string here
11. End Sub
```

程序清单 16.1 包括两段最短的程序，说明怎样在程序之间传递信息：

- ◆ 第 2 行中第一段程序 GetCustomerInfo 声明 3 个字符串型变量 strCustName、strCustC-

- ◆ 第 3 行为注释行，说明在何处将信息赋值给变量。
- ◆ 第 4 行调用 CreateCustomer 程序，并将变量 strCustName、strCustCity 和 strCustPhone 传递过去。因为没有使用 Call 关键词，不需要用括号。
- ◆ 第 7 行为 CreateCustomer 程序的开始，声明 3 个参数，前两个参数 strCName 和 strCCity 按地址传递，最后一个参数 strCphone 按值传递。
- ◆ 第 8 行声明字符变量 strCustomer。第 9 行对 strCustomer 赋值。
- ◆ 第 10 行为注释行，说明对 strCustomer 开始处理（例如存入某数据库），第 11 行结束程序。

用 Private 或 Public 变量传递信息

在程序之间传递信息的另一个方法是使用私有或公共变量。如果共享信息的程序在一个模块中，就可以使用私有变量；如果在不同的模块中，就应当使用公共变量。

警告： 使用私有或公共变量传递信息通常被看做很不好的编程方法。这样做需要更多的内存，信息不容易跟踪，特别是当有多个程序介入时。不过，这种传递信息的方法有时很有用——需要处理别人的代码时可能很有用。

程序清单 16.2 给出使用私有变量传递信息的例子。

程序清单 16.2

```

1. Private strPassMe As String
2.
3. Sub PassingInfo()
4.     strPassMe = "Hello."
5.     PassingInfoBack
6.     MsgBox strPassMe
7. End Sub
8.
9. Sub PassingInfoBack()
10.    strPassMe = strPassMe & " How are you?"
11. End Sub

```

程序清单 16.2 在代码页开始声明私有变量 strPassMe，该变量对该模块的所有程序都有效。

第 3 行到第 7 行 PassingInfo 程序中，先对 strPassMe 赋值（第 4 行），第 5 行调用另一个程序。执行转到第 9 行，开始另一个程序。第 10 行加上 “How are you?”。第 11 行结束程序，此时回到第 6 行，显示信息 “Hello. How are you?”, 第 7 行结束程序。

生成及使用类

类是对象的正式定义，一般指自定义对象。设定类可以构建用户自己的对象。类的工作方式就像对象模板。一旦建立了类，就可以根据类构建对象。

类模块的作用

可以使用类存储信息，处理信息，使软件中各种对象访问信息。例如，如果从正在使用的宿主软件外部获取信息，并且使该软件的 VBA 程序访问信息，就可以将信息封装在类中以简化访问方式。和使用公共变量保存并访问信息相比，使用类更加简洁、更有效率。

简述

要生成类，必须在工程中插入一个类模块，对类进行命名，以便使用。然后在类代码页中编写类（常量和变量声明、程序和函数），从而定义类的属性和方法。一旦定义完成，类就包括所有自定义对象需要的信息，用来执行命令和保存数据。

执行类模块中的指令不同于执行普通模块中的指令，它必须声明类的对象变量，然后使用代码中该方法的对象和属性。

有时候类似乎很难掌握，因此本节给出一个简单的例子把类用于具体的实物，例如一本书。该例描述一个名称为 Book 的类，其中包括一本书的主要信息。生成类以后，输入书的信息以便了解类的工作方式。

注意：该例可用于任何 VBA 宿主软件。

计划类

在生成类之前，应考虑以下方面：

- ◆ 类描述的对象起什么作用。
- ◆ 类需要什么样的信息以包括该对象执行的任务。使用变量或属性保存该信息。使用变量在对象内部保存信息；用属性提供信息，可在对象的外部调用。可构建只读和读/写属性。
- ◆ 希望类执行怎样的命令。编写程序和函数实施可以操作的方法，程序用于执行命令的方法，函数用于返回值的方法。

每个基于 Book 类的对象包括书的信息。该类需要保存信息的属性，例如名称、作者以及价格，还需要方法同时显示书的信息。

生成类模块

生成类模块的第一步是在相应的工程中插入类模块。生成类模块的方法和生成普通模块的方法基本相同。

在工程浏览器中右击目标工程或工程包括的某个项目，从菜单中选择“插入”>“类模块”。还可以从菜单栏上选择“插入”>“类模块”，或者在“标准”工具栏上单击“插入”按钮，然后从下拉列表中选择“类模块”。Visual Basic 编辑器生成新的类模块，命名为“类 n”（n 表示序号），并打开“类模块”窗口。如果工程不包括“类模块”文件夹，VBA 会添加一个，并在工程浏览器中显现它。

如果选择了“要求变量声明”复选框（在 Visual Basic 编辑器的“选项”对话框的“编辑”页中），Visual Basic 编辑器会将代码页的声明区加上 Option Explicit 语句。

注意：如果没有选中“要求变量声明”复选框，应该加上 Option Explicit 语句，要

求类模块中的变量进行显式声明。

用卦的对类

命名类

现在应将类的名称改成有特定含义的名称，而不是“类 n”。显示“属性”窗口（如果没有显示的话），在“名称”栏中输入一个新的名称。使名称有含义，因为需要在代码中使用，并要立即了解其功能。本例的类名为 Book。在“Visual Basic 编辑器”窗口中回车或在其他部位任意单击一下，使改变的名称生效。

设定 Instancing 属性

Instancing 属性用来决定含有一个类的工程被引用后该类是否可见。默认设定为 1，在其他工程中不可见也不能使用。另一个设定为 2，使该类的工程被引用后类得以显现，并且可以使用。引用的工程不可以自己生成类中的项目。

如果希望别的工程而非自己生成的类可以访问，必须将 Instancing 属性设定为 2。否则不要动该设定，保留其为默认设定 1。

声明类的变量和常量

下一步声明变量和常量，用于类内部的操作。这些声明和本书介绍过的声明相同，除了可能在使用名称时希望指名具体的类。本例在常量和变量前使用 Book 作为前缀，表明为 Book 类的一部分。在下面的程序中声明了一个常量和五个变量：

```
Const bookName = "Book Project"
Dim bookTitle As String
Dim bookAuthor As String
Dim bookPages As Long
Dim bookPrice As Currency
Dim bookPublicationDate As Date
```

增加类的属性

现在增加类的属性。表 16.2 列出了 Book 类可以使用的属性。

可用两种方法构建类的属性。第一种不如第二种正式，但对属性的控制力较小。

表 16.2 Book 类的属性

属性	描述
ISBN	只读，字符型属性，为书的国际标准书号。生成一个新的对象，提供国际标准书号，此后该属性不可更改
Title	读/写，字符型属性，用来设定并返回书的正式名称
Author	读/写，字符型属性，用来设定并返回作者的名称
Pages	读/写，长整型属性，用来设定并返回书的页数
Price	只读，货币型属性，用来设定并返回书的价格
PublicationDate	只读，日期型属性，用来设定并返回书的出版日期
HardCover	只读，逻辑型属性，用来设定书的封面为硬壳或软壳
CD	只读，字节型属性，用来设定该书包括的 CD 数量

要，向新 *Collection* 上映射，通过其“即真量变木类”中教育效果，意主

用公共变量生成属性

第一种生成属性的方法是在类模块中声明公共变量。这样做按变量名称生成读/写属性。例如，下面的语句生成一个名为 HardCover 的读/写逻辑型属性：

```
Public Hardcover As Boolean
```

使用公共变量是生成属性的简单方法，然而却受到一定的限制：不可以选择属性为只读（或者只写），当设定或返回该属性的值时，不可以执行其他指令。

可以用正式的方法设定或返回属性。例如，已经在 Book 类的 MastVBA 项目中生成了 Hardcover 的逻辑型属性。可用下面的语句设定属性以及显示信息：

```
MastVBA.HardCover = False
MsgBox MastVBA.HardCover
```

使用属性过程生成属性

第二个比较正式的方法是使用属性过程生成属性。有三种属性过程：Property Let、Property Get 和 Property Set：

- ◆ Property Let 过程对属性赋值。
- ◆ Property Get 过程从属性中返回值。
- ◆ Property Set 过程设定对象的引用。

一般来说，这些过程成对使用，Property Get 和 Property Let 成对或者 Property Set 和 Property Let 成对。Property Let 可单独使用以生成一个只读属性。

用 Property Let 过程对属性赋值

对属性赋值，需要使用 Property Let 过程。Property Let 过程的语法如下

```
Property Let name ([arglist] value)
    [statements]
End Property
```

该语法的结构如下：

- ◆ Property 关键词引出过程，End Property 关键词结束过程。
- ◆ name 为必选参数，指明生成属性过程的名称。如果用 Property Get 过程生成属性，必须在 Property Let 过程中使用同样的名称。
- ◆ arglist 为必选参数，指明进入过程的参数。如果 arglist 含有多个参数，必须用逗号隔开。

例如，下面的 Property Let 过程生成字符型属性 Title，将指明参数 NewTitle，并将该值送给变量 bookTitle。

```
Property Let Title(NewTitle As String)
    bookTitle = NewTitle
End Property
```

属性过程的末尾有只写属性 Title。只写属性不常用，因此下一步是为指定属性的写入方法。

用 Property Get 过程返回属性的值

要返回属性的值，应使用 Property Get 过程。Property Get 过程的语法如下：

```
Property Get name [ (arglist) ] [ As type ]
    [statements]
End Property
```

该语法与 Property Let 过程相同，不同的地方有两处：

- ◆ 其一，Property Get 加上可选的 type 参数，指明属性的数据类型。
- ◆ 其二，在 Property Get 中 arglist 是可选的。Property Get 过程中有的参数往往不需要。如果使用参数，名称和数据类型必须与 Property Let 过程中的参数相吻合。

例如，下面的 Property Get 过程生成了字符属性 Title，并将 bookTitle 变量的内容赋值给它：

```
Property Get Title() As String
    Title = bookTitle
End Property
```

Property Get 过程生成只读属性。然而当和 Property Let 成对使用时就产生了读/写属性——使用这两个过程，属性 Title 就可以使用了。

用 Property Set 过程指定对象的属性

除了可以给属性赋值外，还可以赋给属性对象。要这样做，应使用 Property Set 过程而不是 Property Let 过程。Property Set 过程的语法如下：

```
Property Set name ( [arglist,] reference )
    [statements]
End Property
```

其语法和 Property Let 过程的语法相同，但 Property Set 使用了 reference 参数，而不是 value 参数。reference 是必选参数，用来指定对象的引用。

例如，下面的 Property Set 过程用来生成对象属性 Where，指定一个范围：

```
Property Set Where(rngR As Range)
    bookRange = rngR
End Property
```

提示：对于对象变量，可同时使用 Property Set 过程和 Property Let 过程，但在大多数情况下仅使用 Property Set 过程。

Book 类的属性

程序清单 16.3 给出 Book 类的所有属性。

程序清单 16.3

1. Public Property Let Title(strT As String)
2. bookTitle = strT
3. End Property
- 4.

```

5. Public Property Get Title() As String
6.     Title = bookTitle
7. End Property
8.
9. Public Property Let Author(strA As String)
10.    bookAuthor = strA
11. End Property
12.

13. Public Property Get Author() As String
14.     Author = bookAuthor
15. End Property
16.
17. Public Property Let Pages(intPages As Integer)
18.     bookPages = intPages
19. End Property
20.
21. Public Property Get Pages() As Integer
22.     Pages = bookPages
23. End Property
24.

25. Public Property Let Price(curP As Currency)
26.     bookPrice = curP
27. End Property
28.
29. Public Property Get Price() As Currency
30.     Price = bookPrice
31. End Property
32.

33. Public Property Let PublicationDate(dtePD As Date)
34.     bookPublicationDate = dtePD
35. End Property
36.
37. Public Property Get PublicationDate() As Date
38.     PublicationDate = bookPublicationDate
39. End Property
40.
41. Public Property Get Available() As Boolean
42.     Available = Date >= bookPublicationDate
43. End Property

```

在程序清单 16.3 中，Book 类的每一个属性都被声明为 Public，因此可以全局调用。

每一个 Property Let 过程后都给出 Property Get 过程。第 1 行到第 3 行的 Property Let Title 过程和第 5 行到第 7 行的 Property Get Title 过程对应，其他属性也是如此。过程配对使程序容易阅读，以保证每一个过程都有相应的对应过程以及参数互相吻合。然而，如果需要这两个成对的过程 Property Let 和 Property Get 也可分开。

第 41 行到第 43 行的 Property Get Available 没有相应的 Property Let 过程。Available 属性是只读的，其值由对象的内部产生。

添加类的方法

必要时，添加过程和函数可以添加类的方法。这些代码必须在类模块的内部，以此在类

的属性和方法列表中呈现为方法，这些过程和函数与普通模块中使用的过程和函数相同。

Book 类只有一个方法 ShowInfo，用于显示带有书属性的信息框。程序清单 16.4 为方法 Show Info 的过程。

程序清单 16.4

```

1. Sub ShowInfo()
2.     Dim strM As String
3.     strM = "Title:" & vbTab & bookTitle & vbCrLf
4.     strM = strM & "Author:" & vbTab & bookAuthor & vbCrLf
5.     strM = strM & "Pages:" & vbTab & bookPages & vbCrLf
6.     strM = strM & "Price:" & vbTab & "$" & bookPrice & vbCrLf
7.     strM = strM & "Date:" & vbTab & Me.PublicationDate & vbCrLf
8.     If Me.Available Then strM = strM & vbCrLf & "AVAILABLE NOW"
9.     MsgBox strM, vbOKOnly + vbInformation, bookName _
        & " Information"
10. End Sub

```

ShowInfo 过程构建一个字符串，其中包括从类中得到的信息，然后在信息框中显示出来。过程解释如下：

- ◆ 第 2 行声明字符型变量 strM，它用来保存信息框中 prompt 参数的数据。
- ◆ 第 3 行使 strM 加入 Title：，加入退格、bookTitle 变量的内容（书的名称）以及回车。
- ◆ 第 4 行继续构建 strM，加上作者信息。同样地，第 5 行加上页数，第 6 行加上价格（包括美元符号）。
- ◆ 第 7 行继续构建 strM，加上日期信息。然而没有使用内部变量（book Publication Date）返回日期，而调用了 PublicationDate 属性（用 Me 关键词指定）。这样做是为了举例，返回内部变量同样也是可以的。然而应当了解返回对象信息时的差别：VBA 没有提供变量，而是用 PublicationDate 过程返回信息。
- ◆ 第 8 行返回对象（由 Me 关键词指定）的 Available 属性。如果 Available 为 True，该语句向 strM 增加一个空行（另一个 vbCrLf）并加上字符串 AVAILABLE NOW。
- ◆ 第 9 行显示包括 strM 的确定信息框，该信息框的名称为 book Name（包括文本 Book Project 的常量）和 Information，信息框使用了信息。

使用类

要使用类，必须用 New 关键词用于 Dim 语句或 Set 语句生成一个新的对象。例如，下面的语句生成一个新的 Book 类对象变量：

```
Dim myBook As New Book
```

下面的语句声明一个对象变量，然后指定一个新的 Book 对象：

```
Dim bookAnotherBook As Object
Set bookAnotherBook = New Book
```

可以访问 Book 对象的属性和方法，如同访问其他 VBA 对象的属性和方法那样。例如，

下面的语句设定了 bookAnotherBook 的 Price 属性：

```
bookAnotherBook.Price = 54.99
```

程序清单 16.5 为 Class _ Test 程序，用于显示 Book 类的内容。

程序清单 16.5

```
1. Sub Class_Test()
2.
3.     Dim myBook As New Book
4.
5.     myBook.Title = "Mastering VBA Second Edition"
6.     myBook.Price = 39.99
7.     myBook.Author = "Anonymous"
8.     myBook.Pages = 704
9.     myBook.PublicationDate = #9/23/2005#
10.
11.    myBook.ShowInfo
12.
13. End Sub
```

程序清单 16.5 显示使用新类的示例，说明如下：

- ◆ 第 1 行开始程序，第 13 行结束程序。
- ◆ 第 2 行为空行，第 3 行声明新的对象变量 my Book，第 4 行又是空行。
- ◆ 第 5 行到第 9 行设定对象的 5 个属性，正如对另一个对象设定属性那样。
- ◆ 第 10 行为空行。第 11 行调用了对象的方法，类似于调用另一个对象的方法。

还有一处需要测试。Available 属性是只读逻辑属性。在声明 myBook 之后的 Class: Test 程序的“代码”窗口中输入下面的语句：

```
myBook.Available = True
```

在输入 myBook 后按下“.”键就可以在属性和方法列表中见到 Available 字样。一旦输入 Available 和等号 (My Book. Available=)，VBA 就不能像读写逻辑属性那样在自动列表成员中给出 False 和 True，这是因为 Available 在这里不存在。第二，如果要运行这段代码，就会得到变异错误“不可对只读属性赋值”。

第17章 调试代码和处理错误

- ◆ 了解调试的基本原理
- ◆ 4种不同的错误
- ◆ 不可捕捉的错误
- ◆ VBA 的调试工具
- ◆ 处理运行错误
- ◆ 处理用户中断

本章介绍在 VBA 代码中可能出现的错误以及怎样处理。本章将给出错误类型，从简单的输入错误到死循环以及偶尔出现的错误。

本章一开始简单介绍调试的原理，然后说明 VBA 提供的程序调试工具，并使用它们把错误从一些语句中去除。本章的结尾讨论处理错误的各种方法以及使用的场合。

调试原理

bug 是会造成程序运行不正常的软件或硬件错误。调试意味着将硬件或软件中的错误去除。

注意：在上下文中错误有不同的解释，可以是不知名的小昆虫在电路板中造成计算机工作不正常，或者由神话中的妖怪派生出的含义。然而实际上该词来自早期的电报而不是出自计算机时代。要了解更多情况，可参阅网上计算机字典中的 bug 一词，网址为 <http://foldoc.doc.ic.ac.uk/foldoc/>。

调试的目的是将所有发现的错误尽可能迅速有效地排除。具体的任务可能如下：

1. 首先，测试代码看其是否工作正常。如果肯定能正常工作，那么针对相应的文件或适合的数据简单运行程序一到两次。即使看上去工作正常，仍应当在样板文件中测试一段时间后才对外发布（或者交给其他同事）。
2. 如果代码运行不正常，就需要调试。这就意味着应当按照本章中的要求找出错误，然后去除。如果去除了所有可以认定的错误，按第一步的说明调试代码。
3. 在调试代码时，应当预见到用户不按规章的用法。例如编写一段程序处理 Word 文档，前提是用户使用程序时文档被打开。应当对样本文档进行测试，并且必须保证每次都工作正常。然而，如果用户使用程序时没有先打开文档，那每次操作都将失败。
4. 在准备发布程序时，应当写出用户说明。在说明中应当指出不能纠正的错误，或者程序不能正常运行的环境。

调试程序是一项很特殊的工作，没有任何魔杖可以把程序中的错误一扫而光（尽管

Visual Basic 编辑器可以很好地帮助清除程序中的某些类型的错误。而且，有些简单的错误（例如忘记对变量初始化），可能导致重大错误。每个人都可能有自己调试程序的方法，因为编写程序有自己的风格，然而在调试时应当重点关注程序的功能，然后观察程序实际的运行情况。从这两点出发，就可以掌握调试的原则。

提示：程序越复杂，含有错误的几率越高，应当使程序尽可能简单，使其分解为独立的程序或者模块，正如第 16 章所示。

不同的错误类型

程序中的错误有 4 种基本类型：

◆ 语言错误；

◆ 编译错误；

◆ 运行错误；

◆ 逻辑错误。

本节将依次讨论这些错误类型，以及怎样纠正它们，然后介绍 VBA 的纠错工具。

语言错误

第一种错误是语言错误（也可看做是语法错误）。在“代码”窗口输入单词时，遗漏重要的标点，写错语句，或者结构没有结尾，这些都是语法错误。到目前为止，读者可能在学习的过程中已经出现过很多次语言错误，包括一些简单的打字错误。

VBA 可以排除许多语言错误，这将在下一小节中介绍。那些 Visual Basic 编辑器不能捕捉的语言错误通常表现为编译错误，因此下一小节将同时介绍语言错误和编译错误。

编译错误

当 VBA 不能正确编译语句时，就发生编译错误，即 VBA 不能将语句转换成指令。例如，要 VBA 使用一个对象并不存在的属性，编译错误就会产生。

针对语言错误和编译错误的好消息是，当光标离开出现错误的行时，Visual Basic 编辑器可以发现许多语言错误和一些编译错误。例如，在“代码”窗口中输入下列语句，并回车产生一个新行时（或者单击其他行，或者用↑和↓转入其他行）：

```
If X > Y
```

Visual Basic 编辑器将显示编译错误“缺少：Then 或 GoTo”（见图 17.1），说明语句缺少重要内容，应当写成 If X > Y Then 或者 If X > Y GoTo。Visual Basic 编辑器这样的提醒可以防止这类错误进入运行阶段。

注意：本章假定 VBA 的“自动语法检验”功能以及其他功能处于打开状态。有些开发人员会将这些功能关闭，因为不希望被打扰，如果没有这些功能，将会证明治疗比疾病更可怕。



图 17.1 Visual Basic 编辑器可帮助调试程序，在输入语句时识别许多编译错误

Visual Basic 编辑器解决前一个问题十分简单，然而也可能造成一些语言错误，在光标离开输入行时，Visual Basic 编辑器不能发现。VBA 会把这些错误归结为编译错误。例如，在使用 Word 软件时，在“代码”窗口中输入下面的语句，Visual Basic 编辑器不会发现任何错误。然而在运行程序时 VBA 编译代码，发现了错误并指出（如图 17.2 所示）：

```
ActiveDocument.SaveAs FileName:="My File.doc"
```

这是个简单的打字错误，把 FileName 打成 FileMame，但 VBA 却不能在运行前找出错误。

Visual Basic 编辑器的确可以帮助发现这样的错误。例如，要输入 Documents.Close 语句，却将 Documents 输入成 Docments。在这种情况下，Visual Basic 编辑器就不能显示属性方法列表，因为没有给出一个正确的对象（可以认为没有生成一个自定义集合或对象，名称为 Docments，而且有 Close 方法，这样的想法显然不正确）。见不到属性方法列表，就意味着什么地方出了错。如果继续输入 Documents.Close 语句，Visual Basic 编辑器也不会指出错误。只有在运行程序时才会显示“运行错误 424：需要对象”（如果没有 Option Explicit）。（如果使用了 Option Explicit 就会出现变量未定义这样的编译错误。）

类似地，如果使用一个对象并不存在的属性或者方法，VBA 会给出编译错误。例如，忘记了 Add 方法而使用了 Documents.Create，VBA 将使出错的单词呈高亮显示并指出编译错误“方法和数据成员未找到”（如图 17.3 所示），指出 Documents 集合中没有 Create 方法。



图 17.2 只有在运行时才会出现的错误



图 17.3 “方法和数据成员未找到”错误表明，使用了对象没有的方法或属性

运行错误

第三种错误是运行错误，在代码运行时出现，如果写一条语句造成 VBA 不能执行，就会产生运行错误。例如，打开一个并不存在的文档，关闭一个没有打开的文件，或者执行一个不可能实现的数学运算（例如，0 作为除数）。不能处理的运行错误导致 VBA 显示错误的代码，如图 17.4 所示。

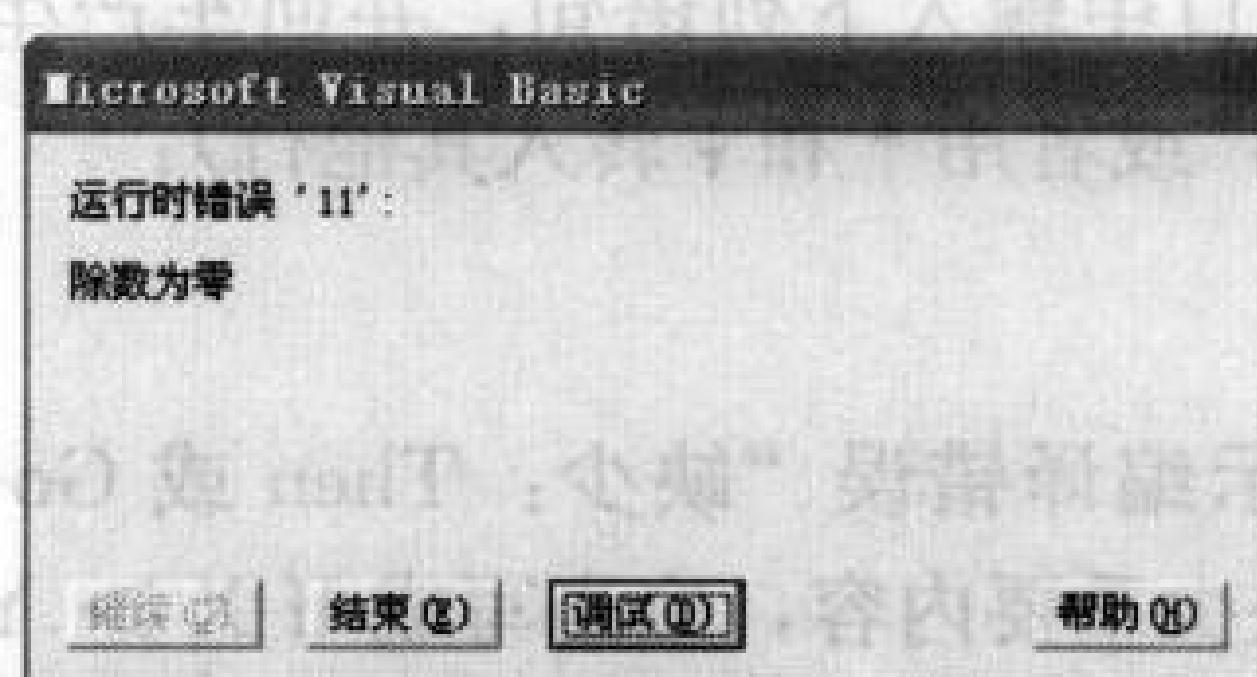


图 17.4 无法处理的运行错误导致 VBA 显示一个信息框

用 0 做除数可以看做一个不可能实现的操作。下面的语句显示信息“运行错误 11：0 作为除数”：

```
Dim DZ  
DZ = 1 / 0
```

在程序中编写的内容不可能出现这样明显的错误，这样的语句毫无疑问会产生除 0 的错误，因为除数是 0。然而很容易输入一个合法的等式，例如月收入=工资/月份，而忘记对月份赋值（如果数值变量为空，将看做 0 值）或者月份在加减运算中产生了 0 值。

要避免运行错误，可使用“监视”窗口跟踪变量的值（在本章后面的“监视”窗口中讨论）。

程序逻辑错误

第四种错误是程序逻辑错误，这样的错误产生不正确的结果。逻辑错误在语法上没有什么问题，VBA 可以编译，执行时不会产生任何错误，然而却得到不同的结果。逻辑错误可能相对比较明显（例如：执行运算时，工作簿没有选对，因为代码未检测当前的窗口），或者很隐蔽（例如：范围扩展时使用了错误的字符或者单元格）。在第一个例子中，处理程序可能运行十分完美，但得出的结果却和希望得到的结果完全不相同。在第二个例子中可以得到几乎正确的结果，或者有时候结果是正确的，有时候稍微有些错误。

逻辑错误是最难捕捉的错误。要发现逻辑错误，必须跟踪代码的执行，搞清什么时候开始出错。要这样做，就需要使用下面讨论的工具。

注意：还有两种错误可能出现——即使不应该出现。第一种是微软的 VBA 内容和实际工作的不相同。这种情况不应当出现，但因为 VBA 的复杂性，就有可能出现。如果发现代码完全不能工作，就应当考虑代码没有正确安装。用 VBA 关键词在网络上搜索，看其他人是否有类似的问题，并了解别人是怎样解决的。第二种错误和第一种有些类似，VBA 的不同版本的运行结果不一样，例如在 Word 2000 中编写的代码运行很正常，然而必须修改才能在 Word 2003 中正常运行。理想状态下，这样的情况不应当出现，然而事物总不是理想的。

VBA 调试工具

VBA 提供了一系列工具去除程序中的错误，最主要的工具是“立即”窗口、“本地”窗口和“监视”窗口。可以有数种方法调用这些程序，其中一种是使用调试工具栏（如图 17.5 所示）。三个按钮（运行子过程/用户窗体，中断以及重新设置）可以在标准工具条中使用，本章后面将介绍大多数其他功能。

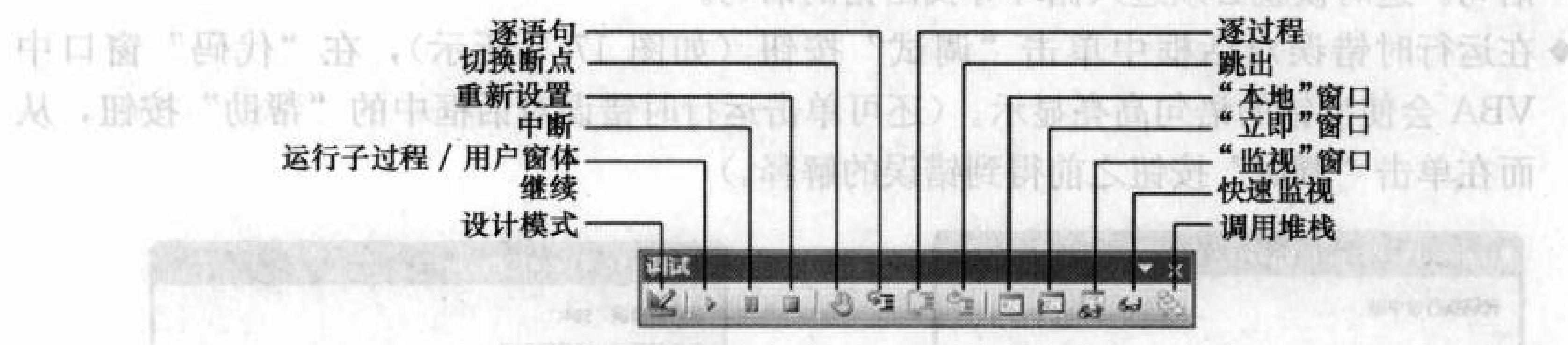


图 17.5 调试工具栏提供 13 种命令调试程序

不可捕捉的错误

程序越复杂，越容易产生真正难以捕捉的错误。通常只要有信心而且比较灵活，就可以找出程序中的错误。然而如果错误的出现由多种不可预见的环境同时造成，就很难找出来。

例如，如果用户在对话框中进行选择时，程序发生的错误就很容易捕捉。然而，用户在对话框中做出两个选择时发生的错误就很难认定。如果错误在用户输入三项选择时出现，或者是由于程序运行的文件中某个成分造成的，查找起来就困难许多。

据说黑客对错误命名采用哲学和量子物理的原理。例如：“heisenbug”定义为“人们查找或发现时消失或警告的错误”。heisenbug 十分可怕，正如波尔和扁桃酸（如果好奇，可在网上查找详细内容）。然而，最麻烦的是 schroeddingbug，这是个设计或实施错误，一直处于禁止状态，直到阅读代码或者发现代码不能工作，因此只有代码逻辑上正确了，错误才能停止。

当然这样的错误十分荒谬，必须找出代码中的问题，而且还需要向上级汇报。

中断模式

中断模式是调试程序中十分重要的工具，因为可以在程序窗口中逐步观察程序的执行情况。例如，If… Then… ElseIf… Else 语句运行时不正确，就可以采用中断模式观察哪一条语句造成了错误。

进入中断模式的方法很简单：

- ◆ 将光标插入“代码”窗口中要执行的程序，然后按 F8 键（或者在调试工具栏上单击“逐语句”，或者选择“调试”>“逐语句”，以此进入中断模式。）
- ◆ 在程序中设定一个或多个断点以产生中断模式。如果程序中出现重要的分界处，断点可使程序在特定之处停止执行。最简单的方法是单击“代码”窗口左面的边框。（也可在某一行上右击并选择“切换”>“断点”。）可以设定多个断点，如果在程序中查找错误，这样做十分有用，因为可以正常执行没有错误的代码，然后在可能有错误的程序中停止。可以就此进入有问题的语句，观察其执行情况。

还有不同的方法可以进入中断模式：

- ◆ 按 Ctrl+Break 键，然后单击对话框中的“调试”按钮（如图 17.6 所示）。一般情况下，使用这样的方式进入中断模式的唯一理由是代码进入死循环。（此时代码长时间不工作，或者不正确地重复执行。）按 Ctrl+Break 键时，VBA 将正在执行的语句高亮显示，然而（取决于运行时间）不可能是该语句造成了问题，而是在循环中的某条语句。这时候就必须进入循环寻找出错的语句。
- ◆ 在运行时错误对话框中单击“调试”按钮（如图 17.7 所示），在“代码”窗口中 VBA 会使出错的语句高亮显示。（还可单击运行时错误对话框中的“帮助”按钮，从而在单击“调试”按钮之前得到错误的解释。）

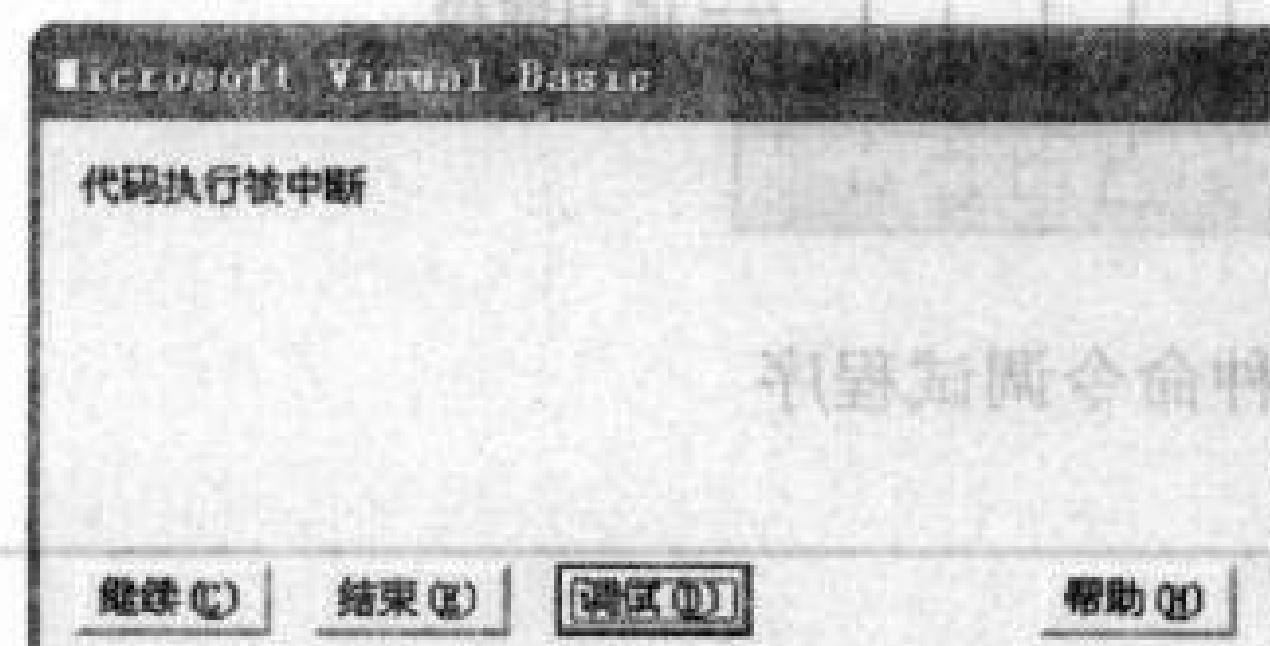


图 17.6 按 Ctrl+Break 键进入中断模式，然后在对话框中单击“调试”按钮

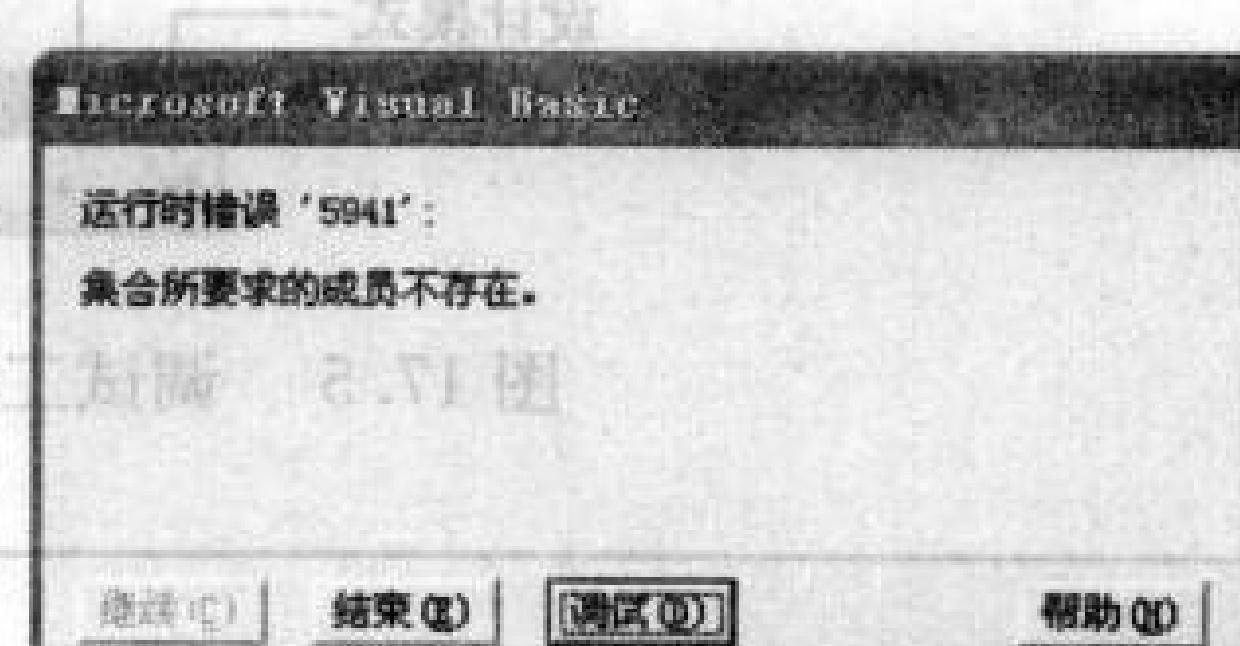


图 17.7 在这样的运行时错误对话框中进入中断模式就可直接找到出错的语句

“逐过程”命令和“跳出”命令

本书在第 3 章中介绍过怎样使用 F8 键进入“逐语句”命令。（也可在调试工具栏上选择“调试”>“逐语句”。）逐语句可以准确地掌握每一条语句的执行过程，然而有时候希望越过一些工作正常的代码而进入有问题的部分。

中断模式提供三个功能以跳过代码：逐过程、跳出和运行到光标处。

注意：逐过程和跳出只有在中断模式的情况下才能使用（例如，使用逐语句）。

逐过程（可按 Shift+F8 键，在调试工具栏上单击“逐过程”按钮，或者选择“调试”>“逐过程”）执行当前程序中调用的整个程序或函数，不再向“逐语句”命令那样一步步执行调用的程序。（调用的程序或函数被越过了。）使用“逐过程”命令可调试调用其他程序或函数的程序，这时被调用的程序运行正常，不需要一步一步地测试。

“跳出”命令（按 Ctrl+Shift+F8 键，单击调试工具栏上的“跳出”按钮，或者选择“调试”>“跳出”）运行当前程序中的其余指令。如果逐步测试的部分已经结束，就可以使用“跳出”命令迅速执行其余的指令。

“运行到光标处”命令（可按 Ctrl+F8 键或者选择“调试”>“运行到光标处”）快速运行指令，直到光标当前所在的位置，就是说进入一个断点。在使用该命令之前，应当使光标设定在合适的语句上。

“本地”窗口

“本地（Locals）”窗口针对当前的程序按树形结构给出所有表达式的值和类型（如图 17.8 所示）。“表达式（Expression）”一栏显示每个表达式的名称，在程序名称的下面给出。“值（Value）”栏显示表达式当前的值（包括空，如果表达式为空，或者为 null，以及为 nothing）。“类型（Type）”栏显示表达式的数据类型，不定型列出 Variant 以及赋值的数据类型（例如 Variant/String 为不定型赋值成字符型数据）。

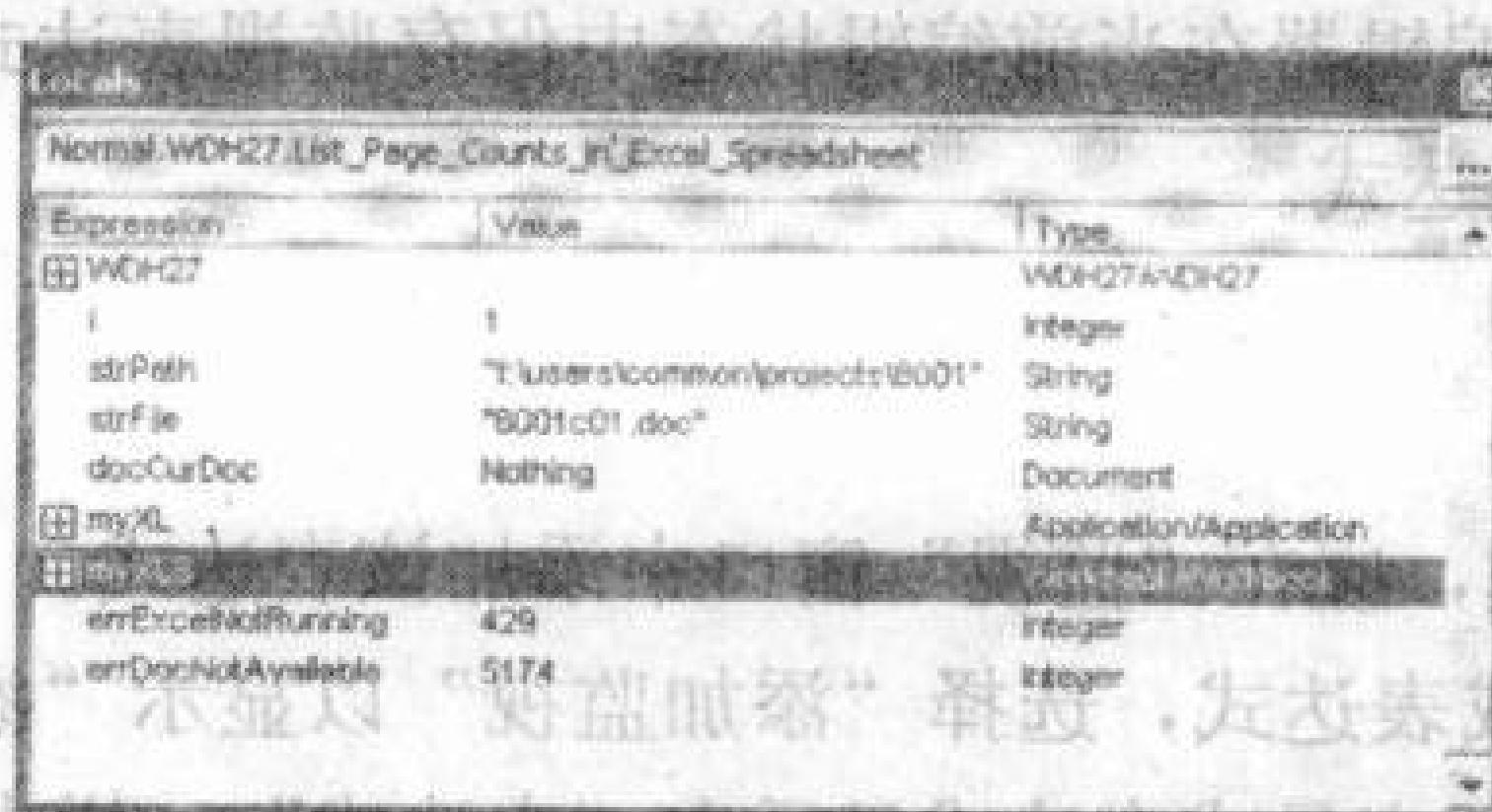


图 17.8 使用 Locals 窗口可以见到当前程序中的所有表达式

要显示“本地”窗口，可单击调试工具栏上的“本地窗口”按钮或者选择“视图”>“本地窗口”。要隐藏“本地”窗口应单击关闭按钮。

在“本地”窗口中，可单击标有省略号的按钮以显示“调用堆栈”信息框。“调用堆栈”信息框在本章的后面将讨论。标有省略号的按钮只有在中断模式中可用。

“监视”窗口

“监视（Watches）”窗口（如图 17.9 所示）是个独立的窗口，用来跟踪代码执行时变

量和表达式的值。要显示“监视”窗口，单击工具栏上的“监视窗口”按钮，或者选择“视图”>“监视窗口”。要隐藏“监视”窗口，可单击关闭按钮（单击“监视窗口”按钮，或者再次选择“视图”>“监视窗口”可再次显示它）。

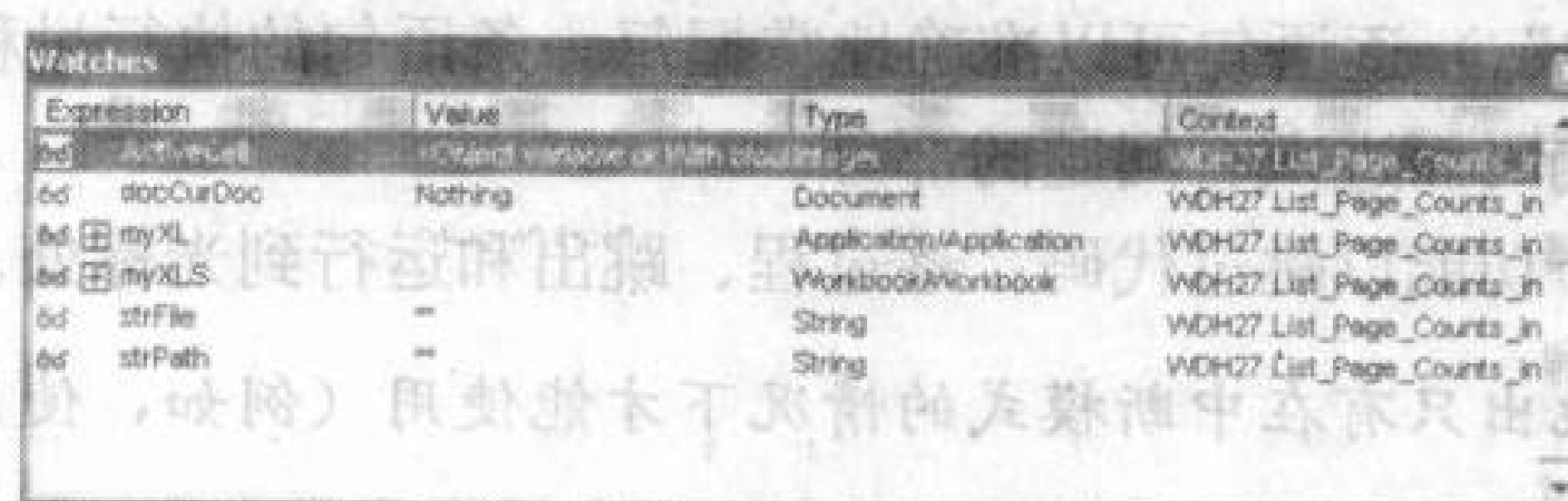


图 17.9 使用“监视(Watches)”窗口跟踪代码中变量和表达式的值

“监视”窗口显示表达式运行的变量或表达式。该信息可在程序执行时发现变量或表达式不正确的结果。“监视”窗口列出变量或表达式的名称、值、数据类型（整型、字节型、字符型、长整型，等等）以及程序名称，因此要跟踪一个变量的值只需要在中断模式中观察“监视”窗口即可。

注意：如果“监视”窗口中的变量或表达式没有初始化，“监视”窗口会在“值(Value)”栏显示溢出上下文，在“类型(Type)”栏显示空或者不定型/空。

Visual Basic 编辑器在“监视”窗口中更新所有的监视表达式，无论何时进入中断模式以及无论何时在“立即”窗口中执行一条语句。因此如果在“代码”窗口中按 F8 键（即中断模式），就可以监视每一条语句执行时的变量或表达式的值。这是查找错误或错误结果最好的方法，比使用鼠标一个个检查表达式方便得多。

在“监视”窗口中显示变量之前，必须进行声明（否则，Visual Basic 编辑器将出现“未生成变量”这样的错误），这是另一个很好的理由，说明应当在程序开始对变量进行显式声明，而不是在程序中间隐式声明。

因为监视表达式减低了运行速度，所以 Visual Basic 编辑器不能在代码中保存内容，必须单独存放。Visual Basic 编辑器在当前编辑状态中保存监视表达式，因此可以在程序之间移动，而不至于丢失监视表达式。

设定监视表达式

要设定一个监视表达式，应在“监视”窗口中添加该表达式。

1. 右击代码中的变量或表达式，选择“添加监视”以显示“添加监视”对话框（如图 17.10 所示），右击的变量或表达式显示在“表达式”一栏中。

注意：也可在“代码”窗口中选择变量或表达式，并选择“调试”>“添加监视”显示“添加监视”对话框。如果采取选择“调试”>“添加监视”显示的方法，而没有选择变量或表达式，就应当手工输入表达式，不过这样做会浪费时间。

2. 如果有必要，在“上下文”栏中调整设定。将“过程”下拉列表设定为当前过程，“模块”下拉列表设定为当前的模块。
3. 在“监视类型”栏中调整选项：

◆ 默认设定为“监视表达式”，用于在“监视”窗口中添加变量或表达式。

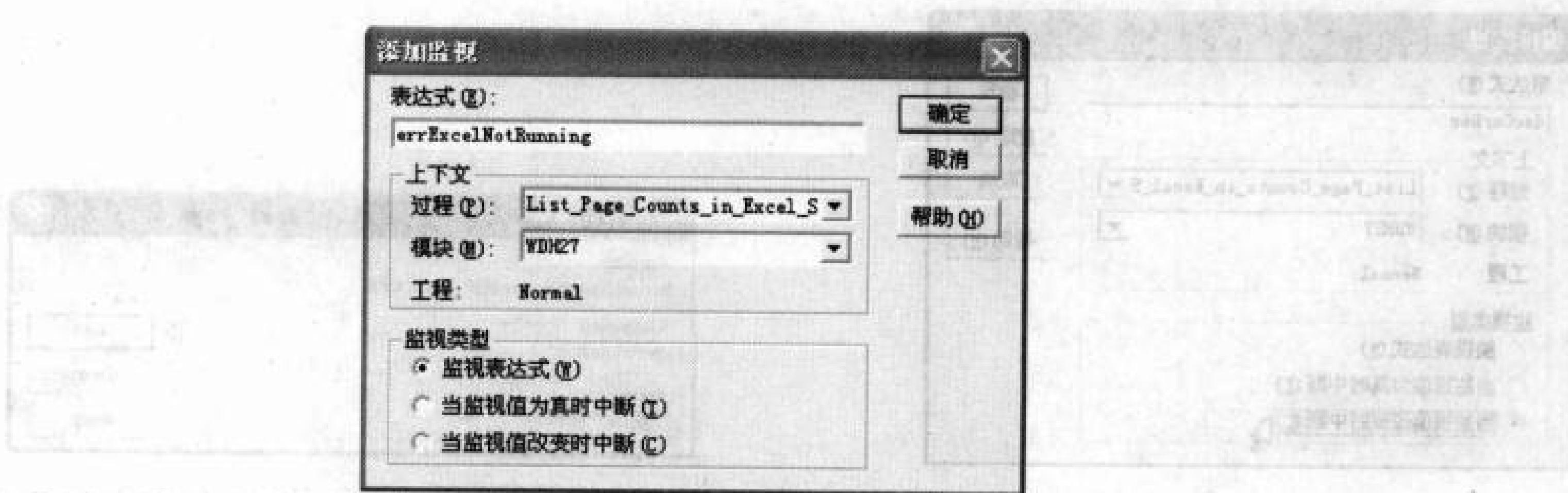


图 17.10 在“监视”窗口中指定需要添加的监视表达式

- ◆ “当监视值为真时中断”可使变量或表达式改变为真时，VBA 进入中断模式。
- ◆ “当监视值改变时中断”可使表达式改变时，VBA 进入中断模式。该设定用于监视一个表达式，其内容不应该改变，却似乎正在改变，或者想了解一个表达式的每一次变化。

提示：“当监视值为真时中断”选项按钮可在运行代码时不需要进入每一个表达式未改变成真的每一个语句。“当监视值改变时中断”选项按钮可运行代码，并在每次表达式变化时停止。

4. 单击“确定”按钮将监视表达式添加到“监视”窗口。

提示：也可将变量和表达式拖曳到“监视”窗口；将当前的上下文设定为监视表达式。选择“当监视值为真时中断”或者“当监视值改变时中断”，拖曳后编辑监视表达式。

编辑监视表达式

要编辑监视表达式，应当在“监视”窗口中右击，并选择“编辑监视”，或者在“监视”窗口中选择“调试”>“编辑监视”。这两种方法都可显示“编辑监视”对话框并给出要监视的表达式，如图 17.11 所示。可改变“上下文”栏中的选择以及“监视类型”栏中的选择改变监视内容，然后单击“确定”按钮以完成改变。

删除监视表达式

要删除监视表达式，可右击“监视”窗口，选择上下文菜单中的“删除监视”项。也可以在“编辑监视”对话框中单击“删除”按钮，删除当前的监视表达式。

使用快速监视

如果不想对某个表达式或变量生成监视表达式，可使用“快速监视”功能，该功能显示“快速监视（Quick Watch）”对话框（如图 17.12 所示），其中显示选择表达式的上下文和值。要使用快速监视，可选择“代码”窗口中的表达式和变量，然后单击工具栏的“快速监视”按钮，或选择“调试”>“快速监视”，或按 Shift + F9 键（如果已经使用“快速监视”对话框，可单击“添加（Add）按钮”添加表达式）。

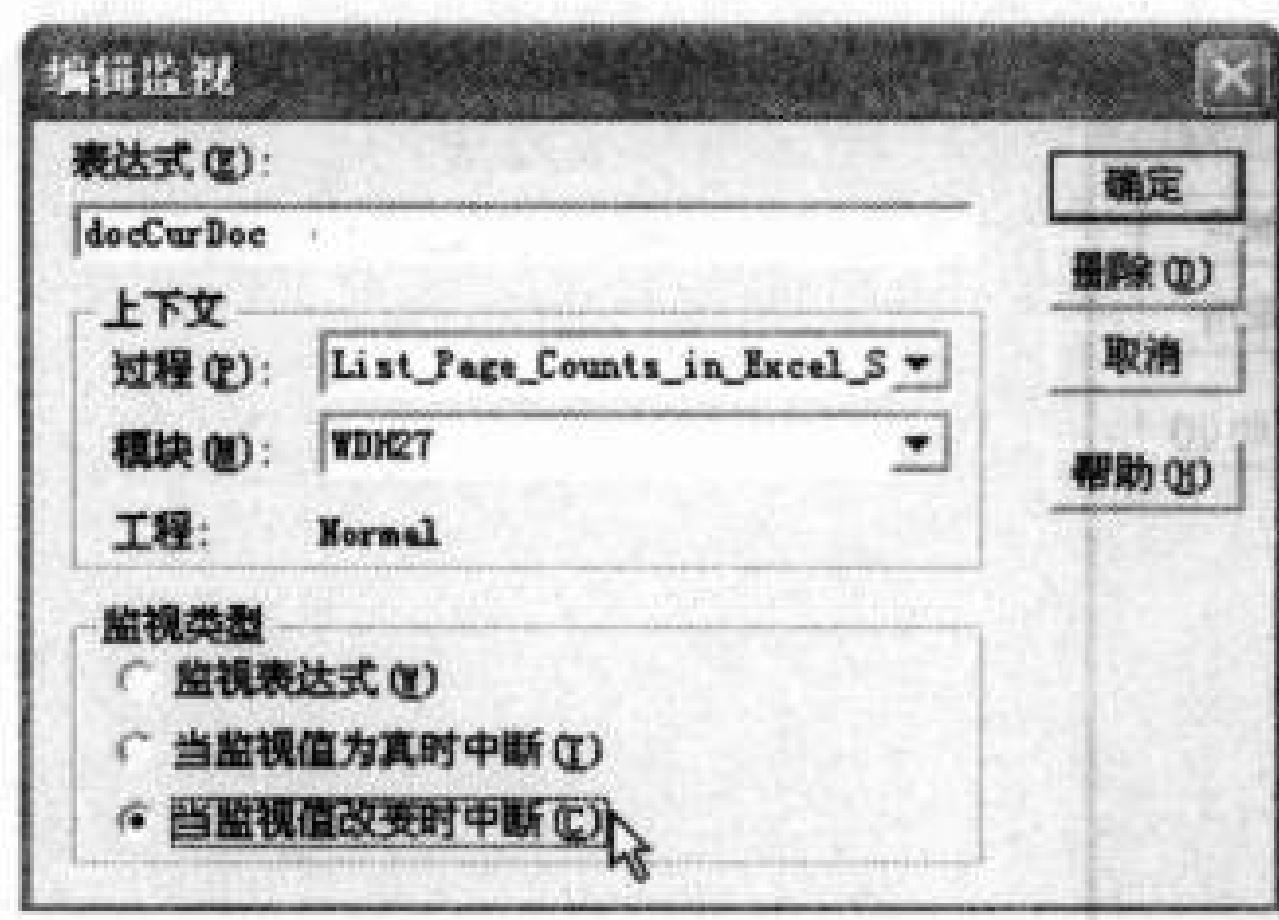


图 17.11 可在“编辑监视”对话框中编辑监视表达式

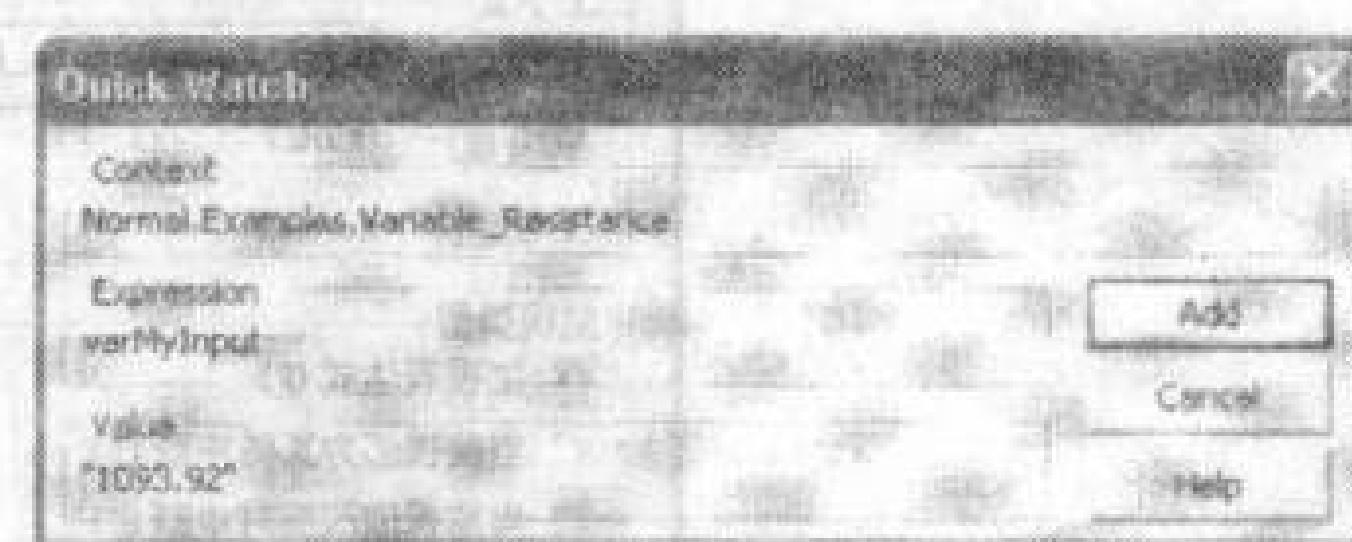


图 17.12 使用“快速监视(Quick Watch)”对话框获得变量或表达式的信息，可以不用在“监视”窗口中设定监视表达式

“立即”窗口

可使用“立即”窗口临时输入需要测试的代码，而不需要将代码输入到程序中，或显示信息以便有助于检查程序执行时的变量值。第一种情况下，将代码输入到“立即”窗口中；第二种情况下，使用在代码框中输入的语句显示“立即”窗口中的信息。

要显示“立即”窗口，可单击“调试”工具栏的“立即窗口”按钮，或者选择“视图”>“立即窗口”，或者按 Ctrl + G 键。要关闭“立即”窗口，可单击关闭按钮。（单击“立即窗口”按钮，选择“视图”>“立即窗口”，或者按下 Ctrl + G 键，并不能在“立即”窗口显示时关闭“立即”窗口）。

在“立即”窗口中，中断模式和设计模式都可在“立即”窗口中执行程序。

“立即”窗口的限制

使用“立即”窗口有一些限制：

- ◆ 不可使用声明语句（例如 Dim、Private、Public、Option Explicit、Static 或者 Type）或者控制流语句（例如，GoTo、Sub 或者 Function），这些语句会导致在“立即窗口中无效”这样的错误。
- ◆ 不可使用多行语句（例如 If 语句块，或者 For…Next 语句），因为在“立即”窗口中不同的行之间没有逻辑联系，每行单独处理。
- ◆ 不可在“立即”窗口中设定断点。

在“立即”窗口中输入代码

“立即”窗口支持许多标准窗口组合键，例如 Ctrl + X（剪切）、Ctrl + C（复制）、Ctrl + V（粘贴）、Ctrl + Home（将光标移至窗口开始处）、Ctrl + End（将光标移至窗口结尾处）、Delete（删除当前的选择）以及 Shift + F10（显示上下文菜单）。

“立即”窗口还支持以下的 Visual Basic 编辑器组合键：

- ◆ F5 继续执行程序。
- ◆ Alt + F5 执行当前程序的错误处理代码。
- ◆ F8 单语句程序逐句执行。
- ◆ Shift + F8 按过程执行。
- ◆ Alt + F8 进入错误处理。

- ◆ F2 显示对象浏览器。

最后，“立即”窗口还有两个自己的命令：

- ◆ 按 Enter 键执行当前行的命令。
- ◆ 按 Ctrl + Enter 键插入回车。

在“立即”窗口中打印信息

除了可以在“立即”窗口中输入语句进行快速测试外，还可以使用调试对象的打印方法将程序中的信息打印到“立即”窗口中。用这样的方法打印可在程序运行时查看信息，而不需要进入中断模式，或者使用中断程序的信息框或对话框。

打印方法的语法如下：

Debug.Print [outputlist]

outputlist 是可选参数，指明需要打印的表达式。也许每次都要包括 outputlist，否则 Print 方法将打印空行，没有实际用途。用下面的语句构建 outputlist：

[Spc (n) | Tab (n)] expression

这里，Spc (n) 插入空字符，Tab (n) 插入制表符，n 是空格或表格的数量。两者都是可选参数，简单的输出不需要使用它们。

expression 为可选参数，指定要打印的数字表达式或字符表达式：

- ◆ 要指定多项表达式，应用空格或冒号隔开。
- ◆ 逻辑值的打印结果为 True 或者 False。
- ◆ 如果 outputlist 为空，就没有打印结果。如果 outputlist 为 Null，打印的结果也是 Null。
- ◆ 如果 outputlist 的结果为错误，打印的结果为 Error errorcode，其中 errorcode 指错误的类型。

在下面的代码中，可在“立即”窗口中指定表达式 Cust Name、Address1、Address2、City、State 和 Zip 的内容：

```
Debug.Print CustName
Debug.Print Address1 & ", " & Address2
Debug.Print City & ", " & State & " " & Zip
```

下面的程序将在“立即”窗口中给出当前工作簿的名称和路径：

```
Sub Debug_Print_All_Workbook_Names()
    Dim oBook As Workbook
    For Each oBook In Workbooks
        Debug.Print oBook.FullName
    Next
End Sub
```

“调用堆栈”对话框

在中断模式中可使用“调用堆栈”对话框（如图 17.13 所示）显示当前的程序调用，即当前程序中调用的过程。当开始运行程序时，该程序加入“调用堆栈”对话框的列表中。如果该程序调用另一个过程，第二个过程的名称在程序执行时加入到堆栈列表中，然后从列表中去除。使用“调用堆栈”对话框可以了解哪一个过程被别的过程调用，这样有助于指明需要检查

错误的代码。

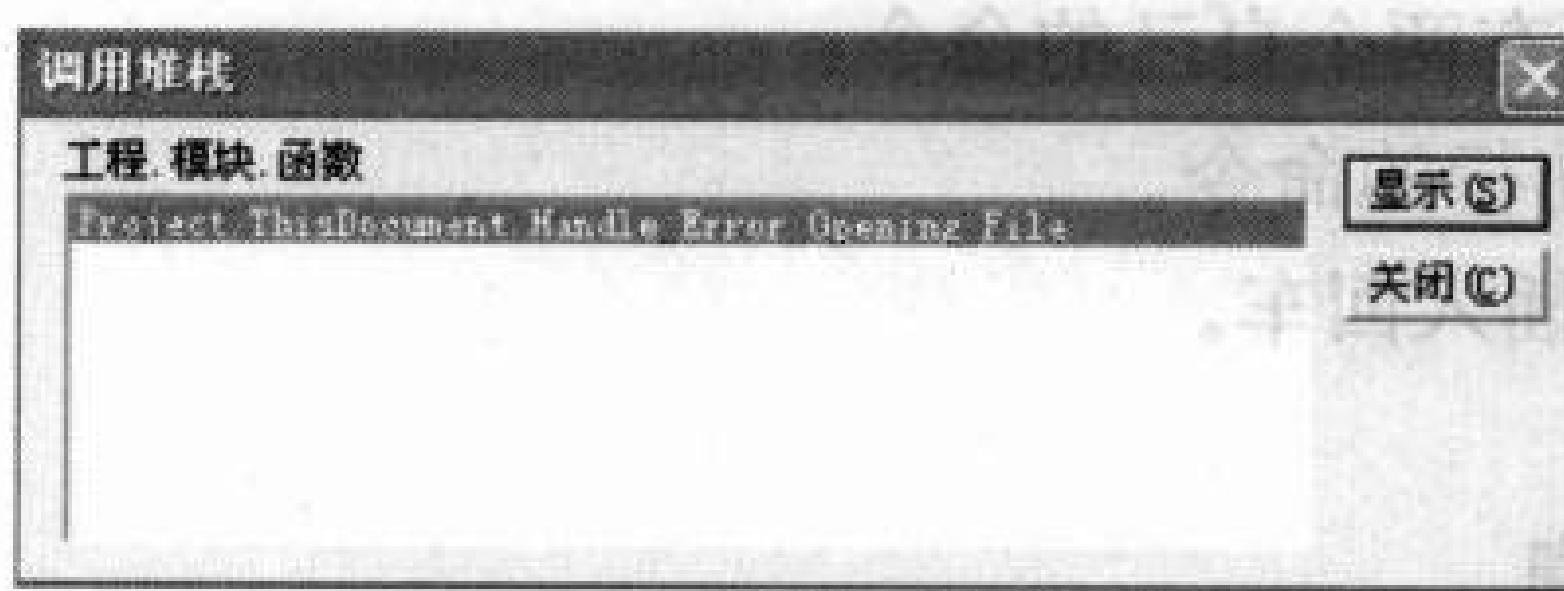


图 17.13 使用“调用堆栈”对话框可以见到当前程序调用的过程列表

要显示“调用堆栈”对话框，可单击“调试”工具栏上的“调用堆栈”按钮，或按 Ctrl + L 键，或者选择“视图”>“调用堆栈”。要显示“调用堆栈”对话框中的某个过程，选择“工程.模块.函数”列表中的项目，并单击“显示”按钮。要关闭“调用堆栈”对话框，可单击关闭按钮。

处理死循环

发现程序进入死循环并不困难：可看到程序不能停止执行。要中断死循环，可按 Ctrl + Break 键。Visual Basic 编辑器将显示代码执行已经终止的对话框。

有几种情况会导致死循环，例如使用 GoTo 语句而没有 If 条件，或者 Do 循环中没有 While 或 Until 语句。死循环很容易就可以避免，然而即使如此它也有可能出现，因为不能预见给出的条件。检测和消除死循环的最好方法是使用断点或者监视表达式锁定程序在什么地方进入了死循环。如果出现，可使用逐语句进入程序。然后使用“监视”窗口或“本地”窗口观察循环中的变量或表达式，这时将提示什么时候出了错误以及造成了死循环。

如果循环只执行数次却怀疑出现了死循环，就可使用循环中的计数器以及 Exit For 语句或者 Exit Do 语句在运行数次后退出循环。

处理运行错误

尽管 VBA 可以帮助消除语言错误和编译错误，但是运行错误仍然令人不愉快。程序迟早会有错误但不一定会停止执行。VBA 可以进行错误处理，即用代码捕捉错误，进行分析，并且在符合错误代码时采取措施。

什么时候需要编写错误处理程序

在下面的情况中应考虑编写错误处理程序：

- ◆ 当运行错误会导致程序彻底不工作时。在 PowerPoint 软件中，一段程序处理幻灯片上的几个对象，一般不需要错误处理程序。如果程序用来生成、删除或者移动文件，就应当使用错误处理程序。
- ◆ 当能够识别某种可能发生的错误，而且可以捕捉时。例如要打开一个文件，可能出现的错误有文件并不存在，文件正在被其他计算机使用，或者在网络盘、软盘、光盘、移动盘等存储器上，而且这些存储器当时都不能访问，还可能出现的错误是，使用一个打印机或远程装置（如扫描仪或数码相机）而这些设备都没有安装，未接电源，或

者设定不正确。同样地如果该对象不存在或者不能访问，在文档中处理特定对象的程序（例如 Excel 软件中的图表）都可能出现错误。

注意：有些情况下，捕捉错误比预见以及定义可能出错的条件更简单。例如在打开或处理文件之前不要通过检测以保证文件必须存在，应当捕捉如果文件不存在会导致的错误。

捕捉错误

捕捉错误意味着找出错误后进行处理。通常应避免错误中断 VBA 指令，然而也可以预见特定的错误，并指定发生错误后代码执行的路线。

要捕捉错误，可使用 On Error 语句。On Error 语句的语法为：

```
On Error GoTo line
```

在这里 line 是标记，指定程序在出错时的执行路线。例如，下面的结构指定了标记 ErrorHandler：

```
Sub ErrorDemo()
    On Error GoTo ErrorHandler
    'statements here
    Exit Sub
ErrorHandler:
    'error-handler statements here
End Sub
```

用来处理错误的标记可命名为任何有效的名称，不一定起名为 ErrorHandler 或者其他类似的名称。有人认为有含义的标记比普通的名称更清楚，例如把出错的类型放在名称中，命名为 HandleErrorNoFileOpen。有人比较喜欢普通的名称，例如 HandleErr。

通常应当将捕捉错误的代码放在程序前端，用来在后面的程序中捕捉错误。若有必要，可在需要的地方输入多个 On Error 语句，构成若干个不同的错误捕捉语句，然而，每次只能有一个起作用。就是说，出现错误时程序会进入错误处理语句中。如果语句需要捕捉不同类型的错误，那么使用多个程序处理语句是十分有用的。在下面的例子中，第一个 On Error 语句使程序进入 ErrorHandler1，第二个 On Error 语句使程序进入 ErrorHandler2：

```
Sub ErrorDemo2()
    On Error GoTo ErrorHandler1
    'statements here
    On Error GoTo ErrorHandler2
    Exit Sub
    'statements here
ErrorHandler1:
    'statements for first error handler here
ErrorHandler2:
    'statements for second error handler here
End Sub
```

每一个错误处理只对自己的程序有效，因此，不同的程序需要有不同的错误处理语句，以便在程序执行中依次生效。

因为错误处理语句在程序中表现为代码应当确保没有错误时不运行。为此，可在处理错误语句之前使用 Exit Sub 语句（即结束程序）或者 GoTo 语句使其进入标记。如果将错误

处理语句放在程序的末尾，使用 Exit Sub 语句更好，这是一种标准的处理方式，而且比较合理。如果把程序处理语句放在别的地方，使用 GoTo 语句会更简单。

注意：对于函数，应使用 Exit Function 语句而不是 Exit Sub 语句。对于属性，应当使用 Exit Property 语句。

下面的例子使用了 Exit Sub 语句，如果不出错，程序在错误处理语句之前终止：

```
Sub ErrorDemo3()
    On Error GoTo ErrorHandler
    'statements
    Exit Sub
ErrorHandler:
    'statements for error handler
End Sub
```

下面的例子使用了 GoTo 语句，在不出现错误时，越过错误处理指令（放在程序的中间）。当命令执行到 GoTo SkipErrorHandler 语句时，执行指向 SkipErrorHandler 标记，越过了处理错误的代码。

```
Sub ErrorDemo4()
    On Error GoTo ErrorHandler
    'statements
    GoTo SkipErrorHandler
ErrorHandler:
    'statements for error handler
SkipErrorHandler:
    'statements
End Sub
```

前面提到过，有些人不喜欢使用 GoTo 语句。如果 GoTo 语句使程序流不容易跟踪，可能应该同意这儿的用法（在 On Error 语句中，GoTo 语句是不可避免的）。

使错误捕捉无效

错误捕捉指令只对出现的程序有效，当程序中的指令执行完毕后，VBA 将使其无效。也可在程序结束前使其无效，应使用下面的语句：

```
On Error GoTo 0
```

在测试代码时，可能希望使错误捕捉无效，这时希望保留错误捕捉在程序的前半部分有效，而此后自己查找出错的错误。

错误后恢复

使用 Resume 语句在捕捉错误或处理错误之后恢复程序执行。Resume 语句有三种形式：Resume、Resume Next 和 Resume *line*。

使用 Resume 语句

Resume 使程序在出现错误的行中恢复执行。应当将 Resume 和错误处理语句一同使用，发现错误后修复出错的语句。例如，参阅程序清单 17.1，VBA 在不能使用 Word 中的特定

样式时进行出错处理。

程序清单 17.1

```
1. Sub StyleError()
2.
3.     On Error GoTo Handler
4.
5.     Selection.Style = "Executive Summary"
6.
7.     'the rest of the procedure happens here
8.
9.     'exit the procedure once execution gets this far
10.    Exit Sub
11.
12. Handler:
13.
14.     If Err = 5834 Then
15.         ActiveDocument.Styles.Add _
16.             Name:="Executive Summary", Type:=wdStyleTypeParagraph
17.         Resume
18.
19. End Sub
```

以下是对程序清单 17.1 错误处理的说明：

- ◆ 第 1 行开始程序，第 19 行结束程序。第 2、4、6、8、11、13 行以及第 18 行为空行。
- ◆ 第 3 行使用了 On Error 语句，起名为 Handler，出现在第 12 行。
- ◆ 第 5 行对当前的内容使用了样式 Executive Summary。如果执行顺利，将继续到第 7 行。第 7 行只是注释，表明下面的程序将开始。
- ◆ 第 9 行为注释，引出第 10 行，即 Exit Sub 语句，用来在错误处理之前结束程序。
- ◆ 如果第 5 行的 Selection. Style 语句导致了错误，程序将转到第 12 行的 Handler 标记，错误处理语句被激活。第 14 行比较错误的值。如果该样式不存在，就出现错误。如果比较结果吻合，第 15 行加入缺少的样式，第 16 行的 Resume 语句将执行返回到导致错误的地方，即第 5 行。因为现在需要的样式已经获得，Selection. Style 语句可以正常执行。

提示：要获得错误代码，可查找 VBA 的帮助文件，或者出错后记录信息框的代码和描述。

使用 Resume Next 语句

Resume Next 可使程序出错后恢复执行下面的语句。以下两种情况应当使用 Resume Next 语句：

- ◆ 用错误处理语句忽略错误，使程序继续执行而不理会出错的语句。
- ◆ 直接使用 On Error Resume Next 语句可使程序出错后继续执行下面的语句，而不需要用错误处理纠正错误。

在前面的例子中，指定的样式如果不存在，可使用 Resume Next 语句跳过：

```
Sub StyleError2()
    On Error GoTo Handler
    Selection.Style = "Executive Summary"
    'the rest of the procedure happens here
    'exit the procedure once execution gets this far
    Exit Sub
Handler:
    Resume Next
End Sub
```

Resume 和 Resume Next 用于含有错误处理的程序中。然而，如果错误发生在另外的程序中，Resume 将恢复最后的语句，调用含有错误处理的程序，Resume Next 在调用含有错误处理语句的最后一行之后恢复程序。

使用 Resume Line 语句

Resume Line 指定程序恢复的行。使用标记指定行，必须在错误处理语句的同一个程序当中。

例如，如果程序要打开一个特定的文件，可使用简单的错误处理语句，即 Resume Line，如程序清单 17.2 所示。该程序用于 Word。在其他软件中，应替换第 15 行的错误代码。

程序清单 17.2

```
1. Sub Handle_Error_Opening_File()
2.
3.     Dim strFName As String
4.
5.     StartHere:
6.
7.     On Error GoTo ErrorHandler
8.     strFName = InputBox("Enter the name of the file to open.", _
    "Open File")
9.     If strFName = "" Then End
10.    Documents.Open strFName
11.    Exit Sub
12.
13. ErrorHandler:
14.
15.     If Err = 5174 Or Err = 5273 Then MsgBox _
        "The file " & strFName & " does not exist." & vbCrLf & _
        "Please enter the name again.", _
        vbOKOnly + vbCritical, "File Error"
16.     Resume StartHere
17.
18. End Sub
```

以下是程序清单 17.2 的解释：

- ◆ 第 1 行开始程序，第 18 行结束程序。
- ◆ 第 2 行为空行，第 3 行声明字符变量 strFName，第 4 行为空行。
- ◆ 第 5 行为标记 StartHere，第 16 行的 Resume 语句将在此恢复。第 6 行为空行。
- ◆ 第 7 行使用 On Error 语句激活错误处理语句 ErrorHandler。
- ◆ 第 8 行显示信息框，提示需要打开的文件名称，并将名称保存在变量 strFName 中。
- ◆ 第 10 行打开该文件，第 9 行检查名称是否为空字符串，如果为空，将结束程序。
- ◆ 如果文件存在并可打开，程序执行到第 11 行，这里为 Exit Sub 语句，用来结束程序。否则将出现错误，转入第 13 行的 ErrorHandler，此时错误处理开始工作。
- ◆ 第 14 行为空行，第 15 行测试错误的值是否为 5174（找不到文件时的错误代码）或 5273（文档名称或路径不合法的错误代码）。如果这两者有一个相吻合，第 15 行显示信息框，通知错误，要求输入正确的名称。
- ◆ 第 16 行的 Resume 语句返回到第 5 行。第 17 行为空行。

提示：有些程序应当建立计数机制以防止用户多次重复同样的错误，因为不能找出错误的原因。错误处理每调用二次，计数变量就增加并检查结果值，当该值到达某一数值时，采用其他的措施。

警告：不能在错误处理以外使用 Resume 语句，如果那样，VBA 将报错。

错误的描述

要查看当前错误的描述，需返回 Err 对象的 Description 属性：

MsgBox Err.Description

错误信息很简明，不容易理解，对最终用户没有什么帮助。在显示错误描述之前，认真考虑一下。通常自己设定的错误描述可以起到更好的效果，用户可以解决。

建立自己的错误描述

在测试时常常需要产生错误，从而了解错误处理是否正常工作。

要产生错误，可使用 Err 对象的 Raise 方法，并且只给出 number 参数，number 指长整型参数，给出希望产生错误的代码。例如，以下的语句产生错误 5121：

Err.Raise 5121

取消警告显示

许多构建的程序使用信息框和对话框让用户在程序中选择。有些软件，例如 Word、Excel 和 PowerPoint 可使用 Application 对象的 DisplayAlerts 属性取消信息框的显示以及程序运行时的报错显示。

- ◆ 在 Word 软件中，可将 DisplayAlerts 设定为 wdAlertsNone (0) 以取消显示警告和信息框，设定为 wdAlertsMessageBox (-2) 来取消显示警告但显示信息框，或 wdAlertsAll (-1, default) 显示所有警告和信息框。DisplayAlerts 是个比较麻烦的设

定，必须在四种情况中明确指定一种，如果将设定改成 False，那以后想恢复警告，必须设定为 True 或者 wdAlertsAll，或者 wdAlertsNone 以及 wdAlertsMessageBox。VBA 在重新启动 Word 软件时恢复默认值。

- ◆ 在 Excel 中，DisplayAlerts 是读/写逻辑属性，可设定为 True 以显示警告，或设定为 False 以取消显示警告。除非修改设定，否则它将不做改变，或者重新启动 VBA 将此设定恢复为 True。
- ◆ 在 PowerPoint 中，DisplayAlerts 是读/写属性，可设定为 ppAlertsAll 以显示所有的警告，或 ppAlertsNone 以取消所有的警告。设定后不做改变将持续下去或者重新启动 VBA 将此设定恢复为 ppAlertsNone。

在 Word、Excel 和 Project 中处理用户中断

错误似乎已经很麻烦，但是还需要搞清楚用户使用 Ctrl + Break 键中断程序时会出现什么问题。有些 VBA 宿主软件（包括 Word 和 Excel）提供 3 种选择：

- ◆ 允许用户中断程序。这是一种简单的处理方法（不需要做什么工作），然而在复杂的程序中可能会出现问题。
- ◆ 不让用户使用中断，通过在程序运行时不让用户输入。这样做很简单，然而也会出现问题，一旦进入死循环，就不能停止程序。
- ◆ 作为前两种方式的折中，可以允许用户在某些程序中使用中断，在程序的关键部位不允许使用中断。

程序运行时不让用户输入

在程序运行时阻止用户输入，应当使 Ctrl + Break 组合键无效。在 Word 中，将 Application 对象的 EnableCancelKey 属性设定为 wdCancelDisabled，而在 Excel 中设定为 xlDisabled：

```
Application.EnableCancelKey = wdCancelDisabled      'Word
Application.EnableCancelKey = xlDisabled            'Excel
```

当程序停止执行时，VBA 会自动恢复用户输入。也可在程序当中恢复用户输入，即将 EnableCancelKey 属性设定为 wdCancelInterrupt（Word 中）或者 xlInterrupt（Excel 中）：

```
Application.EnableCancelKey = wdCancelInterrupt     'Word
Application.EnableCancelKey = xlInterrupt            'Excel
```

Excel 提供第三种设定，即 xlErrorHandler，用以捕捉 Ctrl + Break 组合键，错误代码为 18。可像处理其他错误一样处理这个错误。举例如下：

```
Sub CancelKey_Example()
    Dim i As Long
    On Error GoTo EH
    Application.EnableCancelKey = xlErrorHandler
    For i = 1 To 100000000
        Application.StatusBar = i
    Next i
EH:
```

```

If Err.Number = 18 Then
    If MsgBox("Do you want to stop the procedure?" _ 
        & vbCr & vbCr & "If not, stop pressing Ctrl+Break!", _
        vbYesNo + vbCritical, "User Interrupt Detected") = vbYes Then End
End If
End Sub

```

部分程序运行时阻止用户输入

有时希望在不允许中断的程序执行时，暂时阻止用户输入，然后在允许用户输入时恢复输入功能。例如，程序的功能是将若干文件从一个目录转移到另一个目录，此时用户就不能在目录转移的中途终止程序。下面的例子用于 Word 软件：

```

'interruptible actions up to this point
Application.EnableCancelKey = wdCancelDisabled
For i = 1 to LastFile
    SourceFile = Source & "\Section" & i
    DestFile = Destination & "\Section" & i
    Name SourceFile As DestFile
Next i
Application.EnableCancelKey = wdCancelInterrupt
'interruptible actions after this point

```

警告：千万不要对可能出现死循环的程序使用户输入无效。如果这样做，必须从任务管理器中关闭程序（右击提示区，选择任务管理器，并在“应用程序”页中选择相应的应用程序，然后单击“结束任务”按钮），这样做会丢失未保存的数据。

整理代码

整理代码可以大大简化调试程序。整理代码最好的办法是加注释，可以在生成代码时加注释，也可以在结束时加注释。

在生成代码时进行整理，可以研究其方法并尝试用不同的手段实现目的。加上注释说明语句要实现的功能。一旦程序可以运行，应当研究代码，删除不使用的语句，并注释程序中不使用的部分。指明仍然有效的内容，保留和其余代码功能相关的注释。

注意：对于已不再使用但功能相同的方法，应当保留注释。例如，如果想重写一段程序，使之运行更快、要节省时间和精力，就应当记录下来，还可对这部分代码的其他应用进行说明，有助于在其他程序中再次使用时方便获取。

类似地，改变现有的程序也应当给出注释，这样就不会丢失改变的线索。一旦程序运行满意，应去除不必要的注释，将不清楚的注释重新整理。

在结束编写时整理代码，只需要进入注释行。如果在开始编写程序时很清楚代码的功能，就应当这样做。程序一旦完成，稍加注释就可以十分清楚。

要整理程序，首先用单引号注释，或者用 Rem 关键词。也可以整行注释，或部分行注释。单引号或 Rem 关键词的右边为注释内容。部分行注释，单引号通常是较好的方法，如果使用 Rem 关键词，还需在前面使用冒号（有些语句接受 Rem 而不需要冒号，有些却可能

产生错误)：

```
Rem This is a comment line.
Documents.Add: Rem create a document based on Normal.dot
```

一般来说，用单引号加上空格在语句后面注释比使用 Rem 容易阅读。

```
'This is a comment line
Documents.Add      'create a document based on Normal.dot
```

很容易产生这样的想法：不需要注释程序，因为可以记住。然而一旦完成大量的程序，就不可能记住了。写完程序 6 个月后重新阅读，就会发现很陌生，好像不是自己编写的。如果在 VBA 的用法中取得进步，甚至很难想象当时使用过的笨办法。

大多数程序员讨厌注释代码。对有些人来说，讨厌几乎是病态的。其理由是在编写代码时注释降低了速度，影响了注意力；一旦程序可以运行，注释又成了负担。另外，任何有能力的人都应该读懂代码，了解其功能，难道不是吗？

也许这是对的，但应当这样考虑。首先，人们不可能总是编写代码，有时候也需要别人来处理，读代码的时候得到帮助，别人会十分感激。其次，代码不会永远属于自己，也许有时候还需要调试别人写的代码，这时候就需要注释。

部分默認

默認部分如主窗口，窗口标题栏名称默认为“Microsoft Word”。

默認的字体是宋体，字号是 10 号，颜色是黑色，加粗与否由用户自己决定。

默認的行距是单倍，段落间距是 0 磅，每段首行缩进 24 磅，每段尾行缩进 0 磅。

默認的页边距是上、下各 24 磅，左、右各 12 磅，页眉和页脚高度各 12 磅。

默認的纸张大小是 A4，方向是横向，打印机是默認的，分辨率是 96 dpi，颜色模式是 RGB。

默認的字体是宋体，字号是 10 号，颜色是黑色，加粗与否由用户自己决定。

默默认定的字符间距是 0，磅值是 100%，行间距是 1.5 倍，段落间距是 0 磅。

默默认定的段落格式是无边距，段落首行缩进 24 磅，段落尾行缩进 0 磅。

默默认定的字体是宋体，字号是 10 号，颜色是黑色，加粗与否由用户自己决定。

用户操作界面不耐，而是钟情于图形“面页”界面时称 *WYSIWYG*，即“所见即所得”。
· 通过

第 18 章 构建出色的代码

- ◆ 什么是出色的程序
- ◆ 保留和恢复用户环境
- ◆ 让用户了解程序的执行情况
- ◆ 检查程序执行的条件
- ◆ 在程序完成后重置

一旦编写好有用的程序并且可连续的执行，也许就需要发送给许多同事或者网络上的更多用户。在发送之前应该保证程序尽可能完善，包括用户界面以及可通过计算机选择的设定。很容易出现这样的情况：程序表面上很完善，但只满足了用户的基本要求，或者在某种情况下就会出现意想不到的错误。本章将介绍怎样避免这样的问题，怎样构建程序，用户在使用中不出现任何问题。

本章重点讲述编程的原则。出色的程序因软件不同，定义也不同，而原则可以用于各种软件。本章给出一些例子。

什么是出色的程序

出色的程序指用户不会遭遇多余的命令。包括下列几点：

- ◆ 适合用户的各种环境，如果程序需要改变，就能够恢复到最初的设定（例如，保证运行正常）。
- ◆ 针对程序向用户提供不同的选择，程序执行完毕后给出相关的信息。
- ◆ 在程序执行时，告诉用户执行情况。
- ◆ 在程序执行前保证其条件能够正常执行。
- ◆ 能够预见或捕捉错误，程序不至于崩溃。
- ◆ 即使在意外的环境中出现崩溃，应当尽可能减小对用户的损失。
- ◆ 程序执行完毕后应保证用户能够继续工作。
- ◆ 清除程序执行时所产生的辅助性文档、目录，等等。

如果暂停一会儿，可能会想到一些例子，其中使用的软件不是很完善。例如，Word 软件会出现这样的情况。

- ◆ 如果在处理文档时按 Page Up 键，然后再按 Page Down 键，光标就不能回到同一个点。如果在文档中翻页，然后返回上页，必须检查光标是否还在老地方，否则输入的字符可能出错。
- ◆ 同样，如果在 Word 中使用打印预览，并选择“视图”>“页眉和页脚”，Word 将切换到“页面”视图。大多数人接受“打印布局”视图用来操作“页眉和页脚”，即使早期的版本使用“页眉和页脚”窗格。然而，单击“页眉”和“页脚”工具栏的关闭

按钮回到主文档时，Word 将切换到“页面”视图之前的界面，而不是回到打印预览。

商业化软件界面上出现的这种缺陷，往往在开发人员中引起两种反应。第一，如果用户习惯这样的问题，重新选择一下，或者改变一下视图，在运行一段程序之后执行同样的动作，就没有什么问题。第二种反应十分坚定，要完全恢复用户的环境，即使软件公司似乎不能完成这项功能。

第一种情况比较经济，第二种则需要创造力。要完成开发，必须保证头脑清醒，并在这两种极端的情况下寻找正确的路线。

保留或恢复用户环境

在很多情况下，程序不需要改变用户的环境。然而，如果确实需要改变用户环境，应尽可能地恢复到先前的状态。这取决于宿主软件，下面是一些在 Word、Excel 和 PowerPoint 中的实例：

- ◆ 在 Word 中：改变版本标志设定就可以改变文本，而不作为版本标志。
- ◆ 在 Word 或者 PowerPoint 中：改变视图可以进行在原始视图中不可执行的特定操作。
- ◆ 在 Excel 中：创建临时的工作表，可以使用户放心，所有的单元格没有被使用过。
- ◆ 在所有可使用查找/替换功能的软件中，使用查找或替换功能处理部分文档，然后恢复到最后的查找（和最后的替换），这样就可以再一次正常工作。现在的问题是大多数软件的查找和替换保留了设定，用户不需要重新输入就可以执行同样的查找或者替换工作。如果更新了查找和替换的内容，那么在下一次查找或替换中就很不方便。

要恢复这样的数据，必须使用私有变量、公共变量或者自定义对象。

保证用户始终处于最佳的工作状态

如果程序结束运行，用户需要处于最佳的状态继续工作。最佳的状态根据情况而定，不过有 3 条提示：

- ◆ 通常，会让用户面对运行程序之前的同一个文档。然而有一些例外，例如程序生成了新的文件并且用户希望使用该文件，这时应当按基本法则处理。
- ◆ 如果文件未使用（至少按用户的观点），就应当回到用户运行程序之前的状态。要恢复选择，必须在程序之前定义一个范围，在程序结束之后返回这个范围。在有的软件中，也可以使用书签或者命名的范围，但是如果这样做，用完之后必须删除。
- ◆ 如果程序产生了新的对象并且用户希望进行操作，就应当在程序结束后选择该对象。

让用户了解程序的执行情况

出色的程序的一个重要特点就是在整个进程中时刻让用户了解程序的执行情况。有些情况下，有的宏任务十分简单甚至有些乏味，这时只需要在宏的描述部分给出该宏的说明，让用户确信他们在“宏”对话框中选择了正确的程序即可。在较复杂的程序中，要完成的指令较多，这时就需要向用户提供更多的信息：也许需要显示一个开始信息框或对话框，在进程

中状态栏上显示相应的信息；显示结束信息框；或者创建一个日志文件保存信息，帮助用户记录在程序执行过程中发生了什么。

首先，决定是否要在程序执行中禁止用户输入。在 Word 和 Excel 中，可以设定 Application 对象的 EnableCancelKey 属性（就像在第 17 章的“程序运行时不让用户输入”小节中提到的）来禁止用户输入，保护程序中的敏感部分。当这样做时，最好在程序的开始就通知用户，输入功能已被禁用，并解释原因。否则，用户可能误以为程序无法响应而采取某些措施——试图通过 Microsoft Office 的修复功能或任务管理器强制关闭软件。

如何让用户了解程序的执行情况，将在之后的章节中具体讨论。下节首先将解释如何在 Word 和 Excel 中禁用屏幕更新，从而对用户“隐藏”程序的执行情况（以及这样做的原因）。

禁用屏幕更新

Word 和 Excel 允许禁用屏幕更新——即停止文档工作区内的信息更新。软件窗体的其他部分——标题栏、命令栏、状态栏、滚动条等——仍然保持不断更新，但这些控件与文档工作区相比往往相对静态，因此也不会有很大变化。当然，如果改变了软件窗体或文档窗口的大小，即使禁用屏幕更新，仍然能看到这样的变化。

禁用屏幕更新的好处有两个：

- ◆ 首先，可以在一定程度上加速程序的运行。这样的改进似乎只在上个世纪的条件下才显得很重要，但是，对于那些显卡功能不够强劲的低配置的计算机，这一改进还是很明显的。许多在 2003 年以后生产的计算机都有相对较快的显卡，因此禁用屏幕更新并不会造成很大差异。通过关闭屏幕更新来提升运行速度的方法，仅限于那些大量改变屏幕上显示内容的程序。例如，假设一个 Word 程序要剪切当前文档的某个特定部分，并把它粘贴到一个新文档里，为其建立表格，并指定表格的格式。计算机会耗费一定的资源来更新监视器上显示的内容。如果用户不需要看到每一步的执行情况，因此对计算机资源的浪费，因此最好禁用屏幕更新。
- ◆ 其次，可以对用户隐藏那些不希望他们看到的程序进程。这似乎有些像极权主义者的行为，但往往更像是仁爱的专政政府和公共媒体之间的冲突：人们不应该看到某些会令他们痛苦的事物，而且有许多确实是大部分人不需要了解的。从 VBA 的角度考虑：如果用户不明白程序会按部就班地执行并最后达到目的，当他们看到屏幕上显示的信息时，可能会十分惊讶甚至感到不安。例如，如果某个程序需要移动一个打开的文件，就可能需要对用户隐藏那些过程——关闭文件，移动文件，然后重新打开已经移动过的文件。通过禁用屏幕更新就可以做到这一点。

禁用屏幕更新的一个主要缺点是使用户无法看到程序执行中的信息，而这些信息可能会对他们有用。在最不理想的情况下，由于屏幕没有适时更新，用户便认为程序进入了一个死循环或者计算机被挂起；然后他们可能会试着按 Ctrl + Break 键中断程序或按 Ctrl + Alt + Delete 键来打开“任务管理器”窗口，强制关闭软件。（一般在 VBA 运行代码时，任务管理器会显示宿主软件“没有响应”，所以这样做实际上没有用。）

为了阻止用户以上面那两种非正常方式关闭程序或软件，最好提前告知他们屏幕更新将会禁用。例如，可以在程序一开始就用信息框提示用户，或者可以显示一个对话框，让用户自己选择是关闭屏幕更新让程序执行更快一些，还是启用屏幕更新让程序以一般的速度运行并且显示可能有价值的信息。

如果不在程序一开始显示信息框或对话框，也可以在状态栏上显示这些信息，告诉用户程序正在怎样运行。Word 和 Excel 会持续更新软件的状态栏和标题栏，即使禁用屏幕更新，当然前提是状态栏和标题栏是可见的。要在状态栏上显示相应的信息，需要给 Application 对象的 StatusBar 属性赋一个恰当的字符串：

```
Application.StatusBar = "Word is creating 308 new documents for you to edit. Please wait..."
```

另外，也可以根据需要在两者之间做出选择：可以在程序运行某些部分时禁用屏幕更新，然后再启用，或在运行到其他部分时刷新屏幕。试想一下，某个程序需要根据一个已有的文档创建许多文档，并且套用它的格式。如果在程序最开始关闭屏幕更新，然后当一个文档创建完毕并套用好格式后刷新屏幕，用户就可以一次看到每一个文档（报告了程序的进程）而不会看到具体的套用格式的过程。另外，程序也会运行得快些。

要禁用屏幕更新，需要将 Application 对象的 ScreenUpdating 属性设为 False：

```
Application.ScreenUpdating = False
```

要启用屏幕更新，需要将 Application 对象的 ScreenUpdating 属性重新设为 True：

```
Application.ScreenUpdating = True
```

在 Word 中，要刷新屏幕使其显示缓存中的当前内容，需要使用 Application 对象的 ScreenRefresh 方法：

```
Application.ScreenRefresh
```

操控指针

在把 Word 和 Excel 当做宿主软件使用时，可能需要操控指针（鼠标指针）。在很多程序中都会用到这个功能。因为 VBA 会在程序运行时自动显示程序忙时的指针（沙漏），然后当程序运行结束时再恢复成正常的指针。然而有时也会需要人工控制指针。

警告：即使是使用计算机的时间仅仅只有几个月的用户，都会对指针有条件反射，沙漏指针（上升方向）意味着稍微喘口气（或计算机很慢），正好去拿一杯咖啡或者和同事聊两句，或者开始恐慌：“死机了，而前面 3 个小时的工作还没有保存。”通常情况下，并不需要给这样的条件反射添乱。因此，在程序忙时显示 I 型指针或其他的“普通”指针，或者在程序运行完以后显示程序忙指针，都是错误的做法。

在 Word 中操控指针

Word 通过 System 对象实现了指针。要操控指针，需要对 Cursor 属性进行设置。这是一个读/写的长整型属性，可以被设为以下的值：wdCursorIBeam (1)，表示 I 型指针；wdCursorNormal (2)，表示普通指针；wdCursorNorthWestArrow (3)，表示向左转了 45° 的箭头（尖指针）；以及 wdCursorWait (0)，表示程序忙指针。指针的精确形状取决于用户所选择的指针方案。

例如，下面的语句显示程序忙指针：

```
System.Cursor = wdCursorWait
```

在 Excel 中操控指针

Excel 允许通过 Application 对象的 Cursor 属性操控指针。这是一个读/写的长整型属性，可以被设为以下的值：xlDefault (-4143)，表示默认指针；xlWait (2)，表示沙漏指针；xlNorthwestArrow (1)，表示向左转了 45° 的箭头；xlIBeam (3)，表示 I 型指针。

例如，下面的语句表示显示沙漏指针：

```
Application.Cursor = xlWait
```

注意：当在 Excel 中显式设定 Application 对象的 Cursor 属性时，不要忘记在程序结束前重置，因为不这样做的话，指针将会一直保持修改后的样子。

在程序开始显示信息

在许多程序的开始，都可能需要显示信息框或对话框。为此，一般可以使用是/否信息框或确定/取消信息框，以便通知用户该程序将完成怎样的指令，并让他们有机会取消程序。

对话框通常会显示程序的选项（例如，选项按钮上的一些交互式独立选项或者需要在方框中勾选的非独立选项），允许用户输入信息（通过使用文本框、下拉列表框或组合框）以及让用户在由于失误而启动程序时取消程序。如果有时间为程序创建一个帮助文件或者用户要求这么做，可以为每一个信息框或对话框加入一个“帮助”按钮，链接到帮助文件中的相关主题。

提示：就像前面提到过的，也可使用信息框或对话框警告用户程序将关闭屏幕更新。相应地，如果程序为了保证其部分或全部的持续性，禁止了用户中断，也要通知用户。

程序结束时在信息框或对话框中显示信息

在某些程序中，需要为程序的指令收集一些信息，这样就可以在程序结束时把这些信息显示在信息框或对话框中告知用户。就像在第 13 章中提到的，信息框的使用十分简单，但是对显示在上面的文本有着很严格的限制——不能使用空格、制表符、回车和着重号来达到一些目的。另一方面，如果使用对话框，就可以随心所欲地加入任何文本（通过使用标签或文本框）甚至可以在必要的时候加入图片。

在程序执行时，最简单的收集信息的方法是建立一个或多个字符串变量来保存需要显示的信息。举一个这样的例子，回到第 12 章的程序清单 12.2，其中 cmdOK_Click 过程在创建一系列文件夹时收集相关信息，然后显示一个信息框，告诉用户程序做了哪些工作。

创建日志文件

如果需要在程序运行的过程中收集很多信息，并在程序结束之后把它们呈献给用户，或者只是为了日后参考，可以考虑使用日志文件，而不是信息框或对话框。日志文件对那些十分冗长而又包含了重要数据的程序十分有用：通过周期性地向日志文件写入信息（并保存），可以记录程序在崩溃之前做了哪些工作。

注意：如果想让日志文件既有足够的技术性，又能为普通用户所使用，就需要在包含了所有技术信息的同时，可以被普通用户读懂。例如，像这样一条信息“Office 文件 ‘Madrid’ (madrid060430.xls) 和 Office 文件 ‘Taos’ (taos060430.xls)

.xls) 的数据文件无法在指定地址 \\server2 \\ data \\ dayfiles \\ 找到，因此无法包含其信息”。通常这样的信息会比意义模糊的“错误代码 44E：必要信息丢失”的提示更有帮助。

例如，假设有一个 Word 程序每天从各种各样的资料中收集信息并记录在报告中。就可能需要在日志文件中记录下资料中的信息是否被成功获取，以及何时获取。程序清单 18.1 提供了这样一个例子。在程序的末尾，可以打开日志文件以方便用户检查程序是否成功地创建了报告文件，或者打开摘要文件，让用户自己阅读报告。

程序清单 18.1

```
1. Sub Create_Log_File()
2.
3.     Dim strDate As String
4.     Dim strPath As String
5.     Dim strCity(10) As String
6.     Dim strLogText As String
7.     Dim strLogName As String
8.     Dim strSummary As String
9.     Dim strFile As String
10.    Dim i As Integer
11.
12.    On Error GoTo Crash
13.
14.    strCity(1) = "Chicago"
15.    strCity(2) = "Toronto"
16.    strCity(3) = "New York"
17.    strCity(4) = "London"
18.    strCity(5) = "Lyons"
19.    strCity(6) = "Antwerp"
20.    strCity(7) = "Copenhagen"
21.    strCity(8) = "Krakow"
22.    strCity(9) = "Pinsk"
23.    strCity(10) = "Belgrade"
24.
25.    strDate = Month(Date) & "-" & Day(Date) & "-" _
& Year(Date)
26.    strPath = "f:\Daily Data\" 
27.    strLogName = strPath & "Reports\Log for " _
& strDate & ".doc"
28.    strSummary = strPath & "Reports\Summary for " _
& strDate & ".doc"
29.    Documents.Add
30.    ActiveDocument.SaveAs strSummary
31.
32.    For i = 1 To 10
33.        strFile = strPath & strCity(i) & " " & strDate & ".doc"
34.        If Dir(strFile) <> "" Then
35.            Documents.Open strFile
36.            Documents(strFile).Paragraphs(1).Range.Copy
37.            Documents(strFile).Close _
SaveChanges:=wdDoNotSaveChanges
38.            With Documents(strSummary)
```

```
40.             Selection.EndKey Unit:=wdStory
41.             Selection.Paste
42.             .Save
43.         End With
44.         strLogText = strLogText & strCity(i) -
& vbTab & "OK" & vbCrLf
45.     Else
46.         strLogText = strLogText & strCity(i) -
& vbTab & "No file" & vbCrLf
47.     End If
48. Next i
49.
50. Crash:
51.
52. Documents.Add
53. Selection.TypeText strLogText
54. ActiveDocument.SaveAs strLogName
55. Documents(strLogName).Close
56. Documents(strSummary).Close
57.
58. End Sub
```

在程序清单 18.1 中创建了一个新文档来保存摘要，逐个打开一系列文件，复制每个文件的第一段并粘贴到摘要文件中，然后关闭之前打开的文件。在程序执行时，用一个字符变量保存日志信息，根据这些信息在程序末尾或有错误发生时创建日志文件。以下是程序的运行过程：

- ◆ 第 3 行到第 9 行声明了 6 个字符型变量——strDate、strPath、strLogText、strLogName、strSummary 和 strFile——以及一个字符数组 strCity，其中包含了 10 个元素。（程序使用了 Option Base 1 语句，而没有列出来，因此 strCity(10) 数组中包含了 10 个元素而不是 11 个。）第 10 行声明了整型变量 i，在程序中用做计数器。
- ◆ 第 11 行是空行。第 12 行用 On Error GoTo 语句开始了一个处理错误的代码，在有错误发生时将程序引导到标签 Crash 处。第 13 行为空行。
- ◆ 第 14 行到第 23 行将假设的公司的 10 个虚构的办事处的名称赋值给数组 strCity。第 24 行为空行。
- ◆ 第 25 行给变量 strDate 赋值。其内容为当天的日期，包括月日年（用短划线连接各部分）。月日年的值分别使用相应的函数求得。例如，2006 年 1 月 21 日将会表示为 1—21—2006。（用这样的格式表示日期是因为 Windows 无法处理含有斜杠（\）的文件名——斜杠被用于表示文件路径。）
- ◆ 第 26 行将 strPath 赋值为 f:\Daily Data\ 文件夹。然后第 27 行在 \Reports\ 子目录中构建日志文件的文件名。第 28 行构建摘要文件的文件名，同样也在 \Reports\ 子目录下。
- ◆ 第 29 行基于 Normal.dot 模板创建一个新文件，第 30 行用变量 strSummary 中储存的文件名保存文件。第 31 行是空行。
- ◆ 第 32 行开始 For … Next Each 循环，从 i = 1 运行到 i = 10。第 33 行将字符变量 strFile 赋值，作为文件名使用。该文件名包含了数组 strCity 中储存的城市名称：str-

Path & strCity & " " & strDate & ". doc"。

- ◆ 第 34 行开始了一个 If 语句块，检查 Dir (strFile) 是否返回了一个空字符串。如果没有，第 35 行打开文件，其文件名为 strFile 变量中的值，第 36 行复制该文件的第一段，第 37 行关闭文件而不保存。程序并没有对文档进行任何修改，但是，如果文档中包含了某些即时更新区域（比如日期或者链接），它们会在打开文档时自动更新，这样就对文档进行了修改。加入一个 SaveChanges 参数可以确保用户不会看到一个意想不到的信息框弹出来提醒他们保存文档，而事实上他们知道自己并没有修改这些文件。（另一个方法是将文档的 Saved 属性设为 True，然后关闭文件。这样就不需要使用 SaveChanges 参数了）。
- ◆ 第 39 行到第 43 行是一个 With 语句块，用于处理上面的由 strSummary 指定的 Documents 对象。第 40 行使用把 Unit 参数指定为 wdStory 的 EndKey 方法，将选区移动到文档的末尾。第 41 行把从刚才打开的文件中复制的信息粘贴到摘要文件中。第 42 行保存文档。第 43 行结束 With 语句块。
- ◆ 第 44 行向 strLogText 变量中添加如下信息：strCity (i) 中的内容、一个制表符、文本“OK”以及回车。这样将制作一个简单的表格，其中列出了城市以及它们的报告的状态。
- ◆ 如果第 34 行的条件没有满足，程序转到第 45 行的 Else 分支语句，第 46 行向 strLogText 添加如下信息：strCity (i) 中的内容、一个制表符、文本“No file”以及回车。第 47 行结束 If 语句块，第 48 行结束 For … Next 循环，返回第 32 行继续执行程序。
- ◆ 第 49 行是空行。第 50 行为 Crash 标签，表示错误处理代码的开始。不像很多其他的程序，你不需要在执行错误处理代码之前停止程序的运行——就像这里所看到的，即使有错误发生，仍然需要执行这些语句（创建日志文件）。第 51 行是空行。
- ◆ 第 52 行基于默认模板创建一个新文档。第 53 行向该文档中输入 strLogText 中的信息。第 54 行用 strLogName 中的文件名保存文档。第 55 行关闭这个新文档（另外，也可以不关闭文档以便让用户阅读）。第 56 行关闭摘要文档（该文档从创建之时就一直保持打开状态；同样，可以让它继续保持打开状态以便用户阅读或者让用户选择是否关闭该文档）。第 57 行是空行，第 58 行结束程序。

确保程序在适当的条件下运行

编写一个出色的程序的另一个要点是确定程序是否在适当的条件下运行。无论在何种情况下，几乎都不可能达到完全理想的条件，但是仍然可以采取以下的基本措施：

- ◆ 如果程序需要一个文件保持打开的状态，就需要确保该文件已经打开——否则运行一次都会接到错误的提示。例如，在 Excel 中，可以检查 Workbooks 集合的 Count 属性，确保至少有一个工作簿是打开的：

```
If Workbooks.Count = 0 Then
    MsgBox "This procedure will not run without a "
    & "workbook open.", vbOKOnly + vbExclamation,
    "No Workbook Is Open"
```

- ◆ 如果程序中有某个部分定义了它所要处理的对象，那么就要确定程序所处理的对象就

是所定义的。例如，在一个 Excel 程序中，对用户所选定的图表应用了一个复杂的格式，就必须确保所选择的图表就是用户要处理的那个。如果试图使用与图表有关的命令来操控另一个对象，就很有可能导致错误。

- ◆ 确保文件中包含有程序需要的元素。（如果不是这样，就很可能导致错误。）另外，也可以专门编写代码来处理这种由于元素不存在而造成的错误。

在程序完成后重置

程序就像是孩子或室友，必须学会把自己用过的东西整理好并且打扫干净。程序的重置包括以下几项：

- ◆ 撤销所有为了方便自己运行而做的修改。
- ◆ 关闭所有不再需要打开的文件。
- ◆ 移除程序运行时产生的所有临时文件。

撤销程序做出的修改

有时候需要对文档进行修改以保证程序能够正常运行。这里有两个例子：

- ◆ 在 Word 中，可能需要对表格的一半应用某种格式，而对另一半则不需要。这时，比较好的办法应该是把表格拆分开来，这样就可以方便地选择需要的那一部分，对它进行操作，而不会影响剩下的那一部分。如果执行了这样的操作，之后就需要在删除之前在表格之间插入的空行，重新合并两个表格。要完成这一操作，最简单的办法是在插入的空行处加入书签；这样就可以回到书签并同时删除书签和空行。也可以采取另一种办法：用 Set 语句将空行定义为一个范围（range），然后回到该范围，移除空行。
- ◆ 在 Excel 中，可能需要在工作簿中定义一个命名范围，这样就可以方便地在代码中引用他们。（通常，最好经由 VBA 来使用范围，这样就不会在工作簿中留下不需要的范围。）当使用完这些范围后删除它们。

移除临时文件和临时文件夹

在一个复杂的程序里，可能需要创建一些临时文件来临时储存或操控一些信息，也可能需要创建一些临时文件夹来储存文件。例如，在 Word 中，如果需要修改一个长文档的某几段文字的格式，比较好的方法是复制要修改的部分，把它们粘贴到一个新的空文档中，然后对它们进行操作。这样做比在同一个文件中工作要容易得多，而且不必担心会无意中改变文章的其他段落。同样，在 PowerPoint 中，可能需要创建一个新的演示文稿，用来临时存储或者备份那些错综复杂的对象。

创建临时文件通常是基于对安全和顺利运行程序的考虑，但这对于计算机用户是不利的：这些信息对于他们来说可能毫无用处，却把他们的硬盘搞得杂乱不堪。创建临时文件夹来保存临时文件的方法则更加不利。因此，一定要有这样的意识：每次都必须花一些额外的精力来清理程序运行留下的痕迹，删除那些程序运行时创建的临时文件和文件夹。也许有人会质疑：“不是这样的，没有一个商业软件是这样做的，甚至 Microsoft 自己的软件也没有做到。”事实确实如此，但这并不意味着可以参考这些不好的事情。

如果程序能够移除所有运行时创建的临时文件，就可以很成功地让用户无法察觉这些文件的创建以及之后的删除。这样的想法似乎很诱人，但在大多数情况下并不明智。最好还是警告用户，程序中会产生一些临时文件。甚至可以让用户指定或者创建合适的文件夹来储存这些临时文件，或者给用户一个清单，在上面列出创建了哪些文件以及这些文件是否被成功删除。如果这样做，那么在程序出错或在执行中被中断时，用户就可以安全地删除任何留在他们计算机上的临时文件。

另一种方法是使用 API（应用程序接口）命令的 GetTempDir 和 GetTempFileName 命令来返回计算机的临时文件夹和临时文件名以供使用。API 命令的知识已经超出了本书的范围，但是即使使用了临时文件夹，仍然必须删除那些创建在其中而又不需要再使用的临时文件。同样，能做到这一点的商业软件开发者的数量少到令人失望。

建立临时文件夹

可以使用 MkDir 语句来创建一个文件夹。例如，下面的语句在 C 盘根目录下创建了一个名为 Scratch Folder 的临时文件夹：

```
MkDir "c:\Scratch Folder"
```

在创建文件夹之前，用 Dir 语句来检查一下该文件夹的名称是否已经被使用。（如果相同名称的文件夹已经存在，那么就会发生错误。）

提示：可以根据日期或者时间来创建临时文件夹的名称，这样出现同名文件夹的可能性就会大大减小。也可以用 Rnd 函数来产生随机数以作为文件夹名称的一个部分。

删除临时文件夹

可以使用 RmDir 语句来删除一个空的文件夹。（必须保证文件夹为空，否则 RmDir 命令将无法执行。）例如，下面的语句删除了 C 盘根目录下的名为 Scratch Folder 的临时文件夹：

```
RmDir "c:\Scratch Folder"
```

第 19 章 用 VBA 的安全特性保护代码

- ◆ 理解 VBA 如何实现安全性
- ◆ 为宏工程添加数字签名
- ◆ 获取数字证书
- ◆ 选择适当的安全级
- ◆ 指定可靠发行商或可靠来源
- ◆ 锁定代码

这一章将讨论如何使用 VBA 提供的安全工具来分发和保护宏和 VBA 代码。VBA 的安全性包括 3 个部分：保护软件免遭 VBA 恶意程序的侵害；确保自己的代码不是恶意程序以保证其能运行；保护程序、防止盗用、修改或窥测。

理解 VBA 如何实现安全性

要保护软件免遭 VBA 恶意程序的侵害，可以在运行 VBA 代码时选择软件希望使用的安全级，这样软件就只会运行那些可靠来源（也称做可靠发行商）的程序。可以自己指定哪些发行商是可靠的以及他们的可靠程度。可靠发行商有可能是在同一公司工作的同事；或者持有可靠的数字证书的第三方，比如 VeriSign 授权证书。因为（在本例中）VeriSign 是可以信任的，而从 VeriSign 获得了数字证书的第三方就是可以信任的。

要确保自己的代码是安全可靠的，需要在包含了自定义控件或宏工程（代码模块、类模块或用户窗体）的文档工程或项目样板中签署数字签名。该签名根据数字证书产生，用来表明开发者或其公司的独一无二的身份。本章首先介绍这一技术，因为这是指定安全级的基础。

要保护代码，可以用密码锁定宏，这样就没有人能打开代码了。不仅可以阻止别人修改代码，而且也可以阻止别人中断代码的运行或恶意获取代码，保护知识产权：如果没有人能看到代码，那么谁也不能偷走里面的创意。本章的最后将介绍如何做到这一点。

注意：并不是所有的 VBA 宿主软件都支持 VBA 安全特性——因此，本章的内容可能不一定适合读者的情况。特别是使用 VBA 5.0 而不是 VBA 6.0 的软件是不支持安全特性的。

为宏工程添加数字签名

许多字段入个

VBA 提供一种安全机制，用数字签名来保护宏工程。数字签名提供了一种可以标记项目发行商的方法，可以帮助用户判断代码是否可靠。如果信任代码可以生成有用的程序，就可打开工程，并且运行代码。如果对代码的来源或信息心存疑惑，就可以不打开该项目或者

打开项目但禁用宏。

对于其他人也是一样，需要在自己的项目里签名，让别人知道项目是从哪里来的以及是谁创建了它们。一旦在项目中添加了签名，在任何指定自己为可靠发行商的软件中，代码都可以顺利运行。（这里假设软件中的安全级被设定为中、高或非常高。在本章的较后部分将介绍如何设定安全级。）

这一部分讨论什么是数字证书，它的实际意义是什么，如何获取以及如何利用它来创建数字签名。

注意：VBA 的安全机制以及证书的列表和可靠来源，在计算机中的所有可使用 VBA 的软件中共享。因此如果在某一个软件中指定了一个可靠来源，其他所有支持 VBA 安全特性的软件都会承认这个可靠来源。例如，如果在 Word 中打开了一个带有代码的文档，并且选择信任该代码的来源，那么 Excel 和 Outlook 都会信任该来源，在打开来自该来源的项目时就不会再弹出提示窗口。

什么是数字证书

数字证书是一段代码，用来表明项目所有者的身份。可以利用数字证书来为项目创建数字签名。项目可以是文档工程、模板工程或加载项。并不是只有包含宏、过程、用户窗体、类或 VBA 代码的项目才能添加数字签名，尽管通常是因为有了那些内容才需要添加数字签名。

数字签名应用于整个宏工程——一般为一个文档工程或模板工程。不能仅仅对宏工程中的某一部分添加数字签名，比如一个模块的代码或一个用户窗体。数字签名将包括模块工程中的所有项目，即模块、用户窗体、类以及引用。

获取数字证书

共有下面的几种数字证书：程序员自己创建的、从公司或组织里得到的以及从商业权威机构处获得的授权证书或认证证书（CA）。

程序员自己创建的数字证书除了自己以及自己相信的人以外用处不大。而从商业权威机构获得的证书则可以在世界范围内广泛使用。由公司颁发的证书，适用范围在两者之间：在大多数情况下，公司都会持有商业权威机构的认证证书，也就是说该权威机构可以担保这个公司是值得信赖的；公司选择他们认为可以信赖的人，并颁发证书是另一回事，这是另一套建立信赖的体系。然而，像 Windows Server 2003 这样的服务器软件有独立的认证系统，并不需要通过商业权威机构的认证，因此需要注意哪些证书是可以信赖的。在后面的“证书的提供者及证书的含义”小节中，将会介绍如何辨别证书的来源和意义。

创建个人数字证书

最快和最简单的获得数字证书的方法是自己创建一个。Microsoft Office 2003, Office XP 和 Office 2000 都提供了这项功能。

要了解数字证书的工作原理，最好自己多创建几个并把它们应用于文件。通过指定某些签名可靠而另一些不可靠，就可以清楚地了解数字证书是如何工作的，从而有效地禁止可疑代码在系统上运行。

要打开“创建数字证书”对话框（如图 19.1 所示），可进行以下操作：

- ◆ 针对 Office 2003，选择“开始”>“所有程序”>“Microsoft Office”>“Microsoft Office 工具”>“VBA 项目的数字证书”。
- ◆ 针对 Office XP，选择“开始”>“运行”，键入“%programfiles%”，然后回车在“Windows 浏览器”窗口中打开“Program Files”文件夹。打开“Microsoft Office\Office”文件夹，然后双击“SELFCERT.EXE”文件。
- ◆ 针对 Office 2000，选择“开始”>“运行”，键入“%programfiles%”，然后回车在“Windows 浏览器”窗口中打开“Program Files”文件夹。打开“Microsoft Office\Office”文件夹，然后双击“SELFCERT.EXE”文件。或者也可以采用另一种方法，可以运行 Office 安装光盘的“PFiles\MSOffice\Office”目录下的“SELFCERT.EXE”文件。

在文本框中键入证书的名称，然后单击“确定”按钮。SelfCert 软件就创建了相应的数字签名，并自动安装，然后显示“SelfCert 成功”对话框（见图 19.2）

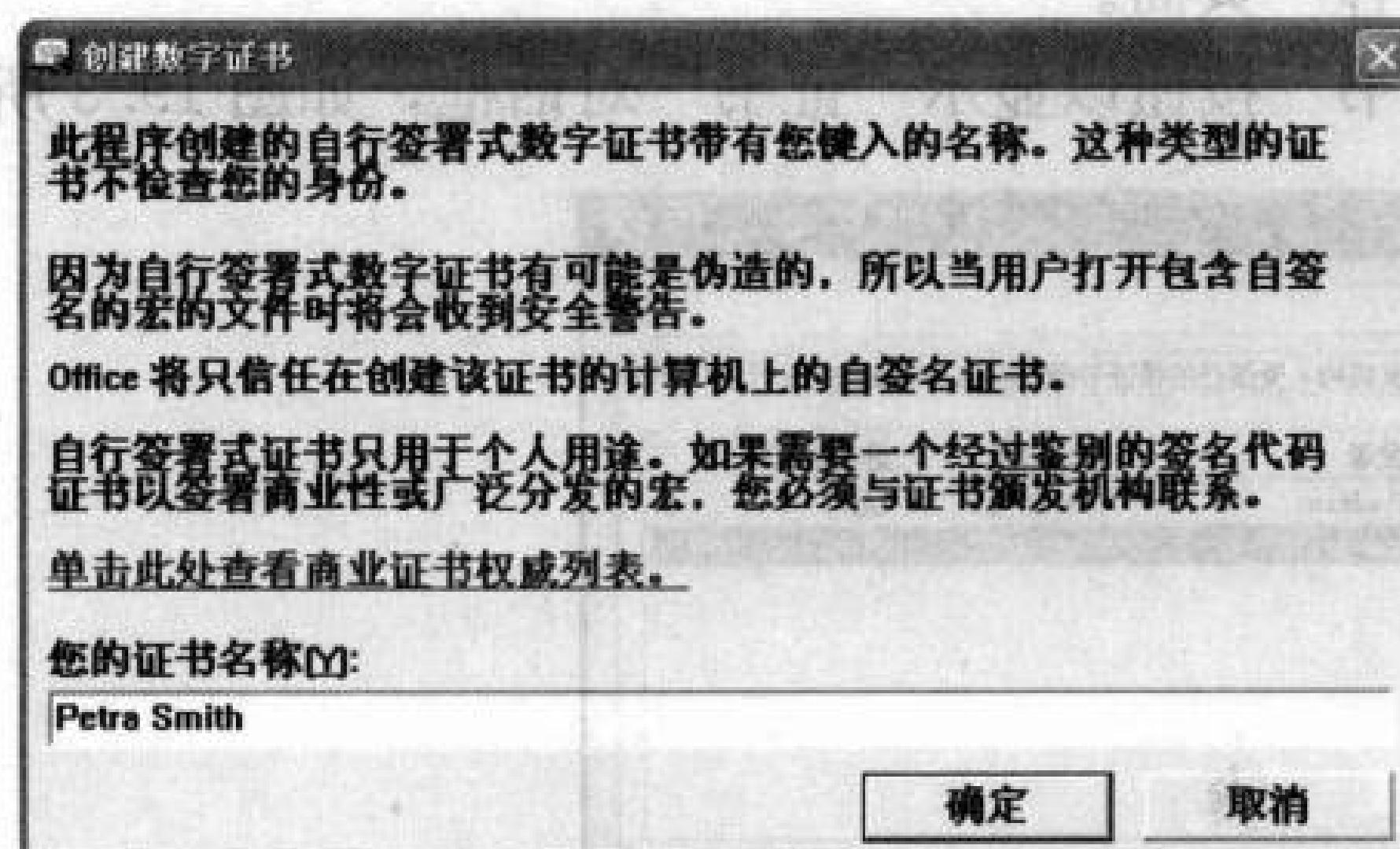


图 19.1 在“创建数字证书”对话框中输入希望给出的数字证书的名称



图 19.2 “SelfCert 成功”对话框提示证书已经创建完成

从公司处获取数字证书

第二个选择是向公司申请使用公司所持有的的数字证书。具体的流程因公司的不同而不同。公司通过数字证书服务器提供的证书和商业认证机构提供的数字证书（这一内容将在下一部分介绍）基本相同。区别在于，公司是向一个由它分配的群体里发放证书，不要对每一个证书进行认证；或者，即使没有从权威机构获得授权，公司也可以发放内容一致的证书。

从商业权威机构获得数字证书

第三个选择是从商业权威机构获得认证。例如 VeriSign (www.verisign.com)、Thawte 公司 (www.thawte.com, 一个 VeriSign 公司) 或 GlobalSign NV-AS (www.globalsign.net)。

证书的种类有很多，可根据需要进行选择。如果想要开发并发布软件，就可能需要考虑获取开发者的证书。

具体的程序取决于不同的认证机构以及证书的种类。一般来说，证书所能代表的可靠性越高，需要提供的证明材料也就越多。例如，只需要提供一个有效的 E-mail 地址就可以获得一个基本的证书，但这样的证书并不足以让大部分人相信此人是可靠的。有些认证需要当

事人亲自到场注册，并提供完整的书面材料（比如护照、驾照或其他身份证明材料）。这样的证书比前一种显得更加可靠。

安装数字证书

获取了数字证书后，需要进行安装，这样 Windows 及其他相关软件才能正常使用它。

注意：Office SelfCert 程序在创建个人证书时就自动安装了它们。如果在计算机上创建了个人数字证书，则不需要再在同一台计算机上安装。如果想要熟悉安装步骤，就需要换一台计算机。

可以遵循以下的步骤安装数字证书：

1. 选择“开始”>“控制面板”，打开“控制面板”窗口。在“分类视图”中，单击“网络与 Internet 连接”，然后单击“Internet 选项”图标。在“经典视图”中，双击“Internet 选项”图标，Windows 将会显示“Internet”属性对话框。
2. 单击“内容”选项卡，显示“内容”页面。
3. 在“证书”分组框中，单击“证书”按钮以显示“证书”对话框，如图 19.3 所示。

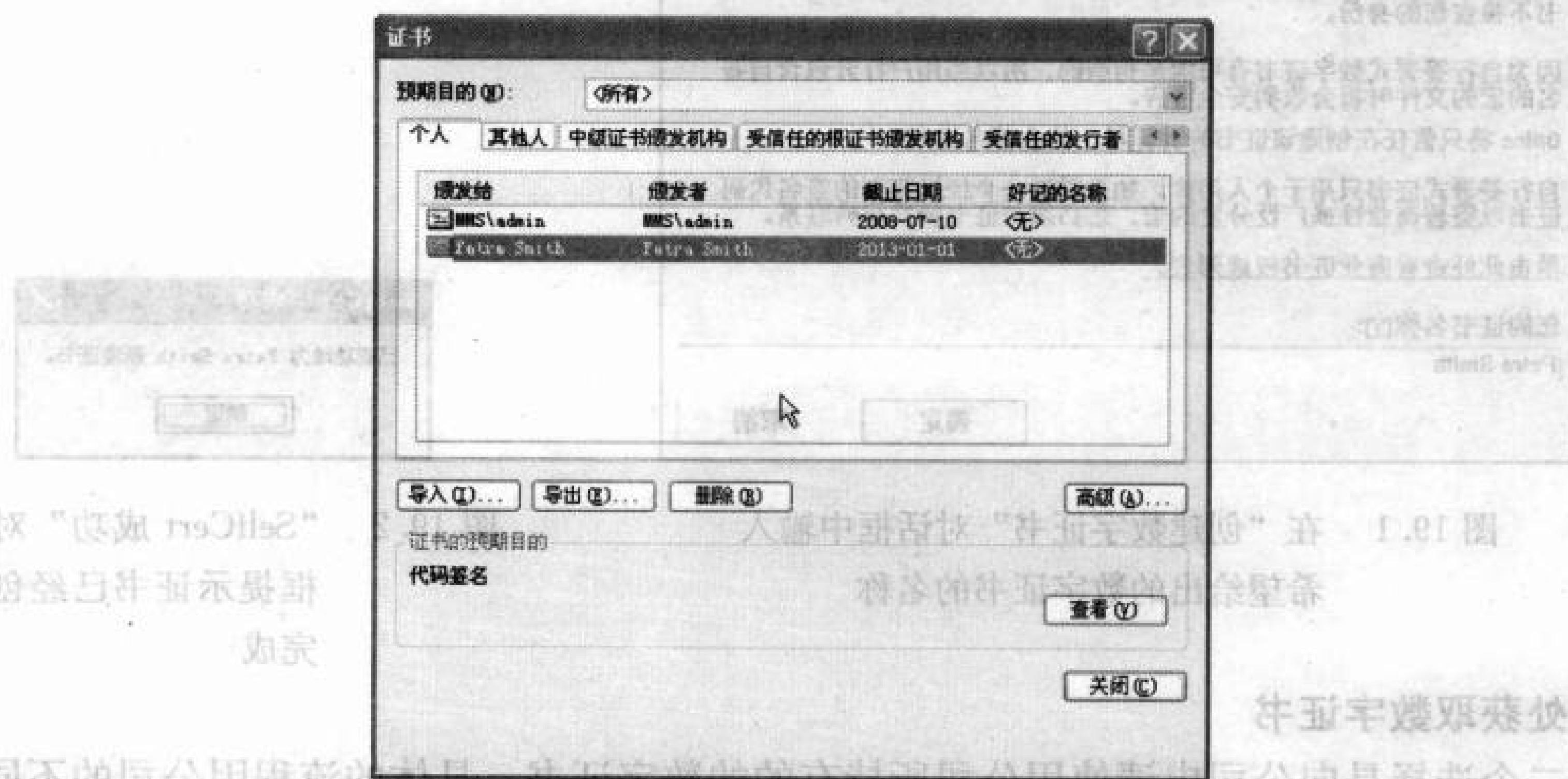


图 19.3 Windows 提供“证书”对话框来管理数字证书

4. 单击“导入”按钮，开始证书导入向导，然后单击“下一步”按钮，显示“证书导入向导”的“要导入的文件”页面。
5. 在“文件名”文本框中输入需要导入的证书文件名称。
 - ◆ 手工输入证书的名称（也可以使用复制、粘贴的方法），或者单击“浏览”按钮，显示“打开”对话框，找到证书的位置，然后单击“打开”按钮。注意“文件类型”下拉列表中的文件类型必须与证书的文件类型相一致，这样证书的文件才会出现在对话框中。例如，如果保存证书的是.CER 类型的文件，则需要在下拉列表中选择“X.509 证书”，这样扩展名为.CER 和.CRT 的文件会在出现在上方的列表中。
 - ◆ 单击“下一步”按钮显示“证书导入向导”对话框的“证书存储”页面，如图 19.4 所示。
6. 选择储存证书的方式。
 - ◆ 要让 Windows 根据证书的文件类型自动分类并储存在默认的证书库中，可以选择

- “根据证书类型，自动选择证书存储”选项按钮。
- ◆ 要让 Windows 把证书储存在指定文件夹内，选择“将所有的证书放入下列存储”选项按钮。要指定存储区的位置，可单击“浏览”按钮以显示“选择证书存储”对话框，如图 19.5 所示，选择证书存储区（例如，个人），然后单击“确定”按钮。要在证书存储区内指定保存路径，可勾选“显示物理存储区”选项，然后单击存储区左边的加号（+），显示其子文件夹，指定所要的文件夹，然后单击“确定”按钮。

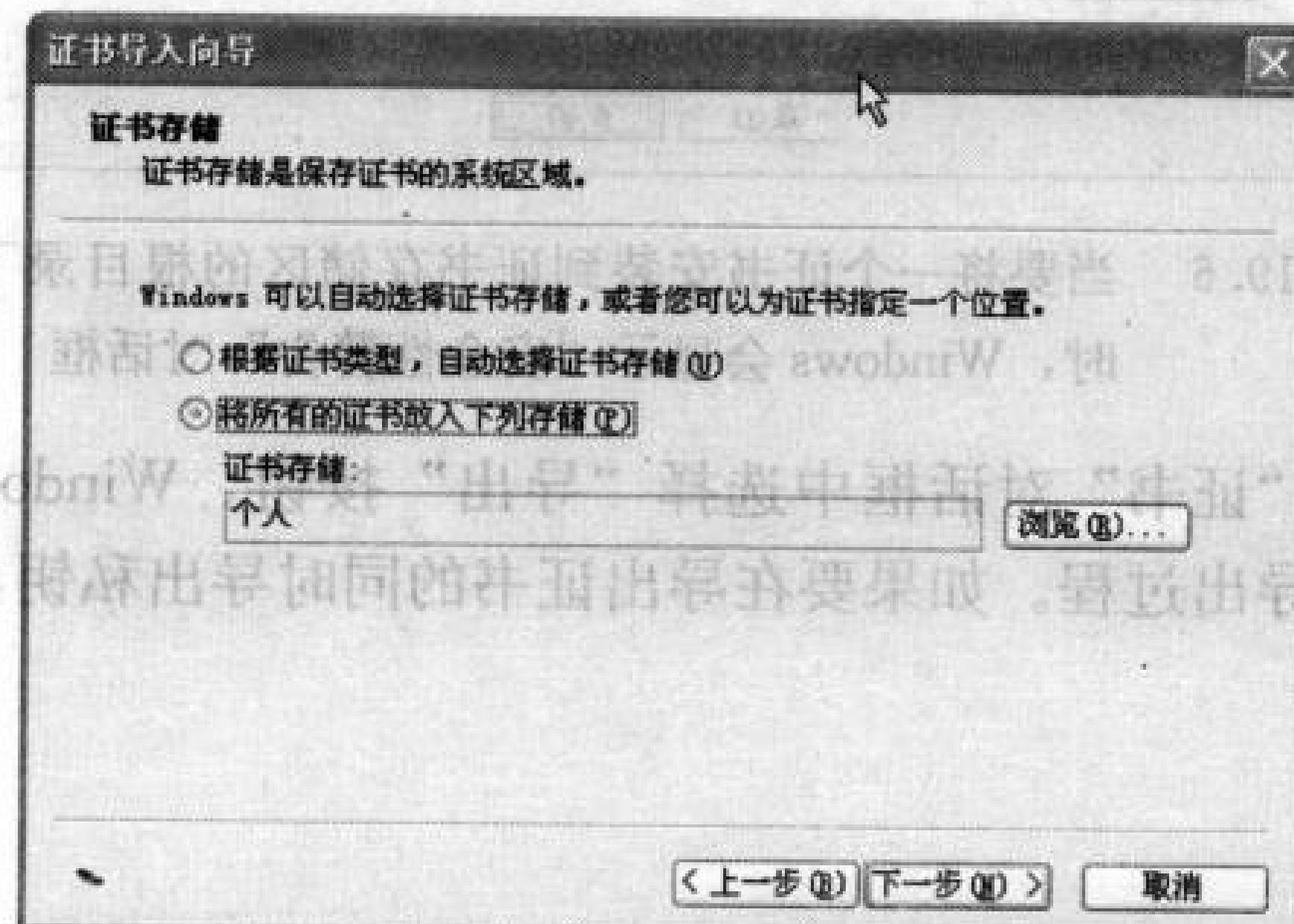


图 19.4 在“证书导入向导”对话框的“证书存储”页面上，选择要将导入的证书储存在哪个证书库中

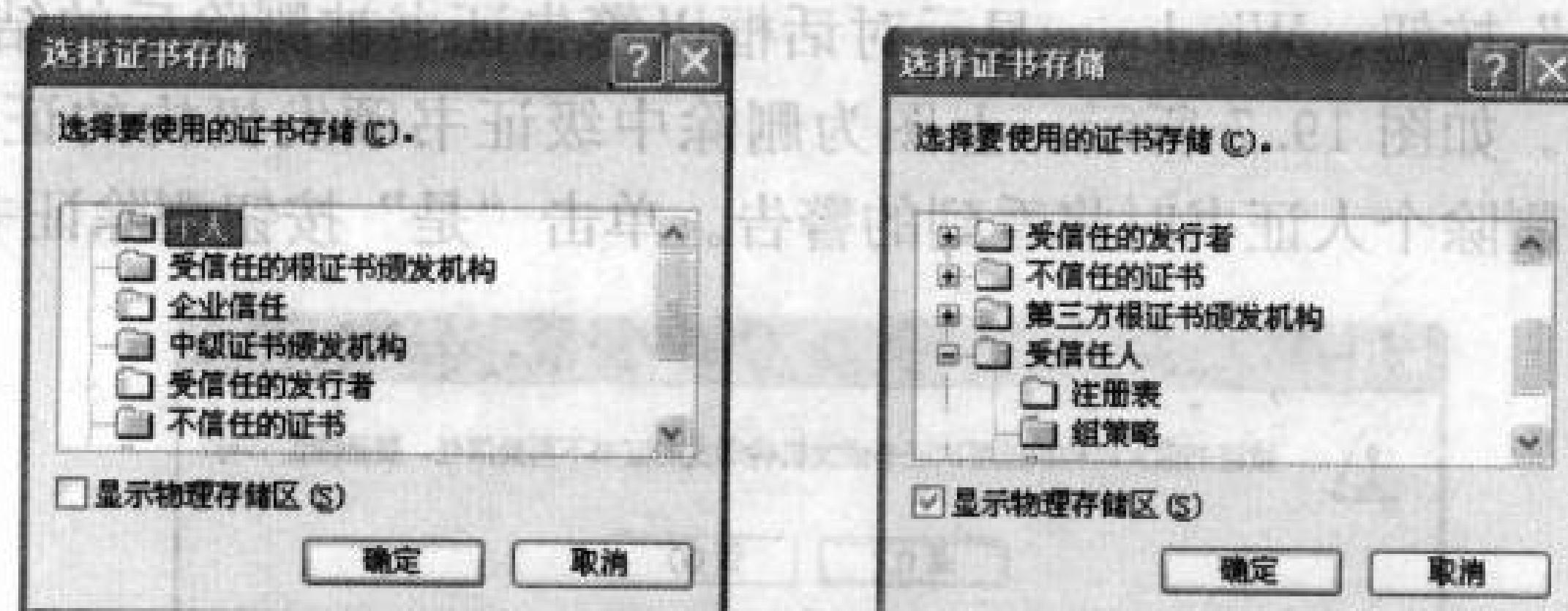


图 19.5 使用“选择证书存储”对话框指定证书存储区。左图显示的是证书的存储类别，右图显示具体的存放地点

7. 单击“下一步”按钮，完成导入进程的设置，此时将会显示“完成证书导入向导”对话框以确认设置的内容。
8. 检查设置，然后单击“完成”按钮，证书导入向导导入证书，并确认导入成功。

如果想把证书储存在证书存储区的根目录下而不是其中的某一个存储区，那么 Windows 会显示一个“安全性警告”对话框（如图 19.6 所示），确保用户明白如果这样做，Windows 将会自动认为所有由该认证机构颁发的证书都是可靠的。双击证书，就像在其他存储类别中保存证书那样。

证书已经安装完毕，相应的页面上会显示“证书”对话框。

导出数字证书

可能会需要导出证书作为备份（这样就可以把它们存在计算机以外的移动存储器上）或安装在另一台计算机中。出于安全的考虑，在证书安装完之后，不应该继续把证书文件储存

在硬盘上，因为这有一定的风险。

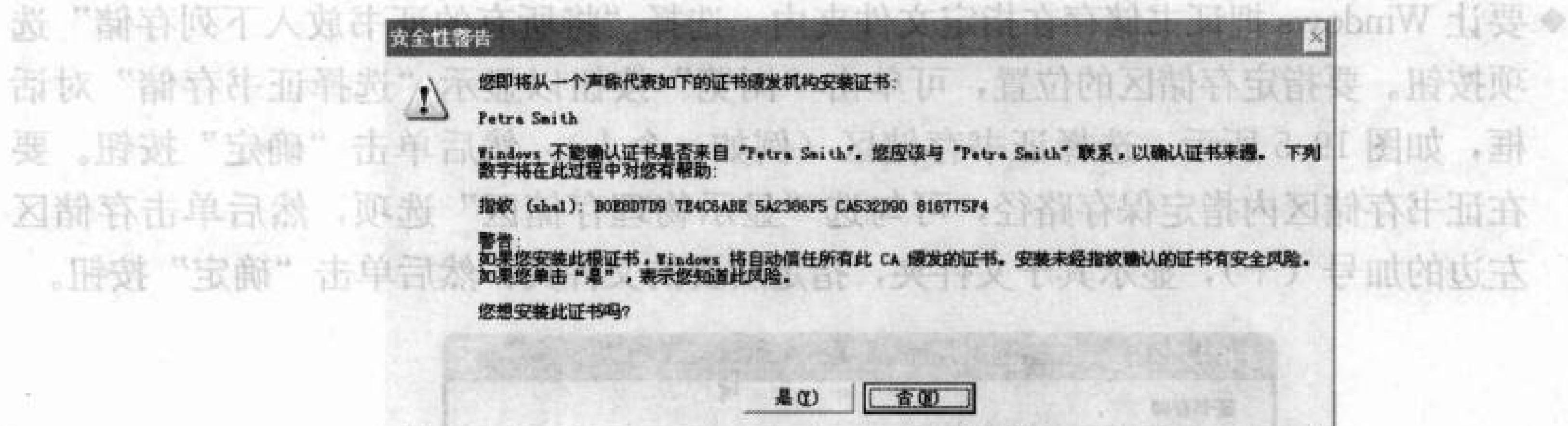


图 19.6 当要将一个证书安装到证书存储区的根目录下时，Windows 会显示“安全性警告”对话框

要导出证书，是在“证书”对话框中选择“导出”按钮。Windows 会开始证书导出向导，引导用户完成整个导出过程。如果要在导出证书的同时导出私钥，注意一定要用密码加以保护。

移除数字证书

要从 Windows 的证书存储区中移除数字证书，可遵循以下步骤：

1. 调用“证书”对话框。
2. 选择包含涉及的证书的选项卡，然后选择需要移除的证书。
3. 单击“删除”按钮，Windows 显示对话框以警告证书被删除后的结果，并让用户确认删除操作。如图 19.7 所示，上图为删除中级证书颁发机构的证书时将看到的警告，下图为删除个人证书时将看到的警告。单击“是”按钮删除证书。

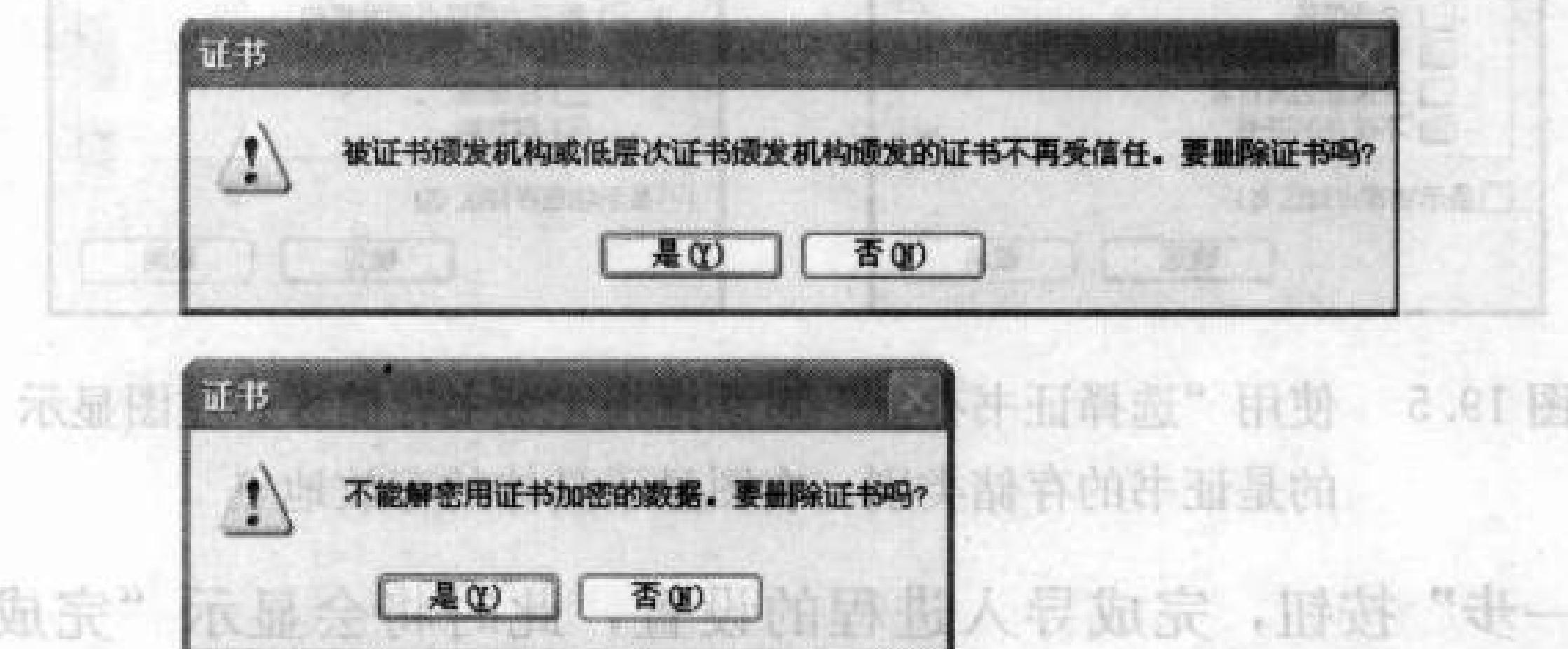


图 19.7 在删除证书时证书管理器将给出的两条警告信息

为宏工程添加数字签名

一旦完成了宏工程并做好了发布的准备，就需要给宏添加数字签名，这样那些使用高的或非常高的安全级别的用户就可以使用它了。

给宏工程添加数字签名，可遵循以下步骤：

1. 在 Visual Basic 编辑器中，定位到包含该宏的文档或模板工程。
2. 在“工程资源管理器”中选择该工程。
3. 选择“工具”>“数字签名”以显示“数字签名”对话框（如图 19.8 所示）。

注意：如果“数字签名”对话框已经为区域列出了所需要的签名，只要单击“确定”按钮，继续使用该证书即可。

4. 单击“选择”按钮以显示“选择证书”对话框（如图 19.9 所示）。
5. 为宏工程选择所需的证书。
6. 单击“确定”按钮应用选择，并关闭“选择证书”对话框。
7. 单击“确定”按钮关闭“数字证书”对话框。
8. 单击“标准”工具栏上的“保存”按钮，按 Ctrl + S 键或选择“文件”>“保存”保存已有数字签名的文档或模板工程。

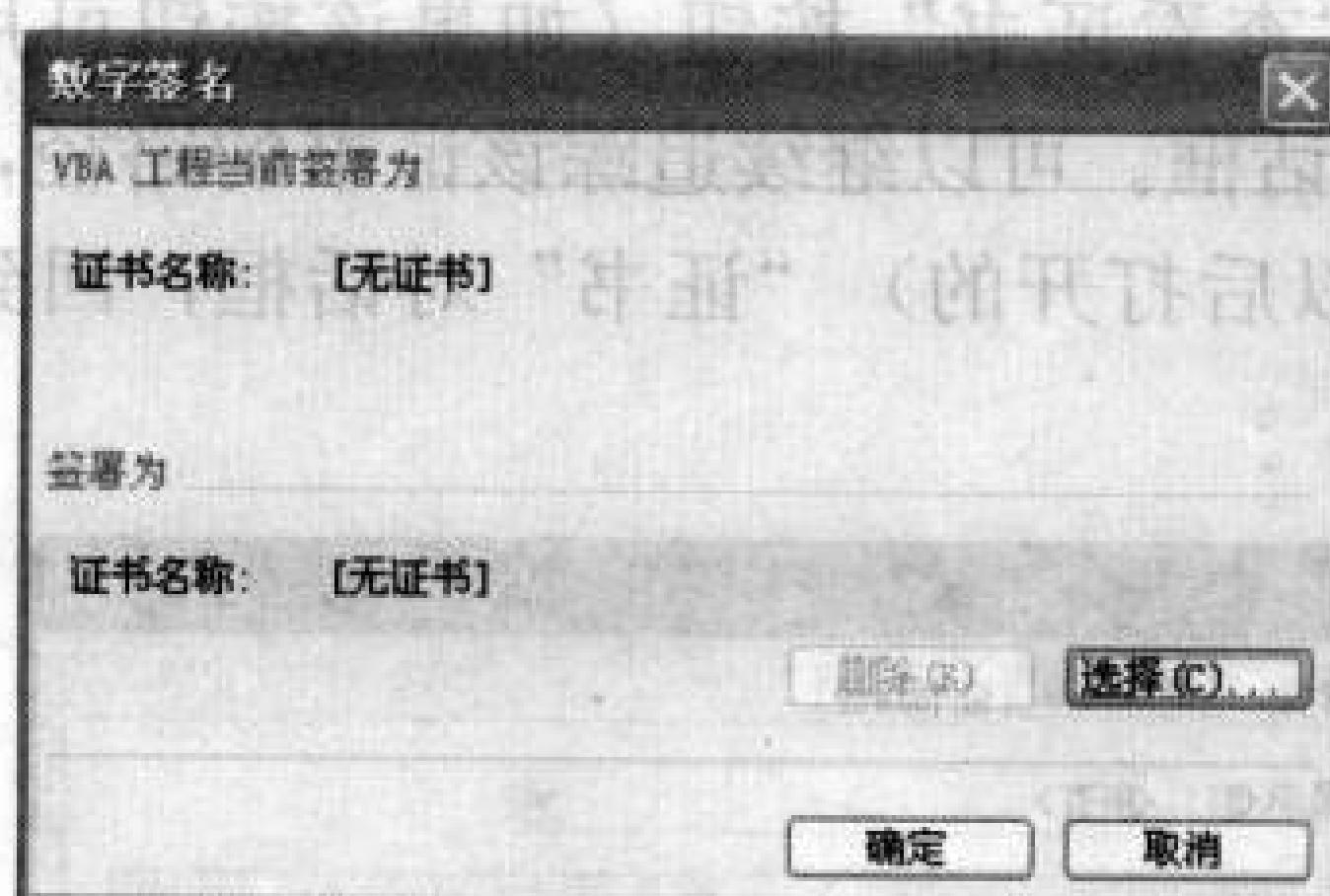


图 19.8 使用“数字签名”对话框
为宏工程指定数字签名

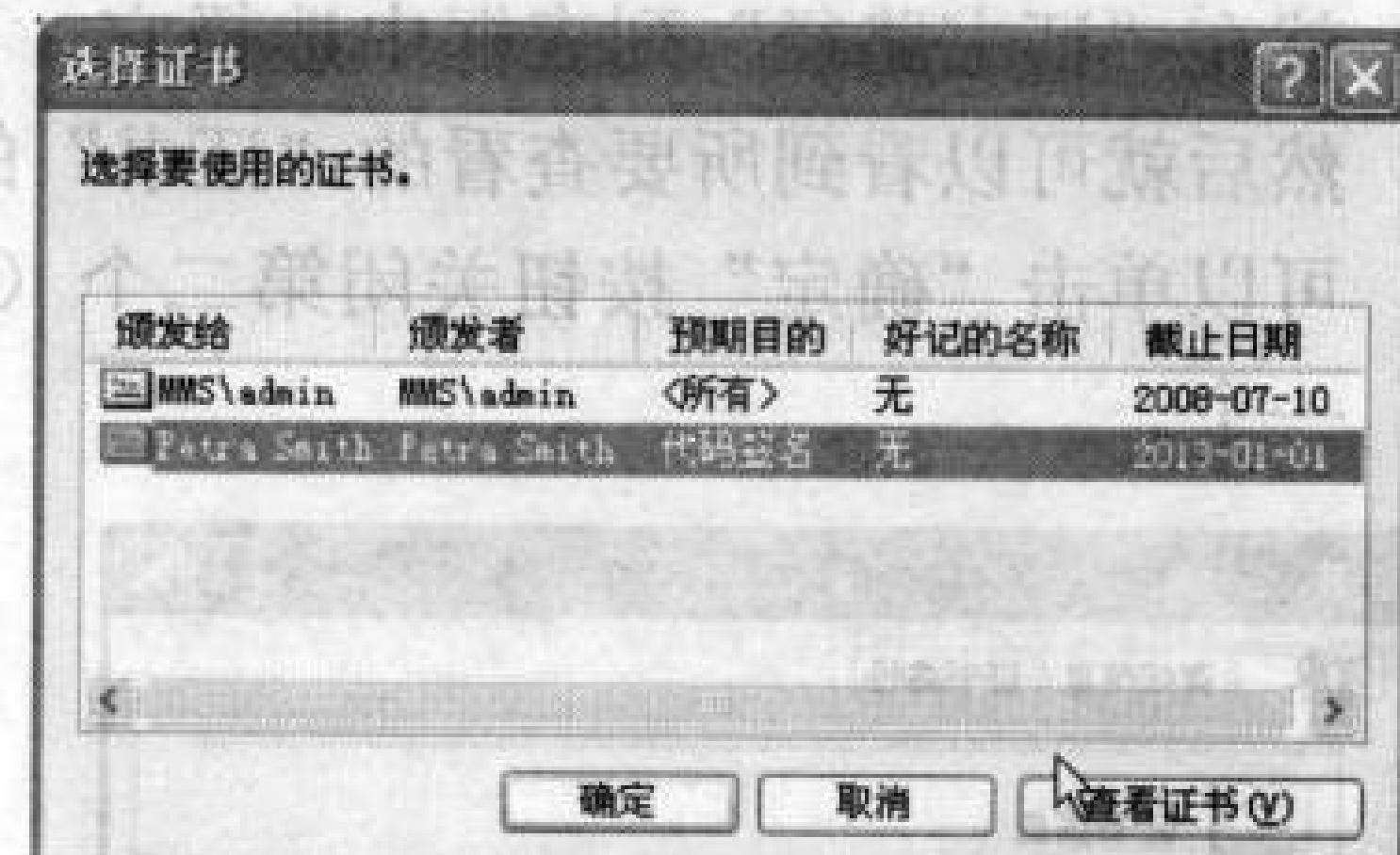


图 19.9 使用“选择证书”对话框指
定签署宏工程得到的证书

从宏工程中移除数字签名

要从宏工程中移除数字签名，可遵循以下步骤：

1. 在 Visual Basic 编辑器中，转到需要移除数字签名的文档或模板工程。
2. 在“工程资源管理器”中选择工程。
3. 选择“工具”>“数字签名”以显示“数字签名”对话框。
4. 单击“删除”按钮。“数字签名”对话框中的“VBA 工程当前签署为”区域和“签署为”区域中的证书名称都变成“无证书”，表明该工程目前没有包含任何数字证书。
5. 单击“确定”按钮关闭“数字签名”对话框。

如果又需要发布宏工程，仍可以采取前面的方法为宏工程添加数字签名。

证书的提供者及证书的含义

当得到一个由数字签名签署的项目时，会希望知道是谁签署了项目以及使用的是何种证书。要看到数字证书的具体内容，可遵循以下操作：

1. 在 Visual Basic 编辑器中，转到包含该宏的文档或模板工程。
2. 在“工程资源管理器”中选择工程。
3. 选择“工具”>“数字签名”以显示“数字签名”对话框。
4. 单击“查看证书”按钮以显示“证书”对话框（如图 19.10 所示）。

注意：如果想查看自己持有的证书，可以在“数字签名”对话框中单击“选择”按钮，在“选择证书”对话框中选择要查看的证书，然后单击“查看证书”按

单击“证书”对话框中的“确定”按钮以显示“证书”对话框。由鼠标右键单击“签名字”图标，选择“查看证书”菜单项，再单击“查看证书”按钮以显示“证书”对话框。

提示：也可以直接双击“证书”对话框中的内容阅读证书。

“证书”对话框包含3个页面：

- ◆“常规”页面显示证书的基本信息：以何种目的设立该证书，该证书颁发给谁颁发者是谁以及证书的有效期。
- ◆“详细信息”页面（如图19.11所示）包含了大约20种证书的特征。单击列表框中的一个域，下方的文本框中就会显示该项目的值。
- ◆“证书路径”页面显示了从颁发机构到目前持有者的证书路径。要检查其中的某一项，就在“证书路径”列表框中选择它，再单击“查看证书”按钮（如果该按钮可用）。然后就可以看到所要查看的“证书”的证书对话框。可以继续追踪该证书的路径，也可以单击“确定”按钮关闭第二个（或者说以后打开的）“证书”对话框，回到前一个。

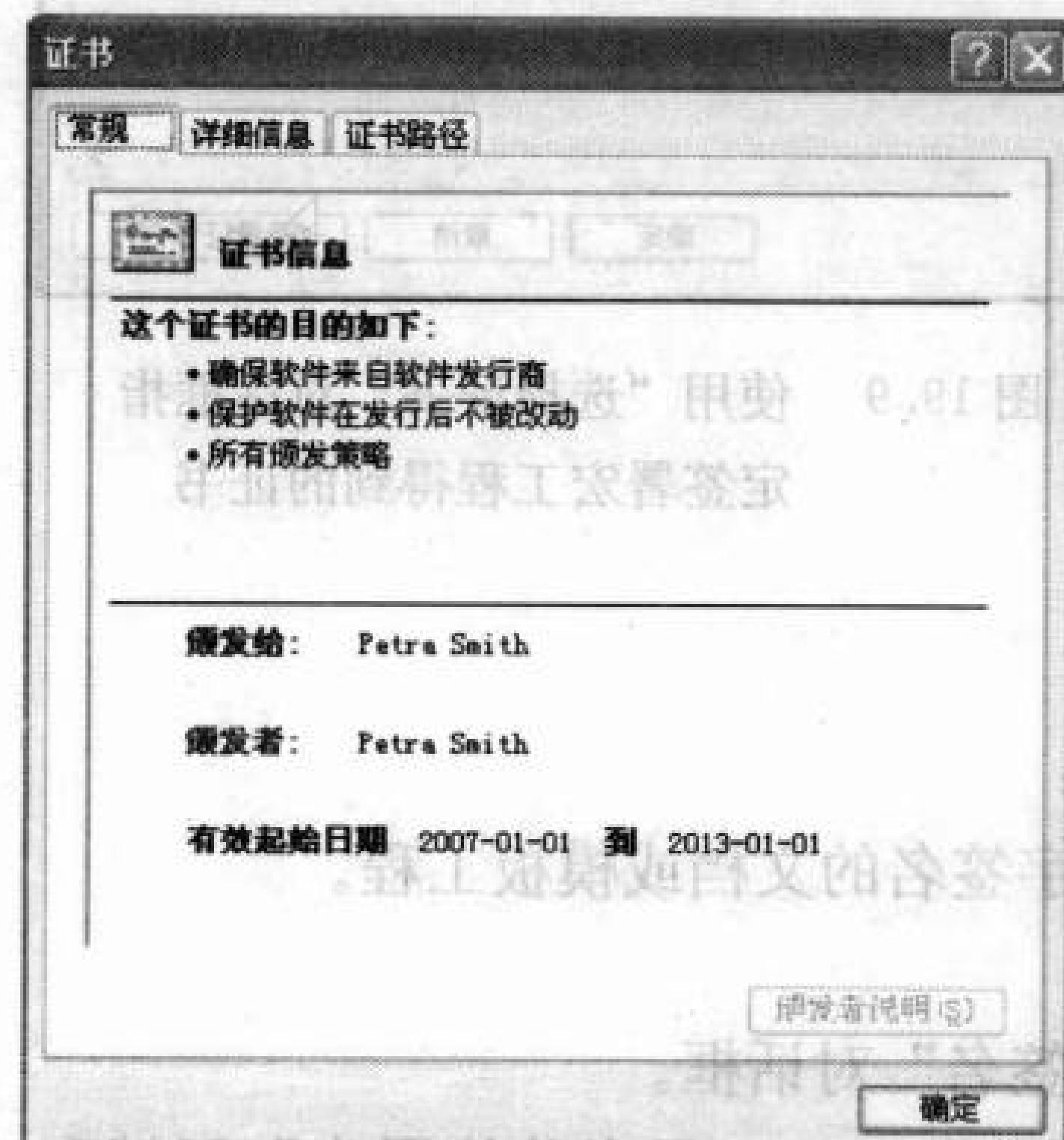


图19.10 使用“证书”对话框查看证书的属性

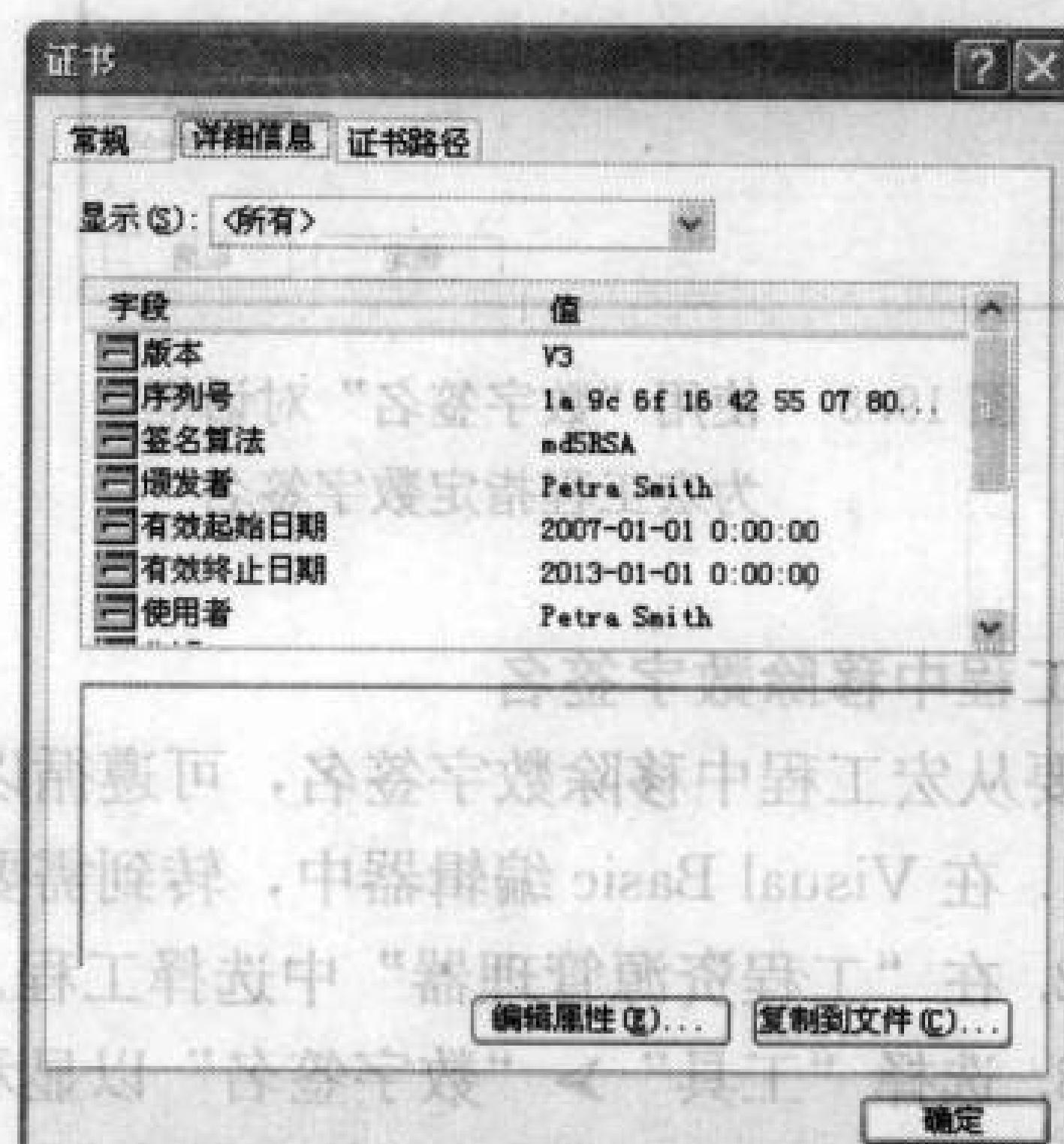


图19.11 “证书”对话框中的“详细信息”页面包含了证书的详细信息

结束浏览证书后，单击“确定”按钮，关闭“证书”对话框。

选择适当的安全级

要使VBA有效率地工作，必须选择一个适当的安全级——一方面，不能过低，必须能够抵御恶意代码和无效代码的威胁；另一方面，不能过高，不至于阻止一些需要运行的代码。

理解VBA隐含的安全威胁

随着这些年来宏语言功能的增强和完善，不当使用宏语言引起的威胁也与日俱增。使用相对简单的VBA命令，可以创建文件、删除文件、控制已有的数据以及控制其他软件。

甚至那些善意开发的程序，因运行环境不当，也可能毁坏一台计算机——例如，一个程序

可能会删除有用的数据或重要的文件，导致计算机崩溃。这样不经意间导致的毁坏经常发生，但是更加需要注意的是一些宏病毒或其他恶意软件中的恶意代码有意识地造成的毁坏。

注意：宏病毒是指用 VBA 这样的宏语言编写的计算机病毒程序。计算机病毒程序是指可以在计算机之间自我复制和自我传播的恶意代码。

防御宏病毒

要保护计算机（以及与之联网的计算机）免遭宏病毒的侵袭，需要做到以下三点：

1. 安装并运行杀毒软件，例如，诺顿（www.symantec.com）或 McAfee VirusScan（<http://www.mcafee.com>）。经常有规律地升级病毒库。（大部分的杀毒软件都提供自动升级功能。）
2. 为使用的软件设定适当的安全级，特别是那些 VBA 的宿主软件或使用其他编程语言或脚本语言的软件。例如，像后面将介绍的那样进行 VBA 的安全设定。
3. 在打开可能包含代码的软件时要特别小心。现在大部分软件在遇到可能有问题的文件时，会发出警告。许多宏病毒企图用欺骗方式躲过这样的警告，而不是使用精妙的程序语言。例如，宏病毒可以在电子邮件软件中通过附件传给联系人。上面的信息和附件暗示其中的内容十分有趣或好笑——比如说是笑话或者可疑图片。因为邮件的发件人是用户熟悉并且信任的人，而且内容看上去十分诱人，所以很多用户就会打开文件而不理会安全警告。打开文件就很可能触发其中的恶意代码。

指定适当的安全设定

首先根据自己的需要设定一个适当的安全级。选择“工具”>“宏”>“安全性”以显示“安全性”对话框（如图 19.12 所示）。

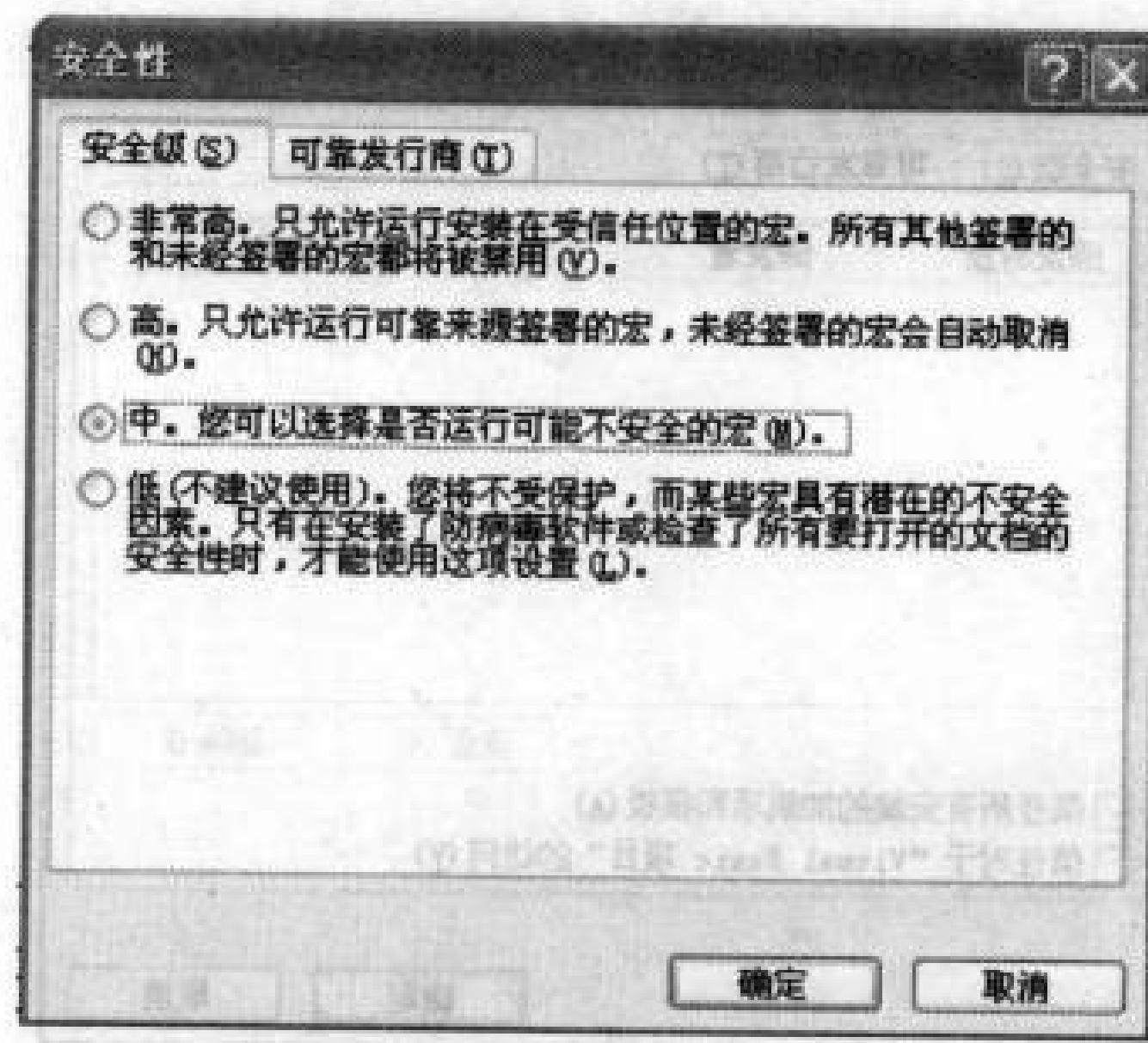


图 19.12 在“安全性”对话框的“安全级”页面上选择适当的安全级别

根据所使用的软件的不同，VBA 提供 3 到 4 个安全级别：非常高（只在某些软件中存在）、高、中和低。以下是对这些级别的解释：

非常高 只允许运行安装在受信任位置的宏（下一个部分将介绍如何指定受信任的位置）。不管宏是否被签署——软件只关心宏的安装位置。这个设定十分适合编写代码的用户，因为只要代码是安装在受信任的位置的，就不需要签署它们了。

高 只允许可靠发行商签署的宏（后文会介绍如何指定可靠的发行商）。高安全级适合

大部分的公司和家庭用户。通过将安全级设为高，签署自己发布的代码以及通过对每一台计算机上的可靠发行商的列表进行严格控制，就可以让用户了解那些程序（比如，哪些是在家里开发的）而不会碰到任何问题，同时也可以防止用户运行一些不可靠的代码。

中 允许用户自己选择是否运行那些没有被签署的程序，而不是信任所有的程序。中安全级别最适用于那些经常会使用未签署的代码的家庭用户。如果软件没有提供“非常高”这个安全级，就可以选择使用“中”安全级以获得一定的机动性。

低 不阻止任何未经签署或由非可靠发行商签署的程序运行——也不会警告可能面临的威胁。一般来说，不要在任何储存了重要信息的计算机上选择“低”安全级。除非这台计算机不与局域网（或 Internet）相连接，而且不会从别人那里复制任何文件。

指定可靠发行商

VBA 有两种方法指定可靠发行商：一种是指定安装在计算机中的模板工程和加载项中的程序来源为可靠发行商；另一种是由用户将带有代码或自定义控件的文档中所包含的来源添加为可靠发行商。

注意：Office 2003 软件使用的是“可靠发行商”的说法，而 Office XP 和 Office 2000 使用的是“可靠来源”。

计算机已经认定的可靠发行商有哪些

要查阅已经被计算机认为可靠的发行商有哪些，可以选择“工具”>“宏”>“安全性”以显示“安全性”对话框，然后单击“可靠发行商”或“可靠来源”选项卡以显示“可靠发行商”或“可靠来源”页面。图 19.13 显示的是 Excel 2003 中的“安全性”对话框的“可靠发行商”页面。

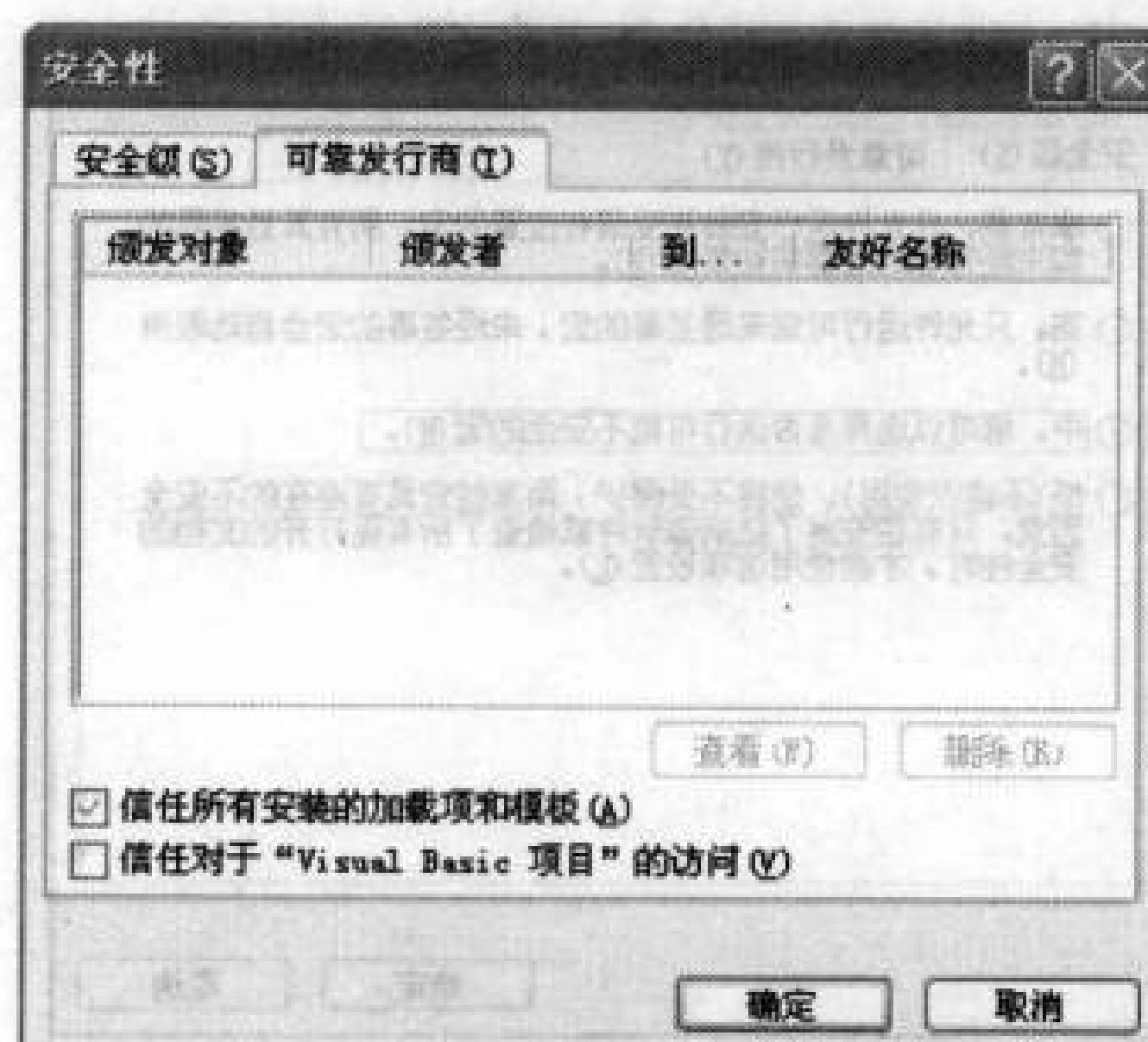


图 19.13 “安全性”对话框中的“可靠发行商”页面上列出了当前计算机认定为可靠的代码和自定义控件来源

指定已存在的发行商为可靠发行商

在 Microsoft Office 软件中，可以选择已经安装在计算机中的发行商为可靠发行商。只要勾选“安全性”对话框中的“可靠发行商”（或“可靠来源”）页面上的“信任所有安装的加载项和模板”复选框。

信任已安装的加载项和模板操作十分方便，可以了解计算机上这些已安装项目的来源。例如，在某个公司中，管理员在安装 Word 和 PowerPoint 时安装了一系列可用的模板和加载项，可以通过选择该复选框显示所有隐含的可靠项目。另一方面，对于个人用户，也可以确保他们知道 Office 安装了什么，而不是在不知情的情况下就安装了软件。

添加可靠发行商

要添加一个可靠发行商，需要打开带有来自该发行商的 VBA 代码的文档或模板。软件会探测到非可靠发行商代码，并显示“安全警告”对话框（如图 19.14 所示）。

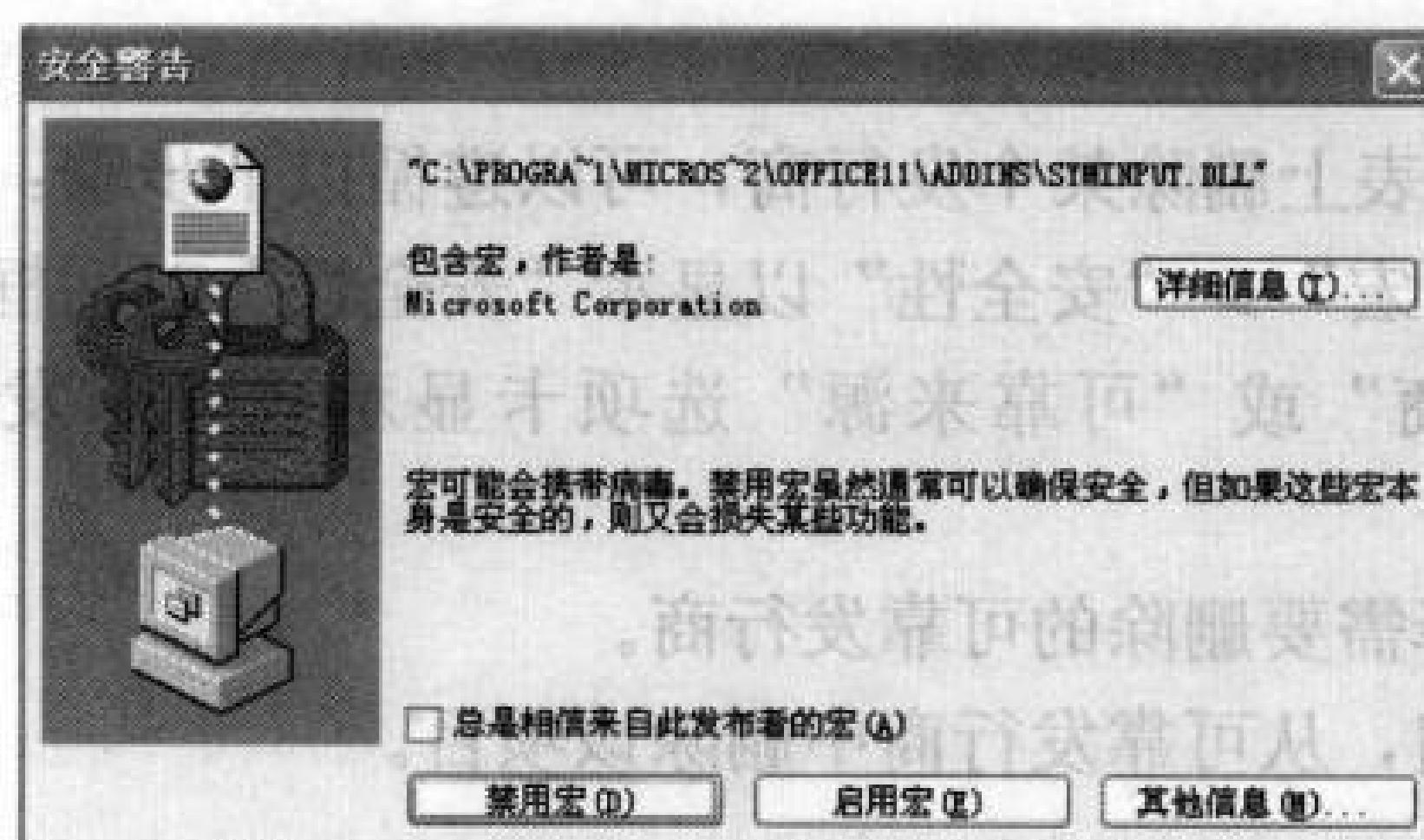


图 19.14 当打开一个文件（或加载某个加载项）时，如果其包含的代码发行商还没有被添加到可靠发行商中，软件会显示“安全警告”对话框。要把该发行商添加到可靠发行商中，勾选“总是相信来自此发布者的宏”复选框，并单击“启用宏”按钮。

“安全警告”对话框提供了以下信息：

- ◆ 带有非可靠发行商的代码的文件的名称和位置。
- ◆ 创建代码的个人或公司的名称。
- ◆ 该数字签名所使用的证书是否可靠（图中的显示为不可靠）。
- ◆ 目前的安全级别是否为高。（如果使用的是“非常高”级别，VBA 会打开文件，并自动禁用宏。如果使用的是“中”级别，“安全警告”对话框不会显示这条信息。如果使用的级别为“低”，不会出现“安全警告”对话框。）

在“安全警告”对话框中，会显示 3 到 5 个选择项：

- ◆ 可以单击“详细信息”按钮显示“证书”对话框，然后检查证书的信息，就像本章前面提到过的那样。可能，在刚刚拿到一个不太确定的文件时，不需要先进行这一步。检查证书，可以帮助从下面的两项中做出选择。
- ◆ 可以单击“禁用宏”按钮，打开文件，但禁用宏。选择禁用宏则阻止了宏的运行，代码中提供的所有功能都无法使用。
- ◆ 如果“启用宏”按钮可用，可以单击它，打开文件并使程序和自定义控件可用。如果安全级别被定为“中”，可以选择是否“总是相信来自此发布者的宏”，要相信，勾选该复选框。如果安全级别为“高”，则必须勾选该复选框才能启用宏。如果发行商不在可靠发行商内，“启用宏”按钮将不可用。
- ◆ 可以单击“安全警告”对话框的“关闭”按钮（“×”按钮）关闭对话框，但不打开文件。当无法决定或想要修改安全级时都可以这样做。例如，如果当前的安全级为“高”，