



Excel VBA 解读



基础入门篇

完美 Excel 出品

微信公众号: *excelperfect*



探索 Excel 开发的奥妙，层层解读 Excel VBA 开发技术。

在个人学习和实践的基础上，整理、归纳、总结和提炼 Excel VBA 开发的原理、技术、方法、技巧以及实例，与 Excel 爱好者分享、探讨和交流。一方面，促进自己不断地学习和提高；另一方面，将学习研究成果与大家分享，共同提高。

本电子书是《Excel VBA 解读》系列的基础入门篇。

坚持分享最好的Excel与VBA技术知识

微信公众号: *excelperfect*

知嗒知识号: *完美Excel*



目录

1. 引子	1
了解 VBA.....	1
实例介绍.....	2
编程能力快速提升法.....	4
关于《Excel VBA 解读》	4
2. Excel 中的加强版“录像机”	5
录制宏.....	5
运行宏.....	6
录制宏的三大用处及好处.....	7
录制宏的技巧.....	8
3. VBA 代码之家.....	9
代码在哪儿.....	9
初识 VBE 界面	10
输入代码.....	11
执行代码.....	12
示例：使用“立即窗口”调试或执行代码	13
4. 如影随形的帮助	15
官方帮助.....	15
VBE 中的 F1 键和 F2	16
网上资源.....	18
5. 认识对象和 Excel 对象模型	19

对象及其属性和方法.....	19
集合.....	20
对象模型.....	21
6. 看看 Excel 的那些常用对象	23
Application 对象	24
Workbook 对象.....	24
Window 对象.....	25
小结.....	26
7. 看看 Excel 的那些常用对象（续 1）	29
Worksheet 对象.....	29
8. 看看 Excel 的那些常用对象（续 2）	33
Range 对象	33
ActiveCell 属性.....	35
Cells 属性	35
Selection 属性.....	35
小结.....	35
Rows 属性	37
Columns 属性	37
9. 看看 Excel 的那些常用对象（续 3）	39
Comment 对象	39
Chart 对象.....	40
拾遗——工作表与图表工作表.....	41
ActiveChart 属性.....	41
10. 神奇的句点	43
省略 Application 对象限定	45
11. 神奇的句点（续）	47
对象方法的参数.....	48
对象属性的参数.....	49
参数的表示方法.....	49
默认属性.....	50

12. 再回首，说透对象	51
小结.....	54
13. 一个简单的 VBA 程序.....	55
[复习]准备代码输入环境	55
输入程序代码.....	56
注释.....	56
VBA 程序的基本结构	57
VBA 程序语言元素	57
[复习]运行程序代码	58
14. VBA 的数据类型	59
为什么会有数据类型.....	60
15. 变量和常量	63
一个示例.....	63
什么是变量.....	64
命名变量.....	64
变量的数据类型.....	65
声明变量.....	65
常量.....	66
声明常量.....	67
附：VBA 的保留字	67
16. VBA 的运算符	69
算术运算符.....	69
关系运算符.....	70
Is 运算符	71
逻辑运算符.....	71
字符串运算符.....	73
运算符的优先级.....	73
赋值运算符.....	74
17. 谈谈对象变量	77
声明对象变量.....	77

给对象变量赋值.....	78
使用对象变量的好处.....	79
18. With... End With 结构，专为对象而生	81
19. 员工管理系统开发 V1.0	85
系统初步规划.....	85
表格设计.....	86
代码.....	86
20. 用户交互初体验—MsgBox 函数.....	89
显示信息.....	89
MsgBox 函数的语法.....	90
获取响应的值.....	92
21. 向左走，向右走——使用 If 语句选择	94
22. 方便的“多选一”结构——Select Case 结构.....	99
23. 有限次的循环	103
24. 有条件的循环（1）	107
25. 有条件的循环（2）	113
26. 在对象中循环——For Each-Next 结构的使用.....	117
27. 看看 VBA 的 Sub 过程和 Function 过程	121
28. 简单问答	125
问题 1：为什么在打开有些 Excel 文件时出现安全警告？	125
问题 2：为什么在保存 Excel 文件时会出现额外的提示信息？	126
问题 3：为什么 Excel 界面中没有“开发工具”选项卡？	126
问题 4：VBA 代码应该放置在哪里？	127
问题 5：如何运行 VBA 代码？	128
结语.....	131
关于完美 Excel	134

主要内容

1. 引子

简单地介绍了一下 VBA，并以自己开发使用的 2 个小实例简要展示了 VBA 在满足个性化需求方面的能力，同时介绍了快速提升编程能力的 5 大方法，最后介绍了《Excel VBA 解读》要达到的目的及文字内容编排。

2. Excel 中的加强版录像机

简要地介绍了 Excel 的录制宏功能，以及录制宏的 3 大用处和好处、技巧。在以后的许多文章中，特别是在介绍对象的属性和方法时，我们常常会通过录制宏来引入对象的属性和方法，方便读者对照理解。此外，还简单地介绍了基本的运行宏的方法。

3. VBA 代码之家

简要地介绍了如何进入 VBE 代码编辑器及其界面、如何输入代码、如何执行代码。同时，介绍了一个使用“立即窗口”调试或执行代码的示例。

4. 如影随形的帮助

在学习或者编写 VBA 代码时，可以利用有许多有效的帮助资源。本文介绍了 VBA 的帮助系统及使用，以及 2 个专业的 Excel 网站。

5.认识对象和 Excel 对象模型

以日常生活中的一些例子作为对比,帮助读者理解 Excel 的对象及其属性和方法、集合以及对象模型的概念,这是编写 Excel VBA 程序的基础。

6.看看 Excel 的那些常用对象

初步介绍了 `Application` 对象、`Workbook` 对象及 `Window` 对象,让读者认识这些对象并知道怎么在 VBA 代码中表示它们。

7.看看 Excel 的那些常用对象 (续 1)

初步介绍了 `Worksheet` 对象及相关属性(`ActiveSheet` 属性),给出了在 VBA 代码中表示工作表的 3 种方法。

8.看看 Excel 的那些常用对象 (续 2)

初步介绍了 `Range` 对象及相关属性(`ActiveCell` 属性、`Cells` 属性、`Selection` 属性、`Rows` 属性、`Columns` 属性),给出了表示单个单元格的 7 种方法和表示单元格区域的 6 种方法。

9.看看 Excel 的那些常用对象 (续 3)

初步介绍了 `Comment` 对象、`Chart` 对象及相关属性(`ActiveChart` 属性),让读者认识这些对象(包括工作表与图表工作表)并知道怎么在 VBA 代码中表示它们。

10.神奇的句点

介绍了 VBA 中引用对象的点运算符,以及使用点运算符引用对象的 2 条规则。

11.神奇的句点（续）

介绍了 VBA 中使用点运算符引用对象的方法和属性的规则，以及对象的方法或属性的参数的表示方法。

12.再回首，说透对象

使用比喻的方法形象地讲解在 Excel VBA 代码中如何使用对象的方法、属性及其参数。

13.一个简单的 VBA 程序

通过一个简单的 VBA 程序介绍程序的基本结构（Sub 过程和 Function 过程）以及一些语言元素（变量、数据类型、保留字、赋值运算符）。

14.VBA 的数据类型

介绍什么是数据类型，为什么要有数据类型，并列出了 VBA 的基本数据类型（Boolean 型、Byte 型、Integer 型、Long 型、Single 型、Double 型、Currency 型、Decimal 型、Date 型、Object 型、String 型及 Variant 型）。

15.变量和常量

介绍了程序中使用的变量和常量的概念，以及其命名规则和声明方式。

16.VBA 的运算符

介绍了程序中使用的算术运算符：加（+）、减（-）、乘（*）、除（/）、整除（\）、取模（MOD）、求幂（^），关系运算符：等于（=）、大于（>）、小于（<）、大于等于（>=）、小于等于（<=）、不等于（<>）、Is 运算符，逻辑运算符：逻辑非（Not）、逻辑与（And）、逻辑或（Or），字符串运算符（&）、赋值运算符（=），以及运算符的优先级。

17.谈谈对象变量

介绍了如何声明对象变量（Object 类型的变量）、给对象变量赋值，以及使用对象变量的 4 大好处。

18.With ... End With 结构，专为对象而生

通过示例代码的一步一步简化，介绍了 With ... End With 结构的使用。该结构是 VBA 中一种高效处理重复引用对象的方式。

19.学生管理系统开发 V1.0

使用已经学过的知识，初步规划一个简单的应用系统，并使用一些直观的代码实现简单的数据转移功能。

20.用户交互初体验—MsgBox 函数

详细介绍了 VBA 内置的 MsgBox 函数（语法及参数），实现与用户的简单交互：向用户显示信息和获取用户的操作信息。

21.向左走，向右走—使用 If 语句选择

详细介绍了 If 语句的语法结构及多种表现形式。

22.方便的“多选一”结构—Select Case 结构

详细介绍了 Select Case 结构的语法，以及判断表达式的形式。

23.有限次的循环

详细介绍了 `For-Next` 循环语句的语法结构。

24.有条件的循环（1）

详细介绍了 `Do While` 循环语句的语法结构（2 种形式）。

25.有条件的循环（2）

详细介绍了 `Do Until` 循环语句的语法结构（2 种形式）。

26.在对象中循环——For Each-Next 结构的使用

详细介绍了专为对象设计的 `For Each-Next` 循环语句的语法结构。该结构可以在对象组成的集合中循环。

27.看看 VBA 的 Sub 过程和 Function 过程

初步介绍了 `Sub` 过程和 `Function` 过程的基本形式，以及（`Function` 过程的）简单使用方式。

28.简单问答

解答初学者遇到的一些疑惑，包括打开有些 Excel 工作簿时会出现安全警告、保存有些 Excel 工作簿时会出现额外的提示信息、Excel 界面中如何调出“开发工具”选项卡、VBA 代码的放置位置、如何运行 VBA 代码，等等。

在完美 Excel 微信公众号中发送消息：**VBA 基础**，即可获得本电子书文档下载链接和密码。

1. 引子

您自己的电脑,或者留意周围的人的电脑,是不是都装有电子表格软件?可以说,电子表格软件几乎是每台电脑必备的软件之一,更是很多人每天必须使用的软件之一,而 MS Excel 是电子表格软件的代表。

Excel 首先是作为数据处理和分析平台而出现的,然而,自从引入 VBA 后,Excel 为用户提供了二次开发能力,这也使 Excel 产生了质的飞跃,使其更加强大。如果您会使用 Excel VBA,就能更自如地操控 Excel,不仅增强其功能和提高工作效率,而且让她按您的习惯和方式“行事”,让她真正变成属于您自己的个性化数据分析与处理平台。

了解 VBA

VBA 是 Visual Basic of Applications 的简称,与大名鼎鼎的 Visual Basic 语言几乎相同,只是它“寄生”在应用软件内部,可以通过它编写的程序控制主应用程序的方方面面。VBA 也“寄生”在 Excel 内部,只要安装了 Excel 就已经有了 VBA。同样,装了 MS Office 的其它套件后,例如 Word、PowerPoint,也带有了 VBA。此外,一些常用软件如 AutoCAD 也带有 VBA,可以使用它方便地对现有软件进行二次开发。这样,无意中形成一个“VBA 家族”,它们使用同样的语言,因此它们之间可以顺畅地进行“交流”,相互合作,发挥各自的优势,形成强大的整体实力。

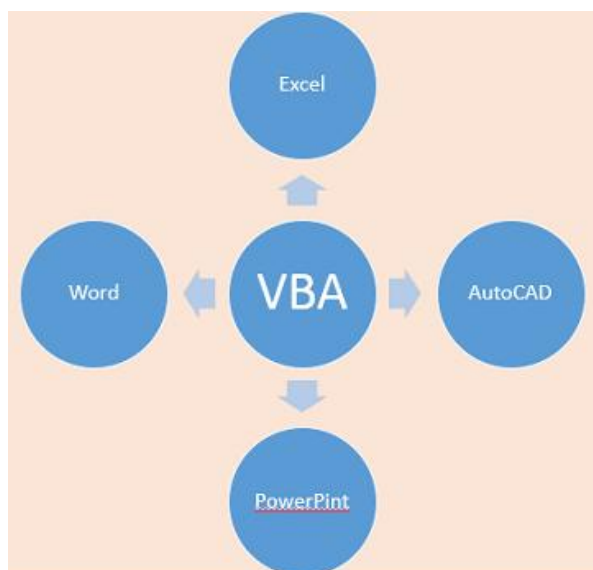


图 1.1

一组 VBA 指令就构成了我们通常所说的“宏”。

实例介绍

为了进一步直观地了解 VBA，抛出自己为方便工作而开发的两个小实例（篇幅所限），这只是冰山的一角。

实例 1：考勤管理系统



图 1.2

在图 1.2 中，左边的数据输入区，只需简单地在单元格中输入 1、2、3 等与相应考勤类型对应的数字，有时需填写弹出的对话框补充信息，即可完成考勤数据的输入。右边的统计区，会自动统计各考勤项。单击上方的汇总按钮，则会在一张

新工作表中详细列出每位员工的考勤记录。同时，会在另外相关联的工作表中自动统计出每位员工的加班费。此外，该工作表与员工绩效考核工作表相关联并运用相关规则，可以评出每名员工相应的得分，进行排名从而确定月绩效系数。（简单地输入基础数据，Excel VBA 自动完成汇总、统计、评价及排序，简化了手工工作，大大提高了效率）

实例 2：小型工程项目管理系统



图 1.3

在图 1.3 所示的工作表单元格下拉列表中选择相应的工程项目，与该项目相关的信息会自动分类并显示在下方，一目了然，便于了解工程项目的有关情况。

需要说明的是，上述实例，在事先经过仔细规划后，编写的代码并不是很多，却实现了满足自己工作需求的很方便的功能。您是不是也有点手痒痒了，哈哈！

编程能力快速提升法

动手做一遍抵得上看一千遍，因此，提升编程能力的唯一方法就是自己输入或编写代码，在编辑器中反复调试，达到自己想要的效果。

- 模仿优秀的程序。首先是照着输入，调试运行，体会其思路 and 过程，看运行结果。其次是改编程序，反复体味。
- 在实践中体验。尝试着解决一些问题，小小成就就会激发更多的实践，积累后会有质的提升。
- 代码与主界面和运行结果对比。这正是 Excel VBA 独特的地方，运行的每一行代码我们都可以在 Excel 中看到代码产生的结果，使我们能更好地理解代码和相对应的 Excel 对象。
- 不怕出错，哪怕是很低级的错误，重要的是从这些错误中汲取教训，这正是学习的好时机。
- 坚持。只有坚持，才能在已有的成果上获得更多的成果。在学习过程中，遇到不懂的，可以先跳过去，以后回头看，可能就已经理解了这些原来不懂的内容。当您坚持下去，若干时间后，蓦然回首，发觉自己已是“高手”。

关于《Excel VBA 解读》

本系列文章就是为您解读 Excel VBA，为有兴趣深入学习 Excel 二次开发的朋友打下基础，让您逐步具备 Excel 开发技能，充分发挥 Excel 的效率。

为便于您阅读，文章都不会很长，正文通常会控制在 1000~1500 字以内，并配有图片作为所讲述内容的解释，力图深入揭示 Excel VBA 开发的奥妙。无论您是想要了解和学习 Excel VBA，还是已经有了一定的 VBA 基础，都希望您能享受阅读的乐趣，从中受益。

那么，就让我们开始解读 Excel VBA 之旅吧。

2. Excel 中的加强版“录像机”

虽然现在的电视基本上都有回放功能，但在过去为了避免错过喜欢的电视节目，我们常常会使用录像机把节目录制下来，以便于有时间时再看。在 Excel 中，也有一个类似录像机的功能，那就是“录制宏”。

使用“录制宏”功能，Excel 可以把您在工作表中的每个操作命令保存下来，之后可以重复这些命令。下面，我们以一个极简单的示例来说明。

录制宏

单击“开发工具”选项卡中的“录制宏”，如下图 2.1 所示。



图 2.1

弹出下图 2.2 所示对话框。

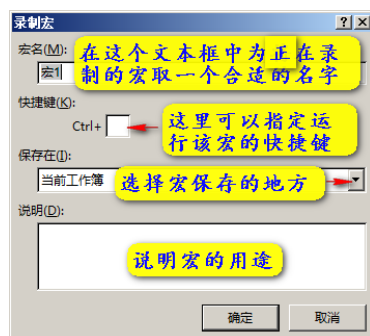


图 2.2

保持默认设置，单击“确定”。这时，Excel 开启录制功能。

单击“加粗”和“倾斜”，再单击“停止录制”按钮（如下图 2.3 所示）。至此，完成了一个简单的宏的录制过程。注意，录制完后一定要按“停止录制”按钮，否则录制工作会一直持续下去。（技巧：按 Alt+F11 组合键即可快速调出 VBE 界面）

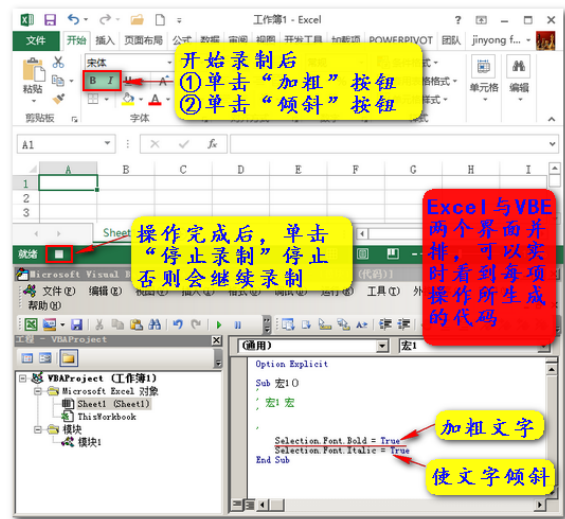


图 2.3

运行宏

选择要执行宏命令的单元格，本例中为 B2。单击“开发工具”选项卡中的“宏”命令，弹出下图 2.4 所示的宏对话框，列出了刚才录制的宏名称。（技巧：按 Alt+F8 组合键可以快速弹出这个对话框）

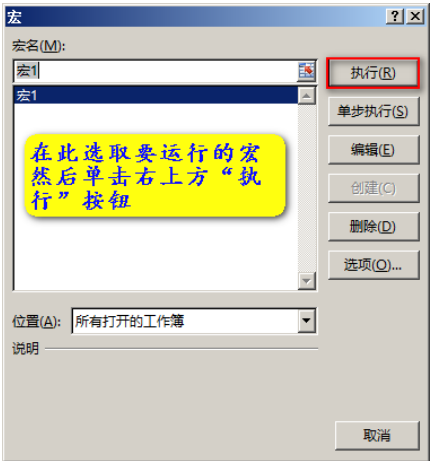


图 2.4

选择宏名后，单击“执行”按钮，自动将 B2 单元格中的文字变为加粗斜体，如图 2.5 所示。

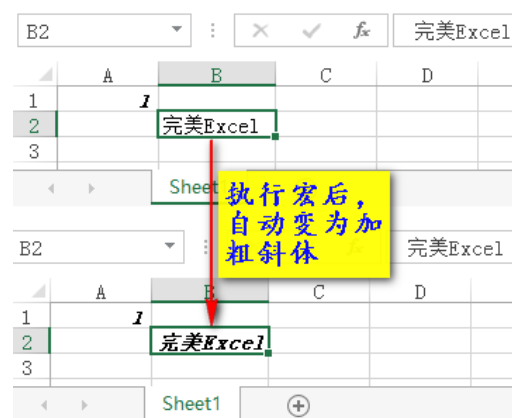


图 2.5

说明：我之所以把这个功能称为加强版的“录像机”，是因为它录制的操作命令，不仅可以原封不动地执行录制时所在单元格的操作（“回放”功能），而且可以对其它单元格甚至其它工作表中的单元格进行相同的操作（“克隆”功能）。

录制宏的三大用处及好处

①**用处：**组合各选项卡和对话框中的一系列命令（例如上例中录制的 2 个命令）、以及自动输入一些模板化的数据。

好处：提高效率。将大量重复性的操作录制下来，以后只需一个命令就能执行这些操作，避免在选项卡和对话框中来回重复操作，以及重复输入一些模板化的数据。

②**用处：**极好的用于学习 Excel VBA 的工具

好处：录制宏后，我们可以通过研究其代码学习 VBA。尤其好的是，如图 3 所示，将 Excel 工作表界面和 VBE 编辑器界面并排排列，可以在录制时一边操作一边查看生成的代码，对照学习体会。

③**用处：**获取代码的源泉

好处：Excel 的对象、属性繁多，将它们全部记住不可能也无必要。在编写代码时，通过录制相关的代码，找到所需的对象、属性，稍作修改或调整，即可运用到自己的程序代码中。

录制宏的技巧

在录制宏前，最好先演示下要录制的操作，制定操作步骤，再开始录制，避免无用的操作生成大量无关的代码。

3.VBA 代码之家

代码在哪儿

无需单独安装代码编辑器，Excel 都已经给您考虑好了，在安装 Office 时，就自带了代码编辑器，我们将其简称为“VBE”。

如下图 3.1 所示，在 Excel 的“开发工具”选项卡中，单击“Visual Basic”按钮，即可进入 VBE 界面。



图 3.1

最简单快捷的方式是，直接按 **Alt+F11** 组合键，立即打开 VBE 界面。（重复按 **Alt+F11** 键，可以在 Excel 主界面和 VBE 界面之间来回切换）

在没有录制宏或者没有插入代码模块时，如下图 2 所示，可单击菜单左侧的 Excel 图标回到 Excel 主界面。在图 3.2 中，右侧区域曾现灰色，这里就是存放 VBA 代码的地方，不仅可以查看宏录制的代码，也可以在其中自己编写代码，以及修改和调试代码。

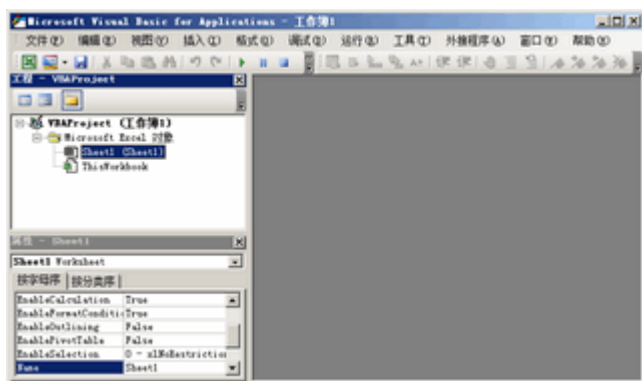


图 3.2

当录制宏后，代码就会自动存放在 VBE 中。下图 3.3 显示的是查看《Excel VBA 解读》系列 2 中录制的宏代码的过程。

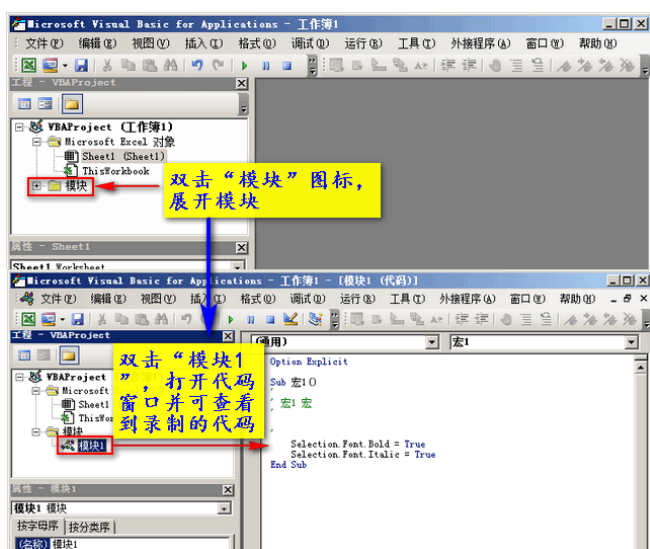


图 3.3

初识 VBE 界面

在刚打开 VBE 时，如果没有录制代码，那么 VBE 界面如上图 3.2 所示，此时可以直接在 VBE 中插入空代码模块，调出代码窗口并在其中编写代码。单击菜单栏中的“插入—模块”，插入代码模块并打开代码窗口，也可以直接按 F7 键调出代码窗口。



图 3.4

通常的 VBE 界面如上图 3.4 所示，分两大区：上面是“菜单栏和工具栏区”，下面是“窗口区”。其中，“工程资源管理器窗口”列出工作簿和工作表对象、模块、类模块、用户窗体清单，用于组织和管理代码模块。“属性窗口”列出了工程资源管理器中所选模块对象的各种属性及默认设置值，可以在这里设置或修改相应的属性值。“代码窗口”是我们平时编写、修改和调试代码的地方，也是我们主要的工作场所。当然，还有其它方便调试代码的窗口，以后将逐一介绍。

提示：可以在“视图”菜单中方便地调出各种窗口。

输入代码

初步了解 VBE 界面后，让我们试着输入简单的代码体验一下。

在代码窗口中输入下面的代码，如图 3.5 所示。

```
Sub 测试()

    Selection.Value = "Excel VBA"

End Sub
```

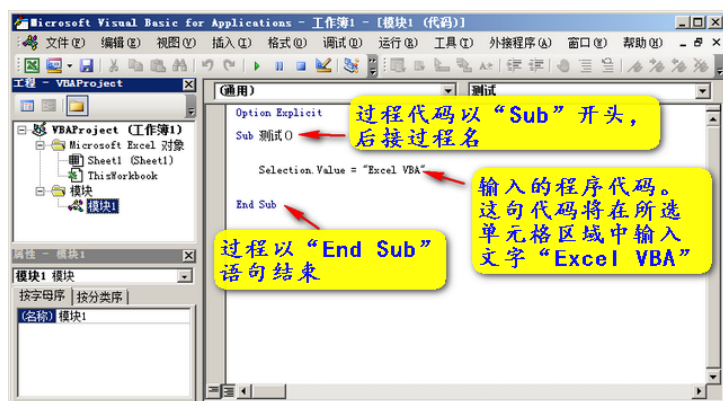


图 3.5

在 VBA 中，过程以关键字“Sub”开头，以“End Sub”结尾。在“Sub”和“End Sub”之间输入程序代码。Sub 后是过程名，使用英文名或中文名均可，但要符合 VBA 规定的命名规范。过程名将会出现在“宏”对话框中，如下图 7 所示。

上述程序代码的意图是：在工作表所选单元格或单元格区域中输入“Excel VBA”。

执行代码

有三种方法执行代码：①单击菜单栏中的“运行——运行子过程/用户窗体”命令；

②单击工具栏中的“运行子过程/用户窗体”按钮，如下图 3.6 所示；



图 3.6

③直接按 F5 键。

在执行上述任一操作命令时，若光标不在过程内部，则会弹出“宏”对话框，在其中选择要运行的过程名后，单击“运行”即可，如图 3.7 所示。因此，最好将光标置于要运行的过程内部，即“Sub”和“End Sub”之间，再操作上述命令执行代码。

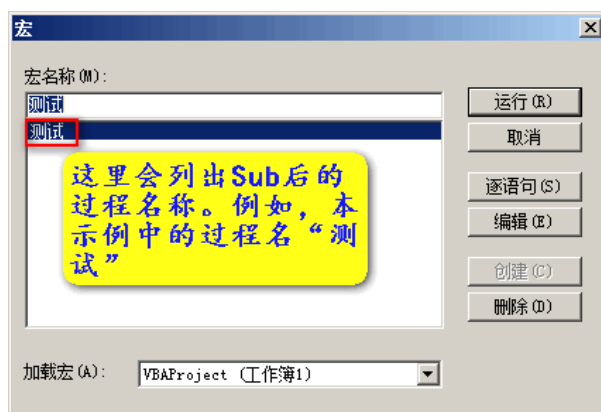


图 3.7

执行图 3.5 中的程序代码后，按 Alt+F11 组合键返回 Excel 主界面，将看到在工作表所选单元格中已经自动输入了文字“Excel VBA”。

示例：使用“立即窗口”调试或执行代码

在 VBE 中，单击菜单栏“视图——立即窗口”，或者按 Ctrl+G 组合键，调出立即窗口，如下图 3.8 所示。

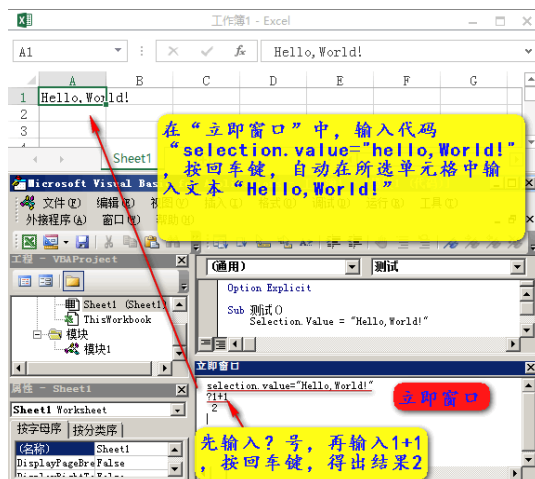


图 3.8

在“立即窗口”中，可以即时执行代码并看到结果。如图 3.8 所示，在立即窗口中输入执行操作的代码后，按回车键，在 Excel 工作表中将得到相应的结果。如果需要进行计算结果操作，则需要先输入一个问号（？），接着输入求值表达式，按回车键后，即可得到结果。

在 VBE 中，输入和执行代码都很简单，试试看！

4. 如影随形的帮助

在阅读和编写代码的过程中，难免会遇到疑惑，如果适时寻求帮助，就会较快地解决问题，也能得到学习。

官方帮助

在 Excel 的各个版本中，Microsoft 都为您准备了丰富的帮助文档资源。下图 4.1 为 Excel 2010 版本中的 VBE 与帮助相关的界面。



图 4.1

单击图 4.1 中①②标识处的帮助按钮或菜单命令，将会调出下图 4.2 所示的 Excel 帮助系统。如果在图 4.1 中③标识处输入关键词或者在下图 4.2 中的搜索框中输入关键词，则会调出相关的帮助文档列表，可以打开相应的文档查看。



图 4.2

Excel 2013 改进了帮助系统，在 VBE 中取消了右上角的帮助搜索框，并将帮助文档移到了 Microsoft 的网站中，使帮助资料更新更丰富。单击图 1 中①②标识处的帮助按钮或菜单命令，直接进入网上帮助系统，如下图 4.3 所示。



图 4.3

无论是 Excel 自带的帮助系统还是网上帮助系统，都是一份丰富的学习资源库和权威资料，研究和学习它们，将毫无疑问地会促进 Excel VBA 编程水平的快速提高。

VBE 中的 F1 键和 F2

即时提供光标所在代码的帮助

在 VBE 中，按 F1 键，可以快速调出光标所处代码的帮助信息。如下图 4.4 所示，将光标置于代码中代表选择对象的 Selection 词内，按 F1 键，在网页中会打开

关于 Selection 的帮助信息。

注：在 Excel 2010 及以前的版本中，会在 Excel 自带的帮助系统中（如上图 4.2 所示）打开相应的帮助信息。

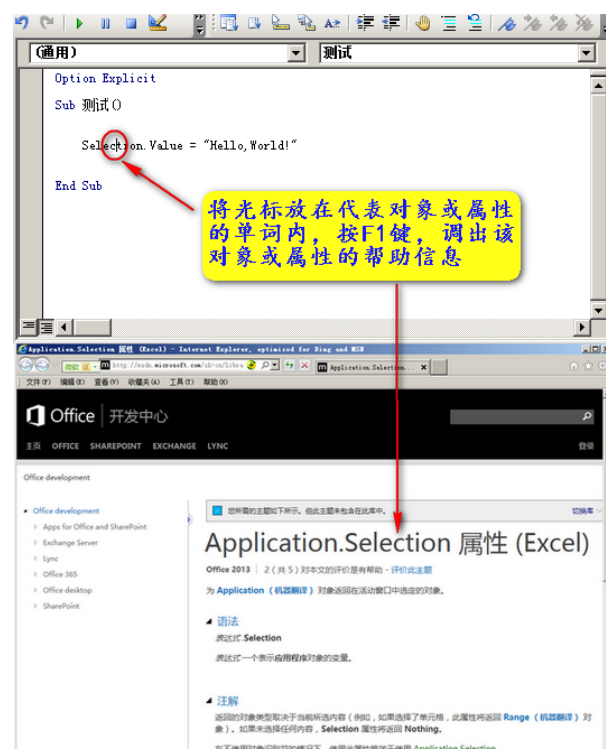


图 4.4

对象浏览器

在 VBE 中，直接按 F2 键或者单击工具栏中的“对象浏览器”按钮，即可调出下图 4.5 所示的对象浏览器窗口。

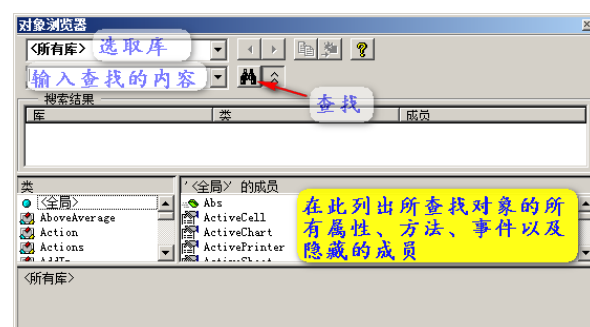


图 4.5

在对象浏览器中，可以搜索对象的属性、方法、事件，甚至隐藏的成员。在“类”条目下列出了所有的对象，如果选择其中某一对象，则在右侧列出该对象的属性、方法和事件。

在代表对象及其属性、方法和事件的窗口中，选择某一条目，单击右上方的问号（?）图标，则会打开关于该条目详细的帮助信息。

当然，Excel 提供的辅助帮助信息还有很多，这靠您在实践中发掘，我也会在会面的文章中及时提到。

网上资源

碰到问题或疑惑时，我们首先可能想到的是谷歌或百度。搜索后，会出现大量的相关网站或文档信息。然而，专业的网站应该是您寻求帮助或学习查找资源的首选。

下面介绍两个专业的网站：

ExcelHome

网址：<http://club.excelhome.net/forum.php>

国内最早开办的 Excel 专业网站之一，汇集了大量的 Excel 专业技术贴子和 Excel 精英，近几年策划出版的一系列 Excel 图书也颇受读者欢迎。

VBAX

网址：<http://www.vbaexpress.com>

国外非常好的一个 Excel VBA 专业网站，活跃着很多 Excel 编程高手，发出的求助贴，能够很快得到专业的回复。（要求您的 E 文有一定的基础哟，呵呵！）

当然，Excel 专业网站或论坛有很多，您可以从前面两个网站中，找到很多其他的较好的 Excel 专业网站。

另外，自认为本人的微信公众号也是非常好的学习的地方，几年来我学习和研究 Excel 的心得都汇聚在此，至今已推送 300 余篇技术文章，并且还在不断的更新哟。

完美 Excel 微信公众号：*excelperfect*

5. 认识对象和 Excel 对象模型

在开始进一步讲解之前，了解一些与 Excel VBA 编程相关的基本概念将有助于知识的学习和理解。

对象及其属性和方法

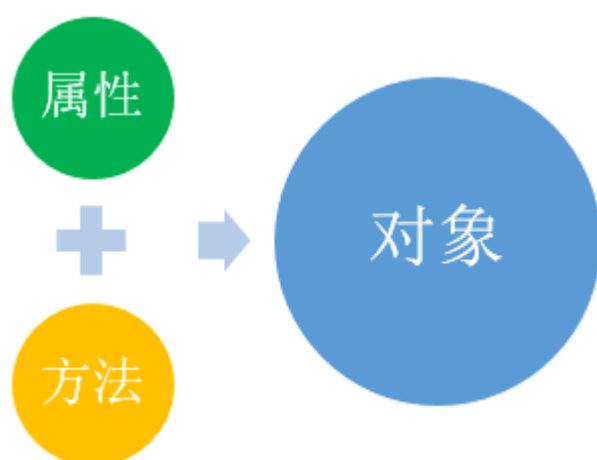


图 5.1

对象

在我们生活的世界中，每天都会看到各种各样的对象，对象无处不在。例如，家里的桌子、冰箱、电视机，马路上跑的汽车，甚至我们自己，这些都是对象。我们使用为桌子、冰箱、电视机、汽车.....来代表相应的对象。

同样，我们可以把 Excel 工作簿当作一个小小世界，它里面也存在各种对象。例如，工作簿本身就是一个对象，使用 `Workbook` 来代表它；而工作表对象则使用 `Worksheet` 来代表；单元格区域使用 `Range` 来代表。

对象的属性

在日常生活中，我们通常会描述物体对象的特征，例如汽车的品牌、颜色、长度、高度等，这些都称为该物体的属性。

Excel 中的对象也或多或少地存在着描述它们特征的属性。例如，**Workbook** 对象有一个 **Name** 属性，代表工作簿的名称；有一个 **Author** 属性，代表工作簿的作者名。**Range** 对象有一个 **RowHeight** 属性，可以设置或者获取单元格区域的行高。

对象的方法

方法就是对象所能做的动作，或者说我们可以对对象做的动作。例如，汽车的行驶、刹车等动作。

在 Excel 中，如果我们要在工作簿中添加一个工作表，那么就要对 **Worksheet** 对象执行添加动作，使用 **Add** 代表，**Add** 方法就是 **Worksheet** 对象的一个方法。

在 Excel 编程开发中，我们就是通过使用 VBA 代码操作 Excel 里各种对象的属性、方法，以实现我们的需要的功能。因此，应该理解对象及其属性、方法的基本概念，熟悉 Excel 常用对象。（后面的系列文章中，我们将通过图形化的方式加速您对 Excel 常用对象及其属性和方法的熟悉）

集合

在我们小区，地下车库里所有的汽车构成了一个汽车集合；地面上所有单元楼房构成了小区楼房集合。集合是相同对象的集，其本身也是一个对象，包含相同的相关对象。

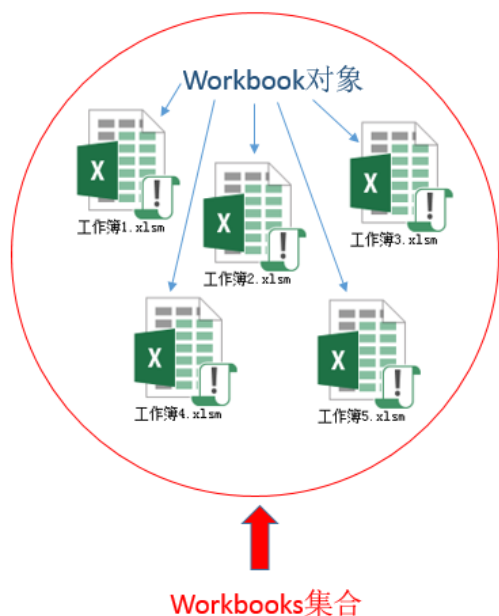


图 5.2

如图 5.2 所示，在 Excel 中，所有打开的工作簿（Workbook 对象）构成了工作簿集合（Workbooks 对象），某工作簿中的所有工作表（Worksheet 对象）构成了一个工作表集合（Worksheets 对象）。可以看出，集合对象名称是其所包含对象的复数形式。

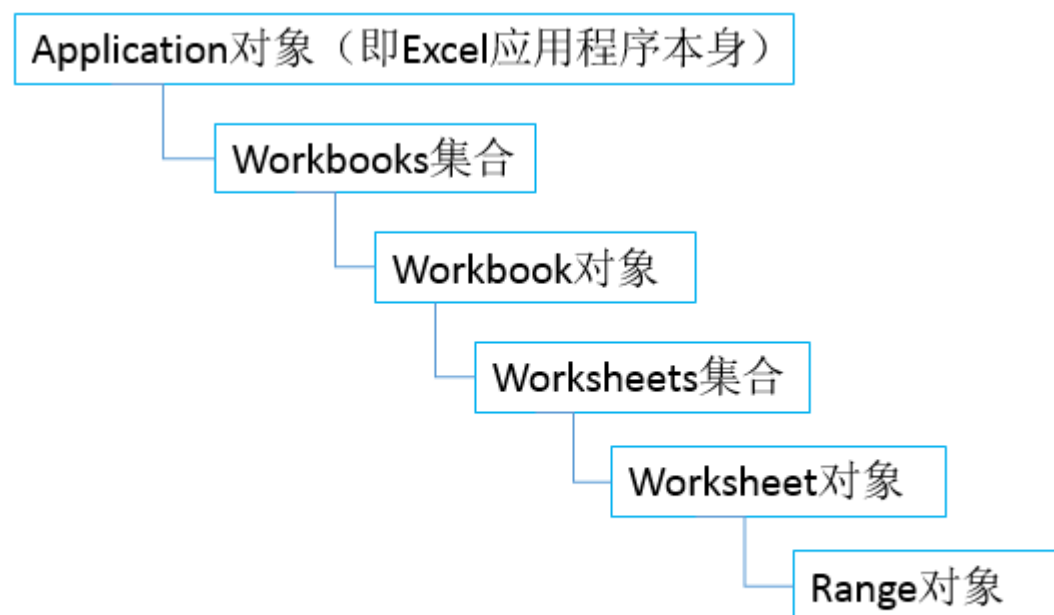
对象模型

前面我们所说的生活中的各种对象，其本身也是由对象组成。例如将汽车逐步分解，由发动机、底盘、车身和电气设备构成，它们又由方向盘、刹车、车轮……等部件组成，这些部件分别又由更小的部件组成，这样直至分解到每个螺丝。所有这些部件按汽车组成组织起来就形成了组成汽车的对象模型。

同样地，Excel 中的对象也是经过层层组织，构成了 Excel 对象层次模型：

Application 对象代表 Excel 应用程序本身，有一个工作簿集合对象（Workbooks 集合对象），包含所有打开的工作簿对象（Workbook 对象），而每个 Workbook 对象有一个工作表集合对象（Worksheets 集合对象），包含工作簿里所有的工作表对象（Worksheet 对象），而每个 Worksheet 对象又包含单元格区域对象（Range 对象）。

Excel 对象模型层次结构如下图 5.3 所示。



Excel对象模型层次结构

图 5.3

6. 看看 Excel 的那些常用对象

下午，我们按约好的时间去足球场踢球。这里，足球就是我们要操作的对象，我们要踢足球，前提是先要找到一个能够踢的足球。

同样，在 Excel 中，我们通过使用 VBA 来操控 Excel 对象，从而实现对 Excel 的全面控制以及在 Excel 界面中获得想要的结果。然而，我们最先要做的事情是找到那个要操控的 Excel 对象。这些对象就像我们前面提到的足球一样，有其特定的名称和表示方式。

下面，我们就来认识 Excel 的一些常用对象，知道怎么样表示它们，这是进入 Excel VBA 编程大门的基础。

接下来的内容我们将依次简单地认识 Excel 的下列常用对象：

- Application 对象
- Workbook 对象
- Window 对象
- Worksheet 对象
- Range 对象
- Comment 对象
- Chart 对象
- 返回常用对象的一些属性：ThisWorkbook 属性、ActiveWorkbook 属性、ActiveWindow 属性、ActiveSheet 属性、Selection 属性、ActiveCell 属性、Cells 属性、Rows 属性、Columns 属性、ActiveChart 属性

Application 对象

顾名思义，Application 对象即代表应用程序本身。在 Excel 中，Application 对象就代表 Excel 应用程序。它是 Excel 对象模型的根，所有的 Excel 对象都是从它这里开始逐层扩展，开枝散叶。

Workbook 对象

Workbook 对象代表工作簿，所有 Workbook 对象组成 Workbooks 集合。换句话说，单个的 Workbook 对象是 Workbooks 集合中的一个成员。我们可以在 Workbooks 集合中指定工作簿的名称来表示要处理的工作簿，如下图 6.1 所示。例如，如果我们要在名为“工作簿 1”的工作簿中执行操作，那么在代码中就使用：

```
Workbooks("工作簿 1.xlsm")
```

来代表该工作簿。也可以使用索引号来引用相应的工作簿，此时就需要知道该工作簿在 Workbooks 集合中的索引号，如果工作簿 1 的索引为 1，那么在代码中可以使用：

```
Workbooks(1)
```

来代表该工作簿。

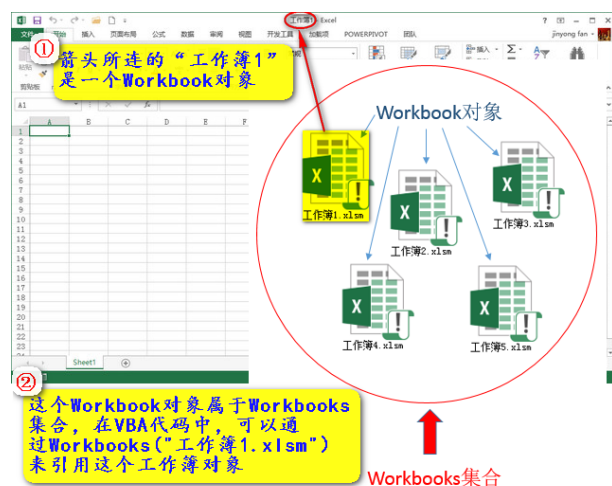


图 6.1

ThisWorkbook 属性

我们总会在 VBE 的工程资源管理器中看到“ThisWorkbook 对象模块”，如下图 6.2 所示，它代表当前工作簿。Application 对象的 ThisWorkbook 属性与其名称相同，

使用该属性可以获取包含正在运行的代码的工作簿对象。

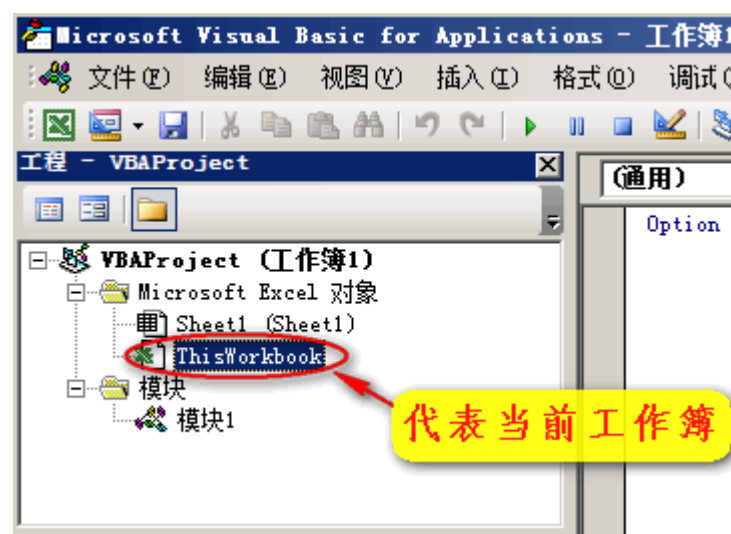


图 6.2

ActiveWorkbook 属性

ActiveWorkbook 属性也属于 Application 对象。使用该属性可以获取当前工作簿对象。

Window 对象

Window 对象代表窗口。如下图 6.3 所示，打开两个工作簿，分别为：工作簿 1 和工作簿 2，每个工作簿都是一个窗口，所有窗口对象组成 Windows 集合。我们可以在 Windows 集合中指定窗口名称来表示要处理的窗口。例如，如果我们要在工作簿 1 所在的窗口中执行操作，那么可以在代码中使用：

```
Windows("工作簿 1.xlsm")
```

来代表工作簿 1。同理，在代码中使用：

```
Windows("工作簿 2.xlsm")
```

来代表工作簿 2。

当然，也可以使用代表窗口的索引号来引用相应的工作簿。例如，如果工作簿 1 所在的窗口索引号为 1，那么可以使用代码：

```
Windows(1)
```

来代表工作簿 1。

说明：当前活动窗口始终是 Windows(1)。

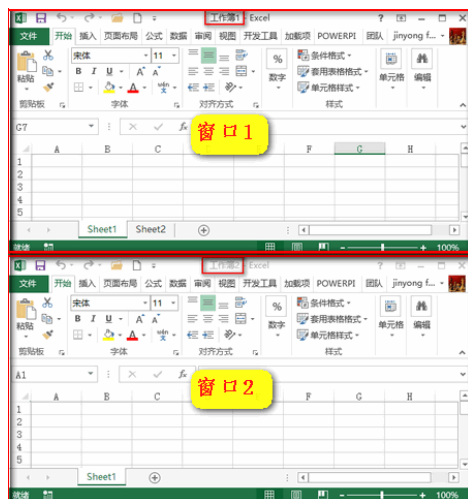


图 6.3

探讨：在单个的工作簿中，也存在 Windows 集合，但这似乎对于 Excel 2013 版来说已不重要了？

ActiveWindow 属性

ActiveWindow 属性属于 Application 对象。使用该属性可以获取当前活动窗口。

小结

观察前面代表 Workbook 对象和 Window 对象的代码表示，可以看出有两种方式：

- (1) 在集合中使用名称来表示具体的对象。
- (2) 在集合中使用索引来表示具体的对象。

我们将其归纳为通用的表达式：

集合(名称)	或	集合(索引)	
Workbooks("工作簿1.xlsm")		Workbooks(1)	→ 都可表示工作簿1
Windows("工作簿1.xlsm")		Windows(1)	→ 都可表示工作簿1所在的窗口1

Excel VBA代码中表示具体的对象的方式

图 6.4

说明：文中所说的“当前工作簿”、“当前活动工作簿”、“当前窗口”、“当前活动窗口”等，表示正在使用中的工作簿、正在使用中的窗口，或者是处于最上面的工作簿或窗口。

7. 看看 Excel 的那些常用对象（续 1）

对于熟悉 Excel 对象模型的人来说，也许会觉得到目前为止所讲的内容都太小儿科了。确实，我也有这种感觉。讲解这些基础的东西对于有一定 Excel VBA 编程功底的人来说，太枯燥了！所以，多些图文，多些趣味，赶走枯燥。

同时，既然是完整的系列文章，我还是得从最简单的东西开始，基于不熟悉 Excel VBA 的朋友，从基本的知识点讲解，由浅入深，慢慢地让他们在无形中就会步入 Excel VBA 编程技术的殿堂。

另一方面，作为一个完整的系列，也应该从基础讲起，慢慢地开始涉及到更深入更实际的知识和实践。这会给熟悉 Excel VBA 的朋友带来一个完整的框架，为他们理清思路的同时，也会给他们带来原来忽视的东西。

随着系列的深入，后面会越来越精彩！

下面继续讲解上一节中未讲完的内容：Excel VBA 编程中常常使用的那些对象到底是什么，如何在代码中表示它们。

Worksheet 对象

Worksheet 对象代表工作表。工作簿中的每个工作表都是一个 Worksheet 对象，所有 Worksheet 对象构成了 Worksheets 集合。

我们使用下面的这一张图来完整解析 Worksheet 对象，如下图 7.1 所示。

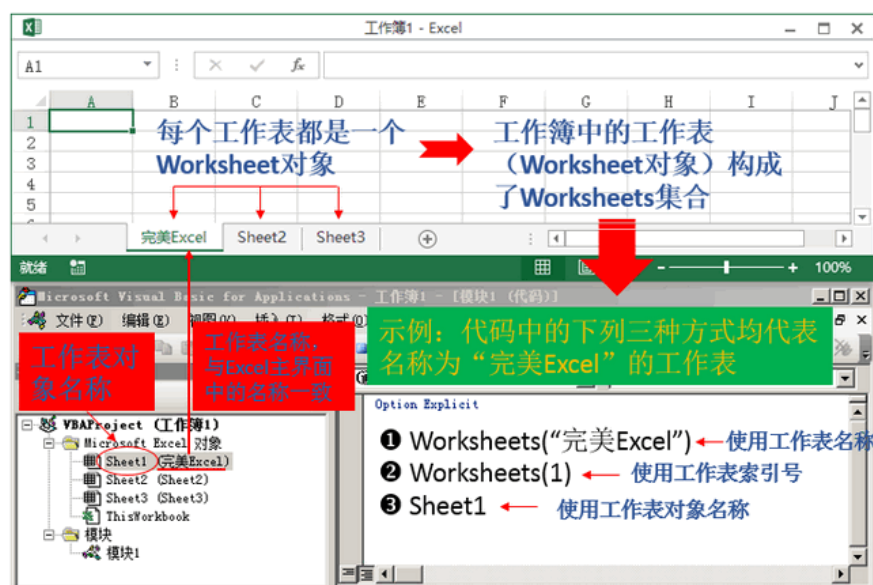


图 7.1

工作表的名称

如图 7.1 所示，上面是 Excel 主界面，下面是 VBE 界面。新建工作簿后，Excel 为我们提供了 1-3 个默认工作表（Excel 2013 中默认为 1 个工作表）。工作表名称默认为 Sheet1、Sheet2、Sheet3。我们可以在 Excel 中更改工作表的名称，例如图 1 中在工作表名称中双击，将默认的名称为“Sheet1”的工作表改名为“完美 Excel”。

再看看 VBE 界面，在左侧的工程资源管理器的“Microsoft Excel 对象”文件夹下，Excel 有几个工作表，就会有几个相对应的工作表代码模块，图 1 中是 3 个。在代码模块标识中，例如 Sheet1（完美 Excel），左侧的“Sheet1”是相应工作表的对象名称，右侧括号中的“完美 Excel”是工作表名称，也就是我们在 Excel 主界面工作表标签中看到的名称。

也就是说，一个工作表有两个名称，一个称为“工作表名”，在 Excel 主界面中可以看到和修改；另一个称为“工作表对象名”，只能在 VBE 界面中看到，也可以在 VBE 中修改。

如何在代码中表示某工作表

按照前一篇文章中归纳出来的公式，要表示某工作表，一般只需要在 Worksheets 集合中指定工作表名称或索引号即可，例如要在代码中指定“完美 Excel”工作表，可以：

1、使用工作表名称：

```
Worksheets(“完美 Excel”)
```

2、使用工作表索引：

```
Worksheets(1)
```

此外，由于工作表还有一个对象名称，因此还可以使用其对象名来指定“完美 Excel”工作表，即：

3、使用工作表对象名：

Sheet1

ActiveSheet 属性

ActiveSheet 属性是 Workbook 对象的一个属性。使用 ActiveSheet 属性可以获取代表当前正在使用的工作表对象。

8. 看看 Excel 的那些常用对象（续 2）

在学习的过程中，我们往往会忽视细节，甚至对有些关键细节也一带而过，结果就会造成一知半解或不懂装懂。虽然过多地纠缠于细节，会耗费很多的精力，有时也会陷于不能自拔而走进死胡同，但是认识一些细节，有助于对所学知识的更深入的理解，更会打下坚实的基础。那么，闲话少说，下面我们就来讲解单元格在 Excel VBA 代码中的表示。

在工作表中，我们面对的或者最主要的工作区域就是一个个单元格了，它们在 Excel VBA 中是使用 Range 对象来表示的。

Range 对象

Range 对象代表单元格或单元格区域。广义地说，单元格区域包括：①单个单元格；②包含连续的或者不连续的多个单元格；③一行或多行；④一列或多列。这些在代码中都可以用 Range 对象来表示。

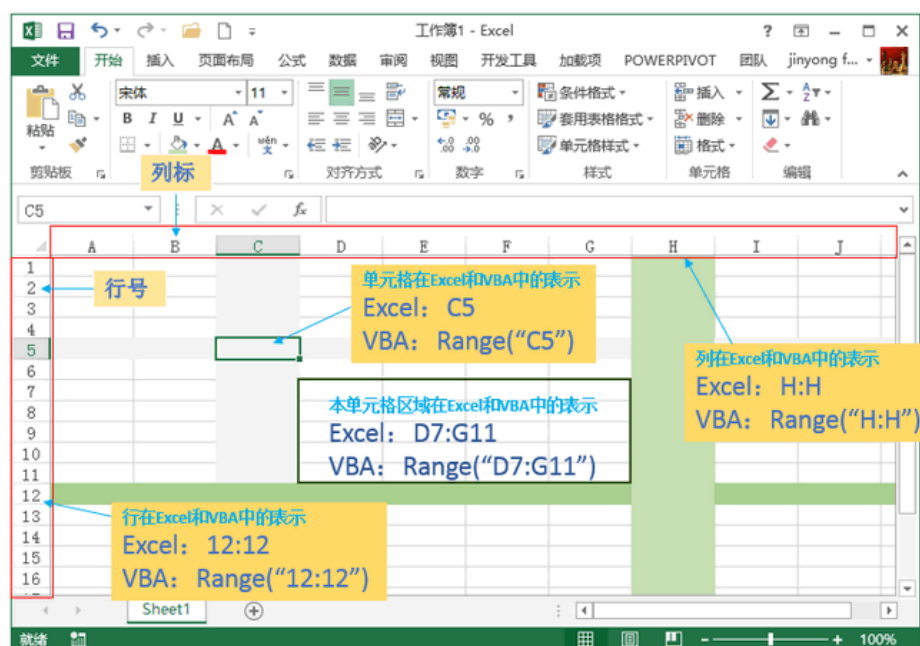


图 8.1

如上图 8.1 所示，在 Excel 中，单元格由行号列标来表示，例如单元格 C5，同样，单元格区域、行、列也由相应的行号列标来表示。

单个单元格、单元格区域、整行、整列等都是 Range 对象。在 VBA 中，由 Range 对象来表示相应的单元格区域。具体的说，如上图 1 所示：

Range(“C5”)

表示当前工作表中的单元格 C5。

Range(“D7:G11”)

表示当前工作表中的单元格区域 D7:G11。

Range(“H:H”)

表示 H 列。

Range(“12:12”)

表示第 12 行。

在图 8.1 中，我们还可以看出这样的规律，在工作表（公式）中表达的单元格或单元格区域，要在 VBA 代码中表示，只需将其放置在 Range 后面的括号中，并加上引号。

ActiveCell 属性

ActiveCell 属性是 **Application** 对象的一个属性。使用该属性可以获取当前工作表中的活动单元格，也就是正在编辑或正准备编辑的单元格。例如图 1 中，如果光标在单元格 C5 内，则可以直接在代码中使用

```
ActiveCell
```

来代表单元格 C5。

Cells 属性

Cells 属性是 **Application** 对象的一个属性。使用该属性可以获取当前工作表中的所有单元格。例如，上图的工作表 Sheet1 中，可以在代码中使用

```
Cells
```

代表该工作表中的全部单元格。

Selection 属性

Selection 属性是 **Application** 对象的一个属性。使用该属性可以获取当前工作表中所选定的单元格或单元格区域。例如，上图 8.1 中，如果光标位于单元格 C5，那么可以在代码中使用

```
Selection
```

来代表单元格 C5。

如果在工作表中选定了单元格区域 D7:G11，那么可以在代码中使用

```
Selection
```

代表该工作表中的单元格区域 D7:G11。

小结

综上，我们来看看在 VBA 代码中表示单个单元格和由多个单元格组成的单元格区域的表示方法。

1、在 VBA 代码中表示单个单元格，如下图 8.2 所示。

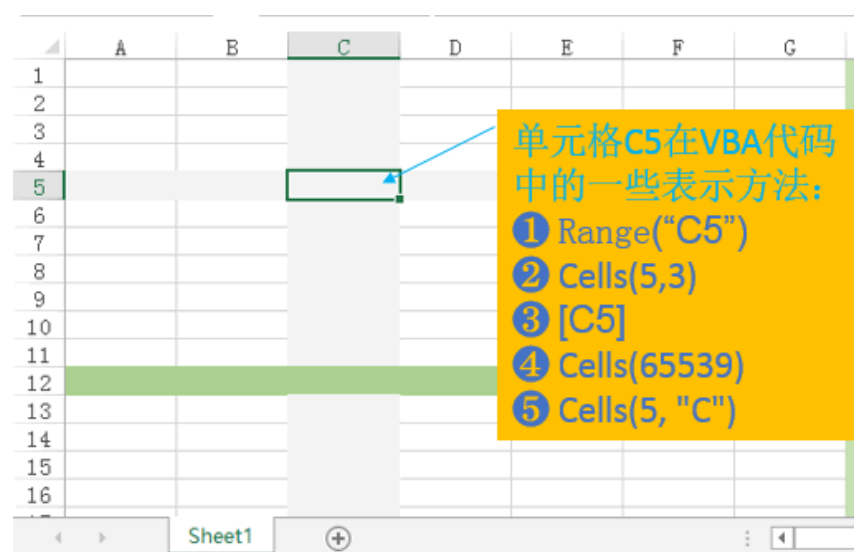


图 8.2

此外，如果光标正处在单元格 C5 中，还可以使用下列代码之一：

⑥ ActiveCell

⑦ Selection

2、在 VBA 代码中表示单元格区域，如下图 8.3 所示。

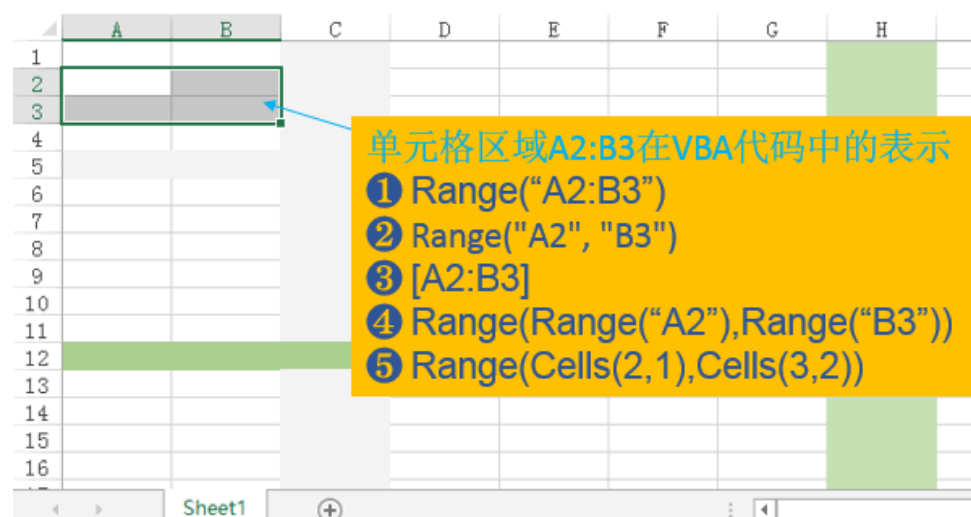


图 8.3

此外，如果在工作表中或之前运行的代码中已经选定单元格区域 A2:B3，还可以使用代码：

Selection

Rows 属性

Rows 属性是 Application 对象的一个属性。使用该属性可以获取工作表或所选定区域的行。

例如图 1 中的第 12 行，还可以使用下面的代码表示：

```
Rows(12)  
Rows("12:12")
```

由第 2 条代码可以推广，如果要获取选定的第 9 行到第 12 行，则可以使用下面的代码：

```
Rows("9:12")
```

Columns 属性

Columns 属性是 Application 对象的一个属性。使用该属性可以获取工作表或所选区域的列。

例如图 1 中的第 8 列，即 H 列，还可以使用下面的代码表示：

```
Columns("H")  
Columns(8)
```

要获取多列，例如第 8 列和第 9 列，可以使用下面的代码表示：

```
Columns("H:I")
```


9. 看看 Excel 的那些常用对象（续 3）

记得上学的时候，老师经常会在作业本上的空白处写一些评语，或者是对某段话的评价，或者是表扬做得认识，等等。Excel 工作表中的批注类似于这些评语，让我们能够为单元格添加评语，为单元格传递更丰富的信息。

Comment 对象

Comment 对象代表单元格批注。如下图 9.1 所示，单击“审阅”选项卡中的“新建批注”命令，为单元格 C5 添加一条批注。这条批注就是一个 Comment 对象。



图 9.1

工作表中所有的 Comment 对象组成了 Comments 集合。如果图 1 中的批注是工作表中的第一个批注，那么在代码中可以使用：

```
Comments(1)
```

代表这个批注。

Chart 对象

Chart 对象代表工作簿中的图表。可以是嵌入在工作表中的嵌入式图表，也可以是单独的图表工作表，如下图 9.2 所示。

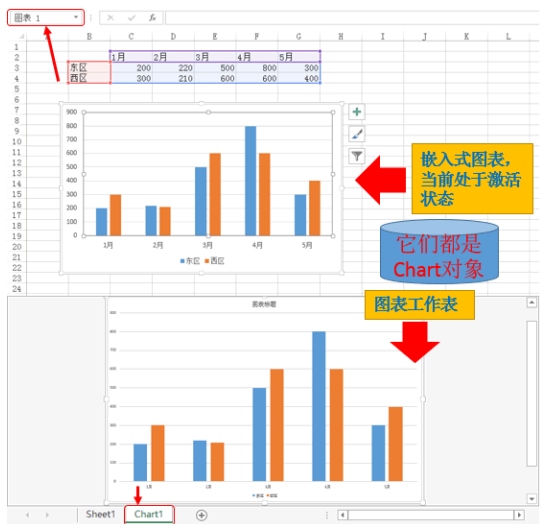


图 9.2

如果是嵌入式图表，则需要通过对象模型层层引用来表示该图表（如下图 9.3 所示的对象层次模型），引用方法将在下篇文章中详述。

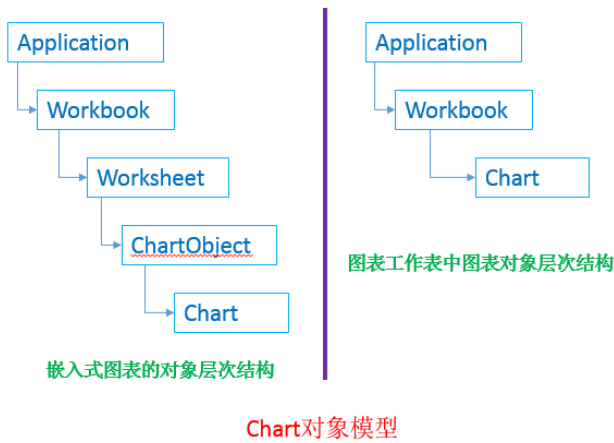


图 9.3

由于所有的图表工作表对象组成了 Charts 集合，因此可以在代码中使用

```
Charts(“Chart1”)
```

来代表图表工作表 Chart1。

拾遗——工作表与图表工作表

在工作簿中，有两类工作表，即工作表和图表工作表，如图 9.2 中的“Sheet1”工作表和“Chart1”工作表。工作表即我们通常在其中进行数据编辑的地方，而图表工作表专门放置单独的图表。

在 VBA 中，**Sheets** 集合包含工作表和图表工作表。这表明，我们前面介绍的 **Worksheet** 对象是 **Sheets** 集合中的成员，可以使用 **Sheets** 集合来表示某 **Worksheet** 对象。

当然，在代码中也可以使用 **Sheets** 集合来表示图表工作表，如下：

```
Sheets(“Chart1”)
```

也表示图表工作表 Chart1。

ActiveChart 属性

ActiveChart 属性属于 **Workbook** 对象的一个属性。使用该属性可以获取当前活动图表，可以是嵌入式图表或者图表工作表。例如，如果嵌入式图表为当前选定的图表，如图 2 所示，那么可以在代码中使用

ActiveChart

来表示当前选定的图表。

好了！到目前为至，通过将 Excel 界面与 VBA 代码表示相比较，我们很清晰地认识了一些常用的对象。相当于是认识了一些基本的对象名称，在 VBA 中的代码是什么以及在 Excel 中是指什么。

下面，我们来看看 VBA 通过什么方式来对这些对象进行操作。

10. 神奇的句点

阳光花园是一个很大的住宅小区，布置有 25 栋楼房，每栋楼由高 18 层的 4 个单元楼组成，每个单元楼住有 2 户，每户按楼层编有门牌号。梦梦同学住在 9 栋 4 单元 601 室。

今天，老师要到梦梦家里家访。进入阳光花园小区后，老师要先找到 9 号楼，再找到 4 单元，乘坐电梯到 6 楼，找到 601 室，就到了梦梦家里。老师的路线图如下图 10.1 所示。

阳光花园 → 9号楼 → 4单元 → 601室

图 10.1

简化一下表示，我们用句点来代替图中的箭头号，那么找到 601 室就可以表示为：

阳光花园.9 号楼.4 单元.601 室

那么，在 Excel VBA 中，我们怎样才能找到工作簿 Book1.xlsm 中的工作表 Sheet1 中的单元格 A1 呢？经过前面的系列介绍，我们知道单元格 A1 在代码中的表示为：

Range(“A1”)

而工作表 Sheet1 属于 Worksheets 集合，在代码中的表示为：

Worksheets(“Sheet1”)

工作簿 Book1.xlsm 属于 Workbooks 集合，在代码中的表示为：

Workbooks(“Book1.xlsm”)

根据 Excel 对象层次模型，我们要找到单元格 A1，就要经过下图 10.2 所示的路线：

Application → Workbooks(“Book1.xlsm”) → Worksheets(“Sheet1”) → Range(“A1”)

图 10.2

同样，我们用句点来代替图中的箭头号，那么在 VBA 中找到单元格 A1 可以表示为：

```
Application.Workbooks("Book1.xlsm").Worksheets("Sheet1").Range("A1")
```

实际上，这就是 VBA 中引用对象的方式，即通过使用句点层层限定直至达到要引用的对象。

由此，我们引出 VBA 中引用对象的一条规则：

规则 1：对象.(对象...对象).对象

在对象模型中，处于被引用对象上层的对象位于左侧，被引用的对象位于最右侧。

实际上，我们通常就是在 Excel 中操作，所以可以省略掉 Application 对象限定，即引用单元格 A1 的代码可以简化为：

```
Workbooks("Book1.xlsm").Worksheets("Sheet1").Range("A1")
```

如果我们只是在工作簿 Book1.xlsm 中操作，将该工作簿作为当前工作簿，那么代码可以进一步简化为：

```
Worksheets("Sheet1").Range("A1")
```

如果工作表 Sheet1 为当前工作表，代码不会应用到其它工作表中，那么代码再次简化：

```
Range("A1")
```

这代表在当前活动工作簿中的当前工作表中的单元格 A1。

这就好比我们都在一个办公楼里工作，当你问我现在在哪里时，如果我正在 601 室，那么我只需回答在 601 室就够了，因为大家彼此都知道，而无须回答我在××区××写字楼××层××室这么繁琐了。

由此，我们又可以引出 VBA 中引用对象的一条规则：

规则 2：若对象就是当前对象，则可以省略句点左侧的对象限定，简化代码。

注意规则的前提是，我们引用的对象就是当前活动对象，如果不是当前活动对象，就会产生不想要的结果或错误。例如，当前活动工作表为 Sheet2，而我们要找的是工作表 Sheet1 中的单元格 A1，此时，如果只写代码：

```
Range("A1")
```

它代表的是工作表 Sheet2 中的单元格 A1。要找到工作表 Sheet1 中的单元格 1，必须在左侧加上限定：

```
Worksheets("Sheet1").Range("A1")
```

省略 Application 对象限定

在编写代码时，对于 **Application** 对象的那些返回对象的属性，我们常常省略掉 **Application** 对象限定，例如前面的文章中的 **ActiveWorkbook** 属性、**ActiveWindow** 属性、**ActiveCell** 属性、**Cells** 属性、**Selection** 属性、**Rows** 属性、**Columns** 属性，等。只需要直接以属性开头进行引用，例如，代码：

```
ActiveCell
```

代表当前活动工作表中的活动单元格。

11. 神奇的句点（续）

在前面的内容中，我们看到了通过在对象之间加上句点可以逐步得到想要的对象，以及引用对象的省略写法。

在获得对象之后，我们就要对这个对象进行相应的操作。例如，给对象赋值，让对象做相应的动作，这些就要用到对象的属性和方法。

如何使用对象的属性和方法呢？仍然是用句点，例如：

```
Worksheets.Add
```

表示添加工作表。

```
Range("A1").Clear
```

表示完全清除单元格 A1 中的内容和格式。

```
Range("A1").Value
```

可以获得或设置单元格 A1 中的值。

```
Range("A1").Address
```

可以获得或设置单元格的地址表示。

.....

于是，我们可以得到使用对象的属性和方法的规则：

规则 3：对象.方法或者对象.属性

也就是通过在对象后加上句点来使用对象的属性和方法，如下图 11.1 所示。

看一看，与对象的引用相同，都是使用句点。

我们在编写 VBA 代码或查看别人编写的 VBA 代码时，会大量用到或看到这样的表示方法。实际上，这是 VBA 语言中最基础的结构，如下图 11.1 所示。

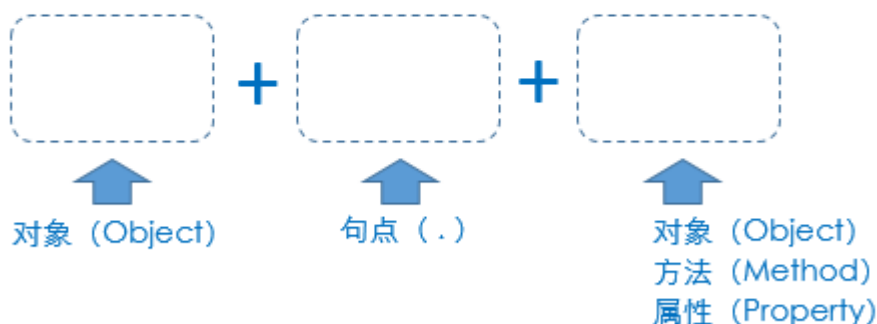
Worksheets . Add

对象 句点 方法

Range("A1") . Address

对象 句点 属性

使用对象的属性和方法



引用对象及其属性和方法的基本结构

图 11.1

现在,我们已经知道了对象有属性和方法,以及表示对象的属性及其方法的语法,那么如何描述它们的属性或方法,具体告诉对象是什么或者该做些什么呢?这就要使用参数了。

对象方法的参数

有很多书中,经常使用踢球的例子。要踢的球是一个对象,它有一个方法,就是踢,使用上面的语法表示,就是:

球.踢

或者:

Ball.Kick

踢多高,向哪个方向去踢,这就是参数要描述的内容。也就是说,方法的参数告诉踢球的具体行为:

球.踢方向:=向左,高度:=3 米

或者:

Ball.Kick Direction:=Left,Height:=3

在 Excel VBA 中，大多数对象的方法也带有参数，告诉该方法具体的行为，例如，在当前工作簿中添加工作表，使用 Worksheets 对象的 Add 方法，如果我们要将新工作表添加到名为 Sheet1 的工作表之后，就要使用 Add 方法的 After 参数，代码如下：

```
Worksheets.Add After:=Worksheets("Sheet1")
```

对象属性的参数

还是使用足球的例子。看到足球后，我们就知道这个球的颜色，大多数是白黑相间的颜色，假如白色有 7 块，黑色有 6 块。可以用下面的式子表示：

球.颜色(白色:=7,黑色:=6)

或者：

```
Ball.Color(White:=7, Black:=6)
```

在 Excel 中，不少属性也带有参数。例如，基于当前单元格 A1 向下偏移 1 行，向右偏移 2 列的单元格，代码表示如下：

```
Range("A1").Offset(RowOffset:=1, ColumnOffset:=2)
```

参数的表示方法

综合上述说明的关于表示对象方法或属性的参数的用法，可以总结出参数表示方法的规则如下：

规则 4： 参数名:=参数值

方法或属性的参数可能不止一个，各参数之间使用逗号（,）分隔开。

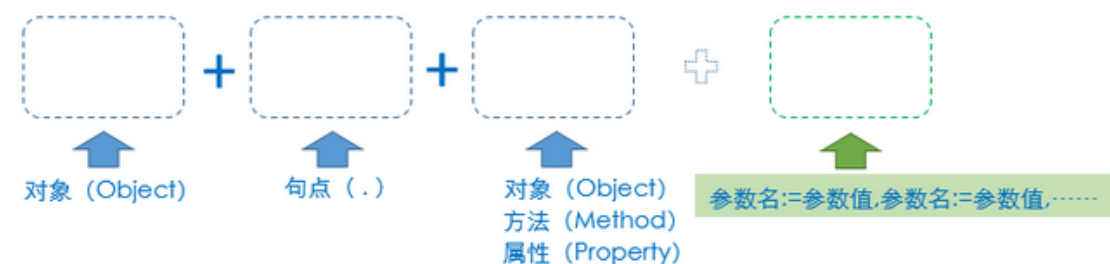


图 11.2

有些情况下，我们可以省略参数名，直接使用参数值，多个参数之间使用逗号(,)分隔。例如：

```
Range("A1").Offset(1, 2)
```

也表示基于当前单元格 A1 向下偏移 1 行，向右偏移 2 列的单元格。

注意，省略参数名虽然简便，但要严格按照参数的顺序来设置相应的参数值，即便有些参数的值无须设置，也要按其顺序使用逗号隔开。例如，基于当前单元格 A1 向右偏移 2 列的单元格：

```
Range("A1").Offset(, 2)
```

Range 对象的 Offset 属性有两个参数，其顺序是：第一个参数 RowOffset,第二个参数 ColumnOffset，上面的代码中，虽然省略了第一个参数，但仍要使用逗号分隔。

如果加上参数名，则无须使用逗号：

```
Range("A1").Offset(ColumnOffset:=2)
```

因此，使用参数名加上:=设置参数是一种较好的编写代码的方法，不仅无须考虑参数的顺序，根据参数名也知道参数设置的具体含义。

对于方法的参数也是如此，在此不再赘述。

默认属性

对于对象的方法或属性来说，有些属性是默认的，也就是说，在我们没有设置属性时，对象就会自动表现出的行为。

例如，Worksheets 对象的 Add 方法，如果我们不设置参数，执行下面的语句：

```
Worksheets.Add
```

默认为在当前工作表前添加一个工作表，新工作表插入到当前工作表的前面。

大多数情况下，我们会省略掉 Range 对象的 Value 属性，例如：

```
Range("A1").Value = "Excel VBA"
```

与

```
Range("A1") = "Excel VBA"
```

相同，将直接在单元格 A1 中输入文本“Excel VBA”。

12. 再回首，说透对象

至此，我们已经了解了对象、属性和方法的基本概念，认识了 Excel 的对象模型以及常用的对象在 VBA 中的表示，以及如何使用对象的属性和方法，这些都是 Excel VBA 编程最基础的部分。熟悉和了解这些知识，对进一步学习 VBA 编程很重要。

为了进一步巩固已经学习的知识，我们引用 Excel 先生（Bill Jelen）的著作《VBA and Macros for Microsoft Excel》中的例子，稍作修改，再次形象化地说明相关的概念和表示对象的基础语法结构。

假设我们使用 VBA 来玩足球，那么踢球的指令应该类似下面这样：

“Kick the Ball”（中文为：“踢球”）

这就是我们平时说话的方式，具有很明确的意义，句子中有动词（Kick）和名词（the Ball）。还记得前面的系列文章中的 VBA 代码，也有动词（Add）和名词（Worksheets）。

虽然说平时我们用于踢球的指令听起来已经很自然，但是 VBA 代码并不是这样来表示的。如果使用 VBA 来踢足球，那么基础的语言结构应该是：

Ball.Kick

这里的名词（Ball），位于最前面，在 VBA 中，就是**对象**，也有动词（Kick）紧随在名词之后，在 VBA 中，就是**方法**。

这也就是 VBA 中最基础的结构：

对象.方法（Object.Method）

如果使用 VBA 编写程序，就一定要习惯于这样的结构。

接着来做假设。在绿色的草地上，放在你面前的有 5 个球，分别是足球、篮球、棒球、保龄球和网球。现在，你给足球队员发出一条指令：

Kick the soccer ball（中文为：踢足球）

如果你仅仅告诉他踢球（或者 `Ball.Kick`），没有明确踢哪一个球，可能他就会踢离他最近的那一个，例如保龄球，这明显会出问题。

对于任何名词或者对象，在 VBA 中，都会有一类对象集合。看看 Excel 中，可以有一行，也可以有多行；可以有一个单元格，也可以有多个单元格；可以有一个工作表，也可以有多个工作表。对象与对象集合的惟一不同之处就是需要在对象的名称后面加上一个表示复数的“s”，例如：

Workbook 成为 Workbooks

Worksheet 成为 Worksheets

Ball 成为 Balls

当所指的是某个对象集合时，就需要告诉 VBA 具体要操作的对象。可以使用不同的方法。

一种方法是使用数字来指明具体的对象：

Balls(2).Kick

然而，这种方式虽然可以达到目的，但也比较危险。例如，可能最初代码工作得很好，但某个时候某人调整了球的顺序，那么 **Balls(2).Kick** 可能就变为踢到其他的球了。

另一种方法是使用名称来指明具体的对象，这是一种比较安全的方式：

Balls("Soccer").Kick

很明确地知道是要踢足球。

在 Excel VBA 中，对于大多数动词或者方法，都会有参数告诉如何执行动作，这些参数相当于副词。你可能想把足球踢到左边并且是使劲踢：

Balls("Soccer").Kick Direction:=Left,Force:=Hard

多数方法都有大量的参数告诉程序如何执行方法。

在查看 VBA 代码时，当看到冒号和等号的组合时，就说明是使用参数描述动词如何执行动作。（而对于属性来说，则是进一步描述形容词的特性）

有时，某个方法可能会有 10 个参数，其中有些参数是可选的，可能 Kick 方法有一个 Elevation 参数，使用下面的代码：

Balls("Soccer").Kick Direction:=Left,Force:=Hard,Elevation:=High

这里容易让人搞糊涂。每个方法的参数都有默认的顺序，如果你碰巧知道参数的位置，那么可以不使用参数的名称，下面的代码等效于上面的代码：

Balls("Soccer").Kick Left,Hard,High

这也可能影响我们对代码的理解，没有冒号和等号的组合，参数的含义就不那么

明显了。当然，知道参数的顺序，是可以理解其意思的，但大多数参数的顺序我们是无法全部记住的。

上面例子中的 **Left**、**Hard** 和 **Hign** 还是比较容易理解，但使用下面的参数时：

```
WordArt.Add Left:=10,Top:=20,Width:=100,Height:=200
```

将其简写成下面的代码：

```
WordArt.Add 10,20,100,200
```

这就让人困惑了！这也是一条有效的代码，但除非你知道 **Add** 方法默认的顺序分别是 **Left**、**Top**、**Width**、**Height**，否则这条代码的意思很难明白。

再复杂一些。用户在一条代码中可以指定某个参数的名称，省略掉另外参数的名称。例如，用户在不指明参数名称的情况下在参数的默认位置设置参数，然后突然带上一个指明名称的参数，而且这个参数不一定在默认的位置上。如果想把踢高高地踢到左边，而不介意力量的大小（使用默认的力量），那么下面的两条语句是等效的：

```
Balls("Soccer").Kick Direction:=Left, Elevation:=High
```

```
Balls("Soccer").Kick Left, Elevation:=High
```

一旦开始指明参数名，后面的代码就都需要指明参数名称。

灵活的参数表示，方便的同时也带来了问题！

对于对象的属性的参数来说，也有类似的规则。

有些方法很简单，例如要模拟按下 **F9** 键（即重新计算工作表），可以使用如下代码：

```
Application.Calculate
```

有些方法可以执行一个动作并创建新的对象，例如，下面的代码添加一张工作表：

```
Worksheets.Add Before:=Worksheets(1)
```

可以将 **Add** 方法创建的新对象指定给一个变量，此时，必须使用括号将参数包住：

```
Set MyWorksheet= Worksheets.Add(Before:=Worksheets(1))
```

与很多讲解对象的书相似，一般代表对象的是名词，代表方法的是动词，而代表属性的则是形容词。前面我们讲过，代码中的：

ActiveCell

代表活动单元格。

现在，我们想将活动单元格的顏色更改为黄色，要用到其属性 **Interior.ColorIndex**，

代码如下：

```
ActiveCell.Interior.ColorIndex = 6
```

虽然代码中用到了两个句点，看似复杂一点，但是其结构仍然是“名词+点+其他内容”，这次是 **Object.Property**（对象.属性）。

差别在于，给属性赋值时，等号前面没有冒号。

对象的属性总是被设置成等于什么，或者是获取属性的值给变量或其它对象的属性。

例如，要将当前单元格的颜色设置成单元格 **A1** 的颜色，代码如下：

```
ActiveCell.Interior.ColorIndex = Range("A1").Interior.ColorIndex
```

Interior.ColorIndex 是属性，通过更改属性的值，可以将对象的表现进行相应的更改。更改形容词就可以对单元格进行某些设置。我们通常会说“将单元格设置为红色的”，而 **VBA** 会这样“说”：

```
ActiveCell.Interior.ColorIndex = 3
```

小结

下表总结了 **VBA** 的“部分语言”

VBA 元素	类似于	说明
对象	名词	
集合	名词复数	一般用于指定具体的对象： Worksheets(1)
方法	动词	Object.Method
参数	副词	列出方法后的参数，将参数名和值用:= 分隔
属性	形容词	可以设置属性： ActiveCell.Height=10 或者查询属性的值： x=ActiveCell.Height

13. 一个简单的 VBA 程序

VBA 是一种“寄生”在主应用程序（例如 Excel）中的语言，正是因为这样的特殊性，所以我们先介绍了与主应用程序相关的对象模型和很多对象，以及如何引用这些对象及其属性和方法。

我的思路（当然也是笔者在当初学习 Excel VBA 时的经历）是，先带领读者认识一些常用的对象及其表示。因为只有你先知道了要使用的对象，才能达到自己的目的。如果你都不知道你要操作什么，空有一番本领，也会感觉一头雾水。

既然你现在已经认识了不少的 Excel 对象，也知道了它们是如何在 Excel 中表示的，知道了对象的方法和属性的表达方法，那么就可以进入下一步了。

下面，我们主要介绍 VBA 语言本身。与此同时，进一步认识 Excel 的对象，并一步一步地让你具备开发出满足自己需求的程序的能力。

先编写一个非常简单的 Excel VBA 程序。

[复习]准备代码输入环境

按 Alt+F11 组合键，打开 VBE 编辑器。单击菜单“插入——模块”，插入一个标准模块，VBE 会自动命名为“模块 1”，如图 13.1 中的箭头所示。

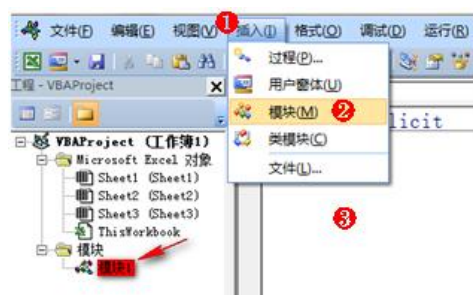


图 13.1

在图 1 中③所标识的代码编辑窗口中，输入代码。

说明：本系列文章中以后所说的输入代码，如无特殊说明，要么是在插入新模块后的代码窗口中输入代码，要么是双击相应的代码模块后在其代码窗口中输入代码。这些细致的差别看似简单，却往往是一些初识 VBA 的朋友迷惑或者出问题的地方。

输入程序代码

在“模块 1”的代码窗口中，输入下图 13.2 所示的代码。

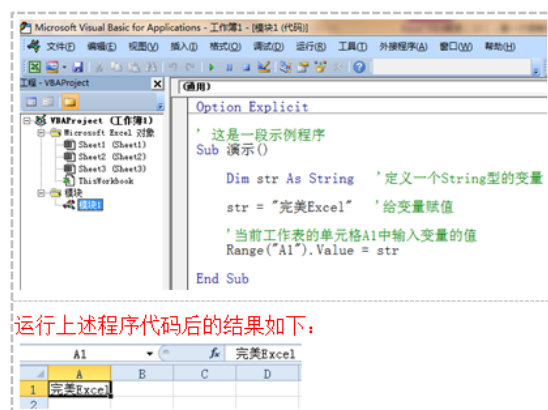


图 13.2

观察图 13.2 中的代码，我们可以看到 3 种颜色：绿色、蓝色、黑色。一般来说，代码注释显示为绿色，VBA 的保留字显示为蓝色，而其他内容显示为黑色。

说明：当输入保留字时，VBE 会自动将其首字母大写。保留字不可以用作变量名，否则程序会报错。

注释

代码注释以撇号（'）开头。除了代码语句中间外，注释可以放置在程序的任何地方。

在示例程序中，我给每行代码都添加了注释，表明代码的功能。有的注释是单独占有一行，有的注释放置在代码语句后面。

注释可以用于说明整个程序的功能和相关信息，也可以用于说明某段代码或某条语句的作用。在很久之后重新打开程序时，注释让我们能够更容易地理解程序，也能够让阅读程序的人尽快了解程序，因此应养成对程序进行注释的习惯，特别对于较大的程序来说，注释必不可少。

VBA 程序的基本结构

VBA 有两种基本的过程，分别称为 Sub 过程和 Function 过程，如下图 13.3 所示。

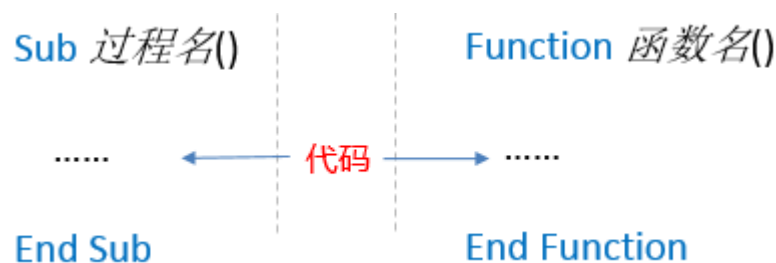


图 13.3

VBA 程序就是以这两种基本的结构组成的。

一般来说，Sub 过程用于放置直接执行的代码，执行过程相当于操作 Excel 中的各种命令按钮或菜单。Function 过程相当于 Excel 中的各类函数，通常会返回值。

VBA 程序语言元素

再次观察图 13.2 中的代码，在 Sub 过程中，有一些 VBA 语言元素，除了注释或固定的基本结构外，还包括：

变量	Str
数据类型	string
保留字	Dim,A S
赋值运算	=

符

此外，有些程序中可能还会包括 VBA 函数、条件或循环语句，这些都是组成 VBA 程序的元素。在后续的系列中，我们将对这些内容逐一讲解。

[复习]运行程序代码

按照 **3. VBA 代码之家** 中所述的执行代码，可得到图 13.2 下方所示的结果。

说明：大家可以参照 **3. VBA 代码之家** 中的“执行代码”节执行代码，也可以使用自己习惯的执行 VBA 代码的方式。关于执行 VBA 代码的方法有很多，以后的系列文章中我们专题讲述。

14. VBA 的数据类型

在 Excel 中，我们会往工作表中输入数字、日期、文本，有时还会粘贴上图片，这些输入的内容都是数据。我们在 Excel 中制作了一个学生入学成绩信息的工作表，如下图 14.1 所示。

	A	B	C	D	E	F
1	序号	姓名	入学时间	分数	是否合格	备注
2	1	张三	2011-9-6	83.5	True	
3	2	李四	2012-9-5	50.3	False	

图 14.1

上面有各种各样的信息，都是通过数据来表现这些信息的，例如学生的姓名、入学时间、考试分数、是否合格以及图片，这些都是数据。

观察一下图 14.1 中的数据，我们可以将其分一下类：

- 有些数据是数字，例如表示序号的 1、2、...和表示分数的 83.5、50.3、...等。
- 有些数据是文本，例如表示姓名的张三、李四、...以及表头的标题如序号、姓名、入学时间、...等。
- 有些数据是日期，例如表示入学时间的 2011-9-6。
- 有些数据是逻辑值，例如表示是否合格的 True 和 False。

在 VBA 中，数据也会被分成不同的类型，例如：

- 处理数字的整型（Integer 型）、长整型（Long 型）、单精度浮点型（Single 型）、双精度浮点型（Double 型）。
- 处理文本的字符串型（String 型）。
- 处理日期时间的日期型（Date 型）。
- 处理真假判断的布尔型（Boolean 型）。

上述都是 VBA 的基本数据类型。本文的后面附有来源于 VBA 帮助所定义的 VBA 的内置数据类型及其内存需求、以及每种类型可以处理的数值范围，方便查阅参考。

为什么会有数据类型

计算机程序离不开数据。通常，我们都是通过程序代码来处理各种数据，得到我们需要的结果。计算机为了实现最优的运算，会以不同的方式存储不同类型的数据，例如不同类别的数字以不同的方式存储，存储数字的方式与存储文本的方式不同。计算机在处理数据时，会根据具体的数据类型以相适应的方式在内存中存储数据，便于程序在运行时区分和使用这些不同类型的数据。

在 VBA 的基本数据类型中，我们通常应用较多的会有 Boolean 型、Integer 型、Long 型、Single 型、Date 型、Object 型、String 型。

在 VBA 中，我们可以处理位于工作表中的数据，也可以处理存放在变量和常量中的数据。接下来，我们就来详细谈谈变量和常量。

附：VBA 基本数据类型

数据类型		大小	数据值范围
(中文)	(英文)	(占用内存空间)	
布尔型	Boolean	2 字节	0～255
字节型	Byte	1 字节	True 或 False
整型	Integer	2 字节	-32768～32767
长整型	Long	4 字节	-2147483648～2147483647
单精度浮点型	Single	4 字节	-3.402823E38～-1.401298E-45(负数)
			1.401298E-45～3.402823E38(正数)
双精度浮点型	Double	8 字节	-1.79769313486231E308～-4.94065645841247E-324(负数)

15. 变量和常量

在编写程序时，少不了要使用变量和常量，它们都是程序中很重要的元素，是组织完美程序的基础。

一个示例

下面是一个简单的 VBA 程序，用来计算半径为 5 的圆的面积。

```
Sub 示例 2()  
    Const PI = 3.1415926  
    Dim r As Integer  
    Dim area As Double  
  
    r = 5  
  
    ' 计算半径为 5 的圆的面积  
    area = PI * r * r  
End Sub
```

其中，r 和 area 就是变量，PI 就是常量。

什么是变量

变量就是在计算机中根据其数据类型预先占用一个位置，在这个位置里可以放置相应的数据。换句话说，我们要在计算机中保存数据，就要有相应的空间，变量就是用来指明这些空间的。

我们可以将值存储在变量中，供程序使用，如下图 15.1 所示。

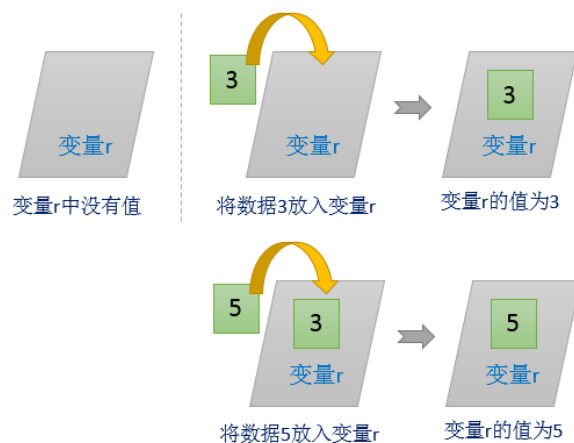


图 15.1

注意，如果将另一个值放入已经存在值的变量中，那么原值就被新值所取代。

命名变量

在程序中，我们经常会使用多个变量。这样，就需要给变量起不同的名字，以区分这些变量，例如图 1 中的 r 就是一个变量的名字。

VBA 规定了变量名的命名规则：

- 可以使用字母、数字和下划线，但必须以字母开头。
- 变量名最长不可以超过 255 个字符。
- 变量名不区分大小写。
- 变量名不能与 VBA 保留字同名，不能与 VBA 中的函数、语句和方法同名。
- 变量名中不能使用空格和句点。

这就说明，我们在命名变量名称时：

- 不能以数字或下划线开头。

- 不能在变量名中使用特殊类型的声明字符（#、\$、%、&或!）。
- 不能在变量名中使用运算符（+、-、*、/、<、>、|、.、,、:）。
- 下列字符也不能使用（?、=、”、;、\、@、^）。
- 在同一作用域内不能有相同的名字。

变量的数据类型

在前一篇中，我们讲过程序中的各种数据都有自己的数据类型（例如整型、字符串型、布尔型等），变量也有自己的数据类型。

在定义变量时，通常要说明变量要存储什么数据类型的值。也就是说，创建变量后，也决定了存储在该变量中的值的数据类型。

例如，上面示例中的：

```
Dim r As Integer
Dim area As Double
```

定义变量时，指定变量 r 存放 Integer 型的数据，而变量 area 存放 Double 型的数据。

说明：与很多编程语言不同，VBA 比较特殊，它在定义变量时，并不一定需要声明存储在变量中的数据类型。VBA 可以自动处理运用数据时涉及到的细节，这对程序编写来说的确很省事，但你会发现，这样的程序执行的速度会更慢，使用内存的效率也不高。如果我们在定义变量时严格指明该变量的数据类型，那么不仅能有效利用计算机空间，并且该变量在进行数据类型转换时，计算机会自动告诉你发生的问题。

声明变量

如上所述，示例中已经给出了声明变量的方法，即使用 Dim 语句来对变量进行声明（或定义）。声明变量的一般语法结构是：

Dim 变量名 As 数据类型

↑ ↑ ↑

空格 空格 空格

图 15.2

当然，还可以使用 **Public** 来声明一个公有变量。正如前面所说的，你可以省略掉 **As** 及之后的数据类型来声明变量，让 **VBA** 自己处理数据，但这并不会带来多少好处。

确切地说，当省略数据类型时，**VBA** 会使用默认的数据类型 **Variant**，在程序处理过程中，会根据所处理内容来改变变量的数据类型。

说明：**VBA** 中“古老的”声明变量的方法

VBA 也延续了 **BASIC** 定义变量数据类型的方法，即将一个字符加到变量名称后面来指定变量的数据类型，例如：

```
Dim r%
```

将变量 **r** 声明为整型。下表列出了有类型声明字符的 **VBA** 数据类型。

数据类型	类型声明字符
Integer（整型）	%
Long（长整型）	&
Single（单精度浮点型）	!
Double（双精度浮点型）	#
Currency（货币型）	@
String（字符串型）	\$

常量

在程序中，如果我们处理的某些数据从不会改变，那么我们可以将它们定义成常量，即决不会发生改变的数值或字符串。

声明常量

VBA 规定使用 **Const** 语句来声明常量。例如本文开头示例中的：

```
Const PI = 3.1415926
```

定义了一个名为 **PI** 的常量，它的值是 **3.1415926**，在程序中使用该常量，其值不会发生变化。我们没有指明 **PI** 的数据类型，**VBA** 会根据它的值确定数据类型，**PI** 的数据类型是 **Double** 型。当然，我们可以在声明的同时明确指定常量的数据类型：

```
Const PI As Double = 3.1415926
```

定义了常量之后，如果要在程序中修改它的值，例如给常量赋新值，将会产生错误。和变量一样，也可以在前面添加 **Public** 来声明一个公有常量。

说明：声明常量的好处

在程序中，将多处使用的值声明为常量，至少有两大好处：

- 如果要修改值，只需手动修改常量定义即可，不必到多个地方修改同一值，不仅繁琐，也容易遗漏。
- 给常量起一个好的名字，能够清楚地表明其意图，让代码更具可读性。

除了我们定义的常量外，**VBA** 还提供了很多预定义的常量，这些常量不用明确声明就可使用。**VBA** 预定义的常量通常以 “**xl**” 或 “**vb**” 开头。

附：VBA 的保留字

按照《Visual Basic for Applications language reference for Office 2013》中的定义，关键字作为 Visual Basic 编程语言的一部分，是一个单词或符号，例如语句、函数名或操作符。下面的网址中列出了所有的关键字。

关键字：[http://msdn.microsoft.com/en-us/library/office/jj692803\(v=office.15\).aspx](http://msdn.microsoft.com/en-us/library/office/jj692803(v=office.15).aspx)

关键字汇总：

[http://msdn.microsoft.com/en-us/library/office/jj692797\(v=office.15\).aspx](http://msdn.microsoft.com/en-us/library/office/jj692797(v=office.15).aspx)

16. VBA 的运算符

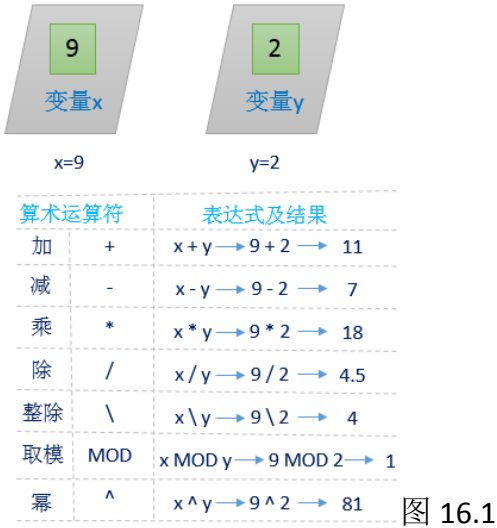
还记得小时候学习数学的经历吗？很小很小的时候，我们会接触到数字，大人们会教我们数数，认识简单的数字，慢慢地我们会开始学习简单的加减法，再大一点，会学习乘法，背诵乘法口诀，随着学习的深入，我们会逐渐学习到更复杂的运算和规则。

运算无处不在。在编写程序代码时，很多内容都是在进行各种各样的运算，因此，运算符非常重要。

当然，最开始我们都是从了解最基本的运算开始，逐渐深入到更多的运算内容。最先应该了解的当然是加、减、乘、除等算术运算符。

算术运算符

常用的 VBA 算术运算符有：加 (+)、减 (-)、乘 (*)、除 (/)、整除 (\)、取模 (MOD)、求幂 (^)。基本的运算如下图 16.1 所示。



说明：

- 对于除法 (/) 运算符：假设 $z=x/y$ ，如果将 z 声明为整型数，则结果为 4，即直接舍弃掉小数位；如果将 z 声明为 Single 型或 Double 型，则结果为 4.5。
- 整除 (\) 运算符，即在除法运算中，结果直接取商，而不管余数。
- 取模 (MOD) 运算符，即在除法运算中，结果取余数。
- 将变量、常量或其他元素使用运算符连接起来，就组成了表达式，特别地，单个的变量或常量也看作是表达式。

关系运算符

关系运算符用于比较，也称为比较运算符。VBA 的关系运算符有：等于 (=)、大于 (>)、小于 (<)、大于等于 (>=)、小于等于 (<=)、不等于 (<>)。基本的比较运算如下图 16.2 所示。

<div><div>9</div><div>变量x</div></div> <div>x=9</div>	<div><div>2</div><div>变量y</div></div> <div>y=2</div>
关系运算符	表达式及结果
等于	= $x = y \rightarrow 9 = 2 \rightarrow \text{False}$
大于	> $x > y \rightarrow 9 > 2 \rightarrow \text{True}$
小于	< $x < y \rightarrow 9 < 2 \rightarrow \text{False}$
大于等于	>= $x >= y \rightarrow 9 >= 2 \rightarrow \text{True}$
小于等于	<= $x <= y \rightarrow 9 <= 2 \rightarrow \text{False}$
不等于	<> $x <> y \rightarrow 9 <> 2 \rightarrow \text{True}$

图 16.2

在 VBA 中，关系运算符用于比较运算符两侧的表达式结果，比较的结果为 True（真）、False（假）或者 Null。因此，关系运算符常用于条件判断中。

Is 运算符

在 VBA 中，经常要在对象变量之间进行判断，例如判断两个对象变量是否引用同一个对象，这时就要用到 Is 运算符。

- **对象变量 1 Is 对象变量 2**

如果对象变量 1 和对象变量 2 都指向同一个对象，其结果是 True，否则是 False。

- **对象变量 Is Nothing**

用于判断该对象变量是否有已设置对象引用，如果没有则为 False。这是程序中经常会用到的语句。

示例：

结合前面学习的系列知识，我们来看看下面语句的意思。

- **Intersect(ActiveCell, Range("A1:B2")) Is Nothing**

该语句判断当前单元格是否在单元格区域 A1:B2 内，如果当前单元格不在该区域内，则为 True。Intersect 方法属于 Application 对象，返回一个 Range 对象，代表该方法指定参数的交叉区域。

逻辑运算符

逻辑运算符用于对表达式进行运算并返回一个逻辑值。VBA 支持 6 个逻辑运算符，即 Not（逻辑非）、And（逻辑与）、Or（逻辑或）、Xor（逻辑异或）、Eqv（逻辑与或）、Imp（逻辑蕴涵）。其中，Not、And、Or 在我们通常的程序中经常会用到，应重点了解。

Not

对表达式进行逻辑非运算后，如果表达式的结果为 True，那么 Not 运算使其值变为 False；如果表达式的结果为 False，那么 Not 运算使其值变为 True。其格式为：

Not 表达式

运算结果表如下：

表达式	Not 表达式
True	False
False	True

And

如果执行逻辑与运算的表达式都为 True，则其值为 True，只要有一个表达式的结果为 False，则其值为 False。其格式为：

表达式 1 And 表达式 2 And 表达式 3 And ... 表达式 n

表达式至少有 2 个。

运算结果表如下：

表达式 1	表达式 2	表达式 1 And 表达式 2
True	True	True
True	False	False
False	True	False
False	False	False

Or

如果执行逻辑或运算的表达式中至少有一个为 True，则其值为 True，如果表达式的结果都为 False，则其值为 False。其格式为：

表达式 1 Or 表达式 2 Or 表达式 3 Or ... 表达式 n

表达式至少有 2 个。

运算结果表如下：

表达式 1	表达式 2	表达式 1 Or 表达式 2
True	True	True
True	False	True
False	True	True
False	False	False

示例

结合前面学习的系列知识，我们来看看下面语句的意思。

- `ActiveSheet.Name="工作表 1" Or ActiveSheet.Name="工作表 2"`

如果当前工作表是“工作表 1”，或者是“工作表 2”，则为 True。

- `ActiveSheet.Name="工作表 1" And Range("A1").Value="工作清单"`

如果当前工作表是“工作表 1”，并且当前工作表的单元格 A1 中的值为“工作清单”，则为 True。

● `ActiveCell.Font.Bold = Not ActiveCell.Font.Bold`

如果当前单元格中的字体为常规，则将其设置为加粗；如果当前单元格中的字体已加粗，则将其设置为常规。

补充说明：上述 6 个逻辑运算符也可作为按位运算符使用，并且我们只介绍了常用的 3 个，在后面的系列中如果用到其它的 3 个运算符，我们会再进行详细介绍。

字符串运算符

VBA 中的字符串运算符仅一个，即字符串连接运算符，使用 & 表示，用于将一个或多个独立的字符串连接在一起。例如：

`"My "& "Family"`

的结果为：

`My Family`

运算符的优先级

如果在同一个表达式中包含有多个运算符，那么就需要了解 VBA 是如何安排那个运算符先运算，那个运算符后运算，即运算的优先顺序。不了解运算符的优先顺序，随意组合运算符，可能得不到我们想要的结果。

下图 16.3 列出了 VBA 中运算符的优先级。

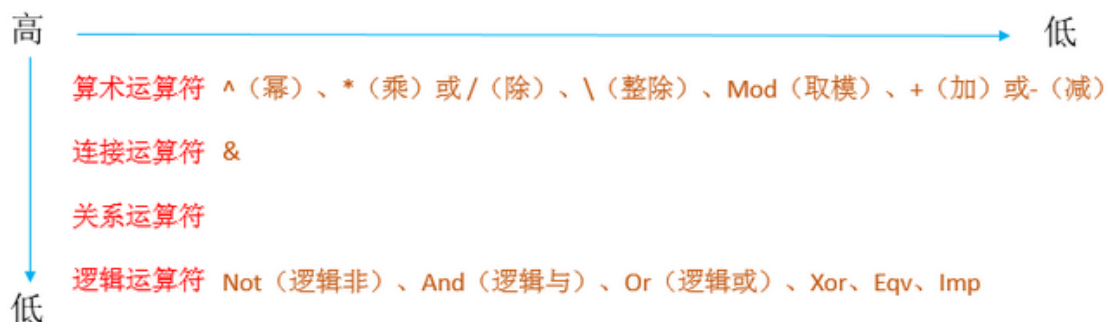


图 16.3

说明：

- 四类运算符由高到低的顺序依次为：算术运算符、连接运算符、关系运算符、逻辑运算符。
 - 算术运算符由高到低的顺序依次为： \wedge （幂）、 $*$ （乘）或 $/$ （除）、 \backslash （整除）、Mod（取模）、 $+$ （加）或 $-$ （减）。
- 同一表达式中多次使用同一个算术运算符时，按从左到右的顺序运算。
- 同一表达式中多个关系运算符，按从左到右的顺序运算。
 - 同一表达式中多次使用同一逻辑运算符时，按从左到右的顺序运算。
 - 如果想改变运算符的优先顺序，可以在表达式中使用() $[]$ 。

赋值运算符

VBA 使用等号(=)作为赋值运算符，用于将表达式的结果赋给变量。通常，接受结果的变量位于赋值运算符(=)的左侧，表达式位于赋值运算符的右侧。通过赋值运算符将右侧表达式的结果赋值给左侧的变量。赋值运算符可以修改变量的值。

下图 16.4 所示为一些简单的赋值示例。

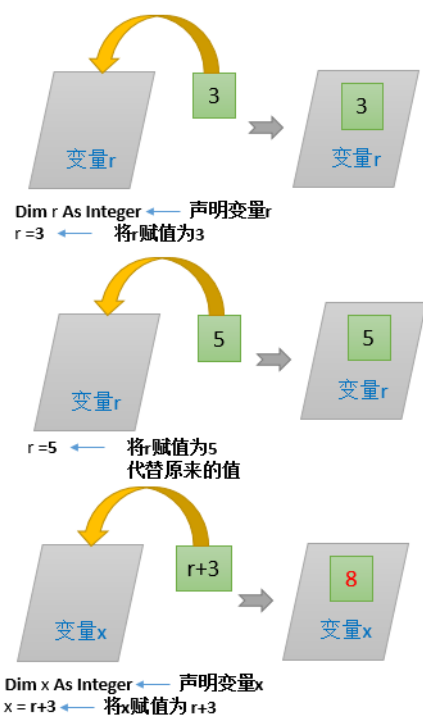


图 16.4

前面我们讲过，可以通过 VBA 来改变对象属性值，因此，也可以将表达式的结果赋值给对象的属性。下面是一些示例：

- `Range("A1").Value = 100`

该语句在单元格 A1 中输入数据 100。

- `Range("A2").Value = "工作清单"`

该语句在单元格 A2 中输入文本“工作清单”。

- `ActiveCell.Interior.Color = vbYellow`

该语句将当前单元格的背景色设置为黄色。

注意：不要将赋值运算符与关系运算符中的“=”混淆，要在不同的表达式中区分“=”是赋值运算符还是等于运算符。

17. 谈谈对象变量

通过前面两篇文章的学习，我们已经了解了 Excel VBA 的变量，知道了如何给变量赋值。下面，我们详细谈谈 Excel 中很重要的对象变量。

变量除了可以表示数值、字符串等外，还可以表示一个工作簿、一张工作表、一个单元格，这就是对象变量（Object 类型的变量）。

声明对象变量

对象变量的声明没有什么特殊之处，仍然是使用 Dim 语句或者 Public 语句。下图 17.1 是一些常用的对象声明示例：


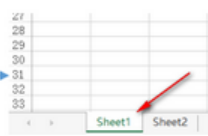
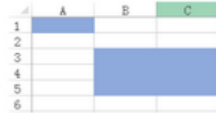
Dim obj As Object	声明变量obj为Object对象(通用的)
Dim wb As Workbook	声明变量wb为Workbook对象 
Dim wks As Worksheet	声明变量wks为Worksheet对象 
Dim rng As Range	声明变量rng为Range对象 

图 17.1

注意：当我们不知道对象变量的具体类型时，可以将其声明为通用的 Object 类

型。

给对象变量赋值

对象变量的赋值与普通变量的赋值不同，必须使用关键字 **Set**。

例如，使用下面的语句将变量 `rng` 声明为 `Range` 对象：

```
Dim rng As Range
```

然后，使用下面的语句给变量 `rng` 赋值：

```
Set rng = Worksheets("Sheet1").Range("A1:B2")
```

现在，我们就可以使用变量 `rng` 来代表工作表 `Sheet1` 中的单元格区域 `A1:B2`。

此时，再运行代码：

```
rng.Value = "示例"
```

将在单元格区域 `A1:B2` 中输入“示例”文本，如图 17.2 所示。

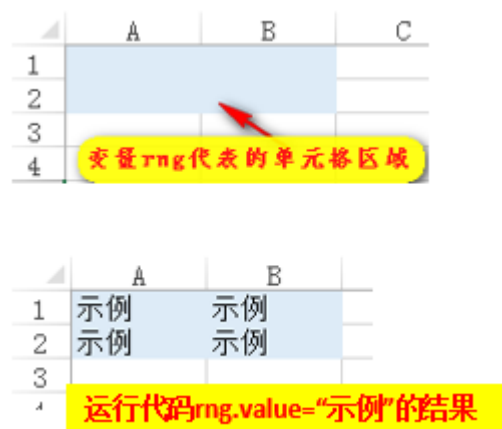


图 17.2

注意：在给对象变量赋值时，一定要加上关键字 **Set**。很多代码错误往往就是在给对象变量赋值时没有使用 **Set**。

给对象变量赋值的通用结构为：

```
Set 对象变量名= 对象
```

使用对象变量的好处

仍然使用上面给单元格区域 A1:B2 输入文本“示例”的例子。如果不使用对象变量，那么将使用下面的代码来操作该区域：

```
Sub test()  
    Worksheets("Sheet1").Range("A1:B2").Value = "示例"  
Worksheets("Sheet1").Range("A1:B2").Font.Bold = True  
Worksheets("Sheet1").Range("A1:B2").Font.Size = 19  
Worksheets("Sheet1").Range("A1:B2").Interior.Color = vbYellow  
End Sub
```

该段代码首先在工作表 Sheet1 的单元格区域 A1:B2 中输入文本“示例”，然后将字体加粗，字号大小调整为 19 号，将单元格背景色设置为黄色。

让我们再来看看使用对象变量达到同样效果的代码：

```
Sub testUpdate()  
    Dim rng As Range  
  
    Set rng = Worksheets("Sheet1").Range("A1:B2")  
  
    rng.Value = "示例"  
    rng.Font.Bold = True  
    rng.Font.Size = 19  
    rng.Interior.Color = vbYellow  
End Sub
```

代码显得更简洁，似乎运行的速度也加快了。因此，我们总结使用对象变量的好处有：

- 可以很好地简化代码，使代码更容易阅读。
- 可以提高代码的执行速度。在多次引用相同的对象时，使用对象变量后，点运算符的数目明显减少，用于解析对象引用的时间也会减少，自然就提高了代码的执行速度。
- 方便地操作其他工作表或工作簿中的数据。例如，在操作当前工作表的数据时，我们要复制或使用其他工作表中的数据，此时，可以将其他工作表中的数据区域赋值给对象变量，就可以方便地使用了，而不用担心

代码究竟在操纵哪个工作表。

- 方便创建新的对象实例。例如下面的代码：

```
Sub CreateNewWorkbook()  
    '声明工作簿和工作表对象变量  
    Dim wb As Workbook  
    Dim wks As Worksheet  
  
    '创建新的对象实例并赋值  
    Set wb = Workbooks.Add  
    Set wks = wb.Worksheets("Sheet1")  
  
    '对工作表进行操作  
    wks.Name = "我的工作表" '重命名工作表  
    wks.Range("A1") = "Test" '给工作表中的单元格 A1 填充值  
End Sub
```

18. With... End With 结构 , 专为对象而生

上一章中专门介绍了对象变量,因为在 Excel VBA 中,我们主要是和对象打交道,这包括各种各样的对象以及由这些对象组成的集合对象。我们来看看并学习 Excel 内部是如何处理对象的。

我们先来录制一个宏。

在 Excel 中,打开宏录制器,录制下面的操作:在工作表单元格 A1 中输入“完美 Excel”,设置其字体为红色,单元格背景色为黄色。录制的代码如下:

```
Sub 宏 3()  
'  
' 宏 3 宏  
'  
'  
  
    ActiveCell.FormulaR1C1 = "完美 Excel"  
Range("A1").Select  
    With Selection.Font  
        .Color = -16776961  
        .TintAndShade = 0  
    End With  
    With Selection.Interior  
        .Pattern = xlSolid
```

```

        .PatternColorIndex = xlAutomatic
        .Color = 65535
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
End Sub

```

观察代码，我们发现，宏录制器为设置字体和单元格背景的两段代码都使用了 **With ... End With** 结构，这就是 VBA 为我们提供的处理对象的有效方法之一。

在这里，宏录制器自动优化了代码，在 **With ... End With** 结构中对同一个对象执行多项操作。当需要对某个对象执行多项操作时（例如，为同一对象的多个属性赋值），使用 **With ... End With** 结构的代码比在每个语句中都显示地引用对象的代码要更快。

在 17. 谈谈对象变量中，我们展示了一个示例：在单元格区域 **A1:B2** 中输入文本“示例”，将字体加粗，字号大小调整为 19 号，并将单元格背景设置为黄色。

代码 1：不使用对象变量

```

Sub test()
    Worksheets("Sheet1").Range("A1:B2").Value = "示例"
    Worksheets("Sheet1").Range("A1:B2").Font.Bold = True
    Worksheets("Sheet1").Range("A1:B2").Font.Size = 19
    Worksheets("Sheet1").Range("A1:B2").Interior.Color = vbYellow
End Sub

```

代码 2：使用对象变量

```

Sub testUpdate()
    Dim rng As Range

    Set rng = Worksheets("Sheet1").Range("A1:B2")

    rng.Value = "示例"

```



```
rng.Font.Bold = True  
rng.Font.Size = 19  
rng.Interior.Color = vbYellow  
End Sub
```

上述两段代码是我们已经学过的代码。现在，我们学习了 With ... End With 结构，又可以将代码进行改写而获得相同的效果。

代码 3：使用 With ... End With 结构（不使用对象变量）

```
Sub test1()  
    With Worksheets("Sheet1").Range("A1:B2")  
        .Value = "示例"  
        .Font.Bold = True  
        .Font.Size = 19  
        .Interior.Color = vbYellow  
    End With  
End Sub
```

代码 4：使用 With ... End With 结构（使用对象变量）

```
Sub testUpdate1()  
    Dim rng As Range  
  
    Set rng = Worksheets("Sheet1").Range("A1:B2")  
  
    With rng  
        .Value = "示例"  
        .Font.Bold = True  
        .Font.Size = 19  
        .Interior.Color = vbYellow  
    End With
```

End Sub

实际上，我们还可以进一步深入，将相同的对象全部归于 With ... End With 结构内，这样就可以得到下面的代码。

代码 5：完全使用 With ... End With 结构

```
Sub testUpdate2()  
    Dim rng As Range  
  
    Set rng = Worksheets("Sheet1").Range("A1:B2")  
  
    With rng  
        .Value = "示例"  
        With .Font  
            .Bold = True  
            .Size = 19  
        End With  
        .Interior.Color = vbYellow  
    End With  
End Sub
```

With ... End With 结构为我们提供了更高效的处理重复引用对象的方式，虽然难以理解一点，也更难阅读，但它确实能带来运行速度上的提高。

19. 员工管理系统开发 V1.0

“什么？才学了点皮毛就可以开发系统了？”我想您读到这儿，看到本篇的题目，一定会感到非常疑惑。

如果您仔细体会了前 18 章的内容，就可以开始着手来试着做一些有趣的事情了。不错！现在我们就尝试用已学到的知识来逐步开发一套小系统。为何这么快就可以运用 VBA 来开始实战了呢？这就是 Excel 作为一个开发平台的优势！

不过，在未开始之前，还是得声明一点，此时编写的代码会很“古板”，有很多约束，实现的功能也有限，你会觉得完全没有必要用 VBA 代码来实现。但我们此刻的目的是让您能够体会到 Excel VBA 的初步魅力，以及使用 VBA 实现手动操作的方法，更重要的是让您巩固前一阶段所学习到的知识。随着更加深入的学习，我们会逐步改善这个系统，使用的知识和技巧也会越多，系统功能也会更强大，系统也会越来越完善。

系统初步规划

员工管理系统的初步构想是：

- 员工按表格要求填写相应的信息。
- 自动汇总员工所填写的信息。
- 查询某员工的信息。
- 查询某员工的信息并能够自动显示在员工信息表中，方便打印输出。
- 分析员工的信息，例如毕业 5 年的员工有多少、具备高级职称的员工有多少。
- 其他一些扩展功能。（暂时未想到，根据需要扩展）

表格设计

根据初步构想，先在 Excel 中设计两个工作表。

- “员工基本信息表”：便于员工填写信息。
- “员工信息数据库”：汇总员工填写的信息。

工作表初步设计如下图 19.1 所示：

1	员工基本信息表							
2	编号:				填表时间:			
3	姓名		性别		民族		照片	
4	出生日期		籍贯		政治面貌			
5	毕业院校				毕业时间			
6	初始学历		学位		专业			
7	工作单位				职称			
8	所在部门		职务		职称时间			
9	主要特长		业余爱好		联系电话			
10	个人简历							
11	主要工作业绩							
12	获奖情况							

在此表中填写信息

1	编号	填表时间	姓名	性别	民族	出生日期	籍贯	政治面貌
2								
3								

信息汇总到下表中

图 19.1

其中，“员工信息数据库”表实际上是将“员工基本信息表”中所填写的数据按数据库记录格式汇总到一张表中。

代码

下面的代码实现将“员工基本信息表”中填写的数据写入到“员工信息数据库”中：

```
Sub TotalData()  
    '声明 Worksheet 变量  
    Dim wksInfo As Worksheet  
    Dim wksBaseInfo As Worksheet  
  
    '给 wksInfo 变量赋值  
    Set wksInfo = ThisWorkbook.Worksheets("员工信息数据库")
```

```
Set wksBaseInfo = ThisWorkbook.Worksheets("员工基本信息表")
```

```
With wksInfo
```

```
.Range("A2").Value = wksBaseInfo.Range("B2").Value  
.Range("B2").Value = wksBaseInfo.Range("F2").Value  
.Range("C2").Value = wksBaseInfo.Range("B3").Value  
.Range("D2").Value = wksBaseInfo.Range("D3").Value  
.Range("E2").Value = wksBaseInfo.Range("F3").Value  
.Range("F2").Value = wksBaseInfo.Range("B4").Value  
.Range("G2").Value = wksBaseInfo.Range("D4").Value  
.Range("H2").Value = wksBaseInfo.Range("F4").Value  
.Range("I2").Value = wksBaseInfo.Range("B5").Value  
.Range("J2").Value = wksBaseInfo.Range("F5").Value  
.Range("K2").Value = wksBaseInfo.Range("B6").Value  
.Range("L2").Value = wksBaseInfo.Range("D6").Value  
.Range("M2").Value = wksBaseInfo.Range("F6").Value  
.Range("N2").Value = wksBaseInfo.Range("B7").Value  
.Range("O2").Value = wksBaseInfo.Range("F7").Value  
.Range("P2").Value = wksBaseInfo.Range("B8").Value  
.Range("Q2").Value = wksBaseInfo.Range("D8").Value  
.Range("R2").Value = wksBaseInfo.Range("F8").Value  
.Range("S2").Value = wksBaseInfo.Range("B9").Value  
.Range("T2").Value = wksBaseInfo.Range("D9").Value  
.Range("U2").Value = wksBaseInfo.Range("F9").Value  
.Range("V2").Value = wksBaseInfo.Range("B10").Value  
.Range("W2").Value = wksBaseInfo.Range("B11").Value  
.Range("X2").Value = wksBaseInfo.Range("B12").Value
```

```
End With
```

```
End Sub
```

代码很简单，用到的都是前面我们学到的知识。在"员工基本信息表"中填写相应

的数据，运行上面的代码，可以看到数据会自动填充到"员工信息数据库"的第二行中。

这里的代码只是机械地将一个表中的数据复制到另一个表中相应的单元格，并且只能实现填充固定一行的数据。

接下来，我们边进一步学习，边完善我们的系统，使系统逐渐灵活方便起来。

20. 用户交互初体验—MsgBox 函数

下面，继续做一些有趣的事情。让 VBA 能够给我们弹出对话框，让用户作出响应，实现与用户交互。这里要使用 VBA 的内置函数：MsgBox 函数和 InputBox 函数。

与 Excel 工作表函数一样，VBA 也包含有各种内置函数，以简化计算和操作，而 MsgBox 函数和 InputBox 函数就是其中常用的函数。这里，先介绍 MsgBox 函数。

显示信息

MsgBox 函数可以为我们提供输出信息，将 VBA 代码的运行结果告诉我们。看下面简单的示例，代码运行后会弹出一个对话框，如图 20.1 右侧所示。

```
Sub MsgBoxTest()  
    Dim i As Long  
    i = 5  
    MsgBox "变量i的值是" & i  
End Sub
```



图 20.1

此时，Excel 告诉我们相关信息，并会等待我们作出响应，单击“确定”，对话框消失。当然，这只是 MsgBox 函数最基本的用法。我们可以让它显示更丰富的信息，例如，显示更有意义的标题、更多的选择按钮、图标，甚至获取用户响应后的值。

将上面的程序代码修改如下：

```
Sub MsgBoxTest()  
    Dim i As Long  
    i = 5  
    MsgBox Prompt:="变量 i 的值是" & i, Buttons:=vbOKCancel + vbCritical, Title:="显示变量的值"  
End Sub
```

运行后的结果如下图 20.2：

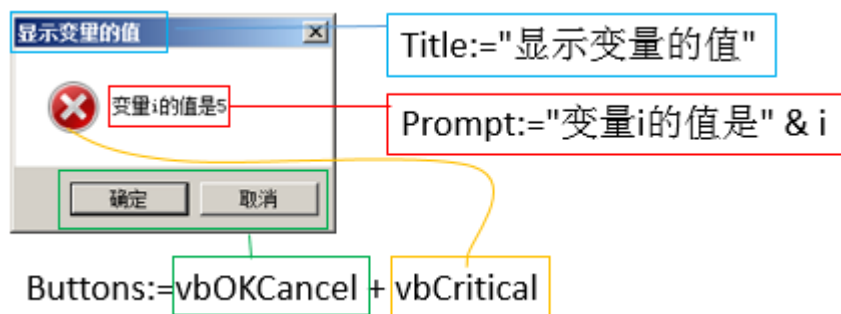


图 20.2

图 20.2 中附加说明了 MsgBox 函数中各参数在对话框中对应的显示。

MsgBox 函数的语法

MsgBox 函数的语法如下：

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

其中，方括号内的参数为可选参数。参数说明如下图 20.3 所示：

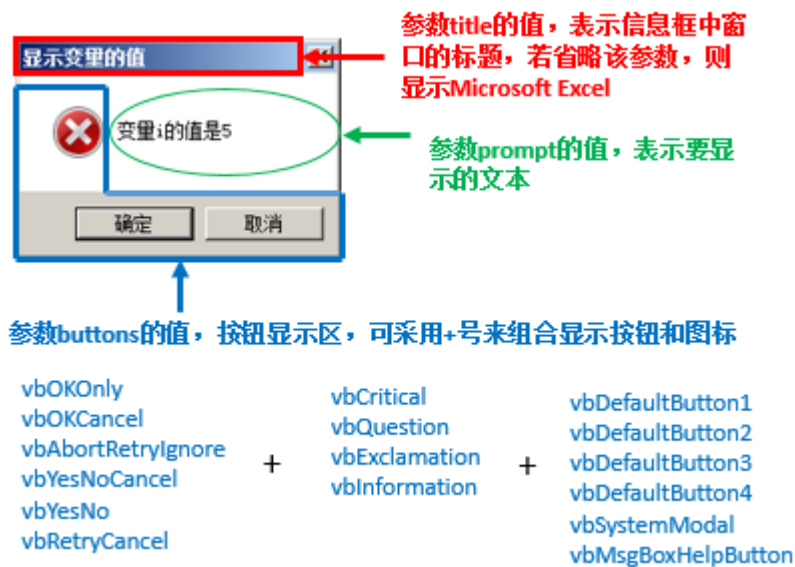


图 20.3

还有两个参数 helpfile 和 context，我们暂且不要管它。

下表列出了参数 buttons 的值及说明：

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“警告询问”图标
vbExclamation	48	显示“警告消息”图标
vbInformation	64	显示“通知消息”图标
vbDefaultButton1	0	第一个按钮为默认按钮
vbDefaultButton2	256	第二个按钮为默认按钮
vbDefaultButton3	512	第三个按钮为默认按钮
vbDefaultButton4	768	第四个按钮为默认按钮
vbSystemModel	4096	所有应用程序都暂停，直至用户对消息框作出反应

常量	值	说明
vbMsgBoxHelpButton	16384	显示帮助按钮

获取响应的值

在用户单击了 MsgBox 函数对话框中的按钮后，我们可以获取用户的行为，即用户单击了哪个按钮。这样，我们可以根据用户的选择作出相应的处理。

我们可以将 MsgBox 函数的结果赋给某个变量，此时需要将参数放在括号中，例如：

```
Sub MsgBoxTest1()
    Dim Msg As Integer
    Msg = MsgBox("您要继续运行吗?", vbYesNo)
    MsgBox Msg
End Sub
```

代码运行结果如下图 20.4 所示：

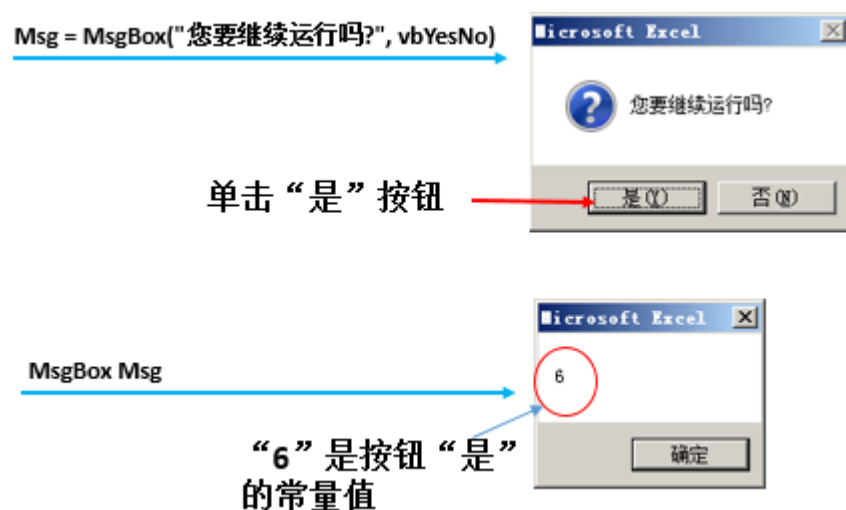


图 20.4

接下来，我们就可以编写代码对用户单击的选择进行相应的回应。这种情形经常使用在条件判断语句中，在讲述相关内容时，我们再给出使用示例。

下表列出了 MsgBox 函数的返回值：

常量	值	相应的按钮
vbOK	1	确定
vbCancel	2	取消
vbAbort	3	终止
vbRetry	4	重试
vbIgnore	5	忽略
vbYes	6	是
vbNo	7	否

21. 向左走，向右走—— 使用 If 语句选择

在日常生活中，我们常常要做出选择，例如，如果明天不下雨，我们就去郊游。在 VBA 中，也有类似的语句，让我们控制程序的执行方向，例如，如果单元格 A1 的值为 0，那么就弹出警告消息“单元格 A1 的值不能为 0”。

将上面的描述用语句来表达，如下图 21.1 所示：



图 21.1

VBA 中 If-Then 语句结构的基本语法如下：

```
If 条件 Then 条件为真时执行的语句 [Else 条件为假时执行的语句]
```

说明：

- 条件为真（假）时执行的语句可以为一条或多条。
- 方括号中的 Else 子句可选。

前述示例完整的过程代码为：

```
Sub testIf()  
    If Range("A1").Value = 0 Then MsgBox "单元格 A1 的值不能为 0"  
End Sub
```

或者：

```
Sub testIf2()
```

```
If Range("A1").Value = 0 Then
    MsgBox "单元格 A1 的值不能为 0"
End If
End Sub
```

注意，在过程 testIf2 中，将 If-Then 语句写在多行中，此时应加上 End If 语句。在 If-Then 行与 End If 行之间，可以添加多条语句。

再看看下面的代码：

```
Sub testIf3()
    If Range("A1").Value = 0 Then
        MsgBox "单元格 A1 的值不能为 0"
    Else
        Range("B1").Value = 20 / Range("A1").Value
    End If
End Sub
```

代码的意思是：如果单元格 A1 的值为 0，那么就弹出警告消息“单元格 A1 的值不能为 0”；否则，就用 20 来除以单元格 A1 中的值，并将结果填写到单元格 B1 中。

当然，下面的代码也可以实现上述功能，只是使用了更多的 If-Then 语句：

```
Sub testIf4()
    If Range("A1").Value = 0 Then MsgBox "单元格 A1 的值不能为 0"
    If Range("A1").Value <> 0 Then Range("B1").Value = 20 / Range("A1").Value
End Sub
```

我们再多一些条件，如下面的代码：

```
Sub testIf5()
    If Range("A1").Value = 0 Then MsgBox "单元格 A1 的值不能为 0"
    If Range("A1").Value <= 10 Then Range("B1").Value = 20 / Range("A1").Value
    If Range("A1").Value > 10 Then Range("B1").Value = 100 / Range("A1").Value
End Sub
```

如果单元格 A1 的值为 0，那么就弹出警告消息“单元格 A1 的值不能为 0”；如果单元格 A1 中的值小于等于 10，就用 20 来除以单元格 A1 中的值，并将结果填

写到单元格 B1 中；如果单元格 A1 中的值大于 10，就用 100 来除以单元格 A1 中的值，并将结果填写到单元格 B1 中。

改写 If-Then 结构实现相同的效果，代码如下：

```
Sub testIf6()  
    If Range("A1").Value = 0 Then  
        MsgBox "单元格 A1 的值不能为 0"  
    ElseIf Range("A1").Value <= 10 Then  
        Range("B1").Value = 20 / Range("A1").Value  
    Else  
        Range("B1").Value = 100 / Range("A1").Value  
    End If  
End Sub
```

由上可知，在多条件下 If-Then 结构的扩展语法结构为：

```
If 条件 Then  
    条件为真时执行的语句  
[Elseif 另一条件 Then  
    另一条件为真时执行的语句  
[Else  
    所有条件都为假时执行的语句  
End If
```

说明：

- 可以有多个 Elseif 子句，但最多只有一个 Else 子句。

当然，还可以使用嵌套的 If-Then-Else 结构，即一个 If-Then-Else 结构嵌套于另一个 If-Then-Else 结构中。例如，将上例改写为：

```
Sub testIf7()  
    If Range("A1").Value = 0 Then  
        MsgBox "单元格 A1 的值不能为 0"  
    Else  
        If Range("A1").Value <= 10 Then
```

```

Range("B1").Value = 20 / Range("A1").Value
    Else
Range("B1").Value = 100 / Range("A1").Value
    End If
End If
End Sub

```

实现的效果相同，但是应特别注意 If 与 Else 和 End If 的配对正确。

下面举一个例子，说明 If-Then 结构的应用。

如下表所示的规则，根据员工的工龄来确定员工的年休假天数：

工龄（年） 年休假天数

10 年及以下	5
10～20	10
20～25	15
25 年以上	20

转换成 VBA 代码如下：

```

Sub NianXiuTian()
    '声明变量,用来表示年休天数和工龄
    Dim lngDays As Long
    Dim lngYears As Long

    lngYears = Range("A1").Value

    '根据工龄确定相应的年休天数
    If lngYears >= 0 And lngYears <= 10 Then
lngDays = 5
        ElseIf lngYears > 10 And lngYears <= 20 Then
lngDays = 10
        ElseIf lngYears > 20 And lngYears <= 25 Then
lngDays = 15
        Else

```

```
IngDays = 20
```

```
End If
```

```
MsgBox "工龄: " & IngYears & vbCrLf & "年休天数:" & IngDays
```

```
End Sub
```

此时，您可以在工作表单元格 A1 中输入代表工龄的数字，运行代码后，会显示相应的年休天数，如下图 21.2 所示。

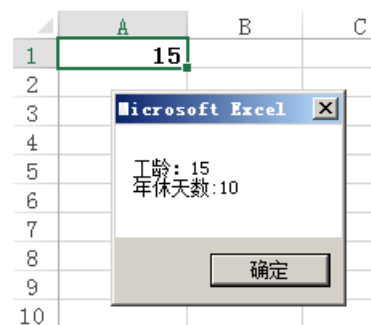


图 21.2

最后，将 If-Then 语句结构及其变形归纳如下图 21.3：

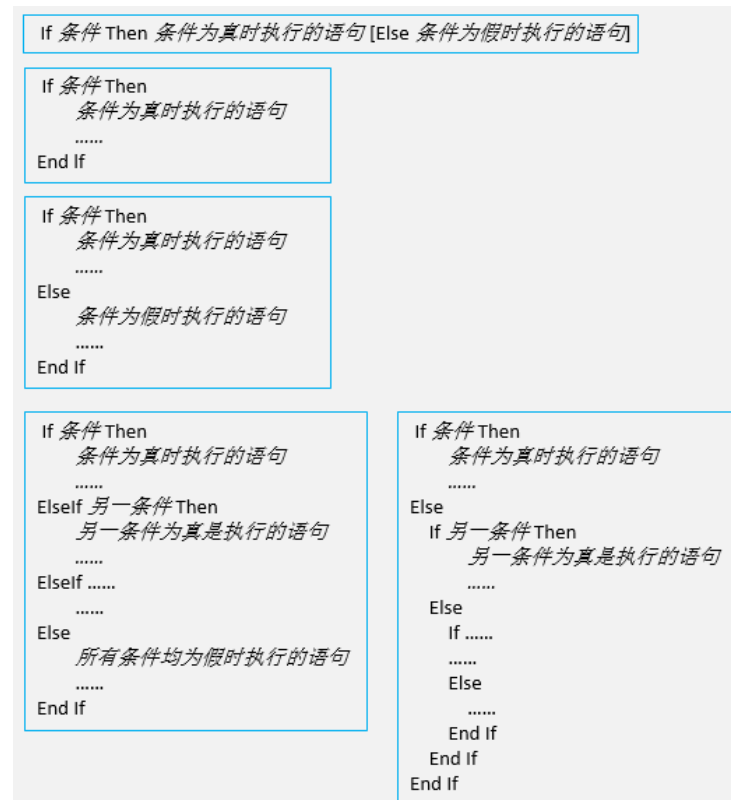


图 21.3

22. 方便的“多选一”结构—— Select Case 结构

人生经常要面临“多选一”的情况，在编写程序代码时也是如此。在上篇文章中，我们讲解了使用 If-Then 结构来进行判断，也举了很多个小例子。我们看到，随着条件的增多，代码的结构也变得复杂或难以理解。幸运的是，VBA 为我们提供了 Select Case 结构，让我们根据多个条件判断选择时，能够写出容易理解的代码。

上篇文章的结尾举了一个例子：按照一定的规则，根据员工的工龄来确定员工的年休假天数：

工龄（年）	年休假天数
10 年及以下	5
10～20	10
20～25	15
25 年以上	20

现在，我们使用 Select Case 结构来编写代码，实现相同的功能。首先看看示例代码：

示例代码 1

```
Sub NianXiuTianWithSelectCase()  
    '声明变量,用来表示年休天数和工龄  
    Dim lngDays As Long  
    Dim lngYears As Long  
  
    lngYears = Range("A1").Value  
  
    '根据工龄确定相应的年休天数
```

```

        Select Case lngYears
            Case 0 To 10
lngDays = 5
            Case 10 To 20
lngDays = 10
            Case 20 To 25
lngDays = 15
            Case Is > 25
lngDays = 20
        End Select

        MsgBox "工龄：" & lngYears & vbCrLf & "年休天数：" & lngDays
    End Sub

```

同样，您可以在工作表单元格 **A1** 中输入代表工龄的数字，运行代码后，会显示相应的年休天数，就像在上篇文章中所示的图一样。

与上篇文章中的代码相比，既简单又易理解。

Select Case 结构的语法如下：

```

Select Case 条件（表达式）
    [Case 表达式值 1
        [语句块]
    [Case 表达式值 2
        [语句块]
    .....
    [Case Else
        [语句块]
End Select

```

说明：

- 当某个 **Case** “表达式值”满足“条件（表达式）”的结果时，执行相应的语句块，并退出 **Select Case** 结构。如果没有满足“条件（表达式）”的结果时，就执行 **Case Else** 后的语句块。

- 方括号表示可选。Case 子句可以有多个，Case Else 子句可以有也可以没有。

Case 后的表达式可以是数值范围，如示例代码 1 所示。下面再举一些简单的例子来说明 Select Case 结构的用法，这些例子本身没有多大的实用价值，只是方便您理解 Select Case 结构。

Case 后的表达式也可以是逗号分隔的数值或字符串，如下面的示例代码所示：

示例代码 2：逗号分隔数值

```
Sub NumWithSelectCase()  
    Select Case Range("A1").Value  
        Case 1, 3, 5  
            MsgBox "单元格 A1 中的值是 5 以内的奇数."  
    End Select  
End Sub
```

如果单元格 A1 中是数值 1 或 3 或 5，运行代码后就会弹出“单元格 A1 中的值是 5 以内的奇数。”的消息框。

示例代码 3：逗号分隔字符

```
Sub CharWithSelectCase()  
    Select Case Range("A1").Value  
        Case "A", "E", "I", "O", "U"  
            MsgBox "单元格 A1 中是大写元音字母."  
        Case Else  
            MsgBox "单元格 A1 中不是大写元音字母."  
    End Select  
End Sub
```

如果单元格 A1 中是字母“A”、“E”、“I”、“O”、“U”之一，运行代码后就会弹出“单元格 A1 中是大写元音字母”的消息框。

我们还可以嵌套使用 Select Case 结构，如下面的示例代码所示：

示例代码 4：嵌套的 Select Case 结构

```
Sub qtWithSelectCase()
```

```

Select Case Range("A1").Value
    Case "工作表"
        Select Case Worksheets.Count
            Case 1
                MsgBox "工作簿中有 1 个工作表"
            Case 2
                MsgBox "工作簿中有 2 个工作表"
            Case Else
                MsgBox "工作簿中的工作表超过了 2 个"
        End Select
    Case Else
        MsgBox "请在单元格 A1 中输入文本：工作表"
End Select
End Sub

```

如果单元格 A1 中的值为文本“工作表”，就判断工作簿中的工作表数量（语句 Worksheets.Count），并在消息框中显示相应的信息。

说明：

- Select Case 结构可以多层嵌套，但要注意每个 Select Case 语句要对应一条 End Select 语句。
- 在书写嵌套结果时，在代码中使用缩进排列可以使程序清晰易懂。

如果 Case 子句后只有一条指令，那么可以将指令与 Case 书写在一行上。例如，我们重写示例代码 2 如下：

```

Sub NumWithSelectCase()
    Select Case Range("A1").Value
        Case 1, 3, 5: MsgBox "单元格中的数字是 5 以内的奇数."
    End Select
End Sub

```

说明：

- 如果将 Case 子句与后面的指令写在一行上，那么必须在它们之间加上冒号 (:)。

23. 有限次的循环

下面，该介绍 VBA 的循环语句结构了。这里，简要地谈谈有限次的循环结构，也就是 For-Next 循环。还是先看一个例子。

我们如果想要在工作表单元格区域 A1:A10 中依次填写数字 1 至 10，可以使用下面的代码：

```
Sub ForNextTest1()  
    Dim i As Integer '声明整型变量 i  
  
    '使用循环为单元格填充数字  
    For i = 1 To 10  
Cells(i, 1).Value = i  
        Next i  
    End Sub
```

运行后，当前工作表中单元格区域 A1:A10 会填充数字 1 至 10。上述代码中，Cells(i, 1)代表第 i 行第 1 列的单元格，语句 Cells(i, 1) = i 将 i 的值放入第 i 行第 1 列的单元格中。

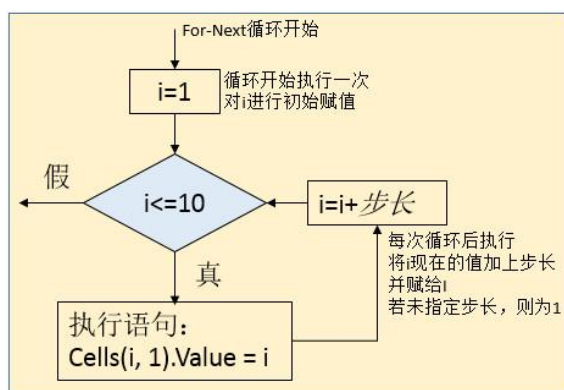
For-Next 循环的基本语法结构如下：

```
For 计数变量=开始值 To 结束值 [step 步长]  
    [语句块]  
    [Exit For]  
    [语句块]  
Next [计数变量]
```

说明：

- 按照惯例，方括号中的内容表示可选项。
- 步长可以是正值或者负值。正确地设置某个值，以满足特定的需求；若不设置，默认循环计数变量每次增加 1，如上面的例子所示。
- ExitFor 语句表示提前退出循环。
- For-Next 循环从计数变量设置的开始值开始，按步长增加计数变量值，直至达到结束值时终止循环。

将前面示例中的循环用框图表示如下：



下面的代码求 1 至 100 的和，并显示结果。

```
Sub ForNextTest2()

    Dim sum As Integer '声明存储结果值的变量

    Dim i As Integer '声明计数变量

    sum = 0 '赋初值

    For i = 1 To 100

sum = sum + i

    Next i

    MsgBox "1 至 100 的和为: " & sum

End Sub
```

下面的代码求 1 至 100 之间的偶数和并显示结果。

```
Sub ForNextTest3()
```

```

Dim sum As Integer '声明存储结果值的变量

Dim i As Integer '声明计数变量

sum = 0 '赋初值

For i = 0 To 100 Step 2
sum = sum + i
Next i

MsgBox "1 至 100 之间的偶数和为: " & sum

End Sub

```

其中，计数变量 i 的值从 0 开始，依次为 2、4、6、8 等，最后达到 100。但是，在循环结束时，i 的值为 102。有兴趣的朋友可以添加一个语句，打印出循环结束后 i 的值。

我们使计数变量从 100 开始，让步长为负值递减，也可以得到同样的效果：

```

Sub ForNextTest4()

Dim sum As Integer '声明存储结果值的变量

Dim i As Integer '声明计数变量

sum = 0 '赋初值

For i = 100 To 0 Step -2
sum = sum + i
Next i

MsgBox "1 至 100 之间的偶数和为: " & sum

End Sub

```

For-Next 循环可以嵌套其他的 For-Next 循环。如下例所示，将当前工作表中的单元格区域 A1:J10 中的内容都填充为数字 1。

```

Sub ForNextTest5()

Dim i As Integer '声明计数变量

Dim j As Integer '声明计数变量

For i = 1 To 10
For j = 1 To 10
Cells(i, j).Value = 1 '填充单元格

```

```
        Next j
    Next i
End Sub
```

下面的例子说明使用 **Exit For** 语句退出循环：

```
Sub ForNextTest6()
    Dim i As Integer '声明计数变量
    For i = 1 To 10
        If Cells(i, 1).Value = 0 Then '判断单元格中的值为 0
            Exit For
        End If
    Next i
    MsgBox "单元格 A" & i & "中的值为 0."
End Sub
```

在单元格区域 **A1:A10** 中，如果某个单元格的值为 **0**，则退出循环，并继续执行 **Next** 后面的语句，本例中是 **MsgBox** 函数。

24. 有条件的循环（1）

在上一篇文章中，我们介绍的 **For-Next** 循环，能够按照指定的数值进行有限次的循环。下面介绍在满足指定的条件时才执行相应代码块的循环结构，先来看看 **Do While** 循环。

按照惯例，先举一个例子。下面的代码从当前单元格开始，逐行显示单元格的内容，直到碰到空单元格为止。如果第一个单元格就为空，则不会执行循环。

```
Sub DoWhile0()  
    Dim i As Integer '声明变量  
    i = 1 '初始化变量  
  
    Do While Cells(i, 1).Value <> ""  
        MsgBox "单元格 A" & i & "的内容为:" & Cells(i, 1).Value  
        i = i + 1  
    Loop  
  
End Sub
```

例如，在单元格区域 **A1:A5** 中依次输入数字 1、2、3、4、5，将当前单元格置于 **A1** 单元格（如图 24.1 所示），运行上面的代码，将依次显示数字 1、2、3、4、5。

	A	B	C
1	1		
2	2		
3	3		
4	4		
5	5		
6			
7			
8			

将A1置为当前
单元格后，运
行代码

图 24.1

Do While 循环的基本语法结构如下：

Do [While 条件语句]

[语句块]

[Exit Do]

[语句块]

Loop

或者：

Do

[语句块]

[Exit Do]

[语句块]

Loop [While 条件语句]

说明：

- 按照惯例，方括号中的内容表示可选项。
- 条件语句可以放在循环的开始处或者结尾处。如果放在开始处，那么先评估条件，满足条件要求，就执行循环里面的语句；这种情况下，当始终不满足条件时，就有可能一次也不会执行循环里面的语句。如果放在结尾处，则至少会执行一次循环里面的语句。
- ExitDo 语句表示提前退出循环。

将条件放置在结尾处，前面的示例代码修改如下：

Sub DoWhile01()

Dim i As Integer '声明变量

```

i = 1 '初始化变量

Do

    MsgBox "单元格 A" & i & "的内容为:" & Cells(i, 1).Value

    i = i + 1

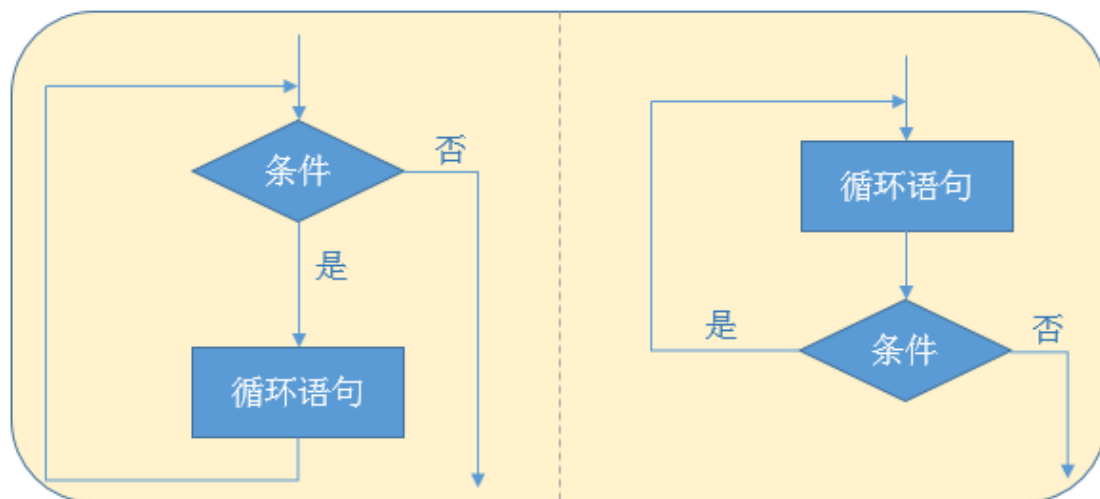
Loop While Cells(i, 1).Value <> ""

End Sub

```

此时，如果单元格 A1 为空，也会显示一条消息框，然后退出循环。

用框图分别表示 Do While 循环的两种语法形式如下：



如果熟悉了 Excel 中的常用对象及其属性，也可以将上面的代码修改如下，实现相同的结果。

```

Sub DoWhile1()

    Do While ActiveCell.Value <> ""

        MsgBox "当前单元格的内容为:" & ActiveCell.Value

        ActiveCell.Offset(1, 0).Activate

    Loop

End Sub

```

其中，Offset 属性表示下移一个单元格，即当前单元格下面的单元格。（单元格的常用属性将在后续文章中详细介绍）

将条件放置在结尾处，前面的示例代码修改如下：

```
Sub DoWhile2()  
    Do  
        MsgBox "当前单元格的内容为:" & ActiveCell.Value  
    ActiveCell.Offset(1, 0).Activate  
    Loop While ActiveCell.Value <> ""  
End Sub
```

下面，为更深入理解 Do While 循环，我们来改写上一篇中用来说明 For-Next 循环的示例。

示例 1：如果想要在工作表单元格区域 A1:A10 中依次填写数字 1 至 10，也可以使用下面的包含 Do While 循环的代码：

```
Sub DoWhile3()  
    Dim i As Integer '声明变量  
  
    i = 1 '给变量赋初始值  
  
    '循环  
    Do While i <= 10  
Cells(i, 1).Value = i  
        i = i + 1  
    Loop  
End Sub
```

注意，与 For-Next 循环不同的是，在进入循环之前，需要先给变量 i 赋初始值。在循环中，还需要能够增加变量的值。

示例 2：下面的代码使用 Do While 循环求 1 至 100 的和，并显示结果。

```
Sub DoWhile4()  
    '声明变量  
    Dim i As Integer
```

```

Dim sum As Integer

'给变量赋初值
i = 1
sum = 0

'循环并显示结果
Do While i <= 100
sum = sum + i
    i = i + 1
Loop
MsgBox "1 至 100 的和为: " & sum
End Sub

```

示例 3： 下面的代码使用 Do While 循环求 1 至 100 之间的偶数和并显示结果。

```

Sub DoWhile5()
'声明变量
Dim i As Integer
Dim sum As Integer

'给变量赋初值
i = 1
sum = 0

'循环并显示结果
Do While i <= 100
    If (i Mod 2 = 0) Then
sum = sum + i
        End If
        i = i + 1
Loop

```

```
MsgBox "1 至 100 的和为: " & sum
```

```
End Sub
```

在编写程序时，我们可以使用多种方法实现想要的结果，您可以根据具体情况选择实现方式，这也是编程的一大美妙之处。

25. 有条件的循环（2）

下面继续介绍在满足指定的条件时才执行相应代码块的循环结构，接着来看看 Do Until 循环。

Do Until 循环与 Do While 循环的结构相似，最本质的区别在于循环条件的判断。顾名思义，在 Do While 循环中，当条件为真（True）时，就执行循环；而在 Do Until 循环中，执行循环直到条件为真（True）时，退出循环。

Do Until 循环的基本语法结构如下：

```
Do [Until 条件语句]
```

```
    [语句块]
```

```
    [Exit Do]
```

```
    [语句块]
```

```
Loop
```

或者：

```
Do
```

```
    [语句块]
```

```
    [Exit Do]
```

```
    [语句块]
```

```
Loop [Until 条件语句]
```

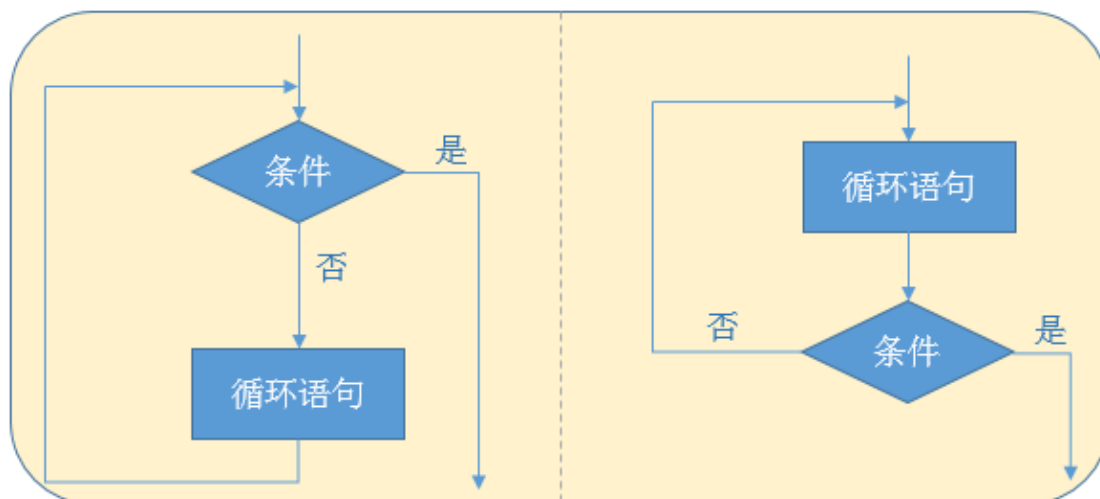
说明：

- 按照惯例，方括号中的内容表示可选项。
- 条件语句可以放在循环的开始处或者结尾处。如果放在开始处，那么先评估条件，若不满足条件要求，就执行循环里面的语句；这种情况下，当始终满足条件时，就有可能一次也不会执行循环里面的语句。如果放

在结尾处，则至少会执行一次循环里面的语句。

- ExitDo 语句表示提前退出循环。

用框图分别表示 Do Until 循环的两种语法形式如下：



与上一篇文章介绍的 Do While 循环相比较，结构一样，只是执行循环的条件相反。

接下来，为方便对两种循环结构的理解，我们改写上一篇文章中的部分示例。

示例 1：如果想要在工作表单元格区域 A1:A10 中依次填写数字 1 至 10，也可以使用下面的包含 Do Until 循环的代码：

```
Sub DoUntil1()  
    Dim i As Integer '声明变量  
  
    i = 1 '给变量赋初始值  
  
    '循环  
    Do Until i > 10  
        Cells(i, 1).Value = i  
        i = i + 1  
    Loop  
End Sub
```


注意，与 For-Next 循环不同的是，在进入循环之前，需要先给变量 i 赋初始值。在循环中，还需要能够增加变量的值。

示例 2： 下面的代码使用 Do Until 循环求 1 至 100 的和，并显示结果。

```
Sub DoUntil2()  
    '声明变量  
    Dim i As Integer  
    Dim sum As Integer  
  
    '给变量赋初值  
    i = 1  
    sum = 0  
  
    '循环并显示结果  
    Do Until i > 100  
        sum = sum + i  
        i = i + 1  
    Loop  
    MsgBox "1 至 100 的和为: " & sum  
End Sub
```

示例 3： 下面的代码使用 Do Until 循环求 1 至 100 之间的偶数和并显示结果。

```
Sub DoUntil3()  
    '声明变量  
    Dim i As Integer  
    Dim sum As Integer  
  
    '给变量赋初值  
    i = 1  
    sum = 0
```

'循环并显示结果

```
Do Until i > 100
    If (i Mod 2 = 0) Then
sum = sum + i
    End If
    i = i + 1
Loop
MsgBox "1 至 100 的和为: " & sum

End Sub
```

26. 在对象中循环—— For Each-Next 结构的使用

前面我们介绍过 Excel 的对象模型及一些常用的对象，也讲解过对象变量的概念及声明对象变量的方法。其实，在绝大多数情况下，我们都是通过使用 Excel VBA 操纵 Excel 对象来达到我们的目的。例如，我们可以操作工作表对象，为工作表命名、排序工作表、统计工作表个数；我们可以操作单元格对象，在单元格区域中填充内容、查找有指定内容的单元格；等等。

为方便操控 Excel 对象，VBA 提供了 For Each-Next 结构，可以在对象组成的集合中循环，给集合中的所有对象执行操作，或者集合中满足相关条件的对象执行操作。

使用 For Each-Next 结构，我们无需知道集合中对象的数目，只需要声明相应的对象变量，编写执行操作的指令。

For Each-Next 结构的语法如下：

```
For Each 对象变量 In 对象集合
    [语句块]
    [Exit For]
    [语句块]
Next [对象变量]
```

说明：

- 按照惯例，方括号中的内容表示可选项。
- 对象变量在使用前需要进行声明。
- Exit For 语句表示提前退出循环。
- 无需知道集合中的对象数目。
- 可以与其它语法结构相互嵌套。

下面通过示例来演示 For Each-Next 结构的使用。

示例 1： 下面的代码遍历当前工作簿中的工作表并依次显示工作表的名字。

```
Sub ForEach1()  
    Dim wks As Worksheet '声明工作表对象变量  
  
    '遍历工作表集合并依次显示工作表的名字  
    For Each wks In Worksheets  
        MsgBox "工作表的名字是:" & wks.Name  
    Next wks  
End Sub
```

示例 2： 通过改进上面的示例 1，在显示工作表名字时表明是第几个工作表，最后统计出工作簿中的工作表数，代码如下。

```
Sub ForEach2()  
    Dim wks As Worksheet '声明工作表对象变量  
    Dim i As Long '声明用于统计工作表数的变量  
  
    i = 0 '给变量赋初始值  
  
    For Each wks In Worksheets  
        i = i + 1  
        MsgBox "第" & i & "个工作表的名字是:" & wks.Name  
    Next wks  
  
    MsgBox "本工作簿共有" & i & "个工作表."  
End Sub
```

示例 3： 下面的代码遍历当前工作表的单元格区域 A1:A5，并依次显示每个单元

格中的内容。

```
Sub ForEach3()  
    Dim cell As Range '声明单元格对象变量  
  
    '遍历单元格区域 A1:A5 并依次显示单元格内容  
    For Each cell In Range("A1:A5")  
        MsgBox cell.Value  
    Next cell  
End Sub
```

示例 4：下面的代码与示例 3 的效果相同，但我们设置了代表单元格区域的对象变量并赋值，让 For Each 结构在对象变量代表的区域内循环遍历。

```
Sub ForEach4()  
    Dim cell As Range, rng As Range '声明单元格对象变量  
  
    '设置 rng 变量的值  
    Set rng = Range("A1:A5")  
  
    '遍历 rng 对象变量代表的单元格区域并依次显示单元格内容  
    For Each cell In rng  
        MsgBox cell.Value  
    Next cell  
End Sub
```

示例 5：下面的代码结合 If-Then 结构判断满足相应条件的单元格数。

如下图 26.1 所示，要统计成绩大于 80 分的学生人数。

	A	B
1	姓名	成绩
2	张三	90
3	李四	78
4	王五	85
5	赵六	66
6	东邪	75
7	西毒	95
8	南帝	67
9	北丐	99

图 26.1

运行下面的代码即可。

```
Sub ForEach5()
    Dim cell As Range, rng As Range '声明单元格对象变量
    Dim i As Long '声明计数变量

    '设置 rng 变量的值
    Set rng = Range("B2:B9")
    i = 0

    '遍历 rng 对象变量代表的单元格区域
    '并判断单元格中的值是否大于 80
    For Each cell In rng
        If cell.Value > 80 Then
            i = i + 1
        End If
    Next cell

    MsgBox "共有" & i & "名学生超过 80 分."
End Sub
```

代码在 For Each 循环结构中使用 If-Then 语句来检查每个单元格中的值，如果其值大于 80，变量 i 就增加 1。

27. 看看 VBA 的 Sub 过程和 Function 过程

经过前面的一系列文章的示例学习，我们应该已经熟悉了 Sub 过程。通常我们看到的 Sub 过程的基本形式是：

```
Sub 过程名称()
```

```
[此处放置代码语句]
```

```
Exit Sub
```

```
End Sub
```

这也是我们在编写大多数 VBA 程序时用到的结构。其中：

- 过程名称为编程人员给过程命名的名称，其命名规则与变量名称的规则相同。在同一模块中的过程名称不能同名。
- Sub 和 End Sub 配套出现，它们之间构成一个完整的程序过程。
- 过程中的代码可以放置 Exit Sub 语句，表示提前退出过程。

通常，在 VBE 编辑器中，写下 sub 和过程名称，按下回车键后，会自动添加 End Sub 语句。Sub 过程用来执行操作，而 Function 过程可以返回值，创建自定义函数。Function 过程的基本语法形式为：

```
Function 过程名称()
```

```
[此处放置代码语句]
```

```
[过程名称=表达式或值]
```

[Exit Function]

[此处放置代码语句]

[过程名称=表达式或值]

End Function

可以看出，其形式与 Sub 过程的基本形式相似，只是将关键词 Sub 换成了 Function。注意：

- 在 Function 过程的代码语句中，应至少有一次把某个确定的值赋值给函数过程的名称。
- 建议避免使用 Excel 内置函数名称、单元格引用、单元格区域的名称作为函数过程的名称。
- Function 过程不能对单元格区域进行修改，不能执行与对象相关的操作。
- Function 过程可以应用在工作表公式中，就像 Excel 的内置函数一样；Function 过程也可以应用在 VBA 代码中。
- 过程代码中可以放置 Exit Function 语句，表示提前退出过程。
- 按惯例，方括号表示可选。

下面的示例代码统计当前可见工作簿的数量：

```
Function lWkbNum()
```

```
    Dim lCount As Long '声明计数变量
```

```
    Dim wkb As Workbook '声明工作簿对象变量
```

```
    '遍历应用程序中的工作簿
```

```
    For Each wkb In Application.Workbooks
```

```
        '如果为可见工作簿则增加 1
```

```
        If wkb.Windows(1).Visible Then
```

```
lCount = lCount + 1
```

```
        End If
```


Next wkb

'将结果赋值给函数名

IWkbNum = ICount

End Function

就像运用 Excel 内置函数一样，在工作表中输入“=IWkbNum()”，显示当前已打开的可见工作簿的数量，如下图 27.1 所示。

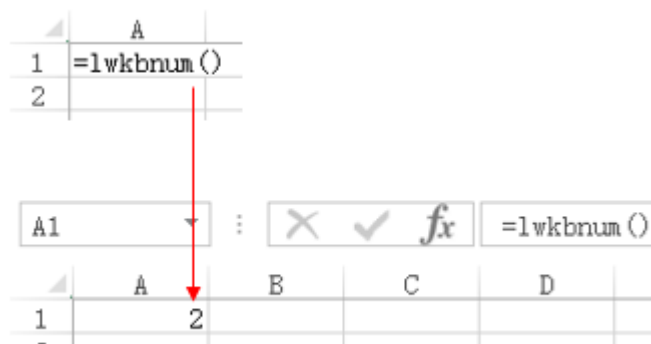


图 27.1: 在单元格 A1 中输入=IWkbNum(), 显示 2, 即当前打开的可见工作簿数为 2 个

下面的代码在 Sub 过程中调用自定义的 IWkbNum 函数:

Sub testIWkbNum()

MsgBox "当前可见工作簿的数量为:" & IWkbNum

End Sub

运行后显示如下图 27.2 所示的消息框:



图 27.2

由于之前的一系列文章中都是使用的 Sub 过程作为的例子, 所以本文详细介绍了 Function 过程的最基础知识, 给您一个初步的认识。

不管是 Sub 过程还是 Function 过程, 都可以带有参数, Function 过程还可以设置返回值的类型, 我们将在后续逐步深入这些知识。

28. 简单问答

Excel 的某些行为往往会让用户感到疑惑、对 VBA 不甚了解的人在初次使用代码时也会碰到一些貌似奇怪的问题。下面我们就从最简单的说起。

问题 1：为什么在打开有些 Excel 文件时出现安全警告？

在打开 Excel 文件时，我们时常会看到下图所示的安全警告消息框。根据 Excel 的版本不同或者在 Excel 2007 以上版本中打开 Excel 2003 版本的文件，大致会出现图 28.1 中所示的 3 类消息框。



图 28.1

这是因为我们打开的工作簿中含有代码，并且在“安全性”（Excel 2003 版）或者“信任中心”中对宏进行了较高级别的设置。默认情况下，为了防止宏病毒的侵害，Excel 会自动采用较高级别的宏安全设置。

如果我们事先知道宏是安全的，就可以单击启用宏。如果不太确定宏的安全性，可以先选择禁用宏，然后在 VBE 中查看宏代码，确认是否安全。

问题 2：为什么在保存 Excel 文件时会出现额外的提示信息？

在 Excel 2007 以上的版本中，当我们保存含有宏代码的工作簿时，有时会出现下图 28.2 所示的信息框。

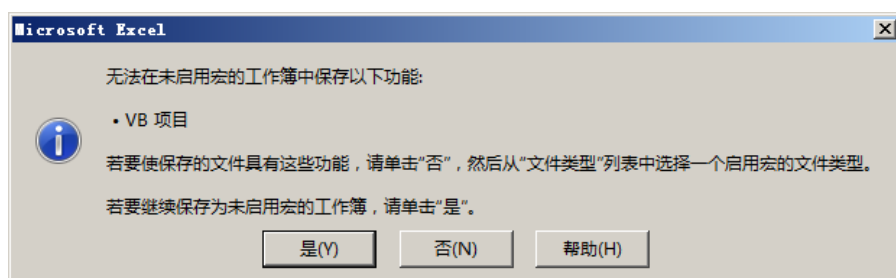


图 28.2

在 Excel 2007 以上的版本中，Microsoft 采用了新的文件格式，扩展名为.xlsx 的文件中不包含宏代码，不能将代码添加到具有.xlsx 扩展名的 Excel 文件中，即使将宏代码添加到.xlsx 的文件中，在保存时 Excel 也会将代码丢弃。

因此，在以.xlsx 为扩展名保存 Excel 文件时，如果工作簿中含有宏代码，就会出现上图所示的信息框。

在 Excel 2007 以上的版本中，需要使用扩展名为.xlsm 保存包含宏代码的文件。

问题 3：为什么 Excel 界面中没有“开发工具”选项卡？

通常，Excel 默认不会在功能区中显示“开发工具”选项卡。您可以选择“文件”——“选项”命令后，在弹出的对话框中选取“开发工具”，使“开发工具”

选项卡显示在 Excel 界面中，如下图 28.3 所示。

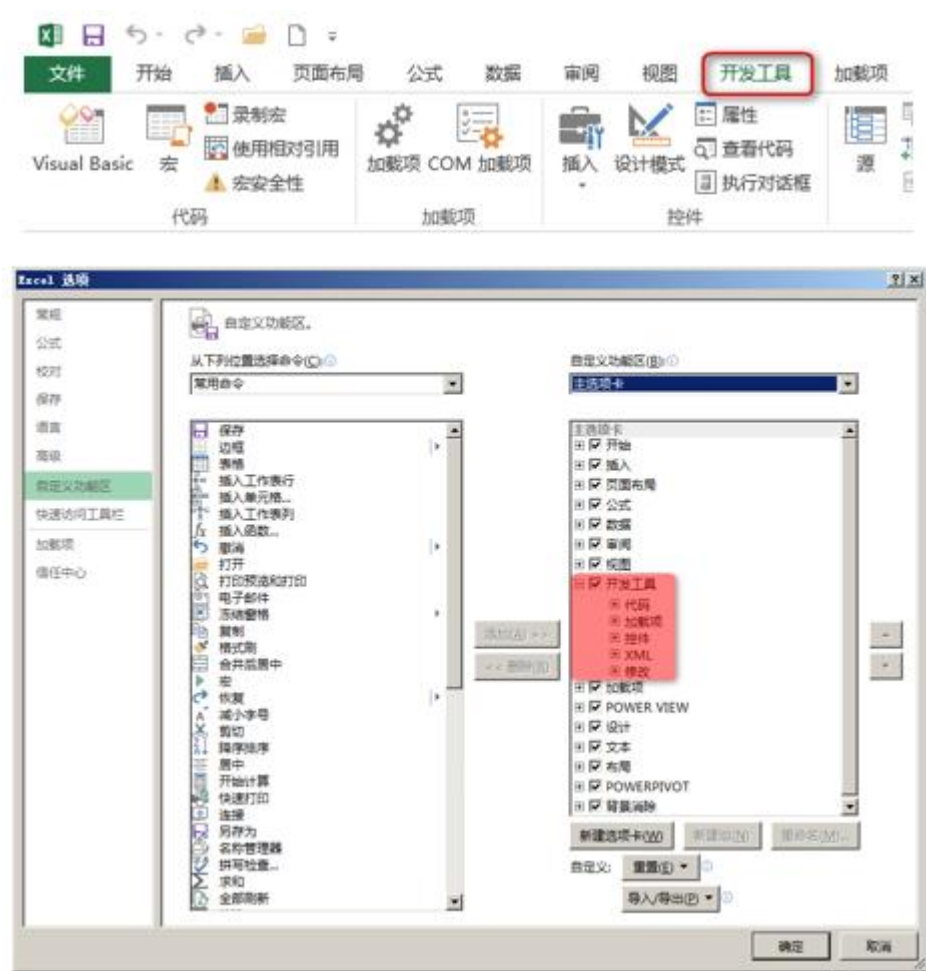


图 28.3

问题 4：VBA 代码应该放置在哪里？

一般来说，VBA 代码可以放置在任意的代码模块里，无论是代表工作簿或工作表的对象模块，还是标准的代码模块，没有硬性的规定或约束。

VBE 界面中工程资源管理器里的常见模块如下图 28.4。

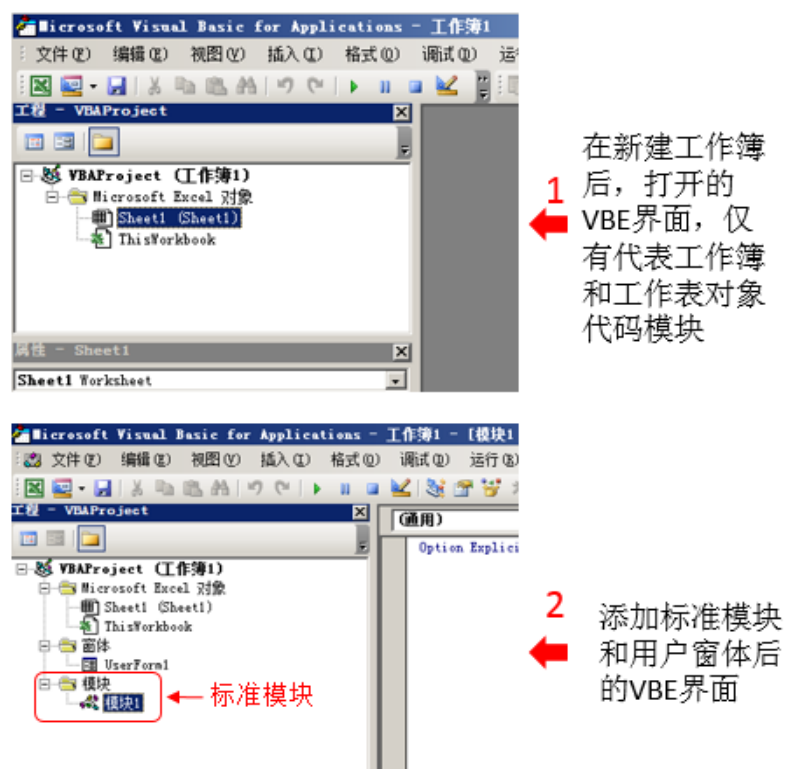


图 28.4

通常，我们应该把代码放置在标准模块中，包括 Sub 过程和 Function 过程，这也便于程序的良好组织。在工作簿对象代码模块和工作表对象代码模块中编写响应工作簿事件和工作表事件的代码，在用户窗体中编写对窗体和窗体中的控件进行操作的代码。

问题 5：如何运行 VBA 代码？

对于初学者来说，运行 VBA 代码最基本的方式有下图 28.5 所示的 3 种方式。

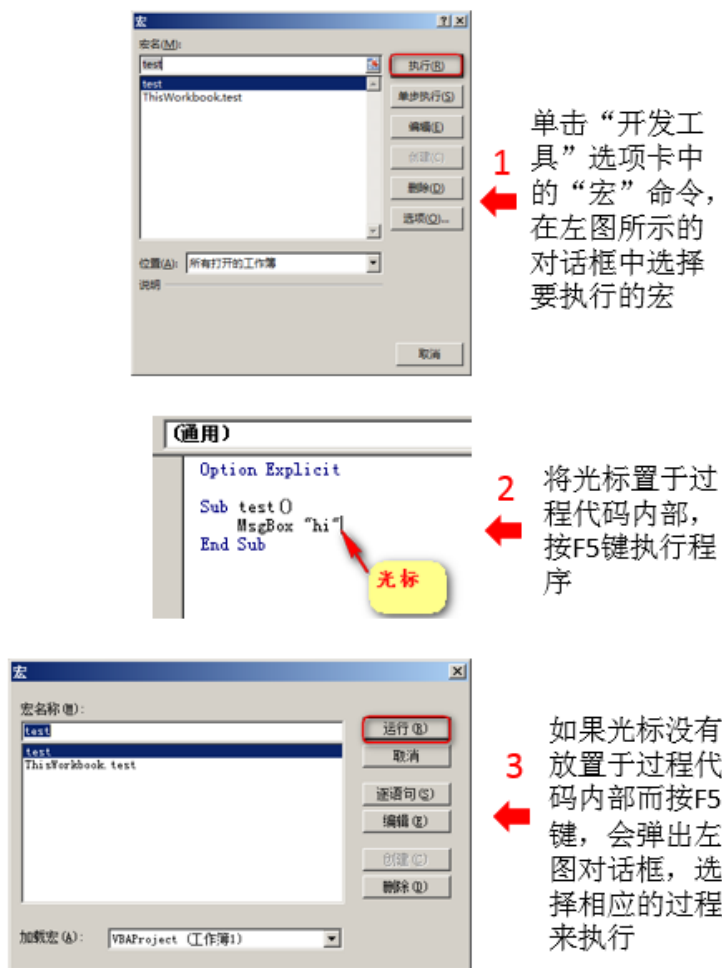


图 28.5

此外，在 **3. VBA 代码之家** 中，还介绍了几种基本的方法。

当然，运行 VBA 代码的方式有很多，包括自定义工具栏、功能区、在工作表中放置控件或图像，等等，我们将在单独的专题中详细介绍。

结语

终于，Excel VBA 解读系列第一季：**基础入门篇**全部文章写完了。第一季，是初学季。虽然这些知识很基础，但也花了我不少的时间和精力。怕写得太深，初学者看不懂；怕写得太死板，大家不愿看；怕写得很简单，让人“嗤之以鼻”，觉得没有“内涵”总之，断断续续，写完了，就这么些内容，任由你评说，也欢迎你评说。

下面列出已完成的第一季的目录和每篇文章的主要内容。

1.引子

2.Excel 中的加强版录像机

简单介绍 Excel 的录制宏功能。

3.VBA 代码之家

简单介绍 VBE 界面，输入代码，执行代码，立即窗口调试代码。

4.如影随形的帮助

VBA 的帮助系统及使用，网上资源。

5.认识对象和 Excel 对象模型

理解 Excel 的对象及其属性和方法、集合以及对象模型的概念。

6.看看 Excel 的那些常用对象

初步介绍 Application 对象、Workbook 对象及 Window 对象。

7.看看 Excel 的那些常用对象（续 1）

初步介绍 Worksheet 对象及相关属性。

8.看看 Excel 的那些常用对象（续 2）

初步介绍 Range 对象及相关属性。

9.看看 Excel 的那些常用对象（续 3）

初步介绍 Comment 对象、Chart 对象及相关属性。

10.神奇的句点

介绍 VBA 中的点运算符：引用对象的规则。

11.神奇的句点（续）

介绍 VBA 中的点运算符：引用方法和属性、以及方法或属性的参数表示。

12.再回首，说透对象

形象说明对象的方法及属性的使用。

13.一个简单的 VBA 程序

通过一个简单的 VBA 程序介绍程序的基本结构、程序的语言元素。

14. VBA 的数据类型

介绍并列出了 VBA 的基本数据类型。

15.变量和常量

介绍变量和常量，及其命名规则和声明方式。

16.VBA 的运算符

介绍算术运算符、关系运算符、逻辑运算符、字符串运算符及其规则，运算符的优先级，赋值运算符。

17.谈谈对象变量

如何声明对象变量、给对象变量赋值，以及使用对象变量的好处。

18.With ... End With 结构，专为对象而生

介绍 With ... End With 结构的使用。

19.学生管理系统开发 V1.0

初步规划系统，随着知识的深入，逐步完善。

20.用户交互初体验—MsgBox 函数

详细介绍 MsgBox 函数。

21.向左走，向右走—使用 If 语句选择介绍 If 语句。

22.方便的“多选一”结构—Select Case 结构

介绍 Select Case 结构。

23.有限次的循环

介绍 For-Next 语句。

24.有条件的循环（1）

介绍 Do While 循环语句结构。

25.有条件的循环（2）

介绍 Do Until 循环语句结构。

26.在对象中循环——For Each-Next 结构的使用

介绍 For Each-Next 循环语句结构。

27.看看 VBA 的 Sub 过程和 Function 过程

初步介绍 Sub 过程和 Function 过程的简单使用。

28.简单问答

解答初学者遇到的一些疑惑。

不管你怎么说，VBA 入门其实很简单。你熟悉好我们在第一季介绍的“这么一点儿”内容，你就入门了，就可以摸索着试写一些简单的小程序来优化你的工作了。

然而，要更好地利用好 VBA 这个强大的工具，还需要不断地学习和实践。

关于完美 Excel

完美 Excel 是我创建的一个微信公众号，自 2017 年 5 月 15 日开始，每天推送一篇 Excel 与 VBA 技术和应用方面的文章。目前，共有 300 余篇实用文章可供广大 Excel 爱好者和使用者学习交流。这本电子书就是根据完美 Excel 上发表的《Excel VBA 解读》系列文章中的前 29 篇整理而成的。

每天早晨，完美 Excel 微信公众号：**excelperfect** 都会推送一篇关于 Excel 与 VBA 的相关文章。如果你有兴趣学习 Excel 和 VBA 的相关知识和实战技巧，可以关注完美 Excel 微信公众号，绝对不会让你失望！

可以通过下列方法关注[完美 Excel]微信公众号：

方法 1—在通讯录中搜索“完美 Excel”或者“**excelperfect**”后点击关注。

方法 2—扫一扫下面的二维码

