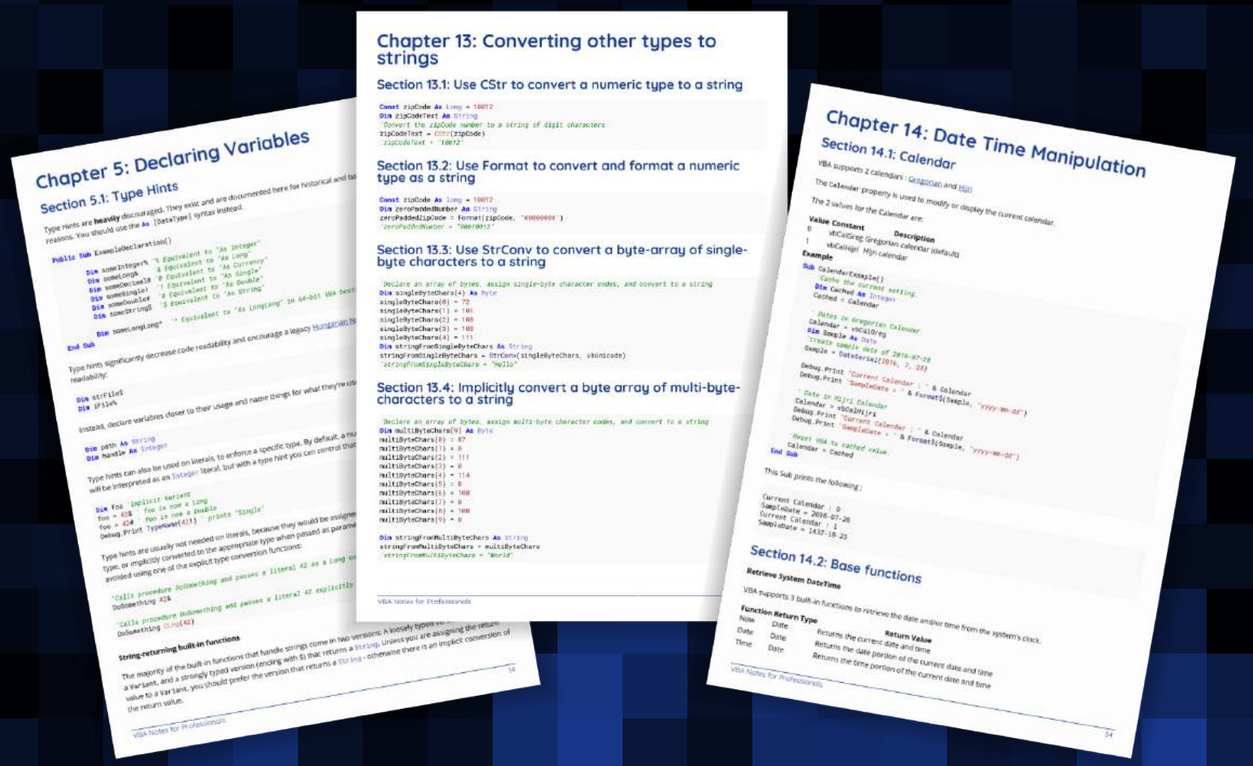
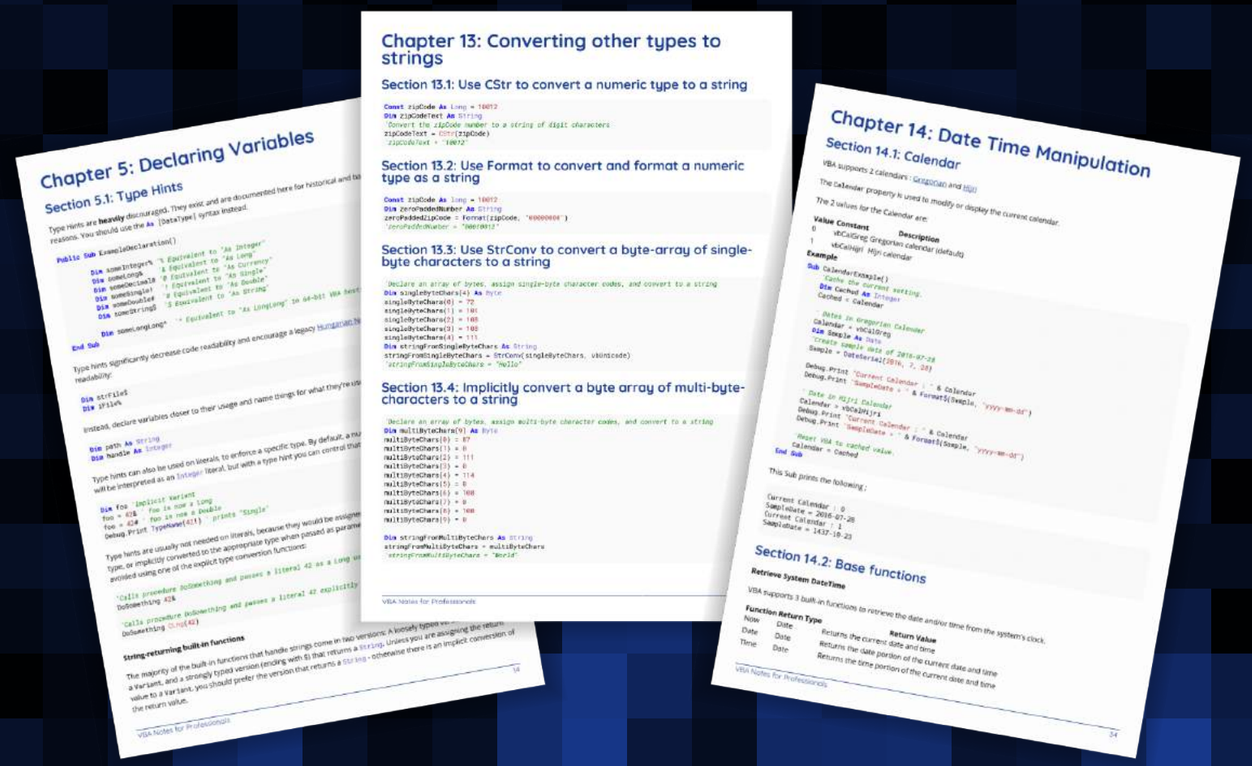
VBA
专业人员笔记



100多页

专业提示和技巧

VBA
Notes for Professionals



100+ pages

of professional hints and tricks

目录

Contents

关于

请随意免费与任何人分享此PDF，
本书的最新版本可从以下网址下载：
https://goalkicker.com/VBABook

本*VBA专业笔记*一书汇编自Stack Overflow Documentation，内容由Stack Overflow的优秀人士撰写。
文本内容采用知识共享署名-相同方式共享许可协议发布，详见本书末尾对各章节贡献者的致谢。图片版权归各自所有者所有，除非另有说明。

这是一本非官方的免费书籍，旨在教育用途，与官方VBA组织或公司及Stack Overflow无关。所有商标和注册商标均为其各自公司所有者所有。

本书所提供的信息不保证正确或准确，使用风险自负。

请将反馈和更正发送至web@petercv.com

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/VBABook

This *VBA Notes for Professionals* book is compiled from Stack Overflow Documentation, the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official VBA group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：VBA入门

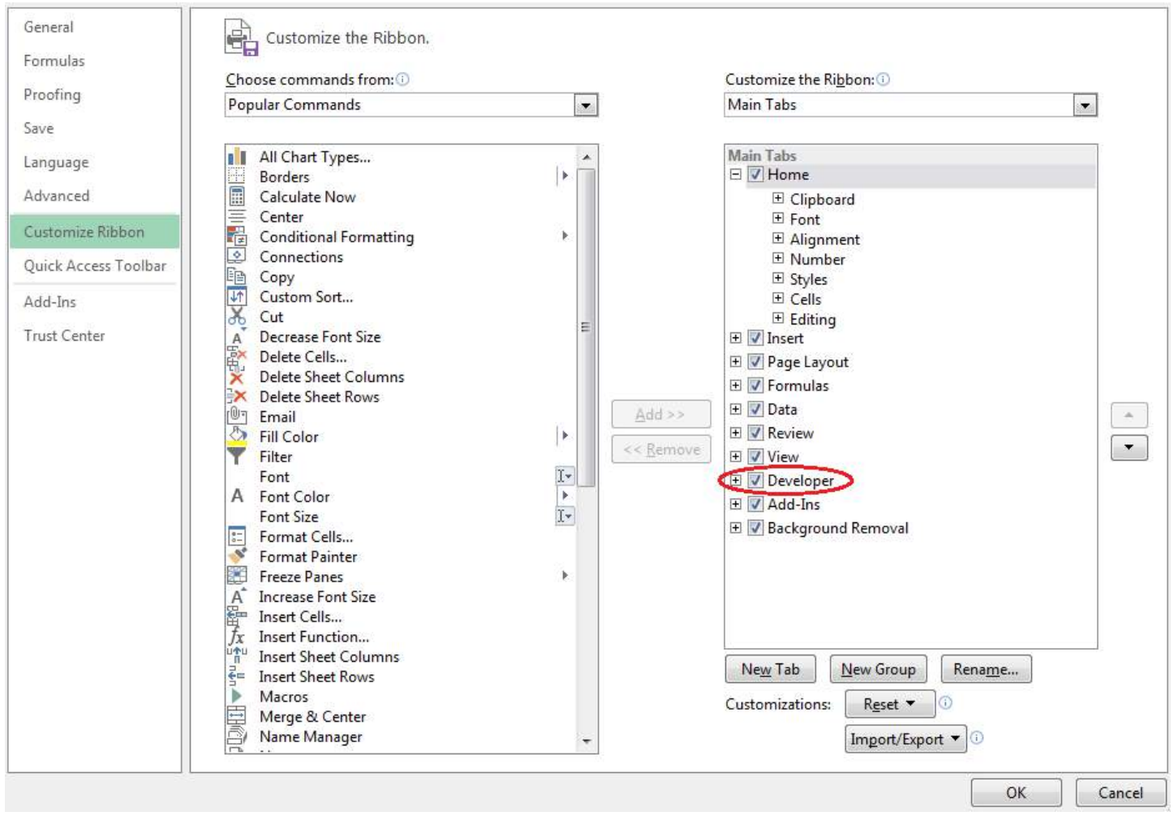
版本	Office 版本	发布日期说明	发布日期
Vba6	? - 2007	[1992-06-30之后某个时间][1]	
Vba7	2010 - 2016	[blog.techkit.com][2]	2010-04-15
Mac 版 VBA	2004，2011 - 2016		2004-05-11

第1.1节：在 Microsoft

Office 中访问 Visual Basic 编辑器

您可以通过按下以下键在任何 Microsoft Office 应用程序中打开 VB 编辑器 Alt+F11 或者转到“开发工具”选项卡并点击“Visual Basic”按钮。如果你在功能区看不到“开发工具”选项卡，请检查是否已启用该选项。

默认情况下，“开发工具”选项卡是禁用的。要启用“开发工具”选项卡，请转到“文件”->“选项”，在左侧列表中选择“自定义功能区”。在右侧的“自定义功能区”树视图中找到“开发工具”项，并勾选“开发工具”复选框。点击“确定”关闭选项对话框。



现在“开发工具”选项卡已在功能区中可见，你可以点击“Visual Basic”打开Visual Basic编辑器。或者你也可以点击“查看代码”直接查看当前活动元素的代码窗格，例如工作表、图表、形状。

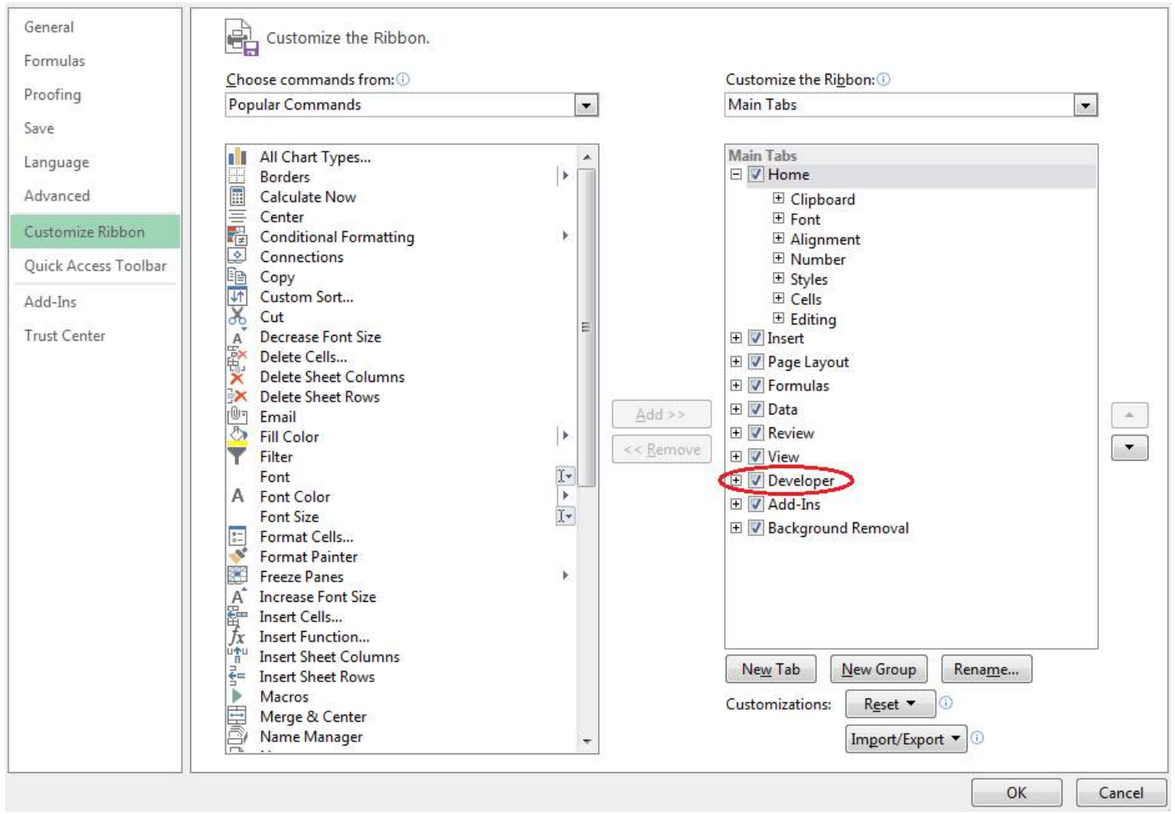
Chapter 1: Getting started with VBA

Version	Office Versions	Release Date Notes	Release Date
Vba6	? - 2007	[Sometime after][1]	1992-06-30
Vba7	2010 - 2016	[blog.techkit.com][2]	2010-04-15
VBA for Mac	2004, 2011 - 2016		2004-05-11

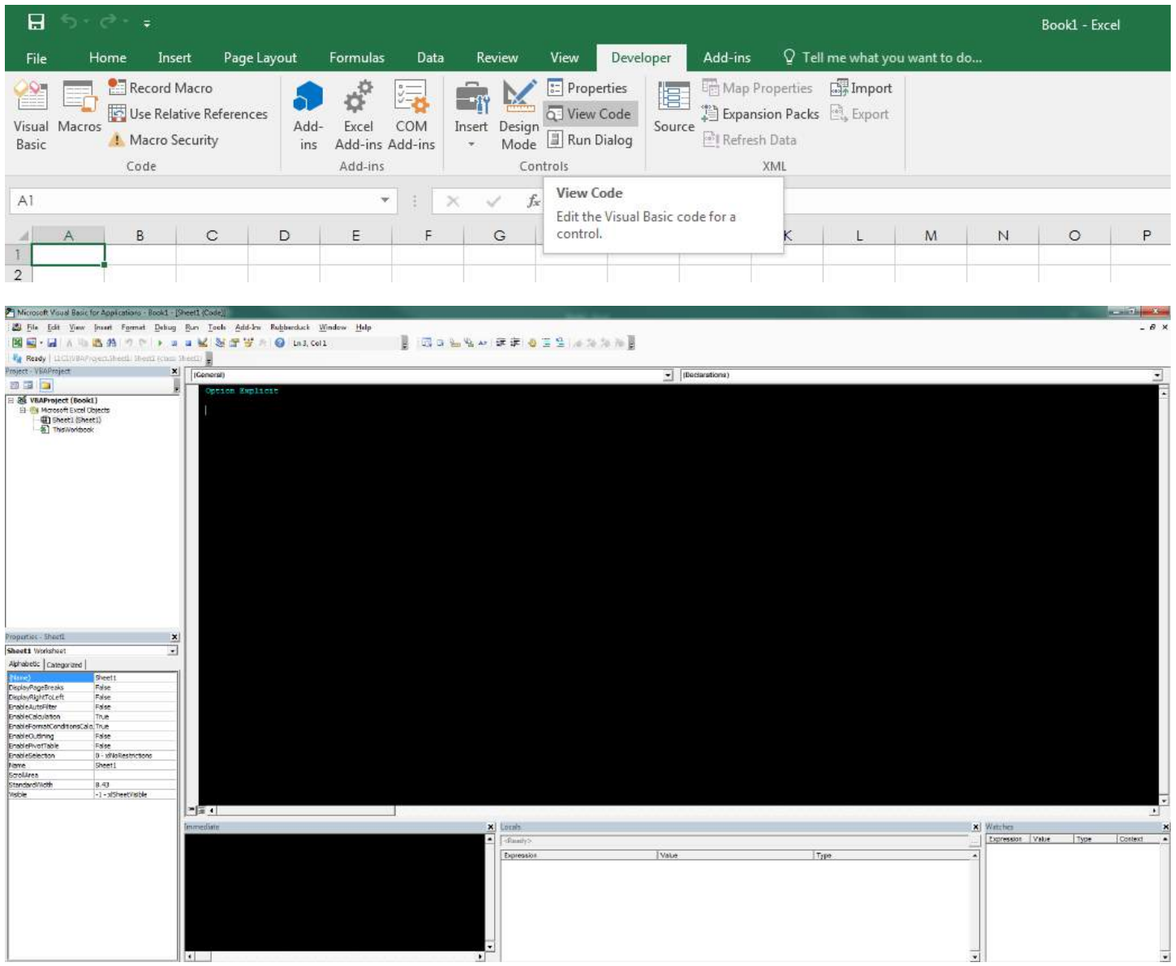
Section 1.1: Accessing the Visual Basic Editor in Microsoft Office

You can open the VB editor in any of the Microsoft Office applications by pressing Alt+F11 or going to the Developer tab and clicking on the "Visual Basic" button. If you don't see the Developer tab in the Ribbon, check if this is enabled.

By default the Developer tab is disabled. To enable the Developer tab go to File -> Options, select Customize Ribbon in the list on the left. In the right "Customize the Ribbon" treeview find the Developer tree item and set the check for the Developer checkbox to checked. Click Ok to close the Options dialog.



The Developer tab is now visible in the Ribbon on which you can click on "Visual Basic" to open the Visual Basic Editor. Alternatively you can click on "View Code" to directly view the code pane of the currently active element, e.g. WorkSheet, Chart, Shape.



你可以使用VBA自动执行几乎所有可以交互（手动）完成的操作，还能提供Microsoft Office中没有的功能。VBA可以创建文档、添加文本、格式化、编辑并保存，所有这些都无需人工干预。

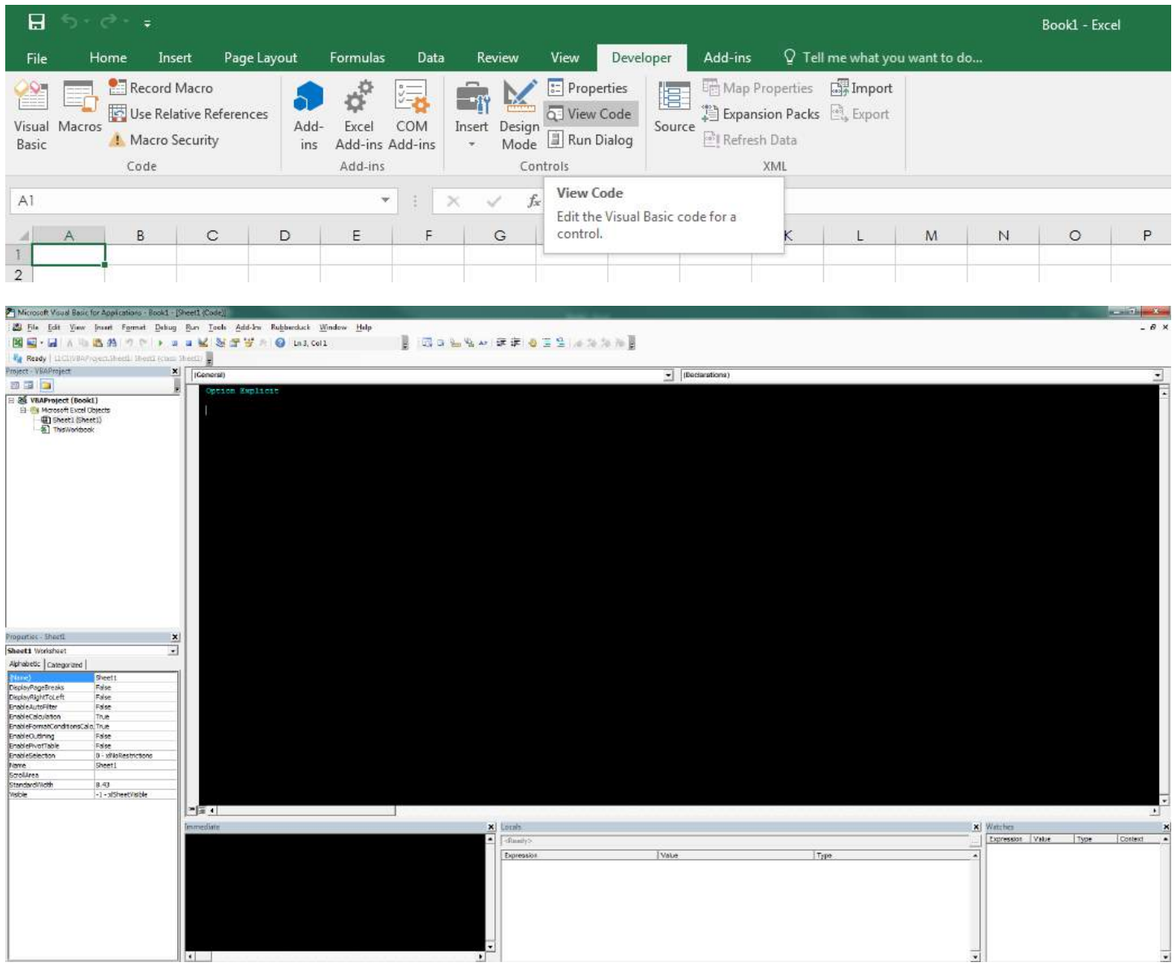
第1.2节：调试

调试是一种非常强大的方法，可以更仔细地查看并修复运行不正确（或无法运行）的代码。

逐步运行代码

调试时你首先需要做的是在特定位置停止代码，然后逐行运行，查看是否按预期执行。

- 断点（ F9 ，调试 - 切换断点）：你可以在任何执行的代码行添加断点（例如，不能添加在声明行），当执行到该点时会停止，并将控制权交给用户。
- 你也可以在空白行添加Stop关键字，使代码在运行时停在该位置。例如，在无法添加断点的声明行之前，这非常有用。 F9
- 单步进入（ F8 ，调试 - 单步进入）：只执行一行代码，如果该行是用户定义的子程序/函数调用，则逐行执行该子程序/函数。
- 单步跳过（ Shift + F8 ，调试 - 单步跳过）：执行一行代码，不进入用户定义的子程序/函数。
- 走出（ Ctrl + Shift + F8 调试 - 跳出（Debug - Step out）：退出当前子程序/函数（运行代码直到结束）。



You can use VBA to automate almost any action that can be performed interactively (manually) and also provide functionality that is not available in Microsoft Office. VBA can create a document, add text to it, format it, edit it, and save it, all without human intervention.

Section 1.2: Debugging

Debugging is a very powerful way to have a closer look and fix incorrectly working (or non working) code.

Run code step by step

First thing you need to do during debugging is to stop the code at specific locations and then run it line by line to see whether that happens what's expected.

- Breakpoint (F9 , Debug - Toggle breakpoint): You can add a breakpoint to any executed line (e.g. not to declarations), when execution reaches that point it stops, and gives control to user.
- You can also add the `Stop` keyword to a blank line to have the code stop at that location on runtime. This is useful if, for example, before declaration lines to which you can't add a breakpoint with F9
- Step into (F8 , Debug - Step into): executes only one line of code, if that's a call of a user defined sub / function, then that's executed line by line.
- Step over (Shift + F8 , Debug - Step over): executes one line of code, doesn't enter user defined subs / functions.
- Step out (Ctrl + Shift + F8 , Debug - Step out): Exit current sub / function (run code until its end).

- 运行到光标处（`Ctrl`+`F8`调试 - 运行到光标（Debug - Run to cursor））：运行代码直到到达光标所在行。
- 你可以使用Debug.Print在运行时将行打印到立即窗口。你也可以使用Debug.?作为Debug.Print的快捷方式

监视窗口

逐行运行代码只是第一步，我们需要了解更多细节，其中一个工具是监视窗口（视图 - 监视窗口），在这里你可以看到已定义表达式的值。要将变量添加到监视窗口，可以：

- 右键点击变量，然后选择“添加监视”。
- 在监视窗口中右键点击，选择“添加监视”。
- 进入调试 - 添加监视。

当你添加一个新表达式时，可以选择仅查看其值，或者在表达式为真或其值发生变化时中断代码执行。

立即窗口

立即窗口允许你执行任意代码或打印项目，只需在它们前面加上`Print`关键字或单个问号"?"

一些示例：

- `? ActiveSheet.Name` - 返回活动工作表的名称
- `Print ActiveSheet.Name` - 返回活动工作表的名称
- `? foo` - 返回foo的值*
- `x = 10` 将 x 设置为10*

* 通过立即窗口获取/设置变量的值只能在运行时进行

调试最佳实践

每当你的代码未按预期工作时，首先应仔细阅读代码，查找错误。

如果这无济于事，那么开始调试；对于简短的过程，逐行执行可能更有效，对于较长的过程，你可能需要设置断点或监视表达式的中断，目标是找到未按预期工作的代码行。

一旦你找到了给出错误结果的代码行，但原因尚不清楚，尝试简化表达式，
或用常量替换变量，这有助于判断变量的值是否错误。

如果你仍然无法解决问题，并寻求帮助：

- 尽可能只包含少量代码以便理解你的问题
- 如果问题与变量的值无关，则用常量替代它们。（例如，不是`Sheets(a*b*c+d^2).Range(addressOfRange)`而是写成`Sheets(4).Range("A2")`）
- 描述哪一行出现了错误行为，以及具体是什么（错误、错误结果等）

第1.3节：第一个模块和Hello World

要开始编写代码，首先需要在左侧列表中右键点击你的VBA项目并添加一个新模块。你的
第一个*Hello-World*代码可能如下所示：

- Run to cursor (`Ctrl`+`F8`, Debug - Run to cursor): run code until reaching the line with the cursor.
- You can use `Debug.Print` to print lines to the Immediate Window at runtime. You may also use `Debug.?` as a shortcut for `Debug.Print`

Watches window

Running code line by line is only the first step, we need to know more details and one tool for that is the watch window (View - Watch window), here you can see values of defined expressions. To add a variable to the watch window, either:

- Right-click on it then select "Add watch".
- Right-click in watch window, select "Add watch".
- Go to Debug - Add watch.

When you add a new expression you can choose whether you just want to see it's value, or also break code execution when it's true or when its value changes.

Immediate Window

The immediate window allows you to execute arbitrary code or print items by preceeding them with either the `Print` keyword or a single question mark "?"

Some examples:

- `? ActiveSheet.Name` - returns name of the active sheet
- `Print ActiveSheet.Name` - returns the name of the active sheet
- `? foo` - returns the value of `foo`*
- `x = 10` sets x to 10*

* Getting/Setting values for variables via the Immediate Window can only be done during runtime

Debugging best practices

Whenever your code doesn't work as expected first thing you should do is to read it again carefully, looking for mistakes.

If that doesn't help, then start debugging it; for short procedures it can be efficient to just execute it line by line, for longer ones you probably need to set breakpoints or breaks on watched expressions, the goal here is to find the line not working as expected.

Once you have the line which gives the incorrect result, but the reason is not yet clear, try to simplify expressions, or replace variables with constants, that can help understanding whether variables' value are wrong.

If you still can't solve it, and ask for help:

- Include as small part of your code as possible for understanding of your problem
- If the problem is not related to the value of variables, then replace them by constants. (so, instead of `Sheets(a*b*c+d^2).Range(addressOfRange)` write `Sheets(4).Range("A2")`)
- Describe which line gives the wrong behaviour, and what it is (error, wrong result...)

Section 1.3: First Module and Hello World

To start coding in the first place, you have to right click your VBA Project in the left list and add a new Module. Your first *Hello-World* Code could look like this:

```vba
Sub HelloWorld()
    MsgBox "Hello, World!"
End Sub
```

要测试它，点击工具栏上的播放按钮，或者直接按 F5 关键。恭喜！你已经创建了第一个自己的VBA模块。


```vba
Sub HelloWorld()
    MsgBox "Hello, World!"
End Sub
```


To test it, hit the *Play*-Button in your Toolbar or simply hit the F5 key. Congratulations! You've built your first own VBA Module.

第二章：注释

2.1节：撇号注释

注释以撇号（'）标记，代码执行时会被忽略。注释有助于向未来的读者（包括你自己）解释代码。

由于所有以注释开头的行都会被忽略，它们也可以用来防止代码执行（在你调试或重构时）。在代码前放置撇号'会将其变成注释。（这称为注释掉该行。）

```
Sub InlineDocumentation()
    '注释以“”开始

    '它们可以放在代码行前，防止该行执行
    'Debug.Print "Hello World"

    '它们也可以放在语句后面
    '语句仍会执行，直到编译器遇到注释
Debug.Print "Hello World"   '打印欢迎信息

'注释可以没有缩进.....
        '.....也可以有任意多的缩进

    "" 注释可以包含多个撇号 ""

    '注释可以跨行（使用行续符） _
但这会使代码难以阅读

    如果需要多行注释，通常更容易在每行使用一个撇号


    续行语法（:）被视为注释的一部分，因此
    无法在注释后放置可执行语句
    这段代码无法运行：Debug.Print "Hello World"
End Sub

注释可以出现在过程内或过程外
```

第2.2节：REM注释

```
子程序 RemComments()
    Rem 注释以"Rem" 开头（VBA会将任何大小写变体转换为"Rem"）
    Rem是 Remark的缩写，类似于DOS语法
    Rem是 添加注释的传统方法，建议优先使用撇号

    Rem 注释不能出现在语句后面，应该使用撇号语法
    Rem 除非它们前面有指令分隔符
    Debug.Print "Hello World": Rem 打印欢迎信息
Debug.Print "Hello World" '打印欢迎信息

    'Rem 不能紧跟以下字符 "!,@,#,$,%,&"'
    '而撇号语法后面可以跟任意可打印字符。'

End Sub

Rem 注释可以出现在过程内部或外部
```


Chapter 2: Comments

Section 2.1: Apostrophe Comments

A comment is marked by an apostrophe (`'`), and ignored when the code executes. Comments help explain your code to future readers, including yourself.

Since all lines starting with a comment are ignored, they can also be used to prevent code from executing (while you debug or refactor). Placing an apostrophe `'` before your code turns it into a comment. (This is called *commenting out* the line.)

```
Sub InlineDocumentation()
  'Comments start with an "'"

  'They can be place before a line of code, which prevents the line from executing
  'Debug.Print "Hello World"

  'They can also be placed after a statement
  'The statement still executes, until the compiler arrives at the comment
  Debug.Print "Hello World"  'Prints a welcome message

'Comments can have 0 indention....
    '... or as much as needed

  '''' Comments can contain multiple apostrophes ''''

  'Comments can span lines (using line continuations) _
    but this can make for hard to read code

  'If you need to have mult-line comments, it is often easier to
  'use an apostrophe on each line

  'The continued statement syntax (:) is treated as part of the comment, so
  'it is not possible to place an executable statement after a comment
  'This won't run : Debug.Print "Hello World"
End Sub

'Comments can appear inside or outside a procedure
```

Section 2.2: REM Comments

```
Sub RemComments()
  Rem Comments start with "Rem" (VBA will change any alternate casing to "Rem")
  Rem is an abbreviation of Remark, and similar to DOS syntax
  Rem Is a legacy approach to adding comments, and apostrophes should be preferred

  Rem Comments CANNOT appear after a statement, use the apostrophe syntax instead
  Rem Unless they are preceded by the instruction separator token
  Debug.Print "Hello World": Rem prints a welcome message
  Debug.Print "Hello World" 'Prints a welcome message

  'Rem cannot be immediately followed by the following characters "!,@,#,$,%,&"
  'Whereas the apostrophe syntax can be followed by any printable character.

End Sub

Rem Comments can appear inside or outside a procedure
```

第3章：字符串字面量 - 转义、不可打印字符和换行续行

第3.1节：转义 " 字符

VBA 语法要求字符串字面量必须出现在"标记内，因此当你的字符串需要*包含*引号时，你需要用额外的"字符转义/前置"字符，以便 VBA 理解你想要""被解释为一个"字符串。

```
'以下两行代码产生相同的输出'
Debug.Print "那个人说，""永远不要使用空气引号"""
Debug.Print "那个人说，" & """ & "永远不要使用空气引号" & """

'输出：'
'那个人说，"永远不要使用空气引号"'
'那个人说，"永远不要使用空气引号"'
```

第3.2节：分配长字符串字面量

VBA编辑器每行只允许1023个字符，但通常只有前100-150个字符在不滚动的情况下可见。如果你需要赋值很长的字符串字面量，但又想保持代码的可读性，就需要使用行连接符和字符串连接来赋值。

```
Debug.Print "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " & _
            "Integer hendrerit maximus arcu, ut elementum odio varius " & _
            "nec. Integer ipsum enim, iaculis et egestas ac, condiment" & _
            "um ut tellus."
'输出：'
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer hendrerit maximus arcu, ut
elementum odio varius nec. Integer ipsum enim, iaculis et egestas ac, condimentum ut tellus.
```

VBA允许你使用有限数量的行连接符（实际数量取决于续行块中每行的长度），所以如果你有非常长的字符串，就需要通过连接赋值和重新赋值。

```
Dim loremIpsum As String

'赋值字符串的第一部分
loremIpsum = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " & _
             "Integer hendrerit maximus arcu, ut elementum odio varius "
'用之前的值和字符串的下一部分重新赋值
loremIpsum = loremIpsum & _
             "nec. Integer ipsum enim, iaculis et egestas ac, condiment" & _
             "um ut tellus."

Debug.Print loremIpsum

'输出：'
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer hendrerit maximus arcu, ut
elementum odio varius nec. Integer ipsum enim, iaculis et egestas ac, condimentum ut tellus.
```

第3.3节：使用VBA字符串常量

VBA 定义了许多特殊字符的字符串常量，例如：

Chapter 3: String Literals - Escaping, non-printable characters and line-continuations

Section 3.1: Escaping the " character

VBA syntax requires that a string-literal appear within `"` marks, so when your string needs to *contain* quotation marks, you'll need to escape/prepend the `"` character with an extra `"` so that VBA understands that you intend the `""` to be interpreted as a `"` string.

```
'The following 2 lines produce the same output
Debug.Print "The man said, ""Never use air-quotes"""
Debug.Print "The man said, " & """ & "Never use air-quotes" & """

'Output:
'The man said, "Never use air-quotes"
'The man said, "Never use air-quotes"
```

Section 3.2: Assigning long string literals

The VBA editor only allows 1023 characters per line, but typically only the first 100-150 characters are visible without scrolling. If you need to assign long string literals, but you want to keep your code readable, you'll need to use line-continuations and concatenation to assign your string.

```
Debug.Print "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " & _
            "Integer hendrerit maximus arcu, ut elementum odio varius " & _
            "nec. Integer ipsum enim, iaculis et egestas ac, condiment" & _
            "um ut tellus."
'Output:
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer hendrerit maximus arcu, ut
elementum odio varius nec. Integer ipsum enim, iaculis et egestas ac, condimentum ut tellus.
```

VBA will let you use a limited number of line-continuations (the actual number varies by the length of each line within the continued-block), so if you have very long strings, you'll need to assign and re-assign with concatenation.

```
Dim loremIpsum As String

'Assign the first part of the string
loremIpsum = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " & _
             "Integer hendrerit maximus arcu, ut elementum odio varius "
'Re-assign with the previous value AND the next section of the string
loremIpsum = loremIpsum & _
             "nec. Integer ipsum enim, iaculis et egestas ac, condiment" & _
             "um ut tellus."

Debug.Print loremIpsum

'Output:
'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer hendrerit maximus arcu, ut
elementum odio varius nec. Integer ipsum enim, iaculis et egestas ac, condimentum ut tellus.
```

Section 3.3: Using VBA string constants

VBA defines a number of string constants for special characters like:

- vbCr ：回车符，等同于 C 语言风格中的 "\r"。
- vbLf ：换行符，等同于 C 语言风格中的 ""。
- vbCrLf ：回车符和换行符（Windows 中的新行）
- vbTab：制表符
- vbNullString：空字符串，类似于 ""

你可以使用这些常量结合连接和其他字符串函数来构建包含特殊字符的字符串字面量。

```
Debug.Print "Hello " & vbCrLf & "World"
'输出：'
'Hello
'World

Debug.Print vbTab & "Hello" & vbTab & "World"
'输出：'
'你好    世界

Dim EmptyString As String
EmptyString = vbNullString
Debug.Print EmptyString = ""
'输出：'
'True
```

使用 vbNullString 被认为比使用等价的 "" 更好，因为两者在代码编译方式上存在差异。字符串是通过指向分配内存区域的指针来访问的，VBA 编译器足够智能，能够使用空指针来表示 vbNullString。字面量 "" 会被分配内存，就像它是一个字符串类型的 Variant，因此使用该常量更高效：

```
Debug.Print StrPtr(vbNullString)    '打印 0。
Debug.Print StrPtr("")              '打印一个内存地址。
```


- vbCr : Carriage-Return 'Same as "\r" in C style languages.
- vbLf : Line-Feed 'Same as "\n" in C style languages.
- vbCrLf : Carriage-Return & Line-Feed (a new-line in Windows)
- vbTab: Tab Character
- vbNullString: an empty string, like ""

You can use these constants with concatenation and other string functions to build string-literals with special-characters.

```
Debug.Print "Hello " & vbCrLf & "World"
'Output:
'Hello
'World

Debug.Print vbTab & "Hello" & vbTab & "World"
'Output:
'    Hello    World

Dim EmptyString As String
EmptyString = vbNullString
Debug.Print EmptyString = ""
'Output:
'True
```

Using `vbNullString` is considered better practice than the equivalent value of `""` due to differences in how the code is compiled. Strings are accessed via a pointer to an allocated area of memory, and the VBA compiler is smart enough to use a null pointer to represent `vbNullString`. The literal `""` is allocated memory as if it were a String typed Variant, making the use of the constant much more efficient:

```
Debug.Print StrPtr(vbNullString)    'Prints 0.
Debug.Print StrPtr("")              'Prints a memory address.
```

第4章：VBA Option 关键字

Option	详细
Explicit	要求在指定的模块中声明变量（理想情况下是所有模块）；启用此选项后，使用未声明（或拼写错误）的变量将导致编译错误。
比较文本	使模块的字符串比较不区分大小写，基于系统区域设置，优先考虑字母等价（例如 "a" = "A"）。
比较二进制	默认的字符串比较模式。使模块的字符串比较区分大小写，使用每个字符的二进制表示/数值进行字符串比较（例如 ASCII）。
比较数据库（仅限 MS-Access）	使模块的字符串比较方式与 SQL 语句中的行为一致。
私有模块	防止模块的Public成员被项目外部访问该模块所在的位置，有效地将过程隐藏于主机应用程序之外（即不可用作宏或用户自定义函数）。
选项 基础 0	默认设置。在模块中将隐式数组下界设置为0。当声明数组时未指定显式下界值，将使用0。
选项 基础 1	在模块中将隐式数组下界设置为1。当声明数组时未指定显式下界值，将使用1。

第4.1节：Option Explicit

在VBA中始终使用Option Explicit被认为是最佳实践，因为它强制开发者在使用变量前声明所有变量。这还有其他好处，例如已声明变量名的自动大写和IntelliSense。

```
Option Explicit

Sub OptionExplicit()
    Dim a As Integer
a = 5
b = 10 '// 导致编译错误，因为未声明变量 'b'
End Sub
```

在 VBE 的工具 ▸ 选项 ▸ 编辑器 属性页中设置 要求变量声明 将会在每个新创建的代码表顶部自动添加 **Option Explicit** 语句。

Chapter 4: VBA Option Keyword

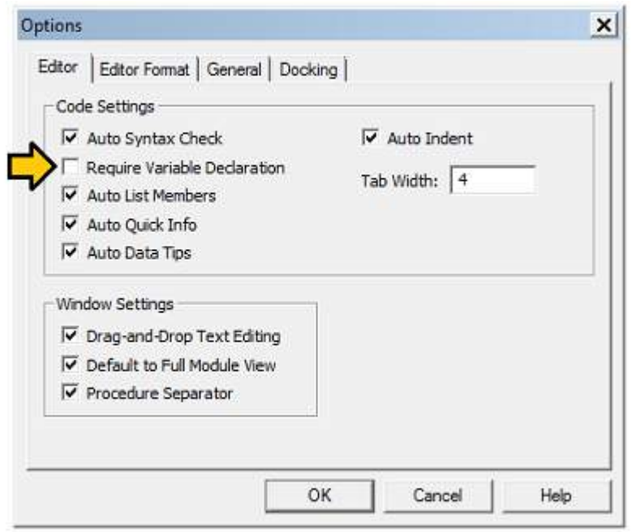
Option	Detail
Explicit	*Require variable declaration* in the module it's specified in (ideally all of them); with this option specified, using an undeclared (/mispelled) variable becomes a compilation error.
Compare Text	Makes the module's string comparisons be case-insensitive, based on system locale, prioritizing alphabetical equivalency (e.g. "a" = "A").
Compare Binary	Default string comparison mode. Makes the module's string comparisons be case sensitive, comparing strings using the binary representation / numeric value of each character (e.g. ASCII).
Compare Database	(MS-Access only) Makes the module's string comparisons work the way they would in an SQL statement.
Private Module	Prevents the module's **Public** member from being accessed from outside of the project that the module resides in, effectively hiding procedures from the host application (i.e. not available to use as macros or user-defined functions).
Option Base 0	Default setting. Sets the implicit array lower bound to `0` in a module. When an array is declared without an explicit lower boundary value, `0` will be used.
Option Base 1	Sets the implicit array lower bound to `1` in a module. When an array is declared without an explicit lower boundary value, `1` will be used.

Section 4.1: Option Explicit

It is deemed best practice to always use **Option** `Explicit` in VBA as it forces the developer to declare all their variables before use. This has other benefits too, such as auto-capitalization for declared variable names and IntelliSense.

```
Option Explicit

Sub OptionExplicit()
    Dim a As Integer
    a = 5
    b = 10 '// Causes compile error as 'b' is not declared
End Sub
```

Setting **Require Variable Declaration** within the VBE's Tools ▸ Options ▸ Editor property page will put the **Option Explicit** statement at the top of each newly created code sheet.



这将避免拼写错误等愚蠢的编码错误，并且促使你在变量声明中使用正确的变量类型。（更多示例见“始终使用 Option Explicit”。）

第4.2节：Option Base {0 | 1}

Option Base 用于声明数组元素的默认下界。它在模块级别声明，仅对当前模块有效。

默认情况下（即未指定 Option Base 时），下界为0。这意味着模块中声明的任何数组的第一个元素索引为0。

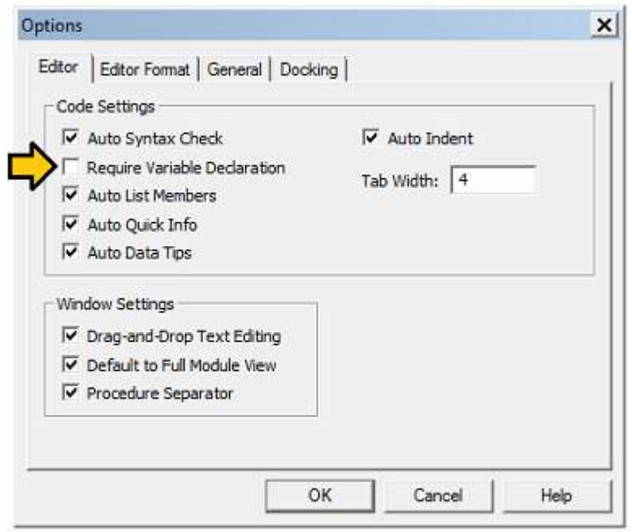
如果指定了 `Option Base 1`，则第一个数组元素的索引为1

Base 0 的示例：

```
Option Base 0

Sub BaseZero()

    Dim myStrings As Variant

    ' 将 Variant 创建为包含3个水果元素的数组
    myStrings = Array("Apple", "Orange", "Peach")

    Debug.Print LBound(myStrings) ' 输出 "0"
    Debug.Print UBound(myStrings) ' 输出 "2"，因为有3个元素，索引从0开始 ->
0,1,2

    对于 i = 0 到 UBound(myStrings)

Debug.Print myStrings(i) ' 这将依次打印 "Apple"、"Orange"、"Peach"

    Next i

End Sub
```

相同示例，基数为1

```
Option Base 1
```

This will avoid silly coding mistakes like misspellings as well as influencing you to use the correct variable type in the variable declaration. (Some more examples are given at ALWAYS Use "Option Explicit".)

Section 4.2: Option Base {0 | 1}

`Option Base` is used to declare the default lower bound of **array** elements. It is declared at module level and is valid only for the current module.

By default (and thus if no Option Base is specified), the Base is 0. Which means that the first element of any array declared in the module has an index of 0.

If `Option Base 1` is specified, the first array element has the index 1

Example in Base 0 :

```
Option Base 0

Sub BaseZero()

    Dim myStrings As Variant

    ' Create an array out of the Variant, having 3 fruits elements
    myStrings = Array("Apple", "Orange", "Peach")

    Debug.Print LBound(myStrings) ' This Prints "0"
    Debug.Print UBound(myStrings) ' This print "2", because we have 3 elements beginning at 0 ->
0,1,2

    For i = 0 To UBound(myStrings)

        Debug.Print myStrings(i) ' This will print "Apple", then "Orange", then "Peach"

    Next i

End Sub
```

Same Example with Base 1

```
Option Base 1
```

```
Sub BaseOne()

    Dim myStrings As Variant

    ' 从Variant创建一个包含3个水果元素的数组
    myStrings = Array("Apple", "Orange", "Peach")

    Debug.Print LBound(myStrings) ' 这将打印 "1"
    Debug.Print UBound(myStrings) ' 这将打印 "3"，因为我们有3个元素，起始索引为1 ->
1,2,3

    对于 i = 0 到 UBound(myStrings)

Debug.Print myStrings(i) ' 这会触发错误9“下标越界”

    Next i

End Sub
```

第二个示例在第一次循环阶段产生了下标越界（错误9），因为尝试访问数组的索引0，而该索引不存在，因为模块声明为Base 1

基数为1时的正确代码是：

```
    对于 i = 1 到 UBound(myStrings)

Debug.Print myStrings(i) ' 这将依次打印 "Apple"、"Orange"、"Peach"

    Next i
```

需要注意的是，Split 函数**总是**创建一个以零为基准的元素索引的数组，无论任何 `Option Base` 设置如何。关于如何使用**Split**函数的示例可以在这里找到

> Split 函数
>
> 返回一个以零为基准的一维数组，包含指定数量的子字符串。

在 Excel 中，`Range.Value` 和 `Range.Formula` 属性对于多单元格范围*总是*返回一个以1为基准的二维 Variant 数组。

同样，在 ADO 中，Recordset.GetRows 方法总是返回一个以1为基准的二维数组。

一个推荐的“最佳实践”是始终使用LBound和UBound函数来确定数组的范围。

```
'对于一维数组
Debug.Print LBound(arr) & ":" & UBound(arr)
声明 i 为 Long 类型
For i = LBound(arr) To UBound(arr)
    Debug.Print arr(i)
Next i

二维数组的示例
Debug.Print LBound(arr, 1) & ":" & UBound(arr, 1)
Debug.Print LBound(arr, 2) & ":" & UBound(arr, 2)
Dim i As Long, j As Long
For i = LBound(arr, 1) To UBound(arr, 1)
    For j = LBound(arr, 2) To UBound(arr, 2)
        Debug.Print arr(i, j)
    Next j
```



```
Sub BaseOne()

    Dim myStrings As Variant

    ' Create an array out of the Variant, having 3 fruits elements
    myStrings = Array("Apple", "Orange", "Peach")

    Debug.Print LBound(myStrings) ' This Prints "1"
    Debug.Print UBound(myStrings) ' This print "3", because we have 3 elements beginning at 1 ->
1,2,3

    For i = 0 To UBound(myStrings)

        Debug.Print myStrings(i) ' This triggers an error 9 "Subscript out of range"

    Next i

End Sub
```

The second example generated a Subscript out of range (Error 9) at the first loop stage because an attempt to access the index 0 of the array was made, and this index doesn't exists as the module is declared with `Base 1`

The correct code with Base 1 is :

```
    For i = 1 To UBound(myStrings)

        Debug.Print myStrings(i) ' This will print "Apple", then "Orange", then "Peach"

    Next i
```

It should be noted that the Split function **always** creates an array with a zero-based element index regardless of any `Option Base` setting. Examples on how to use the **Split** function can be found here

> Split Function
>
> Returns a zero-based, one-dimensional array containing a specified number of substrings.

In Excel, the `Range.Value` and `Range.Formula` properties for a multi-celled range *always* returns a 1-based 2D Variant array.

Likewise, in ADO, the `Recordset.GetRows` method *always* returns a 1-based 2D array.

One recommended 'best practice' is to always use the LBound and UBound functions to determine the extents of an array.

```
'for single dimensioned array
Debug.Print LBound(arr) & ":" & UBound(arr)
Dim i As Long
For i = LBound(arr) To UBound(arr)
    Debug.Print arr(i)
Next i

'for two dimensioned array
Debug.Print LBound(arr, 1) & ":" & UBound(arr, 1)
Debug.Print LBound(arr, 2) & ":" & UBound(arr, 2)
Dim i As long, j As Long
For i = LBound(arr, 1) To UBound(arr, 1)
    For j = LBound(arr, 2) To UBound(arr, 2)
        Debug.Print arr(i, j)
    Next j
```

```
Next i
```

如果要使数组始终以下界为1创建，则必须在每个创建或重新定义数组的代码模块顶部放置 Option Base 1。

第4.3节：Option Compare {Binary | Text | Database}

Option Compare Binary

二进制比较使模块/类中的所有字符串相等检查区分大小写。技术上讲，使用此选项时，字符串比较是根据每个字符的二进制表示的排序顺序进行的。

> A < B < E < Z < a < b < e < z

如果模块中未指定 Option Compare，则默认使用 Binary。

```
Option Compare Binary

Sub CompareBinary()

    Dim foo As String
    Dim bar As String

    '// 区分大小写
foo = "abc"
bar = "ABC"

Debug.Print (foo = bar) '// 输出 "False"

    '// 仍然区分带重音符号的字符
foo = "ábc"
bar = "abc"

Debug.Print (foo = bar) '// 输出 "False"

    '// "b" (Chr 98) 大于 "a" (Chr 97)
foo = "a"
bar = "b"

Debug.Print (bar > foo) '// 输出 "True"

    '// "b" (Chr 98) 不大于 "á" (Chr 225)
foo = "á"
bar = "b"

Debug.Print (bar > foo) '// 输出 "False"

End Sub
```

Option Compare Text

Option Compare Text 使模块/类中的所有字符串比较都使用不区分大小写的比较。

> (A | a) < (B | b) < (Z | z)


```
Next i
```

The `Option Base 1` must be at the top of every code module where an array is created or re-dimensioned if arrays are to be consistently created with an lower boundary of 1.

Section 4.3: Option Compare {Binary | Text | Database}

Option Compare Binary

Binary comparison makes all checks for string equality within a module/class case *sensitive*. Technically, with this option, string comparisons are performed using sort order of the binary representations of each character.

> A < B < E < Z < a < b < e < z

If no Option Compare is specified in a module, Binary is used by default.

```
Option Compare Binary

Sub CompareBinary()

    Dim foo As String
    Dim bar As String

    '// Case sensitive
    foo = "abc"
    bar = "ABC"

    Debug.Print (foo = bar) '// Prints "False"

    '// Still differentiates accented characters
    foo = "ábc"
    bar = "abc"

    Debug.Print (foo = bar) '// Prints "False"

    '// "b" (Chr 98) is greater than "a" (Chr 97)
    foo = "a"
    bar = "b"

    Debug.Print (bar > foo) '// Prints "True"

    '// "b" (Chr 98) is NOT greater than "á" (Chr 225)
    foo = "á"
    bar = "b"

    Debug.Print (bar > foo) '// Prints "False"

End Sub
```

Option Compare Text

Option Compare Text makes all string comparisons within a module/class use a case *insensitive* comparison.

> (A | a) < (B | b) < (Z | z)

```
Option Compare Text

Sub CompareText()

    Dim foo As String
    Dim bar As String

    '// 不区分大小写
foo = "abc"
bar = "ABC"

Debug.Print (foo = bar) '// 输出 "True"

    '// 仍然区分带重音符号的字符
foo = "ábc"
bar = "abc"

Debug.Print (foo = bar) '// 输出 "False"

    '// "b" 仍然排在 "a" 或 "á" 之后
foo = "á"
bar = "b"

Debug.Print (bar > foo) '// 输出 "True"

End Sub
```

Option Compare Database

Option Compare Database 仅在 MS Access 中可用。它设置模块/类使用当前数据库设置来确定是使用文本模式还是二进制模式。

注意：除非模块用于编写自定义 Access UDF（用户定义函数），且需要以与该数据库中 SQL 查询相同的方式处理文本比较，否则不建议使用此设置。


```
Option Compare Text

Sub CompareText()

    Dim foo As String
    Dim bar As String

    '// Case insensitivity
    foo = "abc"
    bar = "ABC"

    Debug.Print (foo = bar) '// Prints "True"

    '// Still differentiates accented characters
    foo = "ábc"
    bar = "abc"

    Debug.Print (foo = bar) '// Prints "False"

    '// "b" still comes after "a" or "á"
    foo = "á"
    bar = "b"

    Debug.Print (bar > foo) '// Prints "True"

End Sub
```

Option Compare Database

Option Compare Database is only available within MS Access. It sets the module/class to use the current database settings to determine whether to use Text or Binary mode.

Note: The use of this setting is discouraged unless the module is used for writing custom Access UDFs (User defined functions) that should treat text comparisons in the same manner as SQL queries in that database.

第5章：声明变量

第5.1节：类型提示

类型提示强烈不建议使用。它们存在且在此处记录，主要是出于历史和向后兼容的原因。你应该改用As [数据类型]语法。

```
Public Sub ExampleDeclaration()

        Dim someInteger% '% 等同于 "As Integer"
        Dim someLong&    '& 等同于 "As Long"
        Dim someDecimal@ '@ 等同于 "As Currency"
        Dim someSingle!  '! 等同于 "As Single"
        Dim someDouble#  '# 等同于 "As Double"
        Dim someString$  '$ 等同于 "As String"

        Dim someLongLong^  '^ 在64位VBA环境中等同于 "As LongLong"
End Sub
```

类型提示显著降低代码可读性，并且鼓励使用一种遗留的匈牙利命名法，这也阻碍了可读性：

```
Dim strFile$
Dim iFile%
```

相反，应在变量使用的地方附近声明变量，并根据用途命名，而不是根据类型命名：

```
Dim path As String
Dim handle As Integer
```

类型提示也可以用于字面量，以强制指定类型。默认情况下，小于32,768的数字字面量会被解释为`Integer`字面量，但通过类型提示可以控制这一点：

```
Dim foo '隐式 Variant
foo = 42& ' foo 现在是 Long 类型
foo = 42# ' foo 现在是 Double 类型
Debug.Print TypeName(42!) ' 输出 "Single"
```

通常字面量不需要类型提示，因为它们会被赋值给显式声明类型的变量，或者在作为参数传递时隐式转换为适当的类型。可以使用显式类型转换函数来避免隐式转换：

```
'调用过程 DoSomething 并使用类型提示将字面量 42 作为 Long 传递
DoSomething 42&

调用过程 DoSomething 并传递一个显式转换为 Long 类型的字面量 42
DoSomething CLng(42)
```

返回字符串的内置函数

大多数处理字符串的内置函数有两个版本：一个是返回 Variant 类型的弱类型版本，另一个是以 $ 结尾的强类型版本，返回 String 类型。除非你将返回值赋给 Variant，否则应优先使用返回 String 的版本——否则返回值会被隐式转换。

Chapter 5: Declaring Variables

Section 5.1: Type Hints

Type Hints are **heavily** discouraged. They exist and are documented here for historical and backward-compatibility reasons. You should use the `As [DataType]` syntax instead.

```
Public Sub ExampleDeclaration()

        Dim someInteger% '% Equivalent to "As Integer"
        Dim someLong&    '& Equivalent to "As Long"
        Dim someDecimal@ '@ Equivalent to "As Currency"
        Dim someSingle!  '! Equivalent to "As Single"
        Dim someDouble#  '# Equivalent to "As Double"
        Dim someString$  '$ Equivalent to "As String"

        Dim someLongLong^  '^ Equivalent to "As LongLong" in 64-bit VBA hosts
End Sub
```

Type hints significantly decrease code readability and encourage a legacy Hungarian Notation which *also* hinders readability:

```
Dim strFile$
Dim iFile%
```

Instead, declare variables closer to their usage and name things for what they're used, not after their type:

```
Dim path As String
Dim handle As Integer
```

Type hints can also be used on literals, to enforce a specific type. By default, a numeric literal smaller than 32,768 will be interpreted as an `Integer` literal, but with a type hint you can control that:

```
Dim foo 'implicit Variant
foo = 42& ' foo is now a Long
foo = 42# ' foo is now a Double
Debug.Print TypeName(42!) ' prints "Single"
```

Type hints are usually not needed on literals, because they would be assigned to a variable declared with an explicit type, or implicitly converted to the appropriate type when passed as parameters. Implicit conversions can be avoided using one of the explicit type conversion functions:

```
'Calls procedure DoSomething and passes a literal 42 as a Long using a type hint
DoSomething 42&

'Calls procedure DoSomething and passes a literal 42 explicitly converted to a Long
DoSomething CLng(42)
```

String-returning built-in functions

The majority of the built-in functions that handle strings come in two versions: A loosely typed version that returns a `Variant`, and a strongly typed version (ending with $) that returns a `String`. Unless you are assigning the return value to a `Variant`, you should prefer the version that returns a `String` - otherwise there is an implicit conversion of the return value.

```
Debug.Print Left(foo, 2)  'Left 返回 Variant
Debug.Print Left$(foo, 2) 'Left$ 返回 String
```

这些函数包括：

- VBA.Conversion.Error -> VBA.Conversion.Error$
- VBA.Conversion.Hex -> VBA.Conversion.Hex$
- VBA.Conversion.Oct -> VBA.Conversion.Oct$
- VBA.Conversion.Str -> VBA.Conversion.Str$
- VBA.FileSystem.CurDir -> VBA.FileSystem.CurDir$
- VBA.[_HiddenModule].Input -> VBA.[_HiddenModule].Input$
- VBA.[_HiddenModule].InputB -> VBA.[_HiddenModule].InputB$
- VBA.Interaction.Command -> VBA.Interaction.Command$
- VBA.Interaction.Environ -> VBA.Interaction.Environ$
- VBA.Strings.Chr -> VBA.Strings.Chr$
- VBA.Strings.ChrB -> VBA.Strings.ChrB$
- VBA.Strings.ChrW -> VBA.Strings.ChrW$
- VBA.Strings.Format -> VBA.Strings.Format$
- VBA.Strings.LCase -> VBA.Strings.LCase$
- VBA.Strings.Left -> VBA.Strings.Left$
- VBA.Strings.LeftB -> VBA.Strings.LeftB$
- VBA.Strings.LTtrim -> VBA.Strings.LTrim$
- VBA.Strings.Mid -> VBA.Strings.Mid$
- VBA.Strings.MidB -> VBA.Strings.MidB$
- VBA.Strings.Right -> VBA.Strings.Right$
- VBA.Strings.RightB -> VBA.Strings.RightB$
- VBA.Strings.RTrim -> VBA.Strings.RTrim$
- VBA.Strings.Space -> VBA.Strings.Space$
- VBA.Strings.Str -> VBA.Strings.Str$
- VBA.Strings.String -> VBA.Strings.String$
- VBA.Strings.Trim -> VBA.Strings.Trim$
- VBA.Strings.UCase -> VBA.Strings.UCase$

注意，这些是函数别名，而不完全是类型提示。Left 函数对应隐藏的 B_Var_Left 函数，而 Left$ 版本对应隐藏的 B_Str_Left 函数。

在 VBA 的早期版本中，$符号不是允许的字符，函数名必须用方括号括起来。在 Word Basic 中，有更多返回字符串且以 $ 结尾的函数。

第5.2节：变量

作用域

变量可以按可见性级别递增的顺序声明：

- 在过程级别，使用Dim关键字在任何过程内；称为局部变量。
- 在模块级别，使用Private关键字在任何类型的模块中；称为私有字段。
- 在实例级别，使用Friend关键字在任何类型的类模块中；称为友元字段。
- 在实例级别，使用Public关键字在任何类型的类模块中；称为公共字段。
- 全局范围内，使用Public关键字在标准模块中；称为全局变量。

变量应始终以尽可能小的作用域声明：优先通过参数传递给过程，而不是声明全局变量。


```
Debug.Print Left(foo, 2)  'Left returns a Variant
Debug.Print Left$(foo, 2) 'Left$ returns a String
```

These functions are:

- VBA.Conversion.Error -> VBA.Conversion.Error$
- VBA.Conversion.Hex -> VBA.Conversion.Hex$
- VBA.Conversion.Oct -> VBA.Conversion.Oct$
- VBA.Conversion.Str -> VBA.Conversion.Str$
- VBA.FileSystem.CurDir -> VBA.FileSystem.CurDir$
- VBA.[_HiddenModule].Input -> VBA.[_HiddenModule].Input$
- VBA.[_HiddenModule].InputB -> VBA.[_HiddenModule].InputB$
- VBA.Interaction.Command -> VBA.Interaction.Command$
- VBA.Interaction.Environ -> VBA.Interaction.Environ$
- VBA.Strings.Chr -> VBA.Strings.Chr$
- VBA.Strings.ChrB -> VBA.Strings.ChrB$
- VBA.Strings.ChrW -> VBA.Strings.ChrW$
- VBA.Strings.Format -> VBA.Strings.Format$
- VBA.Strings.LCase -> VBA.Strings.LCase$
- VBA.Strings.Left -> VBA.Strings.Left$
- VBA.Strings.LeftB -> VBA.Strings.LeftB$
- VBA.Strings.LTtrim -> VBA.Strings.LTrim$
- VBA.Strings.Mid -> VBA.Strings.Mid$
- VBA.Strings.MidB -> VBA.Strings.MidB$
- VBA.Strings.Right -> VBA.Strings.Right$
- VBA.Strings.RightB -> VBA.Strings.RightB$
- VBA.Strings.RTrim -> VBA.Strings.RTrim$
- VBA.Strings.Space -> VBA.Strings.Space$
- VBA.Strings.Str -> VBA.Strings.Str$
- VBA.Strings.String -> VBA.Strings.String$
- VBA.Strings.Trim -> VBA.Strings.Trim$
- VBA.Strings.UCase -> VBA.Strings.UCase$

Note that these are function *aliases*, not quite *type hints*. The `Left` function corresponds to the hidden `B_Var_Left` function, while the `Left$` version corresponds to the hidden `B_Str_Left` function.

In very early versions of VBA the $ sign isn't an allowed character and the function name had to be enclosed in square brackets. In Word Basic, there were many, many more functions that returned strings that ended in $.

Section 5.2: Variables

Scope

A variable can be declared (in increasing visibility level):

- At procedure level, using the **`Dim`** keyword in any procedure; a *local variable*.
- At module level, using the **`Private`** keyword in any type of module; a *private field*.
- At instance level, using the **`Friend`** keyword in any type of class module; a *friend field*.
- At instance level, using the **`Public`** keyword in any type of class module; a *public field*.
- Globally, using the **`Public`** keyword in a *standard module*; a *global variable*.

Variables should always be declared with the smallest possible scope: prefer passing parameters to procedures, rather than declaring global variables.

有关更多信息，请参见访问修饰符。

局部变量

使用Dim关键字声明一个局部变量：

```
Dim 标识符名称 [As 类型][, 标识符名称 [As 类型], ...]
```

声明语法中的[As 类型]部分是可选的。指定时，它设置变量的数据类型，决定为该变量分配多少内存。这声明了一个String变量：

```
Dim 标识符名称 As String
```

当未指定类型时，类型隐式为Variant：

```
Dim 标识符名称 'As Variant 是隐式的
```

VBA语法还支持在一条语句中声明多个变量：

```
Dim someString As String, someVariant, someValue As Long
```

注意[As 类型]必须为每个变量指定（除了'Variant'类型的变量）。这是一个相对常见的陷阱：

```
Dim integer1, integer2, integer3 As Integer '只有 integer3 是整数。
                                            '其余的是变量。
```

静态变量

局部变量也可以是静态的。在VBA中，使用Static关键字使变量“记住”上次过程调用时的值：

```
Private Sub DoSomething()
    Static values As Collection
    If values Is Nothing Then
        Set values = New Collection
        values.Add "foo"
        values.Add "bar"
    End If
    DoSomethingElse values
End Sub
```

这里values集合被声明为静态局部变量；因为它是一个对象变量，所以初始化为Nothing。
声明后的条件判断验证对象引用是否已被Set——如果是过程第一次运行，集合将被初始化。DoSomethingElse可能会添加或删除项目，下次调用DoSomething时，这些项目仍然会在集合中。

替代方案

> VBA的Static关键字很容易被误解——尤其是对于通常在其他语言中工作的资深程序员。在许多语言中，static用于使类成员（字段、属性、方法等）属于类型而非实例。静态上下文中的代码不能引用实例上下文中的代码。VBA的Static关键字含义完全不同。

See Access Modifiers for more information.

Local variables

Use the `Dim` keyword to declare a *local variable*:

```
Dim identifierName [As Type][, identifierName [As Type], ...]
```

The `[As Type]` part of the declaration syntax is optional. When specified, it sets the variable's data type, which determines how much memory will be allocated to that variable. This declares a `String` variable:

```
Dim identifierName As String
```

When a type is not specified, the type is implicitly `Variant`:

```
Dim identifierName 'As Variant is implicit
```

The VBA syntax also supports declaring multiple variables in a single statement:

```
Dim someString As String, someVariant, someValue As Long
```

Notice that the `[As Type]` has to be specified for each variable (other than 'Variant' ones). This is a relatively common trap:

```
Dim integer1, integer2, integer3 As Integer 'Only integer3 is an Integer.
                                            'The rest are Variant.
```

Static variables

Local variables can also be `Static`. In VBA the `Static` keyword is used to make a variable "remember" the value it had, last time a procedure was called:

```
Private Sub DoSomething()
    Static values As Collection
    If values Is Nothing Then
        Set values = New Collection
        values.Add "foo"
        values.Add "bar"
    End If
    DoSomethingElse values
End Sub
```

Here the `values` collection is declared as a `Static` local; because it's an *object variable*, it is initialized to `Nothing`. The condition that follows the declaration verifies if the object reference was `Set` before - if it's the first time the procedure runs, the collection gets initialized. `DoSomethingElse` might be adding or removing items, and they'll still be in the collection next time `DoSomething` is called.

Alternative

> VBA's `Static` keyword can easily be misunderstood - *especially* by seasoned programmers that usually work in other languages. In many languages, `static` is used to make a class member (field, property, method, ...) belong to the *type* rather than to the *instance*. Code in `static` context cannot reference code in *instance* context. The VBA `Static` keyword means something wildly different.

通常，一个Static局部变量同样可以实现为Private的模块级变量（字段）——然而这会挑战变量应以最小可能作用域声明的原则；相信你的直觉，使用你喜欢的任意一种——两者都能工作……但在不了解Static的作用时使用它，可能会导致有趣的错误。

Dim 与 Private

Dim关键字在过程级和模块级都是合法的；它在模块级的用法等同于使用Private关键字：

```
Option Explicit
Dim privateField1 As Long '与 Private privateField2 as Long 相同
Private privateField2 As Long '与 Dim privateField2 as Long 相同
```

Private关键字仅在模块级合法；这促使人们保留Dim用于局部变量，并用Private声明模块变量，尤其是与必须使用的对比鲜明的Public关键字来声明公共成员。或者在任何地方都使用Dim——重要的是一致性：

“私有字段”

- 应使用Private来声明模块级变量。
- 应使用Dim来声明局部变量。
- 不应使用Dim来声明模块级变量。

到处都是Dim

- 请使用Dim来声明任何私有/局部变量。
- 不要使用Private来声明模块级变量。
- 避免声明Public字段。*

*一般来说，应避免声明Public或Global字段。

字段

在模块级别、模块主体顶部的声明部分声明的变量称为字段。在标准模块中声明的Public字段是全局变量：

```
Public PublicField As Long
```

具有全局作用域的变量可以从任何地方访问，包括引用其所在项目的其他VBA项目。

要使变量成为全局/公共，但仅在项目内部可见，请使用Friend修饰符：

```
Friend FriendField As Long
```

这在加载项中特别有用，因为其目的是让其他VBA项目引用加载项项目并能够使用公共API。

```
Friend FriendField As Long '项目内公共，也称为“友元”代码
Public PublicField As Long '项目内外公共
```

Friend字段在标准模块中不可用。

Often, a `Static` local could just as well be implemented as a `Private`, module-level variable (field) - however this challenges the principle by which a variable should be declared with the smallest possible scope; trust your instincts, use whichever you prefer - both will work... but using `Static` without understanding what it does could lead to interesting bugs.

Dim vs. Private

The `Dim` keyword is legal at procedure and module levels; its usage at module level is equivalent to using the `Private` keyword:

```
Option Explicit
Dim privateField1 As Long 'same as Private privateField2 as Long
Private privateField2 As Long 'same as Dim privateField2 as Long
```

The `Private` keyword is only legal at module level; this invites reserving `Dim` for local variables and declaring module variables with `Private`, especially with the contrasting `Public` keyword that would have to be used anyway to declare a public member. Alternatively use `Dim` *everywhere* - what matters is *consistency*:

"Private fields"

- **DO** use `Private` to declare a module-level variable.
- **DO** use `Dim` to declare a local variable.
- **DO NOT** use `Dim` to declare a module-level variable.

"Dim everywhere"

- **DO** use `Dim` to declare anything private/local.
- **DO NOT** use `Private` to declare a module-level variable.
- **AVOID** declaring `Public` fields.*

*In general, one should avoid declaring `Public` or `Global` fields anyway.

Fields

A variable declared at module level, in the *declarations section* at the top of the module body, is a *field*. A `Public` field declared in a *standard module* is a *global variable*:

```
Public PublicField As Long
```

A variable with a global scope can be accessed from anywhere, including other VBA projects that would reference the project it's declared in.

To make a variable global/public, but only visible from within the project, use the `Friend` modifier:

```
Friend FriendField As Long
```

This is especially useful in add-ins, where the intent is that other VBA projects reference the add-in project and can consume the public API.

```
Friend FriendField As Long 'public within the project, aka for "friend" code
Public PublicField As Long 'public within and beyond the project
```

Friend fields are not available in standard modules.

实例字段

在模块级别声明的变量，位于类模块主体顶部的声明部分（包括ThisWorkbook、ThisDocument、Worksheet、UserForm和类模块），是一个实例字段：它只在类的实例存在时存在。

```
'> Class1
Option Explicit
Public PublicField As Long
```

```
'> Module1
Option Explicit
Public Sub DoSomething()
    'Class1.PublicField在这里没有意义
    With New Class1
        .PublicField = 42
    结束于
    'Class1.PublicField 在这里没有任何意义
End Sub
```

封装字段

实例数据通常被Private（私有）保存，并称为封装。可以使用Property过程来公开私有字段。为了公开私有变量但不允许调用者写入，类模块（或标准模块）实现了一个Property Get成员：

```
Option Explicit
Private 封装变量 As Long

Public Property Get SomeValue() As Long
    SomeValue = 封装变量
End Property

Public Sub DoSomething()
    封装变量 = 42
End Sub
```

类本身可以修改封装的值，但调用代码只能访问Public成员（如果调用者在同一项目中，还可以访问Friend成员）。

为了允许调用者修改：

- 一个封装的值，模块公开一个Property Let成员。
- 封装的对象引用，模块公开一个属性集成员。

第5.3节：常量（Const）

如果您的应用程序中有一个永远不会改变的值，您可以定义一个命名常量，并用它代替字面值。

您只能在模块或过程级别使用Const。这意味着变量的声明上下文必须是类、结构、模块、过程或代码块，不能是源文件、命名空间或接口。

```
Public Const GLOBAL_CONSTANT As String = "Project Version #1.000.000.001"
Private Const MODULE_CONSTANT As String = "Something relevant to this Module"

Public Sub ExampleDeclaration()
```


Instance Fields

A variable declared at module level, in the *declarations section* at the top of the body of a class module (including `ThisWorkbook`, `ThisDocument`, `Worksheet`, `UserForm` and *class modules*), is an *instance field*: it only exists as long as there's an *instance* of the class around.

```
'> Class1
Option Explicit
Public PublicField As Long
```

```
'> Module1
Option Explicit
Public Sub DoSomething()
    'Class1.PublicField means nothing here
    With New Class1
        .PublicField = 42
    End With
    'Class1.PublicField means nothing here
End Sub
```

Encapsulating fields

Instance data is often kept **`Private`**, and dubbed *encapsulated*. A private field can be exposed using a **`Property`** procedure. To expose a private variable publicly without giving write access to the caller, a class module (or a standard module) implements a **`Property Get`** member:

```
Option Explicit
Private encapsulated As Long

Public Property Get SomeValue() As Long
    SomeValue = encapsulated
End Property

Public Sub DoSomething()
    encapsulated = 42
End Sub
```

The class itself can modify the encapsulated value, but the calling code can only access the **`Public`** members (and **`Friend`** members, if the caller is in the same project).

To allow the caller to modify:

- An encapsulated **value**, a module exposes a **`Property Let`** member.
- An encapsulated **object reference**, a module exposes a **`Property Set`** member.

Section 5.3: Constants (Const)

If you have a value that never changes in your application, you can define a named constant and use it in place of a literal value.

You can use Const only at module or procedure level. This means the declaration context for a variable must be a class, structure, module, procedure, or block, and cannot be a source file, namespace, or interface.

```
Public Const GLOBAL_CONSTANT As String = "Project Version #1.000.000.001"
Private Const MODULE_CONSTANT As String = "Something relevant to this Module"

Public Sub ExampleDeclaration()
```

```
    Const SOME_CONSTANT As String = "Hello World"

    Const PI As Double = 3.141592653

End Sub
```

虽然指定常量类型可以被视为良好实践，但并非严格要求。不指定类型
仍然会得到正确的类型：

```
Public Const GLOBAL_CONSTANT = "Project Version #1.000.000.001" '仍然是字符串
Public Sub ExampleDeclaration()

    Const SOME_CONSTANT = "Hello World"           '仍然是字符串
    Const DERIVED_CONSTANT = SOME_CONSTANT        'DERIVED_CONSTANT 也是字符串
    Const VAR_CONSTANT As Variant = SOME_CONSTANT 'VAR_CONSTANT 是 Variant/字符串

    Const PI = 3.141592653        '仍然是双精度数
    Const DERIVED_PI = PI         'DERIVED_PI 也是双精度数
    Const VAR_PI As Variant = PI  'VAR_PI 是 Variant/双精度数

End Sub
```

请注意，这仅适用于常量，与变量不同，变量如果不指定类型则默认为 Variant 类型。

虽然可以显式声明常量为字符串，但不能使用固定长度字符串语法声明常量为字符串

```
这是一个有效的5字符字符串常量
Const FOO As String = "ABCDE"

'这不是声明5字符字符串常量的有效语法
Const FOO As String * 5 = "ABCDE"
```

第5.4节：声明固定长度字符串

在VBA中，字符串可以声明为特定长度；它们会自动填充或截断以保持声明的长度。

```
Public Sub TwoTypesOfStrings()

    Dim FixedLengthString As String * 5 ' 声明一个5字符的字符串
    Dim NormalString As String

Debug.Print FixedLengthString      ' 输出 "     "
    Debug.Print NormalString             ' 输出 ""

    FixedLengthString = "123"            ' FixedLengthString 现在等于 "123  "
    NormalString = "456"                 ' NormalString 现在等于 "456"

FixedLengthString = "123456"         ' FixedLengthString 现在等于 "12345"
    NormalString = "456789"              ' NormalString 现在等于 "456789"

End Sub
```



```
    Const SOME_CONSTANT As String = "Hello World"

    Const PI As Double = 3.141592653

End Sub
```

Whilst it can be considered good practice to specify Constant types, it isn't strictly required. Not specifying the type will still result in the correct type:

```
Public Const GLOBAL_CONSTANT = "Project Version #1.000.000.001" 'Still a string
Public Sub ExampleDeclaration()

    Const SOME_CONSTANT = "Hello World"           'Still a string
    Const DERIVED_CONSTANT = SOME_CONSTANT        'DERIVED_CONSTANT is also a string
    Const VAR_CONSTANT As Variant = SOME_CONSTANT 'VAR_CONSTANT is Variant/String

    Const PI = 3.141592653        'Still a double
    Const DERIVED_PI = PI         'DERIVED_PI is also a double
    Const VAR_PI As Variant = PI  'VAR_PI is Variant/Double

End Sub
```

Note that this is specific to Constants and in contrast to variables where not specifying the type results in a Variant type.

While it is possible to explicitly declare a constant as a String, it is not possible to declare a constant as a string using fixed-width string syntax

```
'This is a valid 5 character string constant
Const FOO As String = "ABCDE"

'This is not valid syntax for a 5 character string constant
Const FOO As String * 5 = "ABCDE"
```

Section 5.4: Declaring Fixed-Length Strings

In VBA, Strings can be declared with a specific length; they are automatically padded or truncated to maintain that length as declared.

```
Public Sub TwoTypesOfStrings()

    Dim FixedLengthString As String * 5 ' declares a string of 5 characters
    Dim NormalString As String

    Debug.Print FixedLengthString       ' Prints "     "
    Debug.Print NormalString            ' Prints ""

    FixedLengthString = "123"           ' FixedLengthString now equals "123  "
    NormalString = "456"                ' NormalString now equals "456"

    FixedLengthString = "123456"        ' FixedLengthString now equals "12345"
    NormalString = "456789"             ' NormalString now equals "456789"

End Sub
```

第5.5节：何时使用静态变量

在子过程退出时，局部声明的静态变量不会被销毁，也不会丢失其值。
对该过程的后续调用不需要重新初始化或赋值，尽管你可能想要“清零”任何记忆的值。

当在反复调用的“辅助”子程序中进行晚绑定对象时，这些特别有用。

代码片段1： 在多个工作表之间重用 Scripting.Dictionary 对象

```
Option Explicit

Sub main()
    Dim w As Long

    For w = 1 To Worksheets.Count
       processDictionary ws:=Worksheets(w)
    Next w
End Sub

Sub processDictionary(ws As Worksheet)
    Dim i As Long, rng As Range
    Static dict As Object

    If dict Is Nothing Then
        初始化并设置字典对象
        设置 dict = CreateObject("Scripting.Dictionary")
        dict.CompareMode = vbTextCompare
    否则
        '移除所有预先存在的字典条目' 如果希望使用来自所有工
        作表的单一字典条目，可能需要或不需要这样做

        dict.RemoveAll
    End If

    With ws

        '为每个工作表使用一个新的字典对象
        ' 无需每次都构造/销毁新对象
        ' 或者在后续使用时不清空字典，
        ' 并构建包含所有工作表条目的字典

    End With
End Sub
```

代码片段 2： 创建一个工作表用户自定义函数（UDF），该函数延迟绑定 VBScript.RegExp 对象

```
Option Explicit

Function numbersOnly(str As String, _
                     Optional delim As String = ", ")
    Dim n As Long, nums() As Variant
    Static rgx As Object, cmat As Object

    ‘rgx 作为静态变量，只需创建一次
    ' 这在用此 UDF 填充长列时很有用
    If rgx Is Nothing Then
        Set rgx = CreateObject("VBScript.RegExp")
    Else
        Set cmat = Nothing
```


Section 5.5: When to use a Static variable

A Static variable declared locally is not destructed and does not lose its value when the Sub procedure is exited. Subsequent calls to the procedure do not require re-initialization or assignment although you may want to 'zero' any remembered value(s).

These are particularly useful when late binding an object in a 'helper' sub that is called repeatedly.

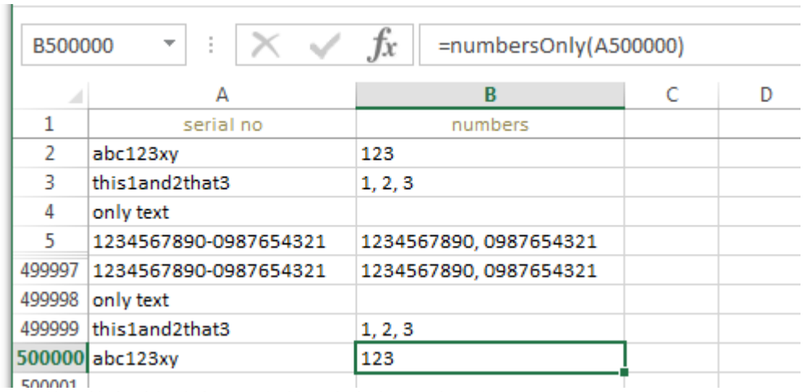
Snippet 1: Reuse a Scripting.Dictionary object across many worksheets

```
Option Explicit

Sub main()
    Dim w As Long

    For w = 1 To Worksheets.Count
        processDictionary ws:=Worksheets(w)
    Next w
End Sub

Sub processDictionary(ws As Worksheet)
    Dim i As Long, rng As Range
    Static dict As Object

    If dict Is Nothing Then
        'initialize and set the dictionary object
        Set dict = CreateObject("Scripting.Dictionary")
        dict.CompareMode = vbTextCompare
    Else
        'remove all pre-existing dictionary entries
        ' this may or may not be desired if a single dictionary of entries
        ' from all worksheets is preferred
        dict.RemoveAll
    End If

    With ws

        'work with a fresh dictionary object for each worksheet
        ' without constructing/destructing a new object each time
        ' or do not clear the dictionary upon subsequent uses and
        ' build a dictionary containing entries from all worksheets

    End With
End Sub
```

Snippet 2: Create a worksheet UDF that late binds the VBScript.RegExp object

```
Option Explicit

Function numbersOnly(str As String, _
                     Optional delim As String = ", ")
    Dim n As Long, nums() As Variant
    Static rgx As Object, cmat As Object

    'with rgx as static, it only has to be created once
    'this is beneficial when filling a long column with this UDF
    If rgx Is Nothing Then
        Set rgx = CreateObject("VBScript.RegExp")
    Else
        Set cmat = Nothing
```

```
    End If

    With rgx
        .Global = True
        .MultiLine = True
        .Pattern = "[0-9]{1,999}"
        If .Test(str) Then
            Set cmat = .Execute(str)
            '调整 nums 数组大小以接受匹配项
            ReDim nums(cmat.Count - 1)
            '用匹配项填充 nums 数组
            For n = LBound(nums) To UBound(nums)
              nums(n) = cmat.Item(n)
            Next n
            将 nums 数组转换为分隔字符串
          numbersOnly = Join(nums, delim)
        否则
            numbersOnly = vbNullString
        End If
    End With
End Function
```



	A	B	C	D
1	serial no	numbers		
2	abc123xy	123		
3	this1and2that3	1, 2, 3		
4	only text			
5	1234567890-0987654321	1234567890, 0987654321		
499997	1234567890-0987654321	1234567890, 0987654321		
499998	only text			
499999	this1and2that3	1, 2, 3		
500000	abc123xy	123		

使用静态对象填充50万行的用户自定义函数示例

> * 使用用户自定义函数填充50万行的耗时：
> - 使用Dim rgx As Object：148.74秒
> - 使用Static rgx As Object：26.07秒
> * 这些数据仅供相对比较参考。您的实际结果将根据
> 操作的复杂性和
> 范围而有所不同。

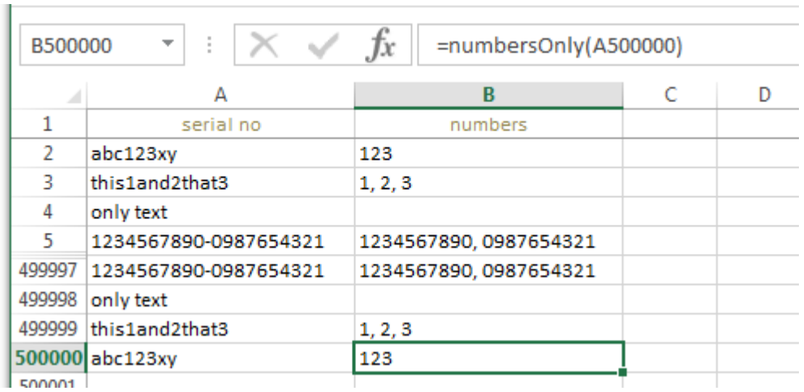
请记住，用户自定义函数（UDF）不会在工作簿的生命周期内只计算一次。即使是非易失性UDF，只要其引用的范围内的值发生变化，也会重新计算。
每次后续的重新计算事件
都会增加静态声明变量的优势。

- 静态变量在模块的生命周期内有效，而不是在声明和赋值的过程或函数的生命周期内有效。

- 静态变量只能在局部声明。
- 静态变量具有许多与私有模块级变量相同的属性，但作用域更受限制。

相关参考：Static（Visual Basic）


```
    End If

    With rgx
        .Global = True
        .MultiLine = True
        .Pattern = "[0-9]{1,999}"
        If .Test(str) Then
            Set cmat = .Execute(str)
            'resize the nums array to accept the matches
            ReDim nums(cmat.Count - 1)
            'populate the nums array with the matches
            For n = LBound(nums) To UBound(nums)
                nums(n) = cmat.Item(n)
            Next n
            'convert the nums array to a delimited string
            numbersOnly = Join(nums, delim)
        Else
            numbersOnly = vbNullString
        End If
    End With
End Function
```



	A	B	C	D
1	serial no	numbers		
2	abc123xy	123		
3	this1and2that3	1, 2, 3		
4	only text			
5	1234567890-0987654321	1234567890, 0987654321		
499997	1234567890-0987654321	1234567890, 0987654321		
499998	only text			
499999	this1and2that3	1, 2, 3		
500000	abc123xy	123		

Example of UDF with Static object filled through a half-million rows

> *Elapsed times to fill 500K rows with UDF:
> - with **Dim rgx As Object**: 148.74 seconds
> - with **Static rgx As Object**: 26.07 seconds
> * These should be considered for relative comparison only. Your own results will vary according to the complexity and
> scope of the operations performed.

Remember that a UDF is not calculated once in the lifetime of a workbook. Even a non-volatile UDF will recalculate whenever the values within the range(s) it references are subject to change. Each subsequent recalculation event only increases the benefits of a statically declared variable.

- A Static variable is available for the lifetime of the module, not the procedure or function in which it was declared and assigned.
- Static variables can only be declared locally.
- Static variable hold many of the same properties of a private module level variable but with a more restricted scope.

Related reference: Static (Visual Basic)

第5.6节：隐式和显式声明

如果代码模块顶部没有包含Option Explicit，则编译器会在你使用变量时自动（即"隐式"）为你创建变量。它们默认的变量类型为Variant。

```
Public Sub ExampleDeclaration()

    someVariable = 10                   '
    someOtherVariable = "Hello World"
    '这两个变量都是Variant类型。

End Sub
```

在上述代码中，如果指定了Option Explicit，代码将中断，因为缺少对someVariable和someOtherVariable的必要Dim语句。

```
Option Explicit

Public Sub ExampleDeclaration()

    Dim someVariable As Long
    someVariable = 10

    Dim someOtherVariable As String
    someOtherVariable = "Hello World"

End Sub
```

在代码模块中使用 Option Explicit 被认为是最佳实践，以确保声明所有变量。

请参阅 VBA 最佳实践，了解如何默认设置此选项。

第 5.7 节：访问修饰符

Dim 语句应保留用于局部变量。在模块级别，建议使用显式访问修饰符：

- **Private 用于私有字段，只能在声明它们的模块内访问。**
- **Public 用于公共字段和全局变量，任何调用代码都可以访问。**
- **Friend 用于项目内公共变量，但其他引用的 VBA 项目无法访问（适用于加载项）。**

- Global 也可用于标准模块中的 Public 字段，但在类模块中非法且已废弃——建议使用 Public 修饰符代替。此修饰符对过程也不合法。

访问修饰符同样适用于变量和过程。

```
Private ModuleVariable As String
Public GlobalVariable As String

Private Sub ModuleProcedure()

ModuleVariable = "这只能在同一模块内完成"

End Sub

Public Sub GlobalProcedure()

GlobalVariable = "这可以在本项目中的任何模块内完成"
```


Section 5.6: Implicit And Explicit Declaration

If a code module does not contain `Option Explicit` at the top of the module, then the compiler will automatically (that is, "implicitly") create variables for you when you use them. They will default to variable type `Variant`.

```
Public Sub ExampleDeclaration()

    someVariable = 10                   '
    someOtherVariable = "Hello World"
    'Both of these variables are of the Variant type.

End Sub
```

In the above code, if `Option Explicit` is specified, the code will interrupt because it is missing the required `Dim` statements for `someVariable` and `someOtherVariable`.

```
Option Explicit

Public Sub ExampleDeclaration()

    Dim someVariable As Long
    someVariable = 10

    Dim someOtherVariable As String
    someOtherVariable = "Hello World"

End Sub
```

It is considered best practice to use Option Explicit in code modules, to ensure that you declare all variables.

See VBA Best Practices how to set this option by default.

Section 5.7: Access Modifiers

The `Dim` statement should be reserved for local variables. At module-level, prefer explicit access modifiers:

- `Private` for private fields, which can only be accessed within the module they're declared in.
- `Public` for public fields and global variables, which can be accessed by any calling code.
- `Friend` for variables public within the project, but inaccessible to other referencing VBA projects (relevant for add-ins)
- `Global` can also be used for `Public` fields in standard modules, but is illegal in class modules and is obsolete anyway - prefer the `Public` modifier instead. This modifier isn't legal for procedures either.

Access modifiers are applicable to variables and procedures alike.

```
Private ModuleVariable As String
Public GlobalVariable As String

Private Sub ModuleProcedure()

    ModuleVariable = "This can only be done from within the same Module"

End Sub

Public Sub GlobalProcedure()

    GlobalVariable = "This can be done from any Module within this Project"
```

```
End Sub
```

Option Private Module

标准模块中的无参数公共`Sub`过程作为宏公开，可以附加到主文档中的控件和键盘快捷键。

相反，标准模块中的公共`Function`过程作为用户自定义函数（UDF）在主应用程序中公开。

在标准模块顶部指定`Option Private Module`可以防止其成员作为宏和UDF向主应用程序公开。


```
End Sub
```

Option Private Module

Public parameterless `Sub` procedures in standard modules are exposed as macros and can be attached to controls and keyboard shortcuts in the host document.

Conversely, public `Function` procedures in standard modules are exposed as user-defined functions (UDF's) in the host application.

Specifying `Option Private Module` at the top of a standard module prevents its members from being exposed as macros and UDF's to the host application.

第6章：声明和赋值字符串

第6.1节：字节数组的赋值与转换

字符串可以直接赋值给字节数组，反之亦然。请记住，字符串存储在多字节字符集（见下文备注）中，因此结果数组中只有每隔一个索引位置的部分字符属于ASCII范围内的字符。

```
Dim bytes() As Byte
Dim example As String

example = "Testing."
bytes = example                 '直接赋值。

'遍历字符。由于宽字符编码，步长为2。
声明 i 为 Long 类型
For i = LBound(bytes) To UBound(bytes) Step 2
    Debug.Print Chr$(bytes(i))  '打印 T, e, s, t, i, n, g, .
Next

Dim reverted As String
reverted = bytes                '直接赋值。
Debug.Print reverted            '打印 "Testing."
```

第6.2节：声明字符串常量

```
Const appName As String = "那个应用程序"
```

第6.3节：声明可变长度字符串变量

```
Dim surname As String '姓氏可以接受可变长度的字符串
surname = "Smith"
surname = "Johnson"
```

第6.4节：声明并赋值定长字符串

```
'声明并赋值一个1字符定长字符串
Dim middleInitial As String * 1 '中间名首字母必须是1个字符长度
middleInitial = "M"

声明并赋值一个2字符固定宽度的字符串 `stateCode`,
必须是2个字符长度
Dim stateCode As String * 2
stateCode = "TX"
```

第6.5节：声明并赋值字符串数组

```
'声明、定义并赋值一个包含3个元素的字符串数组
Dim departments(2) As String
departments(0) = "工程部"
departments(1) = "财务部"
departments(2) = "市场部"

'声明一个未定义维度的字符串数组，然后动态赋值为返回字符串数组的函数结果
```


Chapter 6: Declaring and assigning strings

Section 6.1: Assignment to and from a byte array

Strings can be assigned directly to byte arrays and visa-versa. Remember that Strings are stored in a Multi-Byte Character Set (see Remarks below) so only every other index of the resulting array will be the portion of the character that falls within the ASCII range.

```
Dim bytes() As Byte
Dim example As String

example = "Testing."
bytes = example             'Direct assignment.

'Loop through the characters. Step 2 is used due to wide encoding.
Dim i As Long
For i = LBound(bytes) To UBound(bytes) Step 2
    Debug.Print Chr$(bytes(i))  'Prints T, e, s, t, i, n, g, .
Next

Dim reverted As String
reverted = bytes            'Direct assignment.
Debug.Print reverted        'Prints "Testing."
```

Section 6.2: Declare a string constant

```
Const appName As String = "The App For That"
```

Section 6.3: Declare a variable-width string variable

```
Dim surname As String 'surname can accept strings of variable length
surname = "Smith"
surname = "Johnson"
```

Section 6.4: Declare and assign a fixed-width string

```
'Declare and assign a 1-character fixed-width string
Dim middleInitial As String * 1 'middleInitial must be 1 character in length
middleInitial = "M"

'Declare and assign a 2-character fixed-width string `stateCode`,
'must be 2 characters in length
Dim stateCode As String * 2
stateCode = "TX"
```

Section 6.5: Declare and assign a string array

```
'Declare, dimension and assign a string array with 3 elements
Dim departments(2) As String
departments(0) = "Engineering"
departments(1) = "Finance"
departments(2) = "Marketing"

'Declare an undimensioned string array and then dynamically assign with
'the results of a function that returns a string array
```

```
Dim stateNames() As String
stateNames = VBA.Strings.Split("Texas;California;New York", ";")

'声明、定义并赋值一个定长字符串数组
Dim stateCodes(2) As String * 2
stateCodes(0) = "TX"
stateCodes(1) = "CA"
stateCodes(2) = "NY"
```

第6.6节：使用Mid语句在字符串中分配特定字符

VBA提供了Mid函数用于返回字符串中的子字符串，但它也提供了Mid语句，可以用来分配字符串中的子字符串或单个字符。

Mid函数通常出现在赋值语句的右侧或条件中，但
Mid语句通常出现在赋值语句的左侧。

```
Dim surname As String
surname = "Smith"

'使用Mid语句更改字符串中的第3个字符
Mid(surname, 3, 1) = "y"
Debug.Print surname

'输出：'
'Smyth
```

注意：如果需要对字符串中的单个字节而非单个字符进行赋值（参见下文关于多字节字符集的说明），可以使用MidB语句。在这种情况下，MidB语句的第二个参数是替换开始的字节位置（基于1），因此上面示例的等效代码为MidB(surname, 5, 2) = "y"。


```
Dim stateNames() As String
stateNames = VBA.Strings.Split("Texas;California;New York", ";")

'Declare, dimension and assign a fixed-width string array
Dim stateCodes(2) As String * 2
stateCodes(0) = "TX"
stateCodes(1) = "CA"
stateCodes(2) = "NY"
```

Section 6.6: Assign specific characters within a string using Mid statement

VBA offers a Mid function for *returning* substrings within a string, but it also offers the Mid *Statement* which can be used to assign substrings or individual characters withing a string.

The `Mid` function will typically appear on the right-hand-side of an assignment statement or in a condition, but the `Mid` Statement typically appears on the left hand side of an assignment statement.

```
Dim surname As String
surname = "Smith"

'Use the Mid statement to change the 3rd character in a string
Mid(surname, 3, 1) = "y"
Debug.Print surname

'Output:
'Smyth
```

Note: If you need to assign to individual *bytes* in a string instead of individual *characters* within a string (see the Remarks below regarding the Multi-Byte Character Set), the `MidB` statement can be used. In this instance, the second argument for the `MidB` statement is the 1-based position of the byte where the replacement will start so the equivalent line to the example above would be `MidB(surname, 5, 2) = "y"`.

第7章：字符串连接

第7.1节：使用Join函数连接字符串数组

```
声明并赋值字符串数组
Dim widgetNames(2) As String
widgetNames(0) = "foo"
widgetNames(1) = "bar"
widgetNames(2) = "fizz"

使用Join连接，并用3个字符的字符串分隔每个元素
concatenatedString = VBA.Strings.Join(widgetNames, " > ")
'concatenatedString = "foo > bar > fizz"

'使用Join连接，并用零宽字符串分隔每个元素
concatenatedString = VBA.Strings.Join(widgetNames, vbNullString)
'concatenatedString = "foobarfizz"
```

第7.2节：使用&运算符连接字符串

```
Const string1 As String = "foo"
Const string2 As String = "bar"
Const string3 As String = "fizz"
Dim concatenatedString As String

'连接两个字符串
concatenatedString = string1 & string2
'concatenatedString = "foobar"

连接三个字符串
concatenatedString = string1 & string2 & string3
'concatenatedString = "foobarfizz"
```


Chapter 7: Concatenating strings

Section 7.1: Concatenate an array of strings using the Join function

```
'Declare and assign a string array
Dim widgetNames(2) As String
widgetNames(0) = "foo"
widgetNames(1) = "bar"
widgetNames(2) = "fizz"

'Concatenate with Join and separate each element with a 3-character string
concatenatedString = VBA.Strings.Join(widgetNames, " > ")
'concatenatedString = "foo > bar > fizz"

'Concatenate with Join and separate each element with a zero-width string
concatenatedString = VBA.Strings.Join(widgetNames, vbNullString)
'concatenatedString = "foobarfizz"
```

Section 7.2: Concatenate strings using the & operator

```
Const string1 As String = "foo"
Const string2 As String = "bar"
Const string3 As String = "fizz"
Dim concatenatedString As String

'Concatenate two strings
concatenatedString = string1 & string2
'concatenatedString = "foobar"

'Concatenate three strings
concatenatedString = string1 & string2 & string3
'concatenatedString = "foobarfizz"
```

第8章：常用字符串操作

使用INSTR、FIND和LEN的MID、LEFT和RIGHT字符串函数快速示例。

如何找到两个搜索词之间的文本（例如：冒号后和逗号前）？如何获取单词的剩余部分（使用MID或RIGHT）？这些函数中哪些使用零基参数并返回代码，哪些使用一基？出错时会发生什么？它们如何处理空字符串、未找到结果和负数？

第8.1节：字符串操作常用示例

更好的MID()及其他字符串提取示例，目前网络上缺乏。请帮我做一个好的示例，或者完善这里的这个。类似这样的：

```
DIM strEmpty 作为String, strNull 作为String, theText 作为String
DIM idx 作为Integer
DIM letterCount 作为Integer
DIM result 作为String

strNull = NOTHING
strEmpty = ""
theText = "1234, 78910"

' -----------------
' 提取逗号 ", " 后和 "910" 前的单词 结果："78" ***
' -----------------

使用INSTR获取逗号的位置（索引）
idx = ...   ' 这里是一些解释
if idx < ... ' 检查文本中是否未找到逗号

' 或使用 FIND 获取逗号的索引
idx = ...   ' 这里是一些解释... 注意：区别在于...
if idx < ...  ' 检查文本中是否未找到逗号

result = MID(theText, ..., LEN(...

' 获取逗号后的剩余单词
result = MID(theText, idx+1, LEN(theText) - idx+1)

' 使用 LEFT 获取逗号前的单词
result = LEFT(theText, idx - 1)

' 使用 RIGHT 获取逗号和空格后的剩余文本
result = ...

' 出错时会发生什么
result = MID(strNothing, 1, 2)     ' 这会导致 ...
result = MID(strEmpty, 1, 2) ' 这会导致...
result = MID(theText, 30, 2) ' 现在...
result = MID(theText, 2, 999) ' 不用担心...
result = MID(theText, 0, 2)
result = MID(theText, 2, 0)
result = MID(theText -1, 2)
result = MID(theText 2, -1)
idx = INSTR(strNothing, "123")
idx = INSTR(theText, strNothing)
```


Chapter 8: Frequently used string manipulation

Quick examples for MID LEFT and RIGHT string functions using INSTR FIND and LEN.

How do you find the text between two search terms (Say: after a colon and before a comma)? How do you get the remainder of a word (using MID or using RIGHT)? Which of these functions use Zero-based params and return codes vs One-based? What happens when things go wrong? How do they handle empty strings, unfound results and negative numbers?

Section 8.1: String manipulation frequently used examples

Better MID() and other string extraction examples, currently lacking from the web. Please help me make a good example, or complete this one here. Something like this:

```
DIM strEmpty as String, strNull as String, theText as String
DIM idx as Integer
DIM letterCount as Integer
DIM result as String

strNull = NOTHING
strEmpty = ""
theText = "1234, 78910"

' -----------------
' Extract the word after the comma ", "  and before "910"  result: "78" ***
' -----------------

' Get index (place) of comma using INSTR
idx = ...   ' some explanation here
if idx < ... ' check if no comma found in text

' or get index of comma using FIND
idx = ...   ' some explanation here... Note: The difference is...
if idx < ...  ' check if no comma found in text

result = MID(theText, ..., LEN(...

' Retrieve remaining word after the comma
result = MID(theText, idx+1, LEN(theText) - idx+1)

' Get word until the comma using LEFT
result = LEFT(theText, idx - 1)

' Get remaining text after the comma-and-space using RIGHT
result = ...

' What happens when things go wrong
result = MID(strNothing, 1, 2)     ' this causes ...
result = MID(strEmpty, 1, 2) ' which causes...
result = MID(theText, 30, 2) ' and now...
result = MID(theText, 2, 999) ' no worries...
result = MID(theText, 0, 2)
result = MID(theText, 2, 0)
result = MID(theText -1, 2)
result = MID(theText 2, -1)
idx = INSTR(strNothing, "123")
idx = INSTR(theText, strNothing)
```

```
idx = INSTR(theText, strEmpty)
i = LEN(strEmpty)
i = LEN(strNothing) '...
```

请随意编辑此示例并使其更好。只要保持清晰，并包含常用的
实践。


```
idx = INSTR(theText, strEmpty)
i = LEN(strEmpty)
i = LEN(strNothing) '...
```

Please feel free to edit this example and make it better. As long as it remains clear, and has in it common usage practices.

第9章：子字符串

第9.1节：使用 Left 或 Left$ 获取字符串中最左边的3个字符

```
Const baseString As String = "Foo Bar"

Dim leftText As String
leftText = Left$(baseString, 3)
'leftText = "Foo"
```

第9.2节：使用 Right 或 Right$ 获取字符串中最右边的3个字符

```
Const baseString As String = "Foo Bar"
Dim rightText As String
rightText = Right$(baseString, 3)
'rightText = "Bar"
```

第9.3节：使用 Mid 或 Mid$ 从字符串中获取特定字符

```
Const baseString As String = "Foo Bar"

'获取从第2个字符开始到第6个字符的字符串
Dim midText As String
midText = Mid$(baseString, 2, 5)
'midText = "oo Ba"
```

第9.4节：使用Trim获取去除任何前导或尾随空格的字符串副本

```
'去除字符串中的前导和尾随空格
Const paddedText As String = "    Foo Bar    "
Dim trimmedText As String
trimmedText = Trim$(paddedText)
'trimmedText = "Foo Bar"
```


Chapter 9: Substrings

Section 9.1: Use Left or Left$ to get the 3 left-most characters in a string

```
Const baseString As String = "Foo Bar"

Dim leftText As String
leftText = Left$(baseString, 3)
'leftText = "Foo"
```

Section 9.2: Use Right or Right$ to get the 3 right-most characters in a string

```
Const baseString As String = "Foo Bar"
Dim rightText As String
rightText = Right$(baseString, 3)
'rightText = "Bar"
```

Section 9.3: Use Mid or Mid$ to get specific characters from within a string

```
Const baseString As String = "Foo Bar"

'Get the string starting at character 2 and ending at character 6
Dim midText As String
midText = Mid$(baseString, 2, 5)
'midText = "oo Ba"
```

Section 9.4: Use Trim to get a copy of the string without any leading or trailing spaces

```
'Trim the leading and trailing spaces in a string
Const paddedText As String = "    Foo Bar    "
Dim trimmedText As String
trimmedText = Trim$(paddedText)
'trimmedText = "Foo Bar"
```

第10章：在字符串中搜索子字符串的存在

第10.1节：使用InStr判断字符串是否包含子字符串

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

'检查baseString是否包含"bar"（不区分大小写）
containsBar = InStr(1, baseString, "bar", vbTextCompare) > 0
containsBar = True

检查 baseString 是否包含 bar（不区分大小写）
containsBar = InStr(1, baseString, "bar", vbBinaryCompare) > 0
containsBar = False
```

第10.2节：使用 InStrRev 查找子字符串最后一次出现的位置

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

查找最后一个 "B" 的位置
Dim posX As Long
注意 InStrRev 参数的数量和顺序不同
posX = InStrRev(baseString, "X", -1, vbBinaryCompare)
posX = 0
```

第10.3节：使用 InStr 查找子字符串第一次出现的位置

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

Dim posB As Long
posB = InStr(1, baseString, "B", vbBinaryCompare)
'posB = 5
```


Chapter 10: Searching within strings for the presence of substrings

Section 10.1: Use InStr to determine if a string contains a substring

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

'Check if baseString contains "bar" (case insensitive)
containsBar = InStr(1, baseString, "bar", vbTextCompare) > 0
'containsBar = True

'Check if baseString contains bar (case insensitive)
containsBar = InStr(1, baseString, "bar", vbBinaryCompare) > 0
'containsBar = False
```

Section 10.2: Use InStrRev to find the position of the last instance of a substring

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

'Find the position of the last "B"
Dim posX As Long
'Note the different number and order of the parameters for InStrRev
posX = InStrRev(baseString, "X", -1, vbBinaryCompare)
'posX = 0
```

Section 10.3: Use InStr to find the position of the first instance of a substring

```
Const baseString As String = "Foo Bar"
Dim containsBar As Boolean

Dim posB As Long
posB = InStr(1, baseString, "B", vbBinaryCompare)
'posB = 5
```

第11章：赋值包含重复字符的字符串

第11.1节：使用String函数赋值包含n个重复字符的字符串

```
Dim lineOfHyphens As String
'赋值一个包含80个重复连字符的字符串
lineOfHyphens = String$(80, "-")
```

第11.2节：使用String和Space函数赋值一个n字符的字符串

```
Dim stringOfSpaces As String

'使用Space$赋值一个包含255个重复空格的字符串
stringOfSpaces = Space$(255)

'使用String$赋值一个包含255个重复空格的字符串
stringOfSpaces = String$(255, " ")
```


Chapter 11: Assigning strings with repeated characters

Section 11.1: Use the String function to assign a string with n repeated characters

```
Dim lineOfHyphens As String
'Assign a string with 80 repeated hyphens
lineOfHyphens = String$(80, "-")
```

Section 11.2: Use the String and Space functions to assign an n-character string

```
Dim stringOfSpaces As String

'Assign a string with 255 repeated spaces using Space$
stringOfSpaces = Space$(255)

'Assign a string with 255 repeated spaces using String$
stringOfSpaces = String$(255, " ")
```

第12章：测量字符串的长度

第12.1节：使用Len函数确定字符串中的字符数

```
Const baseString As String = "Hello World"

Dim charLength As Long

charLength = Len(baseString)
'charlength = 11
```

第12.2节：使用LenB函数确定字符串中的字节数

```
Const baseString As String = "Hello World"

Dim byteLength As Long

byteLength = LenB(baseString)
'byteLength = 22
```

第12.3节：优先使用`If Len(myString) = 0 Then`而非`If myString = "" Then`

在检查字符串是否为空时，更好的做法且更高效的是检查字符串的长度，而不是将字符串与空字符串进行比较。

```
Const myString As String = vbNullString

'在检查 myString 是否为空字符串时，优先使用此方法
If Len(myString) = 0 Then
Debug.Print "myString 是空字符串"
End If

'避免在检查 myString 是否为空字符串时使用此方法
If myString = vbNullString Then
    Debug.Print "myString 是空字符串"
End If
```


Chapter 12: Measuring the length of strings

Section 12.1: Use the Len function to determine the number of characters in a string

```
Const baseString As String = "Hello World"

Dim charLength As Long

charLength = Len(baseString)
'charlength = 11
```

Section 12.2: Use the LenB function to determine the number of bytes in a string

```
Const baseString As String = "Hello World"

Dim byteLength As Long

byteLength = LenB(baseString)
'byteLength = 22
```

Section 12.3: Prefer `If Len(myString) = 0 Then` over `If myString = "" Then`

When checking if a string is zero-length, it is better practice, and more efficient, to inspect the length of the string rather than comparing the string to an empty string.

```
Const myString As String = vbNullString

'Prefer this method when checking if myString is a zero-length string
If Len(myString) = 0 Then
    Debug.Print "myString is zero-length"
End If

'Avoid using this method when checking if myString is a zero-length string
If myString = vbNullString Then
    Debug.Print "myString is zero-length"
End If
```

第13章：将其他类型转换为字符串

第13.1节：使用 CStr 将数值类型转换为字符串

```
Const zipCode As Long = 10012
Dim zipCodeText As String
'将 zipCode 数字转换为数字字符的字符串
zipCodeText = CStr(zipCode)
'zipCodeText = "10012"
```

第13.2节：使用 Format 将数值类型转换并格式化为字符串

```
Const zipCode As long = 10012
Dim zeroPaddedNumber As String
zeroPaddedZipCode = Format(zipCode, "00000000")
'zeroPaddedNumber = "00010012"
```

第13.3节：使用StrConv将单字节字符的字节数组转换为字符串

```
'声明一个字节数组，赋值单字节字符代码，并转换为字符串
Dim singleByteChars(4) As Byte
singleByteChars(0) = 72
singleByteChars(1) = 101
singleByteChars(2) = 108
singleByteChars(3) = 108
singleByteChars(4) = 111
Dim stringFromSingleByteChars As String
stringFromSingleByteChars = StrConv(singleByteChars, vbUnicode)
'stringFromSingleByteChars = "Hello"
```

第13.4节：隐式将多字节字符的字节数组转换为字符串

```
'声明一个字节数组，赋值多字节字符代码，并转换为字符串
Dim multiByteChars(9) As Byte
multiByteChars(0) = 87
multiByteChars(1) = 0
multiByteChars(2) = 111
multiByteChars(3) = 0
multiByteChars(4) = 114
multiByteChars(5) = 0
multiByteChars(6) = 108
multiByteChars(7) = 0
multiByteChars(8) = 100
multiByteChars(9) = 0

Dim stringFromMultiByteChars As String
stringFromMultiByteChars = multiByteChars
'stringFromMultiByteChars = "World"
```


Chapter 13: Converting other types to strings

Section 13.1: Use CStr to convert a numeric type to a string

```
Const zipCode As Long = 10012
Dim zipCodeText As String
'Convert the zipCode number to a string of digit characters
zipCodeText = CStr(zipCode)
'zipCodeText = "10012"
```

Section 13.2: Use Format to convert and format a numeric type as a string

```
Const zipCode As long = 10012
Dim zeroPaddedNumber As String
zeroPaddedZipCode = Format(zipCode, "00000000")
'zeroPaddedNumber = "00010012"
```

Section 13.3: Use StrConv to convert a byte-array of single-byte characters to a string

```
'Declare an array of bytes, assign single-byte character codes, and convert to a string
Dim singleByteChars(4) As Byte
singleByteChars(0) = 72
singleByteChars(1) = 101
singleByteChars(2) = 108
singleByteChars(3) = 108
singleByteChars(4) = 111
Dim stringFromSingleByteChars As String
stringFromSingleByteChars = StrConv(singleByteChars, vbUnicode)
'stringFromSingleByteChars = "Hello"
```

Section 13.4: Implicitly convert a byte array of multi-byte-characters to a string

```
'Declare an array of bytes, assign multi-byte character codes, and convert to a string
Dim multiByteChars(9) As Byte
multiByteChars(0) = 87
multiByteChars(1) = 0
multiByteChars(2) = 111
multiByteChars(3) = 0
multiByteChars(4) = 114
multiByteChars(5) = 0
multiByteChars(6) = 108
multiByteChars(7) = 0
multiByteChars(8) = 100
multiByteChars(9) = 0

Dim stringFromMultiByteChars As String
stringFromMultiByteChars = multiByteChars
'stringFromMultiByteChars = "World"
```

第14章：日期时间操作

第14.1节：日历

VBA支持2种日历：公历和伊斯兰历

使用`Calendar`属性来修改或显示当前日历。

Calendar的2个取值为：

值	常量	说明
0	vbCalGreg	公历（默认）
1	vbCalHijri	伊斯兰历

示例

```
Sub CalendarExample()
    '缓存当前设置。
    Dim Cached As Integer
    Cached = Calendar

    ' 公历日期
    Calendar = vbCalGreg
    Dim Sample As Date
    '创建2016-07-28的示例日期
    Sample = DateSerial(2016, 7, 28)

    Debug.Print "当前日历：" & Calendar
    Debug.Print "示例日期 = " & Format$(Sample, "yyyy-mm-dd")

    ' 伊斯兰历日期
    Calendar = vbCalHijri
Debug.Print "当前日历：" & Calendar
    Debug.Print "示例日期 = " & Format$(Sample, "yyyy-mm-dd")

    '重置VBA为缓存值。
    Calendar = Cached
End Sub
```

此子程序打印以下内容；

```
当前日历：0
示例日期 = 2016-07-28
当前日历：1
示例日期 = 1437-10-23
```

第14.2节：基本函数

获取系统日期时间

VBA支持3个内置函数来从系统时钟获取日期和/或时间。

函数	返回类型	返回值
Now	Date	返回当前日期和时间
日期	Date	返回当前日期和时间的日期部分
时间	Date	返回当前日期和时间的时间部分

Chapter 14: Date Time Manipulation

Section 14.1: Calendar

VBA supports 2 calendars : Gregorian and Hijri

The `Calendar` property is used to modify or display the current calendar.

The 2 values for the Calendar are:

Value	Constant	Description
0	vbCalGreg	Gregorian calendar (default)
1	vbCalHijri	Hijri calendar

Example

```
Sub CalendarExample()
    'Cache the current setting.
    Dim Cached As Integer
    Cached = Calendar

    ' Dates in Gregorian Calendar
    Calendar = vbCalGreg
    Dim Sample As Date
    'Create sample date of 2016-07-28
    Sample = DateSerial(2016, 7, 28)

    Debug.Print "Current Calendar : " & Calendar
    Debug.Print "SampleDate = " & Format$(Sample, "yyyy-mm-dd")

    ' Date in Hijri Calendar
    Calendar = vbCalHijri
    Debug.Print "Current Calendar : " & Calendar
    Debug.Print "SampleDate = " & Format$(Sample, "yyyy-mm-dd")

    'Reset VBA to cached value.
    Calendar = Cached
End Sub
```

This Sub prints the following ;

```
Current Calendar : 0
SampleDate = 2016-07-28
Current Calendar : 1
SampleDate = 1437-10-23
```

Section 14.2: Base functions

Retrieve System DateTime

VBA supports 3 built-in functions to retrieve the date and/or time from the system's clock.

Function	Return Type	Return Value
Now	Date	Returns the current date and time
Date	Date	Returns the date portion of the current date and time
Time	Date	Returns the time portion of the current date and time

```
Sub DateTimeExample()

    ' 注：欧盟系统默认日期格式为日/月/年

Debug.Print Now    ' 输出 28/07/2016 10:16:01（以下输出假设此日期和时间）
    Debug.Print Date   ' 输出 28/07/2016
Debug.Print Time   ' 输出 10:16:01

    ' 对当前日期或时间应用自定义格式
Debug.Print Format$(Now, "dd mmmm yyyy hh:nn")   ' 输出 28 July 2016 10:16
    Debug.Print Format$(Date, "yyyy-mm-dd")          ' 输出 2016-07-28
Debug.Print Format$(Time, "hh") & " 小时 " & _
                Format$(Time, "nn") & " 分钟 " & _
                Format$(Time, "ss") & " 秒 "       ' 输出 10 小时 16 分钟 01 秒

End Sub
```

计时器函数

`Timer` 函数返回一个 Single 类型的数值，表示自午夜以来经过的秒数。精度为百分之一秒。

```
子程序 TimerExample()

Debug.Print Time     ' 打印 10:36:31   （执行时的时间）
    Debug.Print Timer    ' 打印 38191,13   （自午夜以来的秒数）

End Sub
```

由于 Now 和 Time 函数的精度仅到秒，Timer 提供了一种方便的方法来提高时间测量的准确性：

```
子程序 GetBenchmark()

    Dim StartTime  As Single
StartTime = Timer        '存储当前时间

    声明 i 为 Long 类型
    Dim temp As String
    For i = 1 To 0000000     '查看 Left$ 执行 1,000,000 次所需时间
       temp = Left$("Text", 2)
    Next i

    维度 已用时间 作为 单一
    Elapsed = Timer - StartTime
Debug.Print "代码完成于 " & CInt(Elapsed * 1000) & " 毫秒"

End Sub
```

IsDate()

IsDate() 用于测试表达式是否为有效日期。返回一个 Boolean 值。

```
Sub IsDate示例()

    Dim anything As Variant
```



```
Sub DateTimeExample()

    ' Note : EU system with default date format DD/MM/YYYY

    Debug.Print Now    ' prints 28/07/2016 10:16:01 (output below assumes this date and time)
    Debug.Print Date   ' prints 28/07/2016
    Debug.Print Time   ' prints 10:16:01

    ' Apply a custom format to the current date or time
    Debug.Print Format$(Now, "dd mmmm yyyy hh:nn")   ' prints 28 July 2016 10:16
    Debug.Print Format$(Date, "yyyy-mm-dd")          ' prints 2016-07-28
    Debug.Print Format$(Time, "hh") & " hour " & _
                Format$(Time, "nn") & " min " & _
                Format$(Time, "ss") & " sec "        ' prints 10 hour 16 min 01 sec

End Sub
```

Timer Function

The `Timer` function returns a Single representing the number of seconds elapsed since midnight. The precision is one hundredth of a second.

```
Sub TimerExample()

    Debug.Print Time     ' prints 10:36:31  (time at execution)
    Debug.Print Timer    ' prints 38191,13  (seconds since midnight)

End Sub
```

Because `Now` and `Time` functions are only precise to seconds, `Timer` offers a convenient way to increase accuracy of time measurement:

```
Sub GetBenchmark()

    Dim StartTime  As Single
    StartTime = Timer        'Store the current Time

    Dim i As Long
    Dim temp As String
    For i = 1 To 1000000     'See how long it takes Left$ to execute 1,000,000 times
        temp = Left$("Text", 2)
    Next i

    Dim Elapsed As Single
    Elapsed = Timer - StartTime
    Debug.Print "Code completed in " & CInt(Elapsed * 1000) & " ms"

End Sub
```

IsDate()

IsDate() tests whether an expression is a valid date or not. Returns a `Boolean`.

```
Sub IsDateExamples()

    Dim anything As Variant
```

```
anything = "2001年9月11日"
    Debug.Print IsDate(anything)     '打印 True
   anything = #9/11/2001#
Debug.Print IsDate(anything)     '打印 True
    anything = "只是一个字符串"
Debug.Print IsDate(anything)     '打印 False
    anything = vbNull
Debug.Print IsDate(anything)     '打印 False
End Sub
```

第14.3节：提取函数

这些函数以一个可以转换为Date的Variant作为参数，并返回一个表示日期或时间部分的Integer。如果参数不能转换为Date，将导致运行时错误13：类型不匹配。

函数	说明	返回值
Year()	返回日期参数的年份部分。	整数（100到9999）
Month()	返回日期参数的月份部分。	整数（1到12）
Day()	返回日期参数的日期部分。	整数（1 到 31）
WeekDay()	返回日期参数对应的星期几。接受一个可选的第二个参数定义一周第一天的参数	整数（1 到 7）
Hour()	返回日期参数中的小时部分。	整数（0 到 23）
Minute()	返回日期参数中的分钟部分。	整数（0 到 59）
Second()	返回日期参数中的秒数部分。	整数（0 到 59）

示例：

```
Sub ExtractionExamples()
    Dim MyDate As Date
MyDate = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)
    Debug.Print Format$(MyDate, "yyyy-mm-dd hh:nn:ss") '输出 2016-07-28 12:34:56
    Debug.Print Year(MyDate)                          '输出 2016
Debug.Print Month(MyDate)                         '输出 7
    Debug.Print Day(MyDate)                           '输出 28
    Debug.Print Hour(MyDate)                          '输出 12
    Debug.Print Minute(MyDate)                        '输出 34
    Debug.Print Second(MyDate)                        '输出 56
    Debug.Print Weekday(MyDate)                       '输出 5
    '根据地区不同 - 例如在欧盟输出 4，在美国输出 5
   Debug.Print Weekday(MyDate, vbUseSystemDayOfWeek)
    Debug.Print Weekday(MyDate, vbMonday)             '输出 4
    Debug.Print Weekday(MyDate, vbSunday)             '输出 5
```



```
    anything = "September 11, 2001"
    Debug.Print IsDate(anything)    'Prints True
    anything = #9/11/2001#
    Debug.Print IsDate(anything)    'Prints True
    anything = "just a string"
    Debug.Print IsDate(anything)    'Prints False
    anything = vbNull
    Debug.Print IsDate(anything)    'Prints False
End Sub
```

Section 14.3: Extraction functions

These functions take a `Variant` that can be cast to a `Date` as a parameter and return an `Integer` representing a portion of a date or time. If the parameter can not be cast to a `Date`, it will result in a run-time error 13: Type mismatch.

Function	Description	Returned value
Year()	Returns the year portion of the date argument.	Integer (100 to 9999)
Month()	Returns the month portion of the date argument.	Integer (1 to 12)
Day()	Returns the day portion of the date argument.	Integer (1 to 31)
WeekDay()	Returns the day of the week of the date argument. Accepts an optional second argument defininig the first day of the week	Integer (1 to 7)
Hour()	Returns the hour portion of the date argument.	Integer (0 to 23)
Minute()	Returns the minute portion of the date argument.	Integer (0 to 59)
Second()	Returns the second portion of the date argument.	Integer (0 to 59)

Examples:

```
Sub ExtractionExamples()
    Dim MyDate As Date
    MyDate = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)
    Debug.Print Format$(MyDate, "yyyy-mm-dd hh:nn:ss") ' prints 2016-07-28 12:34:56
    Debug.Print Year(MyDate)                          ' prints 2016
    Debug.Print Month(MyDate)                         ' prints 7
    Debug.Print Day(MyDate)                           ' prints 28
    Debug.Print Hour(MyDate)                          ' prints 12
    Debug.Print Minute(MyDate)                        ' prints 34
    Debug.Print Second(MyDate)                        ' prints 56
    Debug.Print Weekday(MyDate)                       ' prints 5
    'Varies by locale - i.e. will print 4 in the EU and 5 in the US
    Debug.Print Weekday(MyDate, vbUseSystemDayOfWeek)
    Debug.Print Weekday(MyDate, vbMonday)             ' prints 4
    Debug.Print Weekday(MyDate, vbSunday)             ' prints 5
```

```
End Sub
```

DatePart() 函数

DatePart() 也是一个返回日期部分的函数，但工作方式不同，且比上述函数提供更多可能性。例如，它可以返回年份的季度或年份的周数。

语法：

```
DatePart ( interval, date  [, firstdayofweek] [, firstweekofyear] )
```

interval 参数可以是：

Interval	说明
"yyyy"	年份 (100 到 9999)
y	一年中的第几天（1 到 366）
"m"	月份（1 到 12）
"q"	季度（1 到 4）
"ww"	周数（1 到 53）
"w"	星期几（1 到 7）
"d"	月份中的某一天（1 到 31）
“h”	小时（0 到 23）
“n”	分钟（0 到 59）
“s”	秒（0 到 59）

*firstdayofweek*是可选的。它是一个常量，用于指定一周的第一天。如果未指定，vbSunday将被假定。

*firstweekofyear*是可选的。它是一个常量，用于指定一年的第一周。如果未指定，第一周将被假定为包含1月1日的那一周。

示例：

```
Sub DatePartExample()

    Dim MyDate As Date

MyDate = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)

    Debug.Print Format$(MyDate, "yyyy-mm-dd hh:nn:ss") ' 输出 2016-07-28 12:34:56

    Debug.Print DatePart("yyyy", MyDate)              ' 输出 2016
Debug.Print DatePart("y", MyDate)                 ' 输出 210
    Debug.Print DatePart("h", MyDate)                 ' 输出 12
    Debug.Print DatePart("Q", MyDate)                 ' 输出 3
    Debug.Print DatePart("w", MyDate)                 ' 输出 5
    Debug.Print DatePart("ww", MyDate)                ' 输出 31

End Sub
```

第14.4节：计算函数

DateDiff()


```
End Sub
```

DatePart() Function

`DatePart()` is also a function returning a portion of a date, but works differently and allow more possibilities than the functions above. It can for instance return the Quarter of the year or the Week of the year.

Syntax:

```
DatePart ( interval, date  [, firstdayofweek] [, firstweekofyear] )
```

interval argument can be :

Interval	Description
"yyyy"	Year (100 to 9999)
"y"	Day of the year (1 to 366)
"m"	Month (1 to 12)
"q"	Quarter (1 to 4)
"ww"	Week (1 to 53)
"w"	Day of the week (1 to 7)
"d"	Day of the month (1 to 31)
"h"	Hour (0 to 23)
"n"	Minute (0 to 59)
"s"	Second (0 to 59)

firstdayofweek is optional. it is a constant that specifies the first day of the week. If not specified, vbSunday is assumed.

firstweekofyear is optional. it is a constant that specifies the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

Examples:

```
Sub DatePartExample()

    Dim MyDate As Date

    MyDate = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)

    Debug.Print Format$(MyDate, "yyyy-mm-dd hh:nn:ss") ' prints 2016-07-28 12:34:56

    Debug.Print DatePart("yyyy", MyDate)              ' prints 2016
    Debug.Print DatePart("y", MyDate)                 ' prints 210
    Debug.Print DatePart("h", MyDate)                 ' prints 12
    Debug.Print DatePart("Q", MyDate)                 ' prints 3
    Debug.Print DatePart("w", MyDate)                 ' prints 5
    Debug.Print DatePart("ww", MyDate)                ' prints 31

End Sub
```

Section 14.4: Calculation functions

DateDiff()

DateDiff() 返回一个 Long 类型，表示两个指定日期之间的时间间隔数。

语法

```
DateDiff ( interval, date1, date2  [, firstdayofweek] [, firstweekofyear] )
```

- *interval* 可以是 `DatePart()` 函数中定义的任何间隔类型
- *date1* 和 *date2* 是你想用于计算的两个日期
- *firstdayofweek* 和 *firstweekofyear* 是可选参数。具体说明请参见 `DatePart()` 函数

示例

```
Sub DateDiffExamples()

    ' 检查2016年是否为闰年。
    Dim NumberOfDays As Long
NumberOfDays = DateDiff("d", #1/1/2016#, #1/1/2017#)

    If NumberOfDays = 366 Then
Debug.Print "2016年是闰年。"                '这将输出。
    End If

    ' 一天中的秒数
    Dim StartTime As Date
    Dim EndTime As Date
StartTime = TimeSerial(0, 0, 0)
    EndTime = TimeSerial(24, 0, 0)
Debug.Print DateDiff("s", StartTime, EndTime)        '打印86400

End Sub
```

DateAdd()

DateAdd() 返回一个日期，该日期是在指定日期或时间间隔上加上指定值后的结果。

语法

```
DateAdd ( 间隔, 数值, 日期  )
```

- 间隔 可以是 DatePart() 函数中定义的任何间隔数值 是你想要添加的间
- 隔数量的数值表达式。它可以是正数（获取未来的日期）或负数（获取过去的日期）。
- 日期 是一个 Date 或表示要添加间隔的日期的字面值

示例：

```
Sub DateAdd示例()

    Dim Sample As Date
    '创建2016-07-28 12:34:56的示例日期和时间
    Sample = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)

    ' 5个月前的日期（打印2016-02-28）：
Debug.Print Format$(DateAdd("m", -5, Sample), "yyyy-mm-dd")

    ' 10个月前的日期（打印2015-09-28）：
Debug.Print Format$(DateAdd("m", -10, Sample), "yyyy-mm-dd")
```


`DateDiff()` returns a `Long` representing the number of time intervals between two specified dates.

Syntax

```
DateDiff ( interval, date1, date2  [, firstdayofweek] [, firstweekofyear] )
```

- *interval* can be any of the intervals defined in the `DatePart()` function
- *date1* and *date2* are the two dates you want to use in the calculation
- *firstdayofweek* and *firstweekofyear* are optional. Refer to `DatePart()` function for explanations

Examples

```
Sub DateDiffExamples()

    ' Check to see if 2016 is a leap year.
    Dim NumberOfDays As Long
    NumberOfDays = DateDiff("d", #1/1/2016#, #1/1/2017#)

    If NumberOfDays = 366 Then
        Debug.Print "2016 is a leap year."              'This will output.
    End If

    ' Number of seconds in a day
    Dim StartTime As Date
    Dim EndTime As Date
    StartTime = TimeSerial(0, 0, 0)
    EndTime = TimeSerial(24, 0, 0)
    Debug.Print DateDiff("s", StartTime, EndTime)       'prints 86400

End Sub
```

DateAdd()

`DateAdd()` returns a `Date` to which a specified date or time interval has been added.

Syntax

```
DateAdd ( interval, number, date  )
```

- *interval* can be any of the intervals defined in the `DatePart()` function
- *number* Numeric expression that is the number of intervals you want to add. It can be positive (to get dates in the future) or negative (to get dates in the past).
- *date* is a `Date` or literal representing date to which the interval is added

Examples :

```
Sub DateAddExamples()

    Dim Sample As Date
    'Create sample date and time of 2016-07-28 12:34:56
    Sample = DateSerial(2016, 7, 28) + TimeSerial(12, 34, 56)

    ' Date 5 months previously (prints 2016-02-28):
    Debug.Print Format$(DateAdd("m", -5, Sample), "yyyy-mm-dd")

    ' Date 10 months previously (prints 2015-09-28):
    Debug.Print Format$(DateAdd("m", -10, Sample), "yyyy-mm-dd")
```

```
    ' 8个月后的日期（打印2017-03-28）：
Debug.Print Format$(DateAdd("m", 8, Sample), "yyyy-mm-dd")

    ' 18小时前的日期/时间（打印2016-07-27 18:34:56）：
Debug.Print Format$(DateAdd("h", -18, Sample), "yyyy-mm-dd hh:nn:ss")

    ' 36小时后的日期/时间（打印2016-07-30 00:34:56）：
Debug.Print Format$(DateAdd("h", 36, Sample), "yyyy-mm-dd hh:nn:ss")

End Sub
```

第14.5节：转换与创建

CDate()

CDate() 将任意数据类型转换为Date数据类型

```
Sub CDateExamples()

    Dim sample As Date

    ' 将表示日期和时间的字符串转换为日期
    sample = CDate("2001年9月11日 12:34")
    Debug.Print Format$(sample, "yyyy-mm-dd hh:nn:ss")       ' 输出 2001-09-11 12:34:00

    ' 将包含日期的字符串转换为日期
sample = CDate("2001年9月11日")
Debug.Print Format$(sample, "yyyy-mm-dd hh:nn:ss")       ' 输出 2001-09-11 00:00:00

    ' 将包含时间的字符串转换为日期
sample = CDate("12:34:56")
Debug.Print Hour(sample)                            ' 输出 12
    Debug.Print Minute(sample)                          ' 输出 34
    Debug.Print Second(sample)                          ' 输出 56

    ' 查找从纪元日期1899-12-31起的第10000天
sample = CDate(10000)
Debug.Print Format$(sample, "yyyy-mm-dd")        ' 输出 1927-05-18

End Sub
```

请注意，VBA 还有一个弱类型的 CVDate() 函数，其功能与 CDate() 函数相同，区别在于前者返回的是类型为 Variant 的日期，而后者返回的是强类型的 Date。传递给 Date 参数或赋值给 Date 变量时，应优先使用 CDate() 版本；传递给 Variant 参数或赋值给 Variant 变量时，应优先使用 CVDate() 版本。这样可以避免隐式类型转换。

DateSerial()

DateSerial() 函数用于创建日期。它返回指定年、月、日的 Date 类型值。

语法：

```
DateSerial ( 年, 月, 日 )
```

年、月、日参数必须是有效的整数（年份范围为 100 到 9999，月份范围为 1 到 12，日期范围为 1 到 31）。


```
    ' Date in 8 months (prints 2017-03-28):
    Debug.Print Format$(DateAdd("m", 8, Sample), "yyyy-mm-dd")

    ' Date/Time 18 hours previously (prints 2016-07-27 18:34:56):
    Debug.Print Format$(DateAdd("h", -18, Sample), "yyyy-mm-dd hh:nn:ss")

    ' Date/Time in 36 hours (prints 2016-07-30 00:34:56):
    Debug.Print Format$(DateAdd("h", 36, Sample), "yyyy-mm-dd hh:nn:ss")

End Sub
```

Section 14.5: Conversion and Creation

CDate()

`CDate()` converts something from any datatype to a `Date` datatype

```
Sub CDateExamples()

    Dim sample As Date

    ' Converts a String representing a date and time to a Date
    sample = CDate("September 11, 2001 12:34")
    Debug.Print Format$(sample, "yyyy-mm-dd hh:nn:ss")       ' prints 2001-09-11 12:34:00

    ' Converts a String containing a date to a Date
    sample = CDate("September 11, 2001")
    Debug.Print Format$(sample, "yyyy-mm-dd hh:nn:ss")       ' prints 2001-09-11 00:00:00

    ' Converts a String containing a time to a Date
    sample = CDate("12:34:56")
    Debug.Print Hour(sample)                            ' prints 12
    Debug.Print Minute(sample)                          ' prints 34
    Debug.Print Second(sample)                          ' prints 56

    ' Find the 10000th day from the epoch date of 1899-12-31
    sample = CDate(10000)
    Debug.Print Format$(sample, "yyyy-mm-dd")        ' prints 1927-05-18

End Sub
```

Note that VBA also has a loosely typed `CVDate()` that functions in the same way as the `CDate()` function other than returning a date typed `Variant` instead of a strongly typed `Date`. The `CDate()` version should be preferred when passing to a `Date` parameter or assigning to a `Date` variable, and the `CVDate()` version should be preferred when when passing to a `Variant` parameter or assigning to a `Variant` variable. This avoids implicit type casting.

DateSerial()

`DateSerial()` function is used to create a date. It returns a `Date` for a specified year, month, and day.

Syntax:

```
DateSerial ( year, month, day )
```

With year, month and day arguments being valid Integers (Year from 100 to 9999, Month from 1 to 12, Day from 1 to 31).

示例

```
Sub DateSerialExamples()

    ' 构建一个特定日期
    Dim sample As Date
sample = DateSerial(2001, 9, 11)
Debug.Print Format$(sample, "yyyy-mm-dd")                    ' 输出 2001-09-11

    查找某个日期所在月份的第一天。
sample = DateSerial(Year(sample), Month(sample), 1)
    Debug.Print Format$(sample, "yyyy-mm-dd")                    ' 输出 2001-09-11

    ' 查找上个月的最后一天。
sample = DateSerial(Year(sample), Month(sample), 1) - 1
    Debug.Print Format$(sample, "yyyy-mm-dd")                    ' 输出 2001-09-11

End Sub
```

注意 `DateSerial()` 会接受“无效”的日期并计算出一个有效日期。这可以被创造性地用于

良好用途：

正面示例

```
Sub GoodDateSerialExample()

    '计算从今天起45天后的日期
    Dim today As Date
today = DateSerial (2001, 9, 11)
    Dim futureDate As Date
futureDate = DateSerial(Year(today), Month(today), Day(today) + 45)
    Debug.Print Format$(futureDate, "yyyy-mm-dd")            '打印 2009-10-26

End Sub
```

然而，当尝试从未经验证的用户输入创建日期时，更可能引发问题：

负面示例

```
Sub BadDateSerialExample()

    '允许用户输入未经验证的日期信息
    Dim myYear As Long
myYear = InputBox("输入年份")
            '假设用户输入2009
    Dim myMonth As Long
myMonth = InputBox("输入月份")
            假设用户输入2
    Dim myDay As Long
myDay = InputBox("请输入日期")
            '假设用户输入31
Debug.Print Format$(DateSerial(myYear, myMonth, myDay), "yyyy-mm-dd")
            '输出   2009-03-03

End Sub
```


Examples

```
Sub DateSerialExamples()

    ' Build a specific date
    Dim sample As Date
    sample = DateSerial(2001, 9, 11)
    Debug.Print Format$(sample, "yyyy-mm-dd")                    ' prints 2001-09-11

    ' Find the first day of the month for a date.
    sample = DateSerial(Year(sample), Month(sample), 1)
    Debug.Print Format$(sample, "yyyy-mm-dd")                    ' prints 2001-09-11

    ' Find the last day of the previous month.
    sample = DateSerial(Year(sample), Month(sample), 1) - 1
    Debug.Print Format$(sample, "yyyy-mm-dd")                    ' prints 2001-09-11

End Sub
```

Note that `DateSerial()` will accept "invalid" dates and calculate a valid date from it. This can be used creatively for good:

Positive Example

```
Sub GoodDateSerialExample()

    'Calculate 45 days from today
    Dim today As Date
    today = DateSerial (2001, 9, 11)
    Dim futureDate As Date
    futureDate = DateSerial(Year(today), Month(today), Day(today) + 45)
    Debug.Print Format$(futureDate, "yyyy-mm-dd")            'prints 2009-10-26

End Sub
```

However, it is more likely to cause grief when attempting to create a date from unvalidated user input:

Negative Example

```
Sub BadDateSerialExample()

    'Allow user to enter unvalidate date information
    Dim myYear As Long
    myYear = InputBox("Enter Year")
            'Assume user enters 2009
    Dim myMonth As Long
    myMonth = InputBox("Enter Month")
            'Assume user enters 2
    Dim myDay As Long
    myDay = InputBox("Enter Day")
            'Assume user enters 31
    Debug.Print Format$(DateSerial(myYear, myMonth, myDay), "yyyy-mm-dd")
            'prints  2009-03-03

End Sub
```

第15章：数据类型和限制

第15.1节：Variant（变体）

```
Dim Value As Variant    '显式声明
Dim Value               '隐式声明
```

Variant是一种COM数据类型，用于存储和交换任意类型的值，VBA中的任何其他类型都可以赋值给Variant。未显式指定类型（通过As [Type]声明）的变量默认类型为Variant。

Variant在内存中以VARIANT结构存储，该结构由一个字节类型描述符（VARTYPE）后跟6个保留字节，再后面是8字节的数据区组成。对于数值类型（包括日期和布尔值），底层值存储在Variant本身。对于所有其他类型，数据区包含指向底层值的指针。

VARTYPE		Reserved						Data area			
0	1	2	3	4	5	6	7	8	9	10	11

Variant 的底层类型可以通过 VarType() 函数确定，该函数返回存储在类型描述符中的数值，或者通过 TypeName() 函数确定，该函数返回字符串表示：

```
Dim 示例 As Variant
示例 = 42
Debug.Print VarType(示例)    '打印 2 (VT_I2)
Debug.Print TypeName(示例)   '打印 "Integer"
示例 = "Some text"
Debug.Print VarType(示例)    '打印 8 (VT_BSTR)
Debug.Print TypeName(示例)   '打印 "String"
```

由于 Variant 可以存储任何类型的值，未带类型提示的字面量赋值将根据下表隐式转换为相应类型的 Variant。带有类型提示的字面量将被转换为提示类型的 Variant。

值	结果类型
字符串值	字符串
整数范围内的非浮点数	整数
长整型范围内的非浮点数	长整型
长整型范围外的非浮点数	双精度浮点数
所有浮点数	双精度浮点数

注意：除非有特定原因使用Variant类型（例如For Each循环中的迭代器或API要求），
否则一般应避免在常规任务中使用该类型，原因如下：

- 它们不是类型安全的，增加了运行时错误的可能性。例如，存储整数值的Variant会静默地将自身转换为长整型，而不是溢出。
- 它们引入了处理开销，至少需要额外一次指针解引用。
- Variant的内存需求总是比存储其底层类型所需的内存至少多8字节。

转换为Variant的强制类型转换函数是CVar()。

Chapter 15: Data Types and Limits

Section 15.1: Variant

```
Dim Value As Variant    'Explicit
Dim Value               'Implicit
```

A Variant is a COM data type that is used for storing and exchanging values of arbitrary types, and any other type in VBA can be assigned to a Variant. Variables declared without an explicit type specified by `As [Type]` default to Variant.

Variants are stored in memory as a VARIANT structure that consists of a byte type descriptor (VARTYPE) followed by 6 reserved bytes then an 8 byte data area. For numeric types (including Date and Boolean), the underlying value is stored in the Variant itself. For all other types, the data area contains a pointer to the underlying value.

VARTYPE		Reserved						Data area			
0	1	2	3	4	5	6	7	8	9	10	11

The underlying type of a Variant can be determined with either the `VarType()` function which returns the numeric value stored in the type descriptor, or the `TypeName()` function which returns the string representation:

```
Dim Example As Variant
Example = 42
Debug.Print VarType(Example)    'Prints 2 (VT_I2)
Debug.Print TypeName(Example)   'Prints "Integer"
Example = "Some text"
Debug.Print VarType(Example)    'Prints 8 (VT_BSTR)
Debug.Print TypeName(Example)   'Prints "String"
```

Because Variants can store values of any type, assignments from literals without type hints will be implicitly cast to a Variant of the appropriate type according to the table below. Literals with type hints will be cast to a Variant of the hinted type.

Value	Resulting type
String values	String
Non-floating point numbers in Integer range	Integer
Non-floating point numbers in Long range	Long
Non-floating point numbers outside of Long range	Double
All floating point numbers	Double

Note: Unless there is a specific reason to use a Variant (i.e. an iterator in a For Each loop or an API requirement), the type should generally be avoided for routine tasks for the following reasons:

- They are not type safe, increasing the possibility of runtime errors. For example, a Variant holding an Integer value will silently change itself into a Long instead of overflowing.
- They introduce processing overhead by requiring at least one additional pointer dereference.
- The memory requirement for a Variant is always ***at least*** 8 bytes higher than needed to store the underlying type.

The casting function to convert to a Variant is `CVar()`.

第15.2节：布尔型

```
Dim Value As Boolean
```

Boolean 用于存储可以表示为 True 或 False 的值。内部，该数据类型存储为一个 16 位的值，0 表示 False，任何其他值表示 True。

需要注意的是，当 Boolean 被转换为数值类型时，所有位都会被设置为 1。这导致带符号类型的内部表示为 -1，无符号类型（Byte）的内部表示为最大值。

```
Dim Example As Boolean
Example = True
Debug.Print CInt(Example)   '打印 -1
Debug.Print CBool(42)       '打印 True
Debug.Print CByte(True)     '打印 255
```

转换为 Boolean 的转换函数是 CBool()。尽管它在内部表示为 16 位数字，从超出该范围的值转换为 Boolean 是安全的，不会溢出，尽管它会将所有 16 位都设置为 1：

```
Dim Example As Boolean
Example = CBool(2 ^ 17)
Debug.Print CInt(Example)    '打印 -1
Debug.Print CByte(Example)   '打印 255
```

第 15.3 节：字符串

字符串表示一系列字符，有两种类型：

可变长度

```
Dim Value As String
```

可变长度字符串允许追加和截断，并以 COM BSTR 形式存储在内存中。它由一个4 字节无符号整数组成，用于存储字符串的字节长度，后面跟着字符串数据本身，作为宽字符（每个字符 2 字节）存储，并以 2 个空字节结尾。因此，VBA 能处理的最大字符串长度为2,147,483,647 个字符。

指向该结构的内部指针（可通过 StrPtr() 函数获取）指向数据的内存位置，而不是长度前缀。这意味着 VBA 字符串可以直接传递给需要字符数组指针的 API 函数。

由于长度可以变化，VBA 每次给字符串变量赋值时都会重新分配内存，这可能会对反复修改字符串的过程造成性能损失。

固定长度

```
Dim Value As String * 1024    '声明一个长度为 1024 字符的固定长度字符串。
```

固定长度字符串为每个字符分配 2 字节，并以简单的字节数组形式存储在内存中。分配后，字符串长度不可变。它们在内存中不以空字符结尾，因此用非空字符填满分配内存的字符串不适合传递给期望空字符结尾字符串的 API 函数。

固定长度字符串继承了 16 位索引的限制，因此最大长度只能达到 65,535 个字符。
尝试赋值超过可用内存空间的字符串不会导致运行时错误——结果值将被简单截断：

Section 15.2: Boolean

```
Dim Value As Boolean
```

A Boolean is used to store values that can be represented as either True or False. Internally, the data type is stored as a 16 bit value with 0 representing False and any other value representing True.

It should be noted that when a Boolean is cast to a numeric type, all of the bits are set to 1. This results in an internal representation of -1 for signed types and the maximum value for an unsigned type (Byte).

```
Dim Example As Boolean
Example = True
Debug.Print CInt(Example)   'Prints -1
Debug.Print CBool(42)       'Prints True
Debug.Print CByte(True)     'Prints 255
```

The casting function to convert to a Boolean is `CBool()`. Even though it is represented internally as a 16 bit number, casting to a Boolean from values outside of that range is safe from overflow, although it sets all 16 bits to 1:

```
Dim Example As Boolean
Example = CBool(2 ^ 17)
Debug.Print CInt(Example)    'Prints -1
Debug.Print CByte(Example)   'Prints 255
```

Section 15.3: String

A String represents a sequence of characters, and comes in two flavors:

Variable length

```
Dim Value As String
```

A variable length String allows appending and truncation and is stored in memory as a COM BSTR. This consists of a 4 byte unsigned integer that stores the length of the String in bytes followed by the string data itself as wide characters (2 bytes per character) and terminated with 2 null bytes. Thus, the maximum string length that can be handled by VBA is 2,147,483,647 characters.

The internal pointer to the structure (retrievable by the `StrPtr()` function) points to the memory location of the *data*, not the length prefix. This means that a VBA String can be passed directly API functions that require a pointer to a character array.

Because the length can change, VBA reallocates memory for a String *every time the variable is assigned to*, which can impose performance penalties for procedures that alter them repeatedly.

Fixed length

```
Dim Value As String * 1024    'Declares a fixed length string of 1024 characters.
```

Fixed length strings are allocated 2 bytes for each character and are stored in memory as a simple byte array. Once allocated, the length of the String is immutable. They are ***not*** null terminated in memory, so a string that fills the memory allocated with non-null characters is unsuitable for passing to API functions expecting a null terminated string.

Fixed length strings carry over a legacy 16 bit index limitation, so can only be up to 65,535 characters in length. Attempting to assign a value longer than the available memory space will not result in a runtime error - instead the resulting value will simply be truncated:

```
Dim Foobar As String * 5
Foobar = "Foo" & "bar"
Debug.Print Foobar          '打印 "Fooba"
```

用于转换为任一类型字符串的强制转换函数是 CStr()。

第15.4节：字节

```
Dim Value As Byte
```

字节是一种无符号的8位数据类型。它可以表示0到255之间的整数，尝试存储超出该范围的值将导致运行时错误6：溢出。字节是VBA中唯一的内置无符号类型。

转换为字节的强制类型转换函数是CByte()。对于从浮点类型的转换，结果会四舍五入到最接近的整数，.5时向上取整。

字节数组和字符串

字符串和字节数组可以通过简单赋值相互替换（无需转换函数）。

例如：

```
Sub ByteToStringAndBack()

Dim str As String
str = "Hello, World!"

Dim byt() As Byte
byt = str

Debug.Print byt(0)  ' 72

Dim str2 As String
str2 = byt

Debug.Print str2    ' Hello, World!

End Sub
```

为了能够编码Unicode字符，字符串中的每个字符在数组中占用两个字节，且低位字节在前。例如：

```
Sub UnicodeExample()

Dim str As String
str = ChrW(&H2123) & "."  ' 诗句字符和一个点

Dim byt() As Byte
byt = str

Debug.Print byt(0), byt(1), byt(2), byt(3)  ' 输出：35,33,46,0

End Sub
```



```
Dim Foobar As String * 5
Foobar = "Foo" & "bar"
Debug.Print Foobar          'Prints "Fooba"
```

The casting function to convert to a String of either type is `CStr()`.

Section 15.4: Byte

```
Dim Value As Byte
```

A Byte is an unsigned 8 bit data type. It can represent integer numbers between 0 and 255 and attempting to store a value outside of that range will result in runtime error 6: Overflow. Byte is the only intrinsic unsigned type available in VBA.

The casting function to convert to a Byte is `CByte()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Byte Arrays and Strings

Strings and byte arrays can be substituted for one another through simple assignment (no conversion functions necessary).

For example:

```
Sub ByteToStringAndBack()

Dim str As String
str = "Hello, World!"

Dim byt() As Byte
byt = str

Debug.Print byt(0)  ' 72

Dim str2 As String
str2 = byt

Debug.Print str2    ' Hello, World!

End Sub
```

In order to be able to encode Unicode characters, each character in the string takes up two bytes in the array, with the least significant byte first. For example:

```
Sub UnicodeExample()

Dim str As String
str = ChrW(&H2123) & "."  ' Versicle character and a dot

Dim byt() As Byte
byt = str

Debug.Print byt(0), byt(1), byt(2), byt(3)  ' Prints: 35,33,46,0

End Sub
```

第15.5节：货币

```
Dim Value As Currency
```

Currency 是一种带符号的 64 位浮点数据类型，类似于 Double，但通过乘以 10,000 进行缩放，以提供小数点后 4 位数字的更高精度。Currency 变量可以存储的值范围是 -922,337,203,685,477.5808 到 922,337,203,685,477.5807，使其在32位应用程序中拥有所有内置类型中最大的容量。正如数据类型名称所示，使用此数据类型被认为是最佳实践表示货币计算，因为缩放有助于避免舍入误差。

转换为货币的强制类型转换函数是CCur()。

第15.6节：小数

```
Dim Value As Variant
Value = CDec(1.234)

将值设置为最小可能的十进制值
Value = CDec("0.0000000000000000000000000001")
```

十进制（Decimal）数据类型仅作为Variant的子类型提供，因此您必须将任何需要包含十进制值的变量声明为Variant，然后使用CDec函数赋值为十进制。关键字Decimal是保留字（这表明VBA最终可能会为该类型添加一流支持），因此Decimal不能用作变量名或过程名。

Decimal 类型需要 14 字节的内存（除了父 Variant 所需的字节外），并且可以存储最多 28 位小数的数字。对于没有小数位的数字，允许的值范围是 -79,228,162,514,264,337,593,543,950,335 到 +79,228,162,514,264,337,593,543,950,335（含）。对于具有最大 28 位小数的数字，允许的值范围是 -7.9228162514264337593543950335 到 +7.9228162514264337593543950335（含）。

第15.7节：整数

```
Dim Value As Integer
```

整数（Integer）是一种有符号的16位数据类型。它可以存储范围在-32,768到32,767之间的整数，尝试存储超出该范围的值将导致运行时错误6：溢出（Overflow）。

整数在内存中以小端序（little-endian）存储，负数以二进制补码（two's complement）表示。

注意，通常情况下，除非较小的数据类型是某个类型（Type）的成员，或者由于API调用约定或其他原因必须为2字节，否则最好使用Long而不是Integer。在大多数情况下，VBA内部将Integer视为32位，因此使用较小类型通常没有优势。此外，每次使用Integer类型时都会产生性能损失，因为它会被隐式转换为Long。

转换为Integer的强制类型转换函数是CInt()。对于从浮点类型的转换，结果会四舍五入到最接近的整数，.5时向上取整。

第15.8节：Long

```
Dim Value As Long
```

Long是一种有符号的32位数据类型。它可以存储范围在-2,147,483,648到2,147,483,647之间的整数，

Section 15.5: Currency

```
Dim Value As Currency
```

A Currency is a signed 64 bit floating point data type similar to a Double, but scaled by 10,000 to give greater precision to the 4 digits to the right of the decimal point. A Currency variable can store values from -922,337,203,685,477.5808 to 922,337,203,685,477.5807, giving it the largest capacity of any intrinsic type in a 32 bit application. As the name of the data type implies, it is considered best practice to use this data type when representing monetary calculations as the scaling helps to avoid rounding errors.

The casting function to convert to a Currency is `CCur()`.

Section 15.6: Decimal

```
Dim Value As Variant
Value = CDec(1.234)

'Set Value to the smallest possible Decimal value
Value = CDec("0.0000000000000000000000000001")
```

The `Decimal` data-type is *only* available as a sub-type of `Variant`, so you must declare any variable that needs to contain a `Decimal` as a `Variant` and *then* assign a `Decimal` value using the `CDec` function. The keyword `Decimal` is a reserved word (which suggests that VBA was eventually going to add first-class support for the type), so `Decimal` cannot be used as a variable or procedure name.

The `Decimal` type requires 14 bytes of memory (in addition to the bytes required by the parent Variant) and can store numbers with up to 28 decimal places. For numbers without any decimal places, the range of allowed values is -79,228,162,514,264,337,593,543,950,335 to +79,228,162,514,264,337,593,543,950,335 inclusive. For numbers with the maximum 28 decimal places, the range of allowed values is -7.9228162514264337593543950335 to +7.9228162514264337593543950335 inclusive.

Section 15.7: Integer

```
Dim Value As Integer
```

An Integer is a signed 16 bit data type. It can store integer numbers in the range of -32,768 to 32,767 and attempting to store a value outside of that range will result in runtime error 6: Overflow.

Integers are stored in memory as little-endian values with negatives represented as a two's complement.

Note that in general, it is better practice to use a Long rather than an Integer unless the smaller type is a member of a Type or is required (either by an API calling convention or some other reason) to be 2 bytes. In most cases VBA treats Integers as 32 bit internally, so there is usually no advantage to using the smaller type. Additionally, there is a performance penalty incurred every time an Integer type is used as it is silently cast as a Long.

The casting function to convert to an Integer is `CInt()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Section 15.8: Long

```
Dim Value As Long
```

A Long is a signed 32 bit data type. It can store integer numbers in the range of -2,147,483,648 to 2,147,483,647 and

尝试存储超出该范围的值将导致运行时错误6：溢出（Overflow）。

Long在内存中以小端序（little-endian）存储，负数以二进制补码（two's complement）表示。

请注意，由于 Long 在 32 位操作系统中与指针的宽度相匹配，Long 通常用于存储和传递指针给 API 函数及从 API 函数接收指针。

转换为 Long 的强制转换函数是 CLng()。对于从浮点类型的转换，结果会四舍五入到最接近的整数值，.5 时向上取整。

第 15.9 节：Single（单精度浮点数）

```
Dim Value As Single
```

Single 是一种有符号的 32 位浮点数据类型。它在内部使用 小端 IEEE 754 内存布局存储。因此，该数据类型可以表示的值没有固定范围——限制的是存储值的精度。Single 可以存储范围在 -16,777,216 到 16,777,216 之间的 整数 值而不会丢失精度。浮点数的精度取决于指数部分。

如果赋值超过大约 2^128，Single 会溢出。对于负指数不会溢出，尽管在达到上限之前可用的精度可能会受到质疑。

与所有浮点数一样，进行相等比较时应谨慎。最佳做法是包含一个适合所需精度的误差值（delta）。

转换为 Single 的强制转换函数是 CSng()。

第 15.10 节：Double（双精度浮点数）

```
Dim Value As Double
```

Double 是一种有符号的 64 位浮点数据类型。与 Single 类似，它在内部使用小端 IEEE 格式存储 754 内存布局和相同的精度注意事项应被遵守。Double 可以存储整数
数值范围在-9,007,199,254,740,992到9,007,199,254,740,992之间，且不会丢失精度。浮点数的精度取决于指数。

如果赋值大约大于2的1024次方，Double类型将会溢出。负指数不会导致溢出，
尽管在达到上限之前，可用的精度可能会受到质疑。

将类型转换为双精度浮点数的函数是CDbl()。

第15.11节：日期

日期值

日期类型在内部表示为带符号的64位浮点数据类型，小数点左侧的值表示自1899年12月30日纪元日期起的天数（但请参见下面的注释）。小数点右侧的值表示时间，作为一天的分数。因此，整数日期的时间部分为上午12:00:00，x.5的时间部分为下午12:00:00。

日期的有效值介于公元100年1月1日和公元9999年12月31日之间。由于Double类型的范围更大，因此通过赋予超出该范围的值，有可能导致日期溢出。

attempting to store a value outside of that range will result in runtime error 6: Overflow.

Longs are stored in memory as little-endian values with negatives represented as a two's complement.

Note that since a Long matches the width of a pointer in a 32 bit operating system, Longs are commonly used for storing and passing pointers to and from API functions.

The casting function to convert to a Long is `CLng()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Section 15.9: Single

```
Dim Value As Single
```

A Single is a signed 32 bit floating point data type. It is stored internally using a little-endian IEEE 754 memory layout. As such, there is not a fixed range of values that can be represented by the data type - what is limited is the precision of value stored. A Single can store a value ***integer*** values in the range of -16,777,216 to 16,777,216 without a loss of precision. The precision of floating point numbers depends on the exponent.

A Single will overflow if assigned a value greater than roughly 2128. It will not overflow with negative exponents, although the usable precision will be questionable before the upper limit is reached.

As with all floating point numbers, care should be taken when making equality comparisons. Best practice is to include a delta value appropriate to the required precision.

The casting function to convert to a Single is `CSng()`.

Section 15.10: Double

```
Dim Value As Double
```

A Double is a signed 64 bit floating point data type. Like the Single, it is stored internally using a little-endian IEEE 754 memory layout and the same precautions regarding precision should be taken. A Double can store ***integer*** values in the range of -9,007,199,254,740,992 to 9,007,199,254,740,992 without a loss of precision. The precision of floating point numbers depends on the exponent.

A Double will overflow if assigned a value greater than roughly 21024. It will not overflow with negative exponents, although the usable precision will be questionable before the upper limit is reached.

The casting function to convert to a Double is `CDbl()`.

Section 15.11: Date

```
Dim Value As Date
```

A Date type is represented internally as a signed 64 bit floating point data type with the value to the left of the decimal representing the number of days from the epoch date of December 30th, 1899 (although see the note below). The value to the right of the decimal represents the time as a fractional day. Thus, an integer Date would have a time component of 12:00:00AM and x.5 would have a time component of 12:00:00PM.

Valid values for Dates are between January 1st 100 and December 31st 9999. Since a Double has a larger range, it is possible to overflow a Date by assigning values outside of that range.

因此，它可以与 Double 互换用于日期计算：

```
Dim MyDate As Double
MyDate = 0                                      '纪元日期。
Debug.Print Format$(MyDate, "yyyy-mm-dd")    '输出 1899-12-30。
MyDate = MyDate + 365
Debug.Print Format$(MyDate, "yyyy-mm-dd")    '输出 1900-12-30。
```

转换为日期的强制转换函数是 CDate()，接受任何数值类型或字符串日期/时间表示。需要注意的是，字符串日期表示将根据当前使用的区域设置进行转换，因此如果代码需要可移植性，应避免直接强制转换。

第15.12节：LongLong

```
Dim Value As LongLong
```

LongLong 是有符号的64位数据类型，仅在64位应用程序中可用。在64位操作系统上运行的32位应用程序中不可用。它可以存储的整数值范围是 -9,223,372,036,854,775,808 到 9,223,372,036,854,775,807，尝试存储超出该范围的值将导致运行时错误6：溢出。

LongLongs 在内存中以小端格式存储，负数以二进制补码表示。

LongLong 数据类型是作为 VBA 对 64 位操作系统支持的一部分引入的。在 64 位应用程序中，该值可用于存储和传递指向 64 位 API 的指针。

转换为 LongLong 的强制转换函数是CLngLng()。对于从浮点类型的转换，结果会四舍五入到最接近的整数值，.5 向上取整。

第 15.13 节：LongPtr

```
Dim Value As LongPtr
```

LongPtr 是为了支持 64 位平台而引入 VBA 的。在 32 位系统上，它被视为 Long 类型；在 64 位系统上，它被视为 LongLong 类型。

它的主要用途是在两种架构上提供一种可移植的方式来存储和传递指针（参见编译时更改代码行为）。

虽然在 API 调用中操作系统将其视为内存地址，但应注意 VBA 将其视为有符号类型（因此会受到无符号到有符号溢出的影响）。因此，使用 LongPtr 进行指针运算时，不应使用>或<比较。这个“怪癖”也意味着简单地添加指向内存中有效地址的偏移量可能会导致溢出错误，因此在 VBA 中操作指针时应谨慎。

转换为 LongPtr 的强制转换函数是CLngPtr()。对于从浮点类型的转换，结果会四舍五入到最接近的整数值，.5 向上取整（尽管由于它通常是内存地址，将其用作浮点计算的赋值目标充其量是危险的）。

As such, it can be used interchangeably with a Double for Date calculations:

```
Dim MyDate As Double
MyDate = 0                                      'Epoch date.
Debug.Print Format$(MyDate, "yyyy-mm-dd")    'Prints 1899-12-30.
MyDate = MyDate + 365
Debug.Print Format$(MyDate, "yyyy-mm-dd")    'Prints 1900-12-30.
```

The casting function to convert to a Date is `CDate()`, which accepts any numeric type string date/time representation. It is important to note that string representations of dates will be converted based on the current locale setting in use, so direct casts should be avoided if the code is meant to be portable.

Section 15.12: LongLong

```
Dim Value As LongLong
```

A LongLong is a signed 64 bit data type and is only available in 64 bit applications. It is **not** available in 32 bit applications running on 64 bit operating systems. It can store integer values in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 and attempting to store a value outside of that range will result in runtime error 6: Overflow.

LongLongs are stored in memory as little-endian values with negatives represented as a two's complement.

The LongLong data type was introduced as part of VBA's 64 bit operating system support. In 64 bit applications, this value can be used to store and pass pointers to 64 bit APIs.

The casting function to convert to a LongLong is `CLngLng()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Section 15.13: LongPtr

```
Dim Value As LongPtr
```

The LongPtr was introduced into VBA in order to support 64 bit platforms. On a 32 bit system, it is treated as a Long and on 64 bit systems it is treated as a LongLong.

It's primary use is in providing a portable way to store and pass pointers on both architectures (See Changing code behavior at compile time.

Although it is treated by the operating system as a memory address when used in API calls, it should be noted that VBA treats it like signed type (and therefore subject to unsigned to signed overflow). For this reason, any pointer arithmetic performed using LongPtrs should not use `>` or `<` comparisons. This "quirk" also makes it possible that adding simple offsets pointing to valid addresses in memory can cause overflow errors, so caution should be taken when working with pointers in VBA.

The casting function to convert to a LongPtr is `CLngPtr()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up (although since it is usually a memory address, using it as an assignment target for a floating point calculation is dangerous at best).

第 16 章：命名约定

第 16.1 节：变量名

变量保存数据。命名时应根据其用途命名，而非根据数据类型或作用域，使用名词。如果你觉得有必要给变量编号（例如thing1，thing2，thing3），那么应考虑使用合适的数据结构代替（例如数组、集合(Collection)或字典(Dictionary)）。

表示可迭代*集合(set)*的变量名称——例如数组、集合(Collection)、字典(Dictionary)或单元格范围(Range)——应使用复数形式。

一些常见的VBA命名规范如下：

过程级变量：

驼峰式命名(camelCase)

```
Public Sub ExampleNaming(ByVal inputValue As Long, ByRef inputVariable As Long)

    Dim procedureVariable As Long
    Dim someOtherVariable As String

End Sub
```

模块级变量：

帕斯卡命名法(PascalCase)

```
Public GlobalVariable As Long
Private ModuleVariable As String
```

对于常量：

SHOUTY_SNAKE_CASE 通常用于区分常量和变量：

```
Public Const GLOBAL_CONSTANT As String = "Project Version #1.000.000.001"
Private Const MODULE_CONSTANT As String = "Something relevant to this Module"

Public Sub SomeProcedure()

    Const PROCEDURE_CONSTANT As Long = 10

End Sub
```

然而 PascalCase 命名使代码看起来更整洁，而且同样有效，因为 IntelliSense 使用不同的图标来区分变量和常量：

Chapter 16: Naming Conventions

Section 16.1: Variable Names

Variables hold data. Name them after what they're used for, **not after their data type** or scope, using a **noun**. If you feel compelled to *number* your variables (e.g. `thing1`, `thing2`, `thing3`), then consider using an appropriate data structure instead (e.g. an array, a `Collection`, or a `Dictionary`).

Names of variables that represent an iteratable *set* of values - e.g. an array, a `Collection`, a `Dictionary`, or a `Range` of cells, should be plural.

Some common VBA naming conventions go thus:

For procedure-level Variables:

`camelCase`

```
Public Sub ExampleNaming(ByVal inputValue As Long, ByRef inputVariable As Long)

    Dim procedureVariable As Long
    Dim someOtherVariable As String

End Sub
```

For module-level Variables:

`PascalCase`

```
Public GlobalVariable As Long
Private ModuleVariable As String
```

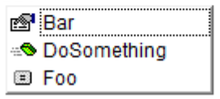
For Constants:

`SHOUTY_SNAKE_CASE` is commonly used to differentiate constants from variables:

```
Public Const GLOBAL_CONSTANT As String = "Project Version #1.000.000.001"
Private Const MODULE_CONSTANT As String = "Something relevant to this Module"

Public Sub SomeProcedure()

    Const PROCEDURE_CONSTANT As Long = 10

End Sub
```

However `PascalCase` names make cleaner-looking code and are just as good, given IntelliSense uses different icons for variables and constants:

```
Option Explicit
Public Const Foo As String = "foo"
Public Bar As String

Sub DoSomething()
    Module1.
End Sub
```



匈牙利命名法

> 根据用途为它们命名，而不是根据数据类型或作用域命名。

“匈牙利命名法使得更容易看出变量的类型”

如果你编写的代码（如过程）遵循单一职责原则（正如应该的那样），你永远不应该在任何过程的顶部看到一屏的变量声明；应尽可能靠近首次使用处声明变量，并且如果你用显式类型声明它们，它们的数据类型将始终一目了然。

VBE 的 `Ctrl`+`i` 快捷键也可以用来在工具提示中显示变量的类型。

变量的用途比其数据类型更有用，尤其是在像VBA这样的语言中
，它会根据需要自动且隐式地将一种类型转换为另一种类型。

请考虑本例中的 iFile 和 strFile：

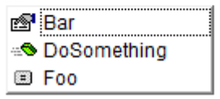
```
Function bReadFile(ByVal strFile As String, ByRef strData As String) As Boolean
    Dim bRetVal As Boolean
    Dim iFile As Integer

    On Error GoTo CleanFail

iFile = FreeFile
    Open strFile For Input As #iFile
   Input #iFile, strData

bRetVal = True

CleanExit:
Close #iFile
   bReadFile = bRetVal
    Exit Function
CleanFail:
bRetVal = False
    Resume CleanExit
End Function
```

比较于：

```
Function CanReadFile(ByVal path As String, ByRef outContent As String) As Boolean
    On Error GoTo CleanFail

    Dim handle As Integer
   handle = FreeFile
```



```
Option Explicit
Public Const Foo As String = "foo"
Public Bar As String

Sub DoSomething()
    Module1.
End Sub
```



Hungarian Notation

> Name them after what they're used for, **not after their data type** or scope.

"Hungarian Notation makes it easier to see what the type of a variable is"

If you write your code such as procedures adhere to the *Single Responsibility Principle* (as it should), you should never be looking at a screenful of variable declarations at the top of any procedure; declare variables as close as possible to their first usage, and their data type will always be in plain sight if you declare them with an explicit type. The VBE's `Ctrl`+`i` shortcut can be used to display a variable's type in a tooltip, too.

What a variable is used for is much more useful information than its data type, *especially* in a language such as VBA which happily and implicitly converts a type into another as needed.

Consider `iFile` and `strFile` in this example:

```
Function bReadFile(ByVal strFile As String, ByRef strData As String) As Boolean
    Dim bRetVal As Boolean
    Dim iFile As Integer

    On Error GoTo CleanFail

    iFile = FreeFile
    Open strFile For Input As #iFile
    Input #iFile, strData

    bRetVal = True

CleanExit:
    Close #iFile
    bReadFile = bRetVal
    Exit Function
CleanFail:
    bRetVal = False
    Resume CleanExit
End Function
```

Compare to:

```
Function CanReadFile(ByVal path As String, ByRef outContent As String) As Boolean
    On Error GoTo CleanFail

    Dim handle As Integer
    handle = FreeFile
```

```
Open path For Input As #handle
    Input #handle, outContent

    Dim result As Boolean
    result = True

CleanExit:
Close #handle
    CanReadFile = result
    Exit Function
CleanFail:
result = False
    Resume CleanExit
End Function
```

strData 在上面的示例中是通过ByRef传递的，但除了我们有幸看到它被明确地这样传递之外，没有任何迹象表明strData实际上是被函数返回的。

下面的例子将其命名为outContent；这个**out**前缀正是匈牙利命名法发明的目的：帮助明确*变量的用途*，在本例中清楚地将其标识为“输出”参数。

这是有用的，因为仅靠 IntelliSense 并不会显示ByRef，即使参数是明确地通过引用传递的：

```
Public Sub DoSomething()
    if CanReadFile(path, 
End Sub CanReadFile(ByVal path As String, outContent As String) As Boolean
```

这导致了.....

正确使用匈牙利命名法

匈牙利命名法最初与变量类型无关。事实上，正确使用的匈牙利命名法实际上是有用的。考虑这个小例子（为简洁起见去掉了**ByVal**和**As** Integer）：

```
Public Sub Copy(iX1, iY1, iX2, iY2)
End Sub
```

比较于：

```
Public Sub Copy(srcColumn, srcRow, dstColumn, dstRow)
End Sub
```

这里的src和dst是匈牙利命名法前缀，它们传达了有用的信息，这些信息无法仅通过参数名或 IntelliSense 显示的声明类型推断出来。

当然，有更好的方式来传达所有信息，使用合适的抽象和可以大声朗读且有意义的真实词汇——作为一个人为的例子：

```
Type Coordinate
    RowIndex As Long
    ColumnIndex As Long
结束 类型

Sub Copy(source As Coordinate, destination As Coordinate)
End Sub
```



```
    Open path For Input As #handle
    Input #handle, outContent

    Dim result As Boolean
    result = True

CleanExit:
    Close #handle
    CanReadFile = result
    Exit Function
CleanFail:
    result = False
    Resume CleanExit
End Function
```

`strData` is passed **ByRef** in the top example, but beside the fact that we're lucky enough to see that it's *explicitly* passed as such, there's no indication that `strData` is actually *returned* by the function.

The bottom example names it `outContent`; this **out** prefix is what Hungarian Notation was invented for: to help clarify *what a variable is used for*, in this case to clearly identify it as an "out" parameter.

This is useful, because IntelliSense by itself doesn't display **ByRef**, even when the parameter is *explicitly* passed by reference:

```
Public Sub DoSomething()
    if CanReadFile(path, 
End Sub CanReadFile(ByVal path As String, outContent As String) As Boolean
```

Which leads to...

Hungarian Done Right

Hungarian Notation originally didn't have anything to do with variable types. In fact, Hungarian Notation *done right* is actually useful. Consider this small example (**ByVal** and **As** Integer removed for brevety):

```
Public Sub Copy(iX1, iY1, iX2, iY2)
End Sub
```

Compare to:

```
Public Sub Copy(srcColumn, srcRow, dstColumn, dstRow)
End Sub
```

`src` and `dst` are *Hungarian Notation* prefixes here, and they convey *useful* information that cannot otherwise already be inferred from the parameter names or IntelliSense showing us the declared type.

Of course there's a better way to convey it all, using proper *abstraction* and real words that can be pronounced out loud and make sense - as a contrived example:

```
Type Coordinate
    RowIndex As Long
    ColumnIndex As Long
End Type

Sub Copy(source As Coordinate, destination As Coordinate)
End Sub
```

第16.2节：过程名称

过程执行某事。用动词来命名它们，反映它们正在做的事情。如果无法准确命名一个过程，说明该过程可能做了太多事情，需要拆分成更小、更专业的过程。

一些常见的VBA命名规范如下：

对于所有过程：

帕斯卡命名法(PascalCase)

```
Public Sub DoThing()

End Sub

Private Function ReturnSomeValue() As [DataType]

End Function
```

对于事件处理程序过程：

对象名_事件名

```
Public Sub Workbook_Open()

End Sub

Public Sub Button1_Click()

End Sub
```

事件处理程序通常由VBE自动命名；如果重命名它们而不重命名对象和/或所处理的事件，代码将会出错——代码可以运行和编译，但处理程序过程将变成孤立状态，永远不会被执行。

布尔成员

考虑一个返回布尔值的函数：

```
Function bReadFile(ByVal strFile As String, ByRef strData As String) As Boolean
End Function
```

比较于：

```
Function CanReadFile(ByVal path As String, ByRef outContent As String) As Boolean
End Function
```

前缀Can的作用与前缀b相同：它标识函数的返回值为Boolean。但Can比b读起来更自然：

```
If CanReadFile(path, content) Then
```

相比之下：

```
If bReadFile(strFile, strData) Then
```


Section 16.2: Procedure Names

Procedures *do something*. Name them after what they're doing, using a **verb**. If accurately naming a procedure is not possible, likely the procedure is *doing too many things* and needs to be broken down into smaller, more specialized procedures.

Some common VBA naming conventions go thus:

For all Procedures:

`PascalCase`

```
Public Sub DoThing()

End Sub

Private Function ReturnSomeValue() As [DataType]

End Function
```

For event handler procedures:

`ObjectName_EventName`

```
Public Sub Workbook_Open()

End Sub

Public Sub Button1_Click()

End Sub
```

Event handlers are usually automatically named by the VBE; renaming them without renaming the object and/or the handled event will break the code - the code will run and compile, but the handler procedure will be orphaned and will never be executed.

Boolean Members

Consider a Boolean-returning function:

```
Function bReadFile(ByVal strFile As String, ByRef strData As String) As Boolean
End Function
```

Compare to:

```
Function CanReadFile(ByVal path As String, ByRef outContent As String) As Boolean
End Function
```

The `Can` prefix *does* serve the same purpose as the `b` prefix: it identifies the function's return value as a `Boolean`. But `Can` reads better than `b`:

```
If CanReadFile(path, content) Then
```

Compared to:

```
If bReadFile(strFile, strData) Then
```

考虑在返回布尔值的成员（函数和属性）前使用前缀，如Can、Is或Has，但仅在有实际意义时使用。这符合当前微软命名指南的要求。

Consider using prefixes such as `Can`, `Is` or `Has` in front of Boolean-returning members (functions and properties), but only when it adds value. This conforms with the current Microsoft naming guidelines.

第17章：数据结构

[待办：本主题应作为所有基础计算机科学101数据结构的示例，并附带一些关于如何在VBA中实现数据结构的概述说明。这将是一个很好的机会，将VBA文档中与类相关主题介绍的概念结合并加以强化。]

第17.1节：链表

此链表示例实现了集合抽象数据类型的操作。

单链节点类

```
Option Explicit

私有值作为Variant类型
私有下一个节点作为单链节点"Next"是VBA中的关键字，因此不是有效的变量名
```

链表类

```
Option Explicit

私有头节点作为单链节点

'集合类型操作

公共子程序添加(值作为Variant类型)
    定义节点作为单链节点

    设置 node = 新建 单链节点
    node.value = value
    设置 node.nextNode = head

    设置 head = node
End Sub

公共子程序 Remove(value 作为 Variant)
    声明 node 作为 单链节点
    声明 prev 作为 单链节点

    设置 node = head

    当 node 不为 空
        如果 node.value = value 则
            '移除节点
            如果 node 是 head 则
                设置 head = node.nextNode
            否则
                设置 prev.nextNode = node.nextNode
            结束如果
            退出子程序
        End If
        设置 prev = node
        设置 node = node.nextNode
    循环结束

End Sub
```


Chapter 17: Data Structures

[TODO: This topic should be an example of all the basic CS 101 data structures along with some explanation as an overview of how data structures can be implemented in VBA. This would be a good opportunity to tie in and reinforce concepts introduced in Class-related topics in VBA documentation.]

Section 17.1: Linked List

This linked list example implements Set abstract data type operations.

SinglyLinkedNode class

```
Option Explicit

Private Value As Variant
Private NextNode As SinglyLinkedNode '"Next" is a keyword in VBA and therefore is not a valid
variable name
```

LinkedList class

```
Option Explicit

Private head As SinglyLinkedNode

'Set type operations

Public Sub Add(value As Variant)
    Dim node As SinglyLinkedNode

    Set node = New SinglyLinkedNode
    node.value = value
    Set node.nextNode = head

    Set head = node
End Sub

Public Sub Remove(value As Variant)
    Dim node As SinglyLinkedNode
    Dim prev As SinglyLinkedNode

    Set node = head

    While Not node Is Nothing
        If node.value = value Then
            'remove node
            If node Is head Then
                Set head = node.nextNode
            Else
                Set prev.nextNode = node.nextNode
            End If
            Exit Sub
        End If
        Set prev = node
        Set node = node.nextNode
    Wend

End Sub
```

```
公共函数 Exists(value 作为 Variant) 作为 布尔值
    声明 node 作为 单链节点

    设置 节点 = 头节点
    While Not node Is Nothing
        If node.value = value Then
            Exists = True
            Exit Function
        End If
        Set node = node.nextNode
    Wend
End Function

Public Function Count() As Long
    Dim node As SinglyLinkedNode

    设置 节点 = 头节点

    While Not node Is Nothing
        Count = Count + 1
        Set node = node.nextNode
    Wend

End Function
```

第17.2节：二叉树

这是一个不平衡的二叉搜索树的示例。二叉树在概念上结构为从共同根节点向下延伸的节点层级，每个节点有两个子节点：左子节点和右子节点。例如，假设数字7、5、9、3、11、6、12、14和15被插入到一个二叉树中。其结构如下所示。请注意，这棵二叉树不是平衡的，平衡性是保证查找性能的一个理想特性——有关自平衡二叉搜索树的示例，请参见AVL树。

```
          7
        / \
       5   9
      / \   \
     3   6   11
              \
               12
                 \
                  14
                 \
                    15
```

二叉树节点类

```
Option Explicit

Public left As 二叉树节点
Public right As 二叉树节点
Public key As Variant
Public value As Variant
```

二叉树类

[TODO]


```
Public Function Exists(value As Variant) As Boolean
    Dim node As SinglyLinkedNode

    Set node = head
    While Not node Is Nothing
        If node.value = value Then
            Exists = True
            Exit Function
        End If
        Set node = node.nextNode
    Wend
End Function

Public Function Count() As Long
    Dim node As SinglyLinkedNode

    Set node = head

    While Not node Is Nothing
        Count = Count + 1
        Set node = node.nextNode
    Wend

End Function
```

Section 17.2: Binary Tree

This is an example of an unbalanced binary search tree. A binary tree is structured conceptually as a hierarchy of nodes descending downward from a common root, where each node has two children: left and right. For example, suppose the numbers 7, 5, 9, 3, 11, 6, 12, 14 and 15 were inserted into a BinaryTree. The structure would be as below. Note that this binary tree is not balanced, which can be a desirable characteristic for guaranteeing the performance of lookups - see AVL trees for an example of a self-balancing binary search tree.

```
          7
        / \
       5   9
      / \   \
     3   6   11
              \
               12
                 \
                  14
                 \
                    15
```

BinaryTreeNode class

```
Option Explicit

Public left As BinaryTreeNode
Public right As BinaryTreeNode
Public key As Variant
Public value As Variant
```

BinaryTree class

[TODO]

第18章：数组

第18.1节：多维数组

多维数组

顾名思义，多维数组是包含多个维度的数组，通常是二维或三维，但最多可以有32个维度。

多维数组的工作原理类似于具有多个层级的矩阵，以一维、二维和三维的比较为例。

一维是典型的数组，看起来像一个元素列表。

```
Dim 1D(3) as Variant

*1D - 视觉效果*
(0)
(1)
(2)
```

二维看起来像数独网格或Excel表格，初始化数组时需要定义数组的行数和列数。

```
Dim 2D(3,3) as Variant
这将生成一个3x3的网格

*2D - 视觉效果*
(0,0) (0,1) (0,2)
(1,0) (1,1) (1,2)
(2,0) (2,1) (2,2)
```

三维开始看起来像魔方，初始化数组时需要定义行数、列数和层数/深度。

```
Dim 3D(3,3,2) as Variant
这将生成一个3x3x3的网格

*3D - 视觉效果*
第一层                  第二层                  第三层
        前面                    中间                    后面
(0,0,0) (0,0,1) (0,0,2) ¦ (1,0,0) (1,0,1) (1,0,2) ¦ (2,0,0) (2,0,1) (2,0,2)
(0,1,0) (0,1,1) (0,1,2) ¦ (1,1,0) (1,1,1) (1,1,2) ¦ (2,1,0) (2,1,1) (2,1,2)
(0,2,0) (0,2,1) (0,2,2) ¦ (1,2,0) (1,2,1) (1,2,2) ¦ (2,2,0) (2,2,1) (2,2,2)
```

进一步的维度可以被视为3D的乘积，因此一个4D(1,3,3,3)将是两个并排的3D数组。

二维数组

创建

下面的例子将是一个员工列表的汇编，每个员工在列表中将有一组信息（名字、姓氏、地址、邮箱、电话……），这个例子本质上是将信息存储在数组中

Chapter 18: Arrays

Section 18.1: Multidimensional Arrays

Multidimensional Arrays

As the name indicates, multi dimensional arrays are arrays that contain more than one dimension, usually two or three but it can have up to 32 dimensions.

A multi array works like a matrix with various levels, take in example a comparison between one, two, and three Dimensions.

One Dimension is your typical array, it looks like a list of elements.

```
Dim 1D(3) as Variant

*1D - Visually*
(0)
(1)
(2)
```

Two Dimensions would look like a Sudoku Grid or an Excel sheet, when initializing the array you would define how many rows and columns the array would have.

```
Dim 2D(3,3) as Variant
'this would result in a 3x3 grid

*2D - Visually*
(0,0) (0,1) (0,2)
(1,0) (1,1) (1,2)
(2,0) (2,1) (2,2)
```

Three Dimensions would start to look like Rubik's Cube, when initializing the array you would define rows and columns and layers/depths the array would have.

```
Dim 3D(3,3,2) as Variant
'this would result in a 3x3x3 grid

*3D - Visually*
      1st layer                2nd layer                3rd layer
        front                    middle                   back
(0,0,0) (0,0,1) (0,0,2) ¦ (1,0,0) (1,0,1) (1,0,2) ¦ (2,0,0) (2,0,1) (2,0,2)
(0,1,0) (0,1,1) (0,1,2) ¦ (1,1,0) (1,1,1) (1,1,2) ¦ (2,1,0) (2,1,1) (2,1,2)
(0,2,0) (0,2,1) (0,2,2) ¦ (1,2,0) (1,2,1) (1,2,2) ¦ (2,2,0) (2,2,1) (2,2,2)
```

Further dimensions could be thought as the multiplication of the 3D, so a 4D(1,3,3,3) would be two side-by-side 3D arrays.

Two-Dimension Array

Creating

The example below will be a compilation of a list of employees, each employee will have a set of information on the list (First Name, Surname, Address, Email, Phone ...), the example will essentially be storing on the array

(员工,信息)中(0,0)是第一个员工的名字。

```
Dim Bosses As Variant
'将bosses设置为Variant，这样我们可以输入任何数据类型

Bosses = [{"Jonh","Snow","President";"Ygritte","Wild","Vice-President"}]
'直接通过填充信息初始化一个二维数组，结果将是一个大小为(1,2)的数组
2x3 = 6个元素

Dim Employees As Variant
'将Employees数组初始化为Variant'
'初始化并重新定义Employee数组，使其成为动态数组而非静态数组，因此被VBA编译器区别对待'

ReDim Employees(100, 5)
声明一个二维数组，可以存储100名员工，每名员工有6个信息元素，但开始时为空

数组大小为101 x 6，包含606个元素

对于 employee = 0 到 UBound(Employees, 1)
对于数组中的每个员工/行，二维数组的UBound将获取数组的最后一个元素

需要两个参数，第一个是你想检查的数组，第二个是维度，这里1 = 员工，2 = 信息

    对于 information_e = 0 到 UBound(Employees, 2)
    对于数组中的每个信息元素/列

Employees(employee, information_e) = InformationNeeded ' InformationNeeded 是填充数组的数据

        遍历整个数组将允许直接将信息赋值到元素坐标中

    下一步
下一步
```

调整大小

像对一维数组进行常规调整大小或ReDim Preserve多维数组会出错，取而代之的是需要将信息转移到一个临时数组中，该数组的大小等于原数组加上要添加的行/列数。在下面的示例中，我们将看到如何初始化一个临时数组，将信息从原数组转移过来，填充剩余的空元素，并用临时数组替换原数组。

```
Dim TempEmp As Variant
'将临时数组初始化为Variant类型
ReDim TempEmp(UBound(Employees, 1) + 1, UBound(Employees, 2))
'ReDim/调整临时数组大小为二维数组，大小为UBound(Employees)+1 = （Employees第一维的最后一个元素）+ 1，

'第二维保持与原数组相同。我们实际上在Employees数组中添加了一行


'转移
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    For info = LBound(Employees, 2) To UBound(Employees, 2)
        '为了将Employees转移到TempEmp，我们遍历两个数组，并用Employees中对应的元素值填充TempEmp

        TempEmp(emp, info) = Employees(emp, info)

    下一步
下一步

填充剩余部分
                        传输后，Temp数组末尾仍有未使用的元素，因为它被扩展了
```


(employee,information) being the (0,0) is the first employee's first name.

```
Dim Bosses As Variant
'set bosses as Variant, so we can input any data type we want

Bosses = [{"Jonh","Snow","President";"Ygritte","Wild","Vice-President"}]
'initialise a 2D array directly by filling it with information, the redult wil be a array(1,2) size
2x3 = 6 elements

Dim Employees As Variant
'initialize your Employees array as variant
'initialize and ReDim the Employee array so it is a dynamic array instead of a static one, hence
treated differently by the VBA Compiler
ReDim Employees(100, 5)
'declaring an 2D array that can store 100 employees with 6 elements of information each, but starts
empty
'the array size is 101 x 6 and contains 606 elements

For employee = 0 To UBound(Employees, 1)
'for each employee/row in the array, UBound for 2D arrays, which will get the last element on the
array
'needs two parameters 1st the array you which to check and 2nd the dimension, in this case 1 =
employee and 2 = information
    For information_e = 0 To UBound(Employees, 2)
    'for each information element/column in the array

        Employees(employee, information_e) = InformationNeeded ' InformationNeeded would be the
data to fill the array
        'iterating the full array will allow for direct attribution of information into the element
coordinates
    Next
Next
```

Resizing

Resizing or `ReDim Preserve` a Multi-Array like the norm for a One-Dimension array would get an error, instead the information needs to be transferred into a Temporary array with the same size as the original plus the number of row/columns to add. In the example below we'll see how to initialize a Temp Array, transfer the information over from the original array, fill the remaining empty elements, and replace the temp array by the original array.

```
Dim TempEmp As Variant
'initialise your temp array as variant
ReDim TempEmp(UBound(Employees, 1) + 1, UBound(Employees, 2))
'ReDim/Resize Temp array as a 2D array with size UBound(Employees)+1 = (last element in Employees 1st
dimension) + 1,
'the 2nd dimension remains the same as the original array. we effectively add 1 row in the Employee
array

'transfer
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    For info = LBound(Employees, 2) To UBound(Employees, 2)
        'to transfer Employees into TempEmp we iterate both arrays and fill TempEmp with the
corresponding element value in Employees
        TempEmp(emp, info) = Employees(emp, info)

    Next
Next

'fill remaining
'after the transfers the Temp array still has unused elements at the end, being that it was increased
```

```
为了填充剩余的元素，从最后一个有值的"行"迭代到数组的最后一行在这种情况下，Temp数组的最后一行将是Employees数组行数加1，因为Employees数组的最后一行已经填充到Temp数组中


For emp = UBound(Employees, 1) + 1 To UBound(TempEmp, 1)
    For info = LBound(TempEmp, 2) To UBound(TempEmp, 2)

TempEmp(emp, info) = InformationNeeded & "NewRow"

    下一步
下一步

清除Employees，将Temp数组赋值给Employees，然后清除Temp数组
Erase Employees
Employees = TempEmp
Erase TempEmp
```

更改元素值

要更改某个元素的值，只需调用该坐标并赋予它一个新值：Employees(0, 0) = "NewValue"

或者通过坐标迭代，使用条件匹配对应参数所需的值：

```
对于 emp = 0 到 UBound(Employees)
    如果 Employees(emp, 0) = "Gloria" 且 Employees(emp, 1) = "Stephan" 则
    '找到值时
Employees(emp, 1) = "已婚，改姓"
        退出循环
        '除非必要，否则不要遍历整个数组
    结束条件判断
Next
```

读取

访问数组中的元素可以通过嵌套循环（遍历每个元素）、循环和坐标（迭代行并直接访问列），或者直接使用坐标访问。

```
'嵌套循环，将遍历所有元素
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    对于 info = LBound(Employees, 2) 到 UBound(Employees, 2)
        Debug.Print Employees(emp, info)
    下一步
下一步

循环并协调，遍历所有行，在每一行中直接访问所有列
对于 emp = LBound(Employees, 1) 到 UBound(Employees, 1)
    Debug.Print Employees(emp, 0)
Debug.Print Employees(emp, 1)
    Debug.Print Employees(emp, 2)
    Debug.Print Employees(emp, 3)
    Debug.Print Employees(emp, 4)
    Debug.Print Employees(emp, 5)
Next

'直接访问带坐标的元素
```



```
'to fill the remaining elements iterate from the last "row" with values to the last row in the array
'in this case the last row in Temp will be the size of the Employees array rows + 1, as the last row
of Employees array is already filled in the TempArray

For emp = UBound(Employees, 1) + 1 To UBound(TempEmp, 1)
    For info = LBound(TempEmp, 2) To UBound(TempEmp, 2)

        TempEmp(emp, info) = InformationNeeded & "NewRow"

    Next
Next

'erase Employees, attribute Temp array to Employees and erase Temp array
Erase Employees
Employees = TempEmp
Erase TempEmp
```

Changing Element Values

To change/alter the values in a certain element can be done by simply calling the coordinate to change and giving it a new value: `Employees(0, 0) = "NewValue"`

Alternatively iterate through the coordinates use conditions to match values corresponding to the parameters needed:

```
For emp = 0 To UBound(Employees)
    If Employees(emp, 0) = "Gloria" And Employees(emp, 1) = "Stephan" Then
    'if value found
        Employees(emp, 1) = "Married, Last Name Change"
        Exit For
        'don't iterate through a full array unless necessary
    End If
Next
```

Reading

Accessing the elements in the array can be done with a Nested Loop (iterating every element), Loop and Coordinate (iterate Rows and accessing columns directly), or accessing directly with both coordinates.

```
'nested loop, will iterate through all elements
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    For info = LBound(Employees, 2) To UBound(Employees, 2)
        Debug.Print Employees(emp, info)
    Next
Next

'loop and coordinate, iteration through all rows and in each row accessing all columns directly
For emp = LBound(Employees, 1) To UBound(Employees, 1)
    Debug.Print Employees(emp, 0)
    Debug.Print Employees(emp, 1)
    Debug.Print Employees(emp, 2)
    Debug.Print Employees(emp, 3)
    Debug.Print Employees(emp, 4)
    Debug.Print Employees(emp, 5)
Next

'directly accessing element with coordinates
```

```
Debug.Print Employees(5, 5)
```

请记住，使用多维数组时，保留一个数组映射总是很方便的，因为它们很容易引起混淆。

三维数组

对于三维数组，我们将使用与二维数组相同的前提，另外还要存储员工和信息，以及他们工作的建筑物。

三维数组将包含员工（可以视为行）、信息（列）和建筑物，可以看作是Excel文档中的不同工作表，它们之间大小相同，但每个工作表的单元格/元素中包含不同的信息。三维数组将包含 ***n*** 个二维数组。

创建

一个三维数组需要3个坐标来初始化Dim 3Darray(2,5,5) As Variant数组的第一个坐标将表示建筑物/工作表的数量（不同的行和列集合），第二个坐标表示行，第三个坐标表示列。上述Dim将生成一个包含108个元素的三维数组（3*6*6），实际上包含3个不同的二维数组集合。

```
Dim ThreeDArray As Variant
'将ThreeDArray数组初始化为Variant类型
ReDim ThreeDArray(1, 50, 5)
'声明一个三维数组，可以存储两组各有51名员工，每名员工有6个信息元素，
但初始为空
'数组大小为2 x 51 x 6，包含612个元素

For building = 0 To UBound(ThreeDArray, 1)
    '遍历数组中的每个建筑物/集合
    For employee = 0 To UBound(ThreeDArray, 2)
    '遍历数组中的每个员工/行
        For information_e = 0 To UBound(ThreeDArray, 3)
        '遍历数组中的每个信息元素/列

ThreeDArray(building, employee, information_e) = InformationNeeded ' InformationNeeded
是用于填充数组的数据
        遍历整个数组将允许直接将信息赋值到元素坐标中

        下一步
    下一步
Next
```

调整大小

调整三维数组大小类似于调整二维数组，首先创建一个与原数组大小相同的临时数组，在需要增加的坐标上加一，第一个坐标增加数组中的集合数量，第二和第三个坐标增加每个集合中的行数或列数。

下面的示例将每个集合中的行数增加一，并用新信息填充新添加的元素。

```
Dim TempEmp As Variant
'将临时数组初始化为Variant类型
ReDim TempEmp(UBound(ThreeDArray, 1), UBound(ThreeDArray, 2) + 1, UBound(ThreeDArray, 3))
```



```
Debug.Print Employees(5, 5)
```

***Remember**, it's always handy to keep an array map when using Multidimensional arrays, they can easily become confusion.*

Three-Dimension Array

For the 3D array, we'll use the same premise as the 2D array, with the addition of not only storing the Employee and Information but as well Building they work in.

The 3D array will have the Employees (can be thought of as Rows), the Information (Columns), and Building that can be thought of as different sheets on an excel document, they have the same size between them, but every sheets has a different set of information in its cells/elements. The 3D array will contain ***n*** number of 2D arrays.

Creating

A 3D array needs 3 coordinates to be initialized `Dim 3Darray(2,5,5) As Variant` the first coordinate on the array will be the number of Building/Sheets (different sets of rows and columns), second coordinate will define Rows and third Columns. The `Dim` above will result in a 3D array with 108 elements (3*6*6), effectively having 3 different sets of 2D arrays.

```
Dim ThreeDArray As Variant
'initialise your ThreeDArray array as variant
ReDim ThreeDArray(1, 50, 5)
'declaring an 3D array that can store two sets of 51 employees with 6 elements of information each,
but starts empty
'the array size is 2 x 51 x 6 and contains 612 elements

For building = 0 To UBound(ThreeDArray, 1)
    'for each building/set in the array
    For employee = 0 To UBound(ThreeDArray, 2)
    'for each employee/row in the array
        For information_e = 0 To UBound(ThreeDArray, 3)
        'for each information element/column in the array

            ThreeDArray(building, employee, information_e) = InformationNeeded ' InformationNeeded
would be the data to fill the array
        'iterating the full array will allow for direct attribution of information into the element
coordinates
        Next
    Next
Next
```

Resizing

Resizing a 3D array is similar to resizing a 2D, first create a Temporary array with the same size of the original adding one in the coordinate of the parameter to increase, the first coordinate will increase the number of sets in the array, the second and third coordinates will increase the number of Rows or Columns in each set.

The example below increases the number of Rows in each set by one, and fills those recently added elements with new information.

```
Dim TempEmp As Variant
'initialise your temp array as variant
ReDim TempEmp(UBound(ThreeDArray, 1), UBound(ThreeDArray, 2) + 1, UBound(ThreeDArray, 3))
```

```
将临时数组重新定义/调整大小为一个三维数组，大小为 UBound(ThreeDArray)+1 = （员工二维数组的最后一个元素）+ 1，

另一维度保持与原数组相同。我们实际上在三维数组的每个"集合"中增加了一行


'转移
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        For info = LBound(ThreeDArray, 3) To UBound(ThreeDArray, 3)
            通过遍历三维数组中的所有集合，将 ThreeDArray 转移到 TempEmp 中，并用每行每个集合中的对应元素值
填充 TempEmp
            TempEmp(building, emp, info) = ThreeDArray(building, emp, info)

        下一步
    下一步
Next

填充剩余部分
要填充剩余元素，我们需要从最后一个有值的"行"迭代到数组中每个集合的最后一行，记住第一个空元素是原数组的 UBound() 加 1

For building = LBound(TempEmp, 1) To UBound(TempEmp, 1)
    For emp = UBound(ThreeDArray, 2) + 1 To UBound(TempEmp, 2)
        For info = LBound(TempEmp, 3) To UBound(TempEmp, 3)

TempEmp(building, emp, info) = InformationNeeded & "NewRow"

        下一步
    下一步
Next

清除Employees，将Temp数组赋值给Employees，然后清除Temp数组
Erase ThreeDArray
ThreeDArray = TempEmp
Erase TempEmp
```

更改元素值和读取

读取和更改三维数组中的元素可以类似于二维数组的方式进行，只需调整循环和坐标中的额外层级即可。

```
Do
' 使用 Do ... While 进行提前退出
    For building = 0 To UBound(ThreeDArray, 1)
        For emp = 0 To UBound(ThreeDArray, 2)
            If ThreeDArray(building, emp, 0) = "Gloria" And ThreeDArray(building, emp, 1) =
"Stephan" Then
            如果找到值
ThreeDArray(建筑, 员工, 1) = "已婚，改姓"
                退出循环
                不要在非必要情况下遍历整个数组
            End If
        Next
    Next
循环条件为假

嵌套循环，将遍历所有元素
对于 建筑 = LBound(ThreeDArray, 1) 到 UBound(ThreeDArray, 1)
    对于 员工 = LBound(ThreeDArray, 2) 到 UBound(ThreeDArray, 2)
        对于 信息 = LBound(ThreeDArray, 3) 到 UBound(ThreeDArray, 3)
          Debug.Print ThreeDArray(建筑, 员工, 信息)
        下一个
```



```
'ReDim/Resize Temp array as a 3D array with size UBound(ThreeDArray)+1 = (last element in Employees
2nd dimension) + 1,
'the other dimension remains the same as the original array. we effectively add 1 row in the for each
set of the 3D array

'transfer
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        For info = LBound(ThreeDArray, 3) To UBound(ThreeDArray, 3)
            'to transfer ThreeDArray into TempEmp by iterating all sets in the 3D array and fill
TempEmp with the corresponding element value in each set of each row
            TempEmp(building, emp, info) = ThreeDArray(building, emp, info)

        Next
    Next
Next

'fill remaining
'to fill the remaining elements we need to iterate from the last "row" with values to the last row in
the array in each set, remember that the first empty element is the original array Ubound() plus 1
For building = LBound(TempEmp, 1) To UBound(TempEmp, 1)
    For emp = UBound(ThreeDArray, 2) + 1 To UBound(TempEmp, 2)
        For info = LBound(TempEmp, 3) To UBound(TempEmp, 3)

            TempEmp(building, emp, info) = InformationNeeded & "NewRow"

        Next
    Next
Next

'erase Employees, attribute Temp array to Employees and erase Temp array
Erase ThreeDArray
ThreeDArray = TempEmp
Erase TempEmp
```

Changing Element Values and Reading

Reading and changing the elements on the 3D array can be done similarly to the way we do the 2D array, just adjust for the extra level in the loops and coordinates.

```
Do
' using Do ... While for early exit
    For building = 0 To UBound(ThreeDArray, 1)
        For emp = 0 To UBound(ThreeDArray, 2)
            If ThreeDArray(building, emp, 0) = "Gloria" And ThreeDArray(building, emp, 1) =
"Stephan" Then
            'if value found
                ThreeDArray(building, emp, 1) = "Married, Last Name Change"
                Exit Do
                'don't iterate through all the array unless necessary
            End If
        Next
    Next
Loop While False

'nested loop, will iterate through all elements
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        For info = LBound(ThreeDArray, 3) To UBound(ThreeDArray, 3)
            Debug.Print ThreeDArray(building, emp, info)
        Next
```

```
    下一步
下一步

循环和坐标，将遍历所有行集并请求该行加上我们选择的列值

For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    对于 员工 = LBound(ThreeDArray, 2) 到 UBound(ThreeDArray, 2)
        Debug.Print ThreeDArray(建筑, 员工, 0)
        Debug.Print ThreeDArray(建筑, 员工, 1)
        Debug.Print ThreeDArray(建筑, 员工, 2)
        Debug.Print ThreeDArray(建筑, 员工, 3)
        Debug.Print ThreeDArray(建筑, 员工, 4)
        Debug.Print ThreeDArray(建筑, 员工, 5)
    下一步
下一步

'直接访问带坐标的元素
Debug.Print Employees(0, 5, 5)
```

第18.2节：动态数组（数组调整大小和动态处理）

动态数组

动态增加和减少数组中的变量是一个巨大优势，尤其是在处理的信息变量数量不固定时。

动态添加值

您可以使用`ReDim`语句简单地调整数组大小，这将调整数组的大小，但如果您希望保留数组中已存储的信息，则需要使用`Preserve`部分。

在下面的示例中，我们创建一个数组，并在每次迭代中增加一个变量，同时保留数组中已有的值。

```
Dim Dynamic_array As Variant
' 首先将 Dynamic_array 设置为 Variant 类型

For n = 1 To 100

    If IsEmpty(Dynamic_array) Then
        'IsEmpty() 将检查我们是否需要向数组添加第一个值或后续值

        ReDim Dynamic_array(0)
        'ReDim Dynamic_array(0) 将数组大小调整为仅包含一个变量
       Dynamic_array(0) = n

    Else
        ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) + 1)
        '上面这一行代码将数组大小从变量0调整到 UBound() = 最后一个变量，再加上一个，有效地将数组大小增加了一个
       Dynamic_array(UBound(Dynamic_array)) = n
        将属性值赋给动态数组的最后一个变量
    End If

Next
```

动态移除值


```
    Next
Next

'loop and coordinate, will iterate through all set of rows and ask for the row plus the value we choose for the columns
For building = LBound(ThreeDArray, 1) To UBound(ThreeDArray, 1)
    For emp = LBound(ThreeDArray, 2) To UBound(ThreeDArray, 2)
        Debug.Print ThreeDArray(building, emp, 0)
        Debug.Print ThreeDArray(building, emp, 1)
        Debug.Print ThreeDArray(building, emp, 2)
        Debug.Print ThreeDArray(building, emp, 3)
        Debug.Print ThreeDArray(building, emp, 4)
        Debug.Print ThreeDArray(building, emp, 5)
    Next
Next

'directly accessing element with coordinates
Debug.Print Employees(0, 5, 5)
```

Section 18.2: Dynamic Arrays (Array Resizing and Dynamic Handling)

Dynamic Arrays

Adding and reducing variables on an array dynamically is a huge advantage for when the information you are treating does not have a set number of variables.

Adding Values Dynamically

You can simply resize the Array with the `ReDim` Statement, this will resize the array but to if you which to retain the information already stored in the array you'll need the part `Preserve`.

In the example below we create an array and increase it by one more variable in each iteration while preserving the values already in the array.

```
Dim Dynamic_array As Variant
' first we set Dynamic_array as variant

For n = 1 To 100

    If IsEmpty(Dynamic_array) Then
        'isempty() will check if we need to add the first value to the array or subsequent ones

        ReDim Dynamic_array(0)
        'ReDim Dynamic_array(0) will resize the array to one variable only
        Dynamic_array(0) = n

    Else
        ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) + 1)
        'in the line above we resize the array from variable 0 to the UBound() = last variable, plus one effectivelly increeasing the size of the array by one
        Dynamic_array(UBound(Dynamic_array)) = n
        'attribute a value to the last variable of Dynamic_array
    End If

Next
```

Removing Values Dynamically

我们可以利用相同的逻辑来减少数组。在示例中，值“last”将从数组中移除。

```
Dim Dynamic_array As Variant
Dynamic_array = Array("first", "middle", "last")

ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) - 1)
' 在保留的同时调整大小，去掉最后一个值
```

重置数组并动态重用

我们也可以重新利用创建的数组，以避免内存中存在过多数组，从而导致运行时间变慢。这对于各种大小的数组都很有用。你可以使用的一个代码片段是将数组通过ReDim调整回(0)，给数组赋值一个变量，然后可以自由地再次增加数组大小。

在下面的代码片段中，我构造了一个包含1到40的数组，清空数组，然后重新填充40到100的值，所有操作均为动态完成。

```
Dim Dynamic_array As Variant

For n = 1 To 100

    If IsEmpty(Dynamic_array) Then
        ReDim Dynamic_array(0)
        Dynamic_array(0) = n

    ElseIf Dynamic_array(0) = "" Then
        如果第一个变量为空（= ""），则赋值为 n
       Dynamic_array(0) = n
    Else
        重新定义 保留 Dynamic_array(0 到 UBound(Dynamic_array) + 1)
       Dynamic_array(UBound(Dynamic_array)) = n
    End If
    如果 n = 40 则
        ReDim Dynamic_array(0)
        '将数组大小调整回单变量，不保留原内容，
        '使数组的第一个值为空
    End If

Next
```

第18.3节：锯齿数组（数组的数组）

锯齿数组不是多维数组

数组的数组（锯齿数组）与多维数组在视觉上不同多维数组看起来像矩阵（矩形），其维度（内部数组）中元素数量固定，而锯齿数组则像年度日历，内部数组的元素数量不同，类似不同月份的天数。

虽然锯齿数组由于嵌套层级较多且类型安全性较差，使用起来相当混乱和棘手，但它们非常灵活，允许你轻松操作不同类型的数据，且不需要包含未使用或空的元素。

创建锯齿数组

在下面的示例中，我们将初始化一个锯齿数组，包含两个数组，一个用于姓名，另一个用于

We can utilise the same logic to to decrease the the array. In the example the value "last" will be removed from the array.

```
Dim Dynamic_array As Variant
Dynamic_array = Array("first", "middle", "last")

ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) - 1)
' Resize Preserve while dropping the last value
```

Resetting an Array and Reusing Dynamically

We can as well re-utilise the arrays we create as not to have many on memory, which would make the run time slower. This is useful for arrays of various sizes. One snippet you could use to re-utilise the array is to `ReDim` the array back to (`0`), attribute one variable to to the array and freely increase the array again.

In the snippet below I construct an array with the values 1 to 40, empty the array, and refill the array with values 40 to 100, all this done dynamically.

```
Dim Dynamic_array As Variant

For n = 1 To 100

    If IsEmpty(Dynamic_array) Then
        ReDim Dynamic_array(0)
        Dynamic_array(0) = n

    ElseIf Dynamic_array(0) = "" Then
        'if first variant is empty ( = "") then give it the value of n
        Dynamic_array(0) = n
    Else
        ReDim Preserve Dynamic_array(0 To UBound(Dynamic_array) + 1)
        Dynamic_array(UBound(Dynamic_array)) = n
    End If
    If n = 40 Then
        ReDim Dynamic_array(0)
        'Resizing the array back to one variable without Preserving,
        'leaving the first value of the array empty
    End If

Next
```

Section 18.3: Jagged Arrays (Arrays of Arrays)

Jagged Arrays NOT Multidimensional Arrays

Arrays of Arrays(Jagged Arrays) are not the same as Multidimensional Arrays if you think about them visually Multidimensional Arrays would look like Matrices (Rectangular) with defined number of elements on their dimensions(inside arrays), while Jagged array would be like a yearly calendar with the inside arrays having different number of elements, like days in on different months.

Although Jagged Arrays are quite messy and tricky to use due to their nested levels and don't have much type safety, but they are very flexible, allow you to manipulate different types of data quite easily, and don't need to contain unused or empty elements.

Creating a Jagged Array

In the below example we will initialise a jagged array containing two arrays one for Names and another for

数字，然后访问每个元素中的一个

```
Dim OuterArray() As Variant
Dim Names() As Variant
Dim Numbers() As Variant
'数组声明为Variant类型，以便我们可以为其元素分配任何数据类型的属性

Names = Array("Person1", "Person2", "Person3")
Numbers = Array("001", "002", "003")

OuterArray = Array(Names, Numbers)
'直接给OuterArray赋值一个包含Names和Numbers数组的数组

Debug.Print OuterArray(0)(1)
Debug.Print OuterArray(1)(1)
'通过给出元素的坐标访问锯齿状数组中的元素
```

动态创建和读取锯齿状数组

我们也可以在构建数组的方法上更加动态，假设我们有一个客户数据表在Excel中，我们想构建一个数组来输出客户详情。

```
姓名 -   电话   -  邮箱  - 客户编号
Person1 - 153486231 - 1@STACK - 001
Person2 - 153486242 - 2@STACK - 002
Person3 - 153486253 - 3@STACK - 003
Person4 - 153486264 - 4@STACK - 004
Person5 - 153486275 - 5@STACK - 005
```

我们将动态构建一个标题数组和一个客户数组，标题数组将包含列标题，客户数组将包含每个客户/行的信息，形式为数组。

```
Dim Headers As Variant
' 带有客户数据表顶部部分的标题数组
    For c = 1 To 4
        If IsEmpty(Headers) Then
            ReDim Headers(0)
Headers(0) = Cells(1, c).Value
        Else
            ReDim Preserve Headers(0 To UBound(Headers) + 1)
            Headers(UBound(Headers)) = Cells(1, c).Value
        End If
    Next

Dim Customers As Variant
“客户数组将包含客户值的数组
Dim Customer_Values As Variant”
“Customer_Values 将是包含客户元素（姓名-电话-邮箱-客户编号）的数组”

    “For r = 2 To 6”
    “‘遍历客户/行
        For c = 1 To 4”
        “‘遍历值/列”

            “‘构建包含客户值的数组”
            “If IsEmpty(Customer_Values) Then
                ReDim Customer_Values(0)”
“             Customer_Values(0) = Cells(r, c).Value
            ElseIf Customer_Values(0) = "" Then”
```


Numbers, and then accessing one element of each

```
Dim OuterArray() As Variant
Dim Names() As Variant
Dim Numbers() As Variant
'arrays are declared variant so we can access attribute any data type to its elements

Names = Array("Person1", "Person2", "Person3")
Numbers = Array("001", "002", "003")

OuterArray = Array(Names, Numbers)
'Directly giving OuterArray an array containing both Names and Numbers arrays inside

Debug.Print OuterArray(0)(1)
Debug.Print OuterArray(1)(1)
'accessing elements inside the jagged by giving the coordenades of the element
```

Dynamically Creating and Reading Jagged Arrays

We can as well be more dynamic in our approx to construct the arrays, imagine that we have a customer data sheet in excel and we want to construct an array to output the customer details.

```
   Name -   Phone   -  Email  - Customer Number
Person1 - 153486231 - 1@STACK - 001
Person2 - 153486242 - 2@STACK - 002
Person3 - 153486253 - 3@STACK - 003
Person4 - 153486264 - 4@STACK - 004
Person5 - 153486275 - 5@STACK - 005
```

We will Dynamically construct an Header array and a Customers array, the Header will contain the column titles and the Customers array will contain the information of each customer/row as arrays.

```
Dim Headers As Variant
' headers array with the top section of the customer data sheet
    For c = 1 To 4
        If IsEmpty(Headers) Then
            ReDim Headers(0)
            Headers(0) = Cells(1, c).Value
        Else
            ReDim Preserve Headers(0 To UBound(Headers) + 1)
            Headers(UBound(Headers)) = Cells(1, c).Value
        End If
    Next

Dim Customers As Variant
'Customers array will contain arrays of customer values
Dim Customer_Values As Variant
'Customer_Values will be an array of the customer in its elements (Name-Phone-Email-CustNum)

    For r = 2 To 6
    'iterate through the customers/rows
        For c = 1 To 4
        'iterate through the values/columns

            'build array containing customer values
            If IsEmpty(Customer_Values) Then
                ReDim Customer_Values(0)
                Customer_Values(0) = Cells(r, c).Value
            ElseIf Customer_Values(0) = "" Then
```

```
“                Customer_Values(0) = Cells(r, c).Value”
            Else
                ReDim Preserve Customer_Values(0 To UBound(Customer_Values) + 1)
                Customer_Values(UBound(Customer_Values)) = Cells(r, c).Value
            End If
        Next

        '将 customer_values 数组添加到 Customers 数组
        If IsEmpty(Customers) Then
            ReDim Customers(0)
Customers(0) = Customer_Values
        Else
            ReDim Preserve Customers(0 To UBound(Customers) + 1)
            Customers(UBound(Customers)) = Customer_Values
        End If

        '重置 Customer_Values 以便在需要时重建新数组
        ReDim Customer_Values(0)
    Next

    Dim Main_Array(0 To 1) As Variant
    '主数组将包含标题和客户

Main_Array(0) = Headers
    Main_Array(1) = Customers
```

为了更好地理解如何动态构建一维数组，请查看数组文档中的动态数组（数组重设和动态处理）。

上述代码片段的结果是一个锯齿数组，其中包含两个数组，一个数组有4个元素，缩进层级为2，另一个数组本身是另一个锯齿数组，包含5个数组，每个数组有4个元素，缩进层级为3，结构如下所示：

```
Main_Array(0) - 头部 - 数组("姓名","电话","邮箱","客户编号")
          (1) - 客户(0) - 数组("Person1",153486231,"1@STACK",001)
                客户(1) - 数组("Person2",153486242,"2@STACK",002)
                   ...
                客户(4) - 数组("Person5",153486275,"5@STACK",005)
```

要访问信息，您必须记住您创建的锯齿数组的结构，在上述
示例中，您可以看到主数组包含一个头部数组和一个数组的数组（客户），因此
访问元素的方式不同。

现在我们将读取主数组的信息，并以信息类型:的形式打印出每个客户的信息
信息.

```
对于 n = 0 到 UBound(Main_Array(1))
    ' n 用于遍历 Main_Array(1) 中的第一个到最后一个数组

    对于 j = 0 到 UBound(Main_Array(1)(n))
        ' j 将遍历 Main_Array(1) 中每个数组的第一个到最后一个元素

Debug.Print Main_Array(0)(j) & ": " & Main_Array(1)(n)(j)
        '打印 Main_Array(0)(j) 即头部，以及 Main_Array(1)(n)(j) 即客户数组中的元素

        我们可以用 j 作为头部调用头部数组，因为头部数组的结构与客户数组相同

    下一步
下一步
```



```
                Customer_Values(0) = Cells(r, c).Value
            Else
                ReDim Preserve Customer_Values(0 To UBound(Customer_Values) + 1)
                Customer_Values(UBound(Customer_Values)) = Cells(r, c).Value
            End If
        Next

        'add customer_values array to Customers Array
        If IsEmpty(Customers) Then
            ReDim Customers(0)
            Customers(0) = Customer_Values
        Else
            ReDim Preserve Customers(0 To UBound(Customers) + 1)
            Customers(UBound(Customers)) = Customer_Values
        End If

        'reset Custumer_Values to rebuild a new array if needed
        ReDim Customer_Values(0)
    Next

    Dim Main_Array(0 To 1) As Variant
    'main array will contain both the Headers and Customers

    Main_Array(0) = Headers
    Main_Array(1) = Customers
```

To better understand the way to Dynamically construct a one dimensional array please check Dynamic Arrays (Array Resizing and Dynamic Handling) on the Arrays documentation.

The Result of the above snippet is an Jagged Array with two arrays one of those arrays with 4 elements, 2 indention levels, and the other being itself another Jagged Array containing 5 arrays of 4 elements each and 3 indention levels, see below the structure:

```
Main_Array(0) - Headers - Array("Name","Phone","Email","Customer Number")
          (1) - Customers(0) - Array("Person1",153486231,"1@STACK",001)
                Customers(1) - Array("Person2",153486242,"2@STACK",002)
                ...
                Customers(4) - Array("Person5",153486275,"5@STACK",005)
```

To access the information you'll have to bear in mind the structure of the Jagged Array you create, in the above example you can see that the `Main Array` contains an Array of `Headers` and an Array of Arrays (`Customers`) hence with different ways of accessing the elements.

Now we'll read the information of the `Main Array` and print out each of the Customers information as `Info Type: Info`.

```
For n = 0 To UBound(Main_Array(1))
    'n to iterate from first to last array in Main_Array(1)

    For j = 0 To UBound(Main_Array(1)(n))
        'j will iterate from first to last element in each array of Main_Array(1)

        Debug.Print Main_Array(0)(j) & ": " & Main_Array(1)(n)(j)
        'print Main_Array(0)(j) which is the header and Main_Array(0)(n)(j) which is the element in
the customer array
        'we can call the header with j as the header array has the same structure as the customer
array
    Next
Next
```

请记住跟踪你的锯齿数组的结构，在上面的例子中，访问客户的名称是通过访问Main_Array -> Customers -> CustomerNumber -> Name完成的，这有三级，要返回"Person4"，你需要知道 Customers 在 Main_Array 中的位置，然后是 Customers 锯齿数组中第四个客户的位置，最后是你需要的元素的位置，在本例中是Main_Array(1)(3)(0)，即Main_Array(Customers)(CustomerNumber)(Name)。

第18.4节：在 VBA 中声明数组

声明数组与声明变量非常相似，只是你需要在名称后面声明数组的维度：

```
Dim myArray(9) As String '声明一个最多包含10个字符串的数组
```

默认情况下，VBA 中的数组是从零开始索引的，因此括号内的数字不是指数组的大小，而是最后一个元素的索引

访问元素

访问数组的元素是通过使用数组名称，后跟元素的索引，放在括号内完成的：

```
myArray(0) = "第一个元素"
myArray(5) = "第六个元素"
myArray(9) = "最后一个元素"
```

数组索引

您可以通过将此行放在模块顶部来更改数组索引方式：

```
Option Base 1
```

通过这行代码，模块中声明的所有数组将从1开始索引。

特定索引

你也可以使用To关键字以及下限和上限（=

```
Dim mySecondArray(1 To 12) As String '从1到12索引的12个字符串数组
Dim myThirdArray(13 To 24) As String '从13到24索引的12个字符串数组
```

动态声明

当你在声明数组之前不知道数组的大小时，可以使用动态声明，以及
ReDim 关键字：

```
Dim myDynamicArray() As Strings '创建一个元素数量未知的字符串数组
ReDim myDynamicArray(5) '这会将数组重置为6个元素
```

注意，使用ReDim关键字会清除数组中之前的所有内容。为防止这种情况，可以在ReDim后使用Preserve关键字：

```
Dim myDynamicArray(5) As String
myDynamicArray(0) = "我想保留的内容"
```


REMEMBER to keep track of the structure of your Jagged Array, in the example above to access the Name of a customer is by accessing `Main_Array -> Customers -> CustomerNumber -> Name` which is three levels, to return `"Person4"` you'll need the location of Customers in the Main_Array, then the Location of customer four on the Customers Jagged array and lastly the location of the element you need, in this case `Main_Array(1)(3)(0)` which is `Main_Array(Customers)(CustomerNumber)(Name)`.

Section 18.4: Declaring an Array in VBA

Declaring an array is very similar to declaring a variable, except you need to declare the dimension of the Array right after its name:

```
Dim myArray(9) As String 'Declaring an array that will contain up to 10 strings
```

By default, Arrays in VBA are **indexed from ZERO**, thus, the number inside the parenthesis doesn't refer to the size of the array, but rather to **the index of the last element**

Accessing Elements

Accessing an element of the Array is done by using the name of the Array, followed by the index of the element, inside parenthesis:

```
myArray(0) = "first element"
myArray(5) = "sixth element"
myArray(9) = "last element"
```

Array Indexing

You can change Arrays indexing by placing this line at the top of a module:

```
Option Base 1
```

With this line, all Arrays declared in the module will be **indexed from ONE**.

Specific Index

You can also declare each Array with its own index by using the `To` keyword, and the lower and upper bound (= index):

```
Dim mySecondArray(1 To 12) As String 'Array of 12 strings indexed from 1 to 12
Dim myThirdArray(13 To 24) As String 'Array of 12 strings indexed from 13 to 24
```

Dynamic Declaration

When you do not know the size of your Array prior to its declaration, you can use the dynamic declaration, and the `ReDim` keyword:

```
Dim myDynamicArray() As Strings 'Creates an Array of an unknown number of strings
ReDim myDynamicArray(5) 'This resets the array to 6 elements
```

Note that using the `ReDim` keyword will wipe out any previous content of your Array. To prevent this, you can use the `Preserve` keyword after `ReDim`:

```
Dim myDynamicArray(5) As String
myDynamicArray(0) = "Something I want to keep"
```

```
ReDim Preserve myDynamicArray(8) '将大小扩展到最多9个字符串
Debug.Print myDynamicArray(0) '仍然打印该元素
```

第18.5节：使用Split从字符串创建数组

Split 函数

返回一个基于零的一维数组，包含指定数量的子字符串。

语法

Split(expression [, delimiter [, limit [, compare]]])

部分	说明
表达式	必需。包含子字符串和分隔符的字符串表达式。如果 expression是零长度字符串（""或vbNullString），Split返回一个不包含任何元素和数据的空数组。在这种情况下，返回的数组的LBound为0，UBound为-1。
分隔符	可选。用于标识子字符串边界的字符串字符。如果省略，默认分隔符为空格字符（" "）。如果 delimiter是零长度字符串，则返回一个包含整个 expression字符串的单元素数组。
限制	可选。要返回的子字符串数量；-1 表示返回所有子字符串。
比较	可选。数值，指示在评估子字符串时使用的比较类型。详见设置部分的取值说明。

设置

compare 参数可以取以下值：

常量	值	说明
说明	-1	使用 Option Compare 语句的设置执行比较。
vbBinaryCompare	0	执行二进制比较。
vbTextCompare	1	执行文本比较。
vbDatabaseCompare	2	仅限 Microsoft Access。基于数据库中的信息执行比较。

示例

本例演示了 Split 的工作原理，通过展示多种样式。注释将显示每种不同 Split 选项的结果集。最后演示如何遍历返回的字符串数组
。

```
子程序 Test

    声明 textArray() 为字符串

textArray = Split("Tech on the Net")
    '结果: {"Tech", "on", "the", "Net"}

textArray = Split("172.23.56.4", ".")
    '结果: {"172", "23", "56", "4"}

textArray = Split("A;B;C;D", ";")
    结果：{"A", "B", "C", "D"}

textArray = Split("A;B;C;D", ";", 1)
    结果：{"A;B;C;D"}
```



```
ReDim Preserve myDynamicArray(8) 'Expand the size to up to 9 strings
Debug.Print myDynamicArray(0) ' still prints the element
```

Section 18.5: Use of Split to create an array from a string

Split Function

returns a zero-based, one dimensional array containing a specified number of substrings.

Syntax

Split(expression [, **delimiter** [, **limit** [, **compare**]]])

Part	Description
expression	Required. String expression containing substrings and delimiters. If *expression* is a zero-length string("" or vbNullString), **Split** returns an empty array containing no elements and no data. In this case, the returned array will have a LBound of 0 and a UBound of -1.
delimiter	Optional. String character used to identify substring limits. If omitted, the space character (" ") is assumed to be the delimiter. If ***delimiter*** is a zero-length string, a single-element array containing the entire ***expression*** string is returned.
limit	Optional. Number of substrings to be returned; -1 indicates that all substrings are returned.
compare	Optional. Numeric value indicating the kind of comparison to use when evaluating substrings. See Settings section for values.

Settings

The ***compare*** argument can have the following values:

Constant	Value	Description
Description	-1	Performs a comparison using the setting of the **Option Compare** statement.
vbBinaryCompare	0	Performs a binary comparison.
vbTextCompare	1	Performs a textual comparison.
vbDatabaseCompare	2	Microsoft Access only. Performs a comparison based on information in your database.

Example

In this example it is demonstrated how Split works by showing several styles. The comments will show the result set for each of the different performed Split options. Finally it is demonstrated how to loop over the returned string array.

```
Sub Test

    Dim textArray() as String

    textArray = Split("Tech on the Net")
    'Result: {"Tech", "on", "the", "Net"}

    textArray = Split("172.23.56.4", ".")
    'Result: {"172", "23", "56", "4"}

    textArray = Split("A;B;C;D", ";")
    'Result: {"A", "B", "C", "D"}

    textArray = Split("A;B;C;D", ";", 1)
    'Result: {"A;B;C;D"}
```

```
textArray = Split("A;B;C;D", ";", 2)
    结果：{"A", "B;C;D"}

textArray = Split("A;B;C;D", ";", 3)
    结果：{"A", "B", "C;D"}

textArray = Split("A;B;C;D", ";", 4)
    结果：{"A", "B", "C", "D"}

    你可以遍历创建的数组
    Dim counter As Long

    For counter = LBound(textArray) To UBound(textArray)
       Debug.Print textArray(counter)
    Next
End Sub
```

第18.6节：遍历数组元素

For...Next

使用迭代变量作为索引号是遍历数组元素最快的方法：

```
Dim items As Variant
items = Array(0, 1, 2, 3)

Dim index As Integer
For index = LBound(items) To UBound(items)
    '假设值可以隐式转换为字符串：
    Debug.Print items(index)
Next
```

嵌套循环可用于遍历多维数组：

```
Dim items(0 To 1, 0 To 1) As Integer
items(0, 0) = 0
items(0, 1) = 1
items(1, 0) = 2
items(1, 1) = 3

Dim outer As Integer
Dim inner As Integer
For outer = LBound(items, 1) To UBound(items, 1)
    For inner = LBound(items, 2) To UBound(items, 2)
        '假设值可以隐式转换为字符串：
        Debug.Print items(outer, inner)
    下一步
下一步
```

For Each...Next

如果性能不重要，也可以使用For Each...Next循环来遍历数组：

```
Dim items As Variant
items = Array(0, 1, 2, 3)

Dim item As Variant '必须是Variant类型
For Each item In items
    '假设值可以隐式转换为字符串：
```



```
    textArray = Split("A;B;C;D", ";", 2)
    'Result: {"A", "B;C;D"}

    textArray = Split("A;B;C;D", ";", 3)
    'Result: {"A", "B", "C;D"}

    textArray = Split("A;B;C;D", ";", 4)
    'Result: {"A", "B", "C", "D"}

    'You can iterate over the created array
    Dim counter As Long

    For counter = LBound(textArray) To UBound(textArray)
        Debug.Print textArray(counter)
    Next
End Sub
```

Section 18.6: Iterating elements of an array

For...Next

Using the iterator variable as the index number is the fastest way to iterate the elements of an array:

```
Dim items As Variant
items = Array(0, 1, 2, 3)

Dim index As Integer
For index = LBound(items) To UBound(items)
    'assumes value can be implicitly converted to a String:
    Debug.Print items(index)
Next
```

Nested loops can be used to iterate multi-dimensional arrays:

```
Dim items(0 To 1, 0 To 1) As Integer
items(0, 0) = 0
items(0, 1) = 1
items(1, 0) = 2
items(1, 1) = 3

Dim outer As Integer
Dim inner As Integer
For outer = LBound(items, 1) To UBound(items, 1)
    For inner = LBound(items, 2) To UBound(items, 2)
        'assumes value can be implicitly converted to a String:
        Debug.Print items(outer, inner)
    Next
Next
```

For Each...Next

A `For Each...Next` loop can also be used to iterate arrays, if performance doesn't matter:

```
Dim items As Variant
items = Array(0, 1, 2, 3)

Dim item As Variant 'must be variant
For Each item In items
    'assumes value can be implicitly converted to a String:
```

```
    Debug.Print item
Next
```

一个**For Each**循环将从外到内遍历所有维度（与元素在
内存中的排列顺序相同），因此不需要嵌套循环：

```
Dim items(0 To 1, 0 To 1) As Integer
items(0, 0) = 0
items(1, 0) = 1
items(0, 1) = 2
items(1, 1) = 3

Dim item As Variant '必须是 Variant 类型
For Each item In items
    '假设值可以隐式转换为字符串：
    Debug.Print item
Next
```

注意，For Each循环最好用于遍历Collection对象，如果性能很重要的话。

以上所有4个代码片段产生相同的输出：

```
0
1
2
3
```



```
    Debug.Print item
Next
```

A **For Each** loop will iterate all dimensions from outer to inner (the same order as the elements are laid out in memory), so there is no need for nested loops:

```
Dim items(0 To 1, 0 To 1) As Integer
items(0, 0) = 0
items(1, 0) = 1
items(0, 1) = 2
items(1, 1) = 3

Dim item As Variant 'must be Variant
For Each item In items
    'assumes value can be implicitly converted to a String:
    Debug.Print item
Next
```

Note that **For Each** loops are best used to iterate `Collection` objects, if performance matters.

All 4 snippets above produce the same output:

```
0
1
2
3
```

第19章：复制、返回和传递数组

第19.1节：向过程传递数组

数组可以通过在数组变量名后加上()来传递给过程。

```
Function countElements(ByRef arr() As Double) As Long
    countElements = UBound(arr) - LBound(arr) + 1
End Function
```

数组必须通过引用传递。如果没有指定传递机制，例如myFunction(arr())，那么VBA默认会假设ByRef，但良好的编码习惯是明确指出。尝试通过值传递数组，例如
myFunction(ByVal arr()) 会导致“数组参数必须是ByRef”的编译错误（如果VBE选项中未勾选自动语法检查，则会出现“语法错误”的编译错误）。

通过引用传递意味着对数组的任何更改都会保留在调用过程里。

```
Sub testArrayPassing()
    Dim source(0 To 1) As Long
    source(0) = 3
source(1) = 1

Debug.Print doubleAndSum(source)  '输出 8
    Debug.Print source(0); source(1)  '输出 6 2
End Sub

Function doubleAndSum(ByRef arr() As Long)
    arr(0) = arr(0) * 2
arr(1) = arr(1) * 2
    doubleAndSum = arr(0) + arr(1)
End Function
```

如果你想避免改变原始数组，那么要小心编写函数，确保它不修改任何元素。

```
函数 doubleAndSum(ByRef arr() As Long)
   doubleAndSum = arr(0) * 2 + arr(1) * 2
结束函数
```

或者创建数组的工作副本并使用该副本进行操作。

```
函数 doubleAndSum(ByRef arr() As Long)
    Dim copyOfArr() As Long
    copyOfArr = arr

copyOfArr(0) = copyOfArr(0) * 2
    copyOfArr(1) = copyOfArr(1) * 2

    doubleAndSum = copyOfArr(0) + copyOfArr(1)
End Function
```

第19.2节：复制数组

您可以使用=运算符将VBA数组复制到相同类型的数组中。数组必须是相同类型的

Chapter 19: Copying, returning and passing arrays

Section 19.1: Passing Arrays to Proceedures

Arrays can be passed to proceedures by putting `()` after the name of the array variable.

```
Function countElements(ByRef arr() As Double) As Long
    countElements = UBound(arr) - LBound(arr) + 1
End Function
```

Arrays *must* be passed by reference. If no passing mechanism is specified, e.g. `myFunction(arr())`, then VBA will assume **`ByRef`** by default, however it is good coding practice to make it explicit. Trying to pass an array by value, e.g. `myFunction(ByVal arr())` will result in an "Array argument must be ByRef" compilation error (or a "Syntax error" compilation error if `Auto Syntax Check` is not checked in the VBE options).

Passing by reference means that any changes to the array will be preserved in the calling proceedure.

```
Sub testArrayPassing()
    Dim source(0 To 1) As Long
    source(0) = 3
    source(1) = 1

    Debug.Print doubleAndSum(source)  ' outputs 8
    Debug.Print source(0); source(1)  ' outputs 6 2
End Sub

Function doubleAndSum(ByRef arr() As Long)
    arr(0) = arr(0) * 2
    arr(1) = arr(1) * 2
    doubleAndSum = arr(0) + arr(1)
End Function
```

If you want to avoid changing the original array then be careful to write the function so that it doesn't change any elements.

```
Function doubleAndSum(ByRef arr() As Long)
    doubleAndSum = arr(0) * 2 + arr(1) * 2
End Function
```

Alternatively create a working copy of the array and work with the copy.

```
Function doubleAndSum(ByRef arr() As Long)
    Dim copyOfArr() As Long
    copyOfArr = arr

    copyOfArr(0) = copyOfArr(0) * 2
    copyOfArr(1) = copyOfArr(1) * 2

    doubleAndSum = copyOfArr(0) + copyOfArr(1)
End Function
```

Section 19.2: Copying Arrays

You can copy a VBA array into an array of the same type using the = operator. The arrays must be of the same type

否则代码将抛出“无法赋值给数组”的编译错误。

```
Dim source(0 到 2) As Long
Dim destinationLong() As Long
Dim destinationDouble() As Double

destinationLong = source     '将 source 的内容复制到 destinationLong
destinationDouble = source   '无法编译
```

源数组可以是固定的或动态的，但目标数组必须是动态的。尝试复制到固定数组会导致“无法赋值给数组”的编译错误。接收数组中任何预先存在的数据都会丢失，其边界和维度会更改为与源数组相同。

```
Dim source() As Long
ReDim source(0 To 2)

Dim fixed(0 To 2) As Long
Dim dynamic() As Long

fixed = source   '无法编译
dynamic = source '可以编译

Dim dynamic2() As Long
ReDim dynamic2(0 to 6, 3 to 99)

dynamic2 = source ' dynamic2 现在的维度是 (0 到 2)
```

复制完成后，这两个数组在内存中是分开的，即这两个变量不是指向相同底层数据的引用，因此对一个数组所做的更改不会出现在另一个数组中。

```
Dim source(0 To 2) As Long
Dim destination() As Long

source(0) = 3
source(1) = 1
source(2) = 4

destination = source
destination(0) = 2

Debug.Print source(0); source(1); source(2)                '输出: 3 1 4
Debug.Print destination(0); destination(1); destination(2) '输出: 2 1 4
```

复制对象数组

对于对象数组，复制的是对这些对象的引用，而不是对象本身。如果对一个数组中的对象进行更改，另一个数组中对应的对象也会显示为已更改——它们都引用同一个对象。然而，在一个数组中将某个元素设置为不同的对象，不会使另一个数组中的对应元素也变成该对象。

```
Dim source(0 To 2) As Range
Dim destination() As Range

Set source(0) = Range("A1"): source(0).Value = 3
Set source(1) = Range("A2"): source(1).Value = 1
Set source(2) = Range("A3"): source(2).Value = 4

destination = source
```


otherwise the code will throw a "Can't assign to array" compilation error.

```
Dim source(0 to 2) As Long
Dim destinationLong() As Long
Dim destinationDouble() As Double

destinationLong = source     ' copies contents of source into destinationLong
destinationDouble = source   ' does not compile
```

The source array can be fixed or dynamic, but the destination array must be dynamic. Trying to copy to a fixed array will throw a "Can't assign to array" compilation error. Any preexisting data in the receiving array is lost and its bounds and dimenions are changed to the same as the source array.

```
Dim source() As Long
ReDim source(0 To 2)

Dim fixed(0 To 2) As Long
Dim dynamic() As Long

fixed = source   ' does not compile
dynamic = source ' does compile

Dim dynamic2() As Long
ReDim dynamic2(0 to 6, 3 to 99)

dynamic2 = source ' dynamic2 now has dimension (0 to 2)
```

Once the copy is made the two arrays are separate in memory, i.e. the two variables are not references to same underlying data, so changes made to one array do not appear in the other.

```
Dim source(0 To 2) As Long
Dim destination() As Long

source(0) = 3
source(1) = 1
source(2) = 4

destination = source
destination(0) = 2

Debug.Print source(0); source(1); source(2)                ' outputs: 3 1 4
Debug.Print destination(0); destination(1); destination(2) ' outputs: 2 1 4
```

Copying Arrays of Objects

With arrays of objects the *references* to those objects are copied, not the objects themselves. If a change is made to an object in one array it will also appear to be changed in the other array - they are both referencing the same object. However, setting an element to a different object in one array won't set it to that object the other array.

```
Dim source(0 To 2) As Range
Dim destination() As Range

Set source(0) = Range("A1"): source(0).Value = 3
Set source(1) = Range("A2"): source(1).Value = 1
Set source(2) = Range("A3"): source(2).Value = 4

destination = source
```

```
Set destination(0) = Range("A4")    ' 仅更改了 destination 中的引用，source 未变

destination(0).Value = 2            ' 仅影响 destination 中的对象
destination(1).Value = 5            ' 影响 source 和 destination 中的对象

Debug.Print source(0); source(1); source(2)                    ' 输出 3 5 4
Debug.Print destination(0); destination(1); destination(2)   ' 输出 2 5 4
```

包含数组的变量

你也可以将数组复制到变量（Variant）中，或从变量中复制数组。当从变量复制时，变量必须包含与接收数组类型相同的数组，否则会抛出"类型不匹配"运行时错误。

```
Dim var As Variant
Dim source(0 To 2) As Range
Dim destination() As Range

var = source
目标 = 变量

变量 = 5
destination = var  ' 抛出运行时错误
```

第19.3节：从函数返回数组

普通模块中的函数（但不是类模块）可以通过在数据类型后加上()来返回数组。

```
Function arrayOfPiDigits() As Long()
    Dim outputArray(0 To 2) As Long

    outputArray(0) = 3
    outputArray(1) = 1
    outputArray(2) = 4

    arrayOfPiDigits = outputArray
End Function
```

函数的结果可以赋给同类型的动态数组或Variant变量。元素也可以通过使用第二组括号直接访问，但这会每次调用函数，因此如果计划多次使用，最好将结果存储在新数组中。

```
Sub arrayExample()

    Dim destination() As Long
    Dim var As Variant

destination = arrayOfPiDigits()
    var = arrayOfPiDigits

Debug.Print destination(0)           ' 输出3
    Debug.Print var(1)                    ' 输出1
    Debug.Print arrayOfPiDigits()(2)    ' 输出4

End Sub
```

注意，返回的实际上是函数内部数组的副本，而不是引用。因此，如果函数返回的是静态数组的内容，调用过程无法更改其数据。


```
Set destination(0) = Range("A4")    'reference changed in destination but not source

destination(0).Value = 2            'affects an object only in destination
destination(1).Value = 5            'affects an object in both source and destination

Debug.Print source(0); source(1); source(2)                    ' outputs 3 5 4
Debug.Print destination(0); destination(1); destination(2)   ' outputs 2 5 4
```

Variants Containing an Array

You can also copy an array into and from a variant variable. When copying from a variant, it must contain an array of the same type as the receiving array otherwise it will throw a "Type mismatch" runtime error.

```
Dim var As Variant
Dim source(0 To 2) As Range
Dim destination() As Range

var = source
destination = var

var = 5
destination = var  ' throws runtime error
```

Section 19.3: Returning Arrays from Functions

A function in a normal module (but not a Class module) can return an array by putting () after the data type.

```
Function arrayOfPiDigits() As Long()
    Dim outputArray(0 To 2) As Long

    outputArray(0) = 3
    outputArray(1) = 1
    outputArray(2) = 4

    arrayOfPiDigits = outputArray
End Function
```

The result of the function can then be put into a dynamic array of the same type or a variant. The elements can also be accessed directly by using a second set of brackets, however this will call the function each time, so its best to store the results in a new array if you plan to use them more than once

```
Sub arrayExample()

    Dim destination() As Long
    Dim var As Variant

    destination = arrayOfPiDigits()
    var = arrayOfPiDigits

    Debug.Print destination(0)           ' outputs 3
    Debug.Print var(1)                    ' outputs 1
    Debug.Print arrayOfPiDigits()(2)    ' outputs 4

End Sub
```

Note that what is returned is actually a copy of the array inside the function, not a reference. So if the function returns the contents of a Static array its data can't be changed by the calling procedure.

通过输出参数输出数组

通常情况下，过程的参数应作为输入，并通过返回值输出，这是良好的编码习惯。
然而，VBA 的限制有时使得过程必须通过ByRef参数输出数据。

输出到固定数组

```
Sub threePiDigits(ByRef destination() As Long)
    destination(0) = 3
destination(1) = 1
    destination(2) = 4
End Sub

Sub printPiDigits()
    Dim digits(0 To 2) As Long

    threePiDigits digits
Debug.Print digits(0); digits(1); digits(2) ' 输出 3 1 4
End Sub
```

从类方法输出数组

输出参数也可以用于从类模块中的方法/过程输出数组

```
' 类模块 '数学常数 (MathConstants)
子程序 threePiDigits(ByRef destination() As Long)
    重新定义 destination(0 到 2)

destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

' 标准代码模块
Sub printPiDigits()
    定义 digits() 为 Long
    定义 mathConsts 为 新建 MathConstants

    mathConsts.threePiDigits digits
   调试打印 digits(0); digits(1); digits(2) ' 输出 3 1 4
结束子程序
```


Outputting an Array via an output argument

It is normally good coding practice for a procedure's arguments to be inputs and to output via the return value. However, the limitations of VBA sometimes make it necessary for a procedure to output data via a `ByRef` argument.

Outputting to a fixed array

```
Sub threePiDigits(ByRef destination() As Long)
    destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

Sub printPiDigits()
    Dim digits(0 To 2) As Long

    threePiDigits digits
    Debug.Print digits(0); digits(1); digits(2) ' outputs 3 1 4
End Sub
```

Outputting an Array from a Class method

An output argument can also be used to output an array from a method/proceedure in a Class module

```
' Class Module 'MathConstants'
Sub threePiDigits(ByRef destination() As Long)
    ReDim destination(0 To 2)

    destination(0) = 3
    destination(1) = 1
    destination(2) = 4
End Sub

' Standard Code Module
Sub printPiDigits()
    Dim digits() As Long
    Dim mathConsts As New MathConstants

    mathConsts.threePiDigits digits
    Debug.Print digits(0); digits(1); digits(2) ' outputs 3 1 4
End Sub
```

第20章：集合

第20.1节：获取集合的项目数量

可以通过调用Collection的.Count函数来获取其中项目的数量：

语法：

```
.Count()
```

示例用法：

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
        .Add "Four"
    结束于

Debug.Print foo.Count    '打印 4
End Sub
```

第20.2节：确定键或项目是否存在于Collection

键

与Scripting.Dictionary不同，Collection没有方法来判断给定的键是否存在，也没有方法来检索Collection中存在的键。确定键是否存在的唯一方法是使用错误处理程序：

```
Public Function KeyExistsInCollection(ByVal key As String, _
                                      ByRef container As Collection) As Boolean
    With Err
        如果 容器 为 Nothing 则 引发 91
        出错时继续下一条
        Dim temp As Variant
       temp = container.Item(key
        出错时转到0

        如果 .Number = 0 则
KeyExistsInCollection = True
        否则如果 .Number <> 5 则
            .引发 .Number
        End If
    End With
End Function
```

项目

确定一个项目是否包含在Collection中的唯一方法是遍历Collection直到找到该项目。注意，由于Collection可以包含基本类型或对象，因此在比较过程中需要一些额外处理以避免运行时错误：

Chapter 20: Collections

Section 20.1: Getting the Item Count of a Collection

The number of items in a `Collection` can be obtained by calling its `.Count` function:

Syntax:

```
.Count()
```

Sample Usage:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
        .Add "Four"
    End With

    Debug.Print foo.Count    'Prints 4
End Sub
```

Section 20.2: Determining if a Key or Item Exists in a Collection

Keys

Unlike a Scripting.Dictionary, a `Collection` does not have a method for determining if a given key exists *or* a way to retrieve keys that are present in the `Collection`. The only method to determine if a key is present is to use the error handler:

```
Public Function KeyExistsInCollection(ByVal key As String, _
                                      ByRef container As Collection) As Boolean
    With Err
        If container Is Nothing Then .Raise 91
        On Error Resume Next
        Dim temp As Variant
        temp = container.Item(key)
        On Error GoTo 0

        If .Number = 0 Then
            KeyExistsInCollection = True
        ElseIf .Number <> 5 Then
            .Raise .Number
        End If
    End With
End Function
```

Items

The only way to determine if an item is contained in a `Collection` is to iterate over the `Collection` until the item is located. Note that because a `Collection` can contain either primitives or objects, some extra handling is needed to avoid run-time errors during the comparisons:

```
公共函数 ItemExistsInCollection(ByRef target As Variant, _
                                ByRef container As Collection) As Boolean
    Dim candidate As Variant
    Dim found As Boolean

    对于每个候选项在容器中
        选择情况为真
            情况候选项是对象且目标是对象
              found = 候选项是目标
            情况候选项是对象，目标是对象
                found = 假
            其他情况
found = (候选项 = 目标)
        结束选择
        如果 found 则
ItemExistsInCollection = 真
            退出函数
        End If
    Next
End Function
```

第20.3节：向集合中添加项目

通过调用Collection的.Add方法添加项目：

语法：

```
.Add(项目, [键], [之前, 之后])
```

参数	说明
项目	要存储在Collection中的项目。它可以是变量可以赋值的任何值，包括基本类型、数组、对象和Nothing。
键	可选。一个String，作为从Collection中检索项目的唯一标识符。如果指定的键已存在于Collection中，将导致运行时错误457："此键已与该集合的元素关联"。
之前	可选。一个现有的键（String值）或索引（数值）用于在Collection中插入项目之前。如果给定了该值，则after参数必须为空，否则将导致运行时错误5："无效的过程调用或参数"。如果传入的String键在Collection中不存在，将导致运行时错误5："无效的过程调用或参数"。如果传入的数值索引在Collection中不存在，将导致运行时错误9："下标越界"。
之后	可选。一个现有的键（String值）或索引（数值）用于在Collection中插入项目之后。如果给定了该值，则before参数必须为空。引发的错误与before参数相同。

注意：

- 键名不区分大小写。添加"Bar"、"Foo"和添加"Baz"、"foo"将导致键名冲突。
- 如果未提供可选的before或after参数，项目将被添加到集合中的最后一个项目之后集合。
- 通过指定before或after参数进行的插入将更改现有成员的数字索引，以匹配它们的新位置。这意味着在使用数字索引的循环中进行插入时应当小心。

示例用法：

```
公共子程序示例()
```



```
Public Function ItemExistsInCollection(ByRef target As Variant, _
                                       ByRef container As Collection) As Boolean
    Dim candidate As Variant
    Dim found As Boolean

    For Each candidate In container
        Select Case True
            Case IsObject(candidate) And IsObject(target)
                found = candidate Is target
            Case IsObject(candidate), IsObject(target)
                found = False
            Case Else
                found = (candidate = target)
        End Select
        If found Then
            ItemExistsInCollection = True
            Exit Function
        End If
    Next
End Function
```

Section 20.3: Adding Items to a Collection

Items are added to a `Collection` by calling its `.Add` method:

Syntax:

```
.Add(item, [key], [before, after])
```

Parameter	Description
item	The item to store in the `Collection`. This can be essentially any value that a variable can be assigned to, including primitive types, arrays, objects, and `Nothing`.
key	Optional. A `String` that serves as a unique identifier for retrieving items from the `Collection`. If the specified key already exists in the `Collection`, it will result in a Run-time error 457: "This key is already associated with an element of this collection".
before	Optional. An existing key (`String` value) *or* index (numeric value) to insert the item before in the `Collection`. If a value is given, the *after* parameter ***must*** be empty or a Run-time error 5: "Invalid procedure call or argument" will result. If a `String` key is passed that does not exist in the `Collection`, a Run-time error 5: "Invalid procedure call or argument" will result. If a numeric index is passed that is does not exist in the `Collection`, a Run-time error 9: "Subscript out of range" will result.
after	Optional. An existing key (`String` value) *or* index (numeric value) to insert the item after in the `Collection`. If a value is given, the *before* parameter ***must*** be empty. Errors raised are identical to the *before* parameter.

Notes:

- Keys are ***not*** case-sensitive. `.Add "Bar", "Foo"` and `.Add "Baz", "foo"` will result in a key collision.
- If neither of the optional *before* or *after* parameters are given, the item will be added after the last item in the `Collection`.
- Insertions made by specifying a *before* or *after* parameter will alter the numeric indexes of existing members to match thier new position. This means that care should be taken when making insertions in loops using numeric indexes.

Sample Usage:

```
Public Sub Example()
```

```
    声明foo为新集合

    使用foo
        .Add "One"              '无键名。此项目只能通过索引检索。
        .Add "Two", "Second"    '指定了键名。可以通过键名或索引检索。
        .Add "Three", , 1       '插入到集合的开头。
        .Add "Four", , , 1      '插入到索引2处。
    结束于

    Dim member As Variant
    对于每个成员在foo中
Debug.Print 成员      '打印 "Three, Four, One, Two"
    下一条
End Sub
```

第20.4节：从集合中移除项目

通过调用集合的.Remove方法来移除项目：

语法：

```
.Remove(索引)
```

参数	说明
索引	要从集合中移除的项目。如果传入的值是数值类型或带有数值子类型的Variant，则将其解释为数值索引。如果传入的值是String或包含字符串的Variant，则将其解释为键。如果传入的字符串键在集合中不存在， 将导致运行时错误5："无效的过程调用或参数"。如果传入的数值索引在集合中不存在， 将导致运行时错误9："下标越界"。

注意：

- 从集合中移除项目会改变集合中该项目之后所有项目的数值索引。使用数值索引并移除项目的For循环应倒序运行（Step -1），以防止下标异常和跳过项目。
- 通常不应在For Each循环内部从集合中移除项目，因为这可能导致不可预测的结果。

示例用法：

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two", "Second"
        .Add "Three"
        .Add "Four"
    结束于

foo.Remove 1              '移除第一个项目。
foo.Remove "Second"       '移除键为 "Second" 的项目。
foo.Remove foo.Count      '移除最后一个项目。

    Dim member As Variant
    For Each member In foo
       Debug.Print member  '打印 "Three"

End Sub
```



```
    Dim foo As New Collection

    With foo
        .Add "One"              'No key. This item can only be retrieved by index.
        .Add "Two", "Second"    'Key given. Can be retrieved by key or index.
        .Add "Three", , 1       'Inserted at the start of the collection.
        .Add "Four", , , 1      'Inserted at index 2.
    End With

    Dim member As Variant
    For Each member In foo
        Debug.Print member      'Prints "Three, Four, One, Two"
    Next
End Sub
```

Section 20.4: Removing Items From a Collection

Items are removed from a `Collection` by calling its `.Remove` method:

Syntax:

```
.Remove(index)
```

Parameter	Description
index	The item to remove from the `Collection`. If the value passed is a numeric type or `Variant` with a numeric sub-type, it will be interpreted as a numeric index. If the value passed is a `String` or `Variant` containing a string, it will be interpreted as the a key. If a String key is passed that does not exist in the `Collection`, a Run-time error 5: "Invalid procedure call or argument" will result. If a numeric index is passed that is does not exist in the `Collection`, a Run-time error 9: "Subscript out of range" will result.

Notes:

- Removing an item from a `Collection` will change the numeric indexes of all the items after it in the `Collection`. `For` loops that use numeric indexes and remove items should run *backwards* (`Step -1`) to prevent subscript exceptions and skipped items.
- Items should generally ***not*** be removed from a `Collection` from inside of a `For Each` loop as it can give unpredictable results.

Sample Usage:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two", "Second"
        .Add "Three"
        .Add "Four"
    End With

    foo.Remove 1              'Removes the first item.
    foo.Remove "Second"       'Removes the item with key "Second".
    foo.Remove foo.Count      'Removes the last item.

    Dim member As Variant
    For Each member In foo
        Debug.Print member    'Prints "Three"
    Next
End Sub
```

第20.5节：从集合中检索项目

可以通过调用.Item函数从Collection中检索项目。

语法：

```
.Item(index)
```

参数	说明
索引	要从Collection中检索的项目。如果传入的值是数值类型或带有数值子类型的Variant，则将其解释为数值索引。如果传入的值是String或包含字符串的Variant，则将其解释为键。如果传入的字符串键在将导致运行时错误5：“无效的过程调用或参数”。如果传入的数值索引在集合中不存在，将导致运行时错误9：“下标越界”。

注意：

- .Item是Collection的默认成员。这允许语法上的灵活性，如下面的示例用法所示。
- 数值索引是从1开始的。
- 键不区分大小写。.Item("Foo")和.Item("foo")指的是相同的键。
- index参数不会从String隐式转换为数字，反之亦然。完全有可能.Item(1)和.Item("1")指向Collection中的不同项目。

示例用法（索引）：

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
        .Add "Four"
    结束于

    Dim index As Long
    For index = 1 To foo.Count
Debug.Print foo.Item(index) '打印 One, Two, Three, Four
    Next
End Sub
```

示例用法（键）：

```
Public Sub Example()
    Dim keys() As String
    keys = Split("Foo,Bar,Baz", ",")
    Dim values() As String
values = Split("One,Two,Three", ",")

    Dim foo As New Collection
    Dim index As Long
    For index = LBound(values) To UBound(values)
       foo.Add values(index), keys(index)
    Next

Debug.Print foo.Item("Bar") '打印 "Two"
End Sub
```


Section 20.5: Retrieving Items From a Collection

Items can be retrieved from a `Collection` by calling the `.Item` function.

Syntax:

```
.Item(index)
```

Parameter	Description
index	The item to retrieve from the `Collection`. If the value passed is a numeric type or `Variant` with a numeric sub-type, it will be interpreted as a numeric index. If the value passed is a `String` or `Variant` containing a string, it will be interpreted as the a key. If a String key is passed that does not exist in the `Collection`, a Run-time error 5: "Invalid procedure call or argument" will result. If a numeric index is passed that is does not exist in the `Collection`, a Run-time error 9: "Subscript out of range" will result.

Notes:

- `.Item` is the default member of `Collection`. This allows flexibility in syntax as demonstrated in the sample usage below.
- Numeric indexes are 1-based.
- Keys are ***not*** case-sensitive. `.Item("Foo")` and `.Item("foo")` refer to the same key.
- The *index* parameter is ***not*** implicitly cast to a number from a `String` or visa-versa. It is entirely possible that `.Item(1)` and `.Item("1")` refer to different items of the `Collection`.

Sample Usage (Indexes):

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
        .Add "Four"
    End With

    Dim index As Long
    For index = 1 To foo.Count
        Debug.Print foo.Item(index) 'Prints One, Two, Three, Four
    Next
End Sub
```

Sample Usage (Keys):

```
Public Sub Example()
    Dim keys() As String
    keys = Split("Foo,Bar,Baz", ",")
    Dim values() As String
    values = Split("One,Two,Three", ",")

    Dim foo As New Collection
    Dim index As Long
    For index = LBound(values) To UBound(values)
        foo.Add values(index), keys(index)
    Next

    Debug.Print foo.Item("Bar") 'Prints "Two"
End Sub
```

示例用法（替代语法）：

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One", "Foo"
        .Add "Two", "Bar"
        .Add "Three", "Baz"
    结束于

    '以下所有行均打印 "Two"
Debug.Print foo.Item("Bar")      '显式调用语法。
Debug.Print foo("Bar")           '默认成员调用语法。
Debug.Print foo!Bar              'Bang 语法。
End Sub
```

请注意，允许使用感叹号（!）语法，因为.Item是默认成员，并且可以接受单个String参数。
这种语法的实用性值得怀疑。

第20.6节：清除集合中的所有项

清除Collection中所有项的最简单方法是直接用一个新的Collection替换它，然后让旧的超出作用域：

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
    End With

Debug.Print foo.Count    '打印 3
    Set foo = New Collection
Debug.Print foo.Count    '打印 0
End Sub
```

但是，如果有多个对该Collection的引用，这种方法只会给被赋值的变量一个空的
Collection。

```
Public Sub Example()
    Dim foo As New Collection
    Dim bar As Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
    结束于

    设置 bar = foo
    设置 foo = 新建 集合

调试打印 foo.Count    '打印 0
    调试打印 bar.Count    '打印 3
结束子程序
```


Sample Usage (Alternate Syntax):

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One", "Foo"
        .Add "Two", "Bar"
        .Add "Three", "Baz"
    End With

    'All lines below print "Two"
    Debug.Print foo.Item("Bar")      'Explicit call syntax.
    Debug.Print foo("Bar")           'Default member call syntax.
    Debug.Print foo!Bar              'Bang syntax.
End Sub
```

Note that bang (`!`) syntax is allowed because `.Item` is the default member and can take a single `String` argument. The utility of this syntax is questionable.

Section 20.6: Clearing All Items From a Collection

The easiest way to clear all of the items from a `Collection` is to simply replace it with a new `Collection` and let the old one go out of scope:

```
Public Sub Example()
    Dim foo As New Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
    End With

    Debug.Print foo.Count    'Prints 3
    Set foo = New Collection
    Debug.Print foo.Count    'Prints 0
End Sub
```

However, if there are multiple references to the `Collection` held, this method will only give you an empty `Collection` *for the variable that is assigned*.

```
Public Sub Example()
    Dim foo As New Collection
    Dim bar As Collection

    With foo
        .Add "One"
        .Add "Two"
        .Add "Three"
    End With

    Set bar = foo
    Set foo = New Collection

    Debug.Print foo.Count    'Prints 0
    Debug.Print bar.Count    'Prints 3
End Sub
```

在这种情况下，清空内容的最简单方法是通过循环遍历集合中的项目数量，
并反复移除最小的项目：

```
公共子程序 清空集合(ByRef 容器 As 集合)
    定义 索引 As 长整型
    对于 索引 = 1 到 容器.Count
       容器.移除 1
    Next
End Sub
```

In this case, the easiest way to clear the contents is by looping through the number of items in the `Collection` and repeatedly remove the lowest item:

```
Public Sub ClearCollection(ByRef container As Collection)
    Dim index As Long
    For index = 1 To container.Count
        container.Remove 1
    Next
End Sub
```

第21章：运算符

第21.1节：连接运算符

VBA 支持两种不同的连接运算符，+ 和 &，当与 String 类型一起使用时，两者执行完全相同的功能——右侧的 String 会被追加到左侧的 String 末尾。

如果&运算符用于除String以外的变量类型，则在连接之前会隐式转换为String。

请注意，+连接运算符是+加法运算符的重载。+的行为由操作数的变量类型和运算符类型的优先级决定。如果两个操作数都被定义为String类型或
带有String子类型的Variant，则它们会被连接：

```
公共子程序示例()
    Dim left As String
    Dim right As String

left = "5"
right = "5"

Debug.Print left + right    '打印 "55"
End Sub
```

如果任一操作数是数值类型，另一侧是可以强制转换为数字的String，则数学运算符的类型优先级会使该运算符被视为加法运算符，数值将被相加：

```
公共子程序示例()
    Dim left As Variant
    Dim right As String

left = 5
right = "5"

Debug.Print left + right    '打印 10
End Sub
```

这种行为可能导致细微且难以调试的错误——尤其是在使用Variant类型时，因此通常应仅使用&运算符进行连接。

第21.2节：比较运算符

标记	名称	说明
=	等于	如果左操作数和右操作数相等，则返回True。注意，这是一种赋值运算符的重载。
<>	不等于	如果左操作数和右操作数不相等，则返回True。
>	大于	如果左操作数大于右操作数，则返回True。
<	小于	如果左操作数小于右操作数，则返回True。
>=	大于或等于	如果左操作数大于或等于右操作数，则返回True
<=	小于或等于	如果左操作数小于或等于右操作数，则返回True。

Chapter 21: Operators

Section 21.1: Concatenation Operators

VBA supports 2 different concatenation operators, `+` and `&` and both perform the exact same function when used with `String` types - the right-hand `String` is appended to the end of the left-hand `String`.

If the `&` operator is used with a variable type other than a `String`, it is implicitly cast to a `String` before being concatenated.

Note that the `+` concatenation operator is an overload of the `+` addition operator. The behavior of `+` is determined by the variable types of the operands and precedence of operator types. If both operands are typed as a `String` or `Variant` with a sub-type of `String`, they are concatenated:

```
Public Sub Example()
    Dim left As String
    Dim right As String

    left = "5"
    right = "5"

    Debug.Print left + right    'Prints "55"
End Sub
```

If *either* side is a numeric type and the other side is a `String` that can be coerced into a number, the type precedence of mathematical operators causes the operator to be treated as the addition operator and the numeric values are added:

```
Public Sub Example()
    Dim left As Variant
    Dim right As String

    left = 5
    right = "5"

    Debug.Print left + right    'Prints 10
End Sub
```

This behavior can lead to subtle, hard to debug errors - especially if `Variant` types are being used, so only the `&` operator should typically be used for concatenation.

Section 21.2: Comparison Operators

Token	Name	Description
=	Equal to	Returns `True` if the left-hand and right-hand operands are equal. Note that this is an overload of the assignment operator.
<>	Not equal to	Returns `True` if the left-hand and right-hand operands are not equal.
>	Greater than	Returns `True` if the left-hand operand is greater than the right-hand operand.
<	Less than	Returns `True` if the left-hand operand is less than the right-hand operand.
>=	Greater than or equal	Returns `True` if the if the left-hand operand is greater than or equal to the right-hand operand.
<=	Less than or equal	Returns `True` if the if the left-hand operand is less than or equal to the right-hand operand.

是	引用相等	如果左侧对象引用与右侧对象引用是同一个实例，则返回True。它也可以用于任一侧为Nothing（空对象引用）的情况。注意：Is运算符会尝试将两个操作数强制转换为 Object后再进行比较。如果任一侧是基本类型或不包含对象的Variant（非对象子类型或vtEmpty），比较将导致运行时错误424 - “需要对象”。如果任一操作数属于同一对象的不同接口，比较将返回True。如果需要同时测试实例和接口的相等性，请使用 ObjPtr(left) = ObjPtr(right)代替。

备注

VBA语法允许“链式”比较运算符，但通常应避免使用这些结构。
比较总是从左到右对两个操作数一次进行，每次比较的结果是一个布尔值。例如，表达式...

```
a = 2: b = 1: c = 0
expr = a > b > c
```

...在某些情况下可以理解为测试b是否在 a和 c之间。在VBA中，计算过程如下：

```
a = 2: b = 1: c = 0
expr = a > b > c
expr = (2 > 1) > 0
expr = True > 0
expr = -1 > 0 ' CInt(True) = -1
expr = False
```

除Is之外的任何比较运算符，如果操作数是Object对象，将对该Object的默认成员的返回值进行比较。如果对象没有默认成员，比较将导致运行时错误438 - “对象不支持此属性或方法”。

如果Object未初始化，比较将导致运行时错误91 - “对象变量或With块变量未设置”。

如果使用字面量Nothing与除Is之外的任何比较运算符，将导致编译错误 - “对象的无效使用”。

如果Object的默认成员是另一个Object，VBA将不断调用每个连续返回值的默认成员，直到返回基本类型或引发错误。例如，假设SomeClass有一个默认成员Value，该成员是ChildClass的实例，且ChildClass有一个默认成员ChildValue。比较操作...

```
Set x = New SomeClass
Debug.Print x > 42
```

...将被计算为：

```
Set x = New SomeClass
Debug.Print x.Value.ChildValue > 42
```

如果任一操作数是数值类型，而另一个操作数是字符串或字符串子类型的Variant，则将执行数值比较。在这种情况下，如果字符串无法转换为数字，则比较将导致运行时错误13 - “类型不匹配”。

如果两个操作数都是字符串或字符串子类型的Variant，则将根据

`Is`	Reference equity	Returns **`True`** if the left-hand object reference is the same instance as the right-hand object reference. It can also be used with **`Nothing`** (the null object reference) on either side. **Note:** The `Is` operator will attempt to coerce both operands into an `Object` before performing the comparison. If either side is a primitive type *or* a `Variant` that does not contain an object (either a non-object subtype or `vtEmpty`), the comparison will result in a Run-time error 424 - "Object required". If either operand belongs to a different *interface* of the same object, the comparison will return **`True`**. If you need to test for equity of both the instance *and* the interface, use `ObjPtr(left) = ObjPtr(right)` instead.

Notes

The VBA syntax allows for "chains" of comparison operators, but these constructs should generally be avoided. Comparisons are always performed from left to right on only 2 operands at a time, and each comparison results in a `Boolean`. For example, the expression...

```
a = 2: b = 1: c = 0
expr = a > b > c
```

...may be read in some contexts as a test of whether `b` is between `a` and `c`. In VBA, this evaluates as follows:

```
a = 2: b = 1: c = 0
expr = a > b > c
expr = (2 > 1) > 0
expr = True > 0
expr = -1 > 0 'CInt(True) = -1
expr = False
```

Any comparison operator other than `Is` used with an `Object` as an operand will be performed on the return value of the `Object`'s default member. If the object does not have a default member, the comparison will result in a Run-time error 438 - "Object doesn't support his property or method".

If the `Object` is unintitialized, the comparison will result in a Run-time error 91 - "Object variable or With block variable not set".

If the literal **`Nothing`** is used with any comparison operator other than `Is`, it will result in a Compile error - "Invalid use of object".

If the default member of the `Object` is *another* `Object`, VBA will continually call the default member of each successive return value until a primitive type is returned or an error is raised. For example, assume `SomeClass` has a default member of `Value`, which is an instance of `ChildClass` with a default member of `ChildValue`. The comparison...

```
Set x = New SomeClass
Debug.Print x > 42
```

...will be evaluated as:

```
Set x = New SomeClass
Debug.Print x.Value.ChildValue > 42
```

If either operand is a numeric type and the *other* operand is a `String` or `Variant` of subtype `String`, a numeric comparison will be performed. In this case, if the `String` cannot be cast to a number, a Run-time error 13 - "Type mismatch" will result from the comparison.

If ***both*** operands are a `String` or a `Variant` of subtype `String`, a string comparison will be performed based on the

代码模块的Option Compare设置执行字符串比较。这些比较是逐字符进行的。
注意，包含数字的字符串的字符表示与数字值的比较不相同：

```
公共子程序示例()
    Dim left As Variant
    Dim right As Variant

left = "42"
right = "5"
Debug.Print left > right              '打印 False
    Debug.Print Val(left) > Val(right)     '打印 True
End Sub
```

因此，确保在对字符串或Variant变量执行数值不等比较之前，将其转换为数字。

如果一个操作数是日期类型，当另一个操作数是数值类型或可以转换为数值类型时，将对其底层的Double值执行数值比较。

如果另一个操作数是字符串或当前区域设置下可转换为日期的字符串子类型的Variant，则该字符串将被转换为日期。如果无法在当前区域设置下转换为日期，则比较将导致运行时错误13 - “类型不匹配”。

在比较Double或Single值与布尔值时应当小心。与其他数值类型不同，由于VBA在涉及浮点数的比较时会将数据类型提升为Double，非零值不能被假定为True：

```
公共子程序示例()
    Dim Test As Double

Test = 42        Debug.Print CBool(Test)             '打印 True。
    'True 被提升为 Double 类型 - Test 未被转换为 Boolean 类型
Debug.Print Test = True             '打印 False

    '使用显式转换：
Debug.Print CBool(Test) = True       '打印 True
    Debug.Print CDbl(-1) = CDbl(True)    '打印 True
End Sub
```

第21.3节：按位\逻辑运算符

VBA 中所有的逻辑运算符都可以看作是同名按位运算符的“重载”。
从技术上讲，它们总是被视为按位运算符。VBA 中所有的比较运算符返回一个布尔值，该布尔值要么所有位都未设置（False），要么所有位都已设置（True）。但它会将任何位被设置的值视为True。这意味着将表达式的按位结果转换为Boolean（参见比较运算符）的结果，总是与将其视为逻辑表达式的结果相同。

使用这些运算符之一赋值表达式的结果将得到按位结果。请注意，在下面的真值表中，0等同于False，1等同于True。

与

如果表达式两边的值都计算为True，则返回True。

左操作数 右操作数 结果

Option Compare setting of the code module. These comparisons are performed on a character by character basis. Note that the *character representation* of a `String` containing a number is ***not*** the same as a comparison of the numeric values:

```
Public Sub Example()
    Dim left As Variant
    Dim right As Variant

    left = "42"
    right = "5"
    Debug.Print left > right              'Prints False
    Debug.Print Val(left) > Val(right)    'Prints True
End Sub
```

For this reason, make sure that `String` or `Variant` variables are cast to numbers before performing numeric inequity comparisons on them.

If one operand is a `Date`, a numeric comparison on the underlying Double value will be performed if the other operand is numeric or can be cast to a numeric type.

If the other operand is a `String` or a `Variant` of subtype `String` that can be cast to a `Date` using the current locale, the `String` will be cast to a `Date`. If it cannot be cast to a `Date` in the current locale, a Run-time error 13 - "Type mismatch" will result from the comparison.

Care should be taken when making comparisons between `Double` or `Single` values and Booleans. Unlike other numeric types, non-zero values cannot be assumed to be **`True`** due to VBA's behavior of promoting the data type of a comparison involving a floating point number to `Double`:

```
Public Sub Example()
    Dim Test As Double

    Test = 42        Debug.Print CBool(Test)             'Prints True.
    'True is promoted to Double - Test is not cast to Boolean
    Debug.Print Test = True             'Prints False

    'With explicit casts:
    Debug.Print CBool(Test) = True      'Prints True
    Debug.Print CDbl(-1) = CDbl(True)   'Prints True
End Sub
```

Section 21.3: Bitwise \ Logical Operators

All of the logical operators in VBA can be thought of as "overrides" of the bitwise operators of the same name. Technically, they are *always* treated as bitwise operators. All of the comparison operators in VBA return a Boolean, which will always have none of its bits set (**`False`**) or *all* of its bits set (**`True`**). But it will treat a value with *any* bit set as **`True`**. This means that the result of the casting the bitwise result of an expression to a `Boolean` (see Comparison Operators) will always be the same as treating it as a logical expression.

Assigning the result of an expression using one of these operators will give the bitwise result. Note that in the truth tables below, `0` is equivalent to **`False`** and `1` is equivalent to **`True`**.

`And`

Returns **`True`** if the expressions on both sides evaluate to **`True`**.

Left-hand Operand Right-hand Operand Result

0	0	0
0	1	0
1	0	0
1	1	1

或

如果表达式任一边的值计算为True，则返回True。

左操作数	右操作数	结果
0	0	0
0	1	1
1	0	1
1	1	1

非

如果表达式计算为False则返回True，若表达式计算为True则返回False。

右操作数	结果
0	1
1	0

非是唯一没有左操作数的操作符。Visual Basic 编辑器会自动简化带有左操作数的表达式
如果你输入...

```
Debug.Print x Not y
```

...VBE 将把该行更改为：

```
Debug.Print Not x
```

对任何包含左操作数（包括表达式）的表达式，类似的简化也将应用于
Not。

Xor

也称为“异或”。当两个表达式的结果不同，返回True。

左操作数	右操作数	结果
0	0	0
0	1	1
1	0	1
1	1	0

注意，虽然Xor运算符可以用作逻辑运算符，但完全没有必要这样做，因为它的结果与比较运算符<>相同。

Eqv

也称为“等价”。当两个表达式的结果相同时，返回True。

左操作数	右操作数	结果
0	0	1

0	0	0
0	1	0
1	0	0
1	1	1

Or

Returns **True** if either side of the expression evaluates to **True**.

Left-hand Operand	Right-hand Operand	Result
0	0	0
0	1	1
1	0	1
1	1	1

Not

Returns **True** if the expression evaluates to **False** and **False** if the expression evaluations to **True**.

Right-hand Operand	Result
0	1
1	0

Not is the only operand without a Left-hand operand. The Visual Basic Editor will automatically simplify expressions with a left hand argument. If you type...

```
Debug.Print x Not y
```

...the VBE will change the line to:

```
Debug.Print Not x
```

Similar simplifications will be made to any expression that contains a left-hand operand (including expressions) for **Not**.

Xor

Also known as "exclusive or". Returns **True** if both expressions evaluate to different results.

Left-hand Operand	Right-hand Operand	Result
0	0	0
0	1	1
1	0	1
1	1	0

Note that although the **Xor** operator can be *used* like a logical operator, there is absolutely no reason to do so as it gives the same result as the comparison operator `<>`.

Eqv

Also known as "equivalence". Returns **True** when both expressions evaluate to the same result.

Left-hand Operand	Right-hand Operand	Result
0	0	1

0	1	0
1	0	0
1	1	1

注意，Eqv函数非常少用，因为x Eqv y等价于更易读的Not (x Xor y)。

`Imp`

也称为“蕴含”。如果两个操作数相同或第二个操作数为True，则返回True。

左操作数	右操作数	结果
0	0	1
0	1	1
1	0	0
1	1	1

请注意，Imp函数很少使用。一个好的经验法则是，如果你无法解释它的含义，
你应该使用其他结构。

第21.4节：数学运算符

按优先级顺序列出：

符号	名称	说明
^	指数运算	返回将左操作数提升到右操作数幂的结果。 请注意，指数运算返回的值始终是Double类型，无论被除数的值类型如何。任何对结果的类型强制转换都发生在计算完成之后。
/	除法1	返回左操作数除以右操作数的结果。注意，除法返回的值始终是Double类型，无论被除数的值类型如何。任何对结果的类型强制转换都发生在计算完成之后。
*	乘法1	返回两个操作数的乘积。
\	整数除法	返回左操作数除以右操作数的整数结果 ***在双方均按0.5四舍五入（向下取整）后。除法的任何余数都会被忽略。*** 如果右操作数（除数）为0，将导致运行时错误11：除以零错误。 注意这是在所有四舍五入完成之后——表达式如3 \ 0.4也会导致除以零错误。
取模	模	返回左操作数除以右操作数的整数余数。 除法前，两边的操作数都会被四舍五入为整数，.5向下取整。例如，8.6 Mod 3 和 12 Mod 2.6 的结果均为 0。如果右操作数（除数）为 0，将导致运行时错误11：除以零。请注意，这是在所有四舍五入完成之后——表达式如 3 Mod 0.4 也会导致除以零错误。
-	减法2	返回左操作数减去右操作数的结果。
+	加法2	返回两个操作数的和。注意，当该符号应用于字符串时，也被视为连接运算符。详见连接运算符。

1 乘法和除法被视为具有相同的优先级。

2 加法和减法被视为具有相同的优先级。

0	1	0
1	0	0
1	1	1

Note that the Eqv function is *very* rarely used as `x Eqv y` is equivalent to the much more readable **`Not`** `(x` **`Xor`** `y)`.

`Imp`

Also known as "implication". Returns **`True`** if both operands are the same *or* the second operand is **`True`**.

Left-hand Operand	Right-hand Operand	Result
0	0	1
0	1	1
1	0	0
1	1	1

Note that the `Imp` function is very rarely used. A good rule of thumb is that if you can't explain what it means, you should use another construct.

Section 21.4: Mathematical Operators

Listed in order of precedence:

Token	Name	Description
^	Exponentiation	Return the result of raising the left-hand operand to the power of the right-hand operand. Note that the value returned by exponentiation is *always* a `Double`, regardless of the value types being divided. Any coercion of the result into a variable type takes place ***after*** the calculation is performed.
/	Division1	Returns the result of dividing the left-hand operand by the right-hand operand. Note that the value returned by division is *always* a `Double`, regardless of the value types being divided. Any coercion of the result into a variable type takes place ***after*** the calculation is performed.
*	Multiplication1	Returns the product of 2 operands.
\	Integer Division	Returns the integer result of dividing the left-hand operand by the right-hand operand ***after*** rounding both sides with .5 rounding down. Any remainder of the division is ignored. If the right-hand operand (the divisor) is `0`, a Run-time error 11: Division by zero will result. Note that this is ***after*** all rounding is performed - expressions such as `3 \ 0.4` will also result in a division by zero error.
`Mod`	Modulo	Returns the integer remainder of dividing the left-hand operand by the right-hand operand. The operand on each side is rounded to an integer *before* the division, with .5 rounding down. For example, both `8.6` **`Mod`** `3` and `12` **`Mod`** `2.6` result in `0`. If the right-hand operand (the divisor) is `0`, a Run-time error 11: Division by zero will result. Note that this is ***after*** all rounding is performed - expressions such as `3` **`Mod`** `0.4` will also result in a division by zero error.
-	Subtraction2	Returns the result of subtracting the right-hand operand from the left-hand operand.
+	Addition2	Returns the sum of 2 operands. Note that this token also treated as a concatenation operator when it is applied to a `String`. See **Concatenation Operators**.

1 Multiplication and division are treated as having the same precedence.

2 Addition and subtraction are treated as having the same precedence.

第22章：排序

与.NET框架不同，Visual Basic for Applications库不包含用于排序数组的例程。

有两种解决方法：1）从头实现排序算法，或2）使用其他常见库中的排序例程
。

第22.1节：算法实现——一维数组的快速排序

来自VBA数组排序函数？

```
Public Sub QuickSort(vArray As Variant, inLow As Long, inHi As Long)

  Dim pivot   As Variant
  Dim tmpSwap As Variant
  Dim tmpLow  As Long
  Dim tmpHi   As Long

  tmpLow = inLow
  tmpHi = inHi

pivot = vArray((inLow + inHi) \ 2)

  当 (tmpLow <= tmpHi)

     当 (vArray(tmpLow) < pivot 且 tmpLow < inHi)
       tmpLow = tmpLow + 1
     循环结束

     当 (pivot < vArray(tmpHi) 且 tmpHi > inLow)
       tmpHi = tmpHi - 1
     循环结束

     如果 (tmpLow <= tmpHi) 则
        tmpSwap = vArray(tmpLow)
        vArray(tmpLow) = vArray(tmpHi)
        vArray(tmpHi) = tmpSwap
tmpLow = tmpLow + 1
        tmpHi = tmpHi - 1
     End If

  循环结束

  如果 (inLow < tmpHi) 则 QuickSort vArray, inLow, tmpHi
  如果 (tmpLow < inHi) 则 QuickSort vArray, tmpLow, inHi

End Sub
```

第22.2节：使用Excel库对一维数组进行排序

此代码利用了Microsoft Excel对象库中的Sort类。

更多阅读，请参见：

- 将范围复制到虚拟范围

Chapter 22: Sorting

Unlike the .NET framework, the Visual Basic for Applications library does not include routines to sort arrays.

There are two types of workarounds: 1) implementing a sorting algorithm from scratch, or 2) using sorting routines in other commonly-available libraries.

Section 22.1: Algorithm Implementation - Quick Sort on a One-Dimensional Array

From VBA array sort function?

```
Public Sub QuickSort(vArray As Variant, inLow As Long, inHi As Long)

  Dim pivot   As Variant
  Dim tmpSwap As Variant
  Dim tmpLow  As Long
  Dim tmpHi   As Long

  tmpLow = inLow
  tmpHi = inHi

  pivot = vArray((inLow + inHi) \ 2)

  While (tmpLow <= tmpHi)

     While (vArray(tmpLow) < pivot And tmpLow < inHi)
        tmpLow = tmpLow + 1
     Wend

     While (pivot < vArray(tmpHi) And tmpHi > inLow)
        tmpHi = tmpHi - 1
     Wend

     If (tmpLow <= tmpHi) Then
        tmpSwap = vArray(tmpLow)
        vArray(tmpLow) = vArray(tmpHi)
        vArray(tmpHi) = tmpSwap
        tmpLow = tmpLow + 1
        tmpHi = tmpHi - 1
     End If

  Wend

  If (inLow < tmpHi) Then QuickSort vArray, inLow, tmpHi
  If (tmpLow < inHi) Then QuickSort vArray, tmpLow, inHi

End Sub
```

Section 22.2: Using the Excel Library to Sort a One-Dimensional Array

This code takes advantage of the `Sort` class in the Microsoft Excel Object Library.

For further reading, see:

- Copy a range to a virtual range

- 如何将选定范围复制到给定数组？

```
Sub testExcelSort()

Dim arr As Variant

InitArray arr
ExcelSort arr

End Sub

Private Sub InitArray(arr As Variant)

Const size = 10
ReDim arr(size)

Dim i As Integer

' 向数组添加降序数字以开始
For i = 0 To size
    arr(i) = size - i
Next i

End Sub

Private Sub ExcelSort(arr As Variant)

' 初始化Excel对象（必需）
Dim xl As New Excel.Application
Dim wbk As Workbook
Set wbk = xl.Workbooks.Add
Dim sht As Worksheet
Set sht = wbk.ActiveSheet

' 将数组复制到Range对象
Dim rng As Range
Set rng = sht.Range("A1")
Set rng = rng.Resize(UBound(arr, 1), 1)
rng.Value = xl.WorksheetFunction.Transpose(arr)

' 在工作表的范围上运行排序例程
Dim MySort As Sort
Set MySort = sht.Sort

With MySort
    .SortFields.Clear
    .SortFields.Add rng, xlSortOnValues, xlAscending, xlSortNormal
    .SetRange rng
    .Header = xlNo
    .Apply
结束于

' 将结果复制回数组
CopyRangeToArray rng, arr

' 清除对象
Set rng = Nothing
wbk.Close False
xl.Quit

End Sub
```


- How to copy selected range into given array?

```
Sub testExcelSort()

Dim arr As Variant

InitArray arr
ExcelSort arr

End Sub

Private Sub InitArray(arr As Variant)

Const size = 10
ReDim arr(size)

Dim i As Integer

' Add descending numbers to the array to start
For i = 0 To size
    arr(i) = size - i
Next i

End Sub

Private Sub ExcelSort(arr As Variant)

' Ininitialize the Excel objects (required)
Dim xl As New Excel.Application
Dim wbk As Workbook
Set wbk = xl.Workbooks.Add
Dim sht As Worksheet
Set sht = wbk.ActiveSheet

' Copy the array to the Range object
Dim rng As Range
Set rng = sht.Range("A1")
Set rng = rng.Resize(UBound(arr, 1), 1)
rng.Value = xl.WorksheetFunction.Transpose(arr)

' Run the worksheet's sort routine on the Range
Dim MySort As Sort
Set MySort = sht.Sort

With MySort
    .SortFields.Clear
    .SortFields.Add rng, xlSortOnValues, xlAscending, xlSortNormal
    .SetRange rng
    .Header = xlNo
    .Apply
End With

' Copy the results back to the array
CopyRangeToArray rng, arr

' Clear the objects
Set rng = Nothing
wbk.Close False
xl.Quit

End Sub
```

```vba
Private Sub CopyRangeToArray(rng As Range, arr)

Dim i As Long
Dim c As Range

'不能直接将数组设置为 Range.value（会增加一个维度）
For Each c In rng.Cells
arr(i) = c.Value
    i = i + 1
Next c

End Sub
```



```vba
Private Sub CopyRangeToArray(rng As Range, arr)

Dim i As Long
Dim c As Range

' Can't just set the array to Range.value (adds a dimension)
For Each c In rng.Cells
    arr(i) = c.Value
    i = i + 1
Next c

End Sub
```

第23章：流程控制结构

第23.1节：For 循环

For 循环用于重复执行括号内的代码指定次数。以下简单示例
说明了基本语法：

```
Dim i as Integer            '声明变量 i
For i = 1 to 10             '声明循环执行的次数
    Debug.Print i           '重复执行的代码段
Next i                  '循环结束
```

上述代码声明了一个 Integer 类型的变量 i。For 循环将 1 到 10 之间的每个值赋给 i，随后执行 Debug.Print i - 即代码将数字1到10打印到即时窗口。注意，循环变量由Next语句递增，也就是说递增发生在包含的代码执行之后，而不是之前。

默认情况下，每次循环执行时计数器会递增1。然而，可以指定一个Step来改变递增的量，该值可以是字面量或函数的返回值。如果起始值、结束值或Step值是浮点数，则会四舍五入到最接近的整数。Step可以是正值也可以是负值。

```
Dim i As Integer
For i = 1 To 10 Step 2
Debug.Print i       '打印1、3、5、7和9
Next
```

通常，For循环用于在循环开始前已知需要执行多少次包含代码的情况（否则Do或While循环可能更合适）。这是因为退出条件在第一次进入循环后即固定，如下代码所示：

```
Private Iterations As Long              '模块范围

Public Sub Example()
    声明 i 为 Long 类型
Iterations = 10
    For i = 1 To Iterations
Debug.Print Iterations      '打印10到1，递减。
Iterations = Iterations - 1
    Next
End Sub
```

可以使用Exit For语句提前退出For循环：

```
Dim i As Integer

对于 i = 1 到 10
    如果 i > 5 那么
        退出循环
    End If
Debug.Print i       '在循环提前退出前打印1, 2, 3, 4, 5。
Next
```


Chapter 23: Flow control structures

Section 23.1: For loop

The `For` loop is used to repeat the enclosed section of code a given number of times. The following simple example illustrates the basic syntax:

```
Dim i as Integer            'Declaration of i
For i = 1 to 10             'Declare how many times the loop shall be executed
    Debug.Print i           'The piece of code which is repeated
Next i                      'The end of the loop
```

The code above declares an Integer `i`. The `For` loop assigns every value between 1 and 10 to `i` and then executes `Debug.Print i` - i.e. the code prints the numbers 1 through 10 to the immediate window. Note that the loop variable is incremented by the `Next` statement, that is after the enclosed code executes as opposed to before it executes.

By default, the counter will be incremented by 1 each time the loop executes. However, a `Step` can be specified to change the amount of the increment as either a literal or the return value of a function. If the starting value, ending value, or `Step` value is a floating point number, it will be rounded to the nearest integer value. `Step` can be either a positive or negative value.

```
Dim i As Integer
For i = 1 To 10 Step 2
    Debug.Print i       'Prints 1, 3, 5, 7, and 9
Next
```

In general a `For` loop would be used in situations where it is known before the loop starts how many times to execute the enclosed code (otherwise a `Do` or `While` loop may be more appropriate). This is because the exit condition is fixed after the first entry into loop, as this code demonstrates:

```
Private Iterations As Long              'Module scope

Public Sub Example()
    Dim i As Long
    Iterations = 10
    For i = 1 To Iterations
        Debug.Print Iterations      'Prints 10 through 1, descending.
        Iterations = Iterations - 1
    Next
End Sub
```

A `For` loop can be exited early with the `Exit For` statement:

```
Dim i As Integer

For i = 1 To 10
    If i > 5 Then
        Exit For
    End If
    Debug.Print i       'Prints 1, 2, 3, 4, 5 before loop exits early.
Next
```

第23.2节：选择案例

`SELECT CASE` 可用于多种不同条件的情况。条件从上到下依次检查，只有第一个匹配的案例会被执行。

```
子程序 TestCase()
    定义 MyVar 为 字符串

    选择案例 MyVar     '我们选择变量 MyVar 进行操作
        案例 "Hello"       '现在我们只检查想要检查的案例
            MsgBox "这个案例"
        案例 "World"
MsgBox "重要"
        案例 "How"
            MsgBox "东西"
        Case "是"
MsgBox "我快没主意了"
        Case "你？", "今天"   '你可以用逗号分隔多个条件
            MsgBox "呃..." '如果匹配任意条件，将进入该case
        Case Else                '如果没有其他case匹配
MsgBox "所有其他情况都失败了"
    结束选择

    Dim i As Integer
    Select Case i
        Case Is > 2 '"Is" 可以用来代替条件中的变量。
MsgBox "i 大于 2"
        'Case 2 < Is '"Is" 只能用在条件的开头。
        'Case Else 是可选的
    End Select
End Sub
```

`SELECT CASE` 代码块的逻辑也可以反转，以支持对不同变量的测试，在这种情况下我们也可以使用逻辑运算符：

```
Dim x As Integer
Dim y As Integer

x = 2
y = 5

选择情况为真
    案例 x > 3
        MsgBox "x 大于 3"
    Case y < 2
        MsgBox "y 小于 2"
    案例 x = 1
        MsgBox "x 等于 1"
    情况 x = 2 异或 y = 3
        MsgBox "去了解一下“异或”"
    情况 非 y = 5
        MsgBox "y 不是 5"
    情况 x = 3 或者 x = 10
MsgBox "x = 3 或 10"
    情况 y < 10 且 x < 10
        MsgBox "x 和 y 都小于 10"
    其他情况
MsgBox "未找到匹配项"
结束选择
```


Section 23.2: Select Case

`SELECT CASE` can be used when many different conditions are possible. The conditions are checked from top to bottom and only the first case that match will be executed.

```
Sub TestCase()
    Dim MyVar As String

    Select Case MyVar     'We Select the Variable MyVar to Work with
        Case "Hello"      'Now we simply check the cases we want to check
            MsgBox "This Case"
        Case "World"
            MsgBox "Important"
        Case "How"
            MsgBox "Stuff"
        Case "Are"
            MsgBox "I'm running out of ideas"
        Case "You?", "Today"  'You can separate several conditions with a comma
            MsgBox "Uuuhm..." 'if any is matched it will go into the case
        Case Else             'If none of the other cases is hit
            MsgBox "All of the other cases failed"
    End Select

    Dim i As Integer
    Select Case i
        Case Is > 2 '"Is" can be used instead of the variable in conditions.
            MsgBox "i is greater than 2"
        'Case 2 < Is '"Is" can only be used at the beginning of the condition.
        'Case Else is optional
    End Select
End Sub
```

The logic of the `SELECT CASE` block can be inverted to support testing of different variables too, in this kind of scenario we can also use logical operators:

```
Dim x As Integer
Dim y As Integer

x = 2
y = 5

Select Case True
    Case x > 3
        MsgBox "x is greater than 3"
    Case y < 2
        MsgBox "y is less than 2"
    Case x = 1
        MsgBox "x is equal to 1"
    Case x = 2 Xor y = 3
        MsgBox "Go read about ""Xor"""
    Case Not y = 5
        MsgBox "y is not 5"
    Case x = 3 Or x = 10
        MsgBox "x = 3 or 10"
    Case y < 10 And x < 10
        MsgBox "x and y are less than 10"
    Case Else
        MsgBox "No match found"
End Select
```

情况语句也可以使用算术运算符。当算术运算符用于SELECT CASE 值时，应在前面加上Is关键字：

```
Dim x 作为 整数

x = 5

Select Case x
    情况 1
MsgBox "x 等于 1"
    情况 2, 3, 4
MsgBox "x 是 2、3 或 4"
    案例 7 至 10
MsgBox "x 在 7 到 10 之间（含）"
    Case Is < 2
MsgBox "x 小于 1"
    Case Is >= 7
MsgBox "x 大于或等于 7"
    Case Else
MsgBox "未找到匹配项"
End Select
```

第 23.3 节：For Each 循环

For Each循环结构非常适合遍历集合中的所有元素。

```
Public Sub 遍历集合(ByVal items As Collection)

    'For Each 迭代器必须始终是变体
    Dim element As Variant

    For Each element In items
        '假设 element 可以转换为字符串
        Debug.Print element
    Next

End Sub
```

在遍历对象集合时使用 For Each：

```
Dim sheet As Worksheet
For Each sheet In ActiveWorkbook.Worksheets
    Debug.Print sheet.Name
Next
```

遍历数组时避免使用 `For Each`；使用 `For` 循环在数组上会有显著更好的性能。相反，遍历 `Collection` 时，For Each 循环会有更好的性能。

语法

```
For Each [item] In [collection]
    [statements]
Next [item]
```

`Next` 关键字后面可以选择性地跟上迭代变量；这有助于澄清嵌套循环，尽管还有更好的方法来澄清嵌套代码，比如将内层循环提取到独立的过程。

```
Dim book As Workbook
For Each book In Application.Workbooks
```


Case statements can also use arithmetic operators. Where an arithmetic operator is being used against the `SELECT CASE` value it should be preceded with the `Is` keyword:

```
Dim x As Integer

x = 5

Select Case x
    Case 1
        MsgBox "x equals 1"
    Case 2, 3, 4
        MsgBox "x is 2, 3 or 4"
    Case 7 To 10
        MsgBox "x is between 7 and 10 (inclusive)"
    Case Is < 2
        MsgBox "x is less than one"
    Case Is >= 7
        MsgBox "x is greater than or equal to 7"
    Case Else
        MsgBox "no match found"
End Select
```

Section 23.3: For Each loop

The `For Each` loop construct is ideal for iterating all elements of a collection.

```
Public Sub IterateCollection(ByVal items As Collection)

    'For Each iterator must always be variant
    Dim element As Variant

    For Each element In items
        'assumes element can be converted to a string
        Debug.Print element
    Next

End Sub
```

Use `For Each` when iterating object collections:

```
Dim sheet As Worksheet
For Each sheet In ActiveWorkbook.Worksheets
    Debug.Print sheet.Name
Next
```

Avoid `For Each` when iterating arrays; a `For` loop will offer significantly better performance with arrays. Conversely, a `For Each` loop will offer better performance when iterating a `Collection`.

Syntax

```
For Each [item] In [collection]
    [statements]
Next [item]
```

The `Next` keyword may optionally be followed by the iterator variable; this can help clarify nested loops, although there are better ways to clarify nested code, such as extracting the inner loop into its own procedure.

```
Dim book As Workbook
For Each book In Application.Workbooks
```

```
    Debug.Print book.FullName

    定义 sheet 作为 工作表
    对于每个 sheet 在 ActiveWorkbook.Worksheets中
        调试打印 sheet.Name
    下一个 sheet
下一个 book
```

第23.4节：Do循环

```
 公共子程序 DoLoop()
    定义 entry 作为 字符串
entry = ""
    '等同于While循环，将持续询问字符串直到输入"Stop"
    '建议使用While循环代替这种形式的Do循环
    当 entry <> "Stop"时循环
entry = 输入框("输入字符串，输入Stop结束")
        调试打印 entry
    循环

    等同于上述循环，但条件仅在循环的第一次迭代之后检查，因此即使在进入循环之前 entry 等于"S
    top"，它也至少会执行一次（如本例所示）

    Do
entry = 输入框("输入字符串，输入Stop结束")
        调试打印 entry
    循环 当 entry <> "Stop"


    等同于写作 Do While Not entry="Stop"

    因为 Until 在循环顶部，当评估条件时 entry 仍然等于"Stop"，所以
    不会执行

    执行 直到 entry = "Stop"
entry = 输入框("输入字符串，输入Stop结束")
        调试打印 entry
    循环

    等同于写作 Do ... Loop While Not i >= 100
    Do
entry = 输入框("输入字符串，输入Stop结束")
        调试打印 entry
    循环 直到 entry = "Stop"
结束子程序
```

第23.5节：While 循环

```
将返回数组中是否存在某个元素
Public Function IsInArray(values() As String, ByVal whatToFind As String) As Boolean
    Dim i As Integer
i = 0

    当 i < UBound(values) 且 values(i) <> whatToFind
        i = i + 1
    循环结束

    IsInArray = values(i) = whatToFind
End Function
```



```
    Debug.Print book.FullName

    Dim sheet As Worksheet
    For Each sheet In ActiveWorkbook.Worksheets
        Debug.Print sheet.Name
    Next sheet
Next book
```

Section 23.4: Do loop

```
Public Sub DoLoop()
    Dim entry As String
    entry = ""
    'Equivalent to a While loop will ask for strings until "Stop" in given
    'Prefer using a While loop instead of this form of Do loop
    Do While entry <> "Stop"
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop

    'Equivalent to the above loop, but the condition is only checked AFTER the
    'first iteration of the loop, so it will execute even at least once even
    'if entry is equal to "Stop" before entering the loop (like in this case)
    Do
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop While entry <> "Stop"


    'Equivalent to writing Do While Not entry="Stop"
    '
    'Because the Until is at the top of the loop, it will
    'not execute because entry is still equal to "Stop"
    'when evaluating the condition
    Do Until entry = "Stop"
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop

    'Equivalent to writing Do ... Loop While Not i >= 100
    Do
        entry = InputBox("Enter a string, Stop to end")
        Debug.Print entry
    Loop Until entry = "Stop"
End Sub
```

Section 23.5: While loop

```
'Will return whether an element is present in the array
Public Function IsInArray(values() As String, ByVal whatToFind As String) As Boolean
    Dim i As Integer
    i = 0

    While i < UBound(values) And values(i) <> whatToFind
        i = i + 1
    Wend

    IsInArray = values(i) = whatToFind
End Function
```

第24章：按引用（ByRef）或按值（ByVal）传递参数

ByRef 和 ByVal 修饰符是过程签名的一部分，用于指示参数如何传递给过程。在VBA中，参数默认按 ByRef 传递，除非另有说明（即如果缺失，ByRef 是隐式的）。

注意 在许多其他编程语言（包括VB.NET）中，如果未指定修饰符，参数默认按值传递：建议显式指定 ByRef 修饰符以避免可能的混淆。

第24.1节：按引用（ByRef）和按值（ByVal）传递简单变量

按 ByRef 或 ByVal 传递表示调用过程（CallingProcedure）是将参数的实际值传递给被调用过程（CalledProcedure），还是传递一个引用（在某些其他语言中称为指针）。

被调用过程（CalledProcedure）。

如果参数按 ByRef 传递，则传递的是参数的内存地址给被调用过程（CalledProcedure），被调用过程对该参数的任何修改都会反映到调用过程（CallingProcedure）中的值。

如果一个参数以ByVal方式传递，传递给CalledProcedure的是实际值，而不是变量的引用。

一个简单的例子可以清楚地说明这一点：

```
Sub CalledProcedure(ByRef X As Long, ByVal Y As Long)
    X = 321
Y = 654
End Sub

Sub CallingProcedure()
    Dim A As Long
    Dim B As Long
A = 123
B = 456

Debug.Print "调用前 => A: " & CStr(A), "B: " & CStr(B)
    "结果 : 调用前 => A: 123 B: 456

    CalledProcedure X:=A, Y:=B

Debug.Print "调用后 =  A: " & CStr(A), "B: " & CStr(B)
    "结果 : 调用后 => A: 321 B: 456
End Sub
```

另一个例子：

```
Sub Main()
    Dim IntVarByVal As Integer
    Dim IntVarByRef As Integer

IntVarByVal = 5
IntVarByRef = 10

SubChangeArguments IntVarByVal, IntVarByRef '5 作为"副本"传入。10 作为引用传入
    Debug.Print "IntVarByVal: " & IntVarByVal '打印 5（SubChangeArguments 未修改）
    Debug.Print "IntVarByRef: " & IntVarByRef '打印 99（变量在
SubChangeArguments 中被修改）
End Sub
```


Chapter 24: Passing Arguments ByRef or ByVal

The **ByRef** and **ByVal** modifiers are part of a procedure's signature and indicate how an argument is passed to a procedure. In VBA a parameter is passed **ByRef** unless specified otherwise (i.e. **ByRef** is implicit if absent).

Note In many other programming languages (including VB.NET), parameters are implicitly passed by value if no modifier is specified: consider specifying **ByRef** modifiers explicitly to avoid possible confusion.

Section 24.1: Passing Simple Variables ByRef And ByVal

Passing **ByRef** or **ByVal** indicates whether the actual value of an argument is passed to the `CalledProcedure` by the `CallingProcedure`, or whether a reference (called a pointer in some other languages) is passed to the `CalledProcedure`.

If an argument is passed **ByRef**, the memory address of the argument is passed to the `CalledProcedure` and any modification to that parameter by the `CalledProcedure` is made to the value in the `CallingProcedure`.

If an argument is passed **ByVal**, the actual value, not a reference to the variable, is passed to the `CalledProcedure`.

A simple example will illustrate this clearly:

```
Sub CalledProcedure(ByRef X As Long, ByVal Y As Long)
    X = 321
    Y = 654
End Sub

Sub CallingProcedure()
    Dim A As Long
    Dim B As Long
    A = 123
    B = 456

    Debug.Print "BEFORE CALL => A: " & CStr(A), "B: " & CStr(B)
    ''Result : BEFORE CALL => A: 123 B: 456

    CalledProcedure X:=A, Y:=B

    Debug.Print "AFTER CALL =  A: " & CStr(A), "B: " & CStr(B)
    ''Result : AFTER CALL => A: 321 B: 456
End Sub
```

Another example:

```
Sub Main()
    Dim IntVarByVal As Integer
    Dim IntVarByRef As Integer

    IntVarByVal = 5
    IntVarByRef = 10

    SubChangeArguments IntVarByVal, IntVarByRef '5 goes in as a "copy". 10 goes in as a reference
    Debug.Print "IntVarByVal: " & IntVarByVal 'prints 5 (no change made by SubChangeArguments)
    Debug.Print "IntVarByRef: " & IntVarByRef 'prints 99 (the variable was changed in
SubChangeArguments)
End Sub
```

```
子程序 SubChangeArguments(ByVal ParameterByVal 作为 整数, ByRef ParameterByRef 作为 整数)
    ParameterByVal = ParameterByVal + 2 ' 5 + 2 = 7（仅在此子程序内更改）
ParameterByRef = ParameterByRef + 89 ' 10 + 89 = 99（更改了 IntVarByRef 本身 - 在
主子程序中）
End Sub
```

第24.2节：ByRef（按引用传递）

默认修饰符

如果未为参数指定修饰符，则该参数隐式按引用传递。

```
公共子程序 DoSomething1(foo 作为 长整数)
结束子程序
```

```
公共子程序 DoSomething2(ByRef foo 作为 长整数)
结束子程序
```

参数foo在DoSomething1和DoSomething2中均按ByRef传递。

> 注意！如果你有其他语言的经验后开始使用VBA，这很可能与你习惯的行为完全相反。在许多其他编程语言（包括VB.NET）中，隐式/默认修饰符是按值传递参数的。

按引用传递

- 当一个值以引用传递(ByRef)时，过程接收该值的引用。

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething foo
    Debug.Print foo
End Sub

Private Sub DoSomething(ByRef foo As Long)
    foo = foo * 2
End Sub
```

调用上述Test过程输出84。DoSomething接收了foo的引用，因此操作的是与调用者相同的内存地址。

- 当通过ByRef传递引用时，过程接收的是指针的引用。

```
Public Sub Test()
    Dim foo As Collection
    Set foo = New Collection
    DoSomething foo
    Debug.Print foo.Count
End Sub

Private Sub DoSomething(ByRef foo As Collection)
    foo.Add 42
```



```
Sub SubChangeArguments(ByVal ParameterByVal As Integer, ByRef ParameterByRef As Integer)
    ParameterByVal = ParameterByVal + 2 ' 5 + 2 = 7 (changed only inside this Sub)
    ParameterByRef = ParameterByRef + 89 ' 10 + 89 = 99 (changes the IntVarByRef itself - in the
Main Sub)
End Sub
```

Section 24.2: ByRef

Default modifier

If no modifier is specified for a parameter, that parameter is implicitly passed by reference.

```
Public Sub DoSomething1(foo As Long)
End Sub
```

```
Public Sub DoSomething2(ByRef foo As Long)
End Sub
```

The `foo` parameter is passed `ByRef` in both `DoSomething1` and `DoSomething2`.

> **Watch out!** If you're coming to VBA with experience from other languages, this is very likely the exact opposite behavior to the one you're used to. In many other programming languages (including VB.NET), the implicit/default modifier passes parameters by value.

Passing by reference

- When a *value* is passed `ByRef`, the procedure receives **a reference** to the value.

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething foo
    Debug.Print foo
End Sub

Private Sub DoSomething(ByRef foo As Long)
    foo = foo * 2
End Sub
```

Calling the above `Test` procedure outputs 84. `DoSomething` is given `foo` and receives a *reference* to the value, and therefore works with the same memory address as the caller.

- When a *reference* is passed `ByRef`, the procedure receives **a reference** to the pointer.

```
Public Sub Test()
    Dim foo As Collection
    Set foo = New Collection
    DoSomething foo
    Debug.Print foo.Count
End Sub

Private Sub DoSomething(ByRef foo As Collection)
    foo.Add 42
```

```
    Set foo = Nothing
End Sub
```

上述代码会引发运行时错误91，因为调用者正在调用一个已不存在对象的Count成员，原因是DoSomething接收了对象指针的引用并在返回前将其赋值为Nothing。

在调用处强制使用 ByVal

在调用处使用括号，可以覆盖ByRef并强制参数以ByVal方式传递：

```
Public Sub Test()
    Dim foo As Long
    foo = 42
   DoSomething (foo)
   Debug.Print foo
End Sub

Private Sub DoSomething(ByRef foo As Long)
    foo = foo * 2
End Sub
```

上述代码输出42，无论是否隐式或显式指定了ByRef。

> 注意！ 因此，在过程调用中使用多余的括号很容易引入错误。
> 注意过程名和参数列表之间的空格：
>
> ```
> bar = DoSomething(foo) '函数调用，无空格；括号是参数列表的一部分
> DoSomething (foo) '过程调用，注意空格；括号不是参数列表的一部分
> DoSomething foo '过程调用不会强制将foo参数设为ByVal
> ```

第24.3节：ByVal

按值传递

- 当一个值以ByVal方式传递时，过程接收的是该值的副本。

```
Public Sub Test()
    Dim foo As Long
    foo = 42
   DoSomething foo
   Debug.Print foo
End Sub

Private Sub DoSomething(ByVal foo As Long)
    foo = foo * 2
End Sub
```

调用上述Test过程输出42。DoSomething接收foo的副本。该副本被乘以2，过程结束时被丢弃；调用者的副本从未被修改。

- 当一个引用以值传递(ByVal)方式传递时，过程接收的是指针的副本。


```
    Set foo = Nothing
End Sub
```

The above code raises run-time error 91, because the caller is calling the `Count` member of an object that no longer exists, because `DoSomething` was given a *reference* to the object pointer and assigned it to **`Nothing`** before returning.

Forcing ByVal at call site

Using parentheses at the call site, you can override **`ByRef`** and force an argument to be passed **`ByVal`**:

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething (foo)
    Debug.Print foo
End Sub

Private Sub DoSomething(ByRef foo As Long)
    foo = foo * 2
End Sub
```

The above code outputs 42, regardless of whether **`ByRef`** is specified implicitly or explicitly.

> **Watch out!** Because of this, using extraneous parentheses in procedure calls can easily introduce bugs.
> Pay attention to the whitespace between the procedure name and the argument list:
>
> ```
> bar = DoSomething(foo) 'function call, no whitespace; parens are part of args list
> DoSomething (foo) 'procedure call, notice whitespace; parens are NOT part of args list
> DoSomething foo 'procedure call does not force the foo parameter to be ByVal
> ```

Section 24.3: ByVal

Passing by value

- When a *value* is passed **`ByVal`**, the procedure receives **a copy** of the value.

```
Public Sub Test()
    Dim foo As Long
    foo = 42
    DoSomething foo
    Debug.Print foo
End Sub

Private Sub DoSomething(ByVal foo As Long)
    foo = foo * 2
End Sub
```

Calling the above `Test` procedure outputs 42. `DoSomething` is given `foo` and receives **a copy** of the value. The copy is multiplied by 2, and then discarded when the procedure exits; the caller's copy was never altered.

- When a *reference* is passed **`ByVal`**, the procedure receives **a copy** of the pointer.

```
Public Sub Test()
    Dim foo As Collection
    Set foo = New Collection
    DoSomething foo
    Debug.Print foo.Count
End Sub

Private Sub DoSomething(ByVal foo As Collection)
    foo.Add 42
    Set foo = Nothing
End Sub
```

调用上述Test过程输出1。DoSomething接收foo并获得指向Collection对象的指针的副本。因为Test作用域中的foo对象变量指向同一个对象，在DoSomething中添加项目即是向同一个对象添加项目。由于它是指针的副本，将其引用设置为Nothing不会影响调用者自己的副本。


```
Public Sub Test()
    Dim foo As Collection
    Set foo = New Collection
    DoSomething foo
    Debug.Print foo.Count
End Sub

Private Sub DoSomething(ByVal foo As Collection)
    foo.Add 42
    Set foo = Nothing
End Sub
```

Calling the above `Test` procedure outputs 1. `DoSomething` is given `foo` and receives *a copy* of **the pointer** to the `Collection` object. Because the `foo` object variable in the `Test` scope points to the same object, adding an item in `DoSomething` adds the item to the same object. Because it's *a copy* of the pointer, setting its reference to `Nothing` does not affect the caller's own copy.

第25章：Scripting.FileSystemObject

第25.1节：仅从文件路径中获取路径

GetParentFolderName方法返回任何路径的父文件夹。虽然这也可以用于文件夹，
但它更适合从绝对文件路径中提取路径：

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetParentFolderName("C:\Users\Me\My Documents\SomeFile.txt")
```

输出

```
C:\Users\Me\My Documents
```

注意，返回的字符串中不包含尾部的路径分隔符。

第25.2节：仅从文件名中获取扩展名

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetExtensionName("MyFile.something.txt")
```

打印 txt 注意 GetExtensionName() 方法已经处理了文件名中多个点的情况。

第25.3节：递归枚举文件夹和文件

早绑定（引用了Microsoft Scripting Runtime）

```
Sub EnumerateFilesAndFolders( _
    FolderPath As String, _
    Optional MaxDepth As Long = -1, _
    Optional CurrentDepth As Long = 0, _
    Optional Indentation As Long = 2)

    Dim FSO As Scripting.FileSystemObject
    Set FSO = New Scripting.FileSystemObject

    '检查文件夹是否存在
    If FSO.FolderExists(FolderPath) Then
        Dim fldr As Scripting.Folder
        Set fldr = FSO.GetFolder(FolderPath)

        '输出起始目录路径
        If CurrentDepth = 0 Then
         Debug.Print fldr.Path
        End If

        列举子文件夹
        Dim subFldr As Scripting.Folder
        For Each subFldr In fldr.SubFolders
Debug.Print Space$((CurrentDepth + 1) * Indentation) & subFldr.Name
            If CurrentDepth < MaxDepth Or MaxDepth = -1 Then
                '递归调用 EnumerateFilesAndFolders
EnumerateFilesAndFolders subFldr.Path, MaxDepth, CurrentDepth + 1, Indentation
            End If
        Next subFldr
```


Chapter 25: Scripting.FileSystemObject

Section 25.1: Retrieve only the path from a file path

The GetParentFolderName method returns the parent folder for any path. While this can also be used with folders, it is arguably more useful for extracting the path from an absolute file path:

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetParentFolderName("C:\Users\Me\My Documents\SomeFile.txt")
```

Prints

```
C:\Users\Me\My Documents
```

Note that the trailing path separator is not included in the returned string.

Section 25.2: Retrieve just the extension from a file name

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetExtensionName("MyFile.something.txt")
```

Prints `txt` Note that the `GetExtensionName()` method already handles multiple periods in a file name.

Section 25.3: Recursively enumerate folders and files

Early Bound (with a reference to `Microsoft Scripting Runtime`)

```
Sub EnumerateFilesAndFolders( _
    FolderPath As String, _
    Optional MaxDepth As Long = -1, _
    Optional CurrentDepth As Long = 0, _
    Optional Indentation As Long = 2)

    Dim FSO As Scripting.FileSystemObject
    Set FSO = New Scripting.FileSystemObject

    'Check the folder exists
    If FSO.FolderExists(FolderPath) Then
        Dim fldr As Scripting.Folder
        Set fldr = FSO.GetFolder(FolderPath)

        'Output the starting directory path
        If CurrentDepth = 0 Then
          Debug.Print fldr.Path
        End If

        'Enumerate the subfolders
        Dim subFldr As Scripting.Folder
        For Each subFldr In fldr.SubFolders
            Debug.Print Space$((CurrentDepth + 1) * Indentation) & subFldr.Name
            If CurrentDepth < MaxDepth Or MaxDepth = -1 Then
                'Recursively call EnumerateFilesAndFolders
                EnumerateFilesAndFolders subFldr.Path, MaxDepth, CurrentDepth + 1, Indentation
            End If
        Next subFldr
```

```
        '枚举文件
        Dim fil As Scripting.File
        For Each fil In fldr.Files
Debug.Print Space$((CurrentDepth + 1) * Indentation) & fil.Name
        Next fil
    结束 如果
结束 子程序
```

调用时带有参数时的输出：EnumerateFilesAndFolders"C:\Test"

```
C:\Test
文档
    个人
      预算.xls
      食谱.doc
    工作
计划.doc
  下载
FooBar.exe
  说明.txt
```

调用时带有参数时的输出：EnumerateFilesAndFolders"C:\Test", 0

```
C:\Test
  文档
  下载
  说明.txt
```

调用时带有参数时的输出：EnumerateFilesAndFolders"C:\Test", 1, 4

```
C:\Test
文档
        个人
        工作
    下载
         FooBar.exe
    说明.txt
```

第25.4节：从文件名中去除文件扩展名

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetBaseName("MyFile.something.txt")
```

打印 MyFile.something

注意 GetBaseName() 方法已经处理了文件名中的多个点。

第25.5节：使用FileSystemObject枚举目录中的文件

早期绑定（需要引用 Microsoft Scripting Runtime）：

```
Public Sub 枚举目录()
    Dim fso As Scripting.FileSystemObject
    Set fso = New Scripting.FileSystemObject

    Dim 目标文件夹 As Folder
```



```
        'Enumerate the files
        Dim fil As Scripting.File
        For Each fil In fldr.Files
            Debug.Print Space$((CurrentDepth + 1) * Indentation) & fil.Name
        Next fil
    End If
End Sub
```

Output when called with arguments like: `EnumerateFilesAndFolders "C:\Test"`

```
C:\Test
  Documents
    Personal
      Budget.xls
      Recipes.doc
    Work
      Planning.doc
  Downloads
    FooBar.exe
  ReadMe.txt
```

Output when called with arguments like: `EnumerateFilesAndFolders "C:\Test", 0`

```
C:\Test
  Documents
  Downloads
  ReadMe.txt
```

Output when called with arguments like: `EnumerateFilesAndFolders "C:\Test", 1, 4`

```
C:\Test
    Documents
        Personal
        Work
    Downloads
        FooBar.exe
    ReadMe.txt
```

Section 25.4: Strip file extension from a file name

```
Dim fso As New Scripting.FileSystemObject
Debug.Print fso.GetBaseName("MyFile.something.txt")
```

Prints `MyFile.something`

Note that the `GetBaseName()` method already handles multiple periods in a file name.

Section 25.5: Enumerate files in a directory using FileSystemObject

Early bound (requires a reference to Microsoft Scripting Runtime):

```
Public Sub EnumerateDirectory()
    Dim fso As Scripting.FileSystemObject
    Set fso = New Scripting.FileSystemObject

    Dim targetFolder As Folder
```

```
    Set 目标文件夹 = fso.GetFolder("C:\")

    Dim 找到的文件 As Variant
    For Each 找到的文件 In 目标文件夹.Files
       Debug.Print 找到的文件.Name
    Next
End Sub
```

晚期绑定：

```
Public Sub 枚举目录()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim targetFolder As Object
    Set targetFolder = fso.GetFolder("C:\")

    Dim foundFile As Variant
    For Each foundFile In targetFolder.Files
       Debug.Print foundFile.Name
    Next
End Sub
```

第25.6节：创建FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8

Sub FsoExample()
    Dim fso As Object ' 声明变量
    Set fso = CreateObject("Scripting.FileSystemObject") ' 将其设置为文件系统对象

    ' 现在用它来检查文件是否存在
    Dim myFilePath As String
    myFilePath = "C:\mypatho\myfile.txt"
    If fso.FileExists(myFilePath) Then
        ' 执行某些操作
    Else
        '文件'不存在
MsgBox "文件不存在"
    结束 如果
结束 子程序
```

第25.7节：使用FileSystemObject读取文本文件

```
Const 读取模式 = 1
Const 写入模式 = 2
Const 追加模式 = 8

Sub 读取文本文件示例()
    Dim fso 作为 对象
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim sourceFile 作为 对象
    Dim myFilePath 作为 字符串
    Dim myFileText 作为 字符串

myFilePath = "C:\mypatho\myfile.txt"
```



```
    Set targetFolder = fso.GetFolder("C:\")

    Dim foundFile As Variant
    For Each foundFile In targetFolder.Files
        Debug.Print foundFile.Name
    Next
End Sub
```

Late bound:

```
Public Sub EnumerateDirectory()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim targetFolder As Object
    Set targetFolder = fso.GetFolder("C:\")

    Dim foundFile As Variant
    For Each foundFile In targetFolder.Files
        Debug.Print foundFile.Name
    Next
End Sub
```

Section 25.6: Creating a FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8

Sub FsoExample()
    Dim fso As Object ' declare variable
    Set fso = CreateObject("Scripting.FileSystemObject") ' Set it to be a File System Object

    ' now use it to check if a file exists
    Dim myFilePath As String
    myFilePath = "C:\mypath\to\myfile.txt"
    If fso.FileExists(myFilePath) Then
        ' do something
    Else
        ' file doesn't exist
        MsgBox "File doesn't exist"
    End If
End Sub
```

Section 25.7: Reading a text file using a FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8

Sub ReadTextFileExample()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim sourceFile As Object
    Dim myFilePath As String
    Dim myFileText As String

    myFilePath = "C:\mypath\to\myfile.txt"
```

```
    Set sourceFile = fso.OpenTextFile(myFilePath, 读取模式)
    myFileText = sourceFile.ReadAll ' myFileText 现在包含文本文件的内容
    sourceFile.Close ' 关闭文件
    '对文本执行你需要做的任何操作

    ' 你也可以逐行读取它
    Dim line As String
    Set sourceFile = fso.OpenTextFile(myFilePath, ForReading)
    While Not sourceFile.AtEndOfStream ' 当我们还没有读完文件时
        line = sourceFile.ReadLine
        '对该行执行某些操作...
    循环结束
    sourceFile.Close
End Sub
```

第25.8节：使用FileSystemObject创建文本文件

```
Sub CreateTextFileExample()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim targetFile As Object
    Dim myFilePath As String
    Dim myFileText As String

myFilePath = "C:\mypatho\myfile.txt"
    Set targetFile = fso.CreateTextFile(myFilePath, True)  ' 这将覆盖任何现有文件
    targetFile.Write "这是一些新文本"
targetFile.Write " 并且这段文本将紧跟在第一段文本之后。"
    targetFile.WriteLine "这段文本包含换行符，以确保每次写入都占据
自己的一行。"
targetFile.Close ' 关闭文件
End Sub
```

第25.9节：使用FSO.BuildPath从文件夹路径和文件名构建完整路径

如果您接受用户输入的文件夹路径，可能需要在构建文件路径之前检查是否有尾部反斜杠（\）。FSO.BuildPath方法使这更简单：

```
  Const sourceFilePath As String = "C:\Temp"  ' <-- 无尾部反斜杠
  Const targetFilePath As String = "C:\Temp\" ' <-- 有尾部反斜杠

  Const fileName As String = "Results.txt"

  Dim FSO As FileSystemObject
  Set FSO = New FileSystemObject

Debug.Print FSO.BuildPath(sourceFilePath, fileName)
  Debug.Print FSO.BuildPath(targetFilePath, fileName)
```

输出：

```
C:\Temp\Results.txt
C:\Temp\Results.txt
```



```
    Set sourceFile = fso.OpenTextFile(myFilePath, ForReading)
    myFileText = sourceFile.ReadAll ' myFileText now contains the content of the text file
    sourceFile.Close ' close the file
    ' do whatever you might need to do with the text

    ' You can also read it line by line
    Dim line As String
    Set sourceFile = fso.OpenTextFile(myFilePath, ForReading)
    While Not sourceFile.AtEndOfStream ' while we are not finished reading through the file
        line = sourceFile.ReadLine
        ' do something with the line...
    Wend
    sourceFile.Close
End Sub
```

Section 25.8: Creating a text file with FileSystemObject

```
Sub CreateTextFileExample()
    Dim fso As Object
    Set fso = CreateObject("Scripting.FileSystemObject")

    Dim targetFile As Object
    Dim myFilePath As String
    Dim myFileText As String

    myFilePath = "C:\mypath\to\myfile.txt"
    Set targetFile = fso.CreateTextFile(myFilePath, True)  ' this will overwrite any existing file
    targetFile.Write "This is some new text"
    targetFile.Write " And this text will appear right after the first bit of text."
    targetFile.WriteLine "This bit of text includes a newline character to ensure each write takes
its own line."
    targetFile.Close ' close the file
End Sub
```

Section 25.9: Using FSO.BuildPath to build a Full Path from folder path and file name

If you're accepting user input for folder paths, you might need to check for trailing backslashes (\) before building a file path. The `FSO.BuildPath` method makes this simpler:

```
  Const sourceFilePath As String = "C:\Temp"  '<-- Without trailing backslash
  Const targetFilePath As String = "C:\Temp\" '<-- With trailing backslash

  Const fileName As String = "Results.txt"

  Dim FSO As FileSystemObject
  Set FSO = New FileSystemObject

  Debug.Print FSO.BuildPath(sourceFilePath, fileName)
  Debug.Print FSO.BuildPath(targetFilePath, fileName)
```

Output:

```
C:\Temp\Results.txt
C:\Temp\Results.txt
```

第25.10节：使用FileSystemObject写入现有文件

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8


Sub WriteTextFileExample()
    Dim oFso
    Set oFso = CreateObject("Scripting.FileSystemObject")

    Dim oFile as Object
    Dim myFilePath as String
    Dim myFileText as String

myFilePath = "C:\mypatho\myfile.txt"
    ' 首先检查文件是否存在
    If oFso.FileExists(myFilePath) Then
        ' 这将用你发送给文件的内容覆盖任何现有文件内容
        ' 若要向现有文件末尾追加数据，请使用 ForAppending
        Set oFile = oFso.OpenTextFile(myFilePath, ForWriting)
    Else
        ' 否则创建文件
        Set oFile = oFso.CreateTextFile(myFilePath) ' 跳过可选的覆盖布尔值，因为我们已经检查文件不存在。

    End If
oFile.Write "这是一些新文本"
oFile.Write " 这段文本将紧跟在第一段文本之后。"    oFile.WriteLine "这段文本包含换行符，确保每次
写入都占据独立一行。"

oFile.Close ' 关闭文件
End Sub
```


Section 25.10: Writing to an existing file with FileSystemObject

```
Const ForReading = 1
Const ForWriting = 2
Const ForAppending = 8


Sub WriteTextFileExample()
    Dim oFso
    Set oFso = CreateObject("Scripting.FileSystemObject")

    Dim oFile as Object
    Dim myFilePath as String
    Dim myFileText as String

    myFilePath = "C:\mypath\to\myfile.txt"
    ' First check if the file exists
    If oFso.FileExists(myFilePath) Then
        ' this will overwrite any existing filecontent with whatever you send the file
        ' to append data to the end of an existing file, use ForAppending instead
        Set oFile = oFso.OpenTextFile(myFilePath, ForWriting)
    Else
        ' create the file instead
        Set oFile = oFso.CreateTextFile(myFilePath) ' skipping the optional boolean for overwrite if
exists as we already checked that the file doesn't exist.
    End If
    oFile.Write "This is some new text"
    oFile.Write " And this text will appear right after the first bit of text."
    oFile.WriteLine "This bit of text includes a newline character to ensure each write takes its
own line."
    oFile.Close ' close the file
End Sub
```

第26章：不使用文件和目录的操作文件系统对象

第26.1节：确定文件夹和文件是否存在

文件：

要确定文件是否存在，只需将文件名传递给Dir$函数并测试其是否返回结果。注意Dir$支持通配符，因此要测试一个特定的文件，传入的pathName应被测试以确保不包含通配符。下面的示例会引发错误——如果这不是期望的行为，可以将函数修改为简单返回False。

```
公共函数 FileExists(pathName 作为字符串) 作为布尔值
    如果 InStr(1, pathName, "*") 或 InStr(1, pathName, "?") 那么
        '退出函数    '遇到通配符返回False。
Err.Raise 52    '遇到通配符时引发错误。
    End If
FileExists = Dir$(pathName) <> vbNullString
End Function
```

文件夹（Dir$ 方法）：

函数Dir$()也可以通过为可选的attributes参数传递vbDirectory来确定文件夹是否存在。在这种情况下，传入的path Name值必须以路径分隔符（\）结尾，因为匹配文件名会导致误报。请记住，通配符只允许出现在最后一个路径分隔符之后，因此下面的示例函数如果输入包含通配符，将抛出运行时错误52——“文件名或号码错误”。如果这不是期望的行为，可以取消注释函数顶部的On Error Resume Next。此外还要记住，Dir$支持相对文件路径（例如..\Foo\Bar），因此只要当前工作目录不变，结果才保证有效。

```
公共函数 FolderExists(ByVal pathName 作为字符串) 作为布尔值'如果路径中包含通
    配符时希望返回False而不是引发错误，请取消注释“On Error”行。

    出错时继续下一条
    如果 pathName = vbNullString 或Right$(pathName, 1) <> "\" 则
        退出函数
    End If
FolderExists = Dir$(pathName, vbDirectory) <> vbNullString
End Function
```

文件夹（ChDir 方法）：

ChDir语句也可以用来测试文件夹是否存在。请注意，此方法会临时更改VBA运行的环境，因此如果这是一个考虑因素，应改用Dir$方法。它的优点是对参数的容错性较低。此方法也支持相对文件路径，因此与Dir$方法有相同的注意事项。

```
Public Function FolderExists(ByVal pathName As String) As Boolean
    '缓存当前工作目录
    Dim cached As String
    cached = CurDir$

    出错时继续下一条
```


Chapter 26: Working With Files and Directories Without Using FileSystemObject

Section 26.1: Determining If Folders and Files Exist

Files:

To determine if a file exists, simply pass the filename to the `Dir$` function and test to see if it returns a result. Note that `Dir$` supports wild-cards, so to test for a *specific* file, the passed `pathName` should to be tested to ensure that it does not contain them. The sample below raises an error - if this isn't the desired behavior, the function can be changed to simply return **`False`**.

```
Public Function FileExists(pathName As String) As Boolean
    If InStr(1, pathName, "*") Or InStr(1, pathName, "?") Then
        'Exit Function   'Return False on wild-cards.
        Err.Raise 52    'Raise error on wild-cards.
    End If
    FileExists = Dir$(pathName) <> vbNullString
End Function
```

Folders (Dir$ method):

The `Dir$()` function can also be used to determine if a folder exists by specifying passing `vbDirectory` for the optional `attributes` parameter. In this case, the passed `pathName` value must end with a path separator (\), as matching *filenames* will cause false positives. Keep in mind that wild-cards are only allowed after the last path separator, so the example function below will throw a run-time error 52 - "Bad file name or number" if the input contains a wild-card. If this isn't the desired behavior, uncomment **`On Error Resume Next`** at the top of the function. Also remember that `Dir$` supports relative file paths (i.e. `..\Foo\Bar`), so results are only guaranteed to be valid as long as the current working directory is not changed.

```
Public Function FolderExists(ByVal pathName As String) As Boolean
    'Uncomment the "On Error" line if paths with wild-cards should return False
    'instead of raising an error.
    'On Error Resume Next
    If pathName = vbNullString Or Right$(pathName, 1) <> "\" Then
        Exit Function
    End If
    FolderExists = Dir$(pathName, vbDirectory) <> vbNullString
End Function
```

Folders (ChDir method):

The `ChDir` statement can also be used to test if a folder exists. Note that this method will temporarily change the environment that VBA is running in, so if that is a consideration, the `Dir$` method should be used instead. It does have the advantage of being much less forgiving with its parameter. This method also supports relative file paths, so has the same caveat as the `Dir$` method.

```
Public Function FolderExists(ByVal pathName As String) As Boolean
    'Cache the current working directory
    Dim cached As String
    cached = CurDir$

    On Error Resume Next
```

```
ChDir pathName
    FolderExists = Err.Number = 0
    出错时转到0
    '切换回缓存的工作目录。
    ChDir cached
End Function
```

第26.2节：创建和删除文件夹

注意： 为简洁起见，以下示例使用本主题中**确定文件夹和文件是否存在的示例中的 FolderExists 函数。**

MkDir 语句可用于创建新文件夹。它接受包含驱动器字母的路径（C:\Foo）、UNC 名称（\\Server\Foo）、相对路径（..\Foo）或当前工作目录（Foo）。

如果省略驱动器或 UNC 名称（即 \Foo），则文件夹将在当前驱动器上创建。该驱动器可能与当前工作目录所在驱动器相同，也可能不同。

```
Public Sub MakeNewDirectory(ByVal pathName As String)
    'MkDir 如果目录已存在则会失败。
    If FolderExists(pathName) Then Exit Sub
    '这仍可能因权限等原因失败。
    MkDir pathName
End Sub
```

RmDir 语句可用于删除现有文件夹。它接受与 MkDir 相同形式的路径，并且与当前工作目录和驱动器的关系相同。请注意，该语句类似于 Windowsrd 命令，因此如果目标目录非空，将抛出运行时错误 75："路径/文件访问错误"。

```
Public Sub DeleteDirectory(ByVal pathName As String)
    If Right$(pathName, 1) <> "\" Then
pathName = pathName & "\"
    End If
    '如果目录不存在，Rmdir 将失败。
    如果 文件夹不存在(pathName) 则退出子程序
    '如果目录中包含文件，Rmdir 将失败。
    如果 Dir$(pathName & "*") <> vbNullString 则退出子程序

    '如果目录中包含子目录，Rmdir 将失败。
    声明 subDir 为 字符串
subDir = Dir$(pathName & "*", vbDirectory)
    Do
        如果 subDir <> "." 且 subDir <> ".." 则退出子程序
       subDir = Dir$(, vbDirectory)
    循环直到 subDir = vbNullString

    '这可能仍会因权限等原因失败。
    RmDir pathName
End Sub
```



```
    ChDir pathName
    FolderExists = Err.Number = 0
    On Error GoTo 0
    'Change back to the cached working directory.
    ChDir cached
End Function
```

Section 26.2: Creating and Deleting File Folders

NOTE: For brevity, the examples below use the FolderExists function from the **Determining If Folders and Files Exist** example in this topic.

The MkDir statement can be used to create a new folder. It accepts paths containing drive letters (C:\Foo), UNC names (\\Server\Foo), relative paths (..\Foo), or the current working directory (Foo).

If the drive or UNC name is omitted (i.e. \Foo), the folder is created on the current drive. This may or may not be the same drive as the current working directory.

```
Public Sub MakeNewDirectory(ByVal pathName As String)
    'MkDir will fail if the directory already exists.
    If FolderExists(pathName) Then Exit Sub
    'This may still fail due to permissions, etc.
    MkDir pathName
End Sub
```

The RmDir statement can be used to delete existing folders. It accepts paths in the same forms as MkDir and uses the same relationship to the current working directory and drive. Note that the statement is similar to the Windows rd shell command, so will throw a run-time error 75: "Path/File access error" if the target directory is not empty.

```
Public Sub DeleteDirectory(ByVal pathName As String)
    If Right$(pathName, 1) <> "\" Then
        pathName = pathName & "\"
    End If
    'Rmdir will fail if the directory doesn't exist.
    If Not FolderExists(pathName) Then Exit Sub
    'Rmdir will fail if the directory contains files.
    If Dir$(pathName & "*") <> vbNullString Then Exit Sub

    'Rmdir will fail if the directory contains directories.
    Dim subDir As String
    subDir = Dir$(pathName & "*", vbDirectory)
    Do
        If subDir <> "." And subDir <> ".." Then Exit Sub
        subDir = Dir$(, vbDirectory)
    Loop While subDir <> vbNullString

    'This may still fail due to permissions, etc.
    RmDir pathName
End Sub
```

第27章：在

VBA中以二进制读取超过2GB的文件及文件哈希

VBA内置了一个简单的方法以二进制方式读取文件，但它有2GB（2,147,483,647字节 - Long数据类型的最大值）的限制。随着技术的发展，这个2GB限制很容易被突破。例如，操作系统安装DVD光盘的ISO镜像。微软确实提供了一种通过低级Windows API克服此限制的方法，以下是其备份。

还演示了（读取部分）如何计算文件哈希，而无需使用微软的外部程序如fciv.exe。

第27.1节：这必须在类模块中，稍后示例中称为“Random”

```
如何突破VBA的2GB文件限制
来源：https://support.microsoft.com/en-us/kb/189981（已存档）
此代码必须放在类模块中

Option Explicit

Public Enum W32F_Errors
    W32F_UNKNOWN_ERROR  =45600
    W32F_FILE_ALREADY_OPEN
    W32F_PROBLEM_OPENING_FILE
    W32F_FILE_ALREADY_CLOSED
    W32F_Problem_seeking
End Enum

Private Const W32F_SOURCE = "Win32File 对象"
Private Const GENERIC_WRITE = &H40000000
Private Const GENERIC_READ = &H80000000
Private Const FILE_ATTRIBUTE_NORMAL = &H80
Private Const CREATE_ALWAYS = 2
Private Const OPEN_ALWAYS = 4
Private Const INVALID_HANDLE_VALUE = -1

Private Const FILE_BEGIN = 0, FILE_CURRENT = 1, FILE_END = 2

Private Const FORMAT_MESSAGE_FROM_SYSTEM = &H1000

Private Declare Function FormatMessage Lib "kernel32" Alias "FormatMessageA" ( _
    ByVal dwFlags As Long, _
lpSource As Long, _
    ByVal dwMessageId As Long, _
    ByVal dwLanguageId As Long, _
    ByVal lpBuffer As String, _
    ByVal nSize As Long, _
参数 作为 Any) 作为 Long

Private Declare Function ReadFile 库 "kernel32" ( _
    ByVal hFile 作为 Long, _
lpBuffer 作为 Any, _
    ByVal nNumberOfBytesToRead 作为 Long, _
lpNumberOfBytesRead 作为 Long, _
    ByVal lpOverlapped 作为 Long) 作为 Long

Private Declare Function CloseHandle 库 "kernel32" (ByVal hObject 作为 Long) 作为 Long

Private Declare Function WriteFile 库 "kernel32" ( _
```


Chapter 27: Reading 2GB+ files in binary in VBA and File Hashes

There is a built in easy way to read files in binary within VBA, however it has a restriction of 2GB (2,147,483,647 bytes - max of Long data type). As technology evolves, this 2GB limit is easily breached. e.g. an ISO image of Operating System install DVD disc. Microsoft does provide a way to overcome this via low level Windows API and here is a backup of it.

Also demonstrate (Read part) for calculating File Hashes without external program like `fciv.exe` from Microsoft.

Section 27.1: This have to be in a Class module, examples later referred as "Random"

```
' How To Seek Past VBA's 2GB File Limit
' Source: https://support.microsoft.com/en-us/kb/189981 (Archived)
' This must be in a Class Module

Option Explicit

Public Enum W32F_Errors
    W32F_UNKNOWN_ERROR = 45600
    W32F_FILE_ALREADY_OPEN
    W32F_PROBLEM_OPENING_FILE
    W32F_FILE_ALREADY_CLOSED
    W32F_Problem_seeking
End Enum

Private Const W32F_SOURCE = "Win32File Object"
Private Const W32F_SOURCE = "Win32File Object"
Private Const GENERIC_WRITE = &H40000000
Private Const GENERIC_READ = &H80000000
Private Const FILE_ATTRIBUTE_NORMAL = &H80
Private Const CREATE_ALWAYS = 2
Private Const OPEN_ALWAYS = 4
Private Const INVALID_HANDLE_VALUE = -1

Private Const FILE_BEGIN = 0, FILE_CURRENT = 1, FILE_END = 2

Private Const FORMAT_MESSAGE_FROM_SYSTEM = &H1000

Private Declare Function FormatMessage Lib "kernel32" Alias "FormatMessageA" ( _
    ByVal dwFlags As Long, _
    lpSource As Long, _
    ByVal dwMessageId As Long, _
    ByVal dwLanguageId As Long, _
    ByVal lpBuffer As String, _
    ByVal nSize As Long, _
    Arguments As Any) As Long

Private Declare Function ReadFile Lib "kernel32" ( _
    ByVal hFile As Long, _
    lpBuffer As Any, _
    ByVal nNumberOfBytesToRead As Long, _
    lpNumberOfBytesRead As Long, _
    ByVal lpOverlapped As Long) As Long

Private Declare Function CloseHandle Lib "kernel32" (ByVal hObject As Long) As Long

Private Declare Function WriteFile Lib "kernel32" ( _
```

```
    ByVal hFile 作为 Long, _
lpBuffer 作为 Any, _
    ByVal nNumberOfBytesToWrite 作为 Long, _
    lpNumberOfBytesWritten 作为 Long, _
    ByVal lpOverlapped 作为 Long) 作为 Long

Private Declare Function CreateFile 库 "kernel32" 别名 "CreateFileA" ( _
    ByVal lpFileName 作为 String, _
    ByVal dwDesiredAccess As Long, _
    ByVal dwShareMode As Long, _
    ByVal lpSecurityAttributes As Long, _
    ByVal dwCreationDisposition As Long, _
    ByVal dwFlagsAndAttributes As Long, _
    ByVal hTemplateFile As Long) As Long

Private Declare Function SetFilePointer Lib "kernel32" ( _
    ByVal hFile As Long, _
    ByVal lDistanceToMove As Long, _
lpDistanceToMoveHigh As Long, _
    ByVal dwMoveMethod As Long) As Long

Private Declare Function FlushFileBuffers Lib "kernel32" (ByVal hFile As Long) As Long

Private hFile As Long, sFName As String, fAutoFlush As Boolean

Public Property Get FileHandle() As Long
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FileHandle = hFile
End Property

Public Property Get FileName() As String
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FileName = sFName
End Property

Public Property Get IsOpen() As Boolean
   IsOpen = hFile <> INVALID_HANDLE_VALUE
End Property

Public Property Get AutoFlush() As Boolean
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    AutoFlush = fAutoFlush
End Property

Public Property Let AutoFlush(ByVal NewVal As Boolean)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
fAutoFlush = NewVal
End Property

Public Sub OpenFile(ByVal sFileName As String)
    If hFile <> INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_OPEN, sFName
    End If
hFile = CreateFile(sFileName, GENERIC_WRITE Or GENERIC_READ, 0, 0, OPEN_ALWAYS,
```



```
    ByVal hFile As Long, _
    lpBuffer As Any, _
    ByVal nNumberOfBytesToWrite As Long, _
    lpNumberOfBytesWritten As Long, _
    ByVal lpOverlapped As Long) As Long

Private Declare Function CreateFile Lib "kernel32" Alias "CreateFileA" ( _
    ByVal lpFileName As String, _
    ByVal dwDesiredAccess As Long, _
    ByVal dwShareMode As Long, _
    ByVal lpSecurityAttributes As Long, _
    ByVal dwCreationDisposition As Long, _
    ByVal dwFlagsAndAttributes As Long, _
    ByVal hTemplateFile As Long) As Long

Private Declare Function SetFilePointer Lib "kernel32" ( _
    ByVal hFile As Long, _
    ByVal lDistanceToMove As Long, _
    lpDistanceToMoveHigh As Long, _
    ByVal dwMoveMethod As Long) As Long

Private Declare Function FlushFileBuffers Lib "kernel32" (ByVal hFile As Long) As Long

Private hFile As Long, sFName As String, fAutoFlush As Boolean

Public Property Get FileHandle() As Long
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FileHandle = hFile
End Property

Public Property Get FileName() As String
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FileName = sFName
End Property

Public Property Get IsOpen() As Boolean
    IsOpen = hFile <> INVALID_HANDLE_VALUE
End Property

Public Property Get AutoFlush() As Boolean
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    AutoFlush = fAutoFlush
End Property

Public Property Let AutoFlush(ByVal NewVal As Boolean)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    fAutoFlush = NewVal
End Property

Public Sub OpenFile(ByVal sFileName As String)
    If hFile <> INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_OPEN, sFName
    End If
    hFile = CreateFile(sFileName, GENERIC_WRITE Or GENERIC_READ, 0, 0, OPEN_ALWAYS,
```

```
FILE_ATTRIBUTE_NORMAL, 0)
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_PROBLEM_OPENING_FILE, sFileName
    End If
sFName = sFileName
End Sub

Public Sub CloseFile()
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_CLOSED
    End If
CloseHandle hFile
    sFName = ""
fAutoFlush = False
   hFile = INVALID_HANDLE_VALUE
End Sub

Public Function ReadBytes(ByVal ByteCount As Long) As Variant
    Dim BytesRead As Long, Bytes() As Byte
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    ReDim Bytes(0 To ByteCount - 1) As Byte
    ReadFile hFile, Bytes(0), ByteCount, BytesRead, 0
   ReadBytes = Bytes
End Function

Public Sub WriteBytes(DataBytes() As Byte)
    Dim fSuccess As Long, BytesToWrite As Long, BytesWritten As Long
    If hFile = INVALID_HANDLE_VALUE Then
RaiseError W32F_FILE_ALREADY_CLOSED
    End If
BytesToWrite = UBound(DataBytes) - LBound(DataBytes) + 1
    fSuccess = WriteFile(hFile, DataBytes(LBound(DataBytes)), BytesToWrite, BytesWritten, 0)
    If fAutoFlush Then Flush
End Sub

公共子程序 Flush()
    如果 hFile = INVALID_HANDLE_VALUE 则
       引发错误 W32F_FILE_ALREADY_CLOSED
    结束如果
刷新文件缓冲区 hFile
End Sub

公共子程序 绝对定位(ByVal HighPos 作为 Long, ByVal LowPos 作为 Long)
    如果 hFile = INVALID_HANDLE_VALUE 则
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
LowPos = 设置文件指针(hFile, LowPos, HighPos, FILE_BEGIN)
End Sub

公共子程序 相对定位(ByVal Offset 作为 Long)
    定义 TempLow 作为 Long, TempErr 作为 Long
    If hFile = INVALID_HANDLE_VALUE Then
       RaiseError W32F_FILE_ALREADY_CLOSED
    End If
TempLow = 设置文件指针(hFile, Offset, ByVal 0&, FILE_CURRENT)
    如果 TempLow = -1 则
        TempErr = Err.LastDllError
        如果 TempErr 则
引发错误 W32F_Problem_seeking, "错误 " & TempErr & "." & vbCrLf & CStr(TempErr)
        End If
```



```
FILE_ATTRIBUTE_NORMAL, 0)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_PROBLEM_OPENING_FILE, sFileName
    End If
    sFName = sFileName
End Sub

Public Sub CloseFile()
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    CloseHandle hFile
    sFName = ""
    fAutoFlush = False
    hFile = INVALID_HANDLE_VALUE
End Sub

Public Function ReadBytes(ByVal ByteCount As Long) As Variant
    Dim BytesRead As Long, Bytes() As Byte
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    ReDim Bytes(0 To ByteCount - 1) As Byte
    ReadFile hFile, Bytes(0), ByteCount, BytesRead, 0
    ReadBytes = Bytes
End Function

Public Sub WriteBytes(DataBytes() As Byte)
    Dim fSuccess As Long, BytesToWrite As Long, BytesWritten As Long
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    BytesToWrite = UBound(DataBytes) - LBound(DataBytes) + 1
    fSuccess = WriteFile(hFile, DataBytes(LBound(DataBytes)), BytesToWrite, BytesWritten, 0)
    If fAutoFlush Then Flush
End Sub

Public Sub Flush()
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    FlushFileBuffers hFile
End Sub

Public Sub SeekAbsolute(ByVal HighPos As Long, ByVal LowPos As Long)
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    LowPos = SetFilePointer(hFile, LowPos, HighPos, FILE_BEGIN)
End Sub

Public Sub SeekRelative(ByVal Offset As Long)
    Dim TempLow As Long, TempErr As Long
    If hFile = INVALID_HANDLE_VALUE Then
        RaiseError W32F_FILE_ALREADY_CLOSED
    End If
    TempLow = SetFilePointer(hFile, Offset, ByVal 0&, FILE_CURRENT)
    If TempLow = -1 Then
        TempErr = Err.LastDllError
        If TempErr Then
            RaiseError W32F_Problem_seeking, "Error " & TempErr & "." & vbCrLf & CStr(TempErr)
        End If
```

```
    End If
End Sub

Private Sub Class_Initialize()
    hFile = INVALID_HANDLE_VALUE
End Sub

私有子程序 Class_Terminate()
    如果 hFile <> INVALID_HANDLE_VALUE 则 关闭句柄 hFile
结束子程序

私有子程序 RaiseError(ByVal ErrorCode 作为 W32F_Errors, 可选 sExtra)
    定义 Win32Err 作为 长整型, Win32Text 作为 字符串
    Win32Err = Err.LastDllError
    如果 Win32Err 则
Win32Text = vbCrLf & "错误 " & Win32Err & vbCrLf & _
        DecodeAPIErrors(Win32Err)
    End If
    选择情况 ErrorCode
        情况 W32F_FILE_ALREADY_OPEN
Err.Raise W32F_FILE_ALREADY_OPEN, W32F_SOURCE, "文件 '" & sExtra & "' 已经
打开。" & Win32Text
        情况 W32F_PROBLEM_OPENING_FILE
Err.Raise W32F_PROBLEM_OPENING_FILE, W32F_SOURCE, "打开文件 '" & sExtra & "' 时出错。" &
Win32Text
        Case W32F_FILE_ALREADY_CLOSED
Err.Raise W32F_FILE_ALREADY_CLOSED, W32F_SOURCE, "没有打开的文件。"
        Case W32F_Problem_seeking
Err.Raise W32F_Problem_seeking, W32F_SOURCE, "查找错误。" & vbCrLf & sExtra
        Case Else
Err.Raise W32F_UNKNOWN_ERROR, W32F_SOURCE, "未知错误。" & Win32Text
    结束选择
End Sub

Private Function DecodeAPIErrors(ByVal ErrorCode As Long) As String
    Dim sMessage As String, MessageLength As Long
sMessage = Space$(256)
MessageLength = FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0&, ErrorCode, 0&, sMessage, 256&,
0&)
    如果 MessageLength > 0 则
DecodeAPIErrors = Left(sMessage, MessageLength)
    Else
DecodeAPIErrors = "未知错误。"
    End If
End Function
```

第27.2节：标准模块中计算文件哈希的代码

```
Private Const HashTypeMD5 As String = "MD5" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.1
10).aspx
Private Const HashTypeSHA1 As String = "SHA1" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1cryptoserviceprovider(v=vs.
110).aspx
Private Const HashTypeSHA256 As String = "SHA256" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256cryptoserviceprovider(v=v
s.110).aspx
Private Const HashTypeSHA384 As String = "SHA384" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384cryptoserviceprovider(v=v
s.110).aspx
```



```
    End If
End Sub

Private Sub Class_Initialize()
    hFile = INVALID_HANDLE_VALUE
End Sub

Private Sub Class_Terminate()
    If hFile <> INVALID_HANDLE_VALUE Then CloseHandle hFile
End Sub

Private Sub RaiseError(ByVal ErrorCode As W32F_Errors, Optional sExtra)
    Dim Win32Err As Long, Win32Text As String
    Win32Err = Err.LastDllError
    If Win32Err Then
        Win32Text = vbCrLf & "Error " & Win32Err & vbCrLf & _
        DecodeAPIErrors(Win32Err)
    End If
    Select Case ErrorCode
        Case W32F_FILE_ALREADY_OPEN
            Err.Raise W32F_FILE_ALREADY_OPEN, W32F_SOURCE, "The file '" & sExtra & "' is already
open." & Win32Text
        Case W32F_PROBLEM_OPENING_FILE
            Err.Raise W32F_PROBLEM_OPENING_FILE, W32F_SOURCE, "Error opening '" & sExtra & "'." &
Win32Text
        Case W32F_FILE_ALREADY_CLOSED
            Err.Raise W32F_FILE_ALREADY_CLOSED, W32F_SOURCE, "There is no open file."
        Case W32F_Problem_seeking
            Err.Raise W32F_Problem_seeking, W32F_SOURCE, "Seek Error." & vbCrLf & sExtra
        Case Else
            Err.Raise W32F_UNKNOWN_ERROR, W32F_SOURCE, "Unknown error." & Win32Text
    End Select
End Sub

Private Function DecodeAPIErrors(ByVal ErrorCode As Long) As String
    Dim sMessage As String, MessageLength As Long
    sMessage = Space$(256)
    MessageLength = FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM, 0&, ErrorCode, 0&, sMessage, 256&,
0&)
    If MessageLength > 0 Then
        DecodeAPIErrors = Left(sMessage, MessageLength)
    Else
        DecodeAPIErrors = "Unknown Error."
    End If
End Function
```

Section 27.2: Code for Calculating File Hash in a Standard module

```
Private Const HashTypeMD5 As String = "MD5" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.1
10).aspx
Private Const HashTypeSHA1 As String = "SHA1" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1cryptoserviceprovider(v=vs.
110).aspx
Private Const HashTypeSHA256 As String = "SHA256" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256cryptoserviceprovider(v=v
s.110).aspx
Private Const HashTypeSHA384 As String = "SHA384" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384cryptoserviceprovider(v=v
s.110).aspx
```

```
Private Const HashTypeSHA512 As String = "SHA512" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.110).aspx

Private uFileSize As Double ' 如果不通过 FileHashes() 测试性能，则注释掉

Sub FileHashes()
    Dim tStart As Date, tFinish As Date, sHash As String, aTestFiles As Variant, oTestFile As
Variant, aBlockSizes As Variant, oBlockSize As Variant
    Dim BLOCKSIZE As Double

    ' 这段代码对不同文件大小和块大小进行性能测试
aBlockSizes = Array("2^12-1", "2^13-1", "2^14-1", "2^15-1", "2^16-1", "2^17-1", "2^18-1",
"2^19-1", "2^20-1", "2^21-1", "2^22-1", "2^23-1", "2^24-1", "2^25-1", "2^26-1")
aTestFiles = Array("C:\ISO\clonezilla-live-2.2.2-37-amd64.iso",
"C:\ISO\HPIP201.2014_0902.29.iso", "C:\ISO\SW_DVD5_Windows_Vista_Business_W32_32BIT_English.ISO",
"C:\ISO\Win10_1607_English_x64.iso",
"C:\ISO\SW_DVD9_Windows_Svr_Std_and_DataCtr_2012_R2_64Bit_English.ISO")
    Debug.Print "测试文件: " & Join(aTestFiles, " | ")
Debug.Print "块大小: " & Join(aBlockSizes, " | ")
    对于每个 oTestFile 在 aTestFiles 中
        调试.打印 oTestFile
          对于每个 oBlockSize 在 aBlockSizes 中
            BLOCKSIZE = 计算(oBlockSize)
            tStart = 现在
sHash = GetFileHash(CStr(oTestFile), BLOCKSIZE, HashTypeMD5)
            tFinish = Now
Debug.Print sHash, uFileSize, Format(tFinish - tStart, "hh:mm:ss"), oBlockSize & " (" &
BLOCKSIZE & ")"
          Next
    Next
End Sub

Private Function GetFileHash(ByVal sFile As String, ByVal uBlockSize As Double, ByVal sHashType As
String) As String
    Dim oFSO As Object ' "Scripting.FileSystemObject"
    Dim oCSP As Object ' "CryptoServiceProvider"之一
    Dim oRnd As Random ' 由微软提供的 "Random" 类，必须在同一文件中
    Dim uBytesRead As Double, uBytesToRead As Double, bDone As Boolean
    Dim aBlock() As Byte, aBytes As Variant ' 用于存储字节的数组
    Dim aHash() As Byte, sHash As String, i As Long
    'Dim uFileSize As Double ' 如果单独使用 GetFileHash()，请取消注释

    Set oRnd = New Random ' 由微软提供的 Random 类
    Set oFSO = CreateObject("Scripting.FileSystemObject")
    Set oCSP = CreateObject("System.Security.Cryptography." & sHashType & "CryptoServiceProvider")

    If oFSO Is Nothing Or oRnd Is Nothing Or oCSP Is Nothing Then
MsgBox "无法创建一个或多个所需对象"
        GoTo CleanUp
    End If

uFileSize = oFSO.GetFile(sFile).Size ' FILELEN() 最大支持2GB！
uBytesRead = 0
bDone = False
sHash = String(oCSP.HashSize / 4, "0") ' 每个十六进制有4位

    Application.ScreenUpdating = False
    ' 以 uBlockSize 或更小的块处理文件
    If uFileSize = 0 Then
        ReDim aBlock(0)
oCSP.TransformFinalBlock aBlock, 0, 0
```



```
Private Const HashTypeSHA512 As String = "SHA512" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.110).aspx

Private uFileSize As Double ' Comment out if not testing performance by FileHashes()

Sub FileHashes()
    Dim tStart As Date, tFinish As Date, sHash As String, aTestFiles As Variant, oTestFile As
Variant, aBlockSizes As Variant, oBlockSize As Variant
    Dim BLOCKSIZE As Double

    ' This performs performance testing on different file sizes and block sizes
    aBlockSizes = Array("2^12-1", "2^13-1", "2^14-1", "2^15-1", "2^16-1", "2^17-1", "2^18-1",
"2^19-1", "2^20-1", "2^21-1", "2^22-1", "2^23-1", "2^24-1", "2^25-1", "2^26-1")
    aTestFiles = Array("C:\ISO\clonezilla-live-2.2.2-37-amd64.iso",
"C:\ISO\HPIP201.2014_0902.29.iso", "C:\ISO\SW_DVD5_Windows_Vista_Business_W32_32BIT_English.ISO",
"C:\ISO\Win10_1607_English_x64.iso",
"C:\ISO\SW_DVD9_Windows_Svr_Std_and_DataCtr_2012_R2_64Bit_English.ISO")
    Debug.Print "Test files: " & Join(aTestFiles, " | ")
    Debug.Print "BlockSizes: " & Join(aBlockSizes, " | ")
    For Each oTestFile In aTestFiles
        Debug.Print oTestFile
        For Each oBlockSize In aBlockSizes
            BLOCKSIZE = Evaluate(oBlockSize)
            tStart = Now
            sHash = GetFileHash(CStr(oTestFile), BLOCKSIZE, HashTypeMD5)
            tFinish = Now
            Debug.Print sHash, uFileSize, Format(tFinish - tStart, "hh:mm:ss"), oBlockSize & " (" &
BLOCKSIZE & ")"
        Next
    Next
End Sub

Private Function GetFileHash(ByVal sFile As String, ByVal uBlockSize As Double, ByVal sHashType As
String) As String
    Dim oFSO As Object ' "Scripting.FileSystemObject"
    Dim oCSP As Object ' One of the "CryptoServiceProvider"
    Dim oRnd As Random ' "Random" Class by Microsoft, must be in the same file
    Dim uBytesRead As Double, uBytesToRead As Double, bDone As Boolean
    Dim aBlock() As Byte, aBytes As Variant ' Arrays to store bytes
    Dim aHash() As Byte, sHash As String, i As Long
    'Dim uFileSize As Double ' Un-Comment if GetFileHash() is to be used individually

    Set oRnd = New Random ' Class by Microsoft: Random
    Set oFSO = CreateObject("Scripting.FileSystemObject")
    Set oCSP = CreateObject("System.Security.Cryptography." & sHashType & "CryptoServiceProvider")

    If oFSO Is Nothing Or oRnd Is Nothing Or oCSP Is Nothing Then
        MsgBox "One or more required objects cannot be created"
        GoTo CleanUp
    End If

    uFileSize = oFSO.GetFile(sFile).Size ' FILELEN() has 2GB max!
    uBytesRead = 0
    bDone = False
    sHash = String(oCSP.HashSize / 4, "0") ' Each hexadecimal has 4 bits

    Application.ScreenUpdating = False
    ' Process the file in chunks of uBlockSize or less
    If uFileSize = 0 Then
        ReDim aBlock(0)
        oCSP.TransformFinalBlock aBlock, 0, 0
```

```
bDone = True
    Else
        With oRnd
            .OpenFile sFile
            Do
                如果 uBytesRead + uBlockSize < uFileSize 那么
                    uBytesToRead = uBlockSize
                Else
uBytesToRead = uFileSize - uBytesRead
                    bDone = True
                End If
                读取一些字节
aBytes = .ReadBytes(uBytesToRead)
                aBlock = aBytes
                如果 bDone 则
oCSP.TransformFinalBlock aBlock, 0, uBytesToRead
                 uBytesRead = uBytesRead + uBytesToRead
                Else
                uBytesRead = uBytesRead + oCSP.TransformBlock(aBlock, 0, uBytesToRead, aBlock,
0)
                End If
DoEvents
            循环 直到 bDone
            .关闭文件
        结束于
    End If
    如果 bDone 则
        将哈希字节数组转换为十六进制字符串
        aHash = oCSP.hash
        对于 i = 0 到 UBound(aHash)
            Mid$(sHash, i * 2 + (aHash(i) > 15) + 2) = Hex(aHash(i))
        Next
    End If
Application.ScreenUpdating = True
    '清理
oCSP.Clear
清理:
    Set oFS0 = Nothing
    Set oRnd = Nothing
    Set oCSP = Nothing
   GetFileHash = sHash
End Function
```

> 输出结果相当有趣，我的测试文件表明 BLOCKSIZE = 131071 (2^17-1) 在 Windows 7 x64 上的 32 位 Office 2010 中整体性能最佳，次佳为 2^16-1 (65535)。注意 2^27-1会导致 内存不足。

文件大小（字节）	文件名
146,800,640	clonezilla-live-2.2.2-37-amd64.iso
798,210,048	HPIP201.2014_0902.29.iso
2,073,016,320	SW_DVD5_Windows_Vista_Business_W32_32BIT_English.ISO
4,380,387,328	Win10_1607_English_x64.iso
5,400,115,200	SW_DVD9_Windows_Svr_Std_and_DataCtr_2012_R2_64Bit_English.ISO

第27.3节：从根文件夹计算所有文件的哈希值

上述代码的另一种变体在你想获取根文件夹中所有文件的哈希码时提供了更高的性能


```
        bDone = True
    Else
        With oRnd
            .OpenFile sFile
            Do
                If uBytesRead + uBlockSize < uFileSize Then
                    uBytesToRead = uBlockSize
                Else
                    uBytesToRead = uFileSize - uBytesRead
                    bDone = True
                End If
                ' Read in some bytes
                aBytes = .ReadBytes(uBytesToRead)
                aBlock = aBytes
                If bDone Then
                    oCSP.TransformFinalBlock aBlock, 0, uBytesToRead
                    uBytesRead = uBytesRead + uBytesToRead
                Else
                    uBytesRead = uBytesRead + oCSP.TransformBlock(aBlock, 0, uBytesToRead, aBlock,
0)
                End If
                DoEvents
            Loop Until bDone
            .CloseFile
        End With
    End If
    If bDone Then
        ' convert Hash byte array to an hexadecimal string
        aHash = oCSP.hash
        For i = 0 To UBound(aHash)
            Mid$(sHash, i * 2 + (aHash(i) > 15) + 2) = Hex(aHash(i))
        Next
    End If
    Application.ScreenUpdating = True
    ' Clean up
    oCSP.Clear
CleanUp:
    Set oFS0 = Nothing
    Set oRnd = Nothing
    Set oCSP = Nothing
    GetFileHash = sHash
End Function
```

> The output is pretty interesting, my test files indicates that **`BLOCKSIZE = 131071` (2^17-1)** gives overall best performance with 32bit Office 2010 on Windows 7 x64, next best is *2^16-1 (65535)*. Note `2^27-1` yields *Out of memory*.

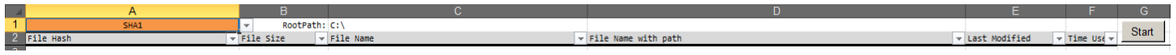
File Size (bytes)	File Name
146,800,640	clonezilla-live-2.2.2-37-amd64.iso
798,210,048	HPIP201.2014_0902.29.iso
2,073,016,320	SW_DVD5_Windows_Vista_Business_W32_32BIT_English.ISO
4,380,387,328	Win10_1607_English_x64.iso
5,400,115,200	SW_DVD9_Windows_Svr_Std_and_DataCtr_2012_R2_64Bit_English.ISO

Section 27.3: Calculating all Files Hash from a root Folder

Another variation from the code above gives you more performance when you want to get hash codes of all files

包括所有子文件夹。

工作表示例：

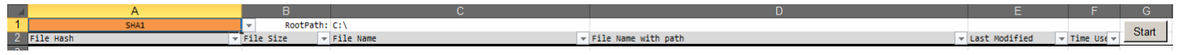


代码

```
Option Explicit

Private Const HashTypeMD5 As String = "MD5" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA1 As String = "SHA1" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA256 As String = "SHA256" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA384 As String = "SHA384" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA512 As String = "SHA512" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.110).aspx

Private Const BLOCKSIZE As Double = 131071 ' 2^17-1

Private oFSO As Object
Private oCSP As Object
Private oRnd As Random ' 需要来自 Microsoft 的类
https://support.microsoft.com/en-us/kb/189981
Private sHashType As String
Private sRootFDR As String
Private oRng As Range
Private uFileCount As Double

Sub AllFileHashes() ' Active-X 按钮调用此过程
    Dim oWS As Worksheet
    ' | A: 文件哈希 | B: 文件大小 | C: 文件名 | D: 文件名和路径 | E: 文件最后修改时间 | F: 计算哈希码所需时间（秒）

    With ThisWorkbook
        ' 清除所有工作表上的旧条目
        For Each oWS In .Worksheets
            Set oRng = Intersect(oWS.UsedRange, oWS.UsedRange.Offset(2))
            如果 oRng 不为空 则 oRng.ClearContents
        下一步
        使用 .Worksheets(1)
sHashType = Trim(.Range("A1").Value) ' A1单元格
            sRootFDR = Trim(.Range("C1").Value) ' C1单元格，B列为文件大小
            如果 Len(sHashType) = 0 或 Len(sRootFDR) = 0 则退出子程序
            设置 oRng = .Range("A3") ' 首页的第一个条目
        结束使用
    结束于

uFileCount = 0
    如果 oRnd 为空 则设置 oRnd = 新建 Random ' 微软类：Random
    如果 oFSO 为空 则设置 oFSO = CreateObject("Scripting.FileSystemObject") ' 仅用于获取正确的文件大小
    如果 oCSP 为空 则设置 oCSP = CreateObject("System.Security.Cryptography." & sHashType & "CryptoServiceProvider")
```


from a root folder including all sub folders.

Example of Worksheet:



Code

```
Option Explicit

Private Const HashTypeMD5 As String = "MD5" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.md5cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA1 As String = "SHA1" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha1cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA256 As String = "SHA256" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA384 As String = "SHA384" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha384cryptoserviceprovider(v=vs.110).aspx
Private Const HashTypeSHA512 As String = "SHA512" '
https://msdn.microsoft.com/en-us/library/system.security.cryptography.sha512cryptoserviceprovider(v=vs.110).aspx

Private Const BLOCKSIZE As Double = 131071 ' 2^17-1

Private oFSO As Object
Private oCSP As Object
Private oRnd As Random ' Requires the Class from Microsoft
https://support.microsoft.com/en-us/kb/189981
Private sHashType As String
Private sRootFDR As String
Private oRng As Range
Private uFileCount As Double

Sub AllFileHashes() ' Active-X button calls this
    Dim oWS As Worksheet
    ' | A: FileHash | B: FileSize | C: FileName | D: FilaName and Path | E: File Last Modification Time | F: Time required to calculate has code (seconds)
    With ThisWorkbook
        ' Clear All old entries on all worksheets
        For Each oWS In .Worksheets
            Set oRng = Intersect(oWS.UsedRange, oWS.UsedRange.Offset(2))
            If Not oRng Is Nothing Then oRng.ClearContents
        Next
        With .Worksheets(1)
            sHashType = Trim(.Range("A1").Value) ' Range(A1)
            sRootFDR = Trim(.Range("C1").Value) ' Range(C1) Column B for file size
            If Len(sHashType) = 0 Or Len(sRootFDR) = 0 Then Exit Sub
            Set oRng = .Range("A3") ' First entry on First Page
        End With
    End With

    uFileCount = 0
    If oRnd Is Nothing Then Set oRnd = New Random ' Class by Microsoft: Random
    If oFSO Is Nothing Then Set oFSO = CreateObject("Scripting.FileSystemObject") ' Just to get correct FileSize
    If oCSP Is Nothing Then Set oCSP = CreateObject("System.Security.Cryptography." & sHashType & "CryptoServiceProvider")
```

```
处理文件夹 oFSO.GetFolder(sRootFDR)

    Application.StatusBar = False
Application.ScreenUpdating = True
    oCSP.Clear
    Set oCSP = Nothing
    Set oRng = Nothing
    Set oFSO = Nothing
    Set oRnd = Nothing
Debug.Print "总文件数: " & uFileCount
End Sub

Private Sub ProcessFolder(ByRef oFDR As Object)
    Dim oFile As Object, oSubFDR As Object, sHash As String, dStart As Date, dFinish As Date
    Application.ScreenUpdating = False
    For Each oFile In oFDR.Files
uFileCount = uFileCount + 1
Application.StatusBar = uFileCount & ": " & Right(oFile.Path, 255 - Len(uFileCount) - 2)
        oCSP.Initialize '重新初始化 CryptoServiceProvider
        dStart = Now
sHash = GetFileHash(oFile, BLOCKSIZE, sHashType)
       dFinish = Now
        With oRng
            .Value = sHash
            .Offset(0, 1).Value = oFile.Size '文件大小（字节）
            .Offset(0, 2).Value = oFile.Name '带扩展名的文件名
            .Offset(0, 3).Value = oFile.Path '完整文件名和路径
            .Offset(0, 4).Value = FileDateTime(oFile.Path) '文件的最后修改时间戳
            .Offset(0, 5).Value = dFinish - dStart '计算哈希码所需时间
        结束于
        If oRng.Row = Rows.Count Then
            '达到最大行数，开始处理下一张工作表
            If oRng.Worksheet.Index + 1 > ThisWorkbook.Worksheets.Count Then
              MsgBox "所有工作表的所有行已用完，请创建更多工作表"
                End
            End If
            Set oRng = ThisWorkbook.Sheets(oRng.Worksheet.Index + 1).Range("A3")
          oRng.Worksheet.Activate
        Else
            '否则移动到下一行
            Set oRng = oRng.Offset(1)
        End If
    Next
    'Application.StatusBar = False
    Application.ScreenUpdating = True
   oRng.Activate
    对于每个 oSubFDR 在 oFDR.SubFolders
       处理文件夹 oSubFDR
    下一个
End Sub

Private Function GetFileHash(ByVal sFile As String, ByVal uBlockSize As Double, ByVal sHashType As
String) As String
    Dim uBytesRead 作为 Double, uBytesToRead 作为 Double, bDone 作为 Boolean
    Dim aBlock() 作为 Byte, aBytes 作为 Variant '用于存储字节的数组
    Dim aHash() 作为 Byte, sHash 作为 String, i 作为 Long, oTmp 作为 Variant
    Dim uFileSize 作为 Double '如果单独使用 GetFileHash()，取消注释

    如果 oRnd 是 Nothing 则设置 oRnd = 新建 Random '微软的类：Random
    如果 oFSO 是 Nothing 则设置 oFSO = 创建对象("Scripting.FileSystemObject") '仅用于获取
正确的文件大小
    如果 oCSP 是 Nothing 则设置 oCSP = 创建对象("System.Security.Cryptography." & sHashType &
```



```
    ProcessFolder oFSO.GetFolder(sRootFDR)

    Application.StatusBar = False
    Application.ScreenUpdating = True
    oCSP.Clear
    Set oCSP = Nothing
    Set oRng = Nothing
    Set oFSO = Nothing
    Set oRnd = Nothing
    Debug.Print "Total file count: " & uFileCount
End Sub

Private Sub ProcessFolder(ByRef oFDR As Object)
    Dim oFile As Object, oSubFDR As Object, sHash As String, dStart As Date, dFinish As Date
    Application.ScreenUpdating = False
    For Each oFile In oFDR.Files
        uFileCount = uFileCount + 1
        Application.StatusBar = uFileCount & ": " & Right(oFile.Path, 255 - Len(uFileCount) - 2)
        oCSP.Initialize ' Reinitialize the CryptoServiceProvider
        dStart = Now
        sHash = GetFileHash(oFile, BLOCKSIZE, sHashType)
        dFinish = Now
        With oRng
            .Value = sHash
            .Offset(0, 1).Value = oFile.Size ' File Size in bytes
            .Offset(0, 2).Value = oFile.Name ' File name with extension
            .Offset(0, 3).Value = oFile.Path ' Full File name and Path
            .Offset(0, 4).Value = FileDateTime(oFile.Path) ' Last modification timestamp of file
            .Offset(0, 5).Value = dFinish - dStart ' Time required to calculate hash code
        End With
        If oRng.Row = Rows.Count Then
            ' Max rows reached, start on Next sheet
            If oRng.Worksheet.Index + 1 > ThisWorkbook.Worksheets.Count Then
                MsgBox "All rows in all worksheets have been used, please create more sheets"
                End
            End If
            Set oRng = ThisWorkbook.Sheets(oRng.Worksheet.Index + 1).Range("A3")
            oRng.Worksheet.Activate
        Else
            ' Move to next row otherwise
            Set oRng = oRng.Offset(1)
        End If
    Next
    'Application.StatusBar = False
    Application.ScreenUpdating = True
    oRng.Activate
    For Each oSubFDR In oFDR.SubFolders
        ProcessFolder oSubFDR
    Next
End Sub

Private Function GetFileHash(ByVal sFile As String, ByVal uBlockSize As Double, ByVal sHashType As
String) As String
    Dim uBytesRead As Double, uBytesToRead As Double, bDone As Boolean
    Dim aBlock() As Byte, aBytes As Variant ' Arrays to store bytes
    Dim aHash() As Byte, sHash As String, i As Long, oTmp As Variant
    Dim uFileSize As Double ' Un-Comment if GetFileHash() is to be used individually

    If oRnd Is Nothing Then Set oRnd = New Random ' Class by Microsoft: Random
    If oFSO Is Nothing Then Set oFSO = CreateObject("Scripting.FileSystemObject") ' Just to get
correct FileSize
    If oCSP Is Nothing Then Set oCSP = CreateObject("System.Security.Cryptography." & sHashType &
```

```
"CryptoServiceProvider")

    如果 oFSO 是 Nothing 或 oRnd 是 Nothing 或 oCSP 是 Nothing 则
       弹出消息框 "无法创建一个或多个所需对象"
        退出函数
    End If

uFileSize = oFSO.GetFile(sFile).Size ' FILELEN() 最大支持2GB
    uBytesRead = 0
bDone = False
sHash = 字符串(oCSP.HashSize / 4, "0") ' 每个十六进制位为4位二进制

    ' 以 uBlockSize 或更小的块处理文件
    If uFileSize = 0 Then
        ReDim aBlock(0)
oCSP.TransformFinalBlock aBlock, 0, 0
        bDone = True
    Else
        With oRnd
            出错时转到 无法打开文件
            .OpenFile sFile
            Do
                如果 uBytesRead + uBlockSize < uFileSize 那么
                    uBytesToRead = uBlockSize
                Else
uBytesToRead = uFileSize - uBytesRead
                    bDone = True
                End If
                读取一些字节
aBytes = .ReadBytes(uBytesToRead)
                aBlock = aBytes
                如果 bDone 则
oCSP.TransformFinalBlock aBlock, 0, uBytesToRead
                uBytesRead = uBytesRead + uBytesToRead
                Else
                uBytesRead = uBytesRead + oCSP.TransformBlock(aBlock, 0, uBytesToRead, aBlock,
0)
                End If
DoEvents
            循环 直到 bDone
            .关闭文件
无法打开文件：
            如果 Err.Number <> 0 则 ' 将哈希码更改为错误描述
              oTmp = Split(Err.Description, vbCrLf)
              sHash = oTmp(1) & ":" & oTmp(2)
            End If
        结束于
    End If
    如果 bDone 则
        将哈希字节数组转换为十六进制字符串
        aHash = oCSP.hash
        对于 i = 0 到 UBound(aHash)
            Mid$(sHash, i * 2 + (aHash(i) > 15) + 2) = Hex(aHash(i))
        Next
    End If
    GetFileHash = sHash
End Function
```



```
"CryptoServiceProvider")

    If oFSO Is Nothing Or oRnd Is Nothing Or oCSP Is Nothing Then
        MsgBox "One or more required objects cannot be created"
        Exit Function
    End If

    uFileSize = oFSO.GetFile(sFile).Size ' FILELEN() has 2GB max
    uBytesRead = 0
    bDone = False
    sHash = String(oCSP.HashSize / 4, "0") ' Each hexadecimal is 4 bits

    ' Process the file in chunks of uBlockSize or less
    If uFileSize = 0 Then
        ReDim aBlock(0)
        oCSP.TransformFinalBlock aBlock, 0, 0
        bDone = True
    Else
        With oRnd
            On Error GoTo CannotOpenFile
            .OpenFile sFile
            Do
                If uBytesRead + uBlockSize < uFileSize Then
                    uBytesToRead = uBlockSize
                Else
                    uBytesToRead = uFileSize - uBytesRead
                    bDone = True
                End If
                ' Read in some bytes
                aBytes = .ReadBytes(uBytesToRead)
                aBlock = aBytes
                If bDone Then
                    oCSP.TransformFinalBlock aBlock, 0, uBytesToRead
                    uBytesRead = uBytesRead + uBytesToRead
                Else
                    uBytesRead = uBytesRead + oCSP.TransformBlock(aBlock, 0, uBytesToRead, aBlock,
0)
                End If
                DoEvents
            Loop Until bDone
            .CloseFile
CannotOpenFile:
            If Err.Number <> 0 Then ' Change the hash code to the Error description
                oTmp = Split(Err.Description, vbCrLf)
                sHash = oTmp(1) & ":" & oTmp(2)
            End If
        End With
    End If
    If bDone Then
        ' convert Hash byte array to an hexadecimal string
        aHash = oCSP.hash
        For i = 0 To UBound(aHash)
            Mid$(sHash, i * 2 + (aHash(i) > 15) + 2) = Hex(aHash(i))
        Next
    End If
    GetFileHash = sHash
End Function
```

第28章：创建过程

第28.1节：过程简介

Sub 是执行特定任务但不返回特定值的过程。

```
Sub ProcedureName ([argument_list])
    [statements]
End Sub
```

如果未指定访问修饰符，过程默认是Public。

Function 是一个接收数据并返回值的过程，理想情况下没有全局或模块范围的副作用。

```
Function ProcedureName ([argument_list]) [As ReturnType]
    [statements]
End Function
```

属性（Property）是一种封装模块数据的过程。属性最多可以有3个访问器：Get 用于返回一个值或对象引用，Let 用于赋值，和/或 Set 用于赋予对象引用。

```
Property Get|Let|Set PropertyName([argument_list]) [As ReturnType]
    [statements]
End Property
```

属性通常用于类模块（尽管标准模块中也允许使用），用于暴露调用代码无法访问的数据的访问器。仅暴露 Get 访问器的属性是“只读”的；仅暴露 Let 和/或 Set 访问器的属性是“只写”的。只写属性不被认为是良好的编程实践——如果客户端代码可以写入一个值，它也应该能够读取该值。

考虑实现一个 Sub 过程来替代只写属性。

返回值

函数（Function）或 Property Get 过程可以（且应该）向调用者返回一个值。这是通过给过程标识符赋值来完成的：

```
Property Get Foo() As Integer
    Foo = 42
End Property
```

第28.2节：带示例的函数

如上所述，函数是较小的过程，包含可能在过程内部重复的小代码片段。

函数用于减少代码冗余。

与过程类似，函数可以声明时带有或不带有参数列表。

函数被声明为有返回类型，因为所有函数都会返回一个值。函数的名称和返回变量是相同的。

Chapter 28: Creating a procedure

Section 28.1: Introduction to procedures

A `Sub` is a procedure that performs a specific task but does not return a specific value.

```
Sub ProcedureName ([argument_list])
    [statements]
End Sub
```

If no access modifier is specified, a procedure is `Public` by default.

A `Function` is a procedure that is given data and returns a value, ideally without global or module-scope side-effects.

```
Function ProcedureName ([argument_list]) [As ReturnType]
    [statements]
End Function
```

A `Property` is a procedure that *encapsulates* module data. A property can have up to 3 accessors: `Get` to return a value or object reference, `Let` to assign a value, and/or `Set` to assign an object reference.

```
Property Get|Let|Set PropertyName([argument_list]) [As ReturnType]
    [statements]
End Property
```

Properties are usually used in class modules (although they are allowed in standard modules as well), exposing accessor to data that is otherwise inaccessible to the calling code. A property that only exposes a `Get` accessor is "read-only"; a property that would only expose a `Let` and/or `Set` accessor is "write-only". Write-only properties are not considered a good programming practice - if the client code can *write* a value, it should be able to *read* it back. Consider implementing a `Sub` procedure instead of making a write-only property.

Returning a value

A `Function` or `Property Get` procedure can (and should!) return a value to its caller. This is done by assigning the identifier of the procedure:

```
Property Get Foo() As Integer
    Foo = 42
End Property
```

Section 28.2: Function With Examples

As stated above Functions are smaller procedures that contain small pieces of code which may be repetitive inside a Procedure.

Functions are used to reduce redundancy in code.

Similar to a Procedure, A function can be declared with or without an arguments list.

Function is declared as a return type, as all functions return a value. The Name and the Return Variable of a function are the Same.

1. 带参数的函数：

```
函数 check_even(i 作为整数) 作为布尔值
如果 (i 模 2) = 0 则
check_even = 真
否则
check_even=假
结束如果
结束函数
```

2. 无参数的函数：

```
函数 greet() 作为字符串
greet= "Hello Coder!"
结束函数
```

函数可以在另一个函数内部以多种方式调用。由于声明了返回类型的函数本质上是一个变量，因此它的使用类似于变量。

函数调用：

```
调用 greet() "类似于过程调用，只是允许过程使用
             "变量 greet
string_1=greet() "函数的返回值用于变量
                 赋值"
```

此外，函数还可以用作 if 语句和其他条件语句的条件。

```
for i = 1 到 10
if check_even(i) 则
msgbox i & " 是偶数"
else
msgbox i & " 是奇数"
end if
next i
```

此外，函数的参数还可以有 By ref 和 By val 等修饰符。

1. Function With Parameter:

```
Function check_even(i as integer) as boolean
if (i mod 2) = 0 then
check_even = True
else
check_even=False
end if
end Function
```

2. Function Without Parameter:

```
Function greet() as String
greet= "Hello Coder!"
end Function
```

The Function can be called in various ways inside a function. Since a Function declared with a return type is basically a variable. it is used similar to a variable.

Functional Calls:

```
call greet() 'Similar to a Procedural call just allows the Procedure to use the
             'variable greet
string_1=greet() 'The Return value of the function is used for variable
                 'assignment
```

Further the function can also be used as conditions for if and other conditional statements.

```
for i = 1 to 10
if check_even(i) then
msgbox i & " is Even"
else
msgbox i & " is Odd"
end if
next i
```

Further more Functions can have modifiers such as By ref and By val for their arguments.

第29章：过程调用

参数	信息
IdentifierName	要调用的过程名称。
参数	以逗号分隔的参数列表，将传递给过程。

第29.1节：这很令人困惑。为什么不总是使用括号呢？

括号用于括住*函数调用*的参数。将它们用于*过程调用*可能会导致
意想不到的问题。

因为它们可能引入错误，既可能在运行时通过传递可能非预期的值给过程引发错误，也可能在
编译时因语法无效而出错。

运行时

多余的括号可能引入错误。假设有一个过程接受对象引用作为参数……

```
Sub DoSomething(ByRef target As Range)
End Sub
```

……并用括号调用：

```
DoSomething (Application.ActiveCell) '在运行时引发错误
```

这将引发“需要对象”运行时错误 #424。在其他情况下可能会出现其他错误：这里
Application.ActiveCell Range对象引用正在被求值并按值传递，无论过程签名指定target将被ByRef传递。上述代码片段中
实际按ByVal传递给DoSomething的值是Application.ActiveCell.Value。

括号强制 VBA 计算括号内表达式的值，并将结果`ByVal`传递给被调用的
过程。当计算结果的类型与过程预期的类型不匹配且无法
隐式转换时，将引发运行时错误。

编译时

这段代码将无法编译：

```
MsgBox ("无效代码！", vbCritical)
```

因为表达式("无效代码！", vbCritical)无法被求值为一个值。

这将能够编译并运行：

```
MsgBox ("无效代码！"), (vbCritical)
```

但肯定会显得很傻。避免多余的括号。

第29.2节：隐式调用语法

```
过程名称
```


Chapter 29: Procedure Calls

Parameter	Info
IdentifierName	The name of the procedure to call.
arguments	A comma-separated list of arguments to be passed to the procedure.

Section 29.1: This is confusing. Why not just always use parentheses?

Parentheses are used to enclose the arguments of *function calls*. Using them for *procedure calls* can cause unexpected problems.

Because they can introduce bugs, both at run-time by passing a possibly unintended value to the procedure, and at compile-time by simply being invalid syntax.

Run-time

Redundant parentheses can introduce bugs. Given a procedure that takes an object reference as a parameter...

```
Sub DoSomething(ByRef target As Range)
End Sub
```

...and called with parentheses:

```
DoSomething (Application.ActiveCell) 'raises an error at runtime
```

This will raise an "Object Required" runtime error #424. Other errors are possible in other circumstances: here the `Application.ActiveCell` `Range` object reference is being *evaluated* and passed by value **regardless** of the procedure's signature specifying that `target` would be passed `ByRef`. The actual value passed `ByVal` to `DoSomething` in the above snippet, is `Application.ActiveCell.Value`.

Parentheses force VBA to evaluate the value of the bracketed expression, and pass the result `ByVal` to the called procedure. When the type of the evaluated result mismatches the procedure's expected type and cannot be implicitly converted, a runtime error is raised.

Compile-time

This code will fail to compile:

```
MsgBox ("Invalid Code!", vbCritical)
```

Because the expression `("Invalid Code!", vbCritical)` cannot be *evaluated* to a value.

This would compile and work:

```
MsgBox ("Invalid Code!"), (vbCritical)
```

But would definitely look silly. Avoid redundant parentheses.

Section 29.2: Implicit Call Syntax

```
ProcedureName
```

```
过程名称 参数1，参数2
```

通过名称调用过程时不使用任何括号。

边缘情况

只有在一种边缘情况下才需要使用Call关键字：

```
Call DoSomething : DoSomethingElse
```

`DoSomething` 和 `DoSomethingElse` 是被调用的过程。如果去掉`Call`关键字，那么 `DoSomething` 会被解析为*行标签*而不是过程调用，这将导致代码出错：

```
DoSomething: DoSomethingElse '只有 DoSomethingElse 会运行
```

第29.3节：可选参数

有些过程有可选参数。可选参数总是在必需参数之后，但
过程可以在没有它们的情况下被调用。

例如，如果函数ProcedureName有两个必需参数（argument1，argument2）和一个可选参数optArgument3，则调用方式至少有四种：

```
'无可选参数
result = ProcedureName("A", "B")

'有可选参数
result = ProcedureName("A", "B", "C")

'使用命名参数（允许不同顺序）
result = ProcedureName(optArgument3:="C", argument1:="A", argument2:="B")

'混合使用命名和未命名参数
result = ProcedureName("A", "B", optArgument3:="C")
```

这里调用的函数头结构大致如下：

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As String)
As String
```

可选（Optional）关键字表示该参数可以省略。如前所述——在头文件中引入的任何可选参数必须出现在所有必需参数之后的末尾。

您还可以为参数提供一个默认值，以防函数未传入该参数的值：

```
函数 ProcedureName(argument1 作为字符串, argument2 作为字符串, 可选 optArgument3 作为字符串 =
"C") 作为字符串
```

在此函数中，如果未提供c参数，则其值将默认为"C"。如果提供了值，则该值将覆盖默认值。

第29.4节：显式调用语法

```
调用 ProcedureName
```



```
ProcedureName argument1, argument2
```

Call a procedure by its name without any parentheses.

Edge case

The `Call` keyword is only required in one edge case:

```
Call DoSomething : DoSomethingElse
```

`DoSomething` and `DoSomethingElse` are procedures being called. If the `Call` keyword was removed, then `DoSomething` would be parsed as a *line label* rather than a procedure call, which would break the code:

```
DoSomething: DoSomethingElse 'only DoSomethingElse will run
```

Section 29.3: Optional Arguments

Some procedures have optional arguments. Optional arguments always come after required arguments, but the procedure can be called without them.

For example, if the function, `ProcedureName` were to have two required arguments (`argument1`, `argument2`), and one optional argument, `optArgument3`, it could be called at least four ways:

```
' Without optional argument
result = ProcedureName("A", "B")

' With optional argument
result = ProcedureName("A", "B", "C")

' Using named arguments (allows a different order)
result = ProcedureName(optArgument3:="C", argument1:="A", argument2:="B")

' Mixing named and unnamed arguments
result = ProcedureName("A", "B", optArgument3:="C")
```

The structure of the function header being called here would look something like this:

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As String)
As String
```

The `Optional` keyword indicates that this argument can be omitted. As mentioned before - any optional arguments introduced in the header **must** appear at the end, after any required arguments.

You can also provide a *default* value for the argument in the case that a value isn't passed to the function:

```
Function ProcedureName(argument1 As String, argument2 As String, Optional optArgument3 As String =
"C") As String
```

In this function, if the argument for c isn't supplied it's value will default to `"C"`. If a value *is* supplied then this will override the default value.

Section 29.4: Explicit Call Syntax

```
Call ProcedureName
```

```
调用 ProcedureName(argument1, argument2)
```

显式调用语法需要Call关键字和参数列表的括号；如果没有参数，括号是多余的。当VB添加了更现代的隐式调用语法后，此语法已被废弃。

第29.5节：返回值

要获取过程调用的结果（例如函数或属性获取过程），请将调用放在赋值语句的右侧：

```
result = ProcedureName
result = ProcedureName(argument1, argument2)
```

如果有参数，必须带有括号。如果过程没有参数，括号是多余的。


```
Call ProcedureName(argument1, argument2)
```

The explicit call syntax requires the `Call` keyword and parentheses around the argument list; parentheses are redundant if there are no parameters. This syntax was made obsolete when the more modern implicit call syntax was added to VB.

Section 29.5: Return Values

To retrieve the result of a procedure call (e.g. `Function` or `Property Get` procedures), put the call on the right-hand side of an assignment:

```
result = ProcedureName
result = ProcedureName(argument1, argument2)
```

Parentheses must be present if there are parameters. If the procedure has no parameters, the parentheses are redundant.

第30章：条件编译

第30.1节：在编译时改变代码行为

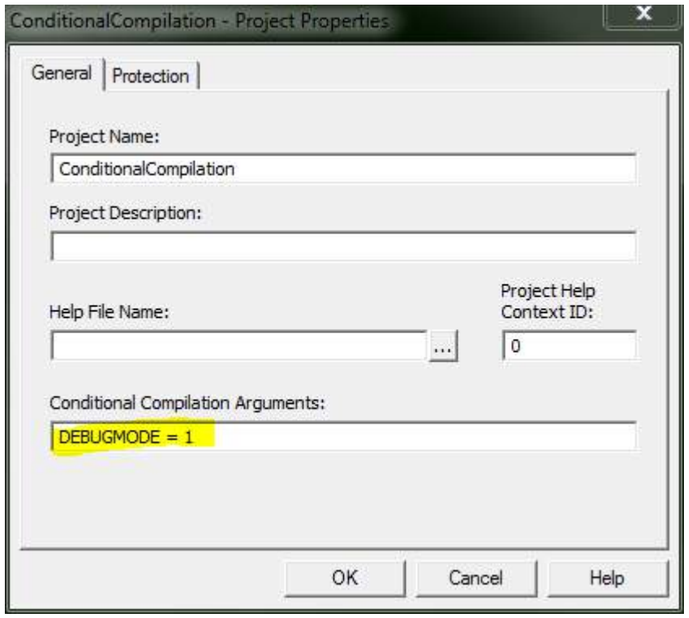
#Const 指令用于定义自定义的预处理器常量。之后可以通过 #If 来控制哪些代码块被编译和执行。

```
#Const DEBUGMODE = 1

#If DEBUGMODE Then
    Const filepath As String = "C:\Users\UserName\Path\To\File.txt"
#Else
    Const filepath As String = "\\server\share\patho\file.txt"
#End If
```

这将导致 filepath 的值被设置为 "C:\Users\UserName\Path\To\File.txt"。移除 #Const 行，或将其改为 #Const DEBUGMODE = 0 会导致 filepath 被设置为 "\\server\share\patho\file.txt"。

#Const 范围

#Const 指令仅对单个代码文件（模块或类）有效。您必须在每个希望使用自定义常量的文件中声明它。或者，您可以通过进入“工具 >> [您的项目名称] 项目属性”来为整个项目全局声明一个 #Const 。这将弹出项目属性对话框，我们将在其中输入常量声明。在 “条件编译参数” 框中，输入 [constName] = [value]。您可以通过用冒号分隔来输入多个常量，例如 [constName1] = [value1] : [constName2] = [value2]。



预定义常量

一些编译常量已经预定义。具体有哪些取决于您运行 VBA 的 Office 版本的位数。请注意，Vba7 是随着 Office 2010 一起引入的，用于支持 Office 的 64 位版本。

Chapter 30: Conditional Compilation

Section 30.1: Changing code behavior at compile time

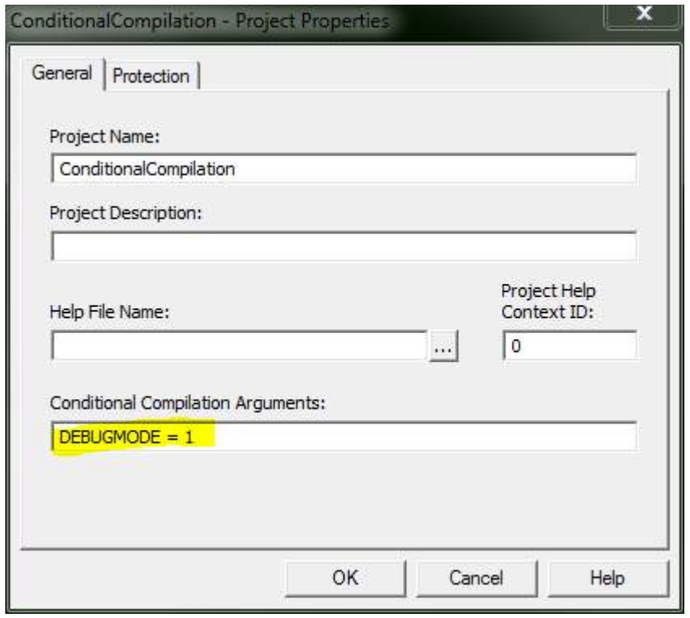
The `#Const` directive is used to define a custom preprocessor constant. These can later be used by `#If` to control which blocks of code get compiled and executed.

```
#Const DEBUGMODE = 1

#If DEBUGMODE Then
    Const filepath As String = "C:\Users\UserName\Path\To\File.txt"
#Else
    Const filepath As String = "\\server\share\path\to\file.txt"
#End If
```

This results in the value of `filepath` being set to `"C:\Users\UserName\Path\To\File.txt"`. Removing the `#Const` line, or changing it to `#Const DEBUGMODE = 0` would result in the `filepath` being set to `"\\server\share\path\to\file.txt"`.

#Const Scope

The `#Const` directive is only effective for a single code file (module or class). It must be declared for each and every file you wish to use your custom constant in. Alternatively, you can declare a `#Const` globally for your project by going to Tools >> [Your Project Name] Project Properties. This will bring up the project properties dialog box where we'll enter the constant declaration. In the “Conditional Compilation Arguments” box, type in `[constName] = [value]`. You can enter more than 1 constant by separating them with a colon, like `[constName1] = [value1] : [constName2] = [value2]`.



Pre-defined Constants

Some compilation constants are already pre-defined. Which ones exist will depend on the bitness of the office version you're running VBA in. Note that Vba7 was introduced alongside Office 2010 to support 64 bit versions of Office.

常量	16 位	32 位	64 位
Vba6	False 如果是 Vba6 则为 False		
Vba7	False 如果是 Vba7 则为 True		
Win16	True	False	False
Win32	False	True	True
Win64	False	False	True
Mac	False	If Mac	If Mac

请注意，Win64/Win32 指的是 Office 版本，而不是 Windows 版本。例如，即使操作系统是 64 位版本的 Windows，32 位 Office 中 Win32 = TRUE。

第30.2节：使用适用于所有 Office 版本的 Declare 导入

```
#If Vba7 Then
    "首先检查 Win64 非常重要，
    因为当 Win64 返回 true 时，Win32 也会返回 true。"

    #If Win64 Then
        Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
    #Else
        Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
    #End If
#Else
    ' 必须是 Vba6，当时没有 PtrSafe 关键字，
    ' 所以我们需要以不同于上面的方法声明 Win32 导入。

    #If Win32 Then
        Declare Function GetFoo Lib "exampleLib32"() As Long
    #Else
        Declare Function GetFoo Lib "exampleLib"() As Integer
    #End If
#End If
```

这可以根据你需要支持的 Office 版本稍作简化。例如，现在很少有人还支持 16 位版本的 Office。16 位 Office 的最后一个版本是 4.3 版，于 1994年，因此以下声明对于几乎所有现代情况（包括Office 2007）都足够了。

```
#If Vba7 Then
    "首先检查 Win64 非常重要，
    因为当 Win64 返回 true 时，Win32 也会返回 true。"

    #If Win64 Then
        Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
    #Else
        Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
    #End If
#Else
    必须是Vba6。我们不支持16位Office，因此必须是Win32。

    Declare Function GetFoo Lib "exampleLib32"() As Long
#End If
```

如果不需要支持早于Office 2010的版本，这个声明就完全可用。

```
我们只有2010版本的安装，因此我们已经知道使用的是Vba7。
```


Constant	16 bit	32 bit	64 bit
Vba6	False	If Vba6	False
Vba7	False	If Vba7	True
Win16	True	False	False
Win32	False	True	True
Win64	False	False	True
Mac	False	If Mac	If Mac

Note that Win64/Win32 refer to the Office version, not the Windows version. For example Win32 = TRUE in 32-bit Office, even if the OS is a 64-bit version of Windows.

Section 30.2: Using Declare Imports that work on all versions of Office

```
#If Vba7 Then
    ' It's important to check for Win64 first,
    ' because Win32 will also return true when Win64 does.

    #If Win64 Then
        Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
    #Else
        Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
    #End If
#Else
    ' Must be Vba6, the PtrSafe keyword didn't exist back then,
    ' so we need to declare Win32 imports a bit differently than above.

    #If Win32 Then
        Declare Function GetFoo Lib "exampleLib32"() As Long
    #Else
        Declare Function GetFoo Lib "exampleLib"() As Integer
    #End If
#End If
```

This can be simplified a bit depending on what versions of office you need to support. For example, not many people are still supporting 16 bit versions of Office. The last version of 16 bit office was version 4.3, released in 1994, so the following declaration is sufficient for nearly all modern cases (including Office 2007).

```
#If Vba7 Then
    ' It's important to check for Win64 first,
    ' because Win32 will also return true when Win64 does.

    #If Win64 Then
        Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
    #Else
        Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
    #End If
#Else
    ' Must be Vba6. We don't support 16 bit office, so must be Win32.

    Declare Function GetFoo Lib "exampleLib32"() As Long
#End If
```

If you don't have to support anything older than Office 2010, this declaration works just fine.

```
' We only have 2010 installs, so we already know we have Vba7.
```

```
#If Win64 Then
    Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
#Else
    Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
#End If
```



```
#If Win64 Then
    Declare PtrSafe Function GetFoo64 Lib "exampleLib32" () As LongLong
#Else
    Declare PtrSafe Function GetFoo Lib "exampleLib32" () As Long
#End If
```

第31章：面向对象的VBA

第31.1节：抽象

抽象层次有助于确定何时进行拆分。

抽象是通过用越来越详细的代码实现功能来实现的。宏的入口点
应该是一个具有*高抽象层次*的小过程，使人一眼就能轻松理解正在发生的事情：

```
Public Sub DoSomething()
    With New SomeForm
        设置.Model = CreateViewModel
        .Show vbModal
        如果.IsCancelled则退出子程序
       处理用户数据 .Model
    结束 With
End Sub
```

这个DoSomething过程具有较高的*抽象层次*：我们可以看出它正在显示一个表单并创建某个
模型，并将该对象传递给某个ProcessUserData过程，该过程知道如何处理它——模型
是如何创建的则是另一个过程的工作：

```
私有函数 CreateViewModel() 作为 ISomeModel
    定义 result 作为 ISomeModel
    设置 result = SomeModel.Create(Now, Environ$("UserName"))
    result.AvailableItems = GetAvailableItems
    设置 CreateViewModel = result
结束函数
```

函数CreateViewModel仅负责创建某个ISomeModel实例。其职责的一部分
是获取一个*可用项目*数组——这些项目如何获取是一个实现细节，已被抽象
隐藏在GetAvailableItems过程之后：

```
私有函数 GetAvailableItems() 作为 Variant
    GetAvailableItems = DataSheet.Names("AvailableItems").RefersToRange
结束函数
```

这里该过程是从DataSheet工作表上的一个命名范围读取可用值。它也可以从数据库读取，或者这些值可以是硬编码的：
这是一个实现细节，对任何更高的抽象层次都无关紧要。

第31.2节：封装

封装将实现细节隐藏于客户端代码之外。

处理 QueryClose 示例演示了封装：表单有一个复选框控件，但其客户端代码并不直接操作它——复选框是一个实
现细节，客户端代码需要知道的是设置是否启用。

当复选框的值发生变化时，处理程序会给一个私有字段成员赋值：

```
私有 类型 TView
IsCancelled 作为 布尔值
    SomeOtherSetting 作为 布尔值
    '其他属性为简洁起见省略
```


Chapter 31: Object-Oriented VBA

Section 31.1: Abstraction

Abstraction levels help determine when to split things up.

Abstraction is achieved by implementing functionality with increasingly detailed code. The entry point of a macro should be a small procedure with a *high abstraction level* that makes it easy to grasp at a glance what's going on:

```
Public Sub DoSomething()
    With New SomeForm
        Set .Model = CreateViewModel
        .Show vbModal
        If .IsCancelled Then Exit Sub
        ProcessUserData .Model
    End With
End Sub
```

The DoSomething procedure has a high *abstraction level*: we can tell that it's displaying a form and creating some model, and passing that object to some ProcessUserData procedure that knows what to do with it - how the model is created is the job of another procedure:

```
Private Function CreateViewModel() As ISomeModel
    Dim result As ISomeModel
    Set result = SomeModel.Create(Now, Environ$("UserName"))
    result.AvailableItems = GetAvailableItems
    Set CreateViewModel = result
End Function
```

The CreateViewModel function is only responsible for creating some ISomeModel instance. Part of that responsibility is to acquire an array of *available items* - how these items are acquired is an implementation detail that's abstracted behind the GetAvailableItems procedure:

```
Private Function GetAvailableItems() As Variant
    GetAvailableItems = DataSheet.Names("AvailableItems").RefersToRange
End Function
```

Here the procedure is reading the available values from a named range on a DataSheet worksheet. It could just as well be reading them from a database, or the values could be hard-coded: it's an *implementation detail* that's none of a concern for any of the higher abstraction levels.

Section 31.2: Encapsulation

Encapsulation hides implementation details from client code.

The Handling QueryClose example demonstrates encapsulation: the form has a checkbox control, but its client code doesn't work with it directly - the checkbox is an *implementation detail*, what the client code needs to know is whether the setting is enabled or not.

When the checkbox value changes, the handler assigns a private field member:

```
Private Type TView
    IsCancelled As Boolean
    SomeOtherSetting As Boolean
    'other properties skipped for brievety
```

```
结束 类型
私有 this 作为 TView

'...

私有子程序 SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub
```

当客户端代码想读取该值时，不需要关心复选框——它只需简单地使用SomeOtherSetting属性：

```
公共属性获取 SomeOtherSetting() 作为 布尔值
    SomeOtherSetting = this.SomeOtherSetting
结束属性
```

SomeOtherSetting属性封装了复选框的状态；客户端代码不需要知道有复选框的存在，只需知道有一个布尔值的设置。通过封装该布尔值，我们为复选框添加了一个抽象层。

使用接口来强制不可变性

让我们更进一步，通过封装表单的模型到一个专用的类模块中。但如果我们做了一个对于UserName和Timestamp的公共属性，我们必须公开Property Let访问器，使属性可变，但我们不希望客户端代码在设置后有能力更改这些值。

在Abstraction示例中，CreateViewModel函数返回一个ISomeModel类：这就是我们的接口，其结构大致如下：

```
Option Explicit

Public Property Get Timestamp() As Date
End Property

Public Property Get UserName() As String
End Property

Public Property Get AvailableItems() As Variant
End Property

Public Property Let AvailableItems(ByRef value As Variant)
End Property

Public Property Get SomeSetting() As String
End Property

Public Property Let SomeSetting(ByVal value As String)
End Property

Public Property Get SomeOtherSetting() As Boolean
End Property

Public Property Let SomeOtherSetting(ByVal value As Boolean)
End Property
```

Notice `Timestamp` and `UserName` properties only expose a `Property Get` accessor. Now the `SomeModel` class can implement that interface:


```
End Type
Private this As TView

'...

Private Sub SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub
```

And when the client code wants to read that value, it doesn't need to worry about a checkbox - instead it simply uses the `SomeOtherSetting` property:

```
Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property
```

The `SomeOtherSetting` property *encapsulates* the checkbox' state; client code doesn't need to know that there's a checkbox involved, only that there's a setting with a Boolean value. By *encapsulating* the `Boolean` value, we've added an *abstraction layer* around the checkbox.

Using interfaces to enforce immutability

Let's push that a step further by *encapsulating* the form's *model* in a dedicated class module. But if we made a `Public Property` for the `UserName` and `Timestamp`, we would have to expose `Property Let` accessors, making the properties mutable, and we don't want the client code to have the ability to change these values after they're set.

The `CreateViewModel` function in the **Abstraction** example returns an `ISomeModel` class: that's our *interface*, and it looks something like this:

```
Option Explicit

Public Property Get Timestamp() As Date
End Property

Public Property Get UserName() As String
End Property

Public Property Get AvailableItems() As Variant
End Property

Public Property Let AvailableItems(ByRef value As Variant)
End Property

Public Property Get SomeSetting() As String
End Property

Public Property Let SomeSetting(ByVal value As String)
End Property

Public Property Get SomeOtherSetting() As Boolean
End Property

Public Property Let SomeOtherSetting(ByVal value As Boolean)
End Property
```

Notice `Timestamp` and `UserName` properties only expose a `Property Get` accessor. Now the `SomeModel` class can implement that interface:

```
Option Explicit
Implements ISomeModel

Private Type TModel
    Timestamp As Date
    UserName As String
    SomeSetting As String
    SomeOtherSetting As Boolean
    AvailableItems As Variant
End Type
Private this As TModel

Private Property Get ISomeModel_Timestamp() As Date
    ISomeModel_Timestamp = this.Timestamp
End Property

Private Property Get ISomeModel_UserName() As String
    ISomeModel_UserName = this.UserName
End Property

Private Property Get ISomeModel_AvailableItems() As Variant
    ISomeModel_AvailableItems = this.AvailableItems
End Property

Private Property Let ISomeModel_AvailableItems(ByRef value As Variant)
    this.AvailableItems = value
End Property

Private Property Get ISomeModel_SomeSetting() As String
    ISomeModel_SomeSetting = this.SomeSetting
End Property

私有属性 Let ISomeModel_SomeSetting(ByVal value As String)
    this.SomeSetting = value
End Property

Private Property Get ISomeModel_SomeOtherSetting() As Boolean
    ISomeModel_SomeOtherSetting = this.SomeOtherSetting
End Property

Private Property Let ISomeModel_SomeOtherSetting(ByVal value As Boolean)
    this.SomeOtherSetting = value
End Property

Public Property Get Timestamp() As Date
    Timestamp = this.Timestamp
End Property

Public Property Let Timestamp(ByVal value As Date)
    this.Timestamp = value
End Property

Public Property Get UserName() As String
    UserName = this.UserName
End Property

Public Property Let UserName(ByVal value As String)
    this.UserName = value
End Property

Public Property Get AvailableItems() As Variant
    AvailableItems = this.AvailableItems
```



```
Option Explicit
Implements ISomeModel

Private Type TModel
    Timestamp As Date
    UserName As String
    SomeSetting As String
    SomeOtherSetting As Boolean
    AvailableItems As Variant
End Type
Private this As TModel

Private Property Get ISomeModel_Timestamp() As Date
    ISomeModel_Timestamp = this.Timestamp
End Property

Private Property Get ISomeModel_UserName() As String
    ISomeModel_UserName = this.UserName
End Property

Private Property Get ISomeModel_AvailableItems() As Variant
    ISomeModel_AvailableItems = this.AvailableItems
End Property

Private Property Let ISomeModel_AvailableItems(ByRef value As Variant)
    this.AvailableItems = value
End Property

Private Property Get ISomeModel_SomeSetting() As String
    ISomeModel_SomeSetting = this.SomeSetting
End Property

Private Property Let ISomeModel_SomeSetting(ByVal value As String)
    this.SomeSetting = value
End Property

Private Property Get ISomeModel_SomeOtherSetting() As Boolean
    ISomeModel_SomeOtherSetting = this.SomeOtherSetting
End Property

Private Property Let ISomeModel_SomeOtherSetting(ByVal value As Boolean)
    this.SomeOtherSetting = value
End Property

Public Property Get Timestamp() As Date
    Timestamp = this.Timestamp
End Property

Public Property Let Timestamp(ByVal value As Date)
    this.Timestamp = value
End Property

Public Property Get UserName() As String
    UserName = this.UserName
End Property

Public Property Let UserName(ByVal value As String)
    this.UserName = value
End Property

Public Property Get AvailableItems() As Variant
    AvailableItems = this.AvailableItems
```

```
End Property

Public Property Let AvailableItems(ByRef value As Variant)
    this.AvailableItems = value
End Property

Public Property Get SomeSetting() As String
    SomeSetting = this.SomeSetting
End Property

Public Property Let SomeSetting(ByVal value As String)
    this.SomeSetting = value
End Property

公共属性获取 SomeOtherSetting() 作为 布尔值
    SomeOtherSetting = this.SomeOtherSetting
结束属性

Public Property Let SomeOtherSetting(ByVal value As Boolean)
    this.SomeOtherSetting = value
End Property
```

接口成员全部是Private，且必须实现接口的所有成员，代码才能编译。那些Public成员不属于接口，因此不会暴露给针对ISomeModel接口编写的代码。

使用工厂方法模拟构造函数

通过使用 VB_PredeclaredId 属性，我们可以让SomeModel类拥有一个*默认实例*，并编写一个函数，该函数像类型级别的成员（VB.NET 中的`Shared`，C# 中的`static`）一样，客户端代码可以调用它而无需先创建实例，就像我们这里所做的：

```
私有函数 CreateViewModel() 作为 ISomeModel
    定义 result 作为 ISomeModel
    设置 result = SomeModel.Create(Now, Environ$("UserName"))
    result.AvailableItems = GetAvailableItems
    设置 CreateViewModel = result
结束函数
```

这个*工厂方法*为从ISomeModel接口访问时只读的属性赋值，这里是Timestamp和UserName：

```
Public Function Create(ByVal pTimeStamp As Date, ByVal pUserName As String) As ISomeModel
    With New SomeModel
        .Timestamp = pTimeStamp
        .UserName = pUserName
        Set Create = .Self
    结束于
End Function

Public Property Get Self() As ISomeModel
    Set Self = Me
End Property
```

现在我们可以针对ISomeModel接口进行编码，该接口将Timestamp和UserName作为只读属性暴露，且这些属性永远不能被重新赋值（只要代码是针对该接口编写的）。


```
End Property

Public Property Let AvailableItems(ByRef value As Variant)
    this.AvailableItems = value
End Property

Public Property Get SomeSetting() As String
    SomeSetting = this.SomeSetting
End Property

Public Property Let SomeSetting(ByVal value As String)
    this.SomeSetting = value
End Property

Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property

Public Property Let SomeOtherSetting(ByVal value As Boolean)
    this.SomeOtherSetting = value
End Property
```

The interface members are all `Private`, and all members of the interface must be implemented for the code to compile. The `Public` members are not part of the interface, and are therefore not exposed to code written against the `ISomeModel` interface.

Using a Factory Method to simulate a constructor

Using a VB_PredeclaredId attribute, we can make the `SomeModel` class have a *default instance*, and write a function that works like a type-level (`Shared` in VB.NET, `static` in C#) member that the client code can call without needing to first create an instance, like we did here:

```
Private Function CreateViewModel() As ISomeModel
    Dim result As ISomeModel
    Set result = SomeModel.Create(Now, Environ$("UserName"))
    result.AvailableItems = GetAvailableItems
    Set CreateViewModel = result
End Function
```

This *factory method* assigns the property values that are read-only when accessed from the `ISomeModel` interface, here `Timestamp` and `UserName`:

```
Public Function Create(ByVal pTimeStamp As Date, ByVal pUserName As String) As ISomeModel
    With New SomeModel
        .Timestamp = pTimeStamp
        .UserName = pUserName
        Set Create = .Self
    End With
End Function

Public Property Get Self() As ISomeModel
    Set Self = Me
End Property
```

And now we can code against the `ISomeModel` interface, which exposes `Timestamp` and `UserName` as read-only properties that can never be reassigned (as long as the code is written against the interface).

第31.3节：多态性

多态性是为不同的底层实现提供相同接口的能力。

实现接口的能力允许完全将应用逻辑与用户界面、数据库或某个工作表解耦。

假设你有一个ISomeView接口，表单本身实现了该接口：

```
Option Explicit

Public Property Get IsCancelled() As Boolean
End Property

Public Property Get Model() As ISomeModel
End Property

Public Property Set Model(ByVal value As ISomeModel)
End Property

Public Sub Show()
End Sub
```

表单的后台代码可能如下所示：

```
Option Explicit
Implements ISomeView

Private Type TView
    IsCancelled As Boolean
    Model As ISomeModel
End Type
私有 这个 作为 TView

私有属性获取 ISomeView_IsCancelled() 作为 布尔值
    ISomeView_IsCancelled = this.IsCancelled
End Property

私有属性获取 ISomeView_Model() 作为 ISomeModel
    设置 ISomeView_Model = this.Model
End Property

私有属性设置 ISomeView_Model(ByVal 值 作为 ISomeModel)
    设置 this.Model = 值
End Property

私有子程序 ISomeView_Show()
    Me.Show vbModal
End Sub

私有子程序 SomeOtherSettingInput_Change()
    this.Model.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
结束子程序

'...其他事件处理程序...

私有子程序 OkButton_Click()
    Me.Hide
End Sub
```


Section 31.3: Polymorphism

Polymorphism is the ability to present the same interface for different underlying implementations.

The ability to implement interfaces allows completely decoupling the application logic from the UI, or from the database, or from this or that worksheet.

Say you have an `ISomeView` interface that the form itself implements:

```
Option Explicit

Public Property Get IsCancelled() As Boolean
End Property

Public Property Get Model() As ISomeModel
End Property

Public Property Set Model(ByVal value As ISomeModel)
End Property

Public Sub Show()
End Sub
```

The form's code-behind could look like this:

```
Option Explicit
Implements ISomeView

Private Type TView
    IsCancelled As Boolean
    Model As ISomeModel
End Type
Private this As TView

Private Property Get ISomeView_IsCancelled() As Boolean
    ISomeView_IsCancelled = this.IsCancelled
End Property

Private Property Get ISomeView_Model() As ISomeModel
    Set ISomeView_Model = this.Model
End Property

Private Property Set ISomeView_Model(ByVal value As ISomeModel)
    Set this.Model = value
End Property

Private Sub ISomeView_Show()
    Me.Show vbModal
End Sub

Private Sub SomeOtherSettingInput_Change()
    this.Model.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub

'...other event handlers...

Private Sub OkButton_Click()
    Me.Hide
End Sub
```

```
私有子程序 CancelButton_Click()
    this.IsCancelled = True
    我隐藏
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
Cancel = True
        this.IsCancelled = True
        Me.Hide
    结束 如果
结束 子程序
```

但是，没有任何东西禁止创建另一个实现ISomeView接口的类模块，且不必是*用户窗体*——这可以是一个SomeViewMock类：

```
Option Explicit
Implements ISomeView

Private Type TView
    IsCancelled As Boolean
    Model As ISomeModel
End Type
私有 这个 作为 TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

Public Property Let IsCancelled(ByVal value As Boolean)
    this.IsCancelled = value
End Property

私有属性获取 ISomeView_IsCancelled() 作为 布尔值
    ISomeView_IsCancelled = this.IsCancelled
End Property

私有属性获取 ISomeView_Model() 作为 ISomeModel
    设置 ISomeView_Model = this.Model
End Property

私有属性设置 ISomeView_Model(ByVal 值 作为 ISomeModel)
    设置 this.Model = 值
End Property

Private Sub ISomeView_Show()
    '什么也不做
End Sub
```

现在我们可以修改处理UserForm的代码，使其基于ISomeView接口工作，例如通过将窗体作为参数传入，而不是实例化它：

```
Public Sub DoSomething(ByVal view As ISomeView)
    With view
        Set .Model = CreateViewModel
        .Show
        If .IsCancelled Then Exit Sub
       ProcessUserData .Model
    End With
End Sub
```



```
Private Sub CancelButton_Click()
    this.IsCancelled = True
    Me.Hide
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
        Cancel = True
        this.IsCancelled = True
        Me.Hide
    End If
End Sub
```

But then, nothing forbids creating another class module that implements the ISomeView interface *without being a user form* - this could be a SomeViewMock class:

```
Option Explicit
Implements ISomeView

Private Type TView
    IsCancelled As Boolean
    Model As ISomeModel
End Type
Private this As TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

Public Property Let IsCancelled(ByVal value As Boolean)
    this.IsCancelled = value
End Property

Private Property Get ISomeView_IsCancelled() As Boolean
    ISomeView_IsCancelled = this.IsCancelled
End Property

Private Property Get ISomeView_Model() As ISomeModel
    Set ISomeView_Model = this.Model
End Property

Private Property Set ISomeView_Model(ByVal value As ISomeModel)
    Set this.Model = value
End Property

Private Sub ISomeView_Show()
    'do nothing
End Sub
```

And now we can change the code that works with a UserForm and make it work off the ISomeView interface, e.g. by giving it the form as a parameter instead of instantiating it:

```
Public Sub DoSomething(ByVal view As ISomeView)
    With view
        Set .Model = CreateViewModel
        .Show
        If .IsCancelled Then Exit Sub
        ProcessUserData .Model
    End With
End Sub
```

因为DoSomething方法依赖于接口（即*抽象*），而不是*具体类*（例如特定的UserForm），我们可以编写自动化单元测试，确保当view.IsCancelled为True时，ProcessUserData不会被执行，方法是让测试创建一个SomeViewMock实例，将其IsCancelled属性设置为`True`，并将其传递给DoSomething。

可测试的代码依赖于抽象

在VBA中可以编写单元测试，市面上有一些插件甚至将其集成到IDE中。但当代码与工作表、数据库、窗体或文件系统紧密耦合时，单元测试就需要一个实际的工作表、数据库、窗体或文件系统——而这些依赖项是新的失控故障点，测试代码应当隔离这些依赖项，以便单元测试不需要实际的工作表、数据库、窗体或文件系统。

通过针对接口编写代码，使测试代码能够注入存根/模拟实现（如上文的SomeViewMock示例），你可以在“受控环境”中编写测试，模拟表单数据上42种可能的用户交互排列组合中的每一种，而无需实际显示表单并手动点击表单控件。

Because the `DoSomething` method depends on an interface (i.e. an *abstraction*) and not a *concrete class* (e.g. a specific `UserForm`), we can write an automated unit test that ensures that `ProcessUserData` isn't executed when `view.IsCancelled` is `True`, by making our test create a `SomeViewMock` instance, setting its `IsCancelled` property to `True`, and passing it to `DoSomething`.

Testable code depends on abstractions

Writing unit tests in VBA can be done, there are add-ins out there that even integrate it into the IDE. But when code is *tightly coupled* with a worksheet, a database, a form, or the file system, then the unit test starts requiring an actual worksheet, database, form, or file system - and these *dependencies* are new out-of-control failure points that testable code should isolate, so that unit tests *don't* require an actual worksheet, database, form, or file system.

By writing code against interfaces, in a way that allows test code to *inject* stub/mock implementations (like the above `SomeViewMock` example), you can write tests in a "controlled environment", and simulate what happens when every single one of the 42 possible permutations of user interactions on the form's data, without even once displaying a form and manually clicking on a form control.

第32章：创建自定义类

第32.1节：向类添加属性

Property过程是一系列语句，用于检索或修改模块上的自定义属性。

属性访问器有三种类型：

1. Get过程用于返回属性的值。
2. 一个Let过程，将（非Object）值赋给对象。
3. 一个Set过程，赋予Object引用。

属性访问器通常成对定义，对每个属性同时使用Get和Let/Set。只有Get过程的属性为只读，只有Let/Set过程的属性为只写。

在下面的示例中，为DateRange类定义了四个属性访问器：

1. StartDate（读/写）。表示范围中较早日期的日期值。每个过程都使用模块变量mStartDate的值。
   模块变量mStartDate的值。
2. EndDate（读/写）。表示范围中较晚日期的日期值。每个过程都使用模块变量mEndDate的值。
   模块变量mEndDate的值。
3. DaysBetween（只读）。计算的整数值，表示两个日期之间的天数。
   由于只有一个Get过程，此属性不能被直接修改。
4. RangeToCopy（只写）。一个Set过程，用于复制现有DateRange对象的值。

```
Private mStartDate As Date                  '模块变量，用于保存起始日期
Private mEndDate As Date                    '模块变量，用于保存结束日期

'返回当前的起始日期值
Public Property Get StartDate() As Date
    StartDate = mStartDate
End Property

'设置起始日期值。注意有两种方法名为StartDate
Public Property Let StartDate(ByVal NewValue As Date)
    mStartDate = NewValue
End Property

'同样的操作，但用于结束日期
Public Property Get EndDate() As Date
    EndDate = mEndDate
End Property

Public Property Let EndDate(ByVal NewValue As Date)
    mEndDate = NewValue
End Property

只读属性，返回两个日期之间的天数
Public Property Get DaysBetween() As Integer
    DaysBetween = DateDiff("d", mStartDate, mEndDate)
End Function

只写属性，传递一个范围对象引用以进行克隆
Public Property Set RangeToCopy(ByRef ExistingRange As DateRange)

Me.StartDate = ExistingRange.StartDate
Me.EndDate = ExistingRange.EndDate
```


Chapter 32: Creating a Custom Class

Section 32.1: Adding a Property to a Class

A `Property` procedure is a series of statement that retrieves or modifies a custom property on a module.

There are three types of property accessors:

1. A `Get` procedure that returns the value of a property.
2. A `Let` procedure that assigns a (non-`Object`) value to an object.
3. A `Set` procedure that assigns an `Object` reference.

Property accessors are often defined in pairs, using both a `Get` and `Let`/`Set` for each property. A property with only a `Get` procedure would be read-only, while a property with only a `Let`/`Set` procedure would be write-only.

In the following example, four property accessors are defined for the `DateRange` class:

1. `StartDate` (*read/write*). Date value representing the earlier date in a range. Each procedure uses the value of the module variable, `mStartDate`.
2. `EndDate` (*read/write*). Date value representing the later date in a range. Each procedure uses the value of the module variable, `mEndDate`.
3. `DaysBetween` (*read-only*). Calculated Integer value representing the number of days between the two dates. Because there is only a `Get` procedure, this property cannot be modified directly.
4. `RangeToCopy` (*write-only*). A `Set` procedure used to copy the values of an existing `DateRange` object.

```
Private mStartDate As Date                  ' Module variable to hold the starting date
Private mEndDate As Date                    ' Module variable to hold the ending date

' Return the current value of the starting date
Public Property Get StartDate() As Date
    StartDate = mStartDate
End Property

' Set the starting date value. Note that two methods have the name StartDate
Public Property Let StartDate(ByVal NewValue As Date)
    mStartDate = NewValue
End Property

' Same thing, but for the ending date
Public Property Get EndDate() As Date
    EndDate = mEndDate
End Property

Public Property Let EndDate(ByVal NewValue As Date)
    mEndDate = NewValue
End Property

' Read-only property that returns the number of days between the two dates
Public Property Get DaysBetween() As Integer
    DaysBetween = DateDiff("d", mStartDate, mEndDate)
End Function

' Write-only property that passes an object reference of a range to clone
Public Property Set RangeToCopy(ByRef ExistingRange As DateRange)

Me.StartDate = ExistingRange.StartDate
Me.EndDate = ExistingRange.EndDate
```

```
End Property
```

第32.2节：类模块的作用域、实例化和重用

默认情况下，新建的类模块是私有类，因此它仅在定义它的VB项目内可用于实例化和使用。你可以在同一个项目中的任何地方声明、实例化和使用该类：

```
'类List的实例化设置为私有
'在同一项目的任何其他模块中，你可以使用：

Dim items As List
Set items = New List
```

但通常你会编写希望在其他项目中使用的类，而无需在项目之间复制模块。如果你在ProjectA中定义了一个名为List的类，并想在ProjectB中使用该类，则需要执行4个操作：

1. 在属性窗口中，将ProjectA中List类的实例化属性从Private更改为 `PublicNotCreatable`
2. 在ProjectA中创建一个公共的“工厂”函数，该函数创建并返回一个List类的实例。通常工厂函数会包含用于初始化类实例的参数。需要工厂函数是因为该类可以被ProjectB使用，但ProjectB不能直接创建
   ProjectA的类的实例。

```
Public Function CreateList(ParamArray values() As Variant) As List
    Dim tempList As List
    Dim itemCounter As Long
    Set tempList = New List
    For itemCounter = LBound(values) to UBound(values)
       tempList.Add values(itemCounter)
    Next itemCounter
    Set CreateList = tempList
End Function
```

3. 在ProjectB中，使用工具..引用...菜单添加对ProjectA的引用。
4. 在ProjectB中，声明一个变量并使用ProjectA中的工厂函数为List分配一个实例

```
Dim items As ProjectA.List
Set items = ProjectA.CreateList("foo","bar")

'使用 items 列表的方法和属性
items.Add "fizz"
Debug.Print items.ToString()
'销毁 items 对象
Set items = Nothing
```

第32.3节：为类添加功能

类模块中的任何公共Sub、Function或Property都可以通过在调用前加上对象引用来调用：

```
对象.过程
```



```
End Property
```

Section 32.2: Class module scope, instancing and re-use

By default, a new class module is a Private class, so it is *only* available for instantiation and use within the VBProject in which it is defined. You can declare, instantiate and use the class anywhere in the *same* project:

```
'Class List has Instancing set to Private
'In any other module in the SAME project, you can use:

Dim items As List
Set items = New List
```

But often you'll write classes that you'd like to use in other projects *without* copying the module between projects. If you define a class called `List` in `ProjectA`, and want to use that class in `ProjectB`, then you'll need to perform 4 actions:

1. Change the instancing property of the `List` class in `ProjectA` in the Properties window, from **`Private`** to `PublicNotCreatable`
2. Create a public "factory" function in `ProjectA` that creates and returns an instance of a `List` class. Typically the factory function would include arguments for the initialization of the class instance. The factory function is required because the class can be used by `ProjectB` but `ProjectB` cannot directly create an instance of `ProjectA`'s class.

```
Public Function CreateList(ParamArray values() As Variant) As List
    Dim tempList As List
    Dim itemCounter As Long
    Set tempList = New List
    For itemCounter = LBound(values) to UBound(values)
        tempList.Add values(itemCounter)
    Next itemCounter
    Set CreateList = tempList
End Function
```

3. In `ProjectB` add a reference to `ProjectA` using the `Tools..References...` menu.
4. In `ProjectB`, declare a variable and assign it an instance of `List` using the factory function from `ProjectA`

```
Dim items As ProjectA.List
Set items = ProjectA.CreateList("foo","bar")

'Use the items list methods and properties
items.Add "fizz"
Debug.Print items.ToString()
'Destroy the items object
Set items = Nothing
```

Section 32.3: Adding Functionality to a Class

Any public **`Sub`**, **`Function`**, or **`Property`** inside a class module can be called by preceding the call with an object reference:

```
Object.Procedure
```

在DateRange类中，可以使用Sub向结束日期添加天数：

```
Public Sub AddDays(ByVal NoDays As Integer)
   mEndDate = mEndDate + NoDays
End Sub
```

一个函数可以返回下个月月底的最后一天（注意GetFirstDayOfMonth因为是私有的，类外不可见）：

```
Public Function GetNextMonthEndDate() As Date
    GetNextMonthEndDate = DateAdd("m", 1, GetFirstDayOfMonth())
End Function

Private Function GetFirstDayOfMonth() As Date
    GetFirstDayOfMonth = DateAdd("d", -DatePart("d", mEndDate), mEndDate)
End Function
```

过程可以接受任何类型的参数，包括对正在定义的类的对象的引用。

下面的示例测试当前的DateRange对象是否有起始日期和结束日期，且包含另一个DateRange对象的起始和结束日期。

```
Public Function ContainsRange(ByRef TheRange As DateRange) As Boolean
    ContainsRange = TheRange.StartDate >= Me.StartDate And TheRange.EndDate <= Me.EndDate
End Function
```

注意使用Me表示法作为访问运行代码的对象的值的一种方式。

In a `DateRange` class, a **`Sub`** could be used to add a number of days to the end date:

```
Public Sub AddDays(ByVal NoDays As Integer)
    mEndDate = mEndDate + NoDays
End Sub
```

A **`Function`** could return the last day of the next month-end (note that `GetFirstDayOfMonth` would not be visible outside the class because it is private):

```
Public Function GetNextMonthEndDate() As Date
    GetNextMonthEndDate = DateAdd("m", 1, GetFirstDayOfMonth())
End Function

Private Function GetFirstDayOfMonth() As Date
    GetFirstDayOfMonth = DateAdd("d", -DatePart("d", mEndDate), mEndDate)
End Function
```

Procedures can accept arguments of any type, including references to objects of the class being defined.

The following example tests whether the current `DateRange` object has a starting date and ending date that includes the starting and ending date of another `DateRange` object.

```
Public Function ContainsRange(ByRef TheRange As DateRange) As Boolean
    ContainsRange = TheRange.StartDate >= Me.StartDate And TheRange.EndDate <= Me.EndDate
End Function
```

Note the use of the `Me` notation as a way to access the value of the object running the code.

第33章：接口

接口（Interface）是一种定义类将执行的一组行为的方法。接口的定义是一组方法签名（名称、参数和返回类型）。拥有所有这些方法的类被称为“实现”该接口。

在 VBA 中，使用接口可以让编译器检查模块是否实现了其所有方法。变量或参数可以根据接口而不是特定类来定义。

第33.1节：一个类中的多个接口——可飞和可游

以Flyable接口为起点，我们可以添加第二个接口Swimmable，代码如下：

```
Sub Swim()
    ' 无代码
End Sub
```

Duck对象可以同时实现飞行和游泳功能：

```
Implements Flyable
Implements Swimmable

Public Sub Flyable_Fly()
    Debug.Print "用翅膀飞行！"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function

Public Sub Swimmable_Swim()
    Debug.Print "漂浮在水面上"
End Sub
```

一个Fish类也可以实现Swimmable接口：

```
Implements Swimmable

Public Sub Swimmable_Swim()
    Debug.Print "水下游泳"
End Sub
```

现在，我们可以看到Duck对象一方面可以作为Flyable传递给Sub，另一方面也可以作为Swimmable传递：

```
Sub InterfaceTest()

Dim MyDuck As New Duck
Dim MyAirplane As New Airplane
Dim MyFish As New Fish

Debug.Print "飞行检查..."

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```


Chapter 33: Interfaces

An **Interface** is a way to define a set of behaviors that a class will perform. The definition of an interface is a list of method signatures (name, parameters, and return type). A class having all of the methods is said to "implement" that interface.

In VBA, using interfaces lets the compiler check that a module implements all of its methods. A variable or parameter can be defined in terms of an interface instead of a specific class.

Section 33.1: Multiple Interfaces in One Class - Flyable and Swimable

Using the `Flyable` example as a starting point, we can add a second interface, `Swimmable`, with the following code:

```
Sub Swim()
    ' No code
End Sub
```

The `Duck` object can `Implement` both flying and swimming:

```
Implements Flyable
Implements Swimmable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Wings!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function

Public Sub Swimmable_Swim()
    Debug.Print "Floating on the water"
End Sub
```

A `Fish` class can implement `Swimmable`, too:

```
Implements Swimmable

Public Sub Swimmable_Swim()
    Debug.Print "Swimming under the water"
End Sub
```

Now, we can see that the `Duck` object can be passed to a Sub as a `Flyable` on one hand, and a `Swimmable` on the other:

```
Sub InterfaceTest()

Dim MyDuck As New Duck
Dim MyAirplane As New Airplane
Dim MyFish As New Fish

Debug.Print "Fly Check..."

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```

```
Debug.Print "游泳检查..."

TrySwimming MyDuck
TrySwimming MyFish

End Sub

Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
     Debug.Print F.GetAltitude
End Sub

Public Sub TrySwimming(S As Swimmable)
    S.Swim
End Sub
```

这段代码的输出是：

> 飞行检查中...
>
> 带翅膀飞行！
>
> 30
>
> 用喷气发动机飞行！
>
> 10000
>
> 游泳检查...
>
> 漂浮在水面上
>
> 水下游泳

第33.2节：简单接口 - 可飞行

接口Flyable是一个类模块，包含以下代码：

```
Public Sub Fly()
    '无代码。
End Sub

Public Function GetAltitude() As Long
    '无代码。
End Function
```

一个类模块Airplane使用Implements关键字告诉编译器，除非它有两个方法：一个Flyable_Fly()子程序和一个返回Long类型的Flyable_GetAltitude()函数，否则会报错。

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "用喷气发动机飞行！"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 10000
```



```
Debug.Print "Swim Check..."

TrySwimming MyDuck
TrySwimming MyFish

End Sub

Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub

Public Sub TrySwimming(S As Swimmable)
    S.Swim
End Sub
```

The output of this code is:

> Fly Check...
>
> Flying With Wings!
>
> 30
>
> Flying With Jet Engines!
>
> 10000
>
> Swim Check...
>
> Floating on the water
>
> Swimming under the water

Section 33.2: Simple Interface - Flyable

The interface `Flyable` is a class module with the following code:

```
Public Sub Fly()
    ' No code.
End Sub

Public Function GetAltitude() As Long
    ' No code.
End Function
```

A class module, `Airplane`, uses the **`Implements`** keyword to tell the compiler to raise an error unless it has two methods: a `Flyable_Fly()` sub and a `Flyable_GetAltitude()` function that returns a `Long`.

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Jet Engines!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 10000
```

```
End Function
```

第二个类模块，Duck，也实现了Flyable接口：

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "用翅膀飞行！"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function
```

我们可以编写一个例程，接受任何Flyable类型的值，知道它会响应Fly或GetAltitude命令：

```
Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub
```

因为接口已定义，IntelliSense弹出窗口会显示Fly和GetAltitude供F使用。

当我们运行以下代码时：

```
Dim MyDuck As New Duck
Dim MyAirplane As New Airplane

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```

输出是：

```
带翅膀飞行！
30
用喷气发动机飞行！
10000
```

请注意，尽管子程序在Airplane和Duck中都命名为Flyable_Fly，但当变量或参数被定义为Flyable时，可以调用为Fly。如果变量被具体定义为Duck，则必须调用为Flyable_Fly。


```
End Function
```

A second class module, `Duck`, also implements `Flyable`:

```
Implements Flyable

Public Sub Flyable_Fly()
    Debug.Print "Flying With Wings!"
End Sub

Public Function Flyable_GetAltitude() As Long
    Flyable_GetAltitude = 30
End Function
```

We can write a routine that accepts any `Flyable` value, knowing that it will respond to a command of `Fly` or `GetAltitude`:

```
Public Sub FlyAndCheckAltitude(F As Flyable)
    F.Fly
    Debug.Print F.GetAltitude
End Sub
```

Because the interface is defined, the IntelliSense popup window will show `Fly` and `GetAltitude` for `F`.

When we run the following code:

```
Dim MyDuck As New Duck
Dim MyAirplane As New Airplane

FlyAndCheckAltitude MyDuck
FlyAndCheckAltitude MyAirplane
```

The output is:

```
Flying With Wings!
30
Flying With Jet Engines!
10000
```

Note that even though the subroutine is named `Flyable_Fly` in both `Airplane` and `Duck`, it can be called as `Fly` when the variable or parameter is defined as `Flyable`. If the variable is defined specifically as a `Duck`, it would have to be called as `Flyable_Fly`.

第34章：递归

调用自身的函数称为递归函数。递归逻辑通常也可以实现为循环。递归必须通过参数进行控制，以便函数知道何时停止递归并停止加深调用栈。无限递归最终会导致运行时错误28："堆栈空间不足"。

参见递归。

第34.1节：阶乘

```
函数 Factorial(Value 作为 Long) 作为 Long
    如果 Value = 0 或 Value = 1 则
Factorial = 1
    Else
Factorial = Factorial(Value - 1) * Value
    End If
End Function
```

第34.2节：文件夹递归

早绑定（引用了Microsoft Scripting Runtime）

```
    子程序 EnumerateFilesAndFolders( _
        FolderPath 作为 字符串, _
        Optional MaxDepth As Long = -1, _
        Optional CurrentDepth As Long = 0, _
        Optional Indentation As Long = 2)

        Dim FSO As Scripting.FileSystemObject
        Set FSO = New Scripting.FileSystemObject

        '检查文件夹是否存在
        If FSO.FolderExists(FolderPath) Then
            Dim fldr As Scripting.Folder
            Set fldr = FSO.GetFolder(FolderPath)

            '输出起始目录路径
            如果 CurrentDepth = 0 则
              Debug.Print fldr.Path
            结束如果

            列举子文件夹
            Dim subFldr As Scripting.Folder
            For Each subFldr In fldr.SubFolders
Debug.Print Space$((CurrentDepth + 1) * Indentation) & subFldr.Name
                如果 CurrentDepth < MaxDepth 或 MaxDepth = -1 则
                    '递归调用 EnumerateFilesAndFolders
EnumerateFilesAndFolders subFldr.Path, MaxDepth, CurrentDepth + 1, Indentation
                结束如果
            Next subFldr

            '枚举文件
            Dim fil As Scripting.File
            For Each fil In fldr.Files
Debug.Print Space$((CurrentDepth + 1) * Indentation) & fil.Name
            下一个 fil
        End If
```


Chapter 34: Recursion

A function that calls itself is said to be *recursive*. Recursive logic can often be implemented as a loop, too. Recursion must be controlled with a parameter, so that the function knows when to stop recursing and deepening the call stack. *Infinite recursion* eventually causes a run-time error '28': "Out of stack space".

See Recursion.

Section 34.1: Factorials

```
Function Factorial(Value As Long) As Long
    If Value = 0 Or Value = 1 Then
         Factorial = 1
    Else
       Factorial = Factorial(Value - 1) * Value
    End If
End Function
```

Section 34.2: Folder Recursion

Early Bound (with a reference to `Microsoft Scripting Runtime`)

```
    Sub EnumerateFilesAndFolders( _
        FolderPath As String, _
        Optional MaxDepth As Long = -1, _
        Optional CurrentDepth As Long = 0, _
        Optional Indentation As Long = 2)

        Dim FSO As Scripting.FileSystemObject
        Set FSO = New Scripting.FileSystemObject

        'Check the folder exists
        If FSO.FolderExists(FolderPath) Then
            Dim fldr As Scripting.Folder
            Set fldr = FSO.GetFolder(FolderPath)

            'Output the starting directory path
            If CurrentDepth = 0 Then
              Debug.Print fldr.Path
            End If

            'Enumerate the subfolders
            Dim subFldr As Scripting.Folder
            For Each subFldr In fldr.SubFolders
                Debug.Print Space$((CurrentDepth + 1) * Indentation) & subFldr.Name
                If CurrentDepth < MaxDepth Or MaxDepth = -1 Then
                    'Recursively call EnumerateFilesAndFolders
                    EnumerateFilesAndFolders subFldr.Path, MaxDepth, CurrentDepth + 1, Indentation
                End If
            Next subFldr

            'Enumerate the files
            Dim fil As Scripting.File
            For Each fil In fldr.Files
                Debug.Print Space$((CurrentDepth + 1) * Indentation) & fil.Name
            Next fil
        End If
```

```
End Sub
```


End Sub

第35章：事件

第35.1节：源和处理程序

什么是事件？

VBA 是事件驱动的：VBA 代码响应主机应用程序或主机文档触发的事件而运行——理解事件是理解 VBA 的基础。

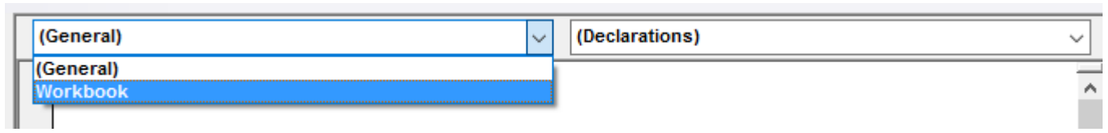
API 通常会暴露一些对象，这些对象会在各种状态下触发多个事件。例如，一个Excel.Application对象在每次创建、打开、激活或关闭新工作簿时都会触发一个事件。或者每当工作表被计算时。或者就在文件保存之前。或者紧接着保存之后。窗体上的按钮在用户点击时会触发一个Click事件，用户窗体本身在激活后会触发一个事件，在关闭之前会触发另一个事件。

从 API 的角度来看，事件是扩展点：客户端代码可以选择实现处理这些事件的代码，并在事件触发时执行自定义代码：这就是你如何通过处理在任何工作表上选择更改时触发的事件，自动执行自定义代码的方式。

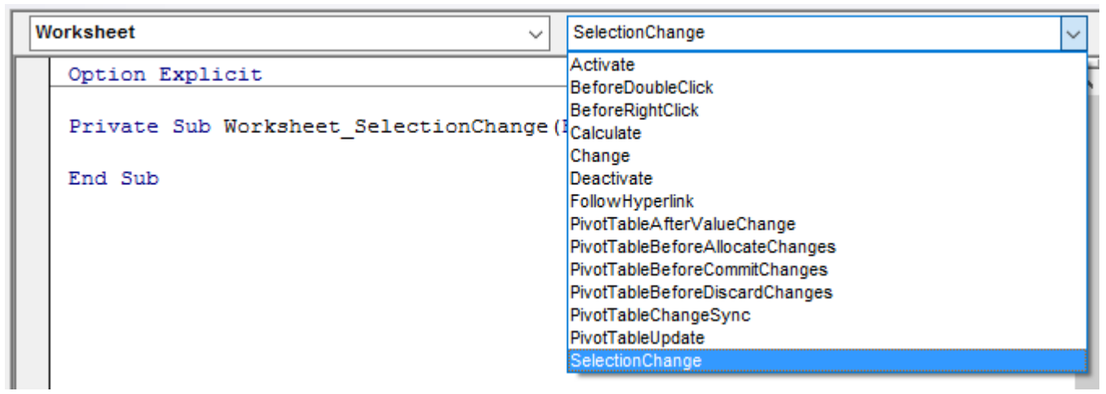
暴露事件的对象称为事件源。处理事件的方法称为处理程序。

处理程序

VBA 文档模块（例如ThisDocument、ThisWorkbook、Sheet1等）和UserForm模块是类模块，它们实现特殊接口，暴露多个事件。你可以在代码窗格顶部左侧的下拉菜单中浏览这些接口：



右侧的下拉菜单列出了左侧下拉菜单中选定接口的成员：



当右侧列表中选择某个项目时，VBE 会自动生成一个事件处理程序的代码框架，或者如果处理程序已存在，则导航到该位置。

你可以在任何模块中定义一个模块范围的WithEvents变量：

Chapter 35: Events

Section 35.1: Sources and Handlers

What are events?

VBA is *event-driven*: VBA code runs in response to events raised by the host application or the host document - understanding events is fundamental to understanding VBA.

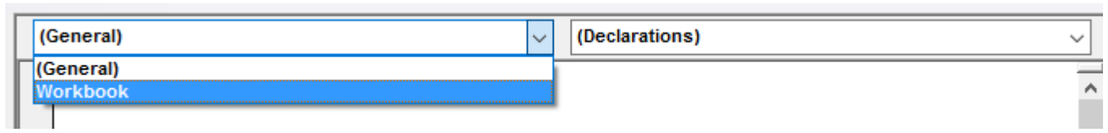
APIs often expose objects that raise a number of *events* in response to various states. For example an `Excel.Application` object raises an event whenever a new workbook is created, opened, activated, or closed. Or whenever a worksheet gets calculated. Or just before a file is saved. Or immediately after. A button on a form raises a `Click` event when the user clicks it, the user form itself raises an event just after it's activated, and another just before it's closed.

From an API perspective, events are *extension points*: the client code can chose to implement code that *handles* these events, and execute custom code whenever these events are fired: that's how you can execute your custom code automatically every time the selection changes on any worksheet - by handling the event that gets fired when the selection changes on any worksheet.

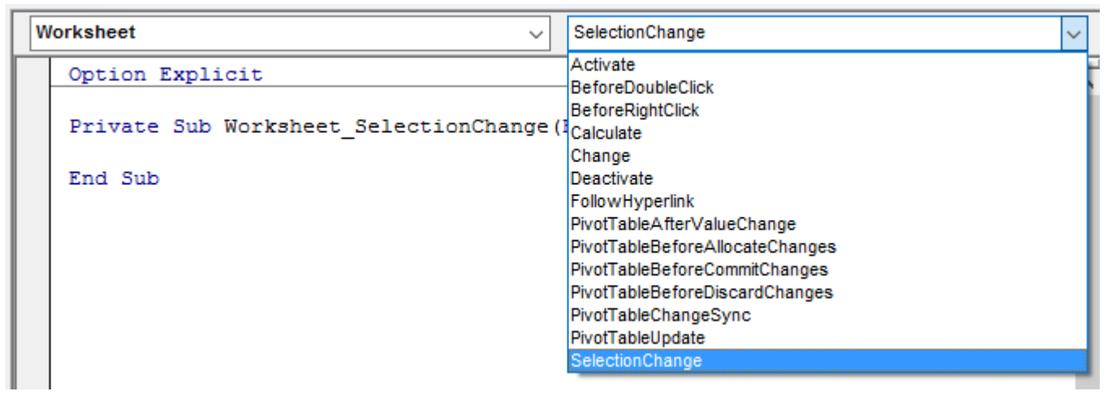
An object that exposes events is an *event source*. A method that handles an event is a *handler*.

Handlers

VBA document modules (e.g. `ThisDocument`, `ThisWorkbook`, `Sheet1`, etc.) and `UserForm` modules are *class modules* that *implement* special interfaces that expose a number of *events*. You can browse these interfaces in the left-side dropdown at the top of the code pane:



The right-side dropdown lists the members of the interface selected in the left-side dropdown:



The VBE automatically generates an event handler stub when an item is selected on the right-side list, or navigates there if the handler exists.

You can define a module-scoped **`WithEvents`** variable in any module:

```
Private WithEvents Foo 作为 Workbook
Private WithEvents Bar 作为 Worksheet
```

每个WithEvents声明都会出现在左侧下拉菜单中可供选择。当在右侧下拉菜单中选择一个事件时，VBE会生成一个事件处理程序的代码框架，名称由WithEvents对象名和事件名用下划线连接组成：

```
Private WithEvents Foo 作为 Workbook
Private WithEvents Bar 作为 Worksheet

Private Sub Foo_Open()

End Sub

Private Sub Bar_SelectionChange(ByVal Target 作为 Range)

End Sub
```

只有至少暴露一个事件的类型才能与WithEvents一起使用，且WithEvents声明不能使用New关键字即时赋值。以下代码是非法的：

```
Private WithEvents Foo 作为 New Workbook '非法
```

对象引用必须显式使用Set赋值；在类模块中，通常在Class_Initialize处理程序中进行赋值是个好地方，因为这样类实例存在期间会处理该对象的事件。

来源

任何类模块（或文档模块，或用户窗体）都可以是事件源。使用Event关键字在模块的声明部分定义事件的签名：

```
公共事件 SomethingHappened(ByVal something As String)
```

事件的签名决定了事件如何被触发，以及事件处理程序的形式。

事件只能在其定义的类中触发——客户端代码只能处理它们。事件通过RaiseEvent关键字触发；事件的参数在此时提供：

```
Public Sub DoSomething()
    RaiseEvent SomethingHappened("hello")
End Sub
```

如果没有处理SomethingHappened事件的代码，运行DoSomething过程仍会触发事件，但不会有任何反应。假设事件源是名为Something的类中的上述代码，以下ThisWorkbook中的代码将在每次调用 test.DoSomething时显示一个消息框，内容为“hello”：

```
Private WithEvents test As Something

Private Sub Workbook_Open()
    Set test = New Something
   test.DoSomething
End Sub

Private Sub test_SomethingHappened(ByVal bar As String)
'当'test'触发'SomethingHappened'事件时，此过程会运行
```



```
Private WithEvents Foo As Workbook
Private WithEvents Bar As Worksheet
```

Each `WithEvents` declaration becomes available to select from the left-side dropdown. When an event is selected in the right-side dropdown, the VBE generates an event handler stub named after the `WithEvents` object and the name of the event, joined with an underscore:

```
Private WithEvents Foo As Workbook
Private WithEvents Bar As Worksheet

Private Sub Foo_Open()

End Sub

Private Sub Bar_SelectionChange(ByVal Target As Range)

End Sub
```

Only types that expose at least one event can be used with `WithEvents`, and `WithEvents` declarations cannot be assigned a reference on-the-spot with the `New` keyword. This code is illegal:

```
Private WithEvents Foo As New Workbook 'illegal
```

The object reference must be `Set` explicitly; in a class module, a good place to do that is often in the `Class_Initialize` handler, because then the class handles that object's events for as long as its instance exists.

Sources

Any class module (or document module, or user form) can be an event source. Use the `Event` keyword to define the *signature* for the event, in the *declarations section* of the module:

```
Public Event SomethingHappened(ByVal something As String)
```

The signature of the event determines how the event is raised, and what the event handlers will look like.

Events can only be *raised* within the class they're defined in - client code can only *handle* them. Events are raised with the `RaiseEvent` keyword; the event's arguments are provided at that point:

```
Public Sub DoSomething()
    RaiseEvent SomethingHappened("hello")
End Sub
```

Without code that handles the `SomethingHappened` event, running the `DoSomething` procedure will still raise the event, but nothing will happen. Assuming the event source is the above code in a class named `Something`, this code in `ThisWorkbook` would show a message box saying "hello" whenever `test.DoSomething` gets called:

```
Private WithEvents test As Something

Private Sub Workbook_Open()
    Set test = New Something
    test.DoSomething
End Sub

Private Sub test_SomethingHappened(ByVal bar As String)
'this procedure runs whenever 'test' raises the 'SomethingHappened' event
```

```
    MsgBox bar
End Sub
```

第35.2节：向事件源传回数据

使用按引用传递的参数

事件可以定义一个ByRef参数，用于返回给调用者：

```
Public Event BeforeSomething(ByRef cancel As Boolean)
Public Event AfterSomething()

Public Sub DoSomething()
    Dim cancel As Boolean
    RaiseEvent BeforeSomething(cancel)
    If cancel Then Exit Sub

    'todo: 实际执行某些操作

    RaiseEvent AfterSomething
End Sub
```

如果BeforeSomething事件有处理程序将其cancel参数设置为True，则当执行从处理程序返回时，cancel将为True，且AfterSomething事件将永远不会被触发。

```
Private WithEvents foo As Something

Private Sub foo_BeforeSomething(ByRef cancel As Boolean)
    cancel = MsgBox("取消？", vbYesNo) = vbYes
End Sub

Private Sub foo_AfterSomething()
    MsgBox "未取消！"
End Sub
```

假设foo对象引用已在某处赋值，当foo.DoSomething运行时，会弹出一个消息框询问是否取消，只有当 否 被选中。

使用可变对象

你也可以传递一个可变对象的副本**ByVal**，并让处理程序修改该对象的属性；调用者然后可以读取修改后的属性值并据此采取行动。

```
类模块 ReturnBoolean
Option Explicit
私有封装作为Boolean

公共属性获取 ReturnValue()作为Boolean
'属性 ReturnValue.VB_UserMemId = 0
ReturnValue = 封装
End Property

公共属性设置 ReturnValue(ByVal 值 作为 Boolean)
    封装 = 值
End Property
```

结合Variant类型，可以用来创建相当不明显的方式将值返回给调用者：


```
    MsgBox bar
End Sub
```

Section 35.2: Passing data back to the event source

Using parameters passed by reference

An event may define a **ByRef** parameter meant to be returned to the caller:

```
Public Event BeforeSomething(ByRef cancel As Boolean)
Public Event AfterSomething()

Public Sub DoSomething()
    Dim cancel As Boolean
    RaiseEvent BeforeSomething(cancel)
    If cancel Then Exit Sub

    'todo: actually do something

    RaiseEvent AfterSomething
End Sub
```

If the `BeforeSomething` event has a handler that sets its `cancel` parameter to **True**, then when execution returns from the handler, `cancel` will be **True** and `AfterSomething` will never be raised.

```
Private WithEvents foo As Something

Private Sub foo_BeforeSomething(ByRef cancel As Boolean)
    cancel = MsgBox("Cancel?", vbYesNo) = vbYes
End Sub

Private Sub foo_AfterSomething()
    MsgBox "Didn't cancel!"
End Sub
```

Assuming the `foo` object reference is assigned somewhere, when `foo.DoSomething` runs, a message box prompts whether to cancel, and a second message box says "didn't cancel" only when No was selected.

Using mutable objects

You could also pass a copy of a mutable object **ByVal**, and let handlers modify that object's properties; the caller can then read the modified property values and act accordingly.

```
'class module ReturnBoolean
Option Explicit
Private encapsulated As Boolean

Public Property Get ReturnValue() As Boolean
'Attribute ReturnValue.VB_UserMemId = 0
    ReturnValue = encapsulated
End Property

Public Property Let ReturnValue(ByVal value As Boolean)
    encapsulated = value
End Property
```

Combined with the `Variant` type, this can be used to create rather non-obvious ways to return a value to the caller:

```
公共事件 SomeEvent(ByVal foo 作为 Variant)

公共子程序 DoSomething()
    定义 result 作为 ReturnBoolean
    result = 新建 ReturnBoolean

    RaiseEvent SomeEvent(result)

    如果 result 那么 ' 如果 result.ReturnValue 为真
        '处理程序将值更改为 True
    Else
        '处理程序未修改该值
    结束 如果
结束 子程序
```

处理程序看起来像这样：

```
Private Sub source_SomeEvent(ByVal foo As Variant) 'foo 实际上是一个 ReturnBoolean 对象
    foo = True 'True 实际上赋值给 foo.ReturnValue，类的默认成员
End Sub
```



```
Public Event SomeEvent(ByVal foo As Variant)

Public Sub DoSomething()
    Dim result As ReturnBoolean
    result = New ReturnBoolean

    RaiseEvent SomeEvent(result)

    If result Then ' If result.ReturnValue Then
        'handler changed the value to True
    Else
        'handler didn't modify the value
    End If
End Sub
```

The handler would look like this:

```
Private Sub source_SomeEvent(ByVal foo As Variant) 'foo is actually a ReturnBoolean object
    foo = True 'True is actually assigned to foo.ReturnValue, the class' default member
End Sub
```

第36章：Scripting.Dictionary 对象

必须通过 VBE 的“工具 → 引用”命令将 Microsoft Scripting Runtime 添加到 VBA 项目中，
以实现 Scripting Dictionary 对象的早期绑定。该库引用随项目一起携带；
在将 VBA 项目分发并在另一台计算机上运行时，无需重新引用。

第36.1节：属性和方法

一个 Scripting Dictionary 对象 以键/项对的形式存储信息。键必须唯一且不能是数组，
但关联的项可以重复（它们的唯一性由对应的键保持），且可以是任何类型的
变量或对象。

字典可以被视为一个具有主唯一索引的两字段内存数据库，索引位于第一个“字段”
（即 *Key*）。键属性上的此唯一索引允许非常快速的“查找”，以检索键关联的项值。

属性

姓名	读/写	类型	描述
CompareMode	*读 / 写*	CompareMode 常量	CompareMode 的设置只能在空的字典上执行。接受的值为 0 (vbBinaryCompare)、1 (vbTextCompare)、2 (vbDatabaseCompare)。
计数	只读无符号长整数		脚本中键/项对的基于1的计数字典对象。
键	*读 / 写*	非数组变量	字典中每个唯一的键。
Item(Key)	*读 / 写*	任意变量	默认属性。与字典中键关联的每个单独项字典。注意，尝试使用字典中不存在的键检索项时，将隐式添加传入的键。

方法

姓名	描述
Add(Key,Item)	向字典中添加一个新的键和值。新键在字典当前中必须不存在键集合，但一个项目可以在多个唯一键中重复。
Exists(Key)	布尔测试，用于确定字典中是否已存在某个键。
键	返回唯一键的数组或集合。
项目	返回关联项目的数组或集合。
Remove(Key)	删除单个字典键及其关联的项目。
RemoveAll	清除字典对象的所有键和值。

示例代码

```
'填充、枚举、定位并删除通过晚绑定创建的字典中的条目
'
Sub iterateDictionaryLate()
    Dim k As Variant, dict As Object

    Set dict = CreateObject("Scripting.Dictionary")
    dict.CompareMode = vbTextCompare          '不区分大小写的比较模式

    '填充字典
dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"
```


Chapter 36: Scripting.Dictionary object

You must add Microsoft Scripting Runtime to the VBA project through the VBE's Tools → References command in order to implement early binding of the Scripting Dictionary object. This library reference is carried with the project; it does not have to be re-referenced when the VBA project is distributed and run on another computer.

Section 36.1: Properties and Methods

A Scripting Dictionary object stores information in Key/Item pairs. The Keys must be unique and not an array but the associated Items can be repeated (their uniqueness is held by the companion Key) and can be of any type of variant or object.

A dictionary can be thought of as a two field in-memory database with a primary unique index on the first 'field' (the *Key*). This unique index on the Keys property allows very fast 'lookups' to retrieve a Key's associated Item value.

Properties

name	read/write	type	description
CompareMode	*read / write*	CompareMode constant	Setting the CompareMode can only be performed on an empty dictionary. Accepted values are 0 (vbBinaryCompare), 1 (vbTextCompare), 2 (vbDatabaseCompare).
Count	*read only*	unsigned long integer	A one-based count of the key/item pairs in the scripting dictionary object.
Key	*read / write*	non-array variant	Each individual unique key in the dictionary.
Item(*Key*)	*read / write*	any variant	Default property. Each individual item associated with a key in the dictionary. Note that attempting to retrieve an item with a key that does not exist in the dictionary will *implicitly add* the passed key.

Methods

name	description
Add(*Key*,*Item*)	Adds a new Key and Item to the dictionary. The new key must not exist in the dictionary's current Keys collection but an item can be repeated among many unique keys.
Exists(*Key*)	Boolean test to determine if a Key already exists in the dictionary.
Keys	Returns the array or collection of unique keys.
Items	Returns the array or collection of associated items.
Remove(*Key*)	Removes an individual dictionary key and its associated item.
RemoveAll	Clears all of a dictionary object's keys and items.

Sample Code

```
'Populate, enumerate, locate and remove entries in a dictionary that was created
'with late binding
Sub iterateDictionaryLate()
    Dim k As Variant, dict As Object

    Set dict = CreateObject("Scripting.Dictionary")
    dict.CompareMode = vbTextCompare          'non-case sensitive compare model

    'populate the dictionary
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"
```

```
    '遍历键
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    '查找Green对应的项
Debug.Print dict.Item("Green")

    '从字典中移除键/项对
dict.Remove "blue"       '通过键移除单个键/项对
    dict.RemoveAll           '移除所有剩余的键/项对

End Sub

在使用早期绑定创建的字典中填充、枚举、定位和删除条目（参见备注）

Sub iterateDictionaryEarly()
    Dim d As Long, k As Variant
    Dim dict As New Scripting.Dictionary

dict.CompareMode = vbTextCompare          '不区分大小写的比较模式

    '填充字典
dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"
    dict.Add Key:="White", Item:="Balloon"

    '遍历键
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    '通过计数遍历键
    For d = 0 To dict.Count - 1
Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    Next d

    通过键集合的边界遍历键
    对于 d = LBound(dict.Keys) 到 UBound(dict.Keys)
       Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    Next d

    '查找Green对应的项
Debug.Print dict.Item("Green")
    '定位第一个键的项
    Debug.Print dict.Item(dict.Keys(0))
    '定位最后一个键的项
    Debug.Print dict.Item(dict.Keys(UBound(dict.Keys)))

    '从字典中移除键/项对
dict.Remove "blue"                              '通过键移除单个键/项对
    dict.Remove dict.Keys(0)                        '通过索引位置移除第一个键/项
    dict.Remove dict.Keys(UBound(dict.Keys))    '通过索引位置移除最后一个键/项
    dict.RemoveAll                                  '移除所有剩余的键/项对

End Sub
```



```
    'iterate through the keys
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    'locate the Item for Green
    Debug.Print dict.Item("Green")

    'remove key/item pairs from the dictionary
    dict.Remove "blue"       'remove individual key/item pair by key
    dict.RemoveAll           'remove all remaining key/item pairs

End Sub

'Populate, enumerate, locate and remove entries in a dictionary that was created
'with early binding (see Remarks)
Sub iterateDictionaryEarly()
    Dim d As Long, k As Variant
    Dim dict As New Scripting.Dictionary

    dict.CompareMode = vbTextCompare          'non-case sensitive compare model

    'populate the dictionary
    dict.Add Key:="Red", Item:="Balloon"
    dict.Add Key:="Green", Item:="Balloon"
    dict.Add Key:="Blue", Item:="Balloon"
    dict.Add Key:="White", Item:="Balloon"

    'iterate through the keys
    For Each k In dict.Keys
        Debug.Print k & " - " & dict.Item(k)
    Next k

    'iterate through the keys by the count
    For d = 0 To dict.Count - 1
        Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    Next d

    'iterate through the keys by the boundaries of the keys collection
    For d = LBound(dict.Keys) To UBound(dict.Keys)
        Debug.Print dict.Keys(d) & " - " & dict.Items(d)
    Next d

    'locate the Item for Green
    Debug.Print dict.Item("Green")
    'locate the Item for the first key
    Debug.Print dict.Item(dict.Keys(0))
    'locate the Item for the last key
    Debug.Print dict.Item(dict.Keys(UBound(dict.Keys)))

    'remove key/item pairs from the dictionary
    dict.Remove "blue"                          'remove individual key/item pair by key
    dict.Remove dict.Keys(0)                    'remove first key/item by index position
    dict.Remove dict.Keys(UBound(dict.Keys))    'remove last key/item by index position
    dict.RemoveAll                              'remove all remaining key/item pairs

End Sub
```

第37章：使用ADO

第37.1节：连接数据源

通过ADO访问数据源的第一步是创建一个ADOConnection对象。通常使用连接字符串来指定数据源参数，尽管也可以通过将DSN、用户ID和密码传递给.Open方法来打开DSN连接。

注意，通过ADO连接数据源不需要DSN——任何具有ODBC提供程序的数据源都可以通过适当的连接字符串连接。虽然不同提供程序的具体连接字符串超出本主题范围，但ConnectionStrings.com是查找适合您提供程序的字符串的极好参考。

```
Const SomeDSN As String = "DSN=SomeDSN;Uid=UserName;Pwd=MyPassword;"

Public Sub Example()
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
    If Not database Is Nothing Then
        ……做工作。
database.Close          '确保关闭所有数据库连接。
    End If
End Sub

Public Function OpenDatabaseConnection(ConnString As String) As ADODB.Connection
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = New ADODB.Connection

    With database
        .ConnectionString = ConnString
        .ConnectionTimeout = 10             '值以秒为单位。
        .Open
    结束于

    OpenDatabaseConnection = database

    Exit Function
Handler:
Debug.Print "数据库连接失败。请检查您的连接字符串。"
End Function
```

请注意，上述示例中的连接字符串包含数据库密码仅为清晰起见。最佳实践是不将数据库密码存储在代码中。这可以通过用户输入密码或使用Windows身份验证来实现。

第37.2节：创建参数化命令

任何通过ADO连接执行的SQL语句如果需要包含用户输入，最佳实践是对其进行参数化，以尽量减少SQL注入的风险。该方法比长串联更易读，并且有助于编写更健壮且易维护的代码（例如，通过使用返回参数数组的函数）

参数）。

在标准ODBC语法中，参数在查询文本中用?“占位符”表示，然后参数按照它们在查询中出现的顺序附加到Command对象中。

Chapter 37: Working with ADO

Section 37.1: Making a connection to a data source

The first step in accessing a data source via ADO is creating an ADO `Connection` object. This is typically done using a connection string to specify the data source parameters, although it is also possible to open a DSN connection by passing the DSN, user ID, and password to the `.Open` method.

Note that a DSN is not required to connect to a data source via ADO - any data source that has an ODBC provider can be connected to with the appropriate connection string. While specific connection strings for different providers are outside of the scope of this topic, ConnectionStrings.com is an excellent reference for finding the appropriate string for your provider.

```
Const SomeDSN As String = "DSN=SomeDSN;Uid=UserName;Pwd=MyPassword;"

Public Sub Example()
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
    If Not database Is Nothing Then
        '... Do work.
        database.Close          'Make sure to close all database connections.
    End If
End Sub

Public Function OpenDatabaseConnection(ConnString As String) As ADODB.Connection
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = New ADODB.Connection

    With database
        .ConnectionString = ConnString
        .ConnectionTimeout = 10             'Value is given in seconds.
        .Open
    End With

    OpenDatabaseConnection = database

    Exit Function
Handler:
    Debug.Print "Database connection failed. Check your connection string."
End Function
```

Note that the database password is included in the connection string in the example above only for the sake of clarity. Best practices would dictate ***not*** storing database passwords in code. This can be accomplished by taking the password via user input or using Windows authentication.

Section 37.2: Creating parameterized commands

Any time SQL executed through an ADO connection needs to contain user input, it is considered best practice to parameterize it in order to minimize the chance of SQL injection. This method is also more readable than long concatenations and facilitates more robust and maintainable code (i.e. by using a function that returns an array of `Parameter`).

In standard ODBC syntax, parameters are given `?` "placeholders" in the query text, and then parameters are appended to the `Command` in the same order that they appear in the query.

请注意，下面的示例为了简洁，使用了OpenDatabaseConnection函数来建立数据源连接。

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        '构建传递给数据源的命令。
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = ? WHERE Bar = ?"
            .CommandType = adCmdText

            创建参数。
            Dim fooValue As ADODB.Parameter
            Set fooValue = .CreateParameter("FooValue", adNumeric, adParamInput)
            fooValue.Value = 42

            Dim condition As ADODB.Parameter
            Set condition = .CreateParameter("Condition", adBSTR, adParamInput)
            condition.Value = "Bar"

            将参数添加到命令中
            .Parameters.Append fooValue
            .Parameters.Append condition
            .Execute
        End With
    End If
清理退出：
    If Not database Is Nothing And database.State = adStateOpen Then
       database.Close
    End If
    退出子程序
处理程序：
Debug.Print "错误 " & Err.Number & ": " & Err.Description
    恢复 清理退出
结束子程序
```

注意：上面的示例演示了一个参数化的 UPDATE 语句，但任何 SQL 语句都可以传递参数。

第37.3节：使用查询检索记录

查询可以通过两种方式执行，这两种方式都会返回一个 ADO Recordset 对象，该对象是返回行的集合。请注意，下面的两个示例都使用了“连接到数据源示例”中的 OpenDatabaseConnection 函数，以简化代码。请记住，传递给数据源的 SQL 语法是特定于提供程序的。

第一种方法是将 SQL 语句直接传递给 Connection 对象，这是执行简单查询最简单的方法：

```
公共子程序 DisplayDistinctItems()
    出错转到 处理程序
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
```


Note that the example below uses the `OpenDatabaseConnection` function from the Making a connection to a data source for brevity.

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        'Build the command to pass to the data source.
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = ? WHERE Bar = ?"
            .CommandType = adCmdText

            'Create the parameters.
            Dim fooValue As ADODB.Parameter
            Set fooValue = .CreateParameter("FooValue", adNumeric, adParamInput)
            fooValue.Value = 42

            Dim condition As ADODB.Parameter
            Set condition = .CreateParameter("Condition", adBSTR, adParamInput)
            condition.Value = "Bar"

            'Add the parameters to the Command
            .Parameters.Append fooValue
            .Parameters.Append condition
            .Execute
        End With
    End If
CleanExit:
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Note: The example above demonstrates a parameterized UPDATE statement, but any SQL statement can be given parameters.

Section 37.3: Retrieving records with a query

Queries can be performed in two ways, both of which return an ADO `Recordset` object which is a collection of returned rows. Note that both of the examples below use the `OpenDatabaseConnection` function from the Making a connection to a data source example for the purpose of brevity. Remember that the syntax of the SQL passed to the data source is provider specific.

The first method is to pass the SQL statement directly to the Connection object, and is the easiest method for executing simple queries:

```
Public Sub DisplayDistinctItems()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)
```

```
    如果 database 不为空
        Dim records As ADODB.Recordset
        设置 records = database.Execute("SELECT DISTINCT Item FROM Table")
        '遍历返回的记录集。
        Do While Not records.EOF      '当还有更多记录时，EOF为假。
            '单个字段可以通过名称或基于0的序号索引。
            '注意这里使用的是Recordset的默认.Fields成员。
Debug.Print records("Item")
            '移动到下一条记录。
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
       database.Close
    End If
    退出子程序
处理程序：
Debug.Print "错误 " & Err.Number & ": " & Err.Description
    恢复 清理退出
结束子程序
```

第二种方法是为你想执行的查询创建一个ADO Command 对象。这需要更多代码，但为了使用参数化查询，这是必要的：

```
公共子程序 DisplayDistinctItems()
    出错转到 处理程序
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim query As ADODB.Command
        Set query = New ADODB.Command
        '构建传递给数据源的命令。
        With query
            .ActiveConnection = database
            .CommandText = "SELECT DISTINCT Item FROM Table"
            .CommandType = adCmdText
        结束于
        Dim records As ADODB.Recordset
        '执行命令以检索记录集。
        Set records = query.Execute()

        Do While Not records.EOF
            Debug.Print records("Item")
          records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
       database.Close
    End If
    退出子程序
处理程序：
Debug.Print "错误 " & Err.Number & ": " & Err.Description
    恢复 清理退出
结束子程序
```



```
    If Not database Is Nothing Then
        Dim records As ADODB.Recordset
        Set records = database.Execute("SELECT DISTINCT Item FROM Table")
        'Loop through the returned Recordset.
        Do While Not records.EOF      'EOF is false when there are more records.
            'Individual fields are indexed either by name or 0 based ordinal.
            'Note that this is using the default .Fields member of the Recordset.
            Debug.Print records("Item")
            'Move to the next record.
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

The second method is to create an ADO `Command` object for the query you want to execute. This requires a little more code, but is necessary in order to use parametrized queries:

```
Public Sub DisplayDistinctItems()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim query As ADODB.Command
        Set query = New ADODB.Command
        'Build the command to pass to the data source.
        With query
            .ActiveConnection = database
            .CommandText = "SELECT DISTINCT Item FROM Table"
            .CommandType = adCmdText
        End With
        Dim records As ADODB.Recordset
        'Execute the command to retrieve the recordset.
        Set records = query.Execute()

        Do While Not records.EOF
            Debug.Print records("Item")
            records.MoveNext
        Loop
    End If
CleanExit:
    If Not records Is Nothing Then records.Close
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

请注意，发送到数据源的命令容易受到SQL注入攻击，无论是有意还是无意。
通常，不应通过拼接任何用户输入来创建查询。相反，应使用参数化查询（参见创建参数化命令）。

第37.4节：执行非标量函数

ADO连接可用于执行提供程序通过SQL支持的几乎所有数据库功能。在这种情况下，不一定总是需要使用Execute函数返回的Recordset，尽管它对于在INSERT语句后使用@@Identity或类似SQL命令获取主键分配很有用。请注意，下面的示例为了简洁起见，使用了“连接到数据源示例”中的OpenDatabaseConnection函数。

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        '构建传递给数据源的命令。
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = 42 WHERE Bar IS NULL"
            .CommandType = adCmdText
            .Execute        '我们不需要数据库的返回值，所以忽略它。
        End With
    End If
清理退出：
    If Not database Is Nothing And database.State = adStateOpen Then
       database.Close
    End If
    退出子程序
处理程序：
Debug.Print "错误 " & Err.Number & ": " & Err.Description
    恢复 清理退出
结束子程序
```

请注意，发送到数据源的命令容易受到SQL注入攻击，无论是有意还是无意。
一般来说，不应通过连接任何用户输入来创建 SQL 语句。相反，应使用参数化（参见创建参数化命令）。

Note that commands sent to the data source are **vulnerable to SQL injection**, either intentional or unintentional. In general, queries should not be created by concatenating user input of any kind. Instead, they should be parameterized (see Creating parameterized commands).

Section 37.4: Executing non-scalar functions

ADO connections can be used to perform pretty much any database function that the provider supports via SQL. In this case it isn't always necessary to use the `Recordset` returned by the `Execute` function, although it can be useful for obtaining key assignments after INSERT statements with @@Identity or similar SQL commands. Note that the example below uses the `OpenDatabaseConnection` function from the Making a connection to a data source example for the purpose of brevity.

```
Public Sub UpdateTheFoos()
    On Error GoTo Handler
    Dim database As ADODB.Connection
    Set database = OpenDatabaseConnection(SomeDSN)

    If Not database Is Nothing Then
        Dim update As ADODB.Command
        Set update = New ADODB.Command
        'Build the command to pass to the data source.
        With update
            .ActiveConnection = database
            .CommandText = "UPDATE Table SET Foo = 42 WHERE Bar IS NULL"
            .CommandType = adCmdText
            .Execute        'We don't need the return from the DB, so ignore it.
        End With
    End If
CleanExit:
    If Not database Is Nothing And database.State = adStateOpen Then
        database.Close
    End If
    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & ": " & Err.Description
    Resume CleanExit
End Sub
```

Note that commands sent to the data source are **vulnerable to SQL injection**, either intentional or unintentional. In general, SQL statements should not be created by concatenating user input of any kind. Instead, they should be parameterized (see Creating parameterized commands).

第38章：属性

第38.1节：VB_PredeclaredId

创建类的全局默认实例。默认实例通过类名访问。

声明

```
VERSION 1.0 CLASS
BEGIN
MultiUse = -1  'True
END
Attribute VB_Name = "Class1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Public Function GiveMeATwo() As Integer
    GiveMeATwo = 2
End Function
```

调用

```
Debug.Print Class1.GiveMeATwo
```

在某些方面，这模拟了其他语言中静态类的行为，但与其他语言不同的是，你仍然可以创建该类的实例。

```
Dim cls As Class1
Set cls = New Class1
Debug.Print cls.GiveMeATwo
```

第38.2节：VB_[Var]UserMemId

VB_VarUserMemId（用于模块范围变量）和VB_UserMemId（用于过程）属性在VBA中主要用于两件事。

指定类的默认成员

一个封装了Collection的List类会希望有一个Item属性，这样客户端代码可以这样写：

```
For i = 1 To myList.Count 'VBA集合对象是从1开始的
    Debug.Print myList.Item(i)
Next
```

但是在Item属性上设置了VB_UserMemId属性为0后，客户端代码可以这样写：

```
对于 i = 1 到 myList.Count 'VBA 集合对象是从 1 开始的
    Debug.Print myList(i)
Next
```

在任何给定的类中，只有一个成员可以合法地拥有`VB_UserMemId = 0`。对于属性，请在

`Get`访问器中指定该属性：

Chapter 38: Attributes

Section 38.1: VB_PredeclaredId

Creates a Global Default Instance of a class. The default instance is accessed via the name of the class.

Declaration

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "Class1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Public Function GiveMeATwo() As Integer
    GiveMeATwo = 2
End Function
```

Call

```
Debug.Print Class1.GiveMeATwo
```

In some ways, this simulates the behavior of static classes in other languages, but unlike other languages, you can still create an instance of the class.

```
Dim cls As Class1
Set cls = New Class1
Debug.Print cls.GiveMeATwo
```

Section 38.2: VB_[Var]UserMemId

`VB_VarUserMemId` (for module-scope variables) and `VB_UserMemId` (for procedures) attributes are used in VBA mostly for two things.

Specifying the default member of a class

A `List` class that would encapsulate a `Collection` would want to have an `Item` property, so the client code can do this:

```
For i = 1 To myList.Count 'VBA Collection Objects are 1-based
    Debug.Print myList.Item(i)
Next
```

But with a `VB_UserMemId` attribute set to 0 on the `Item` property, the client code can do this:

```
For i = 1 To myList.Count 'VBA Collection Objects are 1-based
    Debug.Print myList(i)
Next
```

Only one member can legally have `VB_UserMemId = 0` in any given class. For properties, specify the attribute in the `Get` accessor:

```
Option Explicit
Private internal As New Collection

Public Property Get Count() As Long
    Count = internal.Count
End Property

Public Property Get Item(ByVal index As Long) As Variant
Attribute Item.VB_Description = "获取或设置指定索引处的元素。"
Attribute Item.VB_UserMemId = 0
'获取指定索引处的元素。
    Item = internal(index)
End Property

Public Property Let Item(ByVal index As Long, ByVal value As Variant)
'设置指定索引处的元素。
    With internal
        If index = .Count + 1 Then
            .Add item:=value
        ElseIf index = .Count Then
            .Remove index
            .Add item:=value
        ElseIf index < .Count Then
            .Remove index
            .Add item:=value, before:=index
        End If
    End With
End Property
```

使类可以使用For Each循环结构迭代通过魔法值-4，VB_User

MemId属性告诉VBA该成员返回一个枚举器——这允许客户端代码执行如下操作：

```
Dim item As Variant
For Each item In myList
    Debug.Print item
Next
```

实现此方法最简单的方式是调用内部/封装的Collection上的隐藏[_NewEnum]属性获取器；由于前导下划线使其成为非法的VBA标识符，因此该标识符需要用方括号括起来：

```
Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_Description = "获取一个用于遍历列表的枚举器。"
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40" '会在VB6中隐藏该成员，VBA不支持。
'获取一个用于遍历列表的枚举器。
    Set NewEnum = internal.[_NewEnum]
End Property
```

第38.3节：VB_Exposed

控制类的实例化特性。

```
属性 VB_Exposed = False
```

使类为Private。不能在当前项目之外访问。


```
Option Explicit
Private internal As New Collection

Public Property Get Count() As Long
    Count = internal.Count
End Property

Public Property Get Item(ByVal index As Long) As Variant
Attribute Item.VB_Description = "Gets or sets the element at the specified index."
Attribute Item.VB_UserMemId = 0
'Gets the element at the specified index.
    Item = internal(index)
End Property

Public Property Let Item(ByVal index As Long, ByVal value As Variant)
'Sets the element at the specified index.
    With internal
        If index = .Count + 1 Then
            .Add item:=value
        ElseIf index = .Count Then
            .Remove index
            .Add item:=value
        ElseIf index < .Count Then
            .Remove index
            .Add item:=value, before:=index
        End If
    End With
End Property
```

Making a class iteratable with a `For Each` loop construct

With the magic value `-4`, the `VB_UserMemId` attribute tells VBA that this member yields an enumerator - which allows the client code to do this:

```
Dim item As Variant
For Each item In myList
    Debug.Print item
Next
```

The easiest way to implement this method is by calling the hidden `[_NewEnum]` property getter on an internal/encapsulated `Collection`; the identifier needs to be enclosed in square brackets because of the leading underscore that makes it an illegal VBA identifier:

```
Public Property Get NewEnum() As IUnknown
Attribute NewEnum.VB_Description = "Gets an enumerator that iterates through the List."
Attribute NewEnum.VB_UserMemId = -4
Attribute NewEnum.VB_MemberFlags = "40" 'would hide the member in VB6. not supported in VBA.
'Gets an enumerator that iterates through the List.
    Set NewEnum = internal.[_NewEnum]
End Property
```

Section 38.3: VB_Exposed

Controls the instancing characteristics of a class.

```
Attribute VB_Exposed = False
```

Makes the class **`Private`**. It cannot be accessed outside of the current project.

```
属性 VB_Exposed = True
```

将类公开Public，允许在项目外部访问。但是，由于在VBA中忽略VB_Createable，类的实例不能被直接创建。这相当于以下VB.Net类。

```
Public Class Foo
    Friend Sub New()
    End Sub
End Class
```

为了从项目外部获取实例，必须公开一个工厂来创建实例。实现此目的的一种方法是使用常规的Public模块。

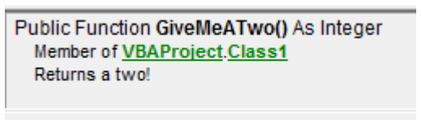
```
Public Function CreateFoo() As Foo
    CreateFoo = New Foo
End Function
```

由于公共模块可以被其他项目访问，这使我们能够创建新的`Public Not Createable`类的实例。

第38.4节：VB_Description

为类或模块成员添加文本描述，该描述会在对象资源管理器中显示。理想情况下，公共接口/API的所有公共成员都应有描述。

```
Public Function GiveMeATwo() As Integer
    Attribute GiveMeATwo.VB_Description = "返回数字2！"
    GiveMeATwo = 2
End Property
```



注意：属性的所有访问器成员（Get、Let、Set）使用相同的描述。

第38.5节：VB_Name

VB_Name指定类或模块的名称。

```
Attribute VB_Name = "Class1"
```

可以用以下方式创建该类的新实例

```
Dim myClass As Class1
myClass = new Class1
```

第38.6节：VB_GlobalNameSpace

在 VBA 中，此属性被忽略。 它没有从 VB6 移植过来。

在 VB6 中，它创建了类的默认全局实例（一个“快捷方式”），以便可以在不使用类名的情况下访问类成员。例如，DateTime（如DateTime.Now）实际上是VBA.Conversion


```
Attribute VB_Exposed = True
```

Exposes the class `Public`ly, outside of the project. However, since `VB_Createable` is ignored in VBA, instances of the class can not be created directly. This is equivalent to a the following VB.Net class.

```
Public Class Foo
    Friend Sub New()
    End Sub
End Class
```

In order to get an instance from outside the project, you must expose a factory to create instances. One way of doing this is with a regular `Public` module.

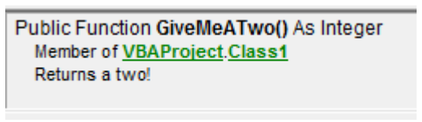
```
Public Function CreateFoo() As Foo
    CreateFoo = New Foo
End Function
```

Since public modules are accessible from other projects, this allows us to create new instances of our `Public - Not Createable` classes.

Section 38.4: VB_Description

Adds a text description to a class or module member that becomes visible in the Object Explorer. Ideally, all public members of a public interface / API should have a description.

```
Public Function GiveMeATwo() As Integer
    Attribute GiveMeATwo.VB_Description = "Returns a two!"
    GiveMeATwo = 2
End Property
```



Note: all accessor members of a property (`Get`, `Let`, `Set`) use the same description.

Section 38.5: VB_Name

VB_Name specifies the class or module name.

```
Attribute VB_Name = "Class1"
```

A new instance of this class would be created with

```
Dim myClass As Class1
myClass = new Class1
```

Section 38.6: VB_GlobalNameSpace

In VBA, this attribute is ignored. It was not ported over from VB6.

In VB6, it creates a Default Global Instance of the class (a "shortcut") so that class members can be accessed without using the class name. For example, `DateTime` (as in `DateTime.Now`) is actually part of the `VBA.Conversion`

类的一部分。

```
Debug.Print VBA.Conversion.DateTime.Now
Debug.Print DateTime.Now
```

第38.7节：VB_Createable

此属性被忽略。 它没有从 VB6 移植过来。

在 VB6 中，它与VB_Exposed属性结合使用，以控制当前项目之外的类的可访问性。

```
VB_Exposed=True
VB_Creatable=True
```

将导致一个Public Class，可以从其他项目访问，但此功能在 VBA 中不存在。

class.

```
Debug.Print VBA.Conversion.DateTime.Now
Debug.Print DateTime.Now
```

Section 38.7: VB_Createable

This attribute is ignored. It was not ported over from VB6.

In VB6, it was used in combination with the `VB_Exposed` attribute to control accessibility of classes outside of the current project.

```
VB_Exposed=True
VB_Creatable=True
```

Would result in a **`Public Class`**, that could be accessed from other projects, but this functionality does not exist in VBA.

第39章：用户窗体

第39.1节：最佳实践

用户窗体（UserForm）是一个带有设计器和默认实例的类模块。可以通过按下 Shift键访问设计器（designer）+ F7 在查看代码背后（code-behind）时，按下可以访问代码背后（code-behind） F7 在查看设计器（designer）时。

每次都使用一个新的实例。

作为一个类模块，窗体因此是一个对象的蓝图。因为窗体可以保存状态和数据，使用类的新实例比使用默认/全局实例更好：

```
使用 New UserForm1
    .Show vbModal
    如果不是.IsCancelled则
        '...
    结束 If
结束 With
```

而不是：

```
UserForm1.Show vbModal
If Not UserForm1.IsCancelled Then
    '...
End If
```

使用默认实例可能会导致细微的错误，尤其是在使用红色“X”按钮关闭窗体和/或代码中使用Unload Me时。

请在其他地方实现逻辑。

窗体应该只关注展示：一个按钮的Click事件处理程序如果连接数据库并根据用户输入运行参数化查询，就是做了太多事情。

相反，应在负责显示窗体的代码中实现应用逻辑，或者更好地，在专门的模块和过程里实现。

编写代码时应确保UserForm只负责知道如何显示和收集数据：数据来自哪里，或者数据之后如何处理，都不应是它关心的内容。

调用者不应被控件所干扰。

为窗体设计一个定义良好的模型，可以在其专用的类模块中，或者封装在窗体的代码背后——通过Property Get过程暴露模型，并让客户端代码使用它们：这使得窗体成为控件及其细节的抽象，只向客户端代码暴露相关数据。

这意味着代码看起来像这样：

```
使用 New UserForm1
```


Chapter 39: User Forms

Section 39.1: Best Practices

A `UserForm` is a class module with a designer and a **default instance**. The *designer* can be accessed by pressing Shift + F7 while viewing the *code-behind*, and the *code-behind* can be accessed by pressing F7 while viewing the *designer*.

Work with a new instance every time.

Being a *class module*, a form is therefore a *blueprint* for an *object*. Because a form can hold state and data, it's a better practice to work with a new *instance* of the class, rather than with the default/global one:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        '...
    End If
End With
```

Instead of:

```
UserForm1.Show vbModal
If Not UserForm1.IsCancelled Then
    '...
End If
```

Working with the default instance can lead to subtle bugs when the form is closed with the red "X" button and/or when `Unload Me` is used in the code-behind.

Implement the logic elsewhere.

A form should be concerned with nothing but *presentation*: a button `Click` handler that connects to a database and runs a parameterized query based on user input, is **doing too many things**.

Instead, implement the *applicative logic* in the code that's responsible for displaying the form, or even better, in dedicated modules and procedures.

Write the code in such a way that the UserForm is only ever responsible for knowing how to display and collect data: where the data comes from, or what happens with the data afterwards, is none of its concern.

Caller shouldn't be bothered with controls.

Make a well-defined *model* for the form to work with, either in its own dedicated class module, or encapsulated within the form's code-behind itself - expose the *model* with `Property Get` procedures, and have the client code work with these: this makes the form an *abstraction* over controls and their nitty-gritty details, exposing only the relevant data to the client code.

This means code that looks like this:

```
With New UserForm1
```

```
    .Show vbModal
    如果 Not .IsCancelled Then
        MsgBox .Message, vbInformation
    End If
结束于
```

而不是这样：

```
使用 New UserForm1
    .Show vbModal
    如果 Not .IsCancelled Then
        MsgBox .txtMessage.Text, vbInformation
    End If
结束于
```

处理 QueryClose 事件。

窗体通常有一个 关闭 按钮，提示/对话框有 好的 和 取消 按钮；用户也可以使用窗体的控制框（红色"X"按钮）关闭窗体，默认情况下这会销毁窗体实例（这也是每次都使用新实例的另一个好理由）。

```
使用 New UserForm1
    .Show vbModal
    如果 Not .IsCancelled 则 '如果不处理 QueryClose，可能会引发运行时错误。
        '...
    End With
End With
```

处理QueryClose事件的最简单方法是将Cancel参数设置为True，然后*隐藏*窗体而不是*关闭*它：

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Cancel = True
    Me.Hide
End Sub
```

这样"X"按钮就永远不会销毁实例，调用者可以安全地访问所有公共成员。

隐藏，不要关闭。

创建对象的代码应负责销毁它：卸载和终止自身不是窗体的责任。

避免在窗体的代码后置中使用Unload Me。改为调用Me.Hide，这样调用代码在窗体关闭时仍然可以使用它创建的对象。

给事物命名。

使用properties工具窗口（ F4 ）仔细命名窗体上的每个控件。控件的名称在代码后置中使用，因此除非你使用能够处理此问题的重构工具，否则重命名控件会破坏代码——所以一开始正确命名要比事后弄清20个

文本框TextBox12代表什么要容易得多。

传统上，UserForm控件使用匈牙利命名法前缀：


```
    .Show vbModal
    If Not .IsCancelled Then
        MsgBox .Message, vbInformation
    End If
End With
```

Instead of this:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        MsgBox .txtMessage.Text, vbInformation
    End If
End With
```

Handle the QueryClose event.

Forms typically have a `Close` button, and prompts/dialogs have `Ok` and `Cancel` buttons; the user may close the form using the form's *control box* (the red "X" button), which destroys the form instance by default (another good reason to *work with a new instance every time*).

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then 'if QueryClose isn't handled, this can raise a runtime error.
        '...
    End With
End With
```

The simplest way to handle the `QueryClose` event is to set the `Cancel` parameter to `True`, and then to *hide* the form instead of *closing* it:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Cancel = True
    Me.Hide
End Sub
```

That way the "X" button will never destroy the instance, and the caller can safely access all the public members.

Hide, don't close.

The code that creates an object should be responsible for destroying it: it's not the form's responsibility to unload and terminate itself.

Avoid using `Unload Me` in a form's code-behind. Call `Me.Hide` instead, so that the calling code can still use the object it created when the form closes.

Name things.

Use the *properties* toolwindow (F4) to carefully name each control on a form. The name of a control is used in the code-behind, so unless you're using a refactoring tool that can handle this, **renaming a control will break the code** - so it's much easier to do things right in the first place, than try to puzzle out exactly which of the 20 textboxes `TextBox12` stands for.

Traditionally, UserForm controls are named with Hungarian-style prefixes:

- lblUserName表示指示用户名的Label控件。
- txtUserName 用于TextBox 控件，用户可以在其中输入用户名。
- cboUserName 用于一个组合框（ComboBox）控件，用户可以输入或选择用户名。
- lstUserName 用于一个列表框（ListBox）控件，用户可以选择用户名。
- btnOk 或 cmdOk 用于一个标记为“确定”的按钮（Button）控件。

问题在于，例如当界面重新设计，组合框（ComboBox）变成了列表框（ListBox）时，名称需要更改以反映新的控件类型：最好根据控件所代表的内容来命名，而不是根据控件类型命名——这样可以尽可能地解耦（decouple）代码与界面。

- UserNameLabel 用于一个只读标签，表示用户名。
- UserNameInput 用于一个控件，用户可以输入或选择用户名。
- OkButton 用于一个标记为“确定”的命令按钮。

无论选择哪种风格，总比让所有控件使用默认名称要好。命名风格的一致性也是理想的。

第39.2节：处理QueryClose事件

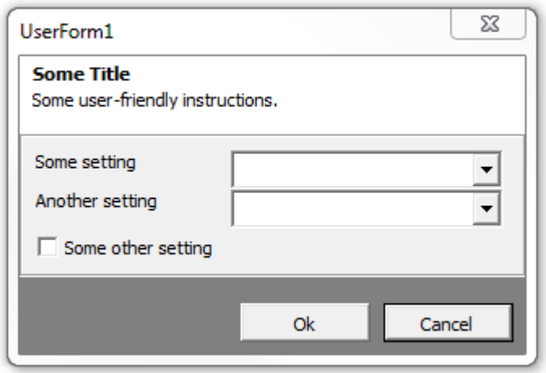
QueryClose 事件在窗体即将关闭时触发，无论是用户操作还是程序控制。参数CloseMode 包含一个VbQueryClose 枚举值，指示窗体关闭的方式：

常量	说明	值
vbFormControlMenu	窗体因用户操作而关闭	0
vbFormCode	窗体因 Unload 语句而关闭	1
vbAppWindows	Windows 会话正在结束	2
vbAppTaskManager	Windows 任务管理器正在关闭主机应用程序	3
vbFormMDIForm	VBA 中不支持	4

为了更好的可读性，最好使用这些常量，而不是直接使用它们的值。

可取消的用户窗体

给定一个带有 取消 按钮



表单的后台代码可能如下所示：

- `lblUserName` for a `Label` control that indicates a user name.
- `txtUserName` for a `TextBox` control where the user can enter a user name.
- `cboUserName` for a `ComboBox` control where the user can enter or pick a user name.
- `lstUserName` for a `ListBox` control where the user can pick a user name.
- `btnOk` or `cmdOk` for a `Button` control labelled "Ok".

The problem is that when e.g. the UI gets redesigned and a `ComboBox` changes to a `ListBox`, the name needs to change to reflect the new control type: it's better to name controls for what they represent, rather than after their control type - to *decouple* the code from the UI as much as possible.

- `UserNameLabel` for a read-only label that indicates a user name.
- `UserNameInput` for a control where the user can enter or pick a user name.
- `OkButton` for a command button labelled "Ok".

Whichever style is chosen, anything is better than leaving all controls their default names. Consistency in naming style is ideal, too.

Section 39.2: Handling QueryClose

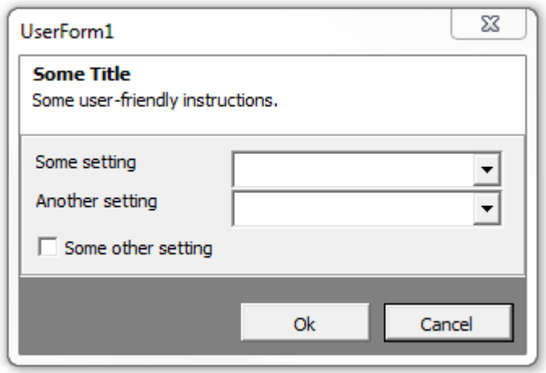
The `QueryClose` event is raised whenever a form is about to be closed, whether it's via user action or programmatically. The `CloseMode` parameter contains a `VbQueryClose` enum value that indicates how the form was closed:

Constant	Description	Value
`vbFormControlMenu`	Form is closing in response to user action	0
`vbFormCode`	Form is closing in response to an `Unload` statement	1
`vbAppWindows`	Windows session is ending	2
`vbAppTaskManager`	Windows Task Manager is closing the host application	3
`vbFormMDIForm`	Not supported in VBA	4

For better readability, it's best to use these constants instead of using their value directly.

A Cancellable UserForm

Given a form with a `Cancel` button



The form's code-behind could look like this:

```
Option Explicit
私有 类型 TView
IsCancelled 作为 布尔值
    SomeOtherSetting 作为 布尔值
    '为简洁起见省略其他属性
结束 类型
私有 这个 作为 TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

公共属性获取 SomeOtherSetting() 作为 布尔值
    SomeOtherSetting = this.SomeOtherSetting
结束属性

'...更多属性...

私有子程序 SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub

私有子程序 OkButton_Click()
    Me.Hide
End Sub

私有子程序 CancelButton_Click()
    this.IsCancelled = True
    我隐藏
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
Cancel = True
        this.IsCancelled = True
        Me.Hide
    结束 如果
结束 子程序
```

调用代码随后可以显示该表单，并知道它是否被取消：

```
公共子程序 DoSomething()
    使用 新的 UserForm1
        .Show vbModal
        如果 .IsCancelled 则退出子程序
        如果 .SomeOtherSetting 则
            '设置已启用
        否则
            '设置已禁用
        结束 如果
    结束于
End Sub
```

IsCancelled 属性在 取消 按钮被点击时，或用户使用 控制框 关闭表单时返回 True。


```
Option Explicit
Private Type TView
    IsCancelled As Boolean
    SomeOtherSetting As Boolean
    'other properties skipped for brievety
End Type
Private this As TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property

'...more properties...

Private Sub SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub

Private Sub OkButton_Click()
    Me.Hide
End Sub

Private Sub CancelButton_Click()
    this.IsCancelled = True
    Me.Hide
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
        Cancel = True
        this.IsCancelled = True
        Me.Hide
    End If
End Sub
```

The calling code could then display the form, and know whether it was cancelled:

```
Public Sub DoSomething()
    With New UserForm1
        .Show vbModal
        If .IsCancelled Then Exit Sub
        If .SomeOtherSetting Then
            'setting is enabled
        Else
            'setting is disabled
        End If
    End With
End Sub
```

The `IsCancelled` property returns `True` when the `Cancel` button is clicked, or when the user closes the form using the *control box*.

第40章：CreateObject 与 GetObject

第40.1节：演示GetObject和CreateObject

MSDN-GetObject函数

> 返回由ActiveX组件提供的对象引用。

> 当对象已有当前实例或您想用已加载文件创建对象时，使用GetObject函数。如果没有当前实例，且您不想用已加载文件启动对象，则使用CreateObject函数。

```
子程序 CreateVSGet()
    声明 ThisXLApp 为 Excel.Application '早期绑定示例
    声明 AnotherXLApp 为 Object '晚期绑定示例
    声明 ThisNewWB 为 Workbook
    声明 AnotherNewWB 为 Workbook
    声明 wb 为 Workbook

    '获取此Excel实例
    设置 ThisXLApp = GetObject(ThisWorkbook.Name).Application
    '创建另一个Excel实例
    Set AnotherXLApp = CreateObject("Excel.Application")
    使第二个实例可见
AnotherXLApp.Visible = True
    '向第二个实例添加工作簿
    设置 AnotherNewWB = AnotherXLApp.Workbooks.Add
    '向第二个实例添加工作表
    AnotherNewWB.Sheets.Add

    现在你应该有两个Excel实例打开
    第一个实例有一个工作簿：Book1
    第二个实例有一个工作簿：Book2

    让我们向第一个实例添加另一个工作簿
    设置 ThisNewWB = ThisXLApp.Workbooks.Add
    现在循环遍历工作簿并显示它们的名称
    对于每个 wb 在 ThisXLApp.Workbooks中
        Debug.Print wb.Name
    下一个
    现在第一个实例有两个工作簿：Book1 和 Book3
    如果你关闭第一个Excel实例，
    Book1 和 Book3 会关闭，但 Book2 仍然会打开

End Sub
```


Chapter 40: CreateObject vs. GetObject

Section 40.1: Demonstrating GetObject and CreateObject

MSDN-GetObject Function

> Returns a reference to an object provided by an ActiveX component.

> Use the GetObject function when there is a current instance of the object or if you want to create the object with a file already loaded. If there is no current instance, and you don't want the object started with a file loaded, use the CreateObject function.

```
Sub CreateVSGet()
    Dim ThisXLApp As Excel.Application 'An example of early binding
    Dim AnotherXLApp As Object 'An example of late binding
    Dim ThisNewWB As Workbook
    Dim AnotherNewWB As Workbook
    Dim wb As Workbook

    'Get this instance of Excel
    Set ThisXLApp = GetObject(ThisWorkbook.Name).Application
    'Create another instance of Excel
    Set AnotherXLApp = CreateObject("Excel.Application")
    'Make the 2nd instance visible
    AnotherXLApp.Visible = True
    'Add a workbook to the 2nd instance
    Set AnotherNewWB = AnotherXLApp.Workbooks.Add
    'Add a sheet to the 2nd instance
    AnotherNewWB.Sheets.Add

    'You should now have 2 instances of Excel open
    'The 1st instance has 1 workbook: Book1
    'The 2nd instance has 1 workbook: Book2

    'Lets add another workbook to our 1st instance
    Set ThisNewWB = ThisXLApp.Workbooks.Add
    'Now loop through the workbooks and show their names
    For Each wb In ThisXLApp.Workbooks
        Debug.Print wb.Name
    Next
    'Now the 1st instance has 2 workbooks: Book1 and Book3
    'If you close the first instance of Excel,
    'Book1 and Book3 will close, but book2 will still be open

End Sub
```

第41章：非拉丁字符

VBA可以使用Unicode读取和写入任何语言或文字的字符串。然而，对于
标识符标记有更严格的规则。

第41.1节：VBA代码中的非拉丁文本

在电子表格单元格A1中，我们有以下阿拉伯语全字母句：

صِفْ خَلْقَ خَوْدٍ كَمِثْلِ الشَّمْسِ إِذْ بَزَغَتْ — يَحْظَى الضَّجِيعُ بِهَا نَجْلَاءَ مِعْطَارِ

VBA提供了AscW和ChrW函数来处理多字节字符代码。我们也可以使用Byte数组直接操作字符串变量：

```
Sub NonLatinStrings()

Dim rng As Range
Set rng = Range("A1")
Do Until rng = ""
    Dim MyString As String
    MyString = rng.Value

    ' AscW函数
    Dim char As String
    char = AscW(Left(MyString, 1))
Debug.Print "第一个字符 (ChrW): " & char
    Debug.Print "第一个字符（二进制）: " & BinaryFormat(char, 12)

    ' ChrW 函数
    Dim uString As String
uString = ChrW(char)
Debug.Print "字符串值（文本）: " & uString        ' 失败！显示为'?'
    Debug.Print "字符串值（AscW）: " & AscW(uString)

    ' 使用字节字符串
    Dim StringAsByt() As Byte
    StringAsByt = MyString
    声明 i 为 Long 类型
    For i = 0 To 1 Step 2
Debug.Print "字节值（十进制）: " & _
            StringAsByt(i) & "|" & StringAsByt(i + 1)
        Debug.Print "字节值（二进制）: " & _
            BinaryFormat(StringAsByt(i)) & "|" & BinaryFormat(StringAsByt(i + 1))
    Next i
Debug.Print ""

    ' 将整个字符串打印到即时窗口失败（全部显示为'?'）
Debug.Print "整个字符串" & vbNewLine & rng.Value
    Set rng = rng.Offset(1)
Loop

End Sub
```

这将为阿拉伯字母Sad生成以下输出：

> 第一个字符（ChrW）：1589
> 第一个字符（二进制）：00011000110101

Chapter 41: Non-Latin Characters

VBA can read and write strings in any language or script using Unicode. However, there are stricter rules in place for Identifier Tokens.

Section 41.1: Non-Latin Text in VBA Code

In spreadsheet cell A1, we have the following Arabic pangram:

صِفْ خَلْقَ خَوْدٍ كَمِثْلِ الشَّمْسِ إِذْ بَزَغَتْ — يَحْظَى الضَّجِيعُ بِهَا نَجْلَاءَ مِعْطَارِ

VBA provides the `AscW` and `ChrW` functions to work with multi-byte character codes. We can also use `Byte` arrays to manipulate the string variable directly:

```
Sub NonLatinStrings()

Dim rng As Range
Set rng = Range("A1")
Do Until rng = ""
    Dim MyString As String
    MyString = rng.Value

    ' AscW functions
    Dim char As String
    char = AscW(Left(MyString, 1))
    Debug.Print "First char (ChrW): " & char
    Debug.Print "First char (binary): " & BinaryFormat(char, 12)

    ' ChrW functions
    Dim uString As String
    uString = ChrW(char)
    Debug.Print "String value (text): " & uString        ' Fails! Appears as '?'
    Debug.Print "String value (AscW): " & AscW(uString)

    ' Using a Byte string
    Dim StringAsByt() As Byte
    StringAsByt = MyString
    Dim i As Long
    For i = 0 To 1 Step 2
        Debug.Print "Byte values (in decimal): " & _
            StringAsByt(i) & "|" & StringAsByt(i + 1)
        Debug.Print "Byte values (binary): " & _
            BinaryFormat(StringAsByt(i)) & "|" & BinaryFormat(StringAsByt(i + 1))
    Next i
    Debug.Print ""

    ' Printing the entire string to the immediate window fails (all '?'s)
    Debug.Print "Whole String" & vbNewLine & rng.Value
    Set rng = rng.Offset(1)
Loop

End Sub
```

This produces the following output for the Arabic Letter Sad:

> First char (ChrW): 1589
> First char (binary): 00011000110101

> 字符串值（文本）：?
> 字符串值（AscW）：1589
> 字节值（十进制）：53|6
> 字节值（二进制）：00110101|00000110
>
> 整个字符串
> ??? ????? ????? ??????? ??????? ??? ??????? — ????? ???????? ???? ??????? ???????

请注意，尽管字符串函数能正常工作，VBA仍无法将非拉丁文本打印到立即窗口。
这是IDE的限制，而非语言本身的限制。

第41.2节：非拉丁标识符和语言覆盖范围

VBA标识符（变量和函数名）可以使用拉丁字母，也可能支持日语、韩语、简体中文和繁体中文字符。

扩展拉丁字母对许多语言提供了完整支持：
英语、法语、西班牙语、德语、意大利语、布列塔尼语、加泰罗尼亚语、丹麦语、爱沙尼亚语、芬兰语、冰岛语、印尼语、爱尔兰语、逻辑语、马普杜恩语、挪威语、葡萄牙语、苏格兰盖尔语、瑞典语、他加禄语

某些语言仅部分涵盖：
阿塞拜疆语、克罗地亚语、捷克语、世界语、匈牙利语、拉脱维亚语、立陶宛语、波兰语、罗马尼亚语、塞尔维亚语、斯洛伐克语、斯洛文尼亚语、土耳其语、约鲁巴语、威尔士语

某些语言几乎没有或完全没有涵盖：
阿拉伯语、保加利亚语、切罗基语、不丹语、希腊语、印地语、马其顿语、马拉雅拉姆语、蒙古语、俄语、梵语、泰语、藏语、乌尔都语、维吾尔语

以下变量声明均有效：

```
Dim Yec'hed As String '布列塔尼语
Dim «Dóna» As String '加泰罗尼亚语
Dim fræk As String '丹麦语
Dim tšellomängija As String '爱沙尼亚语
Dim Törkylempijävongahdus As String '芬兰语
Dim j'examine As String '法语
Dim Paß As String '德语
Dim þjófum As String '冰岛语
Dim hÓighe As String '爱尔兰语
Dim sofybakni As String '逻辑邦语（.o'不适用）
Dim ñizol As String '马普切语
Dim Vår As String '挪威语
Dim «brações» As String '葡萄牙语
Dim d'fhàg As String '苏格兰盖尔语
```

请注意，在VBA集成开发环境中，变量名中的单个撇号不会将该行变为注释（与Stack Overflow上的情况不同）。

此外，使用两个角度符号表示引号的语言«»允许在变量名中使用这些符号，尽管""类型的引号不允许。

> String value (text): ?
> String value (AscW): 1589
> Byte values (in decimal): 53|6
> Byte values (binary): 00110101|00000110
>
> Whole String
> ??? ????? ????? ??????? ??????? ??? ??????? — ????? ???????? ???? ??????? ???????

Note that VBA is unable to print non-Latin text to the immediate window even though the string functions work correctly. This is a limitation of the IDE and not the language.

Section 41.2: Non-Latin Identifiers and Language Coverage

VBA Identifiers (variable and function names) can use the Latin script and may also be able to use Japanese, Korean, Simplified Chinese, and Traditional Chinese scripts.

The extended Latin script has full coverage for many languages:
English, French, Spanish, German, Italian, Breton, Catalan, Danish, Estonian, Finnish, Icelandic, Indonesian, Irish, Lojban, Mapudungun, Norwegian, Portuguese, Scottish Gaelic, Swedish, Tagalog

Some languages are only partially covered:
Azeri, Croatian, Czech, Esperanto, Hungarian, Latvian, Lithuanian, Polish, Romanian, Serbian, Slovak, Slovenian, Turkish, Yoruba, Welsh

Some languages have little or no coverage:
Arabic, Bulgarian, Cherokee, Dzongkha, Greek, Hindi, Macedonian, Malayalam, Mongolian, Russian, Sanskrit, Thai, Tibetan, Urdu, Uyghur

The following variable declarations are all valid:

```
Dim Yec'hed As String 'Breton
Dim «Dóna» As String 'Catalan
Dim fræk As String 'Danish
Dim tšellomängija As String 'Estonian
Dim Törkylempijävongahdus As String 'Finnish
Dim j'examine As String 'French
Dim Paß As String 'German
Dim þjófum As String 'Icelandic
Dim hÓighe As String 'Irish
Dim sofybakni As String 'Lojban (.o'i does not work)
Dim ñizol As String 'Mapudungun
Dim Vår As String 'Norwegian
Dim «brações» As String 'Portuguese
Dim d'fhàg As String 'Scottish Gaelic
```

Note that in the VBA IDE, a single apostrophe within a variable name does not turn the line into a comment (as it does on Stack Overflow).

Also, languages that use two angles to indicate a quote «» are allowed to use those in variable names desipte the fact that the ""-type quotes are not.

第42章：API调用

API代表应用程序编程接口

VBA的API意味着一组允许与操作系统直接交互的方法

可以通过执行DLL文件中定义的过程来进行系统调用

第42.1节：Mac API

微软官方不支持API但通过一些研究可以在网上找到更多声明

Mac版Office 2016是沙盒环境

与支持VBA的其他版本Office应用不同，Mac版Office 2016应用是沙盒环境。

沙盒限制应用访问应用容器外的资源。这会影响任何涉及文件访问或跨进程通信的加载项或宏。你可以通过使用下一节描述的新命令来最大限度地减少沙盒的影响。Mac版Office 2016的新VBA命令

以下VBA命令是Mac版Office 2016中新增加且独有的。

命令	用途
GrantAccessToMultipleFiles	请求用户许可一次访问多个文件
AppleScriptTask	从VB调用外部AppleScript脚本
MAC_OFFICE_VERSION	编译时不同Mac Office版本之间的IFDEF

Mac版Office 2011

```
Private Declare Function system Lib "libc.dylib" (ByVal command As String) As Long
Private Declare Function popen Lib "libc.dylib" (ByVal command As String, ByVal mode As String) As
Long
Private Declare Function pclose Lib "libc.dylib" (ByVal file As Long) As Long
Private Declare Function fread Lib "libc.dylib" (ByVal outStr As String, ByVal size As Long, ByVal
items As Long, ByVal stream As Long) As Long
Private Declare Function feof Lib "libc.dylib" (ByVal file As Long) As Long
```

Mac版Office 2016

```
Private Declare PtrSafe Function popen Lib "libc.dylib" (ByVal command As String, ByVal mode As
String) As LongPtr
Private Declare PtrSafe Function pclose Lib "libc.dylib" (ByVal file As LongPtr) As Long
Private Declare PtrSafe Function fread Lib "libc.dylib" (ByVal outStr As String, ByVal size As
LongPtr, ByVal items As LongPtr, ByVal stream As LongPtr) As Long
Private Declare PtrSafe Function feof Lib "libc.dylib" (ByVal file As LongPtr) As LongPtr
```

第42.2节：获取总显示器数量和屏幕分辨率

```
Option Explicit

'GetSystemMetrics32 信息：http://msdn.microsoft.com/en-us/library/ms724385(VS.85).aspx
#If Win64 Then
    Private Declare PtrSafe Function GetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
#ElseIf Win32 Then
```


Chapter 42: API Calls

API stands for Application Programming Interface

API's for VBA imply a set of methods that allow direct interaction with the operating system

System calls can be made by executing procedures defined in DLL files

Section 42.1: Mac APIs

Microsoft doesn't officially support APIs but with some research more declarations can be found online

Office 2016 for Mac is sandboxed

Unlike other versions of Office apps that support VBA, Office 2016 for Mac apps are sandboxed.

Sandboxing restricts the apps from accessing resources outside the app container. This affects any add-ins or macros that involve file access or communication across processes. You can minimize the effects of sandboxing by using the new commands described in the following section. New VBA commands for Office 2016 for Mac

The following VBA commands are new and unique to Office 2016 for Mac.

Command	Use to
GrantAccessToMultipleFiles	Request a user's permission to access multiple files at once
AppleScriptTask	Call external AppleScript scripts from VB
MAC_OFFICE_VERSION	IFDEF between different Mac Office versions at compile time

Office 2011 for Mac

```
Private Declare Function system Lib "libc.dylib" (ByVal command As String) As Long
Private Declare Function popen Lib "libc.dylib" (ByVal command As String, ByVal mode As String) As
Long
Private Declare Function pclose Lib "libc.dylib" (ByVal file As Long) As Long
Private Declare Function fread Lib "libc.dylib" (ByVal outStr As String, ByVal size As Long, ByVal
items As Long, ByVal stream As Long) As Long
Private Declare Function feof Lib "libc.dylib" (ByVal file As Long) As Long
```

Office 2016 for Mac

```
Private Declare PtrSafe Function popen Lib "libc.dylib" (ByVal command As String, ByVal mode As
String) As LongPtr
Private Declare PtrSafe Function pclose Lib "libc.dylib" (ByVal file As LongPtr) As Long
Private Declare PtrSafe Function fread Lib "libc.dylib" (ByVal outStr As String, ByVal size As
LongPtr, ByVal items As LongPtr, ByVal stream As LongPtr) As Long
Private Declare PtrSafe Function feof Lib "libc.dylib" (ByVal file As LongPtr) As LongPtr
```

Section 42.2: Get total monitors and screen resolution

```
Option Explicit

'GetSystemMetrics32 info: http://msdn.microsoft.com/en-us/library/ms724385(VS.85).aspx
#If Win64 Then
    Private Declare PtrSafe Function GetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
#ElseIf Win32 Then
```

```
    Private Declare Function GetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics" (ByVal nIndex
As Long) As Long
#End If

'VBA 封装函数：
Public Function dllGetMonitors() As Long
    Const SM_CMONITORS = 80
    dllGetMonitors = GetSystemMetrics32(SM_CMONITORS)
End Function

Public Function dllGetHorizontalResolution() As Long
    Const SM_CXVIRTUALSCREEN = 78
    dllGetHorizontalResolution = GetSystemMetrics32(SM_CXVIRTUALSCREEN)
End Function

Public Function dllGetVerticalResolution() As Long
    Const SM_CYVIRTUALSCREEN = 79
    dllGetVerticalResolution = GetSystemMetrics32(SM_CYVIRTUALSCREEN)
End Function

Public Sub ShowDisplayInfo()
Debug.Print "总显示器数量: " & vbTab & vbTab & dllGetMonitors
    Debug.Print "水平分辨率: " & vbTab & dllGetHorizontalResolution
    Debug.Print "垂直分辨率: " & vbTab & dllGetVerticalResolution

    '总显示器数量:         1
    '水平分辨率:   1920
    '垂直分辨率:     1080
End Sub
```

第42.3节：FTP和区域API

modFTP

```
Option Explicit
Option Compare Text
Option Private Module

'http://msdn.microsoft.com/en-us/library/aa384180(v=VS.85).aspx
'http://www.dailydoseofexcel.com/archives/2006/01/29/ftp-via-vba/
'http://www.15seconds.com/issue/981203.htm

'打开Internet对象
Private Declare Function InternetOpen Lib "wininet.dll" Alias "InternetOpenA" ( _
    ByVal sAgent As String, _
    ByVal lAccessType As Long, _
    ByVal sProxyName As String, _
    ByVal sProxyBypass As String, _
    ByVal lFlags As Long _
) As Long
'例如: lngINet = InternetOpen("MyFTP 控制", 1, vbNullString, vbNullString, 0)

'连接到网络
Private Declare Function InternetConnect Lib "wininet.dll" Alias "InternetConnectA" ( _
    ByVal hInternetSession As Long, _
    ByVal sServerName As String, _
    ByVal nServerPort As Integer, _
    ByVal sUsername As String, _
    ByVal sPassword As String, _
    ByVal lService As Long, _
```



```
    Private Declare Function GetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics" (ByVal nIndex
As Long) As Long
#End If

'VBA Wrappers:
Public Function dllGetMonitors() As Long
    Const SM_CMONITORS = 80
    dllGetMonitors = GetSystemMetrics32(SM_CMONITORS)
End Function

Public Function dllGetHorizontalResolution() As Long
    Const SM_CXVIRTUALSCREEN = 78
    dllGetHorizontalResolution = GetSystemMetrics32(SM_CXVIRTUALSCREEN)
End Function

Public Function dllGetVerticalResolution() As Long
    Const SM_CYVIRTUALSCREEN = 79
    dllGetVerticalResolution = GetSystemMetrics32(SM_CYVIRTUALSCREEN)
End Function

Public Sub ShowDisplayInfo()
    Debug.Print "Total monitors: " & vbTab & vbTab & dllGetMonitors
    Debug.Print "Horizontal Resolution: " & vbTab & dllGetHorizontalResolution
    Debug.Print "Vertical Resolution: " & vbTab & dllGetVerticalResolution

    'Total monitors:         1
    'Horizontal Resolution:  1920
    'Vertical Resolution:    1080
End Sub
```

Section 42.3: FTP and Regional APIs

modFTP

```
Option Explicit
Option Compare Text
Option Private Module

'http://msdn.microsoft.com/en-us/library/aa384180(v=VS.85).aspx
'http://www.dailydoseofexcel.com/archives/2006/01/29/ftp-via-vba/
'http://www.15seconds.com/issue/981203.htm

'Open the Internet object
Private Declare Function InternetOpen Lib "wininet.dll" Alias "InternetOpenA" ( _
    ByVal sAgent As String, _
    ByVal lAccessType As Long, _
    ByVal sProxyName As String, _
    ByVal sProxyBypass As String, _
    ByVal lFlags As Long _
) As Long
'ex: lngINet = InternetOpen("MyFTP Control", 1, vbNullString, vbNullString, 0)

'Connect to the network
Private Declare Function InternetConnect Lib "wininet.dll" Alias "InternetConnectA" ( _
    ByVal hInternetSession As Long, _
    ByVal sServerName As String, _
    ByVal nServerPort As Integer, _
    ByVal sUsername As String, _
    ByVal sPassword As String, _
    ByVal lService As Long, _
```

```
    ByVal lFlags As Long, _
    ByVal lContext As Long _
) As Long
'例如: lngINetConn = InternetConnect(lngINet, "ftp.microsoft.com", 0, "anonymous",
"wally@wallyworld.com", 1, 0, 0)

获取文件
Private Declare Function FtpGetFile Lib "wininet.dll" Alias "FtpGetFileA" ( _
    ByVal hFtpSession As Long, _
    ByVal lpszRemoteFile As String, _
    ByVal lpszNewFile As String, _
    ByVal fFailIfExists As Boolean, _
    ByVal dwFlagsAndAttributes As Long, _
    ByVal dwFlags As Long, _
    ByVal dwContext As Long _
) As Boolean
'例如: blnRC = FtpGetFile(lngINetConn, "dirmap.txt", "c:|dirmap.txt", 0, 0, 1, 0)

'发送文件
Private Declare Function FtpPutFile Lib "wininet.dll" Alias "FtpPutFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszLocalFile As String, _
    ByVal lpszRemoteFile As String, _
    ByVal dwFlags As Long, ByVal dwContext As Long _
) As Boolean
'例如: blnRC = FtpPutFile(lngINetConn, "c:\dirmap.txt", "dirmap.txt", 1, 0)

'删除文件
Private Declare Function FtpDeleteFile Lib "wininet.dll" Alias "FtpDeleteFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszFileName As String _
) As Boolean
'示例: blnRC = FtpDeleteFile(lngINetConn, "test.txt")

'关闭 Internet 对象
Private Declare Function InternetCloseHandle Lib "wininet.dll" (ByVal hInet As Long) As Integer
'示例: InternetCloseHandle lngINetConn
'示例: InternetCloseHandle lngINet



Private Declare Function FtpFindFirstFile Lib "wininet.dll" Alias "FtpFindFirstFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszSearchFile As String, _
    lpFindFileData As WIN32_FIND_DATA, _
    ByVal dwFlags As Long, _
    ByVal dwContent As Long _
) As Long
Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
End Type
Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
nFileSizeLow As Long
```



```
    ByVal lFlags As Long, _
    ByVal lContext As Long _
) As Long
'ex: lngINetConn = InternetConnect(lngINet, "ftp.microsoft.com", 0, "anonymous",
"wally@wallyworld.com", 1, 0, 0)

'Get a file
Private Declare Function FtpGetFile Lib "wininet.dll" Alias "FtpGetFileA" ( _
    ByVal hFtpSession As Long, _
    ByVal lpszRemoteFile As String, _
    ByVal lpszNewFile As String, _
    ByVal fFailIfExists As Boolean, _
    ByVal dwFlagsAndAttributes As Long, _
    ByVal dwFlags As Long, _
    ByVal dwContext As Long _
) As Boolean
'ex: blnRC = FtpGetFile(lngINetConn, "dirmap.txt", "c:\dirmap.txt", 0, 0, 1, 0)

'Send a file
Private Declare Function FtpPutFile Lib "wininet.dll" Alias "FtpPutFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszLocalFile As String, _
    ByVal lpszRemoteFile As String, _
    ByVal dwFlags As Long, ByVal dwContext As Long _
) As Boolean
'ex: blnRC = FtpPutFile(lngINetConn, "c:\dirmap.txt", "dirmap.txt", 1, 0)

'Delete a file
Private Declare Function FtpDeleteFile Lib "wininet.dll" Alias "FtpDeleteFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszFileName As String _
) As Boolean
'ex: blnRC = FtpDeleteFile(lngINetConn, "test.txt")

'Close the Internet object
Private Declare Function InternetCloseHandle Lib "wininet.dll" (ByVal hInet As Long) As Integer
'ex: InternetCloseHandle lngINetConn
'ex: InternetCloseHandle lngINet



Private Declare Function FtpFindFirstFile Lib "wininet.dll" Alias "FtpFindFirstFileA" _
( _
    ByVal hFtpSession As Long, _
    ByVal lpszSearchFile As String, _
    lpFindFileData As WIN32_FIND_DATA, _
    ByVal dwFlags As Long, _
    ByVal dwContent As Long _
) As Long
Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
End Type
Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
        nFileSizeLow As Long
```

```
dwReserved0 作为Long
        dwReserved1 作为Long
        cFileName 作为字符串 * MAX_FTP_PATH
        cAlternate 作为字符串 * 14
结束 类型
'示例: lngHINet = FtpFindFirstFile(lngINetConn, "*.*", pData, 0, 0)

Private Declare Function InternetFindNextFile Lib "wininet.dll" Alias "InternetFindNextFileA" _
( _
    ByVal hFind As Long, _
    lpvFindData As WIN32_FIND_DATA _
) As Long
'示例: blnRC = InternetFindNextFile(lngHINet, pData)


Public Sub showLatestFTPVersion()
    Dim ftpSuccess As Boolean, msg As String, lngFindFirst As Long
    Dim lngINet As Long, lngINetConn As Long
    Dim pData As WIN32_FIND_DATA
    '初始化文件名缓冲区
   pData.cFileName = String(260, 0)

   msg = "FTP 错误"
lngINet = InternetOpen("MyFTP 控制", 1, vbNullString, vbNullString, 0)
    If lngINet > 0 Then
lngINetConn = InternetConnect(lngINet, FTP_SERVER_NAME, FTP_SERVER_PORT, FTP_USER_NAME,
FTP_PASSWORD, 1, 0, 0)
        If lngINetConn > 0 Then
FtpPutFile lngINetConn, "C:\Tmp\ftp.cls", "ftp.cls", FTP_TRANSFER_BINARY, 0
                'lngFindFirst = FtpFindFirstFile(lngINetConn, "ExcelDiff.xlsm", pData, 0, 0)
            If lngINet = 0 Then
msg = "DLL 错误: " & Err.LastDllError & ", 错误编号: " & Err.Number & ", 错误
描述: " & Err.Description
            Else
msg = left(pData.cFileName, InStr(1, pData.cFileName, String(1, 0),
vbBinaryCompare) - 1)
            End If
InternetCloseHandle lngINetConn
        End If
    InternetCloseHandle lngINet
    End If
MsgBox msg
End Sub
```

modRegional:

```
Option Explicit

Private Const LOCALE_SDECIMAL = &HE
Private Const LOCALE_SLIST = &HC

Private Declare Function GetLocaleInfo Lib "Kernel32" Alias "GetLocaleInfoA" (ByVal Locale As Long,
ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As Long
Private Declare Function SetLocaleInfo Lib "Kernel32" Alias "SetLocaleInfoA" (ByVal Locale As Long,
ByVal LCType As Long, ByVal lpLCData As String) As Boolean
Private Declare Function GetUserDefaultLCID% Lib "Kernel32" ()

Public Function getTimeSeparator() As String
    getTimeSeparator = Application.International(xlTimeSeparator)
End Function
Public Function getDateSeparator() As String
```



```
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * MAX_FTP_PATH
        cAlternate As String * 14
End Type
'ex: lngHINet = FtpFindFirstFile(lngINetConn, "*.*", pData, 0, 0)

Private Declare Function InternetFindNextFile Lib "wininet.dll" Alias "InternetFindNextFileA" _
( _
    ByVal hFind As Long, _
    lpvFindData As WIN32_FIND_DATA _
) As Long
'ex: blnRC = InternetFindNextFile(lngHINet, pData)


Public Sub showLatestFTPVersion()
    Dim ftpSuccess As Boolean, msg As String, lngFindFirst As Long
    Dim lngINet As Long, lngINetConn As Long
    Dim pData As WIN32_FIND_DATA
    'init the filename buffer
    pData.cFileName = String(260, 0)

    msg = "FTP Error"
    lngINet = InternetOpen("MyFTP Control", 1, vbNullString, vbNullString, 0)
    If lngINet > 0 Then
        lngINetConn = InternetConnect(lngINet, FTP_SERVER_NAME, FTP_SERVER_PORT, FTP_USER_NAME,
FTP_PASSWORD, 1, 0, 0)
        If lngINetConn > 0 Then
                FtpPutFile lngINetConn, "C:\Tmp\ftp.cls", "ftp.cls", FTP_TRANSFER_BINARY, 0
                'lngFindFirst = FtpFindFirstFile(lngINetConn, "ExcelDiff.xlsm", pData, 0, 0)
            If lngINet = 0 Then
                msg = "DLL error: " & Err.LastDllError & ", Error Number: " & Err.Number & ", Error
Desc: " & Err.Description
            Else
                msg = left(pData.cFileName, InStr(1, pData.cFileName, String(1, 0),
vbBinaryCompare) - 1)
            End If
            InternetCloseHandle lngINetConn
        End If
    InternetCloseHandle lngINet
    End If
        MsgBox msg
End Sub
```

modRegional:

```
Option Explicit

Private Const LOCALE_SDECIMAL = &HE
Private Const LOCALE_SLIST = &HC

Private Declare Function GetLocaleInfo Lib "Kernel32" Alias "GetLocaleInfoA" (ByVal Locale As Long,
ByVal LCType As Long, ByVal lpLCData As String, ByVal cchData As Long) As Long
Private Declare Function SetLocaleInfo Lib "Kernel32" Alias "SetLocaleInfoA" (ByVal Locale As Long,
ByVal LCType As Long, ByVal lpLCData As String) As Boolean
Private Declare Function GetUserDefaultLCID% Lib "Kernel32" ()

Public Function getTimeSeparator() As String
    getTimeSeparator = Application.International(xlTimeSeparator)
End Function
Public Function getDateSeparator() As String
```

```
    getDateSeparator = Application.International(xlDateSeparator)
End Function
Public Function getListSeparator() As String
    Dim ListSeparator As String, iRetVal1 As Long, iRetVal2 As Long, lpLCDataVar As String,
Position As Integer, Locale As Long
    Locale = GetUserDefaultLCID()
iRetVal1 = GetLocaleInfo(Locale, LOCALE_SLIST, lpLCDataVar, 0)
    ListSeparator = String$(iRetVal1, 0)
iRetVal2 = GetLocaleInfo(Locale, LOCALE_SLIST, ListSeparator, iRetVal1)
    Position = InStr(ListSeparator, Chr$(0))
    如果 Position > 0 则 ListSeparator = Left$(ListSeparator, Position - 1) 否则 ListSeparator =
vbNullString
    getListSeparator = ListSeparator
End Function

Private Sub ChangeSettingExample()  '更改显示为小数分隔符的字符设置。

    调用 SetLocalSetting(LOCALE_SDECIMAL, ",")  '更改为 ","
    停止                                        '检查控制面板以验证或使用
GetLocaleInfo API 函数
    调用 SetLocalSetting(LOCALE_SDECIMAL, ".")  '恢复更改为 "."
End Sub

Private Function SetLocalSetting(LC_CONST As Long, Setting As String) As Boolean
    调用 SetLocaleInfo(GetUserDefaultLCID(), LC_CONST, Setting)
End Function
```

第42.4节：API声明和使用

声明DLL过程以兼容不同VBA版本：

```
Option Explicit

#If Win64 Then

    Private Declare PtrSafe Sub xLib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

#ElseIf Win32 Then

    Private Declare Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

#End If
```

上述声明告诉 VBA 如何调用定义在 Kernel32.dll 文件中的 "Sleep" 函数

Win64 和 Win32 是用于条件编译的预定义常量

预定义常量

一些编译常量已经预定义。具体有哪些取决于您运行 VBA 的 Office 版本的位数。请注意，Vba7 是随着 Office 2010 一起引入的，用于支持 Office 的 64 位版本。

常量	16 位	32 位	64 位
Vba6	False 如果是 Vba6 则为 False		
Vba7	False 如果是 Vba7 则为 True		
Win16	True	False	False
Win32	False	True	True


```
    getDateSeparator = Application.International(xlDateSeparator)
End Function
Public Function getListSeparator() As String
    Dim ListSeparator As String, iRetVal1 As Long, iRetVal2 As Long, lpLCDataVar As String,
Position As Integer, Locale As Long
    Locale = GetUserDefaultLCID()
    iRetVal1 = GetLocaleInfo(Locale, LOCALE_SLIST, lpLCDataVar, 0)
    ListSeparator = String$(iRetVal1, 0)
    iRetVal2 = GetLocaleInfo(Locale, LOCALE_SLIST, ListSeparator, iRetVal1)
    Position = InStr(ListSeparator, Chr$(0))
    If Position > 0 Then ListSeparator = Left$(ListSeparator, Position - 1) Else ListSeparator =
vbNullString
    getListSeparator = ListSeparator
End Function

Private Sub ChangeSettingExample()  'change the setting of the character displayed as the decimal
separator.
    Call SetLocalSetting(LOCALE_SDECIMAL, ",")  'to change to ","
    Stop                                        'check your control panel to verify or use the
GetLocaleInfo API function
    Call SetLocalSetting(LOCALE_SDECIMAL, ".")  'to back change to "."
End Sub

Private Function SetLocalSetting(LC_CONST As Long, Setting As String) As Boolean
    Call SetLocaleInfo(GetUserDefaultLCID(), LC_CONST, Setting)
End Function
```

Section 42.4: API declaration and usage

Declaring a DLL procedure to work with different VBA versions:

```
Option Explicit

#If Win64 Then

    Private Declare PtrSafe Sub xLib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

#ElseIf Win32 Then

    Private Declare Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)

#End If
```

The above declaration tells VBA how to call the function "Sleep" defined in file Kernel32.dll

Win64 and Win32 are predefined constants used for conditional compilation

Pre-defined Constants

Some compilation constants are already pre-defined. Which ones exist will depend on the bitness of the office version you're running VBA in. Note that Vba7 was introduced alongside Office 2010 to support 64 bit versions of Office.

Constant	16 bit	32 bit	64 bit
Vba6	False	If Vba6	False
Vba7	False	If Vba7	True
Win16	True	False	False
Win32	False	True	True

Win64	False	False	True
Mac	False	If Mac	If Mac

这些常量指的是 Office 版本，而不是 Windows 版本。例如，Win32 = TRUE 表示 32 位 Office，即使操作系统是 64 位的 Windows 版本。

声明 API 时的主要区别在于 32 位和 64 位 Office 版本，后者引入了新的参数类型（详见备注部分）

> 注意：
>
> - 声明放置在模块顶部，且位于任何 Sub 或 Function 之外
> - 在标准模块中声明的过程默认是公共的
> - 要声明模块私有过程，需在声明前加上 **Private** 关键字
> - 在其他类型模块中声明的 DLL 过程对该模块是私有的

Sleep API 调用的简单示例：

```
Public Sub TestPause()

    Dim start As Double

    start = Timer

Sleep 9000      '暂停执行9秒

    Debug.Print "暂停了 " & Format(Timer - start, "#,###.000") & " 秒"

    '立即窗口结果：暂停了9.000秒

End Sub
```

建议创建一个专用的API模块，以便通过VBA包装器轻松访问系统函数——普通的VBA子程序或函数，封装实际系统调用所需的细节，如库中使用的参数及这些参数的初始化

该模块可以包含所有声明和依赖项：

- 方法签名和所需的数据结构
- 执行输入验证的包装器，确保所有参数按预期传递

要声明DLL过程，请在代码窗口的声明部分添加一个Declare语句。

如果过程返回值，则将其声明为Function：

```
声明函数 publicname 库 "libname" [别名 "alias"] [([[按值传递] 变量 [作为 类型] [,[按值传递]
变量 [作为 类型]]...])] 作为 类型
```

如果一个过程不返回值，则将其声明为Sub：

```
声明 Sub publicname Lib "libname" [Alias "alias"] [([[ByVal] 变量 [As 类型] [,[ByVal]
变量 [As 类型]]...])]
```


Win64	False	False	True
Mac	False	If Mac	If Mac

These constants refer to the Office version, not the Windows version. For example Win32 = TRUE in 32-bit Office, even if the OS is a 64-bit version of Windows.

The main difference when declaring APIs is between 32 bit and 64 bit Office versions which introduced new parameter types (see Remarks section for more details)

> Notes:
>
> - Declarations are placed at the top of the module, and outside any Subs or Functions
> - Procedures declared in standard modules are public by default
> - To declare a procedure private to a module precede the declaration with the **Private** keyword
> - DLL procedures declared in any other type of module are private to that module

Simple example for the Sleep API call:

```
Public Sub TestPause()

    Dim start As Double

    start = Timer

    Sleep 9000      'Pause execution for 9 seconds

    Debug.Print "Paused for " & Format(Timer - start, "#,###.000") & " seconds"

    'Immediate window result: Paused for 9.000 seconds

End Sub
```

It is recommended to create a dedicated API module to provide easy access to the system functions from VBA wrappers -- normal VBA Subs or Functions that encapsulate the details needed for the actual system call such as parameters used in libraries, and initialization of those parameters

The module can contain all declarations and dependencies:

- Method signatures and required data structures
- Wrappers that perform input validation, and ensure all parameters are passed as expected

To declare a DLL procedure, add a **Declare** statement to the Declarations section of the code window.

If the procedure returns a value, declare it as a **Function**:

```
Declare Function publicname Lib "libname" [Alias "alias"] [([[ByVal] variable [As type] [,[ByVal]
variable [As type]]...])] As Type
```

If a procedure does not return a value, declare it as a **Sub**:

```
Declare Sub publicname Lib "libname" [Alias "alias"] [([[ByVal] variable [As type] [,[ByVal]
variable [As type]]...])]
```

- !!

> 还需注意的是，大多数对API的无效调用会导致Excel崩溃，并可能损坏数据文件

- !!

Mac版Office 2011

```
Private Declare Function system Lib "libc.dylib" (ByVal command As String) As Long

Sub RunSafari()
    Dim result As Long
result = system("open -a Safari --args http://www.google.com")
    Debug.Print Str(result)
End Sub
```

下面的示例（Windows API - 专用模块（1和2））展示了一个包含Win64和Win32通用声明的API模块

第42.5节：Windows API - 专用模块（1/2）

```
Option Explicit

#If Win64 Then  'Win64 = True, Win32 = False, Win16 = False
    Private Declare PtrSafe Sub apiCopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (MyDest As Any,
MySource As Any, ByVal MySize As Long)
    Private Declare PtrSafe Sub apiExitProcess Lib "Kernel32" Alias "ExitProcess" (ByVal uExitCode
As Long)
    Private Declare PtrSafe Sub apiSetCursorPos Lib "User32" Alias "SetCursorPos" (ByVal X As
Integer, ByVal Y As Integer)
    Private Declare PtrSafe Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As
Long)
    Private Declare PtrSafe Function apiAttachThreadInput Lib "User32" Alias "AttachThreadInput"
(ByVal idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
    Private Declare PtrSafe Function apiBringWindowToTop Lib "User32" Alias "BringWindowToTop"
(ByVal lngHWnd As Long) As Long
    Private Declare PtrSafe Function apiCloseWindow Lib "User32" Alias "CloseWindow" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiDestroyWindow Lib "User32" Alias "DestroyWindow" (ByVal
hWnd As Long) As Boolean
    Private Declare PtrSafe Function apiEndDialog Lib "User32" Alias "EndDialog" (ByVal hWnd As
Long, ByVal result As Long) As Boolean
    Private Declare PtrSafe Function apiEnumChildWindows Lib "User32" Alias "EnumChildWindows"
(ByVal hWndParent As Long, ByVal pEnumProc As Long, ByVal lParam As Long) As Long
    Private Declare PtrSafe Function apiExitWindowsEx Lib "User32" Alias "ExitWindowsEx" (ByVal
uFlags As Long, ByVal dwReserved As Long) As Long
    Private Declare PtrSafe Function apiFindExecutable Lib "Shell32" Alias "FindExecutableA" (ByVal
lpFile As String, ByVallpDirectory As String, ByVal lpResult As String) As Long
    Private Declare PtrSafe Function apiFindWindow Lib "User32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpWindowName As String) As Long
    Private Declare PtrSafe Function apiFindWindowEx Lib "User32" Alias "FindWindowExA" (ByVal
hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
    Private Declare PtrSafe Function apiGetActiveWindow Lib "User32" Alias "GetActiveWindow" () As
Long
    Private Declare PtrSafe Function apiGetClassNameA Lib "User32" Alias "GetClassNameA" (ByVal
hWnd As Long, ByVal szClassName As String, ByVal lLength As Long) As Long
    Private Declare PtrSafe Function apiGetCommandLine Lib "Kernel32" Alias "GetCommandLineW" () As
Long
```


- !!

> Also of note is that **most invalid calls to the API's will crash Excel**, and possibly corrupt data files

- !!

Office 2011 for Mac

```
Private Declare Function system Lib "libc.dylib" (ByVal command As String) As Long

Sub RunSafari()
    Dim result As Long
    result = system("open -a Safari --args http://www.google.com")
    Debug.Print Str(result)
End Sub
```

The examples bellow (Windows API - Dedicated Module (1 and 2)) show an API module that includes common declarations for Win64 and Win32

Section 42.5: Windows API - Dedicated Module (1 of 2)


```
Option Explicit

#If Win64 Then  'Win64 = True, Win32 = False, Win16 = False
    Private Declare PtrSafe Sub apiCopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (MyDest As Any,
MySource As Any, ByVal MySize As Long)
    Private Declare PtrSafe Sub apiExitProcess Lib "Kernel32" Alias "ExitProcess" (ByVal uExitCode
As Long)
    Private Declare PtrSafe Sub apiSetCursorPos Lib "User32" Alias "SetCursorPos" (ByVal X As
Integer, ByVal Y As Integer)
    Private Declare PtrSafe Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As
Long)
    Private Declare PtrSafe Function apiAttachThreadInput Lib "User32" Alias "AttachThreadInput"
(ByVal idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
    Private Declare PtrSafe Function apiBringWindowToTop Lib "User32" Alias "BringWindowToTop"
(ByVal lngHWnd As Long) As Long
    Private Declare PtrSafe Function apiCloseWindow Lib "User32" Alias "CloseWindow" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiDestroyWindow Lib "User32" Alias "DestroyWindow" (ByVal
hWnd As Long) As Boolean
    Private Declare PtrSafe Function apiEndDialog Lib "User32" Alias "EndDialog" (ByVal hWnd As
Long, ByVal result As Long) As Boolean
    Private Declare PtrSafe Function apiEnumChildWindows Lib "User32" Alias "EnumChildWindows"
(ByVal hWndParent As Long, ByVal pEnumProc As Long, ByVal lParam As Long) As Long
    Private Declare PtrSafe Function apiExitWindowsEx Lib "User32" Alias "ExitWindowsEx" (ByVal
uFlags As Long, ByVal dwReserved As Long) As Long
    Private Declare PtrSafe Function apiFindExecutable Lib "Shell32" Alias "FindExecutableA" (ByVal
lpFile As String, ByVallpDirectory As String, ByVal lpResult As String) As Long
    Private Declare PtrSafe Function apiFindWindow Lib "User32" Alias "FindWindowA" (ByVal
lpClassName As String, ByVal lpWindowName As String) As Long
    Private Declare PtrSafe Function apiFindWindowEx Lib "User32" Alias "FindWindowExA" (ByVal
hWnd1 As Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
    Private Declare PtrSafe Function apiGetActiveWindow Lib "User32" Alias "GetActiveWindow" () As
Long
    Private Declare PtrSafe Function apiGetClassNameA Lib "User32" Alias "GetClassNameA" (ByVal
hWnd As Long, ByVal szClassName As String, ByVal lLength As Long) As Long
    Private Declare PtrSafe Function apiGetCommandLine Lib "Kernel32" Alias "GetCommandLineW" () As
Long
```

```
    Private Declare PtrSafe Function apiGetCommandLineParams Lib "Kernel32" Alias "GetCommandLineA"
() As Long
    Private Declare PtrSafe Function apiGetDiskFreeSpaceEx Lib "Kernel32" Alias
"GetDiskFreeSpaceExA" (ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As Currency,
lpTotalNumberOfBytes As Currency, lpTotalNumberOfFreeBytes As Currency) As Long
    Private Declare PtrSafe Function apiGetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal
nDrive As String) As Long
    Private Declare PtrSafe Function apiGetExitCodeProcess Lib "Kernel32" Alias
"GetExitCodeProcess" (ByVal hProcess As Long, lpExitCode As Long) As Long
    Private Declare PtrSafe Function apiGetForegroundWindow Lib "User32" Alias
"GetForegroundWindow" () As Long
    Private Declare PtrSafe Function apiGetFrequency Lib "Kernel32" Alias
"QueryPerformanceFrequency" (cyFrequency As Currency) As Long
    Private Declare PtrSafe Function apiGetLastError Lib "Kernel32" Alias "GetLastError" () As
Integer
    Private Declare PtrSafe Function apiGetParent Lib "User32" Alias "GetParent" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiGetSystemMetrics Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
    Private Declare PtrSafe Function apiGetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
    Private Declare PtrSafe Function apiGetTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long
    Private Declare PtrSafe Function apiGetTickCountMs Lib "Kernel32" Alias "GetTickCount" () As
Long
    Private Declare PtrSafe Function apiGetUserName Lib "AdvApi32" Alias "GetUserNameA" (ByVal
lpBuffer As String, nSize As Long) As Long
    Private Declare PtrSafe Function apiGetWindow Lib "User32" Alias "GetWindow" (ByVal hWnd As
Long, ByVal wCmd As Long) As Long
    Private Declare PtrSafe Function apiGetWindowRect Lib "User32" Alias "GetWindowRect" (ByVal
hWnd As Long, lpRect As winRect) As Long
    Private Declare PtrSafe Function apiGetWindowText Lib "User32" Alias "GetWindowTextA" (ByVal
hWnd As Long, ByVal szWindowText As String, ByVal lLength As Long) As Long
    Private Declare PtrSafe Function apiGetWindowThreadProcessId Lib "User32" Alias
"GetWindowThreadProcessId" (ByVal hWnd As Long, lpdwProcessId As Long) As Long
    Private Declare PtrSafe Function apiIsCharAlphaNumericA Lib "User32" Alias
"IsCharAlphaNumericA" (ByVal byChar As Byte) As Long
    Private Declare PtrSafe Function apiIsIconic Lib "User32" Alias "IsIconic" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiIsWindowVisible Lib "User32" Alias "IsWindowVisible" (ByVal
hWnd As Long) As Long
    Private Declare PtrSafe Function apiIsZoomed Lib "User32" Alias "IsZoomed" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiLStrCpynA Lib "Kernel32" Alias "lstrcpynA" (ByVal
pDestination As String, ByVal pSource As Long, ByVal iMaxLength As Integer) As Long
    Private Declare PtrSafe Function apiMessageBox Lib "User32" Alias "MessageBoxA" (ByVal hWnd As
Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As Long
    Private Declare PtrSafe Function apiOpenIcon Lib "User32" Alias "OpenIcon" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiOpenProcess Lib "Kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Private Declare PtrSafe Function apiPathAddBackslashByPointer Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As Long) As Long
    Private Declare PtrSafe Function apiPathAddBackslashByString Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As String) As Long
'http://msdn.microsoft.com/en-us/library/aa155716%28office.10%29.aspx
    Private Declare PtrSafe Function apiPostMessage Lib "User32" Alias "PostMessageA" (ByVal hWnd
As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Private Declare PtrSafe Function apiRegQueryValue Lib "AdvApi32" Alias "RegQueryValue" (ByVal
hKey As Long, ByVal sValueName As String, ByVal dwReserved As Long, ByRef lValueType As Long, ByVal
sValue As String, ByRef lResultLen As Long) As Long
    Private Declare PtrSafe Function apiSendMessage Lib "User32" Alias "SendMessageA" (ByVal hWnd
```



```
    Private Declare PtrSafe Function apiGetCommandLineParams Lib "Kernel32" Alias "GetCommandLineA"
() As Long
    Private Declare PtrSafe Function apiGetDiskFreeSpaceEx Lib "Kernel32" Alias
"GetDiskFreeSpaceExA" (ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As Currency,
lpTotalNumberOfBytes As Currency, lpTotalNumberOfFreeBytes As Currency) As Long
    Private Declare PtrSafe Function apiGetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal
nDrive As String) As Long
    Private Declare PtrSafe Function apiGetExitCodeProcess Lib "Kernel32" Alias
"GetExitCodeProcess" (ByVal hProcess As Long, lpExitCode As Long) As Long
    Private Declare PtrSafe Function apiGetForegroundWindow Lib "User32" Alias
"GetForegroundWindow" () As Long
    Private Declare PtrSafe Function apiGetFrequency Lib "Kernel32" Alias
"QueryPerformanceFrequency" (cyFrequency As Currency) As Long
    Private Declare PtrSafe Function apiGetLastError Lib "Kernel32" Alias "GetLastError" () As
Integer
    Private Declare PtrSafe Function apiGetParent Lib "User32" Alias "GetParent" (ByVal hWnd As
Long) As Long
    Private Declare PtrSafe Function apiGetSystemMetrics Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
    Private Declare PtrSafe Function apiGetSystemMetrics32 Lib "User32" Alias "GetSystemMetrics"
(ByVal nIndex As Long) As Long
    Private Declare PtrSafe Function apiGetTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long
    Private Declare PtrSafe Function apiGetTickCountMs Lib "Kernel32" Alias "GetTickCount" () As
Long
    Private Declare PtrSafe Function apiGetUserName Lib "AdvApi32" Alias "GetUserNameA" (ByVal
lpBuffer As String, nSize As Long) As Long
    Private Declare PtrSafe Function apiGetWindow Lib "User32" Alias "GetWindow" (ByVal hWnd As
Long, ByVal wCmd As Long) As Long
    Private Declare PtrSafe Function apiGetWindowRect Lib "User32" Alias "GetWindowRect" (ByVal
hWnd As Long, lpRect As winRect) As Long
    Private Declare PtrSafe Function apiGetWindowText Lib "User32" Alias "GetWindowTextA" (ByVal
hWnd As Long, ByVal szWindowText As String, ByVal lLength As Long) As Long
    Private Declare PtrSafe Function apiGetWindowThreadProcessId Lib "User32" Alias
"GetWindowThreadProcessId" (ByVal hWnd As Long, lpdwProcessId As Long) As Long
    Private Declare PtrSafe Function apiIsCharAlphaNumericA Lib "User32" Alias
"IsCharAlphaNumericA" (ByVal byChar As Byte) As Long
    Private Declare PtrSafe Function apiIsIconic Lib "User32" Alias "IsIconic" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiIsWindowVisible Lib "User32" Alias "IsWindowVisible" (ByVal
hWnd As Long) As Long
    Private Declare PtrSafe Function apiIsZoomed Lib "User32" Alias "IsZoomed" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiLStrCpynA Lib "Kernel32" Alias "lstrcpynA" (ByVal
pDestination As String, ByVal pSource As Long, ByVal iMaxLength As Integer) As Long
    Private Declare PtrSafe Function apiMessageBox Lib "User32" Alias "MessageBoxA" (ByVal hWnd As
Long, ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As Long
    Private Declare PtrSafe Function apiOpenIcon Lib "User32" Alias "OpenIcon" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiOpenProcess Lib "Kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Private Declare PtrSafe Function apiPathAddBackslashByPointer Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As Long) As Long
    Private Declare PtrSafe Function apiPathAddBackslashByString Lib "ShlwApi" Alias
"PathAddBackslashW" (ByVal lpszPath As String) As Long
'http://msdn.microsoft.com/en-us/library/aa155716%28office.10%29.aspx
    Private Declare PtrSafe Function apiPostMessage Lib "User32" Alias "PostMessageA" (ByVal hWnd
As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Private Declare PtrSafe Function apiRegQueryValue Lib "AdvApi32" Alias "RegQueryValue" (ByVal
hKey As Long, ByVal sValueName As String, ByVal dwReserved As Long, ByRef lValueType As Long, ByVal
sValue As String, ByRef lResultLen As Long) As Long
    Private Declare PtrSafe Function apiSendMessage Lib "User32" Alias "SendMessageA" (ByVal hWnd
```

```
As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    Private Declare PtrSafe Function apiSetActiveWindow Lib "User32" Alias "SetActiveWindow" (ByVal
hWnd As Long) As Long
    Private Declare PtrSafe Function apiSetCurrentDirectoryA Lib "Kernel32" Alias
"SetCurrentDirectoryA" (ByVal lpPathName As String) As Long
    Private Declare PtrSafe Function apiSetFocus Lib "User32" Alias "SetFocus" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiSetForegroundWindow Lib "User32" Alias
"SetForegroundWindow" (ByVal hWnd As Long) As Long
    Private Declare PtrSafe Function apiSetLocalTime Lib "Kernel32" Alias "SetLocalTime" (lpSystem
As SystemTime) As Long
    Private Declare PtrSafe Function apiSetWindowPlacement Lib "User32" Alias "SetWindowPlacement"
(ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Declare PtrSafe Function apiSetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hWnd
As Long, ByVal hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As Long, ByVal
cy As Long, ByVal wFlags As Long) As Long
    Private Declare PtrSafe Function apiSetWindowText Lib "User32" Alias "SetWindowTextA" (ByVal
hWnd As Long, ByVal lpString As String) As Long
    Private Declare PtrSafe Function apiShellExecute Lib "Shell32" Alias "ShellExecuteA" (ByVal
hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String,
ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
    Private Declare PtrSafe Function apiShowWindow Lib "User32" Alias "ShowWindow" (ByVal hWnd As
Long, ByVal nCmdShow As Long) As Long
    Private Declare PtrSafe Function apiShowWindowAsync Lib "User32" Alias "ShowWindowAsync" (ByVal
hWnd As Long, ByVal nCmdShow As Long) As Long
    Private Declare PtrSafe Function apiStrCpy Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination
As String, ByVal pSource As String, ByVal iMaxLength As Integer) As Long
    Private Declare PtrSafe Function apiStringLen Lib "Kernel32" Alias "lstrlenW" (ByVal lpString
As Long) As Long
    Private Declare PtrSafe Function apiStrTrimW Lib "ShlwApi" Alias "StrTrimW" () As Boolean
    Private Declare PtrSafe Function apiTerminateProcess Lib "Kernel32" Alias "TerminateProcess"
(ByVal hWnd As Long, ByVal uExitCode As Long) As Long
    Private Declare PtrSafe Function apiTimeGetTime Lib "Winmm" Alias "timeGetTime" () As Long
    Private Declare PtrSafe Function apiVarPtrArray Lib "MsVbVm50" Alias "VarPtr" (Var() As Any) As
Long
    Private Type browseInfo      '用于 apiBrowseForFolder
        hOwner As Long
pidlRoot 作为 Long
pszDisplayName 作为 String
lpszTitle 作为 String
ulFlags 作为 Long
lpfn 作为 Long
lParam 作为 Long
iImage 作为 Long
    结束 类型
    Private Declare PtrSafe Function apiBrowseForFolder Lib "Shell32" Alias "SHBrowseForFolderA"
(lpBrowseInfo 作为 browseInfo) 作为 Long
    Private Type CHOOSECOLOR     '由 apiChooseColor 使用；http://support.microsoft.com/kb/153929 和
http://www.cpearson.com/Excel/Colors.aspx
lStructSize 作为 Long
hWndOwner 作为 Long
        hInstance 作为 Long
        rgbResult 作为 Long
        lpCustColors 作为 String
        flags 作为 Long
lCustData 作为 长整型
        lpfnHook 作为 长整型
        lpTemplateName 作为 字符串
    结束 类型
    Private Declare PtrSafe 函数 apiChooseColor 库 "ComDlg32" 别名 "ChooseColorA"
(pChoosecolor 作为 CHOOSECOLOR )作为 长整型
                Private 类型 FindWindowParameters    '用于传入/传出钩子枚举函数参数的自定义结构体
```



```
As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    Private Declare PtrSafe Function apiSetActiveWindow Lib "User32" Alias "SetActiveWindow" (ByVal
hWnd As Long) As Long
    Private Declare PtrSafe Function apiSetCurrentDirectoryA Lib "Kernel32" Alias
"SetCurrentDirectoryA" (ByVal lpPathName As String) As Long
    Private Declare PtrSafe Function apiSetFocus Lib "User32" Alias "SetFocus" (ByVal hWnd As Long)
As Long
    Private Declare PtrSafe Function apiSetForegroundWindow Lib "User32" Alias
"SetForegroundWindow" (ByVal hWnd As Long) As Long
    Private Declare PtrSafe Function apiSetLocalTime Lib "Kernel32" Alias "SetLocalTime" (lpSystem
As SystemTime) As Long
    Private Declare PtrSafe Function apiSetWindowPlacement Lib "User32" Alias "SetWindowPlacement"
(ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Declare PtrSafe Function apiSetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hWnd
As Long, ByVal hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As Long, ByVal
cy As Long, ByVal wFlags As Long) As Long
    Private Declare PtrSafe Function apiSetWindowText Lib "User32" Alias "SetWindowTextA" (ByVal
hWnd As Long, ByVal lpString As String) As Long
    Private Declare PtrSafe Function apiShellExecute Lib "Shell32" Alias "ShellExecuteA" (ByVal
hWnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String,
ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long
    Private Declare PtrSafe Function apiShowWindow Lib "User32" Alias "ShowWindow" (ByVal hWnd As
Long, ByVal nCmdShow As Long) As Long
    Private Declare PtrSafe Function apiShowWindowAsync Lib "User32" Alias "ShowWindowAsync" (ByVal
hWnd As Long, ByVal nCmdShow As Long) As Long
    Private Declare PtrSafe Function apiStrCpy Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination
As String, ByVal pSource As String, ByVal iMaxLength As Integer) As Long
    Private Declare PtrSafe Function apiStringLen Lib "Kernel32" Alias "lstrlenW" (ByVal lpString
As Long) As Long
    Private Declare PtrSafe Function apiStrTrimW Lib "ShlwApi" Alias "StrTrimW" () As Boolean
    Private Declare PtrSafe Function apiTerminateProcess Lib "Kernel32" Alias "TerminateProcess"
(ByVal hWnd As Long, ByVal uExitCode As Long) As Long
    Private Declare PtrSafe Function apiTimeGetTime Lib "Winmm" Alias "timeGetTime" () As Long
    Private Declare PtrSafe Function apiVarPtrArray Lib "MsVbVm50" Alias "VarPtr" (Var() As Any) As
Long
    Private Type browseInfo      'used by apiBrowseForFolder
        hOwner As Long
        pidlRoot As Long
        pszDisplayName As String
        lpszTitle As String
        ulFlags As Long
        lpfn As Long
        lParam As Long
        iImage As Long
    End Type
    Private Declare PtrSafe Function apiBrowseForFolder Lib "Shell32" Alias "SHBrowseForFolderA"
(lpBrowseInfo As browseInfo) As Long
    Private Type CHOOSECOLOR     'used by apiChooseColor; http://support.microsoft.com/kb/153929 and
http://www.cpearson.com/Excel/Colors.aspx
        lStructSize As Long
        hWndOwner As Long
        hInstance As Long
        rgbResult As Long
        lpCustColors As String
        flags As Long
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
    End Type
    Private Declare PtrSafe Function apiChooseColor Lib "ComDlg32" Alias "ChooseColorA"
(pChoosecolor As CHOOSECOLOR) As Long
    Private Type FindWindowParameters    'Custom structure for passing in the parameters in/out of
```

```
钩子枚举函数的参数传递；也可以使用全局变量，但这样更好
strTitle 作为 字符串  '输入
hWnd 作为 长整型        '输出
    结束 类型                          '从所有打开的窗口列表中查找具有动态标题的特定窗口：

http://www.everythingaccess.com/tutorials.asp?ID=Bring-an-external-application-window-to-the-foregrou
nd
    Private Declare PtrSafe 函数 apiEnumWindows 库 "User32" 别名 "EnumWindows" (ByVal
lpEnumFunc 作为 LongPtr, ByVal lParam 作为 LongPtr) 作为 长整型
    Private 类型 lastInputInfo  '由 apiGetLastInputInfo 使用，获取最后输入时间
        cbSize 作为 长整型
dwTime 作为 Long
    结束 类型
    Private Declare PtrSafe Function apiGetLastInputInfo Lib "User32" Alias "GetLastInputInfo"
(ByRef plii 作为 lastInputInfo) 作为 Long
    'http://www.pgacon.com/visualbasic.htm#Take%20Advantage%20of%20Conditional%20Compilation
    'Visual Basic中的逻辑和按位运算符：
http://msdn.microsoft.com/en-us/library/wz3k228a(v=vs.80).aspx 和
http://stackoverflow.com/questions/1070863/hidden-features-of-vba
    Private Type SystemTime
          wYear          作为 Integer
          wMonth         作为 Integer
          wDayOfWeek     作为 Integer
          wDay           作为 Integer
          wHour          作为 Integer
          wMinute        作为 Integer
          wSecond        作为 Integer
          wMilliseconds  作为 Integer
    结束 类型
    Private Declare PtrSafe Sub apiGetLocalTime Lib "Kernel32" Alias "GetLocalTime" (lpSystem 作为
SystemTime)
    Private Type pointAPI       '用于 apiSetWindowPlacement
         X 作为 Long
Y 作为 Long
    结束 类型
    Private Type rectAPI        '用于 apiSetWindowPlacement
        Left_Renamed 作为 Long
Top_Renamed 作为 Long
Right_Renamed 作为Long
Bottom_Renamed 作为 Long
    结束 类型
    Private 类型 winPlacement   '由 apiSetWindowPlacement 使用
        length 作为 Long
flags 作为 Long
showCmd 作为 Long
ptMinPosition 作为 pointAPI
        ptMaxPosition 作为 pointAPI
        rcNormalPosition 作为 rectAPI
    结束 类型
    Private 声明 PtrSafe 函数 apiGetWindowPlacement 库 "User32" 别名 "GetWindowPlacement"
(ByVal hWnd 作为 Long, ByRef lpwndpl 作为 winPlacement) 作为 Long
    Private 类型 winRect     '由 apiMoveWindow 使用
Left 作为 Long
        Top 作为 Long
        Right 作为 Long
        Bottom 作为 Long
    结束 类型
    Private 声明 PtrSafe 函数 apiMoveWindow 库 "User32" 别名 "MoveWindow" (ByVal hWnd 作为
Long, xLeft 作为 Long, ByVal yTop 作为 Long, wWidth 作为 Long, ByVal hHeight 作为 Long, ByVal repaint 作为
Long) 作为 Long

                Private 声明 PtrSafe 函数 apiInternetOpen 库 "WiniNet" 别名 "InternetOpenA" (ByVal
```



```
the hook enumeration function; could use global variables instead, but this is nicer
        strTitle As String  'INPUT
        hWnd As Long        'OUTPUT
    End Type                            'Find a specific window with dynamic caption from a list of
all open windows:
http://www.everythingaccess.com/tutorials.asp?ID=Bring-an-external-application-window-to-the-foregrou
nd
    Private Declare PtrSafe Function apiEnumWindows Lib "User32" Alias "EnumWindows" (ByVal
lpEnumFunc As LongPtr, ByVal lParam As LongPtr) As Long
    Private Type lastInputInfo  'used by apiGetLastInputInfo, getLastInputTime
        cbSize As Long
        dwTime As Long
    End Type
    Private Declare PtrSafe Function apiGetLastInputInfo Lib "User32" Alias "GetLastInputInfo"
(ByRef plii As lastInputInfo) As Long
    'http://www.pgacon.com/visualbasic.htm#Take%20Advantage%20of%20Conditional%20Compilation
    'Logical and Bitwise Operators in Visual Basic:
http://msdn.microsoft.com/en-us/library/wz3k228a(v=vs.80).aspx and
http://stackoverflow.com/questions/1070863/hidden-features-of-vba
    Private Type SystemTime
          wYear          As Integer
          wMonth         As Integer
          wDayOfWeek     As Integer
          wDay           As Integer
          wHour          As Integer
          wMinute        As Integer
          wSecond        As Integer
          wMilliseconds  As Integer
    End Type
    Private Declare PtrSafe Sub apiGetLocalTime Lib "Kernel32" Alias "GetLocalTime" (lpSystem As
SystemTime)
    Private Type pointAPI       'used by apiSetWindowPlacement
         X As Long
         Y As Long
    End Type
    Private Type rectAPI        'used by apiSetWindowPlacement
        Left_Renamed As Long
        Top_Renamed As Long
        Right_Renamed As Long
        Bottom_Renamed As Long
    End Type
    Private Type winPlacement   'used by apiSetWindowPlacement
        length As Long
        flags As Long
        showCmd As Long
        ptMinPosition As pointAPI
        ptMaxPosition As pointAPI
        rcNormalPosition As rectAPI
    End Type
    Private Declare PtrSafe Function apiGetWindowPlacement Lib "User32" Alias "GetWindowPlacement"
(ByVal hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Type winRect     'used by apiMoveWindow
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
    End Type
    Private Declare PtrSafe Function apiMoveWindow Lib "User32" Alias "MoveWindow" (ByVal hWnd As
Long, xLeft As Long, ByVal yTop As Long, wWidth As Long, ByVal hHeight As Long, ByVal repaint As
Long) As Long

    Private Declare PtrSafe Function apiInternetOpen Lib "WiniNet" Alias "InternetOpenA" (ByVal
```

```
sAgent 作为 字符串, ByVal lAccessType 作为 Long, ByVal sProxyName 作为 字符串, ByVal sProxyBypass 作为
字符串, ByVal lFlags 作为 Long) 作为 Long     '打开 Internet 对象     '例如: lngINet =
InternetOpen("MyFTP 控制", 1, vbNullString, vbNullString, 0)
    Private Declare PtrSafe Function apiInternetConnect Lib "WiniNet" Alias "InternetConnectA"
(ByVal hInternetSession As Long, ByVal sServerName As String, ByVal nServerPort As Integer, ByVal
sUsername As String, ByVal sPassword As String, ByVal lService As Long, ByVal lFlags As Long, ByVal
lContext As Long) As Long  '连接到网络  '例如: lngINetConn = InternetConnect(lngINet,
"ftp.microsoft.com", 0, "anonymous", "wally@wallyworld.com", 1, 0, 0)
    Private Declare PtrSafe Function apiFtpGetFile Lib "WiniNet" Alias "FtpGetFileA" (ByVal
hFtpSession As Long, ByVal lpszRemoteFile As String, ByVal lpszNewFile As String, ByVal
fFailIfExists As Boolean, ByVal dwFlagsAndAttributes As Long, ByVal dwFlags As Long, ByVal
dwContext As Long) As Boolean     '获取文件 '例如: blnRC = FtpGetFile(lngINetConn, "dirmap.txt",
"c:\dirmap.txt", 0, 0, 1, 0)
    Private Declare PtrSafe Function apiFtpPutFile Lib "WiniNet" Alias "FtpPutFileA" (ByVal
hFtpSession As Long, ByVal lpszLocalFile As String, ByVal lpszRemoteFile As String, ByVal dwFlags
As Long, ByVal dwContext As Long) As Boolean  '发送文件  '例如: blnRC = FtpPutFile(lngINetConn,
"c:\dirmap.txt", "dirmap.txt", 1, 0)
    Private Declare PtrSafe Function apiFtpDeleteFile Lib "WiniNet" Alias "FtpDeleteFileA" (ByVal
hFtpSession As Long, ByVal lpszFileName As String) As Boolean '删除文件 '例如: blnRC =
FtpDeleteFile(lngINetConn, "test.txt")
    Private Declare PtrSafe Function apiInternetCloseHandle Lib "WiniNet" (ByVal hInet As Long) As
Integer  '关闭 Internet 对象  '例如: InternetCloseHandle lngINetConn     '例如: InternetCloseHandle
lngINet
    Private Declare PtrSafe Function apiFtpFindFirstFile Lib "WiniNet" Alias "FtpFindFirstFileA"
(ByVal hFtpSession As Long, ByVal lpszSearchFile As String, lpFindFileData As WIN32_FIND_DATA,
ByVal dwFlags As Long, ByVal dwContent As Long) As Long
    Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
    End Type
    Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
nFileSizeLow As Long
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * 1 '最大 FTP 路径
        cAlternate As String * 14
    End Type     '例如: lngHINet = FtpFindFirstFile(lngINetConn, "*.*", pData, 0, 0)
    Private Declare PtrSafe Function apiInternetFindNextFile Lib "WiniNet" Alias
"InternetFindNextFileA" (ByVal hFind As Long, lpvFindData As WIN32_FIND_DATA) As Long  '例如: blnRC =
InternetFindNextFile(lngHINet, pData)
#ElseIf Win32 Then  'Win32 = True, Win16 = False
```

(续第二个示例)

第42.6节：Windows API - 专用模块（2/2）

```
#ElseIf Win32 Then  'Win32 = True, Win16 = False
    Private Declare Sub apiCopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (MyDest As Any, MySource
As Any, ByVal MySize As Long)
    Private Declare Sub apiExitProcess Lib "Kernel32" Alias "ExitProcess" (ByVal uExitCode As Long)
    'Private Declare Sub apiGetStartupInfo Lib "Kernel32" Alias "GetStartupInfoA" (lpStartupInfo As
STARTUPINFO)
    Private Declare Sub apiSetCursorPos Lib "User32" Alias "SetCursorPos" (ByVal X As Integer,
ByVal Y As Integer)  'Visual Basic中的逻辑和按位运算符：
http://msdn.microsoft.com/en-us/library/wz3k228a(v=vs.80).aspx 和
http://stackoverflow.com/questions/1070863/hidden-features-of-vba
```



```
sAgent As String, ByVal lAccessType As Long, ByVal sProxyName As String, ByVal sProxyBypass As
String, ByVal lFlags As Long) As Long    'Open the Internet object    'ex: lngINet =
InternetOpen("MyFTP Control", 1, vbNullString, vbNullString, 0)
    Private Declare PtrSafe Function apiInternetConnect Lib "WiniNet" Alias "InternetConnectA"
(ByVal hInternetSession As Long, ByVal sServerName As String, ByVal nServerPort As Integer, ByVal
sUsername As String, ByVal sPassword As String, ByVal lService As Long, ByVal lFlags As Long, ByVal
lContext As Long) As Long  'Connect to the network  'ex: lngINetConn = InternetConnect(lngINet,
"ftp.microsoft.com", 0, "anonymous", "wally@wallyworld.com", 1, 0, 0)
    Private Declare PtrSafe Function apiFtpGetFile Lib "WiniNet" Alias "FtpGetFileA" (ByVal
hFtpSession As Long, ByVal lpszRemoteFile As String, ByVal lpszNewFile As String, ByVal
fFailIfExists As Boolean, ByVal dwFlagsAndAttributes As Long, ByVal dwFlags As Long, ByVal
dwContext As Long) As Long    'Get a file 'ex: blnRC = FtpGetFile(lngINetConn, "dirmap.txt",
"c:\dirmap.txt", 0, 0, 1, 0)
    Private Declare PtrSafe Function apiFtpPutFile Lib "WiniNet" Alias "FtpPutFileA" (ByVal
hFtpSession As Long, ByVal lpszLocalFile As String, ByVal lpszRemoteFile As String, ByVal dwFlags
As Long, ByVal dwContext As Long) As Boolean  'Send a file  'ex: blnRC = FtpPutFile(lngINetConn,
"c:\dirmap.txt", "dirmap.txt", 1, 0)
    Private Declare PtrSafe Function apiFtpDeleteFile Lib "WiniNet" Alias "FtpDeleteFileA" (ByVal
hFtpSession As Long, ByVal lpszFileName As String) As Boolean 'Delete a file 'ex: blnRC =
FtpDeleteFile(lngINetConn, "test.txt")
    Private Declare PtrSafe Function apiInternetCloseHandle Lib "WiniNet" (ByVal hInet As Long) As
Integer  'Close the Internet object  'ex: InternetCloseHandle lngINetConn    'ex: InternetCloseHandle
lngINet
    Private Declare PtrSafe Function apiFtpFindFirstFile Lib "WiniNet" Alias "FtpFindFirstFileA"
(ByVal hFtpSession As Long, ByVal lpszSearchFile As String, lpFindFileData As WIN32_FIND_DATA,
ByVal dwFlags As Long, ByVal dwContent As Long) As Long
    Private Type FILETIME
        dwLowDateTime As Long
        dwHighDateTime As Long
    End Type
    Private Type WIN32_FIND_DATA
        dwFileAttributes As Long
        ftCreationTime As FILETIME
        ftLastAccessTime As FILETIME
        ftLastWriteTime As FILETIME
        nFileSizeHigh As Long
        nFileSizeLow As Long
        dwReserved0 As Long
        dwReserved1 As Long
        cFileName As String * 1 'MAX_FTP_PATH
        cAlternate As String * 14
    End Type    'ex: lngHINet = FtpFindFirstFile(lngINetConn, "*.*", pData, 0, 0)
    Private Declare PtrSafe Function apiInternetFindNextFile Lib "WiniNet" Alias
"InternetFindNextFileA" (ByVal hFind As Long, lpvFindData As WIN32_FIND_DATA) As Long  'ex: blnRC =
InternetFindNextFile(lngHINet, pData)
#ElseIf Win32 Then  'Win32 = True, Win16 = False
```

(continued in second example)

Section 42.6: Windows API - Dedicated Module (2 of 2)

```
#ElseIf Win32 Then  'Win32 = True, Win16 = False
    Private Declare Sub apiCopyMemory Lib "Kernel32" Alias "RtlMoveMemory" (MyDest As Any, MySource
As Any, ByVal MySize As Long)
    Private Declare Sub apiExitProcess Lib "Kernel32" Alias "ExitProcess" (ByVal uExitCode As Long)
    'Private Declare Sub apiGetStartupInfo Lib "Kernel32" Alias "GetStartupInfoA" (lpStartupInfo As
STARTUPINFO)
    Private Declare Sub apiSetCursorPos Lib "User32" Alias "SetCursorPos" (ByVal X As Integer,
ByVal Y As Integer)  'Logical and Bitwise Operators in Visual Basic:
http://msdn.microsoft.com/en-us/library/wz3k228a(v=vs.80).aspx and
http://stackoverflow.com/questions/1070863/hidden-features-of-vba
```

```
'http://www.pgacon.com/visualbasic.htm#Take%20Advantage%20of%20Conditional%20Compilation
    Private Declare Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
    Private Declare Function apiAttachThreadInput Lib "User32" Alias "AttachThreadInput" (ByVal
idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
    Private Declare Function apiBringWindowToTop Lib "User32" Alias "BringWindowToTop" (ByVal
lngHWnd As Long) As Long
    Private Declare Function apiCloseHandle Lib "Kernel32" (ByVal hObject As Long) As Long
    Private Declare Function apiCloseWindow Lib "User32" Alias "CloseWindow" (ByVal hWnd As Long)
As Long
    'Private Declare Function apiCreatePipe Lib "Kernel32" (phReadPipe As Long, phWritePipe As Long,
lpPipeAttributes As SECURITY_ATTRIBUTES, ByVal nSize As Long) As Long
    'Private Declare Function apiCreateProcess Lib "Kernel32" Alias "CreateProcessA" (ByVal
lpApplicationName As Long, ByVal lpCommandLine As String, lpProcessAttributes As Any,
lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long,
lpEnvironment As Any, ByVal lpCurrentDriectory As String, lpStartupInfo As STARTUPINFO,
lpProcessInformation As PROCESS_INFORMATION) As Long
    Private Declare Function apiDestroyWindow Lib "User32" Alias "DestroyWindow" (ByVal hWnd As
Long) As Boolean
    Private Declare Function apiEndDialog Lib "User32" Alias "EndDialog" (ByVal hWnd As Long, ByVal
result As Long) As Boolean
    Private Declare Function apiEnumChildWindows Lib "User32" Alias "EnumChildWindows" (ByVal
hWndParent As Long, ByVal pEnumProc As Long, ByVal lParam As Long) As Long
    Private Declare Function apiExitWindowsEx Lib "User32" Alias "ExitWindowsEx" (ByVal uFlags As
Long, ByVal dwReserved As Long) As Long
    Private Declare Function apiFindExecutable Lib "Shell32" Alias "FindExecutableA" (ByVal lpFile
As String, ByVallpDirectory As String, ByVal lpResult As String) As Long
    Private Declare Function apiFindWindow Lib "User32" Alias "FindWindowA" (ByVal lpClassName As
String, ByVal lpWindowName As String) As Long
    Private Declare Function apiFindWindowEx Lib "User32" Alias "FindWindowExA" (ByVal hWnd1 As
Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
    Private Declare Function apiGetActiveWindow Lib "User32" Alias "GetActiveWindow" () As Long
    Private Declare Function apiGetClassNameA Lib "User32" Alias "GetClassNameA" (ByVal hWnd As
Long, ByVal szClassName As String, ByVal lLength As Long) As Long
    Private Declare Function apiGetCommandLine Lib "Kernel32" Alias "GetCommandLineW" () As Long
    Private Declare Function apiGetCommandLineParams Lib "Kernel32" Alias "GetCommandLineA" () As
Long
    Private Declare Function apiGetDiskFreeSpaceEx Lib "Kernel32" Alias "GetDiskFreeSpaceExA"
(ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As Currency, lpTotalNumberOfBytes As
Currency, lpTotalNumberOfFreeBytes As Currency) As Long
    Private Declare Function apiGetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal nDrive As
String) As Long
    Private Declare Function apiGetExitCodeProcess Lib "Kernel32" (ByVal hProcess As Long,
lpExitCode As Long) As Long
    Private Declare Function apiGetFileSize Lib "Kernel32" (ByVal hFile As Long, lpFileSizeHigh As
Long) As Long
    Private Declare Function apiGetForegroundWindow Lib "User32" Alias "GetForegroundWindow" () As
Long
    Private Declare Function apiGetFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
    Private Declare Function apiGetLastError Lib "Kernel32" Alias "GetLastError" () As Integer
    Private Declare Function apiGetParent Lib "User32" Alias "GetParent" (ByVal hWnd As Long) As
Long
    Private Declare Function apiGetSystemMetrics Lib "User32" Alias "GetSystemMetrics" (ByVal
nIndex As Long) As Long
    Private Declare Function apiGetTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long
    Private Declare Function apiGetTickCountMs Lib "Kernel32" Alias "GetTickCount" () As Long
    Private Declare Function apiGetUserName Lib "AdvApi32" Alias "GetUserNameA" (ByVal lpBuffer As
String, nSize As Long) As Long
    Private Declare Function apiGetWindow Lib "User32" Alias "GetWindow" (ByVal hWnd As Long, ByVal
wCmd As Long) As Long
        Private Declare Function apiGetWindowRect Lib "User32" Alias "GetWindowRect" (ByVal hWnd As
```



```
'http://www.pgacon.com/visualbasic.htm#Take%20Advantage%20of%20Conditional%20Compilation
    Private Declare Sub apiSleep Lib "Kernel32" Alias "Sleep" (ByVal dwMilliseconds As Long)
    Private Declare Function apiAttachThreadInput Lib "User32" Alias "AttachThreadInput" (ByVal
idAttach As Long, ByVal idAttachTo As Long, ByVal fAttach As Long) As Long
    Private Declare Function apiBringWindowToTop Lib "User32" Alias "BringWindowToTop" (ByVal
lngHWnd As Long) As Long
    Private Declare Function apiCloseHandle Lib "Kernel32" (ByVal hObject As Long) As Long
    Private Declare Function apiCloseWindow Lib "User32" Alias "CloseWindow" (ByVal hWnd As Long)
As Long
    'Private Declare Function apiCreatePipe Lib "Kernel32" (phReadPipe As Long, phWritePipe As Long,
lpPipeAttributes As SECURITY_ATTRIBUTES, ByVal nSize As Long) As Long
    'Private Declare Function apiCreateProcess Lib "Kernel32" Alias "CreateProcessA" (ByVal
lpApplicationName As Long, ByVal lpCommandLine As String, lpProcessAttributes As Any,
lpThreadAttributes As Any, ByVal bInheritHandles As Long, ByVal dwCreationFlags As Long,
lpEnvironment As Any, ByVal lpCurrentDriectory As String, lpStartupInfo As STARTUPINFO,
lpProcessInformation As PROCESS_INFORMATION) As Long
    Private Declare Function apiDestroyWindow Lib "User32" Alias "DestroyWindow" (ByVal hWnd As
Long) As Boolean
    Private Declare Function apiEndDialog Lib "User32" Alias "EndDialog" (ByVal hWnd As Long, ByVal
result As Long) As Boolean
    Private Declare Function apiEnumChildWindows Lib "User32" Alias "EnumChildWindows" (ByVal
hWndParent As Long, ByVal pEnumProc As Long, ByVal lParam As Long) As Long
    Private Declare Function apiExitWindowsEx Lib "User32" Alias "ExitWindowsEx" (ByVal uFlags As
Long, ByVal dwReserved As Long) As Long
    Private Declare Function apiFindExecutable Lib "Shell32" Alias "FindExecutableA" (ByVal lpFile
As String, ByVallpDirectory As String, ByVal lpResult As String) As Long
    Private Declare Function apiFindWindow Lib "User32" Alias "FindWindowA" (ByVal lpClassName As
String, ByVal lpWindowName As String) As Long
    Private Declare Function apiFindWindowEx Lib "User32" Alias "FindWindowExA" (ByVal hWnd1 As
Long, ByVal hWnd2 As Long, ByVal lpsz1 As String, ByVal lpsz2 As String) As Long
    Private Declare Function apiGetActiveWindow Lib "User32" Alias "GetActiveWindow" () As Long
    Private Declare Function apiGetClassNameA Lib "User32" Alias "GetClassNameA" (ByVal hWnd As
Long, ByVal szClassName As String, ByVal lLength As Long) As Long
    Private Declare Function apiGetCommandLine Lib "Kernel32" Alias "GetCommandLineW" () As Long
    Private Declare Function apiGetCommandLineParams Lib "Kernel32" Alias "GetCommandLineA" () As
Long
    Private Declare Function apiGetDiskFreeSpaceEx Lib "Kernel32" Alias "GetDiskFreeSpaceExA"
(ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As Currency, lpTotalNumberOfBytes As
Currency, lpTotalNumberOfFreeBytes As Currency) As Long
    Private Declare Function apiGetDriveType Lib "Kernel32" Alias "GetDriveTypeA" (ByVal nDrive As
String) As Long
    Private Declare Function apiGetExitCodeProcess Lib "Kernel32" (ByVal hProcess As Long,
lpExitCode As Long) As Long
    Private Declare Function apiGetFileSize Lib "Kernel32" (ByVal hFile As Long, lpFileSizeHigh As
Long) As Long
    Private Declare Function apiGetForegroundWindow Lib "User32" Alias "GetForegroundWindow" () As
Long
    Private Declare Function apiGetFrequency Lib "Kernel32" Alias "QueryPerformanceFrequency"
(cyFrequency As Currency) As Long
    Private Declare Function apiGetLastError Lib "Kernel32" Alias "GetLastError" () As Integer
    Private Declare Function apiGetParent Lib "User32" Alias "GetParent" (ByVal hWnd As Long) As
Long
    Private Declare Function apiGetSystemMetrics Lib "User32" Alias "GetSystemMetrics" (ByVal
nIndex As Long) As Long
    Private Declare Function apiGetTickCount Lib "Kernel32" Alias "QueryPerformanceCounter"
(cyTickCount As Currency) As Long
    Private Declare Function apiGetTickCountMs Lib "Kernel32" Alias "GetTickCount" () As Long
    Private Declare Function apiGetUserName Lib "AdvApi32" Alias "GetUserNameA" (ByVal lpBuffer As
String, nSize As Long) As Long
    Private Declare Function apiGetWindow Lib "User32" Alias "GetWindow" (ByVal hWnd As Long, ByVal
wCmd As Long) As Long
    Private Declare Function apiGetWindowRect Lib "User32" Alias "GetWindowRect" (ByVal hWnd As
```

```
Long, lpRect As winRect) As Long
    Private Declare Function apiGetWindowText Lib "User32" Alias "GetWindowTextA" (ByVal hWnd As
Long, ByVal szWindowText As String, ByVal lLength As Long) As Long
    Private Declare Function apiGetWindowThreadProcessId Lib "User32" Alias
"GetWindowThreadProcessId" (ByVal hWnd As Long, lpdwProcessId As Long) As Long
    Private Declare Function apiIsCharAlphaNumericA Lib "User32" Alias "IsCharAlphaNumericA" (ByVal
byChar As Byte) As Long
    Private Declare Function apiIsIconic Lib "User32" Alias "IsIconic" (ByVal hWnd As Long) As Long
    Private Declare Function apiIsWindowVisible Lib "User32" Alias "IsWindowVisible" (ByVal hWnd As
Long) As Long
    Private Declare Function apiIsZoomed Lib "User32" Alias "IsZoomed" (ByVal hWnd As Long) As Long
    Private Declare Function apiLStrCpynA Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination As
String, ByVal pSource As Long, ByVal iMaxLength As Integer) As Long
    Private Declare Function apiMessageBox Lib "User32" Alias "MessageBoxA" (ByVal hWnd As Long,
ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As Long
    Private Declare Function apiOpenIcon Lib "User32" Alias "OpenIcon" (ByVal hWnd As Long) As Long
    Private Declare Function apiOpenProcess Lib "Kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Private Declare Function apiPathAddBackslashByPointer Lib "ShlwApi" Alias "PathAddBackslashW"
(ByVal lpszPath As Long) As Long
    Private Declare Function apiPathAddBackslashByString Lib "ShlwApi" Alias "PathAddBackslashW"
(ByVal lpszPath As String) As Long
'http://msdn.microsoft.com/en-us/library/aa155716%28office.10%29.aspx
    Private Declare Function apiPostMessage Lib "User32" Alias "PostMessageA" (ByVal hWnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Private Declare Function apiReadFile Lib "Kernel32" (ByVal hFile As Long, lpBuffer As Any,
ByVal nNumberOfBytesToRead As Long, lpNumberOfBytesRead As Long, lpOverlapped As Any) As Long
    Private Declare Function apiRegQueryValue Lib "AdvApi32" Alias "RegQueryValue" (ByVal hKey As
Long, ByVal sValueName As String, ByVal dwReserved As Long, ByRef lValueType As Long, ByVal sValue
As String, ByRef lResultLen As Long) As Long
    Private Declare Function apiSendMessage Lib "User32" Alias "SendMessageA" (ByVal hWnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    Private Declare Function apiSetActiveWindow Lib "User32" Alias "SetActiveWindow" (ByVal hWnd As
Long) As Long
    Private Declare Function apiSetCurrentDirectoryA Lib "Kernel32" Alias "SetCurrentDirectoryA"
(ByVal lpPathName As String) As Long
    Private Declare Function apiSetFocus Lib "User32" Alias "SetFocus" (ByVal hWnd As Long) As Long
    Private Declare Function apiSetForegroundWindow Lib "User32" Alias "SetForegroundWindow" (ByVal
hWnd As Long) As Long
    Private Declare Function apiSetLocalTime Lib "Kernel32" Alias "SetLocalTime" (lpSystem As
SystemTime) As Long
    Private Declare Function apiSetWindowPlacement Lib "User32" Alias "SetWindowPlacement" (ByVal
hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Declare Function apiSetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hWnd As Long,
ByVal hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As Long, ByVal cy As
Long, ByVal wFlags As Long) As Long
    Private Declare Function apiSetWindowText Lib "User32" Alias "SetWindowTextA" (ByVal hWnd As
Long, ByVal lpString As String) As Long
    Private Declare Function apiShellExecute Lib "Shell32" Alias "ShellExecuteA" (ByVal hWnd As
Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal
lpDirectory As String, ByVal nShowCmd As Long) As Long
    Private Declare Function apiShowWindow Lib "User32" Alias "ShowWindow" (ByVal hWnd As Long,
ByVal nCmdShow As Long) As Long
    Private Declare Function apiShowWindowAsync Lib "User32" Alias "ShowWindowAsync" (ByVal hWnd As
Long, ByVal nCmdShow As Long) As Long
    Private Declare Function apiStrCpy Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination As
String, ByVal pSource As String, ByVal iMaxLength As Integer) As Long
    Private Declare Function apiStringLen Lib "Kernel32" Alias "lstrlenW" (ByVal lpString As Long)
As Long
    Private Declare Function apiStrTrimW Lib "ShlwApi" Alias "StrTrimW" () As Boolean
    Private Declare Function apiTerminateProcess Lib "Kernel32" Alias "TerminateProcess" (ByVal
hWnd As Long, ByVal uExitCode As Long) As Long
```



```
Long, lpRect As winRect) As Long
    Private Declare Function apiGetWindowText Lib "User32" Alias "GetWindowTextA" (ByVal hWnd As
Long, ByVal szWindowText As String, ByVal lLength As Long) As Long
    Private Declare Function apiGetWindowThreadProcessId Lib "User32" Alias
"GetWindowThreadProcessId" (ByVal hWnd As Long, lpdwProcessId As Long) As Long
    Private Declare Function apiIsCharAlphaNumericA Lib "User32" Alias "IsCharAlphaNumericA" (ByVal
byChar As Byte) As Long
    Private Declare Function apiIsIconic Lib "User32" Alias "IsIconic" (ByVal hWnd As Long) As Long
    Private Declare Function apiIsWindowVisible Lib "User32" Alias "IsWindowVisible" (ByVal hWnd As
Long) As Long
    Private Declare Function apiIsZoomed Lib "User32" Alias "IsZoomed" (ByVal hWnd As Long) As Long
    Private Declare Function apiLStrCpynA Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination As
String, ByVal pSource As Long, ByVal iMaxLength As Integer) As Long
    Private Declare Function apiMessageBox Lib "User32" Alias "MessageBoxA" (ByVal hWnd As Long,
ByVal lpText As String, ByVal lpCaption As String, ByVal wType As Long) As Long
    Private Declare Function apiOpenIcon Lib "User32" Alias "OpenIcon" (ByVal hWnd As Long) As Long
    Private Declare Function apiOpenProcess Lib "Kernel32" Alias "OpenProcess" (ByVal
dwDesiredAccess As Long, ByVal bInheritHandle As Long, ByVal dwProcessId As Long) As Long
    Private Declare Function apiPathAddBackslashByPointer Lib "ShlwApi" Alias "PathAddBackslashW"
(ByVal lpszPath As Long) As Long
    Private Declare Function apiPathAddBackslashByString Lib "ShlwApi" Alias "PathAddBackslashW"
(ByVal lpszPath As String) As Long
'http://msdn.microsoft.com/en-us/library/aa155716%28office.10%29.aspx
    Private Declare Function apiPostMessage Lib "User32" Alias "PostMessageA" (ByVal hWnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Private Declare Function apiReadFile Lib "Kernel32" (ByVal hFile As Long, lpBuffer As Any,
ByVal nNumberOfBytesToRead As Long, lpNumberOfBytesRead As Long, lpOverlapped As Any) As Long
    Private Declare Function apiRegQueryValue Lib "AdvApi32" Alias "RegQueryValue" (ByVal hKey As
Long, ByVal sValueName As String, ByVal dwReserved As Long, ByRef lValueType As Long, ByVal sValue
As String, ByRef lResultLen As Long) As Long
    Private Declare Function apiSendMessage Lib "User32" Alias "SendMessageA" (ByVal hWnd As Long,
ByVal wMsg As Long, ByVal wParam As Long, lParam As Any) As Long
    Private Declare Function apiSetActiveWindow Lib "User32" Alias "SetActiveWindow" (ByVal hWnd As
Long) As Long
    Private Declare Function apiSetCurrentDirectoryA Lib "Kernel32" Alias "SetCurrentDirectoryA"
(ByVal lpPathName As String) As Long
    Private Declare Function apiSetFocus Lib "User32" Alias "SetFocus" (ByVal hWnd As Long) As Long
    Private Declare Function apiSetForegroundWindow Lib "User32" Alias "SetForegroundWindow" (ByVal
hWnd As Long) As Long
    Private Declare Function apiSetLocalTime Lib "Kernel32" Alias "SetLocalTime" (lpSystem As
SystemTime) As Long
    Private Declare Function apiSetWindowPlacement Lib "User32" Alias "SetWindowPlacement" (ByVal
hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Declare Function apiSetWindowPos Lib "User32" Alias "SetWindowPos" (ByVal hWnd As Long,
ByVal hWndInsertAfter As Long, ByVal X As Long, ByVal Y As Long, ByVal cx As Long, ByVal cy As
Long, ByVal wFlags As Long) As Long
    Private Declare Function apiSetWindowText Lib "User32" Alias "SetWindowTextA" (ByVal hWnd As
Long, ByVal lpString As String) As Long
    Private Declare Function apiShellExecute Lib "Shell32" Alias "ShellExecuteA" (ByVal hWnd As
Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal
lpDirectory As String, ByVal nShowCmd As Long) As Long
    Private Declare Function apiShowWindow Lib "User32" Alias "ShowWindow" (ByVal hWnd As Long,
ByVal nCmdShow As Long) As Long
    Private Declare Function apiShowWindowAsync Lib "User32" Alias "ShowWindowAsync" (ByVal hWnd As
Long, ByVal nCmdShow As Long) As Long
    Private Declare Function apiStrCpy Lib "Kernel32" Alias "lstrcpynA" (ByVal pDestination As
String, ByVal pSource As String, ByVal iMaxLength As Integer) As Long
    Private Declare Function apiStringLen Lib "Kernel32" Alias "lstrlenW" (ByVal lpString As Long)
As Long
    Private Declare Function apiStrTrimW Lib "ShlwApi" Alias "StrTrimW" () As Boolean
    Private Declare Function apiTerminateProcess Lib "Kernel32" Alias "TerminateProcess" (ByVal
hWnd As Long, ByVal uExitCode As Long) As Long
```

```
    Private Declare Function apiTimeGetTime Lib "Winmm" Alias "timeGetTime" () As Long
    Private Declare Function apiVarPtrArray Lib "MsVbVm50" Alias "VarPtr" (Var() As Any) As Long
    Private Declare Function apiWaitForSingleObject Lib "Kernel32" (ByVal hHandle As Long, ByVal
dwMilliseconds As Long) As Long
    Private Type browseInfo     '用于 apiBrowseForFolder
       hOwner As Long
pidlRoot 作为 Long
pszDisplayName 作为 String
lpszTitle 作为 String
ulFlags 作为 Long
lpfn 作为 Long
lParam 作为 Long
iImage 作为 Long
    结束 类型
    Private Declare Function apiBrowseForFolder Lib "Shell32" Alias "SHBrowseForFolderA"
(lpBrowseInfo As browseInfo) As Long
    Private Type CHOOSECOLOR    '由 apiChooseColor 使用；http://support.microsoft.com/kb/153929 和
http://www.cpearson.com/Excel/Colors.aspx
lStructSize 作为 Long
hWndOwner 作为 Long
        hInstance 作为 Long
        rgbResult 作为 Long
        lpCustColors 作为 String
        flags 作为 Long
lCustData 作为 长整型
        lpfnHook 作为 长整型
        lpTemplateName 作为 字符串
    结束 类型
    Private Declare Function apiChooseColor Lib "ComDlg32" Alias "ChooseColorA" (pChoosecolor As
CHOOSECOLOR ) As Long
    Private Type FindWindowParameters   '用于在钩子枚举函数中传入和传出参数的自定义结构；也可以使用全局变量，但
这样更好       strTitle As String  '输入

hWnd 作为 长整型        '输出
    结束 类型                          '从所有打开的窗口列表中查找具有动态标题的特定窗口：

http://www.everythingaccess.com/tutorials.asp?ID=Bring-an-external-application-window-to-the-foregrou
nd
    Private Declare Function apiEnumWindows Lib "User32" Alias "EnumWindows" (ByVal lpEnumFunc As
Long, ByVal lParam As Long) As Long
    Private Type lastInputInfo  '由 apiGetLastInputInfo, getLastInputTime 使用
        cbSize 作为 Long
dwTime 作为 Long
    结束 类型
    Private Declare Function apiGetLastInputInfo Lib "User32" Alias "GetLastInputInfo" (ByRef plii
作为 lastInputInfo) 作为 Long
    Private Type SystemTime
wYear           作为 Integer
wMonth          作为 Integer
wDayOfWeek      作为 Integer
wDay            作为 Integer
wHour           作为 Integer
wMinute         作为 Integer
wSecond         作为 Integer
wMilliseconds   作为整数
    结束 类型
    Private Declare Sub apiGetLocalTime Lib "Kernel32" Alias "GetLocalTime" (lpSystem As
SystemTime)
    Private Type pointAPI
X As Long
Y As Long
    结束 类型
```



```
    Private Declare Function apiTimeGetTime Lib "Winmm" Alias "timeGetTime" () As Long
    Private Declare Function apiVarPtrArray Lib "MsVbVm50" Alias "VarPtr" (Var() As Any) As Long
    Private Declare Function apiWaitForSingleObject Lib "Kernel32" (ByVal hHandle As Long, ByVal
dwMilliseconds As Long) As Long
    Private Type browseInfo     'used by apiBrowseForFolder
        hOwner As Long
        pidlRoot As Long
        pszDisplayName As String
        lpszTitle As String
        ulFlags As Long
        lpfn As Long
        lParam As Long
        iImage As Long
    End Type
    Private Declare Function apiBrowseForFolder Lib "Shell32" Alias "SHBrowseForFolderA"
(lpBrowseInfo As browseInfo) As Long
    Private Type CHOOSECOLOR    'used by apiChooseColor; http://support.microsoft.com/kb/153929 and
http://www.cpearson.com/Excel/Colors.aspx
        lStructSize As Long
        hWndOwner As Long
        hInstance As Long
        rgbResult As Long
        lpCustColors As String
        flags As Long
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
    End Type
    Private Declare Function apiChooseColor Lib "ComDlg32" Alias "ChooseColorA" (pChoosecolor As
CHOOSECOLOR) As Long
    Private Type FindWindowParameters   'Custom structure for passing in the parameters in/out of
the hook enumeration function; could use global variables instead, but this is nicer
        strTitle As String  'INPUT
        hWnd As Long        'OUTPUT
    End Type                            'Find a specific window with dynamic caption from a list of
all open windows:
http://www.everythingaccess.com/tutorials.asp?ID=Bring-an-external-application-window-to-the-foregrou
nd
    Private Declare Function apiEnumWindows Lib "User32" Alias "EnumWindows" (ByVal lpEnumFunc As
Long, ByVal lParam As Long) As Long
    Private Type lastInputInfo  'used by apiGetLastInputInfo, getLastInputTime
        cbSize As Long
        dwTime As Long
    End Type
    Private Declare Function apiGetLastInputInfo Lib "User32" Alias "GetLastInputInfo" (ByRef plii
As lastInputInfo) As Long
    Private Type SystemTime
          wYear          As Integer
          wMonth         As Integer
          wDayOfWeek     As Integer
          wDay           As Integer
          wHour          As Integer
          wMinute        As Integer
          wSecond        As Integer
          wMilliseconds  As Integer
    End Type
    Private Declare Sub apiGetLocalTime Lib "Kernel32" Alias "GetLocalTime" (lpSystem As
SystemTime)
    Private Type pointAPI
        X As Long
        Y As Long
    End Type
```

```
    Private Type rectAPI
        Left_Renamed As Long
        Top_Renamed As Long
        Right_Renamed As Long
        Bottom_Renamed As Long
    End Type
    Private Type winPlacement
        length As Long
        flags As Long
        showCmd As Long
        ptMinPosition As pointAPI
        ptMaxPosition As pointAPI
        rcNormalPosition As rectAPI
    结束 类型
    Private Declare Function apiGetWindowPlacement Lib "User32" Alias "GetWindowPlacement" (ByVal
hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Type winRect
Left 作为 Long
        Top 作为 Long
        Right 作为 Long
        Bottom 作为 Long
    结束 类型
    Private Declare Function apiMoveWindow Lib "User32" Alias "MoveWindow" (ByVal hWnd As Long,
xLeft As Long, ByVal yTop As Long, wWidth As Long, ByVal hHeight As Long, ByVal repaint As Long) As
Long
#Else    ' Win16 = True
#End If
```



```
    Private Type rectAPI
        Left_Renamed As Long
        Top_Renamed As Long
        Right_Renamed As Long
        Bottom_Renamed As Long
    End Type
    Private Type winPlacement
        length As Long
        flags As Long
        showCmd As Long
        ptMinPosition As pointAPI
        ptMaxPosition As pointAPI
        rcNormalPosition As rectAPI
    End Type
    Private Declare Function apiGetWindowPlacement Lib "User32" Alias "GetWindowPlacement" (ByVal
hWnd As Long, ByRef lpwndpl As winPlacement) As Long
    Private Type winRect
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
    End Type
    Private Declare Function apiMoveWindow Lib "User32" Alias "MoveWindow" (ByVal hWnd As Long,
xLeft As Long, ByVal yTop As Long, wWidth As Long, ByVal hHeight As Long, ByVal repaint As Long) As
Long
#Else    ' Win16 = True
#End If
```

第43章：自动化或使用其他应用程序库

如果您在Visual Basic应用程序中使用其他应用程序的对象，您可能需要建立对这些应用程序对象库的引用。本手册提供了不同软件库的列表、来源及使用示例，如Windows Shell、Internet Explorer、XML HttpRequest等。

第43.1节：VBScript正则表达式

```
Set createVBScriptRegExObject = CreateObject("vbscript.RegExp")
```

工具 > 引用 > Microsoft VBScript 正则表达式 #.#
关联的DLL：VBScript.dll
来源：Internet Explorer 1.0 和 5.5

- MSDN - Microsoft加强了VBScript的正则表达式功能
- MSDN - 正则表达式语法（脚本）
- experts-exchange - 在Visual Basic for Applications和Visual Basic 6中使用正则表达式
- 如何在Microsoft Excel中使用正则表达式（Regex），包括单元格内和循环操作，见SO。
- regular-expressions.info/vbscript
- regular-expressions.info/vbscriptexample
- WIKI-正则表达式

代码

您可以使用此函数获取正则表达式结果，将所有匹配项（如果超过1个）连接成一个字符串，并在Excel单元格中显示结果。

```
Public Function getRegExResult(ByVal SourceString As String, Optional ByVal RegExPattern As String
= "\d+", _
    Optional ByVal isGlobalSearch As Boolean = True, Optional ByVal isCaseSensitive As Boolean =
False, Optional ByVal Delimiter As String = ";") As String

    Static RegExObject As Object
    If RegExObject Is Nothing Then
        Set RegExObject = createVBScriptRegExObject
    End If

getRegExResult = removeLeadingDelimiter(concatObjectItems(getRegExMatches(RegExObject,
SourceString, RegExPattern, isGlobalSearch, isCaseSensitive), Delimiter), Delimiter)

End Function

Private Function getRegExMatches(ByRef RegExObj As Object, _
    ByVal SourceString As String, ByVal RegExPattern As String, ByVal isGlobalSearch As Boolean,
ByVal isCaseSensitive As Boolean) As Object

    With RegExObj
        .Global = isGlobalSearch
        .IgnoreCase = Not (isCaseSensitive) '使用参数的正面含义（如 isCaseSensitive）比使用负面含义（如 IgnoreCas
e）更符合用户习惯
        .Pattern = RegExPattern
        Set getRegExMatches = .Execute(SourceString)
    结束于

End Function
```


Chapter 43: Automation or Using other applications Libraries

If you use the objects in other applications as part of your Visual Basic application, you may want to establish a reference to the object libraries of those applications. This Documentation provides a list, sources and examples of how to use libraries of different softwares, like Windows Shell, Internet Explorer, XML HttpRequest, and others.

Section 43.1: VBScript Regular Expressions

```
Set createVBScriptRegExObject = CreateObject("vbscript.RegExp")
```

Tools> References> Microsoft VBScript Regular Expressions #.#
Associated DLL: VBScript.dll
Source: Internet Explorer 1.0 and 5.5

- MSDN-Microsoft Beefs Up VBScript with Regular Expressions
- MSDN-Regular Expression Syntax (Scripting)
- experts-exchange - Using Regular Expressions in Visual Basic for Applications and Visual Basic 6
- How to use Regular Expressions (Regex) in Microsoft Excel both in-cell and loops on SO.
- regular-expressions.info/vbscript
- regular-expressions.info/vbscriptexample
- WIKI-Regular expression

Code

You can use this functions to get RegEx results, concatenate all matches (if more than 1) into 1 string, and display result in excel cell.

```
Public Function getRegExResult(ByVal SourceString As String, Optional ByVal RegExPattern As String
= "\d+", _
    Optional ByVal isGlobalSearch As Boolean = True, Optional ByVal isCaseSensitive As Boolean =
False, Optional ByVal Delimiter As String = ";") As String

    Static RegExObject As Object
    If RegExObject Is Nothing Then
        Set RegExObject = createVBScriptRegExObject
    End If

    getRegExResult = removeLeadingDelimiter(concatObjectItems(getRegExMatches(RegExObject,
SourceString, RegExPattern, isGlobalSearch, isCaseSensitive), Delimiter), Delimiter)

End Function

Private Function getRegExMatches(ByRef RegExObj As Object, _
    ByVal SourceString As String, ByVal RegExPattern As String, ByVal isGlobalSearch As Boolean,
ByVal isCaseSensitive As Boolean) As Object

    With RegExObj
        .Global = isGlobalSearch
        .IgnoreCase = Not (isCaseSensitive) 'it is more user friendly to use positive meaning of
argument, like isCaseSensitive, than to use negative IgnoreCase
        .Pattern = RegExPattern
        Set getRegExMatches = .Execute(SourceString)
    End With

End Function
```

```
Private Function concatObjectItems(ByRef Obj As Object, Optional ByVal DelimiterCustom As String =
";") As String
    Dim ObjElement As Variant
    For Each ObjElement In Obj
concatObjectItems = concatObjectItems & DelimiterCustom & ObjElement.Value
    Next
End Function

Public Function removeLeadingDelimiter(ByVal SourceString As String, ByVal Delimiter As String) As
String
    If Left$(SourceString, Len(Delimiter)) = Delimiter Then
        removeLeadingDelimiter = Mid$(SourceString, Len(Delimiter) + 1)
    End If
End Function

Private Function createVBScriptRegExObject() As Object
    Set createVBScriptRegExObject = CreateObject("vbscript.RegExp") 'ex.:
createVBScriptRegExObject.Pattern
End Function
```

第43.2节：脚本文件系统对象

```
Set createScriptingFileSystemObject = CreateObject("Scripting.FileSystemObject")
```

工具> 参考文献> Microsoft 脚本运行时
关联的 DLL：ScrRun.dll
来源：Windows 操作系统

MSDN-使用 FileSystemObject 访问文件

> 文件系统对象（FSO）模型提供了一个基于对象的工具，用于处理文件夹和文件。它允许你使用熟悉的对象.方法语法，配合丰富的属性、方法和事件来处理文件夹和文件。你也可以使用传统的 Visual Basic 语句和命令。

> FSO 模型赋予你的应用程序创建、更改、移动和删除文件夹的能力，或者确定特定文件夹是否存在及其位置。它还使你能够获取有关文件夹的信息，例如它们的名称以及创建或最后修改的日期。

MSDN-FileSystemObject 主题: "...解释 FileSystemObject 的概念及其使用方法。" exceltrick-VBA 中的 FileSystemObject – 详解
Scripting.FileSystemObject

第 43.3 节：Scripting Dictionary 对象

```
Set dict = CreateObject("Scripting.Dictionary")
```

工具> 参考文献> Microsoft 脚本运行时
关联的 DLL：ScrRun.dll
来源：Windows 操作系统

Scripting.Dictionary 对象
MSDN-字典对象


```
Private Function concatObjectItems(ByRef Obj As Object, Optional ByVal DelimiterCustom As String =
";") As String
    Dim ObjElement As Variant
    For Each ObjElement In Obj
        concatObjectItems = concatObjectItems & DelimiterCustom & ObjElement.Value
    Next
End Function

Public Function removeLeadingDelimiter(ByVal SourceString As String, ByVal Delimiter As String) As
String
    If Left$(SourceString, Len(Delimiter)) = Delimiter Then
        removeLeadingDelimiter = Mid$(SourceString, Len(Delimiter) + 1)
    End If
End Function

Private Function createVBScriptRegExObject() As Object
    Set createVBScriptRegExObject = CreateObject("vbscript.RegExp") 'ex.:
createVBScriptRegExObject.Pattern
End Function
```

Section 43.2: Scripting File System Object

```
Set createScriptingFileSystemObject = CreateObject("Scripting.FileSystemObject")
```

Tools> References> Microsoft Scripting Runtime
Associated DLL: ScrRun.dll
Source: Windows OS

MSDN-Accessing Files with FileSystemObject

> The File System Object (FSO) model provides an object-based tool for working with folders and files. It allows you to use the familiar object.method syntax with a rich set of properties, methods, and events to process folders and files. You can also employ the traditional Visual Basic statements and commands.

> The FSO model gives your application the ability to create, alter, move, and delete folders, or to determine if and where particular folders exist. It also enables you to get information about folders, such as their names and the date they were created or last modified.

MSDN-FileSystemObject topics: "*...explain the concept of the FileSystemObject and how to use it.*" exceltrick-FileSystemObject in VBA – Explained
Scripting.FileSystemObject

Section 43.3: Scripting Dictionary object

```
Set dict = CreateObject("Scripting.Dictionary")
```

Tools> References> Microsoft Scripting Runtime
Associated DLL: ScrRun.dll
Source: Windows OS

Scripting.Dictionary object
MSDN-Dictionary Object

第 43.4 节：Internet Explorer 对象

```
Set createInternetExplorerObject = CreateObject("InternetExplorer.Application")
```

工具 > 引用 > Microsoft Internet 控件
关联的 DLL：ieframe.dll
来源：Internet Explorer 浏览器

MSDN-InternetExplorer 对象

> 通过自动化控制 Windows Internet Explorer 的一个实例。

Internet Explorer 对象基础成员

下面的代码将介绍 IE 对象的工作原理以及如何通过 VBA 操作它。我建议
逐步执行，否则在多次导航时可能会出错。

```
子程序 IEGetToKnow()
    声明 IE 为 InternetExplorer '引用 Microsoft Internet 控件
    设置 IE = 新建 InternetExplorer

    使用 IE
        .Visible = True '设置或获取一个值，指示对象是可见还是
隐藏。

        导航
        .Navigate2 "http://www.example.com" '导航浏览器到一个可能无法用URL表示的位置，例如Windows Shell命名空间
中实体的PIDL。
Debug.Print .Busy '获取一个值，指示对象是否正在进行导航或下载操作。

Debug.Print .ReadyState '获取对象的就绪状态。
        .Navigate2 "http://www.example.com/2"
        .GoBack '在历史记录列表中向后导航一项
        .GoForward '在历史记录列表中向前导航一项。
        .GoHome '导航到当前的主页或起始页。
        .Stop '取消挂起的导航或下载，并停止动态页面元素，如背景声音和动画。

        .Refresh '重新加载对象中当前显示的文件。

Debug.Print .Silent '设置或获取一个值，指示对象是否可以显示对话框。

Debug.Print .Type '获取所包含文档对象的用户类型名称。

Debug.Print .Top '设置或获取对象顶部边缘的坐标。
Debug.Print .Left '设置或获取对象左边缘的坐标。
Debug.Print .Height '设置或获取对象的高度。
Debug.Print .Width '设置或获取对象的宽度。
    结束于

IE.Quit '关闭应用程序窗口
End Sub
```

网页抓取

使用IE最常见的操作是抓取网站上的一些信息，或者填写网站表单并提交信息。我们将学习如何操作。

Section 43.4: Internet Explorer Object

```
Set createInternetExplorerObject = CreateObject("InternetExplorer.Application")
```

Tools> References> Microsoft Internet Controls
Associated DLL: ieframe.dll
Source: Internet Explorer Browser

MSDN-InternetExplorer object

> Controls an instance of Windows Internet Explorer through automation.

Internet Explorer Objec Basic Members

The code below should introduce how the IE object works and how to manipulate it through VBA. I recommend stepping through it, otherwise it might error out during multiple navigations.

```
Sub IEGetToKnow()
    Dim IE As InternetExplorer 'Reference to Microsoft Internet Controls
    Set IE = New InternetExplorer

    With IE
        .Visible = True 'Sets or gets a value that indicates whether the object is visible or
hidden.

        'Navigation
        .Navigate2 "http://www.example.com" 'Navigates the browser to a location that might not be
expressed as a URL, such as a PIDL for an entity in the Windows Shell namespace.
        Debug.Print .Busy 'Gets a value that indicates whether the object is engaged in a navigation
or downloading operation.
        Debug.Print .ReadyState 'Gets the ready state of the object.
        .Navigate2 "http://www.example.com/2"
        .GoBack 'Navigates backward one item in the history list
        .GoForward 'Navigates forward one item in the history list.
        .GoHome 'Navigates to the current home or start page.
        .Stop 'Cancels a pending navigation or download, and stops dynamic page elements, such as
background sounds and animations.
        .Refresh 'Reloads the file that is currently displayed in the object.

        Debug.Print .Silent 'Sets or gets a value that indicates whether the object can display
dialog boxes.
        Debug.Print .Type 'Gets the user type name of the contained document object.

        Debug.Print .Top 'Sets or gets the coordinate of the top edge of the object.
        Debug.Print .Left 'Sets or gets the coordinate of the left edge of the object.
        Debug.Print .Height 'Sets or gets the height of the object.
        Debug.Print .Width 'Sets or gets the width of the object.
    End With

    IE.Quit 'close the application window
End Sub
```

Web Scraping

The most common thing to do with IE is to scrape some information of a website, or to fill a website form and submit information. We will look at how to do it.

让我们来看一下example.com的源代码：

```
<!doctype html>
<html>
    <head>
        <title>示例域名</title>
        <meta charset="utf-8" />
        <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <style ... </style>
    </head>

    <body>
        <div>
            <h1>示例域名</h1>
            <p>该域名用于文档中的示例说明。您
可以在示例中使用此
域名，无需事先协调或获得许可。</p>
            <p><a href="http://www.iana.org/domains/example">更多信息...</a></p>
        </div>
    </body>
</html>
```

我们可以使用如下代码来获取和设置信息：

```
Sub IEWebScrape1()
    声明 IE 为 InternetExplorer '引用 Microsoft Internet 控件
    设置 IE = 新建 InternetExplorer

    使用 IE
        .Visible = True
        .Navigate2 "http://www.example.com"

        '我们添加一个循环以确保网站已加载并准备好。
        '不总是有效，不能完全依赖。
        Do While .Busy = True Or .ReadyState <> READYSTATE_COMPLETE '等同于 = .ReadyState <> 4
            ' DoEvents - 值得考虑。使用前请了解其影响。
Application.Wait (Now + TimeValue("00:00:01")) '等待1秒，然后再次检查。
        循环

        '在即时窗口打印信息
        使用.Document"源代码HTML"位于显示页面的"下方"。
            停止 'VBE 停止。逐行继续查看发生了什么。
Debug.Print .GetElementsByTagName("title")(0).innerHtml '打印 "Example Domain"
            Debug.Print .GetElementsByTagName("h1")(0).innerHtml '打印 "Example Domain"
            Debug.Print .GetElementsByTagName("p")(0).innerHtml '打印 "This domain is
established..."
Debug.Print .GetElementsByTagName("p")(1).innerHtml '打印 "<a
href="http://www.iana.org/domains/example">More information...</a>"
Debug.Print .GetElementsByTagName("p")(1).innerText '打印 "More information..."
            Debug.Print .GetElementsByTagName("a")(0).innerText '打印 "More information..."

            '我们可以更改本地显示的网站。不要担心会破坏网站。
            .GetElementsByTagName("title")(0).innerHtml = "Psst, scraping..."
            .GetElementsByTagName("h1")(0).innerHtml = "让我试试一些可疑的东西。" '你刚刚
更改了网站的本地 HTML。
            .GetElementsByTagName("p")(0).innerHtml = "Lorem ipsum..... 结束"
            .GetElementsByTagName("a")(0).innerText = "iana.org"
        End With '.document

        .Quit '关闭应用程序窗口
```


Let us consider *example.com* source code:

```
<!doctype html>
<html>
    <head>
        <title>Example Domain</title>
        <meta charset="utf-8" />
        <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1" />
        <style ... </style>
    </head>

    <body>
        <div>
            <h1>Example Domain</h1>
            <p>This domain is established to be used for illustrative examples in documents. You
may use this
            domain in examples without prior coordination or asking for permission.</p>
            <p><a href="http://www.iana.org/domains/example">More information...</a></p>
        </div>
    </body>
</html>
```

We can use code like below to get and set information:

```
Sub IEWebScrape1()
    Dim IE As InternetExplorer 'Reference to Microsoft Internet Controls
    Set IE = New InternetExplorer

    With IE
        .Visible = True
        .Navigate2 "http://www.example.com"

        'we add a loop to be sure the website is loaded and ready.
        'Does not work consistently. Cannot be relied upon.
        Do While .Busy = True Or .ReadyState <> READYSTATE_COMPLETE 'Equivalent = .ReadyState <> 4
            ' DoEvents - worth considering. Know implications before you use it.
            Application.Wait (Now + TimeValue("00:00:01")) 'Wait 1 second, then check again.
        Loop

        'Print info in immediate window
        With .Document 'the source code HTML "below" the displayed page.
            Stop 'VBE Stop. Continue line by line to see what happens.
            Debug.Print .GetElementsByTagName("title")(0).innerHtml 'prints "Example Domain"
            Debug.Print .GetElementsByTagName("h1")(0).innerHtml 'prints "Example Domain"
            Debug.Print .GetElementsByTagName("p")(0).innerHtml 'prints "This domain is
established..."
            Debug.Print .GetElementsByTagName("p")(1).innerHtml 'prints "<a
href="http://www.iana.org/domains/example">More information...</a>"
            Debug.Print .GetElementsByTagName("p")(1).innerText 'prints "More information..."
            Debug.Print .GetElementsByTagName("a")(0).innerText 'prints "More information..."

            'We can change the localy displayed website. Don't worry about breaking the site.
            .GetElementsByTagName("title")(0).innerHtml = "Psst, scraping..."
            .GetElementsByTagName("h1")(0).innerHtml = "Let me try something fishy." 'You have just
changed the local HTML of the site.
            .GetElementsByTagName("p")(0).innerHtml = "Lorem ipsum..... The End"
            .GetElementsByTagName("a")(0).innerText = "iana.org"
        End With '.document

        .Quit 'close the application window
```

```
    以'ie 结尾

End Sub
```

发生了什么？这里的关键角色是**.Document**，也就是HTML源代码。我们可以应用一些查询
来获取我们想要的集合或对象。
例如`IE.Document.GetElementsByTagName("title")(0).innerHtml`。`GetElementsByTagName`返回一个
包含"title"标签的HTML元素集合。源代码中只有一个这样的标签。该集合
是从0开始计数的。因此要获取第一个元素，我们加上`(0)`。现在，在我们的例子中，我们只想要`innerHtml`（一个字符串），而不是
元素对象本身。所以我们指定我们想要的属性。

点击

要在网站上跟随链接，我们可以使用多种方法：

```
子程序 IEGoToPlaces()
    声明 IE 为 InternetExplorer '引用 Microsoft Internet 控件
    设置 IE = 新建 InternetExplorer

    使用 IE
        .Visible = True
        .Navigate2 "http://www.example.com"
        停止 'VBE 停止。逐行继续查看发生了什么。

        '点击
        .Document.GetElementsByTagName("a")(0).Click
        停止 'VBE 停止。

        返回
        .返回
        停止 'VBE 停止。

        使用<a>标签中的href属性或“link”进行导航
        .Navigate2 .Document.GetElementsByTagName("a")(0).href
        停止 'VBE 停止。

        .退出 '关闭应用程序窗口
    结束 With
End Sub
```

微软HTML对象库或IE的最佳搭档

为了充分利用加载到IE中的HTML，你可以（或应该）使用另一个库，即微软
HTML对象库。更多内容将在另一个示例中介绍。

IE的主要问题

IE的主要问题是确认页面已加载完成并准备好进行交互。使用Do While...
循环有帮助，但不可靠。

此外，仅使用IE来抓取HTML内容是大材小用。为什么？因为浏览器是用来浏览的，也就是说
显示带有所有CSS、JavaScript、图片、弹出窗口等的网页。如果你只需要原始数据，考虑采用不同的方法。例如使用XM
L HTTPRequest。关于这点将在另一个示例中详细介绍。


```
    End With 'ie

End Sub
```

What is going on? The key player here is the **.Document**, that is the HTML source code. We can apply some queries to get the Collections or Object we want.
For example the `IE.Document.GetElementsByTagName("title")(0).innerHtml`. `GetElementsByTagName` returns a **Collection** of HTML Elements, that have the "*title*" tag. There is only one such tag in the source code. The **Collection** is 0-based. So to get the first element we add `(0)`. Now, in our case, we want only the `innerHtml` (a String), not the Element Object itself. So we specify the property we want.

Click

To follow a link on a site, we can use multiple methods:

```
Sub IEGoToPlaces()
    Dim IE As InternetExplorer 'Reference to Microsoft Internet Controls
    Set IE = New InternetExplorer

    With IE
        .Visible = True
        .Navigate2 "http://www.example.com"
        Stop 'VBE Stop. Continue line by line to see what happens.

        'Click
        .Document.GetElementsByTagName("a")(0).Click
        Stop 'VBE Stop.

        'Return Back
        .GoBack
        Stop 'VBE Stop.

        'Navigate using the href attribute in the <a> tag, or "link"
        .Navigate2 .Document.GetElementsByTagName("a")(0).href
        Stop 'VBE Stop.

        .Quit 'close the application window
    End With
End Sub
```

Microsoft HTML Object Library or IE Best friend

To get the most out of the HTML that gets loaded into the IE, you can (or should) use another Library, i.e. *Microsoft HTML Object Library*. More about this in another example.

IE Main issues

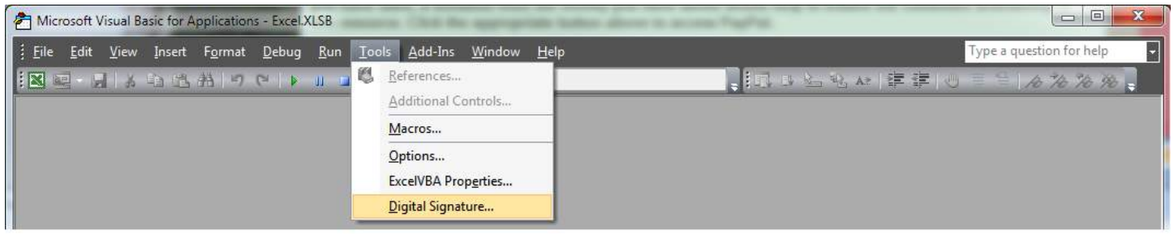
The main issue with IE is verifying that the page is done loading and is ready to be interacted with. The `Do While... Loop` helps, but is not reliable.

Also, using IE just to scrape HTML content is OVERKILL. Why? Because the Browser is meant for browsing, i.e. displaying the web page with all the CSS, JavaScripts, Pictures, Popups, etc. If you only need the raw data, consider different approach. E.g. using XML HTTPRequest. More about this in another example.

第44章：宏安全性及VBA项目/模块的签名

第44.1节：创建有效的自签名数字证书 SELFCERT.EXE

为了运行宏并维护Office应用程序针对恶意代码提供的安全性，有必要通过VBA编辑器 > 工具 > 数字签名对VBAProject.OTM进行数字签名。



Office附带一个实用程序，可以创建自签名数字证书，您可以在电脑上使用该证书为项目签名。

该实用程序**SELFCERT.EXE**位于Office程序文件夹中，

点击"VBA项目的数字证书"以打开证书*向导*。

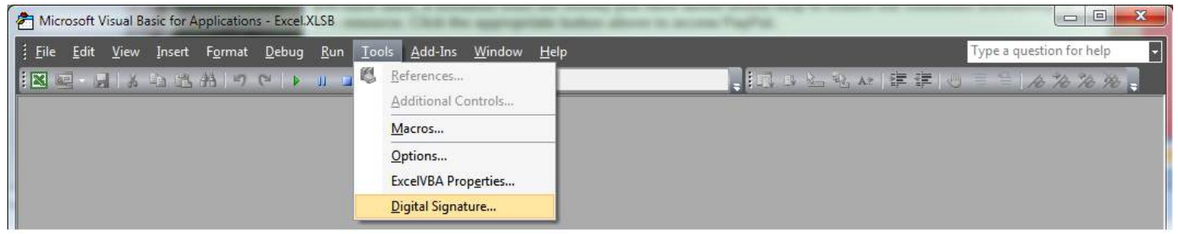
在对话框中输入合适的证书名称，然后点击确定。

如果一切顺利，您将看到确认信息：

Chapter 44: Macro security and signing of VBA-projects/-modules

Section 44.1: Create a valid digital self-signed certificate SELFCERT.EXE

To run macros and maintain the security Office applications provide against malicious code, it is necessary to digitally sign the VBAProject.OTM from the *VBA editor* > *Tools* > *Digital Signature*.



Office comes with a utility to create a self-signed digital certificate that you can employ on the PC to sign your projects.

This utility **SELFCERT.EXE** is in the Office program folder,

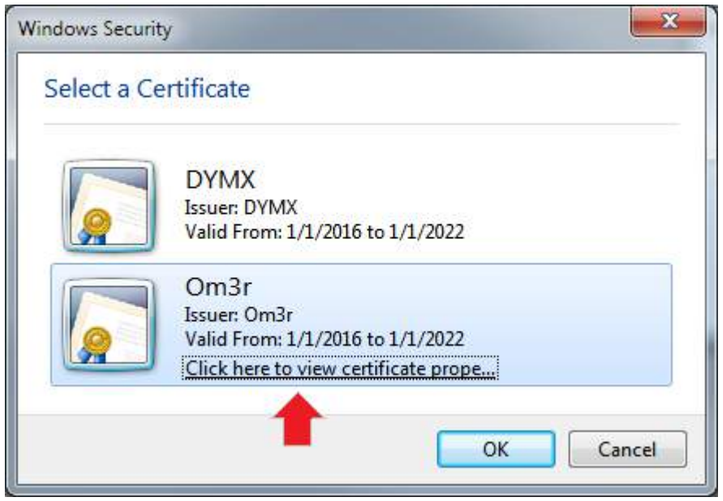
Click on Digital Certificate for VBA Projects to open the certificate *wizard*.

In the dialog enter a suitable name for the certificate and click OK.

If all goes well you will see a confirmation:

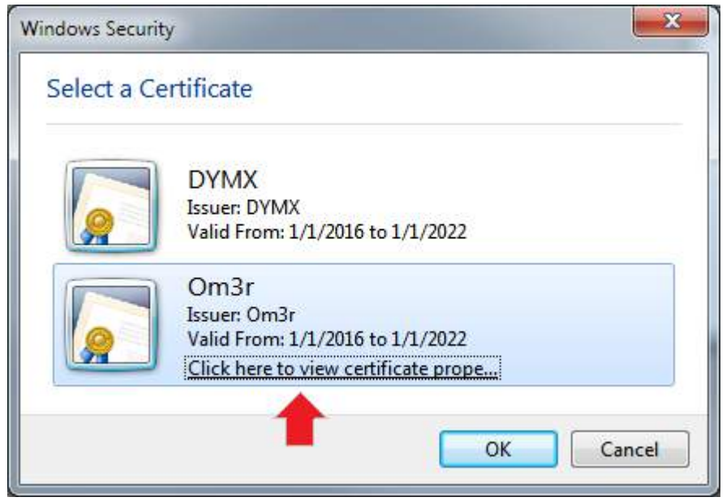
您现在可以关闭SELFCERT向导，转而关注您创建的证书。

如果你尝试使用刚创建的证书并查看其属性



You can now close the **SELFCERT** wizard and turn your attention to the certificate you have created.

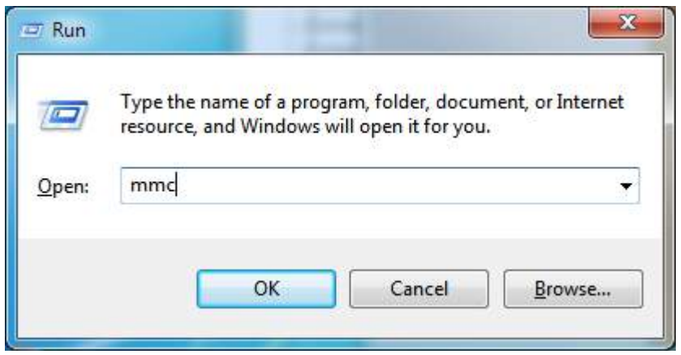
If you try to employ the certificate you have just created and you check its properties



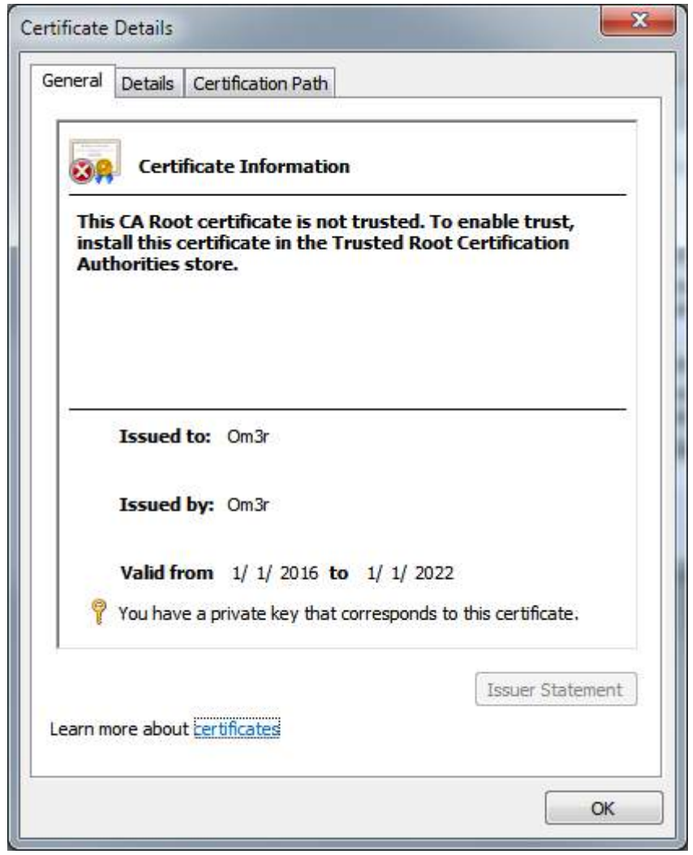
你会看到证书不被信任，原因会在对话框中显示。

证书已创建在“当前用户 > 个人 > 证书”存储区。它需要放在“本地计算机 > 受信任的根证书颁发机构 > 证书”存储区，因此你需要从前者导出并导入到后者。

按下 Windows **键+R**，这将打开“运行”窗口。然后在窗口中输入“mmc”，如下面所示，点击“确定”。



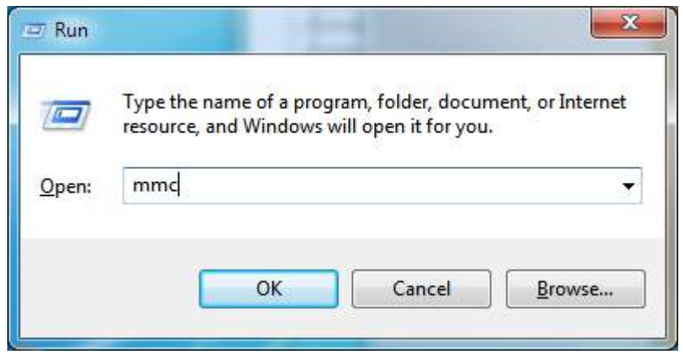
微软管理控制台将打开，界面如下所示。



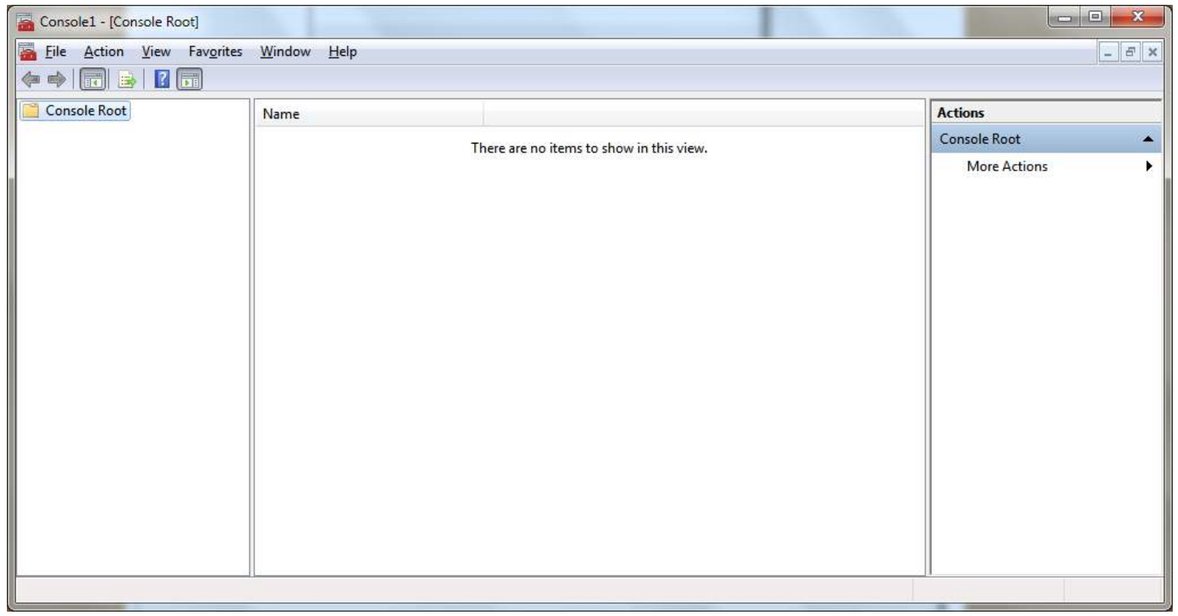
You will see that the certificate is not trusted and the reason is indicated in the dialog.

The certificate has been created in the Current User > Personal > Certificates store. It needs to go in Local Computer > Trusted Root Certificate Authorities > Certificates store, so you need to export from the former and import to the latter.

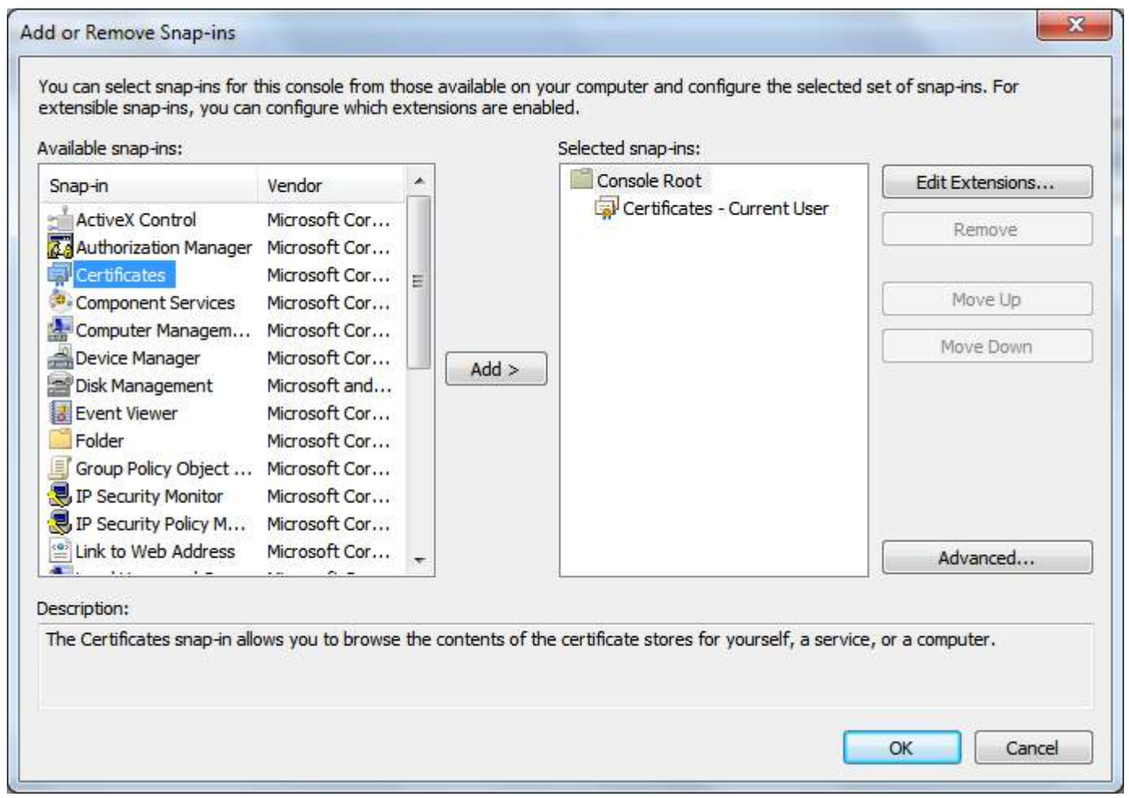
Pressing the Windows **Key+R** which will open the 'Run' Window. then Enter 'mmc' in the window as shown below and click 'OK '.



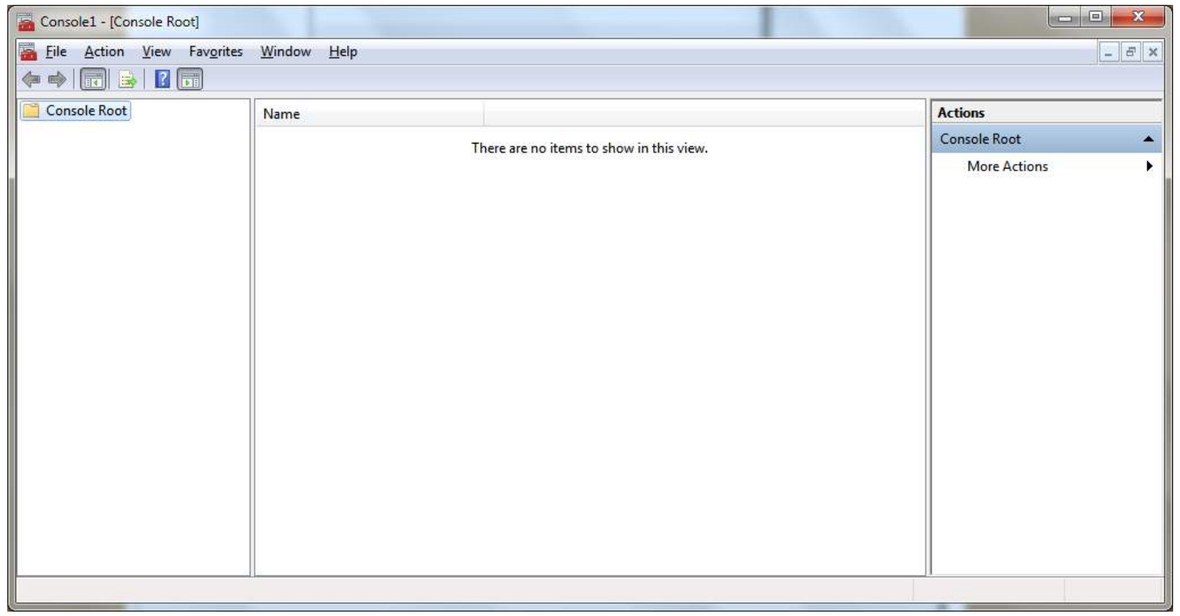
The Microsoft Management Console will open and look like the following.



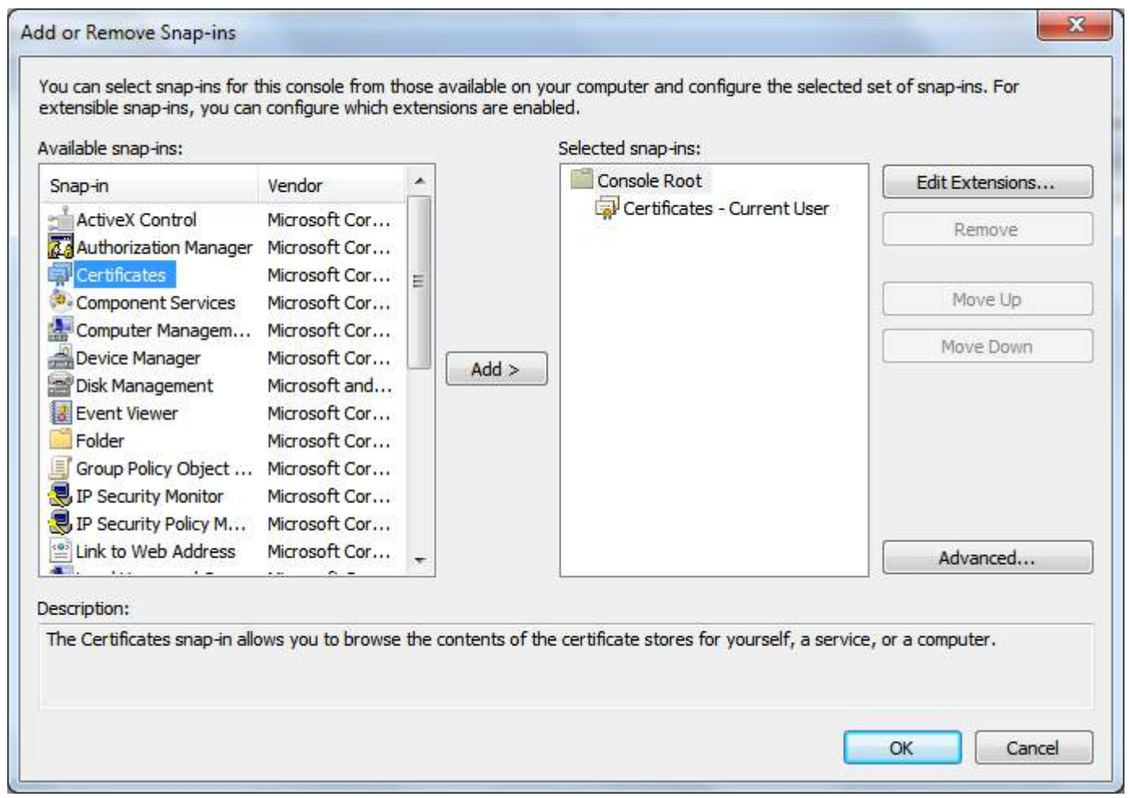
从“文件”菜单中选择“添加/删除管理单元...”，然后在弹出的对话框中双击“证书”，再点击“确定”



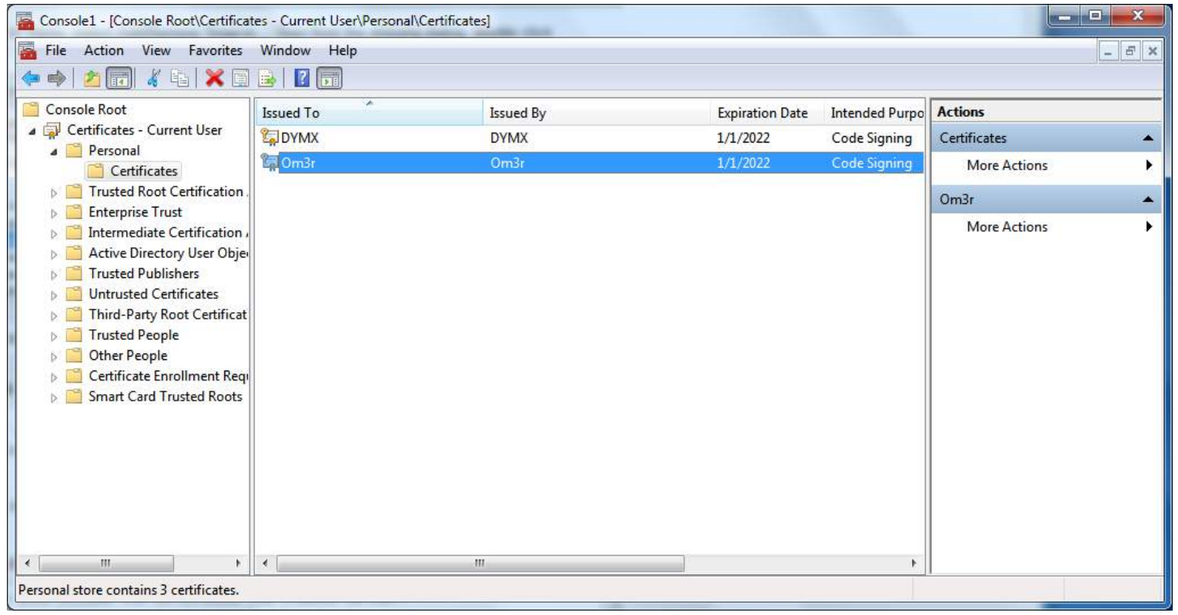
展开左侧窗口中的证书 - 当前用户下拉菜单，并选择如下所示的证书。中间面板将显示该位置的证书，其中包括您之前创建的证书：



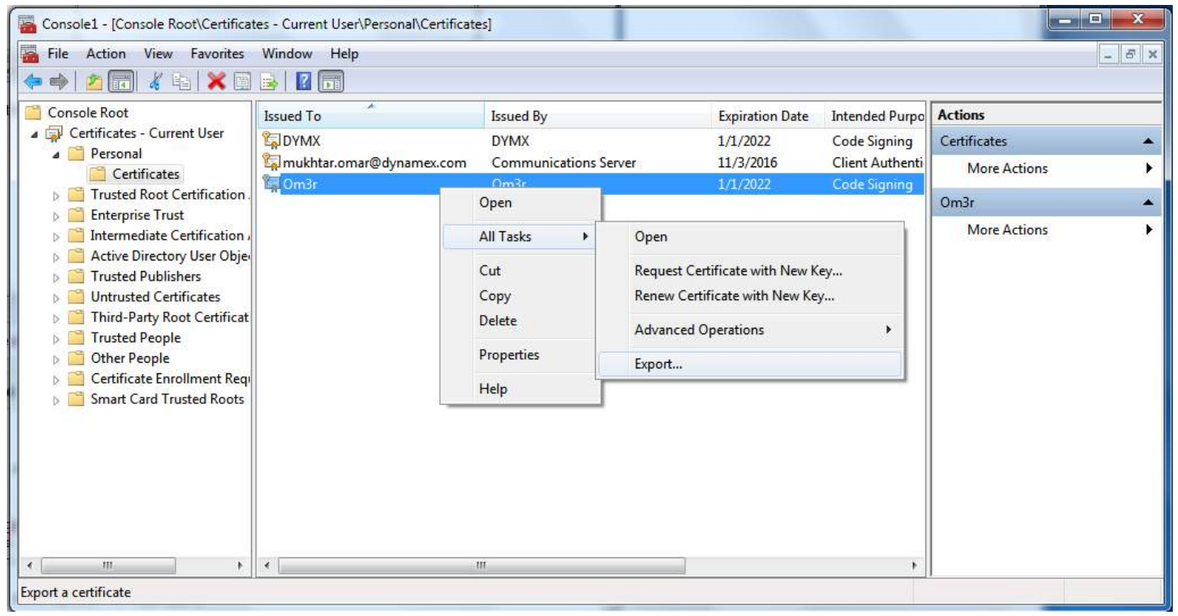
From the File menu, select Add/Remove Snap-in... Then from the ensuing dialog, double click Certificates and then click OK



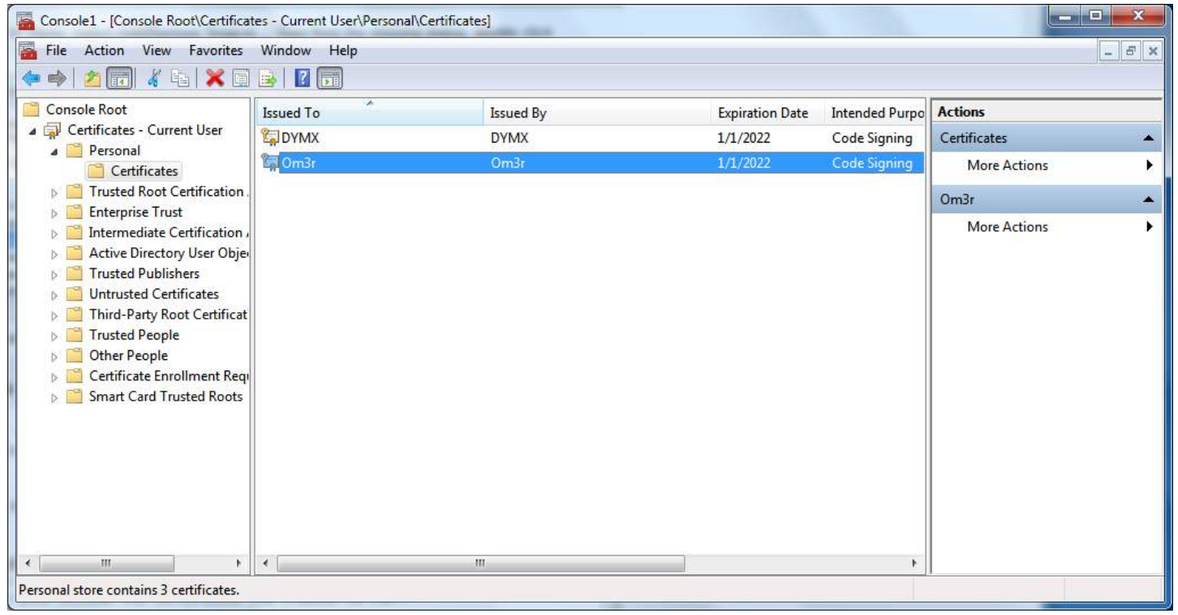
Expand the dropdown in the left window for *Certificates - Current User*' and select certificates as shown below. The center panel will then show the certificates in that location, which will include the certificate you created earlier:



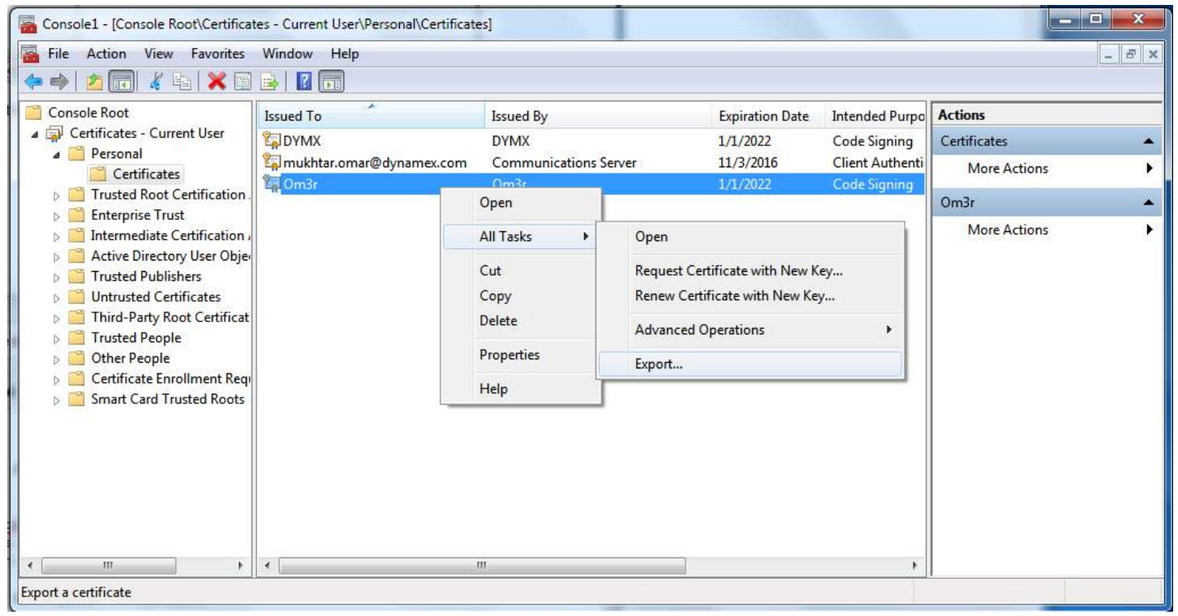
右键点击证书，选择所有任务 > 导出：



导出向导

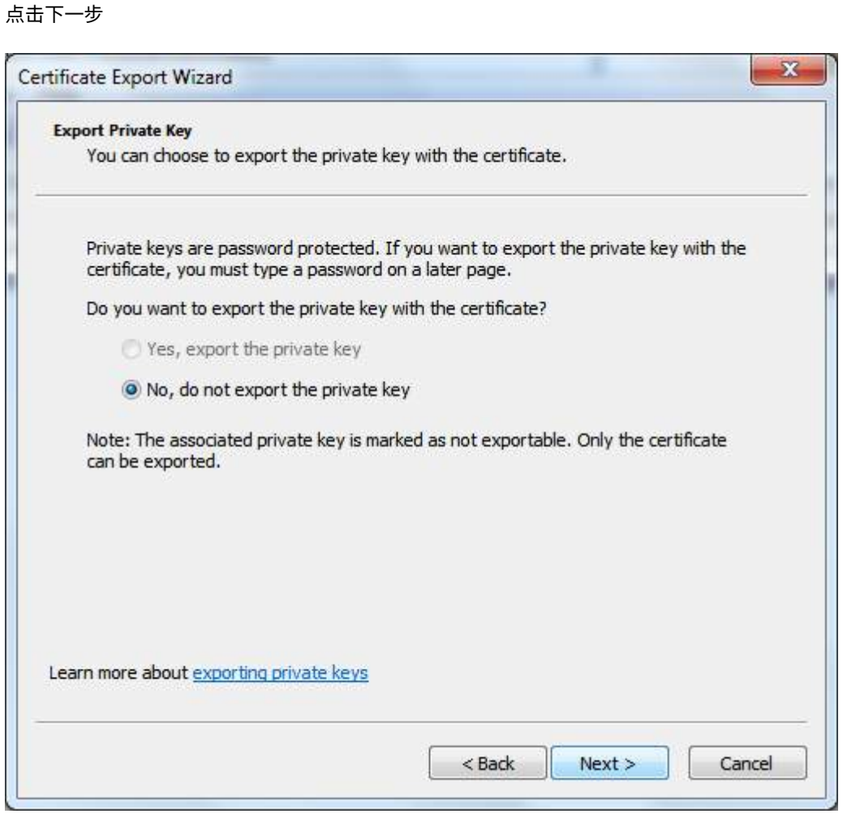


Right click the certificate and select All Tasks > Export:



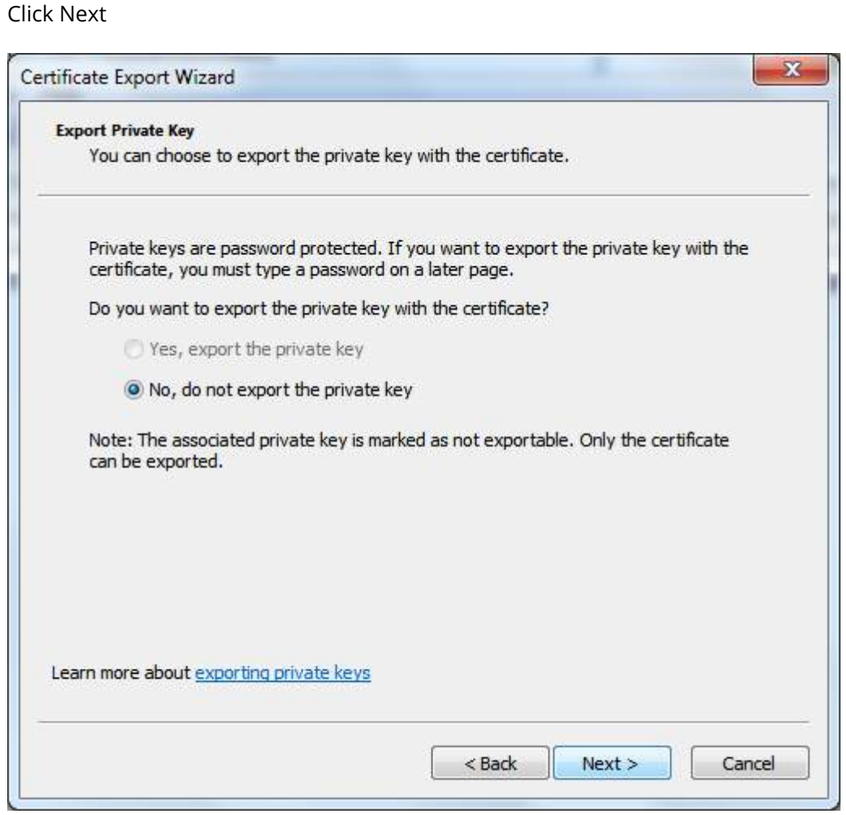
Export Wizard

点击下一步

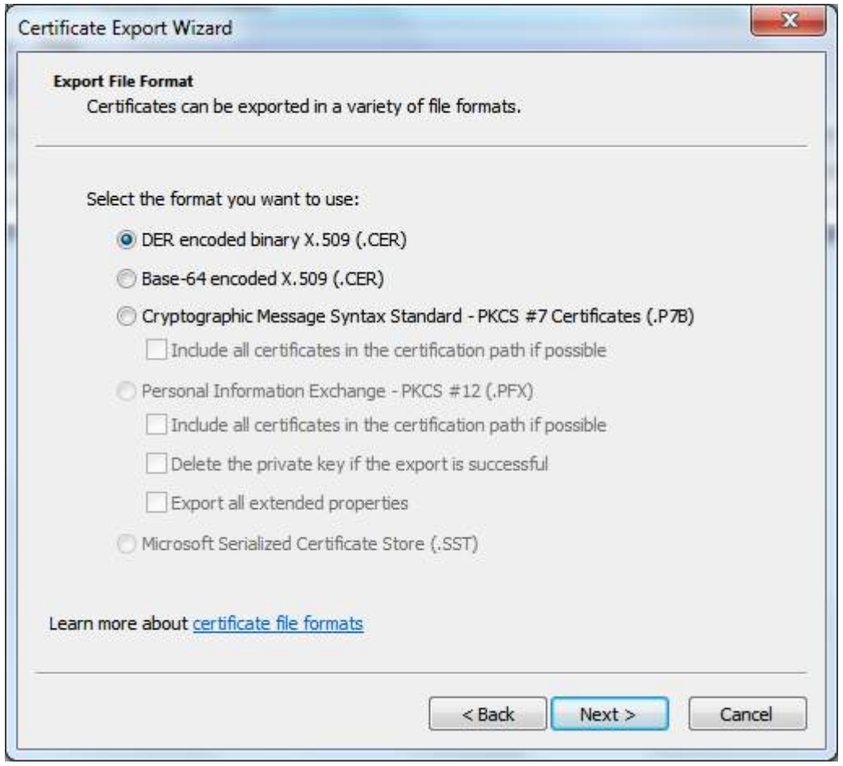


只有一个预选项可用，因此再次点击“下一步”：

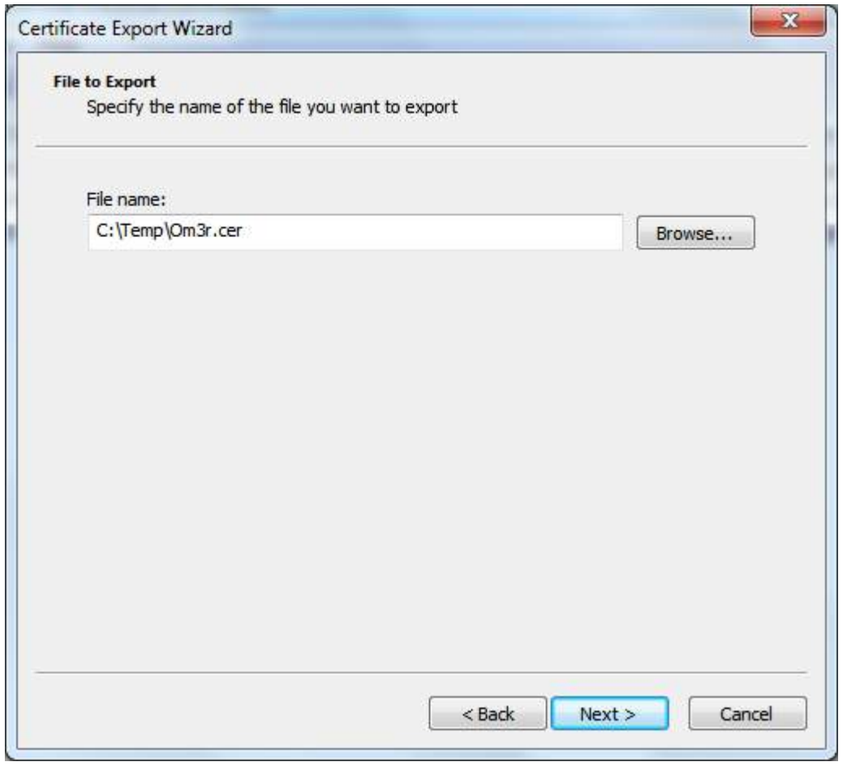
Click Next

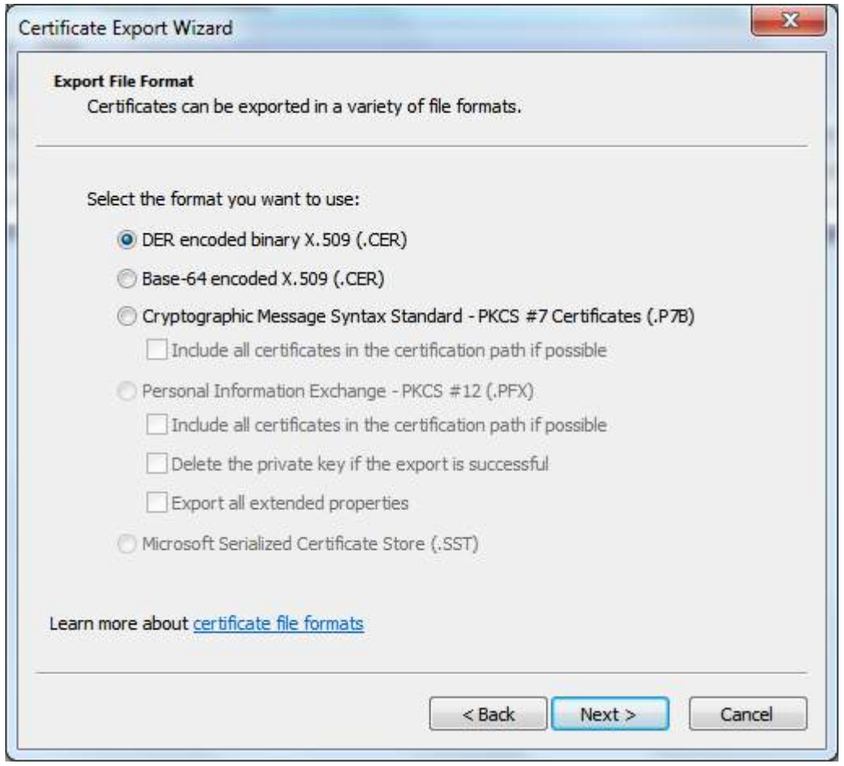


the Only one pre-selected option will be available, so click 'Next' again:

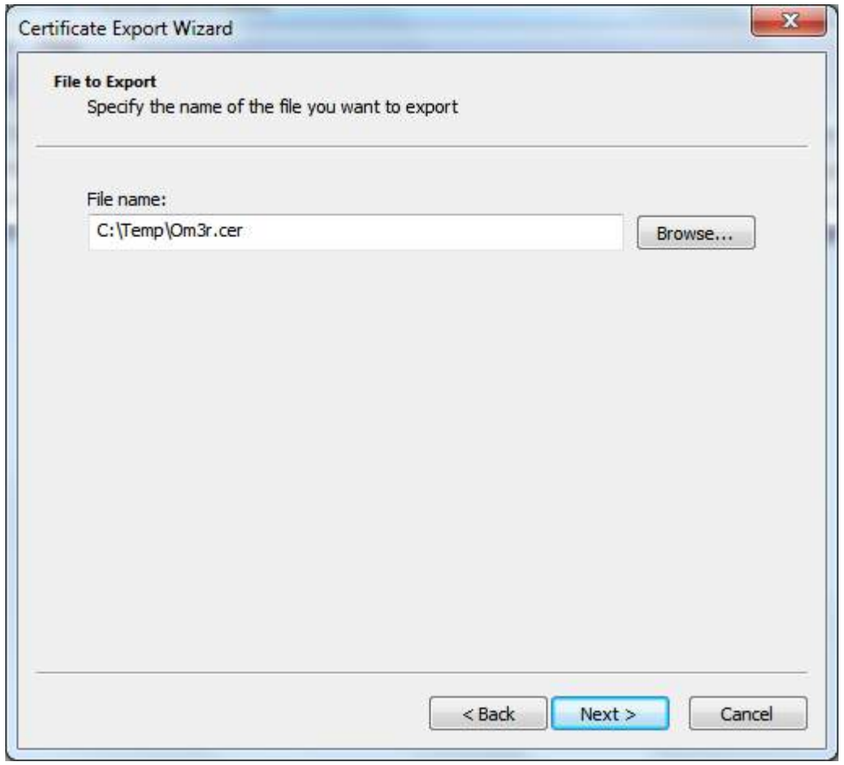


顶部项目将已被预选。再次点击下一步，选择导出证书的名称和保存位置。





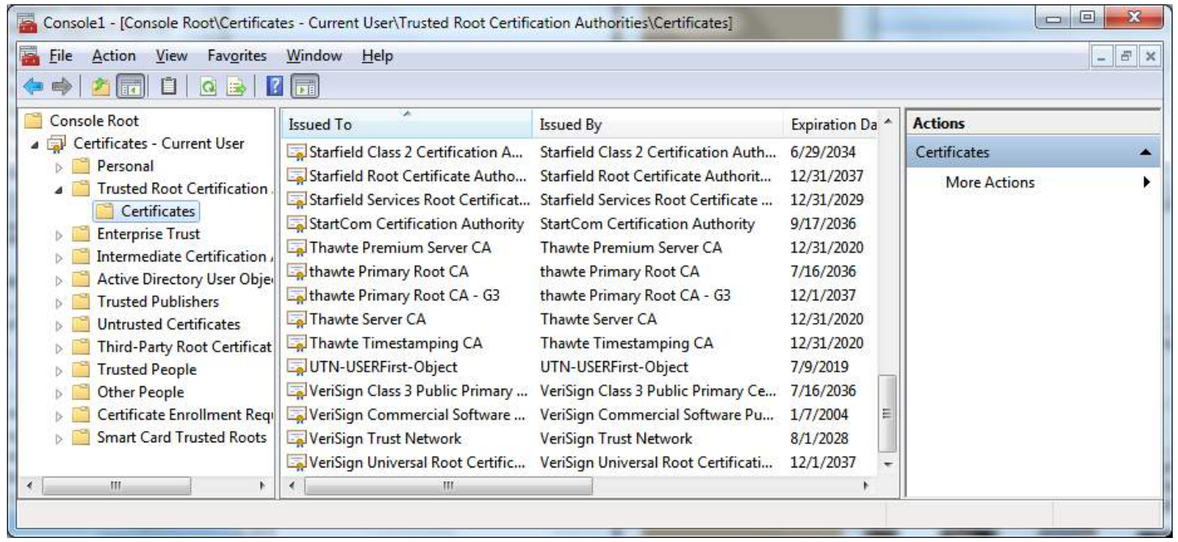
The top item will already be pre-selected. Click Next again and choose a name and location to save the exported certificate.



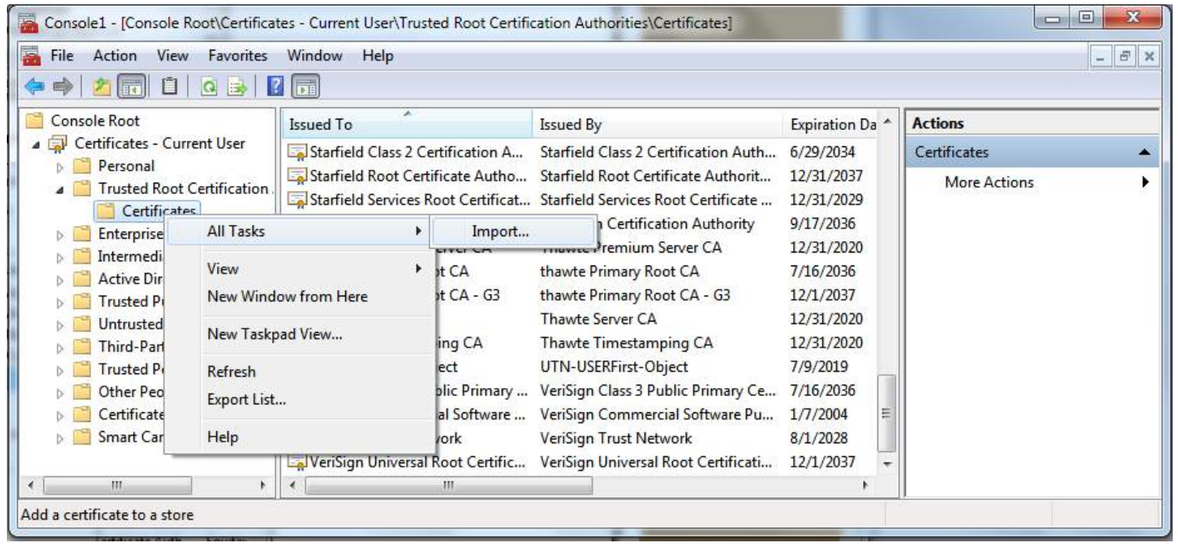
再次点击“下一步”以保存证书

一旦焦点返回到管理控制台。

展开证书菜单，然后从受信任的根证书颁发机构菜单中选择证书。



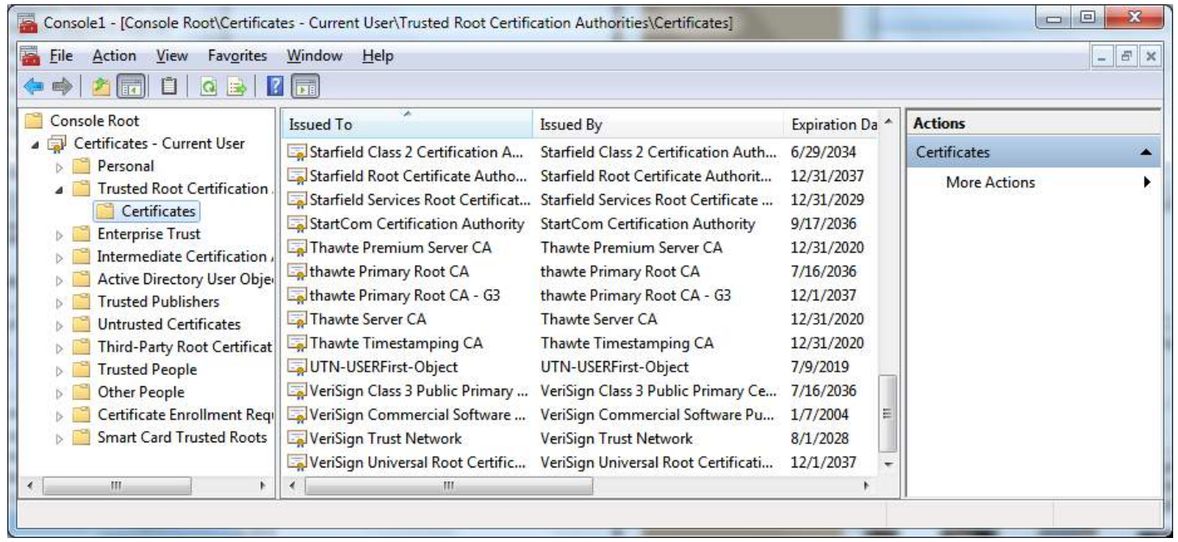
右键点击。选择所有任务和导入



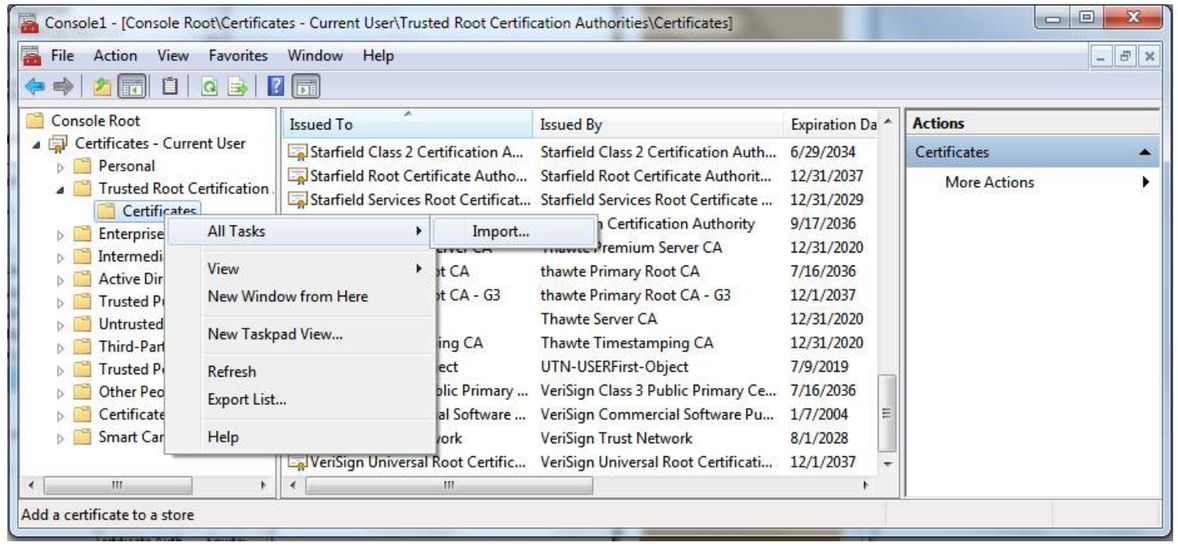
Click Next again to save the certificate

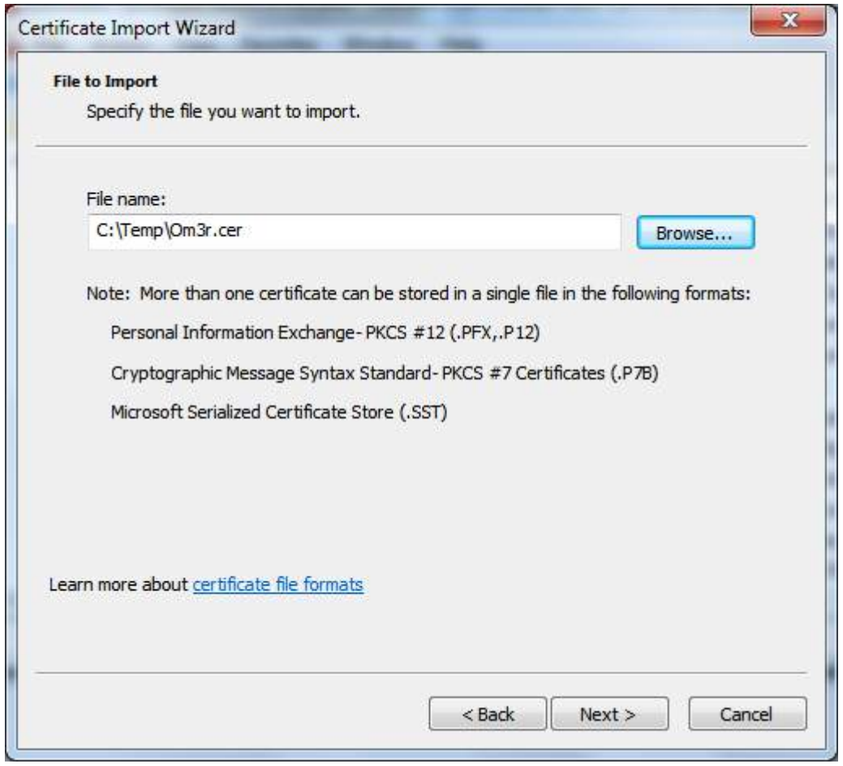
Once focus is returned to the Management Console.

Expand the *Certificates* menu and from the Trusted Root Certification Authorities menu, select *Certificates*.

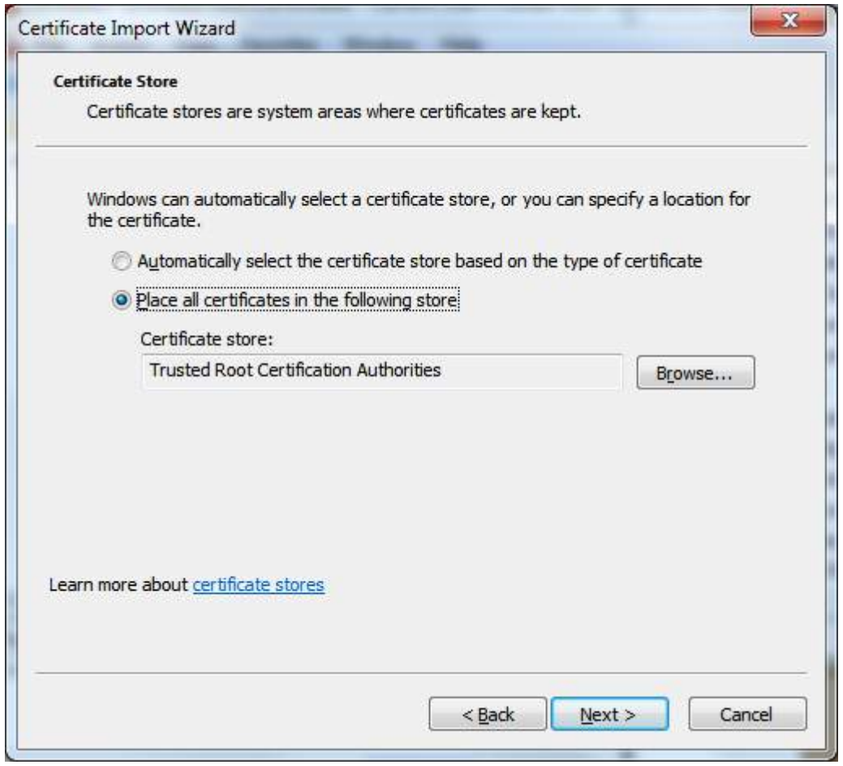


Right click. Select *All Tasks* and *Import*

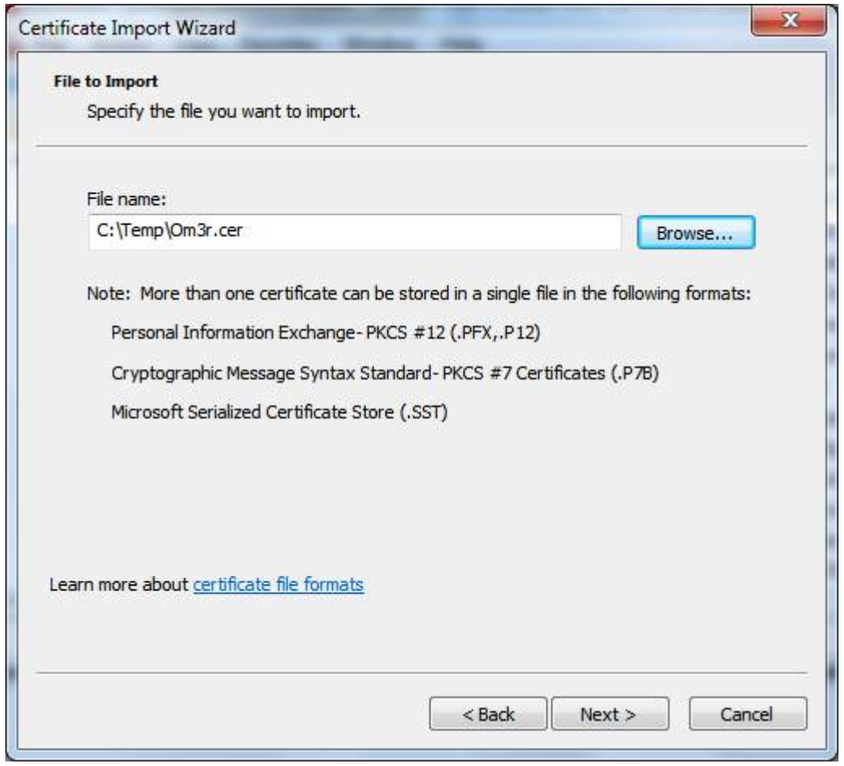




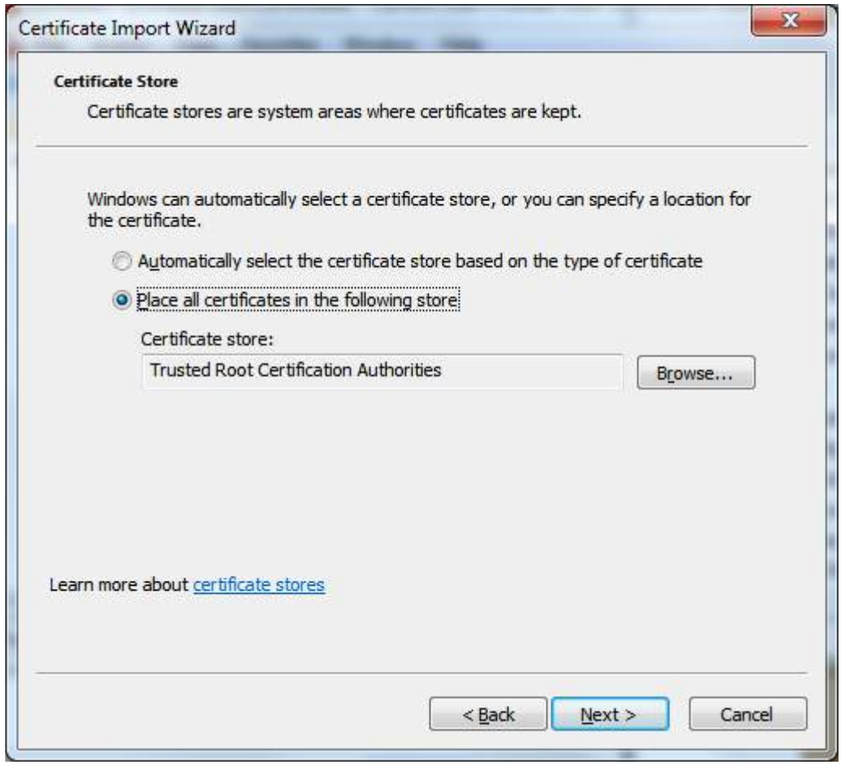
点击下一步并保存到受信任的根证书颁发机构存储：



然后点击下一步 > 完成，现在关闭控制台。



Click next and Save to the *Trusted Root Certification Authorities store*:



Then Next > Finish, now close the Console.

如果你现在使用该证书并检查其属性，你会看到它是一个受信任的证书，你可以用它来签署你的项目：

If you now use the certificate and check its properties, you will see that it is a trusted certificate and you can use it to sign your project:

第45章：VBA运行时错误

能够编译的代码仍然可能在运行时遇到错误。本主题列出了最常见的错误、它们的原因以及如何避免它们。

第45.1节：运行时错误 '6'：溢出

错误的代码

```
Sub DoSomething()
    Dim row As Integer
    For row = 1 To 100000
        '执行操作
    Next
End Sub
```

为什么这段代码不起作用？

Integer 数据类型是一个16位有符号整数，最大值为32,767；赋值超过该值时，会导致溢出并引发此错误。

正确的代码

```
Sub DoSomething()
    Dim row As Long
    For row = 1 To 100000
        '执行操作
    Next
End Sub
```

为什么这段代码能正常工作？

通过使用Long（32位）整数，我们现在可以创建一个循环，迭代次数超过32,767次，而不会导致计数变量类型溢出。

其他说明

有关更多信息，请参见数据类型和限制。

第45.2节：运行时错误 '9'：下标越界

错误的代码

```
Sub DoSomething()
    Dim foo(1 To 10)
    Dim i As Long
    For i = 1 To 100
       foo(i) = i
    Next
End Sub
```

为什么这段代码不起作用？

foo 是一个包含10个元素的数组。当 i 循环计数器达到11时，foo(i) 超出范围。每当使用不存在于该数组或集合中的索引访问数组或集合时，就会发生此错误。

正确的代码

```
Sub DoSomething()
    Dim foo(1 To 10)
    Dim i As Long
```


Chapter 45: VBA Run-Time Errors

Code that compiles can still run into errors, at run-time. This topic lists the most common ones, their causes, and how to avoid them.

Section 45.1: Run-time error '6': Overflow

Incorrect code

```
Sub DoSomething()
    Dim row As Integer
    For row = 1 To 100000
        'do stuff
    Next
End Sub
```

Why doesn't this work?

The `Integer` data type is a 16-bit signed integer with a maximum value of 32,767; assigning it to anything larger than that will *overflow* the type and raise this error.

Correct code

```
Sub DoSomething()
    Dim row As Long
    For row = 1 To 100000
        'do stuff
    Next
End Sub
```

Why does this work?

By using a `Long` (32-bit) integer instead, we can now make a loop that iterates more than 32,767 times without overflowing the counter variable's type.

Other notes

See Data Types and Limits for more information.

Section 45.2: Run-time error '9': Subscript out of range

Incorrect code

```
Sub DoSomething()
    Dim foo(1 To 10)
    Dim i As Long
    For i = 1 To 100
        foo(i) = i
    Next
End Sub
```

Why doesn't this work?

`foo` is an array that contains 10 items. When the `i` loop counter reaches a value of 11, `foo(i)` is *out of range*. This error occurs whenever an array or collection is accessed with an index that doesn't exist in that array or collection.

Correct code

```
Sub DoSomething()
    Dim foo(1 To 10)
    Dim i As Long
```

```
    For i = LBound(foo) To UBound(foo)
       foo(i) = i
    Next
End Sub
```

为什么这段代码能正常工作？

使用 LBound 和 UBound 函数分别确定数组的下界和上界。

其他说明

当索引是字符串时，例如 ThisWorkbook.Worksheets("我不存在")，此错误表示提供的名称在查询的集合中不存在。

不过，实际错误取决于具体实现；Collection 会引发运行时错误5“无效的过程调用或参数”：

```
Sub RaisesRunTimeError5()
    Dim foo As New Collection
    foo.Add "foo", "foo"
    Debug.Print foo("bar")
End Sub
```

第45.3节：运行时错误 '13'：类型不匹配

错误的代码

```
Public Sub DoSomething()
    DoSomethingElse "42?"
End Sub

Private Sub DoSomethingElse(foo As Date)
'    Debug.Print MonthName(Month(foo))
End Sub
```

为什么这段代码不起作用？

VBA 正在努力将"42?"参数转换为Date类型的值。当转换失败时，调用DoSomethingElse无法执行，因为 VBA 不知道传递哪个日期，因此会引发运行时错误 13 *类型不匹配*，因为参数的类型与预期类型不匹配（且无法隐式转换）。

正确的代码

```
Public Sub DoSomething()
    DoSomethingElse Now
End Sub

Private Sub DoSomethingElse(foo As Date)
'    Debug.Print MonthName(Month(foo))
End Sub
```

为什么这段代码能正常工作？

通过向期望Date参数的过程传递Date类型的参数，调用可以成功。

第 45.4 节：运行时错误 '91'：对象变量或 With 块变量未设置

错误的代码

```
Sub DoSomething()
```



```
    For i = LBound(foo) To UBound(foo)
        foo(i) = i
    Next
End Sub
```

Why does this work?

Use LBound and UBound functions to determine the lower and upper boundaries of an array, respectively.

Other notes

When the index is a string, e.g. `ThisWorkbook.Worksheets("I don't exist")`, this error means the supplied name doesn't exist in the queried collection.

The actual error is implementation-specific though; `Collection` will raise run-time error 5 "Invalid procedure call or argument" instead:

```
Sub RaisesRunTimeError5()
    Dim foo As New Collection
    foo.Add "foo", "foo"
    Debug.Print foo("bar")
End Sub
```

Section 45.3: Run-time error '13': Type mismatch

Incorrect code

```
Public Sub DoSomething()
    DoSomethingElse "42?"
End Sub

Private Sub DoSomethingElse(foo As Date)
'    Debug.Print MonthName(Month(foo))
End Sub
```

Why doesn't this work?

VBA is trying really hard to convert the `"42?"` argument into a `Date` value. When it fails, the call to `DoSomethingElse` cannot be executed, because VBA doesn't know what date to pass, so it raises run-time error 13 *type mismatch*, because the type of the argument doesn't match the expected type (and can't be implicitly converted either).

Correct code

```
Public Sub DoSomething()
    DoSomethingElse Now
End Sub

Private Sub DoSomethingElse(foo As Date)
'    Debug.Print MonthName(Month(foo))
End Sub
```

Why does this work?

By passing a `Date` argument to a procedure that expects a `Date` parameter, the call can succeed.

Section 45.4: Run-time error '91': Object variable or With block variable not set

Incorrect code

```
Sub DoSomething()
```

```
    Dim foo As Collection
    With foo
        .Add "ABC"
        .Add "XYZ"
    End With
End Sub
```

为什么这段代码不起作用？

对象变量保存一个引用，引用需要使用Set关键字进行设置。每当对引用为Nothing的对象调用成员时，就会发生此错误。在本例中，foo是一个Collection引用，但它未被初始化，因此引用包含Nothing——我们无法对Nothing调用.Add。

正确的代码

```
Sub DoSomething()
    Dim foo As Collection
    Set foo = New Collection
    With foo
        .Add "ABC"
        .Add "XYZ"
    End With
End Sub
```

为什么这段代码能正常工作？

通过使用Set关键字为对象变量分配一个有效的引用，.Add调用才能成功。

其他说明

通常，函数或属性可以返回一个对象引用——一个常见的例子是 Excel 的`Range.Find`方法，它返回一个`Range`对象：

```
Dim resultRow As Long
resultRow = SomeSheet.Cells.Find("Something").Row
```

但是该函数很可能返回Nothing（如果未找到搜索项），因此链式调用`.Row`成员调用可能会失败。

在调用对象成员之前，使用If Not xxxx Is Nothing条件验证引用是否已设置：

```
Dim result As Range
Set result = SomeSheet.Cells.Find("Something")

Dim resultRow As Long
If Not result Is Nothing Then resultRow = result.Row
```

第45.5节：运行时错误 '20'：无错误时恢复

错误的代码

```
Sub DoSomething()
    On Error GoTo CleanFail
    DoSomethingElse

CleanFail:
    Debug.Print Err.Number
    Resume Next
End Sub
```

为什么这段代码不起作用？


```
    Dim foo As Collection
    With foo
        .Add "ABC"
        .Add "XYZ"
    End With
End Sub
```

Why doesn't this work?

Object variables hold a *reference*, and references need to be *set* using the `Set` keyword. This error occurs whenever a member call is made on an object whose reference is `Nothing`. In this case `foo` is a `Collection` reference, but it's not initialized, so the reference contains `Nothing` - and we can't call `.Add` on `Nothing`.

Correct code

```
Sub DoSomething()
    Dim foo As Collection
    Set foo = New Collection
    With foo
        .Add "ABC"
        .Add "XYZ"
    End With
End Sub
```

Why does this work?

By assigning the object variable a valid reference using the `Set` keyword, the `.Add` calls succeed.

Other notes

Often, a function or property can return an object reference - a common example is Excel's `Range.Find` method, which returns a `Range` object:

```
Dim resultRow As Long
resultRow = SomeSheet.Cells.Find("Something").Row
```

However the function can very well return `Nothing` (if the search term isn't found), so it's likely that the chained `.Row` member call fails.

Before calling object members, verify that the reference is set with a `If Not xxxx Is Nothing` condition:

```
Dim result As Range
Set result = SomeSheet.Cells.Find("Something")

Dim resultRow As Long
If Not result Is Nothing Then resultRow = result.Row
```

Section 45.5: Run-time error '20': Resume without error

Incorrect code

```
Sub DoSomething()
    On Error GoTo CleanFail
    DoSomethingElse

CleanFail:
    Debug.Print Err.Number
    Resume Next
End Sub
```

Why doesn't this work?

如果DoSomethingElse过程引发错误，执行将跳转到CleanFail行标签，打印错误编号，然后Resume Next指令跳回紧跟错误发生行之后的指令，在本例中是Debug.Print指令：错误处理子程序在没有错误上下文的情况下执行，当执行到Resume Next指令时，会引发运行时错误20，因为没有可恢复的地方。

正确代码

```
Sub DoSomething()
    On Error GoTo CleanFail
   DoSomethingElse

    Exit Sub
CleanFail:
    Debug.Print Err.Number
    Resume Next
End Sub
```

为什么这段代码能正常工作？

通过在CleanFail行标签之前引入Exit Sub指令，我们将CleanFail错误处理子程序与过程主体的其余部分分离开来——执行错误处理子程序的唯一方式是通过On Error跳转；因此，正常执行路径不会在没有错误上下文的情况下执行Resume指令，从而避免了运行时错误20。

其他说明

这与运行时错误'3'：Return without GoSub非常相似；在这两种情况下，解决方案都是确保正常执行路径不能进入没有显式跳转的子程序（由行标签标识）（假设On Error GoTo被视为显式跳转）。
Error GoTo被视为显式跳转）。

第45.6节：运行时错误 '3'：Return语句缺少对应的GoSub

错误代码

```
Sub DoSomething()
    GoSub DoThis
DoThis:
Debug.Print "Hi!"
    Return
End Sub
```

为什么这段代码不起作用？

执行进入DoSomething过程，跳转到DoThis标签，向调试输出打印“Hi!”，返回到GoSub调用之后的指令，再次打印“Hi!”，然后遇到Return语句，但现在没有地方可以返回，因为我们不是通过GoSub语句到达这里的。

正确代码

```
Sub DoSomething()
    GoSub DoThis
    Exit Sub
DoThis:
Debug.Print "Hi!"
    Return
End Sub
```

为什么这段代码能正常工作？

通过在DoThis行标签之前引入Exit Sub指令，我们将DoThis子程序与

If the DoSomethingElse procedure raises an error, execution jumps to the CleanFail line label, prints the error number, and the **Resume Next** instruction jumps back to the instruction that immediately follows the line where the error occurred, which in this case is the Debug.Print instruction: the error-handling subroutine is executing without an error context, and when the **Resume Next** instruction is reached, run-time error 20 is raised because there is nowhere to resume to.

Correct Code

```
Sub DoSomething()
    On Error GoTo CleanFail
    DoSomethingElse

    Exit Sub
CleanFail:
    Debug.Print Err.Number
    Resume Next
End Sub
```

Why does this work?

By introducing an **Exit Sub** instruction before the CleanFail line label, we have segregated the CleanFail error-handling subroutine from the rest of the procedure body - the only way to execute the error-handling subroutine is via an **On Error** jump; therefore, no execution path reaches the **Resume** instruction outside of an error context, which avoids run-time error 20.

Other notes

This is very similar to Run-time error '3': Return without GoSub; in both situations, the solution is to ensure that the *normal execution path* cannot enter a sub-routine (identified by a line label) without an explicit jump (assuming **On Error GoTo** is considered an *explicit jump*).

Section 45.6: Run-time error '3': Return without GoSub

Incorrect Code

```
Sub DoSomething()
    GoSub DoThis
DoThis:
    Debug.Print "Hi!"
    Return
End Sub
```

Why doesn't this work?

Execution enters the DoSomething procedure, jumps to the DoThis label, prints "Hi!" to the debug output, *returns* to the instruction immediately after the **GoSub** call, prints "Hi!" again, and then encounters a **Return** statement, but there's nowhere to *return* to now, because we didn't get here with a **GoSub** statement.

Correct Code

```
Sub DoSomething()
    GoSub DoThis
    Exit Sub
DoThis:
    Debug.Print "Hi!"
    Return
End Sub
```

Why does this work?

By introducing an **Exit Sub** instruction *before* the DoThis line label, we have segregated the DoThis subroutine from

过程主体的其余部分分开——执行DoThis子程序的唯一方式是通过GoSub跳转。

其他说明

`GoSub`/`Return` 已被弃用，应避免使用，改用实际的过程调用。一个过程不应包含子程序，除非是错误处理程序。

这与运行时错误 '20'：无错误时恢复 非常相似；在这两种情况下，解决方案是确保*正常执行路径* 不能进入没有显式跳转的子程序（由行标签标识）（假设 `On Error GoTo`被视为显式跳转）。

the rest of the procedure body - the only way to execute the `DoThis` subroutine is via the `GoSub` jump.

Other notes

`GoSub`/`Return` is deprecated, and should be avoided in favor of actual procedure calls. A procedure should not contain subroutines, other than error handlers.

This is very similar to Run-time error '20': Resume without error; in both situations, the solution is to ensure that the *normal execution path* cannot enter a sub-routine (identified by a line label) without an explicit jump (assuming `On Error GoTo` is considered an *explicit jump*).

第46章：错误处理

第46.1节：避免错误情况

当运行时错误发生时，良好的代码应当处理它。最佳的错误处理策略是编写代码
检查错误条件，并简单地避免执行导致运行时错误的代码。

减少运行时错误的一个关键要素是编写执行*单一功能*的小过程。过程失败的原因越少，
整个代码就越容易调试。

避免运行时错误91 - 对象或With块变量未设置：

当在分配引用之前使用对象时，会引发此错误。可能有一个过程
接收一个对象参数：

```
Private Sub DoSomething(ByVal target As Worksheet)
    Debug.Print target.Name
End Sub
```

如果 target 未被分配引用，上述代码将引发错误，这可以通过检查对象
是否包含实际对象引用来轻松避免：

```
Private Sub DoSomething(ByVal target As Worksheet)
    If target Is Nothing Then Exit Sub
    Debug.Print target.Name
End Sub
```

如果target未被赋予引用，则未赋值的引用永远不会被使用，也不会发生错误。

当一个或多个参数无效时，提前退出过程的这种方式称为保护子句（guard clause）。

避免运行时错误9 - 下标越界：

当访问数组超出其边界时，会引发此错误。

```
Private Sub DoSomething(ByVal index As Integer)
    Debug.Print ActiveWorkbook.Worksheets(index)
End Sub
```

如果给定的索引大于ActiveWorkbook中工作表的数量，上述代码将引发运行时
错误。一个简单的保护子句可以避免这种情况：

```
Private Sub DoSomething(ByVal index As Integer)
    If index > ActiveWorkbook.Worksheets.Count Or index <= 0 Then Exit Sub
    Debug.Print ActiveWorkbook.Worksheets(index)
End Sub
```

大多数运行时错误可以通过在使用值之前仔细验证我们使用的值来避免，并使用简单的If语句相应地分支
到另一条执行路径——在保护子句中不做任何假设
并验证过程的参数，甚至在较大过程的主体中也是如此。

第46.2节：自定义错误

通常在编写专用类时，你会希望它抛出自己的特定错误，并且希望调用代码能够干净利落地处理这些自定义错误。

Chapter 46: Error Handling

Section 46.1: Avoiding error conditions

When a runtime error occurs, good code should handle it. The best error handling strategy is to write code that checks for error conditions and simply avoids executing code that results in a runtime error.

One key element in reducing runtime errors, is writing small procedures that *do one thing*. The fewer reasons procedures have to fail, the easier the code as a whole is to debug.

Avoiding runtime error 91 - Object or With block variable not set:

This error will be raised when an object is used before its reference is assigned. One might have a procedure that receives an object parameter:

```
Private Sub DoSomething(ByVal target As Worksheet)
    Debug.Print target.Name
End Sub
```

If `target` isn't assigned a reference, the above code will raise an error that is easily avoided by checking if the object contains an actual object reference:

```
Private Sub DoSomething(ByVal target As Worksheet)
    If target Is Nothing Then Exit Sub
    Debug.Print target.Name
End Sub
```

If `target` isn't assigned a reference, then the unassigned reference is never used, and no error occurs.

This way of early-exiting a procedure when one or more parameter isn't valid, is called a *guard clause*.

Avoiding runtime error 9 - Subscript out of range:

This error is raised when an array is accessed outside of its boundaries.

```
Private Sub DoSomething(ByVal index As Integer)
    Debug.Print ActiveWorkbook.Worksheets(index)
End Sub
```

Given an index greater than the number of worksheets in the `ActiveWorkbook`, the above code will raise a runtime error. A simple guard clause can avoid that:

```
Private Sub DoSomething(ByVal index As Integer)
    If index > ActiveWorkbook.Worksheets.Count Or index <= 0 Then Exit Sub
    Debug.Print ActiveWorkbook.Worksheets(index)
End Sub
```

Most runtime errors can be avoided by carefully verifying the values we're using *before* we use them, and branching on another execution path accordingly using a simple `If` statement - in guard clauses that makes no assumptions and validates a procedure's parameters, or even in the body of larger procedures.

Section 46.2: Custom Errors

Often when writing a specialized class, you'll want it to raise its own specific errors, and you'll want a clean way for

实现这一点的一个巧妙方法是定义一个专用的Enum类型：

```
Option Explicit
Public Enum FoobarError
Err_FooWasNotBarred = vbObjectError + 1024
    Err_BarNotInitialized
    Err_SomethingElseHappened
End Enum
```

使用内置常量vbObjectError确保自定义错误代码不会与保留/现有的错误代码重叠。只需显式指定第一个枚举值，因为每个Enum成员的底层值比前一个成员大1，所以Err_BarNotInitialized的底层值隐式为vbObjectError + 1025。

抛出自定义运行时错误

可以使用Err.Raise语句抛出运行时错误，因此可以如下抛出自定义的Err_FooWasNotBarred错误：

```
Err.Raise Err_FooWasNotBarred
```

Err.Raise方法还可以接受自定义的Description和Source参数——因此，定义常量来保存每个自定义错误的描述是个好主意：

```
Private Const Msg_FooWasNotBarred As String = "该 foo 未被禁止。"
Private Const Msg_BarNotInitialized As String = "该 bar 未被初始化。"
```

然后创建一个专用的私有方法来引发每个错误：

```
Private Sub OnFooWasNotBarredError(ByVal source As String)
    Err.Raise Err_FooWasNotBarred, source, Msg_FooWasNotBarred
End Sub

Private Sub OnBarNotInitializedError(ByVal source As String)
    Err.Raise Err_BarNotInitialized, source, Msg_BarNotInitialized
End Sub
```

该类的实现随后可以简单地调用这些专门的过程来引发错误：

```
Public Sub DoSomething()
    '引发自定义的"BarNotInitialized"错误，源为"DoSomething"：
    If Me.Bar Is Nothing Then OnBarNotInitializedError "DoSomething"
    '...
End Sub
```

客户端代码随后可以像处理其他错误一样，在其自己的错误处理子程序中处理Err_BarNotInitialized。

注意：传统的Error关键字也可以替代Err.Raise使用，但它已过时/弃用。

第46.3节：Resume关键字

错误处理子程序将会：

- 运行到过程结束，在这种情况下，执行将恢复到调用过程。
- 或者，使用Resume关键字在同一过程内恢复执行。

user/calling code to handle these custom errors. A neat way to achieve this is by defining a dedicated `Enum` type:

```
Option Explicit
Public Enum FoobarError
    Err_FooWasNotBarred = vbObjectError + 1024
    Err_BarNotInitialized
    Err_SomethingElseHappened
End Enum
```

Using the `vbObjectError` built-in constant ensures the custom error codes don't overlap with reserved/existing error codes. Only the first enum value needs to be explicitly specified, for the underlying value of each `Enum` member is 1 greater than the previous member, so the underlying value of `Err_BarNotInitialized` is implicitly `vbObjectError + 1025`.

Raising your own runtime errors

A runtime error can be raised using the `Err.Raise` statement, so the custom `Err_FooWasNotBarred` error can be raised as follows:

```
Err.Raise Err_FooWasNotBarred
```

The `Err.Raise` method can also take custom `Description` and `Source` parameters - for this reason it's a good idea to also define constants to hold each custom error's description:

```
Private Const Msg_FooWasNotBarred As String = "The foo was not barred."
Private Const Msg_BarNotInitialized As String = "The bar was not initialized."
```

And then create a dedicated private method to raise each error:

```
Private Sub OnFooWasNotBarredError(ByVal source As String)
    Err.Raise Err_FooWasNotBarred, source, Msg_FooWasNotBarred
End Sub

Private Sub OnBarNotInitializedError(ByVal source As String)
    Err.Raise Err_BarNotInitialized, source, Msg_BarNotInitialized
End Sub
```

The class' implementation can then simply call these specialized procedures to raise the error:

```
Public Sub DoSomething()
    'raises the custom 'BarNotInitialized' error with "DoSomething" as the source:
    If Me.Bar Is Nothing Then OnBarNotInitializedError "DoSomething"
    '...
End Sub
```

The client code can then handle `Err_BarNotInitialized` as it would any other error, inside its own error-handling subroutine.

Note: the legacy `Error` keyword can also be used in place of `Err.Raise`, but it's obsolete/deprecated.

Section 46.3: Resume keyword

An error-handling subroutine will either:

- run to the end of the procedure, in which case execution resumes in the calling procedure.
- or, use the `Resume` keyword to *resume* execution inside the same procedure.

Resume关键字应仅在错误处理子程序内使用，因为如果VBA遇到Resume而不处于错误状态，将引发运行时错误20“无错误时的Resume”。

错误处理子程序可以通过多种方式使用Resume关键字：

- **单独使用Resume，执行将继续在导致错误的语句上。如果错误未被实际处理，则同一错误将再次引发，执行可能进入无限循环。**
- Resume Next继续执行在导致错误的语句之后的语句上。如果错误未被实际处理，则允许执行继续，可能使用无效数据，导致逻辑错误和意外行为。
- Resume [行标签]继续执行在指定的行标签处（如果使用传统行号，则为行号）。这通常允许在干净退出过程之前执行一些清理代码，例如确保数据库连接关闭后返回调用者。

出错时继续下一条

On Error 语句本身可以使用 Resume 关键字来指示 VBA 运行时有效地忽略所有**错误**。

> *如果错误在执行之前没有被**实际处理**，那么执行将被允许继续，可能会使用无效的数据，这可能导致**逻辑错误和意外行为**。*

以上强调的重要性无法过分强调。On Error Resume Next 实际上忽略了所有错误并将它们掩盖起来。一个在遇到无效输入时因运行时错误而崩溃的程序，比起一个在未知/非预期数据下继续运行的程序更好——至少因为错误更容易被识别。On Error Resume Next 很容易隐藏错误。

On Error 语句是过程范围的——这就是为什么在给定的过程里通常应该只有一个单独的On Error语句。

然而，有时错误情况无法完全避免，跳转到错误处理子程序然后 `Resume Next` 只是感觉不对。在这种特定情况下，已知可能失败的语句可以被**包裹在**两个`On Error`语句之间：

```
On Error Resume Next
[可能失败的语句]
Err.Clear '重置当前错误
On Error GoTo 0
```

On Error GoTo 0 指令重置当前过程中的错误处理，这样任何进一步导致运行时错误的指令将在该过程内不被处理，而是传递到调用堆栈上层，直到被活动的错误处理程序捕获。如果调用堆栈中没有活动的错误处理程序，则该错误将被视为未处理的异常。

```
Public Sub Caller()
    On Error GoTo Handler

    Callee

    退出子程序
处理程序：
```


The `Resume` keyword should only ever be used inside an error handling subroutine, because if VBA encounters `Resume` without being in an error state, runtime error 20 "Resume without error" is raised.

There are several ways an error-handling subroutine may use the `Resume` keyword:

- `Resume` used alone, execution continues **on the statement that caused the error**. If the error isn't *actually* handled before doing that, then the same error will be raised again, and execution might enter an infinite loop.
- `Resume Next` continues execution **on the statement immediately following** the statement that caused the error. If the error isn't *actually* handled before doing that, then execution is permitted to continue with potentially invalid data, which may result in logical errors and unexpected behavior.
- `Resume [line label]` continues execution **at the specified line label** (or line number, if you're using legacy-style line numbers). This would typically allow executing some cleanup code before cleanly exiting the procedure, such as ensuring a database connection is closed before returning to the caller.

On Error Resume Next

The `On Error` statement itself can use the `Resume` keyword to instruct the VBA runtime to effectively **ignore all errors**.

> *If the error isn't **actually handled** before doing that, then execution is permitted to continue with potentially invalid data, which may result in **logical errors and unexpected behavior**.*

The emphasis above cannot be emphasized enough. ***On Error Resume Next* effectively ignores all errors and shoves them under the carpet**. A program that blows up with a runtime error given invalid input is a better program than one that keeps running with unknown/unintended data - be it only because the bug is much more easily identifiable. `On Error Resume Next` can easily **hide bugs**.

The `On Error` statement is procedure-scoped - that's why there should *normally* be only **one**, single such `On Error` statement in a given procedure.

However *sometimes* an error condition can't quite be avoided, and jumping to an error-handling subroutine only to `Resume Next` just doesn't feel right. In this specific case, the known-to-possibly-fail statement can be **wrapped** between two `On Error` statements:

```
On Error Resume Next
[possibly-failing statement]
Err.Clear 'resets current error
On Error GoTo 0
```

The `On Error GoTo 0` instruction resets error handling in the current procedure, such that any further instruction causing a runtime error *would be unhandled within that procedure* and instead passed up the call stack until it is caught by an active error handler. If there is no active error handler in the call stack, it will be treated as an unhandled exception.

```
Public Sub Caller()
    On Error GoTo Handler

    Callee

    Exit Sub
Handler:
```

```
Debug.Print "错误 " & Err.Number & " 在 Caller 中。"
End Sub

Public Sub Callee()
    出错时转到处理程序

Err.Raise 1     '这将由被调用者的处理程序处理。
    出错时转到 0 '在此语句之后，错误将向上传递堆栈。
Err.Raise 2     '这将由调用者的处理程序处理。

    退出子程序
处理程序：
Debug.Print "错误 " & Err.Number & " 在被调用者中。"
    Resume Next
End Sub
```

第46.4节：On Error语句

即使有保护子句，也无法现实地总是考虑到过程主体中可能引发的所有错误情况。On Error GoTo语句指示VBA在运行时遇到意外错误时跳转到行标签并进入“错误处理模式”。处理错误后，代码可以使用Resume关键字恢复回“正常”执行。

行标签表示子程序：由于子程序源自传统BASIC代码，使用GoTo和GoSub跳转以及Return语句跳回“主”程序，如果结构不严谨，很容易写出难以理解的意大利面条代码。因此，最好是：

- 一个过程只有**一个且仅有一个**错误处理子程序
- 错误处理子程序**仅在错误状态下运行**

这意味着一个处理其错误的过程，应当结构如下：

```
Private Sub DoSomething()
    On Error GoTo CleanFail

    'procedure code here

清理退出：
    'cleanup code here
    退出子程序

CleanFail:
    'error-handling code here
    恢复 清理退出
结束子程序
```

错误处理策略

有时你希望针对不同的错误采取不同的处理措施。在这种情况下，你将检查全局的Err对象，该对象包含有关引发的错误的信息——并据此采取相应的行动：

```
清理退出：
    退出子程序

CleanFail:
    Select Case Err.Number
        Case 9
MsgBox "指定的编号不存在。请再试一次。", vbExclamation
```



```
    Debug.Print "Error " & Err.Number & " in Caller."
End Sub

Public Sub Callee()
    On Error GoTo Handler

    Err.Raise 1     'This will be handled by the Callee handler.
    On Error GoTo 0 'After this statement, errors are passed up the stack.
    Err.Raise 2     'This will be handled by the Caller handler.

    Exit Sub
Handler:
    Debug.Print "Error " & Err.Number & " in Callee."
    Resume Next
End Sub
```

Section 46.4: On Error statement

Even with *guard clauses*, one cannot realistically *always* account for all possible error conditions that could be raised in the body of a procedure. The `On Error GoTo` statement instructs VBA to jump to a *line label* and enter "error handling mode" whenever an unexpected error occurs at runtime. After handling an error, code can *resume* back into "normal" execution using the `Resume` keyword.

Line labels denote *subroutines*: because subroutines originate from legacy BASIC code and uses `GoTo` and `GoSub` jumps and `Return` statements to jump back to the "main" routine, it's fairly easy to write hard-to-follow *spaghetti code* if things aren't rigorously structured. For this reason, it's best that:

- a procedure has **one and only one** error-handling subroutine
- the error-handling subroutine **only ever runs in an error state**

This means a procedure that handles its errors, should be structured like this:

```
Private Sub DoSomething()
    On Error GoTo CleanFail

    'procedure code here

CleanExit:
    'cleanup code here
    Exit Sub

CleanFail:
    'error-handling code here
    Resume CleanExit
End Sub
```

Error Handling Strategies

Sometimes you want to handle different errors with different actions. In that case you will inspect the global `Err` object, which will contain information about the error that was raised - and act accordingly:

```
CleanExit:
    Exit Sub

CleanFail:
    Select Case Err.Number
        Case 9
            MsgBox "Specified number doesn't exist. Please try again.", vbExclamation
```

```
            Resume
        Case 91
            哇哦，这不应该发生。
            停止执行将在此处中断
            继续按F8跳转到引发错误的行
        Case Else
MsgBox"发生了意外错误：" & vbNewLine & Err.Description, vbCritical
            继续 CleanExit
    End Select
End Sub
```

一般来说，建议为整个子程序或函数开启错误处理，并处理其范围内可能发生的所有错误。如果只需要处理代码中一小部分的错误——则在同一级别开启和关闭错误处理：

```
Private Sub DoSomething(CheckValue as Long)

    如果 CheckValue = 0 则
        On Error GoTo ErrorHandler    ' 开启错误处理
        ' 可能导致错误的代码
        On Error GoTo 0             ' 关闭错误处理 - 同一级别
    End If

清理退出：
    退出子程序

ErrorHandler:
    '这里是错误处理代码
    '不要在这里关闭错误处理
    Resume

End Sub
```

行号

VBA 支持传统风格（例如 QBASIC）的行号。隐藏属性Erl可用于识别引发最后一个错误的行号。如果不使用行号，Erl将始终返回0。

```
Sub DoSomething()
10 出错时转到 50
20调试打印 42 / 0
30 退出子程序
40

    打印 "错误发生在第 " & Erl ' 返回20
结束子程序
```

如果你使用行号，但不一致，那么Erl将返回引发错误的指令之前的最后一个行号。

```
Sub DoSomething()
10 出错时转到 50
   调试打印 42 / 0
30 退出子程序

50调试打印 "错误发生在第 " & Erl '返回10
End Sub
```

请记住，Erl 也仅具有整数精度，并且会悄无声息地溢出。这意味着行号


```
            Resume
        Case 91
            'woah there, this shouldn't be happening.
            Stop 'execution will break here
            Resume 'hit F8 to jump to the line that raised the error
        Case Else
            MsgBox "An unexpected error has occurred:" & vbNewLine & Err.Description, vbCritical
            Resume CleanExit
    End Select
End Sub
```

As a general guideline, consider turning on the error handling for entire subroutine or function, and handle all the errors that may occur within its scope. If you need to only handle errors in the small section section of the code -- turn error handling on and off a the same level:

```
Private Sub DoSomething(CheckValue as Long)

    If CheckValue = 0 Then
        On Error GoTo ErrorHandler    ' turn error handling on
        ' code that may result in error
        On Error GoTo 0               ' turn error handling off - same level
    End If

CleanExit:
    Exit Sub

ErrorHandler:
    ' error handling code here
    ' do not turn off error handling here
    Resume

End Sub
```

Line numbers

VBA supports legacy-style (e.g. QBASIC) line numbers. The `Erl` hidden property can be used to identify the line number that raised the last error. If you're not using line numbers, `Erl` will only ever return 0.

```
Sub DoSomething()
10 On Error GoTo 50
20 Debug.Print 42 / 0
30 Exit Sub
40
50 Debug.Print "Error raised on line " & Erl ' returns 20
End Sub
```

If you *are* using line numbers, but not consistently, then `Erl` will return *the last line number before the instruction that raised the error*.

```
Sub DoSomething()
10 On Error GoTo 50
   Debug.Print 42 / 0
30 Exit Sub

50 Debug.Print "Error raised on line " & Erl 'returns 10
End Sub
```

Keep in mind that `Erl` also only has `Integer` precision, and will silently overflow. This means that line numbers

超出整数范围将导致错误的结果：

```
Sub DoSomething()
99997 On Error GoTo 99999
99998调试打印 42 / 0
99999
Debug.Print Erl    '打印 34462
End Sub
```

行号并不像导致错误的语句那样重要，而且快速编号行会变得繁琐且不太利于维护。

outside of the integer range will give incorrect results:

```
Sub DoSomething()
99997 On Error GoTo 99999
99998 Debug.Print 42 / 0
99999
      Debug.Print Erl    'Prints 34462
End Sub
```

The line number isn't quite as relevant as the statement that caused the error, and numbering lines quickly becomes tedious and not quite maintenance-friendly.

鸣谢

非常感谢所有来自 Stack Overflow Documentation 的人员帮助提供这些内容，
更多更改可以发送至 web@petercv.com 以发布或更新新内容

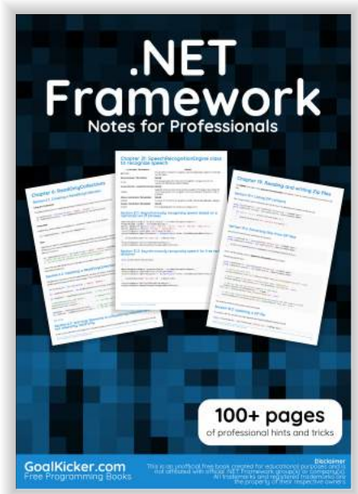
0m3r	第1章和第44章
安迪·特拉拉	第1章
贝诺·格里姆	第1章和第23章
黑鹰	第17章
书食者	第1章
布拉尼斯拉夫·科拉尔	第24章、第40章和第43章
共产国际	第3、5、6、14、15、18、20、21、25、26、28、36、37和46章
dadde	第5章
戴夫	第5、18和25章
Derpcode	第1章
自由人	第14和15章
Hosch250	第二章
Hubisan	第18章
hymced	第38章
IvenBach	第31章
Jeeped	第4、5和36章
Kaz	第5和16章
凯尔	第36章
利亚姆H	第28章
litelite	第1章和第23章
洛根·里德	第46章
马特·尤哈斯	第1章
马尔滕·范斯塔姆	第1、4和18章
宏观人	第1、4、23、29和30章
马克·R	第5和19章
马丁	第23章
马修·甘东	第4、5、9、16、18、23、24、28、29、31、35、38、39、45和46章
米格尔_柳	第18章
尼尔·马塞特	第5、15、22、29、32、33和41章
尼克·德威特	第1章
帕舒特	第8章
帕特里克	第27章
保罗·比卡	第42章
R3uK	第24章
罗兰	第23章
橡皮鸭	第4、25、30和38章
SandPiper	第26章
肖恩·V·威尔逊	第2和5章
西瓦普拉萨斯·瓦迪韦尔	第28章
斯特凡·皮诺	第1和4章
史蒂夫·林兹伯格	第12、25和30章
SWa	第5章
Tazaf	第18章
蒂埃里·达隆	第5章
托马斯·G	第4章和第14章
ThunderFrame	第2、3、4、5、6、7、9、10、11、12、13、15、23、25、32和34章
蒂姆	第40章

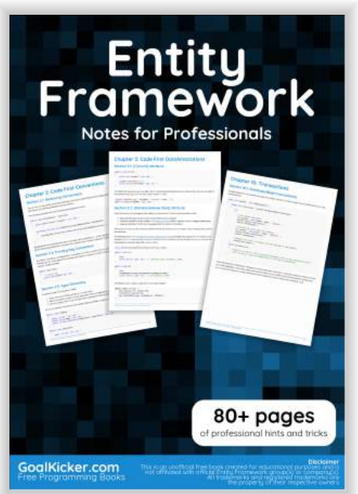
Credits

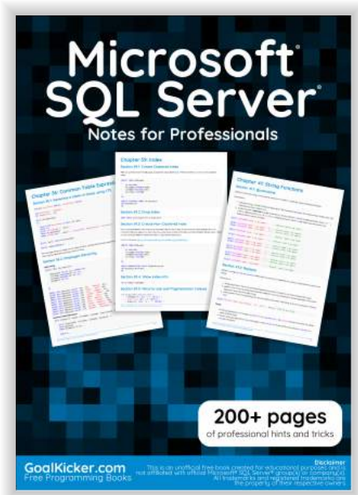
Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

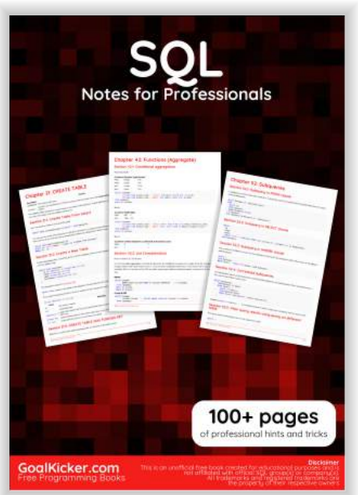
0m3r	Chapters 1 and 44
Andy Terra	Chapter 1
Benno Grimm	Chapters 1 and 23
Blackhawk	Chapter 17
Bookeater	Chapter 1
Branislav Kollár	Chapters 24, 40 and 43
Comintern	Chapters 3, 5, 6, 14, 15, 18, 20, 21, 25, 26, 28, 36, 37 and 46
dadde	Chapter 5
Dave	Chapters 5, 18 and 25
Derpcode	Chapter 1
FreeMan	Chapters 14 and 15
Hosch250	Chapter 2
Hubisan	Chapter 18
hymced	Chapter 38
IvenBach	Chapter 31
Jeeped	Chapters 4, 5 and 36
Kaz	Chapters 5 and 16
Kyle	Chapter 36
LiamH	Chapter 28
litelite	Chapters 1 and 23
Logan Reed	Chapter 46
Máté Juhász	Chapter 1
Maarten van Stam	Chapters 1, 4 and 18
Macro Man	Chapters 1, 4, 23, 29 and 30
Mark.R	Chapters 5 and 19
Martin	Chapter 23
Mathieu Guindon	Chapters 4, 5, 9, 16, 18, 23, 24, 28, 29, 31, 35, 38, 39, 45 and 46
Miguel_Ryu	Chapter 18
Neil Mussett	Chapters 5, 15, 22, 29, 32, 33 and 41
Nick Dewitt	Chapter 1
pashute	Chapter 8
PatricK	Chapter 27
paul bica	Chapter 42
R3uK	Chapter 24
Roland	Chapter 23
RubberDuck	Chapters 4, 25, 30 and 38
SandPiper	Chapter 26
Shawn V. Wilson	Chapters 2 and 5
Sivaprasath Vadivel	Chapter 28
Stefan Pinnow	Chapters 1 and 4
Steve Rindsberg	Chapters 12, 25 and 30
SWa	Chapter 5
Tazaf	Chapter 18
Thierry Dalon	Chapter 5
Thomas G	Chapters 4 and 14
ThunderFrame	Chapters 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 15, 23, 25, 32 and 34
Tim	Chapter 40

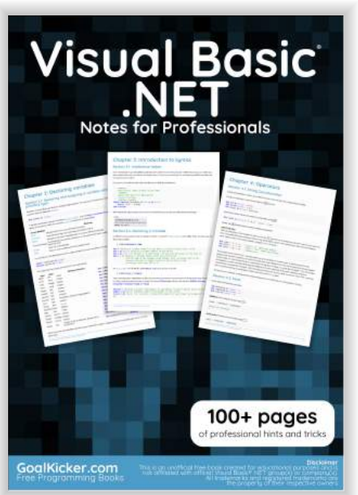
你可能也喜欢











You may also like