# EXCEL 与 VBA 程序设计

| | |
|---|---|
| 标题： | Excel 与 VBA 程序设计 |
| 版本： | 0.10 |
| 作者： | 马维峰 |
| Email： | mweifeng@gmail.com |
| Blog： | http://maweifeng.cnblogs.com |
| 时间： | 开始时间：2005-6-20<br>版本 0.10 |

# 关于本书的说明

笔者大概从 98 年开始学习 VB，从喜欢到失望，从失望到欣喜，从欣喜到平淡，大概是很多自己一样的程序员的学习心路。而对于 Office 的强大功能，对于 VBA，领会的却很晚。例如 Excel，虽然也一直知道其功能很强大，但到底如何强大，有什么有别于其他同类软件的特色，却不甚清楚。大概在 2003 年，应工作需要，仔细查阅了一些 Excel 的资料，开始学习 Excel 数据处理和 VBA 开发。因为当时已是一个熟练的 VB 程序员，所以 VBA 语法并不是难点和重点，而在很多书中没有很清楚讲解的问题，例如一个工作表内的某些数据如何获得，某个或某几个单元格的值怎么高效的获取和赋值，如何打开关闭 Excel 文件，如何正确部署最后的程序，如何绘制复杂的图表，等等诸如此类的问题，反而经常会困惑自己很久。因此，本书将以笔者的学习经验为依据，以一个程序员的角度，讲解 Excel VBA 开发的种种问题，并对一些笔者在实际中遇到的大多数 Excel VBA 开发的书中较少涉及的内容作深入探讨，对于一些设计问题、效率问题、程序风格，书中都会给出笔者的建议。

书中关于运行效率的说明，都经过笔者亲自测试，具体测试方法在运行效率一节有详细说明。书中包括的代码风格之类的部分观点只是个人喜好问题，特此说明，读者可以根据自己的判断取舍。写作过程中，有时会觉得有太多的问题需要说明，却限于篇幅，不能一一展开；有时又不知如何下笔，不知如何才可以清楚简洁的讲清楚一个问题。对于很多自己的经验或者教训，只能在合适的时候插入只言片语，古人言“中有苦心而不能显”，“中有调剂而人不知”，大概如是。

书中的代码、例子和文字是紧密配合的，没有了这些内容，也就失去了全书的灵魂所在。很多的说明文字必须通过代码来体会，这是笔者很多年来自己的体会。

本书不求对 Excel 和 VBA 面面俱到的介绍，而且这也是不可能的。从内容选择和取舍来说，本书更注重实用，从笔者的经验出发，从应用的角度来介绍 Excel VBA 的内容，而不是相反。

目前写一本关于 VBA 的书好像有些不合时宜，毕竟 VBA 和 VB 一样，是属于“落后”和“过时”的技术，VB 6 的使用者，如笔者，大多已经转移到.net 平台之下，那么，VBA 的命运如何，我们还不得而知？但至少，在很长的一段时间内，作为 Office 开发的方式，

VBA 和 VSTO 应该会共存，而对于非专业程序员，首选应该还是 VBA，此为其一。其二，VBA 开发的核心在于 Office 的对象模型的掌握，而这也是本书的重点之所在。

本书的读者应该可以较熟练的使用 Excel，例如可以使用公式，可以自定义公式；熟悉基本的 Excel 的概念和名词，例如宏、加载宏。对于基本没有程序设计经验的读者，第二章比较系统的介绍 VBA 语法和集成开发环境（IDE），对于熟悉 VBA 或 VB 的读者，可以略过这一章。其实对于 Office 系列的开发，语法只是很小的一部分，主要的难点和问题在于相应的对象模型及其应用，所以书中的大多数内容其实只是围绕 Excel 对象模型的解释和讲解。

详细来说，本书的读者可以细分为：

1. 应用 Excel 作为基础平台，提供相应解决方案的程序员；
2. 各类科研工作者，应用 Excel 进行数据处理，这其实是本书最初的写作动机；
3. 在各类企事业部门需要进行大量机械性和重复性的信息、数据处理工作，希望可以利用 Excel 自动化这些工作的人员；
4. 其他对 Excel 自动化和 VBA 编程感兴趣的读者。

本书使用的 Excel 版本是 2003，但书中绝大多数内容并未涉及 Excel 2000 之后的内容；除了少数内容，书中所介绍的内容也与 Excel 97 内容兼容。对于较新版本的内容，在介绍时都尽可能的做了说明。

最后祝学习愉快！

*** *** *** *** *** *** *** ***

**关于书中的符号、提示、代码等的说明**

- 所有关于菜单工具栏的操作以以下形式表示：

***“文件 － 打开”***

- 对于一些技巧，需要提醒说明的问题，文中都已以下形式做了说明：

打开 IDE 环境的方法

- **通过*“工具 — 宏 — VISUAL BASIC 编辑器”***
- **通过快捷键*“ALT + F11”***
- **右键单击工具栏，选择“VISUAL BASIC”，此工具栏有录制宏，打开 VBA IDE 等的快捷按钮**

- 程序代码以以下方式显示（黄底、字体为 Courier New、5 号）：

```
#001 Function MyAdd(varA, varB) As Variant
#002     MyAdd = varA + varB
#003 End Function
```

- 对于一些需要说明的问题，一般以脚注方式列出。

# 目 录



# 1. 前言

## 1.1. 关于 Excel 和 VBA

Microsoft Excel 不仅仅是一个被广泛应用的电子表格软件，Excel 除了具有一般电子表格软件的数据处理、统计分析、图表功能外，Excel 最大的特点是集成了 VBA 环境。从 Office 97 开始，微软为所有的 Office 组件引入了统一的应用程序自动化语言－－Visual Basic For Application（VBA），并提供了 VBA 的 IDE 环境[1]。作为非常流行的应用程序开发语言 Visual Basic 的子集，VBA 具有 VB 语言的大多数特征和易用性，它最大特点就是将 Excel 作为开发平台来开发应用程序，可以应用 Excel 的所有现有功能，例如其数据处理、图表绘制、数据库连接、内置函数等等。

VBA 作为 Visual Basic 的应用程序的版本，与 Visual Basic 的区别包括如下几个方面：

1. Visual Basic 用于创建 Windows 应用程序，其代码最终被编译为可执行程序；而 VBA 是用于使已有的应用程序自动化，始终为解释执行；
2. Visual Basic 具有自己的开发环境，而 VBA 必须“寄生于”已有的应用程序，例如 Office，或者其他应用程序；
3. Visual Basic 开发出的应用程序编译后可脱离 VB 环境执行，但执行 VBA 应用程序要求用户访问相应的被“寄生的”应用程序，例如 Excel 下开发的 VBA 程序，不仅要安装 Excel，而且安装时必须安装 VBA 环境才可以执行；
4. 使用 VBA 开发，可以使用相应“寄生”应用程序的已有功能，大大简化开发，但同时，对于已有应用程序不擅长的任务，则较难实现。

尽管存在这些不同，Visual Basic 和 VBA 在结构上仍然非常相似。如果你已经了解了 Visual Basic，会发现学习 VBA 非常快；相应的掌握了 VBA 会给 Visual Basic 的学习打下坚实的基础。当学会在 Excel 中用 VBA 创建解决方案后，你就已经具备了在其他 Office 应用程序，例如 Word、Access 等中用 VBA 创建解决方案的基本知识。另外，VBA 不仅仅是应用在微软自己的应用程序中，从 VBA 5.0 起，微软开始为其他软件开发商提供 VBA

[1] Office97 使用的 VBA 版本为 5.0，在此之前并非所有 Office 组件都提供 VBA，而且 VBA 并不提供 IDE，类似于现在的 VBScipt。

的许可证[2]，允许在其他应用程序中集成 VBA，例如 CorelDraw、AutoCAD、ArcGIS 等软件目前都集成了 VBA。

VB, VBA, VBScipt

微软是制造概念和混乱的大师，例如 VB 家族，就有很多成员，不算退役的 QBASIC、WORD BASIC 之类的语言，目前被广泛应用的基于 COM 技术的 VB 成员就有 VB、VBA、VBSCRIPT。

从功能和概念上讲，VB > VBA > VBSCRIPT，后者是前者的子集。但实际上在 VB 中，VBA 起着基础的作用，提供了大多数语言级别的支持，打开 VB 和 VBA 工程的引用，你都可以看到 VBA 的引用。虽然在 VB 开发环境和 VBA 环境下二者具体的 DLL 模块不同，但说明了微软内部是共享 VB 语言的基础实现的。

## 1.2. Excel 作为开发平台

应用 Excel 作为开发工具，在目前主要有 2 方面的用途。

第一是作为一种日常事务和工作处理的脚本语言，主要应用于类似办公自动化等领域。例如办公室人员的重复性事务处理，科研人员的数据处理或模拟，公司或企业的简单的数据处理汇总等等，在这种情况下，这也是过去很多年来 Excel 的主要应用方面，在此方面，可以应用 Excel 实现以下功能：

1. 使重复性的任务自动化；
2. 自定义 Excel 中工具栏、菜单和窗体的界面；
3. 简化模板的使用；
4. 为 Excel 环境添加额外的功能；
5. 对数据执行复杂的操作和分析；
6. 自动绘制各类图表并进行自定义。

[2] 参见 http://www.msdn.microsoft.com/vba/companies/company.asp。

Excel 在科学研究中的应用

由于 EXCEL 友好的用户体验、强大的功能及其普及性，再加上 VBA 的帮助，使得 EXCEL 在科学研究中的应用越来越多，很多研究人员使用 EXCEL 记录实验数据，通过公式、自定义公式、各类 EXCEL 加载宏处理数据，应用 EXCEL 编写数学模型的实现。

在地学研究中，很多著名的软件，如 ISOPLOT，都是使用 EXCEL VBA 编写的；每年的 *COMPUTER AND GEOSCIENCES* 上有大量基于 EXCEL 和 VBA 的程序发表。

第二是作为企业应用的一个组件来使用，主要应用于企业应用程序的前端（表现层）或领域层。在表现层，其实就是应用 Excel 开发用户界面，通过 COM 组件、Web Service、ADO 或其他方式连接后端应用。另一种应用方式是通过其他程序，应用 COM 自动化技术来调用 Excel，完成一些在 Excel 中很容易完成，但在其他程序设计语言或环境下比较困难的任务，例如很多公司使用 Excel 作为报表工具。

随着 Office 2003 的发布，微软对智能客户端技术的推广，Office 和 VBA 在企业应用，智能客户端方面的应用会越来越多，针对 VBA 本身的不足，微软推出的 VSTO（Visual Studio Tools for Office），使得开发人员可以应用.net 开发 Office 应用。

本书并没有专门涉及 Excel 在企业开发中的应用，所以内容和例子也基本上是围绕第一类应用而展开的。但实际上，不管是应用 Excel 开发一般的数据处理程序还是应用 Excel 开发企业应用的前端，其技术和思路都是类似的，如何正确使用 Excel 的对象模型，如何正确设计自己的程序是解决所有实际问题的基础。

## 1.3. 宏、加载宏和 VBA

本书并不打算涉及宏的录制和使用，但进行 VBA 开发确实应该熟悉宏的录制和操作[3]。Excel（包括其他 Office 程序）允许用户录制一段宏，并将其记录为 VBA 代码。对于开发者，使用这一功能，一方面可以节省时间，将录制的宏代码作为开发的基础，另一方面，对于不熟悉的操作，例如如何绘制图表，如何删除一行之类操作，可以录制一个宏并，通过查看其 VBA 代码进行学习。录制的宏可以在 VBA 集成开发环境（IDE）中修改编辑，可以为宏指定按钮、快捷键；而实际编写的代码也可以象宏一样在运行。如何录制宏可以

[3] 参见 Excel 帮助，基本上宏的录制非常简单，在需要录制的操作之前按下录制按钮，操作完成后停止录制即可。录制完成后即可在 VBA IDE 下查看其代码。

参考 Excel 帮助或有关书籍。

加载宏程序是一类程序，它们为 Microsoft Excel 添加可选的命令和功能。例如，Excel 的“分析工具库”加载宏程序提供了一套数据统计分析工具，在进行复杂统计或工程分析时，可用它来节省操作步骤。

Excel 有三种类型的加载宏程序：Excel 加载宏、自定义的组件对象模型（COM）加载宏和自动化加载宏。本书所说的加载宏特指 Excel 加载宏（后缀为 xla 的文件）。Excel 加载宏可以通过单击 *“工具 － 加载宏”* 菜单来调用，在加载宏对话框中，可以安装、卸载加载宏，对于没有在对话框中的加载宏，可以通过浏览按钮定位相应的文件（图 1-1）。

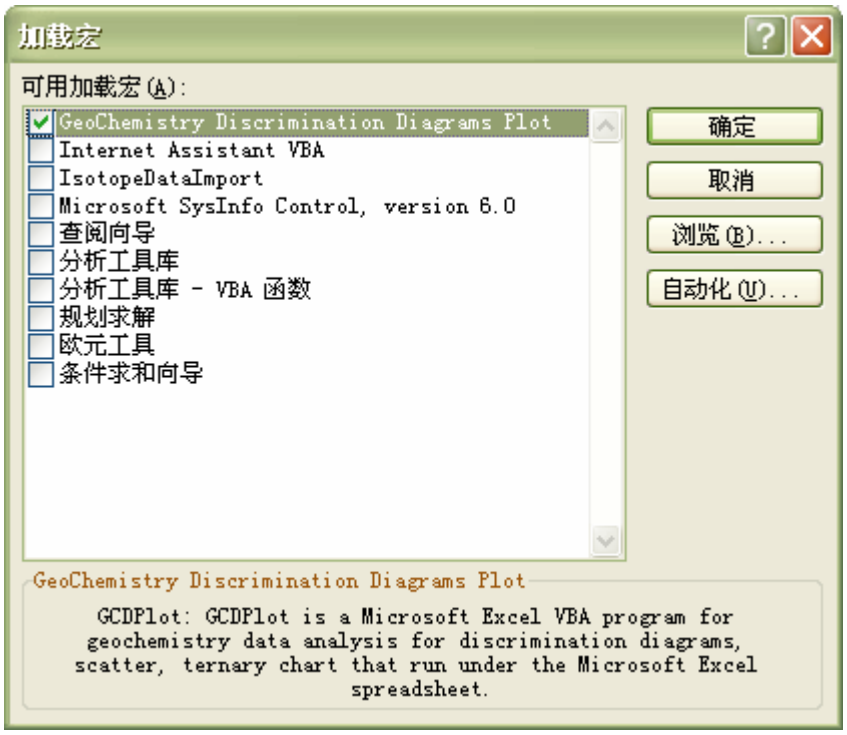



图 1-1 Excel 加载宏对话框

包含 VBA 代码的 Excel 文件，可以通过选择 *“文件 － 另存为”* 对话框保存为加载宏。

VBA 是一种脚本语言，它将 Microsoft Office 中的每一个应用程序都看成一个对象。Office 中，每个应用程序都由各自的 Application 对象代表。例如在 Word 中，Application 对象中包容了 Word 的菜单栏、工具栏、Word 命令以及文档对象等等。文档对象中则包容了所有的文字、表格、图像等文档组成部分的相应对象。在 Excel 中，Application 对象中包容了 Excel 的菜单栏、工具栏、工作薄和工作表对象、图表对象等等。其中，工作表对象和图表对象是 Excel 中的主要对象。VBA 程序设计的主要任务就是通过编写代码操作这些对象来完成一些任务。

VBA 原理的隐喻

VBA 的基本原理可通过下图做示意性解释。


VBA 作为应用 VBA 编写的代码和 OFFICE 对象之间的一个桥梁，为 2 者之间的调用提供支持，这种调用是通过 COM 自动化实现的。例如我们的代码中一句代码，调用 OFFICE 中一个对象的一个属性，那么这个过程大概是类似这样的：VBA 环境解释执行这句代码，如果发现对 OFFICE 对象的调用，就通过 COM 的方式调用这个对象，获取其属性，这样 VBA 代码就可以和 OFFICE 对象进行交互。

## 1.4. 本书的组织

本书划分为以下几个部分：

**VBA 简介**

介绍了 VBA 的 IDE 环境，如何在 IDE 环境下进行开发，VBA 的数据类型，基本语法，模块和过程的概念，调试 VBA 程序，错误处理，如何设计用户窗体，面向对象编程的基本概念及其类模块设计等内容。

**Excel 对象模型**

VBA 对象模型介绍了 Excel 对象模型的架构和组成，Application 对象，Workbook 对象，Worksheet 对象，Range 对象以及与这些对象相关的一些其他对象，分别介绍了各对象的基本属性、方法和事件。

**数据处理**

数据处理部分介绍了数据处理的概念、方式、流程、技术，重点介绍了数据处理的不同方式：“表格驱动”的数据处理方式和基于代码的数据处理方式，介绍了其方式、优缺点和差别；介绍了文件操作、数据操作等具体技术；最后给出了不同的应用实例。

**绘图**

**界面设计**

**其他话题**

本部分作为全书的补充，介绍了在其他部分没有介绍和没有展开介绍的内容，包括Excel VBA 程序的类型及其部署，VBA 程序的保护，在 Excel 中使用 VBA 自动化其他 Office 组件，在 VB 等编程语言中使用 COM 自动化使用 Excel，Excel 数据的导入导出，提高效率的方法和建议等内容。

**附录**

主要介绍了 VBA 的命名规范和代码规范

# 2. VBA 简介

要使用 VBA 进行数据处理，第一要熟悉 VBA 的 IDE 环境，知道如何进行代码书写，如何编写代码，设计窗体，创建类模块（对象），第二要熟悉 VBA 的基本语法和。二者都是 VBA 程序设计的基础，需要认真学习。

VBA 语法不是一章就可以全部介绍完全的，本章介绍的内容是最基本和应该熟练掌握的内容，对于不熟悉或者不理解的内容可以在学习了后面的内容后再反过头来学习。有些内容需要反复练习和熟悉。对于 VBA 语法和用法的很多内容可以随时通过查看帮助来获得相关信息。

本章和下一章（Excel 对象模型）的部分内容，特别是表格内的一些内容，没有必要完全记住，可以作为参考手册来使用。

## 2.1. VBA 及其 IDE 初步

本部分将对 VBA 及其开发环境 IDE（集成开发环境）作一概略的介绍。VBA IDE 是进行程序设计和代码编写的地方，同一版本的 Office 共享同一 IDE。文中会涉及到一些诸如对象、事件等部分读者可能不熟悉或不清楚的概念，对于此类问题可直接忽略之，因为在后面会有详细介绍。本部分也不是一个 VBA 的参考文档，只是其语法、特征的快速浏览和介绍。

### 2.1.1. VBA 集成开发环境（IDE）的组成

VBA 代码和 Excel 文件是保存在一起的，可以通过点击*“工具 － 宏 － Visual Basic 编辑器”*打开 VBA 的 IDE 环境（图 2-1），进行程序设计和代码编写。

打开 IDE 环境的方法

- 通过*“工具 － 宏 － VISUAL BASIC 编辑器”*
- 通过快捷键*“ALT + F11”*
- 右键单击工具栏，选择“Visual Basic”，此工具栏有录制宏，打开 VBA IDE 等的快捷按钮： 

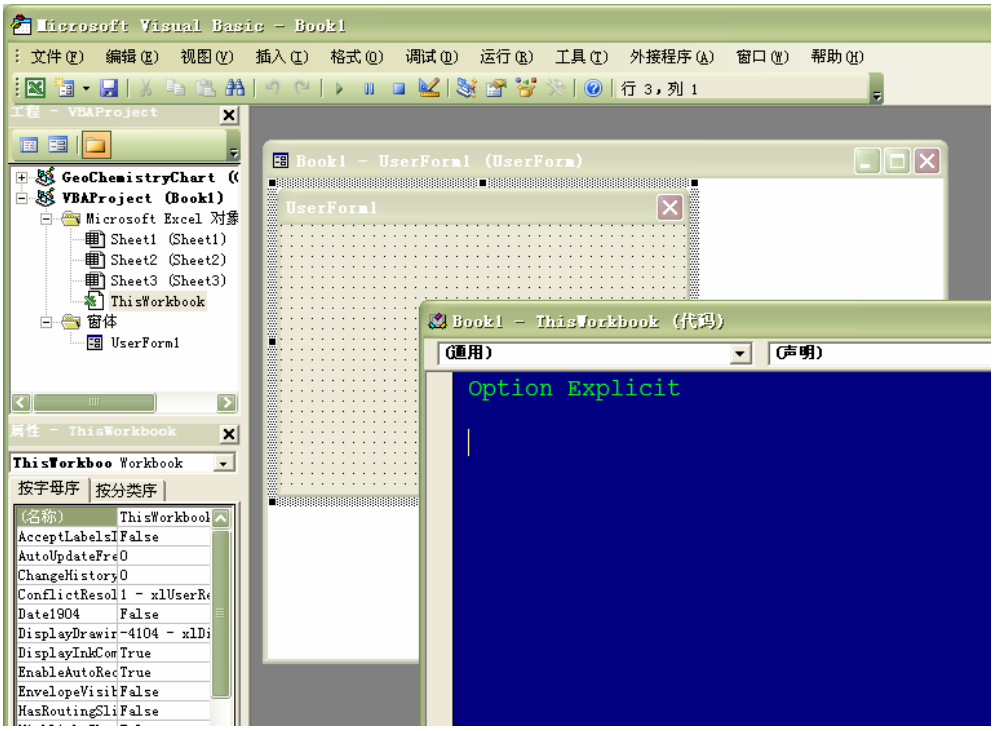



图 2-1 Visual Basic IDE 环境

图 2-1 为 Excel VBA 的 IDE 环境，对于所有使用同一版本 VBA 的应用程序，都共享相同的 IDE 环境。对于同一程序，例如 Excel，不管你打开几个 Excel 文件，但启动的 VBA 的 IDE 环境只有一个。缺省情况下，VBA IDE 环境上方为菜单和工具条（图 2-1），左侧上方窗口为工程资源管理器窗口，资源管理器窗口之下为属性窗口，右侧最大的窗口为代码窗口。

在资源管理器窗口可以看的所有打开和加载的 Excel 文件及其加载宏。每一个 Excel 文件，对应的 VBA 工程都有 4 类对象，包括：Microsoft Excel 对象、窗体、模块和类模块（图 2-2）。Microsoft Excel 对象代表了 Excel 文件及其包括的工作薄等几个对象，包括所有的 Sheet 和一个 Workbook，分别表示文件（工作薄）中所有的工作表（包括图表），例如缺省情况下，Excel 文件包括 3 个 Sheet，在资源管理器窗口就包括 3 个 Sheet，名字分别是各 Sheet 的名字。ThisWorkbook 代表当前 Excel 文件。双击这些对象会打开代码窗口（图 2-1 右侧窗口），在此窗口中可输入相关的代码，响应工作薄或者文件的一些事件，例如文件的打开、关闭，工作薄的激活、内容修改、选择等（有关事件、Excel 对象模型见后）。窗体对象代表了自定义对话框或界面，模块为自定义代码的载体，类模块则是以类或对象的方式编写代码的载体，关于各对象的具体含义和使用见后。在工程资源管理器窗口的右

键菜单下，有添加用户窗体、模块、类模块的选项，也可以将已有的模块移除、导入和导出。在工程资源管理器之下，也可以通过将一个工程中的模块用鼠标拖拽到另一个工程实现模块在工程之间的拷贝。

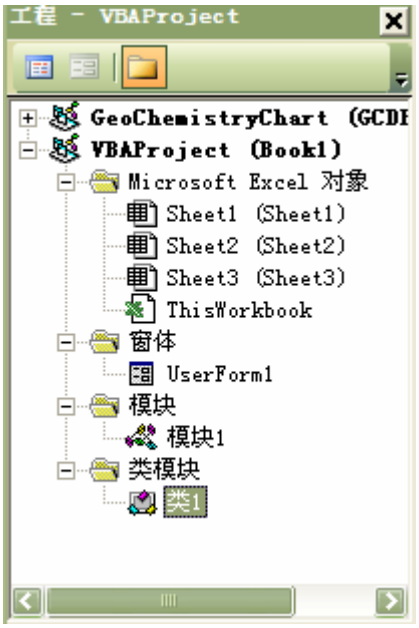



图 2-2 VBA 工程资源管理器窗口

建议随时更改 Excel VBA 工程的名称，其缺省名称为“VBAProject”，可以通过选中工程，在属性窗口更改为有意义的名称，或者在菜单的“工具 － VBAProject 属性”对话框中更改。

在 VBA 工程资源管理器之下是属性窗口（图 2-3），主要用于对象属性的交互式设计和定义，例如选中图 2-2 中的 VBAProject，在属性窗口即可更改其名称。属性窗口除了更改工程、各对象、模块的基本属性外，主要用途是用户窗体（自定义对话框）的交互式设计。图 2-3 显示的就是一个打开的窗体（UserForm）的属性窗口。


图 2-3 VBA 属性窗口

在 IDE 窗口的右侧，可以打开代码窗口。在资源管理器窗口中的每一个对象会对应一个代码窗口（用户窗体包括一个设计窗口和一个代码窗口）。可以通过在对象上双击、在右键菜单或资源管理器工具栏上选择查看代码（或对象）打开代码窗口。

对于 IDE 环境、菜单、工具栏的具体使用和说明，在后面的讲解中会逐步讲解说明。

单击*“视图 － 对象浏览器”*或工具栏上的*“对象浏览器”*按钮即可打开对象浏览器窗口（图 2-4），在此窗口内可查看当前工程及其引用对象的属性、方法和事件。对象浏览器对于熟悉和查看相应的 Excel 对象、引用对象（包括 COM 对象、其他 Excel 程序）所包含的类、属性、方法和事件非常有用，特别是在没有相应的帮助资料或者文档的情况下，对象浏览器是查看一个对象的内容的最有效的工具。

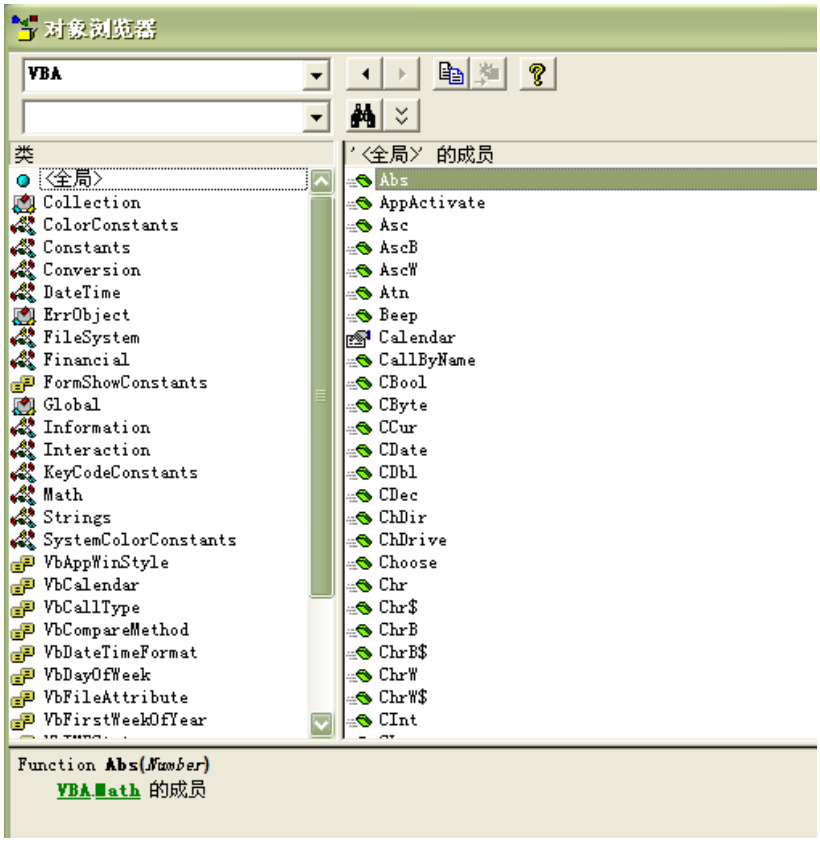



图 2-4 VBA IDE 环境的对象浏览器

## 2.1.2. 在 VBA IDE 下进行开发

熟悉了 VBA 的 IDE 环境后，我们来开发 VBA 之旅的第一个程序。新建一个 Excel 文件，通过菜单或键盘快捷键打开 VBA 集成开发环境，在 VBAProject 上单击右键，选择*“插入 — 模块”*。这样，系统将打开一个代码窗口，在窗口中输入以下代码[4]。

```
#001 Sub MyFirstVBAProgram()
#002     Dim strName As String
#003     Dim strHello As String
#004     strName = InputBox("请输入你的名字：")
#005     strHello = "你好，" & strName & "!"
#006     MsgBox strHello
#007 End Sub
```

将鼠标光标放置在这段代码之内，单击菜单*“运行 — 运行子过程/用户窗体”*，或者在工具栏单击运行按钮，则可运行这段代码。运行结果会显示一个对话框，输入一些内容

[4] 代码内的“#003”为行号，在实际代码中不能输入，在此只为文中叙述方便，之后不再重复。

后，会显示相应的问候语。同样，这段代码可以和宏一样，在 Excel 下选择并执行。

与其他程序设计语言不同，VBA 程序是事件驱动的，没有 Main 函数之类的入口的概念。如果在 IDE 环境下，鼠标光标不在任何过程内，单击工具栏或运行菜单的运行，会显示一个对话框，要你选择要运行的过程。

本质上，VBA 代码应该只是一些完成具体工作的集合，而通过界面元素或者 Excel 的事件驱动执行，你可以通过自定义按钮、菜单，并指定一个宏（VBA 过程，自定义界面也可以通过编程手段完成此类工作），通过单击此按钮即可调用相应的 VBA 代码，或者将调用绑定在 Excel 的某个事件下。

下面我们简单看一下这段代码的组成，代码第 1 行表示这是一个新的过程，名称为“MyFirstVBAProgram”，第 2、3 行定义了 2 个变量，其类型为字符串类型，第 4 行调用 InputBox 这个内置函数，并将返回值赋给 strName 这个变量，第 5 行将几个字符串组合成一个新的字符串，第 6 行调用 MsgBox 这个函数，显示一个对话框，第 7 行表示过程结束。VBA 程序由不同的模块组成，在模块内部，可以定义不同的变量、过程或函数，由此组成一个完整的程序。

代码窗口的设置

中文环境下 VBA IDE 代码窗口缺省的设置比较糟糕，字体为宋体，大小是 9 磅，使用不很方便，可以在“工具 － 选项”对话框下的“编辑器格式”页内设置代码窗口字体、颜色、背景。

在此模块内，再新建一段代码：

```
#001 Function MyAdd(varA, varB) As Variant
#002     MyAdd = varA + varB
#003 End Function
```

此段代码非常简单，只有 3 行，第 1 行表示这是一个函数，具有 2 个参数 varA,和 varB，函数与过程的差别在于函数有返回值，第 2 行将参数 varA,和 varB 的和赋给函数，代表其返回值。函数无法直接运行，必须从工作表或者其他程序调用，例如，我们可以写以下一段简单的程序调用此函数：

```
#001 Sub TestAdd()
#002     Dim a, b, c
#003     a = 12
#004     b = 34
```

```
#005     c = MyAdd(a, b)
#006     MsgBox c
#007 End Sub
```

其中第 5 行为函数 MyAdd 的调用，函数将返回值赋给 c。需要说明的是，VBA 中，调用过程可以使用 Call 语句，也可省略，调用过程时，其参数的括号可以省略，但调用函数必须有括号。

也可以直接在工作表内使用自定义的函数，例如在工作表中，我们可以和 Excel 内置函数一样使用自定义的函数（图 2-5），Excel 会负责参数传递，将返回值赋给相应的单元格，在引用参数改变时会自动重新计算，总之，与内置函数的使用没有什么不同。

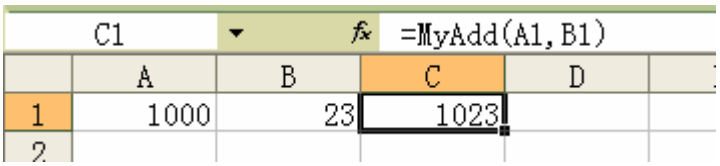



图 2-5 在工作表中使用自定义函数

以上通过 2 个 例子简单介绍了 VBA 编程的过程和概念，后面我们将正式进入 VBA 编程之旅，逐步讲解模块、函数与过程、基本语法、数据类型、类模块与面向对象编程等概念。

## 2.1.3. 善用工具及其他

VBA 集成开发环境，提供了很多便利的工具可以帮助或辅助我们写出好的程序，其中的方便必须亲自使用才可以体会，因此，一定要善用工具[5]。

VBA 的代码编辑器提供了几项非常有用的功能，如代码大小写自动切换，代码自动格式化，即时代码提示。代码自动大小写切换可以帮助我们发现拼写错误，如果我们所有的过程和变量都是按照首字母大写的规则定义的，那么输入这些过程或者变量时，可以全部小写，如果编辑器自动将其首字母改成了大写，那么说明拼写没有错误。代码即时提示可以使我们不必记忆太多的东西，输入对象后会自动列出其属性、方法等内容；输入方法函数后会提示参数信息。代码自动格式化可以使我们在书写代码时不必过于关心格式化的问题，如等号前后加空格之类。

在实际的编程过程中，一定要善于利用在线帮助，VBA 的在线帮助包含了大量对编程

[5] 不仅是 VBA 开发，所有的程序开发工作中都应该善用工具。:）

有用的参考信息，任何人都不可能记得住所有的函数、对象的用法和程序语言的语法，所以一定要利用好帮助。帮助的使用可以在任何关键字上按 F1 键查找相关内容，也可以通过帮助目录浏览，或者通过查找输入关键字查找相关内容。

VBA 的对象浏览器可以浏览一个对象的属性、方法和事件，通过对象浏览器，我们可以获得一个对象的整体概念，特别对于某些没有提供帮助的对象或第三方对象，对象浏览器更有帮助文档的作用。

VBA 中使用了很多常量，可以在编程中直接使用其代表的数字，也可以使用定义好的常量，例如 MsgBox 中的一些参数（按钮参数 vbYesNo 等）。使用常量一者可以获得好的可读性，二者也容易记忆。关于 VBA 以及 Excel 的常量，可以随时通过帮助文档查阅。

在代码书写中，如果一句代码过长，应该使用接行字符（“-”）将其分为几行，而不是书写为一行，一般来说，代码的长度不要超过 80 个字符为宜，这样阅读方便，不需要横向拉动滚动条，也不容易出错。例如以下打开文件语句使用了 3 个接行符：

```
Workbooks.OpenText Filename:=strFilename, _
    Origin:=936, StartRow:=1, DataType:=xlFixedWidth, _
    FieldInfo:=Array(Array(0, 1), Array(37, 1), _
    Array(52,1),Array(64,1)), TrailingMinusNumbers:=True
```

对于代码格式，一定要养成缩进的习惯，在过程之后，循环语句、判断语句之内，如本书的例子样子，缩进 4 个字符，便于阅读。代码中，在一个逻辑或者操作完成之后，应该空一行，以表示其逻辑关系，在过程与过程之间，也应该空一行。

VBA 中，使用单引号“’”表示注释，编写程序时，一定要养成注释的习惯。注释不是所有代码都要注释；一般来说，对一个模块、过程、函数，要大概说明其功能，参数；对于一个过程，如果涉及较复杂的算法，要说明其使用的算法或流程。在过程和函数中，对关键代码，说明其操作的目的、算法或流程。

## 2.2. 模块、函数和过程

### 2.2.1. 模块

模块是自定义的过程、函数保存的地方，也是录制的宏保存的场所。有两种基本类型的模块：类模块和标准模块，本节介绍标准模块，类模块将在专门介绍。模块可以通过右键单击工程资源管理器的工程名，选择“*插入 － 模块*”来新建，新建的模块缺省的名称

为“模块 1”,“模块 2”,建议在属性窗口内更改为有意义的名称。模块有 2 个任务：（1）保存过程和函数；（2）定义模块内的私有变量或整个工程的公有变量。

关于 VBA 中的命名

VBA 中可以为模块、函数、过程、变量赋于中文名称，但笔者不推荐这么做。至于可能出现的在英文系统下的兼容性问题，在 Office 2000 以后，应该已不存在，只是一个编程习惯问题。

另外，变量名称一般应该由 2 部分组成：类型（小写） + 含义（各单词首字母大写）；而过程则由一个或数个表示其意义的单词组成，首字母大写，例如 strName，ChangeName 等。对于过程内部的临时变量，如循环变量，则直接以 i、j、temp 之类命名即可。详细的 VBA 命名规范和代码规范见附录。

在模块中可以声明变量（包括对象），定义过程和函数。过程和函数的定义见下文，变量的声明如下。

变量的声明：

```
Dim i as Long
Dim strName as String
```

Dim 表示声明，i 和 strName 为变量名称，As 之后的 Long 表示数据类型（参见数据类型一节）。变量的声明还可以通过以下两种方式定义：

```
Private i as Long
Public strName as String
```

Private 表示此变量是私有的，只有在此模块内部的函数、过程才可以访问；而 Public 则表示此变量为公用的，可以在其他模块中访问使用。应用 Dim 声明的变量也是私有变量。一般来说，除非必要，一定要少使用公用变量。

要求变量声明

VBA 缺省可以不声明变量，在第一次使用的时候自动声明，但此功能也是 VBA 代码（包括其他 Basic 代码）的一个主要错误之源。试想第一次使用了一个变量 strMyFirstName，之后又通过 strMyFirstNme（少一个 a）来使用它，但由于拼写不同，VBA 以为是一个新的变量，于是会新声明一个变量，这样的错误极其难以发现。

可以通过“工具 － 选项”对话框，在“编辑器”页，选中“要求变量声明”，则在使用变量前，都必须先通过 Dim 语句声明。

## 2.2.2. 过程

过程是最基本的运行单位。一个完整的过程一般类似如下格式：

```
Sub Test()
    … …
End Sub
```

在以上程序中，Sub 代表过程种类，表示运行指定的操作，但不返回运行结果；Test 表示过程名称，最后以 End Sub 结束。

过程的长度（行数）

过程并不限定代码行数，一个过程可以有一行到数百行代码，但随着过程或函数长度的增加，错误也会随之增加，因此一般的编程书籍都建议过程（函数）的长度不要超过计算机屏幕的一屏为宜，对于一些特殊的过程，也不要超过 200 行代码。对于太长的代码可以分拆为几个子过程。

正式的过程描述如下：

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

其组成和含义见表 2-1：

表 2-1 过程的组成部分及其含义

| 部分 | 描述 |
|---|---|
| ***Public*** | 可选的。表示所有模块的所有其它过程都可访问这个 Sub 过程。 如果在包含 Option Private 的模块中使用，则这个过程在该工程外是不可使用的。 |
| ***Private*** | 可选的。表示只有在包含其声明的模块中的其它过程可以访问该 Sub 过程。 |
| ***Static*** | 可选的。表示在调用之间保留 Sub 过程的局部变量的值。Static 属性对在 Sub 外声明的变量不会产生影响，即使过程中也使用了这些变量。 |
| ***name*** | 必需的。Sub 的名称；遵循标准的变量命名约定。 |
| ***arglist*** | 可选的。代表在调用时要传递给 Sub 过程的参数的变量列表。多个变量则用逗号隔开。 |
| ***statements*** | 可选的。Sub 过程中所执行的任何语句组。 |

其中的 arglist 参数的语法以及语法各个部分如下，描述见表 2-2：

```
[Optional] [ByVal | ByRef] [ParamArray] varname[( )] [As type]
    [= defaultvalue]
```

表 2-2 过程参数的语法及其含义

| 部分 | 描述 |
|---|---|
| ***Optional*** | 可选的。表示参数不是必需的关键字。如果使用了该选项，则 *arglist* 中的后续参数都必须是可选的，而且必须都使用 Optional 关键字声明。如果使用了 ParamArray，则任何参数都不能使用 Optional。 |
| ***ByVal*** | 可选的。表示该参数按值传递。 |
| ***ByRef*** | 可选的。表示该参数按地址传递。ByRef 是 Visual Basic 的缺省选项。 |
| ***ParamArray*** | 可选的。只用于 *arglist* 的最后一个参数，指明最后这个参数是一个 Variant 元素的 Optional 数组。使用 ParamArray 关键字可以提供任意数目的参数。ParamArray 关键字不能与 ByVal，ByRef，或 Optional 一起使用。 |
| ***varname*** | 必需的。代表参数的变量的名称；遵循标准的变量命名约定。 |
| ***type*** | 可选的。传递给该过程的参数的数据类型，参加数据类型一节。 |
| ***defaultvalue*** | 可选的。任何常数或常数表达式。只对 Optional 参数合法。如果类型为 Object，则显式的缺省值只能是 Nothing。 |

其中按值传递参数指一种将参数值而不是将地址传递给过程的方式，这就使过程访问到变量的复本。结果，过程不可改变变量的真正值。按地址传递参数指一种将参数地址而不是将值传递给过程的方式，这就使过程访问到实际的变量。结果，过程可改变变量的真正值。VBA 缺省按地址传递参数。

例如以下程序，11 行的过程 ByValTest 的参数 var 是按值传递，因此不会改变参数的值（5 行的调用），而 15 行的过程 ByRefTest 的参数 var 是按引用（地址）传递的，因此会改变参数的值（6 行的调用）。

```
#001 Sub ParameterTest()
#002     Dim var1 As Long, var2 As Long
#003     var1 = 12
#004     var2 = 12
#005     ByValTest var1
#006     ByRefTest var2
#007     MsgBox "还是 12: " & var1
#008     MsgBox "不是 12: " & var2
#009 End Sub
#010
#011 Sub ByValTest(ByVal var As Long)
#012     var = var * 2
#013 End Sub
#014
```

```
#015 Sub ByRefTest(var As Long)
#016     var = var * 2
#017 End Sub
```

不过和变量不同，如果没有使用 Public、Private 显式指定，Sub 过程按缺省情况就是公用的（Public），因此一定要注意把不打算公开的过程定义为私有的。

所有的可执行代码都必须属于某个过程。不能在别的过程中定义 Sub 过程。

Exit Sub 语句使执行立即从一个 Sub 过程中退出。程序接着从调用该 Sub 过程的语句下一条语句执行。在 Sub 过程的任何位置都可以有 Exit Sub 语句。

在 Sub 过程中使用的变量分为两类：一类是在过程内显式定义的，另一类则不是。在过程内显式定义的变量（使用 Dim 方法）都是局部变量（参加数据类型一节）。

过程（包括函数等）的创建可以通过在代码窗口直接键入“Sub…”来创建，也可以使用菜单的“*插入 － 过程*”对话框来创建。

### 2.2.3. 函数

函数是具有返回值的过程，其正式描述如下：

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

各部分含义同 Sub 过程相同，在此不在重复。需要说明的是，函数要返回一个值，其类型通过在定义时通过 As Type 来定义，要从函数返回一个值，只需将该值赋给函数名。在过程的任意位置都可以出现这种赋值。如果没有对 name 赋值，则过程将返回一个缺省值：数值函数返回 0，字符串函数返回一个零长度字符串 (""), Variant 函数则返回 Empty。如果在返回对象引用的 Function 过程中没有将对象引用赋给 name（通过 Set），则函数返回 Nothing。

VBA 中有大量内置函数，例如前边例子里使用过的 MsgBox，InputBox。VBA 的函数主要包括数学函数（包括三角函数、随机数等）、字符串函数等等，熟悉 VBA 的内置函数可以提高工作效率，更好的完成工作。对于 VBA 的函数，可以参考其帮助文档。

## 2.2.4. 调用过程和函数

从其它过程调用一个过程（Sub）时，必须键入过程名称以及任何需要的参数值。Call 语句可有可无，如果使用它，则参数必须以括号括起来。

可以使用 Sub 过程去组织其它的过程。在下面的示例中，Sub 过程 Main 传递参数值 56 去调用 Sub 过程 MultiBeep（第 2 行）。运行 MultiBeep 后，控件返回 Main，然后 Main 调用 Sub 过程 Message（第 3 行）。Message 显示一个信息框；当按“确定”键时，控件会返回 Main，接着 Main 退出执行。

```
#001 Sub Main()
#002     MultiBeep 56
#003     Message
#004 End Sub
#005
#006 Sub MultiBeep(numbeeps)
#007     For counter = 1 To numbeeps
#008         Beep
#009     Next counter
#010 End Sub
#011
#012 Sub Message()
#013     MsgBox "Time to take a break!"
#014 End Sub
```

下面的示例展示了调用具有多个参数的 Sub 过程的两种不同方法。当第二次调用 HouseCalc 时，因为使用 Call 语句（第 3 行），所以需要利用括号将参数括起来。

```
#001 Sub Main()
#002     HouseCalc 99800, 43100
#003     Call HouseCalc(380950, 49500)
#004 End Sub
#005
#006 Sub HouseCalc(price As Single, wage As Single)
#007     If 2.5 * wage <= 0.8 * price Then
#008         MsgBox "You cannot afford this house."
#009     Else
#010         MsgBox "This house is affordable."
#011     End If
#012 End Sub
```

在调用函数（Function）时，为了使用函数的返回值，必须指定函数给变量，并且用括号将参数封闭起来；如下示例所示：

```
Answer3 = MsgBox("Are you happy?", 4, "Question 3")
```

如果不在意函数的返回值，可以用调用 Sub 过程的方式来调用函数。如下面示例所示，可以省略括号，列出参数并且不要将函数指定给变量：

```
MsgBox "Task Completed!", 0, "Task Box"
```

如果在上述例子中包含括号，则语句会导致一个语法错误。

# 2.3. 数据类型与变量

## 2.3.1. 常量和变量

变量用于保存在程序运行过程中需要临时保存的值或对象，变量具有不同的类型，例如整型、浮点型（见后节），变量可能包含不同的数值，在程序运行时，变量的数值可以改变。而当需要存储静态信息时，可以使用常量。使用常量有两个原因，（1）常量可以存放数值供程序运行时多次引用而不改变;（2）使用常量可以增加程序的可读性，例如 BookTitle 比“Excel 与 VBA 程序设计”要容易记忆和修改。

定义变量可以使用 Dim 语句：

```
Dim 变量名 As 数据类型
```

变量的名称变量名必须以字母开始，并且只能包含字母、数字和特定的特殊字符，不能包含空格、句号、惊叹号，也不能包括字符@、&、$和#。名字最大长度为 255 字符。

在 Dim 语句中不必提供数据类型。如果没有提供数据类型，变量将被指定为 Variant 类型，因为 VBA 中默认的数据类型是 Variant。一般应该明确指定数据类型，因为这样程序可读性更强；Variant 类型一般来说，要占用更多空间（16 字节），运行速度也会更慢一些（根据不同数据类型，从基本无差别到大概慢 0.5～1 倍[6]）。对于模块级别的变量，可以使用 Public、Private 来定义（见上节）。

变量定义中要特别注意的一个问题是，VBA 的变量定义，每个变量之后必须加“As 数据类型”，例如：

Dim i As Long, j As Long

而不可以这样：

[6] 本书所涉及的效率和速度快慢的说明虽然也参考了大量资料，但所有数据都经过笔者的测试，测试方法见“其他话题”一章的“提高效率的一些建议”一节，一般都经过 3-5 次测试，取平均结果。

```
Dim i, j As Long
```

这样，只有 j 是 Long 型，i 为 Variant 型。

要声明常量并设定常量的值，需要使用 Const 语句。常量声明后，不能对它赋一个新的数值。例如，假设需要声明一个常量来保存书本价格，可以使用如下语句：

```
Const BOOKPRICE As Long = 23.50
```

可以在 Const 语句中可以指定数据类型。常量的命名惯例是全部字母都用大写，这样就容易区分代码中的变量和常量。

在 VBA 中，赋值表达式使用的是“=”，和比较表达式相同，可以通过表达式给变量赋值，如果表达式左右两侧的数据类型不同，VBA 会尝试进行自动数据类型转换，如果无法转换，会发生一个类型不匹配的运行时错误。例如运行如下程序代码，第一和第二个对话框（5 和 7 行）都可以显示，但第 8 行的赋值会产生一个类型不匹配的运行时错误。

```
#001 Sub TestType()
#002
#003     Dim i As Long
#004     i = 1
#005     MsgBox i
#006     i = 12.12
#007     MsgBox i
#008     i = "123a"
#009     MsgBox i
#010
#011 End Sub
```

## 2.3.2. 数据类型

数据类型，指变量的特性，用来决定可保存何种数据。数据类型包括 Byte、Boolean、Integer、Long、Currency、Decimal、Single、Double、Date、String、Object、Variant（默认）和用户定义类型等。

表 2-3 显示所支持的数据类型，以及存储空间大小与范围。

表 2-3 VBA 的数据类型、存储空间大小、数值范围

| 数据类型 | 存储空间大小 | 范围 |
|---|---|---|
| Byte | 1 个字节 | 0 到 255 |
| Boolean | 2 个字节 | True 或 False |
| Integer | 2 个字节 | -32,768 到 32,767 |

| | | |
|---|---|---|
| Long<br>(长整型) | 4 个字节 | -2,147,483,648 到 2,147,483,647 |
| Single<br>(单精度浮点型) | 4 个字节 | 负数时从 -3.402823E38 到 -1.401298E-45；正数时从 1.401298E-45 到 3.402823E38 |
| Double<br>(双精度浮点型) | 8 个字节 | 负数时从 -1.79769313486231E308 到 -4.94065645841247E-324；正数时从 4.94065645841247E-324 到 1.79769313486232E308 |
| Currency<br>(变比整型) | 8 个字节 | 从 -922,337,203,685,477.5808 到 922,337,203,685,477.5807 |
| Decimal | 14 个字节 | 没有小数点时为 +/-79,228,162,514,264,337,593,543,950,335，而小数点右边有 28 位数时为 +/-7.9228162514264337593543950335；最小的非零值为 +/-0.0000000000000000000000000001 |
| Date | 8 个字节 | 100 年 1 月 1 日 到 9999 年 12 月 31 日 |
| Object | 4 个字节 | 任何 Object 引用 |
| String<br>(变长) | 10 字节加字符串长度 | 0 到大约 20 亿 |
| String<br>(定长) | 字符串长度 | 1 到大约 65,400 |
| Variant<br>(数字) | 16 个字节 | 任何数字值，最大可达 Double 的范围 |
| Variant<br>(字符) | 22 个字节加字符串长度 | 与变长 String 有相同的范围 |
| 用户自定义<br>（利用 Type） | 所有元素所需数目 | 每个元素的范围与它本身的数据类型的范围相同。 |

当使用 Variant 数据类型的时候，VBA 会根据实际需要将数据转换为特定的数据类型，但有时 VBA 的自动转换并不正确，就需要使用类型转换函数。类型转换函数可以如下方式使用：

```
bResult = CBool(expression)
```

其中 expression 参数可以是任何字符串表达式或数值表达式，类型转换函数的类型和使用说明见表 2-4，如果参数超出可以接受的范围会导致一个运行时错误。

表 2-4 VBA 的类型转换函数及其说明

| 函数 | 返回类型 | *expression* 参数范围 |
|---|---|---|
| **CBool** | Boolean | 任何有效的字符串或数值表达式。 |
| **CByte** | Byte | 0 至 255。 |
| **CCur** | Currency | -922,337,203,685,477.5808 至 922,337,203,685,477.5807。 |
| **CDate** | Date | 任何有效的日期表达式。 |
| **CDbl** | Double | 负数从 -1.79769313486231E308 至 -4.94065645841247E-324；正数从 |

| | | |
|---|---|---|
| | | 4.94065645841247E-324 至 1.79769313486232E308。 |
| **CDec** | Decimal | 零变比数值，即无小数位数值，为 +/-79,228,162,514,264,337,593,543,950,335。对于 28 位小数的数值，范围则为+/-7.9228162514264337593543950335；最小的可能非零值是 0.0000000000000000000000000001。 |
| **CInt** | Integer | -32,768 至 32,767，小数部分四舍五入。 |
| **CLng** | Long | -2,147,483,648 至 2,147,483,647，小数部分四舍五入。 |
| **CSng** | Single | 负数为 -3.402823E38 至 -1.401298E-45；正数为 1.401298E-45 至 3.402823E38。 |
| **CStr** | String | 依据 *expression* 参数返回 Cstr。 |
| **CVar** | Variant | 若为数值，则范围与 **Double** 相同；若不为数值，则范围与 **String** 相同。 |

很多资料和书籍会告诉读者不要使用 Variant 类型，因为速度和空间占用。笔者也推荐显式定义数据类型，但原因不是因为速度，而是程序的清晰和可读性。

Variant 代表 Integer 类型时，速度慢 50%，但作为 Double 的 Variant 类型，速度没有什么差别，作为 Currency 类型的 Variant 类型，速度有些时候还要快一些。

## 2.3.3. 运算符

VBA 中的运算符有以下几类运算符：

1. 算术运算符，用来进行数学计算的运算符；
2. 比较运算符，用来进行比较的运算符；
3. 连接运算符，用来合并字符串的运算符；
4. 逻辑运算符，用来执行逻辑运算的运算符。

逻辑运算符和比较运算符将在介绍条件语句时介绍，连接运算符在介绍字符串时做介绍，本小节只介绍算术运算符。

算术运算符有以下运算符：

1. ^ 运算符：求一个数字的某次方，如 A^B；
2. * 运算符：乘法运算；
3. / 运算符：除法运算；
4. \ 运算符：对两个数作除法并返回一个整数；
5. Mod 运算符：求两数的余数；
6. + 运算符：加法运算；

7. - 运算符：减法运算。

当一个表达式牵扯到多个运算符时，就必须考虑运算符的优先顺序。运算符的优先顺序是指在一个表达式中进行若干操作时，每一部分都会按预先确定的顺序进行计算求解，称这个顺序为运算符的优先顺序。

在表达式中，当运算符不止一种时，要先处理算术运算符，接着处理比较运算符，然后再处理逻辑运算符。所有比较运算符的优先顺序都相同；也就是说，要按它们出现的顺序从左到右进行处理。而算术运算符和逻辑运算符则必须按下列优先顺序（由上至下）进行处理（表 2-5）。可以用括号改变优先顺序，强令表达式的某些部分优先运行。括号内的运算总是优先于括号外的运算。但是，在括号之内，运算符的优先顺序不变。

表 2-5 运算符的优先顺序

| 算术 | 比较 | 逻辑 |
|---|---|---|
| 指数运算 (^) | 相等 (=) | **Not** |
| 负数 (–) | 不等 (<>) | **And** |
| 乘法和除法 (*、 /) | 小于 (<) | **Or** |
| 整数除法 (\) | 大于 (>) | **Xor** |
| 求模运算 (**Mod**) | 小于或相等 (<=) | **Eqv** |
| 加法和减法 (+、 –) | 大于或相等 (>=) | **Imp** |
| 字符串连接 (**&**) | **Like Is** | |

## 2.3.4. 数组

数组是具有相同数据类型并且共享同一个名字的一组变量的集合。数组中的元素通过索引数字加以区分。定义数组的语法如下：

```
Dim ArrayName(n) As Type
Dim ArrayName(a to b) As Type
```

其中 n、a、b 是数组中的元素的数目。n 表示数组元素为 0 到 n，共 n+1 个，a 表示数组元素最小索引为 a，最大为 b，元素个数为(a-b+1)个。

例如，如果要创建保存 10 个学生名字的数组，可以使用如下语句：

```
Dim strStudents(9) As String
```

注意，括号中的数字是 9 而不是 10。这是因为在默认情况下，第一个索引数字是 0。数组在处理相似信息时非常有用。假设需要处理 15 门考试成绩，可以创建 15 个独立的变量，也可以创建一个数组来保存考试成绩，具体语句如下：

```
Dim iTestScores(14) As Integer
```

大多数情况下，可以使用像上面例子中类似的一维数组。VBA 也支持多维数组。可以认为二维数组和工作表或者表格具有相似的结构。要创建 4×4 的数组，可以使用如下语句：

```
Dim iTable(3,3) As Integer
```

声明数组时的另一种选择是不给定大小。这样，当程序开始运行时，就具有定义数组大小的灵活性。如果声明数组时没有给定大小，就成为动态数组。声明动态数组的语法如下：

```
Dim DynamicArray() As Type
```

例如，你的应用程序让用户创建一张表格，可以提示用户确定要创建的表格的行和列的数目。通过创建动态数组就可以做到这样，甚至还可以使用户在创建完表格（实际上是数组）后对改变行或列的维数。

对数组进行声明后，可以在运行时用 ReDim 语句重新指定数组的大小：

```
ReDim DynamicArray(size)
```

参数 size 代表数组的新大小。如果要保留数组的数值，请在 ReDim 语句后使用关键字 Preserve，具体语法如下：

```
ReDim Preserve DynamicArray(size)
```

任何数据类型的数组都需要 20 个字节的内存空间，加上每一数组维数占 4 个字节，再加上数据本身所占用的空间。数据所占用的内存空间可以用数据元数目乘上每个元素的大小加以计算。例如，以 4 个 2 字节之 Integer 数据元所组成的一维数组中的数据，占 8 个字节。这 8 个字节加上额外的 24 个字节，使得这个数组所需总内存空间为 32 个字节。包含一数组的 Variant 比单独的一个数组需要多 12 个字节。

## 2.3.5. 自定义数据类型

在模块中可以使用 Type 关键字定义包含一个或多个元素的用户自定义的数据类型。语法如下：

```
[Private | Public] Type varname
    elementname [([subscripts])] As type
    [elementname [([subscripts])] As type]
    … …
End Type
```

其中各关键字和前文说明一致。用户自定义类型可包含一个或多个某种数据类型的数

据元素、数组或一个先前定义的用户自定义类型，一般用于表示数据记录，记录一般由多个不同数据类型的元素组成。例如：

```
Type MyType
    MyName As String          '定义字符串变量存储一个名字。
    MyBirthDate As Date       '定义日期变量存储一个生日。
    MySex As Integer          '定义整型变量存储性别
End Type
```

Type 语句只能在模块级使用。使用 Type 语句声明了一个用户自定义类型后，就可以在该声明范围内的任何位置声明该类型的变量。可以使用 Dim、Private、Public、ReDim 或 Static 来声明用户自定义类型的变量。需要说明的是，在标准模块中，用户自定义类型按缺省设置是公用的，可以使用 Private 关键字来改变其可见性；而在类模块中，用户自定义类型只能是私有的，且使用 Public 关键字也不能改变其可见性。

## 2.3.6. 枚举类型

在 Offfice 和 Excel 对象模型中，有一类常量称为枚举常量，如表示目前操作系统的 XlPlatform 枚举常量，其包含 3 个常量 xlMacintosh、xlMSDOS、xlWindows，在代码中可以如下使用：

```
XlPlatform.xlWindows
```

一方面易于记忆，并可以使用代码窗口的自动完成功能。我们也可以用 Enum 语句自定义枚举变量。变量和参数都可以定义为 Enum 类型。Enum 类型中的元素被初始化为 Enum 语句中指定的常数值。所赋给的值可以包括正数和负数，且在运行时不能改变。例如：

```
Enum SecurityLevel
    IllegalEntry = -1
    SecurityLevel1 = 0
    SecurityLevel2 = 1
End Enum
```

Enum 语句只能在模块级别中出现，可以定义为 Private 类型或者 Public 类型。定义 Enum 类型后，就可以用它来定义变量，参数或返回该类型的过程。不能用模块名来限定 Enum 类型。类模块中的 Public Enum 类型并不是该类的成员；只不过它们也被写入到类型库中。在标准模块中定义的 Enum 类型则不写到类型库中。具有相同名字的 Public Enum 类型不能既在标准模块中定义，又在类模块中定义，因为它们共享相同的命名空间。若不同的类型库中有两个 Enum 类型的名字相同，但成员不同，则对这种类型的变量的引用，将

取决于哪一个类型库具有更高的引用优先级。

例如，可以定义如下的枚举变量：

```
#001 Public Enum MyColors
#002    icMistyRose = &HE1E4FF&
#003    icSlateGray = &H908070&
#004    icDodgerBlue = &HFF901E&
#005    icDeepSkyBlue = &HFFBF00&
#006    icSpringGreen = &H7FFF00&
#007    icForestGreen = &H228B22&
#008    icGoldenrod = &H20A5DA&
#009    icFirebrick = &H2222B2&
#010 End Enum
```

然后，在代码中，我们就可以使用此枚举变量：

```
#001 Sub MyTest()
#002    PrintColor (icForestGreen)
#003 End Sub
#004
#005 Sub PrintColor(color As MyColors)
#006    color = icFirebrick
#007    MsgBox color
#008 End Sub
```

## 2.3.7. 变量的作用域（生存周期）

变量保留其值的这段时间，称为作用域或生存周期。变量的值可能在整个生存周期都在改变，但它仍然保留着一些值。当变量失去了范围之后，它也就不再保存着任一个值。

当过程开始运行时，所有的变量都会被初始化。一个数值变量会初始化成 0，变长字符串被初始化成零长度的字符串 ("")，而定长字符串会被填满 ASCII 字符码 0 所表示的字符或是 Chr(0)。Variant 变量会被初始化成 Empty。用户定义类型中每一个元素变量会被当成个别变量来做初始化。

当声明一个对象变量（详细介绍见后）时，内存中虽有保留空间，但它的值会被设置成 Nothing，直到利用 Set 语句对它指定一个对象引用。

如果在代码的运行期间，变量的值一直没有改变，则它会继续保有它的初始值直到它丢失范围为止。

Dim 语句声明过程的级别变量将保留一个值，直到此过程退出为止。如果该过程调用其它的过程，则在这些过程正在运行的同时，属于调用者过程的变量也保留它的值。

如果过程的级别变量是用 Static 关键字来声明的，则只要代码正在任何模块中运行此变量仍会保留它的值。而当所有的代码都完成运行后，变量会失去它的范围和它的值。所以它的存活期和模块级别的变量是一样的。

模块级别的变量与静态变量是不同的。在标准模块或类模块中变量会保留它的值，直到停止运行代码。在对象类模块中，只要仍有一个属于此对象类的实例存在，则变量会一直保留它的值。模块级别的变量会一直占用内存资源，直到重新设置它们的值，所以只有在必要时才使用它们。

如果在 Sub 或 Function 语句前加上 Static 关键字，则在此过程中所有过程级别的变量的值被保留在调用期间。

### 2.3.8. 字符串

字符串是 VBA 中需要经常处理的一种数据类型，有 2 中字符串类型：变长字符串和定长字符串。变长字符串理论上可以保存大约 2G（$2^{31}$）字节的字符串，实际中其保存的字符串长度由内存大小决定；而定长字符串可以保存大约 65000（$2^{16}$）长度的字符串。

这 2 种字符串都可以使用 Dim 语句来定义，定长字符串只需在其后加一个表示其长度的数字即可。如：

```
Dim MyString as StringDim
Dim MyFixedString as String * 25
```

字符串定义后为空字符串，即没有任何数据的字符串（“”），可以通过以下方式对字符串赋值。字符串在 VBA 中用双引号表示。

```
MyString = "Hello world."
MyFixedString = "This is a fixed string."
MyEmptyString = ""
```

定长字符串必须是其确定的长度，如果赋值时长度过长或过短，则自动以空格添满或截断。

字符串的连接可以使用“&”或者“+”，不过不推荐使用后者，因为容易和数字运算混淆。例如可以使用以下语句连接字符串。

```
MyString = “test”
MyString = MyString & “ Test Added”
MyString = MyString + “ Added Still”
```

字符串处理的常用函数如表 2-6 所示，各函数的具体使用方法可参考 VBA 的帮助文

档。熟悉这些函数会有效的提高你的编程效率。

表 2-6 字符串操作函数索引

| 作用 | 关键字 |
|---|---|
| 比较两个字符串 | StrComp |
| 变换字符串 | StrConv |
| 大小写变换 | Format, LCase, UCase |
| 建立重复字符的字符串 | Space, String |
| 计算字符串长度 | Len |
| 设置字符串格式 | Format |
| 重排字符串 | LSet, RSet |
| 处理字符串 | InStr, Left, LTrim, Mid, Right, RTrim, Trim |
| 设置字符串比较规则 | Option Compare |
| 拆分和连接字符串 | Split, Join (Office2000, VBA 6.0 以后提供) |
| 运用 ASCII 与 ANSI 值 | Asc, Chr |

判断字符串是否为空

VBA 中，判断字符串是否为空可以使用 `STRNAME = ""` 来判断，也可以使用 `LEN(STRNAME) = 0` 来判断，后者要比前者快 0.5 倍到 1 倍，但实际上，后者从可读性上不如前者，所以，如果判断语句是在一个大的循环内，应该尽量使用后者来判断字符串是否为空，否则为了代码可读性也可以使用前者。

字符串处理中，经常需要做的一件事情就是将字符串格式化为特定的格式，例如格式化为标准的长日期格式，保留 2 位小数点的格式等等，VBA 中，可以使用 Format 函数来完成此项工作。Format 函数的语法为：

```
Format(expression[,format])
```

其中 expression 为任何有效的表达式，format 为要格式化的格式，例如可以如下使用 Format 函数：

```
Dim MyTime, MyDate, MyStr
MyTime = #17:04:23#
MyDate = #January 27, 1993#

'以系统设置的长时间格式返回当前系统时间。
MyStr = Format(Time, "Long Time")

'以系统设置的长日期格式返回当前系统日期。
MyStr = Format(Date, "Long Date")
```

```
MyStr = Format(MyTime, "h:m:s")    '返回 "17:4:23"。
MyStr = Format(MyTime, "hh:mm:ss AMPM")   '返回 "05:04:23 PM"。

'如果没有指定格式，则返回字符串。
MyStr = Format(23)    '返回 "23"。

'用户自定义的格式。
MyStr = Format(5459.4, "##,##0.00")    ' 返回 "5,459.40"。
MyStr = Format(334。9, "###0.00")    ' 返回 "334.90"。
MyStr = Format(5, "0.00%")    ' 返回 "500.00%"。
MyStr = Format("HELLO", "<")    ' 返回 "hello"。
MyStr = Format("This is it", ">")    ' 返回 "THIS IS IT"。
```

总之，使用 Format 可以将字符串格式化为任何需要的格式，用好 Format 函数可以在实际开发中省却很多不必要的麻烦和工作，其具体格式化字符串（参数 format）的说明请参考 VBA 的帮助文档。另外，Format 函数比一般的数据类型转换函数（如 CStr）要慢 1 到 10 倍（根据格式的不同），因此，尽量不要频繁使用此函数，更不要滥用此函数。

## 2.3.9. 日期和时间

日期和时间可以存储在 Date 数据类型中，日期的范围是 100-1-1 到 9999-12-31,每天的时间范围是 0:00:00 到 23:59:59。Date 数据类型可以同时保存日期和时间，也可以只在其中保存时间或者日期。相对于字符串等方式保存日期和时间，应用 Date 数据类型，可以在变量之间进行日期运算。

给 Date 数据类型赋值时需要在时间和日期外使用“#”符号包括起来，VBA 基本上可以识别常见的各种时间格式，但为防止引入错误，最好还是使用本机的时间和日期格式。当给一个 Date 类型的变量赋以一个不可识别的时间格式的时间或日期，会引发一个错误。下面是一些使用 Date 数据类型的例子：

```
MyDate = #15 July 1999#
StartDate = #April 8, 2001#
MyTime = #8:47 PM#
StartingDateTime = #05/07/1992 15:56#
```

VBA 的代码编辑器会自动将你输入的日期更改为本机设置的日期格式。如果你输入的年代不是按照 4 位格式输入而是输入 2 位，VBA 将此类时间识别为 1930 年到 2029 年，因此，最好都按 4 位格式输入年代。在处理日期和时间时，VBA 提供了以下函数（表 2-7）。

表 2-7 VBA 中的日期与时间函数

| 作用 | 关键字 |
|---|---|
| 设置当前日期或时间 | Date ,Now, Time |
| 计算日期 | DateAdd, DateDiff, DatePart |
| 返回日期 | DateSerial, DateValue |
| 返回时间 | TimeSerial, TimeValue |
| 设置日期或时间 | Date, Time |
| 计时 | Timer |

## 2.4. VBA 语言基础

本部分将介绍在进行 VBA 程序设计时，需要熟悉的 VBA 语法，包括流程控制语句，最简单的用户交互，一些常用的函数，字符串操作等 VBA 语言的基础内容。

### 2.4.1. 处理简单的用户输入输出

在程序设计时，经常需要进行用户交互，例如显示一条信息反馈给用户，需要用户输入一些信息等。VBA 中，最简单的用户输入输出方法是使用 MsgBox 函数和 InputBox 函数。我们在前面已经使用过这两个函数，下面我们将系统的学习一下这两个函数的使用方法。

MsgBox 函数可以在用户屏幕显示一个对话框，等待用户点击，最后返回一个 Integer 值，代表用户点击的按钮。MsgBox 的语法如下：

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

其中 prompt 代表其显示的消息，之后的参数可以全部省略；buttons 表示对话框显示的按钮和图标，title 表示其标题。例如以下代码片断：

```
#001 Dim iReturn As Integer
#002
#003 iReturn = MsgBox("测试", vbQuestion Or vbYesNo, "标题")
#004
#005 If iReturn = vbYes Then
#006    MsgBox "Yes Clicked", vbInformation
#007 Else
#008    MsgBox "No Clicked", vbInformation
#009 End If
```

第三行显示一个有询问图标，有 Yes 和 No 按钮（是和否）的对话框，单击后根据返回值判断，告诉你点击了那个按钮。

InputBox 可以在用户屏幕显示一个对话框，对话框包括一个提示和用户输入文本框，等待用户输入正文或按下按钮，并返回包含文本框内容的字符串（String 类型）。InputBox 的语法如下：

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

其中各关键字与 MsgBox 重复的在此不再重复。其中 default 表示对话框中缺省文字内容，xpos 和 ypos 表示对话框距屏幕左上角的距离，省略则对话框居中。如果用户单击 OK 或按下 Enter 键，则 InputBox 函数返回文本框中的内容。如果用户单击 Cancel，则此函数返回一个长度为零的字符串 ("")。

如果在 MsgBox 和 InputBox 中同时提供了 *helpfile* 与 *context*，用户可以按 F1 (Windows) 来查看与 context 相应的帮助主题。某些主应用程序，如 Microsoft Excel，会在对话框中自动添加一个 Help 按钮。

## 2.4.2. 控制程序流程

VBA 程序代码的流程同其他程序设计语言一样，有顺序语句：从上到下，由程序的第一行执行到最后一行；判断分支语句：根据条件跳过部分语句，执行另一部分；循环语句：循环执行一段语句。

## 2.4.3. 条件语句

程序中经常需要做的是进行条件判断，根据不同的条件（逻辑判断），执行不同的代码片断，VBA 中的条件判断语句有 If 语句和 Select Case 语句。

### If...Then...Else 语句

If 表达式是根据表达式的值有条件地执行一组语句，如果条件为真，则执行其后的语句，否则到下一个判断条件。其语法为：

```
If condition Then [statements][Else elsestatements]
```

或者，可以使用块形式的语法：

```
If condition Then
    [statements]
[ElseIf condition-n Then
    [elseifstatements] ...
[Else
```

```
    [elsestatements]]
End If
```

If...Then...Else 语句的各部分说明见表 2-8。

表 2-8 If...Then...Else 语句各部分说明

| 部分 | 描述 |
|---|---|
| *condition* | 必要参数。一个或多个具有下面两种类型的表达式：<br>数值表达式或字符串表达式，其运算结果为 True 或 False。如果 *condition* 为 Null，则 *condition* 会视为 False。 |
| *statements* | 在块形式中是可选参数；但是在单行形式中，且没有 Else 子句时，则为必要参数。一条或多条以冒号分开的语句，它们在 *condition* 为 True 时执行。 |
| *condition-n* | 可选参数。与 *condition* 同。 |
| *elseifstatements* | 可选参数。一条或多条语句，它们在相关的 *condition-n* 为 True 时执行。 |
| *elsestatements* | 可选参数。一条或多条语句，它们在前面的 *condition* 或 *condition-n* 都不为 True 时执行。 |

If 语句可以使用单行形式（第一种语法），但是块形式（第二种语法）则提供了更强的结构化与适应性，并且通常也是比较容易阅读、维护及调试，推荐使用。

使用判断语句需要使用比较运算符来构造需要的条件表达式。条件表达式是用来比较 2 个或多个数值并判断其大小，最后返回 True 或者 False（表 2-9）。

表 2-9 比较运算符

| 比较运算符 | 描述 |
|---|---|
| = | 比较 2 个值是否相等 |
| < | 比较左侧的值是否小于右侧的值 |
| > | 比较左侧的值是否大于右侧的值 |
| <= | 比较左侧的值是否小于等于右侧的值 |
| >= | 比较左侧的值是否大于等于右侧的值 |
| <> | 比较左右两侧的值是否不等 |

在使用判断语句时，也会使用到以下逻辑运算符。

1. And 运算符：通过“result = expression1 And expression2”使用，如果两个表达式的值都是 True，则 result 是 True。如果其中一个表达式的值是 False，则 result 是 False。
2. Not 运算符：通过“result = Not expression”使用，如果 expression 是 True，则返回 False，否则返回 True。
3. Or 运算符：使用同 And 运算符，如果两个表达式的值有一个是 True，则 result

是 True。如果两个表达式的值都是 False，则 result 是 False。

下面举一个简单的例子，用来判断输入数据有几位，来说明 If 语句的使用。

```
Dim Number, Digits, MyString
Number = 53      '设置变量初始值。
If Number < 10 Then
   Digits = 1
ElseIf Number < 100 Then
'若判断结果为 True，则完成下一行语句。
   Digits = 2
Else
   Digits = 3
End If
```

在 If 语句中，不要写诸如

If bResult = True Then

这样的语句，而应该这样写成这样：

If bResult Then

同样的，也不要写

If bResult = False Then

而应该写成：

If Not bResult Then

原因包括 2 方面，第一是易于阅读，第二是因为 VBA 不区分赋值和比较运算符，前者容易引入错误。

### Select Case 语句

当一个表达式与几个不同的值相比较时，可以使用 Select Case 语句来交替使用在 If...Then...Else 语句中的 ElseIf。If...Then...Else 语句会计算每个 ElseIf 语句的不同的表达式，Select Case 语句只计算表达式一次。其语法如下：

```
Select Case testexpression
    [Case expressionlist-n
       [statements-n]] ...
    [Case Else
       [elsestatements]]
End Select
```

如果 testexpression 匹配某个 Case expressionlist 表达式，则在 Case 子句之后，直到下一个 Case 子句的 statements 会被执行；如果是最后一个子句，则会执行到 End Select。然后控制权会转移到 End Select 之后的语句。如果 testexpression 匹配一个以上的 Case 子句中的 expressionlist 表达式，则只有第一个匹配后面的语句会被执行。

Case Else 子句用于指明 elsestatements，当 testexpression 和所有的 Case 子句中的 expressionlist 都不匹配时，则会执行这些语句。虽然不是必要的，但是在 Select Case 区块中，最好还是加上 Case Else 语句来处理不可预见的 testexpression 值。如果没有 Case expressionlist 匹配 testexpression，而且也没有 Case Else 语句，则程序会从 End Select 之后的语句继续执行。

可以在每个 Case 子句中使用多重表达式或使用范围，例如，下面的语句是正确的：

```
Case 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

在下面的示例中，Select Case 语句会计算发送给此过程的参数 performance。请注意，每个 Case 语句可以包含一个以上的值，一个值的范围，或是一个值的组合以及比较运算符。如果 Select Case 语句与 Case 语句的任何值相匹配，则可选的 Case Else 语句运行。

```
Function Bonus(performance, salary)
   Select Case performance
      Case 1
         Bonus = salary * 0.1
      Case 2, 3
         Bonus = salary * 0.09
      Case 4 To 6
         Bonus = salary * 0.07
      Case Is > 8
         Bonus = 100
      Case Else
         Bonus = 0
   End Select
End Function
```

## 2.4.4. 循环语句

如果需要反复执行某一句或者某一段语句，则可以使用 VBA 的循环语句。循环允许重复执行一组语句。某些循环重复执行语句直到条件为 False（Do...Loop While）；某些循环重复执行语句直到条件为 True（Do...Loop While）；某些循环执行一指定次数的语句（For…Next）或是集合中的每一个对象（For Each...Next）。以下将逐次介绍这些循环语句。

### Do...Loop 语句

当条件为 True 时，或直到条件变为 True 时，重复执行一个语句块中的命令。其语法如下：

```
Do [{While | Until} condition]
    [statements]
    [Exit Do]
    [statements]
Loop
```

或者可以使用下面这种语法：

```
Do
    [statements]
    [Exit Do]
    [statements]
Loop [{While | Until} condition]
```

Do Loop 语句的语法具有以下几个部分（表 2-10）：

表 2-10 Do Loop 语句各部分说明

| 部分 | 描述 |
|---|---|
| *condition* | 可选参数。数值表达式或字符串表达式，其值为 **True** 或 **False**。如果 *condition* 是 Null，则 *condition* 会被当作 **False**。 |
| *statements* | 一条或多条命令，它们将被重复当或直到 *condition* 为 **True**。 |

在 Do...Loop 中可以在任何位置放置任意个数的 Exit Do 语句，随时跳出 Do...Loop 循环。Exit Do 通常用于条件判断之后，例如 If...Then，在这种情况下，Exit Do 语句将控制权转移到紧接在 Loop 命令之后的语句。如果 Exit Do 使用在嵌套的 Do...Loop 语句中，则 Exit Do 会将控制权转移到 Exit Do 所在位置的外层循环。

### For...Next 语句

For…Next 语句可以以指定次数来重复执行一组语句。其语法如下：

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

For…next 语句的语法具有以下几个部分（表 2-11）：

表 2-11 For Next 语句各部分说明

| 部分 | 描述 |
|---|---|
| *counter* | 必要参数。用做循环计数器的数值变量。这个变量不能是 Boolean 或数组元素。 |
| *start* | 必要参数。*counter* 的初值。 |
| *End* | 必要参数，*counter* 的终值。 |
| *Step* | 可选参数。*counter* 的步长。如果没有指定，则 *step* 的缺省值为 1。 |
| *Statements* | 可选参数。放在 **For** 和 **Next** 之间的一条或多条语句，它们将被执行指定的次数。 |

step 参数可以是正数或负数。step 参数值决定循环的执行情况，如果 step 为正数，则当循环变量 counter 小于等于结束变量 End 时循环执行；如果 step 为负数，则当 counter 大于等于结束变量 End 时循环执行

循环中可以在任何位置放置任意个 Exit For 语句，随时退出循环。Exit For 经常在条件判断之后使用，例如 If...Then，并将控制权转移到紧接在 Next 之后的语句。可以将一个 For...Next 循环放置在另一个 For...Next 循环中，组成嵌套循环。不过在每个循环中的 counter 要使用不同的变量名。下面的代码是正确的。

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...
        Next K
    Next J
Next I
```

为提高程序的执行效率，最好使用 Long 型数据类型的变量作为循环变量，比起其他数据类型的循环变量，如 Variant 类型，使用长整型循环变量可以提高 0.5 倍以上的运行效率。在不影响程序意义的情况下，可以省略 Next 语句之后的 counter。

### For Each...Next 语句

另外一个经常使用的循环是 For Each…Next 循环，此循环主要针对一个数组或集合中的每个元素，重复执行一组语句。其语法如下：

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

For Each...Next 语句的语法具有以下几个部分（表 2-12）。

表 2-12 For Each...Next 语句的组成部分

| 部分 | 描述 |
|---|---|
| *element* | 必要参数。用来遍历集合或数组中所有元素的变量。对于集合来说，*element* 可能是一个 Variant 变量、一个通用对象变量或任何特殊对象变量。对于数组而言，*element* 只能是一个 **Variant** 变量。 |
| *group* | 必要参数。对象集合或数组的名称（用户定义类型的数组除外）。 |
| *statements* | 可选参数，针对 *group* 中的每一项执行的一条或多条语句。 |

如果集合中至少有一个元素，就会进入 For...Each 块执行。一旦进入循环，便先针对 *group* 中第一个元素执行循环中的所有语句。如果 *group* 中还有其它的元素，则会针对它们执行循环中的语句，当 *group* 中的所有元素都执行完了，便会退出循环，然后从 Next 语句之后的语句继续执行。在循环中可以在任何位置放置任意个 Exit For 语句，随时退出循环。可以将一个 For...Each...Next 循环放在另一个之中来组成嵌套式 For...Each...Next 循环。但是每个循环的 *element* 必须是唯一的。

以下示例使用 For Each...Next 语句搜寻集合中的所有成员的 Text 属性，查找“Hello”字符串。

```
Dim Found, MyObject, MyCollection
Found = False                           '设置变量初始值。
For Each MyObject In MyCollection       '对每个成员作一次迭代。
   If MyObject.Text = "Hello" Then
      Found = True      ' 将变量 Found 的值设成 True。
      Exit For          ' 退出循环。
   End If
Next
```

在对集合进行循环时，使用 For Each 循环要比 For 循环快 1/3 以上，因此，尽量对集合对象使用 For Each 循环，因为一方面，For Each 循环不需要设置循环变量，不容易出错，而且循环速度又比 For 循环快。对于数组，For Each 循环的速度基本没有优势，大概可以快 10%左右，因此，可以根据实际情况选择适当的循环。

除了以上介绍的循环语句，VBA 还有一个 While...Wend 语句，此语句只是为了向前兼容保留的，不推荐使用，其效果与 Do…Loop 语句一致，但后者提供了更好的控制结构。

## 2.4.5. With 语句

With 语句可以在一个单一对象或一个用户定义类型上执行一系列的语句，一方面可以

节约代码输入量，一方面 With 语句也可以提高运行效率。With 语句的语法如下：

```
With object
    [statements]
    [statements]
End With
```

Object 代表一个对象或用户自定义类型的名称，statements 表示要执行在 object 上的一条或多条语句。

With 语句可以对某个对象执行一系列的语句，而不用重复指出对象的名称。例如，要改变一个对象的多个属性，可以在 With 控制结构中加上属性的赋值语句，这时候只是引用对象一次而不是在每个属性赋值时都要引用它。下面的例子显示了如何使用 With 语句来给同一个对象的几个属性赋值。

```
With MyLabel
    .Height = 2000
    .Width = 2000
    .Caption = "This is MyLabel"
End With
```

程序一旦进入 With 块，object 就不能改变。因此不能用一个 With 语句来设置多个不同的对象。可以将一个 With 块放在另一个之中，而产生嵌套的 With 语句。但是，由于外层 With 块成员会在内层的 With 块中被屏蔽住，所以必须在内层的 With 块中，使用完整的对象引用来指出在外层的 With 块中的对象成员。

With 语句经常使用在对一个对象或用户自定义类型需要进行反复引用的情况下，特别是在循环中，如果要反复引用某个对象，那么最好通过 With 语句来引用该对象。

## 2.4.6. Exit 语句

我们前边已经使用过多次 Exit 语句，现在做一总结，Exit 语句可以退出 Do...Loop、For...Next、Function、Sub 或 Property 代码块（Property 介绍见类模块一节）。其语法为：

```
Exit Do
Exit For
Exit Function
Exit Property
Exit Sub
```

各用法的说明见表 2-13：

表 2-13 Exit 语句

| 语句 | 描述 |
|---|---|
| **Exit Do** | 提供一种退出 **Do...Loop** 循环的方法，并且只能在 **Do...Loop** 循环中使用。**Exit Do** 会将控制权转移到 **Loop** 语句之后的语句。当 **Exit Do** 用在嵌套的 **Do...Loop** 循环中时，**Exit Do** 会将控制权转移到 **Exit Do** 所在位置的外层循环。 |
| **Exit For** | 提供一种退出 **For** 循环的方法，并且只能在 **For...Next** 或 **For Each...Next** 循环中使用。**Exit For** 会将控制权转移到 **Next** 之后的语句。当 **Exit For** 用在嵌套的 **For** 循环中时，**Exit For** 将控制权转移到 **Exit For** 所在位置的外层循环。 |
| **Exit Function** | 立即从包含该语句的 **Function** 过程中退出。程序会从调用 **Function** 的语句之后的语句继续执行。 |
| **Exit Property** | 立即从包含该语句的 **Property** 过程中退出。程序会从调用 **Property** 过程的语句之后的语句继续执行。 |
| **Exit Sub** | 立即从包含该语句的 **Sub** 过程中退出。程序会从调用 **Sub** 过程的语句之后的语句继续执行。 |

# 2.5. 用户窗体

VBA 包括一类特殊的对象——用户窗体（UserForm），用户窗体是一个窗口或对话框，用以构成应用程序的用户界面部分。使用用户窗体可以提供一个图形用户界面，在此界面上，可以为其添加按钮、图片、文本框等控件，作为一个自定义窗口或者对话框，供用户与程序进行交互。

## 2.5.1. 设计用户窗体

通过在工程资源管理器右键单击，选择“*插入 － 用户窗体*”可以新建一个用户窗体（图 2-6），新建的用户窗体缺省名称为“UserForm1”，新建以后的用户窗体是一个空白区域，可以从工具箱中选择控件为用户窗体添加界面元素。选中用户窗体，在属性窗口内可以设置其属性（如果工具箱和属性窗口没有显示，可以通过视图菜单激活之），也可以拖动改变其大小。窗体具有设计模式和代码模式之分，在资源管理器窗口选择窗体，可以在其工具栏或者通过右键切换窗体的设计模式和代码模式。设计模式下可以通过鼠标拖拽以及属性调整来可视化设计用户窗体的外观以及样式。在代码模式下，可以和在一般的模块窗体一样，定义变量、过程，并且可以响应窗体和控件的事件。

新建一个窗体，在属性窗口，将此窗体的“名称（Name）”设置为“frmMyHelloWorld”，

Caption 设置为“Hello, My First Form”。

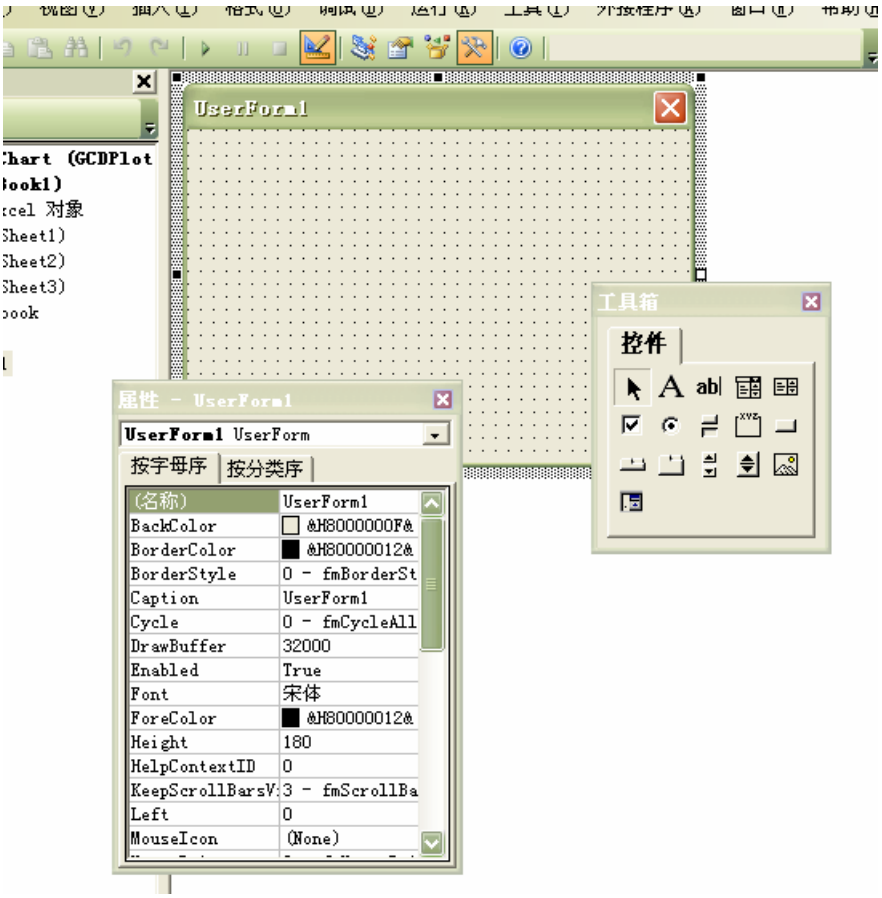



图 2-6 用户窗体

从工具箱中选择不同的控件，然后在用户窗体上按下鼠标左键，拖动即可将控件添加到用户窗体。从工具箱选择“文本框”，添加到用户窗体，设置其名称为“txtHelloMsg”；选择“命令按钮”，添加到用户窗体,设置其名称为“cmdHello”，Caption 为“Hello”。双击按钮，VBA IDE 自动打开代码窗口，并新建一个名为“cmdHello _Click”的过程，这个过程称为事件，当用户点击按钮，此过程内的代码就会执行。在此过程中输入如下代码：

```
Private Sub cmdHello_Click()
    Me.txtHelloMsg.Text = "Hello, My VBA!"
End Sub
```

切换到用户窗体的设计模式，在工具栏上点击“运行”可以测试此用户窗体，刚刚新建的用户窗体将显示在桌面，单击按钮，文本框的文字将会变成“Hello, My VBA!”（图 2-7）。

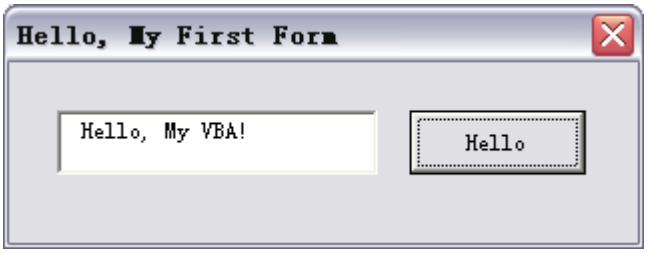



图 2-7 运行的 My First Form

这就是用户窗体编程的基本概念，我们可以操作的窗体，文本框，按钮等都是对象，准确来说，都是 ActiveX 控件，通过对这些对象的可视化设计，设置其属性，然后在其事件内写入需要的操作。

总而言之，用户窗体是 VBA 中的一个对象，我们可以通过设置其属性、调用其方法、响应其事件来操作用户窗体和其之上的控件，当用户对用户窗体的元素进行操作时（通过鼠标、按钮），响应的事件就会被执行，这种编程模式就叫做事件驱动的编程模式。我们可以创建一个过程来显示设计好的用户窗体。

显示一个用户窗体需要经过 2 个步骤。首先这些用户窗体必须被装载（Loaded），接着显示（Show）这个窗体。装载窗体就是为窗体分配内存，初始化这个窗体。显示窗体则是创建一个图形窗口，显示给用户。装载窗体可以通过调用窗体的 Load 方法，显示窗体可以调用窗体的 Show 方法。如果调用 Show 方法时窗体还没有装载，则会首先自动装载窗体。相应的，可以通过 Hide 隐藏窗体，通过 Unload 来卸载（销毁）窗体。如果需要快速显示窗体，在 Show 之前应该先 Load 之，对重复使用的窗体，不要 Unload 而是 Hide 之，可以快速显示窗体，提高效率。

窗体的显示模式有 2 种：模式窗体（modal）和无模式窗体（modeless）。对于 modal 窗体，显示后将停止显示之后的代码直到退出或隐藏此窗体，并且必须退出或隐藏此窗体后，才可以操作非此窗体的其他界面元素（包括宿主程序 Excel 和其他用户窗体），例如 MsgBox 就是模式窗体；而对于无模式窗体，在其显示之后，不阻塞后续代码和界面元素，后续代码继续执行，其他界面元素也可以操作。可以使用以下语句显示一个无模式窗体：

```
UserForm1.Show vbModeless
```

模式窗体显示代码为：

```
UserForm1.Show vbModal
```

对于 Show 方法，如果不指定模式，则缺省为模式窗体。

对于一个窗体，与其创建、显示、销毁相关的有 3 个事件，Activate 事件发生在窗体

称为当前激活窗体之时，Deactivate 事件则发生在窗体不再是当前激活窗体之时，Terminate 发生在窗体销毁（Unload）之时。

## 2.5.2. 事件驱动

下面介绍一下事件的有关概念。事件简单来说，就是由用户或者系统触发的、可以在代码中响应的一段代码。例如，我们移动鼠标、点击窗体和按钮、敲击键盘、窗体的显示移动等等都会产生一系列的事件，通过编写代码中响应这些事件，当发生此类事件时，例如单击了一个按钮，程序代码就会进行相应的操作。例如在前边的例子中，在按钮单击事件中（cmdHello_Click()），我们书写了改变文本框文本的代码。

窗体、窗体上的控件都定义了很多事件，例如鼠标移动、单击等事件。我们自定义的类模块也可以有自定义的事件（参见本章类模块一节）。

用户窗体编程时，编写一个事件的响应代码有 2 种方式，对于缺省事件，双击这个控件，就会自动打开代码编辑器，新建或定位到这个事件。或者，我们可以在代码编辑器上方左侧的“对象框”选择对象，然后在其右侧的“过程/事件框”选择响应的事件，即可定位或创建这个事件（图 2-8）。

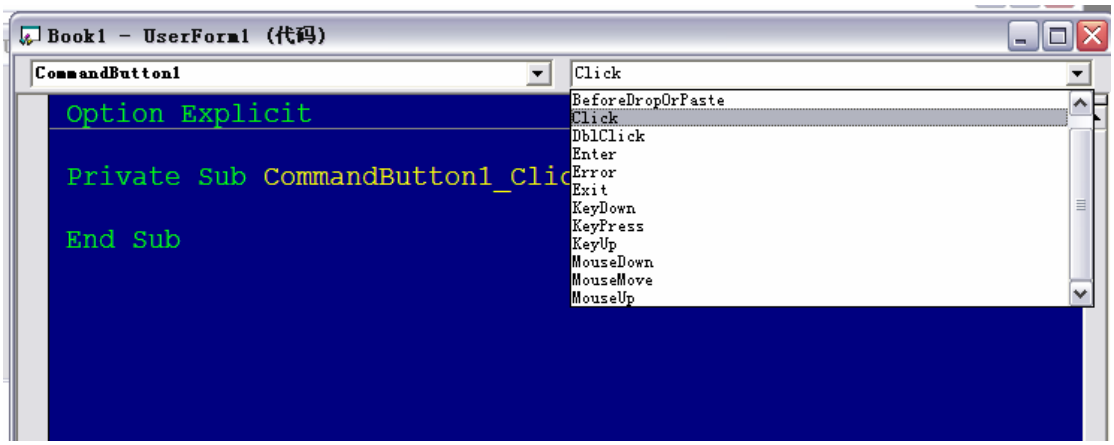



图 2-8 代码编辑器中选择事件

## 2.5.3. 使用控件

用户窗体编程的大部分工作就是在工具箱选择合适的工具，放置到用户窗体，设置其属性。工具箱中包含了一系列可以放置到窗体的控件，例如标签（Label）控件，可以显示静态文本，文本框（TextBox）可以显示动态的可编辑文本。可以直接从工具箱将控件拖拽

到窗体上的合适位置，通过拖拽设置大小、位置，在属性窗口设置其属性。

用户窗体以及工具箱的常用控件包含有一些通用的属性、方法和事件（不一定所有控件都有），这些属性、方法和事件描述如表 2-14。

表 2-14 用户窗体和控件的常用属性、方法、事件描述

| 属性 方法 事件 | 描述 |
|---|---|
| BackColor | 属性：控制控件的背景色。 |
| Caption | 属性：在控件上显示的文本，不能被用户改变。 |
| Change | 事件：当控件的 Value 属性改变时调用。 |
| Click | 事件：用户在控件上单击鼠标时调用。 |
| ControlTipText | 属性：当用户鼠标指针停留在控件上时显示的提示信息。 |
| DblClick | 事件：用户在控件上双击鼠标时调用。 |
| Enabled | 属性：当为 True 时，会获取焦点并响应用户操作。 |
| Enter | 事件：在控件获取了焦点后触发。 |
| Exit | 事件：当控件的焦点转移到其他控件时发生。 |
| Font | 属性：表示控件的字体样式。 |
| ForeColor | 属性：表示控件的前景色，如控件上字体的颜色 |
| Height | 属性：表示控件的高度。 |
| Left | 属性：表示控件左侧距窗体左侧的距离。 |
| Locked | 属性：当为 True 时，用户无法改变其内容。 |
| Name （名称） | 属性: 表示控件的名称。 |
| SpecialEffect | 属性：控件的显示效果，例如平面效果之类。 |
| TabIndex | 属性：设置控件在窗体中的 Tab 顺序。 |
| TabStop | 属性：当用户使用 Tab 键设置焦点到此控件后，控件是否接受焦点。 |
| Top | 属性：控件距窗体上部的距离。 |
| Value | 属性：一般包含控件的状态或者内容。 |
| Visible | 属性：为 True 则显示在窗体上，否则不显示。 |
| Width | 属性：控件宽度。 |

表 2-14 中 Top，Left，Width 和 Height 的单位都是以磅（Point）为单位。Value 属性的具体含义与控件的类型有关，当 Value 改变时，都会触发 Change 事件。Click 和 DblClick 事件发生在用户单击或双击鼠标时，一般来说，客户代码往往都包含在 Click 事件中，例如下拉列表框的选择，按钮的点击等。

对 VBA 用户窗体中的控件的介绍和应用如果有疑问，可以将控件添加到用户窗体，选中控件，单击 F1，将显示控件的帮助文档以及示例。

Excel 编程中要经常使用的一个特殊控件是 RefEdit 控件，在用户窗体上，本控件可以使用户方便的在一张或多张工作表中输入或选定的单元格区域的地址。若要选定某个区域，

请单击控件中的按钮以折叠用户窗体，然后选定该区域，再单击控件中的按钮以展开用户窗体。不能在无模式用户窗体中使用 RefEdit 控件，否则会导致程序死锁或非法退出。

# 2.6. 调试 VBA 代码

## 2.6.1. 错误的类型

程序设计过程中，经常需要定位错误位置并改之，此过程常称为调试。

VBA 中第一类错误为语法错误，如果你打开了代码编辑器的“自动语法检查”，则当你输入了一条错误的语法时，VBA IDE 会即时给出一条提示信息（图 2-9）。

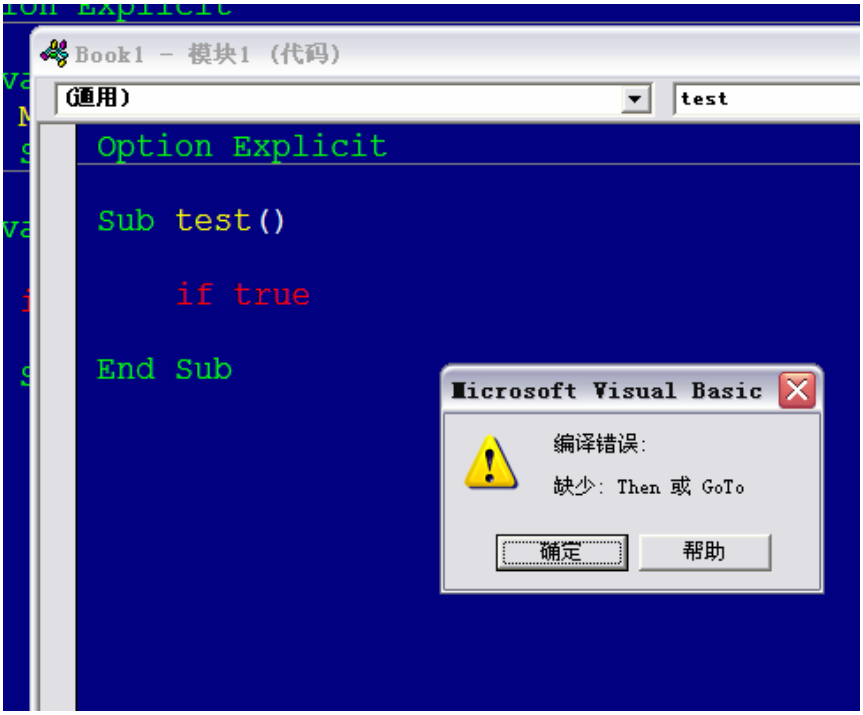



图 2-9 语法错误即时提示信息

第二类错误为运行错误。运行错误是指造成应用程序停止运行的任何错误。有时这是由于错误的拼写，例如对象名字。VBA 不会检查这种类型的错误，除非运行该过程。运行错误也可能是用户的操作所引起的，而用户的操作不是你所能控制和预测的。例如，如果用户不给你编写的函数提供数据，就有可能产生运行错误。对于此类情况，就需要写错误处理代码。

最后一种可能出现的错误是逻辑错误。逻辑错误不会显示在 IDE，问题在于代码的执

行结果和预期的结果不同，这就意味着代码的逻辑或算法出了问题。随着对语言和环境的熟悉，大部分的调试时间可能都用在对逻辑错误的处理上。

## 2.6.2. 使用 Debug 对象

Debug 对象可以在运行时将输出发送到立即（Immediate）窗口。Debug 对象有 2 个方法：Assert 和 Print 方法，Assert 判断 1 条条件语句，如果为 False，则挂起执行，暂停在 Assert 语句的地方，Print 语句则向立即窗口输出一段文本，例如以下代码：

```
#001    Dim i As Long
#002    For i = 1 To 10 Step 1
#003        Debug.Print i
#004        Debug.Assert i < 8
#005    Next i
```

当程序执行到 3 行 Debug.Print 时，程序向立即窗口输出 i 的值，而当 i 大于等于 8 时，Debug.Assert 判断的表达式为 False，程序挂起。

使用 Debug 对象可以对程序进行跟踪判断和调试。

## 2.6.3. VBA 的调试工具

对以上错误查找的过程就称为调试。VBA 提供了以下几种调试工具供使用。这些工具包括：

- 立即窗口：可以交互式运行 VBA 代码；
- 监视窗口：监视变量、对象的值或内容；
- 断点：程序执行到此后会进入中断状态，可以单步执行，查看变量对象等；
- 单步执行代码

当对过程进行调试时，需要对过程中的每一行代码进行处理。为了便于将错误的区域分离出来，VBA 提供了多种方式，以便在某个特定的地方暂停过程的执行，称为将过程设置为中断模式。中断模式暂时挂起过程的执行。通过将某个过程设置为中断模式，可以检查当前变量和属性的数值，也可以运用其他调试技术。当过程处于中断模式时，可以执行如下操作：

- 编辑应用程序的代码；
- 查看变量、属性的数值和语句；

- 为变量和属性输入不同的数值；
- 通过使用立即窗口运行 VBA 语句。

如果知道错误出现在某个语句之后，则可以在此设置断点。在特定的位置暂停过程的方式称为设置断点。将某行代码设置为断点有以下几种方法：

- 选择要设置为断点的语句，再选择从菜单中选择“调试”、“断点开关”；
- 选择要设置为断点的语句，再按下 F9 键；
- 选择要设置为断点的语句，再单击“代码”窗口边缘的指示器栏。

只能将断点设置在可执行的代码行，而不能将断点设置在不可执行的代码行，其中包括注释、变量和常数的声明语句以及空行。如果某行代码已经设置为断点，该行代码的颜色会改变，该行边缘的指示器栏还显示一个圆点。以上三种操作也可以用来删除断点，因为断点是一个开关项。如果在过程中设置了多个断点并希望将它们全部删除，可以选择“*调试 － 清除全部断点*”。

程序中断之后，鼠标指向变量，会在变量上显示数量的当前数值。也可以在监视窗口中设置变量，随时监视其变化。

对于逻辑错误，一个比较有效的调试方法就是定位到错误可能在的地方，然后通过单步执行查找其错误（*调试 － 逐语句/过程*）。

综合利用这些工具，就可以对 VBA 代码进行调试了。

## 2.7. 错误处理

所谓调试，就是对可以预测的问题进行处理并进行纠正的过程。调试只能够发现可以预测的错误，要处理不可预测的和不可避免的错误时，就必须使用错误处理。通过启用错误处理，就可以使应用程序更稳定、更健壮。所谓错误处理程序，就是应用程序中用来捕获和处理错误的实用程序。错误处理程序的开发过程可以分为下面三个步骤：

1. 设置错误捕获：当设置错误捕获时，就是告诉程序当错误发生时，到什么地方去捕获错误;
2. 编写错误处理实用程序：错误处理实用程序就是当错误发生时程序要跳转到的地方；
3. 提供从错误处理程序跳出的出口，换句话说，就是当错误处理完毕时，需要程序做的事情。

### 2.7.1. 设置错误捕获

设置错误捕获在 VBA 中是通 On Erro 语句来实现的。在一个给定的程序中，过程可以拥有一个或数个 On Error 语句，当过程中有多个 On Error 语句的话，只有最近正在执行的那个捕获陷阱才是起作用的。

处理错误有两种不同的方式，其中之一是执行内联错误处理。内联错误处理在 On Error 语句中有一些指令来处理错误。要执行内联错误处理的话，可以使用下面语句中的任何一句：

- On Error Resume　如果有运行时刻的错误发生，那么程序将从导致错误发生的语句处重新开始执行；
- On Error Resume Next　如果有运行时刻的错误发生，那么程序就从导致错误发生的语句的下一句继续执行下去。

要禁止错误的处理程序，可以在程序中初始化 On Error 语句以后，使用 On Error GoTo 0 语句。在测试程序或过程并且不想启用错误处理时，禁止错误处理程序是非常有用的。

错误处理的两种不同处理方式中，不推荐使用内联错误处理方法，最好是采用错误捕获的处理方式，采用错误捕获能够跳转到错误处理实用程序，这样的方式使开发人员能够对各种各样的错误进行灵活的处理。为了设置能够跳转到错误处理实用程序的错误捕获，可以使用 On Error GoTo“行”语句，这里“行”代表的是位于错误处理代码前面的行标号。要创建行标号的话，只要为该行输入一个名称，后面跟一个冒号就可以了。VBA 中的行标号需要独占一行。

### 2.7.2. 编写错误处理实用程序

当错误发生时，VBA 就查找程序中的行标号，并开始跳转到行标号所在的位置继续执行。错误处理实用程序的代码评估所发生的错误并采取相应的措施。评估处理过程要么是采用 If 语句、要么是采用 Select 语句来完成的，在这两个语句中，应该总是包括 Else 子句，用它来处理那些所有没有预料到的错误。

### 2.7.3. 提供从错误处理程序跳出的出口

在错误处理实用程序内，通过测试 Err 对象的 Number 属性的值，确定发生的错误，

然后就有如下四种选择：

- Resume　返回到导致错误发生的语句；
- Resume Next　返回到导致错误发生的语句的下一行语句；
- Resume “行” 跳转到程序中行标号标明的行；
- 结束过程或者整个应用程序。

## 2.7.4. 错误处理的简单示例

本示例选自 VBA 帮助文档，示例首先使用 On Error GoTo 语句在一个过程中指定错误处理的代码所在。本示例中，试图删除一已经打开的文件从而生成的错误码为 55。这个错误将由示例中的错误处理程序码来处理，处理完後，控制会回到发生错误的语句处。On Error GoTo 0 语句关闭错误陷阱。然后 On Error Resume Next 语句用来改变错误陷阱，以便发觉下一个语句产生的错误的范围。请注意示例中使用 Err.Clear 在错误处理完後，清除 Err 对象的属性。

```
#001 Sub OnErrorStatementDemo()
#002     On Error GoTo ErrorHandler    ' 打开错误处理程序。
#003     Open "TESTFILE" For Output As #1    ' 打开输出文件。
#004     Kill "TESTFILE"    ' 试图删除已打开的文件。
#005
#006     On Error Goto 0    ' 关闭错误陷阱。
#007
#008     On Error Resume Next    ' 改变错误陷阱。
#009
#010     ' 试图启动不存在的对象
#011     ObjectRef = GetObject("MyWord.Basic")
#012
#013     '检查可能发生的 Automation 错误。
#014     If Err.Number = 440 Or Err.Number = 432 Then
#015         ' 告诉用户出了什么事。然后清除 Err 对象。
#016         Msg = "自动化错误!"
#017         MsgBox Msg
#018         Err.Clear    ' 清除 Err 对象字段。
#019     End If
#020
#021     Exit Sub       ' 退出程序，以避免进入错误处理程序。
#022
#023 ErrorHandler:    ' 错误处理程序。
#024     Select Case Err.Number    ' 检查错误代号。
```

```
#025         Case 55    ' 发生"文件已打开"的错误。
#026             Close #1    ' 关闭已打开的文件。
#027         Case Else
#028             ' 处理其他错误状态 . . .
#029     End Select
#030     Resume    ' 将控制返回到产生错误的语句。
#031 End Sub
```

# 2.8. 类模块和面向对象

## 2.8.1. 面向对象开发

本部分将介绍 VBA 中面向对象开发的问题和方法。

面向对象的第一个概念是对象（Object），对象是现实中的事物（实体、事件、过程）的数字化代表，例如一个对象可以代表现实中的人、房子，也可以代表一个组织甚至一次交易。对象可以包含对象，即一个对象可以由其他几个对象组成[7]。

类（Class）是一类对象模版，类定义了对象的属性和以及用来控制对象行为的方法。例如我们可以创建一个 People 的类，然后通过这个类创建很多的人的实例（instance）。类是由一系列的变量以及对变量操作的函数、方法组成的。

在 VBA 中，类通过类模块（Class Module）来实现。一个类模块代表一个类，其中包括属性（properties）、方法（methods）、事件（events）以及变量。属性代表了对象的一些特征，例如人的名字、身高；方法代表其可以做的事情，例如一个 Car 对象应该可以 Move，一个 Circle 对象应该可以计算面积等等；事件是用来使对象和创建对象的客户代码进行交互的一种有效手段，简单来说，事件是由客户代码创建的一段程序，而这段程序是由对象自己调用的。例如在用户窗体编程中，按钮点击事件代码是由用户窗体中书写的，而本事件的运行却是按钮对象本身激活的。在类模块中，事件可以用在通知客户数据改变类似用途[8]。

[7] 这里介绍的面向对象的概念只涉及到 VBA 中需要的概念，由于 VBA 并不是一个严格意义的面向对象语言，缺乏诸如继承等概念，因此，此类概念在本书不涉及。
[8] 对于没有事件机制的语言，对于此类通知需要使用 Observer 模式。

## 2.8.2. 对象变量和对象

我们可以使用已有的 VBA、Excel 对象模型等中的已有对象，也可以通过类模块创建自己的对象（Object）。一个对象可以包含属性（properties）、方法（methods）和事件（events）。前面已经讲过，对象必须通过显式创建，一个对象变量只是指向这个对象的一个指针（引用）。下面的介绍中，“对象变量”指我们创建的指向对象的变量，如 objExcel，一个对象可以有 0 个到多个对象变量指向它；对象、对象实例、实例则指对象本身，在计算机中，这是内存中的一段代码和数据[9]。很多 VBA 中，一个对象的创建可以使用以下 2 中语句：

```
Set ObjectVariable = New ClassName
```

或者：

```
Set ObjectVariable = ObjectExpression
```

使用 Set 创建对象实际上要做 2 件事情，首先释放此对象变量指向的对象，然后在将此变量指向已创建的对象 ObjectExpression。在 Dim、Public 和 Private 语句里的 New 语句并不立即创建一个新的对象，而是查找这个对象是否已经创建，如果已经创建，则将此对象变量指向已有的变量，否则新建一个对象。因此，在 Set 语句中的 New 比 Dim 等中高效一些，因为 VBA 无需检查是否新建了对象[10]。

VBA 编程中，除了类似 Excel 的 Appplication、WorkBook 这些已系统自动创建的对象，使用其他对象首先需要声明，声明之后再使用 New 语句创建之。对象的声明也有 2 种方式：

```
Dim ObjectVariable As ClassName
```

或者：

```
Dim ObjectVariable As New ClassName
```

前者只声明一个 ClassName 类型的变量，后者将自动创建一个对象。

VBA 包含有一特殊的值：Nothing。Nothing 表示现在的对象变量没有指向任何实例，使用 Dim 声明的对象默认指向 Nothing。我们可以使用 Nothing 来判断对象是否被创建或指向任何对象实例：

```
If ObjectVariable Is Nothing Then
```

也可以使用以下语句销毁一个对象：

---

[9] 打个比方，对象变量是人的名字、ID 等，对象是人本身，类是人这个物种。程序语言中，对对象的操作必须使用对象变量（ID）来执行。

[10] 在使用 CreateObject 时，一般先使用 GetObject，效果和 New 是类似的。

```
Set ObjectVariable = Nothing
```

在只有一个对象变量指向这个对象的时候，使用此语句即可销毁此对象，如果有多个对象变量指向此对象，则必须将所以对象变量设置为 Nothing 才可以释放其资源。

## 2.8.3. 创建类模块

使用类模块可以创建自定义的类，使用此类作为模版（对象类型）创建对象。在类模块中可以定义对象的属性和以及用来控制对象行为的方法。

类模块包含有一系列的属性、方法和事件，用来和其他对象或创建对象的代码交互。另外，类模块还可以包括一些局部的变量、私有过程、函数为其服务。一个对象（类模块）的属性、方法和事件都可以声明为 Public 或者 Private 两种类型。对被标记为 Public 的属性、方法、事件以及变量可以被使用此对象的任意代码使用，而标记为 Private 的元素只可以被此类模块的代码使用。

在类模块中，缺省的方法、属性是作为 Private 类型，但在实际编码时，最好还是显式声明属性、方法的作用范围：Public 还是 Private，以避免在编码和使用是混淆、迷惑。

通过在 VBA 工程菜单上右键单击选择“插入 — 类模块”可以插入一个类模块，类模块缺省以类 1、类 2（中文系统，英文系统为 Class1、Class2）命名。在资源管理器上选择类，然后在属性窗口中给此模块一个有意义的名字。

### 定义属性

类模块中有 2 类属性，第一类为 Public 类型的模块级变量，第二类为 Property 过程。对于模块级变量，必须在模块的最上部定义，同样，所有过程都必须在模块级变量定义之后。对定义为 Public 的变量，就是一个属性：

```
Public ProductId As Long
```

使用这种方法定义的属性不能返回数组、定长字符串、常量或者用户自定义类型（使用 Type 定义的变量）。如果要使用返回这类变量，需要定义一个 Property 过程。Property 过程是一系列的 VBA 语句，它允许程序员去创建并操作自定义的属性。使用 Property 过程不仅仅是为了返回 Public 级变量不能返回的变量，其主要目的是一个属性在设置、获取其具体值的时候可以进行一系列的运算、验证等操作。例如一个税额的属性，可以针对具

体情况计算其税额，这就是单单一个变量不能做到的。

当创建一个 Property 过程时，它会变成此过程所包含的类模块的一个属性。VBA 提供下列三种类型的 Property 过程：

- Property Let, 用来设置属性值的过程；
- Property Get, 用来返回属性值的过程；
- Property Set, 用来设置对对象引用的过程，只有属性为一个对象时使用。

声明 Property 过程的语法如下所示：

```
[Public | Private] [Static] Property {Get | Let | Set} _
  propertyname [(arguments)] [As type]
    Statements
    statements
End Property
```

Property 过程通常是成对使用的：Property Let 与 Property Get 一组，而 Property Set 与 Property Get 一组。单独声明一个 Property Get 过程就象声明只读属性。三个 Property 过程一起使用时，只有对 Variant 变量有用，因为只有 Variant 才能包含一个对象或其它数据类型的信息。Property Set 本意是使用在对象上；而 Property Let 则不是。

在 Property 过程声明中所需要的参数为：

- Property Get　Property Get propname(1, ..., n) As type
- Property Let　Property Let propname(1, ...,,,, n, n+1)
- Property Set　Property Set propname(1, ..., n, n+1)

在具有相同名称的 Property 过程中，从第一个到最后一个参数（1, ..., n）都必须共享相同的名称与数据类型。

Property Get 过程声明时所需的参数比相关的 Property Let 以及 Property Set 声明少一个。Property Get 过程的数据类型必须与相关的 Property Let 以及 Property Set 声明中的最后(n+1)个参数的类型相同。例如，如果声明下列的 Property Let 过程，则 Property Get 声明所使用参数的名称与数据类型必须 Property Let 过程中所用一样。

```
Property Let Names(intX As Integer, varZ As Variant)
    ' Statement here.
End Property

Property Get Names(intX As Integer) As Variant
    ' Statement here.
End Property
```

在 Property Set 声明中，最后一个参数的数据类型必须是对象类型或是 Variant。

### 定义方法

方法就是一个过程或函数，定义为 Public 的方法可以为使用此类模块的客户代码使用，而定义为 Private 的方法只可以在类内部使用。

### 定义事件

可以使用 Event 语句来定义事件。Event 的语法如下：

```
[Public] Event procedurename [(arglist)]
```

Event 语句包含下面 2 部分：

Public，可选的。指定该 Event 在整个工程中都是可见的。缺省情况下 Events 类型是 Public。应注意，事件只能在所声明的模块中产生。

Procedurename，必需的。事件的名称；遵循标准的变量命名约定。

arglist 参数的语法及语法的各个部分如下，其说明见表 2-15：

```
[ByVal | ByRef] varname[( )] [As type]
```

表 2-15 事件 Event 参数说明

| 部分 | 描述 |
|---|---|
| ByVal | 可选的。表示该参数是按值传递的。 |
| ByRef | 可选的。表示该参数是按地址传递的。ByRef 是 Visual Basic 的缺省设置。 |
| *varname* | 必需的。代表要传递给过程的参数变量的名称；遵循标准的变量命名约定。 |
| *type* | 可选的。指传递给过程的参数的数据类型；可以是 Byte、 Boolean 、Integer、Long、Currency、Single、Double、Decimal（目前尚不支持）、Date、String（只支持变长）、Object、Variant、用户定义类型或对象类型。 |

事件被声明之后，就可以使用 RaiseEvent 语句来产生该事件。不能声明带返回值的事件。在下面的代码段中，给出了声明事件和产生事件的典型事件：

```
#001 '在类模块的模块级中声明一个事件
#002 Public Event DataChangeded (DataID as Long)
#003
#004 Sub DataProcess()
#005    … …
#006    '产生事件
#007    RaiseEvent LogonCompleted(mID)
#008 End Sub
```

以上代码，在第 2 行声明了一个事件，在第 7 行触发了该事件。可以象声明过程的参数一样来声明事件的参数，但有以下不同：事件不能有带命名参数、Optional 参数、或者 ParamArray 参数。事件没有返回值。

在类模块、Excel 对象模块或用户窗体代码中[11]，可以在创建对象时使用 WithEvents 关键字来使用事件，例如对以上定义的事件，我们可以使用以下方法来使用：

```
Dim WithEvents objTemp As MyObject

'事件处理程序
Private Sub objTemp_ DataChangeded()
   MsgBox "Hello，Events！"
End Sub
```

在 VB6 中，New 和 WithEvents 不能一起使用，所以要么首先使用 Dim 时使用 New 创建对象，然后使用 WithEvents 再创建一个对象并绑定到这个对象，或者使用 Dim WithEvents 时不使用 New 创建，而后再其他代码中初始化对象。

VBA defines two special events for all classes, the.

VBA 定义了 2 个特殊的事件，Initialize 事件和 Terminate 事件，Class_Initialize 事件的代码在类被创建时执行，Class_Terminate 的代码在类被销毁时执行。

在类模块或用户窗体代码内部，可以使用 Me 关键字来引用该类模块所有的元素，包括 Public 和 Private 类型以及所有的属性、方法、变量，Me 也可以作为参数传递。

## 2.9. COM 对象的使用

在 VBA 中，可以使用 ActiveX 对象或者使用一般的 COM 对象，使用方法和在 VB 中一致。如果需要使用其他第三方的 ActiveX 对象，可以选择“*工具 － 附加控件*”或者在工具箱上单击右键选择“附加控件”，在附加控件对话框选择任何第三方控件，将其添加到工具箱[12]，添加了的控件，可以和已有控件一样的使用（图 2-10）。

[11] 在一般模块中不能使用 WithEvents 来创建事件。如果不使用 WithEvents，则事件被忽略。

[12] 在此对话框，只显示已经在系统注册表注册的对象，对于没有注册的对象，在 VB 中，有一个浏览按钮可以浏览定位，并自动注册，在 VBA 中，则没有此功能。读者可以使用命令行“Regsvr32”注册或者反注册 COM 组件。


图 2-10 附加控件对话框

对于第三方的 COM 控件，例如其他的 Office 组件、数据库连接组件 ADO 等等，在使用前需要在引用对话框中添加引用。选择“*工具 － 引用*”打开引用对话框（类似于附件控件对话框），选择需要的引用。然后就可以在代码中和使用 Excel 内置函数一样使用添加了引用的 COM 对象。

## 2.10. 集合对象

VBA 编程需要熟悉的另一个对象是 Collection 对象（集合对象），一个集合对象就是一系列对象的集合。例如，一个 WorkBook 对象包括 1 个到多个 WorkSheet 对象。Collection 对象提供了简便方法，直截了当将一组相关的对象视为单一对象来引用。集合中的对象都属于这个集合。当然，集合的成员不一定都是同一种数据类型的。

建立集合的方法与建立其它对象的方法一样：

```
Dim X As New Collection
```

一旦建立集合之后，就可以用 Add 方法添加成员，用 Remove 方法删除成员。在用 For Each...Next 语句循环整个集合时，可以用 Item 方法从集合返回特定成员。

使用集合对象可以使用 For…Next 循环，也可以使用 For Each…Next 循环，使用前者可以使用数字索引每个成员对象，使用后者则不需要。例如，需要遍历工作薄的所有工作薄，可以使用以下代码：

```
For Each objWorkSheet in Worksheets
    objWorkSheet.PageSetup.RightFooter = Path
Next objWorkSheet
```

在 Excel 对象模型中，诸如 WorkSheets，Addins，WorkBooks 等都是集合对象。当我们需要将数个对象统一操作时，就需要使用集合对象或数组，二者的选取见以下说明。

数组和集合的选择

在 VBA 中，可以使用集合对象（见集合对象一节），创建一个可以容纳任何对象或数据变量的集合，可以动态增加、删除对象。那么，如何选择使用集合和动态数组呢？基本原则如下：数组是保存一定数量的相同元素的集合，其大小增长要求不是很频繁，例如需要处理的一组数据，而集合则主要用于保存需要频繁增删元素，例如配置信息；数组要比集合快很多（大概要快 1 个数量级）。因此，集合应该只应用在数据量不大（小于 $10^5$）且需要频繁增加删除元素的情况下。

# 3. Excel 的对象模型

上一章介绍了 VBA 的 IDE 环境和 VBA 语法，这一章介绍 Excel 对象模型，VBA 语法和 Excel 对象模型组成了应用 Excel 和 VBA 进行开发的基础。从某种程度上讲，理解和熟悉 Excel 对象模型的过程，也就是使用 Excel 和 VBA 进行开发的过程。

和使用其他程序开发语言和环境一个很大的差别是，Excel VBA 开发必须时刻牢记，开发的目的是解决问题，解决问题的关键在于使用已有的功能，补充和开发缺乏或者很难用的功能（例如某些函数），才是我们使用 Excel 和 VBA 进行开发的目的。这一切的前提就是熟悉 Excel 对象模型。

另一方面，Excel 对象模型包括了大量的对象、属性和方法，任何人也不可能记住所有的内容，因此，熟悉是指熟悉其结构和组成，对于一个对象的具体方法和属性则没有必要记住，完全可以在开发的时候通过查看帮助或者对象浏览器来获得帮助。

## 3.1. Excel 对象模型简介

前边已经谈到，Visual Basic for Application（VBA）通过对象（Object）来操作和控制 Excel，不管是操作 Excel 程序（Application 对象）、工作薄（Workbook 对象），还是操作工作表（Worksheet 对象）或其中的单元格（Cell 对象），我们都是在操作对象。所有的对象或者由其他对象组成，或者是其他对象的一部分，或者 2 者兼是，例如，Workbook 对象包含有 Worksheets 对象，而 Worksheet 对象包含 Cell 对象。

操作一个对象时，我们可以通过读取和设置其属性，或者调用其方法，例如，我们可以通过 Name 属性修改活动工作薄中名为 Sheet2 的工作薄的名称，并通过 Activate 方法激活它，代码如下：

```
Worksheets("Sheet2").Name = "NewName"
Worksheets("NewName").Activate
```

图 3-1 是 Excel 对象模型图部分，可以通过 Excel 或者 VBA 帮助打开之，单击其中某个对象则可以跳转到对象的说明。从对象模型，我们可以看到对象之间的包含关系。

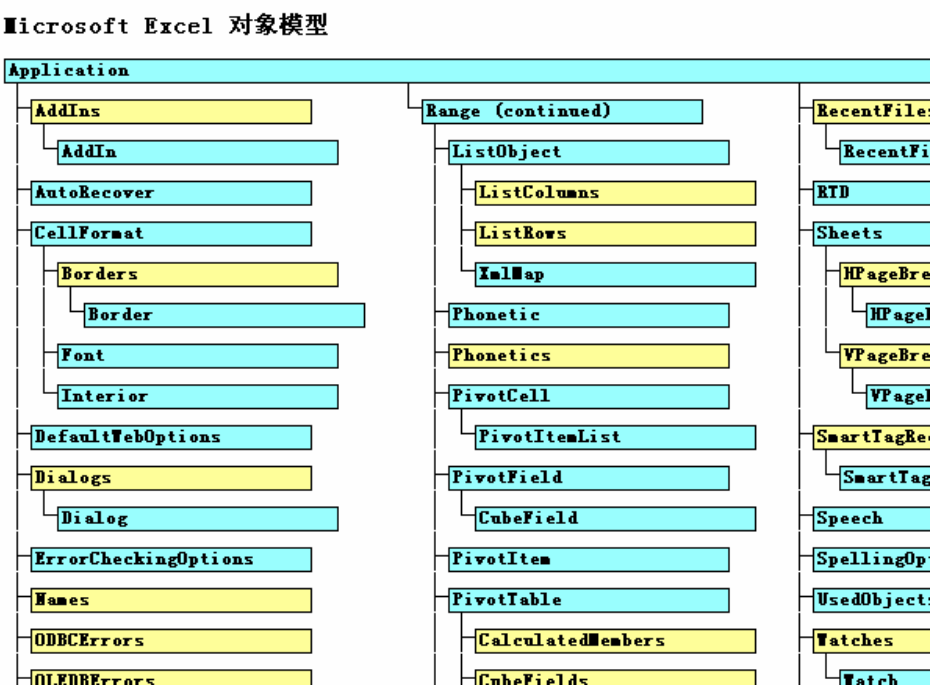



图 3-1 Microsoft Excel 对象模型

Excel 中最顶端的对象为 Application 对象，代表了 Excel 程序本身，Application 对象包含了数个 Workbook 对象（通过 Workbooks 集合对象引用），Workbook 对象则包含数个 Worksheet 对象，Range 对象和 Chart 对象则代表了工作薄中的单元格或图表。

当一个对象包含另一个对象的多个实例，称为集合（Collection），例如 WorkBook 对象有 WorkSheets 集合，WorkSheets 集合中包含了工作表中的所有工作薄，我们可以通过名称或数字序号来引用其中的 Worksheet。本章开头的例子我们就使用了 Worksheets 集合。

可以在 Excel VBA 编程中通过读取和设置属性，或者调用方法来操作 Excel，也可以编写工作表级、图表级、查询表级、工作簿级或应用程序级的事件过程。例如，Activate 事件发生在工作表级，而 SheetActivate 事件既可发生在工作簿级，也可发生在应用程序级。工作簿的 SheetActivate 事件发生在激活该工作簿中的任一工作表时，而应用程序级的 SheetActivate 事件发生在任一打开的工作簿中的任一工作表被激活时。

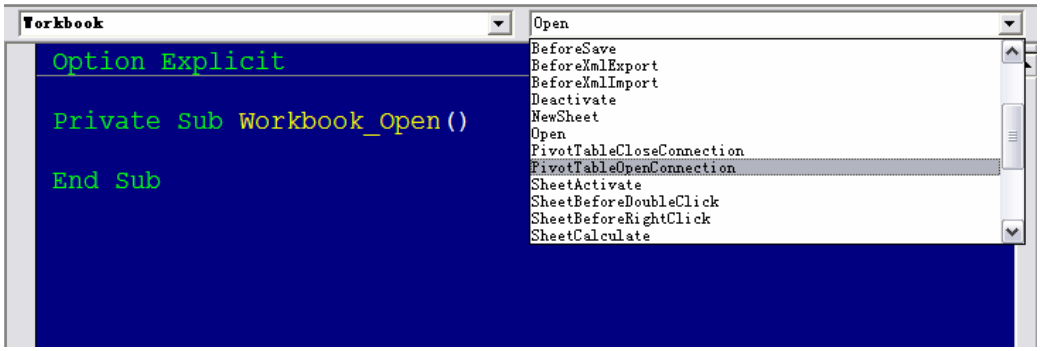



图 3-2 在 VBA IDE 中编写事件过程

工作表、图表工作表和工作簿事件过程对任意打开的工作表或工作簿都有效，可以在各自的模块中编写（图 3-2）。若要为嵌入图表、QueryTable 对象或 Application 对象编写事件过程，则必须在类模块中用 WithEvents 关键字创建新的对象（见 Application 对象一节最后的例子）。也可用 Application 的 EnableEvents 属性来启用或禁用事件。

## 3.2. Application 对象

Application 对象是 Excel 对象模型的最上部的对象，代表了 Excel 应用程序本身。Application 对象提供了大量属性、方法和事件，用来操作 Excel 程序，其中的许多成员我们可能从来不会使用，但是其他的一些成员却是非常重要的。简单来说，可以将这些成员分为以下种类：

- 控制 Excel 状态和显示的成员；
- 返回对象的成员；
- 执行操作的成员；
- Window 对象及其集合；
- Application 的事件；

以下部分将主要介绍以上这些重要的 Application 对象的成员。

因为 Application 对象是 Excel 对象模型的最顶端对象，逻辑上所有 Excel 对象都需要使用 Application 对象来引用，例如要引用第一个工作表的第一个工作表的“A1”单元格，我们需要这么做：

```
Application.Workbooks(1).Worksheets(1).Cells(1,1) = 100
```

为了方便，Excel 允许对一些常用对象直接使用，不需要使用 Application 对象引用，例如 ActiveSheet，Workbooks，Worksheets 等对象。

### 3.2.1. 控制 Excel 状态和显示的属性

Application 对象提供了一个很大的属性集来控制 Excel 的状态。表 3-1 列出了与状态有关的 Application 对象属性的一个子集。

表 3-1 一些控制 Excel 状态的 Application 属性

| 属性 | 类型 | 说明 |
|---|---|---|
| Cursor | XlMousePointer | 获取或设置鼠标指针的外观。 |
| EditDirectlyInCell | 布尔值 | 直接就地获取或设置编辑单元格的能力。如果为 False，则您只能在公式栏中编辑单元格。 |
| EnableEvents | 布尔值 | 可用 EnableEvents 属性来启用或禁用事件。 |
| FixedDecimal | 布尔值 | 如果为 True，则所有的数字值都使用 FixedDecimalPlaces 属性来确定小数位数；否则将忽略 FixedDecimalPlaces 属性（默认值为 False）。 |
| FixedDecimalPlaces | Long | 确定用于数值数据的小数位数（如果 FixedDecimal 属性为 True）。 |
| Interactive | 布尔值 | 获取或设置用户通过键盘和鼠标与 Excel 交互的能力；如果将此属性设置成 False，则一定要确保在异常处理程序中将其重新设置成 True。Excel 不会自动为您重新设置它。 |
| MoveAfterReturn | 布尔值 | 如果为 True，则当您按下 Enter 键时，选择会移到下一个单元格；默认值为 True。 |
| MoveAfterReturnDirection | xlDirection (xlDown, xlToLeft, xlToRight, xlUp) | 指示在按下 Enter 键之后移动的方向（如果 MoveAfterReturn 属性为 True）。默认值为 xlDown。 |
| ScreenUpdating | 布尔值 | 如果为 True，Excel 就会在每个方法调用之后更新其屏幕。为了节省时间并且使您的应用程序看起来更加专业，您可以在代码运行时关掉显示。一旦完成，就一定要再次将此属性值重新设置为 True。Excel 不会自动为您重新设置它。 |
| SheetsInNewWorkbook | Long | 获取或设置 Excel 自动放置在新的工作簿中的工作表的数目。 |
| StandardFont | 字符串 | 获取或设置 Excel 中默认字体的名称；只有在重新启动 Excel 之后才会生效。 |
| StandardFontSize | Long | 获取或设置 Excel 中默认字体的大小；只有在重新启动 Excel 之后才会生效。 |
| StartupPath（只读） | 字符串 | 返回包含 Excel 启动加载项的文件夹的完整路径。 |
| TemplatesPath（只读） | 字符串 | 返回包含模板的文件夹的完整路径；此值代表着一个 Windows 特殊文件夹。 |

在表 3-1 所列出的所有属性中，常用的一个属性是 ScreenUpdating 属性。通过利用这个属性，可以使 Excel 在每次修改数据后不更新显示，因为更新显示会严重影响代码的运行效率，特别是在通过编程方式大量修改数据表数据或图表时。必须记住，当使用 ScreenUpdating 属性时，始终需要在数据修改完成后将其设置为 True（第 9 行）：

```
#001    Dim i As Long
#002
#003    Application.ScreenUpdating = False
#004
#005    For i = 1 To 1000 Step 1
#006        Workbooks(1).Worksheets(1).Cells(i, 1).Value = i
#007    Next i
#008
#009    Application.ScreenUpdating = True
```

可用 EnableEvents 属性来启用或禁用事件。例如，使用 Save 方法保存工作表时，将引发 BeforeSave 事件。可在调用 Save 方法之前将 EnableEvents 属性设置为 False，以防止该事件的发生。

```
Application.EnableEvents = False
ActiveWorkbook.Save
Application.EnableEvents = True
```

Application 对象还提供了一组控制 Excel 中的显示的属性，例如 DisplayAlerts（是否显示警告信息）、DisplayFormulaBar（是否显示公式栏）、DisplayFullScree（全屏）等，可以修改这些属性中的任何一个来改变用户在屏幕上所看到的内容。列出了可用的显示选项的一个子集。

## 3.2.2. 返回对象的属性

许多 Application 对象的属性用来返回其他的对象。我们通常需要利用 Application 对象来引用 Excel 提供的其他对象。例如通过诸如 ActiveWindow 的属性返回当前活动的窗口，通过诸如 Charts 的属性返回图表对象的集合。表 3-2 列出了 Application 对象的返回对象的属性的一个子集。

表 3-2 Application 对象返回对象的属性的一个子集

| 属性 | 类型 | 说明 |
|---|---|---|

| | | |
|---|---|---|
| ActiveCell | 范围 | 返回对活动窗口（顶部的窗口）中当前活动单元格的引用。如果没有活动窗口，此属性会产生一个错误。 |
| ActiveChart | 图表 | 返回对当前活动的图表的引用。对于一个嵌入式图表来说，只有当此图表被选中或被激活时才可认为是活动的。 |
| ActiveSheet | 对象 | 返回对活动工作簿中的活动工作表的引用。 |
| ActiveWindow | 窗口 | 返回对活动窗口（顶部的窗口）的引用；如果没有活动窗口，则不返回任何结果。 |
| Charts | 工作表 | 返回 Sheet 对象（Chart 和 Worksheet 对象的父对象）的集合，这些对象包含对活动工作簿中的每个图表的引用。 |
| Selection | 对象 | 返回应用程序中选中的对象。可能是一个 Range、一个 Worksheet 或任何其他的对象 — 同样适用于 Window 类，在这种情况下，选择通常是一个 Range 对象。如果当前没有对象被选中，则不返回任何结果。 |
| Sheets | 工作表 | 返回 Sheet 对象的集合，这些对象包含对活动工作簿中每个工作表的引用。 |
| Workbooks | 工作簿 | 返回 Workbook 对象的集合，这些对象包含对所有打开的工作簿的引用。 |

例如，与 Application 类的 Workbooks 属性交互能够循环访问打开的工作薄、打开或创建一个新的工作簿。Workbooks 集合使得有可能使用所有打开的工作簿、创建一个新的工作簿以及将数据导入一个新的工作簿。

**创建一个新的工作簿**

使用如下代码（也可以指定一个工作簿模板的名称作为 Add 方法的参数）：

```
Workbooks.Add
```

**关闭所有打开的工作簿**

与大多数的集合不同，Workbooks 集合允许一次性地关闭所有的成员，如果有工作薄没有保存，会提示是否保存。下面的方法调用关闭所有打开的工作簿：

```
Workbooks.Close
```

**打开一个现有的工作簿**

最简单的形式是使用 Open 方法，如下面的代码片段所示。Open 方法提供了大量的可选参数，但是通常不需要使用这些可选参数：

```
Workbooks.Open "C:\YourPath\YourWorkbook.xls"
```

以工作簿的形式打开一个文本文件、数据库或 XML 文件（使用 OpenText、OpenDatabase

或 OpenXml）。在数据处理一章，我们还要回到此话题。例如，可以使用如下代码以工作簿的形式加载一个文本文件（使用逗号作为分隔符，从文本文件中的第三行开始）：

```
Workbooks.OpenText "C:\Test.txt", StartRow:=3, _
    DataType:=xlDelimited, Comma:=True)
```

**引用工作簿**

可以使用整数（指示在集合中的位置,使用 Count 属性返回工作薄个数）或工作簿名作为 Workbooks 集合中的索引。通过名称引用工作簿，必须使用在标题栏看到的名称，例如在保存该文件之前，这个名称不包括“.xls”扩展名，例如：

```
Workbooks(1)
Workbooks("Book1")
Workbooks("Book1.xls")
```

## 3.2.3. 执行操作

Application 对象提供了许多允许执行操作的方法，例如从重新计算当前数据到撤销对数据的更改。以下例举了部分常用方法：

**Calculate**

强制重新计算所有打开的工作簿、特定的工作簿或者特定的范围：

```
Application.Calculate
'或者
Worksheets(1).Calculate
'或者
Application.Range("A3:C23").Calculate
```

除了 Application，Range 和 Worksheet 对象也提供 Calculate 方法。使用可以将计算范围限定在需要重新计算的最小单元格数内的对象的方法。虽然 Excel 中的重新计算引擎非常快，但如果可以限制所涉及到的单元格数，我们还是可以优化这一操作。只有在需要重新计算每个打开的工作簿中的每个未决更改时才使用 Application.Calculate。

**Checkspelling**

返回一个 Boolean 来指示提供的参数是否拼写正确。您可以选择提供一个自定义字典的名称和一个 Boolean 来指示您是否想要忽略大小写。下面的代码片段检查您所提供的值

的拼写，并且在工作表上指示其结果：

**Evaluate**

将一个 Microsoft Excel 名称转换为一个对象或者一个值。这个方法允许您以字符串的形式创建引用，并且在需要时将其转换成一个实际对象引用，或者求表达式的值。这个方法类似与一些脚本语言中的“Eval”方法。使用方法：

```
expression.Evaluate(Name)
```

下列几类 Microsoft Excel 名称可以使用此方法：

- A1-样式引用。可以引用任何以 A1-样式符号表示的单个单元格。所有引用都是绝对引用。
- 单元格区域。可在区域引用中使用区域、交集和联合运算符（分别为冒号、空格和逗号）。
- 已定义的名称。可用宏语言指定任意名称。
- 外部引用。可以使用“!”操作符引用另一工作簿上的单元格或已定义的名称。例如，Evaluate("[BOOK1.XLS]Sheet1!A1")。

另外，使用方括号（例如，"[A1:C5]"）与用字符串参数调用 **Evaluate** 方法是等效的。例如，下列表达式对是等价的。

```
[a1].Value = 25
Evaluate("A1").Value = 25

Set firstCellInSheet = Workbooks("BOOK1.XLS").Sheets(4).[A1]
Set firstCellInSheet = _
    Workbooks("BOOK1.XLS").Sheets(4).Evaluate("A1")
```

使用方括号的优点在于代码较短。使用 Evaluate 的优点在于参数是字符串，这样您既可以在代码中构造该字符串，也可以使用 Visual Basic 变量。

下例将工作表 Sheet1 上 A1 单元格的字体设置为加粗。

```
Worksheets("Sheet1").Activate
boldCell = "A1"
Application.Evaluate(boldCell).Font.Bold = True
```

**发送电子邮件**

Excel 允许您登录您所安装的电子邮件系统、将当前的工作簿作为附件发送、以及从

您所安装的电子邮件系统中注销。MailSystem 属性指出已安装的电子邮件系统，而 MailSession 属性返回对当前电子邮件会话（如果有活动的会话，您就不需要登录）的引用，SendMail 可以发送邮件。下面的示例将示例工作簿作为一个简单的电子邮件消息的附件发送：

```
Private Sub TestEmail()
    If IsNull(Application.MailSession) Then
        Application.MailLogon
    End If
    ActiveWorkbook.SendMail "abc@test.com", "Subject"
    Application.MailLogoff
End Sub
```

**Quit**

通过编程方式退出 Excel。如果您将 DisplayAlerts 属性设置为 False，则系统不会提示您保存任何未保存的数据。此外，如果您将 Workbook 的 Saved 属性设置为 True，则不管您有没有进行更改，Excel 都不会提示您保存它：

```
Application.Quit
```

**Undo**

取消用户在用户界面内进行的最后一次操作。这个方法不会对代码进行的操作产生影响，并且只能撤销单个操作：

```
Application.Undo
```

**WorksheetFunction**

Application 对象包含一个属性 WorksheetFunction，这个属性返回 WorksheetFunction 类的实例。用作可从 VBA 中调用的 Microsoft Excel 工作表函数的容器。下面的示例显示了对单元格区域 A1:A10 应用工作表函数 **Min** 的结果。

```
Set myRange = Worksheets("Sheet1").Range("A1:C10")
answer = Application.WorksheetFunction.Min(myRange)
MsgBox answer
```

另外，也可以直接使用“Application.Excel 工作表函数”的形式使用 Excel 内置函数，但使用 WorksheetFunction 更清晰。Excel 中内置了大量进行数据处理、统计分析、财务计算的函数，使用内置函数不仅可以节约编程时间，提高编程效率，而且运算速度较高。使

用工作表函数还不仅可以使用 Range 作为参数，也可以使用其他类型的参数，例如：

```
dblAverage = Application.WorksheetFunction.Average( _
    12, 14, 13, 19, 21)
```

**Dialogs**

Dialogs 属性，返回一个 Dialogs 集合，此集合代表所有的内置对话框。只读。用来内置对话框，例如显示“文件”菜单的“打开”对话框。

```
Application.Dialogs(xlDialogOpen).Show
```

Dialog 对象是 Dialogs 集合的成员之一。Dialogs 包含所有的 Microsoft Excel 内置对话框。不能新建内置对话框或向该集合中添加内置对话框。用 Dialog 对象所能做的唯一有用的事情是将其与与 Show 方法共用，以显示相应的对话框。可用 Dialogs(index)返回单个 Dialog 对象，其中 index 为用于标识对话框的内置常量。下例运行“文件”菜单中的内置“打开”对话框。如果 Microsoft Excel 成功地打开了文件，则 Show 方法为 True；如果用户取消了对话框，则该值为 False。

```
dlgAnswer = Application.Dialogs(xlDialogOpen).Show
```

Excel 对象库包括了许多内置对话框的内置常量。每个常量都以“xlDialog”打头，后跟对话框的名称。

**OnTime**

安排一个过程在将来的特定时间运行（既可以是具体指定的某个时间，也可以是指定的一段时间之后）。使用方法：

```
expression.OnTime(EarliestTime, Procedure, _
    LatestTime, Schedule)
```

Expression 必须是一个 Application 对象，EarliestTime 设置过程开始运行的时间。Procedure 设置要运行的过程名。LatestTime 表示过程开始运行的最晚时间。例如，LatestTime 参数设为 EarliestTime + 30，当时间到了 EarliestTime 时，如果由于其他程序处于运行状态 Microsoft Excel 不处于“就绪”、“复制”、“剪切”或“查找”模式，则 Microsoft Excel 将等待 30 秒让第一个过程先结束运行。如果 30 秒内 Microsoft Excel 不能回到“就绪”模式，则不运行此过程。如果省略该参数，Microsoft Excel 将一直等待到可以运行该过程为止。

Schedule，可选。如果该值为 True，则安排一个新的 OnTime 过程。如果该值为 False，则清除先前设置的过程。默认值为 True。因此使用 Now + TimeValue(time) 可安排经过一

段时间（从现在开始计时）之后运行某个过程。使用 TimeValue(time) 可安排某个过程只运行指定的时间。

下例设置 15 秒后运行 my_Procedure 过程，从现在开始计时。

```
Application.OnTime Now + TimeValue("00:00:15"), _
    "my_Procedure"
```

下例设置 my_Procedure 在下午 5 点开始运行。

```
Application.OnTime TimeValue("17:00:00"), "my_Procedure"
```

## 3.2.4. Window 对象和 Windows 集合

可以使用 Application 对象的 Windows 属性来打开、关闭、激活和排列 Excel 对象窗口。

Windows 属性返回 Window 对象的集合，并且您可以调用 Arrange 方法来排列所有打开的窗口（可见的窗口）。指定一个 XlArrangeStyle 枚举值来指示您想要以何种方式排列窗口，并且还可以选择指定一些关于您是否只想排列可见的窗口、以及您想如何同步窗口滚动的信息。例如，要在 Excel 工作区中平铺显示窗口，您可以使用如下代码：

```
Application.Windows.Arrange(xlArrangeStyleTiled)
```

可以通过编程方式调用工作簿的 **NewWindow** 方法创建一个新的窗口，设置新窗口的标题，然后并将其激活：

```
With ThisWorkbook.NewWindow
    .Caption = "New Window"
    .Activate
End With
```

Windows 类提供控制相关窗口的外观和行为的属性和方法，包括颜色、标题、窗口特性的可视性、以及滚动行为。

## 3.2.5. Application 事件

Application 对象除了提供大量属性和方法，还有一大组事件可用。根据名称，就可以比较清楚地知道它们的用途。下面将描述了这些事件的一个子集，并讨论如何使用这些事件。事件的使用方法见 VBA 一章。

Application 对象提供了各种与表（包括图表和工作表）相关的事件。

**SheetActivate**

当任何一个表被激活时，SheetActivate 都会发生。Excel 将一个包含对被激活的表的引用的 Object 变量传递给事件处理程序。

**SheetBeforeDoubleClick**

在 Excel 提供默认的双击处理之前，当任何表被双击时，SheetBeforeDoubleClick 都会发生。Excel 将下列参数传递给事件处理程序：一个包含对表的引用的 Object 变量、一个包含离双击位置最近的单元格的 Range 对象、一个允许您取消默认事件处理的 Boolean 值（默认为 False）。

VBA 中所有名称中包括单词“Before”的事件都允许取消默认的事件处理。传递给事件处理程序的参数通常名为 Cancle，具有默认值 False。如果将这个参数设置为 True，Excel 将不会执行事件的默认处理。

**SheetBeforeRightClick**

在 Excel 提供默认的右键单击处理之前，当任何表被右键单击时，都会发生。Excel 将下列参数传递给事件处理程序：一个包含对表的引用、一个包含离右击位置最近的单元格的 Range 对象、一个允许您取消默认事件处理的 Boolean 值（默认为 False）。

**SheetCalculate**

当任何表被重新计算时，SheetCalculate 都会出现。Excel 将一个包含对重新计算的表的引用传递给事件处理程序。

**SheetChange**

当任何工作表中的单元格发生变化（通过用户或者通过运行代码）时，都会发生。Excel 将一个 Object 变量（包含对表的引用）和一个 Range 变量（引用改变的范围）传递给事件处理程序。

**SheetDeactivate**

当任何表单被停用时（即当它不再有焦点时），SheetDeactivate 都会发生。只有当焦点转移到同一工作簿内的另一个表时，这个事件处理程序才会运行。Excel 将一个包含对已

经停用的表的引用的 Object 变量传递给事件处理程序。

**SheetSelectionChange**

当工作表上的选择改变时，SheetSelectionChange 会发生。Excel 将一个引用选择发生改变的表的 Object 变量和一个引用新选择的 Range 变量传递给事件处理程序。

以上事件也可用 Workbook 类提供的事件。如果该事件是由 Application 对象提供的，则它可以被 Excel 内当前打开的任何一个表引发。当它是由 Workbook 对象提供的，则该事件只有在它影响特定工作簿中的一个表时才会发生。此外，Worksheet 也提供的相同事件。

Application 对象和相应的 Workbook 对象提供了各种处理 Window 对象的行为的事件。下面的列表描述了这些事件。例如，当任何窗口被激活时，WindowActivate 都会发生；当任何窗口被停用时，WindowDeactivate 都会发生；当任何工作簿窗口重新调整大小时，WindowResize 都会发生。

在使用 VBA 创建 Excel 程序的时候，大多数情况下是在 Excel 的 Workbook 或者 Worksheet 对象中响应事件，这种情况只需要从代码窗口的对象列表框选择对象，然后在方法列表框选择事件，即可在 IDE 自动生成的事件处理过程中书写事件处理程序。

但对于 Application 事件，需要在用户窗体或类模块中通过 WithEvents 语句定义响应事件的对象，通过此对象编写事件响应过程（见本节最后的例子）。

Application 对象提供了各种与任何 Workbook 对象交互时都会发生的事件。这些事件过程中的每一个都接收 Workbook 变量，该变量指示参与事件的特定工作簿。例如，当创建一个新的工作簿时，NewWorkbook 会发生；当任何工作簿被激活时，WorkbookActivate 都会发生；当一个打开的工作簿关闭时，WorkbookBeforeClose 会发生；当工作簿内的打印刚好在默认事件处理之前开始时，WorkbookBeforePrint 会发生；当刚好在默认事件处理之前保存工作簿时，WorkbookBeforeSave 会发生；当任何工作簿被停用时，WorkbookDeactivate 都会发生；当将新的表添加到工作簿时，WorkbookNewSheet 会发生；当一个工作簿打开时，WorkbookOpen 会发生。另外，Workbook 也提供了类似的一组事件。

我们可以在类模块和用户窗体中使用 Application 事件，对于 Wooksheet 和 Workbook

的事件，可以直接在其模块下书写响应代码，也可以通过 WithEvents 语句在类模块或者其他模块中响应。例如，我们建立如下测试程序：

第一，新建一个窗体，名称为 frmEventForm，在其上添加一个 ListBox，名称为 lstEvent，添加如下代码：

```
#001 Public WithEvents MyAppEvent As Application
#002
#003 Private Sub MyAppEvent_NewWorkbook(ByVal Wb As Workbook)
#004     Me.lstEvent.AddItem "新建工作薄"
#005 End Sub
#006
#007 Private Sub MyAppEvent_SheetActivate(ByVal Sh As Object)
#008     Me.lstEvent.AddItem Sh.Name & "激活"
#009 End Sub
#010
#011 Private Sub MyAppEvent_SheetChange(ByVal Sh As _
          Object, ByVal Target As Range)
#012     Me.lstEvent.AddItem Sh.Name & ":" & _
          Target.Address & "改变了"
#013 End Sub
#014
#015 Private Sub MyAppEvent_WindowActivate(ByVal Wb _
          As Workbook, ByVal Wn As Window)
#016     Me.lstEvent.AddItem Wb.Name & "激活"
#017 End Sub
```

其中第一行表示响应 Application 对象的事件，然后通过在代码窗口上方的对象框和方法事件框就可以选择事件。

第二，新建一个过程，输入如下代码：

```
#001 Dim frm As New frmEventForm
#002 Set frm.MyAppEvent = Application
#003 frm.Show vbModeless
```

第一行新定义这个窗体的实例，然后将窗体的 MyAppEvent 的引用指向 Application 对象，使其响应 Application 对象的事件，然后使其显示为无模式窗体，这样我们还可以和 Excel 交互。运行结果如图 3-3 所示。

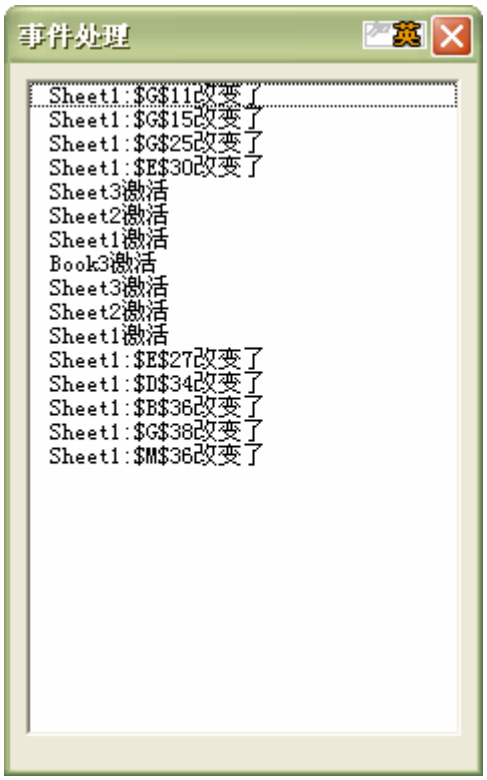



图 3-3 响应 Application 对象的事件

以上方法是使用 Application 事件的一般方法，当然，我们也可以在类模块中响应 Application 事件。通过响应 Application 事件，我们可以精确的控制程序的运行，例如在编写自己的程序时，可以控制程序窗口、工作薄的切换，对任何单元格的改变作出响应，等等。

## 3.3. WorkBook 对象

Workbook 类代表了 Excel 的一个单一的工作簿，WorkSheet 类则代表了工作薄中的一个工作表，本部分将主要介绍 Workbook 类和其相关的类，包括最常使用的属性和方法。

### 3.3.1. Workbooks 集合

Workbooks 集合包含了 Excel 程序中所有打开的工作薄，我们可以使用 For Each…Next 循环来遍历 Workbooks 集合。我们也可以使用 Workbooks 集合创建新的工作薄（Workbook），关闭操作工作薄。

在介绍 Application 对象的时候，我们已经介绍了可以使用 Workbooks.Add 方法来新建

工作薄，同时也可以为工作薄指定一个模版；可以使用 Open 方法打开已有工作薄；使用工作薄名称或者数字索引引用工作薄；或者通过 Close 方法关闭所有工作薄。

## 3.3.2. Workbook 的属性

Workbook 类提供了 90 多个属性，其中可能永远不会被开发者所使用，例如，AutoUpdateFrequency 属性返回共享工作簿的自动更新的分钟数；如果工作簿使用 1904 日期系统，Date1904 属性会返回 True 值；PasswordEncryptionAlgorithm 属性可以让您设置用于加密密码的确切算法，等等。

以下介绍了最常使用的 Workbook 属性：

**Name、FullName、Path 属性**

Name、FullName、Path（字符串，只读）：这些属性分别返回不同版本的工作簿名称。FullName 返回完整路径名称，包括工作簿文件名。Name 只是返回名称部分，而 Path 则只返回路径部分。例如以下代码获取了当前工作薄的名称等属性，运行结果见图 3-4，使用这些属性在组织大的程序，定位程序目录等方面具有非常重要的用途。

```
ActiveSheet.Range("A1").Value = ThisWorkbook.Name
ActiveSheet.Range("A2").Value = ThisWorkbook.Path
ActiveSheet.Range("A3").Value = ThisWorkbook.FullName
```

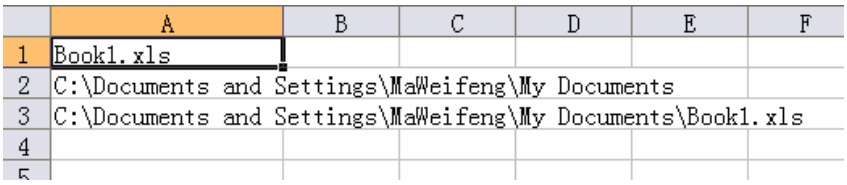



图 3-4 使用 Workbook 属性返回的工作薄名称等信息

**Password 属性**

Password 属性返回或设置密码，在打开指定的工作簿时必须提供该密码。String 类型，可读写。 例如，Microsoft Excel 打开名为 Password.xls 的工作簿，设置它的密码，然后关闭该工作簿。

```
Sub UsePassword()

    Dim wkbOne As Workbook
```

```
    Set wkbOne = Workbooks.Open("C:\Password.xls")

    wkbOne.Password = InputBox ("Enter Password")
    wkbOne.Close

End Sub
```

Password 属性可读取，但返回 ">>**"。

**ReadOnly 属性**

ReadOnly（布尔值，只读）：如果工作簿以只读的方式打开，则此属性返回 True 值。此时如果无法将数据保存到工作簿。

**Saved 属性**

Saved（布尔值）：用来获取或设置工作簿的保存状态。如果用户已经对工作簿的内容或结构进行了修改，则 Saved 属性就为 True。如果试图关闭工作簿或者退出 Excel，将会出现一个警报提示保存工作簿（除非已经将 Application.DisplayAlerts 属性设置成 False）。如果在代码中将 Saved 属性值设置成 False，Excel 就会认为工作簿已经保存，并且不会再次提醒保存。

同其他的 Office 应用程序一样，Excel 允许保存工作簿的同时保存文档属性（可以选择 *文件 － 属性* 来设置其属性）。也可以通过 Workbook 类的 BuiltInDocumentProperties 属性来使用内置属性，并通过 CustomDocumentProperties 属性来使用自定义属性。

**Sheets 和 Worksheets 属性**

Workbook 类提供了一个 Sheets（Worksheets）属性，它返回一个 Sheets 对象。这个对象包含 Sheet 对象集合，其中每个对象既可以是 Worksheet 对象，也可以是 Chart 对象。以下例子列出工作簿中的所有现有的表：

```
Private Sub ListSheets()
    Dim sh As Worksheet
    Dim rng As Range
    Dim i As Integer

    Set rng = ActiveSheet.Range("A5")
    For Each sh In ThisWorkbook.Sheets
```

```
        rng.Offset(i, 0).Value = sh.Name
        i = i + 1
    Next sh
End Sub
```

## 3.3.3. Sheets 集合

Sheets 集合包括以下成员。

**Visible 属性**

Visible 属性可以显示或隐藏一个现有的表。可将 Visibility 属性设置成 XlSheetVisibility 枚举值（XlSheetHidden、XlSheetVeryHidden、xlSheetVisible）中的一个值。使用 XlSheetHidden 可以让用户通过 Excel 界面隐藏表；使用 XlSheetVeryHidden，则要求运行代码来取消隐藏表，例如：

```
Workbooks(1).Sheets(1).Visible = xlSheetHidden
```

**Add 方法**

Add 方法将一个新表添加到工作簿中的表集合中，并且可以接受四个可选参数，这些参数可以指明表的位置、要添加的表数和表的类型（工作表、图表等）：

```
Dim sh As Worksheet
Set sh = ThisWorkbook.Sheets.Add()
```

**Copy 方法**

Copy 方法创建一个表的副本，并且将表插入到指定的位置。

**Delete 方法**

Delete 方法删除一个指定的表.

**FillAcrossSheets 方法**

FillAcrossSheets 方法将工作簿内一个表的范围中的数据复制到所有其他表。可以指定一个范围，以及是否要复制数据、进行格式化，或全部。

**Move 方法**

Move 方法和 Copy 方法很类似，只不过最终您得到的是表的一个实例。

**PrintOut 方法**

PrintOut 方法允许打印选择的对象（这个方法适用于多个不同的对象）。PrintOut 可以指定许多可选的参数，包括：要打印的页数（起始页和终止页）、副本数量、打印前是否进行预览、要使用的打印机的名称、是否打印到一个文件、是否进行逐份打印以及您要打印到的文件名。下面的例子使用默认的打印机打印指定的表、只打印第一页、打印两份副本，并且在打印前预览文档：

```
ThisWorkbook.Sheets(1).PrintOut _
    From:=1, To:=1, Copies:=2, Preview:=True
```

**PrintPreview 方法**

PrintPreview 方法允许您在打印预览窗口显示指定的对象，并且可以选择禁止更改页面布局：

**Select 方法**

Select 方法选择指定的对象，并且改变用户的选择（可使用 Activate 方法使对象获得焦点，而不需改变用户的选择。）您可以有选择地提供一个被当前选择取代的对象的引用。下面的代码片段选择第一个工作表：

```
ActiveWorkbook.Sheets(1).Select()
```

在这一部分中列出的许多方法也适用于其他的类。例如，PrintOut 方法是由以下类提供的：Chart、Charts、Range、Sheets、 Window、Workbook、Worksheet 和 Worksheets。这些方法的具体使用是相同的，只不过是作用在不同的对象上而已。Select 方法几乎适用于任何一种可选择的对象（并且这样的对象有很多）。

## 3.3.4. Workbook 的方法

Workbook 类提供了大量的方法，以下描述了一些最可能使用的方法：

**Activate 方法**

Activate 方法激活一个工作簿，以下代码激活并且选择工作簿中的第一个工作表：

```
ThisApplication.Workbooks(1).Activate
```

**Close 方法**

Close 方法关闭一个指定的工作簿，并且（可选）指定是否保存修改。如果工作簿从未保存过，则可以指定一个文件名。下面的代码片段关闭工作簿，并且不保存修改：

```
Workbooks(1).Close SaveChanges:=False
```

注意，打开和新建工作薄的方法（Add，Open）在 Workbooks 集合对象中，而 Workbook 对象只有关闭（Close）和保存（Save）的方法。

**Protect 和 Unprotect 方法**

Protect 和 Unprotect 方法可以设置保护一个工作簿，从而不能添加或者删除工作表，以及再次取消保护工作簿。可以指定一个密码（可选），并且指明是否保护工作簿的结构（这样用户就不能移动工作表）以及工作簿的窗口（可选）。保护工作簿用户仍可以编辑单元格。要想保护数据，则必须保护工作表。调用 Unprotect 方法（如果需要，还要传递一个密码）可以取消保护工作簿。

**Save 方法**

Save 方法保存工作簿。如果您还未保存过工作簿，则应该调用 SaveAs 方法，这样您可以指定一个路径（如果还未保存过工作簿，Excel 会将其保存在当前文件夹中，并以创建工作簿时所给的名称命名）。

**SaveAs 方法**

SaveAs 方法要比 Save 方法复杂的多。这个方法允许保存指定的工作簿，并且指定名称、文件格式、密码、访问模式和其他更多的选项（可选）。查看联机帮助可以获得所有选项列表。下面的代码片段将当前工作簿保存到一个指定位置，并且存成 XML 格式：

```
ActiveWorkbook.SaveAs "C:\MyWorkbook.xml", _
    FileFormat:=Excel.XlFileFormat.xlXMLSpreadsheet
```

由于保存成某些格式需要一些交互，因此在调用 SaveAs 方法之前可以将

Application.DisplayAlerts 属性设置成 False。例如，在将一个工作表保存成 XML 格式时，Excel 会提醒不能随工作簿保存 VBA 项目。如果将 DisplayAlerts 属性设置成 False，就不会出现这种警告。

**SaveCopyAs 方法**

SaveCopyAs 方法将工作簿的一个副本保存到文件中，但不会修改在内存中打开的工作簿。当您想创建工作簿的备份，同时不修改工作簿的位置时，这个方法非常有用：

## 3.3.5. Workbook 的事件

Workbook 提供了类似 Application 的事件，例如工作薄激活（Activate）、打开（Open）、关闭（BeforeClose，发生在关闭之前）、保存（BeforeSave）、打印（BeforePrint）、工作表改变（NewSheet）、工作表内容改变（SheetChange）、加载宏的加载和卸载（AddinInstall，AddinUninstall）等等。我们可以在 VBA IDE 中打开当前 Excel 工程的 ThisWorkbook 模块（代表了当前工作薄），从左侧选择 Workbook 对象，然后从右侧选择响应的事件，编写事件响应过程（见图 3-2）。

例如下例当指定工作簿作为加载宏安装时，本示例将一个控件添加到常用工具栏中。

```
Private Sub Workbook_AddinInstall()
    With Application.Commandbars("Standard").Controls.Add
        .Caption = "The AddIn's menu item"
        .OnAction = "'ThisAddin.xls'!Amacro"
    End With End Sub
End Sub
```

将 BeforeClose 事件的响应设置为保存工作簿的任何更改。

```
Private Sub Workbook_BeforeClose(Cancel as Boolean)
    If Me.Saved = False Then Me.Save
End Sub
```

如上例，在事件参数里有一个 Cancel 参数，可以通过设置其属性为 True 来取消其事件，比如设置以上 BeforeClose 事件的 Cancel 为 True，则 Excel 不再关闭本工作薄：

```
Private Sub Workbook_BeforeClose(Cancel as Boolean)
    Cancel = True
End Sub
```

如果在 BeforeSave 事件里设置 Cancel，Excel 将不保存此工作薄。

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI _
```

```
    As Boolean, Cancel as Boolean)
    a = MsgBox ("Do you want to save?", vbYesNo)
    If a = vbNo Then
        Cancel = True
    End If
End Sub
```

在其他对象的事件处理过程中的事件也具有同样的作用。

## 3.4. Worksheet 对象

Workbook 类代表了 Excel 的一个单一的工作簿，WorkSheet 类则代表了工作薄中的一个工作表，本部分将主要介绍 WorkSheet 类，包括最常使用的属性和方法。

Worksheet 类提供了大量的成员，但是其大多数的属性、方法和事件与 Application 和（或）Workbook 类提供的成员是相同的或相似的。需要说明，尽管 Excel 提供了一个 Sheets 集合作为 Workbook 对象的属性，但是在 Excel 中没有 Sheet 类。Sheets 集合的每个成员都是 Worksheet 或 Chart 对象。也许在 Office 内部实现时，Worksheet 和 Chart 类是继承自一个没有公开的 Sheet 类，但我们无从考证。

由于很多 Worksheet 的成员也包含在 Application 和 Workbook 对象中，以下将对 Worksheet 主要的属性、方法和事件做一简单介绍。

**Calculate 方法**

重新计算一个工作表，例如：

```
Worksheets(1).Calculate
```

如果一个工作薄有很多工作表，则应该尽量缩小计算范围。

**CheckSpelling 方法**

对一个工作表进行拼写检查，等同于在 Excel 界面下选择“*工具 － 拼写检查*”。

```
Worksheets("Sheet1").CheckSpelling
```

**Comments 属性**

返回当前选择的工作表内的所有注释的集合。例如返回注释的条数：

```
MsgBox Worksheets("sheet2").Comments.Count
```

对于特定单元格的注释，可以使用 Range 对象 Comments 属性。

**Delete 方法**

删除一个工作表，与选择“编辑 － 删除工作表”效果一致。

```
Worksheets("sheet1").Delete
```

**PrintOut 和 PrintPreview 方法**

这两个方法可以打印或者打印预览特定的工作表。

```
Worksheets("sheet2").PrintOut
Worksheets("sheet2").PrintPreview
```

**Protect、Unprotect 方法**

本方法可以保护一个工作表，效果等同于选择“*工具 － 保护 －保护工作表*”，可以为添加一个作为密码的参数。例如：

```
Worksheets("sheet2").Protect
Worksheets("sheet2").Protect ("123")

Worksheets("Sheet2").Unprotect ("123")
```

第一个例子没有密码，因此调用 UnProtect 方法或者从菜单解密时不需要密码，第二个例子的密码为“123”，第三个例子解密 Sheet2。

**Range 属性**

返回一个 Range 对象，该对象代表一个单元格或单元格区域。下一节要详细介绍 Range 对象。

**SaveAs 方法**

保存对不同的文件中的图表或工作表的更改。

```
Worksheets("sheet2").SaveAs ("MyFile")
```

**Select 方法**

选择一个工作表。例如：

```
Worksheets("sheet2").Select
```

**Visible 属性**

设置或显示一个工作薄，例如：

```
Worksheets("sheet2").Visible = False
```

**SelectionChange 事件**

当工作表上的选定区域发生改变时，将产生本事件。

下列滚动工作簿窗口，直至选定区域位于窗口的左上角。

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    With ActiveWindow
        .ScrollRow = Target.Row
        .ScrollColumn = Target.Column
    End With
End Sub
```

**Calculate 事件**

在对工作表进行重新计算之后产生此事件。

**Change 事件**

当用户更改工作表中的单元格，或外部链接引起单元格的更改时产生此事件。

重新计算引起的单元格更改不触发本事件。可使用 Calculate 事件俘获工作表重新计算操作。下例将更改的单元格的颜色设为蓝色。

```
Private Sub Worksheet_Change(ByVal Target as Range)
    Target.Font.ColorIndex = 5
End Sub
```

# 3.5. Range 对象

Range 对象是 Excel 应用程序中最经常使用的对象；在操作 Excel 内的任何区域之前，都需要将其表示为一个 Range 对象，然后使用该 Range 对象的方法和属性。基本上来说，一个 Range 对象代表一个单元格、一行、一列、包含一个或者更多单元块（可以是连续的单元格，也可以式不连续的单元格）的选定单元格，甚至是多个工作表上的一组单元格。对于使用过其他编程语言（例如 VB 或者 C）的读者，可以把 Range 看做一个加强的数组

或者 Grid 对象。也可以简单的说，Range 就是操作 Excel 内的具体内容（单元格）的对象。

## 3.5.1. 返回或获得 Range 对象

本部分将说明如何获得或返回 Range 对象。

**Range 属性**

可用 Range(arg)（其中 arg 为区域名称）来返回代表单个单元格或单元格区域的 Range 对象。下例将单元格 A1 中的值赋给单元格 A5。

```
Worksheets("Sheet1").Range("A5").Value = _
    Worksheets("Sheet1").Range("A1").Value
```

下例设置单元格区域 A1:H8 中每个单元格的公式，用随机数填充该区域。在没有对象识别符（句号左边的对象）的情况下，使用 Range 属性将返回活动表（ActiveSheet）上的一个区域。如果活动表不是工作表，则该方法无效。在没有明确的对象识别符的情况下，使用 Range 属性之前，可用 Activate 方法来激活工作表。

```
Worksheets("Sheet1").Activate
Range("A1:H8").Formula = "=Rand()"
```

下例清除区域 Criteria 中的内容。

```
Worksheets(1).Range("Criteria").ClearContents
```

其中 Criteria 为名称，在 Excel 中可以定义名称，代表单元格、单元格区域、公式或常量值的单词或字符串。名称更易于理解，例如，“产品”可以引用难于理解的区域“Sheet1!C20:C30”。如果用文本参数指定区域地址，必须以 A1 样式记号指定该地址（不能用 R1C1 样式记号）。有关单元格引用和名称，可以参考 Excel 帮助文档或有关书籍。

**Cells 属性**

可用 Cells(row, column)（其中 row 为行号，column 为列标）返回工作表或某个 Range 对象中的单个单元格。下例将单元格 A1 赋值为 24。

```
Worksheets(1).Cells(1, 1).Value = 24
```

下例设置单元格 A2 的公式。

```
ActiveSheet.Cells(2, 1).Formula = "=Sum(B1:B5)"
```

虽然也可用 Range("A1") 返回单元格 A1，但有时用 Cells 属性更为方便，因为可以使用变量来指定行列。下例在 Sheet1 上创建行号和列标。当工作表激活以后，使用 Cells

属性时不必明确声明工作表（它将返回活动工作表上的单元格）。

```
Sub SetUpTable()
    Dim i As Long
    Dim j As Long
    Worksheets("Sheet1").Activate
    For i = 1 To 5
        Cells(1, i + 1).Value = 1990 + i
    Next i
    For j = 1 To 4
        Cells(j + 1, 1).Value = "Q" & j
    Next j
End Sub
```

虽然可用 Visual Basic 字符串函数转换 A1 样式引用，但使用 Cells(1, 1) 记号更为简便（而且也是更好的编程习惯）。

可用 expression.Cells(row, column) 返回区域中的一部分，其中 expression 是返回 Range 对象的表达式，row 和 column 为相对于该区域左上角的偏移量。下例设置单元格 C5 中的公式。

```
Worksheets(1).Range("C5:C10").Cells(1, 1).Formula = _
     "=Rand()"
```

**Range 和 Cells**

可用 Range(cell1, cell2) 返回一个 Range 对象，其中 cell1 和 cell2 为指定起始和终止位置的 Range 对象。下例设置单元格区域 A1:J10 的边框线条的样式。

```
With Worksheets(1)
    .Range(.Cells(1, 1), _
        .Cells(10, 10)).Borders.LineStyle = xlThick
End With
```

注意每个 Cells 属性之前的句点。如果前导的 With 语句应用于该 Cells 属性，那么这些句点就是必需的。本示例中，句点指示单元格处于第一张工作表上。如果没有句点，Cells 属性将返回活动工作表上的单元格。

**Offset 属性**

可用 Offset(row, column)（其中 row 和 column 为行偏移量和列偏移量）返回相对于另一区域在指定偏移量处的区域。下例选定位于当前选定区域左上角单元格的向下三行且向右一列处的单元格。由于必须选定位于活动工作表上的单元格，因此必须先激活工作表。

```
Worksheets("Sheet1").Activate
  'Can't select unless the sheet is active
Selection.Offset(3, 1).Range("A1").Select
```

**Union 方法**

可用 **Union**(range1, range2, ...) 返回多块区域，即该区域由两个或多个连续的单元格区域所组成。下例创建由单元格区域 A1:B2 和 C3:D4 组合定义的对象，然后选定该定义区域。

```
Dim r1 As Range, r2 As Range, myMultiAreaRange As Range
Worksheets("sheet1").Activate
Set r1 = Range("A1:B2")
Set r2 = Range("C3:D4")
Set myMultiAreaRange = Union(r1, r2)
myMultiAreaRange.Select
```

使用包含若干块的选定区域时，Areas 属性很有用。它将一个多块选定区域分割为单个的 Range 对象，然后将这些对象作为一个集合返回。使用返回的集合的 Count 属性可检测一个选定区域是否包含多块区域，如下例所示。

```
Sub NoMultiAreaSelection()
    NumberOfSelectedAreas = Selection.Areas.Count
    If NumberOfSelectedAreas > 1 Then
        MsgBox "You cannot carry out this command " & _
            "on multi-area selections"
    End If
End Sub
```

## 3.5.2. Range 对象的常用属性和方法

以下将对 Range 对象的常用属性、方法做一简单介绍。

**Activate 方法**

Activate 方法激活单个或多个单元格，使其成为当前活动单元格，例如：

```
Worksheets("sheet1").Range("a1").Activate
```

**AddComment 方法**

为单元格增加注释，例如：

```
Worksheets("sheet1").Range("a1").AddComment ("MyComment")
```

如果给已经有注释的单元格增加注释，会导致一个运行时错误，给非单个的单元格增加注释，也会导致一个错误。可以使用 Comments 集合来修改已有注释。

**Address 方法**

可以使用 Address 方法获取一个 Range 的引用地址，例如“A1”，在实际编程中，Address 方法非常有用。例如：

```
MsgBox ActiveCell.Address
```

**Calculate 方法**

重新计算特定的单元格。类似于 Application 和 Worksheet 的同名方法，只不过其范围更小。

```
Worksheets("sheet2").Range("a3.d12").Calculate
```

**Cells 属性**

返回一个 Cells 集合对象，包含所有的单元格，可以使用：

```
Cells(行, 列)
```

返回 Range 中的特定单元格。也可以获取其他属性，例如获得单元格的个数：

```
MsgBox Worksheets("sheet2").Range("a3.d12").Cells.Count
```

**CheckSpelling 方法**

对单个单元格或者范围进行拼写检查。

```
Worksheets("sheet2").Range("a3.d12").CheckSpelling
```

**Clear 方法**

清除 Range 内的一切内容，包括注释和格式。

```
Worksheets("sheet2").Range("a3.d12").Clear
```

**ClearComments 方法**

清除注释，例如：

```
Worksheets("sheet2").Range("a3.d12").ClearComments
```

**ClearContents 方法**

清除单元格或者其集合的内容，不清除格式和注释。

```
Worksheets("sheet2").Range("a3.d12").ClearContents
```

**ClearFormats 方法**

清除 Range 内的格式。

**Column 和 Row 属性**

返回 Range 的第一列或行，例如：

```
MsgBox Worksheets("sheet2").Range("b3.d12").Column
```

返回 2,因为第一列是 B。

```
MsgBox Worksheets("sheet2").Range("b3.d12").Row
```

返回 3,因为第一行是 3。

**Columns 和 Rows 属性**

返回 Range 的列和行，通过其 Count 属性可以获取行数或列数。在使用 For 循环遍历时，就可以使用此属性，获得 Range 对象的大小，然后使用 Cells 属性，获取各个单元格的具体值，例如：

```
#001    Dim i As Long, j As Long
#002    Dim rng As Range
#003    Set rng = ActiveSheet.Range("C1:H26")
#004    With rng
#005        For i = 1 To .Columns.Count
#006            For j = 1 To .Rows.Count
#007                .Cells(j, i).Value = j & " " & i
#008            Next j
#009        Next i
#010    End With
```

上面的例子通过 Columns 和 Rows 属性获取了 Range 的大小，循环给“C1:H26”这个 Range 赋值。

**ColumnWidth 和 RowWidth 属性**

设置行或列的宽度，例如：

```
Worksheets("sheet2").Range("b3.d12").ColumnWidth = 4
Worksheets("sheet2").Range("b3.d12").RowHeight = 10
```

**Copy 和 PasteSpecial 方法**

与编辑菜单的拷贝相同，例如：

```
Worksheets("sheet2").Range("f19.g20").Copy
Worksheets("sheet2").Range("h19").PasteSpecial
```

PasteSpecial 允许设置 Paste 值还是格式。例如：

```
Worksheets("sheet2").Range("h19").PasteSpecial
Type:=xlPasteValues
```

**PrintOut 和 PrintPreview 方法**

打印一定范围的 Range 对象，而不是整个工作表。

**Select 方法**

选中特定的 Range，如：

```
Worksheets("sheet2").Range("f19.g20").Select
```

**Value 属性**

Range 的缺省属性，可以读写单元格的内容，例如：

```
MsgBox Worksheets("sheet2").Range("f26").Value
Worksheets("sheet2").Range("f26").Value = 10
```

Range 对象 Variant 类型，不仅可以表示单个单元格的值，也可以将整个 Range 作为一个数组返回或者设置。Range 对象读入数组可以使用以下方法：

```
vData = ActiveSheet.Range("A1:B10").Value
```

其中 vData 可以是定义好的数组，也可以是一个 Variant 变量。反过来，使用：

```
ActiveSheet.Range("D1:E10").Value = vData
```

就可以将数组 vData 的值赋给 Range，如果 Range 的范围较小，则自动截断。

# 4. 数据处理

## 4.1. 概述

Excel 获得了极其广泛的应用，一个重要原因就是 Excel 强大的数据处理能力。应用 Excel 的函数、排序、分类汇总、数据筛选等功能，可以完成从财务报表到资产管理，从学生成绩表到物质分析结果统计等各种复杂的数据处理工作。但是，在实际的工作中，数据处理的要求千变万化，也有很多使用 Excel 本身功能不能完成，或者很难完成的数据处理工作，这时候就需要使用 VBA 对 Excel 进行扩展。

另一方面，数据处理是一个非常广泛的概念，需要对数据处理作一基本概述都是不可能的。数据处理的核心在于数据的业务流程、数学运算和统计分析，因此，本章只是从 Excel 和 VBA 出发，对使用 Excel 和 VBA 进行数据处理的技术问题做一介绍。本书的数据处理也不是仅仅局限于数学数据的处理，例如方程求解、模型计算等，而是指一切的数据，包括文本、数字、各类数据文件等的各类处理工作，例如内容查找排序，数据的输入输出等等这样的任务。

可以进行数据处理的工具非常之多，对于数值型的数据处理，从一般的程序设计语言 C、Java 到科学运算编程语言 Fortran，从类似 Matlab、IDL 的各类专有语言，到各类其他的电子表格软件和形形色色的工具都可以方便的进行数据处理；对于文本数据和文件处理，例如 Perl、Python、VBScript 这些脚本语言由衷得天独厚的优势。那么使用 Excel 的优势何在呢？

相比其他语言和工具，Excel 的优势在于：

1. Excel 以电子表格为界面，适合数据的输入输出；
2. Excel 具有强大的图表功能，适合数据的可视化表示；
3. Excel 有大量的内置公式，包括了从统计分析到数学运算的方方面面，使用这些公式可以节约大量工作；
4. 可以使用 VBA 对 Excel 进行扩展；
5. VBA 使用简单，功能完备，非一般程序的脚本语言可比，可以完成复杂的数据处理流程；
6. VBA 有比较完备的文件、字符串处理能力；

7. VBA 通过设计用户窗体（Form），修改 Excel 界面的菜单、工具栏，可以完成交互性友好的数据处理程序；
8. 应用 VBA 可以控制 Excel 的全部对象和功能；
9. VBA 可以任意调用 COM 组件，扩展性好。

也许从单一方面，Excel 都不是最优秀的数据处理平台，但综合所有这些方面，Excel 无疑是一个优秀的数据处理平台。对于实际工作者，与其泛泛学习几个平台和语言，还不如深入学习 VBA 和 Excel，应用其来完成大部分的日常数据处理工作。

本章首先介绍使用 Excel 进行数据处理总的方式和流程，特别强调了基于表格计算的数据处理[13]和基于代码的数据处理的不同和优缺点，然后介绍了一些具体的数据处理技术，包括文件操作和数据操作，最后给出了一些实际应用的例子。本章第三节数据文件操作和第四节数据操作的内容，可以作为参考手册在需要的时候阅读。

## 4.2. Excel 数据处理的方式和流程

### 4.2.1. 方式和流程

数据处理可以是一个 Excel 系统或者应用的全部或者部分内容。应用 Excel 和 VBA 进行数据处理，其使用或者处理方式可以简单的划分为 2 大类，第一类可以称之为“表格驱动”的数据处理；第二类为“代码驱动”的数据处理。Excel VBA 数据处理程序的流程对于不同的具体应用，都有不同的流程，但总体还有一定的规律或者模式，都需要进行数据的输入输出，把处理过程划分为一些具体的步骤，然后逐次完成（图 4-1）。

[13] 可以认为基于表格计算的数据处理，及“表格驱动”的数据处理是 Excel，以及电子表格软件可以广泛应用于数据处理的一个重要原因，这种方式的程序设计大大降低了编程的难度，使没有任何程序设计训练的用户可以进行一些程序设计。

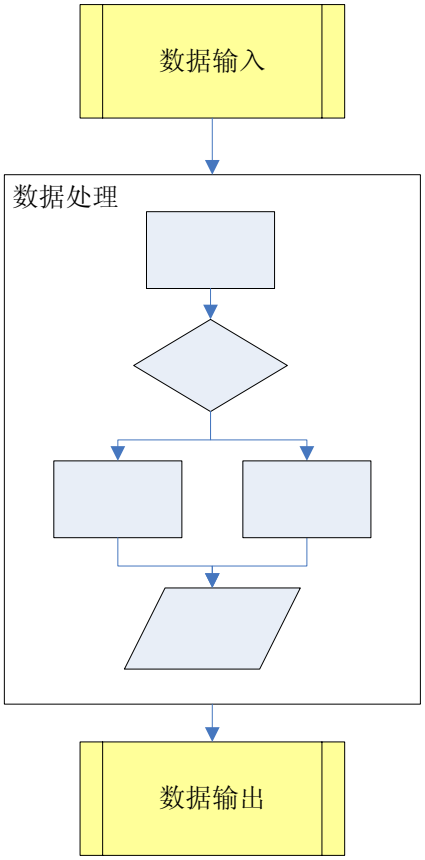



图 4-1 数据处理的一般流程

由图 4-1 可以得知，数据处理首先需要获取数据，这个数据可以是外部的文件，或者是 Excel 工作表中的全部或者部分数据。对于外部文件，可以使用 Excel 直接打开，也可以编程读取。如果文件只有一个或者很少，完全可以使用 Excel 打开，然后再进行处理；如果文件很多，需要逐个处理，比如批量更改多个文件的数据，或者从多个文件中获取数据，那么就可以使用编程的方法来获取数据。使用编程来打开获取数据，可以使用 VBA 处理文本或二进制文件的方式，也可以调用 Excel 对象打开并处理之。对于 Excel 工作表里的数据，则需要通过 Excel 的对象模型来处理，例如通过 Range 对象获取某个或多个单元格的内容。

获取了数据以后，则需要编写具体的数据处理程序，这部分主要的工作是需要通过某种方式来实现需要的数据处理的算法或者模型。具体来说，使用 Excel 完成这部分工作可以使用“表格驱动”的数据处理或者“代码驱动”的数据处理，对于后者，还可以使用面向对象的编程方法或者基于过程的编程方法。数据处理针对应用的差别即在于数据处理的具体流程的差别，也就是这部分。例如个人增值税计算需要判断收入的多少，然后根据收入应用不同的公式来计算；线性方程求解需要通过矩阵运算来获得结果解向量。对于不同

的应用，其复杂程度也不同，有些应用可能只是一个公式的计算，有些应用则需要数千行甚至更多的代码。

需要说明的是，计算机数据处理的很多工作复杂性在于数据的前期处理和后期处理，例如读入的数据文件的格式各不相同，如何转换为可以计算的变量，如何从中提取需要的数据；处理完成后输出为需要的格式，进行可视化的表达，而应用 Excel 的优势正在于 Excel 可以帮助你完成很多此类琐碎的工作，而使你可以专注于数据处理的具体求解。

最后则需要输出结果，对于表格，结果保存在一个或数个单元格；对于代码，可以使用返回值、对话框、写入文件或者单元格的方式来返回结果。

### 4.2.2. “表格驱动”的数据处理

对于第一类，“表格驱动”的数据处理，其含义指大部分或者全部的数据处理工作是通过电子表格以及 Excel 的内置函数来完成的，其运算由 Excel 的“重新计算”驱动。例如求解一个二次方程，使用“表格驱动”的数据处理方式可以如下进行（图 4-2）：

1. 首先选择二次方程参数（a, b, c）输入的位置（即单元格）；
2. 在另一个单元格内计算二次方程的 Δ 的平方根，本例中可以输入以下公式[14]：

   *(D4^2-4*D3*D5)^0.5*

   其中 D3，D4，D5 分别代表二次方程的 a，b，c；
3. 在结果单元格输入最后的求解公式：

   *(-D4+C7)/ (2*D3)*

   和

   *(-D4-C7)/ (2*D3)*


图 4-2 “表格驱动”的二次方程求解

[14] 为了例子的简单，我们没有考虑 Δ 为负的情况。

由此，我们可以对“表格驱动”的数据处理总结如下：“表格驱动”的数据处理其主要机制是使用 Excel 的函数，包括使用 VBA 自定义的函数，和 Excel 电子表格来进行，使用电子表格保存计算的临时值，通过一个或者数个临时值，最后计算得到需要的结果。

“表格驱动”的数据处理的具体方法和流程可以划分为：

1. 将数据处理划分为一定的步骤，每一步都可以通过一个函数和公式的计算得到下一步需要的结果，逐次求解，最后得到需要的计算结果，其核心在于数据处理步骤的划分；
2. 确定输入输出以及每一步计算的临时变量需要放置的单元格，应用前一步骤得到的计算结果，在单元格中输入可以得到此步骤结果的具体公式，依次完成所有步骤；
3. 在输入区域输入数据则得到结果，如果 Excel 的“重新计算”没有打开，可以通过“*工具 － 选项 － 重新计算*”打开之。

“表格驱动”的数据处理的优点有：

1. 使用简单，无需编程即可实现，是广大非程序员进行数据处理的首选方式；
2. 计算速度快，因为 Excel 的重新计算引擎以及内置函数计算速度较好，因此通过此种方式进行运算速度通常比使用 VBA 代码快一些；
3. 输入输出操作简单，输入输出的位置已经确定，可以在公式或函数（包括自定义函数）中直接输入，无需进行位置判断，单元格内容获取和数据类型判断等操作；
4. 结果可视化表达简单，例如报表之类结果可以在编程阶段就直接设计好；
5. 调试修改方便，可以随时得到运算结果，查看每一步骤的运算结果。

“表格驱动”的数据处理的缺点有：

1. 对于分支判断情况处理困难，对所有分支判断需要使用 IF 函数，比较烦琐和易出错，如果判断条件多，输入麻烦，工作量大；
2. 无法直接处理循环运算；
3. 由于以上 2 条，导致“表格驱动”的数据处理难以或者无法完成复杂的数据处理工作；
4. 很难或者较难处理输入参数个数不确定的情况下的数据处理；
5. 异常和错误处理困难。

因此，在实践工作中，需要根据情况选用这种处理方式，很多情况下可以使用表格驱

动的数据处理作为数据处理的主流程或者格架，配合 VBA 代码，来完成复杂的数据处理工作。“表格驱动”的数据处理和基于数据表的用户界面（见界面设计一节）常常一起出现，互为依托。

所得税计算，银行利率，工资表，简单的收支计算这样的小程序都很适合使用“表格驱动”的数据处理方式来完成。

很多小公司的财务、人事管理都在使用 Excel，如果加入一些计算和处理的自动化功能（很多已经在做），应该可以减少很多工作量。

本书主要介绍基于 VBA 代码的数据处理。

### 4.2.3. 基于“过程”代码的数据处理

基于代码的数据处理指我们通过 VBA 代码来完成数据处理工作，例如对于以上求解二次方程根的例子，使用代码来处理可以使用如下代码：

```
#001 Private m_a As Double
#002 Private m_b As Double
#003 Private m_c As Double
#004 Private m_d As Double
#005
#006 Public Sub ParameterInput(a As Double, _
       b As Double, c As Double)
#007     m_a = a
#008     m_b = b
#009     m_c = c
#010 End Sub
#011
#012 Public Sub GetDelta()
#013     m_d = (m_b ^ 2 - 4 * m_a * m_c) ^ 0.5
#014 End Sub
#015
#016 Public Function GetX1() As Double
#017     GetX1 = (-m_b + m_d) / (2 * m_a)
#018 End Function
#019
#020 Public Function GetX2() As Double
#021     GetX2 = (-m_b - m_d) / (2 * m_a)
#022 End Function
```

以上代码前 4 行定义了需要的临时变量，包括二次方程“$ax^2+bx+c=0$”的参数 a、b、c，以及临时变量Δ的平方根。过程 ParameterInput 则输入参数 a、b、c，保存其值在变量 m_a、m_b、m_c 中。过程 GetDelta 计算Δ的平方根。最后通过 GetX1 和 GetX2 计算返回二次方程的结果。程序没有考虑出错和意外情况，如Δ为虚数的情况。

对于此代码，可以通过以下代码使用获取计算结果。结果与上面使用表格计算获得的结果一致。

```
ParameterInput Range("D3"), Range("D4"), Range("D5")
GetDelta
MsgBox GetX1
MsgBox GetX2
```

代码首先输入参数，然后计算Δ，最后输出结果。过程不能有所差错。

由以上例子，我们可以看到，使用 VBA 代码进行数据处理较“表格驱动”的数据处理复杂一些，需要通过代码完成数据的输入，计算和输出。

一般来说，基于 VBA 代码的数据处理方法为：

1. 定义必要的模块级变量，存储全局各个函数都需要使用的参数；
2. 将数据处理划分为一定的步骤，该步骤一般要包括数据输入，一步或数步的计算，结果输出。每个步骤可以定义为一个函数或过程，将计算结果保存于模块级变量；
3. 使用此数据处理代码也需要根据代码本身，进行数据输入、处理、输出，不能颠倒次序。
4. 如果有几组数据需要计算，则需要特别小心每次的数据输入输出，计算顺序。

### 4.2.4. 基于“面向对象”代码的数据处理

通过 VBA 代码来进行数据处理，其代码不仅可以保存在模块中，也可以保存在类模块中，这样，该数据处理方法就变成了一个数据处理的类，使用该类的实例就可以进行具体的数据处理工作。

例如对于以上的二次方程求解，使用类模块可以这样来完成。新建一个类模块，命名为 Equation，输入如下代码：

```
#001 Private m_a As Double
#002 Private m_b As Double
#003 Private m_c As Double
#004 Private m_d As Double
#005 Private m_x1 As Double
```

```
#006 Private m_x2 As Double
#007
#008 Public Property Let A(value As Double)
#009    m_a = value
#010 End Property
#011
#012 Public Property Let B(value As Double)
#013    m_b = value
#014 End Property
#015
#016 Public Property Let C(value As Double)
#017    m_c = value
#018 End Property
#019
#020 Public Property Get X1() As Double
#021    X1 = m_x1
#022 End Property
#023
#024 Public Property Get X2() As Double
#025    X2 = m_x2
#026 End Property
#027
#028 Public Sub Calculate()
#029    m_d = (m_b ^ 2 - 4 * m_a * m_c)
#030
#031    If m_d < 0 Then Exit Sub
#032
#033    m_d = m_d ^ 0.5
#034
#035    m_x1 = (-m_b + m_d) / (2 * m_a)
#036    m_x2 = (-m_b - m_d) / (2 * m_a)
#037 End Sub
#038
#039 Public Function IsHaveRoot() As Boolean
#040    If m_d < 0 Then
#041       IsHaveRoot = False
#042    Else
#043       IsHaveRoot = True
#044    End If
#045 End Function
```

以上代码和前边模块代码没有大的区别，差别在于使用类模块，可以把方程求解封装为一个对象，通过对象的属性和方法来完成方程的求解。类模块以及面向对象的编程我们已经做了介绍。这段代码 1-6 行定义了需要的变量，各个属性（Property）可以设置参数和

返回最终的值，方法 Calculate 进行具体计算，IsHaveRoot 判断是否有根。这个类模块的使用可以使用如下代码。

```
Dim Eq As New Equation
Eq.A = Range("D3")
Eq.B = Range("D4")
Eq.C = Range("D5")
Eq.Calculate
If Eq.IsHaveRoot Then
   MsgBox "x = " & Eq.X1 & " , " & Eq.X2
Else
   MsgBox "No Root!"
End If
```

由以上代码可以看出使用面向对象的方法，数据处理的使用方式有了较大的变化，方程求解首先需要定义一个方程式对象，然后对方程式的属性（相当于方程的参数）进行设置，使用其计算方法，最后得到结果。整体概念和使用方便程度都优于使用一般的模块（基于过程的方法）。对于数个方程，可以新建数个 Equation 对象，分别求解。

一般来说，如果一个数据处理流程比较复杂，使用面向对象的方法更容易设计，也更容易使用。基于面向对象的方法来设计数据处理的方法总结如下：

1. 定义必要的私有变量，存储数据处理需要使用的参数；
2. 如果有必要，在 Class_Initialize 过程中进行必要的初始化；
3. 设计类的接口，包括公有（Public）属性和方法，通过属性来完成输入和输出，通过一个或数个方法来进行运算和结果判断；
4. 将数据处理划分为一定的步骤，每个步骤可以作为一个私有方法，供公有属性或方法调用；
5. 使用此类模块需要新建一个对象，设置属性，调用方法，最后获得结果。

对于复杂的数据处理，可能需要设计多个类协同工作，甚至与其他程序协同工作。例如通过调用已有的 COM 组件来完成一些任务。

## 4.3. 操作数据文件

在 Excel 中进行数据处理的时候，不仅会使用到当前 Excel 文件的数据，也可能需要操作其他数据文件，例如读取其他数据文件中的数据，循环处理分布在多个文件中的数据，将处理结果写入一定格式的文件供其他程序使用，等等。这些文件也许存储于 Excel 格式

的电子表格中，也许存储于其他文件格式，或者存储于数据库中，本节将介绍如何操作 Excel 电子表格文件和一般格式的数据文件，对数据库的操作请参考其他话题一章中 Excel 数据导入导出一节。在 Excel 中操作数据文件的方法有以下几种：

1. 使用 Excel 对象来打开、读写、保存文件；
2. 使用 VBA 的文件处理语句来处理文件；
3. 使用 FileSystemObject 对象处理文件。

以下章节将分别介绍这几种文件的处理方式的方法和其优缺点，适用场合和地点。

## 4.3.1. 使用 Excel 对象操作数据文件

使用 Excel 对象可以打开、读取、保存 Excel 数据格式和文本格式的数据文件。

Excel 的 Application 对象代表了 Excel 应用程序，而其属性 Workbooks 可返回一个 Workbooks 集合，集合包含了 Excel 程序中所有打开的工作薄（Workbook 对象）。关于 Workbooks 集合和 Workbook 对象的使用方法请参考 Excel 对象模型一章，这里只介绍其文件操作的有关功能。我们可以使用 Workbooks 集合的 Open 或 OpenText 方法打开已有工作薄或文本文件；使用 Workbook 对象的 Save 和 SaveAs 方法保存工作薄；使用 Workbook 对象的 Close 方法关闭工作薄；或者使用 Workbooks 集合的 Close 方法关闭所有工作薄。需要注意的是，打开操作使用的是 Workbooks 集合对象，关闭和保存使用的是 Workbook 对象。

### 4.3.1.1. 打开 Excel 文件

可以使用 Workbooks.Open 方法打开一个 Excel 工作簿。其完整语法如下：

```
Workbooks.Open(FileName, …… )
```

其中的 FileName 表示要打开的工作簿的文件名，如果没有指定路径，则代表当前路径。另外，可以选择另外 14 个可选参数表示是否只读打开，指定工作薄密码，更新工作薄等。例如可以使用以下方法打开 Analysis.xls 工作簿，然后运行 Auto_Open 宏。

```
Workbooks.Open "ANALYSIS.XLS"
ActiveWorkbook.RunAutoMacros xlAutoOpen
```

使用 Open 方法也可以打开文本文件，但建议使用 OpenText 方法。

如果不是一个简单的打开操作，需要进行很多选项，那么可以先录制一个宏，然后修

改其代码来完成。

#### 4.3.1.2. 打开文本文件

打开文本文件使用 Workbooks 集合对象的 OpenText 方法。此方法载入一个文本文件，并将其作为包含单个工作表的工作簿进行分列处理，然后在此工作表中放入经过分列处理的文本文件数据。例如，以下代码打开 Data.txt 文件并将制表符作为分隔符对此文件进行分列处理，转换成为工作表。

```
Workbooks.OpenText filename:="DATA.TXT", _
    dataType:=xlDelimited, tab:=True
```

Workbooks.OpenText 的完整语法如下：

```
Workbooks.OpenText(FileName, Origin, StartRow, DataType,
TextQualifier, ConsecutiveDelimiter, Tab, Semicolon, Comma,
Space, Other, OtherChar, FieldInfo, TextVisualLayout,
DecimalSeparator, ThousandsSeparator, TrailingMinusNumbers,
Local)
```

其参数含义为：

FileName，String 类型，必需。指定要载入并作分列处理的文件名。

Origin，Variant 类型，可选。指定文本文件来源。可为以下 XlPlatform 常量之一：xlMacintosh、xlWindows 或 xlMSDOS。此外，它还可以是代表所需代码页的代码页编号的整数。例如，“1256”说明源文本文件的编码是阿拉伯语 (Windows)。如果省略该参数，则此方法就会使用“文本导入向导”中“文件原始格式”选项的当前设置。

StartRow，Variant 类型，可选。作分列处理的起始行号。默认值为 1。

DataType，Variant 类型，可选。在文件中指定数据的列格式。可为以下 XlTextParsingType 常量之一：xlDelimited 或 xlFixedWidth。如果未指定该参数，则 Microsoft Excel 将在打开此文件时确定列格式。

TextQualifier，XlTextQualifier 类型，可选。指定文本识别符。XlTextQualifier 可为以下 XlTextQualifier 常量之一：xlTextQualifierDoubleQuote（默认值）；xlTextQualifierNone；xlTextQualifierSingleQuote。

ConsecutiveDelimiter，Variant 类型，可选。如果该值为 True，则将连续的分隔符号作为一个分隔符号处理。默认值为 False。

Tab，Variant 类型，可选。如果该值为 True，则将分隔符设为制表符（DataType 必

须设为 xlDelimited）。默认值为 False。

Semicolon，Variant 类型，可选。如果该值为 True，则将分隔符设为分号（DataType 必须设为 xlDelimited）。默认值为 False。

Comma，Variant 类型，可选。如果该值为 True，则将分隔符设为逗号（DataType 必须设为 xlDelimited）。默认值为 False。

Space，Variant 类型，可选。如果该值为 True，则分隔符设为空格（DataType 必须设为 xlDelimited）。默认值为 False。

Other，Variant 类型，可选。如果该值为 True，则将分隔符设为由 OtherChar 参数指定的字符（DataType 必须设为 xlDelimited）。默认值为 False。

OtherChar，Variant 类型，可选（如果 Other 为 True，则必需）。当 Other 为 True 时，指定分隔字符。如果指定了多个字符，则将字符串中的第一个字符作为分隔符，并忽略其余的字符。

FieldInfo，xlColumnDataType 类型，可选。包含各数据列分析信息的数组。对本参数的解释取决于 DataType 值。如果此数据由分隔符分隔，本参数为由两元素数组组成的数组，其中每个两元素数组指定一个特定列的转换选项。第一个元素为列标（从 1 开始），第二个元素是 XlColumnDataType 常量之一，用以指定如何分析该列。

其中 XlColumnDataType 可为以下 XlColumnDataType 常量之一：

- xlGeneralFormat 常规
- xlTextFormat 文本
- xlMDYFormat MDY 日期
- xlDMYFormat DMY 日期
- xlYMDFormat YMD 日期
- xlMYDFormat MYD 日期
- xlDYMFormat DYM 日期
- xlYDMFormat YDM 日期
- xlEMDFormat EMD 日期
- xlSkipColumn 忽略列

列识别符可为任意顺序。输入数据中如果某列没有列识别符，则用常规设置对该列进行分列处理。对于该参数，需要注意的是：

- 在指定要跳过的列时，必须明确说明其余各列的类型，否则数据不能正确拆分。

- 如果数据中包含可识别的日期，则工作表中单元格的格式就会设置为“日期”，即便为该列设置的格式为“常规”。此外，如果您为某一列指定了以上日期格式中的一种格式，而数据中并不包含可识别的日期，那么工作表中单元格的格式为“常规”。

本示例将第三列作为 MDY（例如，01/10/1970) 处理，第一列作为文本处理，源数据中其他列以“常规”设置进行分列处理。

```
Array(Array(3, 3), Array(1, 2))
```

如果源数据为定宽列，则每个两元数组的第一个元素指定起始元素在列中的位置，（用整数表示，第一个字符为 0 (零)），第二个元素用 0 到 9 的数字指定分列选项，如前表所示。

TextVisualLayout，Variant 类型，可选。文字的可视布局。

DecimalSeparator，Variant 类型，可选。表示在识别数字时，Microsoft Excel 使用的小数位分隔符。默认设置为系统设置。

ThousandsSeparator，Variant 类型，可选。 表示在识别数字时，Excel 使用的千位分隔符。默认设置为系统设置。

表 4-1 显示了使用不同的导入设置向 Excel 中导入文本时的结果。数字结果显示在最右边的列中。

表 4-1 使用不同的导入设置向 Excel 中导入文本时的结果

| 系统小数位分隔符 | 系统千位分隔符 | 小数位分隔符值 | 千位分隔符值 | 导入的文本 | 单元格的值（数据类型） |
|---|---|---|---|---|---|
| 句点 | 逗号 | 逗号 | 句点 | 123,123.45 | 123,123.45（数字） |
| 句点 | 逗号 | 逗号 | 逗号 | 123,123.45 | 123,123.45（文本） |
| 逗号 | 句点 | 句点 | 逗号 | 123,123.45 | 123,123.45（数字） |
| 句点 | 逗号 | 句点 | 逗号 | 123,123.45 | 123 123.45（文本） |
| 句点 | 逗号 | 句点 | 空格 | 123,123.45 | 123,123.45（数字） |

TrailingMinusNumbers，Variant 类型，可选。

Local，Variant 类型，可选。

以上是参数介绍，在实际的编程中，一般无需对这些复杂的参数进行处理，你可以使用 Excel 的文件打开功能，当打开文本文件时，会自动打开文本导入向导，通过可视化设置，从而以一定的格式打开需要的文本。通过对这一过程录制一个宏即可完成一个打开文

本文件的代码。

绝大多数的 Excel 操作，都可以录制为宏，而录制的宏可以作为开发的基础来使用，这点也是 Excel VBA 开发的一个特点。

打开文本文件的代码录制过程为：

1. 选择 *工具 － 宏 － 录制新宏*;
2. 在录制新宏对话框中选择将宏保存在当前工作薄；单击确定开始；
3. 选择 *文件 － 打开*,然后选择文本文件,在弹出的文本导入向导中设置导入规则；
4. 文本打开后，停止录制；
5. 然后就可以在 VBA IDE 的代码窗口中找到新录制的宏。

例如以下代码即是一个使用宏录制后的打开文本文件的代码，然后将文件名替换为 strOldName，并重新编排格式后的代码：

```
Workbooks.OpenText Filename:= _
       strOldName, Origin:=936, _
       StartRow:=1, DataType:=xlFixedWidth, _
       FieldInfo:=Array(0, 1), _
       TrailingMinusNumbers:=True
```

文本文件的打开是使用 Excel 进行数据处理过程中经常需要做的工作，而使用 Excel 可以大大简化文件打开和导入的难度。

#### 4.3.1.3. 打开其他文件

使用 Excel 对象模型还可以打开 XML 文件和一些文件数据库文件（例如 Access）数据库，对于 XML 文件，需要 Excel 2003 以上版本，例如以下代码打开了 XML 数据文件“customers.xml”并在 XML 列表中显示了此文件的内容。：

```
Sub UseOpenXML()
    Application.Workbooks.OpenXML _
       Filename:="customers.xml", _
       LoadOption:=xlXmlLoadImportToList
End Sub
```

具体的使用方法和参数请参考帮助文档。

以下代码打开了“northwind.mdb”文件。本示例假定“northwind.mdb file”存在于 C 盘

上。

```
Sub UseOpenDatabase()
    'Open the Northwind database.
    Workbooks.OpenDatabase _
        FileName:="C:\northwind.mdb"
End Sub
```

有关 XML 文件和 Access 文件的操作，请参考有关文档。数据库的操作请参考本书其他话题一章。

### 4.3.1.4. 保存文件

文件的保存使用 Workbook 的 Save 或者 SaveAs 方法。Save 方法为简单的保存，SaveAs 方法则为第一次保存或另存文件时使用。

**Save 方法**

Save 方法保存指定工作簿所做的更改。其语法为：

```
Workbook.Save
```

若要将一个工作簿标记为已保存，而不真正写入磁盘，请将该工作簿的 Saved 属性设为 True。如果是第一次保存工作簿，请使用 SaveAs 方法为该文件指定文件名。

以下代码保存当前活动工作簿。

```
ActiveWorkbook.Save
```

以下代码保存所有打开的工作簿，然后关闭 Microsoft Excel。

```
For Each w In Application.Workbooks
    w.Save
Next w
Application.Quit
```

**SaveAs 方法**

SaveAs 方法保存对不同文件中的工作表的更改。其语法为：

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

其主要参数说明如下：

Filename，Variant 类型，可选。该字符串表示要保存的文件名。可包含完整路径。如果不指定路径，Microsoft Excel 将文件保存到当前文件夹中。

FileFormat，Variant 类型，可选。保存文件时使用的文件格式。要得到有效选项的列表，请参阅 FileFormat 属性。对于已有文件，其默认格式是上次指定的文件格式；对于新文件，默认格式为当前使用的 Excel 版本格式。

Password，Variant 类型，可选。它是一个区分大小写的字符串（最长不超过 15 个字符），用于指定文件的保护密码。

以下代码新建一个工作簿，提示用户输入文件名，然后保存该工作簿。

```
Set NewBook = Workbooks.Add
Do
    fName = Application.GetSaveAsFilename
Loop Until fName <> False
NewBook.SaveAs Filename:=fName
```

以上代码的 GetSaveAsFilename 方法为显示标准的“另存为”对话框，获取用户文件名，但并不真正保存任何文件。然后使用代码保存文件。

另外，我们还可以使用 SaveAsXMLData 方法将文件保存为 XML 数据文件。但此方法只适用于 Office 2003 以上版本。

这里基本没有涉及 Excel 对 XML 处理的内容，因为 XML 处理不是简单的保存和读取，从 Office XP 开始，XML 已经成为 Office 的一个重要组成部分和开发方式，因为篇幅关系，本书将不涉及这些内容。

### 4.3.1.5. 关闭文件

关闭文件可以使用 Workbooks 集合对象或者 Workbook 对象的 Close 方法，前者关闭所有打开的工作薄，后者关闭特定的工作薄。

例如以下代码关闭“Book1.xls”，并放弃所有对此工作簿的更改。

```
Workbooks("Book1.xls").Close SaveChanges:=False
```

我们可以使用 SaveChanges 参数表示是否保存更改，对于很多不需要更改的操作，则可以设置为 False，以避免弹出保存更改提示对话框。

以下代码关闭所有打开的工作簿。如果某个打开的工作簿有改变，Microsoft Excel 将

显示询问是否保存更改的对话框和相应提示。

```
Workbooks.Close
```

#### 4.3.1.6. 总结

一般来说，使用 Excel 对象进行文件操作的情况和场合是操作的文件是需要进行处理的“数据文件”，而不是配置文件或者其他文件。对于 Excel 格式，使用 Excel 对象模型是唯一的选择，对于文本格式的数据文件，如果需要进行复杂的格式设置，然后导入为表格格式，那么，使用 Excel 对象模型也是合适的。

### 4.3.2. 使用 VBA 语句操作文件

在 Excel 下如果没有必要，尽量不要使用 VBA 语句操作文件。对于一般的文本格式和 Excel 格式的数据文件，最好使用 Excel 对象来操作；对于文件、目录操作，文本文件，微软建议使用 FileSystemObject 对象来操作，因为 FileSystemObject 提供了面向对象的文件操作方式；但由于 FileSystemObject 对象无法读写二进制格式的文件，因此对于二进制格式的文件还需要使用 VBA 语句来操作。

使用 VBA 语句进行文件操作主要是通过 Open，Close，Get，Input，Write，Print 等语句来进行操作，以下将按照操作目的逐一进行介绍。

#### 4.3.2.1. 处理文件

一般来说，可以使用 Kill 命令删除文件，使用 Name 命令重命名或者移动文件，使用 FileCopy 命令来拷贝文件。

例如以下代码：

```
' 文件操作一定要注意错误处理.
' 所有以下操作会在文件打开的情况下失败.
On Error Resume Next
' 重命名文件或者移动文件
' 重命名
Name "c:\vb6\TempData.tmp" As "c:\vb6\TempData.$$$"
' 移动文件
Name "c:\vb6\TempData.$$$" As "d:\VS98\Temporary.Dat"
' 拷贝文件
FileCopy "d:\VS98\Temporary.Dat", "d:\temporary.$$$"
```

```
' 删除文件
Kill "d:\temporary.*"
```

可以使用 GetAttr 函数来得到文件的属性，使用 SetAttr 命令设置文件的属性。例如以下代码获得文件的属性。

```
Function GetAttrDescr(filename As String) As String
    Dim result As String, attr As Long
    attr = GetAttr(filename)
    ' GetAttr also works with directories.
    If attr And vbDirectory Then result = result & " Directory"
    If attr And vbReadOnly Then result = result & " ReadOnly"
    If attr And vbHidden Then result = result & " Hidden"
    If attr And vbSystem Then result = result & " System"
    If attr And vbArchive Then result = result & " Archive"
    ' Discard the first (extra) space.
    GetAttrDescr = Mid$(result, 2)
End Function
```

同样，可以使用 SetAttr 命令来改变文件（包括目录）的属性，例如：

```
' 设置文件为存档和只读
filename = "d:\VS98\Temporary.Dat"
SetAttr filename, vbArchive + vbReadOnly
' 改变文件是否隐藏
SetAttr filename, GetAttr(filename) Xor vbHidden
```

我们可以使用 FileLen 和 FileDateTime 函数获取文件的大小和时间而不需要打开文件。例如以下代码：

```
Debug.Print FileLen("d:\VS98\Temporary.Dat")
Debug.Print FileDateTime("d:\VS98\Temporary.Dat")
```

当调用 FileLen 函数时，如果所指定的文件已经打开，则返回的值是这个文件在打开前的大小。

### 4.3.2.2. 处理目录

我们可以使用 CurDir 函数返回当前的路径。其参数 drive 是一个字符串表达式，它指定一个存在的驱动器。如果没有指定驱动器，或 drive 是零长度字符串 ("")，则 CurDir 会返回当前驱动器的路径。例如：

```
Debug.Print CurDir
Debug.Print CurDir("c")
```

其输出为：

D:\temp\office\

C:\Documents and Settings\XXX\My Documents

我们可以使用 ChDrive 和 ChDir 命令来改变当前的驱动器和目录。例如：

```
' 使“C:\Windows”成为当前目录
ChDrive "C:"
ChDir "C:\Windows"
```

可以使用 MkDir 和 RmDir 来创建和删除目录。也可以使用 Name 命令来重命名目录，但不可以移动之。

以下代码使用 MkDir 语句来创建目录或文件夹。如果没有指定驱动器，新目录或文件夹将会建在当前驱动器中。

```
MkDir "MYDIR"    ' 建立新的目录或文件夹。
```

以下代码使用 RmDir 语句删除已存在的目录或文件夹。

```
' 假设 MYDIR 为一空的目录或文件夹。
RmDir "MYDIR"    ' 将 MYDIR 删除。
```

需要注意，如果想要使用 RmDir 来删除一个含有文件的目录或文件夹，则会发生错误。在试图删除目录或文件夹之前，先使用 Kill 语句来删除所有文件。

Dir 函数返回一个 String，用以表示一个文件名、目录名或文件夹名称，它必须与指定的模式或文件属性、或磁盘卷标相匹配。其语法为：

```
Dir[(pathname[, attributes])]
```

在第一次调用 Dir 函数时，必须指定 pathname，否则会产生错误。如果也指定了文件属性，那么就必须包括 pathname。

Dir 会返回匹配 pathname 的第一个文件名。若想得到其它匹配 pathname 的文件名，再一次调用 Dir，且不要使用参数。如果已没有合乎条件的文件，则 Dir 会返回一个零长度字符串 ("")。一旦返回值为零长度字符串，并要再次调用 Dir 时，就必须指定 pathname，否则会产生错误。不必访问到所有匹配当前 pathname 的文件名，就可以改变到一个新的 pathname 上。但是，不能以递归方式来调用 Dir 函数。以 vbDirectory 属性来调用 Dir 不能连续地返回子目录。

例如以下代码可以循环目录和子目录，查找需要的文件。

```
Sub FindFile(cPath As String, Optional strFile As String = "")

On Error Resume Next

    'bStop 为全局变量，用于中途结束递归。
```

```
    If bStop Then
        Exit Sub
    End If

    Dim s As String, sDir() As String
    Dim I As Long, d As Long

    '可以在此加入状态指示的指令，例如进度或者正在查找的文件
    '例如：Me.lblInfo = "正在查找： " & cPath
    DoEvents

    '查找目录下的文件
    If Right(cPath, 1) <> "\" Then
        cPath = cPath & "\"
    End If
    s = Dir(cPath & strFile, vbArchive Or _
            vbNormal Or vbReadOnly Or vbHidden Or _
            vbSystem Or vbDirectory)

    Do While s <> ""
        Debug.Print cPath & s
        s = Dir()
    Loop

    '查找目录下的子目录
    s = Dir(cPath, vbArchive Or _
            vbNormal Or vbReadOnly Or vbHidden Or _
            vbSystem Or vbDirectory)

    Do While s <> ""
        If s <> "." And s <> ".." Then
            If (GetAttr(cPath & s) And vbDirectory) = _
                    vbDirectory Then
                d = d + 1
                ReDim Preserve sDir(d)
                sDir(d) = cPath & s
            End If
        End If
        s = Dir()
    Loop

    '开始递归
    For I = 1 To d
        FindFile sDir(I) & "\"
```

```
    Next

End Sub
```

读者可以使用此函数来完成一些查找文件的任务。

由于 VBA 和 VB 没有判断文件是否存在的直接办法，Dir 函数的另外一个非常普遍的应用是判断文件是否存在[15]，例如：

```
Function FileExists(filename As String) As Boolean
    On Error Resume Next
    FileExists = (Dir$(filename) <> "")
End Function
```

#### 4.3.2.3. 处理文本文件

**文件打开和关闭**

文本文件是最简单和基本的文件类型，我们可以使用 Open 语句来打开文本文件，其语法为：

```
'文件输入，读取
Open FileName For Input
'文件输出，写入
Open FileName For Output
'添加内容到文件末尾
Open FileName For Appending
```

例如：

```
' #1 为文件号
Open "readme.txt" For Input As #1
```

如果 Open 语句的 pathname 指定的文件不存在，那么，在用 Append、Binary、Output、或 Random 方式打开文件时，可以建立这一文件。

打开文件之后，必须在使用完成后关闭文件，关闭文件的语句为 Close，Close 语句用于关闭 Open 语句所打开的文件。例如以下代码使用 Close 语句来关闭所有为 Output 而打开的三个文件。

```
Dim I, FileName
For I = 1 To 3    ' 循环三次。
    FileName = "TEST" & I    ' 创建文件名。
    Open FileName For Output As #I    ' 打开文件。
```

[15] FileSystemObject 对象模型提供了判断文件存在的方法。

```
    Print #I, "This is a test."    ' 将字符串写入文件。
Next I
Close    ' 将三个已打开的文件全部关闭。
```

Close 语句的语法为：

```
Close [filenumberlist]
```

可选的 filenumberlist 参数为一个或多个文件号，其中 filenumber 为任何有效的文件号，语法如下：

```
[[#]filenumber] [, [#]filenumber] . . .
```

如果省略 filenumberlist，则将关闭 Open 语句打开的所有活动文件。

对于一个应用程序，一般可以手动指定文件号，也可以使用 FreeFile 函数返回一个整数，获得一个可用的函数号。例如：

```
Dim fnum As Integer
fnum = FreeFile()
Open "readme.txt" For Input As #fnum
```

**读取内容**

文本文件打开之后，可以使用 Line Input # 语句逐行读入文件，该语句从已打开的顺序文件中读出一行并将它分配给 String 变量。其语法为：

```
Line Input #filenumber, varname
```

其中 filenumber 为任何有效的文件号，varname 为有效的 String 变量名。

Line Input # 语句一次只从文件中读出一个字符，直到遇到回车符 (Chr(13)) 或回车–换行符 (Chr(13) + Chr(10)) 为止。回车–换行符将被跳过，而不会被附加到字符串上。例如以下代码本示例使用 Line Input # 语句从顺序文件中读入一行数据，并将该行数据赋予一个变量。本示例假设 TESTFILE 文件内含数行文本数据。

```
Dim TextLine
Open "TESTFILE" For Input As #1    ' 打开文件。
Do While Not EOF(1)    ' 循环至文件尾。
    Line Input #1, TextLine    ' 读入一行数据并将其赋予某变量。
    Debug.Print TextLine    ' 在立即窗口中显示数据。
Loop
Close #1    ' 关闭文件。
```

其中的 EOF 函数返回一个 Integer，它包含 Boolean 值 True，表明已经到达为 Random 或顺序 Input 打开的文件的结尾。其语法为：

```
EOF(filenumber)
```

同样，使用 LOF 函数可以返回一个 Long，表示用 Open 语句打开的文件的大小，该大小以字节为单位。其语法为

```
LOF(filenumber)
```

前面说过，对于尚未打开的文件，使用 FileLen 函数将得到其长度。

使用 Input 语句可以从已打开的顺序文件中读出数据并将数据指定给变量。其语法为：

```
Input #filenumber, varlist
```

其中的 Varlist，是用逗号分界的变量列表，将文件中读出的值分配给这些变量；这些变量不可能是一个数组或对象变量。但是，可以使用变量描述数组元素或用户定义类型的元素。

通常可以用 Write # 将 Input # 语句读出的数据写入文件。为了能够用 Input # 语句将文件的数据正确读入到变量中，在将数据写入文件时，要使用 Write # 语句而不使用 Print # 语句。使用 Write # 语句可以确保将各个单独的数据域正确分隔开。

Write # 和 Input # 语句类似于 C 语言的 fprintf 和 fscanf 语句。

以下示例使用 Input # 语句将文件内的数据读入两个变量中。本示例假设 TESTFILE 文件内含数行以 Write # 语句写入的数据；也就是说，每一行数据中的字符串部分都是用双引号括起来，而与数字用逗号隔开，例如，（"Hello", 234）。

```
Dim MyString, MyNumber
Open "TESTFILE" For Input As #1      ' 打开输入文件。
Do While Not EOF(1)    ' 循环至文件尾。
    Input #1, MyString, MyNumber      ' 将数据读入两个变量。
    Debug.Print MyString, MyNumber    ' 在立即窗口中显示数据。
Loop
Close #1    ' 关闭文件。
```

下面这个函数可以将任意文本文件的数据一次读入到一个字符串。

```
Function ReadTextFileContents(filename As String) As String
    Dim fnum As Integer, isOpen As Boolean
On Error GoTo Error_Handler
    ' 获得下一个文件号
    fnum = FreeFile()
    Open filename For Input As #fnum
    isOpen = True
    ' 读入文件
    ReadTextFileContents = Input(LOF(fnum), fnum)

Error_Handler:
    ' 错误处理和文件关闭
```

```
    If isOpen Then Close #fnum
    If Err Then Debug.Print Err.Number, Err.Description
End Function

' 使用此函数
strContent = ReadTextFileContents("c:\test.txt")
```

**写入内容**

对于需要写入的文件，打开时需要使用 For Output 语句或者 For Append 语句，后者添加内容到文件末尾。文件的写入可以使用 Print # 语句。

Print 语句将格式化显示的数据写入顺序文件中。其语法为：

```
Print #filenumber, [outputlist]
```

例如以下代码使用 Print # 语句将数据写入一个文件。

```
Open "c:\TESTFILE" For Output As #1    ' 打开输出文件。
Print #1, "This is a test"    ' 将文本数据写入文件。
Print #1,    ' 将空白行写入文件。
' 数据写入两个区（print zones）。
Print #1, "Zone 1"; Tab; "Zone 2"
' 以空格隔开两个字符串。
Print #1, "Hello"; " "; "World"
' 在字符串之前写入五个空格。
Print #1, Spc(5); "5 leading spaces "
Print #1, Tab(10); "Hello"   ' 将数据写在第十列。

' 赋值 Boolean、Date、Null 及 Error 等。
Dim MyBool, MyDate, MyNull, MyError
MyBool = False: MyDate = #2/12/1969#: MyNull = Null
MyError = CVErr(32767)
' True、False、Null 及 Error 会根据系统的地区设置自动转换格式。
' 日期将以标准的短式日期的格式显示。
Print #1, MyBool; " is a Boolean value"
Print #1, MyDate; " is a date"
Print #1, MyNull; " is a null value"
Print #1, MyError; " is an error value"
Close #1    ' 关闭文件。
```

以上代码写入的文本文件内容如下所示，其中的“.”表示空格。

```
This.is.a.test

Zone.1........Zone.2
Hello.World
```

```
.....5.leading.spaces.
.........Hello
False.is.a.Boolean.value
1969-2-12..is.a.date
Null.is.a.null.value
Error.32767.is.an.error.value
```

通常用 Line Input # 或 Input 读出 Print # 在文件中写入的数据。

Write # 语句应用一定的格式将数据写入顺序文件。其语法为：

```
Write #filenumber, [outputlist]
```

其中的 Outputlist 为要写入文件的数值表达式或字符串表达式，用一个或多个逗号将这些表达式分界。

通常可以用 Input # 从文件读出 Write # 写入的数据。

如果省略 outputlist，并在 filenumber 之后加上一个逗号，则会将一个空白行打印到文件中。多个表达式之间可用空白、分号或逗号隔开。空白和分号等效。

用 Write # 将数据写入文件时将遵循几个通用的约定，使得无论什么区域都可用 Input # 读出并正确解释数据：

- 在写入数值数据时总使用句号作为十进制分隔符。
- 对于 Boolean 类型的数据，或者打印 #TRUE# 或者打印 #FALSE#。无论在什么地区，都不将 True 和 False 这两个关键字翻译出来。
- 使用通用的日期格式将 Date 类型的数据写入文件中。当日期或时间的部件丢失或为零时，只将现有部分写入文件中。
- 如果 outputlist 的数据为 Empty，则不将任何数据写入文件。但对 Null 数据，则要写入 #NULL#。
- 如果 outputlist 数据为 Null 数据，则将 #NULL# 写入文件中。
- 对于 Error 类型的数据，输出看起来与 #ERROR errorcode# 一样。无论在什么地区，都不将关键字 Error 翻译出来。

与 Print # 语句不同，当要将数据写入文件时，Write # 语句会在项目和用来标记字符串的引号之间插入逗号。没有必要在列表中键入明确的分界符。Write # 语句在将 outputlist 中的最后一个字符写入文件后会插入一个新行字符，即回车换行符，(Chr(13) + Chr(10))。

以下示例使用 Write # 语句将行数据写入顺序文件。

```
Open "c:\TESTFILE" For Output As #1    ' 打开输出文件。
Write #1, "Hello World", 234    ' 写入以逗号隔开的数据。
```

```
Write #1,    ' 写入空白行。

Dim MyBool, MyDate, MyNull, MyError
' 赋值 Boolean、Date、Null 及 Error 等。
MyBool = False: MyDate = #2/12/1969#: MyNull = Null
MyError = CVErr(32767)
' Boolean 数据以 #TRUE# 或 #FALSE# 的格式写入。
' 日期以通用日期格式写入，例如：#1994-07-13# 代表
' 1994 年 1 月 13 日。Null 数据以 #NULL# 格式写入。
' Error 数据以 #ERROR 错误代号# 的格式写入。
Write #1, MyBool; " is a Boolean value"
Write #1, MyDate; " is a date"
Write #1, MyNull; " is a null value"
Write #1, MyError; " is an error value"
Close #1    ' 关闭文件。
```

这段代码写入的文件内容为：

```
"Hello World",234

#FALSE#," is a Boolean value"
#1969-02-12#," is a date"
#NULL#," is a null value"
#ERROR 32767#," is an error value"
```

#### 4.3.2.4. 处理二进制文件

打开二进制文件可以使用 Open FileName For Random 和 Open FileName For Binary 语句。使用二进制模式（Binary），我们可以使用 Put 语句写入内容，使用 Get 语句读入内容。VBA 根据这 2 个函数的参数的数据类型决定写入或者读取多少内容。以下代码以二进制方式写入 2 个变量的内容。

```
Dim numEls As Long, text As String
numEls = 12345
text = "A 16-char string"
' 打开或创建一个二进制文件
Open "data.bin" For Binary As #1
Put #1, , numEls          ' 写入 4 bytes.
Put #1, , text            ' 写入 16 bytes (ANSI format).
Close #1
```

当读入数据时，必须使用相同的顺序来读取，例如以上代码写入的文件可以这样读入：

```
Dim numEls As Long, text As String
Open "data.bin" For Binary As #1
```

```
Get #1, , numEls
text = Space$(16)          ' 准备 16 bytes 的字符串
Get #1, , text             ' 读入
Debug.Print numEls, text
Close #1
```

读入数据时，可以使用 Seek 语句或者指定 Get 语句的第二个参数指定读取位置。Seek 语句的语法为：

```
Seek [#]filenumber, position
```

其中 position 为介于 1 – 2，147，483，647 之间的数字，指出下一个读写操作将要发生的位置。

在 Get 及 Put 语句也可以指定记录号，该记录号将覆盖由 Seek 语句指定的文件位置。若在文件结尾之后进行 Seek 操作，则进行文件写入的操作会把文件扩大。如果试图对一个位置为负数或零的文件进行 Seek 操作，则会导致错误发生。

使用 Get 语句读取文件时，必须使用 LOF 函数（文件的长度）来判断是否到达文件末尾，而不是使用 EOF 函数，例如可以使用 Seek 函数（不是 Seek 语句，其参数只有文件号，返回当前位置）判断当前位置，然后比较 LOF 函数的返回值：

```
Do While Seek(1) < LOF(1)
    ' Continue to read.
      ....
Loop
```

#### 4.3.2.5. 总结

VBA 语句的文件操作函数，涵盖了文件操作的绝大部分内容，可以满足绝大多数情况下的文件操作要求。很多函数的使用也非常简单，例如文件重命名、移动的 Name 语句，或者文件删除的 Kill 语句；一般文件的读写也非常方便；有些函数，例如 Dir，功能也很强大，可以完成复杂的文件夹遍历。但另一方面，VBA 语句的文件操作由于其语句和函数的特征，使用和学习并不方便，对于复杂的文件读写，代码的结构和维护性都不好，因此在 VB 和 VBA 版本 6 之后，微软引入了 FileSystemObject 对象模型，提供了面向对象的类库，来操作驱动器、文件夹和文件。但对于二进制文件的操作，目前还只能使用 VBA 语句。

笔者认为，对于一些简单的文件操作，例如重命名，判断文件是否存在（使用 Dir），写入或读取一个简单的文本文件，使用 VBA 语句更简单和直观，但如果文件操作比较复

杂，还是使用 FileSystemObject 对象模型比较合适，因为后者的面向对象的语法更易于代码的书写和维护。

### 4.3.3. FileSystemObject 对象模型

FileSystemObject (FSO，下面简称为 FSO) 对象模型，具有大量的属性、方法和事件，使用面向对象的“object.method”语法，来处理文件夹和文件，可以在 Office 2000 以后版本使用。FileSystemObject 并不是 VBA 的一部分，它是以一个 COM 组件的形式提供，可以在 VB、VBA、VBScript 中使用。

FSO 对象模型可以创建、改变、移动和删除文件夹，或探测特定的文件夹是否存在，若存在，还可以找出有关文件夹的信息，如名称、被创建或最后一次修改的日期，等等。FSO 对象模型还使文件处理变得很容易。可以创建文件，插入和改变数据，以及输出（读取）数据。FSO 对象模型，支持通过 TextStream 对象来创建和操作文本文件，但不支持二进制文件的创建或操作[16]。

FileSystemObject 对象模型包含下面的对象和集合（表 4-2）。

表 4-2 FileSystemObject 对象模型内的对象和集合

| 对象/集合 | 描　　述 |
|---|---|
| FileSystemObject | 主对象。包含用来创建、删除和获得有关信息，以及通常用来操作驱动器、文件夹和文件的方法和属性。和该对象相关联的许多方法，与其他 FSO 对象中的方法完全相似；它们是为了方便才被提供的。 |
| Drive | 对象。包含用来收集信息的方法和属性，这些信息是关于连接在系统上的驱动器的，如驱动器的共享名和它有多少可用空间。请注意，"drive" 并非必须是硬盘，也可以是 CD-ROM 驱动器，RAM 磁盘等等。并非必须把驱动器实物地连接到系统上；它也可以通过网络在逻辑上被连接起来。 |
| Drives | 集合。提供驱动器的列表，这些驱动器实物地或在逻辑上与系统相连接。Drives 集合包括所有驱动器，与类型无关。要可移动的媒体驱动器在该集合中显现，不必把媒体插入到驱动器中。 |
| File | 对象。包含用来创建、删除或移动文件的方法和属性。也用来向系统询问文件名、路径和多种其他属性。 |
| Files | 集合。提供包含在文件夹内的所有文件的列表。 |
| Folder | 对象。包含用来创建、删除或移动文件夹的方法和属性。也用来向系统询问文件夹名、路径和多种其他属性。 |
| Folders | 集合。提供在 Folder 内的所有文件夹的列表。 |

[16] 微软在有关文档中称计划将来支持二进制文件，不过应该只是计划。:）

| TextStream | 对象。用来读写文本文件。 |
|---|---|

#### 4.3.3.1. 使用方法

使用 FileSystemObject 对象模型进行文件操作的步骤是：

1. 使用 CreateObject 方法来创建 FileSystemObject 对象；
2. 在新创建的对象上使用适当的方法；
3. 访问对象的属性。

**创建 FileSystemObject 对象**

首先，使用 CreateObject 方法来创建 FileSystemObject 对象。

下面代码显示如何创建 FileSystemObject 实例：

```
Dim fso
Set fso = CreateObject("Scripting.FileSystemObject")
```

在这个示例中，Scripting 是类型库的名字，而 FileSystemObject 则是想要创建的对象的名字。

CreateObject 函数创建并返回一个对 ActiveX 对象的引用。其语法为：

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

Class 参数使用 appname.objecttype 这种语法，包括以下部分：

- appname 必需的；Variant（字符串）。提供该对象的应用程序名。
- objecttype 必需的；Variant（字符串）。待创建对象的类型或类。

例如：

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

上述代码创建一个 Microsoft Excel 电子数据表。我们后边还要介绍使用 ActiveX 对象（COM 对象）的方法以及 CreateObject 函数，这里只要知道 CreateObject 函数返回了一个具体的对象实例就可以了。

**使用适当的方法**

其次，使用 FileSystemObject 对象的适当方法。例如，要创建一个新的对象，则使用 CreateTextFile 或 CreateFolder（FSO 对象模型不支持驱动器的创建或删除）。

要删除对象，则使用 FileSystemObject 对象的 DeleteFile 和 DeleteFolder 方法，或 File 和 Folder 对象的 Delete 方法。也可以使用适当的方法，来复制和移动文件与文件夹。

FileSystemObject 对象模型中的某些功能是重复的。例如，可以用 FileSystemObject 对象的 CopyFile 方法，也可以用 File 对象的 Copy 方法来复制文件。这两种方法功能是相同的；两种方法都能使编程灵活。

**访问现有驱动器、文件和文件夹**

要访问现有驱动器、文件或文件夹，则使用 FileSystemObject 对象中的适当的“get”方法：

- GetDrive
- GetFolder
- GetFile

若要访问现有文件：

```
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.GetFile("c:\test.txt")
```

不要对新创建的对象使用“get”方法，因为“create”函数已经返回那个对象的一个实例。例如，如果使用 CreateFolder 方法创建了一个新的文件夹，则不要使用 GetFolder 方法来访问它的属性，如 Name、Path、Size 等等。只需设一个变量给 CreateFolder 函数，来获得新创建文件夹的对象实例，然后访问它的属性、方法和事件。

若要为 CreateFolder 函数创建变量，请使用该语法：

```
Sub CreateFolder
   Dim fso, fldr
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set fldr = fso.CreateFolder("C:\MyTest")
   MsgBox "Created folder: " & fldr.Name
End Sub
```

**访问对象的属性**

一旦有了对象的实例，就可以访问它的属性。例如，要获得特定文件夹的名字，首先创建该对象的一个实例，然后赋给一个对象变量（在这个例子中是 GetFolder 方法，因为该文件夹已经存在了）。

用该代码来获得 GetFolder 方法的一个实例：

```
Set fldr = fso.GetFolder("c:\")
```

现在，已经有了 Folder 对象的实例，就可以检查它的 Name 属性了。

```
MsgBox "Folder name is: " & fldr.Name
```

要找出最后一次修改文件的时间，则使用下面的语法：

```
Dim fso, f1
Set fso = CreateObject("Scripting.FileSystemObject")
' 获得要查询的 File 对象。
Set f1 = fso.GetFile("c:\detlog.txt")
' 打印信息。
MsgBox "File last modified: " & f1.DateLastModified
```

### 4.3.3.2. 处理驱动器和文件夹

使用 FileSystemObject（FSO）对象模型，可以处理驱动器和文件夹，就像在 Windows 资源管理器中交互式地处理它们一样。可以复制和移动文件夹，获取有关驱动器和文件夹的信息，等等。

**获取有关驱动器的信息**

可以用 Drive 对象来获得有关各种驱动器的信息，这些驱动器是实物地或通过网络连接到系统上的。它的属性可以用来获得下面的信息内容：

- 驱动器的总容量，以字节为单位（TotalSize 属性）
- 驱动器的可用空间是多少，以字节为单位（AvailableSpace 或 FreeSpace 属性）
- 哪个号被赋给了该驱动器（DriveLetter 属性）
- 驱动器的类型是什么，如可移动的、固定的、网络的、CD-ROM 或 RAM 磁盘（DriveType 属性）
- 驱动器的序列号（SerialNumber 属性）
- 驱动器使用的文件系统类型，如 FAT、FAT32、NTFS 等等（FileSystem 属性）
- 驱动器是否可以使用（IsReady 属性）
- 共享和/或卷的名字（ShareName 和 VolumeName 属性）
- 驱动器的路径或根文件夹（Path 和 RootFolder 属性）

**Drive 对象的用法示例**

使用 Drive 对象来收集有关驱动器的信息。在下面的代码中，没有对实际的 Drive 对象的引用；相反，使用 GetDrive 方法来获得现有 Drive 对象的引用（在这个例子中就是 drv）。

下面示例示范了如何使用 Drive 对象：

```
Sub ShowDriveInfo(drvPath)
   Dim fso, drv, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set drv = fso.GetDrive(fso.GetDriveName(drvPath))
   s = "Drive " & UCase(drvPath) & " - "
   s = s & drv.VolumeName & vbCrLf
   s = s & "Total Space: " & _
         FormatNumber(drv.TotalSize / 1024, 0)
   s = s & " Kb" & vbCrLf
   s = s & "Free Space: " & _
         FormatNumber(drv.FreeSpace / 1024, 0)
   s = s & " Kb" & vbCrLf
   MsgBox s
End Sub
```

**处理文件夹**

在表 4-3 中，描述了普通的文件夹任务和执行它们的方法。

表 4-3 使用 FSO 处理文件夹的任务和方法

| 任务 | 方法 |
|---|---|
| 创建文件夹 | FileSystemObject.CreateFolder |
| 删除文件夹 | Folder.Delete 或 FileSystemObject.DeleteFolder |
| 移动文件夹 | Folder.Move 或 FileSystemObject.MoveFolder |
| 复制文件夹 | Folder.Copy 或 FileSystemObject.CopyFolder |
| 检索文件夹的名字 | Folder.Name |
| 如果文件夹在驱动器上存在，则找出它 | FileSystemObject.FolderExists |
| 获得现有 **Folder** 对象的实例 | FileSystemObject.GetFolder |
| 找出文件夹的父文件夹名 | FileSystemObject.GetParentFolderName |
| 找出系统文件夹的路径 | FileSystemObject.GetSpecialFolder |

下面的示例示范了如何使用 Folder 和 FileSystemObject 对象，来操作文件夹和获得

有关它们的信息：

```
Sub ShowFolderInfo()
   Dim fso, fldr, s
   ' 获取 FileSystemObject 的一个实例。
   Set fso = CreateObject("Scripting.FileSystemObject")
   ' 获取驱动器对象。
   Set fldr = fso.GetFolder("c:")
   ' 打印父文件夹名。
   MsgBox "Parent folder name is: " & fldr
   ' 打印驱动器名。
   MsgBox "Contained on drive " & fldr.Drive & vbCrLf
   ' 打印根文件名。
   If fldr.IsRootFolder = True Then
      MsgBox "This is the root folder."
   Else
      MsgBox "This folder isn't a root folder."
   End If

   ' 用 FileSystemObject 对象新建文件夹。
   fso.CreateFolder ("C:\Bogus")
   MsgBox "Created folder C:\Bogus"
   ' 打印文件夹的基本名。
   MsgBox "Basename = " & fso.GetBaseName("c:\bogus")
   ' 删除新建文件夹。
   fso.DeleteFolder ("C:\Bogus")
   MsgBox "Deleted folder C:\Bogus"
End Sub
```

### 4.3.3.3. 处理文件

有两种主要的文件处理类型：

- 创建、添加或删除数据，以及读取文件
- 移动、复制和删除文件

#### 创建文件

创建空文本文件有三种方法。

第一种方法是用 CreateTextFile 方法。下面的示例示范了如何用 CreateTextFile 方法创建文本文件：

```
Dim fso, f1
```

```
Set fso = CreateObject("Scripting.FileSystemObject")
Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
```

创建文本文件的第二种方法是，使用 FileSystemObject 对象的 OpenTextFile 方法，并设置 ForWriting 标志。

```
Dim fso, ts
Const ForWriting = 2
Set fso = CreateObject("Scripting. FileSystemObject")
Set ts = fso.OpenTextFile("c:\test.txt", ForWriting, True)
```

创建文本文件的第三种方法是，使用 OpenAsTextStream 方法，并设置 ForWriting 标志。要使用这种方法，使用下面的代码：

```
Dim fso, f1, ts
Const ForWriting = 2
Set fso = CreateObject("Scripting.FileSystemObject")
fso.CreateTextFile ("c:\test1.txt")
Set f1 = fso.GetFile("c:\test1.txt")
Set ts = f1.OpenAsTextStream(ForWriting, True)
```

**添加数据到文件中**

一旦创建了文本文件，使用下面的三个步骤向文件添加数据：

- 打开文本文件；
- 写入数据；
- 关闭文件。

要打开现有的文件，则使用 FileSystemObject 对象的 OpenTextFile 方法或 File 对象的 OpenAsTextStream 方法。

要写数据到打开的文本文件，则根据表 4-4 所述任务使用 TextStream 对象的 Write、WriteLine 或 WriteBlankLines 方法。

表 4-4 FSO 中写数据到文本文件的方法

| 任务 | 方法 |
|---|---|
| 向打开的文本文件写数据，不用后续一个新行字符。 | Write |
| 向打开的文本文件写数据，后续一个新行字符。 | WriteLine |
| 向打开的文本文件写一个或多个空白行。 | WriteBlankLines |

要关闭一个打开的文件，则使用 TextStream 对象的 Close 方法。

下面的例子示范了如何打开文件，和同时使用三种写方法来向文件添加数据，然后关

闭文件：

```
Sub CreateFile()
   Dim fso, tf
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set tf = fso.CreateTextFile("c:\testfile.txt", True)
   ' 写一行，并带有一个新行字符。
   tf.WriteLine("Testing 1, 2, 3.")
   '  向文件写三个新行字符。
   tf.WriteBlankLines(3)
   ' 写一行。
   tf.Write ("This is a test.")
   tf.Close
End Sub
```

**读取文件**

要从文本文件读取数据，则使用 TextStream 对象的 Read、ReadLine 或 ReadAll 方法。表 4-5 描述了不同的任务应使用哪种方法。

表 4-5 FSO 中读取文本文件的方法

| 任务 | 方法 |
|---|---|
| 从文件读取指定数量的字符。 | Read |
| 读取一整行（一直到但不包括新行字符）。 | ReadLine |
| 读取文本文件的整个内容。 | ReadAll |

如果使用 Read 或 ReadLine 方法，并且想跳过数据的特殊部分，则使用 Skip 或 SkipLine 方法。read 方法的结果文本存在一个字符串中，该字符串可以显示在一个控件中，也可以用字符串函数（如 Left、Right 和 Mid）来分析，连接等等。

下面的例子示范了如何打开文件，和如何写数据到文件中并从文件读取数据：

```
Sub ReadFiles
   Dim fso, f1, ts, s
   Const ForReading = 1
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f1 = fso.CreateTextFile("c:\testfile.txt", True)
   ' 写入一行。
   MsgBox "Writing file … "
   f1.WriteLine "Hello World"
   f1.WriteBlankLines(1)
   f1.Close
```

```
   ' 读取文件内容。
   MsgBox "Reading file … "
   Set ts = fso.OpenTextFile("c:\testfile.txt", ForReading)
   s = ts.ReadLine
   MsgBox "File contents = '" & s & "'"
   ts.Close
End Sub
```

### 移动、复制和删除文件

FSO 对象模型各有两种方法移动、复制和删除文件，如下表所述。

表 4-6 FSO 中移动、复制和删除文件的方法

| 任务 | 方法 |
|---|---|
| 移动文件 | File.Move 或 FileSystemObject.MoveFile |
| 复制文件 | File.Copy 或 FileSystemObject.CopyFile |
| 删除文件 | File.Delete 或 FileSystemObject.DeleteFile |

下面示例在驱动器 C 的根目录中创建一个文本文件，向其中写一些信息，然后把它移动到 \tmp 目录中，并在 \temp 中做一个备份，最后把它们从两个目录中删掉。

要运行下面的示例，需要先在驱动器 C 的根目录中创建 \tmp 和 \temp 目录：

```
Sub ManipFiles
   Dim fso, f1, f2, f3, s
   Set fso = CreateObject("Scripting.FileSystemObject")
   Set f1 = fso.CreateTextFile("c:\testfile.txt", True)

   MsgBox "Writing file … "

   ' 写入一行。
   f1.Write ("This is a test.")
   ' 关闭写入到的文件。
   f1.Close

   MsgBox "Moving file to c:\tmp"

   ' 获取到 C:\ 根目录中文件对象。
   Set f2 = fso.GetFile("c:\testfile.txt")
   ' 将文件移到 \tmp 目录。
   f2.Move ("c:\tmp\testfile.txt")

   MsgBox "Copying file to c:\temp"
```

```
    ' 将文件复制到 \temp。
    f2.Copy ("c:\temp\testfile.txt")

    MsgBox "Deleting files"

    ' 获得文件当前位置的对象。
    Set f2 = fso.GetFile("c:\tmp\testfile.txt")
    Set f3 = fso.GetFile("c:\temp\testfile.txt")

    ' 删除文件。
    f2.Delete
    f3.Delete
    MsgBox "All done!"
End Sub
```

#### 4.3.3.4. FSO 总结

由以上代码可以看到，使用 FSO 处理文件、文件夹比使用 VBA 语句的方法具有更易操作的特点，FSO 除了不能处理二进制文件，其文件和文件夹处理的方法也更完备，更直观和易于使用。

## 4.4. 操作数据

上面介绍了数据文件的操作。数据处理的另一部分工作就是操作数据，包括如何从工作表获取数据，获取的数据的验证，不同类型数据的操作方法等。数据操作首先需要进行类型判断和类型转换，然后是不同类型的数据的操作方法和常用函数，很多有关内容在 VBA 简介一章已做了介绍，例如数据类型转换（表 2-3，表 2-4），字符串操作（表 2-6），日期和时间操作（表 2-7）。本章则主要介绍一些重点内容。

### 4.4.1. 工作表数据引用

工作表数据获取的前提是理解 Excel 的对象模型，我们在 Excel 对象模型一章已对整个对象模型做了详细的介绍，这里仅从数据操作的角度来介绍其内容。

如图 4-3 所示，Excel 对象模型的最顶端对象是 Application 对象，Application 对象代表 Excel 应用程序本身，在 VBA 中，Application 是默认对象，所以可以不表明，例如使用

以下代码激活“book1”窗口，可以使用：

```
Application.Windows("book1.xls").Activate
```

也可以使用：

```
Windows("book1.xls").Activate
```

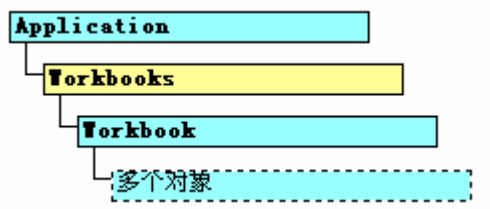



图 4-3 Application, Workbooks, Workbook 对象的关系

Application 对象的 Workbooks 属性返回一个 Workbooks 集合对象，该对象表示所有打开的 Excel 工作薄的集合，使用该集合的 Item 属性可以返回一个 Workbook 对象，代表打开的某个工作薄（图 4-3）。因此可以使用 Workbooks 引用打开的工作薄，例如：

```
Workbooks.Item("test.xla")
```

或者返回一个 Workbook（工作薄）对象：

```
Set wb = Workbooks.Item("myaddin.xla")
```

由于 Item 属性为集合对象的默认属性，因此以上代码也可以简略为：

```
Workbooks("test.xla")
Set wb = Workbooks("myaddin.xla")
```

得到代表工作薄的 Workbook 对象后，即可使用其 Worksheets 属性，返回代表该工作薄所有工作表的 Worksheets 集合（图 4-4）。

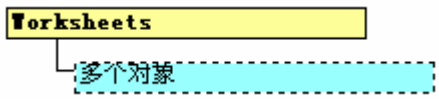



图 4-4 Worksheets 集合对象

例如，可以使用以下代码使用 Worksheets 集合：

```
For Each ws In Workbooks("test.xla").Worksheets
    MsgBox ws.Name
Next ws
```

Application 对象也有 Worksheets 属性，返回的为当前获得工作薄的 Worksheets 集合。

通过 Worksheets 集合，即可获得代表工作表的 Worksheet 对象，每个 Worksheet 对象代表一张工作表（图 4-5）。例如使用以下代码返回当前活动工作薄的的 Worksheets 集合，然后由此集合操作第一个工作表：

```
Worksheets(1).Visible = False
Set Ws = Workbooks("test.xla").Worksheets(1)
```

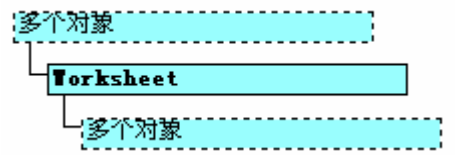



图 4-5 Worksheet 对象

得到 Worksheet 对象后，即可使用其 Range 属性返回 Range 对象，返回一个 Range 对象，该对象代表一个单元格或单元格区域，而获取工作表中的全部或部分单元格（图 4-6）。

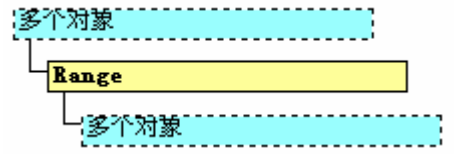



图 4-6 Range 对象

通过 Range 属性，即可对单元格进行操作。Application 对象的 Range 属性代表当前活动工作表的 Range 属性。可以通过如下方法使用 Range 对象：

```
Worksheets("Sheet1").Range("A1").Value = 3.14159
```

以下代码在 Sheet1 的 A1 单元格中创建一个公式。

```
Worksheets("Sheet1").Range("A1").Formula = "=10*RAND()"
```

以下代码在 Sheet1 的单元格区域 A1:D10 上进行循环。如果某个单元格的值小于 0.001，则此代码将用 0（零）来取代该值。

```
For Each c in Worksheets("Sheet1").Range("A1:D10")
    If c.Value < .001 Then
        c.Value = 0
    End If
Next c
```

以下代码在名为“TestRange”的区域上进行循环，并显示该区域中空白单元格的个数。

```
numBlanks = 0
For Each c In Range("TestRange")
```

```
    If c.Value = "" Then
        numBlanks = numBlanks + 1
    End If
Next c
MsgBox "There are " & numBlanks & " empty cells in this range"
```

有关 Range 对象的进一步详细使用说明，请参考 Excel 对象模型一章。这样，我们就可以使用 Excel 对象模型操作工作表中的数据，获取、修改、设置某个或者某几个单元格的内容。

综上所述，我们对工作表中数据的操作做如下图示（图 4-7），在实际编程中，应该时刻明确所操作的数据是什么数据，位于整个对象模型的那个层次，这样才可以更好的处理数据。

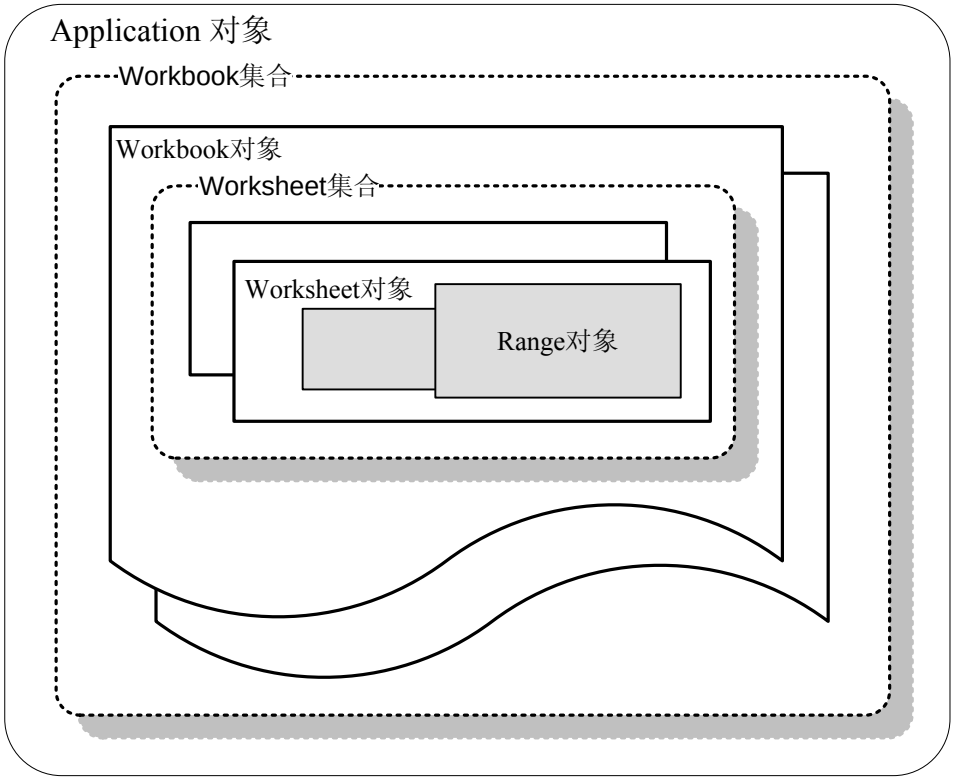



图 4-7 Excel 对象相互关系示意图

## 4.4.2. 操作文本

Excel 和 VBA 不仅可以处理各种数值问题，也可以对文本进行处理。VBA 有一些功能强大的字符串处理函数，以下将按照类别来介绍使用 VBA 进行数据处理的函数和方法。关于 VBA 详细的字符串操作函数列表，请参考 VBA 简介的表 2-6。

#### 4.4.2.1. 基础的文本处理运算符和函数

字符串连接使用“&”运算符，虽然也可以使用“＋”运算符连接字符串，但不推荐这样做。

```
strTest = "ABCDE" & "1234"
```

另外几个常用的字符串函数是 Left, Right, 和 Mid, 分别从开始，结束或中间位置提取字符串的一部分，例如：

```
Text = "123456789"
Debug.Print Left$(text, 3)        ' "123"
Debug.Print Right$(text, 2)       ' "89"
Debug.Print Mid$(text, 3, 4)      ' "3456"
```

Mid 也可以作为命令，来改变一个字符串的一部分，例如：

```
Text = "123456789"
Mid(Text, 3, 4) = "abcd"    ' Text = "12abcd789"
```

Len 函数返回字符串的长度，例如：

```
Debug.Print Len("12345")         ' "5"
If Len(Text) = 0 Then ...   ' 比 Text=""快
```

可以使用 LTrim, RTrim, 和 Trim 来去处字符串头尾的空白，例如：

```
Text = "  abcde  "
Debug.Print LTrim(Text)          ' "abcde  "
Debug.Print RTrim(Text)          ' "  abcde"
Debug.Print Trim(Text)           ' "abcde"
```

Asc 函数返回字符串第一个字符的 Ascii 码，而 Chr 则根据 Ascii 码得到一个字符。

StrComp 函数逐字符比较 2 个字符串，分别返回-1, 0, 或者 1，代表字符串小于、等于和大于。例如：

```
Select Case StrComp(first, second, vbTextCompare)
   Case 0
      ' first = second   (e.g. "VISUAL" vs. "Visual")
   Case -1
      ' first < second   (e.g. "C++" vs. "Visual")
   Case 1
      ' first > second   (e.g. "Basic" vs. "Delphi")
End Select
```

#### 4.4.2.2. 转换函数

最常用的转换函数是 Ucase 和 Lcase，分别转换字符串为大写或小写。例如：

```
Text = "New York, USA"
Debug.Print UCase$(Text)                ' "NEW YORK, USA"
Debug.Print LCase$(Text)                ' "new york, usa"
```

StrConv 函数可以对转换做更多的选择，其语法为：

```
StrConv(string, conversion, LCID)
```

其中 conversion 参数的设置值见表 4-7：

表 4-7 StrConv 函数 conversion 参数设置值

| 常数 | 值 | 说明 |
|---|---|---|
| vbUpperCase | 1 | 将字符串文字转成大写。 |
| vbLowerCase | 2 | 将字符串文字转成小写。 |
| vbProperCase | 3 | 将字符串中每个字的开头字母转成大写。 |
| vbWide* | 4* | 将字符串中单字节字符转成双字节字符。 |
| vbNarrow* | 8* | 将字符串中双字节字符转成单字节字符。 |
| vbKatakana** | 16** | 将字符串中平假名字符转成片假名字符。 |
| vbHiragana** | 32** | 将字符串中片假名字符转成平假名字符。 |
| vbUnicode | 64 | 根据系统的缺省码页将字符串转成 Unicode。 |
| vbFromUnicode | 128 | 将字符串由 Unicode 转成系统的缺省码页。 |

*应用到远东区域。

**仅应用到日本。

表 4-7 中这些常数是由 VBA 指定的。可以在程序中使用它们来替换真正的值。其中大部分是可以组合的，例如 vbUpperCase + vbWide，互斥的常数不能组合，例如 vbUnicode + vbFromUnicode。当在不适用的区域使用常数 vbWide、vbNarrow、vbKatakana，和 vbHiragana 时，就会导致运行时错误。

例如：

```
MsgBox StrConv(Text, vbUpperCase)   ' "NEW YORK, USA"
MsgBox StrConv(Text, vbLowerCase)   ' "new york, usa"
MsgBox StrConv(Text, vbProperCase)  ' "New York, Usa"
```

### 4.4.2.3. 查找和替换子字符串

使用 InStr 可以在字符串中查找一个子字符串，返回最先出现的位置，例如：

```
MsgBox InStr("abcde ABCDE", "ABC")      '"7"
MsgBox InStr(8, "abcde ABCDE", "ABC")  '"0" (开始位置 > 1)
MsgBox InStr(1, "abcde ABCDE", "ABC", vbTextCompare) '"1"
```

InStrRev 函数与 InStr 类似，不过从末尾开始查找。

可以使用 Replace 函数替换字符串中的某个子字符串，并返回替换后的字符串，其语法为：

```
Text = Replace(原字符串, 查找的字符串, 用来替换的子字符串, [开始位置], [替换次数], [比较模式])
```

例如：

```
MsgBox Replace("abc ABC abc", "ab", "123")  ' "123c ABC 123c"
```

StrReverse 函数返回一个反向的字符串。

Split 函数返回一个下标从零开始的一维数组，它包含指定数目的子字符串。其语法为：

```
Split(expression[, delimiter[, limit[, compare]]])
```

其中 expression 为要拆分的字符串， delimiter 是表示子字符串的分隔符，缺省是空格“ ”，limit 表示要返回的子字符串数，–1 表示返回所有的子字符串；compare 表示判别子字符串时使用的比较方式，可以是二进制或者文本方式。

例如以下代码:

```
Dim words() As String
words() = Split("Microsoft Visual Basic 6")
```

words() 现在是一个 4 个元素的数组，其元素为“Microsoft”，“Visual”，“Basic”，“6”。

Join 函数则为连接字符串数组中的各个元素，返回一个字符串，其速度比使用“&”快，例如：

```
Debug.Print Join(words, " ")  ' "Microsoft Visual Basic 6"
```

#### 4.4.2.4. 字符串的格式化

VBA 中，可以使用 Format 函数来完成字符串格式化。Format 函数的语法为：

```
Format(expression[,format])
```

其中 expression 为任何有效的表达式，format 为要格式化的格式，例如可以如下使用 Format 函数：

```
Dim MyTime, MyDate, MyStr
MyTime = #17:04:23#
MyDate = #January 27, 1993#

'以系统设置的长时间格式返回当前系统时间。
MyStr = Format(Time, "Long Time")

'以系统设置的长日期格式返回当前系统日期。
MyStr = Format(Date, "Long Date")
```

```
MyStr = Format(MyTime, "h:m:s")    '返回 "17:4:23"。
MyStr = Format(MyTime, "hh:mm:ss AMPM")   '返回 "05:04:23 PM"。

'如果没有指定格式，则返回字符串。
MyStr = Format(23)    '返回 "23"。

'用户自定义的格式。
MyStr = Format(5459.4, "##,##0.00")    ' 返回 "5,459.40"。
MyStr = Format(334。9, "###0.00")    ' 返回 "334.90"。
MyStr = Format(5, "0.00%")    ' 返回 "500.00%"。
MyStr = Format("HELLO", "<")    ' 返回 "hello"。
MyStr = Format("This is it", ">")    ' 返回 "THIS IS IT"。
```

总之，使用 Format 可以将字符串格式化为任何需要的格式，用好 Format 函数可以在实际开发中省却很多不必要的麻烦和工作，其具体格式化字符串（参数 format）的说明请参考 VBA 的帮助文档。但是要注意，Format 函数比一般的数据类型转换函数（如 CStr）要慢 1 到 10 倍（根据格式的不同），因此，尽量不要频繁使用此函数，更不要滥用此函数。

## 4.4.3. 操作数值

### 4.4.3.1. 数学运算

VBA 算术运算符有以下运算符：

- ^ 运算符：求一个数字的某次方，如 A^B；
- 运算符：乘法运算；
- / 运算符：除法运算；
- \ 运算符：对两个数作除法并返回一个整数；
- Mod 运算符：求两数的余数；
- + 运算符：加法运算；
- - 运算符：减法运算。

以下为一些例子：

```
Dim a As Long, b As Long, result As Long
result = a / b          ' 浮动数除法
result = a \ b          ' 整数除法

Dim x As Double, result As Double
```

```
x = 1.2345
result = x ^ 3
result = x * x * x      '与上面的结果相同
```

#### 4.4.3.2. 数学函数

VBA 具有以下常用的数学函数：

- Abs 函数
- Atn 函数
- Cos 函数
- Exp 函数，返回 Double，指定 e（自然对数的底）的某次方。
- Fix 函数，返回参数的整数部分。
- Int 函数，返回参数的整数部分。
- Log 函数，返回一个 Double，指定参数的自然对数值。
- Rnd 函数，求随机数
- Sgn 函数，返回一个整数，指出参数的正负号。
- Sin 函数
- Sqr 函数
- Tan 函数

以下是一些说明和例子。

自然对数是以 e 为底的对数。常数 e 的值大约是 2.718282。

如下所示，将 x 的自然对数值除以 n 的自然对数值，就可以对任意底 n 来计算数值 x 的对数值：

```
Logn(x) = Log(x) / Log(n)
```

下面的示例说明如何编写一个函数来求以 10 为底的对数值：

```
Static Function Log10(X)
    Log10 = Log(X) / Log(10#)
End Function
```

Int 和 Fix 都会删除 number 的小数部份而返回剩下的整数。

Int 和 Fix 的不同之处在于，如果 number 为负数，则 Int 返回小于或等于 number 的第一个负整数，而 Fix 则会返回大于或等于 number 的第一个负整数。例如，Int 将 -8.4 转换成 -9，而 Fix 将 -8.4 转换成 -8。例如：

```
Dim MyNumber
MyNumber = Int(99.8)    ' 返回 99。
MyNumber = Fix(99.2)    ' 返回 99。

MyNumber = Int(-99.8)    ' 返回 -100。
MyNumber = Fix(-99.8)    ' 返回 -99。

MyNumber = Int(-99.2)    ' 返回 -100。
MyNumber = Fix(-99.2)    ' 返回 -99。
```

我们可以使用这些基本的数学函数，导出以下一些常用的数学函数（表 4-8）。

表 4-8 由基本数学函数导出的非基本数学函数列表

| 函数 | 由基本函数导出之公式 |
|---|---|
| Secant（正割） | Sec(X) = 1 / Cos(X) |
| Cosecant（余割） | Cosec(X) = 1 / Sin(X) |
| Cotangent（余切） | Cotan(X) = 1 / Tan(X) |
| Inverse Sine （反正弦） | Arcsin(X) = Atn(X / Sqr(-X * X + 1)) |
| Inverse Cosine （反余弦） | Arccos(X) = Atn(-X / Sqr(-X * X + 1)) + 2 * Atn(1) |
| Inverse Secant （反正割） | Arcsec(X) = Atn(X / Sqr(X * X - 1)) + Sgn((X) - 1) * (2 * Atn(1)) |
| Inverse Cosecant （反余割） | Arccosec(X) = Atn(X / Sqr(X * X - 1)) + (Sgn(X) - 1) * (2 * Atn(1)) |
| Inverse Cotangent （反余切） | Arccotan(X) = Atn(X) + 2 * Atn(1) |
| Hyperbolic Sine （双曲正弦） | HSin(X) = (Exp(X) - Exp(-X)) / 2 |
| Hyperbolic Cosine （双曲余弦） | HCos(X) = (Exp(X) + Exp(-X)) / 2 |
| Hyperbolic Tangent （双曲正切） | HTan(X) = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X)) |
| Hyperbolic Secant （双曲正割） | HSec(X) = 2 / (Exp(X) + Exp(-X)) |
| Hyperbolic Cosecant（双曲余割） | HCosec(X) = 2 / (Exp(X) - Exp(-X)) |
| Hyperbolic Cotangent（双曲余切） | HCotan(X) = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X)) |
| Inverse Hyperbolic Sine（反双曲正弦） | HArcsin(X) = Log(X + Sqr(X * X + 1)) |
| Inverse Hyperbolic Cosine（反双曲余弦） | HArccos(X) = Log(X + Sqr(X * X - 1)) |
| Inverse Hyperbolic Tangent（反双曲正切） | HArctan(X) = Log((1 + X) / (1 - X)) / 2 |
| Inverse Hyperbolic Secant（反双曲正割） | HArcsec(X) = Log((Sqr(-X * X + 1) + 1) / X) |
| Inverse Hyperbolic Cosecant | HArccosec(X) = Log((Sgn(X) * Sqr(X * X + 1) + 1) / X) |
| Inverse Hyperbolic Cotangent （反双曲余切） | HArccotan(X) = Log((X + 1) / (X - 1)) / 2 |
| 以 N 为底的对数 | LogN(X) = Log(X) / Log(N) |

#### 4.4.3.3. 随机数

VBA 使用一个语句和一个函数来产生随机数，Randomize 语句初始化随机数生成器，其语法为：

```
Randomize [number]
```

Randomize 用 number 将 Rnd 函数的随机数生成器初始化，该随机数生成器给 number 一个新的种子值。如果省略 number，则用系统计时器返回的值作为新的种子值。

如果没有使用 Randomize，则（无参数的）Rnd 函数使用第一次调用 Rnd 函数的种子值。

Rnd 函数返回一个包含随机数值的 Single。其语法为：

```
Rnd[(number)]
```

可选的 number 参数是 Single 或任何有效的数值表达式。

其返回值决定于 number。

表 4-9 Rnd 函数的 number 参数和结果

| 如果 number 的值是 | Rnd 生成 |
|---|---|
| 小于 0 | 每次都使用 number 作为随机数种子得到的相同结果。 |
| 大于 0 | 序列中的下一个随机数。 |
| 等于 0 | 最近生成的数。 |
| 省略 | 序列中的下一个随机数。 |

Rnd 函数返回小于 1 但大于或等于 0 的值。number 的值决定了 Rnd 生成随机数的方式。

如果给定种子，则生成相同的数列，因为每一次调用 Rnd 函数都用数列中的前一个数作为下一个数的种子。在调用 Rnd 之前，先使用无参数的 Randomize 语句初始化随机数生成器，该生成器具有根据系统计时器得到的种子。

为了生成某个范围内的随机整数，可使用以下公式：

```
Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

这里，upperbound 是随机数范围的上限，而 lowerbound 则是随机数范围的下限。

若想得到重复的随机数序列，在使用具有数值参数的 Randomize 之前直接调用具有负参数值的 Rnd。使用具有同样 number 值的 Randomize 是不会得到重复的随机数序列的。

以下代码生成具有相同序列的随机数:

```
dummy = Rnd(-2)           '初始化
For i = 1 To 10
    Debug.Print Int(Rnd * 6) + 1
Next
```

## 4.4.4. Excel 数据表函数

### 4.4.4.1. 使用 Excel 函数

Excel 内置了大量可以处理数据的函数，包括基本的运算、统计以及金融计算等函数，这些函数大多都可以通过 Application.WorksheetFunction 的形式使用。关于那些函数可以通过 VBA 使用，可以参考 VBA 文档和附录的“可用于 Visual Basic 的工作表函数列表”。

Excel 工作表函数接受的参数基本和 VBA 的数据类型对应，对于输入是一个单元格范围的情况，例如“A5:B12”，则调用的时候参数可以是为 Range 对象，也可以是数组，而其输出，如果是数组函数，则返回一个以 1 为底的数组。

例如表 4-10 即为一些常用的 Excel 函数，可以在 VBA 代码中使用。

表 4-10 一些常用 Excel 函数

| 函数 | 说明 |
|---|---|
| AVERAGE | 计算平均数 |
| COUNT | 计算 Range 对象单元格的个数 |
| COUNTA | 计算 Range 对象非空单元格的个数 |
| COUNTBLANK | 计算 Range 对象空单元格的个数 |
| COUNTIF (range, criteria) | 根据条件计算 Range 对象单元格的个数 |
| MAX | 返回最大值 |
| MEDIAN | 返回中值（中位数） |
| MIN | 返回最小值 |
| MODE | 出现频率最多的数值 |
| STDEV | 返回标准偏差 |
| SUM | 求和 |
| SUMIF(range, criteria) | 条件求和 |

例如以下代码使用了这些函数来求和，计算平均值、中值、标准偏差等。

```
Dim a(5) As Double
Dim i As Long
For i = 0 To 5
```

```
    a(i) = Rnd
Next i

With Application.WorksheetFunction
    Debug.Print .Sum(a)
    Debug.Print .Average(a)
    Debug.Print .Median(a)
    Debug.Print .StDev(a)
    Debug.Print .Count(a)
    Debug.Print .Max(a)
    Debug.Print .Min(a)
End With
```

在开发中，使用 Excel 的内置函数可以大大提高开发效率，因此在完成一个任务，编写一个算法的时候，最好参考一下 VBA 文档的“可用于 Visual Basic 的工作表函数列表”，看是不是有可以直接使用的函数。

#### 4.4.4.2. 高级 Excel 函数

应用 Excel 函数可以大大加快开发速度，而且其运算效率一般也远远高于 VBA 代码。下面将对统计和矩阵运算的 Excel 数据表函数做一介绍。

**统计函数**

Excel 有大量统计函数，用于工作表运算和 VBA 代码可以简化相关的工作，下面列出了部分可能会经常用到的 Excel 统计函数（表 4-11）。

表 4-11 一些常用 Excel 统计函数

| 函数 | 描述 |
|---|---|
| AVEDEV | 返回数据点与它们的平均值的绝对偏差平均值 |
| AVERAGE | 返回其参数的平均值 |
| AVERAGEA | 返回其参数的平均值，包括数字、文本和逻辑值 |
| CONFIDENCE | 返回总体平均值的置信区间 |
| CORREL | 返回两个数据集之间的相关系数 |
| COUNT | 计算参数列表中的数字个数 |
| DEVSQ | 返回偏差的平方和 |
| GEOMEAN | 返回几何平均值 |
| HARMEAN | 返回调和平均值 |
| INTERCEPT | 返回线性回归线的截距 |
| KURT | 返回数据集的峰值 |

| | |
|---|---|
| LARGE | 返回数据集中第 k 个最大值 |
| LINEST | 返回线性趋势的参数 |
| MAX | 返回参数列表中的最大值 |
| MEDIAN | 返回给定数值集合的中值 |
| MIN | 返回参数列表中的最小值 |
| SLOPE | 返回线性回归线的斜率 |
| STDEV | 估算基于给定样本的标准偏差 |
| STDEVP | 计算基于给定的样本总体的标准偏差 |
| STEYX | 返回通过线性回归法计算每个 x 的 y 预测值时所产生的标准误差 |
| TREND | 返回沿线性趋势的值 |

对于表 4-11 所列的这些函数，有些在使用 Excel 函数一节已做了介绍，我们可以在 VBA 代码中直接使用，例如计算 2 个数据集的相关系数：

```
#001    Dim a(10) As Double, b(10) As Double
#002    Dim i As Long
#003
#004    '相关性不确定
#005    For i = 0 To 10 Step 1
#006        a(i) = Rnd
#007        b(i) = Rnd
#008    Next i
#009    Debug.Print Application. _
#010        WorksheetFunction.Correl(a, b)
#011
#012    '正相关：1
#013    For i = 0 To 10 Step 1
#014        a(i) = Rnd
#015        b(i) = a(i) * 3
#016    Next i
#017    Debug.Print Application. _
#018        WorksheetFunction.Correl(a, b)
#019
#020    '负相关：-1
#021    For i = 0 To 10 Step 1
#022        a(i) = Rnd
#023        b(i) = -a(i)
#024    Next i
#025    Debug.Print Application. _
#026        WorksheetFunction.Correl(a, b)
```

以上代码，构造了 3 组数据，分别调用 Correl 计算其相关系数。Correl 的参数是 2 个

维数一致的 Range 对象或者一维数组，函数返回二者的相关系数。

下面看一个计算线性回归的例子，使用的函数是 LINEST，要回归的表达式为 y = a x + b，其语法为：

```
LINEST(y, x, const, stats)
```

其中 y 为已知的 y，为一维数组。

其中 x 为已知的 x，为一维或者多维数组，代表要回归的表达式的 x 的值。

Const 为一逻辑值，用于指定是否将常量 b 强制设为 0。

Stats 为一逻辑值，指定是否返回附加回归统计值，具体内容可参考 Excel 文档。

返回根据参数不同，分别返回各个系数，截距和误差分析数据。

例如以下为计算随机构造的 2 组数据的线性回归的例子：

```
#001    Dim x(10) As Double, y(10) As Double
#002    Dim i As Long
#003
#004    For i = 0 To 10 Step 1
#005        x(i) = i
#006        y(i) = Rnd
#007    Next i
#008
#009    Dim t
#010    t = Application. _
#011        WorksheetFunction.LinEst(y, x, , False)
#012    Debug.Print t(1), t(2)
#013    For i = 0 To 10 Step 1
#014        Debug.Print t(1) * x(i) + t(2), y(i)
#015    Next i
```

首先定义变量（1 行），然后随机构造数据（4 到 7 行），然后计算其回归参数（10 行），由于是 x 是 1 维，所以返回 2 个值，为一以 1 为底的数组，第一个元素是斜率 a，第二个元素是截距 b，打印之（12 行），最后打印计算的 y 值和实际的 y 值（13 到 15 行）。

下面是 2 个 x 变量的回归估计，表达式为 y = a1 * x1 + a2 * x2 + b，代码如下：

```
#001    Dim x(1, 100) As Double
#002    Dim y(100) As Double
#003    Dim i As Long
#004
#005    For i = 0 To 100 Step 1
#006        x(0, i) = i
#007        x(1, i) = Rnd
#008        y(i) = Rnd
```

```
#009    Next i
#010
#011    Dim t
#012    Dim yy As Double
#013    t = Application. _
#014        WorksheetFunction.LinEst(y, x, , False)
#015    Debug.Print t(1), t(2), t(3)
#016    For i = 0 To 100 Step 1
#017        yy = t(1) * x(1, i) + t(2) * x(0, i) + t(3)
#018        Debug.Print y(i), yy, yy - y(i)
#019    Next i
```

代码和上面的没有太大差别，需要说明的是 x 是一个 2*101 的数组，返回的系数是从后向前的，也就是 a2 在前边，是返回数组的第一个元素，a2 在后边（17 行）。

Excel 的统计函数很多，语法、参数和返回值也很复杂，使用时需要仔细阅读其说明，并通过数据表测试分析后再在代码中使用，以避免误用。

**矩阵运算**

Excel 有几个可以在 VBA 中使用的矩阵运算函数。

MINVERS，返回数组矩阵的逆距阵。其语法为：

```
MINVERSE(array)
```

其中 Array 是具有相等行数和列数的数值数组。

MMULT，返回两数组的矩阵乘积。结果矩阵的行数与 array1 的行数相同，矩阵的列数与 array2 的列数相同。其语法为：

```
MMULT(array1,array2)
```

其中 array1, array2 是要进行矩阵乘法运算的两个数组。Array1 的列数必须与 array2 的行数相同，而且两个数组中都只能包含数值。

MDETERM，返回一个数组的矩阵行列式的值。其语法为：

```
MDETERM(array)
```

其中 array 是行数和列数相等的数值数组。

下面我们看一个例子：

```
#001    Dim x(1 To 3, 1 To 3) As Double
#002    Dim i As Long
#003    Dim j As Long
#004
```

```
#005    For i = 1 To 3
#006        For j = 1 To 3
#007            x(i, j) = Rnd
#008        Next j
#009    Next i
#010
#011    Dim t, v
#012    t = Application.WorksheetFunction.MInverse(x)
#013    v = Application.WorksheetFunction.MMult(x, t)
#014
#015    Debug.Print "原矩阵："
#016    For i = 1 To 3
#017        Debug.Print x(i, 1), x(i, 2), x(i, 3)
#018    Next i
#019
#020    Debug.Print "逆矩阵："
#021    For i = 1 To 3
#022        Debug.Print t(i, 1), t(i, 2), t(i, 3)
#023    Next i
#024
#025    Debug.Print "原矩阵 × 逆矩阵："
#026    For i = 1 To 3
#027        Debug.Print v(i, 1), v(i, 2), v(i, 3)
#028    Next i
```

代码首先定义了一个数组（矩阵），然后对其求逆，然后计算原矩阵和逆矩阵的乘积，结果为单位矩阵[17]。

对于其他 Excel 数据表函数，也可以使用以上方式在 VBA 代码中使用，以节省开发的工作量，提高效率，附录有所有可以在 VBA 代码中使用的 Excel 数据表函数的列表和简要说明。

## 4.5. 应用实例

以下实例是应用 Excel 进行数据处理的一些实际的例子，其中应用了本章前边所讲述的 Excel 和 VBA 的一些功能。每个例子最后的解决方案不一定是最好的，但每个例子都包

[17] 由于为数值计算，最后的有些结果可能不是 0 或者 1，而是一个接近于 0 或者 1 的值，例如，4.44089209850063E-16，这是浮点运算的限制，不是计算错误。

含了应用 VBA 在 Excel 中进行数据处理和解决问题的思路、方法。实例的讲述采用了逐步求解的讲述方式，以便真实的再现问题的解决过程。

## 4.5.1. 实例 1：在 Excel 中应用 VBA 批量导入数据

### 4.5.1.1. 问题描述

实际工作中，经常需要把一些数据（几十或者几百个数据文件），例如实验数据，导入 Excel 汇总，手工完成费时费力且容易出错，通过录制宏可以部分解决问题，但不是很完美，下面我们看使用 Excel 和 VBA 来如何一步步完成这项工作。

### 4.5.1.2. 通过录制宏导入数据

本例中的数据文件是仪器测试生成一系列文本格式的数据文件，格式完全一样，目的是要把每个数据文件导入到 Excel 中作为一条记录，也就是一行。我们可以使用 VBA 代码通过 File 对象读取每个文件，也可以使用 Excel 对象的功能。对于 VBA 编程，应该时刻牢记：“万不得已不要写代码，尽量使用 Office 的功能”。

数据包括 2 部分，第一部分是文件头，包括一些数据信息，后面是按行放置的数据，包括结果和误差，需要的是后边的数据，要把每行的数据和误差放置到相邻的两列（见下图）。可以使用 Excel 打开这个文本文件，按照弹出的文本文件导入向导对话框的步骤，使用固定列宽导入了需要的数据。

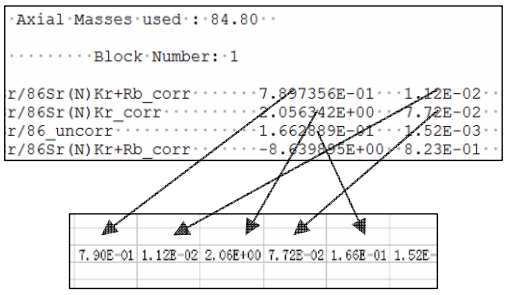



图 4-8 需要导入 Excel 的数据的格式

明白了问题，一切就好办了，打开 Excel，然后开始录制宏：首先打开文件，通过导

入文本文件向导，读入数据，将特定单元格的数据拷贝到一个目标 Excel 文件中，然后关闭这个文本文件，停止录制宏。

录制的宏很长，大概包括 2 部分。第一部分是一句打开文件，格式转换的操作，后边一部分是激活不同的文件，拷贝和粘贴不同的 Range。这时可以删除掉刚才拷贝进来的数据，运行了一下这个宏，测试是否把需要的数据导入进来了。

下一步应该修改宏导入成批数据。这批数据文件名是是“r20041124001357.txt”、“r20041124001358.txt”、“r20041124001359.txt”、“r20041124001360.txt”之类，是时间加序列组成的。因此可以写一个循环就批量导入。

打开了刚才录制的宏，这批数据文件名最后 2 位从 46 到 89，可以写一个 i 从 1 到 44 的循环，把读入文件部分的文件名改为：

"r200411240013" & ( i + 45 ) & ".txt"

把粘贴目的地（Range）表示行数的数字用 i 替换。

OK，按下了执行按钮，每次关闭文件的时候，有一个讨厌的是否保存文件的对话框跳出来，其他好像一切正常，还好，点击了 44 次鼠标后[18]，得到了需要的数据。

### 4.5.1.3. 修改 VBA 代码实现一个可通用的宏

**指定要导入的文件**

我们可以更改循环以导入不同的文件的数据，也可以通过一个打开文件对话框来指定需要的文件。通过一个打开文件对话框，选择一系列文件，然后将文件全路径存入一个集合或数组，然后循环读出这些文件就可以了。

先创建了一个窗体，然后放置了一个按钮，将 CommonDialog 控件引入工程，添加到窗体，在按钮的点击事件里加入如下代码：

```
Dim strFiles As String, i As Long

With CommonDialog1
    .Flags = &H200& Or &H80000 '可以选择多个文件
    .ShowOpen

    If .FileName <> "" Then
```

[18] 可以通过 Sendkeys 语句，可以模拟键盘操作关闭这个对话框，也可以使用 Workbooks(File).Close (False) 关闭，False 表示不需要保存文件。

```
            strFiles = .FileName
        End If
    End With

    '分割返回值，返回值为以 ASCII 码为 0 的分割的字符串
    '字符串第一个为路径，之后为单个文件名
    Files = Split(strFiles, Chr(0))
    For i = 1 To UBound(Files) Step 1
        '连接路径和文件名，组成文件数组
        Files(i) = Files(0) & "\" & Files(i)
    Next i
```

代码不多，最后的文件列表保存在 Files 数组里。使用 CommonDialog 控件打开多个文件，返回的多个文件的分割符号是 ASCII 码为 0 的字符。后面的部分就简单了。

```
For i = 1 To UBound(Files) Step 1
    strFilename = Files(i)
    DoImport strFilename
Next i
```

把原来的宏修改后保存在 DoImport 这个过程里，传入文件名即可导入这个文件，循环导入所有文件就可以了。虽然程序功能复杂了，但代码要有条理一些。

**指定要导入的位置**

可以指定导入的文件，那么也应该可以指定从第几行开始导入。给窗体加一个 RefEdit，点击开始的区域后返回的将是一个引用位置的字符串，使用 Range 函数得到该区域的引用对象（Range 对象），然后就可以得到其开始行数：

```
Range(Me.RefEdit1.Value).Row
```

重构一下 DoImport 这个过程，增加一个 mRow 参数，将导入的数据全部写到第 mRow 行。上面的调用过程就变成了：

```
Dim mRow as long
mRow = Range(Me.RefEdit1.Value).Row
For i = 1 To UBound(Files) Step 1
    strFilename = Files(i)
    DoImport strFilename, mRow
    mRow = mRow + 1
Next i
```

#### 4.5.1.4. 总结

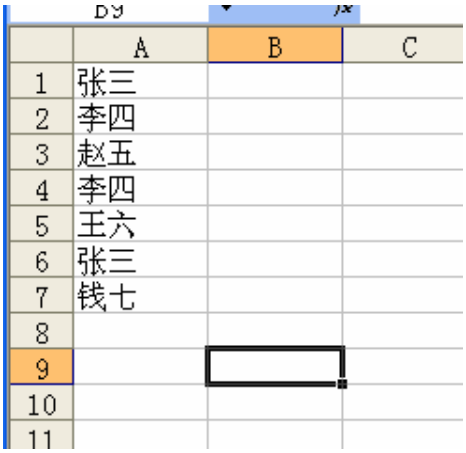
这样，我们就完成了这个任务。由此，我们可以确认，应用 Excel 可以完成很多 Copy、Paste 此类的事情，而且，通过录制并修改、扩展宏代码是进行 VBA 编程很好的起点。

### 4.5.2. 实例 2：在 Excel 中使用 VBA 来筛选数据

#### 4.5.2.1. 问题描述

这个问题是笔者解决过的一个真实的问题，而且此问题也有通过函数的方式来解决的方案，但 VBA 的优势在于一个思路可以解决很多类似的问题。另一方面，此问题涉及了算法设计、Excel 对象的效率[19]等问题，具有较好的参考价值和教学意义。

问题是这样的，一个很大的 Excel 文件，其中有些行是重复的，也就是说，有 2 行是完全一样的，有些行是不重复的，现在的问题是要找出所有不重复或者重复的行（图 4-9）。



| | A | B | C |
|---|---|---|---|
| 1 | 张三 | | |
| 2 | 李四 | | |
| 3 | 赵五 | | |
| 4 | 李四 | | |
| 5 | 王六 | | |
| 6 | 张三 | | |
| 7 | 钱七 | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |

图 4-9 求解重复行的 Excel 表格

现在的问题如图 4-9，图中“张三”、“李四”有 2 个，其他只有一个，需要把他们找出来。我们可以将重复的行在 B 列标记为 1，这样就可以达到目的，所以问题简化为：遍历所有行，对于每一行，顺序查找所有的行，看是否有其他行内容与其相同。

#### 4.5.2.2. VBA 程序

打开 VBA 编辑器，插入一个模块，使用以下代码可以达到我们的目的：

[19] 本书“其他话题”一章专门有一节讲述效率的问题。

```
Sub SelectDouble()

    Dim i As Long, j As Long

    For i = 1 To 7 Step 1
        For j = 1 To 7 Step 1
            '不比较相同的行
            If i <> j Then
                If Range("A" & i).Value = _
                       Range("A" & j).Value Then
                   Range("E" & i).Value = 1
                End If
            End If
        Next j
    Next i

End Sub
```

点击运行，如果内容很少，重复的行在 B 列都标志了 1，没有重复的空着，然后排序就可以了。

#### 4.5.2.3. 效率问题

以上代码没有考虑效率问题。读者可以试着把循环改到 1 到 10000，等着计算结果。几分钟过去后，也许会觉得有些奇怪，通过“Ctrl + Break”，暂停程序，将鼠标放在 i 变量上，显示 i 的值，大概只有几十。完全完成计算在笔者的计算机上需要 2-3 个小时，原因在于 Range 函数太慢。

那么，如何解决这个问题呢，我们提供了以下方案，可以适合不同的情况。

#### 4.5.2.4. 通过数组

数组要比 Range 函数快一些，定义 2 个数组，首先把数据全部读入第一个数组，然后对数组进行操作，对于重复的，把第二个数组的相应部分写为 1，计算完成后，根据第二个数组，把结果写回去。程序代码如下：

```
Sub SelectDouble2()

    Dim i As Long, j As Long
    Dim max As Long
```

```
    Dim a() As String, b() As Long

    max = 10000

    ReDim a(max) As String
    ReDim b(max) As Long

    For i = 1 To max Step 1
        a = Range("A1:A" & max).Value
    Next i

    For i = 1 To max Step 1
        For j = 1 To max Step 1
            '不比较相同的行
            If i <> j Then
                If a(i) = a(j) Then
                    b(i) = 1
                End If
            End If
        Next j
    Next i

    For i = 1 To max Step 1
        Range("F1:F" & max).Value = b(i)
    Next

End Sub
```

此时计算可以在 1 到数分钟内完成，效率可以满足大多数需求，但一个问题是内存占用太大，两个数组会占用很大的内存空间，本例涉及的数据不是很大，如果很大，则会造成一定问题。

#### 4.5.2.5. 使用内置函数

在 VBA 中，可以使用 Application.函数名，调用 Excel 的内置函数。这样，改过的代码如下：

```
Sub SelectDouble3()

    Dim i As Long, j As Long, a, b

    For i = 2 To 9999 Step 1
        a = Application.VLookup(Range("A" & i), _
```

```
            Range("A1:B" & (i - 1)), 2, False)
        b = Application.VLookup(Range("A" & i), _
            Range("A" & (i + 1) & ":B1000"), 2, False)
        If IsError(a) And IsError(b) Then
            Range("G" & i).Value = 0
        End If
    Next i

End Sub
```

代码很短，但有一点问题，循环是从 2 到 9999，因为为了防止 VLOOKUP 函数的 Range 范围失效，所以这两行需要手动处理。IsError 函数来检测返回值，如果两个返回值都是错误，则此行为单一的没有重复的行，标志为 0 即可。

程序执行速度在笔者计算机上比上面的稍微慢一些，但也很快，而且无需分配多余的数组，没有内容大小的限制。

#### 4.5.2.6. 改进算法

这样，我们已经解决了问题，那么是否还有改进的余地和研究的必要吗？学习过数据结构的读者都知道二分查找，对于大数据量，速度可以大幅度提高，前提是数据要排序。

对于我们这个例子，如果数据排好了序，则只需要检查当前数值和下一个是否一样，如果一样，那么把当前和下一个位置标示出来，循环变量加 2，跳过下一个，如果不一样，循环变量加 1 继续比较就可以了，代码如下：

```
Sub SelectDouble4()

    Dim i As Long, Max As Long

    Max = 10000
    i = 1
    Do
        If Range("A" & i).Value = _
               Range("A" & (i + 1)).Value Then
            Range("I" & i).Value = 1
            Range("I" & (i + 1)).Value = 1
            i = i + 2
        Else
            i = i + 1
        End If
    Loop While i < Max
```

```
End Sub
```

这个程序复杂度只有 n，而前边的程序复杂度都是 $n^2$。因此，这种方法的程序执行速度也相当快，内存占用也小。有些满的原因主要是 Range 对象，如果必要，可以使用数组方法，加快其速度。

#### 4.5.2.7. 总结

所以，对于一个问题的解决，需要从多个方面去考虑和钻研，寻求一种好的途径（例如不对每个单元格使用 Range 对象），设计一个好的算法才是关键。在实际工作中，需要不断思考和改进，才可以得到优秀的解决方案。

### 4.5.3. 实例 3：如何在多个文件中查找需要的信息

#### 4.5.3.1. 问题描述

上面的 2 个例子，侧重点不同，叙述了使用 Excel 和 VBA 进行数据处理的不同方面，下面这个例子则稍微复杂一些，但使用的技术和方法都是前边使用过的，因此叙述过程中主要侧重于思路和关键点。

实际工作中有时需要在一系列的文件中查找需要的信息，例如在很多文件中进行批量的查找替换等等，或者在很多文件中查找需要的信息汇总起来。下面的例子需要解决的问题就是这样一个问题。有一系列相同格式的 2 进制文件，压缩保存在不同的文件中，此二进制文件可以应用一个通用程序转化为相同格式的文本文件（如下所示）；

###### ###### ###### ###### ######

###### ###### ###### ###### ######

…

现在需要编制一个程序从这些文件中查找符合某些条件的行，然后保存在另一个文本文件中。

#### 4.5.3.2. 解决思路

对于这样的问题，笔者在学习 Excel 和 VBA 之前，一般会自然而然的想着使用 C 或

者 VB 开发，考虑如何设计一个文件读取功能，如何将文件内容保存在一个数组或者其他数据结构中，如何设计查找算法，等等。

某些读者或许也会考虑使用一个简单的桌面数据库，将数据全部读入数据库，然后再进行操作。

其实使用 Microsoft Excel + VBA 可以快速简洁的实现这类功能，应用 Microsoft Excel + VBA 实现所需要的功能具有如下优势[20]：

1. 文件读取、格式化等操作可以交给 Excel 去做；
2. 数据可以自然的保存在 Excel 的电子表格中，无需考虑复杂的数据结构；
3. 查询可以调用 Excel 的方法，也可以自己写一个简单的顺序查找。

使用 Excel 完成此类任务，可以有两种方法，或者使用 VB 或者其他支持 COM 自动化的程序调用 Excel，或者在 Excel 中使用 VBA 编写需要的代码，在 Excel 下执行。前者可以在其他程序中嵌入使用，而 Excel 可以完全在后台执行，后者无需其他程序开发环境，选取完全取决于要求。对于如何使用 VB 或者其他支持 COM 自动化的程序调用 Excel，在本书“其他话题”一章有专门介绍。

因此，本程序的流程大概为：

1. 调用外部程序解压缩并转化相应的文件；
2. 逐个打开转换后的文本文件，保存在 Excel 数据表中；
3. 在 Excel 数据表中查找需要的内容，如果找到需要的内容，则拷贝到需要的位置；
4. 关闭本文件，到第 2 步；
5. 完成操作，清理临时文件等内容。

### 4.5.3.3. 关键技术

本程序实现的关键技术及其代码如下所述。

**调用外部程序**

在 VBA 中，调用外部程序的函数为“Shell”。

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

[20] 对于本例，使用 Excel 具有优势，如果是一般的文本文件，使用 Word＋VBA 可能更有优势。

执行一个可执行文件，返回一个 Variant (Double)，如果成功的话，代表这个程序的任务 ID，若不成功，则会返回 0。Shell 函数是以异步方式来执行其它程序的。也就是说，用 Shell 启动的程序可能还没有完成执行过程，就已经执行到 Shell 函数之后的语句。

对于需要同步执行的程序，即等待调用的程序执行完毕，再执行下一步，可以使用以下代码：

```
Private Declare Function WaitForSingleObject Lib "kernel32"
_
   (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As
Long

Private Declare Function CloseHandle Lib "kernel32" _
   (ByVal hObject As Long) As Long

Private Declare Function OpenProcess Lib "kernel32" _
   (ByVal dwDesiredAccess As Long, ByVal bInheritHandle As
Long, _
    ByVal dwProcessId As Long) As Long

Private Const INFINITE = -1&
Private Const SYNCHRONIZE = &H100000

Private Sub ShellTest()
    Dim iTask As Long, ret As Long, pHandle As Long
    iTask = Shell("notepad.exe", vbNormalFocus)     '开始执行
    pHandle = OpenProcess(SYNCHRONIZE, False, iTask)
    ret = WaitForSingleObject(pHandle, INFINITE)
    ret = CloseHandle(pHandle)                      '执行结束
    MsgBox "Process Finished!"
End Sub
```

**文件操作**

程序使用的文件操作包括如何打开文件、关闭文件，在打开的文件之间切换等操作。打开文件使用：

```
Application.Workbooks.OpenText        '用于打开文本文件
Application.Workbooks.Open            '用于打开 Excel 文件
```

对于文件打开操作，可以使用宏录制功能，以便实现格式转化，定制列的格式等操作。关闭文件使用：

```
Application.Workbooks(文件名).Close       '关闭文件
```

Close 方法参数可以是 True 或者 False，分别表示是否保存文件。

切换当前文件：

```
Application.Workbooks(文件名). Activate
```

需要注意的是，使用 Workbooks 集合返回 Workbook 对象时，使用的文件名必须是不包括路径但包括扩展名的文件名，例如“abc.txt”，或者“xyz.xls”。

**查询操作的实现**

逐个打开各个文件，就可以进行查询操作了，查询结果可以写入一个临时的 Excel 文件，然后保存之。

查询操作可以调用 Excel 的函数实现，Excel 有数十个查找引用函数，例如 HLookUp，VLookUp 等。

对于本例，可以编写一个顺序查找，要获取各个单元格的值，最简单的方法就是使用 Range 对象，例如，Range(“A1”).Value，返回当前工作表活动 Sheet，单元格 A1 的值。对于有多个打开工作表的情况，可以使用 Windows(文件名).Sheets().Range，或者 Workbooks(文件名). .Sheets().Range 来获得 Range 对象。具体操作和上个例子类似，在此就不重复了。

一般来说，可以直接调用 Excel 函数和功能的，最好调用其实现，因为效率和速度都相对比使用 VBA 完成的代码要好。

**编写界面**

在 Excel 下，一个 VBA 程序的发布形式有 2 种，基于 Excel 文件发布或者发布为一个 Excel 加载宏程序。对于应用界面，可以在 Excel 已有的菜单、工具条上增加新的内容，也可以直接在 Excel 图表中增加控件，或者使用自定义窗体，不同的方式有不同的应用领域[21]。

对于增加的工具条和按钮等界面内容，一般的编程方式是，当你的加载宏或者 Excel 文件打开运行时，将界面内容增加到 Excel 中，而结束或退出时，删除其内容，因此相关代码一般写在 VBA 项目（工程）的 ThisWookbook 对象下，在此对象中，选择 Workbook_Open，以及 Workbook_BeforeClose 事件，在其中添加工具按钮的增加，删除操作；对于加载宏，有 Workbook_AddinInstall，Workbook_AddinUninstall 事件。

[21] 参见“界面设计”和“其他话题”2 章的相关内容。

### 4.5.3.4. 主要代码

**解压和转换文件**

解压和转换文件可以使用 Shell 函数来完成，首先需要解压文件，代码如下：

```
Public Sub Extract(strpath As String)
    Shell strpath & "\Extract.exe e data.bin", vbNormalFocus
End Sub
```

其中的 Extract.exe e data.bin 为在命令行下执行此程序的命令行。解压此文件后，需要逐个转换并将转换后的文件保存在一个文本文件中以备之后查找。

```
#001 Public Sub GetFiles(strpath)
#002
#003    Dim MyPath As String, MyName As String
#004    Dim i As Long
#005
#006    MyPath = strpath & "\*.d"    ' 指定路径。
#007    MyName = Dir(MyPath)    ' 找寻第一项。
#008    Do While MyName <> ""    ' 开始循环。
#009        Shell strpath & "\cvt.exe " & MyName
#010        Files.Add Left(MyName, Len(MyName) - 2) & "txt"
#011        MyName = Dir    ' 查找下一个目录。
#012    Loop
#013
#014 End Sub
```

以上函数转换一个目录下所有的文件并将其转换后的文件保存在全局集合变量 Files 中。第 6 行指定了需要转换的文件路径，及其通配符，即以“d”为后缀的文件；然后使用 Dir 函数循环整个文件夹的所有以“d”为后缀的文件（7-12 行）；第 9 行则调用“cvt.exe”来转换文件为文本文件，转换之后将转换后的文件加入集合变量 Files。

这样，就完成了文件的解压和转换，并将其保存在一个集合变量里。

**内容查找**

内容查找代码如下所示：

```
#001 Public Sub FindInFiles(number As Long, strPath As String)
#002
#003    Dim i As Long, j As Long, k As Long
#004
#005    For i = 1 To Files.Count
#006        OpenFile strPath & "\" & Files.Item(i)
```

```
#007        FindNumber Files.Item(i), number
#008        Workbooks(Files.Item(i)).Close (False)
#009    Next i
#010
#011 End Sub
```

代码循环 Files 集合，首先打开文件（第 6 行，使用 OpenFile 过程，为自定义的该格式的文本文件打开过程，可以通过录制宏完成），然后调用 FindNumbers 过程来查找单个文件（7 行），最后关闭这个文件。

下面是 FindNumbers 过程的实现代码：

```
Private Sub FindNumber(strFile As String, number As Long)

    Dim j As Long, k As Long
    Dim strNumber  As String
    k = 1

    For j = 1 To 65535
        strNumber = _
            Windows(strFile).ActiveSheet.Range("C" & j).Value
        If strNumber = "" Then
            Exit For
        End If
        If Windows(strFile).ActiveSheet.Range("C" & j). _
            Value = number Then

            Windows("Result.xls").ActiveSheet. _
                Range("A" & k & ":D" & k).Value = _
                Windows(strFile).ActiveSheet. _
                Range("B" & j & ":E" & j).Value

            k = k + 1
        End If
    Next j

End Sub
```

代码循环整个文件，查找与 number 相同的值。

**界面和以上代码的整合**

界面编写在关键技术已做了简单介绍，这里不再重复，其所述的技术在后面的界面设计中还要介绍。完成界面后，需要注意的是，以上代码执行的时候，因为 Shell 函数是异

步调用，因此必须分步调用，笔者设计了不同的按钮，分别做解压、转换和查找，以避免造成错误（图 4-10）；也可以使用上面介绍的 API 函数，设计异步调用的程序。

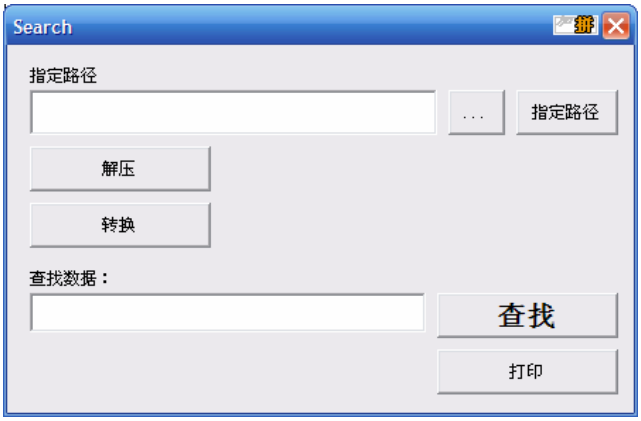



图 4-10 文件批量查找对话框

#### 4.5.3.5. 总结

这样，我们就完成了一个完整的应用。使用 VBA 和 Excel 实现在多个文件中查找所需要的数据，相对于使用其他编程语言直接编写实现，要高效迅速的多，而且程序的可维护性也好很多。

### 4.5.4. 实例 4：批量重命名文件

#### 4.5.4.1. 问题描述

在进行科学研究和实际工作中，有时需要对大量的数据文件进行整理，例如一个常见的整理工作就是根据数据文件的内容来重命名文件，手工工作烦琐且容易出错，我们可以使用 Excel 和 VBA 来完成这个工作。

例如需要根据一系列文本文件的某个描述值来重命名文件，文本文件格式是固定的，这个值位于文本文件的特定的行和列。我们要将文件重命名为此值。

#### 4.5.4.2. 解决思路

问题解决的思路类似于实例 1，其流程为：

1. 首先需要得到需要重命名的文件，保存于一个数组或者集合；

2. 然后循环操作这个集合，取出每一个需要重命名的文件；
3. 对于每一个文件，使用 Excel 打开之，得到需要的新名字；
4. 使用这个名字应用 Name 命令重命名之。

### 4.5.4.3. 关键代码

**重命名文件**

重命名文件使用 Name 命令，其代码如下：

```
#001 Public Sub ReName(strPath As String, strName As String)
#002
#003    Dim strOldName As String
#004    Dim strNewName As String
#005
#006    strOldName = strPath & strName
#007
#008    Workbooks.OpenText Filename:= _
#009        strOldName, Origin:=936, _
#010        StartRow:=1, _
            DataType:=xlFixedWidth, _
            FieldInfo:=Array(0, 1), _
#011        TrailingMinusNumbers:=True
#012
#013    strNewName =  _
        Mid(Workbooks(strName).Sheets(1).Range("A9"), 60)

#014    Workbooks(strName).Close (False)
#015
#016    strNewName = strPath & "Results_" & strNewName
#017
#018    Name strOldName As strNewName
#019
#020 End Sub
```

这段代码定义了一个 ReName 函数，可以根据参数指定的路径和文件，打开相应的文件（8 行），然后从其第一列、第九行（A9）取出字符串，取其第 60 个字符之后的内容（13 行），然后使用 Name 语句重命名文件。需要注意的是，传入的参数是路径＋文件名，2 个参数，是为了易于操作。

打开文件我们说过，最好使用录制宏的方式生成基础代码，然后再修改以适合需求。

**得到需要的文件集合**

这个过程和实例 1 类似，唯一不同的是将文件存入数组时，仅存入文件名，路径则保存在 Files(0)：

```
Files = Split(strFiles, Chr(0))
```

**循环重命名**

然后就可以循环 Files 这个数组，重命名每个文件。

```
    For i = 1 To UBound(Files) Step 1
        ReName Files(0) & "\", Files(i)
    Next I
```

因为 Files(0)保存的是没有以“\”结尾的路径，因此参数可以使用以上代码传递。

### 4.5.4.4. 总结

本示例非常简单，不需要做额外的说明，因为使用 Excel 和 VBA 结合的方案，使问题的解决非常简单。另一方面，我们也可以发现，可以使用 Excel 和 VBA 做的很多工作是仅仅使用 Excel 无法完成的。这也就是 Excel 的魅力吧。

# 7. 其他话题

## 7.1. Excel VBA 程序的类型和部署

### 7.1.1. Excel VBA 程序的类型

应用 VBA 在 Excel 下开发的程序，其程序存储位置和部署形式是相关联的，使用 VBA 编写的 Excel 程序可以存储在两个位置[23]：（1）加载宏；（2）当前的 Excel 文件。而其部署或者发布也相应的可以通过单独的文件来发布，或者通过加载宏的方式来发布。

### 7.1.2. 加载宏和一般电子表格程序的优缺点

通过加载宏和一般电子表格存储和发布 Excel VBA 程序的优缺点，可简述如下：

1. 加载宏可以被很多文件使用，一般作为通用功能程序（工具类应用）发布，例如微软 Excel 自带的一些加载宏（分析工具库等）；
2. 位于文件内部的宏则往往与工作表内的数据互动比较多，需要使用当前定义好的工作表，适合特定应用，比如一个科学研究中专业过程的模拟，一个商业报表或者企业应用的前端；
3. 对于部分企业应用的前端和科学计算程序，由于其数据驱动的特征，使用电子表格制作其用户界面具有其他方法不具有的优越性；
4. 加载宏的界面必须通过给 Excel 增加按钮等方式来实现，而工作表形式的实现则可以在电子表格上增加按钮等控件，并充分利用工作表来实现，具有非常好的灵活性；
5. 对于工作表形式的程序，数据的输入获取直接通过电子表格的特定单元格获取，并且可以充分利用公式等工具，简化程序设计。
6. 默认情况下，加载宏没有安全提示，而电子表格没有数字签名则打开时有提示，

[23] 对于录制的宏，还可以保存在个人宏工作薄，实际上就是一个位于“C:\Documents and Settings\用户名\Application Data\Microsoft\Excel\XLSTART”目录下的名为“PERSONAL.XLS”的一个文件，Excel 启动时会自动加载并隐藏之。

如果安全性设置为高，则默认不可用，必须在安全性设置为中时才可以使用。

### 7.1.3. 部署

不管是加载宏程序还是一般的电子表格程序，其部署都很简单，我们可以直接拷贝这个文件到目标机，对于加载宏，需要在 Excel 中将其加载，对于后者，可以直接双击使用 Excel 打开运行。

而对于应用 VSTO 开发的 Office 应用，其程序的存储位置位于独立的文件和 DLL。另外，如前文所述，我们也可以将一些算法、代码通过 COM 对象封装在独立的 DLL 内，在 Excel 内通过 VBA 调用。对于此类代码，需要在部署前安装和注册相应的（COM）对象。

如果在 VBA 代码里使用了其他文件中的资源，如数据、配置信息等，在发布和部署时需要一起发布，对于这些文件资源的位置，可以通过 Application 对象获取当前文件或加载宏的所在目录，然后通过目录的相对路径来调用。

例如，当前文件的路径为：

```
Application.ActiveWorkbook.Path;
```

加载宏的路径则通过以下方法来获取：

```
Application.AddIns.Item([加载宏名称]).Path
```

对于有多个文件的程序包，可以使用安装工具制作安装包，也可以通过直接 Copy 的方式来部署。

## 7.2. VBA 程序的安全性和保护

VBA 程序可以通过在属性中设置密码，保护 VBA 代码的安全。通过*“工具 － 〔工程名〕属性”*或者右键单击工程资源管理器中的工程选择属性打开工程属性对话框，选择“保护”页，在此可以设置 VBA 工程的密码，可以保护让不知道工程密码的人无法查看此程序的 VBA 代码（图 7-1）。


图 7-1 工程属性对话框（保护工程）

在工程属性对话框中选中锁定工程，并在查看工程属性密码中输入密码，则每次打开此 Excel 文件或加载宏，查看或修改其工程时都需要输入密码，但不影响 VBA 程序的使用和运行。

由于 Excel 工程（一个文件，或者加载宏）可以通过添加另一个文件或加载宏的引用（reference）调用其中的公共过程、函数和公有类（见后文说明）；通过这种方式，在很多程序中就可以使用已有加载宏或者程序的模块，也可以通过这种方式来进行合作开发等等；但同样，由于此功能，可能会暴露一些关键的函数、类或者过程（例如加密、解密代码，注册代码）给其他人，造成不必要的麻烦。因此，对于一些关键性的、不希望别人破解或者利用的代码，一定要设置为私有（Private）。

对于某些关键性代码，例如程序注册、公司或者单位的关键技术等，也可以通过封装在 DLL 中，通过 COM 对象来调用，以增加安全性。

## 7.3. 自动化其他 Office 组件

程序间的交流从一般的数据共享到相互协作，会给实际的工作带来戏剧性的便利。现在的 Windows 程序，可以通过剪贴板和 OLE 嵌入等技术非常容易的共享数据。程序之间

的协作则从 Unix 的管道技术到 Windows 下的 DDE、OLE 以及 COM 自动化[24]，技术的众多正好说明了需求的旺盛。

由于 Office 的各个组件都将自己的组件模型通过 COM 暴露出来，可以使用 COM 自动化技术使用，这就为程序间的协作奠定了基础。在 VBA 开发中，经常需要协作 Office 的各个成员完成一件任务，例如一份文档，可能首先需要从 Access 中获取数据，在 Excel 中进行处理，在 Word 中根据模版生成报表，最后调用 Outlook 自动发送给需要的收件人。为什么需要使用不同程序一起来完成一件事呢？一方面，由于不同的应用程序，具有不同的功能和优势，例如 Excel 擅长数据处理，Word 擅长文字处理，而 Outlook 可以收发 Email，因此，在系统开发中，如果可以直接使用各程序的独特功能，则可以节省很多的开发时间；另一方面，这些成熟的软件其可靠性和稳定程度也优于自己草草实现的代码。

**Excel 工作簿和工作表的部分内部限制**

- 工作表大小：65,536 行，256 列
- 单元格内容（文本）：32,767 个字符
- 工作簿中的工作表数：受可用内存限制
- 工作簿中的名称数：受可用内存限制

因此，Excel 只适合保存中等数量的数据，较大的数据量应该保存在数据库系统，例如桌面数据库 Access，而使用 Excel 作为数据处理的工具。

下面，我们会一起学习如何在 Excel 中启动或激活其他 Office 组件，完成特定的任务。

## 7.3.1. 启动其他 Office 组件

我们可以在 Excel（或者其他程序设计语言）中使用 VBA 控制 Word，这时，Excel 称为 Client 程序，而 Word 是作为 Server 程序，这种在一种程序中控制另一种程序的技术被称为 COM 自动化（或者 OLE 自动化）。当然，我们也可以在 VB 中通过 COM 自动化控制 Excel 或者 Word（见下一节）。

通过 COM 自动化在 Excel 中控制 Office 的其他组件，首先需要创建目标组件的一个对象实例，我们可以通过以下 2 种方法创建需要的组件。

[24] 有关的技术还可以长长的列出，如 DCOM、.net 的 Remoting，Web Service 技术等等

我们可以用以下方式启动 Word，往其中输入部分内容，然后退出。

```
#001 Sub OpenWord()
#002
#003    Dim objWord As Object
#004    Dim objDoc As Object
#005
#006    On Error Resume Next
#007
#008    Set objWord = GetObject(, "Word.Application")
#009    If objWord Is Nothing Then
#010        Set objWord = CreateObject("Word.Application")
#011    End If
#012
#013    On Error GoTo 0
#014
#015    If objWord Is Nothing Then
#016        MsgBox "Word 启动错误!", vbExclamation
#017    End If
#018
#019    With objWord
#020        .Visible = True
#021        Set objDoc = _
#022            .Documents.Add
#023        objDoc.Content = "Test"
#024        objDoc.Close
#025        .Quit
#026    End With
#027
#028    Set objDoc = Nothing
#029    Set objWord = Nothing
#030
#031 End Sub
```

在这段程序中，第 8 行通过 GetObject 获取正在运行的 Word，如果不存在，则通过 CreateObject 方法创建一个 Word 对象，这种将对象定义为 Object 类型，然后通过 CreateObject 或者 GetObject 方法获取其实例的方法称为晚期绑定。19 行到 26 行通过操作 Word 的对象模型，新建一个文档（第 21 行），设置其内容（第 23 行），然后退出（24、25 行）。

另外一种启动其他 Office 组件的方法是在 VBA 的引用中添加其引用（图 7-2），然后即可象创建 Excel 对象一样使用 Dim，New 等关键字声明、创建对象。

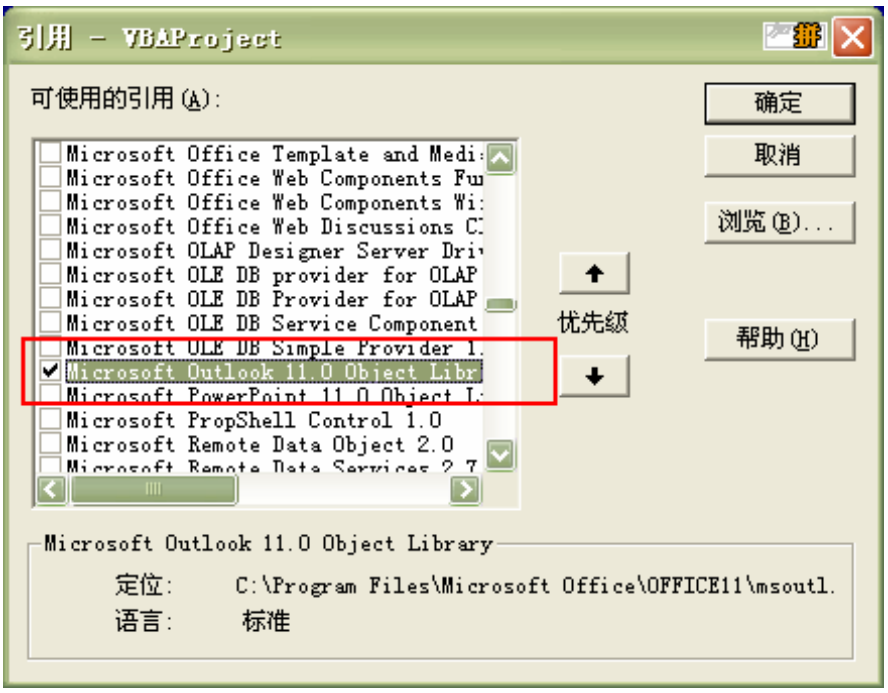



图 7-2 VBA 项目中添加对 Outlook 的引用

以下程序首先需要在工程中添加 Outlook 的引用（图 7-2），然后通过 Dim 和 New 创建需要的对象（3 到 7 行），在第 9 行创建了 Outlook 对象，在第 10 和 11 行获取了联系人列表，13 到 16 行循环读取联系人信息并写入电子表格，之后退出（18、19 行）[25]。

```
#001 Sub DisplayOutlookContactNames()
#002
#003     Dim objOutlook As Outlook.Application
#004     Dim objNameSpace As Outlook.Namespace
#005     Dim objAddresslist As AddressList
#006     Dim objEntry As AddressEntry
#007     Dim i As Long
#008
#009     Set objOutlook = New Outlook.Application
#010     Set objNameSpace = objOutlook.GetNamespace("Mapi")
#011     Set objAddresslist =
             objNameSpace.AddressLists("联系人")
#012
#013     For Each objEntry In objAddresslist.AddressEntries
#014         i = i + 1
#015         Cells(i, 1).Value = objEntry.Name
```

[25] 在运行这段程序时，会有一个对话框显示说明有一个程序访问 Outlook，询问是否允许访问，这是在 Office XP 之后增加的功能，以防止日益严重的电子邮件病毒。

```
#016     Next
#017
#018     objOutlook.Quit
#019     Set objOutlook = Nothing
#020
#021 End Sub
```

现在，我们知道，可以通过早期绑定（early binding）和晚期绑定（late binding）来创建 Office 组件的实例，那么这两种方法有什么优缺点呢。

1. 晚期绑定运行速度要比早期绑定慢一些；
2. 在编码阶段，使用晚期绑定无法使用 IntelliSense（自动列出成员和参数提示等），无法使用对象浏览器浏览对象的信息；
3. 晚期绑定与程序版本无关，例如开发时安装的是 Office 2000，程序在安装 Office XP 的计算机也可以正常运行，早期绑定则必须是相同的版本。
4. 实际开发中，可以在编码阶段使用早期绑定，在最后发布时将代码改为晚期绑定即可（始终使用 CreatObject 创建对象，将所有声明改为 Object 即可）。

## 7.3.2. 与其他 Office 组件交互

与其他 Office 组件交互的关键是熟悉其工作方式和组件模型，其他无他。

例如，我们可以在 Excel 设置一个计划表，如英语复习计划，课程表或者其他，由于 Excel 的自动填充，函数等功能，使其建立此类表格非常方便，而 Outlook 中，则可以添加任务，及时提醒我们。下面我们举一个这样的例子。Excel 的表格如图 7-3 所示，其中的日期和任务都是通过自动填充和自定义函数填充的。

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | 日期 | GRE词汇 | | | | 听力（1.5h） | 写作（1h） |
| 2 | | 初记 | | 复习内容 | 复习内容 | 每天1-1.5小时 | |
| 3 | 9月13日 | 1 | 25 | | | NCE(3) | TOEFL Topic 1 |
| 4 | 9月14日 | 26 | 50 | | | NCE(6) | TOEFL Topic 2 |
| 5 | 9月15日 | 51 | 75 | 9月13日 | | NCE(9) | TOEFL Topic 3 |
| 6 | 9月16日 | 76 | 100 | 9月14日 | | NCE(12) | TOEFL Topic 4 |
| 7 | 9月17日 | 101 | 125 | 9月15日 | 9月13日 | NCE(15) | TOEFL Topic 5 |
| 8 | 9月18日 | 126 | 150 | 9月16日 | 9月14日 | NCE(18) | TOEFL Topic 6 |
| 9 | 9月19日 | 151 | 175 | 9月17日 | 9月15日 | NCE(21) | TOEFL Topic 7 |
| 10 | 9月20日 | 176 | 200 | 9月18日 | 9月16日 | NCE(24) | TOEFL Topic 8 |
| 11 | 9月21日 | 201 | 225 | 9月19日 | 9月17日 | NCE(27) | TOEFL Topic 9 |
| 12 | 9月22日 | 226 | 250 | 9月20日 | 9月18日 | NCE(30) | TOEFL Topic 10 |

图 7-3 Excel 学习计划表

然后，我们在此电子表格中添加对 Outlook 的引用，新建一个模块，输入以下代码：

```
#001 Public Sub WriteToOutlookTask()
#002
#003     Dim Task As Outlook.TaskItem
#004     Dim rng As Range
#005     Dim i As Long
#006
#007     For i = 3 To 34 Step 1
#008       If Range("B" & i).Value <> "" Then
#009         Set Task = _
              Outlook.Application.CreateItem(olTaskItem)
#010         Task.Subject = "初记: " & _
              Range("B" & i).Value & "-" & _
              Range("C" & i).Value
#011         Task.StartDate = Range("A" & i).Value
#012         Task.DueDate = Range("A" & i).Value
#013         Task.Save
#014       End If
#015       If Range("D" & i).Value <> "" Then
#016         Set Task = _
              Outlook.Application.CreateItem(olTaskItem)
#017         Task.Subject = "复习: " & _
              Range("B" & i - 2).Value & _
              "-" & Range("C" & i - 2).Value
#018         Task.StartDate = Range("A" & i).Value
#019         Task.DueDate = Range("A" & i).Value
#020         Task.Save
```

```
#021      End If
#022      If Range("E" & i).Value <> "" Then
#023         Set Task = _
               Outlook.Application.CreateItem(olTaskItem)
#024         Task.Subject = "复习: " & _
               Range("B" & i - 4).Value & "-" & _
               Range("C" & i - 4).Value
#025         Task.StartDate = Range("A" & i).Value
#026         Task.DueDate = Range("A" & i).Value
#027         Task.Save
#028      End If
#030    Next i
#031
#032 End Sub
```

以上代码首先创建了需要的 Outlook 对象（第 3 行），第 7 到 30 行循环对 3-34 行的 Excel 数据表（图 7-3）进行处理，在此循环中，每个 If 语句块处理并添加一个任务。

## 7.4. 通过其他程序自动化 Excel

在其他应用程序中自动化 Excel 和在 Excel 中自动化其他 Office 组件是类似的，我们首先需要创建一个 Excel 的 Application 对象，对象创建可以使用早期绑定，也可以使用晚期绑定，之后通过 Excel 的 Application 创建其他对象，完成需要的工作。

### 7.4.1. 创建 Excel 对象

下面是一段 VB6 的代码，这段代码创建了一个 Excel Application 对象，然后将一些数据导出到 Excel 中。

```
#001 Dim objExcel As Object
#002 Dim objBook As Object
#003 Dim objSheet As Object
#004 Dim objRng As Object
#005 Dim i As Long, j As Long
#006
#007 'Start a new workbook in Excel
#008 Set objExcel = CreateObject("Excel.Application")
#009
#010 Set objBook = objExcel.Workbooks.Add()
#011 Set objSheet = objBook.Worksheets(1)
#012 Set objRng = objSheet.Range("A" & 1 & ":" & "J" & 20000)
```

```
#013
#014 Dim a(20000, 10) As Variant
#015
#016 For j = 0 To 20000 Step 1
#017     For i = 0 To 10 Step 1
#018         a(j, i) = CStr(j)
#019     Next i
#020 Next j
#021
#022 objRng.Value = a
#023
#024 objExcel.Visible = True
```

## 7.4.2. Excel 自动化中的事件

在利用 VB 或者 VB.net 进行 Office 自动化开发时，有时必须知道用户做了什么操作，例如切换打开的文件，是否关闭了 Excel 程序，改变单元格的内容等等。实际上，Office 对象模型中有大量的事件，可以精细控制到单元格的改变、Sheet 的切换、文件打开关闭、加载宏加载等等，通过使用这些事件，我们就可以知道打开的 Office 程序是否被关闭。

以下示例（以 VB.net 为例），新建一个工程，添加对 Excel 的引用，然后在 Form 中声明如下变量：

```
Dim WithEvents objExcel As Excel.Application
Dim objWorkBook As Excel.Workbook
```

打开或激活 Excel：

```
objExcel = New Excel.Application
objWorkBook = objExcel.Workbooks.Add
objExcel.Visible = True
```

然后就可以响应 Excel 对象的事件了，例如：

```
Private Sub objExcel_WorkbookBeforeClose(ByVal Wb _
     As Excel.Workbook, ByRef Cancel As Boolean) _
     Handles objExcel.WorkbookBeforeClose
   Me.lstEvent.Items.Add("将要关闭：" & Wb.Name)
End Sub

Private Sub objExcel_WorkbookBeforeSave(ByVal Wb _
      As Excel.Workbook, ByVal SaveAsUI As Boolean, _
      ByRef Cancel As Boolean) Handles _
      objExcel.WorkbookBeforeSave
   Me.lstEvent.Items.Add("保存：" & Wb.Name)
```

```
End Sub

Private Sub objExcel_WorkbookOpen(ByVal Wb _
      As Excel.Workbook) Handles objExcel.WorkbookOpen
   Me.lstEvent.Items.Add("打开: " & Wb.Name)
End Sub

Private Sub objExcel_SheetChange(ByVal Sh _
     As Object, ByVal Target As Excel.Range) _
     Handles objExcel.SheetChange
   Me.lstEvent.Items.Add("改变了: " & _
       CType(Sh, Excel.Worksheet).Name & " 的 " _
       & Target.Address)
End Sub
```

运行效果如图 7-4 所示，这样，我们就可以很好的和 Excel 交互。

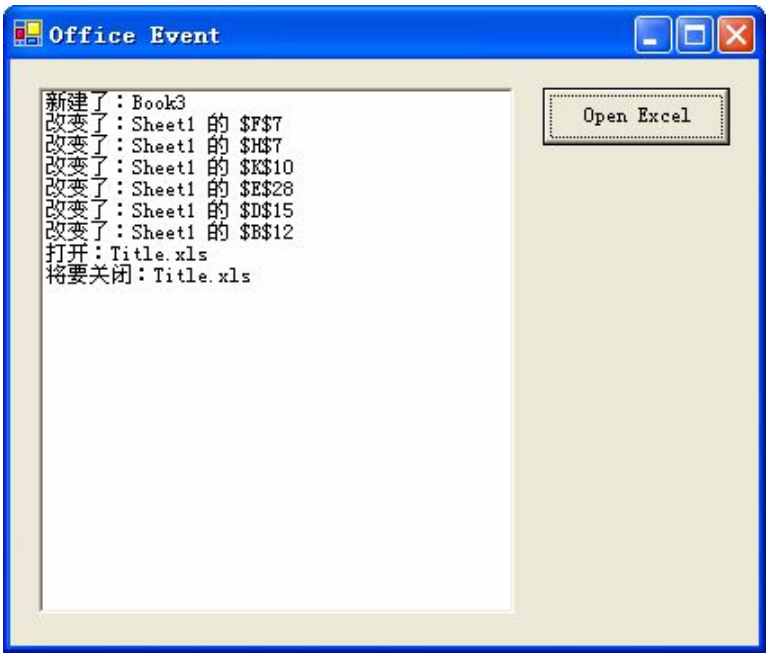



图 7-4 Office 自动化中事件编程示例

## 7.4.3. 使用 Excel 完成业务逻辑

利用 COM 自动化调用 Excel 除了上面数据导入导出的简单例子外，利用 Excel 的强大功能，可以完成很多在其他程序设计语言中不易完成而在 Excel 较为容易的工作，如使用 Excel 进行一些复杂数学模型的计算、图表的绘制等。对于我们在书中前面例举的数据处理、图表绘制的例子，也可以在其他程序设计语言中，调用 Excel 在后台完成。我们也可

以在 Excel 中使用 VBA，函数和自定义函数组合完成一些复杂模型的计算，这种情况下如果需要让电子表格中的公式重新计算，只需要调用 Application（WorkSheet 和 Range 也有此函数）的 Calculate 方法即可。

# 7.5. Excel 数据导入导出的几种方式

由于 Excel 应用的广泛性和 Excel 强大的功能与易用性，经常需要在其他程序中将数据导出为 Excel 格式，或者读入 Excel 格式的文件。在本部分，我们将介绍几种如何导入导出 Excel 数据的方法，包括通过 COM 自动化、ADO 以及第三方类库方式。

## 7.5.1. 使用自动化传输数据

和通过其他程序操作 Excel 一样，我们通过 COM 自动化启动 Excel，然后就可以使用 Excel 的对象模型，写入或读出 Excel 数据。

### 7.5.1.1. 使用“自动化”功能逐单元格传输数据

利用“自动化”功能，我们可以逐单元格地向工作表传输数据：

```
#001    Dim objExcel As Object
#002    Dim objBook As Object
#003    Dim objSheet As Object
#004
#005    '启动 Excel
#006    Set objExcel = CreateObject("Excel.Application")
#007    Set objBook = objExcel.Workbooks.Add
#009
#010    '将数据写入特定的单元格
#011    Set objSheet = objBook.Worksheets(1)
#012    objSheet.Range("A1").Value = "姓"
#013    objSheet.Range("B1").Value = "名字"
#014    objSheet.Range("A1:B1").Font.Bold = True
#015    objSheet.Range("A2").Value = "张三"
#016    objSheet.Range("B2").Value = "李四"
#017
#018    '保存工作薄并退出
#019    objBook.SaveAs "D:\Book1.xls"
#020    objExcel.Quit
```

在数据量较少的情况下逐单元格传输数据是一种完全可以接受的方法。我们可以灵活

地将数据放到工作簿中的任何位置，并可以在运行时根据条件对单元格进行格式设置。不过，如果需要向 Excel 工作簿传输大量数据，则应该使用下面的方法[26]。

#### 7.5.1.2. 使用“自动化”功能将数据数组传输到工作表上的区域

使用“自动化”功能将数据数组传输到工作表上的区域一次可以将一个数据数组传输到多个单元格区域，例如我们在“通过其他程序自动化 Excel”一节中的例子就是使用这种方法，看下面这个例子：

```
#001    Dim objExcel As Object
#002    Dim objBook As Object
#003    Dim objSheet As Object
#004
#005    '启动 Excel
#006    Set objExcel = CreateObject("Excel.Application")
#007    Set objBook = objExcel.Workbooks.Add
#008
#009     '创建一个数组
#010     Dim DataArray(1 To 100, 1 To 3) As Variant
#011     Dim i As Integer
#012     For i = 1 To 100
#013        DataArray(i, 1) = "订单 " & Format(i, "0000")
#014        DataArray(i, 2) = Rnd() * 1000
#015        DataArray(i, 3) = DataArray(i, 2) * 0.7
#016     Next
#017
#018     '在第一行增加表头
#019     Set objSheet = objBook.Worksheets(1)
#020     objSheet.Range("A1:C1").Value =  _
           Array("订单号", "数量", "税额")
#021
#022     '传递数据
#023     objSheet.Range("A2").Resize(100, 3).Value = _
           DataArray
#024
#025     '保存数据，退出 Excel
#026     objBook.SaveAs "C:\Book1.xls"
#027     objExcel.Quit
```

使用数组传输数据而不是逐单元格传输数据，在传输大量数据时，传输性能会大大增

[26] 在“提高效率”一节还将讨论这个问题。

强。程序第 15 行将 300 数组的数据传递到 300 个单元格。此行表示两个接口请求（一个用于 Range 方法返回的 Range 对象，另一个用于 Resize 方法返回的 Range 对象）。只要有可能，尽量使用批量传输数据以及减少所发出的接口请求的数量中受益。

### 7.5.1.3. 使用“自动化”功能将 ADO 记录集传输到工作表区域

Excel 2000 引入了 CopyFromRecordset 方法，使用此方法可以将 ADO（或 DAO）记录集传输到工作表上的某个区域。下面的代码说明了如何自动化 Excel 以及使用 CopyFromRecordset 方法传输 ADO 数据集的内容。

```
#001    '打开数据库，获取数据集
#002    Dim conn As New ADODB.Connection
#003    Dim rs As ADODB.Recordset
#004
#005    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
#006       & Data Source= [数据库路径];"
#007    rs.Open [SQL 语句], _
#008        cnn, adOpenKeyset, adLockOptimistic
#009
#010    '打开 Excel
#011    Dim objExcel As Object
#012    Dim objBook As Object
#013    Dim objSheet As Object
#014    Set objExcel = CreateObject("Excel.Application")
#015    Set objBook = oExcel.Workbooks.Add
#016    Set objSheet = oBook.Worksheets(1)
#017
#018    '传输数据
#019    objSheet.Range("A1").CopyFromRecordset rs
#020
#021    '保存并退出
#022    objBook.SaveAs "C:\Book1.xls"
#023    objExcel.Quit
#024
#025    '关闭数据库连接
#026    rs.Close
#027    conn.Close
```

使用此方法，即可快速的把 ADO 数据集中的数据传输到 Excel 的电子表格。Excel 97 也提供了一种 CopyFromRecordset 方法，但它只能用于 DAO 记录集，不支持 ADO。

### 7.5.1.4. 使用“自动化”功能在工作表上创建 QueryTable

使用 Excel 可以创建 QueryTable，通过导入数据，可以不必重复键入要在 Microsoft Excel 中进行分析的数据。也可以在每次更新数据库时，自动通过原始源数据库中的数据来更新 Excel 报表和汇总数据。QueryTable 对象代表由外部数据源返回的数据构建的表。在 Excel 中，可以通过查阅向导加载宏或者提供指向 OLEDB 或 ODBC 数据源的连接字符串和 SQL 字符串就可以创建 QueryTable。Excel 假定能够生成记录集，并负责将其插入工作表中的指定的位置。使用 QueryTables 可提供优于 CopyFromRecordset 方法的多种优点：

1. Excel 处理记录集的创建并将其放置到工作表中；
2. 查询可以保存在 QueryTable 中，以便在以后能够刷新，以获取更新的记录集；
3. 当向工作表中添加新的 QueryTable 时，可以指定将工作表上的单元格中已经存在的数据移位，以便放置新数据。

我们可以使用编程手段创建 QueryTable，下面的代码演示了如何自动运行 Excel 2000 及后续版本，在 Excel 工作表中创建新的 QueryTable：

```
#001    Dim objExcel As Object
#002    Dim objBook As Object
#003    Dim objSheet As Object
#004    Set objExcel = CreateObject("Excel.Application")
#005    Set objBook = oExcel.Workbooks.Add
#006    Set objSheet = oBook.Worksheets(1)
#007
#008    '创建 QueryTable
#009    Dim objQryTable As Object
#010    Set objQryTable = objSheet.QueryTables.Add( _
           "OLEDB;Provider=Microsoft.Jet.OLEDB.4.0;" & _
           "Data Source= [数据库路径];", & _
           objSheet.Range("A1"), "[SQL 语句]")
#012    objQryTable.RefreshStyle = xlInsertEntireRows
#013    objQryTable.Refresh False
#014
#015    objBook.SaveAs "C:\Book1.xls"
#016    objExcel.Quit
```

## 7.5.2. 使用 ADO 操作 Excel 数据

使用 Microsoft Jet OLE DB 提供程序，通过 ADO 可以将记录添加到现有 Excel 工作簿的一个表中，也可以从表中返回数据。Excel 中的“表”仅是带有定义名称的一个区域。区域中的第一行必须包含标题（或字段名），而且所有后续行都包含记录。下列步骤说明了如何使用名为 MyTable 的空表创建工作簿：

1. 在 Excel 中启动一个新工作簿；
2. 将下面的标题添加到 Sheet1 中的 A1:B1 单元格：A1: FirstName，B1: LastName；
3. 将单元格 B1 的格式设置为右对齐；
4. 选择 A1:B1；
5. 在插入菜单上，选择名称，然后选择定义。输入名称 MyTable，并单击确定；
6. 将新工作簿另存为 C:\Book1.xls 并退出 Excel。

通过 ADO 访问 Excel 电子表格的典型连接字符串格式如下所示：

```
"Provider = Microsoft.Jet.OLEDB.4.0;" & _
"Data Source = C:\Spreadsheets\calculations.xls;" & _
"Extended Properties = Excel 8.0"
```

在连接到 Excel 文件的情况下，可以看到我们必须在连接字符串中使用 Extended Properties 属性以及 Provider 和 Data Source。如果是 Excel 8.0 和更高版本，则使用“Excel 8.0”设置。对于有密码保护的 Excel 文件，则无法打开。

这样，我们就可以使用 ADO 打开、更新 Excel 工作薄，要使用 ADO 将记录添加到上面创建的 MyTable 中，可以使用与以下内容类似的代码：

```
#001    '创建连接到 Excel 的连接
#002    Dim conn As New ADODB.Connection
#003    conn.Open "Provider = Microsoft.Jet.OLEDB.4.0;" & _
#004       "Data Source = C:\Book1.xls; ” & _
#005       “Extended Properties = Excel 8.0;"

#006    conn.Execute "Insert into “ & _
#007        MyTable (FirstName, LastName)" & _
#008       " values ('Bill', 'Brown')"

#009    conn.Execute "Insert into “ & _
#010       ”MyTable (FirstName, LastName)" & _
#011       " values ('Joe', 'Thomas')"
```

```
#012    conn.Close
```

在以此方式将记录添加到该表中后，工作簿中的格式将会保留。在前面的示例中，添加到 B 列中的新字段的格式设置为右对齐。添加到行中的每个记录都将继承它前面的行的格式。请注意，在将一个记录添加到工作表中的一个或多个单元格时，该记录将会覆盖这些单元格中以前存在的任何数据；也就是说，在添加新记录时，工作表中的行不会“向下推移”。在工作表中设计数据的布局时应考虑到这一点。

对于最简单的 Excel 数据检索，需要指定工作表和该表中的单元格范围。需要确保工作表名称后跟美元符号和单元范围（可选）。通过使用冒号分隔单元格范围中的起始单元格和终止单元格，来指定此单元格集。然后，使用括号将整个数据标识字符串括起。使用此类型语法的 SELECT 语句一般如下所示：

```
SELECT SalesMonth, TotalSales, PercentageChange1Year
FROM [Sheet1$A1:E24]
```

因此，若要检索一张工作表内的所有数据，则使用该工作表的名称，后面带有一个美元字符，并用方括号将其括起：例如：

```
strQuery = "SELECT * FROM [Sheet1$]"
```

如果象本节开始那样对区域指定了名称，则只需使用定义的名称。例如：

```
strQuery = "SELECT * FROM MyRange"
```

对未命名的单元格区域作为数据来源，需要在工作表名的后面加上区域范围，并用方括号将其括起。例如：

```
strQuery = "SELECT * FROM [Sheet1$A1:B10]"
```

同样，也可以使用 ADO.net 来操作 Excel 数据，操作方法和 ADO 类似。

## 7.5.3. 使用第三方类库

有很多第三方类库（包括开源的和非开源的）可以读取和写入 Excel 数据文件，使用第三方类库的好处是计算机不需要安装有 Excel 也可以完成此类操作。例如有很多商业的 Excel 文件格式读写工具，如 ComponentOne XLS for .NET[27]，用友华表的 Excel 读写控件[28]，

[27] http://www.component1.com/
[28] http://www.cellsoft.cc/UfHbSite/index.asp

Aspose.Excel 控件，Syncfusion Essential ExcelRW 控件；开源的类库有.net 下的 koogra[29]等，在 SourceForge[30]上可以检索到不同系统、平台下的 Excel 读写开源项目。有关第三方类库的信息可以通过网络检索获取。

## 7.6. 关于 Excel 工程的引用

在 VBA 一章中，我们介绍了如何引用 COM 对象，在程序设计中使用所引用的 COM 组件的功能。在 Excel 中，引用不仅包括对 COM 对象的引用，而且还可以引用其他 Excel 文件、加载宏。打开 VBA IDE 下的*“工具 —— 引用”*可以选择当前工程的引用，所引用的对象可以是 COM 对象和 Excel 文件（图 7-5）。

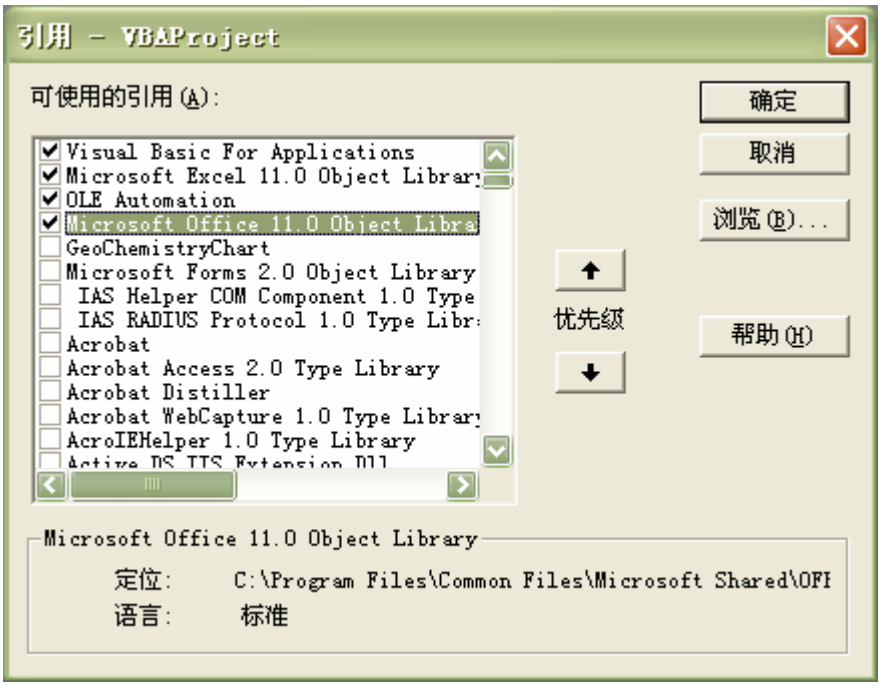



图 7-5 VBA 工程的引用对话框

通过引用，一个 Excel 工程（一个文件，或者加载宏）可以调用其他文件或者加载宏中的过程、函数和公有类。例如，我们使用 VBA 写了一个通用的统计分析的程序库，那么在别的程序中，就不需要将此文件中的这些代码拷贝过去，而只需要添加此文件的引用，即可在程序中使用此程序公开的方法和对象。通过这种方式，在很多程序中就可以使用已有加载宏或者程序的模块，也可以通过这种方式来进行合作开发。

---

[29] http://sourceforge.net/projects/koogra/
[30] http://sourceforge.net/

由于模块缺省属性是公有的（Public），因此在引用该工程时，模块内的公有过程和变量总是可见的。在这种情况下，使用 Option Private Module，可以防止在模块所属的工程外引用该模块的内容。使用方法，在模块中的任何过程之前使用此语句：

```
Option Private Module
```

如果模块中使用了 Option Private Module，则其公用部分（例如，在模块级定义的变量，对象，以及用户定义类型）在该模块所属的工程内仍是可用的，但对其它应用程序或工程则是不可用的。

## 7.7. 提高效率的一些建议

首先，效率的提高应该建立在正确的解决方案和正确的算法的基础上，前者保证了结果的正确性，后者保证了效率。通过改进算法和思路得到的运行效率的提高可以是以下优化方法的十倍百倍，因此，优化首先应该考虑的算法，其次才是本文以下提到的方法。要特别指出，效率的优化必须是针对一些关键代码的优化，对于一些在程序执行过程中，只执行很少次数的代码，没有必要牺牲可读性而进行优化。在此基础上，可以通过注意以下一些问题，提高程序的运行效率。

### 7.7.1. 尽量使用 Excel 的内置函数

应该尽量使用 Excel 的内置函数，使用 Excel 内置函数不仅可以提高运行效率，而且可以节省代码数量。对于 Excel 内置函数可以通过以下方式访问：

Application.函数名()

Application.WorksheetFunction.函数名(myRange)

例如以下求平均值的例子，使用的 VBA 代码如下：

```
For Each c In Worksheet(1).Range(″A1:A1000″)
    TotalValue = TotalValue ＋ c.Value
Next
AverageValue = TotalValue / _
    Worksheet(1).Range(″A1:A1000″).Rows.Count
```

而下面代码程序比上面例子快得多：

```
AverageValue = _
    Application.Average(Worksheets(1).Range("A1:A1000"))
```

其它函数如 Count，Counta，Countif，Match，Lookup 等等，都能代替相同功能的 VBA

程序代码，提高程序的运行速度。

## 7.7.2. 尽量减少使用对象引用

在 VBA 代码中，应该尽量减少使用对象引用，尤其在循环中。每一个 Excel 对象的属性、方法的调用都需要通过 COM 接口的一个或多个调用，这些 COM 调用都是比较费时的，因此，减少使用对象引用能加快 VBA 代码的运行。可以通过以下途径改进效率：

### 7.7.2.1. 使用 With 语句

例如以下语句，可以通过替换为 With 语句，提高运行效率：

```
ActiveSheet.Range("A1:A1000").Font.Name = "Arial"
ActiveSheet.Range("A1:A1000").Font.FontStyle = "Bold"
```

对应的 With 语句：

```
With ActiveSheet.Range("A1:A1000").Font
    .Name = "Arial"
    .FontStyle = "Bold"
End With
```

### 7.7.2.2. 使用对象变量

如果一个对象引用被多次使用，则可以通过定义一个局部变量，将此对象用 Set 设置为对象变量，以减少对对象的访问。如：

```
ActiveSheet.Range("A1").Value = 100
ActiveSheet.Range("A2").Value = 200
```

则以下代码要比上面的要快：

```
Dim objSheet As Object
Set objSheet = ActiveSheet
objSheet.Range("A1").Value = 100
objSheet.Range("A2").Value = 200
```

### 7.7.2.3. 减少循环中的对象访问

例如以下循环，可以通过设置局部变量或者使用 With 语句来提高效率。

```
For k = 1 To 1000
    ActiveSheet.Range("A1000").Cells(1,k).Value = k
```

```
Next k
```

则以下代码比上面的要快（使用 With 语句）：

```
With ActiveSheet.Range("A1000")
    For k = 1 to 1000
        .Cells(1,k).Value = k
    Next k
End With
```

## 7.7.3. 高效使用 Range 对象

VBA 编程中对 Excel 对象的引用是不可避免的，而且是经常性的工作，例如对 Range 对象的引用，但同时这种引用又是非常耗时的，例如与使用数组相比较，使用 Range 对象要慢 $10^3$ 倍到 $10^4$ 倍，因此，一定要避免频繁引用 Range 对象（例如在一些信息查询、矩阵运算等时候），必要的时候，可以通过数组等方式来替代，在运算开始前将 Range 的数据读入数组，运算完成后再写入 Range。

Range 对象读入数组可以使用以下方法：

```
vData = ActiveSheet.Range("A1:B10").Value
```

其中 vData 可以是定义好的数组，也可以是一个 Variant 变量。反过来，使用：

```
ActiveSheet.Range("D1:E10").Value = vData
```

就可以将数组 vData 的值赋给 Range，如果 Range 的范围较小，则自动截断。例如以下这段代码：

```
Set objRng = objSheet.Range("A" & 1 & ":" & "J" & 2000)
With objRng
    For j = 1 To 2000 Step 1
        For i = 1 To 10 Step 1
           .Cells(j, i).Value = CStr(j)
        Next i
    Next j
End With
```

要比下面这段代码慢 $10^3$ 倍左右。

```
Dim a(1 to 2000, 1 to 10) As Variant
For j = 1 To 2000 Step 1
    For i = 1 To 10 Step 1
        a(j, i) = CStr(j)
    Next i
Next j
objRng.Value = a
```

其速度差异的原因是 Range 操作非常耗时，而数组操作相对于 Range 操作，时间快的可以忽略，因此，对于 $10^3$ 的数据量，1 次 Range 对象操作和 $10^3$ 次 Range 对象操作其速度差异可以达 $10^3$ 倍，数据量增大，虽然数组操作和将数组赋给 Range 对象会慢一些，但时间差异将依然以指数级增长。

因此，在要对 Range 对象进行频繁操作时，应该尽量使用数组替换。

### 7.7.4. 减少对象的激活和选择

通过录制宏得到的 VBA 代码充满了对象的激活和选择，例如 Workbooks(xxx).Activate、Sheets(xxx).Select、Range(xxx).Select 等，但事实上大多数情况下这些操作不是必需的，因此，应该尽量避免这样的代码。例如：

```
Sheets("Sheet3").Select
Range("A1").Value = 100
Range("A2").Value = 200
```

可改为：

```
With Sheets("Sheet3")
    .Range("A1").Value = 100
    .Range("A2").Value = 200
End With
```

### 7.7.5. 关闭屏幕更新

避免不断的刷新屏幕，在向工作薄写数据或者绘图时，锁定屏幕刷新；

如果你的 VBA 程序需要经常更新屏幕工作表的内容，则关闭屏幕更新是提高 VBA 程序运行速度的最有效的方法，如果不涉及复杂的计算，可以缩短运行时间 2/3 左右[31]。关闭屏幕更新的方法：

Application.ScreenUpdate = False

请不要忘记 VBA 程序运行结束时再将该值设回来：

Application.ScreenUpdate = True

[31] 这个结果是一个粗略结果，对于需要往屏幕写大量数据或者绘制复杂图片，而计算量不大情况下的测试结果。

## 7.7.6. 提高关键代码的效率

最后要特别指出，不要做不必要的优化。虽然本书给出了很多不同方法执行效率的差异，但千万不要因为追求效率而损失了代码的可读性、清晰性。

效率的优化必须是针对关键代码的优化，对于一些在程序执行过程中，只执行很少次数的代码，没有必要牺牲可读性而进行优化。

对于代码执行效率，千万不要人云亦云，必要时候，自己动手测试一下，结果往往会出乎意料。

本节所建议的效率优化方案，对于使用 COM 自动化来调用 Excel 时也一样适用。

## 7.7.7. 代码执行时间的测算

VBA 和 VB 中，没有专门的代码执行事件测算工具和方法，笔者一般是使用 Timer 函数，其返回值是一个 Single 类型的数值，代表从午夜开始到现在经过的秒数，此数值包括小数部分，但精确程度在 Windows NT，2000 和 XP 下大概接近 10 毫秒。如果要测试一段代码的执行速度，可以使用如下方法：

```
#001 Sub MeasureTime()
#002
#003    Dim Time1 As Single, Time2 As Single
#004    Dim TotalTime As Single
#005    Dim Times As Long
#006    Dim i As Long
#007
#008    Times = 10000
#009    Time1 = Timer
#010
#011    For i = 1 To Times Step 1
#012       Mytest1
#013    Next i
#014
#015    Time2 = Timer
#016
#017    TotalTime = (Time2 - Time1) * 1000
#018    MsgBox "执行时间: " & TotalTime & " 毫秒（次数: " _
#019       & Times & ")"
#020
#021 End Sub
```

```
#022
#023 Sub Mytest1()
#024
#025    Dim i As Long
#026    Dim s As String
#027    i = Rnd
#028    s = Format(i, "#.00")
#029
#030 End Sub
```

过程 MeasureTime 可以测试一个过程的执行速度，因为一般一个过程执行会很快，所以使用循环，执行 n 次（第 8 行设置），在第 12 行调用测试的过程，通过循环前的时间（第 9 行）和循环后的时间（第 15 行），计算总共执行时间（第 17 行）。

使用这个方法，就可以做一些测试，看哪些方法执行效率更高。另外，由于 Windows 的多任务特点，测试时最好关闭其他无关程序，以获得较准确的测试结果。

# 8. 附录

## 8.1. VBA 命名规则

一个好的命名规则可以提高程序的可读性，减少错误发生的概率，命名规则不是一定的，不同的人有不同的规则和习惯，但在编程过程中，对于个人或工作组，一定要遵守相同的命名规则[32]。

### 8.1.1. 变量、常量、自定义类型和枚举

表 8-1 概括了变量、常量的基本命名规则。

表 8-1 变量、常量和枚举类型的命名规则

| 元素 | 命名规则 |
|---|---|
| 变量 | <范围><数组><数据类型>描述（首字母大写） |
| 常量 | <范围><数据类型>描述（全部大写） |
| 用户自定义类型 | Type 描述名称<br>　　<数据类型>描述<br>End Type |
| 枚举类型 | Enum <工程前缀>一般描述<br>　　<工程前缀><一般描述><具体名称 1><br>　　<工程前缀><一般描述><具体名称 2><br>End Enum |

<范围>表示了变量的作用域，对于 Private 类型和模块级变量，一般使用“m”前缀表示，对于 Public 类型的变量，一般使用“g”前缀表示，而对于过程内的局部变量，则不使用前缀。如果是数组，在范围前缀后增加“a”表示变量为数组。

对于数据类型，一般使用表 8-2 的前缀表示。

表 8-2 命名规则常用前缀

| 前缀 | 数据类型 | 前缀 | 数据类型 | 前缀 | 数据类型 |
|---|---|---|---|---|---|
| is | Boolean | cm | ADODB.Command | cmb | MSForms.ComboBox |

[32] 本命名规范以及本章后节的代码规范也适用于 VB 开发。

| byt | Byte | cn | ADODB.Connection | chk | MSForms.CheckBox |
|---|---|---|---|---|---|
| cur | Currency | rs | ADODB.Recordset | cmd | MSForms.CommandButton |
| dte | Date | | | fra | MSForms.Frame |
| dec | Decimal | cht | Excel.Chart | lbl | MSForms.Label |
| f | Double，Single | rng | Excel.Range | lst | MSForms.ListBox |
| i | Integer，Long | wb | Excel.Workbook | mpg | MSForms.MultiPage |
| obj | Object | ws | Excel.Worksheet | opt | MSForms.OptionButton |
| str | String | | | spn | MSForms.SpinButton |
| u | User-defined type | cbr | Office.CommandBar | txt | MSForms.TextBox |
| v | Variant | ctl | Office.CommandBarControl | ref | RefEdit Control |
| col | VBA.Collection | cls | 自定义类 | frm | 用户窗体 |

变量的描述部分最好使用有意义的字符串，使用 1-2 个英文单词表示，首字母大写，例如“strUserName”、“iPeopleAge”。除了循环变量使用 i、j，临时变量使用 tmp 之类的变量外，不要使用太短的命名，但也不要使用太长不易记忆的名称。

常量则一般使用全部大写的方式，以与变量区别。

对于枚举类型，整个工程一定要使用一致的规则，每个枚举常量都包含工程前缀，变量前缀和本身描述几部分，例如：

```
Private Enum schDayType
    schDayTypeUnscheduled
    schDayTypeProduction
    schDayTypeDownTime
    schDayTypeHoliday
End Enum
```

## 8.1.2. 过程和函数

过程和函数命名一般使用“名词 + 动词”的方式，首字母大写，也可以使用“动词 + 名词”方式，对于过程和函数的参数，命名方式见前，为了和局部变量区别，可以不使用表示参数变量类型的前缀。例如，我们可以命名如下的过程：

```
GetUserName(id as long) As String
```

## 8.1.3. 模块、类模块和用户窗体

模块使用类似过程的命名，用几个表示其用途的首字母大写的短语来表示，例如“PlotChartTools”；类模块增加前缀“C”，以与标准模块相区别，例如“CIniTools”、

“CEmployee”等；用户窗体则以“frm”为前缀，如“frmAbout”、“frmRegTools”。这样，在代码中我们可以这样使用类模块：

```
Dim clsMyClass As CMyClass
Set clsMyClass = New CMyClass
```

类模块与其对象差别一目了然。由于 VBA 对于窗体可以使用缺省窗体，不需要创建实例，在代码中可以直接使用，因此，使用了与变量定义一样的前缀。例如：

```
frmRegTools.Show
```

### 8.1.4. VBA 工程

VBA 工程一般使用与其文件名同名的名字，一方面，当打开几个工程的时候可以方便的区分工程，另一方面，在工程之间引用的时候，需要不同的名称。

## 8.2. VBA 代码规范

代码规范表示了如何定义变量、过程、函数（见前），如何组织代码，控制缩进，添加注释等内容。代码规范的目的在于产生一致的代码，提高代码的可读性，使其易于修改和交流。以下规范并非必须遵守，当使用规范破坏了代码的可读性，那么就没有必要遵从代码规范了，这种情况需要自行判断。

### 8.2.1. 代码的排版

**缩进**

一般来说，代码的缩进应该为 4 个空格，在 VBA IDE 中选中自动缩进，并设置为 4 个字符。一个过程的语句要比过程名称缩进 4 个空格，在循环，判断语句、With 语句之后也要缩进。例如：

```
If strText = " " Then
    NoZeroLengthString = Null
Else
    NoZeroLengthString = strText
End If
```

**行的长度**

一行代码尽量不要过长，对于大多数编程规范，建议一行代码的最大长度为 80 个字符，在 VBA 中，可以使用续行赋“-”将长的代码行分为数行，后续行应该缩进以表示与前行的关系。例如：

```
AverageValue = TotalValue / _
    Worksheet(1).Range(″A1:A1000″).Rows.Count
```

**空行**

一个模块内部，过程之间要使用空行隔开，模块的变量定义和过程之间也应该空 1 行。过程内部，变量定义和代码应该空 1 行。在一组操作和另一组操作之间也应该空 1 行显示其逻辑关系。空行可以很好的提高程序的可读性，但同时，空行没有必须遵守的规则，其使用的目的就是要显示程序的逻辑关系。

**不要将多个语句放在同一行上**

虽然 VBA 允许将多条语句放在一行，但不推荐这么做。

## 8.2.2. 注释

书写程序的同时，应该同时对关键代码，模块，过程增加注释，更改程序的同时，必须同时更改注释。必须时刻保证注释与程序代码一致，否则还不如不加注释。对于简短的注释，不需要加句号，否则应该增加句号，组成段落。

如果可能，建议尽量使用英文书写注释，因为这样会带来交流的便利，特别是在正式的开发中。

**区块注释**

区块注释通常描述其下的部分或全部代码，例如模块说明或者过程说明。其缩进要和它所描述的代码一致。模块的注释应该位于模块的所有代码之前，Option 语句之后，过程的注释位于过程定义之后，并保证缩进一致。对于模块的注释，注释结束后应该有一空行，其前后可以加一些修饰以区别与其他注释，而过程注释则不需要。例如：

```
#001 Option Explicit
#002
#003 '************************************************
#004 '主程序模块，提供按钮调用，对话框弹出等服务
```

```
#005 '*************************************************
#006
#007 Public Const strVer As String = "0.31"
#008
#009 Public Sub GeoDrawMain()
#010 '主程序模块，单击按钮后弹出
#011    frmMain.Show vbModal
#012 End Sub
```

**行内注释**

行内注释的形式是在语句的同一行中加注释，行内注释应该简单明了，并不要描述显而易见的事情。行内注释和语句至少应该有 2 个以上空格。可以在语句和注释之间使用多个 Tab 使注释对齐，例如：

```
Dim iAge As Long            '年龄
Dim strName As String       '姓名
```

## 8.2.3. 程序版本

建议在模块注释中包括作者，修改时间，版本等信息，例如：

```
' 模块名称：气压计算模块
' 描述：…
' 作者：Mars
' 创建时间：2004 年 4 月 23 日
' 修改时间：2005 年 7 月 13 日
' 版本：2.5
```

此类注释应该形成自己的风格，在所有的工程中保持一致。对于团队工作和正式开发，应该严格要求在模块注释中包括这些内容。

## 8.2.4. 一些基本原则

**明确说明作用范围**

在代码中，对于模块级的变量，过程，函数，应该总是使用“Public”、“Private”等关键字明确说明其范围。

**使用有意义的名称**

一定要使用有意义的名称，而不要使用简单的 A、B、C 之类的名称（循环变量约定俗成使用 i、j 等名称除外）。

**明确参数和变量的数据类型**

在定义过程参数的时候，一定要明确指定其数据类型和传递方式（ByRef 或者 ByVal），这不仅仅是考虑效率，而是为了方便对这些过程的使用。对于变量定义，也应该养成明确说明其数据类型的习惯。

**模块内的过程排序**

模块内部的过程应该按照“Public”、“Private”的顺序排序，公有的过程在前，私有在后；然后再按照过程名称字母顺序排序。

**使用常量和枚举**

应该尽量使用常量和枚举，而不要在程序代码中使用数字（幻数）。

**语句的选择**

对于 True、False 的判断，使用 If 语句，对于多种可能的判断，使用 Select 语句。对于循环，对于确定循环次数的循环，使用 For 语句，对于不确定循环次数的循环，使用 Do While 语句，尽量对集合使用 For Each 语句。

**Goto 语句**

除了错误处理，不要使用 Goto 语句。

## 8.3. 可用于 VBA 代码的工作表函数列表

本附录按照字母顺序列出了所有可以用于 VBA 代码的工作表函数，供读者查询和使用，Excel 版本为 Excel 2003，其他版本以其文档说明为准。

**A**

Acos，返回数字的反余弦值

Acosh，返回数字的反双曲余弦值

And，如果其所有参数为 TRUE，则返回 TRUE

Asin，返回数字的反正弦值

Asinh，返回数字的反双曲正弦值

Atan2，返回 X 和 Y 坐标的反正切值

Atanh，返回数字的反双曲正切值

AveDev，返回数据点与它们的平均值的绝对偏差平均值

Average，返回其参数的平均值

**B**

BetaDist，返回 Beta 累积分布函数

BetaInv，返回指定 Beta 分布的累积分布函数的反函数

BinomDist，返回一元二项式分布的概率值

**C**

Ceiling，将数字舍入为最接近的整数或最接近的 Significance 的倍数

ChiDist，返回 γ2 分布的单尾概率

ChiInv，返回 γ2 分布的单尾概率的反函数

ChiTest，返回独立性检验值

Choose，从一列值中选择值

Clean，删除文本中不可打印的字符

Combin，返回给定数目对象的组合数

Confidence，返回总体平均值的置信区间

Correl，返回两个数据集之间的相关系数

Cosh，返回数字的双曲余弦值

Count，计算参数列表中的数字个数

CountA，计算参数列表中的数值个数

CountBlank，计算区间内的空白单元格个数

CountIf，计算满足给定条件的区间内的非空单元格个数

Covar，返回协方差，成对偏差乘积的平均值

CritBinom，返回使累积二项式分布小于或等于临界值的最小值

**D**

DAverage，返回选择的数据库条目的平均值

Days360，计算基于一年 360 天的两个日期间的天数

Db，使用固定余额递减法，计算一笔资产在给定期间内的折旧值。

DCount，计算数据库中包含数字的单元格个数

DCountA，计算数据库中的非空单元格

Ddb，使用双倍余额递减法或其他指定方法，计算一笔资产在给定期间内的折旧值

Degrees，将弧度转换为度

DevSq，返回偏差的平方和

DGet，从数据库提取符合指定条件的单个记录

DMax，返回选择的数据库条目的最大值

DMin，返回选择的数据库条目的最小值

Dollar，使用 $（美元）货币格式将数字转换成文本

Dproduct，返回列表或数据库的列中满足指定条件的数值的乘积

DStDev，基于选择的数据库条目的样本估算标准偏差

DStDevP，基于选择的数据库条目的总体计算标准偏差

DSum，将数据库中符合条件的记录的字段列中的数字相加

DVar，基于选择的数据库条目的样本估算方差

DVarP，基于选择的数据库条目的样本总体计算方差

**E**

Even，将数字向上舍入为最接近的偶数

ExponDist，返回指数分布

**F**

Fact，返回数字的阶乘

FDist，返回 F 概率分布

Find，检查两个文本值是否相同

FindB，同上，用于双字节字符

Finv，返回 F 概率分布的反函数值

Fisher，返回 Fisher 变换

FisherInv，返回 Fisher 变换的反函数值

Fixed，将数字按指定的小数位数进行取整

Floor，向绝对值减小的方向舍入数字

Forecast，返回沿线性趋势的值

Frequency，以垂直数组的形式返回频率分布

FTest，返回 F 检验的结果

Fv，返回一笔投资的未来值

**G**

GammaDist，返回伽玛分布

GammaInv，返回伽玛累积分布函数的反函数

GammaLn，返回伽玛函数的自然对数，Γ(x)

GeoMean，返回几何平均值

Growth，返回沿指数趋势的值

**H**

HarMean，返回调和平均值

Hlookup，查找数组的首行，并返回特定单元格的值

HypGeomDist，返回超几何分布

**I**

Index，使用索引从引用或数组选择值

Intercept，返回线性回归线的截距

Ipmt，基于固定利率及等额分期付款方式，返回给定期数内对投资的利息偿还额

Irr，返回由数值代表的一组现金流的内部收益率

IsErr，如果值为除 #N/A 以外的任何错误值，则返回 TRUE

IsError，如果值为任何错误值，则返回 TRUE

IsLogical，如果值为逻辑值，则返回 TRU

IsNA，如果值为 #N/A 错误值，则返回 TRUE

IsNonText，如果值不为文本，则返回 TRUE

IsNumber，如果值为数字，则返回 TRUE

Ispmt，计算特定投资期内要支付的利息

IsText，如果值为文本，则返回 TRUE

**K**

Kurt，返回数据集的峰值

**L**

Large，返回数据集中第 k 个最大值

LinEst，返回线性趋势的参数

Ln，返回数字的自然对数

Log，按所指定的底数，返回数字的对数

Log10，返回数字的以 10 为底的对数

LogEst，返回指数趋势的参数

LogInv，返回对数分布函数的反函数

LogNormDist，返回对数累积分布函数

Lookup，在向量或数组中查找值

**M**

Match，在引用值或数组中查找值

Max，返回参数列表中的最大值

MDeterm，返回数组的矩阵行列式的值

Median，返回给定数值集合的中值

Min，返回参数列表中的最小值

MInverse，返回数组的逆矩阵

MIrr，返回正和负现金流以不同利率进行计算的内部收益率

MMult，返回两个数组的矩阵乘积

Mode，返回在数据集中出现次数最多的值

**N**

NegBinomDist，返回负二项式分布

NormDist，返回正态累积分布

NormInv，返回正态累积分布的反函数

NormSDist，返回标准正态累积分布

NormSInv，返回标准正态累积分布的反函数

NPer，返回投资期间的数量

Npv，返回基于一系列定期现金流和贴现率计算的投资的现净值

**O**

Odd，将数字向上舍入为最接近的奇数

Or，如果所有参数为 TRUE，则返回 TRUE

**P**

Pearson，返回 Pearson 乘积矩相关系数

Percentile，返回区域中数值的第 K 个百分点的值

PercentRank，返回数据集中的值的百分比排位

Permut，返回给定数目对象的排列数量

Phonetic，提取文本字符串中的拼音 (furigana) 字符

Pi，返回 pi 的值

Pmt，返回年金的定期支付金额

Poisson，返回泊松分布

Power，返回给定数字次幂的结果

Ppmt，返回一笔投资在给定期间内偿还的本金

Prob，返回区域中的数值落在指定区间内的概率

Product，将其参数相乘

Proper，将文本值的每个字的首字母大写

Pv，返回投资的现值

**Q**

Quartile，返回数据集的四分位数

**R**

Radians，将角度转换为弧度

Rank，返回一列数字的数字排位

Rate，返回年金的各期利率

Replace，替换文本中的字符

ReplaceB，同上，专为双字节字符使用

Rept，按照给定的次数重复显示文本

Roman，将阿拉伯数字转换为文本形式的罗马数字

Round，将数字按指定位数舍入

RoundDown，向绝对值减小的方向舍入数字

RoundUp，向绝对值增大的方向舍入数字

RSq，返回 Pearson 乘积矩相关系数的平方

RTD，与源连接以接收实时数据

**S**

Search，在一个文本值中查找另一个文本值（不区分大小写）

SearchB，同上，专为双字节字符使用

Sinh，返回数字的双曲正弦值

Skew，返回分布的偏斜度

Sln，返回一笔资产在某个期间内的线性折旧值

Slope，返回线性回归线的斜率

Small，返回数据集中第 K 个最小值

Standardize，返回正态化数值

StDev，估算基于给定样本的标准偏差

StDevP，计算基于给定的样本总体的标准偏差

StEyx，返回通过线性回归法计算每个 x 的 y 预测值时所产生的标准误差

Substitute，在文本字符串中用 new_text 替代 old_text

Subtotal，返回列表或数据库中的分类汇总

Sum，将其参数相加

SumIf，按给定条件对若干单元格求和

SumProduct，返回对应的数组元素的乘积和

SumSq，返回参数的平方和

SumX2MY2，返回两个数组中对应值的平方差之和

SumX2PY2，返回两个数组中对应值的平方和之和

SumXMY2，返回两数组中对应值的平方差的之和

Syd，返回一笔资产按年限总和折旧法计算的指定期间的折旧值

**T**

Tanh，返回数字的双曲正切值

TDist，返回学生的 t 分布

Text，设置数字格式并将其转换为文本

Tinv，返回作为概率和自由度函数的学生 t 分布的 t 值

Transpose，返回数组的转置

Trend，返回沿线性趋势的值

Trim，从文本删除空格

TrimMean，返回数据集的内部平均值

TTest，返回与学生的 t 检验相关的概率

**V**

Var，估算基于样本的方差

VarP，计算基于样本总体的方差

Vdb，使用余额递减法，返回一笔资产在给定期间或部分期间内的折旧值

VLookup，查找数组首列，移动到行并返回单元格的值

**W**

Weekday，将序列号转换为一星期的某天

Weibull，返回韦伯分布

**Z**

ZTest，返回 z 检验的单尾概率值