关于本教程

本教程主要针对那些想学习 AutoIt 但是找不到入门途径的朋友，希望我的这篇拙作能带领你走进 AutoIt 的大门。

说一下本教程的使用方法：

1、 教程中没有对出现的函数进行详细的解释，因为帮助文档已经解释的很清楚了。遇到不懂的函数，请参阅帮助文档。

2、 我尽量为每个小节都写一段实例，以代码来说明问题。但是希望看的朋友能细心点，不要一目十行。这样既表现出你对我辛劳的尊重，也体现了你认真学习的态度。

3、 本教程均由我一人编写完成，其中难免会出现各种漏洞错误，希望这些错误不会给你带去麻烦。如果你能把发现的错误告知于我，我非常感谢！

如果教程中出现了 BUG 或者你有看不懂的地方，可以给我发邮件，我的邮箱地址:382869232@qq.com

或者到我的博客提出你的疑问，我的博客：crossdoor.cublog.cn

希望我们能够共同进步！

CrossDoor 2010-06-01

目　　录

第一章 AutoIt 基础
- 1、关于 AutoIt
- 2、变量、常量和数据结构
- 3、运算符、宏
- 4、流程控制
  - 4.1、选择语句
  - 4.2、分支语句
  - 4.3、循环语句
- 5、函数
  - 5.1、自定义函数
  - 5.2、函数的参数传递
  - 5.3、函数的变量作用域
  - 5.4、函数的嵌套与递归

第二章 窗口
- 1、第一个窗口程序
  - 1.1、窗口消息
  - 1.2、消息拦截
- 2、多窗口程序
  - 2.1、父窗口与子窗口
  - 2.2、GUI 嵌入外部进程窗口

第三章 字符串与变量转换
- 1、字符串处理
  - 1.1、字符串长度
  - 1.2、字符串截取
  - 1.3、字符串替换
  - 1.4、字符串分割
  - 1.5、正则
- 2、变量转换
  - 2.1、转换为指针
  - 2.2、转换为句柄
  - 2.3、转换为整数
  - 2.4、转换为二进制数据

第四章 数组
- 1、一维数组
- 2、二维及多维数组
  - 2.1、数组的维数
  - 2.2、数组调整

第五章 注册表读写

1、读取注册表
2、写入注册表

第六章 文件读写
1、Ini 配置文件读写
2、Txt 文档读写
3、二进制文件读写

第七章 进程管理
1、进程列表
2、进程等待及结束
3、运行文件

第八章 窗口管理
1、窗口列表
2、窗口等待及结束
3、窗口自动化操作
3.1、按键发送
3.2、控件控制

第九章 定时器的应用
1、 内置函数定时器
2、 API 定时器

第十章 Com 对象调用
1、 创建 Com 对象
2、 拦截 Com 对象错误
3、 Com 对象使用实例

第十一章 动态链接库调用
1、 调用动态链接库
2、 系统 API

第十二章 网络编程
1、 Windows Socket 接口简介
2、 TCP 应用程序设计
2.1、TCP 聊天服务端
2.2、TCP 聊天客户端

第十三章 ODBC 数据库编程
1、 SQL 语言及 ODBC 简介
2、 数据库连接与断开
3、 数据库管理
3.1、执行 sql 语句

3.2、获取结果集中的数据

第一章 AutoIt 基础

1、关于 AutoIt

AutoIt v3 官方主页
AutoIt v3 中文论坛

AutoIt v3 是用以编写并生成具有 BASIC 语言风格的脚本程序的免费软件，它被设计用来在 Windows GUI(用户界面)中进行自动操作. 通过它可以组合使用模拟键击,鼠标移动和窗口/控件操作等来实现自动化任务, 而这是其它语言所无法做到或尚无可靠方法实现的 (比如 VBScript 和 SendKeys). AutoIt 非常小巧,完全运行在所有 windows 操作系统上. (thesnow 注:现在已经不再支持 win 9x,微软连 XP 都能放弃,何况一个 win 9x 支持), 并且不需要任何*运行库*.

AutoIt 最初是为 PC(个人电脑)的"批量处理"而设计, 用于对数千台 PC 进行(同样的)配置. 现在,autoit 是一个支持复杂表达式, 自定义函数,循环等的强大脚本软件.

AutoIt 可以做的事:

- 简单易懂的类 BASIC 表达式
- 模拟键盘,鼠标动作事件
- 操作窗口与进程
- 直接与窗口的"标准控件"交互(设置/获取 文字,移动,关闭,等等)
- 脚本可以编译为标准可执行文件
- 创建用户图形界面接口(GUI)
- COM 支持
- 正则表达式
- 直接调用外部 DLL 和 Windows API 函数
- 程序运行为功能(让程序运行于其它账户)
- 详细易懂的帮助文件于基于社区的支持论坛
- 完全兼容于 Windows 2000 / XP / 2003 / Vista / 2008
- Unicode 与 64 位 运算支持
- 高精度,易使用的数学运算
- 可以运行于 Windows Vista Account Control (UAC)

AutoIt 被设计得尽可能小, 并且不用依赖外部 DLL 文件或添加注册表项目即可独立运行. 也可以安全的成为服务运行. 脚本可以使用 **Aut2Exe** 编译为可独立运行的文件

此外我们还设计了 AutoIt 的 ActiveX 和 DLL 版本 —— **AutoItX** 这是个组件化的语言(COM 同一 DLL 文件中的标准 DLL 函数). AutoItX 将使得您可以加入一些 AutoIt 独有的特性到您最常用的脚本语言或程序设计语言中去!

PS：本教程将以汉化版 AutoIt 3.3.6.1 为基础（大家可以到 AutoIt v3 中文论坛下载并安装。）

2、变量、常量和数据结构

AutoIt 中只存在一种数据类型,那就是 Variant，Variant 变量存储任何数据类型，对它执行各种操作和类型转换。需要注意的是，使用这种弱类型的变量会造成不好的编程习惯。

Variant 变量的类型检查和计算在运行期间才进行，编译器不会提示代码中的潜在错误，这些错误在进一步测试中才能发现。与其它的解释性代码一样，AU3 脚本中的许多操作需要直到执行时才能知道，这就是影响脚本代码效率的一大原因。

所谓变量，顾名思义就是一个可以变动的数据。每个变量都有自己的名字，而且必须以英文字符"**$**"开头，其中只能包含 **字母**, **数字** 和下划线_字符。下面是一些有效的变量名：$var1、$my_variable。

AutoIt 中使用关键字 Dim，Local 和 Global 来声明并创建变量：Dim $var1。也可以一次声明多个变量：Dim $var1,$my_variable。声明变量的同时也可以赋值：Dim $var1=1,$my_variable="变量 2"。

Dim，Local，Global 这三者的不同之处在于其声明变量的作用域：Dim = 如果同名的全局变量并不存在则作用域为局部(如果已有同名的全局变量存在则将复用该变量!)。Global = 将创建的变量的作用域强制转换为全局的。Local = 将创建的变量的作用域强制转换为 局部/函数 的。

所谓常量，就是一个不可更改值的数据。例如圆周率 π=3.1415926，这就是一个常量，一旦更改了它的值，那它就不是圆周率了。

常量声明使用 Const 关键字，就像：Const $const1 = 1, $const2=12

声明的常量可以用 Enum 关键字进行初始化,就像：

```
Enum $const1 = 1, $const2, $const3
Enum STEP 2 $incr0, $incr2, $incr4
Enum STEP *2 $mult1, $mult2, $mult4
```

注意：常量不能声明为一个已经存在的变量。

如果把变量比作为只有一个口袋的钱包，那数据结构则可以看成是有很多个口袋的钱包。

一个数据结构中有多个字段，每个字段中储存一个不同类型的变量值。例如 API 函数 RegisterClass 中要用到一个 WNDCLASS 的结构，这个结构按照 C 语言的格式定义如下：

```
typedef struct {
    UINT style;
    WNDPROC lpfnWndProc;
```

```
    int cbClsExtra;
    int cbWndExtra;
    HINSTANCE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
} WNDCLASS, *pWNDCLASS;
```

一个定义好的数据结构是没有储存数据的，它就像是一个制作好但是还未使用的钱包，里面虽然有很多可以放东西的口袋，但却全是空的。

至于数据结构的具体用法，后面的教程中我会参杂在其它的例子中一并介绍。

3、运算符、宏

AutoIt 支持以下这些赋值符号，数学运算符，比较和逻辑运算符。

运算符	详细信息
	赋值运算
=	赋值,如 **$var = 5** (*赋值数字 5 到 $var*)
+=	自增赋值,如 **$var += 1** (*添加 1 到 $var*)
-=	自减赋值.
*=	自乘赋值.
/=	自除赋值.
&=	连续赋值. 如 **$var = "one"**, 然后 **$var &= 10** (*$var 的结果为 "one10"*)
	数学运算
+	使两个数相加. 如 **10 + 20** (*等于 30*)
-	使两个数相减.如 **20 - 10** (*等于 10*)
*	使两个数相乘.如 **20 * 10** (*等于 200*)
/	使两个数相除.如 **20 / 10** (*等于 2*)
&	使两个字符串连接起来.比如 **"one" & 10** (*等于"one10"*)
^	提高某个数的幂.比如 **2 ^ 4** (*2 的 4 次方,等于 16*)
	比较运算 (大小写敏感的字符串需要使用 == 来比较)
=	判断两个值是否相等. 比如 **If $var= 5 Then** (*如果变量 $var 的值为 5 则条件成立*). 用于字符串时**不区分大小写**
==	判断两个**字符串**是否相等.左方和右方的值将会转化成字符串,并**区分大小写**,这个运

	算只能用于区分字符串大小写的比较.
<>	判断两个值是否不相等. 比较会对字符串**大小写敏感**. 要比较一个大小写敏感的不等于操作使用 **Not ("string1" == "string2")**
>	判断第一个值(左边)是否大于第二个值(右边).Strings are compared lexicographically even if the contents of the string happen to be numeric.
>=	判断第一个值(左边)是否大于或等于第二个值(右边).Strings are compared lexicographically even if the contents of the string happen to be numeric.
<	判断第一个值(左边)是否小于第二个值(右边). Strings are compared lexicographically even if the contents of the string happen to be numeric.
<=	判断第一个值(左边)是否小于或等于第二个值(右边). Strings are compared lexicographically even if the contents of the string happen to be numeric.
	逻辑运算
AND	逻辑与运算. 如 **If $var = 5 AND $var2 > 6 Then** (如果变量 *$var 的值为 5 而且变量 $var2 的值大于 6 则条件成立*)
OR	逻辑或运算. 如 **If $var = 5 OR $var2 > 6 Then** (如果变量 *$var 的值为 5 或者变量 $var2 的值大于 6 则条件成立*)
NOT	逻辑非运算. 如 **NOT 1** (*结果为 False*)

当一个表达式内含有多个运算符时, 其结合的先后顺序由**运算符的优先级别**来控制。AutoIt 中运算符的优先级如下所示：（处于同一优先级的两种运算符将按 *从左到右*的顺序结合，越上面的运算符则优先级越高）

NOT
^
* /
+ -
&
< > <= >= = <> ==
AND OR

AutoIt 提供了一组宏，它们具备了常量属性，可以在代码中把它们当成字符串引用，但是不可对它们进行赋值。

下面列出的是 AutoIt 所有的宏：

宏	详细信息
@AppDataCommonDir	公共 Application Data 文件夹所在路径
@AppDataDir	当前用户 Application Data 文件夹所在路径
@AutoItExe	当前脚本的完整路径. 已经编译的文件返回 EXE 文件所在完整路径.
@AutoItPID	当前运行脚本的进程 PID.

@AutoItVersion	AutoIt 版本号,如 3.2.3.12
@AutoItX64	Returns 1 if the script is running under the native x64 version of AutoIt.
@COM_EventObj	Object the COM event is being fired on. Only valid in a COM event Function.
@CommonFilesDir	Common Files 文件夹路径
@Compiled	脚本已经编译,返回 1.未编译,返回 0.
@ComputerName	当前计算机的名称.
@ComSpec	%comspec%的值, 指定的第二个命令解释程序; 主要用于命令行使用, 如. *Run(@ComSpec & " /k help | more")*
@CPUArch	Returns "X86" when the CPU is a 32-bit CPU and "X64" when the CPU is 64-bit.
@CR	回车符, Chr(13); 用于换行.
@CRLF	@CR 和 @LF ;用于换行.
@DesktopCommonDir	公共 Desktop 文件夹路径(桌面)
@DesktopDir	当前用户 Desktop 文件夹路径(桌面)
@DesktopHeight	桌面高度(像素) (垂直分辨率)
@DesktopWidth	桌面宽度(像素) (水平分辨率)
@DesktopDepth	像素颜色位深度(如 32 Bit).
@DesktopRefresh	屏幕刷新率.(如 75 HZ)
@DocumentsCommonDir	公共 Documents 文件夹路径(我的文档)
@error	错误标识. 参见 SetError 函数.
@exitCode	退出代码
@exitMethod	退出方法. 参见 OnAutoItExitRegister() 函数.
@extended	扩展的函数返回值,使用于一些特定函数.如: StringReplace.
@FavoritesCommonDir	公共 Favorites 文件夹路径
@FavoritesDir	当前用户的 Favorites 文件夹路径
@GUI_CtrlId	最后点击的控件标识(Control ID). 只是使用 event 函数时有效. 请参考 GUICtrlSetOnEvent 函数.
@GUI_CtrlHandle	最后点击的控件句柄(Control handle). 只是使用 event 函数时有效. 请参考 GUICtrlSetOnEvent 函数.
@GUI_DragId	拖动控件标识(Control ID). 只是使用 event 函数时有效. 请参考 GUICtrlSetOnEvent 函数.

@GUI_DragFile	拖动文件(到控件)的文件名. 只是使用 event 函数时有效. 请参考 GUICtrlSetOnEvent 函数.
@GUI_DropId	(拖动后)放下控件标识(Control ID). 只是使用 event 函数时有效. 请参考 GUICtrlSetOnEvent 函数.
@GUI_WinHandle	最后点击的 GUI 窗口句柄(GUI window handle). 只是使用 event 函数时有效. 请参考 GUICtrlSetOnEvent 函数.
@HomeDrive	当前用户主目录所在的驱动器号.(主要用于确定系统所在分区)
@HomePath	当前用户主目录所在位置.(不包含盘符),如须得到完整路径,请使用 @HomeDrive , @HomePath.
@HomeShare	服务器和共享名称,包含当前用户主目录.
@HOUR	当前时钟的时值(24 时制),值的范围是 00 ~ 23
@HotKeyPressed	最后按下的热键. 参考 HotKeySet 函数.
@IPAddress1	第一个网络适配器的 IP 地址.在某些电脑上可能会返回 127.0.0.1
@IPAddress2	第二个网络适配器的 IP 地址.若不存在则返回 0.0.0.0
@IPAddress3	第三个网络适配器的 IP 地址.若不存在则返回 0.0.0.0
@IPAddress4	第四个网络适配器的 IP 地址.若不存在则返回 0.0.0.0
@KBLayout	返回当前键盘布局的 代号。
@LF	换行, Chr(10); 代表用户行中断,进入下一行.
@LogonDNSDomain	登录 DNS 域.
@LogonDomain	登录域.
@LogonServer	登录服务器.
@MDAY	当前是一月中的第几天. (01 到 31)
@MIN	当前的分钟数(00 到 59)
@MON	当前月份(01 到 12)
@MSEC	当前时钟毫秒值.范围为(00 到 999)
@MUILang	Returns code denoting Multi Language if available (Vista is OK by default).
@MyDocumentsDir	我的文档的路径.
@NumParams	调用用户函数的参数数量.
@OSArch	Returns one of the following: "X86", "IA64", "X64" - this is the architecture type of the currently running operating system.
@OSBuild	返回操作系统的内部标号(build 号),如:Windows 2003 Server

	返回的是 3790
@OSLang	返回表示操作系统语言的代号。
@OSServicePack	系统已安装的 Service pack 信息,比如"Service Pack 3"
@OSType	Returns "WIN32_NT" for NT/2000/XP/2003/Vista/2008/Win7/2008R2.
@OSVersion	Returns one of the following: "WIN_2008R2", "WIN_7", "WIN_2008", "WIN_VISTA", "WIN_2003", "WIN_XP", "WIN_XPe", "WIN_2000".
@ProgramFilesDir	返回 Program Files 文件夹路径.
@ProgramsCommonDir	「开始」菜单\程序目录所在路径(例：C:\Documents and Settings\All Users\「开始」菜单\程序)公共用户
@ProgramsDir	「开始」菜单\程序 目录所在路径(例：C:\Documents and Settings\All Users\「开始」菜单\程序) 当前用户
@ScriptDir	脚本所在目录. (不包含反斜杠符号"\")
@ScriptFullPath	等价于 @ScriptDir & "\" & @ScriptName
@ScriptLineNumber	当前执行的脚本行号. 在调试循环语句是非常有用. (已经编译的脚本中没意义)
@ScriptName	当前运行的脚本的长文件名.
@SEC	当前时钟的秒值，值域为 00 ～ 59
@StartMenuCommonDir	公共用户「开始」菜单 目录所在路径(例：C:\Documents and Settings\All Users\「开始」菜单)
@StartMenuDir	当前用户的 「开始」菜单目录所在路径
@StartupCommonDir	公共用户的 启动 目录所在路径(例: C:\Documents and Settings\All Users\「开始」菜单\程序\启动)
@StartupDir	当前用户的 启动 目录所在路径
@SW_DISABLE	屏蔽(禁用)指定窗口
@SW_ENABLE	恢复指定窗口(使其重新可用).
@SW_HIDE	隐藏指定窗口并激活其它窗口.
@SW_LOCK	锁定窗口,避免被重画.
@SW_MAXIMIZE	最大化指定窗口.
@SW_MINIMIZE	最小化指定窗口并激活下一个在 Z 轴(垂直屏幕)方向上的顶层窗口.
@SW_RESTORE	激活并显示指定窗口,如果该窗口已最小化或最大化则以其原始大小和位置还原.一般来说,应用程序在还原一个最小化窗口时应该应用此标志.

@SW_SHOW	激活指定窗口并使其以当前大小和位置信息显示.
@SW_SHOWDEFAULT	设置显示状态(SW_值),程序在启动应用程序时需指定该值.
@SW_SHOWMAXIMIZED	激活并最大化指定窗口.
@SW_SHOWMINIMIZED	激活并最小化指定窗口.
@SW_SHOWMINNOACTIVE	最小化显示指定窗口.与 @SW_SHOWMINIMIZED 不同之处在于该窗口将不被激活.
@SW_SHOWNA	令指定窗口根据其当前大小和位置信息显示.与 @SW_SHOW 不同之处在于该窗口将不被激活.
@SW_SHOWNOACTIVATE	令指定窗口以其上一次的大小和位置显示.与 @SW_SHOWNORMAL 不同之处在于该窗口将不被激活.
@SW_SHOWNORMAL	激活并显示指定窗口，如果该窗口已最小化或最大化则以其原始大小和位置还原.一般来说，应用程序在首次显示窗口时应该应用此标志.
@SW_UNLOCK	取消锁定窗口,允许窗口被重画.
@SystemDir	Windows 下的 **System (或 System32)** 文件夹所在路径(例: C:\WINDOWS\system32)
@TAB	Tab 字符, Chr(9)
@TempDir	临时文件夹路径
@TRAY_ID	最后点击的项目标识 (item identifier), 用于 TraySet(Item)OnEvent 函数.
@TrayIconFlashing	如果托盘图标为闪烁状态,返回 1; 反之,返回 0.
@TrayIconVisible	如果托盘图标为可见状态,返回 1; 反之,返回 0.
@UserProfileDir	返回当前用户的 Profile 文件夹路径.
@UserName	当前登录的用户的名称.
@WDAY	指示当天属该周的第几天,值域为 1 ～ 7,依次表示星期天到星期六.
@WindowsDir	**Windows** 文件夹 所在路径,(例: C:\WINDOWS)
@WorkingDir	当前/激活的工作目录(不包括结尾的反斜杠符号)
@YDAY	指示当天属该年的第几天,值域为 001 ~ 366(若不是闰年则为 001 ~ 365)
@YEAR	当前年份(4 位数)

4、流程控制

流程控制是程序中很重要的部分。AutoIt 作为一种顺序执行的脚本，如果没

有了流程控制，那结果将让我不敢想象。

流程控制主要用于对某项条件进行判断，然后选择执行代码。详细说明请看下文分析。

4.1、选择语句

当脚本代码执行到某一行，出现了分歧条件时，就需要使用选择语句来判断程序接下来该如何执行了。

这就像是你开车走到了十字路口，于是接下来的路程你将面临多种选择，比如说回家直走、上班左转、买菜右转，此时你就需要根据自身接下来要做的事情判断下面的路要怎么走了。

选择语句的作用也正是如此，根据条件判断并选择要执行的动作。

AutoIt 中的选择语句有三种，分别为：

- 单一条件选择语句：If...Then

示例代码：（定义变量$a 的值为 1，然后使用单一条件选择语句判断，如果变量$a 的值为 1 就退出脚本。）

```
Dim $a = 1
If $a = 1 Then Exit
```

- 双条件选择语句：If...Else...EndIf

示例代码：（定义变量$a 的值为 2，然后使用双条件选择语句判断，如果变量$a 的值为 1 就退出脚本，否则就弹出消息框提示变量$a 的值。）

```
Dim $a = 2
If $a = 1 Then
    Exit
Else
    MsgBox(0, "变量$a 的值", "变量$a = " & $a)
EndIf
```

- 条件选择语句：If...ElseIf...Else...EndIf

示例代码：（定义变量$a 的值为 2，然后使用条件选择语句判断，如果变量$a 的值为 1 就退出脚本，否则当$a 为 0 的时弹出一个空消息框，否则就弹出消息框提示变量$a 的值。）

```
Dim $a = 2
If $a = 1 Then
    Exit
ElseIf $a = 0 Then
    MsgBox(0, "", "")
Else
    MsgBox(0, "变量$a 的值", "变量$a = " & $a)
EndIf
```

注意：条件选择语句中的 ElseIf 可以接无数条。

4.2、分支语句

当脚本中出现的条件多达十几二十条，甚至更多时，如果继续使用条件选择语句 If...ElseIf...Else...EndIf 来判断脚本接下来的流程，显然需要书写大段代码，而且也不便于理解与今后的代码维护。

此时我们就可以选择使用分支语句来判断脚本接下来的流程了。AutoIt 中的分支语句有两种，具体区别大家看示例：

第一种：Select...Case...EndSelect

示例代码：

```
Dim $a = 2, $b = 1
Select
    Case $a = 1
       MsgBox(0, "", "变量$a = 1")
    Case $b = 2
       MsgBox(0, "", "变量$b = 2")
    Case Else
       MsgBox(0, "", "变量$a 既不等于 1 也不等于 2")
EndSelect
```

第二种：Switch...Case...EndSwitch

示例代码：

```
Dim $a = 2
Switch $a
    Case 1
       MsgBox(0, "", "变量$a = 1")
    Case 2
       MsgBox(0, "", "变量$a = 2")
    Case Else
       MsgBox(0, "", "变量$a 既不等于 1 也不等于 2")
EndSwitch
```

从上面的两个示例可以看出，Select 可以同时判断多个条件，而 Switch 则只能一次判断一个条件。使用效果基本是相同的，大家可以在编写代码时自行选择。

4.3、循环语句

循环就是重复的执行一部分代码。根据重复执行的次数，可以分为有限循环和无限循环。

其中有限循环为：For...To...Step...Next

示例代码 1：（Step 代表步进值，默认为 1）

```
Dim $i
For $i = 0 To 5 Step 1
    MsgBox(0, "变量$i 的值", "变量$i = " & $i)
Next
MsgBox(0, "$i 最后的值", "$i = " & $i)
```

代码说明：变量$i 从 0 开始步进，每次步进的值为 1，步进到 5 时退出循环，并提示$i 的值。

示例代码 2：

```
Dim $i
For $i = 5 To 0 Step -1
    MsgBox(0, "变量$i 的值", "变量$i = " & $i)
Next
MsgBox(0, "$i 最后的值", "$i = " & $i)
```

代码说明：变量$i 从 5 开始步进，每次步进的值为-1，步进到 0 时退出循环，并提示$i 的值。

当不知道需要执行多少次循环时，我们就可以选择使用无限循环。无限循环有两种。

第一种：While...Wend

示例代码：

```
Dim $i = 0
While $i < 100
    $i += 1
Wend
MsgBox(0, "$i 最后的值", "$i = " & $i)
```

代码说明：变量$i 初始定义值为 0，当它的值小于 100 时就一直执行循环。循环中的代码只有一行$i += 1，意思为变量$i 自加 1。当变量$i 自加到大于等于 100 时退出循环，并提示$i 的值。

从上面的例子可以看出 While 循环在条件成立时，将会一直执行循环体内的代码。也就是说 While 循环是先判断循环条件，再执行循环代码。

第二种无限循环结构 Do...Until 则刚好于此相反，它是先执行一次循环代码，然后再判断循环条件。示例代码：

```
Dim $i = 0
Do
    $i += 1
Until $i < 100
MsgBox(0, "$i 最后的值", "$i = " & $i)
```

代码说明：变量$i 初始定义值为 0，当它的值小于 100 时就退出循环。因为变量$i 自加了 1 之后才判断循环的条件，所以当退出循环时，提示$i 的值应该是 1。

5、函数

AutoIt 内置了超过 200 个函数，而且还有大量的 UDF 函数（UDF = User Defined Function，即用户自定义函数），除了这些函数之外，我们还可以把脚本中需要在多处重复使用的代码写成自定义函数。

5.1、自定义函数

使用关键字：Func...Return...EndFunc，可以声明一个用户自定义函数。

示例代码：（写一个求矩形周长的函数，函数的两个参数分别为长和宽。）

```
MsgBox(0, "长 3 宽 2 的矩形周长", _Zhouchang(3, 2))
Func _Zhouchang($Chang, $Kuan)
    Dim $Zhouchang ;定义一个局部变量，用来表示周长
    $Zhouchang = ($Chang + $Kuan) * 2
    Return $Zhouchang ;关键字 Return 返回周长
EndFunc
```

PS：分号 ; 后面的文字表示单行注释

5.2、函数的参数传递

AutoIt 中函数的参数传递有两种方式，一种是传值，一种是传址。

所谓传值，顾名思义就是传递数值。像上面举例写的那个求矩形周长的函数，就是使用的传值方式。这种方式比较常用，理解也比较简单，所以就不多做解释。

我们来详细说明一下传址。从字面上可以看出，传址就是传递地址，也就是把变量在内存中的地址传递到函数中，然后函数直接读取此地址上的数据，或者是改写此地址上的数据。我们依旧使用求周长来举例说明：

```
Dim $Zhouchang ;周长变量我们在函数外部定义
_Zhouchang(3, 2, $Zhouchang)
MsgBox(0, "长 3 宽 2 的矩形周长",$Zhouchang)
;第三个参数前我们加上关键字 byref，这个关键字的作用就是表示后面的变量是传址
Func _Zhouchang($Chang, $Kuan, byref $Zhouchang)
    $Zhouchang = ($Chang + $Kuan) * 2
EndFunc ;函数没有返回值，因为周长直接写到了变量$Zhouchang 里
```

5.3、函数的变量作用域

函数中使用到的变量，作用范围可分为两种。一种是局部变量，一种是全局变量。

所谓局部变量，指的是只在函数内部起作用，不会影响到函数外部，即使在函数外部存在另一个同名变量。

全局变量则不同，它的作用范围是整个程序。也就是说，不管变量在哪里被修改了值，它影响的都是整个程序，而不单单局限于函数内部。

我们举个例子来说明一下：

```
Global $QuanJu = 100 ;定义一个全局变量$QuanJu 等于 100

MsgBox(0,"","函数_A 的局部变量值为:" & _A() & @LF & "函数_B 的局部变量值为:" & _B()
& @LF & "全局变量$QuanJu 的值为:" & $QuanJu)
;使用&符号串联字符串，宏@LF 表示换行

Func _A()
    Dim $JuBu
    $JuBu += 10;给局部变量赋值为自加 10
    Return $JuBu;返回局部变量的值
EndFunc

Func _B()
    Dim $JuBu
    $JuBu = 2 ^ 4;给局部变量赋值为 2 的四次方
    $QuanJu -= $JuBu ;全局变量自减函数_B 中的局部变量
    Return $JuBu;返回局部变量的值
EndFunc
```

上面的实例中，两个自定义函数_A 和_B 里面都有一个名为$JuBu 的局部变量，但是两个函数分别对它的赋值是不同的，从弹出的消息框可以看到结果，虽然两个局部变量名称一样，但是值是不一样的。

而全局变量$QuanJu 在函数_B 中被改变了值，从消息框可以看出，它最后的值就是被改变后的值。

因此，我们可以看出，局部变量的作用范围就是在函数内部，而全局变量则作用于整个程序。

5.4、函数的嵌套与递归

函数的嵌套，从字面上就可以理解，就是在一个函数中调用另一个函数。在上面说明变量作用范围的例子中，我就使用了函数的嵌套。

在内置函数 MsgBox 中，分别使用了自定义函数_A 和_B。所以这里我就不再举例说明了，我们着重看一下函数的递归。

什么是函数的递归？其实递归也是在函数中调用函数，不过与嵌套的区别在于，嵌套是调用其它函数，而递归则是调用自身。

也就是说，AutoIt 的自定义函数可以自己调用自己！需要注意的是，AutoIt 最大支持的递归深度为 5100 级，超过将会报错。也就是说，函数调用自身的次数应该在 5100 次以内。

我们来看一个函数递归的示例：

```
MsgBox(0, "递归阶乘", _Dg(5))
Func _Dg($n)
    If $n = 1 Then
        Return 1
    Else
        Return $n * _Dg($n - 1)
    EndIf
EndFunc
```

上面是一个递归求阶乘的例子，如果看的有点迷糊，那就暂时丢下好了，等以后熟练了再来看。

第二章 窗口

作为一个多任务操作系统，Windows 可以同时运行多个程序。有些程序在后台运行，不需要像使用者显示信息，例如系统服务程序；有些程序则需要接受用户的输入信息，并处理后输出，例如计算器。

为了接收用户输入的信息，以及向用户输出处理的结果，程序就必须建立一个窗口来完成此项任务。

窗口有一个重要的属性，就是句柄。为了区别不同的窗口，从而达到结果输出不会混淆的目的，Windows 系统给每个窗口都分配了一个唯一的句柄。通过这个句柄我们可以操作自己建立的窗口，也可以操作其它程序建立的窗口，也就是 AutoIt 擅长的自动化操作（这里不讲自动化操作）。

这一章我们来学习如何使用 AutoIt 建立窗口。

1、第一个窗口程序

创建一个空白的窗口：（把下面的代码复制到 SCITE 编辑器中，然后保存文件。遇到不懂的函数可以把光标移动函数中间，然后按 F1 获取帮助。）

```
Global Const $GUI_EVENT_CLOSE = -3;窗口关闭消息的值
GUICreate("我的第一个窗口") ; 创建一个居中显示的 GUI 窗口
GUISetState(@SW_SHOW) ; 显示一个空白的窗口

While 1
    $msg = GUIGetMsg();捕获窗口消息
    If $msg = $GUI_EVENT_CLOSE Then ExitLoop;使用关键字 ExitLoop 跳出 While 循环
WEnd
GUIDelete();删除窗口界面
```

注意：ExitLoop 关键字是用来跳出循环的。While、Do、For 循环都可以使用它来跳出。当多层循环嵌套时，它除了能跳出最近一层的循环外，还可以跳出外层的任意一层循环，后面接一个数字，表示第几层循环（后面没跟数字时，表示使用默认值 1，跳出最靠近它的循环）。

一个完整的窗口，还应该包括一些控件，比如输入框、按钮之类的。与窗口一样，这些控件也有句柄，我们称为控件句柄，对控件的操作则通过这些句柄来进行。

因为窗口程序时通过事件驱动，例如按下最小化按钮、关闭按钮等。用户对窗口进行的任何一个操作，都会产生一个消息。Windows 系统根据不同的消息，来响应不同的操作。

程序运行期间会不断的产生消息，为了不错过这些消息我们有两种方式来处理。一种是不断的循环消息，已达到不错过的目的；还有一种是使用 Event 模式，当产生事件时就进行响应。

1.1、窗口消息

首先我们来看一下消息循环模式：

```
Global Const $GUI_EVENT_CLOSE = -3;窗口关闭消息的值

GUICreate("我的第一个窗口") ; 创建一个居中显示的 GUI 窗口
$Input = GUICtrlCreateInput("1111", 10, 35, 300, 20)
$btn = GUICtrlCreateButton("读取输入框", 40, 75, 90, 20)
GUISetState(@SW_SHOW) ; 显示一个空白的窗口

While 1 ;死循环，直到捕获窗口的 退出消息才跳出
    $msg = GUIGetMsg();捕获窗口消息
     Select;使用分支判断窗口消息
        Case $msg = $GUI_EVENT_CLOSE;退出消息
            ExitLoop
         Case $msg = $btn;按钮被点击
             $D = GUICtrlRead($Input);读取输入框数据
             MsgBox(0, "输入框的数据", $D)
    EndSelect
WEnd

GUIDelete();删除窗口界面
```

从上例可以看出，消息循环模式时，脚本处于一个死循环中，并且在循环里不断的捕获窗口上产生的消息，然后再根据消息来进行操作。

这种模式是一直在被动的不断获取窗口的消息，在系统资源的耗费上相当的不划算。而且因为脚本的主循环需要不断的处理消息，使得我们想要在主循环中执行其它代码很不方便。因为这样很容易造成消息无法得到及时的响应。

接着来看一下 Event 模式：

```
Global Const $GUI_EVENT_CLOSE = -3;窗口关闭消息的值

Opt("GUIOnEventMode", 1) ;开启 Event 模式
GUICreate("我的第一个窗口") ; 创建一个居中显示的 GUI 窗口
GUISetOnEvent($GUI_EVENT_CLOSE, "main");注册关闭消息到自定义函数 main 里面进行处理
$Input = GUICtrlCreateInput("1111", 10, 35, 300, 20)
$btn = GUICtrlCreateButton("读取输入框", 40, 75, 90, 20)
GUICtrlSetOnEvent($btn, "main");注册按钮点击消息到自定义函数 main 里面进行处理
GUISetState(@SW_SHOW) ; 显示一个空白的窗口

While 1 ;死循环，保证脚本不会退出
    GUISetBkColor(RandomColor());修改窗口背景颜色
    Sleep(3000) ;睡眠 3 秒
WEnd

Func main()
    Switch @GUI_CtrlId;根据宏@GUI_CtrlId 来判断消息
        Case $GUI_EVENT_CLOSE
            Exit
        Case $btn
            $D = GUICtrlRead($Input);读取输入框数据
            MsgBox(0, "输入框的数据", $D)
    EndSwitch
EndFunc

Func RandomColor()
    Return "0x" & Hex(Random(0, 255, 1), 2) & Hex(Random(0, 255, 1), 2) &
Hex(Random(0, 255, 1), 2)
EndFunc   ;产生一个随机的 RGB 颜色值
```

从上例可以看出，Event 模式时，当窗口产生消息时，才会对消息进行响应，而不会一直处于消息检测状态。这样一来，相较于消息循环模式将会节省大量的 CPU。

而且在主循环中，我们还可以做其它的事情，而不用担心会造成消息的延迟。（如果在消息循环模式下，主循环也睡眠 3 秒的话，那窗口的消息将会被延迟甚至不响应。）

所以我推荐大家在处理窗口消息时，要尽量使用 Event 模式。

1.2、消息拦截

AutoIt 提供了一个内置函数 GUIRegisterMsg，可以让我们为已知 Windows 消息代码(WM_MSG)注册一个自定义的函数来进行操作。

具体的使用我们来看实例：

```
Global Const $GUI_EVENT_CLOSE = -3;窗口关闭消息的值
Global Const $WM_ENTERSIZEMOVE = 0x0231;窗口移动消息的值
Global Const $WM_EXITSIZEMOVE = 0x0232;窗口结束移动消息的值
Opt("GUIOnEventMode", 1) ;开启 Event 模式
$Gui = GUICreate("我的第一个窗口") ; 创建一个居中显示的 GUI 窗口
GUISetOnEvent($GUI_EVENT_CLOSE, "main");注册关闭消息到自定义函数 main 里面进行处理
$Input = GUICtrlCreateInput("1111", 10, 35, 300, 20)
$btn = GUICtrlCreateButton("读取输入框", 40, 75, 90, 20)
GUICtrlSetOnEvent($btn, "main");注册按钮点击消息到自定义函数 main 里面进行处理
GUISetState(@SW_SHOW) ; 显示一个空白的窗口
GUIRegisterMsg($WM_ENTERSIZEMOVE, "WM_ENTERSIZEMOVE");产生窗口移动消息时，执行
自定义函数 WM_ENTERSIZEMOVE
GUIRegisterMsg($WM_EXITSIZEMOVE, "WM_EXITSIZEMOVE");窗口移动结束时，执行自定义函
数 WM_EXITSIZEMOVE

While 1 ;死循环，保证脚本不会退出
    GUISetBkColor(RandomColor());修改窗口背景颜色
    Sleep(3000) ;睡眠 3 秒
WEnd

Func main()
    Switch @GUI_CtrlId;根据宏@GUI_CtrlId 来判断消息
        Case $GUI_EVENT_CLOSE
            Exit
        Case $btn
            $D = GUICtrlRead($Input);读取输入框数据
            MsgBox(0, "输入框的数据", $D)
    EndSwitch
EndFunc

Func RandomColor()
    Return "0x" & Hex(Random(0, 255, 1), 2) & Hex(Random(0, 255, 1), 2) &
Hex(Random(0, 255, 1), 2)
EndFunc   ;产生一个随机的 RGB 颜色值

Func WM_ENTERSIZEMOVE($hWndGUI, $MsgID, $WParam, $LParam)
    WinSetTrans($Gui, "", 130)
EndFunc   ;窗口移动时，设置窗口透明值为 130

Func WM_EXITSIZEMOVE($hWndGUI, $MsgID, $WParam, $LParam)
    WinSetTrans($Gui, "", 255)
EndFunc   ; 窗口结束移动时，设置窗口透明值为 255，也就是不透明
```

上面的代码运行后，当窗口移动时，你会发现窗口变成半透明的了，这是因

为脚本把窗口移动的消息拦截下来了，在发生移动的时候，脚本把窗口设置成了半透明。除此以外，我们还可以拦截其它的消息，比如列表控件的双击、单击消息，组合列表框控件的点击、选择消息等。

对于自己编写的程序的窗口，我们可以使用此种方法来拦截消息。但是对于外部程序的窗口则不能这样做，若非要拦截外部程序窗口的消息，唯一的方法就是注入到目标程序的进程中去。此处不讨论这些，略过。

2、多窗口程序

在实际的应用中，程序可能需要用到多窗口来实现，典型的例子就是 QQ，除了主界面之外，还有聊天窗口、资料设置窗口、好友查找窗口等。

窗口与窗口之间的关系除了平等之外，还有父子窗口。也就是以其中一个窗口为主窗口，其余窗口均为主窗口的下属窗口。平等关系的我们就不说了，因为跟父子窗口差不多，所以我们就说一下父子窗口好了。

2.1、父窗口与子窗口

AutoIt 内置的窗口创建函数 GUICreate 有八个参数，通过查看帮助，我们可以看到，最后一个参数可以为新建的窗口指定父窗口的句柄，这样一来新窗口就将成为一个子窗口。

如果窗口需要指定样式，比如去掉关闭按钮、去掉最小化按钮等等，也可以在使用 GUICreate 创建时一并设置。GUICreate 函数的第六和第七两个参数就是设置窗口样式的，至于具体的样式我们可以查看帮助。如果要使用默认样式可以设置为-1。

GUICreate 函数创建窗口成功后将会返回一个真正的句柄。为什么要这么说呢？这是因为 AutoIt 在创建控件时，返回的并不是真正的控件句柄，而是控件标识，所谓标识就是一个整数。

也就是说，如果你在窗口上创建了十个按钮，那么这十个按钮返回的并非是按钮控件的句柄，而是一到十的十个整数。（这里只是举例，实际应用中可能并非刚好是一到十。）

要取得控件的真正句柄，我们需要用到 GUICtrlGetHandle 这个函数，此函数可以根据控件标识返回控件句柄。

以上这些在此只是顺带说明，不做具体的实例说明。本小节我们还是以父窗口和子窗口为主题。

关于父窗口与子窗口的应用我们来看一段实例：

```
Global Const $GUI_EVENT_CLOSE = -3;窗口关闭消息的值
Dim $Child_Gui;定义一个变量用于存放子窗口的句柄
Opt("GUIOnEventMode", 1) ;开启 Event 模式
$Gui = GUICreate("我的第一个窗口") ; 创建一个居中显示的 GUI 窗口
GUISetOnEvent($GUI_EVENT_CLOSE, "main");注册关闭消息到自定义函数 main 里面进行处理
$Input = GUICtrlCreateInput("1111", 10, 35, 300, 20)
$btn = GUICtrlCreateButton("读取输入框", 40, 75, 90, 20)
GUICtrlSetOnEvent($btn, "main")
$Show_btn = GUICtrlCreateButton("显示子窗口", 40, 100, 90, 20)
GUICtrlSetOnEvent($Show_btn, "main")
GUISetState(@SW_SHOW) ; 显示一个空白的窗口

While 1 ;死循环，保证脚本不会退出
    GUISetBkColor(RandomColor());修改窗口背景颜色
    Sleep(1000)
WEnd

Func main()
    Switch @GUI_CtrlId;根据宏@GUI_CtrlId 来判断消息
        Case $GUI_EVENT_CLOSE
            Switch @GUI_WinHandle;根据宏@GUI_WinHandle 来判断产生关闭消息的窗口消息
                Case $GUI
                    Exit
                Case $Child_Gui
                    GUIDelete($Child_Gui)
            EndSwitch
        Case $btn
            $D = GUICtrlRead($Input);读取输入框数据
            MsgBox(0, "输入框的数据", $D)
        Case $Show_btn
            Child_Gui ()
    EndSwitch
EndFunc

Func RandomColor()
    Return "0x" & Hex(Random(0, 255, 1), 2) & Hex(Random(0, 255, 1), 2) &
Hex(Random(0, 255, 1), 2)
EndFunc   ;产生一个随机的 RGB 颜色值

Func Child_Gui ()
    $Child_Gui = GUICreate("我是子窗口", 200, 40, -1, -1, -1, -1, $Gui)
    GUISetOnEvent($GUI_EVENT_CLOSE, "main")
    GUISetState(@SW_SHOW)
EndFunc   ;子窗口
```

细心的朋友肯定会发现，在自定义函数中我们嵌套使用了两个分支判断语句 Switch。扩展一下，我们还可以在分支判断语句里面使用循环语句 While，而且循环语句内还可以再使用循环语句，具体的使用，大家可以自己写代码熟悉一下。

在上面的代码中，当子窗口显示后，主窗口的背景色将保持不变，转而是子窗口的背景色被不断修改。造成这个现象的原因在于 GUISetBkColor 函数，查看帮助可以看到这个函数有两个参数，其中第二个参数是需要操作的窗口的句柄，这个参数是可以省略的。

在省略状况下，函数将会使用最近一次使用过的窗口句柄。当子窗口出现时，最近一次使用的窗口句柄变成了子窗口的，所以背景颜色被改变的窗口就从父窗口变成了子窗口。

2.2、GUI 嵌入外部进程窗口

除了上节中说明的父子窗口，还有一种特殊的父子窗口。玩过网游的朋友肯定会发现，游戏界面内部的窗口，无论如何都不能被移除游戏界面之外。就像是一个窗口被镶嵌在另一个窗口内部，无法取出一样。

AutoIt 可以使用 API 来达成这种效果，此处我们只演示一下这种效果，不对 API 的调用做详细的解释，等到后文会有具体的说明章节。

我们来看实例代码：（窗口一个窗口，然后嵌入到记事本中）

```
Run("notepad.exe") ;运行记事本
WinWait("无标题 - 记事本") ;等待记事本窗口出现
$Gui = GuiCreate("被装进了记事本", 240, 120)
GuiSetState()
DllCall("user32.dll", "int", "SetParent", "hwnd", $Gui, "hwnd",WinGetHandle("无标题 - 记事本")) ;使用 API 把脚步建立的窗口嵌入记事本窗口中

Do
;Do 循环，当窗口消息等于退出消息，或者记事本窗口消失时，就退出循环
Until GuiGetMsg() =-3 Or Not WinExists("无标题 - 记事本")
```

上面的代码只是创建了一个空白的窗口嵌入到记事本窗口中，实际的运用还要看具体的目的。比如说在全屏游戏时，嵌入一个窗口来显示当前时间，这样就可以不用切换到桌面来查看时间。

第三章 字符串与变量转换

在程序中，字符串的处理与变量之间的转换占了很重要的一席之地。学习好处理字符串与转换变量，将使你写起程序来更得心应手。

1、字符串处理

与大多数编程语言一样，AutoIt 中定义一个字符串也是使用双引号括起来。但是除此之外，AutoIt 还支持使用单引号来表示字符串变量。

例如，下面两种字符串变量的赋值是等价的：

```
Dim $var = '中国人'
Dim $var = "中国人"
```

虽然上面两种方式是等价的，但是推荐大家在大多数情况下使用双引号。

如果我们需要在字符串中包含引号，要怎么办？

因为 AutoIt 支持单双引号定义字符串，于是当我们需要包含一种引号在字符串内时，我们可以使用另一种引号来定义，看代码：

```
Dim $var1 = '"中国人' ;用单引号包含双引号
Dim $var2 = "'中国人" ;用双引号包含单引号
MsgBox(0, "", $var1 & @LF & $var2) ;宏@LF 用于换行，具体说明请查看宏列表
```

那如果一个字符串中，两种引号都需要出现，怎么办？此时我们可以使用字符串连接符&来实现，具体看代码：

```
Dim $var = '"中国人' & "'中国人"
MsgBox(0, "", $var)
```

1.1、字符串长度

获取字符串长度，AutoIt 有内置的函数可用。StringLen 函数可以获取字符串的长度，但是需要注意的是，这个函数是将一个字符当做一长度，这个字符不管是英文字符还是中文字符都被当成长度为一。而在字符的编码中，中文字符的字节数比英文的要长。这里我们只说明一下，不做深入了解。

下面我们看一下如何获取字符串长度：

```
Dim $var = "中国人 Chinese"
$Len = StringLen ($var)
MsgBox(0, "字符串长度", $Len)
```

运行代码后，我们可以从消息框中看到，字符串的长度为 10，其中包括三个中文字符和七个英文字符。

从上面的代码中大家应了解到 StringLen 函数获取的长度是字符串的字符数，而非字节数。AutoIt 没有一个内置函数 SizeOf 来返回字符串的字节数，所以若要取字符串的字节数则需要我们自己写函数来完成此项任务。

1.2、字符串截取

一个字符串，在实际的应用中，有时我们只需要用到左边几个字符或右边几个字符，当然也可能是中间的一段字符。此时我们就需要截取字符串。

AutoIt 用于这方面的函数有以下几个：

StringLeft（返回字符串中从左开始指定数量的字符）

```
Dim $var ="中国的国庆是十一"
MsgBox(0, "", StringLeft ($var, 3)) ;返回字符串的左边三个字符
```

StringRight（返回字符串中从右开始指定数量的字符）

```
Dim $var = "中国的国庆是十一"
MsgBox(0, "", StringRight ($var, 3)) ;返回字符串的右边三个字符
```

StringMid（取字符串的部分字符）

```
Dim $var = "中国的国庆是十一"
MsgBox(0, "", StringMid ($var, 3, 5)) ;从字符串第三个字符开始取 5 个字符
```

StringTrimLeft（删除字符串中从左开始指定数量的字符）

```
Dim $var = "中国的国庆是十一"
MsgBox(0, "", StringTrimLeft ($var, 3)) ;删除字符串左边的三个字符
```

StringTrimRight（删除字符串中从左右始指定数量的字符）

```
Dim $var = "中国的国庆是十一"
MsgBox(0, "", StringTrimRight($var, 3)) ;删除字符串右边的三个字符
```

1.3、字符串替换

在实际应用过程中，有时候我们会需要把字符串中的部分字符替换成其它的，这个时候我们就需要用到 StringReplace 函数。

看实例说明：（把国庆是十一替换成劳动节是五一）

```
Dim $var = "中国的国庆是十一"
MsgBox(0, "", StringReplace ($var, "国庆是十一", "劳动节是五一"))
```

上面的代码会把字符串中符合要求的字符全部替换，可能上例对这点的体现不明显，我们换个例子来说明：

```
Dim $var = "abc 中国的国庆是十一 abc"
MsgBox(0, "", StringReplace ($var, "abc", "123"));abc 替换为 123
```

如果我们只想替换一次，那要怎么办？通过查看帮助，我们可以看到 StringReplace 函数一共有五个参数，其中后两个参数可以省略。第四个参数可以帮助我们实现目的，指定需要替换的次数，看实例：

```
Dim $var = "abc 中国的国庆是十一 abc"
MsgBox(0, "", StringReplace ($var, "abc", "123", 1))
```

StringReplace 函数的第五个参数是用于区分大小写的，在默认的情况下，字符串的替换时不区分大小写的。具体的应用我就不再举例说明，大家可以自己写代码练习使用这个参数。

1.4、字符串分割

实际的应用中，有时候我们会需要把字符串拆分为几段，然后再进行具体的操作。比如说一段文章，我们希望把它的每一句都独立提取出来，这时候我们就可以以句话来进行拆分。

AutoIt 中用于字符串拆分的函数是 StringSplit，从帮助中我们可以知道，字符串拆分后，将会产生一个数组，用于存放拆分后的字符串。数组的第一个元素存放的是拆分后字符串的数量，从第二个元素开始才是拆分后的字符串。

需要注意的是，不过字符串是否拆分成功，结果都将是一个数组。拆分失败时，数组的第一个元素将是 1，第二个元素里面存放的是整个字符串。

我们来看一个简单的实例：

```
Dim $var = "abc 中国的国庆是十一 abc"
$d = StringSplit($var, "b");用字符 b 来拆分字符串
MsgBox(0, "", "子串数量:" & $d[0] & @LF & "第一个子串:" & $d[1])

$d = StringSplit($var, "e");用字符 e 来拆分字符串，因为不存在 e，所以拆分将失败
MsgBox(0, "", "子串数量:" & $d[0] & @LF & "第一个子串:" & $d[1])
```

有时候在表驱动中，使用拆分的字符串也会有意想不到的效果。我用一段帮助文档中的代码来作为实例，至于具体的分析就不做了，等大家以后再自己琢磨吧：

```
$attrib = FileGetAttrib(@ScriptDir) ;获取脚本目录的文件夹属性
If @error Then
    MsgBox(4096,"错误", "无法获得属性.")
    Exit
EndIf
$input = StringSplit("R,A,S,H,N,D,O,C,T",",")
$output = StringSplit("只读 /, 存档 /, 系统 /, 隐藏 /, 普通 /, 目录 /, 脱机文件 /, 压缩 /, 临时 /", ",")
For $i = 1 to 9
    $attrib = StringReplace($attrib, $input[$i], $output[$i], 0, 1)
Next
$attrib = StringTrimRight($attrib, 2) ;移除末尾的反斜杠
MsgBox(0,"完整的文件属性:", $attrib)
```

1.5、正则

正则表达式通常用来验证字符串是否符合某个语法规则，很多情况下使用正则来判断或截获符合规则的字符串，将会是脚本更加简捷。

正则最初是在 Unix 系统中出现的，后来被很多语言所引用，AutoIt 同样也支持正则。要学好正则，建议大家到网上查找一些针对性的教程，我在这里只演示一下正则的作用。

通常情况下，正则除了用来判断字符串是否符合某个规则之外，还可以用来截获符合某个规则的字符串，另外还有个很重要的就是替换。

首先我们来看一下字符串的验证，正则验证字符串使用函数 StringRegExp：

```
Dim $ip = "127.0.0.1"
StringRegExp($ip,"^((25[0-5]|2[0-4]\d|[01]?\d?\d)\.){3}(25[0-5]|2[0-4]\d|[01]?\
d?\d)$")
If Not @error Then
    MsgBox(0," ", "IP 地址格式正确。")
Else
    MsgBox(0," ", "IP 地址格式错误。")
EndIf

Dim $id = "abc124"
StringRegExp($id,"^[\w]{6,15}+$") ;判断一个字符串是否只使用了数字和字母及下划
线，且长度在 6-15 位。一般网上注册账号就是这样判断账号是否符合要求的
If Not @error Then
    MsgBox(0," ", "账号格式正确。")
Else
    MsgBox(0," ", "账号格式错误。")
EndIf
```

接下来我们看看如何用正则来截获字符串：（从一段字符中提取除 IP 地址）

```
Dim $ip = "abc127.0.0.1aaa"
$a =
StringRegExp($ip,"((25[0-5]|2[0-4]\d|[01]?\d?\d)\.){3}(25[0-5]|2[0-4]\d|[01]?\d?
\d)", 2)
If Not @error Then
    MsgBox(0," ", "字符串中的 IP 地址:" & $a[0])
Else
    MsgBox(0," ", "字符串中没有 IP 地址。")
EndIf
```

除了验证字符串是否符合规则以及从字符串中提取符合规则的字串，我们也可以使用正则来替换字符串中符合规则的字串，要完成这项任务，我们需要用到

函数 StringRegExpReplace，我们来看替换的实例：

```
Dim $ip = "abc127.0.0.1aaa"
$a = StringRegExpReplace ($ip,"\d", "中") ;把字符串中的数字全部替换成中字
If Not @error Then
    MsgBox(0," ", "替换了数字后的字符串:" & $a)
Else
    MsgBox(0," ", "字符串中没有数字可以替换")
EndIf

Dim $ip = "abc127.0.0.1aaa"
$a = StringRegExpReplace ($ip,"\d", "中", 2);把字符串中的前 2 个数字替换成中字
If Not @error Then
    MsgBox(0," ", "替换了前 2 个数字后的字符串:" & $a)
Else
    MsgBox(0," ", "字符串中没有数字可以替换")
EndIf
```

上例中的第二个替换，只替换了字符串中的前两个数字，这是因为使用了函数 StringRegExpReplace 的第四个参数来制定需要替换的子串数量，通常情况下如果省略这个参数，将会替换所有符合要求的字串。

从上面的几个例子可以看出，使用正则来判断字符串，关键在于正则表达式，所以如果对这部分内容感兴趣的朋友，可以在网上找一份专门针对正则的教程来学习。在此我就不再深入，因为我的正则也是半桶水，而且这也不属于 AutoIt 的范畴了。

2、变量转换

因为 AutoIt 的变量是 Variant 类型，这就可能导致脚本执行的过程中会出现一些奇怪的问题。比如一个窗口的句柄我们可以直接当成字符串来使用，但是一个符合窗口句柄格式的字符串，我们如果当成当口句柄来使用的话就不行了。

因此为了保证脚本在执行过程中不会出现上述的错误，我们在某些时刻就必须保证变量类型的正确，此时就需要转换变量的类型了。

在其它编程语言中，因为对变量类型的严格控制，在代码编写过程中也经常需要转换变量类型。比如说一个整型变量如果想在消息框中显示出来的话，像 AutoIt 那样直接使用肯定是不行的，需要转换成字符串类型之后再使用。

因此不管是不是在 AutoIt 中，变量类型的转换都很重要。这一节我们就讲一下 AutoIt 中的变量类型转换。

2.1、转换为指针

指针这种类型的变量，在 AutoIt 中是很少见的，因为 AutoIt 没有指针。但是在一些应用中，我们常常需要用到指针，所以 AutoIt 还是提供了一个内置函数

Ptr 来把变量转换成指针类型。

指针在 AutoIt 中一般是在调用 DLL 时需要用到，而且这个指针大多数都是数据结构的指针，比如我们在调用 API 时就会经常需要创建一个数据结构，然后再把结构中某个元素的指针作为调用 API 的参数使用。

因为实在不知道要举一个什么样的例子来说明把字符串变量转换成指针变量，Ptr 函数的使用非常简单，而且要演示指针一时也想不到具体的例子，所以这个实例暂时就不举了。我在这里就解释一下指针的概念吧。

因为程序中所有的变量都需存放在内存，为了能正确的访问到这些数据，就必须给它们进行编号，这个编号就像是变量在内存里的地址，当需要访问的时候，就可以根据地址找到变量。这个地址，就是我们说的指针。而指针变量，就是用于存放变量地址的变量。

2.2、转换为句柄

句柄是 Windows 给程序建立或访问的对象分配的一个唯一标识，在 Windows 系统中存在很多句柄，比如窗口、文件、位图等等。这一节我们要讲的是窗口句柄。

系统分配给窗口的句柄是一个整数，在 AutoIt 中通常会以一个 16 进制数来表示。要把一个整数或字符串转换成句柄，我们需要用到 HWND 函数，我们来看一个实例：

```
$hWnd = WinGetHandle("[class:Shell_TrayWnd]") ;获取任务栏窗口的句柄
WinSetState($hWnd, "", @SW_HIDE) ;使用任务栏句柄来隐藏任务栏
Sleep(1000) ;休息 1 秒
WinSetState($hWnd, "", @SW_SHOW) ;恢复显示任务栏

MsgBox(0, "", "我们把获取到的句柄转换成字符串，然后再次隐藏任务栏试试。")
$sWnd = String($hWnd) ;把句柄转换成字符串，再次隐藏显示任务栏试试
WinSetState($sWnd, "", @SW_HIDE) ;使用任务栏句柄来隐藏任务栏
Sleep(1000) ;休息 1 秒
WinSetState($sWnd, "", @SW_SHOW) ;恢复显示任务栏

MsgBox(0, "", "上面的实验失败了，句柄不能是字符串，所以我们把字符串转换成句柄再试试。")
$Wnd = HWnd($sWnd) ;把句柄转换成字符串，再次隐藏显示任务栏试试
WinSetState($Wnd, "", @SW_HIDE) ;使用任务栏句柄来隐藏任务栏
Sleep(1000) ;休息 1 秒
WinSetState($Wnd, "", @SW_SHOW) ;恢复显示任务栏
```

2.3、转换为整数

脚本在用户的使用过程中，变量可能会产生一些我们不知道的变化，而有时候这个变量又要确保它是一个整数，这样才能保证脚本接下来的功能不会发生错误。

比如说从窗口接收一个用户输入的数据，然后用于计算，如果用户输入的不是数字，那计算的结果肯定会发生错误，此时我们就需要把用户输入的数据先转换成整数，确保变量类型的准确，然后再计算。

把一个变量转换为整数使用的是 Int 函数，这个应该很好理解，简单说明一下：

```
MsgBox(0, "整数转换示例", "10 转换后的结果：" & Int(10) & _
                          @LF & "w 转换后的结果" & Int("w"))
```

上面的代码运行后，w 转换的结果会是 0。根据帮助文档的说明，我们可以知道，不是数字的字符串进行转换的结果就是 0。

再上面的代码中，我们用了一个以前没见过的符号_，这个下划线在 AutoIt 中的作用是换行。也就是说当一行的代码太长时，为了便于阅读，我们就可以使用下划线把这行代码分成两行甚至多行。

2.4、转换为二进制数据

这里说的二进制数据，其实就是字符串编码，AutoIt 中有一个函数可以很方便的把字符串转换成各种编码值，当然这些编码值也可以转换回字符串。这里需要用到的函数有两个，一个把字符串转换成二进制数据，一个把二进制数据转换成字符串。

这两个函数分别是 StringToBinary、BinaryToString。这两个函数都有两个参数，第一个参数是需要转换的数据，第二个参数是转换标志。

标志	[可选参数] 修改数据编码转换格式: 标志 = 1 (默认), 二进制数据为 ANSI 编码 标志 = 2, 二进制数据为 UTF16 小编码 标志 = 3, 二进制数据为 UTF16 大编码 标志 = 4, 二进制数据为 UTF8 编码

因为帮助中的演示代码很详细，所以这里我只举个简单的例子：

```
$sString = "Hello，我是要转换的字符串！"
$ASCII = StringToBinary($sString)
MsgBox(0, "", "字符串的 ASCII 码：" & $ASCII & _
              @LF & "ASCII 码转回的字符串：" & StringToBinary($ASCII))
```

第四章 数组

数组，顾名思义就是一组相同类型的数。在 AutoIt 中，前面那句话是不成立的，由于 AutoIt 特殊的变量类型，数组并不是只能存放同一类型的数据。

在程序中，数组也是一个变量，只是这个变量比较特别，因为它可以存储很多个数据，这些数据可以是相同类型的，也可以是不同类型的。

数组变量可以只有一个成员，也可以有很多个成员，每个成员我们可以认为是一个单独的变量，在 AutoIt 中它可以存放任何数据。

这些成员都通过数组的下标来访问。在 AutoIt 中，数组的开始下标是 0。也就是说，如果一个数组有 4 个成员，那这四个成员的下标依次是 0、1、2、3，请不要惯性思维的认为还有个下标 4。

1、一维数组

在数组中，最简单的就是一维数组。数组的定义方式与变量相同，不同的在于数组变量的名称格式，看一个定义数组的实例：

```
Dim $array[2]
```

上例定义了一个拥有 2 个成员的数组，因为没给成员赋值，所以这个数组现在还是空的，没存放任何数据。下面我们看看如何给数组赋值，有两种方法，一种是定义数组时就赋值：

```
Dim $array[2] = [1, 2]
```

还有一种是定义数组之后再赋值：

```
Dim $array[2]
$array[0] = 1
$array[1] = 2
```

我们用一个表格来说明下上面的数组：

数组名称	$array[0]	$array[1]
数据	1	2

一维数组是最简单也是最容易被理解的数组，所以关于它的说明就到此为止，下面我们看一个应用一维数组的例子：

```
;定义一个 7 元素数组，用来存放一周中每天的名称
Dim $week[7] = ["周一", "周二", "周三", "周四", "周五", "周六", "周日"]
Dim $day = 2 ;这个变量表示我们要查看的数组元素
If $day > 6 Then $day = 6
If $day < 0 Then $day = 0
;用两个 If 来判断$day 变量的大小，是否超出数组
MsgBox(0, "", $week[$day])
```

2、二维及多维数组

相较于一维数组，之上还有二维、三维等多维数组，AutoIt 最大支持的数组维度有 64 维，这么恐怖的数组，不晓得在哪里会用得上。我们一般使用的大多数是一维、二维数组，其它的几乎没什么几乎用到，毕竟我们使用 AutoIt 可不是要做惊天动地的大事，我们只是想写点小程序帮助自己从一些繁琐的操作中解脱出来而已。

关于数组围数的解释，我在网上看过一个比喻很好，这里引用一下。

————————————————————————————

关于多维数组的解释：

————————————————————————————

假如数组名称为“房间”代表房间位置：

一维：一幢平房共 10 个房间，第 5 个房间＝房间[5]

二维：二幢平房，一幢 10 个房间。
则第一幢、第五个房间＝房间[1][5]

三维：二幢 2 层楼房，一幢 10 个房间。
则第一幢、第二层、第五个房间＝房间[1][2][5]

————————————————————————————

上面的解释，非常清楚的解释了多维数组的概念，如果还有朋友看不懂，那实在抱歉，我能力有限，恐怕无法再帮助你了。

这一节的名称中虽然包含多维数组，但是我只对二维数组举一个应用实例。扩展一下上面的一维数组，把它改成一个存放课程表的数组，具体看代码：（一周上三天课，且每天三节课。）

```
Dim $week[4][3] = [["周一", "周二", "周三"], ["语文", "数学", "数学"], ["英语", "语文", "体育"], ["英语", "音乐", "班会"]]
```

上面的数组用一个表格来表示：

2.1、数组的维数

这节我们说一下如何获取数组的维数，AutoIt 中有一个内置函数 UBound，使用它可以获取数组的维数，除此之外，这个函数还可以获取数组各维上的元素个数。我们来看实例：

```
Dim $week[4][3] = [["周一", "周二", "周三"], ["语文", "数学", "数学"], ["英语", "语文", "体育"], ["英语", "音乐", "班会"]]
$rows = UBound($week)
$cols = UBound($week, 2)
$dims = UBound($week, 0)
MsgBox(0, "当前 " & $dims & "-维数组有", $rows & " 行, " & $cols & " 列")
```

2.2、数组调整

当我们需要定义一个数组来存放数据，但是不知道数据有多少个时，我们可以使用 ReDim 关键字来调整数组。看实例：（仍然使用上面用的的数组，我们给每天加一节课）

```
Dim $week[4][3] = [["周一", "周二", "周三"], ["语文", "数学", "数学"], ["英语", "语文", "体育"], ["英语", "音乐", "班会"]] ;先定义数组
ReDim $week[5][3] ;重定义大小
For $i = 0 To 2
    $week[4][$i] = "美术" ;赋值，多加的那节课全部为美术课
Next
```

我们用一个表格来显示数组调整后的数据：

Row	Col 0	Col 1	Col 2
[0]	周一	周二	周三
[1]	语文	数学	数学
[2]	英语	语文	体育
[3]	英语	音乐	班会
[4]	美术	美术	美术

ReDim 函数还有一个用处就是清理数据，因为 AutoIt 没有数组销毁函数，所以如果要清理数组的话，我们可以使用这个函数，使用此函数把数组调成只有一个元素的一维数组，这样就能达到清理数组数据的目的，但是这样做，是否能释放内存资源就不得而知了。（注意：如果原数组是一维数组，那重定义成只有一个元素的一维数组后，第一个元素数据将会被保留，此时可以给它赋一个空值，以达到清理数据的目的。）

使用 ReDim 函数清理数组的实例我就不写了，大家有兴趣可以自己写来试试。

第五章 注册表读写

注册表，是 Windows 系统的一个专用产物。我们可以把它理解成为一个数据库，专门用来储存系统和应用程序设置信息的数据库。

与 VB 不同的是，AutoIt 内置了操作注册表的函数，而不需使用 API 来实现这些功能。AutoIt 中操作注册表是一件很简单的事情，之所以要独立出来做一章讲解，主要是为了方便大家查阅。

1、读取注册表

读取注册表数据，我们使用的函数是 RegRead，通过查看帮助文档，我们知道这个函数有两个参数，第一个是键名，第二个是值项。这里没什么可说的，实例我直接复制帮助文档的好了：

```
$var =
RegRead("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVer
sion", "ProgramFilesDir")
MsgBox(4096, "Program files are in:", $var)
```

关于注册表的读取，还有两个特别的函数，它们分别是 RegEnumVal、RegEnumKey。

RegEnumVal 用来读取指定键名下所有值项的数据，我们来看实例：（读取开机启动的程序）

```
For $i = 1 to 100
    $var =
RegEnumVal("HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Current
Version\Run", $i)
    if @error <> 0 Then ExitLoop
    MsgBox(4096, "开机启动项 #" & $i, $var)
Next
```

RegEnumKey 用来读取知道键名下所有的子键名称，我们来看实例：

```
For $i= 1 to 10
    $var = RegEnumKey("HKEY_LOCAL_MACHINE\SOFTWARE", $i)
    If @error <> 0 then ExitLoop
    MsgBox(4096, "安装的第 " & $i & " 个程序: ", $var)
Next
```

2、写入注册表

注册表写入函数是 RegWrite，查看帮助可以得知这个函数有四个参数，其中后三个参数可以省略，当省略的时候，就表示只创建第一个参数指定的键名，而

不写入值。如果第一个参数是一个新的键名，且后面三个参数也使用了的话，就表示创建新键的同时写入值。

值的写入要注意一个很重要的地方，那就是注册表的数据类型，注册表的数据类型主要有以下四种：

显示类型	数据类型	说明
REG_SZ	字符串	文本字符串
REG_MULTI_SZ	多字符串	含有多个文本值的字符串
REG_BINARY	二进制数	二进制值，以十六进制显示。
REG_DWORD	双字	一个 32 位的二进制值，显示为 8 位的十六进制值。

如果要更新指定键的数据，也同样是使用 RegWrite 函数，使用方法与写入新的数据是一样的。

我们来看一个实例：

```
; 创建一个新键，并同时写入一个单行字符串数据
RegWrite("HKEY_CURRENT_USER\Software\Test", "TestKey", "REG_SZ",
"Hello this is a test")

; 更新刚才写入的字符串数据
RegWrite("HKEY_CURRENT_USER\Software\Test", "TestKey", "REG_SZ", "
我是新来的")

;写入一个多行字符串数据
RegWrite("HKEY_CURRENT_USER\Software\Test", "TestKey1",
"REG_MULTI_SZ", "Hello1" & @LF & "Hello2")

;写入一个二进制数据
RegWrite("HKEY_CURRENT_USER\Software\Test", "TestKey2",
"REG_BINARY", 0x01)

;只创建新建，不写数据
RegWrite("HKEY_CURRENT_USER\Software\Test1")
```

有写就有删，AutoIt 中删除注册表数据的函数是 RegDelete，查看帮助可以知道这个函数有两个参数，第二个参数值项名可以省略，在省略的情况下，表示删除第一个参数指定的键。看一个实例：

```
; 删除上例建立的 TestKey 值项
RegDelete("HKEY_CURRENT_USER\Software\Test", "TestKey")
; 删除上例建立的两个键
RegDelete ("HKEY_CURRENT_USER\Software\Test")
RegDelete ("HKEY_CURRENT_USER\Software\Test1")
```

第六章 文件读写

Windows 的文件 I/O 操作使用的是文件句柄和读写指针，使用 API 操作的文件不仅有普通文本文档，还有串口、磁盘设备、网络文件、控制台和目录等。相对的，因为功能强大，所以 API 函数的使用也就比较复杂。

AutoIt 中提供了内置函数来进行文件读写操作，但是只能操作普通的文件。虽然功能减弱了，不过使用起来也简单许多。

在讲文件读写操作前，还有一点必须要说明的是，Windows 下的文件都有一个后缀名，用于区分文件类型。这个后缀只是用来给用户区分文件类型的，但并代表使用某种后缀的文件就真的是那种类型的文件。

比如说 Windows 下的可执行 PE 文件后缀名一般都是.EXE 和.COM，但是如果你使用.TXT 做后缀的话，它也还是一个可执行 PE 文件，只是我们双击后不能正常运行而已。要正常运行它，只要通过运行来打开就好了。或者修改注册表，把.TXT 文件当成.EXE 来执行。

1、Ini 配置文件读写

AutoIt 读写 ini 文件可直接使用内置函数 IniWrite、IniRead。两个函数都有四个参数，前三个是一样的，区别在于第四个参数，写入数据时，第四个参数是需要写入的数据；读取数据时，第四个参数是读取失败后需要返回的默认值。

两个函数的使用都很简单，看个实例：

```
;在桌面的 test.ini 写入数据，不存在文件时会自动创建
IniWrite(@DesktopDir & "\test.ini", "setup", "test", "haha")
;读取刚才写入的数据
$a1 = IniRead(@DesktopDir & "\test.ini", "setup", "test", "")
;读取一个不存在的键名的数据
$a2 = IniRead(@DesktopDir & "\test.ini", "setup", "test2", "默认")
MsgBox(0, "", "test 键值：" & $a1 & @LF & "test2 键值：" & $a2)
```

要删除 ini 文件中指定字段下的键名，或者删除指定的字段，可以使用 IniDelete 函数。这个函数有三个参数，第三个参数可省略。当省略时，表示要删除指定的字段，不省略的话则表示删除字段下指定的键名。看例子：

```
;删除 test 键名
IniDelete(@DesktopDir & "\test.ini", "setup", "test")
;删除 setup 字段
IniDelete(@DesktopDir & "\test.ini", "setup")
```

如果需要读取 ini 中所有字段的名称，可以使用函数 IniReadSectionNames，看帮助文档中的例子：

```
$var = IniReadSectionNames(@WindowsDir & "\win.ini")
If @error Then
    MsgBox(4096, "", "错误, 读取 INI 文件失败.")
Else
    For $i = 1 To $var[0]
        MsgBox(4096, "", "字段名:" & $var[$i])
    Next
EndIf
```

如果要读取 ini 文件中某个字段下所有的键名和值，可以使用函数 IniReadSection，看例子：

```
$var = IniReadSection(@WindowsDir & "\win.ini")
If @error Then
    MsgBox(4096, "", "错误, 读取 INI 文件失败.")
Else
    For $i = 1 To $var[0][0]
       MsgBox(0, "", "关键字: " & $var[$i][0] & @LF & "值: " & $var[$i][1])

    Next
EndIf
```

有一点需要特别注意的是，使用 IniReadSectionNames 和 IniReadSection 函数时，因为要兼容 9x 系统，这两个函数都只能读取 ini 文件的前 32767 个字符。当文件大小超出限制时，将无法获取到完整的数据。

还有一点就是，像本章开头说的，ini 文件不一定就一定需要使用.ini 作为后缀。如果使用.exe 做后缀的话，只有文件符合 ini 文件的格式，那也一样可以使用 ini 文件读写函数读取和写入数据。

2、Txt 文档读写

AutoIt 读写文本文件，使用的函数有 FileOpen、FileClose、FileWrite、FileWriteLine、FileRead、FileReadLine。

不管是读或写文件，在 AutoIt 中既可以使用文件句柄来进行操作，也可以直接操作。我们先看如何直接操作：

```
;写入一个字符串并以换行符结尾, 也就是按行写
FileWriteLine(@DesktopDir & "\test.txt", "第一行")
;紧接着文本原来的数据写入 test 字符串
FileWrite(@DesktopDir & "\test.txt", "test")
;读取前四个字节的数据
$var1 = FileRead(@DesktopDir & "\test.txt", 4)
;读取第二行的数据
$var2 = FileReadLine(@DesktopDir & "\test.txt", 2)
MsgBox(0, "", "前 4 个字节:  " & $var1 & @LF & "第二行: " & $var2)
```

FileWrite、FileWriteLine 两个函数在写入数据的时候，若写入文件不存在，则会自动创建。不过需要注意的是，当文件的路径也不存在时，就会写入失败。

FileWrite、FileWriteLine、FileRead、FileReadLine 这四个函数的第一个参数，都可以是文件名，或文件句柄，上面的例子就是直接使用文件名来读写数据。这样的做法在少量的读写操作时用起来还是不错的，但是当要进行大量的读写操作时，这样做就会降低脚本的效率。因为牵扯到文件 I/O 操作，这里就不做说明。

我们只要知道当要进行大量次数的读写操作时，使用文件句柄来操作是比较理想的办法。要获得文件的句柄可以使用 FileOpen 函数，成功打开文件后将会返回一个文件句柄，供我们进行读写操作，当操作结束后，记得使用 FileClose 函数关闭句柄。

FileOpen 函数有两个参数，第二个参数是我们要重点注意的地方，这个参数将指定我们获取的句柄能对文件进行何种操作，这个参数默认是只读模式。

模式	[可选参数] 指定以何种模式(读或写)打开文件: 可以是下列几种: 0 = 只读模式(默认) 1 = 写入模式(附加数据到文件尾部) 2 = 写入模式(先删除之前的内容) 8 = 如果目标目录不存在就创建(参考注意). 16 = 强制使用二进制(字节)模式(参考注意) 32 = 使用 Unicode UTF16 小编码读写模式,读取不会覆盖存在的 BOM. 64 = 使用 Unicode UTF16 大编码读写模式,读取不会覆盖存在的 BOM. 128 = 使用 Unicode UTF8 (带 BOM)读写模式,读取不会覆盖存在的 BOM. 256 = 使用 Unicode UTF8 (无 BOM)读写模式. 16384 = 当打开一个文件读取时(文件没有 BOM), 使用完整文件 UTF8 检测. 如果没有使用这一模式,则只会检测文件最前端的 UTF8 标志. 文件夹路径必须存在(如果没有指定模式 '8' - 见注释).

一般情况下我们大多是使用前四种模式组合使用，至于后五种用的比较少，而且解释起来因为要涉及到字符编码问题，所以就不解释了。

我们先来看一个只读模式的例子：

```
$file = FileOpen(@DesktopDir & "\test.txt", 0) ;注意第二个参数 0
; 检查打开的文件是否可读
If $file = -1 Then
    MsgBox(0, "错误", "不能打开文件.")
    Exit
EndIf
MsgBox(0, "全部数据", FileRead($file))
FileClose($file) ;关闭句柄
```

上例中 FileOpen 函数使用的是只读模式打开文件，所以我们只能读取文件的内容，而不能写入，不信的话大家可以自己试试。要记住的是使用 FileOpen 函数获取了文件句柄，在读写操作完毕后，一定要记住使用 FileClose 函数关闭句柄。

接着我们来看看写入模式，写入数据有两个模式 1 和 2，查看上面的表格可以知道两者的区别。1 是在文件末尾接着写入数据，2 则是先清空文件内容然后再写入数据。

还有一个 8 模式，使用这个模式后，当文件的路径不存在时，就会自动创建路径。我们就获取一个 2 加 8 模式的句柄，看实例：

```
$file = FileOpen(@DesktopDir & "\t\test.txt", 2 + 8) ;注意第二个参数
; 检查打开的文件是否可读
If $file = -1 Then
    MsgBox(0, "错误", "不能打开文件.")
    Exit
EndIf
;写入一个字符串并以换行符结尾，也就是按行写
FileWriteLine($file, "行")
;紧接着文本原来的数据写入 test 字符串
FileWrite($file, "t")
MsgBox(0, "全部数据", FileRead($file)) ;句柄是写模式的，所以读数据会失败
FileClose($file) ;关闭句柄
```

上例的消息框显示将会是空，因为句柄不具备读取模式，所以无法读取内容，但是数据写入是成功了的。大家可以手动打开文件看一下。

16 模式我们挪到下一节讲。

3、二进制文件读写

FileOpen 函数获取的还要一个 16 模式，通过这个模式来打开文件进行读写，我们可以读取到文件内容的二进制数据，或者是按二进制修改文件数据。

我们来看一个例子：

```
$file = FileOpen(@WindowsDir & "\Notepad.exe", 0 + 16)
; 检查打开的文件是否可读
If $file = -1 Then
    MsgBox(0, "错误", "不能打开文件.")
    Exit
EndIf
MsgBox(0, "记事本的二进制数据", FileRead($file))
FileClose($file) ;关闭句柄
```

上例中把记事本程序的二进制数据读取了出来，我们在 FileOpen 函数中使用的模式是 0 加 16，也就是二进制读取模式。

除了读取，二进制也可以由写入模式。不过二进制写入一般都是替换原有数据的情况比较多，因为要用到二进制模式写入数据的情况，大多是要修改 EXE 或 DLL 文件，因为这两种文件的特殊性，我们如果对其写入新数据，将会导致原程序的损坏。而且我们大多都只是需要修改其中一些关键的数据而已，其它的数据都希望保持不变。

举个例子说，比如一个网游客户端的主程序里有连接服务端的 IP，我们想要修改这个 IP 的值，但是又要保持程序还能正常运行。此刻，我们就只需要把这个 IP 替换掉即可。

要达成上例中的目的，我们要先把程序的二进制数据读取出来，然后再进行替换，最后再重新写入数据。为什么要先读取再写入呢？这是因为文件句柄不能

同时具备读写两种模式，一个句柄只能是读或写一种模式。

因为教程的关系，无法附带一个 EXE 文件来演示对 PE 文件的二进制数据修改，所以这里我就拿上面例子中创建的 txt 文件进行二进制数据修改的演示，虽然文件类型不同，但是方法都是一样的。

我们看实例：（把文本中的 t 替换成 a）

```
$file = FileOpen(@DesktopDir & "\t\test.txt", 0 + 16)
; 检查打开的文件是否可读
If $file = -1 Then
    MsgBox(0, "错误", "不能打开文件.")
    Exit
EndIf
$dat =  FileRead($file) ;先读出数据
FileClose($file) ;关闭句柄

$a = Hex(Asc("a"), 2) ;获取 a 的 ascii 码值的 16 进制数
$t = Hex(Asc("t"), 2) ;获取 a 的 ascii 码值的 16 进制数
$dat2 =  StringReplace($dat, $t, $a) ;把 t 替换成 a

$file = FileOpen(@DesktopDir & "\t\test.txt", 2 + 16) ;二进制写入
FileWrite($file, $dat2) ;写入修改后的数据
FileClose($file) ;关闭句柄
```

上例脚本执行完毕后，大家可以手动打开文本查看，第二行的 t 已经被替换成 a 了。如果不理解为什么要使用字符的 ascii 码值的 16 进制数来进行替换的，可以把读取到的二进制数据用消息框显示出来观察一下。

在二进制数据替换的过程中，有一个需要大家特别注意的地方，那就是目标数据与替换数据的长度应该相等。当然，在文本文件中这个要求是无意义的。但是在 PE 文件或者动态链接库文件中，因为文件结构的关系，一旦把文件数据修改的与原文件大小不一，就会出现文件文法正常工作的情况。

所以当我们在使用二进制数据替换 EXE 之类的比较特殊的文件时，需要特别注意目标数据与替换数据的长度要一样才行。

除了读写文件，还有个删除文件的函数 FileDelete，这个函数只有一个路径参数，用于指定需要删除的文件。文件可以使用通配符，用来表示要删除某种类型的文件，比如使用*.txt 表示删除所有的 txt 文件。

来看使用实例：

```
FileDelete(@DesktopDir & "\test.txt") ;删除桌面的 test.txt
FileDelete(@DesktopDir & "\t\*.txt") ;删除桌面 t 目录下的所有 txt 文件
```

要删除目录需要使用函数 DirRemove，看实例：

```
DirRemove (@DesktopDir & "\t ") ;删除桌面的 t 目录
```

这个函数，还有第二个参数，是用来指定是否连同目录底下的子目录和文件一起删除的，我就不做演示了，大家自己试试看。

第七章 进程管理

作为一个强大的脚本语言，AutoIt 当然也能创建和结束系统进程。所谓进程就是正在进行中的程序，硬盘上存放的可执行文件是不能称为进程的，在内存中执行的文件才是进程。

进程就是可执行文件使用的资源总和，包括虚拟地址空间、对象句柄、数据、代码等等。同一个可执行文件，之所以能建立多个进程，这是因为不同的进程之间的地址空间等资源是相互隔离的。

1、进程列表

AutoIt 内置函数中有一个获取进程列表的函数 ProcessList，使用这个这个函数可以获取一个简单的进程表单，其中包括进程名和进程 PID。

我们来看看实例：

```
#Include <Array.au3>
$ProcessList = ProcessList()
_ArrayDisplay($ProcessList, "进程列表")
```

上例的第一行出现了一个之前没见过的关键字#Include，这个关键字的作用是把其它脚本包含到当前的脚本当中，这样我们就可以使用在其它脚本中写好的自定义函数了。通常我们把这个包含的脚本叫做 UDF（User Defined Function）。

像上例中的第三行，使用的就是 Array.au3 这个 UDF 中定义的函数。UDF 除了用来定义函数，也同样可以用来定义变量或者数据结构。当我们要写个大工程时，有时一个变量需要在多个脚本中引用，为了方便修改与管理，此时我们就可以把这个变量在一个独立的脚本文件中定义，然后在其它脚本中引用即可。

一个进程除了进程名和 PID 之外，还有许多其它的信息，比如父进程、子进程、子线程、引用的 DLL 模块等等。要得到这些信息，使用内置函数 ProcessList 显然是不可行的。

此时我们可以借助 API 来实现，API 中有一个进程快照函数就可以完全胜任我们的需求，这个函数就是 CreateToolhelp32Snapshot。关于这个函数的用法我就不在此讲了，如果想了解的朋友，可以看下我以前写的一个 AU3 进程管理器，就完全是使用这个函数来实现的。

2、进程等待及结束

AutoIt 中进程等待函数有两个，ProcessWait 等待进程出现，ProcessWaitClose 等待进程结束。这两个函数的用法很简单，此处就不写实例了。

AutoIt 中结束进程可以使用函数 ProcessClose，使用此函数结束进程时，如

果使用的参数是进程名，且这个进程名存在多个进程，那 PID 最高的进程将会被结束。也就是说，如果进程中有很多同名进程的话，使用进程名来结束进程，只能结束一个进程，而不会结束所有的同名进程。这个函数的使用也很简单，所以也不写实例了。

3、运行文件

在 AutoIt 中，如果要运行一个文件，可以使用的函数有 Run、RunWait、RunAs、RunAsWait、ShellExecute、ShellExecuteWait。

Run 和 ShellExecute 两个函数是以系统当前的用户运行一个文件。

RunWait 和 ShellExecuteWait 两个函数是以系统当前的用户运行一个文件，并且等待创建的程序结束后再继续执行脚本。

RunAs 和 RunAsWait 两个函数则可以指定一个系统用户来运行一个文件。

这里我们只讲 Run、RunWait、ShellExecute、ShellExecuteWait 四个函数，另外两个因为不常用所以不讲，如果有兴趣的朋友可以自己试试看。

先来看 Run：（帮助中的例子，最大化执行记事本程序）

```
Run(@WindowsDir & "\Notepad.exe", "", @SW_MAXIMIZE)
```

接着试试 RunWait：（帮助中的例子，等待创建的记事本进程结束）

```
$val = RunWait(@WindowsDir & "\Notepad.exe", "", @SW_MAXIMIZE)
MsgBox(0, "", "退出代码: " & $val)
```

如果要执行批处理指令怎么办？我们可以使用一个宏**@ComSpec** 来达成目的，这个宏的值是系统命令解释程序的路径。看个例子：

```
Run(@ComSpec & " /k help | more ")
```

上例在批处理指令之前加了一个/k 参数，此参数的作用是在指令执行完毕后，保留 cmd 窗口。如果想不想保留 cmd 窗口的话，就使用/c 参数。

Run 和 RunWait 两个函数只能执行可执行程序，且文件扩展名为 EXE、BAT、COM 或 PIF。如果文件扩展名不是这几个，那函数将会执行失败。

如果一个文件是可执行程序，但扩展名不是上述的几个，那要如何执行？办法有两个，一是修改注册表，把此类扩展名的文件设置为 EXE 类型的文件；另一种就是使用 API 来执行文件。

我们先看下改注册表的方法：（把.txx 类型的文件设置为 EXE 类型文件）

```
RegWrite("HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.txx", " Content
Type", "REG_SZ", " application/x-msdownload")
RegWrite("HKEY_LOCAL_MACHINE\SOFTWARE\Classes\.txx", "", "REG_SZ",
"exefile")
;复制一个记事本程序到桌面，并改后缀为.txx
FileCopy(@WindowsDir & "\Notepad.exe", @DesktopDir & "\Notepad.txx")
Run(@DesktopDir & "\Notepad.txx")
```

上例运行后，查看任务管理器，将会看到一个以.txx 为后缀的进程。

Windows 中可以使用 WinExec 和 ShellExecute 这两个 API 来执行一个可执行文件，当然还有一个进程创建函数 CreateProcess 也同样可以用来执行一个可执

行文件。至于三者的区别，有兴趣的朋友可以谷歌看看，我这里不做解释。

三个 API 中，WinExec 是最简单，这里我就用这个来做实例：

```
;复制一个记事本程序到桌面，并改后缀为.aaaaa
FileCopy(@WindowsDir&"\Notepad.exe",@DesktopDir&"\Notepad.aaaaa")
_Exec(@DesktopDir & "\Notepad.aaaaa")

Func _Exec($lpCmdLine, $uCmdShow = @SW_SHOW)
    $Ret = DllCall("kernel32.dll", "Int", "WinExec", "str", $lpCmdLine,
"Int", $uCmdShow)
    If $Ret[0] < 32 Then
        Return SetError(1, 0, $Ret[0])
    EndIf
    Return $Ret[0]
EndFunc
```

上例中 WinExec 这个 API 被我写成了一个自定义函数_Exec。运行成功后，在进程管理器中我们会发现一个后缀为.aaaaa 的进程。

如果要执行的文件不是一个可执行文件，那我们可以使用 ShellExecute、ShellExecuteWait 函数来实现目的。这两个函数会根据文件后缀，寻找关联的程序运行文件，例如一个 rar 压缩包文件，将会使用 winrar 程序来打开它。

我们来看一个实例：

```
ShellExecute(@WindowsDir & " \win.ini")
```

ShellExecuteWait 函数也是一样：

```
$val = ShellExecuteWait(@WindowsDir & " \win.ini")
MsgBox(0, "", "退出代码: " & $val)
```

要知道的是 ShellExecute、ShellExecuteWait 函数也可以执行可执行文件，但是这两个函数执行后不会返回进程的 PID。