

# 第 1 章 VBA 编程概念和工具

本章将介绍编写 VBA 代码需要了解的编程基本概念和 VBA 语言元素，以及调试程序并处理错误的方法。本章内容不仅适用于 Excel，也适用于其他支持 VBA 编程的 Office 应用程序，例如 Word、PowerPoint 和 Access。

## 1.1 VBA 和宏简介

本节将简要介绍 VBA 和宏的一些基本概念，以及宏的相关操作，包括录制宏、运行宏、设置宏的安全性和修改宏。

### 1.1.1 何时需要使用 VBA

并非所有 Excel 用户都需要使用 VBA，这是因为 Excel 界面操作几乎可以完成日常所需的大多数工作，而且学习 VBA 编程也需要付出大量的时间和精力。然而，使用 VBA 可以更好地完成以下任务或者是实现这些任务的唯一途径。

- ❑ 重复执行操作：当需要执行有规律可循的重复性操作时，非常适合使用 VBA。例如，工作簿中有 3 个工作表，需要为每个工作表的数据区域中的偶数行设置蓝色背景，编写一段 VBA 代码可以快速完成这项工作。
- ❑ 增强 Excel 功能：使用 VBA 的主要原因之一是可以扩展 Excel 自身的功能。很多 Excel 无法实现或颇费周折才能完成的任务，可以通过编写 VBA 代码来解决。例如，在 Excel 中无法一次性复制不相邻的多个单元格，通过编写 VBA 可以解决这个问题。
- ❑ 简化专业数据的处理难度：对于一些需要具备专业知识才能处理的数据，可以通过编写 VBA 代码创建一个用于处理数据的特定程序，将复杂的业务逻辑包含在程序内部，只要程序的使用者为程序提供需要处理的数据，程序就会自动对数据进行处理并呈现最终结果，这样程序的使用者不需要直接面对复杂的业务逻辑，降低数据处理的难度。
- ❑ 与其他 Office 程序交换数据：当需要在 Excel 和其他 Office 程序之间交换数据时，虽然可以通过手动复制和粘贴的方式完成，但是使用 VBA 可以简化烦琐的操作过程，提高效率。
- ❑ 为大量用户统一提供 Excel 的增强功能：通过将编写好的 VBA 程序创建为加载项，可以将加载项分发给不同的用户，这些用户在自己的 Excel 程序中安装加载项，然后就可以像使用 Excel 内部功能一样，使用加载项中的 Excel 增强功能。

### 1.1.2 通过录制宏学习 VBA 编程

宏是一段可以反复运行的 VBA 代码。从技术上讲，宏是一个不包含参数的 Sub 过程。当

用户希望重复执行某个特定操作时，可以在 Excel 中将该操作录制下来，Excel 会自动生成相应的 VBA 代码。以后可以反复运行该代码来重复执行相同的操作，这就是所谓的录制宏。

在 Excel 中编写的 VBA 代码主要处理的是 Excel 中的各种对象，例如工作簿、工作表、单元格、字体、边框、背景色等。由于对象的数量众多，且它们之间具有错综复杂的关系，所以在编写 VBA 代码时经常会遇到不知道该使用哪个对象以及如何使用的情况。

用户可以从录制宏时自动生成的代码中，学习执行特定操作时所需使用的对象及其代码编写方法。虽然由 Excel 自动生成的很多代码不够简洁，甚至难以理解，但是它仍然是学习 VBA 编程的一个有用工具。

如需录制在 Excel 中执行的操作，可以单击 Excel 窗口底部状态栏中的  按钮，打开“录制宏”对话框，为即将录制的宏设置名称、运行宏的快捷键、存储位置、宏的简要说明等，如图 1-1 所示。

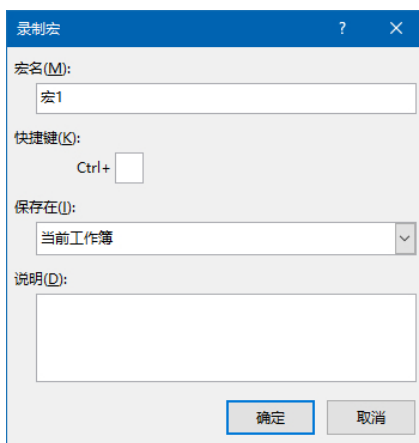


图 1-1 “录制宏”对话框

**注意：**由于录制过程中的所有误操作都会被记录下来，所以在开始录制前，应该预先演练好即将录制的一系列操作。如果录制过程中出现误操作，则在运行宏时也会执行这些误操作，严重影响宏的效率。

为宏设置快捷键时，可以在“快捷键”文本框中输入一个小写或大写的英文字母。例如，如果输入小写字母 e，则可以按 Ctrl+E 组合键运行宏；如果输入大写字母 E，则需要按 Ctrl+Shift+E 组合键运行宏。

录制的宏默认存储在当前工作簿中，可以在“保存在”下拉列表中选择宏的存储位置。

- ☐ 当前工作簿：将录制的宏存储在当前工作簿中。
- ☐ 新工作簿：将录制的宏存储在新建的工作簿中。
- ☐ 个人宏工作簿：将录制的宏存储在 Personal.xlsb 工作簿中，如果它不存在，则 Excel 会自动创建。每次启动 Excel 时会自动打开该工作簿，但是其默认处于隐藏状态。在当前打开的任何工作簿中都可以运行 Personal.xlsb 工作簿中的宏。

设置好宏的相关信息后，单击“确定”按钮，关闭“录制宏”对话框，进入录制状态，用户在工作簿中的大多数操作会被录制下来。如需结束录制，可以单击 Excel 状态栏中的  按钮。

**注意：**如果在停止录制前就运行宏，则将进入死循环，此时可以按 Ctrl+Break 组合键强制中断宏的运行。

宏的录制功能不是万能的，实际上它只能录制一些按顺序进行的简单操作。具体而言，录制的宏无法实现很多功能，包括但不限于以下这些：

- ❑ 不能创建常量和变量。
- ❑ 只能创建 Sub 过程，不能创建 Function 过程，创建的 Sub 过程不能包含参数。
- ❑ 不能生成判断条件和执行循环的代码。
- ❑ 不能生成显示对话框的代码。

### 1.1.3 运行宏

如需运行已录制好的宏，可以先打开“宏”对话框。打开该对话框有以下几种方法：

- ❑ 在功能区的“视图”选项卡中单击“查看宏”按钮，如图 1-2 所示。
- ❑ 在功能区的“开发工具”选项卡中单击“宏”按钮，如图 1-3 所示。
- ❑ 按 Alt+F8 组合键。



图 1-2 单击“查看宏”按钮



图 1-3 单击“宏”按钮

**提示：**在功能区中默认不显示“开发工具”选项卡，需要手动将其添加到功能区中。

打开“宏”对话框，选择想要运行的宏并单击“执行”按钮，即可运行该宏，如图 1-4 所示。如果为宏设置了快捷键，则可以使用快捷键运行宏，而无须打开“宏”对话框。

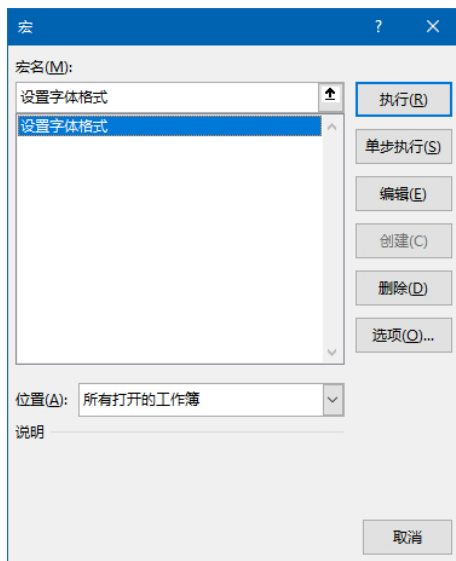


图 1-4 使用“宏”对话框运行宏

**提示：**如果无法运行宏，则可能需要更改宏的安全性设置，具体方法请参考 1.1.4 小节。

### 1.1.4 更改宏的安全性设置

当打开一个包含宏的工作簿时，可能会在功能区下方显示如图 1-5 所示的提示信息，如需运行该工作簿中的宏，可以单击“启用内容”按钮。以后每次打开这个工作簿时，都会显示该提示信息，每次都需要单击“启用内容”按钮，才能运行工作簿中的宏。

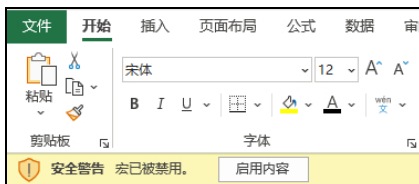


图 1-5 禁用宏时显示的提示信息

如果不理会该提示信息，而是直接使用 1.1.3 小节中的方法运行宏，则会显示如图 1-6 所示的提示信息，无法运行工作簿中的宏。

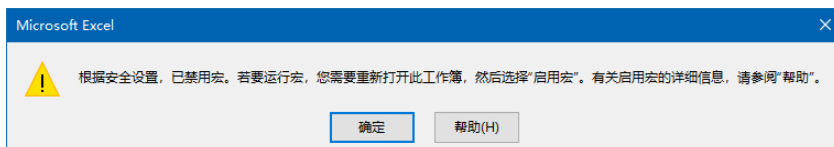


图 1-6 运行宏时显示的阻止信息

如果不想每次依靠单击“启用内容”按钮来获得运行宏的权限，则可以在 Excel 的信任中心中更改宏的安全性设置，操作步骤如下：

- (1) 启用 Excel，单击“文件”按钮，然后选择“选项”命令。
- (2) 打开“Excel 选项”对话框，在左侧选择“信任中心”选项卡，然后单击右侧的“信任中心设置”按钮，如图 1-7 所示。



图 1-7 单击“信任中心设置”按钮

- (3) 打开“信任中心”对话框，在左侧选择“宏设置”选项卡，然后在右侧选中“启用 VBA 宏（不推荐；可能运行危险代码）”单选按钮，如图 1-8 所示。



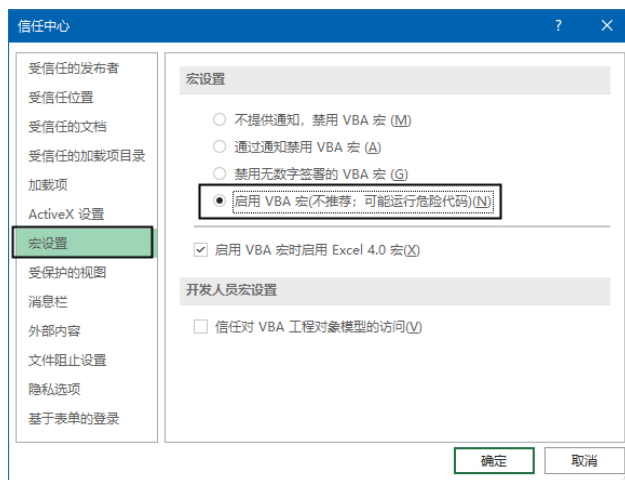


图 1-8 更改宏的安全性设置

**提示：**如果已在功能区中显示“开发工具”选项卡，则可以单击该选项卡中的“宏安全性”按钮，直接打开第（3）步中的设置界面。

（4）连续单击两次“确定”按钮，关闭之前打开的对话框。

如果既想保留 Excel 原来的安全性设置，又不想每次都通过单击“启用内容”按钮才能运行工作簿中的宏，那么可以将包含宏的一个或多个工作簿移动到同一个文件夹中，然后在 Excel 中将该文件夹标记为受信任位置，以后每次打开该文件夹中的工作簿时，Excel 都会认为其中的宏是安全的，这样就不会在功能区下方显示启用宏的提示信息，并允许用户运行这些工作簿中的宏。

如需在 Excel 中将一个文件夹标记为受信任位置，可以使用本小节前面介绍的方法，打开“信任中心”对话框，在左侧选择“受信任位置”选项卡，右侧将显示已标记为受信任位置的文件夹，如图 1-9 所示。如需添加新的文件夹，可以单击“添加新位置”按钮。

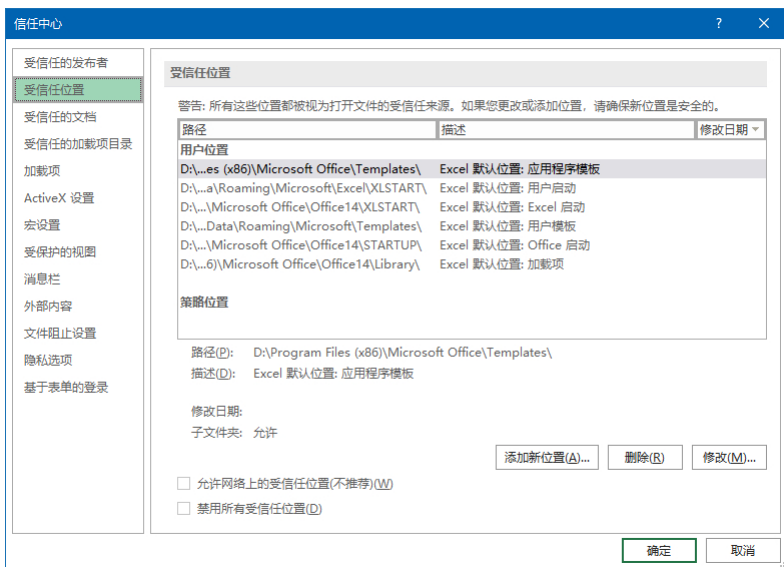


图 1-9 单击“添加新位置”按钮

打开如图 1-10 所示的对话框，单击“浏览”按钮并选择所需的文件夹，然后单击“确定”按钮，即可将选择的文件夹标记为受信任位置，如图 1-11 所示。

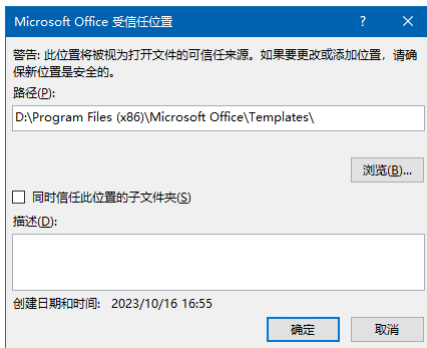


图 1-10 单击“浏览”按钮选择文件夹

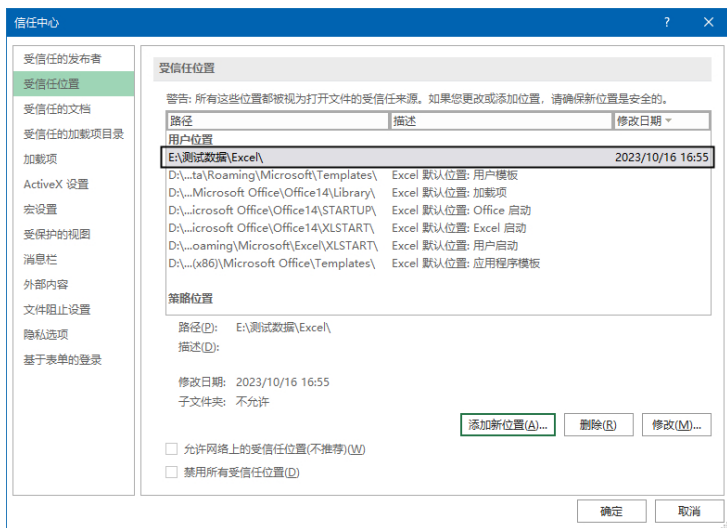


图 1-11 在受信任位置中添加新的文件夹

### 1.1.5 修改宏的相关信息和 VBA 代码

如需修改宏的相关信息，可以使用 1.1.3 小节中的方法打开“宏”对话框，选择需要修改的宏，然后单击“选项”按钮（请参考图 1-4），在打开的对话框中修改宏的快捷键和说明，如图 1-12 所示。

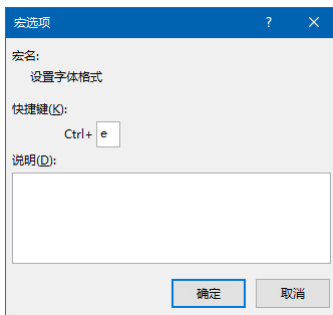


图 1-12 修改宏的相关信息

如需查看或修改宏的 VBA 代码，可以在“宏”对话框中选择一个宏，然后单击“编辑”按钮，打开如图 1-13 所示的窗口，在左侧窗格中展开“模块”并选择其中的“模块 1”，将在右侧显示录制的宏的 VBA 代码。

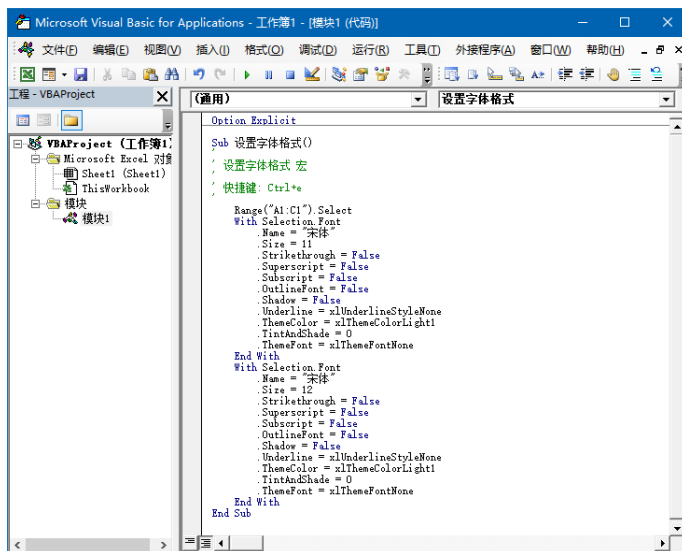


图 1-13 查看宏的 VBA 代码

本例录制宏时执行的操作是：选择 A1:C1 单元格区域，将它们的字体设置为“宋体”，将字号设置为“12”。录制的宏通常包含很多无用代码，完成本例操作只需要以下几行代码：

```
Sub 设置字体格式 ()
    Range("A1:C1").Select
    With Selection.Font
        .Name = "宋体"
        .Size = 12
    End With
End Sub
```

为了减少代码的总行数，还可以将上面的代码改为以下形式：

```
Sub 设置字体格式 ()
    Range("A1:C1").Select
    Selection.Font.Name = "宋体"
    Selection.Font.Size = 12
End Sub
```

如果希望上面的代码适用于任意选中的单元格或区域，而不局限于 A1:C1 单元格区域，则可以将 Range("A1:C1").Select 这行代码删除。

## 1.2 VBA 编程工具

与使用其他编程语言需要先单独安装集成开发环境（IDE）不同，开发 Excel VBA 程序的相关工具位于 Excel 功能区的“开发工具”选项卡中，无须额外安装。在 Excel 中编写 VBA 代码的工具是 VBE（Visual Basic Editor），本节将介绍 VBE 的界面组成及其各个部分的功能和用法。

### 1.2.1 打开 VBE 窗口

打开 VBE 窗口有以下几种方法：

- ❑ 在功能区的“开发工具”选项卡中单击 Visual Basic 按钮。
- ❑ 在“宏”对话框中选择一个宏，然后单击“编辑”按钮。
- ❑ 右击工作表标签，在弹出的快捷菜单中选择“查看代码”命令。
- ❑ 按 Alt+F11 组合键。

VBE 窗口由菜单栏、工具栏、工程资源管理器、属性窗口、代码窗口等部分组成，可以使用菜单栏的“视图”菜单控制在 VBE 窗口中显示哪些部分，并手动调整各个部分的排列方式。在如图 1-14 所示的 VBE 窗口中显示了菜单栏、工具栏、工程资源管理器、属性窗口、代码窗口、立即窗口、监视窗口。

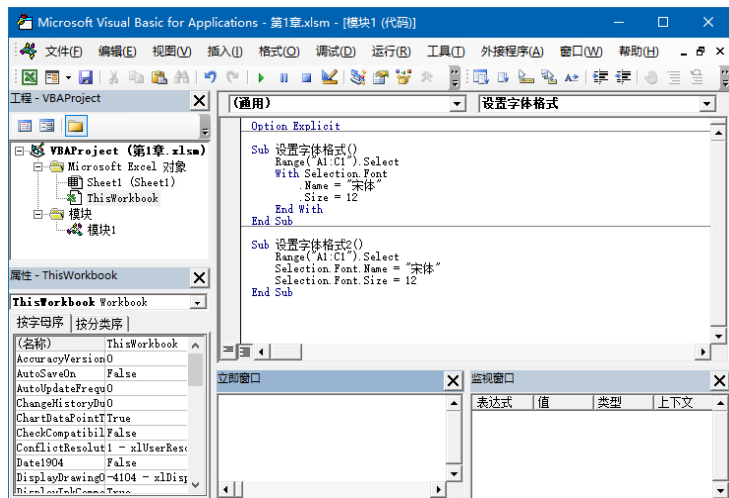


图 1-14 VBE 窗口

### 1.2.2 工程资源管理器

如图 1-15 所示，当前打开的所有工作簿都会显示在工程资源管理器中，每个工作簿都是一个 VBA 工程，工作簿的名称显示在 VBAPROJECT 右侧的小括号中。每个工作簿包含的工作表 (Sheet1、Sheet2 等)、代表工作簿自身的 ThisWorkbook 以及在 VBA 工程中添加的模块，都会以缩进的形式显示在每个 VBA 工程的下方，单击加号或减号可以展开或折叠类别中的项目。工程资源管理器对 VBA 工程的组织方式类似于 Windows 操作系统中的文件资源管理器。

在 VBA 工程中除了包含 ThisWorkbook 和 Sheet1、Sheet2 等固定模块之外，还可以创建以下 3 种模块。

- ❑ 标准模块：录制宏时自动创建的模块就是标准模块，用户也可以手动创建标准模块，在其中创建 Sub 过程或 Function 过程并输入所需的 VBA 代码。
- ❑ 窗体模块：在 VBA 工程中创建的用户窗体就是窗体模块。
- ❑ 类模块：使用类模块创建新的对象。

如需创建上述 3 种模块，可以在工程资源管理器中右击任意一项，然后在弹出的快捷菜单中选择“插入”命令，在弹出的子菜单中选择想要创建的模块，如图 1-16 所示。

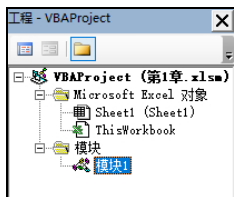


图 1-15 工程资源管理器

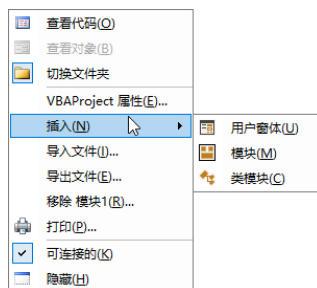


图 1-16 选择想要创建的模块

在如图 1-16 所示的快捷菜单中还可以对模块执行以下操作。

- ❑ 导出文件：使用“导出文件”命令，可以将指定模块以文件的形式保存到计算机中。
- ❑ 导入文件：使用“导入文件”命令，可以将以文件形式保存的模块导入到当前 VBA 工程中，以便重复使用其中的 VBA 代码，节省重新输入代码的时间。
- ❑ 删除模块：使用“移除 xx”命令（xx 表示模块的名称），可以删除不需要的模块。

### 1.2.3 属性窗口

在 VBE 窗口中选中的对象的所有属性的名称和值将显示在属性窗口中。如图 1-17 所示，由于在工程资源管理器中选择的是 ThisWorkbook，所以在属性窗口中将显示与 ThisWorkbook 关联的工作簿的属性。

属性窗口中左列是属性的名称，右列是属性的值。如需更改某个属性的值，可以在属性窗口中单击属性名称，然后在其右侧输入或选择属性的值，如图 1-18 所示。

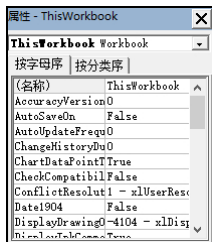


图 1-17 属性窗口

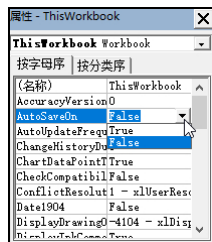


图 1-18 为属性设置预置值

### 1.2.4 代码窗口

用户编写的所有 VBA 代码都显示在代码窗口中。工程资源管理器中的每个模块都有一个与其对应的代码窗口，双击一个模块，将打开与其对应的代码窗口，如图 1-19 所示。



图 1-19 代码窗口

代码窗口的顶部有两个下拉列表，对于不同类型的模块，在左、右两个下拉列表中将显示不同的项目，右侧下拉列表中显示哪些项目取决于在左侧下拉列表中做出的选择。

例如，打开 ThisWorkbook 模块的代码窗口，在其顶部的左侧下拉列表中只有“（通用）”和“Workbook”两项。如果选择“Workbook”，则在右侧的下拉列表中会显示 Workbook 对象的所有事件过程；如果选择“（通用）”，则在右侧的下拉列表中只有“（声明）”一项。

在代码窗口中输入的代码以过程的形式进行组织。VBA 中最常用的 3 种过程分别是 Sub 过程（子过程）、Function 过程（函数过程）和事件过程。一个模块可以包含任意数量的过程，每个过程可以完成不同的任务。在编写一个复杂的 VBA 程序时，将实现不同小功能的代码组织到不同的过程中是一种良好的编程习惯。

如果一个模块包含多个过程，则可以控制是将所有过程同时显示在代码窗口中，还是每次只显示一个过程。只需单击代码窗口左下角的  或  按钮，即可在两种显示方式之间切换。当同时显示所有过程时，两个相邻过程之间使用一条横线分隔。

### 1.2.5 设置 VBE 编程选项

在 VBE 中提供了一些可以提高编程效率的选项，用户可以根据自己的编程习惯更改这些选项。实际上，大多数选项的默认设置几乎是最佳选择，用户通常无须更改这些选项的默认设置。然而，了解这些选项的含义和设置方法仍然是有用的，因为用户可以根据不同的需求和习惯随时更改这些设置。

打开 VBE 窗口，单击菜单栏中的“工具”|“选项”命令，打开“选项”对话框，如图 1-20 所示。

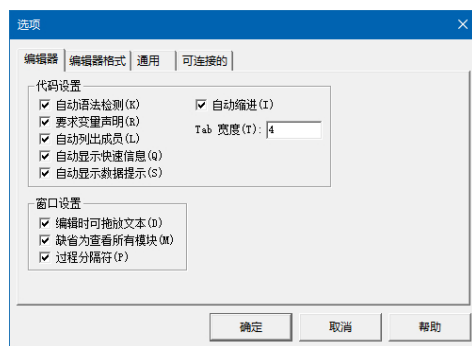


图 1-20 “选项”对话框

“选项”对话框包含 4 个选项卡，用于控制在代码窗口中输入代码的选项显示在“编辑器”选项卡中，此处主要介绍该选项卡中的选项。虽然目前可能还无法体会到这些选项的用处，但是随着对 VBA 不断深入的了解，以后会理解它们能给编程带来很多便利之处。

- ❑ 自动语法检测：启用该选项时，如果在输入 VBA 代码时出现语法错误，则会显示一个对话框，其中包含出错的原因。如果关闭该选项，则在出现语法错误时，将使用特定颜色标记出错代码而不显示对话框。
- ❑ 要求变量声明：启用该选项时，在 VBA 中使用一个变量之前必须先声明它，否则会显示出错信息。如果关闭该选项，则可以直接使用变量而无须事先声明它，但是这不是一个好的编程习惯。

- ❑ 自动列出成员：启用该选项时，在 VBA 代码中输入一个对象和英文句点后，会自动显示该对象的所有属性和方法，如图 1-21 所示。用户只需在列表中使用键盘上的方向键选择所需的属性或方法，然后按 Tab 键，即可自动将选中的属性或方法输入到代码窗口中，提高输入效率的同时也可避免出现拼写错误。
- ❑ 自动显示快速信息：启用该选项时，将自动在输入对象的属性、方法或 VBA 函数时显示参数的提示信息，如图 1-22 所示。
- ❑ 自动显示数据提示：启用该选项时，如果运行的 VBA 代码处于暂停状态，则可以将鼠标指针指向代码中的某个变量，此时会显示该变量的当前值，有利于分析和调试代码。
- ❑ 编辑时可拖放文本：启用该选项时，可以使用鼠标拖动代码来完成移动和复制操作。
- ❑ 缺省为查看所有模块：启用该选项时，在代码窗口中将同时显示当前模块中的所有过程。如果关闭该选项，则每次只显示一个过程。
- ❑ 过程分隔符：启用该选项时，将在两个过程之间显示一条分隔线。
- ❑ 自动缩进：启用该选项时，用户在输入好一行代码并按 Enter 键后，VBE 会自动将下一行代码开头的缩进量设置为与上一行相同。如果关闭该选项，则每次按 Enter 键后，下一行代码的开头不进行缩进。
- ❑ Tab 宽度：为了使程序具有良好的可读性，通常需要为不同的代码设置缩进格式。默认的缩进量是 4 个字符，用户可以根据个人习惯更改该设置。

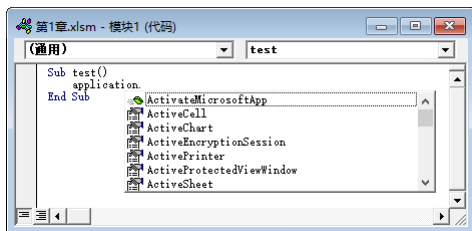


图 1-21 自动列出成员

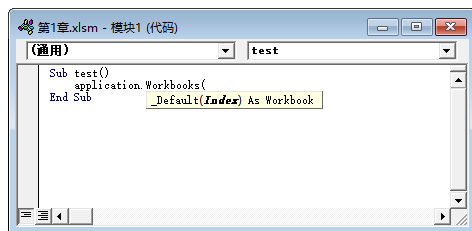


图 1-22 自动显示快速信息

## 1.3 输入和保存 VBA 代码

输入 VBA 代码之前，需要先在工程资源管理器中双击一个模块，打开与其关联的代码窗口。无论一个 VBA 程序的复杂程度如何，它都由一个或多个过程组成，每个过程完成不同的任务，组成一个 VBA 程序的所有代码都位于不同的过程中。在代码窗口中输入 VBA 代码与在 Windows 记事本中输入文本类似，可以使用常规的文本编辑操作，例如剪切、复制、粘贴、删除和撤销等。然而，在代码窗口中输入 VBA 代码有很多需要注意的问题。本节将介绍在输入 VBA 代码时增加代码可读性的常用方法，还将介绍如何保存 VBA 代码。

### 1.3.1 表达式和运算符

表达式由实际值、变量、常量、函数、运算符等多种元素组成。下面是表达式的一个示例，该表达式的含义是：将存储在 intSum 变量中的值与 10 相加，然后将计算结果赋值给该变量并替换原有值。

```
intSum = intSum + 10
```

在上面的表达式中有 4 个元素：一个变量、一个实际值，一个加号和一个等号，加号和等号都是运算符。运算符用于连接表达式中的各个元素，并决定表达式执行的运算类型和运算顺序。VBA 支持以下几种运算符。

- 连接运算符：将多个部分连接成一个整体，VBA 中的连接运算符只有&和+两个。
- 算术运算符：执行数学运算，例如加、减、乘、除等。
- 比较运算符：比较两部分内容并返回一个逻辑值 True 或 False。
- 逻辑运算符：将多个由比较运算符组成的表达式组合在一起，可以构建复杂的判断条件。

不同类型的运算符在运算时具有不同的优先级，优先级决定运算的先后顺序。在所有运算符中，算术运算符的优先级最高，其次是比较运算符，然后是逻辑运算符，连接运算符的优先级最低。

VBA 中的算术运算符、比较运算符和逻辑运算符如表 1-1 所示。表中的算术运算符和逻辑运算符包含的各个运算符按照优先级从高到低的顺序排列，比较运算符中的所有运算符具有相同的优先级。如果在一个表达式中有多个相同优先级的运算符，则这些运算符按照它们在表达式中的位置从左到右执行运算。

表 1-1 算术运算符、比较运算符和逻辑运算符

| 算术运算符 | 说 明 | 比较运算符 | 说 明   | 逻辑运算符 | 说 明  |
|-------|-----|-------|-------|-------|------|
| ^     | 求幂  | =     | 等于    | Not   | 逻辑非  |
| -     | 负号  | <>    | 不等于   | And   | 逻辑与  |
| *     | 乘   | <     | 小于    | Or    | 逻辑或  |
| /     | 除   | >     | 大于    | Xor   | 逻辑异或 |
| \     | 整除  | <=    | 小于或等于 | Eqv   | 逻辑等价 |
| Mod   | 求余  | >=    | 大于或等于 | Imp   | 逻辑蕴含 |
| +     | 加   |       |       |       |      |
| -     | 减   |       |       |       |      |

如需改变运算符的默认优先级，可以将想要优先计算的部分放在一对小括号中。下面的表达式先计算小括号中的加法，然后计算小括号外的乘法，最后的计算结果是 35。如果不使用小括号改变运算优先级，则计算结果是 31。

```
intTotal = (1 + 6) * 5
```

### 1.3.2 使用缩进格式

复杂的 VBA 代码通常都会包含大量用于处理条件判断和循环的代码，这些代码都具有固定的书写格式和要求，例如 If Then 或 Do Loop，当这些具有固定结构的代码彼此嵌套在一起时，会显著增加用户理解代码的难度。

为了增加代码的可读性，同时为了在代码出错时便于排查问题根源，应该在编写代码时使用正确的缩进格式。下面的代码包含两组嵌套在一起的 If Then 结构，由于使用了正确的缩进格式，所以可以很容易看出每组 If Then 结构的起止位置和它们各自包含的代码行。

```
Sub 验证用户名()  
    Dim strUserName As String
```



```

strUserName = InputBox("输入用户名: ")
If strUserName = "Admin" Then
    MsgBox "你是管理员"
Else
    If strUserName = "User" Then
        MsgBox "你是普通用户"
    Else
        MsgBox "你不是有效用户"
    End If
End If
End Sub

```

下面是代码不使用缩进格式时的情况，两组 If Then 结构的起止位置和各自包含哪些代码变得不太容易辨认。当代码出现错误时，这种混乱的格式将加大排查错误的难度。

```

Sub 验证用户名 ()
    Dim strUserName As String
    strUserName = InputBox("输入用户名: ")
    If strUserName = "Admin" Then
        MsgBox "你是管理员"
    Else
        If strUserName = "User" Then
            MsgBox "你是普通用户"
        Else
            MsgBox "你不是有效用户"
        End If
    End If
End Sub

```

除了使用缩进格式之外，当一个 VBA 程序包含很多行代码时，为了使代码更具可读性，可以使用空行将所有代码分隔成多个逻辑部分。

### 1.3.3 将长代码分成多行

代码窗口的宽度总是有限的，如果一行代码过长而超出代码窗口的宽度，那么需要拖动水平滚动条才能看到位于窗口外的代码，如图 1-23 所示。

如需将较长的代码完整显示在代码窗口的可视范围之内，可以将一行长代码分成多行较短的代码。首先将插入点定位到希望开始分行的位置，然后输入一个空格和一条下划线，再按 Enter 键，插入点右侧的代码将被移入到下一行，如图 1-24 所示。虽然设置分行后的两部分代码位于上下两行，但是 VBA 仍然将它们看作一行代码。

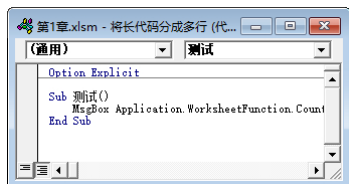


图 1-23 代码的一部分位于窗口的可见范围之外



图 1-24 将代码分成多行

### 1.3.4 为代码添加注释

为 VBA 程序中的重要代码添加注释是一个好习惯，不但可以在以后提醒自己这些代码的含

义或功能，也可以在将代码交给其他人维护时为他们理清思路。注释可以单独占据一行或位于一行代码的结尾，注释显示为绿色。为代码添加注释有以下几种方法：

- ❑ 以单引号（'）开头的内容。
- ❑ 以 **Rem** 关键字开头的内容。如果将注释放在代码行的上方，则需要在 **Rem** 关键字和注释内容之间添加一个空格；如果将注释放在代码行的右侧，则需要在 **Rem** 关键字和代码结尾之间添加一个冒号，并在 **Rem** 和注释内容之间添加一个空格。
- ❑ 选择想要转换为注释的一行或多行内容，然后单击“编辑”工具栏中的“设置注释块”按钮。如需使注释的内容恢复为可运行的 VBA 代码，可以单击“编辑”工具栏中的“解除注释块”按钮.

下面的代码使用单引号和 **Remo** 关键字两种方法添加注释。

```
'本过程的功能是在对话框中显示用户输入的用户名
Sub 设置用户名 ()
    Dim strName As String '声明一个 String 数据类型的变量
    Rem 下一行代码将用户输入的内容赋值给变量
    strName = InputBox("输入用户名：")
    MsgBox "用户名是：" & strName: Rem 显示用户名
End Sub
```

### 1.3.5 使用 InputBox 函数获取用户输入

如果希望 VBA 程序可以处理用户输入的数据，则可以使用 VBA 内置的 **InputBox** 函数。使用该函数可以显示一个对话框，其中有一个接收用户输入的文本框，**InputBox** 函数将返回用户在文本框中输入的内容。无论用户在文本框中输入什么内容，该函数的返回值都是字符串（String）数据类型。**InputBox** 函数的语法如下：

```
InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])
```

- ❑ **prompt**（必需）：显示在对话框顶部标题下方的文本。
- ❑ **title**（可选）：显示在对话框顶部的标题。
- ❑ **default**（可选）：显示在文本框中的默认值。如果不输入任何内容，则 **InputBox** 函数将返回默认值。
- ❑ **xpos、ypos**（可选）：对话框左上角在屏幕中的坐标值。
- ❑ **helpfile、context**（可选）：帮助文件和帮助主题。

**提示：**参数名称右侧带有“必需”二字表示必需为参数提供一个值，“可选”二字表示可以省略参数的值，此时将使用参数的默认值。

运行下面的代码将显示如图 1-25 所示的对话框，由于没有设置 **title** 参数，所以对话框顶部的标题默认为 Microsoft Excel。标题下方的文本由 **prompt** 参数指定。由于将 **default** 参数设置为 **admin**，所以在文本框中以选中的状态显示该默认值，用户输入新数据时会自动替换默认值。为了使后面的代码可以轻松处理用户输入的内容，可以将 **InputBox** 函数的返回值赋值给一个变量，本例的变量是 **strName**。最后使用 **MsgBox** 函数显示 **strName** 变量的值，即用户在对话框中输入的内容。

```
Sub InputBox 函数 ()
    Dim strName As String
```

```

strName = InputBox("输入用户名: ", "admin")
MsgBox "用户名是: " & strName
End Sub

```



图 1-25 由 InputBox 函数创建的对话框

如果用户没有输入任何内容而直接单击“取消”按钮，则会显示默认值 admin。如果希望单击“取消”按钮后不显示任何内容，而是直接退出程序，则可以使用 If Then 语句判断 strName 变量是否是零长度的字符串，如果是，则执行 Exit Sub 语句将直接退出当前的 Sub 过程，而不会执行后面的代码。

```

If strName = "" Then Exit Sub

```

### 1.3.6 使用 MsgBox 函数显示信息

为了在程序运行期间随时向用户发送有关程序运行状况的信息，可以使用 VBA 内置的 MsgBox 函数。使用该函数可以显示一个对话框，其中显示由用户指定的内容。MsgBox 函数的语法如下：

```

MsgBox(prompt[,buttons][,title][,helpfile,context])

```

- prompt（必需）：显示在对话框中的内容。
- buttons（可选）：显示在对话框中的按钮和图标的类型，该参数的值如表 1-2 所示。
- title（可选）：显示在对话框顶部的标题。
- helpfile、context（可选）：帮助文件和帮助主题。

表 1-2 buttons 参数的值

| 常 量                | 值   | 说 明               |
|--------------------|-----|-------------------|
| vbOKOnly           | 0   | 只显示“确定”按钮         |
| vbOKCancel         | 1   | 显示“确定”和“取消”按钮     |
| vbAbortRetryIgnore | 2   | 显示“终止”“重试”和“忽略”按钮 |
| vbYesNoCancel      | 3   | 显示“是”“否”和“取消”按钮   |
| vbYesNo            | 4   | 显示“是”和“否”按钮       |
| vbRetryCancel      | 5   | 显示“重试”和“取消”按钮     |
| vbCritical         | 16  | 显示“关键信息”图标        |
| vbQuestion         | 32  | 显示“询问信息”图标        |
| vbExclamation      | 48  | 显示“警告信息”图标        |
| vbInformation      | 64  | 显示“通知信息”图标        |
| vbDefaultButton1   | 0   | 第 1 个按钮是默认按钮      |
| vbDefaultButton2   | 256 | 第 2 个按钮是默认按钮      |
| vbDefaultButton3   | 512 | 第 3 个按钮是默认按钮      |
| vbDefaultButton4   | 768 | 第 4 个按钮是默认按钮      |

MsgBox 函数返回一个表示用户在对话框中单击了哪一个按钮的值,该函数的返回值如表 1-3 所示。

表 1-3 MsgBox 函数的返回值

| 常 量      | 值 | 说 明       |
|----------|---|-----------|
| vbOK     | 1 | 单击了“确定”按钮 |
| vbCancel | 2 | 单击了“取消”按钮 |
| vbAbort  | 3 | 单击了“终止”按钮 |
| vbRetry  | 4 | 单击了“重试”按钮 |
| vbIgnore | 5 | 单击了“忽略”按钮 |
| vbYes    | 6 | 单击了“是”按钮  |
| vbNo     | 7 | 单击了“否”按钮  |

**提示:** 在 VBA 代码中使用常量值或数字值均可。

运行下面的代码将显示如图 1-26 所示的对话框,由于只设置了 prompt 参数,所以在对话框中只显示“确定”按钮。显示对话框时会暂时中断代码的运行,单击“确定”按钮后将继续运行代码。

```
Sub MsgBox 函数 ()  
    MsgBox "已处理完成!"  
End Sub
```

运行下面的代码将显示如图 1-27 所示的对话框,由于设置了 title 参数,所以该参数的值将显示在对话框顶部的标题栏中。由于省略了位于 title 参数之前 buttons 的参数,所以需要为 buttons 参数保留一个逗号分隔符。

```
Sub MsgBox 函数 2 ()  
    MsgBox "已处理完成!", , "进度提醒"  
End Sub
```

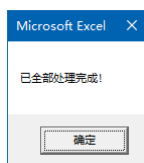


图 1-26 由 MsgBox 函数创建的对话框

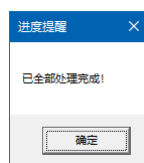


图 1-27 设置对话框的标题

如果不想输入额外的逗号分隔符,则可以使用“命名参数”的方法设置参数值。使用该方法设置参数值的格式如下:

```
参数名:=参数值
```

使用命名参数设置参数值时,如果跳过了某个参数而设置下一个参数,则无须输入额外的逗号分隔符,并且可以任意顺序设置参数值,而无须按照函数语法中的参数顺序依次设置。下面是在上一个示例中使用命名参数时的代码,将原本位于后面的 title 参数放到了第一个位置。在代码中使用命名参数的另一个优点是可使参数值的含义更清晰。

```
Sub MsgBox 函数 3 ()
```

```
MsgBox title:="进度提醒", prompt:="已处理完成!"
End Sub
```

为 MsgBox 函数中的 buttons 参数设置的值可以是表 2-1 中 3 组值的总和，3 组值分别用于设置对话框中的按钮类型、图标类型和默认按钮。例如，如需在对话框中显示“是”按钮、“否”按钮和“询问信息”图标，如图 1-28 所示，可以使用以下代码：

```
Sub MsgBox 函数 4()
    MsgBox "是否需要保存?", vbYesNo + vbQuestion
End Sub
```

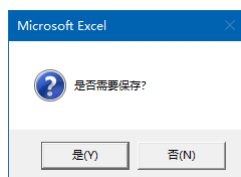


图 1-28 自定义设置对话框中的按钮和图标

当对话框中显示不止一个按钮时，可以通过 MsgBox 函数的返回值，判断用户单击的是哪一个按钮。下面的代码与上一个示例类似，唯一区别是将 MsgBox 函数的返回值赋值给一个变量，然后使用 If Then 语句检查该变量中的值是否等于 vbNo，如果是，则说明用户单击了“否”按钮，此时会直接退出程序，否则将执行下一行代码 ActiveWorkbook.Save，以便保存当前工作簿。本例代码中的 vbNo 是 VBA 内置常量，也可以在代码中使用与该常量等价的数字 7，但是使用常量会使代码更具可读性。

```
Sub MsgBox 函数 5()
    Dim lngAnswer As Long
    lngAnswer = MsgBox("是否需要保存?", vbYesNo + vbQuestion)
    If lngAnswer = vbNo Then Exit Sub
    ActiveWorkbook.Save
End Sub
```

如果显示在对话框中的内容较长，则可以在需要换行的位置插入 vbCrLf 或 vbNewLine 常量。运行下面的代码将显示如图 1-29 所示的对话框，整个内容分 3 行显示，处理方法是不断将不同的文本连接到 strMessage 变量上，最终该变量将存储所有叠加在一起的文本。

```
Sub MsgBox 函数 6()
    Dim strMessage As String
    strMessage = "是否需要保存?" & vbCrLf
    strMessage = strMessage & "保存请单击“是”按钮" & vbCrLf
    strMessage = strMessage & "不保存请单击“否”按钮"
    MsgBox strMessage, vbYesNo + vbQuestion
End Sub
```

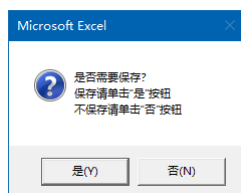


图 1-29 将内容显示为多行

### 1.3.7 保存 VBA 代码

从 Excel 2007 开始, 微软为保存 VBA 代码的工作簿提供了专门的格式, 其文件扩展名是 .xlsm。无论是在工作簿中录制的宏, 还是手动编写的 VBA 代码, 如需将其保存到工作簿中, 需要在“另存为”对话框中将“保存类型”设置为“Excel 启用宏的工作簿”, 如图 1-30 所示。

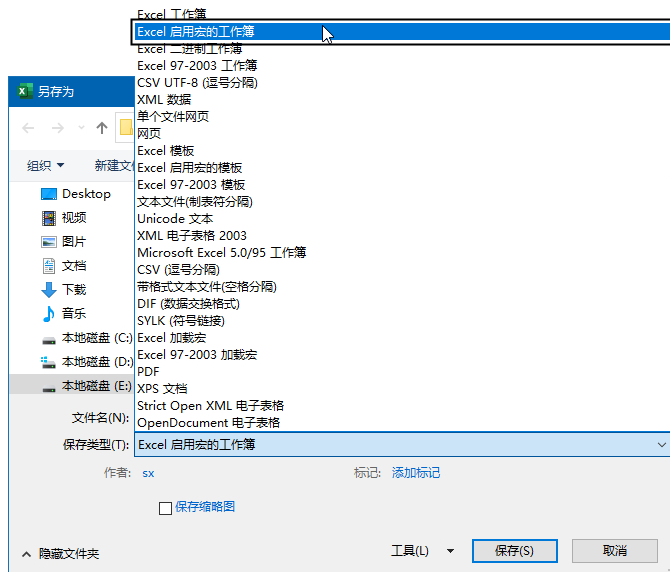


图 1-30 将“保存类型”设置为“Excel 启用宏的工作簿”

## 1.4 变量、常量和数据类型

变量和常量都是使用特定的名称来代表具体的值, 它们之间的主要区别是, 变量的值可以在程序运行期间随时修改, 而常量的值始终都是固定的, 不能随意修改。数据类型是 VBA 可以处理的数据类别, 例如文本、数值、日期、逻辑值等。本节将介绍在 VBA 程序中创建变量和常量的方法, 以及如何检测数据类型并在不同的数据类型之间转换。

### 1.4.1 VBA 支持的数据类型

VBA 支持的数据类型如表 1-4 所示, 每一种数据类型都有特定的取值范围并占用不同的内存空间。为了提高 VBA 程序的运行效率, 编写 VBA 代码时需要考虑不同数据类型之间的差异, 并在完成不同任务时使用最合适的数据类型。

表 1-4 VBA 支持的数据类型

| 数据类型     | 取值范围                                       | 占用的内存空间 |
|----------|--------------------------------------------|---------|
| Boolean  | True 或 False                               | 2 字节    |
| Byte     | 0~255                                      | 1 字节    |
| Currency | -922337203685477.5808~922337203685477.5807 | 8 字节    |
| Date     | 100 年 1 月 1 日~9999 年 12 月 31 日             | 8 字节    |

续表

| 数据类型          | 取值范围                                                                                                                                       | 占用的内存空间     |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| Integer       | -32768~32767                                                                                                                               | 2 字节        |
| Long          | -2147483648~2147483647                                                                                                                     | 4 字节        |
| Single        | 负数: -3.402823E38~-1.401298E-45<br>正数: 1.401298E-45~3.402823E38                                                                             | 4 字节        |
| Double        | 负数: -1.79769313486232E308~-4.49065645841247E-324<br>正数: 4.49065645841247E-324~1.79769313486232E308                                         | 8 字节        |
| Decimal       | 不带小数点: +/-79228162514264337593543950335<br>小数点右边有 28 位: +/-7.9228162514264337593543950335<br>最小的非零值: +/-0.00000000000000000000000000000001 | 14 字节       |
| String (定长)   | 1~65400 个字符                                                                                                                                | 字符串的长度      |
| String (变长)   | 0~20 亿个字符                                                                                                                                  | 10 字节+字符串长度 |
| Object        | 任何对象的引用                                                                                                                                    | 4 字节        |
| Variant (字符型) | 与 String (变长) 的范围相同                                                                                                                        | 22 字节+字符串长度 |
| Variant (数字型) | 与 Double 的范围相同                                                                                                                             | 16 字节       |
| 用户定义          | 各个组成部分的取值范围                                                                                                                                | 各个部分的空间总和   |

## 1.4.2 声明变量

变量是计算机内存中的存储位置,用于存储在 VBA 程序运行期间需要处理的值,变量中的值可以随时修改。每个变量都有一个名称和一种数据类型,变量的数据类型如表 1-4 所示。

编写 VBA 代码时可以直接使用变量,而无须事先声明它。然而,在使用一个变量之前先声明它,可以提高程序的运行效率,而且也是一种良好的编程习惯。为了避免忘记事先声明变量而直接在程序中使用变量,可以强制声明变量,这样在发现包含未经声明就已经使用的变量时会显示错误提示。强制声明变量有以下两种方法:

- ❑ 在模块顶部的声明部分输入 Option Explicit 语句。
- ❑ 使用 1.2.5 小节中的方法,在“选项”对话框的“编辑器”选项卡中勾选“强制变量声明”复选框。该方法会自动在新建模块的顶部添加 Option Explicit 语句,但是对现有模块无效。

声明变量可以使用 Dim 语句,下面的代码声明一个名为 UserName 的变量:

```
Dim UserName
```

如需指定变量的数据类型,可以在 Dim 语句中使用 As 子句。下面的代码声明一个字符串类型的变量,其名称是 UserName。

```
Dim UserName As String
```

**注意:** 如果不使用 As 子句指定变量的数据类型,则变量的数据类型默认为 Variant。为了节省内存空间并提高程序的运行效率,通常应该为变量指定一种数据类型。

下面的代码声明了两个变量,只有第二个变量的数据类型是 Integer,第一个变量的数据类型是 Variant。

```
Dim Row, Col As Integer
```

如果希望将上面的两个变量都声明为 Integer 数据类型,则需要在每个变量的后面加上 As 子句,代码如下:

```
Dim Row As Integer, Col As Integer
```

**提示:**除了使用 Dim 语句声明变量之外,还可以使用 Public、Private 和 Static 语句,它们的主要区别是声明的变量具有不同的作用域和生存期,具体内容请参考 1.4.5 小节。

为变量指定数据类型时,可以使用数据类型的简写形式,从而减少代码输入量。数据类型的简写形式如表 1-5 所示。

表 1-5 数据类型的简写形式

| 数 据 类 型  | 简 写 形 式 |
|----------|---------|
| String   | \$      |
| Integer  | %       |
| Long     | &       |
| Single   | !       |
| Double   | #       |
| Currency | @       |

下面的代码将 UserName 声明为 String 数据类型:

```
Dim UserName$
```

可以在声明变量时混合使用 As 语句和数据类型的简写形式。

```
Dim Row%, Col As Integer
```

### 1.4.3 变量的命名规则

声明变量时为变量起一个有意义的名称,对变量的使用有很大帮助。然而,在 VBA 中对用户创建的变量的名称有一些限制,具体如下:

- ☐ 不能将 VBA 关键字用作变量名。关键字是 VBA 中的保留字,用于标识 VBA 中的特定语言元素,声明变量的 Dim 就是 VBA 中的一个关键字。
- ☐ 变量名的首字符必须使用英文字母或汉字。
- ☐ 变量名可以包含数字和下划线,但是不能包含空格、句点、叹号等符号。
- ☐ 变量名的长度不能超过 255 个字符。

由于在 VBA 中创建的大多数变量都具有特定的数据类型,为了通过变量名就能识别其数据类型,可以在变量名的开头添加由 1~3 个字符组成的表示数据类型的前缀。建议的前缀及其对应的数据类型如表 1-6 所示。

表 1-6 标识数据类型的前缀

| 前 缀 | 数 据 类 型 | 前 缀 | 数 据 类 型 |
|-----|---------|-----|---------|
| str | String  | byt | Byte    |
| int | Integer | dat | Date    |



续表

| 前 缀 | 数 据 类 型 | 前 缀 | 数 据 类 型  |
|-----|---------|-----|----------|
| lng | Long    | cur | Currency |
| sng | Single  | dec | Decimal  |
| dbl | Double  | var | Variant  |
| bln | Boolean | udf | 用户定义     |

下面的代码将 strUserName 变量声明为 String 数据类型，使用 str 前缀标识该变量的数据类型是 String。

```
Dim strUserName As String
```

#### 1.4.4 为变量赋值

声明后的每个变量都有一个初始值，不同数据类型的变量具有不同的初始值。Integer、Long、Single、Double 等数值数据类型的变量的初始值是 0，String 数据类型的变量的初始值是零长度字符串，Boolean 数据类型的变量的初始值是逻辑值 False。

声明变量的目的是将值存储到变量中，然后在程序中使用变量代替实际值，并在需要时修改存储在变量中的值。将值存储在变量中的操作称为“为变量赋值”。如需为一个变量赋值，需要先输入变量的名称，然后在其右侧输入一个等号，再在等号的右侧输入一个值。下面的代码将“Admin”赋值给名为 UserName 的变量。

```
UserName = "Admin"
```

**注意：**如果 UserName 变量的数据类型是 Integer、Long、Single、Double 等，则会出现类型不匹配的错误，因为赋给 UserName 变量的值是一个字符串而非数值。

#### 1.4.5 变量的作用域和生存期

变量的作用域是指可以使用变量的范围。如果在一个过程中声明变量，则该变量只能在声明它的过程中使用，将这种变量称为过程级变量。下面的代码在“打开文件”过程中声明了一个名为 strFileName 的变量，该变量只能在该过程中使用，不能在“保存文件”过程中使用。

```
Sub 打开文件()
    Dim strFileName As String
    Workbooks.Open strFileName
End Sub

Sub 保存文件()
    ActiveWorkbook.Save
End Sub
```

**注意：**在同一个过程中声明的多个变量不能重名。

如果希望一个变量可以被当前模块中的所有过程使用，则需要在模块的顶部声明该变量，即在模块中的所有过程之外的位置声明变量，将这种变量称为模块级变量。下面的代码在两个过程的上方声明一个变量，此时两个过程都可以使用该变量。

```
Dim strFileName As String
```

```
Sub 打开文件 ()
    Workbooks.Open strFileName
End Sub

Sub 保存文件 ()
    ActiveWorkbook.Save
End Sub
```

声明模块级变量时，还可以使用 Private 语句，代码如下：

```
Private strFileName As String
```

**提示：**如果声明了一个模块级变量和一个过程级变量，它们的名称相同，则在包含该同名变量的过程中优先使用过程级变量，而忽略模块级变量。

如果希望声明的变量可以被 VBA 工程中的所有模块中的所有过程使用，则需要在声明模块级变量时使用 Public 语句，代码如下：

```
Public strFileName As String
```

使用 Public 语句声明的模块级变量还可以被其他 VBA 工程使用，使用前需要在其他 VBA 工程中添加对该变量所在工作簿的引用。只需单击菜单栏中的“工具”|“引用”命令，然后在“引用”对话框中勾选要引用的工作簿的 VBA 工程名的复选框，如图 1-31 所示。如果要引用的工作簿未打开，则可以单击“浏览”按钮打开该工作簿。

**提示：**如需为 VBA 工程设置一个有意义的名称，可以在工程资源管理器中右击 VBA 工程中的任意一项，在弹出的快捷菜单中选择“VBAProject 属性”命令，然后在打开的对话框中修改“工程名称”文本框中的值，如图 1-32 所示。



图 1-31 在一个 VBA 工程中引用其他 VBA 工程



图 1-32 修改 VBA 工程的名称

变量的生存期是指在变量中存储的值的保存期限。变量的作用域直接影响变量的生存期。过程级变量中的值在过程结束时自动消失，模块级变量中的值在工作簿打开期间始终存在。如果希望过程级变量具有模块级变量的生存期，则可以在过程中使用 Static 语句将变量声明为静态变量，代码如下：

```
Static lngSales As Long
```

如需将一个过程中的所有变量都变成静态变量，可以在 Sub 过程或 Function 过程的开头添加 Static 关键字。

```
Static Sub 打开文件 ()
```

### 1.4.6 使用常量

使用常量可以存储固定不变的值。在 VBA 中已经内置了很多常量，例如在 MsgBox 函数中使用的 vbYesNo。除了 VBA 内置常量之外，用户还可以创建新的常量，将一些不易记忆的数字或文本存储到常量中，以后可以使用常量代替这些内容。

在 VBA 中使用 Const 语句声明常量，为了从外观上与变量区分，可以将常量名称全部大写。下面的代码声明一个名为 PI 的常量，在其中存储圆周率的值。

```
Const PI = 3.14159265
```

与变量类似，声明常量时也可以为其指定数据类型。下面的代码声明一个 String 数据类型的常量。

```
Const USER_NAME As String = "SongXiang"
```

常量也具有与变量类似的命名规则和作用域，此处不再赘述。

### 1.4.7 检测和转换数据类型

如需检测表达式或变量的数据类型，可以使用以 Is 开头的一系列 VBA 内置函数，如表 1-7 所示，这类函数的返回值是一个逻辑值 True 或 False。

表 1-7 检测数据类型的 Is 类函数

| 函 数       | 说 明                                        |
|-----------|--------------------------------------------|
| IsNumeric | 检测表达式是否是数字，如果是则返回 True，否则返回 False          |
| IsDate    | 检测表达式是否是有效的日期，如果是则返回 True，否则返回 False       |
| IsObject  | 检测变量是否是对象变量或者包含对象引用，如果是则返回 True，否则返回 False |
| IsArray   | 检测变量是否是数组，如果是则返回 True，否则返回 False           |
| IsEmpty   | 检测变量是否已经初始化，如果未初始化则返回 True，否则返回 False      |
| IsNull    | 检测表达式是否是 Null 值，如果是则返回 True，否则返回 False     |
| IsMissing | 检测参数是否已经传递给过程，如果未传递则返回 True，否则返回 False     |
| IsError   | 检测表达式是否是错误值，如果是则返回 True，否则返回 False         |

下面的代码声明一个 String 数据类型的变量，然后将一个数字以字符串的形式赋值给该变量，最后使用 IsNumeric 函数检测该变量的值是否是数字，结果返回 True，如图 1-33 所示。

```
Sub 检测数据类型 ()
    Dim Num As String
    Num = "168"
    MsgBox IsNumeric (Num)
End Sub
```



图 1-33 IsNumeric 函数检测结果

使用 VBA 内置的 `TypeName` 函数可以返回一个表示数据类型的文本,当无法确定变量中存储的值是什么数据类型时可以使用该函数。将上面示例中的代码修改为以下形式,运行结果如图 1-34 所示,证实 `Num` 变量中的值是字符串。

```
Sub 检测数据类型()  
    Dim Num As String  
    Num = "168"  
    MsgBox TypeName(Num)  
End Sub
```

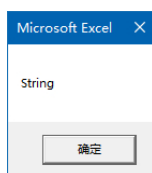


图 1-34 使用 `TypeName` 函数返回表示数据类型的文本

如需转换表达式的数据类型,可以使用以 `C` 开头的一系列 VBA 内置函数,例如 `CBool`、`CByte`、`CStr`、`CInt`、`CLng`、`CSng`、`Cdbl`、`CDate`、`CCur`、`CVar` 等,通过函数名称的拼写可以很容易了解函数将表达式转换成哪种数据类型。

下面的代码与上一个示例类似,唯一区别是使用 `Cint` 函数将 `Num` 变量的数据类型转换为 `Integer`,然后使用 `TypeName` 函数证实数据类型是否转换成功。

```
Sub 转换数据类型()  
    Dim Num As String  
    Num = "168"  
    MsgBox TypeName(CInt(Num))  
End Sub
```

**注意:** 如果转换前的值超出目标数据类型支持的范围,则会出现错误。

## 1.5 创建和调用 Sub 过程

无论一个 VBA 程序包含多少行代码,所有代码都会被组织到一个或多个过程中,这些过程存储在不同的模块中。使用过程可以将复杂的代码分解成多个更小的逻辑单元,以便于代码的编写和调试。在 VBA 中最常使用的是 `Sub` 过程和 `Function` 过程,前者也称为子过程,后者也称为函数过程。在 Excel 中录制的宏是 `Sub` 过程,用户可以在 VBE 中创建 `Sub` 过程,从而实现比录制的宏更灵活、更强大的功能。本节将介绍创建和调用 `Sub` 过程的方法。

### 1.5.1 创建 Sub 过程

一个 `Sub` 过程由 `Sub` 语句开始, `End Sub` 语句结束,在这两个语句之间编写实现特定功能的 VBA 代码。`Sub` 过程的语法如下:

```
[Private | Public] [Static] Sub name [(arglist)]  
    [statements]  
[Exit Sub]  
[statements]  
End Sub
```

- ☐ Private（可选）：声明一个模块级的 Sub 过程，该过程只能被其所在模块中的过程调用，不能被其他模块中的过程调用。
- ☐ Public（可选）：声明一个工程级的 Sub 过程，任何 VBA 工程中的任何过程都可以调用该过程。如果在模块顶部添加 Option Private Module 语句，则会强制将 Sub 过程变成模块级过程。
- ☐ Static（可选）：将 Sub 过程中的所有变量指定为静态变量。
- ☐ Sub（必需）：标志 Sub 过程的开始。
- ☐ name（必需）：Sub 过程的名称，与变量的命名规则相同。
- ☐ arglist（可选）：为 Sub 过程提供的一个或多个参数，参数之间使用逗号分隔。参数是 Sub 过程需要处理的数据，实际上参数就是变量，只不过在将变量传递给过程时，将变量称为参数，所以参数也具有与变量相同的数据类型。如果不提供参数，则必须保留一对空括号。
- ☐ statements（可选）：实现特定功能的 VBA 代码。
- ☐ Exit Sub（可选）：退出 Sub 过程。
- ☐ End Sub（必需）：标志 Sub 过程的结束。

可以手动输入 Sub 过程，也可以使用“添加过程”对话框中的选项创建 Sub 过程。如需手动输入 Sub 过程，可以先打开一个模块的代码窗口，然后在其中输入 Sub 和一个名称，按 Enter 键后，会自动在过程名称右侧添加一对小括号以及 End Sub 语句。

```
Sub 测试()
```

```
End Sub
```

现在可以在 Sub 和 End Sub 之间输入所需的 VBA 代码了。

如需使用“添加过程”对话框创建 Sub 过程，可以单击菜单栏中的“插入”|“过程”命令，打开“添加过程”对话框，在“名称”文本框中输入 Sub 过程的名称，然后在“类型”中选择“子程序”，还可以在“范围”中选择创建的 Sub 过程是模块级还是工程级的，最后单击“确定”按钮，如图 1-35 所示。

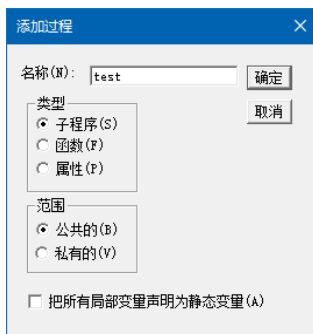


图 1-35 使用“添加过程”对话框创建 Sub 过程

**提示：**用户创建的不包含参数的 Sub 过程会与录制的宏同时显示在“宏”对话框中。在“宏”对话框中不显示用户创建的 Sub 过程有两种方法，一种是在创建过程时使用 Private 关键字，将过程创建为模块级的；另一种是在过程名右侧的小括号中添加至少一个参数。

## 1.5.2 调用 Sub 过程

将一个 VBA 程序实现的所有功能分解到多个 Sub 过程中，每个 Sub 过程只实现一种特定的功能，这种离散的组织方式为代码的编写和测试提供了很大的灵活性。在所有 Sub 过程中，总有一些 Sub 过程实现的是可能被反复使用的功能，此时无须在每个需要这种功能的地方重复编写代码，而可以直接调用实现该功能的 Sub 过程。

调用 Sub 过程最简单的方法是直接输入过程名。如果过程包含参数，则需要在过程名的右侧输入所需的参数值。如果有多个参数，则各个参数之间使用逗号分隔。下面的代码调用名为“OpenFile”的 Sub 过程，并为其提供 FileName 和 ReadOnly 两个参数的值。

```
Sub Main()  
    OpenFile "第1章.xlsm", False  
End Sub  
  
Sub OpenFile(FileName As String, ReadOnly As Boolean)  
  
End Sub
```

调用 Sub 过程的另一种方法是使用 Call 语句。首先输入 Call，然后输入过程名。如果过程包含参数，则需要将所需的参数放在过程名右侧的一对小括号中，各个参数之间使用逗号分隔。下面的代码使用 Call 语句调用上一个示例中的 OpenFile 过程。

```
Call OpenFile("第1章.xlsm", False)
```

如果在 VBA 工程中存在多个同名的 Sub 过程，则在调用它们时，需要在过程名的开头添加过程所在的模块名和一个英文句点，格式如下：

```
模块名.过程名
```

如需调用其他 VBA 工程中的 Sub 过程，可以先在 VBE 的“引用”对话框中添加对该工程的引用，然后使用以下格式的代码调用该工程中的 Sub 过程。

```
工程名.模块名.过程名
```

## 1.5.3 按地址或按值传递参数

创建 Sub 过程时，可以在过程名右侧的小括号中添加所需的一个或多个参数，并为这些参数指定合适的数据类型。传递参数有“按地址”和“按值”两种方式，创建 Sub 过程时可以指定参数的传递方式，在参数名的左侧使用 ByRef 关键字表示按地址传递，使用 ByVal 关键字表示按值传递，省略这两个关键字时默认按地址传递。下面的 Sub 过程包含一个按值传递的参数。

```
Sub OpenFile(ByVal FileName As String)  
  
End Sub
```

按地址传递参数时，如果过程改变参数的值，则这种改变会直接影响变量本身。按值传递参数时，如果过程改变参数的值，则对变量本身没有任何影响。

下面的代码创建一个名为 MySum 的过程，该过程有一个按地址传递的参数，该过程执行的操作是对参数加 1。

```
Sub MySum(ByRef Num As Integer)
```

```
Num = Num + 1
End Sub
```

下面的代码在“测试”过程中声明一个名为 intNum 的变量，然后调用 MySum 过程，并将该变量传递给该过程，最后显示该变量在 MySum 过程中执行加 1 计算后返回“测试”过程时的值。由于 MySum 过程的 Num 参数是按地址传递的，所以在 MySum 过程内部执行加 1 计算的结果会影响 intNum 变量本身，当 intNum 变量返回“测试”过程后其值是 1，如图 1-36 所示。

```
Sub 测试()
    Dim intNum As Integer
    MySum intNum
    MsgBox intNum
End Sub
```

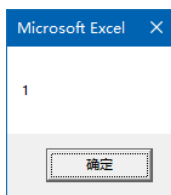


图 1-36 按地址传递参数的结果

如果使用 ByVal 关键字将 MySum 过程中的 Num 参数指定为按值传递，则在“测试”过程中 intNum 变量的值最后会显示为 0。这是因为在 MySum 过程中无论对参数执行什么计算，按值传递时的计算结果都不会影响传递给 MySum 过程的变量本身。

#### 1.5.4 Sub 过程的递归

递归是指在一个 Sub 过程中调用该 Sub 过程，即一个过程调用其自身。任何一个过程都可以递归，关键在于需要在满足条件时退出递归，否则会进入无限循环。

运行下面的代码将显示一个对话框，在其中输入一个名称，系统会检测输入的是否是“admin”。如果不是，则显示提示信息并通过调用该过程重新显示对话框，要求用户重新输入用户名。如果输入的是“admin”，则显示“登录成功”的提示信息并退出程序。

```
Sub 用户登录()
    Dim strUserName As String
    strUserName = InputBox("输入用户名: ")
    If strUserName = "admin" Then
        MsgBox "登录成功!"
        Exit Sub
    Else
        MsgBox "用户名不正确，请重新输入!"
        Call 用户登录
    End If
End Sub
```

**注意：**只有输入与 admin 大小写完全匹配的英文字母，才会退出程序。如果希望输入任意大小写形式都能与 admin 匹配，则可以将 If 语句中的判断条件改为以下形式，使用 VBA 内置的 LCase 函数将用户输入的内容全部转换为小写字母。

```
If LCase(strUserName) = "admin" Then
```

也可以使用 UCase 函数将用户输入的内容全部转换为大写字母，此时需要将等号右侧的内容改为全部大写的 ADMIN。

## 1.6 创建和调用 Function 过程

在本章前面介绍 InputBox 函数时，该函数会返回一个值，该值表示用户在对话框中输入的内容。MsgBox 函数也有一个返回值，表示用户在对话框中单击了哪一个按钮。如果用户想要创建带有返回值的函数，则需要创建 Function 过程。是否带有返回值是 Function 过程和 Sub 过程的主要区别，该区别导致 Function 过程在语法和使用等方面与 Sub 过程存在一些差异。本节将介绍创建和调用 Function 过程的方法，还将介绍在 VBA 中使用 VBA 内置函数和 Excel 工作表函数的方法。

### 1.6.1 创建 Function 过程

Function 过程的语法与 Sub 过程类似，主要区别在于对返回值的处理。一个 Function 过程由 Function 语句开始，End Function 语句结束，在这两个语句之间编写实现特定功能的 VBA 代码。Function 过程的语法如下：

```
[Public | Private] [Static] Function name [(arglist)] [As type]
[statements]
[name = expression]
[Exit Function]
[statements]
[name = expression]
End Function
```

- ❑ **Public**（可选）：声明一个工程级的 Function 过程，任何 VBA 工程中的任何过程都可以调用该过程。如果在模块顶部添加 Option Private Module 语句，则会强制将 Function 过程变成模块级过程。
- ❑ **Private**（可选）：声明一个模块级的 Function 过程，该过程只能被其所在模块中的过程调用，不能被其他模块中的过程调用。
- ❑ **Static**（可选）：将 Function 过程中的所有变量指定为静态变量。
- ❑ **Function**（必需）：标志 Function 过程的开始。
- ❑ **name**（必需）：Function 过程的名称，与变量的命名规则相同。
- ❑ **arglist**（可选）：为 Function 过程提供的一个或多个参数，参数之间使用逗号分隔。参数是 Function 过程需要处理的数据。如果不提供参数，则必须保留一对空括号。
- ❑ **type**（可选）：为 Function 过程的返回值指定数据类型，与变量的数据类型相同。
- ❑ **statements**（可选）：实现特定功能的 VBA 代码。
- ❑ **expression**（可选）：Function 过程的返回值。如果希望 Function 过程可以返回一个值，则需要在 Function 过程结束前将其返回值赋值给 Function 过程的名称。
- ❑ **Exit Function**（可选）：退出 Function 过程。
- ❑ **End Function**（必需）：标志 Function 过程的结束。

创建 Function 过程的步骤与 Sub 过程类似，可以在代码窗口中手动输入 Function 和 End



Function，也可以使用“添加过程”对话框自动输入这两个语句，此时需要在该对话框的“类型”中选择“函数”选项，其他选项与 Sub 过程相同。

下面的代码创建一个计算两个数字之和的 Function 过程，该过程有两个参数，它们表示参与计算的两个数字，将该过程的返回值指定为 Single 数据类型，因为参与计算的数字可能是小数。

```
Function MySums (Num1, Num2) As Single
    MySums = Num1 + Num2
End Function
```

## 1.6.2 调用 Function 过程

调用 Function 过程的方法与调用 Sub 过程类似，包括 Function 过程的作用域规则和调用时的书写格式等方面。用户创建的 Function 过程的使用方法与 VBA 内置函数相同，可以使用连接运算符将 Function 过程与其他表达式组合在一起，也可以将 Function 过程的返回值赋给一个变量，然后使用该变量表示 Function 过程的返回值，以简化代码的输入量。

假设在 1.6.1 小节中创建的 MySums 过程位于名为“创建 Function 过程”的模块中，下面的代码在位于另一个模块的“测试”过程中调用 MySums 过程，并将其与一个字符串组合在一起，以便将它们显示在由 MsgBox 函数创建的对话框中，如图 1-37 所示。

```
Sub 调用 Function 过程 ()
    MsgBox "两个数字的总和是：" & 创建 Function 过程.MySums (2, 6)
End Sub
```

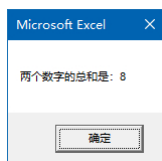


图 1-37 调用 Function 过程

如需在代码中处理 Function 过程的返回值并减少代码的输入量，可以将 Function 过程的返回值赋给一个变量，然后在代码中使用该变量表示 Function 过程的返回值。下面的代码将 MySums 过程的返回值赋给 sngSum 变量，然后在 If 语句中判断该变量的大小并显示指定的信息。

```
Sub 调用 Function 过程 2 ()
    Dim sngSum As Single
    sngSum = 创建 Function 过程.MySums (2, 6)
    If sngSum < 10 Then MsgBox "给定的两个数字太小了！"
End Sub
```

如果创建 Function 过程时，在 Function 的开头添加 Public 关键字或者省略该关键字，则创建的 Function 过程可以在工作表中使用，就像使用 Excel 内置的工作表函数一样。仍以前面创建的 MySums 过程为例，在工作表的 A1 和 B1 两个单元格中分别输入一个数字，然后在 C1 单元格中输入以下公式并按 Enter 键，将在 C1 单元格中显示两个数字之和，如图 1-38 所示。

```
=MySums (A1,B1)
```

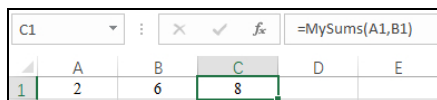


图 1-38 在工作表中使用用户创建的 Function 过程

如果不想在工作表中使用由用户创建的 Function 过程，则可以在创建 Function 过程时添加 Private 关键字，将其创建为模块级的 Function 过程。

### 1.6.3 使用 VBA 内置函数

用户创建的 Function 过程通常用于完成特定的计算或操作。实际上，VBA 已经内置了大量的 Function 过程，它们都是 VBA 内置的函数，这些函数可以执行很多数学和日期方面的计算，还可以处理字符串、文件和文件夹等。

如果想要使用某个 VBA 内置函数但是不知道它的英文拼写，则可以先输入 VBA 和一个英文句点，然后在弹出的列表中选择所需的函数，如图 1-39 所示。输入函数后，可以将开头的 VBA 和英文句点删除。

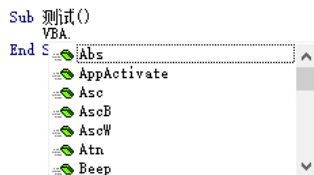


图 1-39 在列表中选择 VBA 内置函数

当用户创建的 Function 过程与 VBA 内置的函数同名时，如果直接输入 Function 过程的名称，则使用的是用户创建的 Function 过程。如需使用同名的 VBA 内置函数，需要在 Function 过程的开头添加 VBA 和一个英文句点。

### 1.6.4 在 VBA 中使用 Excel 工作表函数

在 VBA 代码中除了可以使用 VBA 内置函数和用户创建的 Function 过程之外，还可以使用 Excel 工作表函数。下面的代码就是使用 Excel 工作表函数统计 A1:A10 单元格区域中非空单元格的个数。

```
Application.WorksheetFunction.CountA(Range("A1:A10"))
```

Application 对象的 WorksheetFunction 属性返回 WorksheetFunction 对象，该方法由 Excel 工作表函数组成。在 VBA 代码中输入 Application.WorksheetFunction.后，也会弹出一个列表，可以从中选择所需的工作表函数。由于 WorksheetFunction 是全局成员，所以在代码中可以省略对 Application 对象的引用，上面的代码可以改写成以下形式：

```
WorksheetFunction.CountA(Range("A1:A10"))
```

需要注意的是，如果一个 Excel 工作表函数与某个 VBA 内置函数的功能相同，则在 VBA 中无法使用该工作表函数。例如，工作表函数 UPPER 的功能是将英文字母转换为大写，它与 VBA 内置的 UCase 函数的功能相同，则在 VBA 中不能使用 UPPER 函数。

## 1.7 有选择地执行代码

在 Excel 中录制的宏只能按照用户的操作顺序逐行执行代码，这意味着录制的宏无法灵活处理可能出现的各种情况。在 VBA 中可以使用 If Then Else 和 Select Case 两个语句为程序加入条件判断机制，根据判断结果有选择地执行所需的代码。本节将介绍这两个语句的使用方法。

### 1.7.1 使用 If Then Else 语句根据条件选择要执行的代码

使用 If Then Else 语句可以根据条件是否成立执行一行或多行代码，该语句有单行或多行两种形式。

#### 1. 单行 If Then Else 语句

如果只有一个条件且只执行少量的操作，则可以使用单行 If Then Else 语句，格式如下：

```
If 条件 Then 条件成立时执行的代码 Else 条件不成立时执行的代码
```

下面的代码检测用户输入的数字是否大于 10，如果是，则显示“符合要求”，否则显示“数字太小”。

```
Sub 单行 If 语句()  
    Dim intNum As Integer  
    intNum = Val(InputBox("输入一个整数："))  
    If intNum > 10 Then MsgBox "符合要求" Else MsgBox "数字太小"  
End Sub
```

如果只在条件成立时才执行操作，则可以省略 Else 子句。下面的代码只在数字小于 10 时显示“不符合要求”，其他情况不显示任何信息。

```
Sub 单行 If 语句 2()  
    Dim intNum As Integer  
    intNum = Val(InputBox("输入一个整数："))  
    If intNum < 10 Then MsgBox "不符合要求"  
End Sub
```

#### 2. 多行 If Then Else 语句

很多复杂问题通常需要在条件成立或不成立时执行多个操作，此时可以使用多行 If Then Else 语句，格式如下：

```
If 条件 Then  
    条件成立时执行的一行或多行代码  
Else  
    条件不成立时执行的一行或多行代码  
End If
```

下面的代码是本章前面的一个示例，用于检查用户输入的用户名，如果符合要求，则显示欢迎信息并退出程序，否则显示错误信息并要求用户重新输入用户名。无论条件是否成立，都需要分别执行两行代码。

```
Sub 多行 If 语句()  
    Dim strUserName As String  
    strUserName = InputBox("输入用户名：")  
    If strUserName = "admin" Then
```

```

        MsgBox "登录成功！"
        Exit Sub
    Else
        MsgBox "用户名不正确，请重新输入！"
        Call 用户登录
    End If
End Sub

```

**提示：**与单行 If Then Else 语句类似，如果条件不成立时不需要执行任何操作，则可以省略 Else 子句。单行 If Then Else 语句在条件成立和不成立时也可以执行多行代码，只需将多行代码写在同一行，并在各句代码之间使用英文冒号进行分隔。

处理复杂条件时，可以在多行 If Then Else 语句的内部嵌套另一个或多个 If Then Else 语句，每一个 If Then Else 语句中的 If 和 End If 必须成对出现，格式如下：

```

If 第 1 个条件 Then
    第 1 个条件成立时执行的代码
Else
    If 第 2 个条件 Then
        第 2 个条件成立时执行的代码
    Else
        If 第 n 个条件 Then
            第 n 个条件成立时执行的代码
        Else
            第 n 个条件不成立时执行的代码
        End If
        第 2 个条件不成立时执行的代码
    End If
    第 1 个条件不成立时执行的代码
End If

```

下面是一个嵌套 If Then Else 语句的示例，它也是本章前面的一个示例。

```

Sub 多行 If 语句 2()
    Dim strUserName As String
    strUserName = InputBox("输入用户名：")
    If strUserName = "Admin" Then
        MsgBox "你是管理员"
    Else
        If strUserName = "User" Then
            MsgBox "你是普通用户"
        Else
            MsgBox "你不是有效用户"
        End If
    End If
End Sub

```

如需在多个类似的条件中检查是否符合其中某个条件并执行相应的代码，则可以使用下面的格式：

```

If 第 1 个条件 Then
    第 1 个条件成立时执行的代码
ElseIf 第 2 个条件 Then
    第 1 个条件不成立但第 2 个条件成立时执行的代码
ElseIf 第 n 个条件 Then
    前 n-1 个条件都不成立但第 n 个条件成立时执行的代码

```

```
Else
    前面所有条件都不成立时执行的代码
End If
```

这种结构非常适合检测多个值中是否存在与指定值相匹配的值，并执行预先为各个值编写好的代码。下面的代码检测用户输入的数字，如果大于 100，则显示“数字太大”；如果小于 100，则显示“数字太小”，否则显示“数字有效”。

```
Sub 多行 If 语句 3 ()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字: "))
    If intNum > 100 Then
        MsgBox "数字太大"
    ElseIf intNum < 10 Then
        MsgBox "数字太小"
    Else
        MsgBox "数字有效"
    End If
End Sub
```

实际上，实现上述需求更适合使用 Select Case 语句，将在 1.7.2 小节中介绍。

## 1.7.2 使用 Select Case 语句根据表达式的值执行符合条件的代码

使用 Select Case 语句可以根据表达式的值从多个条件中检测是否存在符合的条件，并执行与第一个符合的条件关联的代码。Select Case 语句的格式如下：

```
Select Case 表达式
    Case 与表达式进行比较的第 1 个值
        满足第 1 个值时执行的代码
    Case 与表达式进行比较的第 2 个值
        满足第 2 个值时执行的代码
    Case 与表达式进行比较的第 n 个值
        满足第 n 个值时执行的代码
    Case Else
        不满足前面所有值时执行的代码
End Select
```

下面是使用 Select Case 语句重新编写 1.7.1 小节最后一个示例之后的代码，代码看起来更清晰。在 Case 子句中使用 Is 关键字表示 intNum 变量的值，将其与特定的值进行比较来构建判断条件。

```
Sub SelectCase 语句 ()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字: "))
    Select Case intNum
        Case Is > 100
            MsgBox "数字太大"
        Case Is < 10
            MsgBox "数字太小":
        Case Else
            MsgBox "数字有效"
    End Select
End Sub
```

如需减少代码的行数，可以将每个 Case 子句中的两行代码写在一行上：

```
Sub SelectCase 语句 ()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字："))
    Select Case intNum
        Case Is > 100: MsgBox "数字太大"
        Case Is < 10: MsgBox "数字太小":
        Case Else: MsgBox "数字有效"
    End Select
End Sub
```

**提示：**上面的示例使用 Is 关键字指定范围的一端，还可以使用 To 关键字指定范围的两端，例如 Case 10 To 20。

如果需要比较的不是范围而是固定的值，则可以将固定的值写在 Case 子句中。下面的代码根据用户输入的内容显示不同的欢迎信息。

```
Sub SelectCase 语句 2 ()
    Select Case LCase(InputBox("输入用户名："))
        Case "admin"
            MsgBox "你好，管理员"
        Case "user"
            MsgBox "你好，普通用户"
        Case Else
            MsgBox "无效用户"
    End Select
End Sub
```

可以在一个 Case 子句中指定需要比较的多个值，在各个值之间使用英文逗号分隔。只要表达式与其中任意一个值匹配，就执行该 Case 子句中的代码。下面的代码检测当前 Excel 程序的版本，在第一个 Case 子句中指定了多个值，如果表达式的值与其中任意一个值匹配，则显示“Excel 2003 之后的版本”，否则显示“Excel 2003 或更早版本”的提示信息。

```
Sub SelectCase 语句 3 ()
    Select Case Application.Version
        Case "16.0", "15.0", "14.0", "12.0"
            MsgBox "Excel 2003 之后的版本"
        Case Else
            MsgBox "Excel 2003 或更早版本"
    End Select
End Sub
```

与嵌套的 If Then Else 语句类似，Select Case 语句也可以互相嵌套，还可以将 If Then Else 和 Select Case 两个语句嵌套在一起，从而构建可以处理复杂条件的代码。

## 1.8 重复执行代码

编程解决的主要问题之一是可以自动处理需要重复执行的操作，最大限度地提高效率并减少人为失误。在 VBA 中可以使用 For Next 和 Do Loop 两个语句处理需要重复执行的操作，本节将介绍它们的使用方法。

### 1.8.1 使用 For Next 语句重复执行代码指定的次数

如果事先知道代码需要重复执行的次数，则可以使用 For Next 语句，该语句的语法如下：

```
For counter = start To end [Step step]
[statements]
[Exit For]
[statements]
Next [counter]
```

- ❑ counter（必需）：计数器，其值在循环期间将不断递增或递减。
- ❑ start（必需）：计数器的起始值。
- ❑ end（必需）：计数器的终止值。
- ❑ Step step（可选）：计数器的步长，即计数器每次递增或递减的增量。以大写字母 S 开头的 Step 是 VBA 中的关键字，以小写字母 s 开头的 step 在编写代码时需要换成实际值。如果步长是 1，则可以写成 Step 1。由于 1 是步长的默认值，所以可以省略 Step 1。
- ❑ statements（可选）：实现特定功能的 VBA 代码。
- ❑ Exit For（可选）：退出 For Next 循环。

下面的代码计算自然数 1~10 的所有数字之和。由于两个相邻自然数的差值是 1，所以本例中的 For Next 语句的步长是 1，辨析代码时可将其省略。

```
Sub ForNext 语句 1()
    Dim intCounter As Integer, intSum As Integer
    For intCounter = 1 To 10
        intSum = intSum + intCounter
    Next intCounter
    MsgBox "数字的总和是：" & intSum
End Sub
```

如需计算 1~10 中的所有偶数之和，可以将步长值设置为 2，并将计数器的起始值设置为 0，代码如下：

```
Sub ForNext 语句 2()
    Dim intCounter As Integer, intSum As Integer
    For intCounter = 0 To 10 Step 2
        intSum = intSum + intCounter
    Next intCounter
    MsgBox "所有偶数的总和是：" & intSum
End Sub
```

如果希望求和的数字范围完全由用户指定，则可以使用 InputBox 函数接收用户输入的数字，然后将其返回值指定为计数器的起始值和终止值。为了避免在输入不能转换为数字的字符串时程序出错，需要使用 VBA 内置的 IsNumeric 函数检测用户两次输入的数字是否都是数字，如果都是数字，才会使用 For Next 语句对数字求和。使用 And 运算符连接两个 IsNumeric 函数可以实现同时满足是数字的条件。

```
Sub ForNext 语句 3()
    Dim intCounter As Integer, intSum As Integer
    Dim intStart As String, intEnd As String
    intStart = InputBox("输入数字范围的下限")
    intEnd = InputBox("输入数字范围的上限")
    If IsNumeric(intStart) And IsNumeric(intEnd) Then
```

```
For intCounter = intStart To intEnd
    intSum = intSum + intCounter
Next intCounter
MsgBox "指定范围内的所有数字的总和是：" & intSum
End If
End Sub
```

### 1.8.2 使用 Do Loop 语句在满足条件时重复执行代码

如果事先无法确定代码需要重复执行的次数，但是知道在什么情况下开始或结束重复执行代码，此时应使用 Do Loop 语句。在 Do Loop 语句中使用 While 或 Until 设置开始或结束循环的条件，共有 4 种形式。

#### 1. 先检测条件是否成立，如果成立则开始循环

```
Do While 条件
    条件成立时执行的代码
Loop
```

下面的代码说明了这种 Do Loop 形式的工作方式，在开始 Do Loop 循环之前，先检测用户输入的数字是否小于 10，如果小于 10，则执行 Do Loop 中的代码，每循环一次 intNum 变量都加 1，直到 intNum 变量的值等于 10 为止。如果最初输入的数字大于或等于 10，则跳过 Do Loop 语句，直接在对话框中显示该数字。

```
Sub DoLoop 语句 ()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字："))
    Do While intNum < 10
        intNum = intNum + 1
    Loop
    MsgBox intNum
End Sub
```

#### 2. 循环一次后检测条件是否成立，如果成立则继续循环

```
Do
    条件成立时执行的代码
Loop While 条件
```

下面的代码说明了这种 Do Loop 形式的工作方式，修改了上一个示例中的代码，此时无论用户输入的是什么数字，都会先进入 Do Loop 循环对该数字加 1，然后判断数字是否小于 10，如果小于 10，则继续对该数字加 1，直到数字等于 10 为止，否则不再执行 Do Loop 中的代码，而直接在对话框中显示 intNum 变量的当前值。

```
Sub DoLoop 语句 2 ()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字："))
    Do
        intNum = intNum + 1
    Loop While intNum < 10
    MsgBox intNum
End Sub
```

当在对话框中输入的数字大于或等于 10 时，本例最后在对话框中显示的数字会比上一个示例的数字大 1，这是因为在本例中无论输入多大的数字，都会先执行加 1 的计算。



### 3. 先检测条件是否成立，如果成立则结束循环

```
Do Until 条件
    条件不成立时执行的代码
Loop
```

下面的代码说明了这种 Do Loop 形式的工作方式，本例代码的运行结果与使用第一种 Do Loop 形式的示例相同。在代码的编写方式上有两个细微的区别，将 While 改成 Until，然后将条件改成相反的条件，即将“小于 10”改成“大于或等于 10”。

```
Sub DoLoop 语句 3()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字: "))
    Do Until intNum >= 10
        intNum = intNum + 1
    Loop
    MsgBox intNum
End Sub
```

### 4. 循环一次后检测条件是否成立，如果成立则结束循环

```
Do
    条件不成立时执行的代码
Loop Until 条件
```

下面的代码说明了这种 Do Loop 形式的工作方式，本例与使用第二种 Do Loop 形式的示例在代码的编写方式上也有与上一个示例类似的两个区别。

```
Sub DoLoop 语句 4()
    Dim intNum As Integer
    intNum = Val(InputBox("输入一个数字: "))
    Do
        intNum = intNum + 1
    Loop Until intNum >= 10
    MsgBox intNum
End Sub
```

无论使用以上哪种形式的 Do Loop 语句，都可以使用 Exit Do 语句立刻结束循环并执行 Loop 语句后面的代码，这给在 Do Loop 语句中加入额外的判断条件提供了机会。

下面的代码在 Do Loop 语句中首先执行显示接收用户输入的对话框的代码，然后使用 If Then 语句判断是否在未输入内容的情况下关闭了对话框，如果是，则使用 Exit Do 语句立刻结束 Do Loop 循环；如果不是，则检测输入的用户名是否是 admin。如果是 admin，则在对话框中显示“欢迎登录系统”，否则重新显示接收用户输入的对话框，用户需要再次输入用户名并重复之前的检测。

```
Sub DoLoop 语句 5()
    Dim strUserName As String
    Do
        strUserName = InputBox("输入用户名: ")
        If strUserName = "" Then Exit Do
        Loop While LCase(strUserName) <> "admin"
        If strUserName <> "" Then MsgBox "欢迎登录系统"
    End Sub
```

## 1.9 对象编程

在 Excel 中编写的 VBA 代码几乎都是在处理各种对象。Excel 中的对象代表 Excel 应用程序中的各种元素，工作簿、工作表、单元格、字体、图片、形状、图表等都是对象，Excel 应用程序本身也是一个对象。编程处理 Excel 中的对象不但可以使界面环境中的手动操作变成自动执行，还可以完成很多在界面环境中无法实现的功能。本节将介绍在 VBA 中进行对象编程的基本概念和一般方法，编程处理 Excel 中的具体对象的方法将在后续章节中详细介绍。

### 1.9.1 Excel 对象模型

为了通过编程完整控制 Excel 应用程序，微软提供了一个与 Excel 应用程序中的各个元素完全对应的 Excel 对象模型，该模型包含可以编程处理的所有 Excel 对象以及对象之间的关系。Excel 对象模型就像一张 Excel 应用程序中所有对象的关系脉络图。

Application 对象在 Excel 对象模型中是最顶层的对象，它代表 Excel 应用程序，该对象是其他所有对象的发源地。在 Excel 应用程序中创建的每一个工作簿都是一个 Workbook 对象，在工作簿中创建的每一个工作表都是一个 Worksheet 对象，工作表中的任意单元格或单元格区域都是 Range 对象。

上面简单介绍的几个对象的层次结构可以表示为以下形式，它们的级别从左到右逐层下降。这几个对象是 Excel 对象模型中最重要、使用率最高的对象，通过它们可以延伸出其他对象，直到蔓延至整个对象模型。

Application ⇨ Workbook ⇨ Worksheet ⇨ Range

使用 VBE 中的对象浏览器可以查看 Excel 对象模型中的所有对象。打开对象浏览器有以下几种方法：

- ❑ 单击菜单栏中的“视图”|“对象浏览器”命令。
- ❑ 单击“标准”工具栏中的“对象浏览器”按钮.
- ❑ 按 F2 键。

打开的对象浏览器如图 1-40 所示。如要查看 Excel 对象模型中的所有对象，可以在顶部的下拉列表中选择 Excel，在“类”列表框中会显示所有 Excel 对象。选择一个 Excel 对象，右侧会显示该对象的属性、方法和事件， 图标表示对象的属性、 图标表示对象的方法， 图标表示对象的事件。

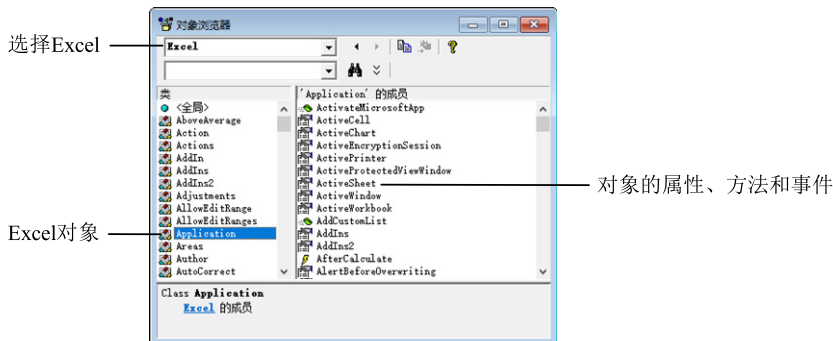


图 1-40 对象浏览器

**提示：**严格地说，在“类”列表框中显示的是每个对象的类。为了易于理解“类”的概念，可以将类看作对象的模板，每个对象都是通过类创建的，与从工作簿模板创建工作簿的方式类似。由类创建的对象称为类的实例。每一个类具有哪些属性、方法和事件是事先被定义好的，使用类创建多个对象后，可以为这些对象的同一个属性设置不同的值，使每个对象具有自己的特征，也可以使用同一种方法操作对象，但是为方法提供不同的参数，从而得到不同的操作结果。

## 1.9.2 引用集合中的对象

当同一种对象的数量不止一个时，这些对象就构成了集合。例如，在 Excel 中打开的所有工作簿构成了工作簿集合，在 Excel 对象模型中表示为 **Workbooks**。一个工作簿中的所有工作表构成了工作表集合，在 Excel 对象模型中表示为 **Worksheets**。

如需引用集合中的某个对象，可以使用两种方法。如果集合中的所有对象都有一个唯一的名称，则可以使用名称引用对象。如果当前工作簿包含名为 2021、2022 和 2023 的 3 个工作表，则下面的代码引用名为“2023”的工作表。

```
Worksheets("2023")
```

如果名为“2023”的工作表是第 3 个工作表，则还可以使用索引号引用该工作表。

```
Worksheets(3)
```

**注意：**如需一直引用某个工作表，则应该使用名称而非索引号来引用这个工作表，因为索引号会根据工作表位置的变动而改变。

如果要引用的工作表所属的工作簿不是当前工作簿，则需要在代码中添加对特定工作簿的引用。

```
Workbooks("总公司").Worksheets("2023")
```

## 1.9.3 使用对象变量引用对象

在 VBA 支持的数据类型中有一个 **Object** 类型，它表示一般对象类型。当无法确定对象的具体类型时，可以将变量声明为 **Object** 数据类型。编程处理 Excel 对象时，通常不会使用 **Object** 数据类型，而是将变量声明为 Excel 中的特定对象类型。将声明为一般或特定对象类型的变量称为对象变量。下面的代码将名为 **wks** 的变量声明为 **Worksheet** 对象类型，以后可以在代码中使用该对象变量引用一个工作表。

```
Dim wks As Worksheet
```

将一个变量声明为特定的对象类型后，还需要使用 **Set** 语句将特定的对象赋值给该变量，然后才能在代码中使用该变量引用特定的对象。下面的代码将当前工作簿中名为“2023”的工作表赋值给 **wks** 变量，以后可以使用 **wks** 变量引用该工作表。

```
Set wks = Worksheets("2023")
```

使用对象变量引用对象不仅可以减少代码的输入量，还可以提高程序的运行效率。当不再使用某个对象变量时，可以使用 **Set** 语句将 **Nothing** 赋值给该对象变量，清除在对象变量中引用的对象，使对象变量恢复到赋值前的状态。

```
Set wks = Nothing
```

在实际编程中，为了避免使用一个没有引用实际对象的对象变量而导致程序出错，可以使用 Is 关键字配合 Nothing 来检测一个对象变量是否正在引用一个对象。下面的代码检测 wks 对象变量是否是 Nothing，如果是，说明该对象变量没有引用任何特定的对象，此时将退出 Sub 过程。

```
If wks Is Nothing Then Exit Sub
```

如需在对象变量不是 Nothing 时执行操作，可以在上面的代码中添加 Not 关键字。

```
If Not wks Is Nothing Then Exit Sub
```

### 1.9.4 对象的属性

属性是对象具有的特征。例如，Worksheet 对象有一个 Name 属性，该属性表示工作表的名称。如需修改工作表的名称，可以为 Name 属性设置一个值，就像为变量赋值一样。下面的代码将名为“2023”的工作表的名称设置为“2023 年”。

```
Worksheets("2023").Name = "2023 年"
```

**提示：**如果当前工作簿中没有名为“2023”的工作表，则运行代码时会出错。

除了可以设置属性的值之外，还可以使用属性的值。下面的代码显示当前工作簿中第 1 个工作表的名称。

```
MsgBox Worksheets(1).Name
```

如需在代码中多次使用属性的值，可以将它赋值给一个变量，然后使用变量代表属性的值。下面的代码将当前工作簿中第 1 个工作表的名称存储在 strSheetName 变量中。

```
strSheetName = Worksheets(1).Name
```

对象的很多属性返回的是一个值，例如上面代码中的 Worksheet 对象的 Name 属性。然而，对象的某些属性返回的也可能是另一个对象。下面的代码将 A1 单元格的字体设置为“黑体”。

```
Range("A1").Font.Name = "黑体"
```

Range("A1")是一个 Range 对象，Font 是该对象的属性，它表示单元格的字体格式（字体、字号、颜色等）。为 Range 对象使用 Font 属性后，返回的是一个 Font 对象。Font 后面的 Name 是 Font 对象的属性，表示字体的名称。上面的代码中位于等号左侧的 3 个部分连在一起，它们的运作方式是：先使用 Range 对象的 Font 属性返回一个 Font 对象，然后使用 Font 对象的 Name 属性设置字体名称。编写 VBA 代码处理 Excel 应用程序时经常会遇到这种情况。

### 1.9.5 对象的方法

方法是对对象可以执行的操作。Workbook 对象有一个 Close 方法，该方法用于关闭工作簿。下面的代码是关闭名为“总公司”的工作簿。

```
Workbooks("总公司").Close
```

**提示：**如果当前没有打开名为“总公司”的工作簿，则运行代码时会出错。

如果工作簿中存在未保存的内容，则运行上面的代码会显示一个对话框，询问用户是否保存工作簿，选择后才会关闭工作簿。如果在关闭工作簿时不想被这个对话框打扰，无论工作簿

中是否存在未保存的内容，都自动保存并关闭工作簿，则可以使用下面的代码，将 `Close` 方法的 `SaveChanges` 参数设置为 `True`。

```
Workbooks("总公司").Close True
```

如需不保存修改而直接关闭工作簿，可以将 `True` 改为 `False`。如果希望代码的含义更清晰，则可以使用命名参数，与本章前面介绍 `MsgBox` 函数时使用命名参数的方法相同。下面的代码使用命名参数的方式指定 `SaveChanges` 参数的值，在参数名及其值之间添加一个冒号和一个等号。

```
Workbooks("总公司").Close SaveChanges:=True
```

与属性类似，对象的某些方法也可以返回另一个对象。例如，`Worksheets` 对象的 `Add` 方法将新建一个工作表，并返回一个 `Worksheet` 对象，该对象表示这个新建的工作表。为了在后面的代码中使用由 `Add` 方法新建的工作表，可以将该方法返回的 `Worksheet` 对象赋值给一个对象变量。

下面的代码是使用 `Worksheets` 对象的 `Add` 方法在当前工作簿中新建一个工作表，并将其返回的 `Worksheet` 对象赋值给 `wks` 对象变量，现在 `wks` 对象变量表示这个新建的工作表。然后为 `wks` 对象变量设置 `Name` 属性以修改新工作表的名称。

```
Sub 对象的方法()  
    Dim wks As Worksheet  
    Set wks = Worksheets.Add  
    wks.Name = "2024"  
End Sub
```

### 1.9.6 父对象和子对象

Excel 对象模型中的各个对象构成了复杂的层次结构，它就像一条线路图，可以在编写代码时为找到特定的对象提供方向。在 Excel 对象模型中，`Application` 对象处于最顶层，比其他所有对象的级别都高。

`Workbook` 对象位于 `Application` 对象的下一层，`Worksheet` 对象位于 `Workbook` 对象的下一层。在相邻的上下两层对象中，将上一层对象称为父对象，将下一层对象称为子对象。所以 `Application` 对象是 `Workbook` 对象的父对象，`Workbook` 对象是 `Application` 对象的子对象，而 `Workbook` 对象又是 `Worksheet` 对象的父对象。很多 Excel 对象同时充当父对象和子对象的角色，就像此处的 `Workbook` 对象。

如需从一个对象返回其子对象，只需使用可以返回特定对象的属性，正如 1.9.4 小节中介绍的。如需从一个对象返回其父对象，可以使用对象的 `Parent` 属性。下面的代码使用 `Worksheet` 对象的 `Parent` 属性，返回 `Worksheet` 对象的父对象 `Workbook` 对象，然后使用 `Workbook` 对象的 `Name` 属性返回特定的工作表所属工作簿的名称。

```
Worksheets("2023").Parent.Name
```

如需返回特定的工作表所属工作簿所在的 Excel 应用程序的版本号，可以使用两次 `Parent` 属性。第一个 `Parent` 属性返回 `Worksheet` 对象的父对象——`Workbook` 对象，第二个 `Parent` 属性是 `Workbook` 对象的属性，返回的是 `Workbook` 对象的父对象——`Application` 对象。

```
Worksheets("2023").Parent.Parent.Version
```

如果起始对象的层次级别很低,则可能需要使用很多个 Parent 属性才能返回 Application 对象。实际上,无论对象处于哪个层次级别,都可以使用 Application 属性直接返回 Application 对象,代码如下:

```
Worksheets("2023").Application.Version
```

### 1.9.7 使用 With 语句提高处理同一个对象的效率

在 VBA 程序中对一个对象或对象变量的每一次引用都会消耗一定的内存。下面的代码为 A1 单元格设置多种字体格式,其中对 A1 单元格的 Font 属性引用了 4 次。

```
Sub 设置多个属性()  
    Range("A1").Font.Name = "宋体"  
    Range("A1").Font.Size = 12  
    Range("A1").Font.Bold = True  
    Range("A1").Font.Color = vbRed  
End Sub
```

一种更好的方法是使用 With 语句在最开始只引用一次 A1 单元格的 Font 属性,之后可以为 A1 单元格的 Font 属性返回的 Font 对象设置一系列的属性和方法,但是必须在属性和方法的开头保留英文句点,整个 With 语句以 With 开始,End With 结束。使用 With 语句既可以提高代码效率,还可以减少代码的输入量。

```
Sub 使用 With 语句设置多个属性()  
    With Range("A1").Font  
        .Name = "宋体"  
        .Size = 12  
        .Bold = True  
        .Color = vbRed  
    End With  
End Sub
```

### 1.9.8 使用 For Each 语句处理集合中的对象

可能经常需要处理一个集合中的每一个对象,尤其是处理集合中满足条件的对象,此时可以使用 For Each 语句,该语句的语法如下:

```
For Each element In group  
    [statements]  
    [Exit For]  
    [statements]  
Next [element]
```

- ❑ element (必需): 循环访问集合中每一个对象的对象变量。
- ❑ group (必需): 对象集合。
- ❑ statements (可选): 处理对象的 VBA 代码。
- ❑ Exit For (可选): 退出 For Each 语句。

For Each 语句与 For Next 语句类似,区别是 For Each 语句不需要事先知道循环的次数。无论集合中有多少个对象,For Each 语句都会依次处理每一个对象,直到处理完所有对象。如果在 For Each 语句中加入 If Then Else 语句,则可以只处理满足条件的对象,或者在满足特定条件

时，使用 Exit For 语句提前退出循环。

下面的代码逐个查看当前工作簿中每个工作表的名称，如果发现名为“2023”的工作表，则激活该工作表并退出循环。

```
Sub ForEach 语句 ()
    Dim wks As Worksheet
    For Each wks In Worksheets
        If wks.Name = "2023" Then
            wks.Activate
            Exit For
        End If
    Next wks
End Sub
```

## 1.10 调试程序并处理错误

无论代码编写的有多么仔细，都无法避免程序出现错误。VBE 提供了一些有用的调试工具，可以在程序出错时帮助用户尽快找到出错原因。本节将介绍常用调试工具的使用方法，还将介绍如何编写错误处理代码来处理运行时错误。

### 1.10.1 错误类型

VBA 中的错误有 3 种类型：编译错误、运行时错误和逻辑错误。

- ❑ 编译错误：在编写代码的过程中出现不符合 VBA 语法规则的代码时，会立刻被检测出来，并要求用户更正错误的代码，否则禁止运行代码。这意味着在运行代码前，必须完全解决所有现存的编译错误。
- ❑ 运行时错误：运行代码期间执行无效的操作时，会立刻中断代码的运行，并使用特定颜色标记出错的代码。解决错误后，才能继续运行代码。
- ❑ 逻辑错误：逻辑错误具有很强的隐蔽性，因为它在 VBA 语法规则和程序运行两个方面都不会出现问题，也永远不会显示任何错误信息，但是程序的运行结果与预期存在偏差或完全不同。

如需设置出错代码的标记颜色，可以在 VBE 窗口中单击菜单栏中的“工具”|“选项”命令，打开“选项”对话框，在“编辑器格式”选项卡中设置，如图 1-41 所示。

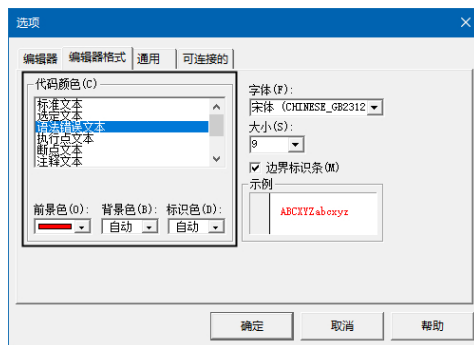


图 1-41 设置出错代码的标记颜色

## 1.10.2 运行代码的几种方式

VBE 提供了几种运行程序的方式，它们适用于不同的需求。

### 1. 运行整个程序

如果是第一次运行一个编写好的 VBA 程序，则通常会以常规方式运行该程序。将插入点定位到想要运行的过程的代码范围内，然后按 F5 键或单击“标准”工具栏中的  按钮，开始运行程序。如果运行过程中出现运行时错误，则会显示一个包含错误信息的对话框，如图 1-42 所示。

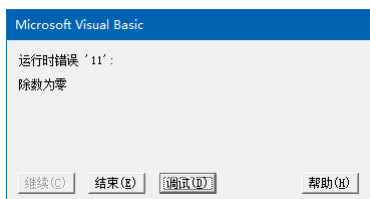


图 1-42 显示运行时错误信息的对话框

单击对话框中的“调试”按钮，将进入中断模式，此时程序暂停运行，并突出显示出错代码，如图 1-43 所示。

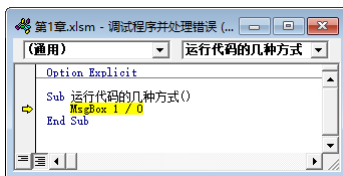


图 1-43 在中断模式下修复错误的代码

更正错误后，按 F5 键从出错代码的位置继续运行程序。如需从头开始运行程序，可以单击“标准”或“调试”工具栏中的“重新设置”按钮，然后按 F5 键。

### 2. 逐行运行代码

如需了解程序中每一行代码的运行情况，可以使用 F8 键逐行运行代码。每按一次 F8 键，将从程序起始位置开始每次向下运行一行代码并进入中断模式。在中断模式下，将鼠标指针移动到变量或表达式上，将显示它们的当前值，便于检查程序是否正按照预期运行。

### 3. 运行到指定位置

如果希望让程序直接运行到可能有问题的代码行之前的位置，以便于逐步排查错误，并节省运行中间代码的时间，则可以为程序设置断点，有以下几种方法：

- ❑ 单击要设置断点的位置，然后单击“标准”工具栏中的“切换断点”按钮 。
- ❑ 单击要设置断点的位置，然后按 F9 键。
- ❑ 在要设置断点的位置的左边缘单击。
- ❑ 右击要设置断点的位置，在弹出的快捷菜单中选择“切换”|“断点”命令，如图 1-44 所示。
- ❑ 将 Stop 语句添加到程序中希望作为断点的位置。

设置为断点位置的代码行会突出显示，并在该行代码的左侧显示一个圆点，如图 1-45 所示。可以在一个程序中设置多个断点。运行包含断点的程序时，会在运行到第一个断点时暂停并进入中断模式。



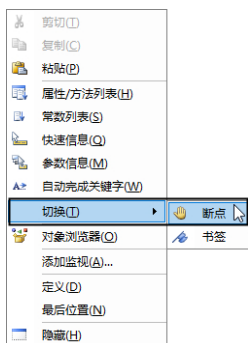


图 1-44 使用鼠标快捷菜单中的命令设置断点

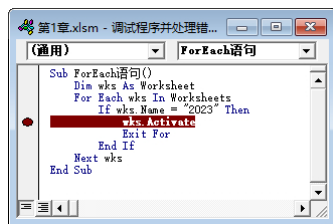


图 1-45 设置断点

清除断点与设置断点的方法相同，只需重复执行前 4 种方法中的任意一种，或者将第 5 种方法中的 Stop 语句从代码中删除。

### 1.10.3 监视程序中的特定值

如需监视程序运行过程中某个变量、属性或表达式的值的变化情况，可以在 VBE 窗口中单击菜单栏中的“调试”|“添加监视”命令，打开如图 1-46 所示的“添加监视”对话框，设置以下几项：

- 在“表达式”文本框中输入想要监视的内容，此处输入的是一个变量。
- 在“上下文”类别中选择实施监视的范围。
- 在“监视类型”类别中选择监视的方式，此处选中“当监视值改变时中断”单选按钮，表示当监视的对象的值发生改变时，会立刻进入中断模式。

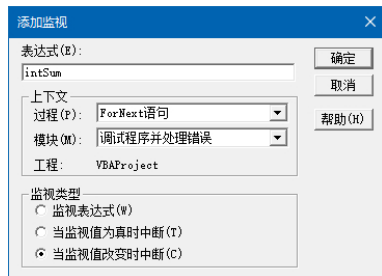


图 1-46 “添加监视”对话框

设置完成后，单击“确定”按钮，关闭“添加监视”对话框。在 VBE 窗口的底部将显示监视窗口，如图 1-47 所示。由于本例将监视类型设置为“当监视值改变时中断”，所以在程序运行过程中，intSum 变量的值每次发生改变时都会自动进入中断模式，并在监视窗口中显示该变量的值。

如需修改或删除现有的监视，可以在监视窗口中右击该项，然后在弹出的快捷菜单中选择“编辑监视”或“删除监视”命令，如图 1-48 所示。

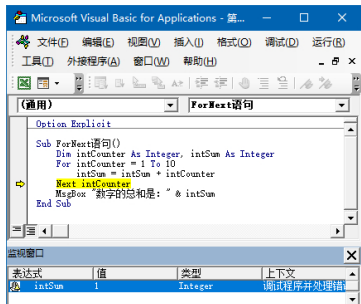


图 1-47 运行程序时监视变量的值

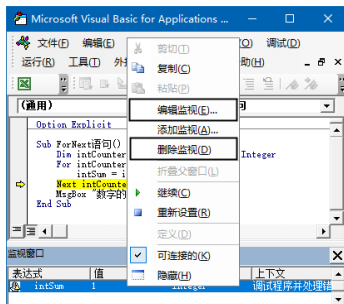


图 1-48 修改或删除监视

### 1.10.4 在立即窗口中测试代码

使用 1.10.3 小节介绍的方法虽然可以监视在程序运行期间某个特定值的变化情况，但是存在一些局限：

- ❑ 当表达式的值不断变化时，同一时间只能显示一个值，下一个值会覆盖上一个值，这样无法查看所有历史值。
- ❑ 每次值发生变化时，都会进入中断模式，影响程序运行的流畅性。
- ❑ 下次重新打开 Excel 文件时，需要重新添加监视的表达式。

使用立即窗口可以解决上述所有问题，使用该窗口可以测试表达式、变量或属性的值，还可以测试过程的执行情况。立即窗口有两种使用方法，一种是直接在立即窗口中使用 `Print` 方法测试代码，另一种方法是在程序中使用 `Debug.Print` 语句添加需要测试的内容，并在程序运行期间自动将测试结果显示在立即窗口中。

#### 1. 在立即窗口中使用 Print 方法

在运行程序时手动或自动进入中断模式后，可以在立即窗口中使用 `Print` 方法测试表达式、变量或属性的值。假设要测试 `intSum` 变量的当前值，可以在立即窗口中输入以下代码，如图 1-49 所示。

```
print intsum
```



图 1-49 在立即窗口中输入要测试的代码

按 `Enter` 键后，将在下一行显示该变量的当前值，如图 1-50 所示。



图 1-50 显示测试结果

英文问号 (?) 是 `Print` 方法的简写形式，在立即窗口中测试代码时可以使用它代替 `Print`。使用英文问号时，在它和表达式之间无须添加空格，如图 1-51 所示。



图 1-51 使用 Print 方法的简写形式

#### 2. 在代码中使用 Debug.Print 语句

如需查看一个变量在一段循环语句中值的变化情况，可以将 `Debug.Print` 语句添加到代码中，在运行程序后会通过该语句自动将该变量的每个值显示到立即窗口中。

在下面的代码中将 `Debug.Print` 语句添加到对 `intSum` 变量求和的代码之后，以便在每次对 `intSum` 变量求和立刻将其值显示在立即窗口中。运行该代码时将在立即窗口中显示如图 1-52 所示的信息。

```
Sub 使用 DebugPrint 语句 ()
```

```

Dim intCounter As Integer, intSum As Integer
For intCounter = 1 To 10
    intSum = intSum + intCounter
    Debug.Print "intSum 变量在第" & intCounter & "次循环时的值是：" & intSum
Next intCounter
MsgBox "数字的总和是：" & intSum
End Sub

```

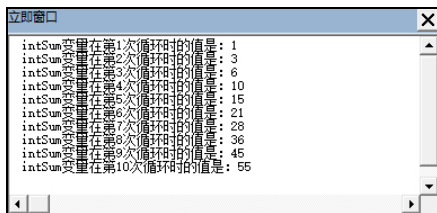


图 1-52 在代码中使用 Debug.Print 语句

### 1.10.5 处理运行时错误

无论是否具备丰富的 VBA 编程经验，在运行编写好的 VBA 程序时，都可能会出现运行时错误。出现运行时错误是正常现象，VBA 程序员要做的是预先考虑到程序可能会执行哪些无效操作，并在程序中事先编写处理无效操作的代码，以便在出现运行时错误，显示有指导意义的提示信息，而不是进入中断模式导致程序无法继续运行。

在 VBA 中可以使用 On Error 语句捕获运行时错误，该语句有以下几种形式：

- ❑ On Error Goto Line: 捕获到运行时错误，将暂停程序的正常运行，并转到由 Line 指定的位置开始执行事先编写好的错误处理程序。Line 由一个字符串和一个冒号组成，表示错误处理程序的起点。在一个 VBA 程序中可以编写多个错误处理程序，每个错误处理程序都有一个对应的 Line。
- ❑ On Error GoTo 0: 关闭正处于活动状态的错误处理程序，并恢复正常的错误捕获状态。如果在程序中使用 On Error Resume Next 语句忽略任何错误，则应该在适当的位置使用 On Error GoTo 0，否则会忽略所有运行时错误。
- ❑ On Error Resume Next: 出现运行时错误时不显示任何提示信息，忽略所有运行时错误并从出错代码的下一行代码继续执行。

通常将处理运行时错误的错误处理程序放在 VBA 程序的结尾，以 Line 作为起点。在错误处理程序中，可以使用 Resume 语句决定完成错误处理后程序的走向。Resume 语句有以下几种形式：

- ❑ Resume: 重新执行导致运行时错误的代码。
- ❑ Resume Next: 从导致运行时错误的代码的下一行代码开始执行。
- ❑ Resume Line: 执行由 Line 标记的代码，此处的 Line 与 On Error Goto Line 中的 Line 的功能和用法类似。

处理运行时错误还可以使用 Err 对象。Err 对象的 Number 属性的值是一个错误号，错误号是一个非零数字。通过检测 Number 属性的值是否是 0，可以判断是否出现了运行时错误。下面通过几个示例介绍使用 On Error 语句、Resume 语句和 Err 对象编写错误处理程序处理运行时错误的方法。

下面的代码将用户在对话框中输入的数字作为除数并被 100 除，然后在对话框中显示计算结果。由于用户输入 0 时将导致运行时错误，所以需要在程序中编写应对该错误的错误处理程序。

首先在可能导致运行时错误的代码的上一行添加 `On Error GoTo Line` 语句，本例将 `Line` 设置为 `ErrTrap`。然后在正常程序的结尾的下一行输入 `ErrTrap` 和一个冒号，按 `Enter` 键后，开始输入处理运行时错误的代码。本例先使用 `MsgBox` 函数显示一个“除数不能是 0”的提示信息，然后递归调用当前 `Sub` 过程，重新显示接收用户输入的对话框，让用户重新输入一个数字。为了防止在正常程序结束后继续执行错误处理程序，所以应该在正常程序的最后一行代码之后添加 `Exit Sub` 语句，在未出现运行时错误时不会执行错误处理程序。

```
Sub 处理运行时错误()  
    Dim strNum As String  
    strNum = InputBox("输入一个整数: ")  
    If strNum = "" Then Exit Sub  
    If Not IsNumeric(strNum) Then Exit Sub  
    On Error GoTo ErrTrap  
    MsgBox "计算结果是: " & 100 / strNum  
    Exit Sub  
ErrTrap:  
    MsgBox "除数不能是 0! "  
    Call 处理运行时错误  
End Sub
```

下面的代码实现相同的功能，但是使用了不同的错误处理方法。先使用 `On Error Resume Next` 语句忽略所有运行时错误，然后检测 `Err` 对象的 `Number` 属性的值，从而判断是否出现了运行时错误，并显示相应的提示信息。本例没有使用 `On Error GoTo 0` 语句，因为处理完错误后没有其他需要执行的代码了，否则应该使用该语句恢复正常的错误捕获状态。

```
Sub 处理运行时错误 2()  
    Dim strNum As String  
    strNum = InputBox("输入一个整数: ")  
    If strNum = "" Then Exit Sub  
    If Not IsNumeric(strNum) Then Exit Sub  
    On Error Resume Next  
    MsgBox "计算结果是: " & 100 / strNum  
    If Err.Number <> 0 Then  
        MsgBox "除数不能是 0! "  
        Call 处理运行时错误 2  
    End If  
End Sub
```

下面的代码在错误处理程序的结尾使用 `Resume` 语句，完成错误处理后会重新执行导致运行时错误的代码。首先使用 `Set` 语句将名为“2023”的工作表赋值给一个 `Worksheet` 类型的对象变量，如果该工作表不存在，则会导致运行时错误，此时将跳转到 `ErrTrap` 标签处开始执行错误处理程序。在错误处理程序中，先显示一个对话框，询问用户是否需要新建一个工作表，如图 1-53 所示。

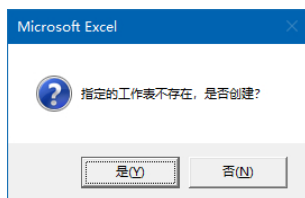


图 1-53 是否创建工作表的提示信息

如果单击对话框中的“是”按钮, 则将新建一个工作表, 并将其命名为“2023”, 然后使用 **Resume** 语句重新执行 **Set** 语句, 此时不会再出现运行时错误, 所以可以正常执行后面的代码。如果单击对话框中的“否”按钮, 则直接退出程序, 不再执行任何操作。

```
Sub 处理运行时错误 3()  
    Dim wks As Worksheet, lngAnswer As Long  
    On Error GoTo ErrTrap  
    Set wks = Worksheets("2023")  
    With wks.Range("A1")  
        .Value = "编号"  
        .Font.Name = "黑体"  
        .Font.Bold = True  
    End With  
    Exit Sub  
ErrTrap:  
    lngAnswer = MsgBox("指定的工作表不存在, 是否创建?", vbYesNo + vbQuestion)  
    If lngAnswer = vbYes Then  
        Worksheets.Add.Name = "2023"  
        Resume  
    Else  
        Exit Sub  
    End If  
End Sub
```

## 第 2 章 控制 Excel 应用程序

Application 对象位于 Excel 对象模型的顶层，它代表 Excel 应用程序。通过 Application 对象可以编程控制 Excel 应用程序，该对象的很多属性和方法用于设置 Excel 应用程序的界面环境和相关选项，与在“Excel 选项”对话框中设置的效果相同。Application 对象的某些方法还可以实现一些特殊操作，例如定时运行程序。本章将介绍使用 Application 对象控制 Excel 应用程序的方法。

### 2.1 Application 对象和全局成员

Application 对象的很多属性和方法都是“全局”成员。“全局”意味着不需要为属性或方法添加对象引用，即可直接使用该对象的属性和方法。例如，ActiveWorkbook 引用活动工作簿，由于它是全局成员，所以下面的两行代码等效。

```
Application.ActiveWorkbook
ActiveWorkbook
```

在 Excel 中有很多与 ActiveWorkbook 类似的全局成员，它们引用不同的活动对象。例如，ActiveCell 引用活动单元格，ActiveSheet 引用活动工作表，ActiveChart 引用活动图表，ActiveWindow 引用活动窗口。

如需查看哪些属性和方法是全局成员，可以在 VBE 窗口中打开对象浏览器，然后在“工程/库”下拉列表中选择“Excel”选项，再在“类”列表框中选择“全局”选项，将在右侧显示 Excel 中的所有全局成员，如图 2-1 所示。

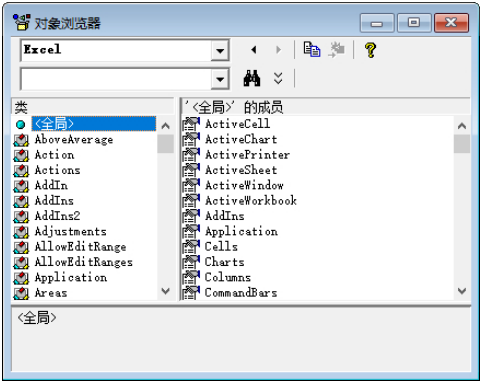


图 2-1 查看 Excel 中的所有全局成员

用于引用活动对象的全局成员为编写通用的 VBA 代码提供了极大的方便。例如，选择一个工作表时，会使其成为活动工作表。如需显示活动工作表的名称，由于事先无法预知哪个工作

表会成为活动工作表，所以可以使用 `ActiveSheet` 动态引用已经成为活动工作表的那个工作表，代码如下。

```
ActiveSheet.Name
```

**提示：**`ActiveSheet` 返回一个 `Worksheet` 对象，使用该对象的 `Name` 属性可以获取工作表的名称。

`Application` 对象的 `Selection` 属性也是全局成员，它表示在活动工作簿的活动工作表中选中的对象，可以是单元格、单元格区域、图表、图形、图片等任何可以选中的对象。下面的代码将选中的单元格区域的字体设置为“宋体”。

```
Selection.Font.Name = "宋体"
```

由于上面的代码事先假设选中的是单元格区域，所以可以正常运行。如果选中的对象不支持 `Font` 属性，例如图片，则运行上面的代码将导致运行时错误。为了避免这种问题，可以先使用 `VBA` 内置的 `TypeName` 函数检测选中对象的类型，然后再执行相应的操作。

`TypeName` 函数返回表示对象类型的字符串，如果被检测的对象类型是单元格或单元格区域，则 `TypeName` 函数将返回“`Range`”。下面的代码使用 `If Then Else` 语句检测 `Selection` 的类型，如果选中的对象不是单元格或单元格区域，则会显示提示信息。

```
Sub Selection 属性 ()
    If TypeName(Selection) = "Range" Then
        Selection.Font.Name = "宋体"
    Else
        MsgBox "需要先选择一个单元格或单元格区域"
    End If
End Sub
```

**注意：**在使用 `TypeName` 函数的表达式的等号右侧输入的字符串的大小写形式，必须与 `TypeName` 函数的返回值的大小写形式完全相同。

在代码中输入 `Selection` 和一个英文句点之后，不会自动显示包含属性和方法的成员列表，这是因为 `Selection` 默认是 `Object` 类型，它代表一般对象类型，这意味着 `VBA` 无法确定 `Selection` 到底是哪种对象。如需解决这个问题，可以先声明一个特定对象类型的变量，然后将 `Selection` 赋值给该变量，之后就会显示包含属性和方法的成员列表了，如图 2-2 所示。

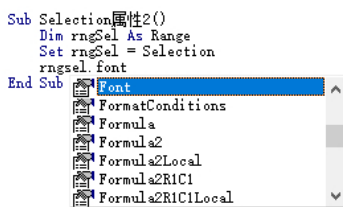


图 2-2 为 `Selection` 显示包含属性和方法的成员列表

## 2.2 获取 Excel 应用程序的基本信息

编程创建 `Excel VBA` 程序时，事先检查程序的一些关键信息是很有必要的，包括 `Excel` 版本号、用户名、安装路径、启动文件夹路径和工作簿模板路径等。本节将介绍使用 `Application`

对象获取这些信息的方法。

### 2.2.1 使用 Version 属性获取 Excel 版本号

使用 Application 对象的 Version 属性将返回 Excel 应用程序的版本号，代码如下。

```
Application.Version
```

Version 属性的返回值与 Excel 版本的对应情况如表 2-1 所示。

表 2-1 Version 属性返回的版本号与 Excel 版本之间的对应情况

| Version 属性的返回值 | 对应的 Excel 版本                       |
|----------------|------------------------------------|
| 16.0           | Excel 2021、Excel 2019 和 Excel 2016 |
| 15.0           | Excel 2013                         |
| 14.0           | Excel 2010                         |
| 12.0           | Excel 2007                         |
| 11.0           | Excel 2003                         |

如果 VBA 程序中的某些功能受到不同 Excel 版本的影响，为了避免程序出现无法预料的问题，应该检测 Excel 应用程序的版本，然后根据不同的 Excel 版本选择相应的处理方式。下面的代码检测当前正在使用的 Excel 应用程序的版本，如果 Excel 版本是 Excel 2007 或更低版本，则显示“当前版本过低，无法使用本程序”，并退出程序，否则显示另一条信息，如图 2-3 所示。

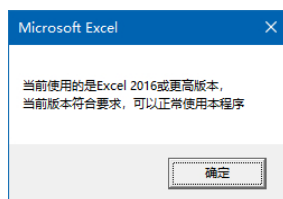


图 2-3 检测 Excel 版本并显示相应的信息

```
Sub 检测 Excel 版本()  
    Select Case Val(Application.Version)  
        Case Is >= 14  
            MsgBox "当前版本符合要求，可以正常使用本程序"  
        Case Else  
            MsgBox "当前版本过低，无法使用本程序，" & vbCrLf & "请安装 Excel 2010 或更高版本"  
            Exit Sub  
        End Select  
    End Sub
```

**提示：**Val 函数是一个 VBA 内置函数，该函数返回字符串中的第一个数字。如果字符串的第一个字符不是数字，则返回 0。例如，Val("666Like888")返回 666。vbCrLf 是一个 VBA 内置常数，表示将插入点移动到下一行的起始位置，可以使用 Chr(13)& Chr(10)代替 vbCrLf。

### 2.2.2 使用 UserName 属性获取 Excel 用户名

使用 Application 对象的 UserName 属性将返回 Excel 用户，即在“Excel 选项”对话框中设置的用户名，如图 2-4 所示。



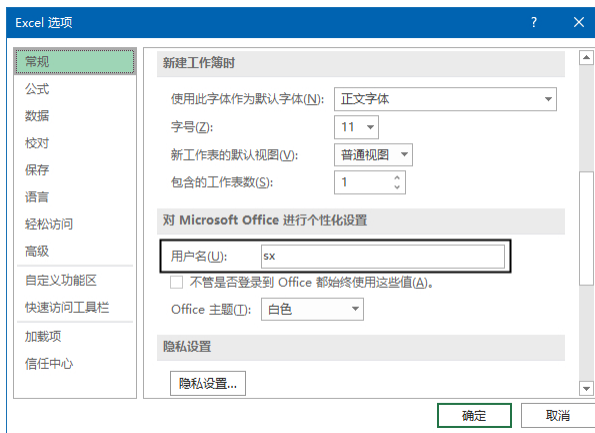


图 2-4 在“Excel 选项”对话框中设置的用户名

下面的代码将用户在对话框中输入的名称设置为 Excel 用户名。如果用户什么都没输入就单击“确定”按钮，或者单击“取消”按钮，则会再次显示输入对话框，直到用户输入内容为止。

```
Sub 设置 Excel 用户名()  
    Dim strUserName As String  
    Do  
        strUserName = InputBox("输入用户名：")  
    Loop While strUserName = ""  
    Application.UserName = strUserName  
End Sub
```

### 2.2.3 使用 Path 属性获取 Excel 的安装路径

使用 Application 对象的 Path 属性将返回 Excel 应用程序的安装路径。

```
Application.Path
```

可以在立即窗口中测试上述代码的运行结果，如图 2-5 所示。

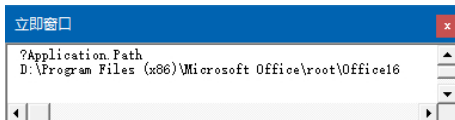


图 2-5 在立即窗口中测试 Path 属性的返回值

### 2.2.4 使用 StartupPath 属性获取启动文件夹路径

位于启动文件夹中的工作簿会在启动 Excel 时自动打开，第 1 章介绍过的 Personal.xlsb 工作簿就位于启动文件夹中。使用 Application 对象的 StartupPath 属性将返回 Excel 启动文件夹的路径，如图 2-6 所示。

```
Application.StartupPath
```

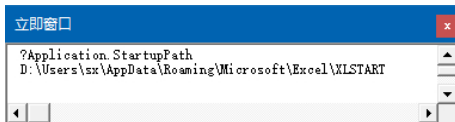


图 2-6 在立即窗口中测试 StartupPath 属性的返回值

## 2.2.5 使用 TemplatesPath 属性获取工作簿模板路径

如需在 VBA 程序中处理工作簿模板，需要先知道工作簿模板的存储位置。使用 Application 对象的 TemplatesPath 属性将返回 Excel 工作簿模板的路径，如图 2-7 所示。

```
Application.TemplatesPath
```



图 2-7 在立即窗口中测试 TemplatesPath 属性的返回值

## 2.3 设置 Excel 应用程序的界面环境

本节将介绍编程控制 Excel 界面环境的方法，其中的一些设置可以在“Excel 选项”对话框中手动完成，而另一些设置只能通过编程才能实现。

### 2.3.1 使用 Visible 属性设置 Excel 应用程序的可见性

启动 Excel 应用程序时，默认会正常显示程序窗口。当编写一个需要验证用户的 VBA 程序时，可能需要在显示程序窗口之前先验证用户的有效性，此时就需要在启动 Excel 时隐藏暂时隐藏程序窗口。

使用 Application 对象的 Visible 属性可以控制 Excel 程序窗口的可见性，该属性为 True 表示正常显示程序窗口，该属性为 False 表示隐藏程序窗口。下面的代码将隐藏 Excel 程序窗口。

```
Application.Visible = False
```

### 2.3.2 使用 WindowState 属性设置 Excel 窗口的显示状态

窗口的显示状态是指以最大化、最小化等方式显示窗口。使用 Application 对象的 WindowState 属性可以返回或设置 Excel 窗口的状态，该属性返回或设置的值由 XlWindowState 常量提供，如表 2-2 所示。

表 2-2 XlWindowState 常量

| 名 称         | 值     | 说 明 |
|-------------|-------|-----|
| xlMaximized | -4137 | 最大化 |
| xlMinimized | -4140 | 最小化 |
| xlNormal    | -4143 | 正常  |

下面的代码将活动的 Excel 窗口最大化显示。

```
Application.WindowState = xlMaximized
```

下面的代码使用 For Each 语句将当前打开的所有 Excel 窗口都设置为最大化。

```
Sub 最大化显示所有 Excel 窗口()
    Dim win As Window
    For Each win In Windows
```

```

win.WindowState = xlMaximized
Next win
End Sub

```

下面的代码实现相同的功能,但是使用的是 For Next 语句,此处使用 Windows 对象的 Count 属性获取所有打开窗口的总数,然后使用索引号引用每一个打开的窗口。Windows 是 Application 对象的属性,该属性返回 Windows 集合,使用该集合的 Count 属性返回窗口总数。

```

Sub 最大化显示所有 Excel 窗口 2 ()
    Dim intIndex As Integer
    For intIndex = 1 To Windows.Count
        Windows(intIndex).WindowState = xlMaximized
    Next intIndex
End Sub

```

### 2.3.3 使用 DisplayFullScreen 属性设置是否全屏显示 Excel

与最大化显示 Excel 窗口不同,全屏显示会将 Excel 窗口中的标题栏、功能区、编辑栏和状态栏都隐藏起来。使用 Application 对象的 DisplayFullScreen 属性可以设置 Excel 窗口是否全屏显示,该属性为 True 表示全屏显示,该属性为 False 表示不全屏显示。下面的代码将 Excel 窗口全屏显示。

```
Application.DisplayFullScreen = True
```

### 2.3.4 使用 Caption 属性设置 Excel 标题栏

在 Excel 窗口顶部的标题栏中默认显示工作簿的名称和 Excel 应用程序的名称,例如“工作簿1 - Excel”,如图 2-8 所示。



图 2-8 Excel 窗口顶部的标题栏

使用 Application 对象的 Caption 属性可以设置标题栏中的 Excel 应用程序的名称。下面的代码将标题栏中默认显示的“Excel”替换为“测试系统”,如图 2-9 所示。

```
Application.Caption = "测试系统"
```



图 2-9 修改标题栏中的 Excel 应用程序的名称

如果不想在标题栏中显示任何内容,可以将 Caption 属性设置为包含空格的字符串。

```
Application.Caption = " "
```

如需恢复标题栏的默认名称,可以将 Caption 属性设置为零长度字符串。

```
Application.Caption = ""
```

使用 Application 对象的 Caption 属性可以返回标题栏中的所有内容,而不只是 Excel 应用程序的名称。下面的代码将在对话框中显示标题栏中的所有内容,如图 2-10 所示。

如果只想返回标题栏中位于“-”后面的内容,则可以使用下面的代码,使用 Mid 函数在整

个标题栏内容中查找每一个字符以确定“-”的位置，然后使用 Right 函数提取位于“-”后面的所有字符。代码的运行效果如图 2-11 所示。

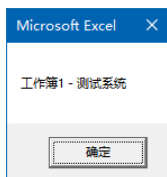


图 2-10 使用 Caption 属性返回标题栏内容

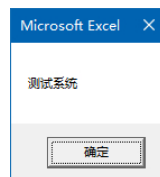


图 2-11 提取标题栏中的程序名称

```
Sub 提取标题栏中的程序名称 ()
    Dim strTitle As String
    Dim strAppName As String
    Dim intStart As Integer
    strTitle = Application.Caption
    For intStart = 1 To Len(strTitle)
        If Mid(strTitle, intStart, 1) = "-" Then
            strAppName = Right(strTitle, Len(strTitle) - intStart)
            Exit For
        End If
    Next intStart
    MsgBox strAppName
End Sub
```

### 2.3.5 使用 DisplayFormulaBar 属性设置是否显示编辑栏

Excel 中的编辑栏显示在单元格区域的上方，是一个类似文本框的长条矩形区域，可以在编辑栏中输入公式来计算工作表中的数据，如图 2-12 所示。

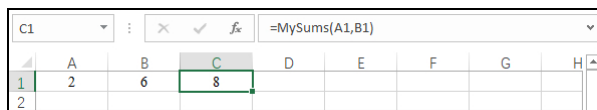


图 2-12 编辑栏

使用 Application 对象的 DisplayFormulaBar 属性可以设置是否显示编辑栏，该属性为 True 表示显示编辑栏，该属性为 False 表示隐藏编辑栏。下面的代码将隐藏编辑栏。

```
Application.DisplayFormulaBar = False
```

### 2.3.6 使用 ShowMenuFloaties 属性设置右击单元格是否显示浮动工具栏

浮动工具栏是在单元格进入编辑模式后选择内容时，或右击单元格时，在单元格附近显示的微型工具栏，其中包含常用的格式选项，可以提高设置格式的 efficiency，如图 2-13 所示。

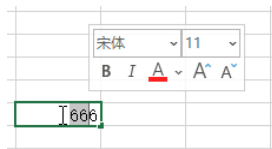


图 2-13 浮动工具栏

使用 Application 对象的 ShowSelectionFloaties 属性可以设置在单元格中选择内容时是否显

示浮动工具栏，该属性为 **False** 表示显示浮动工具栏，该属性为 **True** 表示不显示浮动工具栏。下面的代码在单元格中选择内容时不显示浮动工具栏：

```
Application.ShowSelectionFloaties = True
```

即使将 **ShowSelectionFloaties** 属性设置为 **False** 不显示浮动工具栏，但是在右击单元格时仍会显示浮动工具栏。如果希望在右击单元格时也不显示浮动工具栏，则可以将 **Application** 对象的 **ShowMenuFloaties** 属性设置为 **True**。

```
Application.ShowMenuFloaties = True
```

如需彻底禁止显示浮动工具栏，可以将上面两个属性都设置为 **True**。

### 2.3.7 使用 ShowDevTools 属性设置是否显示“开发工具”选项卡

使用 **Application** 对象的 **ShowDevTools** 属性可以设置在功能区中是否显示“开发工具”选项卡，该属性为 **True** 表示在功能区中显示“开发工具”选项卡，该属性设置为 **False** 表示在功能区中不显示“开发工具”选项卡。下面的代码在功能区中显示“开发工具”选项卡。

```
Application.ShowDevTools = True
```

### 2.3.8 使用 StatusBar 属性设置在状态栏中显示的信息

状态栏位于 Excel 窗口的底部，其中显示在 Excel 中执行操作时的状态信息，例如单元格当前所处的编辑模式。当执行耗时较长的操作时，可以在状态栏中显示有关程序处理进度的信息。使用 **Application** 对象的 **StatusBar** 属性可以返回或设置在状态栏中显示的信息。

下面的代码在 1 万个单元格中依次输入从 1 开始的连续自然数，程序运行期间在状态栏中实时显示当前正在处理第几个单元格，如图 2-14 所示。在程序结束前，应该将 **StatusBar** 属性设置为 **False**，使状态栏恢复到 Excel 默认状态，否则在状态栏中会一直显示由程序设置的信息。



图 2-14 在状态栏中显示操作进度

```
Sub 在状态栏中显示操作进度 ()
    Dim intCell As Integer
    For intCell = 1 To 10000
        Cells(intCell).Value = intCell
        Application.StatusBar = "正在处理第" & intCell & "个单元格"
    Next intCell
    Application.StatusBar = False
End Sub
```

### 2.3.9 使用 DisplayAlerts 属性设置警告信息的显示方式

运行 VBA 程序时，可能会显示来自 Excel 的警告信息，只有对这类信息进行处理后，程序才能继续运行。例如，当 VBA 程序执行删除工作表的操作时，将显示如图 2-15 所示的对话框，只有单击“删除”按钮或“取消”按钮后，程序才能继续运行。

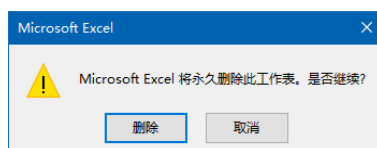


图 2-15 Excel 中的警告信息

如果不想显示该对话框而直接执行删除操作,则可以将 `Application` 对象的 `DisplayAlerts` 属性设置为 `False`,屏蔽任何 Excel 警告信息,此时会自动执行与对话框中默认按钮关联的操作。默认按钮是指对话框中处于焦点的按钮,即在不改变选择的情况下直接按 `Enter` 键生效的按钮。从外观上看,默认按钮的边缘显示为虚线或蓝色。

下面的代码执行活动工作表,为了避免出现警告信息,在执行该操作前将 `DisplayAlerts` 属性设置为 `False`。如果在删除工作表之后还有很多代码,则应该将 `DisplayAlerts` 属性设置为 `True`,使 Excel 可以正常显示警告信息,否则会屏蔽后面可能出现的所有警告信息。

```
Sub 设置警告信息的显示方式()  
    Application.DisplayAlerts = False  
    ActiveSheet.Delete  
    Application.DisplayAlerts = True  
End Sub
```

### 2.3.10 使用 `DefaultFilePath` 属性设置打开文件的默认路径

如需频繁从同一个文件夹中打开 Excel 工作簿,可以将该文件夹设置为打开文件的默认路径。以后每次执行打开文件的操作时,默认都会显示该文件夹。在“Excel 选项”对话框的“保存”选项卡中,通过设置“默认本地文件位置”选项可以实现该功能,如图 2-16 所示。

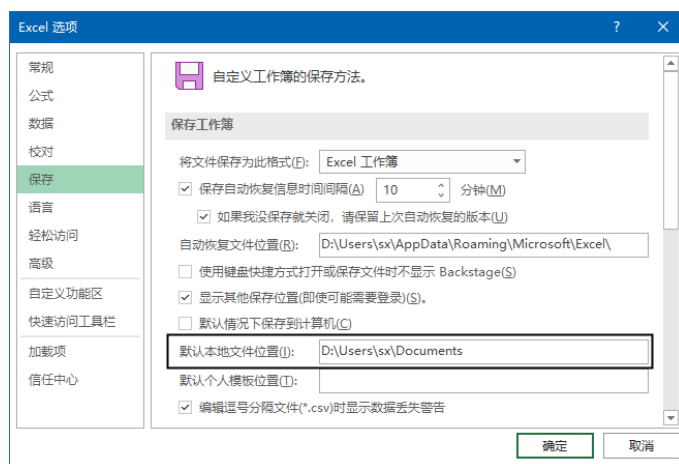


图 2-16 设置打开文件的默认路径

在 VBA 中可以使用 `Application` 对象的 `DefaultFilePath` 属性设置打开文件的默认路径。如果设置的路径不存在,则不会使无效路径的设置生效,也不会出现任何错误提示。下面的代码将指定的路径存储在 `strPath` 变量中,然后使用 VBA 内置的 `Dir` 函数检测该路径是否有效,如果路径有效,则 `Dir` 函数将返回路径结尾的文件夹名称,此时将该路径设置为 `DefaultFilePath` 属性的值;如果路径不存在,则 `Dir` 函数将返回零长度字符串,此时会显示提示信息。

```

Sub 设置打开文件的默认路径 ()
    Dim strPath As String
    strPath = "E:\测试数据\Excel"
    If Dir(strPath, vbDirectory) <> "" Then
        Application.DefaultFilePath = strPath
    Else
        MsgBox "指定的路径无效"
    End If
End Sub

```

### 2.3.11 使用 SheetsInNewWorkbook 属性设置新工作簿中的工作表数

在不同版本的 Excel 中,新建的工作簿中默认包含不同数量的工作表。实际上,可以在“Excel 选项”对话框的“常规”选项卡中通过修改“包含的工作表数”选项来更改默认的工作表数,如图 2-17 所示。

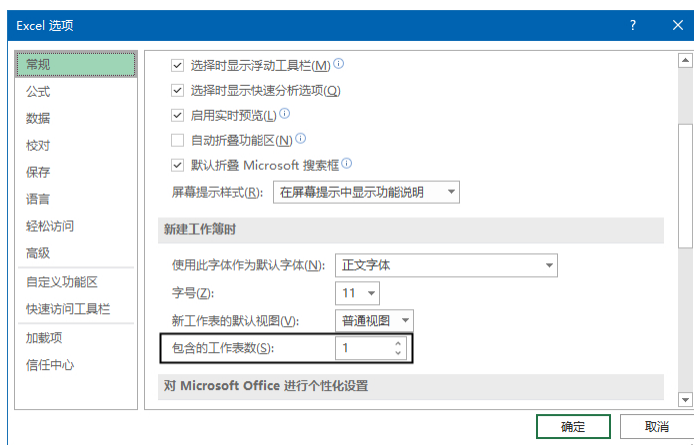


图 2-17 更改新建的工作簿中默认包含的工作表数

在 VBA 中可以使用 Application 对象的 SheetsInNewWorkbook 属性设置新建的工作簿中默认包含的工作表数。下面的代码将在每次新建的工作簿中自动包含 6 个工作表。

```
Application.SheetsInNewWorkbook = 6
```

### 2.3.12 使用 StandardFont 和 StandardFontSize 属性设置工作簿的默认字体和字号

为新建的工作簿设置默认的字体和字号有两种方法,一种是创建一个工作簿模板,在其中设置好字体和字号,保存后将其放置到 2.2.4 小节介绍的 Excel 启动文件夹中,以后每次在快速访问工具栏中单击“新建”按钮时,新建的工作簿中的字体和字号都与工作簿模板保持一致。

另一种方法是在“Excel 选项”对话框的“常规”选项卡中设置“使用此字体作为默认字体”和“字号”两个选项,如图 2-17 所示。以后每次单击“文件”按钮,然后选择“新建”|“空白工作簿”命令时,创建的空白工作簿中的字体和字号将由这两个选项决定。

使用 Application 对象的 StandardFont 和 StandardFontSize 属性可以设置第二种方法中的默认字体和字号。下面的代码将新建的工作簿中的默认字体设置为“黑体”,默认字号设置为“12”。

```
Application.StandardFont = "黑体"  
Application.StandardFontSize = 12
```

注意：运行代码会立刻在“Excel 选项”对话框中显示设置结果，但是只有在退出 Excel 应用程序并再次启动它之后，设置才会真正对以后新建的工作簿有效。

## 2.4 Excel 应用程序的特殊操作

Application 对象的一些属性和方法可以实现一些比较特殊的功能，包括控制屏幕刷新、计算字符串表达式、定时运行程序和为程序设置快捷键，这些功能通常很难在 Excel 界面环境中实现。本节将介绍在 VBA 中使用 Application 对象实现这些功能的方法。

### 2.4.1 使用 ScreenUpdating 属性控制屏幕刷新

屏幕刷新是指在程序运行期间，程序执行的各种操作将导致屏幕中显示的内容不断变化和闪烁，这种现象通常会给用户带来不好的体验。使用 Application 对象的 ScreenUpdating 属性可以开启或关闭屏幕刷新，该属性为 True 表示开启屏幕刷新，该属性为 False 表示关闭屏幕刷新。

在即将开始执行一段包含大量操作的代码时，可以将 ScreenUpdating 属性设置为 False，暂时关闭屏幕刷新，这样在代码运行期间将不会让屏幕显示有任何变化，还能加快代码的运行速度。

```
Application.ScreenUpdating = False
```

在包含大量操作的代码运行结束后，可以将 ScreenUpdating 属性设置为 True，重新开启屏幕刷新。

```
Application.ScreenUpdating = True
```

注意：如需在程序运行期间显示 Excel 内置对话框或用户窗体，应该开启屏幕刷新，否则拖动对话框时会在屏幕上产生橡皮擦的效果。

### 2.4.2 使用 Evaluate 方法将字符串转换为对象或值

使用 Application 对象的 Evaluate 方法可以将一个字符串转换为 Excel 对象或值。Evaluate 方法有以下两种语法：

```
Evaluate("字符串")  
[字符串]
```

使用第一种语法时，必须将字符串放在一对英文双引号中。使用第二种语法时，直接将字符串放在中括号内，不能在字符串的两侧添加英文双引号。下面两行代码都引用 A1 单元格：

```
Evaluate("A1")  
[A1]
```

下面两行代码都返回 3 个数字的乘积：

```
Evaluate("2*3*5")  
[2*3*5]
```

在上面两个示例中，使用第一种语法的优点是可以在字符串中使用&连接符连接变量，从



而灵活构建表达式。使用第二种语法的优点是代码更简洁。下面的代码使用第一种语法，将变量和字符串组合在一起，用作 Evaluate 方法的参数，计算 10 和 strNumber 变量的乘积，该变量的值由用户指定。

```
Sub Evaluate 方法()
    Dim strNumber As String
    strNumber = Val(InputBox("输入一个整数"))
    MsgBox Application.Evaluate("10*" & strNumber)
End Sub
```

第1章曾经介绍过，如果在 Excel 工作表函数和 VBA 内置函数中同时存在完成同一个功能的函数，则在 VBA 程序中不能使用完成该功能的 Excel 工作表函数。然而，Application 对象的 Evaluate 方法打破了这种限制，可以在该方法中以字符串的形式编写使用 Excel 工作表函数的公式，Evaluate 方法会计算其结果。

下面两行代码使用工作表函数 ISBLANK 判断 A1 单元格是否为空。由于该函数与 VBA 内置的 IsEmpty 函数等效，所以不能在 VBA 中使用 ISBLANK 函数，但是可以在 Evaluate 方法中使用该函数。

```
Evaluate("ISBLANK(A1)")
[ISBLANK(A1)]
```

### 2.4.3 使用 OnTime 方法定时运行 VBA 程序

使用 Application 对象的 OnTime 方法可以在指定的时间自动运行 VBA 程序，只要 Excel 应用程序一直处于运行状态，但是不必打开包含 VBA 程序的工作簿，Excel 会在需要时自动打开它。在到达指定的时间之前，用户在 Excel 中的各种操作不受影响。

OnTime 方法的语法如下：

```
Application.OnTime(EarliestTime, Procedure, LatestTime, Schedule)
```

- EarliestTime（必需）：运行 VBA 程序的时间。
- Procedure（必需）：VBA 程序的名称。
- LatestTime（可选）：运行 VBA 程序的最后时间。如果在到达由 EarliestTime 参数指定的时间时，Excel 正在执行其他程序，则会等待该程序结束后再运行由 Procedure 参数指定的 VBA 程序。如果将 LatestTime 参数设置为一个时间，则会等到达该时间时再运行指定的 VBA 程序。
- Schedule（可选）：该参数为 True 表示安排一个新的运行计划，该参数为 False 表示清除当前正处于活动状态的运行计划。省略该参数时，其默认值是 True。

设置 EarliestTime 参数时可以使用 VBA 内置的 TimeValue 或 TimeSerial 函数。TimeValue 函数只有一个参数，它是一个表示时间的字符串。TimeSerial 函数有 3 个参数，分别表示时间中的时、分、秒。下面两种形式都表示当天下午 3 点 30 分：

```
TimeValue("15:30:00")
TimeSerial(15, 30, 0)
```

如需表示指定时间间隔之后的时间，可以使用+连接符连接当前时间和由 TimeValue 或 TimeSerial 函数设置的时间间隔。下面两种形式都表示 30 分钟后的时间，其中的 Now 函数返

回当前系统时间。

```
Now + TimeValue("00:30:00")
Now + TimeSerial(0, 30, 0)
```

了解如何设置时间后，就可以很容易使用 On Time 方法定时运行 VBA 程序了。下面的代码在当天上午 9 点半自动运行名为“例会提醒”的 Sub 过程，向用户发送开会提醒的信息。为了使该功能生效，需要先运行包含 On Time 方法的 Sub 过程。

```
Sub OnTime 方法 ()
    Application.OnTime TimeValue("9:30:00"), "例会提醒"
End Sub

Sub 例会提醒 ()
    MsgBox "今天上午 10 点开例会"
End Sub
```

运行下面的代码，将在 15 分钟后自动运行“例会提醒”Sub 过程，此处使用的是 TimeSerial 函数。

```
Sub OnTime 方法 2 ()
    Application.OnTime Now + TimeSerial(0, 15, 0), "例会提醒"
End Sub
```

如需定时重复运行一个 VBA 程序，可以在该程序内使用 On Time 方法，并将该方法的 Procedure 参数设置为该程序。运行下面的代码，每隔 10 分钟会向用户发送一次例会提醒的信息。

```
Sub 例会提醒 2 ()
    Application.OnTime Now + TimeSerial(0, 10, 0), "例会提醒 2"
    MsgBox "今天上午 10 点开例会"
End Sub
```

如需停止定时重复运行某个 VBA 程序，需要在一个单独的 Sub 过程中编写代码，获取为运行该程序设置的时间，并将 OnTime 方法的 Schedule 参数设置为 False，即取消定时运行计划。为了在同一个模块中的不同 Sub 过程之间共享同一个时间，需要在该模块中声明一个模块级变量，然后在定时重复运行程序的 Sub 过程和停止定时重复运行的 Sub 过程中使用该变量传递时间。

在下面的示例中，先运行名为“例会提醒 3”的 Sub 过程，将每隔 10 分钟发送一次例会提醒。在该过程中，将定制运行程序的时间赋值给名为 dateTime 的模块级变量，然后在“停止例会提醒”Sub 过程中将该变量设置为 On Time 方法的 EarliestTime 参数的值，从而确保两个过程中的时间完全相同。

```
Dim dateTime As Date

Sub 例会提醒 3 ()
    dateTime = Now + TimeSerial(0, 10, 0)
    Application.OnTime dateTime, "例会提醒 3"
    MsgBox "今天上午 10 点开例会"
End Sub

Sub 停止例会提醒 ()
    Application.OnTime dateTime, "例会提醒 3", , False
End Sub
```

## 2.4.4 使用 OnKey 方法为 VBA 程序设置快捷键

使用 Application 对象的 OnKey 方法可以为 VBA 程序设置快捷键，之后可以通过快捷键运行指定的 VBA 程序。OnKey 方法的语法如下：

```
Application.OnKey(Key, Procedure)
```

- Key（必需）：为 VBA 程序设置的快捷键。表 2-3 列出了在 OnKey 方法中表示按键的代码，在该表中没有列出字母键、数字键和符号键，因为这些按键在 OnKey 方法中直接使用按键本身的字符表示，例如 A 键在 OnKey 中表示为"a"，如果写为"A"，则表示同时按 Shift 键和 A 键。如需在快捷键中包含+、^或%，则需要将它们放在一对大括号中，此时表示它们是按键本身，而不代表 Shift 键、Ctrl 键或 Alt 键，例如"^{+}"表示按 Ctrl 键和加号键。
- Procedure（可选）：设置快捷键的 VBA 程序的名称。

表 2-3 OnKey 方法中与按键对应的代码

| 按 键          | 代 码              | 按 键         | 代 码          |
|--------------|------------------|-------------|--------------|
| Shift        | +                | F1~F15      | {F1}~{F15}   |
| Ctrl         | ^                | Tab         | {TAB}        |
| Alt          | %                | Ins         | {INSERT}     |
| Enter        | {ENTER}或~（波形符）   | Break       | {BREAK}      |
| Esc          | {ESCAPE}或{ESC}   | 向上          | {UP}         |
| Backspace    | {BACKSPACE}或{BS} | 向下          | {DOWN}       |
| Delete 或 Del | {DELETE}或{DEL}   | 向左          | {LEFT}       |
| Home         | {HOME}           | 向右          | {RIGHT}      |
| End          | {END}            | Caps Lock   | {CAPSLOCK}   |
| Pageup       | {PGUP}           | Num Lock    | {NUMLOCK}    |
| Pagedown     | {PGDN}           | Scroll Lock | {SCROLLLOCK} |

下面的代码将“显示欢迎信息”过程的快捷键设置为 Ctrl+1，然后运行名为“设置快捷键”的过程，之后可以按 Ctrl+1 组合键运行“显示欢迎信息”过程。在 Excel 中按 Ctrl+1 组合键默认将打开“设置单元格格式”对话框，现在失去该功能，而是运行“显示欢迎信息”过程。

```
Sub 设置快捷键()  
    Application.OnKey "^1", "显示欢迎信息"  
End Sub  
  
Sub 显示欢迎信息()  
    MsgBox "你好"  
End Sub
```

**提示：**退出 Excel 应用程序之前，使用 OnKey 方法设置的快捷键一直有效。

如需恢复上面示例中的 Ctrl+1 组合键在 Excel 中的默认功能，可以省略 OnKey 方法的第 2 个参数。

```
Application.OnKey "^1"
```

如需彻底禁用 Ctrl+1 组合键在 Excel 中的功能，可以将 OnKey 方法的第 2 个参数设置为零长度字符串。

```
Application.OnKey "^1", ""
```