

# Level 1

## 1.1 AutoIt v3 概述

### 1.1.1 AutoIt v3 是什么？

AutoIt v3（简称 AU3），是一款免费的、具有类 BASIC 语法的脚本语言，被设计用于自动控制 Windows 图形用户界面（GUI）和编写常规脚本。

由于 AutoIt 最初被设计为批量、自动的实现对大量 PC 的配置，所以可以非常方便的实现模拟键盘鼠标动作、操纵窗口或进程、与标准化控件进行交互等。AutoIt 被设计的尽可能小巧，可编译为能独立运行的 exe 应用程序，可运行于几乎所有版本的 Windows 操作系统，并且不依赖任何运行库、外部 dll 或注册表项目。

随着时间的发展，越来越多的功能已经融入到 AutoIt 中，如：支持复杂表达式、支持用户自定义函数、支持 GUI、支持 COM、支持正则表达式、可调用外部 dll 和 Windows API、支持 Unicode、支持 64 位、具有放心可用的数字签名、兼容 Windows 账户控制（UAC）等等。现在的 AutoIt v3 已经是一款强大的脚本语言！

最好的一点是，AutoIt 将继续免费下去！

### 1.1.2 AutoIt v3 的优势

诚然，作为轻量级的脚本语言，自然比不上大型编程语言那么强大，所以如果您想成为一个职业程序员或考虑以编程为生，建议您还是去认真学习 C++、C# 或 Java 等。而如果您并非以编程为生，却需要通过编程实现您的想法，从而提高、改善工作效率，那么 AutoIt v3 将是您最棒的选择。

AutoIt v3 独到的优势：

#### （1）上手速度快

AutoIt v3 有着宽松的语法，初学者可以在不必过多考虑语法问题的前提下，将更多的精力集中于学习和实现算法上。虽然宽松的语法不太利于培养优秀的编程习惯，但却非常有利于快速学习和快速掌握。这就像新手开车，自动挡反而能令新手将更多精力集中于前方道路状况，而不用分心于什么时候控制离合器、什么时候增减挡。

#### （2）转化时间短

任何编程语言都存在一个转化时间问题，即从“开始学习编程”到“能够使用编程”需要多久。AutoIt v3 语法宽松，包含大量与 Windows 相关常用操作（如管理注册表、管理文件、管理文件夹、管理进程、模拟键盘鼠标动作、控制 GUI 等），所以只要专心学习，只需要数周时间就可以写出不错的小程序。这相比大型编程语言动辄数月的转化时间要短很多。

#### （3）丰富的函数库

AutoIt v3 对 Windows 系统有着优秀操作性，例如读/写/删注册表值、复制/移动/删除文件或文件夹、GUI 窗口/控件控制等，都可通过函数实现。又有不少高手为 AutoIt v3 添加了大量的用户自定义函数（UDF），使可用功能得到了极大的拓展。使用者可以在不必了解函数工作原理的条件下，

即可使用函数来实现复杂功能，有效提升了程序设计效率。

#### (4) 完善的帮助文档

帮助文档，就像是一本关于 AutoIt v3 的大百科全书，AutoIt v3 大小巨细均在其中。当有什么不理解或遗忘时，可以在帮助文档中快速查到相关内容。我们甚至不需要记忆所有语句和函数的具体功能，待需要时在帮助文档中快速查询一下即可。AutoIt v3 的帮助文档现已有详细的汉化版本，很大程度上方便了学习和使用者。

选择编程语言，一定要根据自身的情况和需求来判断，不一定要选择最强大或最著名的，但一定要选择适合自己、适合使用的！

### 1.1.3 创建 AutoIt v3 编程环境

#### 1、下载

IT 天空为大家提供优质的 AutoIt v3 编程环境汉化版，下载地址为：

<http://www.itiankong.com/download/au3.html>

目前 AutoIt v3 稳定版的版本为 3.3.12.0，本书内代码均基于此版本。今后会有更高版本，但不同版本之间仅有极少数语句或函数存在差异。

AutoIt v3 仅有几十 MB，很快即可下载完毕。

#### 2、安装

下载完成后，双击运行，弹出下图界面（版本升级后界面可能会略有不同）：



(图 1-1)

建议安装全部项目，设定安装路径后，单击右下角的“安装”按钮。稍事等待后，安装完成。

安装完成后，桌面会出现“AutoIt 脚本编辑器 (SciTE)”、“AutoIt 图形界面编辑器 (Koda)”等快捷方式，右键新建菜单中会出现“AutoIt v3 脚本”。

### 3、卸载

如果不再需要 AutoIt v3，可以将其卸载。卸载方法很简单，只需要进入 AutoIt v3 安装目录，找到“AccAu3Uninst.exe”，双击运行按提示操作即可。

## 1.1.4 编写第一个 AutoIt v3 程序

### 1、创建

脚本就是一个以.au3 为后缀的纯文本文件，创建方法很容易：

方法 1：在桌面或其他存储位置，单击鼠标右键 → “新建” → “AutoIt v3 脚本”。

方法 2：运行 SciTE 编辑器，菜单栏 → “文件” → “保存”，选择位置，设定名称，保存即可。

### 2、编辑

编辑脚本的一般方法如下：

方法 1：右键单击 AutoIt v3 脚本文件，选择“编辑(SciTE)”。

方法 2：运行 SciTE 编辑器，菜单栏 → “文件” → “打开”，浏览选中脚本文件，单击“打开”。

以上两种方法均可使用 SciTE 编辑器打开 AutoIt v3 脚本文件。

打开脚本后，我们写入一段简单的代码，作为我们的第一个 AutoIt v3 程序：

```
MsgBox(0, 'Demo', 'Hello World')
```

(友情提示：语句或函数所使用的符号，如括号、逗号、单引号、双引号等均为英文符号)

写好之后，菜单栏 → “文件” → “保存” 或 “Ctrl+S” 对脚本进行保存。建议养成经常“Ctrl+S”的习惯，以随时保存我们的劳动成果。

### 3、运行

运行脚本的一般方法如下：

方法 1：使用 SciTE 编辑器打开要运行的脚本，菜单栏 → “工具” → “执行”，或直接按 F5。

方法 2：在脚本文件上单击鼠标右键，选择“运行”。

方法 3：如果在脚本文件上单击右键，排在第一位的是“运行”，那么双击脚本亦可直接运行。

两种方式均可使脚本运行，上述代码运行后如下图：



(图 1-2)

### 4、编译

什么是“编译”？

AutoIt v3 脚本的编译，简单说就是将 AutoIt v3 脚本转化为 exe 可执行文件。被编译出的 exe 文件，其代码将不可被再行更改，如需变更则需要修改源脚本代码，然后再次执行编译。而编译后的 exe 文件，将不再依赖 AutoIt v3 编程环境，可独立运行于任何安装有 Windows 操作系统的 PC 上。

编译脚本的一般方法如下：

方法 1：使用 SciTE 编辑器打开要编译的脚本，菜单栏 → “工具” → “编译”，或直接按 F7。

方法 2：在脚本文件上单击鼠标右键，选择“编译”。

两种方法均可将 AutoIt v3 脚本编译为 exe 可执行文件。运行编译后的 exe 可执行文件，效果与直接运行脚本一致。

有兴趣可以测试将脚本和编译后的 exe 文件移动到其他安装有 Windows 操作系统的 PC 中，测试脚本是否可运行，exe 文件是否可运行。

## 1.2 算法

### 1.2.1 算法的定义

算法，广义上讲，为解决一个问题而采取的方法和步骤称为算法。例如乘坐火车从 A 地到 B 地，需要经历 A 地 → 买票 → 进站 → 检票 → 乘车 → 到站 → 出站 → B 地等步骤，这些步骤组成了“从 A 地乘火车到 B 地”的算法。

算法并不是一个陌生的存在，我们的生活、学习或工作中无时无刻不在使用算法，例如穿衣服、刷牙、吃饭等等，事实上每件事情都在按照一系列的步骤进行着，只是我们已将它们化为了生活习惯而无需每次都考虑其算法罢了。

算法也有优劣，就像平时工作中一样，同样的事情有些人做起来拖拖拉拉、颠三倒四，而有些人做起来干脆利落、有条不紊，问题就出在他们所使用的算法效率不同。程序设计的亦是如此，解决同样的问题时，好的算法可以事半功倍，差的算法则会事倍功半。所以，在书写程序代码之前认真考虑使用怎样的算法，什么样的算法更为简练高效，是程序设计者必备的素质。

### 1.2.2 算法的描述

当我们在头脑中形成了算法，就需要将其以合理有序的方式描述出来。算法的描述方式很多，如自然语言、流程图、N-S 图、PAD 图、伪代码等等，这里介绍两种比较常用的算法描述方式：

#### 1、流程图

流程图是一种用图形来标识算法的描述方式，它通过几何图形和流程线来描述各步骤的执行，这种方法直观形象、逻辑清晰、便于理解。但在描述大程序时会造成流程图太大、流程转向过多等问题，

以至于难以读懂。不过，流程图对于小程序和初学者而言是十分有帮助的。

流程图中几何图形的意义：

	处理框 ( 矩形 )： 表示完成某种操作		输入输出框 ( 平行四边形 )： 表示输入或输出数据
	起止框 ( 圆角矩形 )： 表示程序的起始或终止		流程线 ( 箭头 ) 表示程序执行的方向
	判断框 ( 菱形 )： 表示进行判断		连接点 ( 圆形 ) 表示两段流程图的连接点

我们举个简单的算法例子：我们要外出，先观察天气，如果下雨，则带雨伞，否则不带。用流程图描述一下：



( 图 1-3 )

可以看出，流程图在描述简单算法时十分直观清晰。建议初学者在写代码前先画个流程图表达一下自己的算法，这样可以在写代码时一边写一边对照，有助于提高思路的条理性，并形成良好的代码书写习惯。

## 2、伪代码

伪代码是介于自然语言和程序语言之间的一种算法描述形式，它结合了程序代码的书写方式和自然语言描述，更有利于将算法转化为程序代码。伪代码并没有太严格的语法要求，只需要能明确表达步骤含义，并尽可能与程序代码语法接近即可（便于向真正代码的转化）。不过因其需要先对程序语法有一定了解，所以不太适合新手，但在熟悉了程序语法后比流程图更有效率，而且在描述大型程序时更为条理易懂。

使用与刚才相同的例子：我们要外出，先观察天气，如果下雨，则带雨伞，否则不带。用伪代码描述一下：

```
准备外出  
观察天气  
If 下雨 Then
```

```

带雨伞
Else
    不带雨伞
EndIf
外出

```

可以看出，伪代码更接近真正代码的写法，而且书写简洁，不需要画几何图形等。但这样的写法会令新接触编程的初学者一头雾水，并且不利于初学者掌握程序逻辑。

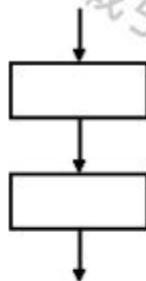
综上，两种算法各有优势。流程图更适合新手和小型程序，可直观的了解代码逻辑；伪代码书写简练，更利于转化为真正的代码，适合有基础的编程者。至于使用哪一种，这取决于程序规模、设计者水平和设计习惯，但无论使用哪一种，请务必养成在写代码前认真写好算法的习惯。

### 1.2.3 算法的三种基本结构

Bohm 和 Jacopini 于 1966 年提出程序有三种基本结构：顺序结构、选择结构和循环结构。这三种结构都具有“只有一个入口和一个出口”的特点，不会出现死循环。

#### 1、顺序结构

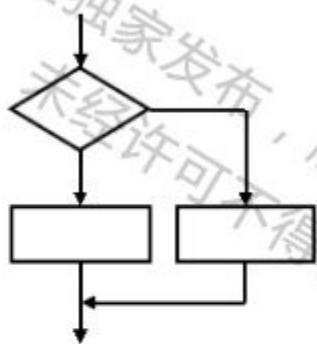
顺序结构是一种线性结构，也是程序设计中最简单、最常用的基本结构。顺序结构程序是把要执行的各步骤依次排列起来，当程序运行后，以自顶向下、自左至右的顺序执行这些语句，直至执行完所有语句或执行到退出代码。



( 图 1-4 )

#### 2、选择结构（分支结构）

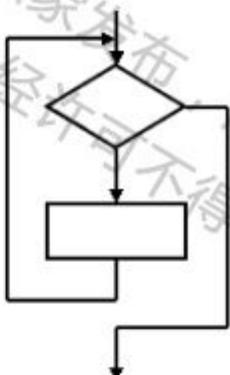
选择结构是一种常用的主要基本结构，是用来描述分支现象的重要手段。选择结构是根据某个条件是否成立来决定下一步执行哪些工作的。在这种条件下，程序不再按照行号的顺序来执行各行语句，而是根据给定的条件来决定选取哪条路径、执行哪些语句。



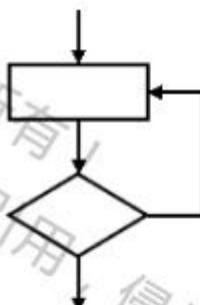
( 图 1-5 )

### 3、循环结构

当需要在某个条件成立或不成立时反复执行某段语句，直到条件不成立或成立后才停止这种重复工作，这类程序结构叫做循环结构。循环语句有助于减少人为的重复性工作，与数组等数据类型关系密切，是一种十分常用的程序控制结构。



( 图 1-6 )



( 图 1-7 )

#### 1.2.4 简单算法举例

##### 1、判断平闰年

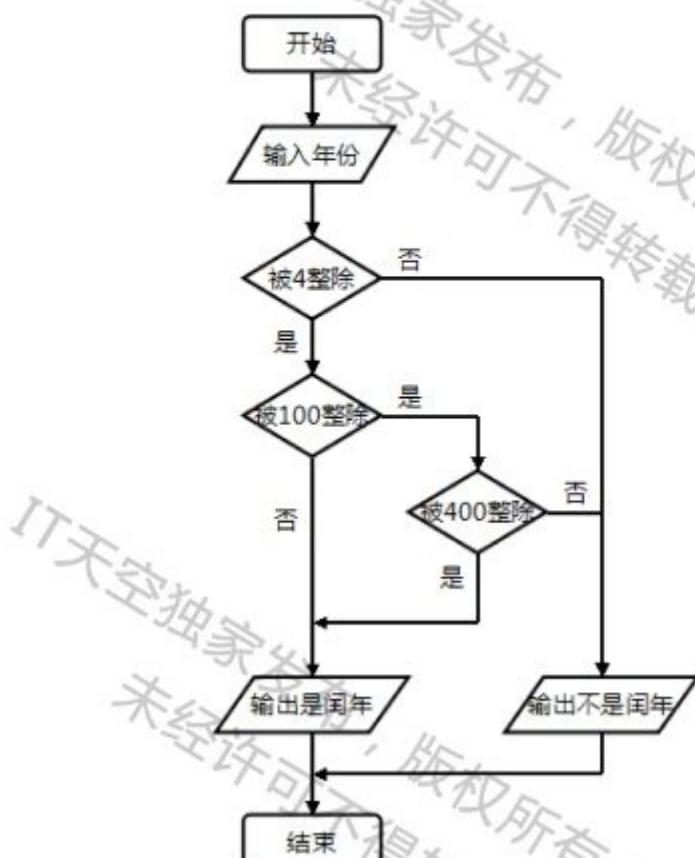
判断某年是否为闰年，条件是：

条件 1：年份能被 4 整除，不能被 100 整除，是闰年。

条件 2：年份能被 4 整除，能被 100 整除，还能被 400 整除，是闰年。

明确这两个条件后，我们要将条件转化为算法。

( 1 ) 以流程图描述：



(图 1-8)

(2) 以伪代码描述：

```

输入：年份

If 年份 可以被 4 整除 Then
  If 年份 可以被 100 整除 Then
    If 年份可以被 400 整除 Then
      输出信息：年份 是闰年
    Else
      输出信息：年份 不是闰年
    EndIf
  Else
    输出信息：年份 是闰年
  EndIf
Else
  输出信息：年份 不是闰年
EndIf

退出
  
```

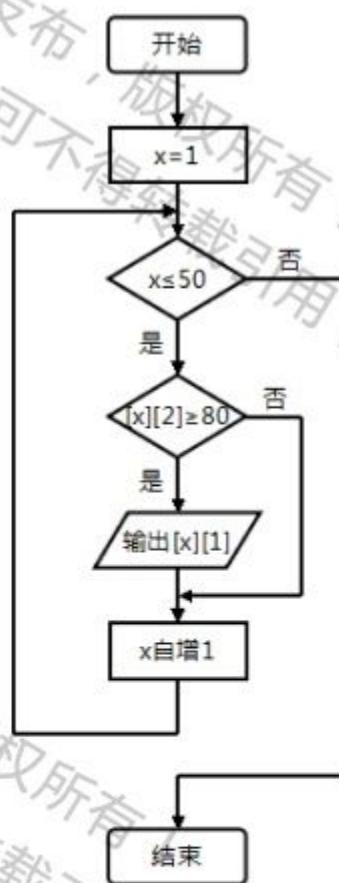
## 2、筛选成绩高于 80 分的学生

有一张表格中包含了 50 个学生的姓名、班级、性别、成绩（形如下表），输出其中成绩高于 80 分的学生姓名。

张娜	93
李帆	72
王洋	87
.....	.....
赵勇	86

假设我们以“表格[1][1]”来表示表格的第 1 行第 1 列，那么“表格[1][1]”的值为“张娜”，而“表格[1][2]”的值为“93”分，以此类推得到规律，表格[x][1]的值为学生的姓名，而表格[x][2]的值为学生的成绩。那么，我们要做的是逐个判断表格[x][2]（x 为 1 到 50）是否大于 80，如果是则输出表格[x][1]。

(1) 以流程图描述：



(图 1-9)

(2) 以伪代码描述：

```

x=1
While x<=50
    If 表格[x][2]>=80 Then
        输出 表格[x][1]
    EndIf
    x 自增 1
WEnd

退出

```

### 1.2.5 算法的特性

#### 1、有穷性

一个算法应包含有限的操作步骤，而不能是无限的。例如设定循环结束的条件为某种不可能达到的条件时（例如“当秒针指向第 61 秒”），则循环将永远无法停止，这就不符合有穷性要求。

另一个层面讲，有穷性还指算法的执行应该在一个合理的范围内，例如一个需要执行 1000 年的算法，很明显不在“合理”的实际范围内。不过“合理范围”并无严格的规范，由人们的常识和需要而定。

#### 2、确定性

算法中的每一个步骤应当是确定的，而不应当是含糊的、模棱两可的。例如“将手举过头顶”这个说法，双手、左手还是右手？举过头顶多少距离？这些都是不确定的，算法中不应出现这类步骤。换句话说，算法应当有唯一的含义，而不应当有两种或两种以上含义。

#### 3、有大于等于 0 个输入

所谓输入是指在算法执行时需要从外界取得必要的信息。例如算法是计算两个数的和，输入 1 和 2 则结果为 3，这里的 1 和 2 就是两个输入。而有些算法不需要输入，例如算法的功能是关闭计算机，那么执行算法则会形成关机的结果。

#### 4、有大于等于 1 个输出

算法的目的是为了达到一定的目的。例如求两数和的算法，得到的加法运算结果即是输出。不过输出并不局限于“值”，而是达到一定的结果，例如关闭计算机的算法，执行后关机就是结果。没有输出结果的算法是没有意义的。

#### 5、有效性

算法中的每一个步骤都应当能有效的执行，并得到确定的结果。例如算法中的某个步骤是计算两

个数的商，当除数为 0 时就不具备有效性。

### 1.2.6 算法的结构化

所谓算法的结构化，是指将一个复杂的算法进行分阶段、分模块处理。结构化后的算法，每个功能模块相对独立，分别执行不同的功能，而所有的模块又能有机的结合，形成一个整体算法。

这就像我们写一本书，要先将内容分章、分节，每章讲述一个方面，每节讲述本章的一个重点。所有的章节集合在一起，就形成了一本完整的书。

对于比较复杂的算法，结构化处理是必须的。我们可以将算法分为数个模块，每个模块下又有数个子模块，而每个子模块下又划分为不同的小功能模块。例如：

```
功能模块 1
    功能模块 1.1
        功能模块 1.1.1
        功能模块 1.1.2
    功能模块 1.2
        功能模块 1.2.1
        功能模块 1.2.2
    功能模块 1.3

功能模块 2
    功能模块 2.1
    功能模块 2.2
        功能模块 2.2.1
        功能模块 2.2.2
        功能模块 2.2.3

功能模块 3
    功能模块 3.1
    功能模块 3.2
    功能模块 3.3
```

这样划分后，程序的整体算法思路变得清晰可见。每个功能模块可以独立开发，使开发更具有集中性。在出现程序问题时，可以将问题定位到模块，从而对问题模块进行修正而不影响其他。

在很多大中型程序中，不同的模块都是交于不同的人或团队进行开发维护，所以结构化思想是尤为重要的。另一方面讲，即便我们只是开发小型应用程序，也应当使用结构化的思想，例如将程序划分为输入、处理、输出等模块进行书写，这有助于我们提高编程思想，形成良好的编程习惯。

## 1.3 语言基础

### 1.3.1 变量

在程序运行过程中，其值可以被改变的量，叫做“变量”。变量一般用于存储在程序执行过程中值存在变化的量。定义变量的一般方法为：

```
Dim <变量>
```

除了 Dim，还可以使用 Local 或 Global 来定义变量，因此二者涉及其他方面知识，故将于后续章节中讲解。

变量有其命名规则，不得违反，否则视为错误。变量的命名规则为：以“\$”开头的，字母、数字、下划线组合。在符合规则的条件下，您可以按照自己的需求来定义变量，例如\$a、\$ab、\$AbC、\$\_a12、\$abc\_123、\$2ac3 等均是合法的变量。

常见的变量定义形式：

#### 1、定义一个变量

一行定义一个变量

```
Dim $a
```

#### 2、定义一个变量，并为其赋值

(1) 先定义一个变量，再为其赋值：

```
Dim $a  
$a = 10
```

(2) 定义一个变量的同时为其赋值（效果与上例等同）：

```
Dim $a=13
```

#### 3、定义多个变量

(1) 分多行，每行定义一个变量：

```
Dim $a  
Dim $b  
Dim $c
```

(2) 一行定义多个变量（效果与上例等同）：

```
Dim $a, $b, $c
```

#### 4、定义多个变量，并为其赋值

(1) 先定义多个变量，再分别为其赋值：

```
Dim $a  
Dim $b  
Dim $c  
$a = 10  
$b = 20  
$c = 30
```

## Let's AutoIt Plus

(2) 定义多个变量的同时分别为其赋值(效果与上例等同)：

```
Dim $a = 10, $b = 20, $c = 30
```

(3) 定义多个变量，为其中某个赋值：

```
Dim $a  
Dim $b  
Dim $c  
$c = 30
```

(4) 定义多个变量的同时为其中某个赋值(效果与上例等同)：

```
Dim $a, $b, $c = 30
```

(值得注意的是，“=”只是将其右侧的值赋值给其左侧的那个变量)

### 说明：

(1) 定义变量时，一般需要使变量名称具有可读性，例如年龄定义为：\$Age，书籍名称定义为：\$BookName 等等。这样可以有效提高程序的可读性，便于维护。

(2) 变量的命名除了要具备可读性以外，还有两种命名习惯。一种习惯是：变量名使用小写，例如\$file、\$path 等；而如果变量名由多个单词组成，则从第二个单词开始首字母大写，例如\$myName、\$appDownloadPath 等。另一种习惯是：变量名首字母大写，例如\$Student、\$Teacher 等；而如果变量名由多个单词组成，则每个单词的首字母大写，例如 \$FileSystem、\$SoftwareInstallPath 等。而无论使用哪一种，请保持习惯，不要两种混用。

(3) AutoIt v3 的变量具备“即时定义并使用”的方式，即可以不必先定义变量，而是在使用时自动定义。但这种方式易造成混乱和错误，并导致后期维护困难。所以强烈建议坚持“先定义、后使用”的原则。

(4) 变量定义的时机也有两种习惯。一种习惯是先定义所有即将使用的变量，而后使用。另一种习惯是在即将使用变量前定义变量。而无论使用哪一种，请保持习惯，不要两种混用。

(5) AutoIt v3 的变量名并不区分大小写，\$var、\$Var、\$VAR 会被认为是同一个变量。

(6) 同一作用域内，如果一个变量被重复定义不会报错，重复定义会被自动忽略。

(7) 变量在被定义后，如果不为其赋值，那么其默认值为空字符串。

(8) 在整个程序的运行过程中，可反复为一个变量赋值多次，赋新值后旧值自动无效。

### 1.3.2 常量

在程序运行过程中，其值不可以被改变的量，叫做“常量”。常量一般用于存储在程序执行过程中值是固定的量。定义常量的一般方法为：

```
Const <常量> = <值>
```

除了 Const，还可以使用 Local Const 或 Global Const 来定义常量，因此二者涉及其他方面知识，故将于后续章节中讲解。

常量命名规则与变量相同，不再赘述，请参照 1.3.1 节。

常见的常量定义形式：

## 1、定义一个常量

一行定义一个常量，并为其赋值

```
Const $M = 100
```

## 2、定义多个常量

(1) 分多行，每行定义一个常量，并为其赋值

```
Const $M = 100
```

```
Const $N = 200
```

(2) 一行定义多个常量，并为其赋值

```
Const $M = 100, $N = 200
```

### 说明：

(1) 定义常量与定义变量最大的区别，在于定义常量的同时必须为其赋值。而常量值一旦定义，则在整个程序运行过程中不可被改变。

(2) 定义常量时，一般需要使常量名称具有可读性，例如最大值定义为：\$MAX，价格定义为：\$PRICE 等等。这样可以有效提高程序的可读性，便于维护。

(3) 常量的命名习惯与变量类似，但常量一般会采用“前缀标记”方式或“全大写”方式与变量加以区分，前缀标记方式会在后续章节内讲解，对于初学者建议使用全大写方式来命名常量，例如\$MIN、\$PI、\$SQLSERVER，便于与变量进行区分。

(4) 常量必须先定义，后使用。

(5) AutoIt v3 的常量名并不区分大小写，\$max、\$Max、\$MAX 会被认为是同一个常量。

(6) 同一作用域内，常量不可被重复定义，否则会报错。

(7) 在常量被定义并赋值后，不要试图用任何方法改变其值，否则会报错。

### 1.3.3 数据类型

在 AutoIt v3 中，定义一个变量或常量时无需指定其数据类型，这与一些强语法程序设计语言有很大不同。例如在 C++ 中，定义一个 Int 类型的变量，那么这个变量就只能存储特定范围内的整数值；而在 AutoIt v3 中，定义一个变量无需声明类型，这个变量既可以存储整数，又可以存储浮点数，还可以存储字符串等。

这种不区分数据类型的方式，可以使程序代码变的灵活，也易于初学者在不必过多考虑语法的条件下专注于算法设计，但这样会令程序变得易出错，后期的代码维护和错误排查变得非常复杂。所以，我们必须自己做到心中有数，主动避免混用数据类型的情况，以形成良好的编程习惯。

因为 AutoIt v3 的变量或常量并不区分数据类型，所以下面只是大体上将常用的数据划分了五种类型，注意，这并不是全部数据类型，亦不是每类都划分的很详细，高级语言中会非常详细的划分出十几种数据类型。

#### 1、数值类型

数值类型数据，是程序设计中最常用的一类数据，与我们平时生活中所提到的“数”的概念接近，个别书写方法与日常生活中略有不同。

(1) 普通十进制值，如：5、68、5.33、-7.7、-30，与数学中书写方法一致；

(2) 指数型十进制值，如：1.2e3，等同于数学中的 $1.2 \times 10^3$ ；

(3) 十六进制值，如：0x312、0x4fff，以“0x”开头以与十进制值区分。

数值型变量的定义举例：

```
Dim $Num1 = -3, $Num2 = 15, $Num3 = 9.2  
Dim $TotalPrice = 1.2e6  
Dim $Color = 0x10fe35
```

## 2、字符串类型

字符串类型数据，一般由一对英文双引号（"）或单引号（'）包含起来的一个或多个字符，如'a'、'abc'、"Tom"、"Hello World !"等。单双引号在使用时并无差别，作用相同，但必须同时使用一对单引号或双引号对字符串进行包含，否则字符串无效。

字符串型变量定义举例：

```
Dim $User = 'Skyfree', $Organization = 'iTianKong.com'  
Dim $WinDir = "C:\Windows"  
Dim $SystemVersion = "Windows 7 Ultimate"
```

除了上述常规字符串，还有一些特殊情况。例如下述软件自动安装命令：

```
D:\Install.exe -Path:"C:\Program Files" -Auto
```

这条命令中本身就带有英文引号，而如果将此命令直接赋值给变量，就会出现问题。

处理这类已包含英文引号的字符串时，可以将字符串中的英文引号连续书写两次，以表明这个引号是字符串内的，而不是用来包含字符串的，例如：

```
Dim $Command = "D:\Install.exe -Path:**C:\Program Files** -Auto"
```

这种方法虽然可以解决问题，但当引号多时非常容易引起混乱。解决这类问题还可以使用单双引号混合的方法，例如：

```
Dim $Command = 'D:\Install.exe -Path:"C:\Program Files" -Auto'
```

这个例子中，字符串两侧的英文单引号用来标明这段内容是字符串，这样就不与字符串中的英文双引号起冲突了。相对来说，这种方法更为简便，在实际编程中常用。

## 3、布尔类型

布尔类型是最简单，但极为常见的值。说其简单，是因为布尔型数据只有两个值：True（真）和False（假）；说其极为常见，则是因为几乎所有的逻辑判断都需要它的存在。

在选择结构中，布尔值决定了分支的走向。例如1.2.2中出门前判断是否下雨而是否带雨伞的算法，如果下雨为True，则带雨伞，如果下雨为False，则不带雨伞。这就是一种简单的布尔值运用。

布尔值变量定义举例：

```
Dim $Flag1 = True, $Flag2 = False
```

## 4、数组类型

数组类型是一种特殊的数据类型，它用于存储一系列相关值，并可以通过元素序号进行访问。由于此数据类型较为复杂，将在后续章节中详细讲述，故此处不再赘述。

## 5、结构体

结构体也是一种特殊的复杂数据类型，对应高级语言中的结构体或类概念，在 AutoIt v3 中经常使用结构体来存储 Windows API 类函数的返回值。由于此数据类型较为复杂，将在后续章节中详细讲述，故此处不再赘述。

### 1.3.4 不同类型数据间的相互转化

不同类型数据间是可以相互转化的，在转化的过程中会产生一定的变化，下面我们就常见的转化做一下了解：

#### 1、数值转字符串

- (1) 普通十进制值，可以直接转化为字符串，例如 1200 转化为 "1200"；
- (2) 指数型十进制值，将实际值转化为字符串，例如 1.2e3 转化为 "1200"；
- (3) 十六进制值，会将值转化为十进制值后再转化为字符串，例如 0x3cd266 转化为 3986022。

#### 2、数值转布尔

- (1) 一切非 0 数值转化为 True，例如 1、25、-31、17.5、3e5；
- (2) 0 数值转化为 False。

#### 3、字符串转数值

- (1) 空字符串 (" 或 "") 转化为 0；
- (2) 如果字符串仅包含普通十进制值，则转化为十进制值，例如 "125" 转化为 125；
- (3) 如果字符串仅包含指数型十进制值，则转化为十进制值结果，例如 "1.5e3" 转化为 1500；
- (4) 如果字符串仅包含十六进制值，则转化为十进制值结果，例如 "0x00e3dc" 转化为 58332；
- (5) 如果字符串不包含任何数值（如'a'、'a\_b'、"abc"、"ab c"），则转化为 0；
- (6) 如果字符串混有数值和字符串内容，则从左至右查找第一个不符合数值规则的字符，并将此字符以左的部分转化为数值，例如 "123abc" 转化为 123，"2.86e3abc" 转化为 2860，"1abc2" 转化为 1，"abc123" 转化为 0。

#### 4、字符串转布尔

- (1) 一切非空字符串转化为 True，例如 'a'、'Hello'、"123"、"0"；
- (2) 空字符串 (" 或 "") 转化为 False。

#### 5、布尔转数值

- (1) True 转化为 1；  
 (2) False 转化为 0。

## 6、布尔转字符串

- (1) True 转化为 "True"；  
 (2) False 转化为 "False"。

# 1.4 顺序结构

顺序结构是最简单、最常用的程序设计结构。“顺序”，就是把要执行的步骤按照先后次序排列起来，在程序运行后，依次执行它们。例如先将水壶内注水，再把水壶放在燃气灶上，然后点燃燃气灶把水烧开，最后将开水倒入保温瓶中，这就是一个简单的顺序结构。

在使用代码表达顺序结构之前，我们还需要了解一些基本的程序设计知识，本节将引导大家从基础开始，逐步加入程序设计者的行列。

## 1.4.1 基本运算

### 1、算数运算：+、-、\*、/、^

算数运算，一般是指数值之间为了得到一定计算结果的运算。AutoIt v3 中的算数运算与日常生活中的数学运算相似，但个别符号的书写方法不同。常见算术运算符如下表：

算数运算符	名称	作用	范例
+	加法运算符	用于数值之间的和运算	$10+20$ ，值为 30
-	减法运算符	用于数值之间的差运算	$20-10$ ，值为 10
*	乘法运算符	用于数值之间的积运算	$3*5$ ，值为 15
/	除法运算符	用于数值之间的商运算	$18/3$ ，值为 6
^	幂运算符	用于计算数值的几次方	$2^4$ ，值为 16

说明：

(1) 算数运算符的优先级是不同的：幂运算的优先级最高，乘除运算的优先级其次，加减运算的优先级最低；而乘、除运算具有相同的优先级，加、减运算具有相同的优先级。

(2) 当运算的优先级不同时，先进行优先级高的运算，后进行优先级低的运算。例如  $1+3*2$ ，要先计算  $3*2$  的值为 6，后计算  $1+6$  的值为 7。

(3) 当运算的优先级相同时，按照自左至右的顺序运算。例如  $1+3-2$ ，要先计算  $1+3$  的值为 4，后计算  $4-2$  的值为 2。

(4) 可使用英文括号 “(” 与 “)” 来改变运算的优先级。例如  $3*(1+2)$ ，要先计算  $1+2$  的值为 3，后计算  $3*3$  的值为 9。

## 2、连接运算：&

连接运算，一般用于将两个字符串进行连接。

连接运算符“&”，作用是将其左右两侧的字符串连接为一个字符串。例如：“abc”&“def”的运算结果为“abcdef”，‘ab’&‘cd’&‘ef’的运算结果为‘abcdef’

## 3、赋值运算：=

赋值运算，是最常见的运算，作用是将值赋给变量（或常量），其实我们已经接触过多次了。

赋值运算符“=”作用是将其右侧的值赋给其左侧量，例如\$a=100是将100赋值给\$a。如果赋值运算符右侧不是明确的值，则需先得出右侧的运算结果，后将运算结果赋给左侧的量，例如\$sum=1+2是将1+2的运算结果3赋值给\$sum。

值得注意的是，当“=”作为赋值运算符时，与普通数学中的等号有着明显的区别。赋值运算符是“将右侧值给左侧量”，而普通数学里的等号则是表明其“左右两侧值相等”，所以在AutoIt v3的代码中不可能出现形如\$a+1=\$b的形式，这是误把赋值运算与普通数学相混淆造成的错误。

## 4、自赋值运算：+=、-=、\*=、/=、&=

在学习自赋值运算之前，我们先来看一小段代码：

```
Dim $a = 2
$a = $a + 1
```

有没有同学感觉这段代码是错的？如果你是这类同学，那么证明你对刚刚学的赋值运算符没有了解透彻。首先要确定的是，这段代码是正确的，而且变量\$a的最终值为3。其实关键在于怎样理解\$a=\$a+1，根据赋值运算符的规则，我们要先计算“=”右侧\$a+1值，由于\$a在定义时被赋值为2，那么\$a+1等同于2+1，则运算结果为3，即“=”右侧值为3，然后将3赋值给“=”左侧的\$a，则\$a的值为3。

其实刚刚\$a所完成的运算，是一种自赋值运算。所谓自赋值运算，即变量自身参与运算，再将运算结果赋给变量自身。在实际编程中，自赋值运算非常常见，特别是在循环结构中，自增、自减运算极其普遍。下面我们来看一下常见的自赋值运算符：

运算符	名称	作用
+=	自增运算符	运算符左侧值与右侧值相加后赋值给左侧量
-=	自减运算符	运算符左侧值与右侧值相减后赋值给左侧量
*=	自乘运算符	运算符左侧值与右侧值相乘后赋值给左侧量
/=	自除运算符	运算符左侧值与右侧值相除后赋值给左侧量
&=	自连接运算符	运算符左侧值与右侧值连接后赋值给左侧量

说明：

- (1)总的来说，即是将“左侧”与“右侧”运算后再赋值给“左侧”，所以不难理解。
- (2)以自增运算举例，假设\$a=1，则\$a+=1等同于\$a=\$a+1，\$a的最终值为2。自减、自乘、自除类同。而自增、自减运算非常常用。
- (3)自连接运算举例，假设\$s='ab'，则\$s&='cd'等同于\$s=\$s&'cd'，\$s的最终值为'abcd'。

(4) 如果自赋值运算符的右侧不是确定的值，应先求出右侧的值，再与左侧形成自赋值运算。假设 \$a=2，则 \$a\*=3+1，应先计算出 3+1 的值为 4，再计算 2\*4 的值为 8，最终将 8 赋值给 \$a。换句话说，\$a\*=3+1 等同于 \$a=\$a\*(3+1)，而不是 \$a=\$a\*3+1。

### 5、运算与变量类型的自动转化

之前已经提到过不同类型数据间的转化，不同类型数据间的转化大多数是根据运算类型而自动转化的。例如：

(1) `100 + "200"`，算术运算符 “+” 的左右两侧均要求为数值型数据，那么字符串“200”会自动转化为数值 200，而后计算 `100 + 200`，值为 300。

(2) `"abc" & 123`，连接运算符 “&” 的左右两侧均要求为字符串型数据，那么数值 123 会自动转化为字符串“123”，而后对“abc”和“123”进行连接，值为“abc123”。

所以，如果要确定数值是怎样转化的，首先是要了解运算符要求的数据是什么类型。一般而言，运算符要求什么样的数据类型，数据就会朝什么类型进行转化。

同样，有些要求布尔运算的场合，如条件选择语句（将在后续章节讲述），数值和字符串则会自动转化为布尔类型值。

### 1.4.2 基本输入输出

AutoIt v3 是支持创建带有图形用户界面 (Graphical User Interface, GUI) 的脚本程序的，可以通过 GUI 与用户形成良好的交互。但在编程学习初期，应将精力集中于实现基本算法上（这就像在初学骑自行车时应将精力集中于掌握平衡上一样），所以在初级阶段我们只使用基本的 `InputBox` 和 `MsgBox` 作为输入输出媒介。

#### 1、数据输入：`InputBox`

`InputBox`（输入框）作为将数据输入到程序的基本方法之一，其语法较为复杂，为了不过多干扰现阶段的学习重点，我们只使用 `InputBox` 函数的部分功能，完整功能将在结束初级阶段后学习。

**基本语法：**

```
InputBox("标题", "提示")
```

**运行效果：**



(图 1-10)

**参数说明：**

标题：对话框顶部的标题文字，例如上图中的“我是标题”。

提示：对话框内、输入框上方文字，例如上图中的“我是提示”。

(请根据实际需要合理安排标题和提示内容)

#### 返回值：

成功：返回用户输入的值。

失败：返回空字符串。

(所谓返回值，即函数运行结束时返回的值)

#### 举例：

```
Dim $r = InputBox("姓名采集", "请输入您的姓名")
```



(图 1-11)

运行时，InputBox 函数会产生一个标题为“姓名采集”、提示为“请输入您的姓名”的对话框。当输入姓名，按下 OK 按钮后，InputBox 函数的返回值为刚才输入的姓名；按下 Cancel、关闭对话框等其他情况，InputBox 函数的返回值为空字符串。而无论 InputBox 函数的返回值是什么，都会将返回值赋值给变量 \$r，换句话说，变量 \$r 接受了由 InputBox 函数返回的值。

## 2、数据输出：MsgBox

MsgBox（消息框）作为数据输出给用户的基本方法之一，其语法较为复杂，为了不过多干扰现阶段的学习重点，我们只使用 MsgBox 函数的部分功能，完整功能将在结束初级阶段后学习。

#### 基本语法：

```
MsgBox(0, "标题", "文本")
```

#### 运行效果：



(图 1-12)

#### 参数说明：

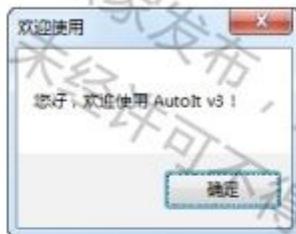
标题：对话框顶部的标题文字，例如上图中的“我是标题”。

文本：对话框内的文字，例如上图中的“我是文本”。

(请根据实际需要合理安排标题和文本内容)

#### 举例：

```
MsgBox(0, "欢迎使用", "您好，欢迎使用 AutoIt v3 !")
```



(图 1-13)

### 1.4.3 代码注释与代码换行

#### 1、代码注释

代码注释，即通过自然语言文字描述代码的含义。大多数时候，注释用于描述一行或一段代码的意图、描述函数的功能和参数意义等，从而使代码所表达的意图清晰可见。在代码测试时，也会使用注释暂时屏蔽部分代码功能，以达到测试的目的。

注释在编译时会被编译器自动忽略掉，所以注释不会被带入编译后的程序，同样不会影响代码的正确执行或增加编译后可执行文件的体积。

代码注释具有非常重要的地位。在平时编程工作中，经常有写了一段代码，隔一段时间就会变得生疏的情况，而注释可以有效的帮助回忆起当时的设计意图。而在多人协作的开发模式中，将代码交给别人维护或由多人共同维护代码时，注释有利于一个人了解另一个人的代码设计意图。所以，建议初学者养成编写代码注释的习惯。

在 AutoIt v3 中，对于单行或少量行的注释，可以使用英文分号（“;”）来表示。一行代码中自英文分号起之后的部分，都会被认为是注释。例如：

```
Dim $a=1 ;定义变量$a，并将其赋值为 1
Dim $b=2 ;定义变量$b，并将其赋值为 2
```

或

```
;定义变量$a，并将其赋值为 1
Dim $a=1
;定义变量$b，并将其赋值为 2
Dim $b=2
```

如果是想表明下列代码段的含义，可以在代码段的开头写多行注释，例如：

```
;如下代码段的功能：
;(1) 定义变量$a 和$b，分别赋值为 1 和 2
;(2) 定义变量$c，并赋值为$a 和$b 的加和
;(3) 使用 MsgBox 输出$c 的值
```

```
Dim $a = 1, $b = 2
Dim $c = $a + $b
MsgBox(0, '求和', '和为：' & $c)
```

而对于多行注释，更简单的方法是在注释文本的前一行写 #comments-start ( 可简写为 #cs ) , 后一行写 #comments-end ( 可简写为 #ce ) , 这样 #cs 和 #ce 之间的文本就会被认为是注释，例如：

```
#cs
如下代码段的功能：
(1) 定义变量 $a 和 $b，分别赋值为 1 和 2
(2) 定义变量 $c，并赋值为 $a 和 $b 的加和
(3) 使用 MsgBox 输出 $c 的值
#ce

Dim $a = 1, $b = 2
Dim $c = $a + $b
MsgBox(0, '求和', '和为：' & $c)
```

## 2. 代码换行

有时一行代码太长，会使阅读和修改的困难。而 AutoIt v3 的表达式或语句是以行为单位的，不能随随便便的换行。

换行需要使用英文下划线（“\_”）作为换行标记，例如：

```
MsgBox(0, '测试即将开始', '运行本测试前，建议关闭其他正在运行的程序，以免造成测试不准确！')
```

上例中，一行中字符串长度太长，可使用换行标记进行换行：

```
 MsgBox(0, '测试即将开始',_
    '运行本测试前，&_
    '建议关闭其他正在运行的程序，' &_
    '以免造成测试不准确！')
```

注意：

- (1) 换行符不能切断表达式中的关键词或函数名等，例如不能切断“MsgBox”，否则会报错；
- (2) 换行符不能切断数值，例如 1200，不能第一行写 12，第二行写 00；
- (3) 换行符不能切断字符串，但可将长字符串切割成数个短字符串，以&进行连接，再进行切断换行，例如上例中 MsgBox 的文本部分。
- (4) 注意，代码的换行并不代表着文本的换行。例如上述两例代码运行的结果是一致的：



(图 1-14)

### 1.4.4 顺序结构程序举例

#### 1、输入圆的半径，自动求圆的面积。

算法分析：

- (1) 假设圆的半径为 r，则圆的面积为： $\pi r^2$ 。

(2) 可使用 InputBox 输入半径，而后用 MsgBox 输出结果。

算法流程图：



(图 1-15)

程序代码：

```
; 定义π（因为π的值为固定的，所以定义为常量）  
Const $PI = 3.1415926  
  
; 输入圆的半径  
Dim $r = InputBox('半径', '请输入圆的半径（单位：厘米）')  
  
; 计算圆的面积，赋值给$s  
Dim $s = $PI * $r ^ 2  
  
; 输出圆的面积  
MsgBox(0, '面积', '圆的面积为：' & $s & '平方厘米')
```

运行图例：



(图 1-16)



(图 1-17)

2、变量\$a 的值为 1，变量\$b 的值为 2，交换两变量的值。

**算法分析：**

- ( 1 ) 不能直接更换两变量的值，形如 \$a=\$b 的代码会导致 \$b 的值直接覆盖 \$a 的值。
- ( 2 ) 那么，交换两变量的值，需要有一个中间变量。
- ( 3 ) 使用 MsgBox 输出结果。

**算法流程图：**

( 图 1-18 )

**程序代码：**

```

;定义$a 和$b，并分别赋初值
Dim $a = 1, $b = 2

;定义中间变量
Dim $t

;将$a 的值赋值给$t
$t = $a
;将$b 的值赋值给$a，此时$a 的值已为$b 的值
$a = $b
;将$t ( 原$a ) 的值赋值给$b,此时$b 的值已为$a 的值
  
```

```
$b = $t  
;  
显示交换后$a 和$b 的值  
MsgBox(0, '交换两变量值', '变量$a 的值为' & $a & ' ' & '变量$b 的值为' & $b)
```

运行图例：



(图 1-19)

其他疑惑：

有同学可能不太明白形如 '变量\$a 的值为' 这类字符串的意思，字符串中怎么会有个“变量”，字符串中的“变量”会怎样输出？

其实如果你认为 '变量\$a 的值为' 中的 \$a 是变量，就已经错了。回顾字符串的定义，字符串是以一对英文引号包含起来的一个或多个字符，所以，只要是英文引号内的部分就都是字符，不具备特殊含义。换句话说，上述字符串中的 "\$" 是一个字符、“a”也是一个字符，它俩就算拼在一起充其量就是俩字符，虽然很像是一个变量，但却不具备变量的意义。

这样解释，大家明白了吗？

## 1.5 选择结构

日常生活中，我们往往会根据实际条件来改变行事方法，例如：如果天气冷了，那么要多穿衣服。程序执行也是如此，往往需要根据条件来改变执行的步骤。而这种根据条件来选择执行哪条分支的结构，称之为选择结构。

### 1.5.1 “真”“假”运算

是否满足条件，是选择哪条分支的重要依据。因算法要求具有确定性（见 1.2.5 节），所以是否满足条件的结果只有“真”或“假”，即布尔值 “True” 或 “False”。在程序设计中，我们一般通过关系运算和逻辑运算来判断条件是否成立，得出 True 或 False。

#### 1、关系运算：=、==、<>、>、>=、<、<=

关系运算（比较运算），一般是将两个值进行比较，并得出比较的结果。常见关系运算符如下表：

关系运算符	名称	作用
-------	----	----

=	等于	判断两个值是否相等
==	绝对等于	判断两个值是否相等(比较字符串时区分大小写)
<>	不等于	判断两个值是否不等于
>	大于	判断左侧值是否大于右侧值
>=	大于等于	判断左侧值是否大于等于右侧值
<	小于	判断左侧值是否小于右侧值
<=	小于等于	判断左侧值是否小于等于右侧值

当关系运算符两侧的值满足关系运算所规定的条件时，结果为 True，否则为 False。

例如：(假设已定义变量\$a 和\$b)

- (1) \$a 值为 10、\$b 值为 10 时，\$a==\$b 的结果为 True
- (2) \$a 值为 15、\$b 值为 12 时，\$a>\$b 的结果为 True
- (3) \$a 值为 21、\$b 值为 30 时，\$a>=\$b 的结果为 False
- (4) \$a 值为 19、\$b 值为 11 时，\$a<\$b 的结果为 False
- (5) \$a 值为 13、\$b 值为 13 时，\$a<=\$b 的结果为 True
- (6) \$a 值为"abc"、\$b 值为"ABC"时，\$a==\$b 的结果为 True
- (7) \$a 值为"abc"、\$b 值为"ABC"时，\$a==\$b 的结果为 False

## 2. 逻辑运算：And、Or、Not

逻辑运算，一般是判断两个布尔值可以达成什么样的结果。常见的逻辑运算符如下表：

逻辑运算符	名称	作用
And	与	当两侧值均为 True 时，结果为 True，否则为 False
Or	或	当两侧值均为 False 时，结果为 False，否则为 True
Not	非	非 True 结果为 False，非 False 结果为 True

例如：(假设已定义变量\$a 和\$b)

\$a	\$b	\$a And \$b	\$a Or \$b	Not \$a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

AutoIt v3 中的 And 和 Or，使用时还需注意：

- (1) 表达式 1 And 表达式 2，如果表达式 1 的结果为 False，则整体结果立即为 False。
  - (2) 表达式 1 Or 表达式 2，如果表达式 1 的结果为 True，则整体结果立即为 True。
- 这两种条件下表达式 2 都不会被运行，因为表达式 1 的结果已足够决定整体的运行结果。

### 1.5.2 运算的优先级

当一个语句中出现多个运算符时，其结合的先后顺序是由运算符的优先级来控制的。其实在之前

## Let's AutoIt Plus

我们已经接触过算术运算符的优先级了(先乘除、后加减),现在并入我们刚学习的关系运算符和逻辑运算符,看看它们的优先级吧。

在 AutoIt v3 中,运算符的优先级如下:

优先级	运算符
高	Not
	^
	* , /
	+ , -
	&
	= , == , <> , < , <= , > , >=
	And , Or

上表中,从上到下按照优先级从高到低的顺序进行排列,越靠上则优先级越高。同一行内的运算符优先级是相同的。

- (1) 同一表达式或语句中,运算符优先级不同时,优先级高的先运算,优先级低的后运算;
- (2) 同一表达式或语句中,运算符优先级相同时,按照从左到右的顺序依次进行运算。

### 1.5.3 If 语句

#### 1、If.....Then.....EndIf

If.....Then.....EndIf 语句是最基本的条件选择语句。

**语法 :**

```
If <条件> Then  
    <语句段>  
EndIf
```

**功能 :**

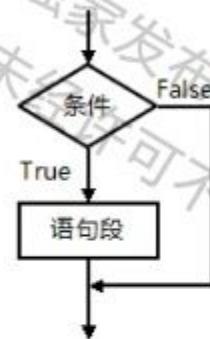
如果满足条件,则执行语句段

**说明 :**

(1) <条件>,一般是逻辑或关系表达式,当条件为真时,执行<语句段>。

(2) <语句段>,由一条或多条语句组成,多条语句需要分多行书写。

**流程图 :**



(图 1-20)

**举例：**

输入一个数值，当数值大于 0 时给出提示。

```

Dim $Num = InputBox("输入数值", "请输入一个数值：")
If $Num > 0 Then
    MsgBox(0, "判断数值是否大于 0", "您输入的是一个大于 0 的数")
EndIf

```

其实，当语句段只有一条语句时，可以使用简写模式。

**语法：**

```
If <条件> Then <语句>
```

这种写法与基本写法的区别有两点：

- (1) 条件满足时，仅能执行一条语句；
- (2) 整条语句要写在同一行里，不能分行写。

**举例：**

```

Dim $Num = InputBox("输入数值", "请输入一个数值：")
If $Num > 0 Then MsgBox(0, "判断数值是否大于 0", "您输入的是一个大于 0 的数")

```

运行步骤和结果与刚才的例子相同。

**2、If.....Then.....Else.....EndIf**

If.....Then.....Else.....EndIf 语句是一种基本的双分支语句。

**语法：**

```

If <条件> Then
    <语句段 1>
Else
    <语句段 2>
EndIf

```

**功能：**

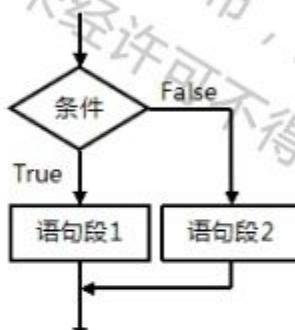
如果满足条件，则执行语句段 1，否则执行语句段 2。

**说明：**

- (1) <条件>，一般是逻辑或关系表达式，当条件为真时，执行对应 <语句段>。

(2) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。

流程图：



(图 1-21)

举例：

输入一个数，当数值大于 0 时、小于等于 0 时分别给出不同的提示。

```
Dim $Num = InputBox('输入数值', '请输入一个数值：')
If $Num > 0 Then
    MsgBox(0, '判断数值是否大于 0', '您输入的是一个大于 0 的数')
Else
    MsgBox(0, '判断数值是否大于 0', '您输入的是一个小于或等于 0 的数')
EndIf
```

### 3、If.....Then.....ElseIf.....Then.....EndIf

If.....Then.....ElseIf.....Then.....EndIf 语句是一种多分支语句。

语法：

```
If <条件 1> Then
    <语句段 1>
ElseIf <条件 2> Then
    <语句段 2>
...
ElseIf <条件 n> Then
    <语句段 n>
[Else
    <语句段 n+1>]
EndIf
```

功能：

如果满足条件 1，则执行语句段 1，否则如果满足条件 2，则执行语句段 2，……，否则如果满足条件 n，则执行语句段 n，否则执行语句段 n+1。

说明：

- (1) <条件>，一般是逻辑或关系表达式，当条件为真时，执行对应<语句段>。
- (2) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。
- (3) 最后的 Else 和<语句段 n+1>可选。

**流程图：**



(图 1-22)

**举例：**

输入一个数，当数值大于 0 时、等于 0、小于 0 时分别给出不同的提示。

```

Dim $Num = InputBox('输入数值', '请输入一个数值：')
If $Num > 0 Then
    MsgBox(0, '判断数值是否大于 0', '您输入的是一个大于 0 的数')
ElseIf $Num = 0 Then
    MsgBox(0, '判断数值是否大于 0', '您输入的是一个等于 0 的数')
Else
    MsgBox(0, '判断数值是否大于 0', '您输入的是一个小于 0 的数')
EndIf
  
```

## 1.5.4 Select 语句与 Switch 语句

### 1、Select.....Case.....EndSelect

Select.....Case.....EndSelect 语句是专门设计用于多分支的语句。

**语法：**

```

Select
  Case <条件1>
    <语句段 1>
  Case <条件 2>
    <语句段 2>
  .....
  Case <条件 n>
  
```

```
<语句段 n>
[Case Else
    <语句段 n+1>]
EndSelect
```

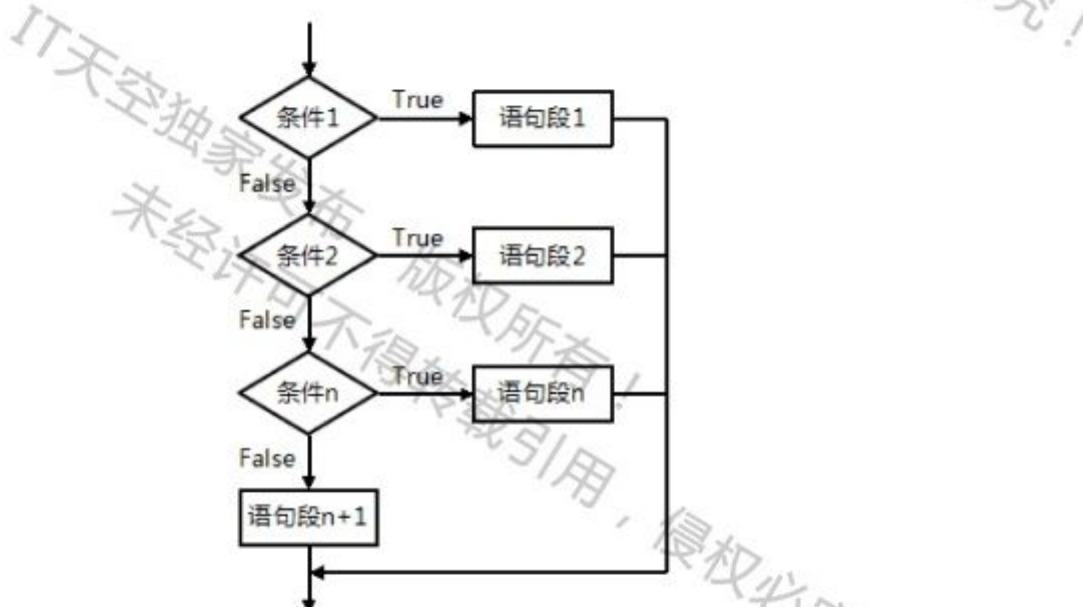
### 功能：

如果满足条件 1，则执行语句段 1，如果满足条件 2，则执行语句段 2……，如果满足条件 n，则执行语句段 n，都不满足则执行语句段 n+1。

### 说明：

- (1) <条件>，一般是逻辑或关系表达式，当条件为真时，执行对应<语句段>。
- (2) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。
- (3) 最后的 Case Else 和<语句段 n+1>可选。

### 流程图：



(图 1-23)

### 举例：

输入学生的成绩，85 分以上输出 “A”，75~84 分输出 “B”，60~74 分输出 “C”，60 分以下输出 “D”。

```
Dim $Score = InputBox('输入得分', '请输入学生的得分 (1~100):')
Select
    Case $Score >= 85 And $Score <= 100
        MsgBox(0, '得分等级', 'A')
    Case $Score >= 75 And $Score <= 84
        MsgBox(0, '得分等级', 'B')
    Case $Score >= 60 And $Score <= 74
        MsgBox(0, '得分等级', 'C')
    Case Else
```

```
    MsgBox(0, '得分等级', 'D')
EndSelect
```

**注意：**

(1) 各个条件之间，是按照自上到下的匹配顺序进行匹配的，即靠上的条件优先匹配。

(2) 当满足某条件时，则执行条件对应的语句段，执行完毕后自动跳出 Select 语句。

基于此，可以简化上例的写法：

```
Dim $Score = InputBox('输入得分', '请输入学生的得分 (1~100):')
Select
    Case $Score >= 85
        MsgBox(0, '得分等级', 'A')
    Case $Score >= 75
        MsgBox(0, '得分等级', 'B')
    Case $Score >= 60
        MsgBox(0, '得分等级', 'C')
    Case Else
        MsgBox(0, '得分等级', 'D')
EndSelect
```

假设输入成绩为 90 分，则满足大于等于 85 分的条件，执行输出 “A”。虽然 90 分实际上讲也满足大于等于 75、大于等于 60 的条件，但因为先满足了大于等于 85 的条件，执行并跳出了 Select 语句，所以不会再向下匹配 75、60 这两个条件了。

即便如此，仍建议大家尽可能的将条件写的完善，毕竟程序除了执行效率外，可读性和可维护性同样重要。

## 2、Switch.....Case.....EndSwitch

Switch.....Case.....EndSwitch 语句也是专门设计用于多分支的语句。与 Select 语句不同的是，Switch 语句是针对同一个表达式的不同结果进行分支判断的。

**语法：**

```
Switch <表达式>
    Case <值 1>
        <语句段 1>
    Case <值 2>
        <语句段 2>
    ...
    Case <值 n>
        <语句段 n>
    [Case Else
        <语句段 n+1>]
EndSwitch
```

**功能：**

先得出表达式的值，后将表达式的值与预定的值依次匹配，匹配成功则执行对应的语句段。

**说明：**

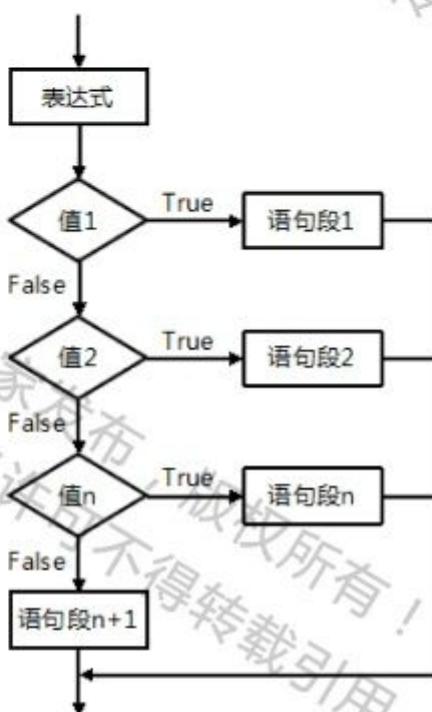
(1) <表达式>，理论上说允许各种表达式，但必须可以得出确定的值。

(2) <值>，预定的值，表达式的值与哪一个预定的值相匹配，则执行对应的语句段。如果有多个值对应同一语句段，则可以使用英文逗号（“,”）进行分隔，例如“1,2,3,4,5”；而如果值是一个连续的区间，则可使用关键词“To”进行连接，例如“1 To 5”。

(3) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。

(4) 最后的 Case Else 和<语句段 n+1>可选。

流程图：



(图 1-24)

举例：

输入学生的成绩，85 分以上输出 “A”，75~84 分输出 “B”，60~74 分输出 “C”，60 分以下输出 “D”。

```
Dim $Score = InputBox('输入得分', '请输入学生的得分 ( 1~100 ) : ')
Switch $Score
    Case 85 To 100
        MsgBox(0, '得分等级', 'A')
    Case 75 To 84
        MsgBox(0, '得分等级', 'B')
    Case 60 To 74
        MsgBox(0, '得分等级', 'C')
    Case Else
        MsgBox(0, '得分等级', 'D')
EndSwitch
```

注意：

- (1) 各个值之间，是按照自上到下的匹配顺序进行匹配的，即靠上的条件优先匹配。  
 (2) 当满足某值时，则执行值对应的语句段，执行完毕后自动跳出 Switch 语句。

### 1.5.5 选择结构的嵌套

所谓“嵌套”，即在一种语句类型中又使用了相同类型的语句，例如在 If 语句的语句段中又使用了 If 语句。举个例子：

输入一个数值，判断其与 0 的大小关系。若大于 0 则进一步判断数值与 100 的大小关系，若小于 0 则进一步判断数值与 -100 的大小关系。

```
Dim $Num = InputBox('输入数值', '请输入一个数值(例如 15、-31 等):')
If $Num > 0 Then
    If $Num > 100 Then
        MsgBox(0, '数值大小判断', '您输入的数大于 0，且大于 100')
    Else
        MsgBox(0, '数值大小判断', '您输入的数大于 0，但小于等于 100')
    EndIf
Else
    If $Num > -100 Then
        MsgBox(0, '数值大小判断', '您输入的数小于 0，但大于 -100')
    Else
        MsgBox(0, '数值大小判断', '您输入的数小于等于 0，且小于等于 -100')
    EndIf
EndIf
```

如上例所示，我们先在一个 If 语句中判断了数值与 0 的大小关系，而后在第一个 If 语句的两条分支中，各使用了第二个 If 语句来判断数值与 100、-100 的大小关系。

除了 If 语句，Select、Switch 等语句同样可以进行嵌套，甚至可以混合嵌套（例如 Select 语句中嵌套 If 语句等），嵌套方法与 If 语句的嵌套类同，所以这里不再赘述，请大家自行理解。

顺道一提的是，当代码逐步复杂起来后，代码的合理“缩进”是十分必要的，特别是在多重嵌套时。以 If 语句举例，Else 和 EndIf，只和最近的没有配对的 If 进行配对。换句话说，配对关系具有层级性，即第一层的 Else 和 EndIf 与第一层的 If 进行配对，而第二层的 Else 和 EndIf 与第二层的 If 进行配对。在这种多层模式下，缩进可以直观的明确层级关系，加强程序的可读性，减少不必要的错误。

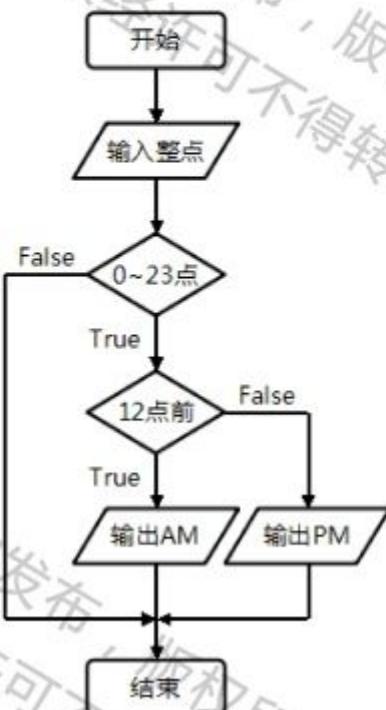
缩进一般使用 Tab 进行，简单说可以理解为“几个 Tab 则是第几层”。养成良好的代码缩进习惯，可有效提高程序书写与维护效率，减少匹配关系错误和不必要的麻烦。

### 1.5.6 选择结构程序举例

#### 1、输入一个整点时间，判断其属于 AM 或是 PM

算法分析：

- (1) 一般而言，AM 和 PM 以 12 点进行分割，0:00~11:59 为 AM，12:00~23:58 为 PM。  
(2) 分支较少，使用 If 语句即可解决。



(图 1-25)

**程序代码：**

```
;输入整点时间
Dim $Hour = InputBox("输入时间", "现在是几点 ( 0~23 ) ? : ")
;输入的整点时间在 0~23 的范围内
If $Hour >= 0 And $Hour <= 23 Then
    If $Hour < 12 Then
        ;12 点之前为 AM
        MsgBox(0, '时段', 'AM')
    Elseif $Hour >= 12 Then
        ;12 点之后为 PM
        MsgBox(0, '时段', 'PM')
    Endif
Endif
```

**运行图例：**



(图 1-26)

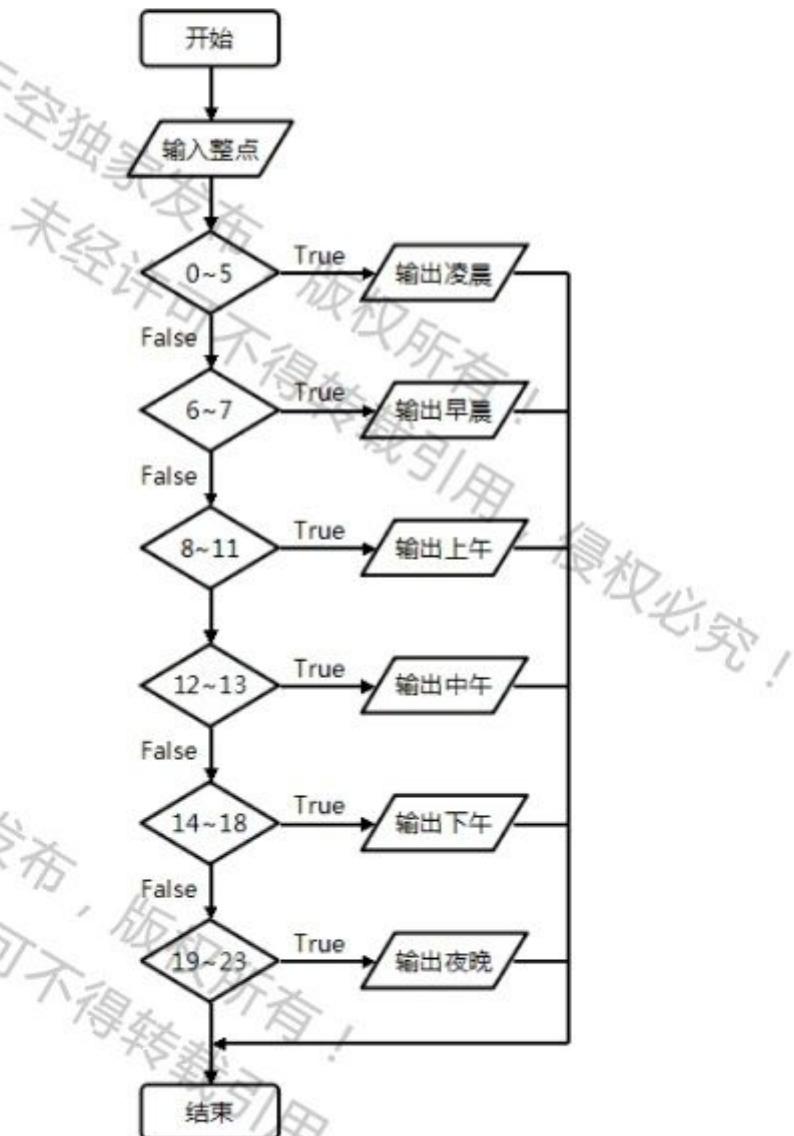


(图 1-27)

## 2、输入一个整点时间，判断其属于凌晨、早晨、上午、中午、下午或是夜晚

算法分析：

基于单一值（整点时间）的多分支判断，使用 Switch 语句解决非常简便。



(图 1-28)

## 程序代码：

```
;输入整点时间  
Dim $Hour = InputBox('输入时间', '现在是几点 ( 0~23 ) ? : ')  
  
;判断整点时间  
Switch $Hour  
    Case 0 To 5  
        ;0~5 点，凌晨  
        MsgBox(0, '时段', '凌晨')  
    Case 6 To 7  
        ;6~7 点，早晨  
        MsgBox(0, '时段', '早晨')  
    Case 8 To 11  
        ;8~11 点，上午  
        MsgBox(0, '时段', '上午')  
    Case 12 To 13  
        ;12~13 点，中午  
        MsgBox(0, '时段', '中午')  
    Case 14 To 18  
        ;14~18 点，下午  
        MsgBox(0, '时段', '下午')  
    Case 19 To 23  
        ;19~23 点，夜晚  
        MsgBox(0, '时段', '夜晚')  
EndSwitch
```

## 运行图例：

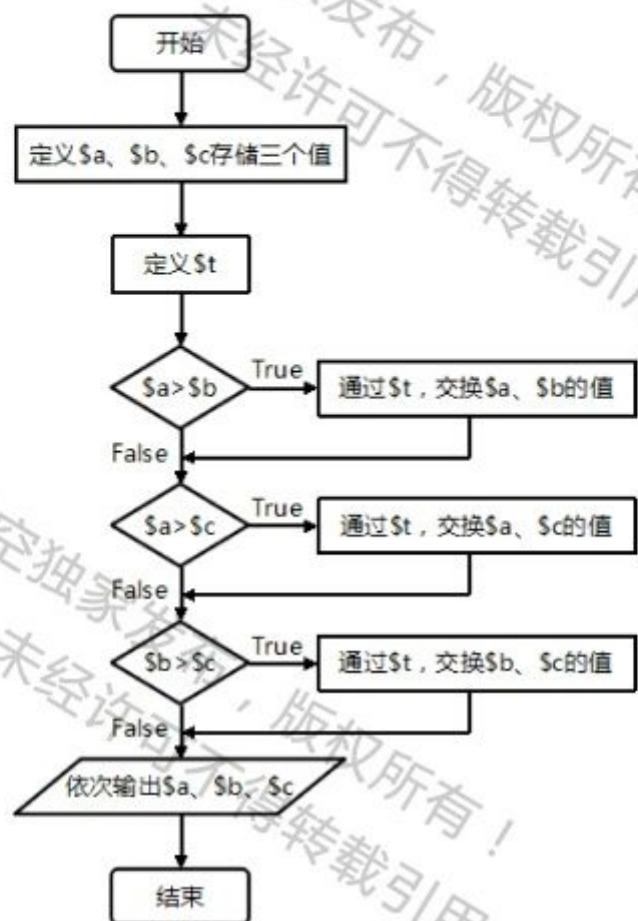


## 3、将三个数值按照从小到大的顺序进行排列

### 算法分析：

- ( 1 ) 假设有三个位置 ( 一号位、二号位、三号位 ) 分别放置三个数；
- ( 2 ) 将一号位与二号位上的数相比较，将较小的放置到一号位，较大的放置到二号位；
- ( 3 ) 再将一号位与三号位上的数相比较，将较小的放置到一号位，较大的放置到三号位；
- ( 4 ) 至此，一号位上的数是三个位置上最小的；
- ( 5 ) 然后二号位与三号位上的数相比较，将较小的放置到二号位，较大的放置到三号位；
- ( 6 ) 至此，一号位、二号位、三号位上的数，按照从小到大的顺序排列完成。

算法流程图：



(图 1-31)

程序代码：

```
; 定义 $a、$b、$c，并分别赋值
Dim $a = 15, $b = 8, $c = 11
```

```
; 定义 $t 作为中间变量
Dim $t
```

```
; 如果 $a 大于 $b，则交换 $a 和 $b 的值
```

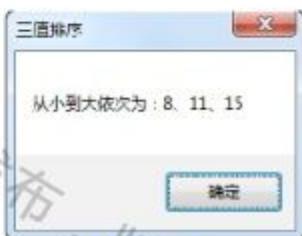
```
If $a > $b Then
    $t = $a
    $a = $b
    $b = $t
EndIf
```

```
; 如果 $a 大于 $c，则交换 $a 和 $c 的值
```

```
If $a > $c Then
    $t = $a
```

```
$a = $c  
$c = $t  
EndIf  
  
;如果$b 大于$c，则交换$b 和$c 的值  
If $b > $c Then  
    $t = $b  
    $b = $c  
    $c = $t  
EndIf  
  
;输出排序后的三个值  
MsgBox(0, '三值排序', '从小到大依次为 : ' & $a & ', ' & $b & ', ' & $c)
```

运行图例：



(图 1-32)

## 1.6 循环结构

日常生活中，我们常常会做一些具有重复性的事情，例如折叠 100 只千纸鹤。在折叠的过程中，每次折叠的步骤是相同的，而当折叠到 100 只时停止折叠。

在程序设计中，当需要反复的按照既定规则执行某段语句，而直到达到某种条件时才终止，应当使用循环结构。循环结构可以有效减少重复性工作，是尤为重要的一种程序控制结构。

### 1.6.1 While 语句与 Do-Until 语句

#### 1、While.....WEnd

While.....WEnd 循环，又被称作“当”型循环，是一种基本的循环结构。

语法：

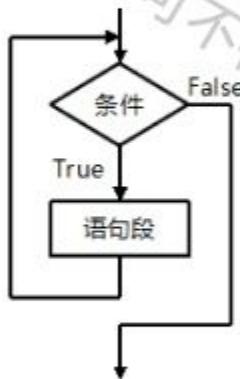
```
While <条件>  
    <语句段>  
WEnd
```

功能：

当满足条件时，反复执行语句段。（这也是 While.....WEnd 被称为“当”型循环的原因）

**说明：**

- (1) <条件>，一般是逻辑或关系表达式，为真时反复执行<语句段>。  
 (2) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。

**流程图：**

(图 1-33)

**举例：**

依次输出：1、2、3、4、5

```

Dim $i = 1
While $i <= 5
    MsgBox(0, '输出', $i)
    $i += 1
WEnd
  
```

**2、Do.....Until**

Do.....Until 循环，又被称作“直到”型循环，与 While 一样是一种基本的循环结构。

**语法：**

```

Do
    <语句段>
Until <条件>
  
```

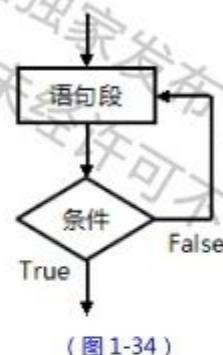
**功能：**

反复执行语句段，直到满足条件。（这也是 Do.....Until 被称为“直到”型循环的原因）

**说明：**

- (1) <条件>，一般是逻辑或关系表达式，为真时结束循环，否则反复执行<语句段>。  
 (2) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。

**流程图：**



举例：

依次输出：1、2、3、4、5

```
Dim $i = 1
Do
    MsgBox(0, '输出', $i)
    $i += 1
Until $i > 5
```

## 1.6.2 For 语句

For...To...Step...Next 循环，是一种极常用的循环结构，使用频率非常高。

语法：

```
For <变量>=<初值> To <终值> [Step <步进值>]
    <语句段>
Next
```

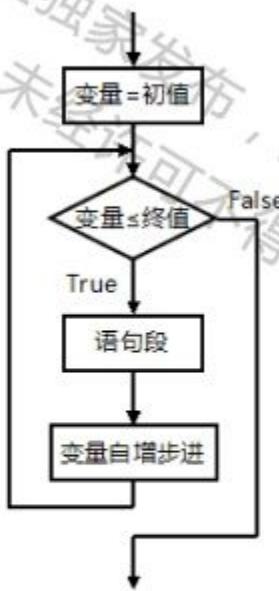
功能：

变量由初值开始每执行一次语句段就自增一个步进值，变量到达或超过终值时循环结束。

说明：

- (1) <变量>，就是一个普通变量，但一般与<语句段>中的代码有一定关系。
- (2) <初值>、<终值>、<变量>的最初值和最终值。
- (3) <步进值>，每次<语句段>执行完毕后，<变量>会自增一个<步进值>。
- (4) <语句段>，由一条或多条语句组成，多条语句需要分多行书写。
- (5) 另外，当<步进值>为 1 时，可省略 Step 和<步进值>部分不写。

流程图：



(图 1-35)

**举例：**

依次输出：1、2、3、4、5

```

Dim $i
For $i = 1 To 5
    MsgBox(0, '输出', $i)
Next
  
```

### 1.6.3 循环结构的嵌套

所谓“嵌套”，即在一种语句类型中又使用了相同类型的语句，例如在 For 语句的语句段中又使用了 For 语句。举个例子：

依次输出：11、12、13、21、22、23、31、32、33

```

Dim $i,$j
For $i=10 To 30 Step 10
    For $j=1 To 3
        MsgBox(0, '输出', $i+$j)
    Next
Next
  
```

通过两层 For...To...Step...Next 循环来实现。

(1) 第一层循环，变量 \$i 从 10 开始，以 10 的步进值递增，达到 30 时结束。则 \$i 的值在第一层循环中依次是 10、20、30。

(2) 第二层循环，变量 \$j 从 1 开始，以 1 为步进值递增，达到 3 时结束。则 \$j 的值在第二层循环中依次是：1、2、3。

(3) 因为第二层循环在第一层循环内，则第一层每次循环时，第二层都要进行完整的循环。例如当 \$i 为 10 时，\$j 会依次为 1、2、3，则输出 \$i+\$j 的加和即可依次得出 11、12、13。

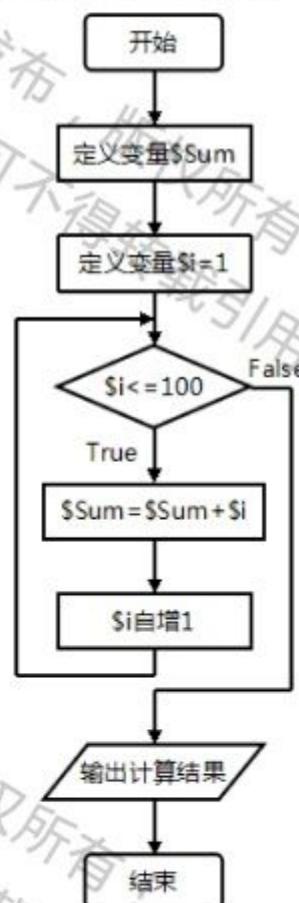
(4) 输出其他值的方式请根据(3)类推。

#### 1.6.4 循环结构程序举例

##### 1、计算 $1+2+3+\dots+99+100$ 的值

算法分析：

- (1) 需要一个变量，例如 \$i，来依次表示 1、2、3、……、99、100。
- (2) 还需要一个变量，例如 \$Sum，来存储加和。
- (3) 那么，使 \$Sum 的初值为 0。而后：
  - 当 \$i 为 1 时，使 \$Sum = \$Sum + \$i (即  $0+1$ )，之后 \$Sum 的值为 1；
  - 当 \$i 为 2 时，使 \$Sum = \$Sum + \$i (即  $1+2$ )，之后 \$Sum 的值为 3；
  - 当 \$i 为 3 时，使 \$Sum = \$Sum + \$i (即  $3+3$ )，之后 \$Sum 的值为 6；
  - ……
- (4) 以此类推，最终得到  $1+2+3+\dots+99+100$  的值。



(图 1-36)

程序代码：

```
; 定义变量$Sum，存储加法计算的和
```

```

Dim $Sum = 0

;定义变量$i，通过循环用来表示 1~100
Dim $i = 1

;当$i<=100 时，进行循环
While $i <= 100
    ;每次循环将$i 的值与当前$Sum 的值相加，并将加和再存入$sum 中
    $Sum = $Sum + $i
    ;每次循环$i 自增 1
    $i = $i + 1
WEnd

;输出计算结果
MsgBox(0, '1~100 的加和', '1+2+3+...+99+100 的和为 : ' & $Sum)

```

运行图例：



( 图 1-37 )

## 2、输出小九九乘法口诀

算法分析：

( 1 ) 使用循环结构解决问题，最先要做的是找到循环的规律，那么先来观察乘法口诀表：

$1 \times 1 = 1$				
$2 \times 1 = 2$	$2 \times 2 = 4$			
$3 \times 1 = 3$	$3 \times 2 = 6$	$3 \times 3 = 9$		
$4 \times 1 = 4$	$4 \times 2 = 8$	$4 \times 3 = 12$	$4 \times 4 = 16$	
$5 \times 1 = 5$	$5 \times 2 = 10$	$5 \times 3 = 15$	$5 \times 4 = 20$	$5 \times 5 = 25$
.....				

( 2 ) 规律 1，乘号前的值，与行号相同，即第一行为 1，第二行为 2，以此类推。

( 3 ) 规律 2，乘号后的值，与列号相同，即第一列为 1，第二列为 2，以此类推。

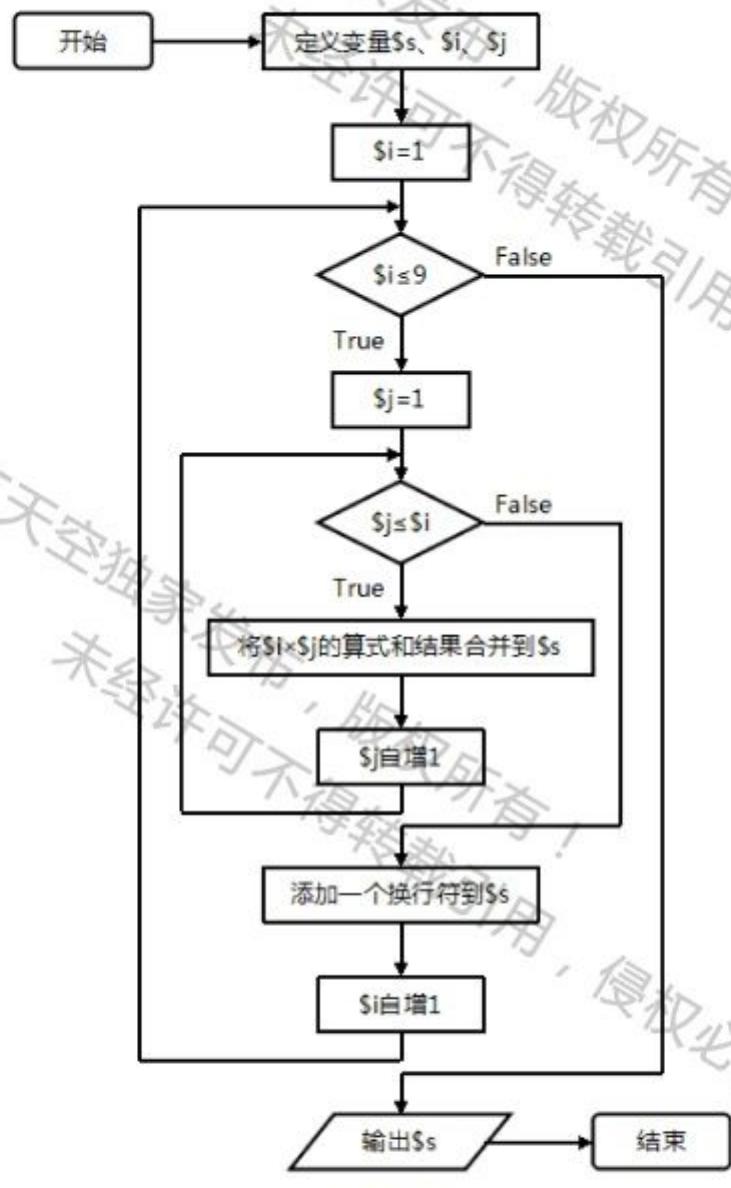
( 4 ) 规律 3，第几行就有几列，第一行有一列，第二行有两列，以此类推。

( 5 ) 假设以变量*\$i* 表示行，变量*\$j* 表示列，那么：

- 第*\$i* 行的第*\$j* 列的算式是  $i \times j$ ，算式结果是  $i \times j$  的值；
- 因小九九口诀有 9 行，则 *\$i* 的初值为 1，终值为 9；
- 第*\$i* 行有 *\$i* 列，则 *\$j* 的初值为 1，终值为当时的 *\$i*。

( 6 ) 每形成一行算式后，要进行换行再形成下一行。字符串换行可使用宏@CRLF 来进行。关于

宏，将在后续章节中详细介绍，本节只要会使用就可以了。



### 程序代码：

```
; 定义变量$s，用于存储要输出的字符串  
Dim $s = ""  
  
; 定义变量$i，用来表示乘号前的 1~9  
; 定义变量$j，用来表示乘号后的 1~9  
Dim $i, $j  
  
; 第一层循环用于表示乘号前的数值  
For $i = 1 To 9
```

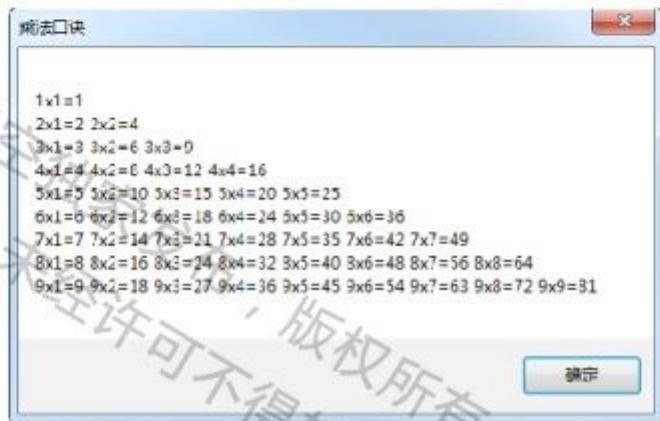
```

;第二层循环用于表示乘号后的数值
For $j = 1 To $i
    ;第二层循环，每次形成一个算式和结果，并合并保存到$s 中
    $s &= $i & 'x' & $j & '=' & $i * $j & ''
Next
;第一层循环即将结束时，本行所有算式已经形成，在行末添加一个换行符号
$s &= @CRLF
Next

;输出乘法口诀字符串
MsgBox(0, '乘法口诀', $s)

```

运行图例：



( 图 1-39 )

### 3、使用总计 20 张面额为 10 元、20 元或 50 元的钞票，组成 600 元

算法分析：

( 1 ) 枚举法，是程序设计中常用的一种方法，即列出所有可能性，并测试每种可能性是否符合条件要求。这个问题适合使用枚举法来解决。

( 2 ) 总计 20 张钞票，则 10 元和 20 元钞票最少可以有 0 张、最多可以有 20 张；而 50 元钞票最少可以有 0 张、最多却只能有 12 张（因为 12 张 50 已经 600 元了）。

( 3 ) 那么我们需要使用三层循环，每层循环表示一种钞票的张数，从而表达所有的组合方式。

( 4 ) 当得到一种组合方式后，需要判定其符合“总计 20 张”和“总计 600 元”两个条件，如果符合，则记录这种组合方法。

( 5 ) 伪代码描述：

```

Dim $s 存储最终输出结果
Dim $i 代表 10 元张数，$j 用于代表 20 元的张数，$k 用于代表 50 元的张数
Dim $n=1 用来为方法编号

For $i=0 张 To 20 张
    For $j=0 张 To 20 张
        For $k=0 张 To 12 张
            If ($i+$j+$k 一共 20 张) And (10 元×$i+20 元×$j+50 元×$k 共 600 元) Then

```

## Let's AutoIt Plus

```
    将“方法$n : $i 张 10 元、$j 张 20 元、$k 张 50 元”合并存入$s 中
EndIf
方法编号$n 自增 1
Next
Next
Next

输出所有方法$s
```

### 程序代码：

```
;定义$s 用于存储最终的输出结果
Dim $s

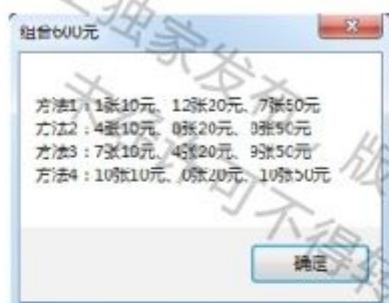
;定义$i 用于代表 10 元的张数
;定义$j 用于代表 20 元的张数
;定义$k 用于代表 50 元的张数
Dim $i, $j, $k

;方法编号
Dim $n = 1

;第一层循环枚举 10 元的张数
For $i = 0 To 20
    ;第二层循环枚举 20 元的张数
    For $j = 0 To 20
        ;第三层循环枚举 50 元的张数
        For $k = 0 To 12
            ;当钞票一共有 20 张且总计面额为 600 元时
            If $i + $j + $k = 20 And 10 * $i + 20 * $j + 50 * $k = 600 Then
                ;将组合方法合并保存到$s 中
                $s &= '方法' & $n & ':' & _
                    $i & '张 10 元、' & _
                    $j & '张 20 元、' & _
                    $k & '张 50 元' & @CRLF
            ;将方法的编号自增 1
            $n += 1
        EndIf
    Next
Next
Next

;输出以总计 20 张 10 元、20 元、50 元组成 600 元的方法
MsgBox(0, '组合 600 元', $s)
```

### 运行图例：



(图 1-40)

## 1.7 Level1 总结

本章的目的，是引领大家进入程序设计的大门。凡事入门难，学习编程也不例外，特别是对于首次接触编程的学习者而言更是如此。但最初的难度，往往来源于对新概念的不熟悉，对算法逻辑的不理解，以及对整体设计思路的无头绪，这类“难度”只是源于对新领域的陌生，多加熟悉就会自然而然的克服。

算法是本章的一个非常重要的点。算法是程序设计的灵魂，使用编程解决问题，就是先将流程转化为算法，再将算法转化为编程语言的过程。由此可见，算法是高于编程语言的一种存在，无论用什么语言实现算法，只不过是语法不同罢了。所以，算法是具有高度的通用性的，不只可以用于 AutoIt v3，还可以用于其他编程语言。

掌握算法之后，我们就学会了怎样以算法的角度分析问题。而后，我们需要学习以怎样的方式和结构来表达问题，所以我们依次学习了语言基础、顺序结构、选择结构和循环结构。

语言基础是简单而重要的一节，很多程序设计的基本概念都在其中。常量和变量是程序设计中最基本的元素，它们用来存储数据，而数据又分为不同类型，不同类型的数据又存在相互转化。而在程序执行中，绝大多数环节都是对数据的处理，所以，掌握如何使用常变量来表达特定类型的数据，是非常重要的一环，不要因其简单而忽略。好的基础是一切的基石。

顺序结构、选择结构和循环结构，是程序设计语言中的三种基本结构。遵循这三种结构，遵守程序的书写规则，才能写出好的程序。顺序结构是最简单，但最普遍的结构，顾名思义，就是一个接一个的执行一条条的语句，这种执行方式常见而普遍，却是不可或缺的重要结构。选择结构形成了分支，可以使程序在不同的条件下执行不同的语句，程序执行的过程中会有千变万化的情况，考虑到每种可能发生的情况，并对每种情况产生应对方案，是在考验编程者的思维全面性。循环结构致力于使用特定规律来实现重复性工作，而如何总结规律，并在头脑中构思出多次循环时数据的变动，是在考验编程者的思维逻辑性。

说到这里，如果您感觉对本章还有哪些没学透彻的地方，还请务必搞清楚、弄明白。本章是基础中的基础，但却是最重要的基础，对本章的稳固掌握，会使后续章节的学习事半功倍，否则将事倍功半。

以伪代码表述如何完成 Level1 的学习：

```
定义变量 $flg, $i  
  
Do  
    使$flg 为 0  
    For $i = 1 To 7  
        If 没学过 1.$i 节 Then  
            学习 1.$i 节  
        Else  
            If 没掌握 1.$i 节 Then  
                复习 1.$i 节  
            EndIf  
        EndIf  
        If 掌握了 1.$i 节 Then $flg 自增 1  
    Next  
Until $flg 的值为 7
```

学好了 Level1 所有章节

看得懂，Level1 就可以毕业了！

## Level 2

## 2.1 函数

### 2.1.1 函数的定义

#### 1、函数的意义

函数 ( Function ) 这个词并不陌生，我们已经多次接触过了，例如 Level1 中我们使用 InputBox 和 MsgBox 函数实现了基本输入输出。

函数是实现特定功能的子模块（例如 Input 用于输入，而 MsgBox 用于输出），是模块化编程思想的重要基石。

一般而言，函数分为标准函数和自定义函数两种，标准函数由 AutoIt v3 本身提供，如 MsgBox 函数和 InputBox 函数；自定义函数由编程者根据实际需求自行书写。本节的主要目的，是与大家共同学习如何书写自定义函数。

#### 2、函数的定义

```
Func <函数名>([参数 1][, 参数 2]...[, 参数 n])
    [语句或语句段]
    [Return <返回值>]
EndFunc
```

说明：

- ( 1 ) 函数必须以 Func 开始，并以 EndFunc 结束；
- ( 2 ) 函数名的命名规则为：“以字母或下划线开头的，字母、下划线、数字组合”，例如 abc、ab12、\_a1b2 均为合法函数名；
- ( 3 ) 函数名应尽可能有意义，可通过函数名推断函数的基本功能，便于后期维护；
- ( 4 ) 同一个程序代码中，函数不可被重复定义，即函数名不可相同；
- ( 5 ) 建议所有自定义函数的函数名以下划线 ( \_ ) 开头，以与标准函数形成区分；
- ( 6 ) 函数可以有一个或多个参数（有参函数），也可以没有参数（无参函数）；
- ( 7 ) 函数功能由一行或多行语句组成，以实现特定功能，但亦可没有语句（空函数）；
- ( 8 ) 一般而言需要为函数指定返回值，亦可不指定。

例 1：

```
Func _TestFunction1($a, $b)
    Dim $c
    $c = $a + $b
    Return $c
EndFunc
```

例 2：

```
Func _TestFunction2()
    MsgBox(0, "", "Hello World!")
EndFunc
```

例 3：

```
Func _TestFunction3()
EndFunc
```

### 3. 函数的执行

AutoIt v3 代码的执行方式，是按照自顶向下的顺序执行，当遇到选择结构、循环结构时则进行既定的跳转或反复。而函数是独立的功能模块，当函数被调用时，脚本即“跳转”至函数部分去执行，执行完毕后再回到被调用的位置，继续向下执行之后的代码段。值得注意的是，函数如果不被调用是不会执行的。

例如：( 暂时看不懂这段代码不要着急，本例只是用来示范函数的执行方法 )

```
Dim $a, $b
$a = 1
$b = 2
Dim $c = _Plus($a, $b)
MsgBox(0, "", $a & "+" & $b & "=" & $c)

Func _Plus($x, $y)
    Dim $sum = $x + $y
    Return $sum
EndFunc
```

本例中，\_Plus 函数用于实现对两个参数的加运算，而后把和返回。代码执行时，在执行到 “Dim \$c = \_Plus(\$a, \$b)” 这行时，代码“跳转”至\_Plus 函数执行，执行完毕后再回到此行，并将返回值赋值给\$c。

而如果函数没有被调用，例如代码只包含一个函数：

```
Func _Plus($x, $y)
    Dim $sum = $x + $y
    Return $sum
EndFunc
```

执行这段代码时不会产生任何结果，就像执行一个没有任何代码的空脚本一样。

## 2.1.2 函数的一般形式

### 1. 有参函数

有参函数，即带有一个或多个参数的函数。参数对函数而言相当于一种数据输入，函数会对参数进行某种处理，或根据参数的不同而执行不同功能。

如\_Plus 函数用于计算两个参数的和：

```
Func _Plus($x, $y)
    Dim $sum = $x + $y
    Return $sum
```

```
EndFunc
```

而\_Message 函数则根据\$Mode 的值不同，输出不同的文本

```
Func _Message($Mode)
    Switch $Mode
        Case 1
            MsgBox(0, "Mode=1", "Hello World!")
        Case 2
            MsgBox(0, "Mode=2", "Hello Au3!")
        Case Else
            MsgBox(0, "Mode=3", "Hello World! Hello Au3!")
    EndSwitch
EndFunc
```

### 2、无参函数

无参函数，即没有参数的函数。无参函数一般用于执行某种固定的处理或操作。

例如\_HelloWorld 函数的功能就是输出 “Hello World!”。

```
Func _HelloWorld()
    MsgBox(0, "", "Hello World!")
EndFunc
```

### 3、空函数

空函数，即函数内没有任何语句的函数。空函数没有实际作用，在程序设计中一般用于“占位”，等待后期补充功能。虽然空函数不会实现任何功能，但仍是模块化程序设计思想中的重要部分。

例如：

```
Func _TestFunction()

EndFunc
```

### 2.1.3 函数的参数与返回值

#### 1、形式参数与实际参数

( 1 ) 定义函数时，函数名右侧括号里的参数，被称为形式参数（形参）。

( 2 ) 函数被调用时，实际给予函数的参数，被称为实际参数（实参）。

仍以刚才的代码为例：

```
Dim $a, $b
$a = 1
$b = 2
Dim $c = _Plus($a, $b)
MsgBox(0, "", $a & "+" & $b & "=" & $c)

Func _Plus($x, $y)
```

```

Dim $sum = $x + $y
Return $sum
EndFunc

```

"Dim \$c=\_Plus(\$a, \$b)" 这行中\$a 和\$b 即为实际参数；

"Func \_Plus(\$x, \$y)" 这行中的的\$x 和\$y 则为形式参数。

## 2、参数的值传递与引用

### (1) 值传递

一般而言，当函数被调用时实参将值传递给形参，即所谓值传递。换句话说，这种情况下形参仅仅是获得了实参的值，那么形参的任何改变亦与实参无关。例如：

```

Dim $a = 1
_PlusOne($a)
MsgBox(0, "", $a)

Func _PlusOne($x)
$x = $x + 1
EndFunc

```

最终结果，\$a 的值仍为 1。因为实参\$a 仅仅是把自己的值 1 传递给了形参\$x，\$x 即便做了自增 1 操作，也不过是形参\$x 的值自增了 1，并不影响实参\$a 的值。

值传递是函数的最常见形式，也是使用最为频繁的形式。形参值的改变不影响实参值，最大程度上体现出了函数（功能模块）的独立性，保证了原始数据不被意外“写脏”。

### (2) 引用

有些特殊时候，的确需要实参随形参的改变而改变，这种情况下需要使用参数的引用。“引用”相当于将形参与实参建立起关联，形参可以理解为是实参的别名，二者所对应的是同一个数据。

建立引用的一般方法，即在形参前加上 “ByRef”，例如：

```

Dim $a = 1, $b = 2
_Exchange($a, $b)
MsgBox(0, "", "$a=" & $a & " " & "$b=" & $b)

Func _Exchange(ByRef $x, ByRef $y)
Dim $t
$t = $x
$x = $y
$y = $t
EndFunc

```

函数\_Exchange 的作用是交换\$x 和\$y 的值，当调用函数\_Exchange 时将\$a 和\$b 设置为实参，因形参\$x 和\$y 为引用模式，则\$x 相当于\$a 的别名、\$y 相当于\$b 的别名，那么当\$x 和\$y 的值发生交换时，\$a 和\$b 的值也发生了交换。所以最终\$a 的值为 2，\$b 的值为 1。

相比于值传递方式，引用方式具有一定的风险，因为此方式将直接改变原始数据的值，虽然在特定条件下这种方式更为简便快捷，但仍请使用者灵活谨慎的使用此方式。

引用方式一般用于两种情况：第一，当数据量很大时，引用方式效率更高，因为引用方式是形参

为实参取了个别名，而值传递需要将实参的数据传递给形参；第二，当函数需要“返回”多个值时，可以借助参数的引用实现（例如上例中的交换操作需要“返回”两个值，而函数只能靠 Return 返回一个值）。

### 3、参数的默认值

可以为某个参数设置默认值，当不指定这个参数的值时则使用默认值。设置默认值的方法即在形式参数的右侧写赋值运算符（=）和默认值，例如：

```
Dim $a = 1, $b = 2
Dim $r1, $r2

$r1 = _Plus($a, $b)
MsgBox(0, 'Test', $r1)
$r2 = _Plus($a)
MsgBox(0, 'Test', $r2)

Func _Plus($x, $y = 1)
    Return $x + $y
EndFunc
```

上例中，\_Test 函数的作用是将形参 \$x 与 \$y 的值相加并返回，其中 \$y 的默认值为 1。

第一次调用\_Test 函数时，指定了实参 \$a 和 \$b，值传递给形参 \$x 和 \$y，那么 \$x + \$y 等同于  $1 + 2$ ，所以返回值为 3；

第二次调用\_Test 函数时，只指定了实参 \$a，则形参 \$x 的值为 \$a 的值，而因未指定形参 \$y 的值，则 \$y 为默认值 1，那么 \$x + \$y 等同于  $1 + 1$ ，所以返回值为 2。

### 4、函数的返回值

#### (1) Return

函数的返回值一般是通过 Return 实现的，Return 的用法很简单：

```
Return <值>
```

注意：

- 函数内一旦执行了 Return 代码，则返回被调用处，如果 Return 代码之后还有其他代码，这些代码将不被执行；
- Return 返回的是一个值，如果是变量则返回变量的值，如果是表达式则返回其运算结果；
- 如果不写 Return，函数的默认返回值为 0；
- Return 一次只能返回一个值。

#### (2) SetError 与 @error、@extended

当函数执行完毕返回时，可通过使用 SetError 函数设置 @error、@extended 值，以令函数带回更多信息。

SetError 语法：

```
SetError(错误值[, 扩展值=0[, 返回值]])
```

参数：

错误值：即设定@error 的值

扩展值：即设定@extended 的值，默认为 0

返回值：即设定 SetError 函数的返回值

返回值：

如指定了返回值参数则返回指定的返回值，如未指定则返回一个未知值

举例：

```

Dim $a

Dim $i = 1
While $i <= 3 ;循环 3 次
    If $i = 1 Then $a = 5 ;第 1 次$a 值为 5
    If $i = 2 Then $a = -3 ;第 2 次$a 值为-3
    If $i = 3 Then $a = 15 ;第 3 次$a 值为 15

    ;调用_Test 函数
    $r = _Test($a)
    Switch @error
        Case 0 ;@error 为 0，则$a 介于 0~9
            MsgBox(0, "", $a & "介于 0~9")
        Case 1 ;@error 为 1，则$a 不在 0~9 之间
            Switch @extended
                Case 1 ;@extended 为 1，则$a<0
                    MsgBox(0, "", $a & "小于 0")
                Case 2 ;@extended 为 2，则$a>9
                    MsgBox(0, "", $a & "大于 9")
            EndSwitch
    EndSwitch
    EndSwitch

    $i += 1
WEnd

Func _Test($x)
    If $x >= 0 And $x <= 9 Then
        ;当$x 的值介于 0~9 时，返回 1，并将@error 设置为 0，@extended 设置为 0
        Return SetError(0, 0, 1)
    Else
        If $x < 0 Then
            ;当$x 的值小于 0 时，返回 0，并将@error 设置为 1，@extended 设置为 1
            Return SetError(1, 1, 0)
        Else
            ;当$x 的值大于 9 时，返回 0，并将@error 设置为 1，@extended 设置为 2
            Return SetError(1, 2, 0)
        EndIf
    EndIf

```

```
EndFunc
```

注意：

- @error 和@extended 的值一般为整数值，作为各种错误的标记；
- 不要企图使用错误值帮助函数返回多个值，这是极不符合编程习惯的做法；
- 如果不为函数设置@error 和@extended 值，则@error 和@extended 均默认为 0；
- 无论是标准函数还是自定义函数，执行后均会改变@error 和@extended 值；
- @error 值一般以 0 表示无错误，以非 0 表示有错误，不同数值标识不同种类的错误；
- @extended 值一般以 0 表示无拓展值，以非 0 表示有拓展，不同数值标识不同情况。

### 2.1.4 函数的嵌套调用与递归调用

#### 1、嵌套调用

函数的嵌套调用，即在函数中再次调用其他函数。

例：三数值排序（第 1.5.6 节，第 3 题），以函数嵌套调用的方式来实现：

```
Dim $a = 3, $b = 1, $c = 2  
_NumSort($a, $b, $c)  
MsgBox(0, "三数值排序", "从小到大: " & $a & ", " & $b & ", " & $c)  
  
Func _NumSort(ByRef $n1, ByRef $n2, ByRef $n3)  
    ;若$n1>$n2，则交换$n1 和$n2 的值  
    If $n1 > $n2 Then _Exchange($n1, $n2)  
    ;若$n1>$n3，则交换$n1 和$n3 的值  
    If $n1 > $n3 Then _Exchange($n1, $n3)  
    ;若$n2>$n3，则交换$n2 和$n3 的值  
    If $n2 > $n3 Then _Exchange($n2, $n3)  
EndFunc  
  
Func _Exchange(ByRef $x, ByRef $y)  
    ;交换两变量数值  
    Dim $t = $x  
    $x = $y  
    $y = $t  
EndFunc
```

\_NumSort 函数用于将三个数值按照从小到大的顺序存放至三个变量中，而 \_Exchange 函数用于交换两个变量的值。在 \_NumSort 函数中，将三个变量的值两两对比，当发现第一个变量值大于第二个变量值时，将两变量数值进行交换。

可以发现的是，以函数的写法解决问题，代码整体更为清晰简练，可读性和可维护性均有提高。

#### 2、递归调用

函数的递归调用，即在函数中再次调用函数自身。这是一个有些绕的概念，举个例子帮助大家理

解：

有五个人坐在一起，第五个人说他比第四个人大 2 岁，第四个人说他比第三个人大 2 岁，第三个人说他比第二个人大 2 岁，第二个人说他比第一人大 2 岁，第一个人说他 10 岁，请问第五个人几岁？

对于这个问题：

(1) 要知道第五个人几岁，就必须先知道第四个人几岁，而要知道第四个人几岁，就必须知道第三个人几岁，以此类推，直至得知第一个人的年龄，再反向回推；

(2) 五个人的年龄差距都是 2 岁，也就是说“第 n 个人的年龄 = 第 (n-1) 个人的年龄 + 2”，而当 n 为 1 ( 即第一个人 ) 时，年龄为 10 。

明确了这些，我们用函数的递归调用来解决：

```
MsgBox(0, "", "第五个人的年龄是：" & _Age(5))
```

;函数\_Age 用于求出年龄，\$n 代表第几个人

```
Func _Age($n)
    If $n = 1 Then
        ;当$n=1 时 ( 第一个人 ), 返回年龄 10
        Return 10
    Else
        ;否则返回上个人的年龄+2
        Return _Age($n - 1) + 2
    EndIf
EndFunc
```

递归过程描述：

```
_Age(5)=_Age(4)+2
    _Age(4)=_Age(3)+2
        _Age(3)=_Age(2)+2
            _Age(2)=_Age(1)+2
                _Age(1)=10
                    _Age(2)=10+2=12
                        _Age(3)=12+2=14
                            _Age(4)=14+2=16
                                Age(5)=16+2=18
```

通过递归得出，第五个人的年龄为 18 岁。

递归算法，是一种需要认真思考的技巧性算法。递归算法可以很巧妙的解决很多复杂的问题，如遍历未知层数的文件夹、注册表等，但递归算法也有不易思考、不易维护的缺点。请大家在使用递归算法时一定要严谨慎重。

## 2.1.5 变量（常量）的作用域

至目前为止，为了方便大家的学习，我们一直是使用 Dim 来定义变量的。在我们学习了函数相关知识后，从本节起要改为使用 Local 和 Global 来定义变量，并明确变量的作用范围。

## 1、局部变量（常量）与其作用域

(1) 局部变量，即只在变量所属函数内有效的变量。

局部变量的定义方法：

```
Local <变量>
```

如：

```
Local $a
```

局部变量的作用域（作用范围）为其所属函数，举例：

```
Func _TestA()
```

```
    Local $a = 5  
    $a += 1  
    Return $a
```

```
EndFunc
```

```
Func _TestB($x, $y)
```

```
    Local $a  
    $a = $x + $y  
    Return $a
```

```
EndFunc
```

在函数\_TestA 和\_TestB 中，都定义了变量\$a，但由于\$a 的作用域仅限于其所属函数，所以这两个函数中的变量\$a 毫无干扰。打个比方，就像两间房间里分别放置了两台一模一样的电视机，这个房间里看什么节目对另一个房间里看什么节目不会产生影响。

值得额外说明的是，如果函数有参数，那么函数参数默认为本函数的局部变量。例如\_TestB 函数的参数\$x 与\$y，均为\_TestB 的局部变量。

(2) 局部常量，即只在常量所属函数内有效的常量。

局部常量的定义方法：

```
Local Const <常量> = <值>
```

如：

```
Local Const $MAX=5
```

与局部变量唯一的不同仅仅是其值不可被改变，其余方面均相同，所以不再赘述，请各位类比。

## 2、全局变量（常量）与其作用域

(1) 全局变量，即在整个代码中均有效的变量。

全局变量的定义方法：

```
Global <变量>
```

如：

```
Global $g_IniFile
```

全局变量的作用域（作用范围）是整个代码，包括每个函数，举例：

```
Global $g_Num = 1
```

```
_TestA()
```

```
_TestB()
```

```

    MsgBox(0, "", $g_Num)

    Func _TestA()
        $g_Num += 1
    EndFunc

    Func _TestB()
        Local $a = 3
        $g_Num = $g_Num + $a
    EndFunc

```

本例中，定义了全局变量\$g\_Num，并赋值为 1。因全局变量的作用域为整个代码，所以执行函数\_TestA 后\$g\_Num 的值增加了 1，执行\_TestB 后\$g\_Num 的值增加了 3。在最后使用 MsgBox 输出时，\$g\_Num 的值为 5。

### (2) 全局常量，即在整个代码中均有效的常量。

全局常量的定义方法：

```
Global Const <常量>=<值>
```

如：

```
Global Const $PI=3.1415926
```

与全局变量唯一的不同仅仅是其值不可被改变，其余方面均相同，所以不再赘述，请各位类比。

## 3、全局变量（常量）与局部变量（常量）重名情况的处理

以变量举例（常量类同），当局部变量与全局变量重名时，局部变量作用域内全局变量被屏蔽。

举例：

```

Global $sum = 0
_Plus()
MsgBox(0, "", $sum)

Func _Plus()
    Local $sum
    Local $x = 1, $y = 2
    $sum = $x + $y
EndFunc

```

这段代码的执行结果为 0。

代码开头定义了全局变量\$sum，并赋值为 0。而在函数\_Plus 中又定义了局部变量\$sum。当\_Plus 函数被调用时，虽然执行了\$sum=\$x+\$y=1+2=3 的操作，但因为局部变量\$sum 与全局变量\$sum 重名，在\_Plus 函数内全局变量\$sum 被屏蔽，所以全局变量\$sum 的值并没有发生任何改变，仍为 0。

但在这里提醒大家，实际编程应用中，应尽可能避免全局变量与局部变量重名。

## 4、关于 Dim、Local 与 Global

### (1) 弃用 Dim 的原因

从本节起，我们弃用 Dim，最主要的原因是 Dim 不能良好的区分局部和全局。Dim 定义变量的

规则：

- 如果是在函数外定义变量则自动认为定义的是全局变量；
- 如果是在函数内定义变量则自动认为定义的是局部变量；
- 当定义的局部变量与全局变量重名时，则自动认为局部变量是全局变量。

前两条规则显得很智能，但在某些特殊情况下（例如一定要在函数内定义全局变量）则不好用。而致命的问题是第三条，这会令全局变量和局部变量发生混淆，易产生运行错误，并造成错误难于排查。例如：

```
Dim $a  
_Test()  
MsgBox(0, "", $a)  
  
Func _Test()  
    Dim $a = 1  
    $a += 1  
EndFunc
```

运行后\$a 的结果为 2。因为全局变量\$a 与局部变量\$a 重名，但使用 Dim 定义时，会自动认为局部变量\$a 就是全局变量\$a，则\_Test 运行后全局变量\$a 的值发生了改变！如果有多个函数交错使用某个与全局变量命名相同的局部变量时，问题就会变得错综复杂。

### (2) 关于 Local 和 Global 其他要知道的

与 Dim 的自动判定不同的是，Local 是强制定义一个局部变量，Global 是强制定义一个全局变量。使用 Local 和 Global 定义，可以非常明确的得知变量的作用域，而只有意图明确的代码才能最大程度的正确执行，并利于代码的阅读和修正。

有一点必须要注意的是，如果在函数外部使用了 Local 声明变量，那么 Local 的作用域将扩展到整个代码，这种条件下 Local 与 Global 的作用域完全相同。这是一种不正规的用法，不建议尝试或使用。

### 2.1.6 代码的函数化

所谓代码的函数化，即以一个主函数开始执行程序，在主函数中调用各子函数以实现不同功能，主函数执行结束后退出程序。这种思路起源于 C 语言，是一种经典的代码书写方式，通过合理的规划变量作用域和函数功能模块，可有效减少程序错误，有利于模块化开发，有助于提高可读性和可维护性。

举例：

```
Global $g_Var1, $g_Var2  
  
;以主函数开始  
_Main()  
;以主函数结束  
Exit
```

```
Func _Main()
Local $a, $b

;调用各子函数
_Function1()
_Function2()
_Function3()

EndFunc

Func _Function1()
Local $s1, $s2
;子函数 1
EndFunc

Func _Function2($x, $y)
;子函数 2
EndFunc

Func _Function3()
;子函数 3
EndFunc
```

程序执行时，执行主函数（\_Main），主函数内调用各子函数以实现各子功能，当主函数执行完毕后执行 Exit 代码退出程序。这种代码书写方式，最突出的优点有二：

- (1) 除了必要的变量声明为全局变量外，其余变量均为局部变量，可使代码整体多个变量充分独立，无相互影响；
- (2) 可使各子功能模块充分独立，便于独立维护每个功能，便于功能函数的移植，有利于多人协同开发，并有助于后期的代码维护与修正。

今后写代码时，强烈推荐使用函数化书写方法。

## 2.2 数组

到目前为止，我们所学习的数据类型都是基本类型，如数值型、字符串、布尔型。基本类型数据是一种“单一”数据，即一个基本类型的常量或变量中只包含一个数据。而本节将要学习的数组是一种构造类型数据，构造类型数据是一种“复合”数据，一个构造类型的常量或变量中往往包含多个数据。

数组是一种最基本的构造类型，数组内的数据是一系列有序或相关数据的集合。数组类型的变量（常量）使用统一的数组名和不同的索引去标识数组内的不同数据，而数组内不同数据又以索引为序号实现逻辑关联。

## 2.2.1 一维数组

### 1、一维数组的定义

定义一维数组的一般方式：

```
Global/Local <数组名> [元素个数]
```

例如：

```
Local $Array[5]
```

上述语句定义了一个数组，数组名为\$Array，此数组拥有 5 个元素。

说明：

(1) 数组名的命名规则与变量命名规则相同，不再赘述；

(2) 值得注意的是，数组的索引是 0 基索引（从 0 开始的索引），所以\$Array 的 5 个元素的索引分别为 0~4，而非 1~5；

(3) 一维数组结构类似下表：

\$Array[0]
\$Array[1]
\$Array[2]
\$Array[3]
\$Array[4]

(4) 一维数组定义后，每个元素的默认值均为空字符串。

### 2、一维数组的赋值

#### (1) 创建时对所有元素进行赋值

例如：定义一个一维数组，数组名为\$Array，数组拥有 5 个元素，对 5 个元素依次为 0、1、2、3、4。

```
Local $Array[5] = [0, 1, 2, 3, 4]
```

这是一种很常见的对数组内所有元素进行初始化的方法。上述数组数据形如：

0
1
2
3
4

#### (2) 创建时对部分元素进行赋值

例如：定义一个一维数组，数组名为\$Array，数组拥有 5 个元素，对前 3 个元素依次赋值 0、1、2。

```
Local $Array[5] = [0, 1, 2]
```

上述数组数据形如：

0
1

2

**(3) 创建时按照赋值自动决定一维数组大小**

可根据赋值元素数量自动决定一维数组的大小，例如：

```
Local $Array[] = [0, 1, 2, 3, 4]
```

本例中，定义\$Array 时并未指定数组大小，而之后的赋值操作中赋值了 5 个元素，所以数组的大小自动为 5。本例的数据效果与（1）中例子完全相同。

**(4) 分别对每个元素进行赋值**

在数组定义后，分别对每个元素进行赋值，这是一种普遍而常见的方式：

```
Local $Array[5]
$array[0]=1
$array[1]=2
$array[2]=3
$array[3]=4
$array[4]=5
```

本例的数据效果与（1）中例子完全相同。

**3、一维数组元素的使用**

使用数组中元素的一般方式：

```
<数组名>[索引]
```

例如：

```
Local $Array[5] = [0, 1, 2, 3, 4]
MsgBox(0, "", $Array[1])
```

输出结果为 1，即\$Array 数组中索引为 1 的元素的值。

其实除了需要写索引以外，使用数组数据与使用普通变量没有什么不同。

**2.2.2 二维数组****1、二维数组的定义**

定义二维数组的一般方式：

```
Global/Local <数组名>[第一维元素个数][第二维元素个数]
```

例如：

```
Local $Array[5][3]
```

上述语句定义了一个数组，数组名为\$Array，此数组第一维拥有 5 个元素，第二维拥有 3 个元素，总计  $5 \times 3 = 15$  个元素。

说明：

（1）数组名的命名规则与变量命名规则相同，不再赘述；

（2）值得注意的是，数组的索引是 0 基索引（从 0 开始的索引）；

(3) 二维数组结构类似下表：

\$Array[0][0]	\$Array[0][1]	\$Array[0][2]
\$Array[1][0]	\$Array[1][1]	\$Array[1][2]
\$Array[2][0]	\$Array[2][1]	\$Array[2][2]
\$Array[3][0]	\$Array[3][1]	\$Array[3][2]
\$Array[4][0]	\$Array[4][1]	\$Array[4][2]

(4) 将二维数组理解为表格，更容易理解二维数组数据的分布，在很多实际应用中，经常使用二维数组来表达表数据，而对于二维数组的两个维，通常也称为行和列；

(5) 二维数组定义后，每个元素的默认值均为空字符串。

## 2、二维数组的赋值

### (1) 创建时对所有元素进行赋值

例如：定义一个3行2列的数组，为第1行赋值a和b，为第2行赋值c和d，为第3行赋值e和f。

```
Local $Array[3][2] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
```

这是一种很常见的对数组内所有元素进行初始化的方法。值得注意的是，除了最外侧的中括号，每行数据还需要再用一对英文中括号进行包含，以明确表示这是一行。上述数组数据形如：

a	b
c	d
e	f

### (2) 创建时对部分元素进行赋值

例如：定义一个3行2列的数组，为第1行赋值a和b，为第2行赋值c和d。

```
Local $Array[3][2] = [['a', 'b'], ['c', 'd']]
```

上述数组数据形如：

a	b
c	d

### (3) 创建时按照赋值自动决定二维数组大小

可根据赋值元素数量自动决定二维数组的大小，例如：

```
Local $Array[] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
```

定义\$Array时，未指定二维中任意一维的大小，那么将根据赋值时最大行数和最大列数自动设定第一维和第二维。上例中，最大行数为3，最大列数为2，则第一维大小自动为3，第二维大小自动为2。本例的数据效果与(1)中例子完全相同。

还可以只指定某一维，例如仅指定第二维：

```
Local $Array[][2] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
```

第一维会根据最大行数自动决定为3。

需要注意的是，如果每行列数不同，则以最大列数为第二维大小。

例如：

```
Local $Array[] = [['a', 'b'], ['c', 'd', 'x'], ['e', 'f']]
```

数据为：

a	b	
c	d	x
e	f	

这是因为第 2 行的是 3 列，所以整体自动改为了 3 列。

#### (4) 分别对每个元素进行赋值

在数组定义后，分别对每个元素进行赋值，这是一种普遍而常见的方式：

```
Local $Array[3][2]
$Array[0][0] = 'a'
$Array[0][1] = 'b'
$Array[1][0] = 'c'
$Array[1][1] = 'd'
$Array[2][0] = 'e'
$Array[2][1] = 'f'
```

本例的数据效果与（1）中例子完全相同。

### 3、二维数组元素的使用

使用数组中元素的一般方式：

```
<数组名>[索引 1][索引 2]
```

例如：

```
Local $Array[3][2] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
MsgBox(0, "", $Array[1][1])
```

输出结果为 d，即\$Array 数组中一维索引为 1、二维索引为 1 的元素的值。

其实除了需要写索引以外，使用数组数据与使用普通变量没有什么不同。

#### 2.2.3 动态数组

所谓动态数组，是指根据实际数据需求，动态调整数组的大小。对数组大小的重定义由 ReDim 来实现，ReDim 的语法：

```
ReDim <数组>[索引 1][索引 2]...[索引 n]
```

例如：

```
;定义一个一维数组，大小为 3
Local $Array[3]
;将数组大小调整至 5
ReDim $Array[5]
```

说明：

（1）对于同一维，如果 ReDim 后的大小大于原大小，则产生新元素，默认值为空字符串，例如：

```
Local $Array[3]=[0,1,2]
ReDim $Array[5]
```

\$Array 的值为：

0
1
2

新增\$Array[3]和\$Array[4]元素，且值为空字符串。

(2) 对于同一维，如果 ReDim 后的大小小于原大小，则多余元素被舍弃，数据将丢失，例如：

```
Local $Array[3]=[0,1,2]
ReDim $Array[2]
```

\$Array 的值为：

0
1

原有的\$Array[2]元素被舍弃，数据 2 丢失。

(3) 在(1)和(2)中的规律用于数组的某一维，而非只用于一维数组，二维数组等更多维数的数组同样适用。无非就是“增长则新增元素，缩短则抛弃元素”。

(4) 当使用 ReDim 改变数组维数时，数组全部数据被抛弃，例如：

```
Local $Array[3]=[0,1,2]
ReDim $Array[3][3]
```

或

```
Local $Array[3][2] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
ReDim $Array[5]
```

前者产生一个  $3 \times 3$  元素的空二维数组，后者产生一个 5 元素的空一维数组，所有元素值均为空字符串。

### 2.2.4 数组的其他相关

#### 1、数组的上限

数组可以有更多的维数，例如三维数组、四维数组，一维数组和二维数组只是比较常见的两种，但维数并不可以无限多，AutoIt v3 中最大允许 64 维数组。同时，数组的元素也并不可以无限多，AutoIt v3 中最大允许数组中包含约 1600 万个元素。

值得注意的是，维数上限和元素上限是同时奏效的，即无论你有多少维，哪怕只有一维，元素的最大数量也不可超过 1600 万个；而无论你有多少元素，哪怕每一维只有 1 个元素，维数最多也不可超过 64 维。

#### 2、数组元素数据类型可以不同

与很多强语法编程语言不同，在 AutoIt v3 中同一个数组中允许存储不同类型的数据，例如使用二维数组记录学生姓名（字符串）和成绩（数值）：

```
Local $Array[3][2] = [{"Skye", 88}, {"Una", 85}, {"GMan", 95}]
```

### 3、不常使用的 0 元素

元素的索引是 0 基索引，这和我们所熟悉的 1 基索引有一定区别，为了便于思维、减少错误，我们在使用数组时一般不使用元素 0，而是从元素 1 开始使用，例如：

```
Local $Array[6] = ["", 1, 2, 3, 4, 5]
```

定义一个数组\$Array，大小为 6，元素 0 保持为空字符串，从元素 1~元素 5 依次赋值 1~5。这样，我们就避开了元素 0，从元素 1 开始使用，与日常的 1 基索引形成了对应。不过元素 0 也没闲着，元素 0 经常用于存储一些特殊数据，如一维数组元素数、二维数组行数、程序进程 PID、GUI 控件 ID 等等。

### 4、数组的相互赋值

可以将一个数组的全部值赋值给另一个数组。被赋值数组的数据将与原数组的数据完全相同，被赋值数组的原始数据将完全丢失。例如：

```
Local $Array[6] = ["", 1, 2, 3, 4, 5]
Local $TestArray[3]
$TestArray = $Array
```

\$TestArray 是具有 3 个元素的数组，被赋值后将具有 6 个元素，各元素值与\$Array 完全相同。简而言之，无论被赋值数组是多少维、多少元素、元素值都是什么，被赋值后均与原数组一模一样。甚至被赋值的变量本不是数组，被赋值后也与原数组相同，例如：

```
Local $Array[6] = ["", 1, 2, 3, 4, 5]
Local $a
$a = $Array
```

\$a 并非数组，但在被赋值后，也成为了与\$Array 完全相同的数组。

注意，数组对数组赋值时，数组只需要写数组名就可以了。

### 5、数组作为函数参数

数组同样可以作为函数的参数，作为参数时只需要数组名即可，例如：

```
_Main()
Exit

Func _Main()
    Local $Array[6] = ["", 1, 2, 3, 4, 5]
    _Test($Array)
EndFunc

Func _Test($TestArray)
    MsgBox(0, "", $TestArray[1])
EndFunc
```

\$Array 作为实参将所有值传递给形参\$TestArray，则\$TestArray 数据与\$Array 完全相同，输出\$TestArray[1]的值为 1。

另外，对于大型数组，建议采用引用方式。因为值传递模式下，实参要将所有数据传递给形参，当数组内数据量很大时，会造成程序执行效率的下降。而引用只是形参作为实参的别名，并没有数据传递这一环节，效率更高。

### 6、清理数组

数组内一般包含较多数据，当一个数组不再被使用时，将数组赋值为空字符串即可达到清理数组的目的，例如：

```
Local $Array[6] = ["", 1, 2, 3, 4, 5]
$Array = ""
```

一个简单的操作，\$Array 数组就被清理了。

注意，这里所讲的清理是彻底抛弃数组的维数和值，并不是将数组内所有元素值抹去，如果要将数组所有值重置为空字符串，需要借助循环结构来实现。

### 7、获得数组的大小

有时当我们得到一个数组，可能并不知道数组的大小，例如动态数组、利用第三方函数得到的数组等，这种情况下可以使用 UBound 函数来获取数组的大小。

**语法：**

```
UBound(数组[, 维数=1])
```

**参数：**

数组：目标数组

维数：可选参数，以指定获取数组第几维的大小，默认为 1（即第一维）

**返回值：**

成功：返回指定维数的大小

失败：返回 0，并设置@error 值

@error：

1，指定的“数组”并非数组

2，指定的数组维数无效

**举例：**

```
Local $Array1[5] = [0, 1, 2, 3, 4]
Local $r1 = UBound($Array1) ;$r1 值为 5

Local $Array2[3][2] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
Local $r2 = UBound($Array2, 1) ;$r2 值为 3
Local $r3 = UBound($Array2, 2) ;$r3 值为 2
```

**注意：**

UBound 获得的是数组某一维的元素数量，而最大索引应该为（元素数量-1）。例如 6 个元素的数组，索引为 0~5，最大索引是 5，即（6-1）。

### 8、展示数组内容

在书写程序时，经常需要反复调试，需要一种直观的方式了解数组内各个元素的值，这个需求可通过\_ArrayDisplay 函数实现。

\_ArrayDisplay 函数参数比较复杂，暂时只了解其常用参数即可，详细参数会在后续章节讲解。

#### 语法：

```
#include <Array.au3>
_ArrayDisplay(数组[, "标题"])
```

#### 参数：

数组：目标数组

标题：可选参数，展示窗体的标题

#### 返回值：

成功：返回 1

失败：返回 0，并设置@error 值

@error 值：

1，指定的“数组”并非数组

2，指定的数组最大维数超过 2 (\_ArrayDisplay 只能展示一维和二维数组)

#### 其他：

\_ArrayDisplay 函数是一个 UDF（用户自定义函数），使用前需要在代码顶部写上 "#include <Array.au3>"。（至于这句代码是什么作用，后续章节学习了#include 后即会明白，这里只需要记得不写这行代码会导致\_ArrayDisplay 函数无法正常使用即可）

#### 举例：

```
#include <Array.au3>

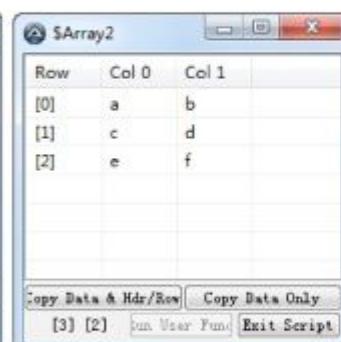
Local $Array1[5] = [0, 1, 2, 3, 4]
_ArrayDisplay($Array1, '$Array1')

Local $Array2[3][2] = [['a', 'b'], ['c', 'd'], ['e', 'f']]
_ArrayDisplay($Array2, '$Array2')
```

#### 效果：



(图 2-1)



(图 2-2)

## 2.2.5 数组类程序举例

### 1、找出数组中所有数值的最小值

在一个由数值数据组成的一维数组中，找到其中最小值。

**算法分析：**

- (1) 定义一个变量(如：\$Min)来存储最小值；
- (2) 首先假定数组的元素1为最小值，并将其值复制给\$Min；
- (3) 而后令\$Min与数组内的其他元素(元素2~元素n)依次与\$Min相比较，谁比\$Min小就将谁的值再次赋给\$Min；
- (4) 最终所获得的\$Min的值，即为整个数组的最小值。

**程序代码：**

```
_Main()
Exit

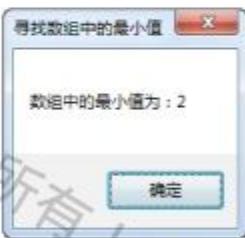
Func _Main()
    ;定义一个包含多个数值的数组
    Local $a_Num[] = [", 5, 8, 10, 4, 15, 13, 2, 9, 7]

    ;假定最小值为元素1
    Local $Min = $a_Num[1]

    Local $i
    ;从元素2开始遍历到数组的最后一个元素
    For $i = 2 To UBound($a_Num, 1) - 1
        ;如果哪个元素比最小值还小，则令最小值为这个元素的值
        If $a_Num[$i] < $Min Then $Min = $a_Num[$i]
    Next

    ;输出最小值
    MsgBox(0, "寻找数组中的最小值", "数组中的最小值为：" & $Min)
EndFunc  ;==>_Main
```

**运行图例：**



(图 2-3)

### 2、将数组中的数值按照从小到大的顺序排序

将一个数组内无序的数值按照从小到大的顺序进行排序。

#### 算法分析：

- (1) 数组排序是数组类问题中的典型问题，本例将介绍最经典的“冒泡排序法”；
- (2) 所谓冒泡排序，就是经过多次对数组的遍历，逐步将最小（或最大）值交换到数组的前部元素位置上，就像气泡上浮一样，所以称之为“冒泡”法；
- (3) 以上例的数组“5、8、10、4、15、13、2、9、7”举例：

循环	执行	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
--	(排序前的无序数组)	5	8	10	4	15	13	2	9	7
第1次	将元素[1]~[9]的最小值放在元素[1]上	2	8	10	5	15	13	4	9	7
第2次	将元素[2]~[9]的最小值放在元素[2]上	2	4	10	8	15	13	5	9	7
第3次	将元素[3]~[9]的最小值放在元素[3]上	2	4	5	10	15	13	8	9	7
第4次	将元素[4]~[9]的最小值放在元素[4]上	2	4	5	7	15	13	10	9	8
第5次	将元素[5]~[9]的最小值放在元素[5]上	2	4	5	7	8	15	13	10	9
第6次	将元素[6]~[9]的最小值放在元素[6]上	2	4	5	7	8	9	15	13	10
第7次	将元素[7]~[9]的最小值放在元素[7]上	2	4	5	7	8	9	10	15	13
第8次	将元素[8]~[9]的最小值放在元素[8]上	2	4	5	7	8	9	10	13	15
--	(排序后的有序数组)	2	4	5	7	8	9	10	13	15

(4) 如上表，每次循环时数组的起始位置比上次多 1（例如第 1 次是元素 1、第 2 次是元素 2），而每次循环的目的都是将此次所遍历部分的最小值交换至起始位置的元素上，如此，通过 8 次循环解决了 9 数排序问题。

#### 程序代码：

```
#include <Array.au3>

_Main()
Exit

Func _Main()
;定义一个包含多个数值的数组
Local $a_NUm[] = ["", 5, 8, 10, 4, 15, 13, 2, 9, 7]
_ArrayDisplay($a_NUm, "排序前的无序数组")

Local $i, $j
;令起始元素依次为第一个元素至倒数第二个元素
For $i = 1 To UBound($a_NUm, 1) - 2
;从起始元素的下一个元素起至倒数第一个元素
For $j = $i + 1 To UBound($a_NUm, 1) - 1
;如果哪个元素比起始元素要小，则令其与起始元素互换值
If $a_NUm[$j] < $a_NUm[$i] Then
    _Exchange($a_NUm[$i], $a_NUm[$j])
EndIf
Next
Next
```

```
;完成排序并输出
_ArrayDisplay($a_Num, "排序后的有序数组")
EndFunc ;==>_Main

Func _Exchange(ByRef $n1, ByRef $n2)
;交换两个变量的值
Local $t = $n1
$n1 = $n2
$t = $n2
EndFunc ;==>_Exchange
```

运行图例：

排序前的无序数组	
Row	Col 0
[0]	
[1]	5
[2]	8
[3]	10
[4]	4
[5]	15
[6]	13
[7]	2
[8]	9
[9]	7

( 图 2-4 )

排序后的有序数组	
Row	Col 0
[0]	
[1]	2
[2]	4
[3]	5
[4]	7
[5]	8
[6]	9
[7]	10
[8]	13
[9]	15

( 图 2-5 )

### 3、将数字矩阵按照对角线进行翻转

设法将数字矩阵：

1	2	3
4	5	6
7	8	9

沿左上角至右下角的对角线进行对换，得到新数字矩阵：

1	4	7
2	5	8
3	6	9

算法分析：

- ( 1 ) 很明显的，这次需要二维数组出马了；
- ( 2 ) “沿左上角至右下角的对角线进行对换”，即将[0][1]元素对换为[1][0]、[0][2]元素对换为[2][0]，以此类推；
- ( 3 ) 如果以变量*\$i*、*\$j* 分别代表第一维和第二维的索引，那么即[\$i][\$j]与[\$j][\$i]进行对换。

程序代码：

```
#include <Array.au3>
```

```

_Main()
Exit

Func _Main()
;定义一个数字矩阵数组
Local $a_Num1[3][3] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
;定义一个新数组，用于存储变换后的数字矩阵
Local $a_Num2[3][3]

;展示对换前数组的值
_ArrayDisplay($a_Num1, "对换前")
Local $i, $j
;遍历原数组的每个值，按照对换规则赋值给新数组
For $i = 0 To 2
    For $j = 0 To 2
        $a_Num2[$i][$j] = $a_Num1[$j][$i]
    Next
Next
;展示对换后数组的值
_ArrayDisplay($a_Num2, "对换后")
EndFunc  ;==> _Main

```

运行图例：

对换前			
Row	Col 0	Col 1	Col 2
[0]	1	2	3
[1]	4	5	6
[2]	7	8	9

[Copy Data & Hdr/Row] [Copy Data Only]  
[3] [3] Run User Func Exit Script

( 图 2-6 )

对换后			
Row	Col 0	Col 1	Col 2
[0]	1	4	7
[1]	2	5	8
[2]	3	6	9

[Copy Data & Hdr/Row] [Copy Data Only]  
[3] [3] Run User Func Exit Script

( 图 2-7 )

#### 4、将总分大于 500 分的学生信息进行汇总

已知一表格包括学生的姓名、性别、班级和总分，筛选表格中总分大于 500 的学生并生成新表格。

算法分析：

- ( 1 ) 二维数组是最适合表达表数据的；
- ( 2 ) 遍历表格，判断学生的分数，如果该学生分数大于 500，则将此学生的信息加入新表格；
- ( 3 ) 并不能预先确定有多少学生分数会大于 500，那么新表格需要动态决定行数。

程序代码：

```
#include <Array.au3>
```

```
_Main()
Exit

Func _Main()
;以常量$N 表示数组的第二维大小(列数)
Local Const $N = 5

;学生信息原数组
;第一维的 0 元素用于表示此列作用，第二维的 0 元素用于标记序号，0 元素均未用于存储学生信息数据
Local $a_Info[][$N] = [{"序号", "姓名", "性别", "班级", "总分"},_
[1, "李伟", "男", "二班", 520],_
[2, "王杨", "男", "一班", 575],_
[3, "张娜", "女", "五班", 397],_
[4, "孙静", "女", "三班", 437],_
[5, "刘昌", "男", "六班", 332],_
[6, "赵盛", "男", "三班", 415],_
[7, "齐露", "女", "二班", 360],_
[8, "金美", "女", "二班", 554],_
[9, "秦川", "男", "四班", 271],_
[10, "马冉", "女", "一班", 536]]

;输出所有学生信息数组
_ArrayDisplay($a_Info, "所有学生的信息")

;定义一个新数组，用于存储筛选后的学生信息
;与原数组具有相同大小的第二维(列数)，但第一维(行数)仅为 1，以便动态拓展
Local $a_Output[1][$N] = [{"序号", "姓名", "性别", "班级", "总分"}]

;定义$p 用于标记数组第一维上限、序号
Local $p = 1

Local $i, $j
;循环遍历学生信息数组
For $i = 1 To UBound($a_Info, 1) - 1
;如果学生的成绩大于 500
If $a_Info[$i][4] > 500 Then
;那么令新数组增加一行
ReDim $a_Output[$p + 1][$N]
;令序号为 1
$a_Output[$p][0] = $p
;其余元素依次与原数组相同
For $j = 1 To UBound($a_Info, 2) - 1
$a_Output[$p][$j] = $a_Info[$i][$j]
Next
;因为新数组添加了新的一行，所以$p 值应自增 1
$p += 1
EndIf
```

Next

```
;输出总分大于 500 分的学生信息数组
_ArrayDisplay($a_Output, "分数大于 500 分的学生信息")
EndFunc  ;==>_Main
```

运行图例：

所有学生的信息

Row	Col 0	Col 1	Col 2	Col 3	Col 4
[0]	序号	姓名	性别	班级	总分
[1]	1	李伟	男	二班	520
[2]	2	王强	男	一班	575
[3]	3	张娜	女	五班	397
[4]	4	孙静	女	三班	437
[5]	5	刘晶	男	六班	332
[6]	6	赵盛	男	三班	415
[7]	7	齐盛	女	二班	360
[8]	8	金美	女	二班	554
[9]	9	秦川	男	四班	271
[10]	10	马冉	女	一班	536

Copy Data & Hdr/Row      Copy Data Only  
[1] [5]      Run User Func      Exit Script

( 图 2-8 )

分数大于 500 分的学生信息

Row	Col 0	Col 1	Col 2	Col 3	Col 4
[0]	序号	姓名	性别	班级	总分
[1]	1	李伟	男	二班	520
[2]	2	王强	男	一班	575
[3]	3	金美	女	二班	554
[4]	4	马冉	女	一班	536

Copy Data & Hdr/Row      Copy Data Only  
[5] [5]      Run User Func      Exit Script

( 图 2-9 )

## 2.3 字符串

在 Level1 中，我们已经对字符串有了最基本的了解，而在 Level2 中，我们将深入了解更多字符串的相关知识。字符串类型作为一种基本数据类型，有着极高的出现频率，很多数据都是以字符串的方式存在，所以，学懂如何处理字符串，就是在驾驭数据方面提高了一个层次。

### 2.3.1 字符串变量的定义与赋值

字符串变量的定义与赋值我们已经在 1.3.3 节第 2 部分学习过了，本节只做简要回顾。如果您对这部分知识有所生疏，建议先回顾 1.3.3 节相关知识。

字符串变量的定义与赋值的一般方式：

```
Local <变量> = '<字符串>'
```

或

```
Local <变量> = "<字符串>"
```

例如：

```
Local $s1 = 'IT 天空'
```

```
Local $s2 = "Hello World! Hello Au3!"
```

注意：

(1) 字符串必须以一对英文单引号('')或一对英文双引号("")进行包含，引号数量必须匹配(有一个引号开头则必须有一个引号结尾)，引号类型必须相同(都是单引号或都是双引号)；

(2) 这里要特别提一下的是，只要符合(1)中的规则，英文单引号或双引号在使用中没有任何区别，力图书写简单则可使用单引号，与其他语言接近则使用双引号，完全是编程者习惯；

(3) 所谓“字符串”，即大于或等于1个字符，“a”、“abc”都被称为字符串，不因字符串仅包含一个字符而被称作“字符”；

(4) 被引号包含的部分就是字符串，不因是数字(例如“123”)就不被称为字符串，虽然在实际应用中变量类型是会根据条件自动转化的，但我们也必须明确数值123和字符串“123”是两种完全不同的类型。

### 2.3.2 字符串的连接

字符串的连接需要通过连接运算符“&”来实现，我们已经在1.4.1节第2部分学习过了，本节只做简要回顾。如果您对这部分知识有所生疏，建议先回顾1.4.1节相关知识。

连接运算符“&”，是将其左右两侧的字符串进行自然连接，而后得出连接后的值，一般方式为：

"<字符串1>" & "<字符串2>"

连接后的值为：

"<字符串1><字符串2>"

例如：

"abc" & "def"

连接过后的值为：

"abcdef"

常见的连接运算举例：

(1) 直接连接两字符串：

Local \$s = "abc" & "def"

\$s的最终值为：abcdef。

(2) 连接变量与字符串：

Local \$s1 = "abc"

Local \$s = \$s1 & "def"

\$s的最终值为：abcdef。

(3) 连接变量：

Local \$s1 = "abc", \$s2 = "def"

Local \$s = \$s1 & \$s2

\$s的最终值为：abcdef。

(4) 多次连接：

Local \$s1 = "abc", \$s2 = "def"

```
Local $s = $s1 & $s2 & "ghi"
```

同一行语句中多次出现连接运算（无括号改变运算优先级），按照从左至右的方式依次执行，即先连接\$s1 与 \$s2，在将\$s1 与 \$s2 的连接结果与 ghi 相连接，所以 \$s 的最终值为：abcdefghi。

(5) 多次自连接：

```
Local $s = ""  
$s &= "abc"  
$s &= "def"  
$s &= "ghi"
```

\$s 被定义为空字符串；

\$s 先与 abc 进行自连接，此时 \$s 的值为 abc；

\$s 再与 def 进行自连接，此时 \$s 的值为 abcdef；

\$s 最后与 ghi 进行自连接，\$s 的最终值为：abcdefghi。

连接运算时的数据类型自动转化：

因连接运算符“&”在运算时，要求其左右两侧均为字符串类型值，所以当“&”任意一侧不为字符串类型时，则自动启用数据类型转化。举例：

(1) 直接转化值

```
Local $s = "abc" & 123
```

因 123 为数值类型值，所以在执行“&”运算时，数值 123 会先自动转化为字符串“123”，而后再与 abc 执行连接运算，\$s 的最终值为：abc123。

(2) 转化变量值

```
Local $n = 123  
Local $s = "abc" & $n
```

因变量 \$n 的值 123 为数值类型，所以在执行“&”运算时，数值 123 会先自动转化为字符串“123”，而后再与 abc 执行连接运算，\$s 的最终值为：abc123。值得注意的是，仅仅是 \$n 的值 123 进行了类型转化，而非变量 \$n 进行了类型转化，换句话说，在执行后 \$n 的值依旧是数值 123，而非字符串“123”。

本小节只以数值型数据为例讲解了连接运算时的数据类型自动转化，更多数据类型与字符串的转化请参见 1.3.4 节，本节不再赘述。

### 2.3.3 字符串的两种引号

字符串可使用一对英文双引号或单引号进行包含，且英文双引号或单引号具有完全相同的功能，那为什么 AutoIt v3 中要有两种引号？目的是什么？

我们先放下这个问题，看另外一个问题，而后大家就会明白单双引号的存在意义了。

字符串是由字符构成的，英文字母、阿拉伯数字、常见符号（如！、@、#、\$、%）、各国语言（如中文）、各国文字符号（如中文逗号“，”、中文句号“。”）等等均是字符，理所当然的，英文双引号、英文单引号也是字符。那么，如果一个字符串中本身就存在英文引号，那我们应当如何表达这个字符

串？例如：

```
He said "Hello"
```

这句话中本身就包含了英文双引号，如果我们按照一般的字符串变量定义方法：

```
Local $s = "He said "Hello""
```

这行语句存在错误，He 左侧的双引号与 Hello 左侧的双引号形成了第一个配对，而 Hello 右侧的两个双引号又形成了第二个配对，那么真正的字符串内容并没有被一对引号包含起来，所以在运行时会发生报错现象。

对于这类字符串本身带有引号的情况，一般的处理方法是将字符串内的引号书写两次，以告知编译器这是字符串内的引号而不是用于标记字符串的引号。不过值得注意的是，字符串内被书写两次的引号仍会被当做一个处理，只是以这种方式告知编译器罢了。解决方法：

```
Local $s = "He said ""Hello"""
```

注意：

- (1) 字符串两侧是用于包含字符串的双引号；
- (2) Hello 左侧的一个双引号，写成两个，以标记这是一个字符串内的双引号；
- (3) Hello 右侧的一个双引号，写成两个，以标记这是一个字符串内的双引号。

通过这种方法，成功的解决了字符串内本身包含引号的问题。同理，如果字符串内是单引号，可采用书写两次单引号的方式来解决问题，方法类同。不过这种解决方法，令字符串变得不直观且难于阅读，并容易出现引号配对错误。

我们可以采用混用两种引号的方式来解决。所谓混用引号，即使用与字符串中引号不同的另一种引号来包含字符串，例如：

```
Local $s = 'He said "Hello"'
```

这个语句中，使用单引号对字符串进行了包含，即使用单引号来表示这是一个字符串。而单引号并不与字符串内的双引号产生匹配，那么只有字符串左侧开头处的单引号与字符串右侧结尾处的单引号形成匹配，所以不会产生与字符串内引号发生错误匹配的问题。

这种解决方法更为直观简洁，不影响字符串本身的可读性，不易发生引号配对错误，又不用改写字符串，是一种推荐的解决方式。

在面对实际问题时，灵活运用两种引号，可以使问题得到充分的解决。

### 2.3.4 字符的 ASCII 代码值

#### 1、什么是 ASCII 代码

ASCII 是基于拉丁字母的一套编码，主要用于显示英文和其他西欧语言，ASCII 是现今最通用的单字节编码系统。ASCII 代码一共有 256 个 ( 0~255 )，其中标准 ASCII 代码 128 个 ( 0~127 )，拓展 ASCII 代码 128 个 ( 128~255 )，一般我们只使用标准 ASCII 值。

简而言之，ASCII 代码就是一套编码，使用 0~255 这 256 个数字每个代表一个字符，例如 ASCII 代码 35 代表 "#"、ASCII 代码 51 代表 "3"、ASCII 代码 69 代表 "E"。

## 2. 标准 ASCII 代码表

控制符（一般不可被直接输出为字符，但均具有一定的控制功能）

字符	十进制值	说明
NUL	0	空 ( Null ) 字符
SOH	1	标题的开始
STX	2	文本的开始
ETX	3	文本的结束
EOT	4	结束传输
ENQ	5	查询
ACK	6	应答
BEL	7	铃声
BS	8	退格 ( Backspace )
HT	9	水平制表符 ( Tab )
LF	10	换行
VT	11	垂直制表符
FF	12	换页
CR	13	回车
SO	14	移出 ( Shift Out ), 切换字符集
SI	15	移入 ( Shift In ), 恢复默认字符集
DLE	16	数据通讯换码
DC1	17	设备控制符 1
DC2	18	设备控制符 2
DC3	19	设备控制符 3
DC4	20	设备控制符 4
NAK	21	否定应答
SYN	22	同步闲置
ETB	23	传送块结束
CAN	24	取消行
EM	25	媒体结束
SUB	26	替换
ESC	27	退出
FS	28	文件分隔符
GS	29	组分隔符
RS	30	记录分隔符
US	31	单元分隔符
DEL	127	删除

标准字符（列表中符号均为英文符号）

字符	十进制值	说明
	32	空格
!	33	叹号
"	34	双引号

#	35	#号
\$	36	美元符号
%	37	百分比符号
&	38	连字符号
'	39	单引号
(	40	左圆括号(左小括号)
)	41	右圆括号(右小括号)
*	42	星号(乘号)
+	43	加号
,	44	逗号
-	45	减号
.	46	点
/	47	斜线(除号)
0	48	数字0
1	49	数字1
2	50	数字2
3	51	数字3
4	52	数字4
5	53	数字5
6	54	数字6
7	55	数字7
8	56	数字8
9	57	数字9
:	58	冒号
;	59	分号
<	60	小于号(左尖括号)
=	61	等号
>	62	大于号(右尖括号)
?	63	问号
@	64	at 符号
A	65	大写字母A
B	66	大写字母B
C	67	大写字母C
D	68	大写字母D
E	69	大写字母E
F	70	大写字母F
G	71	大写字母G
H	72	大写字母H
I	73	大写字母I
J	74	大写字母J
K	75	大写字母K
L	76	大写字母L
M	77	大写字母M

N	78	大写字母 N
O	79	大写字母 O
P	80	大写字母 P
Q	81	大写字母 Q
R	82	大写字母 R
S	83	大写字母 S
T	84	大写字母 T
U	85	大写字母 U
V	86	大写字母 V
W	87	大写字母 W
X	88	大写字母 X
Y	89	大写字母 Y
Z	90	大写字母 Z
[	91	左方括号 ( 左中括号 )
\	92	反斜线
]	93	右方括号 ( 右中括号 )
^	94	插入符号
-	95	下划线
,	96	撇号
a	97	小写字母 a
b	98	小写字母 b
c	99	小写字母 c
d	100	小写字母 d
e	101	小写字母 e
f	102	小写字母 f
g	103	小写字母 g
h	104	小写字母 h
i	105	小写字母 i
j	106	小写字母 j
k	107	小写字母 k
l	108	小写字母 l
m	109	小写字母 m
n	110	小写字母 n
o	111	小写字母 o
p	112	小写字母 p
q	113	小写字母 q
r	114	小写字母 r
s	115	小写字母 s
t	116	小写字母 t
u	117	小写字母 u
v	118	小写字母 v
w	119	小写字母 w
x	120	小写字母 x

y	121	小写字母 y
z	122	小写字母 z
{	123	左大括号
	124	垂直线
}	125	右大括号
~	126	波浪线

面对这么多 ASCII 值，是不是已经眼花缭乱了呢？不用担心，不需要记住所有 ASCII 值，只需要记住部分常用值，其他值到用时再来查找也不迟。

常用部分仅有：

- (1) 0，代表空（并非空格或空字符串），通常这是一种特殊标记；
- (2) 9，代表水平制表符；
- (3) 10、13，分别代表换行和回车，它们无法直接写进字符串中，但可通过 ASCII 代码实现；
- (4) 48~59，分别代表阿拉伯数字 0~9；
- (5) 65~90、97~122，分别代表英文字母 A~Z 和 a~z。

### 3、字符与 ASCII 代码的相互转换 ( Chr、Asc )

在知道了 ASCII 代码与字符的对应关系后，我们可以使用 Chr 和 Asc 函数实现 ASCII 代码与字符的相互转换。

**(1) Chr 函数语法：**

```
Chr(<ASCII 代码>)
```

Chr 函数的参数为 ASCII 代码，返回值为 ASCII 代码所对应的字符串，例如：

```
Local $s = Chr(65)
```

\$s 的值为 A。

而如果给出的 ASCII 代码不在 0~255 的范围，则会返回一个空字符串。

**(2) Asc 函数语法：**

```
Asc("<字符>")
```

Asc 函数的参数为单个字符，返回值为字符所对应的 ASCII 值，例如：

```
Local $n = Asc("A")
```

\$n 的值为 65。

值得注意的是，Asc 函数的参数部分是单个字符，这是因为 ASCII 代码总是与单个字符相互对应，所以在使用 Asc 函数时应严格遵守这个规则。而如果真的错写了多个字符，如 Asc("AB")，函数默认只抓取字符串的第一个字符进行转化，所以 Asc("AB") 与 Asc("A") 的结果相同均为 65，但应极力避免这种错误的写法。

借助 Chr 和 Asc 函数，我们轻松实现了 ASCII 代码与字符的相互转化。

#### 2.3.5 字符串的换行

## 1、使用 Chr(10)、Chr(13)实现换行

如果我们要输出一个多行的字符串，应该怎么写？例如输出：

```
我是第一行
我是第二行
```

一般而言，换行是通过换行符实现的，先来了解一下什么是换行符。

在上一节中，我们了解到 ASCII 代码 10 代表换行符，13 代表回车符，在不同的操作系统中，换行的表示方式不同：

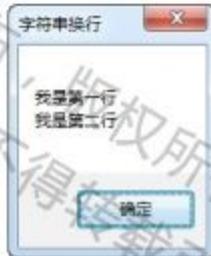
- (1) 在 Dos 或 Windows 中，使用回车符+换行符的方式表示换行；
- (2) 在 Unix 或 Linux 中，使用换行符来表示换行；
- (3) 在 MacOS 中，使用回车符来表示换行。

我们写的程序运行于 Windows 系统中，所以一般使用回车符+换行符的方式表示换行。结合上一节学习的 Chr 函数，则 Chr(13)&Chr(10)即可表达换行的意图。

那么代码书写为：

```
Local $s = "我是第一行" & Chr(13) & Chr(10) & "我是第二行"
MsgBox(0, '字符串换行', $s)
```

效果如下图：



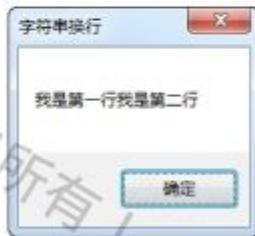
( 图 2-10 )

特别指出的是，字符串的换行与代码的换行是两回事，字符串的换行是令字符串分行显示，而代码的换行只是为了代码书写更为美观易读。不要试图通过使用代码换行的方式使字符串分行显示，完全达不到目的。

例如：

```
Local $s = "我是第一行" &
        "我是第二行"
MsgBox(0, '字符串换行', $s)
```

效果：



( 图 2-11 )

不过可以通过代码换行，令字符串分行的代码更为易读。

例如：

```
Local $s = "说明：" & Chr(13) & Chr(10) &_
    "1、请打开浏览器；" & Chr(13) & Chr(10) &_
    "2、请在地址栏输入：www.itiankong.com；" & Chr(13) & Chr(10) &_
    "3、请按下回车，等待页面打开。"
MsgBox(0, '字符串换行', $s)
```

效果：



(图 2-12)

上述代码，在字符串换行时同时使代码进行了换行，这样做可以使代码维护更容易，当修改第几行文本内容时，对应修改第几行代码即可。不过这只是个代码书写习惯，并不强求多行的字符串必须这样书写。

### 2、使用@CR、@LF、@CRLF 实现换行

总是书写 Chr(10)、Chr(13)挺麻烦的，有没有什么好的方法可以替代？有，我们可以使用@LF 来替代 Chr(10)，使用@CR 替代 Chr(13)，并可使用@CRLF 替代 Chr(13)&Chr(10)。

到这里就会有同学提问了，常量或变量不都是以 "\$" 开头的吗，怎么会有 "@" 开头的？以 "@" 开头的被称为“宏”，例如@CR、@LF 和@CRLF 就是三种宏。关于宏，之后章节会详细介绍，本节不过多讲解，大家目前只需要了解宏可以像普通变量一样使用，但不可对其直接赋值，就可以了。

那么刚才的代码可以改写为：

```
Local $s = "我是第一行" & @CRLF & "我是第二行"
MsgBox(0, '字符串换行', $s)
```

和

```
Local $s = "说明：" & @CRLF &_
    "1、请打开浏览器；" & @CRLF &_
    "2、请在地址栏输入：www.itiankong.com；" & @CRLF &_
    "3、请按下回车，等待页面打开。"
MsgBox(0, '字符串换行', $s)
```

所达到的效果与之前小节的效果是相同的，但书写更为简练。

### 2.3.6 影响字符串的特殊字符

一般而言，一个字符串是从最左侧第一个字符开始到最右侧最后一个字符结束。当输出字符串时，通常是将整个字符串里的所有字符一并输出。但有些字符会影响字符串的输出和判定，例如 ASCII 代码为 0 的“空”字符。

例如：

```
Local $s = 'ab' & 'cd'
MsgBox(0, "", $s)
```

输出结果为：

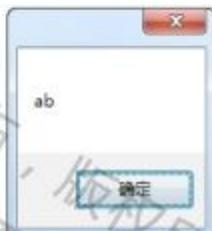


(图 2-13)

而：

```
Local $s = 'ab' & Chr(0) & 'cd'
MsgBox(0, "", $s)
```

输出结果为：



(图 2-14)

这是因为 MsgBox 在输出 \$s 的变量内容时，读取到 ASCII 代码为 0 的字符时就认为字符串结束了，剩余的部分没有输出。

其实在一般的日常使用中，很少会用到 Chr(0)，但本节是为了提醒大家有些特殊字符会影响字符串的正常输出，进而言之，可能会影响某些数据的正确性。所以，大家在处理字符串时一定要慎重，不要太随意。

### 2.3.7 字符串的相关函数

字符串的处理是一门学问，为了便于使用者良好的处理字符串，AutoIt v3 中提供了各种各样的字符串类函数。

#### 1、字符串的处理

##### (1) 截取：StringLeft、StringRight、StringMid

函数	语法
StringLeft 截取字符串左侧指定数量的字符	StringLeft("字符串",数量) 参数： 字符串：目标字符串 数量：截取的数量

	<p><b>返回值：</b> 所截取的字符串</p> <p><b>备注：</b> 数量超过总长时（超界），返回整个字符串 数量为负数时，返回一个空字符串</p>
StringRight 截取字符串右侧指定数量的字符	<p><b>StringRight("字符串",数量)</b></p> <p><b>参数：</b> 字符串：目标字符串 数量：截取的数量</p> <p><b>返回值：</b> 所截取的字符串</p> <p><b>备注：</b> 数量超过总长时（超界），返回整个字符串 数量为负数时，返回一个空字符串</p>
StringMid 从起始位置起，截取字符串中指定数量的字符	<p><b>StringMid("字符串",起始位置,[数量])</b></p> <p><b>参数：</b> 字符串：目标字符串 数量：截取的数量</p> <p><b>返回值：</b> 所截取的字符串</p> <p><b>备注：</b> 数量不指定时，返回自起始位置开始所有字符 起始位置超界时，返回一个空字符串 数量超界时，返回自起始位置开始所有字符</p>

例：

```
Local $s = "abcdef"

Local $s1 = StringLeft($s, 3)
;$s1 的值为"abc"，即"abcdef"左起 3 个字符

Local $s2 = StringRight($s, 3)
;$s2 的值为"def"，即"abcdef"右起 3 个字符

Local $s3 = StringMid($s, 2, 4)
;s3 的值为"bcde"，即"abcdef"第 2 个位置起 4 个字符

Local $s4 = StringMid($s, 2)
;$s4 的值为"bcdef"，即"abcdef"第 2 个位置起所有字符
```

### (2) 删除：StringTrimLeft、StringTrimRight

函数	语法
StringTrimLeft 删除字符串左侧指定数量的字符	<p><b>StringTrimLeft("字符串",数量)</b></p> <p><b>参数：</b> 字符串：目标字符串</p>

	<p>数量：删除的数量 返回值：     删除指定数量字符后的字符串 备注：     数量超过总长时（超界），返回一个空字符串</p>
StringTrimRight 删除字符串右侧指定数量的字符	<p>StringTrimRight("字符串",数量) 参数：     字符串：目标字符串     数量：删除的数量 返回值：     删除指定数量字符后的字符串 备注：     数量超过总长时（超界），返回一个空字符串</p>

例：

```
Local $s = "abcdef"
Local $s1 = StringTrimLeft($s, 2)
;$s1 的值为"cded"，即"abcdef"删除左侧 2 个字符后的字符串

Local $s2 = StringTrimRight($s, 2)
;$s2 的值为"abcd"，即"abcdef"删除右侧 2 个字符后的字符串
```

### ( 3 ) 删除空白符 : StringStripWS

函数	语法
StringStripWS 删除字符串中的空白符  ( 空白符包括 ASCII 代码为 0、9、10、11、12、13、32 的字符，即空字符、水平制表符、换行符、垂直制表符、换页符、回车符、空格符 )	<p>StringStripWS("字符串",标志)</p> <p>参数：     字符串：目标字符串     标志：         1，删除字符串左侧的空白符         2，删除字符串右侧的空白符         4，如有连续多个空白符，则只保留一个         8，删除字符串中的所有空白符（这将忽略其他标志）</p> <p>返回值：     删除指定空白符后的字符串</p>

虽然 StringStripWS 功能多多，但大多数时候我们只是用其清理字符串中多余的空格符。另外，标志中除了 8 以外，其他标志是可以叠加使用的，例如 1+2（或直接写 3）代表删除字符串左右两侧的空白符。

例：

```
Local $s = "Hello World"
Local $s1 = StringStripWS($s, 1)
;$s1 的值为"Hello World"，即"Hello World"去掉左侧空格后的字符串
```

## Let's AutoIt Plus

```
Local $s2 = StringStripWS($s, 2)
;$s2 的值为" Hello World"，即" Hello World "去掉右侧空格后的字符串
```

```
Local $s3 = StringStripWS($s, 1 + 2)
;$s3 的值为"Hello World"，即" Hello World "去掉左右两侧空格后的字符串
```

```
Local $s4 = StringStripWS($s, 4)
;$s4 的值为"Hello World"，即" Hello World "连续多个空格只保留一个后的字符串
```

```
Local $s5 = StringStripWS($s, 8)
;$s5 的值为"HelloWorld"，即" Hello World "去掉所有空格后的字符串
```

### (4) 大小写转换 : StringLower、StringUpper

函数	语法
StringLower 将字符串中所有大写字母转化为小写	StringLower("字符串") 参数： 字符串：目标字符串 返回值： 所有字母转为小写后的字符串
StringUpper 将字符串中所有小写字母转化为大写	StringUpper("字符串",数量) 参数： 字符串：目标字符串 返回值： 所有字母转为大写后的字符串

例：

```
Local $s = "AbCDef"

Local $s1 = StringUpper($s)
;$s1 的值为"ABCDEF"，即"AbCDef"中所有字母转为大写后的字符串

Local $s2 = StringLower($s)
;$s2 的值为"abcdef"，即"AbCDef"中所有字母转为小写后的字符串
```

## 2、字符串的操作

### (1) 长度 : StringLen

函数	语法
StringLen 获取字符串的长度	StringLen("字符串") 参数： 字符串：目标字符串 返回值： 字符串的长度

例：

```
Local $s = "abc123"

Local $l = StringLen($s)
;$l 的值为 6，即"abc123"的长度（总字符数）
```

## (2) 查找 : StringInStr

函数	语法
StringInStr 检查字符串是否包含特定子字符串	<pre>StringInStr("字符串", "子字符串"[, 区分大小写=0[, 出现位置=1[, 开始=1[, 数量]]]])</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>字符串：目标字符串</li> <li>子字符串：要搜索的字符串</li> <li>区分大小写：           <ul style="list-style-type: none"> <li>0 (默认)，不区分大小写</li> <li>1，区分大小写</li> <li>2，不区分大小写（使用基本/快速比较法）</li> </ul> </li> <li>出现位置：           <ul style="list-style-type: none"> <li>指定搜索第几次出现的位置，默认搜索第1次 (为负数则从右向左计数)</li> </ul> </li> <li>开始：           <ul style="list-style-type: none"> <li>从什么位置开始搜索，默认为第1个位置开始 (为负数则从右向左计数)</li> </ul> </li> <li>数量：           <ul style="list-style-type: none"> <li>指定仅搜索目标字符串中一定数量的字符，默认为全部</li> </ul> </li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回子字符串开始的位置</li> <li>失败：返回0， @error：</li> <li>1，“开始”或“出现位置”无效</li> </ul>

例：

```
Local $s = "Hello World! Hello Au3!"

Local $r1 = StringInStr($s, "hello")
;$r1 的值为 1，即第 1 次发现"hello"的位置是目标字符串的第 1 个位置

Local $r2 = StringInStr($s, "hello", 1)
;$r2 的值为 0，即没有在目标字符串中发现"hello"，  
原因是我们在设置时必须匹配大小写，而目标字符串中都是首字母大写的 Hello

Local $r3 = StringInStr($s, "hello", 0, 2)
;$r3 的值为 14，即第 2 次发现"hello"的位置是目标字符串的第 14 个位置

Local $r4 = StringInStr($s, "hello", 0, -1)
;$r4 的值为 14，即从右向左第 1 次发现"hello"的位置是目标字符串的第 14 个位置
```

```
Local $r5 = StringInStr($s, "hello", 0, 1, 7)
;$r5 的值为 14，即从第 7 个位置开始搜索，第 1 次发现"hello"的位置是第 14 个位置
```

```
Local $r6 = StringInStr($s, "au3", 0, 1, 1, 12)
;$r6 的值为 0，即没有发现"au3"，
;原因是设置了只搜索目标字符串的前 12 个字符，而前 12 个字符中没有"au3"子字符串
```

### (3) 替换 : StringReplace

函数	语法
StringReplace 替换字符串中的指定字符串	StringReplace("字符串", "搜索字符串/起始位置", "替换字符串"[数量=0][区分大小写=0])  参数： 字符串：目标字符串 搜索字符串/起始位置：指定将要替换的字符串或位置 替换字符串：用于替换的字符串 区分大小写： 0 (默认)，不区分大小写 1，区分大小写 2，不区分大小写 (使用基本/快速比较法)  返回值： 返回替换后的字符串

例：

```
Local $s = "Hello World! Hello Au3!"
```

```
Local $s1 = StringReplace($s, "Hello", "Hi")
;$s1 的值为"Hi World! Hi Au3!"，即字符串中所有的 Hello 替换成了 Hi
```

```
Local $s2 = StringReplace($s, 20, "ACC")
;$s2 的值为"Hello World! Hello ACC!"，即从第 20 个位置开始 ( Au3 的 A 所在位置 ) 将字符替换为 ACC
```

```
Local $s3 = StringReplace($s, "Hello", "Hi", 1)
;$s3 的值为"Hi World! Hello Au3!"，即替换字符串中的 Hello 到 Hi，但只替换 1 个
```

```
Local $s4 = StringReplace($s, "hello", "Hi", 0, 1)
;$s4 的值为"Hello World! Hello Au3!"，没有任何替换，
;原因是设置了区分大小写，而原文中不包含小写的 hello
```

### (4) 比较 : StringCompare

函数	语法
StringCompare 比较两个字符串	StringCompare("字符串 1", "字符串 2"[区分大小写=0])  参数： 字符串 1：参与比较的第一个字符串 字符串 2：参与比较的第二个字符串

	<p>区分大小写：</p> <ul style="list-style-type: none"> <li>0 (默认), 不区分大小写</li> <li>1, 区分大小写</li> <li>2, 不区分大小写 ( 使用基本/快速比较法 )</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>0, 字符串相等</li> <li>&gt;0, 字符串 1 大于字符串 2</li> <li>&lt;0, 字符串 1 小于字符串 2</li> </ul>
--	---

例：

```
Local $s1, $s2

$s1 = "Alex"
$s2 = "Bell"
Local $r1 = StringCompare($s1, $s2)
;$r1<0, 因为"Alex"的英文排序先于"Bell"

$s1 = "Skye"
$s2 = "GMan"
Local $r2 = StringCompare($s1, $s2)
;$r2>0, 因为"Skye"的英文排序晚于"GMan"

$s1 = "Tom"
$s2 = "tom"
Local $r3 = StringCompare($s1, $s2)
;$r3=0, 因为在不区分大小写的前提下"Tom"与"tom"是相同的
Local $r4 = StringCompare($s1, $s2, 1)
;$r4<0, 因为在区分大小写的前提下"Tom"的英文排序先于"tom"
```

### (5) 分割 : StringSplit

函数	语法
<b>StringSplit</b> 按照指定的分隔符分割字符串 为数组	<code>StringSplit("字符串","分隔符"[标志=0])</code> <p>参数：</p> <ul style="list-style-type: none"> <li>字符串：目标字符串</li> <li>分隔符：一个或多个字符作为分隔符 ( 区分大小写 )</li> <li>标志：</li> </ul> <ul style="list-style-type: none"> <li>0 (默认), 分隔符字符串中的每个字符都作为分割符</li> <li>1, 分隔符字符串整体作为分隔符</li> <li>2, 从元素 0 开始存储分割后的字符串</li> </ul> <p>返回值：</p> <p>返回一个数组，元素 0 为分割后子字符串的数量，从元素 1 开始为每个分割后的子字符串。若未发现分隔符，则元素 0 值为 1，元素 1 为整个字符串。</p>

例：

```
Local $s
```

```
$s = "Una|Skye|GMan"

Local $array1 = StringSplit($s, "|")
;$array1 : 元素 0 为 3、元素 1 为"Una"、元素 2 为"Skye"、元素 3 为"GMan"
;即以 "|" 为分隔符，分割后一共有 3 个子字符串，总数存储于元素 0，子字符串依次存储于元素 1、2、3 中

Local $array2 = StringSplit($s, "|", 2)
;$array2 : 元素 0 为"Una"、元素 1 为"Skye"、元素 2 为"GMan"
;即以 "|" 为分隔符，将分割后的子字符串从元素 0 开始依次进行存储

Local $array3 = StringSplit($s, ",")
;$array3 : 元素 0 为 1、元素 1 为"Una|Skye|GMan"
;即没有找到指定分隔符 ","，则元素 0 为 1、元素 1 为整个字符串

$s = "Hello\nWorld\n!"

Local $array4 = StringSplit($s, "\n", 1)
;$array4 : 元素 0 为 3、元素 1 为"Hello"、元素 2 为"World"、元素 3 为"!"
;即以 "\n" 整体为分隔符，分割后一共有 3 个子字符串，
;总数存储于元素 0，子字符串依次存储于元素 1、2、3 中
```

### 3、@CR 的处理

函数	语法
StringAddCR 在字符串中所有的换行符@LF 前添加回车符@CR	StringAddCR("字符串") 参数： 字符串：需要添加回车符的字符串 返回值： 目标字符串所有@LF 前添加@CR 后的字符串
StringStripCR 删除字符串中所有的回车符 @CR	StringStripCR("字符串") 参数： 字符串：需要删除回车符的字符串 返回值： 删除字符串中所有@CR 后的字符串

例：

```
Local $s = "Hello World!" & @LF

Local $s1 = StringAddCR($s)
;$s1 的值为"Hello World!"后@CRLF，即$s 的字符串中@LF 前添加@CR

Local $s2 = StringStripCR($s1)
;$s2 的值为"Hello World!"后@LF，即$s1 的值去掉@CRLF 中的@CR
```

### 4、字符串与 ASCII 代码的转换

函数	语法
StringToASCIIArray 转换字符串为包含对应 ASCII 代码的数组	<p>StringToASCIIArray("字符串", [开始=0[结束=0[编码=0]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>字符串：目标字符串</li> <li>开始：从字符串的哪个字符开始进行转换（默认为 0）</li> <li>结束：至字符串的哪个字符为止结束转换</li> <li>编码：</li> <ul style="list-style-type: none"> <li>0（默认），使用 UTF-16 字符集</li> <li>1，使用 ANSI 字符集</li> <li>2，使用 UTF-8 字符集</li> </ul> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回数组，元素依次为字符串中每个字符的 ASCII 码</li> <li>失败：返回空字符串</li> </ul> <p>备注：</p> <ul style="list-style-type: none"> <li>“开始”和“结束”为 0 基索引，即第 1 个位置的索引为 0，第二个位置的索引为 1，以此类推</li> </ul>
StringFromASCIIArray 转换 ASCII 代码数组到字符串	<p>StringFromASCIIArray(数组, [开始=0[结束=-1[编码=0]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>数组：包含 ASCII 代码的数组</li> <li>开始：从数组的哪个元素开始进行转换（默认为 0）</li> <li>结束：至数组的哪个元素为止结束转换</li> <li>编码：</li> <ul style="list-style-type: none"> <li>0（默认），使用 UTF-16 字符集</li> <li>1，使用 ANSI 字符集</li> <li>2，使用 UTF-8 字符集</li> </ul> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回转化后的字符串</li> <li>失败：返回空字符串，并设置@error</li> </ul> <p>@error：</p> <ul style="list-style-type: none"> <li>1，“数组”不是数组</li> <li>2，“开始”的索引无效</li> </ul>

例：

```
Local $s1 = "abc"
Local $array = StringToASCIIArray($s1)
Local $s2 = StringFromASCIIArray($array)
```

\$array 为由\$s1 转换得到的数组，元素 0 为 “a” 的 ASCII 代码 97，元素 1 为 “b” 的 ASCII 代码 98，元素 2 为 “c” 的 ASCII 代码 99；

\$s2 为由\$array 转换得到的字符串，为"abc"。

## 5、字符串的检测

函数	语法
StringIsAlNum 检查字符串是否仅包含字母数	<p>StringIsAlNum("字符串")</p> <p>参数：字符串：目标字符串</p>

字 ( alphanumeric ) 字符	返回值 : 是 : 1 , 否 : 0
StringIsAlpha 检查字符串是否仅包含字母 ( alphabetic ) 字符	StringIsAlpha("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsASCII 检查字符串是否仅包含 ASCII 码为 0 到 127 的字符	StringIsASCII("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsDigit 检查字符串是否仅包含十进制数字 ( 0~9 ) 字符	StringIsDigit("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsXDigit 检查字符串是否仅包含十六进制数字 ( 0~9 , a~f ) 字符	StringIsXDigit("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsFloat 检查字符串是否为浮点数	StringIsFloat("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsInt 检查字符串是否为整数	StringIsInt("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsLower 检查字符串是否仅包含小写字母字符	StringIsLower("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsUpper 检查字符串是否仅包含大写字母字符	StringIsUpper("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0
StringIsSpace 检查字符串是否仅包含空白符	StringIsSpace("字符串") 参数 : 字符串 : 目标字符串 返回值 : 是 : 1 , 否 : 0

例 :

```
Local $r

$r = StringIsAlNum("abcABC123"); -> 1, 字符串仅包含字母数字
$r = StringIsAlNum("abc,ABC_123"); -> 0, 字符串包含了逗号、下划线等非字母数字

$r = StringIsAlpha("abcABC"); -> 1, 字符串仅包含字母
$r = StringIsAlpha("abcABC123"); -> 0, 字符串包含了数字, 非纯字母

$r = StringIsASCII("abAB12,\#\@"); -> 1, 字符串中字符均为 ASCII 代码 0~127 的字符
$r = StringIsASCII("abc123 %o±÷"); -> 0, 字符串中包含了 "%o"、"±"、"÷" 等 ASCII 代码非 0~127 的字符

$r = StringIsDigit("123"); -> 1, 字符串仅包含了十进制数字 ( 0~9 )
$r = StringIsDigit("abc123"); -> 0, 字符串中包含了英文字母
```

```

$r = StringIsXDigit("12ac6f") ; -> 1, 字符串仅包含了十六进制数字 ( 0~9、a~f )
$r = StringIsXDigit("12ac6fgh") ; -> 0, 字符串包含了非十六进制数字

$r = StringIsFloat("1.2") ; -> 1, 字符串为浮点数
$r = StringIsFloat("12") ; -> 0, 字符串为整数

$r = StringIsInt("12") ; -> 1, 字符串为整数
$r = StringIsInt("1.2") ; -> 0, 字符串为浮点数

$r = StringIsLower("abc") ; -> 1, 字符串中所有字母均为小写
$r = StringIsLower("aBc") ; -> 0, 字符串中包含了大写字母

$r = StringIsUpper("ABC") ; -> 1, 字符串中所有字母均为大写
$r = StringIsUpper("aBC") ; -> 0, 字符串中包含了小写字母

$r = StringIsSpace(" " & @CR & @LF & @CRLF & @TAB) ; -> 1, 字符串仅包含了空白字符
$r = StringIsSpace("ab12" & " " & @CR & @LF & @CRLF & @TAB) ; -> 0, 字符串中包含了非空白字符

```

### 2.3.8 字符串类程序举例

#### 1、依次输出 26 个英文字母

**算法分析：**

- ( 1 ) 第一感觉，这类题目需要用循环来实现，但循环变量一般只能是数值，不能为字符；
- ( 2 ) 而 ASCII 代码是数值，ASCII 代码 65 到 90 恰巧对应 A 到 Z ；
- ( 3 ) 可通过使用 Chr 函数将 ASCII 代码转化为字符。
- ( 4 ) 那么，创建一个循环，使循环变量从 65 递增至 90，而后逐个将数值转化为字符即可。

**程序代码：**

```

_Main()
Exit

Func _Main()
;定义 $s 存储输出结果
Local $s = ""
;定义 $i 用于循环
Local $i
;$i 的值从 65 ( A 的 ASCII 代码 ) 开始递增至 90 ( Z 的 ASCII 代码 )
For $i = 65 To 90
    $s &= Chr($i);Chr($i) 即获取 ASCII 代码对应的字符
Next
;输出
MsgBox(0, "输出 A-Z", $s)
EndFunc  ;==> _Main

```

运行图例：



(图 2-15)

## 2、字符串函数练习

已知字符串：“Skyfree and ITianKong.Com”

- (1) 获取字符串长度；
- (2) 截取字符串中的“Skyfree”；
- (3) 截取字符串中的“ITianKong.Com”；
- (4) 将“Skyfree”替换为“Una”；
- (5) 判断字符串中是否包含“ITianKong”子字符串；
- (6) 截取字符串中的“free”。

算法分析：

- (1) 获取字符串长度使用函数 StringLen；
- (2) 截取字符串使用函数 StringLeft、StringRight 或 StringMid；或可变通方法，删除多余的字符，即可得到想要的字符串，删除字符串使用函数 StringTrimLeft、StringTrimRight；
- (3) 替换字符串中的特定子字符串使用函数 StringReplace；
- (4) 查找字符串中的特定子字符串使用函数 StringInStr；
- (5) 截取特定段使用 StringMid，并配合使用 StringInStr 确定字符串起始或结束位置即可。

程序代码：

```
_Main()
Exit

Func _Main()
;定义目标字符串$s
Local $s = 'Skyfree and ITianKong.Com'
;定义$str用于存储输出信息
Local $r = ""
$r &= '字符串为: ' & $s & @CRLF & @CRLF

;获取字符串长度
Local $l
;使用 StringLen 函数获取字符串长度
$l = StringLen($s)
$r &= '[1]字符串共包含' & $l & '个字符' & @CRLF

;截取字符串"Skyfree"
```

```

Local $r1
;方法 1，使用 StringLeft 函数直接从左侧截取 7 个字符
$r1 = StringLeft($s, 7)
;方法 2，使用 StringMid 函数获取第 1 个字符起之后的 7 个字符
$r1 = StringMid($s, 1, 7)
;方法 3，使用 StringTrimRight 函数删除右侧 18 个字符
$r1 = StringTrimRight($s, 18)
$r &= '[2]截取字符串中的"Skyfree": ' & $r1 & @CRLF

;截取字符串中的"ITianKong.Com"
Local $r2
;方法 1，使用 StringRight 函数直接从右侧截取 6 个字符
$r2 = StringRight($s, 6)
;方法 2，使用 StringMid 函数获取第 13 个字符起之后的 6 个字符
$r2 = StringMid($s, 13, 6)
;方法 3，使用 StringTrimLeft 函数删除左侧 12 个字符
$r2 = StringTrimLeft($s, 12)
$r &= '[3]截取字符串中的"ITianKong.Com": ' & $r2 & @CRLF

;获取将"Skyfree"为"Una"后的字符串
Local $r3
;使用 StringReplace 函数将字符串中的"Skyfree"替换为"Una"
$r3 = StringReplace($s, "Skyfree", "Una")
$r &= '[4]将"Skyfree"为"Una": ' & $r3 & @CRLF

;判断字符串中是否存在"ITianKong"子字符串
;使用 StringInStr 函数判定字符串中是否包含"ITianKong"
;若包含，则返回字符串的位置（>0 正整数），而非 0 转化为布尔值 True，所以执行第一分支
;若不包含，则返回 0，而 0 转化为布尔值 False，所以执行第二分支
If StringInStr($s, "ITianKong") Then
    $r &= '[5]字符串汇中包含"ITianKong"子字符串' & @CRLF
Else
    $r &= '[5]字符串汇中不包含"ITianKong"子字符串' & @CRLF
EndIf

;截取字符串"free"
Local $r4
;获取从左至右第 1 个 f 的位置
Local $p = StringInStr($s, 'f', 0, 1)
;获取从左至右第 2 个 e 的位置
Local $q = StringInStr($s, 'e', 0, 2)
;使用 StringMid 函数获取第 1 个 f 到第 2 个 e 之间的字符
$r4 = StringMid($s, $p, $q - $p + 1)
$r &= '[6]截取字符串中的"free": ' & $r4 & @CRLF

```

```
;输出  
MsgBox(0, '字符串函数练习', $r)  
EndFunc ;==>_Main
```

运行图例：



( 图 2-16 )

### 3、将三个字符串按照大小（字母优先）顺序进行排序

算法分析：

- ( 1 ) 字符串比较使用函数 StringCompare；
- ( 2 ) 三字符串排序思路与三数字排序思路相同（可参见 1.5.6 第 3 题）；
  - 假设有三个位置（一号位、二号位、三号位）分别放置三个字符串
  - 将一号位与二号位上的字符串相比较，将较小的放置到一号位，较大的放置到二号位
  - 再将一号位与三号位上的字符串相比较，将较小的放置到一号位，较大的放置到三号位
  - 至此，一号位上的字符串是三个位置上最小的
  - 然后二号位与三号位上的字符串相比较，将较小的放置到二号位，较大的放置到三号位
  - 至此，一号位、二号位、三号位上的字符串，按照从小到大的顺序排列完成
- ( 3 ) 因交换两变量值要在此问题中使用三次，写为函数更便于多次调用。

程序代码：

```
_Main()  
Exit  
  
Func _Main()  
;设定三字符串值  
Local $s1 = 'Una', $s2 = 'GMan', $s3 = 'Skye'  
  
;如果$s1 值优先于$s2，则交换$s1 和$s2 的值  
If StringCompare($s1, $s2) > 0 Then _Exchange($s1, $s2)  
;如果$s1 值优先于$s3，则交换$s1 和$s3 的值  
If StringCompare($s1, $s3) > 0 Then _Exchange($s1, $s3)  
;如果$s2 值优先于$s3，则交换$s2 和$s3 的值  
If StringCompare($s2, $s3) > 0 Then _Exchange($s2, $s3)
```

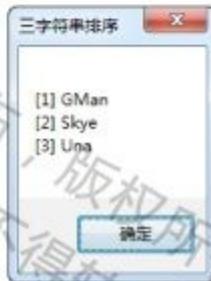
```

;排序
MsgBox(0, '三字符串排序', _
'[1] ' & $s1 & @CRLF &_
'[2] ' & $s2 & @CRLF &_
'[3] ' & $s3)
EndFunc  ;==>_Main

Func _Exchange(ByRef $xs1, ByRef $xs2)
;利用中间变量$ts，交换$xs1 和$xs2 的值
Local $ts
$ts = $xs1
$xs1 = $xs2
$xs2 = $ts
EndFunc  ;==>_Exchange

```

运行图例：



(图 2-17)

## 2.4 代码书写规范

AutoIt v3 语法宽松，便于初学者快速掌握，但宽松语法不利于养成好的代码书写习惯，而差的代码书写习惯又往往使后期代码维护变得繁琐。所以，即便 AutoIt v3 并不过于强调语法，我们也必须建立起一套代码书写规范，以利于代码交流、后期维护、多人协同等等。

刚接触代码规范时，特别是对于小程序，会感觉代码书写变得“麻烦”，但仍请保持规范，以时时刻刻培养自己的代码书写习惯。

### 1、代码的整体书写规范

(1) 代码整体使用函数化书写方式(详见 2.1.6 节)。每个程序必须有一个主函数，程序的运行由主函数开始，主函数内调用各子函数实现各独立功能，主函数运行结束后程序退出。

形如：

```

_Main()
Exit

```

```
Func _Main()
;...
_Function1()
;...
_Function2()
;...
EndFunc

Func _Function10
;...
EndFunc

Func _Function20
;...
EndFunc
```

- (2) 每个子函数应只实现一个独立功能，便于代码维护和函数移植。  
(3) 函数外仅用于定义全局变量或全局常量，可通过执行函数为全局变量或全局常量赋值，但除主函数外不再独立执行任何语句或函数。

形如：

```
Global $g_Var1 = 10
Global $g_Var2 = _GetTempDir()

_Main()
Exit

Func _Main()
;...
EndFunc

Func _GetTempDir()
;...
EndFunc
```

## 2、变量与常量的命名规范

- (1) 变量或常量，必须是以“\$”开头的，字母、数字、下划线组合，其名称能表达其作用。  
(2) 所有变量必须先定义后使用，可以在代码顶部位置加入下述代码以约束：

```
AutoItSetOption("MustDeclareVars", 1)
```

或简写为：

```
Opt("MustDeclareVars", 1)
```

- (3) 对于全局变量，使用“g\_”前缀，以与局部变量进行区分，例如：

```
Global $g_Var1, $g_Var2
```

- (4) 数组类型变量，使用“a\_”前缀，以与基本类型的变量进行区分，例如：

```
Local $a_Info, $a_Num
```

(5) 对于用于临时存储数据的变量，使用“t\_”前缀，例如：

```
Local $t_Val
```

(6) 多个前缀可以叠加使用，例如：

```
Local $ta_McStr ;临时数组
```

```
Global $ga_Config ;全局数组
```

(7) 对于一般常量，建议使用全大写方式，以与变量区分，例如：

```
Local Const $MAX = 5
```

(8) 对于全局常量，除了使用全大写方式，建议使用“\_”前缀，以与局部常量区分，例如：

```
Global Const $_WINDIR = "C:\Windows"
```

(9) 如果不喜欢使用大写命名常量，可以使用“c\_”前缀，与变量区分。

```
Local Const $c_Max = 5
```

```
Global Const $gc_WinDir = "C:\Windows"
```

(10) 如果是通用函数相关的全局变量（常量），为防止与其他代码的全局变量冲突，可将功能类型作为分类前缀，分类前缀与变量名之间使用双下划线“\_”以更明确的标注，例如实现 7z 压缩解压功能的全局变量：

```
Global $g_7zip_DllFilePath
```

```
Global $g_7zip_DllHandle
```

(11) 其他常用的简单变量：

- \$i、\$j、\$k，一般用于循环结构控制
- \$n：数值
- \$s：字符串，\$ts：临时字符串
- \$r：函数返回值
- \$f：文件句柄（后续章节学习）

### 3. 函数的命名规范

(1) 函数名，必须是以字母或下划线开头的，字母、下划线、数字组合，其名称能表达其作用。

(2) 自定义函数应以“\_”开头，以与标准函数相区别，例如：

```
Func _Test()
```

```
; ...
```

```
EndFunc
```

(3) 一个函数应只表达一个独立功能，并有完善的返回值、错误值机制。

(4) 一般而言，子函数只完成特定的功能，而子函数内一般不包括用户交互提示（如 MsgBox）、程序退出语句（如 Exit）等，这些功能应由主函数根据子函数返回值实现。

(5) 如果函数具有通用功能，为防止与其他代码的函数名冲突，可将功能类型作为分类前缀，分类前缀与变量名之间使用双下划线“\_”以更明确的标注，例如实现 7z 压缩解压功能的函数：

```
Func _7zip_Compress()
```

```
; ...
```

```
EndFunc
```

```
Func _7zip_Extrac()
;
EndFunc
```

## 2.5 文件包含

### 1、#include

有时候，我们需要将另一个代码文件中的全部代码包含进当前的代码，例如将一系列实现特定功能的函数写进某个代码文件中，当我们需要其中某个功能时，可以直接将这个代码文件包含进来，而无需去复制其中某个或某几个函数。

例如我们之前接触过的\_ArrayDisplay 函数，使用前必须先 “#include <Array.au3>”，脚本文件 Array.au3 中包含了各种与数组相关功能的函数，将其包含进来，相当于将其中所有内容插入到 “#include <Array.au3>” 这行语句的位置上。

**#include 的语法：**

```
#include <脚本文件名>
```

或：

```
#include "脚本文件路径"
```

**说明：**

(1) 如果使用尖括号 ( <> )，则优先在标准库目录中搜索指定的脚本文件，标准库目录一般位于安装目录下 Include 目录中 ( 如 D:\AutoIt3\Include )。

(2) 如果使用英文引号 ( " " )，则优先在当前目录搜索指定的脚本文件。

(3) 除了标准库目录，还可以创建自定义库目录，由注册表键 “HKCU\Software\AutoIt v3\AutoIt” 的 Include 键值决定。

(4) 尖括号时搜索的完整优先级顺序：标准库目录、自定义库目录、当前目录。

(5) 双引号时搜索的完整优先级顺序：当前目录、自定义库目录、标准库目录。

(6) 使用相对目录时可使用 “.\” 指代当前目录、“..\” 指代上层目录，例如：

```
#include ".\test.au3" ;包含当前目录下的 test.au3，与#include "test.au3"作用无差别
```

```
#include "..\test.au3" ;包含上层目录下的 test.au3
```

```
#include "..\..\MyInclude\test.au3" ;包含上上层目录下 MyInclude 目录下的 test.au3
```

(7) 亦可使用绝对目录，使用绝对目录时尖括号和引号没有区别，但不推荐这样做，不利于代码的移植。

(8) #include 一般写在代码的顶部，不要尝试在语句或函数中使用#include 语句。

**举例：**

将 1.au3 和 2.au3 放置在相同目录下，运行 1.au3：

1.au3

```
#include "2.au3"
```

```

_Main()
Exit

Func _Main()
Local $a = 1, $b = 2
.Exchange($a, $b)
MsgBox(0, "", "$a=" & $a & @CRLF & "$b=" & $b)
EndFunc

```

2.au3

```

Func _Exchange(ByRef $a, ByRef $b)
Local $t = $a
$a = $b
$b = $t
EndFunc

```

运行结果为 \$a=2, \$b=1, 即 \$a 和 \$b 的值完成了互换。虽然 1.au3 中并没有 \_Exchange 函数, 但因为 1.au3 中包含了 2.au3, 相当于把 2.au3 的内容全部插入到了 1.au3 中, 所以 1.au3 中就有了 \_Exchange 函数。

## 2、#include-once

与 #include 的包含概念不同, #include-once 是用于标记当前脚本仅被包含一次。为什么会有这样的标记呢? 举个例子大家就明白了。

将 1.au3、2.au3、a.au3、b.au3 位于同目录, 运行 1.au3:

1.au3

```

#include "a.au3"
#include "b.au3"

_Main()
Exit

Func _Main()
;
EndFunc

```

a.au3

```

#include "2.au3"

Func _TestA()
;
EndFunc

```

b.au3

```

#include "2.au3"

```

```
Func _TestB()
;
EndFunc
```

2.au3

```
Global Const $_TIMEOUT = 5

Func _Timer()
;
EndFunc
```

上述 4 个代码文件中，a.au3 包含了 2.au3，b.au3 也包含了 2.au3，而 1.au3 包含了 a.au3 和 b.au3。那么对于 1.au3 而言，相当于包含了 a.au3 一次、包含了 b.au3 一次、包含了 2.au3 两次。当 2.au3 被包含两次时，常量 \$\_TIMEOUT 就会被定义两次，函数 \_Timer 也会被定义两次，但常量、函数均是不可重复进行定义的，所以程序会报错，不可运行。

为了避免这种情况的发生，将 “#include-once” 加入 2.au3 的顶部：

```
#include-once

Global Const $_TIMEOUT = 5

Func _Timer()
;
EndFunc
```

这样，当 1.au3 发现 2.au3 被多次包含后，会自动只保留其中一次包含，其余的多次包含将被自动舍弃。由此，顺利解决了某个代码文件被多次包含而产生的报错问题。

## 2.6 宏

宏对我们来说已经不是一个陌生的概念了，之前接触过的 @CR、@LF 等都是宏。宏可以理解为是一种特殊的变量，除了以“@”开头这点以外，与普通变量最大的区别是宏是只读的，即只可使用其值却不可对其赋值。虽然少数宏可以通过函数改变其值（如使用 SetError 函数改变 @error 的值），但决不可像对变量赋值那样对宏赋值（如 @Error=1 是错误的）。

值得注意的是，除了某些表达固定信息的宏以外（例如 @CRLF 代表换行），大多数宏的值是会根据系统环境不同而自动改变的，这种改变在 AutoIt v3 程序运行时自动执行，无需人为干预。例如表达计算机名的宏 @ComputerName，在不同计算机名的 PC 上其值是不同的。

在了解了宏的基本概念后，下面我们来详细学习一些常见宏。注意，这并不是所有宏，其他宏将在今后的学习中逐步了解。

### 2.6.1 AutoIt 相关宏

#### (1) @Compiled

@Compiled 用于判断当前脚本是否被编译了，如已被编译则值为 1，否则为 0。例如：

```
MsgBox(0, "", @Compiled)
```

直接运行此脚本时值为 0，而编译后再运行时值为 1。

#### ( 2 ) @Error、@Extended

@Error 和@Extended 用于标注函数执行后的错误信息，可使用函数 SetError 来改变两者值，详见 2.1.3 节 4 ( 2 )。

#### ( 3 ) @NumParams

@NumParams 用于表示函数的实参数量，例如：

```
_Main()
Exit

Func _Main()
    Local $r1 = _Test(1, 2)
    MsgBox(0, "", $r1)
    Local $r2 = _Test(1)
    MsgBox(0, "", $r2)
EndFunc

Func _Test($Parm1, $Parm2 = 2)
    Return @NumParams
EndFunc
```

\_Test 函数有两个形参，其中第一个形参无默认值，第二个形参有默认值。\_Main 函数中第一次调用\_Test 函数时，给出了两个实参，所以 \$r1 的值为 2；而\_Main 函数中第二次调用\_Test 函数时，只给出了一个实参，所以 \$r2 的值为 1

#### ( 4 ) @ScriptName、@ScriptDir、@ScriptFullPath

@ScriptName 用于表示当前脚本的名称；

@ScriptDir 用于表示当前脚本所在目录；

@ScriptFullPath 用于表示脚本的完整路径，相当于 “@ScriptDir&@ScriptName”。

例如当前脚本的完整路径为：

```
D:\Au3Code\Test.au3
```

那么：

@ScriptName 的值为：

```
Test.au3
```

@ScriptDir 的值为：

```
D:\Au3Code
```

@ScriptFullPath 的值为：

```
D:\Au3Code\Test.au3
```

#### ( 5 ) @WorkingDir

@WorkingDir 用于表示当前脚本的工作目录。一般而言，无论是直接运行脚本文件，或是编译后直接运行.exe 程序，工作目录即是当前目录，与@ScriptDir 值相同。但如果脚本被其他程序调用时，例如运行 D:\1\Main.exe 时自动调用 D:\2\Test.exe，那么 Test.exe 的工作目录其实是 D:\1，即

主调用程序的工作目录。

### ( 6 ) @AutoItExe、@AutoItPID、@AutoItVersion、@AutoItX64

@AutoItExe，如果脚本被编译了，其值为编译后.exe 程序的完整路径，与@ScriptFullPath 值相同；如果脚本没被编译，其值为 Au3 安装目录下的 AutoIt3.exe 或 AutoIt3\_x64.exe 的路径（32 位方式运行为前者，64 位方式运行为后者）。

@AutoItPID 用于表示当前脚本运行时的进程标识符（PID），如果脚本被编译了，其值为编译后.exe 程序的 PID；如果脚本没被编译，其值为 Au3 安装目录下的 AutoIt3.exe 或 AutoIt3\_x64.exe 的 PID（32 位方式运行为前者，64 位方式运行为后者）。

@AutoItVersion 用于表示当前脚本所使用的 Au3 版本，例如 3.3.10.2。

@AutoItX64 用于表示当前脚本是否在以 64 位方式运行，是为 1，不是为 0。

### ( 7 ) @CR、@LF、@CRLF、@TAB

@CR 用于表示回车符（ASCII 代码值 13）；

@LF 用于表示换行符（ASCII 代码值 10）；

@CRLF 用于表示回车换行符，等效于 “@CR&@LF”；

@TAB 用于表示制表符（ASCII 代码值 9）。

## 2.6.2 目录宏

### 1、系统路径宏

#### ( 1 ) @HomeDrive、@HomePath

@HomeDrive 用于表示当前用户主目录所在盘符；

@HomePath 用于表示当前用户主目录的路径（无盘符）。

例如当前用户的主目录为：

C:\Users\Skyfree

那么：

@HomeDrive 的值为：

C:

@HomePath 的值为：

\Users\Skyfree

要获取用户主目录完整路径，可使用@HomeDrive&@HomePath。

#### ( 2 ) @WindowsDir、@SystemDir、@ProgramFilesDir、@CommonFilesDir

@WindowsDir 用于表示当前系统安装目录，如：

C:\Windows

@SystemDir 用于表示当前系统文件目录，如：

C:\Windows\system32

@ProgramFilesDir 用于表示程序文件目录，如：

C:\Program Files

@CommonFilesDir 用于表示公共程序文件目录，如：

C:\Program Files\Common Files

**( 3 ) @TempDir**

@TempDir 用于表示当前用户的程序临时目录（当程序需要释放临时文件时常用），如：

C:\Users\Skyfree\AppData\Local\Temp

**( 4 ) @ComSpec**

@ComSpec 用于表示系统命令解释程序的完整路径（当需要调用批处理命令来协助完成某些任务时常用），如：

C:\Windows\system32\cmd.exe

**2、当前用户宏****( 1 ) @AppDataDir、@LocalAppDataDir**

@AppDataDir 用于表示当前用户应用数据目录，如：

C:\Users\Skyfree\AppData\Roaming

@LocalAppDataDir 用于表示当前用户本地应用数据目录，如：

C:\Users\Skyfree\AppData\Local

**( 2 ) @DesktopDir**

@DesktopDir 用于表示当前用户桌面目录，如：

C:\Users\Skyfree\Desktop

**( 3 ) @StartMenuDir、@ProgramsDir、@StartupDir**

@StartMenuDir 用于表示当前用户开始菜单目录，如：

C:\Users\Skyfree\AppData\Roaming\Microsoft\Windows\Start Menu

@ProgramsDir 用于表示当前用户开始菜单程序目录，如：

C:\Users\Skyfree\AppData\Roaming\Microsoft\Windows\Start Menu\Programs

@StartupDir 用于表示当前用户开始菜单启动目录，如：

C:\Users\Skyfree\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup

**( 4 ) @MyDocumentsDir、@FavoritesDir、@UserProfileDir**

@MyDocumentsDir 用于表示当前用户文档目录，如：

C:\Users\Skyfree\Documents

@FavoritesDir 用于表示当前用户收藏目录，如：

C:\Users\Skyfree\Favorites

@UserProfileDir 用于表示当前用户配置文件（NTUSER.DAT）所在目录，如：

C:\Users\Skyfree

**3、所有用户宏****( 1 ) @AppDataCommonDir**

@AppDataCommonDir 用于表示公共应用数据目录，如：

C:\ProgramData

**( 2 ) @DesktopCommonDir**

@DesktopCommonDir 用于表示公共桌面目录，如：

C:\Users\Public\Desktop

### ( 3 ) @StartMenuCommonDir、@ProgramsCommonDir、@StartupCommonDir

@StartMenuCommonDir 用于表示公共开始菜单目录，如：

C:\ProgramData\Microsoft\Windows\Start Menu

@ProgramsCommonDir 用于表示公共开始菜单程序目录，如：

C:\ProgramData\Microsoft\Windows\Start Menu\Programs

@StartupCommonDir 用于表示公共启动菜单目录，如：

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup

### ( 4 ) @DocumentsCommonDir、@FavoritesCommonDir

@DocumentsCommonDir 用于表示公共文档目录，如：

C:\Users\Public\Documents

@FavoritesCommonDir 用于表示公共收藏目录，如：

C:\Users\Public\Favorites

## 2.6.3 系统信息宏

### ( 1 ) @OSArch、@OSLang、@OSType、@OSVersion、@OSBuild、@OSServicePack

@OSArch 用于表示系统位宽，32 位系统时值为 “x86”，64 位系统时值为 “x64”，安腾 64 位系统为 “IA64”；

@OSLang 用于表示系统语言代码，因代码很多请参见帮助文档，常用的值有：

- 简体中文 0804
- 繁体中文 0404 (台湾) 0C04 (香港) 1404 (澳门)
- 英文 0409 (美国) 0809 (英国)

@OSType 用于表示系统类型，一般常见系统系均为 “WIN32\_NT”；

@OSVersion 用于表示系统版本，值包括：

- 桌面系统：WIN\_8.1、WIN\_8、WIN\_7、WIN\_VISTA、WIN\_XP、WIN\_XPe
- 服务器系统：WIN\_2012R2、WIN\_2012、WIN\_2008R2、WIN\_2008、WIN\_2003

@OSBuild 用于表示系统内部版本号，如 “7601” ( Windows 7 SP1 )

@OSServicePack 用于表示服务包号，如 “Service Pack 1”

### ( 2 ) @CPUArch ( 对比：@OSArch )

@CPUArch 用于表示 CPU 的位宽，32 位时值为 “x86”，64 位时值为 “x64”。

与 @OSArch 所表示的系统位宽不同，@CPUArch 是 CPU 的硬件位宽，例如在 64 位的 CPU 硬件系统上安装 32 位系统时，@CPUArch 值为 “x64”，而 @OSArch 值为 “x86”。但绝大多数时候我们所说的 32 位和 64 位都是在指系统位宽，因为程序运行是以系统为平台的，即便 CPU 支持 64 位，如果没有 64 位的系统，仍旧无法运行 64 位的应用程序。

### ( 3 ) @KBLLayout、@MUILang ( 对比：@OSLang )

@KBLLayout 用于表示当前键盘布局代码，代码表与 @OSLang 代码表是同一个，区别在于键盘布局代码比系统语言代码前部多了 4 个 0，例如简体中文系统的键盘布局代码为 “00000804”。

@MUILang 用于表示多语言系统的 UI 语言代码，代码表与 @OSLang 代码表是同一个，区别在于 @MUILang 是 UI 的语言代码，例如在英文语言系统上安装了简体中文语言包后，@OSLang 的值

为 0409，但 @MUILang 的值为 0804。（注意，此例笔者未实际验证）

#### (4) @ComputerName、@UserName

@ComputerName 用于表示计算机名，如：SKYFREE-PC；

@UserName 用于表示当前用户名，如：Skyfree。

#### (5) @IPAddress1、@IPAddress2、@IPAddress3、@IPAddress4

@IPAddress1 用于表示第一个网卡的 IP 地址，如：192.168.1.5（个别可能为 127.0.0.1）

@IPAddress2 用于表示第二个网卡的 IP 地址，若不存在则为 0.0.0.0。

@IPAddress3 用于表示第三个网卡的 IP 地址，若不存在则为 0.0.0.0。

@IPAddress4 用于表示第四个网卡的 IP 地址，若不存在则为 0.0.0.0。

#### (6) @DesktopHeight、@DesktopWidth、@DesktopDepth、@DesktopRefresh

@DesktopWidth 用于表示水平分辨率，如：1920

@DesktopHeight 用于表示垂直分辨率，如：1080

@DesktopDepth 用于表示颜色深度，如：32

@DesktopRefresh 用于表示刷新率，如：60

### 2.6.4 时间与日期宏

#### (1) @YEAR、@MON、@MDAY、@WDAY、@YDAY

@YEAR 用于表示当前年份，4 位数，如：2014（2014 年）

@MON 用于表示当前月份，01~12，如：04（4 月）

@MDAY 用于表示今天是本月的第几天，00~31，如：16（第 16 天，亦等同于 16 日）

@WDAY 用于表示今天是本周的第几天，注意，1~7 对应周日~周六，如：4（周三）

@YDAY 用于表示今天是本年的第几天，001~365（闰年最大值为 366）

例如：

```
_Main()
Exit

Func _Main()
Local $Date = @YEAR & "年" & @MON & "月" & @MDAY & "日"

Local $a_WDay[] = ["", "日", "一", "二", "三", "四", "五", "六"]
Local $WDay = $a_WDay[@WDAY]

Local $ts = $Date & "星期" & $WDay
MsgBox(0, "", $ts)
EndFunc  ;==>_Main
```

输出结果形如：“2014 年 04 月 16 日 星期四”。

#### (2) @HOUR、@MIN、@SEC、@MSEC

@HOUR 用于表示当前时，00~23，如：18（下午 6 时）

@MIN 用于表示当前分，00~59，如：56（56 分）

@SEC 用于表示当前秒，00~59，如：25（25 秒）

@MSEC 用于表示当前毫秒 ( 1 秒 = 1000 毫秒 ), 000~999 , 如 015 :( 15 毫秒 )

例如：

```
_Main()
Exit

Func _Main()
    Local $Time = @HOUR & ":" & @MIN & ":" & @SEC
    MsgBox(0, "", $Time)
EndFunc
```

输出结果形如：“19:08:23”。

## 2.7 Level2 总结

作为对 Level1 的进阶，Level2 着重讲述了几个编程基础中的难点。Level2 的学习结束，代表着对 AutoIt v3 基础学习的结束。不要小看基础知识，任何高级编程都必须以基础知识为根本，有些程序功能很复杂，除了其涉及的知识面更多之外，更多的是其将基础知识反复、整合使用。如果学习者对 Level1 或 Level2 中的某些点仍存在疑惑，建议回头重新阅读一番，将其学扎实。盖楼没有好的地基，盖的越高越危险，编程同理。

本章中，我们首先学习了函数。函数并不是一个难点，但却是编程的重点。说其不难，是因为函数最大的作用是将代码拆分成多个子功能代码段，使整体更利于阅读和维护，便于重复使用和移植，所以说只要理解了参数的传递方式、理解了变量的作用域，剩下的事情就只有拆分功能罢了。而说其重要，是因为函数是实现结构化编程的重要组成部分，拆分功能不是目的，目的是令代码整体有着明确的结构体系，使代码整体调理有序，从而有助于减少代码出错率，利于多人协同开发和代码维护。

而后，我们学习了数组。数组是一个难点，同样是一个重点。数组是我们所接触的第一个构造类型数据，与我们在 Level1 所学习的基本数据不同，构造化数据是由多个数据组成的数据集合，而不是单一的数据。借助索引，我们来获取数组中的某个数据，索引是由 0 开始的自然数，这又与循环结构形成了关联，可以说用到数组的地方几乎一定会用到循环，而循环又是 Level1 的难点，两大难点聚在一起，给学习添加了不小的难度。但无论如何，数组是重要且常用的数据类型，在实际编程中使用频率非常高，即便有难度也一定要克服。

然后，我们进一步学习了字符串。字符串类型数据我们并不陌生，在 Level1 中就已经多次接触了。将字符串相关基础知识单列为一节，最大的原因是大多数数据都是以字符串的形式存储的，对数据的处理往往就是对字符串的处理。除了对字符串加深认识外，我们又学习了字符串相关函数，以用于对字符串进行操作、处理、判断。（对于字符串还有一个重点是正则表达式，因其有一定难度，所以推后讲解，不将过多难点过于集中在一章内）

之后，我们学习了代码书写规范。这套代码书写规范，是笔者在 7 年的 AutoIt v3 编程实践中逐步总结出来，虽不能说这是一套最好或完善的解决方案，但至少是经过长期实践验证的方案。遵守规范书写代码，最初会感觉有些麻烦，但只要习惯了就会变得很自然。代码规范是用良好的编程习惯来

抵消错误的发生机率，大家是愿意用一时的麻烦换来今后的安逸，还是想用一时的安逸换来今后的麻烦？

最后，我们学习了宏。宏其实并不复杂，它就是一种特殊的由系统进行预定义的变量，可以使用但不可为其赋值。对于宏，更多的是需要记忆其功能含义，不过对于有点儿英文基础的同学来说没有太大难度。其实也不必完全死记硬背，只需要知道有哪些宏，用到时再查询帮助文档即可。

最后的最后，请务必重视本章，并以同样的态度重视 Level1。Level1 和 Level2 是基础中的基础，对基础及时的扎实掌握，会令今后的学习事半功倍。

## Level 3

## 3.1 GUI 是什么

### 3.1.1 GUI 的概念

GUI ( Graphical User Interface , 图形用户接口 ) , 一般是指以图形方式与用户进行交互的方法 , 是目前所普遍采用的一种人机交互方式。

UI 有很多种 , 广义的讲 , 能够使人对计算机进行操作的方式都可以被称作 UI 。如 DOS 和 Linux Shell 也是一种 UI , 只不过这种 UI 是命令行方式。而 GUI 更重视 “G” , 即图形方式 , 图形方式有利于用户的理解和操作 , 使机交互变得更为人性化。

### 3.1.2 GUI 的设计

GUI 的设计要遵循一定的原则 , 请时刻谨记 GUI 的本质是 “ 用户接口 ” , 如何使用户能够顺畅的进行操作 , 是 GUI 设计的核心准则。在 GUI 设计中 , 请注意以下原则 :

#### ( 1 ) 用户关注的是任务 , 而不是技术

以怎样的算法去完成一个任务 , 这往往不是用户所关心的 , 用户所关心的往往是 “ 怎样按一个按钮完成所有任务 ” , 虽然这是几乎不可能的 , 但却是用户所向往的 , 所以 GUI 的设计往往以 “ 怎样以最少的操作完成最多的任务 ” 为终极目标。

#### ( 2 ) 从用户的视角看问题 , 使用用户能看懂的词汇

设计者认为理所当然的事情 , 往往不是用户认为理所当然的 , 同样的 , 设计者认为容易理解的词汇 , 常常不是用户容易理解的。 GUI 是给用户操作的 , 那么设计者就必须站在用户的立场上考虑怎样设计和描述。记住 , 最好的 GUI 是 “ 一看即会 ” 的。

#### ( 3 ) 使任务简单化 , 不要让用户解决额外的问题

用户只做最少、最简单的操作来实现任务。例如解压一个压缩包 , 用户只需要设定 “ 哪个压缩包 ” 和 “ 解压到哪里 ” 就可以了 , 除此之外不要再让用户去做其他事情 , 否则这个 GUI 的设计必定是失败的。

#### ( 4 ) 显示信息 , 而不仅仅是数据

程序执行后 , 将数据回馈给用户 , 从而实现交互。但不要简单的认为 “ 数据 ” 就是 “ 信息 ” , 二者有本质的差别。比如对于日期而言 , 数据是 “ 201406031135 ” , 信息是 “ 2014-6-3 , 11:35 ” , 前者是给程序看的 , 后者是给用户看的 , 如果将前者展示给用户 , 那就不要指望用户有什么好心情了。

#### ( 5 ) 保持用户操作习惯的一致性

尽可能保证用户对 GUI 习惯的一致性 , 这对程序的亲和力有着极大的帮助。例如开发一个行业软件 , 已有同类软件数种 , 且用户已形成特定的操作习惯 , 新开发的软件如果符合用户已有的习惯 , 则有助于用户的快速上手 , 并加速软件的普及效率。不要总让用户为你而改变操作习惯 , 除非你是 Steve Jobs 。

### (6) 不要向用户暴露程序执行细节

执行细节是程序的事情，不是用户需要操心的内容。如果需要用户等待，可给出进度条或其他方式提醒，但不要向用户过分呈现程序执行细节，这对于开发者的利益而言也是一种保护。

### (7) 漂亮的 GUI，不一定是好的 GUI

GUI 的第一要务是令用户与计算机形成良好的交互，所以如果达不到这个目的，GUI 做的再漂亮也无济于事。但反过来讲，可以称为“好”的 GUI，几乎一定漂亮，无论是交互还是外观。

## 3.2 GUI 入门

万事开头难，初学 GUI 也是如此。GUI 设计所涉及的知识繁多，如果一味的堆叠概念，在早期过于求细求多，则会导致初学者包袱沉重，学习困难。所以，本章先和大家一起学习 GUI 设计的基础方法，先学起来、用起来，再逐步学精、学深。

### 3.2.1 创建窗体

#### 1. 什么是窗体

窗体（Form）是应用程序与用户进行交互的信息窗口，如：



(图 3-1)

#### 2. 创建窗体

创建窗体需要使用函数：GUICreate。

函数	说明(简单语法)
GUICreate 创建窗体	<p>语法： GUICreate("标题", [宽度, [高度, [左侧=-1, [顶部=-1]]]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>标题：窗体的标题</li> <li>宽度：【可选参数】窗体的宽度(单位：像素)</li> <li>高度：【可选参数】窗体的高度(单位：像素)</li> <li>左侧：【可选参数】窗体左侧与显示区域左侧的距离(单位：像素) 默认值为-1，默认值时窗体在显示区域内左右居中。</li> <li>顶部：【可选参数】窗体顶部与显示区域顶部的距离(单位：像素) 默认值为-1，默认值时窗体在显示区域内上下居中。</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回窗体句柄( Handle )</li> <li>失败：返回 0，并将@error 设置为 1</li> </ul>

举例：创建一个标题为“Form”，宽 300 像素、高 200 像素，且上下、左右居中的窗体。

```
GUICreate("Form", 300, 200, -1,-1)
```

实际上，测试上述代码，将看不到任何效果。原因是 Au3 所创建的窗体默认为隐藏状态( hide )，如果要将窗体显示出来，需要将窗体的设置为显示状态( show )。

### 3、显示窗体

改变窗体的状态需要使用函数：GUISetState。

函数	说明(简单语法)
GUISetState 设置窗体状态	<p>语法： GUISetState([标志, [窗体句柄]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>标志：窗体状态标志，如：</li> <li>@SW_SHOW，显示窗体(默认值)</li> <li>@SW_HIDE，隐藏窗体</li> <li>@SW_DISABLE，禁用窗体</li> <li>@SW_ENABLE，启用窗体</li> </ul> <p>窗体句柄：被设置窗体的句柄，默认值为此前最后创建窗体的句柄。</p> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回 1</li> <li>失败：返回 0</li> </ul>

举例：

```
GUICreate("Form", 300, 200, -1,-1)
GUISetState(@SW_SHOW)
```

或

```
Global $gu_Form = GUICreate("Form", 300, 200, -1, -1)
GUISetState(@SW_SHOW, $gu_Form)
```

写好了上面的代码，运行，终于能看到一个一闪而过的窗体了。为什么会一闪而过？其实只要想一下 Au3 代码运行的机制就明白了，Au3 代码是自顶向下依次执行的，当所有代码执行完毕后就会退出脚本，并释放相关的系统资源。上述代码只有 2 行，那么脚本执行时，遵循：脚本开始->创建窗

体->显示窗体->脚本结束，所以在窗体创建并显示后，程序就退出了，自然就把窗体给删除了。

那么，怎样持续显示一个窗体？

#### 4、维持窗体

要让脚本持续运行而不结束，方法是让这个脚本陷入一个持续执行中，而不能退出。这个状态，可以通过死循环来实现。例如：

```
While 1
```

```
WEnd
```

1 转化成布尔值就是 True，While 循环的条件如果总是为 True 则循环永不退出。但空的死循环会造成高额的系统资源消耗，我们要在循环内做点儿什么，以减少资源消耗，那么我们就暂时让每次循环时程序休眠 1 毫秒吧。

```
While 1
```

```
    Sleep(1) ;脚本休眠 1 毫秒
```

```
WEnd
```

Sleep 是个很简单的函数，可以令程序进入休眠状态，休眠的单位是毫秒（1 秒=1000 毫秒）。Sleep 函数先不多介绍，后续章节会对其详解，目前我们只是借助其降低持续循环对系统资源的消耗。

由此，我们的窗体创建代码改为：

```
Global $gu_Form = GUICreate("Form", 300, 200, -1, -1)
```

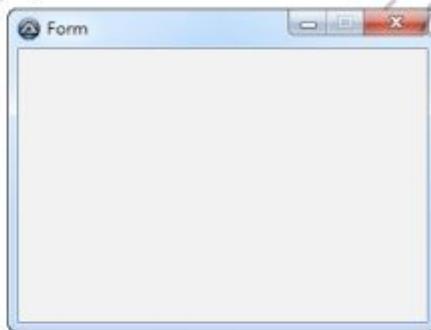
```
GUISetState(@SW_SHOW, $gu_Form)
```

```
While 1
```

```
    Sleep(1)
```

```
WEnd
```

运行，终于可以看见窗体了！



( 图 3-2 )

值得注意的是，这个窗体并不会响应你的任何操作，甚至无法通过右上角的退出按钮退出，这是因为我们并没有书写任何响应代码。（需要退出的话，可单击任务栏脚本的图标，然后退出脚本）

虽然我们奋斗了 4 个小节才写出了一个没有实际功能的窗体，但是请不要气馁，这是一小步，却是学习 GUI 的一大步。请不要小瞧本节，本节内容将有助于详细理解 GUI 的呈现原理，对后续章节（甚至高等级章节）的学习有着至关重要的帮助。

### 3.2.2 创建控件

#### 1、什么是控件

控件 ( Control , 常被简写为 Ctrl ) 是组成 GUI 的元素 , 用于展示信息、响应用户操作 , 是与用户交互的基本单元。控件一般在创建于窗体内 , 换句话说 , 窗体是控件的容器。

控件有很多种类类型 , 常见的控件有标签、文本框、按钮、复选框、单选按钮、菜单、组合框、树状结构、进度条等等 , 我们并不需要在一个 GUI 中用上所有的控件类型 , 只需要根据需求选取最合适的控件即可。本章将介绍几种基本的控件 , 供大家学习体会。



( 图 3-3 )

#### 2、怎样创建

以最简单的三种控件为例 :

- ( 1 ) 标签 ( Label ) 控件 , 一般用于展示信息 ( 提示信息、说明信息、输出结果等 );
- ( 2 ) 文本框 ( Input ) 控件 , 一般用于要求用户输入信息 , 并根据用户输入的信息执行程序 ;
- ( 3 ) 按钮 ( Button ) 控件 , 一般用于响应用户的单击事件 , 常用于开始执行某些任务。

函数	说明 ( 简单语法 )
GUICtrlCreateLabel 创建标签控件	<p>语法 : GUICtrlCreateLabel("文本", 左侧, 顶部, [宽度, [高度]])</p> <p>参数 :</p> <ul style="list-style-type: none"> <li>文本 : 控件的文本</li> <li>左侧 : 控件左侧距离窗体左侧的距离 ( 单位 : 像素 )</li> <li>顶部 : 控件顶部距离窗体顶部的距离 ( 单位 : 像素 )</li> <li>宽度 : 【可选参数】控件的宽 ( 单位 : 像素 ) ( 不指定时根据文本内容自动调整 )</li> <li>高度 : 【可选参数】控件的高 ( 单位 : 像素 )</li> </ul>

	<p>( 不指定时根据文本内容自动调整 )</p> <p>返回值 :</p> <p>成功 : 返回控件 ID 失败 : 返回 0</p>
GUICtrlCreateInput 创建文本框	<p>语法 :</p> <p>GUICtrlCreateInput("文本", 左侧, 顶部, [宽度, [高度]])</p> <p>参数 :</p> <p>文本 : 控件的文本 左侧 : 控件左侧距离窗体左侧的距离 ( 单位 : 像素 ) 顶部 : 控件顶部距离窗体顶部的距离 ( 单位 : 像素 ) 宽度 : 【可选参数】控件的宽 ( 单位 : 像素 ) ( 不指定时根据文本内容自动调整 ) 高度 : 【可选参数】控件的高 ( 单位 : 像素 ) ( 不指定时根据文本内容自动调整 )</p> <p>返回值 :</p> <p>成功 : 返回控件 ID 失败 : 返回 0</p>
GUICtrlCreateButton 创建按钮	<p>语法 :</p> <p>GUICtrlCreateButton("文本", 左侧, 顶部, [宽度, [高度]])</p> <p>参数 :</p> <p>文本 : 控件的文本 左侧 : 控件左侧距离窗体左侧的距离 ( 单位 : 像素 ) 顶部 : 控件顶部距离窗体顶部的距离 ( 单位 : 像素 ) 宽度 : 【可选参数】控件的宽 ( 单位 : 像素 ) ( 不指定时根据文本内容自动调整 ) 高度 : 【可选参数】控件的高 ( 单位 : 像素 ) ( 不指定时根据文本内容自动调整 )</p> <p>返回值 :</p> <p>成功 : 返回控件 ID 失败 : 返回 0</p>

举例 : 设计一个输入圆的半径 , 计算圆的面积的 GUI

```
; 创建一个宽 215 像素、高 125 像素 , 居中显示的窗体
Global $gu_Form1 = GUICreate("计算圆的面积", 215, 125, -1, -1)

; 在距离窗体左侧 16 像素、顶部 19 像素的位置 , 创建一个宽 28 像素、高 17 像素的标签控件
Global $gu_Label1 = GUICtrlCreateLabel("半径", 16, 19, 28, 17)

; 在距离窗体左侧 16 像素、顶部 51 像素的位置 , 创建一个宽 28 像素、高 17 像素的标签控件
Global $gu_Label2 = GUICtrlCreateLabel("面积", 16, 51, 28, 17)

; 在距离窗体左侧 48 像素、顶部 16 像素的位置 , 创建一个宽 145 像素、高 21 像素的文本框控件
Global $gu_Input1 = GUICtrlCreateInput("", 48, 16, 145, 21)

; 在距离窗体左侧 48 像素、顶部 48 像素的位置 , 创建一个宽 145 像素、高 21 像素的文本框控件
Global $gu_Input2 = GUICtrlCreateInput("", 48, 48, 145, 21)
```

```
;在距离窗体左侧 104 像素、顶部 88 像素的位置，创建一个宽 41 像素、高 25 像素的按钮控件
Global $gu_Button1 = GUICtrlCreateButton("计算", 104, 88, 41, 25)

;在距离窗体左侧 152 像素、顶部 88 像素的位置，创建一个宽 41 像素、高 25 像素的按钮控件
Global $gu_Button2 = GUICtrlCreateButton("清空", 152, 88, 41, 25)

;显示窗体
GUICreate("计算圆的面积", 280, 150, 100, 100)
GUICtrlCreateInput("半径", 100, 100, 100, 20)
GUICtrlCreateInput("面积", 100, 120, 100, 20)
GUICtrlCreateButton("计算", 150, 130, 50, 20)
GUICtrlCreateButton("清空", 150, 150, 50, 20)
GUICtrlSetState(-1, $SW_SHOW)

;维持窗体
While 1
    Sleep(1)
WEnd
```

图例：



( 图 3-4 )

通过一番努力，终于成功的创建了窗体和控件，但现在的窗体和控件没有任何实际功能。要让它们具有实际功能，则必须为它们添加响应代码。

### 3.2.3 创建代码

#### 1、消息循环模式

在添加功能代码之前，我们必须先思考一个问题：怎么让程序知道我们操作了控件？例如单击了上例中的“计算”按钮？

其实在操作 GUI 时，GUI 会发出消息（事件），消息一般分为两类：控件消息和系统消息。

（1）控件消息是当某个控件被点击或发生改变时发出的，控件消息代码为正数，并与控件 ID 相关联；

（2）系统消息是当窗体被操作时发出的，消息代码为负数，常量值包含于 GUIConstantsEx.au3 中，其中最常见的是窗口关闭消息：\$GUI\_EVENT\_CLOSE，值为 -3。

这样，如果我们能够获取到控件消息，则可以知道是操作了哪个控件；如果能够获取系统消息，则可以知道窗体被做了什么操作。

要获取这些消息，可使用函数 GUIGetMsg。

函数	说明（简单语法）
----	----------

<b>GUIGetMsg</b> 创建标签控件	语法： <code>GUIGetMsg()</code> 返回值： 系统消息：返回系统消息值 控件消息：返回控件 ID 无消息：返回 0
----------------------------	---

但现在还有一个问题，我们不可能知道用户在什么时间单击控件，所以我们必须一直使用 GUIGetMsg 函数去反复“监听”当前的消息事件。说到“一直”、“反复”，大家应该联想到了之前的那个用于维持窗体的循环了。

我们可以使用循环一直反复获取当前的消息，如果消息不为 0，则对应起控件 ID 或系统消息的代码，则可知道用户进行了什么操作。为此，我们需要形如这样的代码：

```
Local $nMsg
While 1
  $nMsg = GUIGetMsg()
  Switch $nMsg
    Case <系统消息 ID>
      <执行系统消息>
    Case <控件 ID 1>
      <执行对应事件 1>
    Case <控件 ID 2>
      <执行对应事件 2>
  EndSwitch
WEnd
```

此段代码中，定义变量 \$nMsg 用于存储当前系统消息，而后使用 Switch 多分支结构判定 \$nMsg 对应了哪个系统消息或控件 ID，由此，我们就可以将用户对窗体、控件的操作与实际功能代码对应起来了！而因为这种模式是由反复循环方式获取系统消息的，所以又被称为“消息循环模式”。

反应快的同学可能立刻会有一个问题，在代码中反复使用循环获取消息事件，会不会使系统资源利用率陡增？答案是“不会”。GUIGetMsg 是一个为消息循环模式专门设计的函数，当无消息时会自动闲置 CPU，所以没必要担心 GUIGetMsg 函数会带来系统资源消耗。

## 2、获取与修改控件数据

大问题解决完了，还剩下两个小问题，怎样获取和修改控件的数据？例如在 3.2.2 节中创建的“根据半径计算圆的面积”的 GUI，计算前需要用户输入圆的半径值，计算后需要输出圆的面积，怎样读取用户写入文本框的信息？又怎样将计算后的结果输出到另一个文本框？这些问题，需要使用函数 GUICtrlRead 和 GUICtrlSetData 来解决。

函数	说明（简单语法）
GUICtrlRead 获取控件信息	语法： <code>GUICtrlRead(控件 ID)</code>

	<p>参数：</p> <p>控件 ID：控件的 ID</p> <p>返回值：</p> <p>成功：返回控件信息 失败：返回 0</p>
GUICtrlSetData 设置控件信息	<p>语法：</p> <p>GUICtrlSetData(控件 ID, 数据)</p> <p>参数：</p> <p>控件 ID：控件的 ID</p> <p>数据：将要设置的数据（因控件类型不同而不同）</p> <p>返回值：</p> <p>成功：返回 1 失败：返回 0</p>

OK，现在我们终于可以为 3.2.2 节中的代码添加实际功能了！

```
#include <GUIConstantsEx.au3>

AutoItSetOption("MustDeclareVars", 1)

Global Const $gc_Pi = 3.14

Global $gu_Form1 = GUICreate("计算圆的面积", 215, 125, -1, -1)
Global $gu_Label1 = GUICtrlCreateLabel("半径", 16, 19, 28, 17)
Global $gu_Label2 = GUICtrlCreateLabel("面积", 16, 51, 28, 17)
Global $gu_Input1 = GUICtrlCreateInput("", 48, 16, 145, 21)
Global $gu_Input2 = GUICtrlCreateInput("", 48, 48, 145, 21)
Global $gu_Button1 = GUICtrlCreateButton("计算", 104, 88, 41, 25)
Global $gu_Button2 = GUICtrlCreateButton("清空", 152, 88, 41, 25)

_Main()
Exit

Func _Main()
    GUISetState(@SW_SHOW, $gu_Form1)

    Local $r, $s

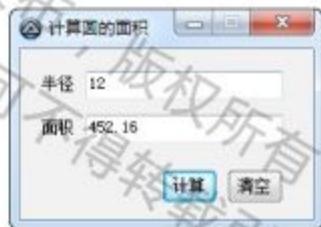
    Local $nMsg
    While 1
        $nMsg = GUIGetMsg()
        Switch $nMsg
            Case $GUI_EVENT_CLOSE ;关闭窗口
                Exit
            Case $gu_Button1 ;单击 "计算" 按钮
                ;计算逻辑
        EndSwitch
    EndWhile
EndFunc
```

```

$r = GUICtrlRead($gu_Input1) ;获取半径值
If $r = "" Then
    MsgBox(0, "错误", "未设置半径") ;必须输入半径才能计算
Else
    $r = Number($r)
    If $r < 0 Then
        MsgBox(0, "错误", "半径不可为负数") ;半径不可为负
    Else
        $s = $gc_Pi * $r ^ 2
        GUICtrlSetData($gu_Input2, $s)
    Endif
Endif
Case $gu_Button2 ;单击“清空”按钮
    GUICtrlSetData($gu_Input1, "") ;将文本框数据设置为空字符串，即为“清空”
    GUICtrlSetData($gu_Input2, "")
EndSwitch
WEnd
EndFunc

```

图例：



(图 3-5)

值得注意的是，上述代码中，对于用户输入的数据加入了数个判断，例如判定输入是否为空，判断输入的数值是否为非负数等。这是因为用户的输入是不可预知的，用户并不像设计者那样懂得这里应该填写什么样的数据。但是上例中的判断也并不完全，例如未判断是否为数值而不是字符串、未判断半径最大上限是多少等等，这些判断需要大家学习更多编程知识才能书写。在这里要和大家明确的一点是，对于用户写入的数据必须严格判断，不要以设计者的思维认为数据应该是怎样的，用户的输入永远是千奇百怪的。

### 3.3 KODA 编辑器

在 3.2 节中，我们手动创建了窗体和控件，想必大家深有体会的是：最麻烦的并不是掌握创建窗体或控件的函数，而是计算使用多大的窗体、要将哪个控件放在哪个位置、控件的长宽各是多少。简单的 GUI 也就罢了，复杂的 GUI 如果都靠如此脑补，非得累死人不可。所以在本节中，为大家介绍一个 GUI 设计辅助工具：KODA。

开篇提醒：KODA 只是 GUI 设计辅助工具，不要过分依赖 KODA，以代码控制 GUI 才是王道。

### 3.3.1 认识 KODA

#### 1、打开 KODA

方法 1：桌面->AutoIt GUI 编辑器

方法 2：开始菜单->所有程序->AutoIt v3 (ITianKong.Com)->编辑图形界面(Koda)

方法 3：任务栏->ACC 工具箱->运行 AutoIt GUI 编辑工具(Koda)

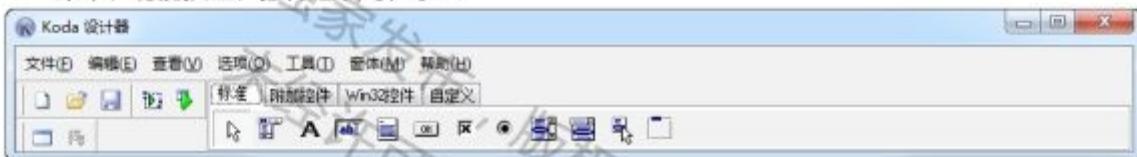
方法 4：AutoIt3 安装目录\SciTE\Koda\FD.exe

#### 2、看懂 KODA

Koda 分为 5 部分：

##### (1) 功能区

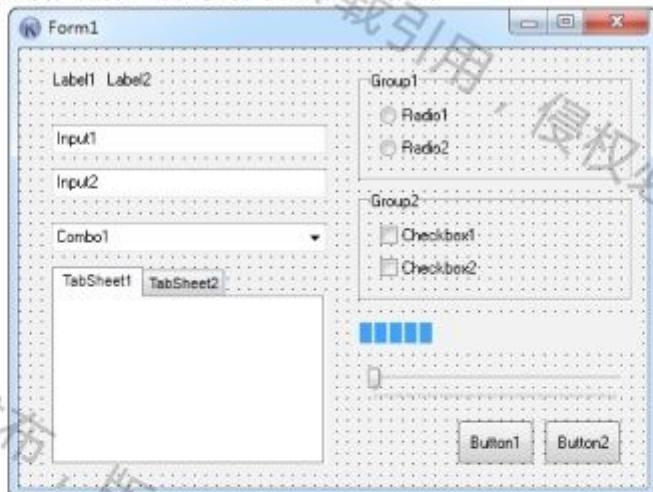
菜单、功能按钮、控件选项均位于此



(图 3-6)

##### (2) 设计区

创建一个窗体，然后在窗体上添加控件，完成 GUI 设计



(图 3-7)

##### (3) 对象列表区 (对象树形图)

窗体、控件会在此以树状结构展示



( 图 3-8 )

#### ( 4 ) 对象属性区 ( 对象检查器 )

窗体和控件的属性、样式、扩展样式会展示于此



( 图 3-9 )

#### ( 5 ) 窗体列表区

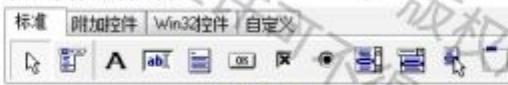
GUI 的所有窗体会在此处以列表形式展示



( 图 3-10 )

### 3.3.2 使用 KODA 创建控件

在功能区的下图位置中，包含多种控件：



(图 3-11)

控件的创建方法很简单，使用鼠标左键单击某种控件，将鼠标移动至设计区，在窗体上拖动出一块区域，松开鼠标左键，即可创建一个控件。

常用控件列表：

控件	控件名称	说明	相关函数
按钮控件	一般用于响应单击	GUICtrlCreateButton	
复选框控件	可多选的控件	GUICtrlCreateCheckbox	
组合框控件	下拉菜单多个选项	GUICtrlCreateCombo	
日期控件	显示日期	GUICtrlCreateDate	
编辑框控件	可输入大量文字	GUICtrlCreateEdit	
分组控件	可使其他控件分组	GUICtrlCreateGroup	
图标控件	显示.ico 图标	GUICtrlCreateIcon	
文本框控件	主要用于输入文字	GUICtrlCreateInput	
静态标签控件	主要用于显示文字	GUICtrlCreateLabel	
列表控件	生成多行单列的简单列表	GUICtrlCreateList	
列表查看器控件	生成多行多列的复杂列表	GUICtrlCreateListView GUICtrlCreateListViewItem	
菜单控件	生成类似菜单栏的菜单项目	GUICtrlCreateMenu GUICtrlCreateMenuItem	
月历控件	显示月历，并可点选设定日期	GUICtrlCreateMonthCal	
图像控件	显示图片，如.jpg、.bmp	GUICtrlCreatePic	
进度条控件	显示进度情况	GUICtrlCreateProgress	
单选框控件	仅可单选的控件	GUICtrlCreateRadio	
滑动条控件	类似播放器的拖动条	GUICtrlCreateSlider	
标签页控件	生成多个标签页	GUICtrlCreateTab GUICtrlCreateTabItem	
树形查看器控件	生成树状结构的信息图	GUICtrlCreateTreeView GUICtrlCreateTreeViewItem	

常用控件种类繁多，但不代表我们需要全部使用，更不需要死记硬背。遵循“多应用、多理解”的学习理念，我们会逐步学习每种控件的使用条件和使用方式。

### 3.3.3 使用 KODA 调整控件属性

每种控件都有其对应的属性，当我们创建一个控件并选中它时，KODA 左下角对象列表区内就会显示此控件的各个属性。属性名称为英文，每个控件都会对应数个属性，每个控件的属性种类又有一定差别。这令很多初学者犯难，但不要被眼花缭乱的表象所迷惑，重复开篇的一句话，KODA 只是 GUI 辅助设计工具，学会利用辅助，用以驾驭本质，总结并归纳，从而将复杂的问题条理化，进而掌握并应用。

使用 KODA 分别创建一个窗体（Form）、标签（Label）、文本框（Input）和按钮（Button），而后我们来看看它们的属性。



(图 3-12)



(图 3-13)



(图 3-14)



(图 3-15)

第一，每个控件都有“属性”、“样式”和“扩展样式”三类值，在 KODA 中我们应将注意力集中在“属性”上，“样式”和“扩展样式”建议通过代码来调整而不是在 KODA 中设定。这样做的好处有二，其一，我们可以更集中精力于调整控件性质，不分散精力于其他方面；其二，可以良好的遵循“以代码控制 GUI”的原则，便于后期维护与调整。

第二，虽然每个控件的属性类别和数量并不相同，但在 KODA 调整中常用的属性并不多，更多的 GUI 属性在代码中进行调整要更为灵活。常用属性包括：

属性	作用
Left	控件左侧距离窗体的距离，或窗体左侧距离显示区域左侧的距离
Top	控件顶部距离窗体的距离，或窗体顶部距离显示区域顶部的距离
Width	控件或窗体的宽
ClientWidth	窗体除去边框后的宽（仅用于窗体）
Height	控件或窗体的高
ClientHeight	窗体除去边框后的高（仅用于窗体）
Name	控件的名称（注意不是文本）
Text 或 Caption	控件的文本
AutoSize	自适应大小（Label 等的特殊属性，一般设置为 False，避免字体产生的问题）
Font	控件或窗体中文本所使用的字体（一般保持默认）

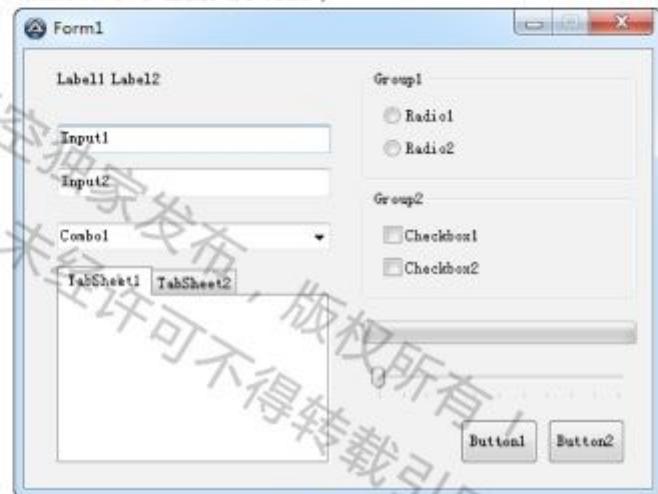
记住这些常用属性，就已经征服了 85% 的控件属性了！所以，一定不要看到这一大堆英文属性就开始发憷，常用的并不多，不常用的到用时再学，先会用，后精通。

修改属性值则很简单，鼠标单击属性右侧的值，修改为指定值，而后鼠标单击一下其他任意位置就可以了。

### 3.3.4 将设计转化为代码

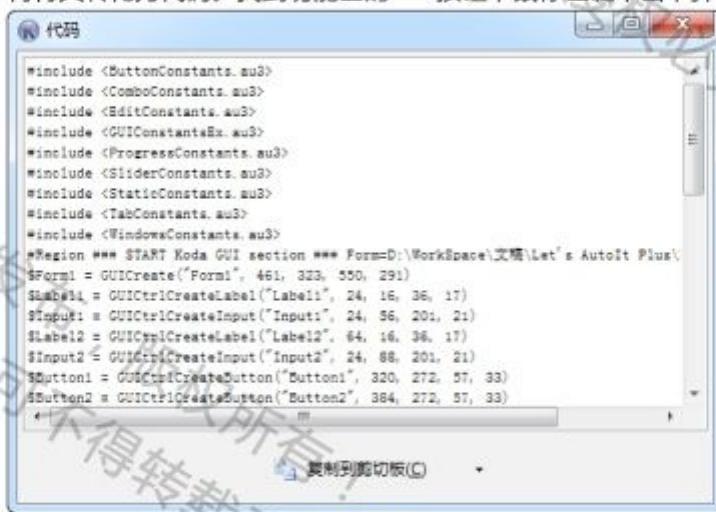
KODA 只是 GUI 设计辅助工具，最终我们还是要用代码来控制 GUI，那么怎样将设计好的 GUI 转化成代码呢？

先别急，先来看看我们的设计效果。找到功能区的 按钮，鼠标左键单击，即可看到 GUI 的实际运行效果（注意，只是效果，不包含实际功能）：



(图 3-16)

效果满意后，再将其转化为代码。找到功能区的 按钮，鼠标左键单击，弹出窗口如下图：



(图 3-17)

复制其中代码，粘贴到脚本文件中即可！

使用 KODA，好处在于可以更为直观方便的设计 GUI，就像画画一样，轻松点点鼠标即可完成。另外，由 KODA 产生的代码，除了自动生成了 GUI 窗体、控件的创建代码以外，也将可能用到的 \*Constants.au3 做了包含 (#include)，并自动生成了消息循环模式的代码。

上述都是 KODA 的便捷之处，但 KODA 生成的代码不一定完全符合我们的需求。例如并不符合我们之前所提到的代码书写规范，没有先定义变量后使用，也没有充分的将代码函数化。这些都需要我们动手改动一下。

综上，KODA 虽然很方便，但至始至终只是 GUI 设计辅助工具，使用代码控制 GUI 才是核心目的。借助 KODA，但不要过分依赖 KODA，发挥 KODA 的便捷之处，融合我们的编程思想，才能将编程学习走的更远。

3.4 窗体

在 3.2.1 节中，我们已经学习了创建窗体的基本方法，现在我们来学习更多相关知识。

## 1. GUICreate

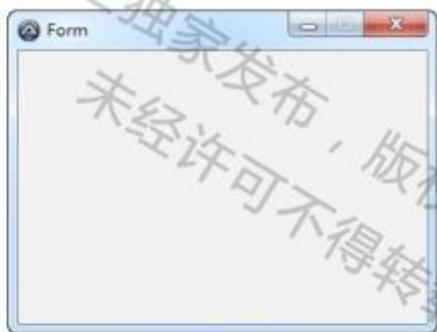
### GUICreate 的完整语法：

函数	说明
GUICreate 创建窗体	<p>语法：</p> <pre>GUICreate("标题", [宽度], [高度], [左侧], [顶部], [样式], [扩展样式], [父窗体句柄])</pre> <p>参数：</p> <ul style="list-style-type: none"><li>标题：窗体的标题</li><li>宽度：【可选参数】窗体的宽度（单位：像素）</li><li>高度：【可选参数】窗体的高度（单位：像素）</li><li>左侧：【可选参数】窗体左侧与显示区域左侧的距离（单位：像素） 默认值为-1， 默认值时窗体在显示区域内左右居中。</li><li>顶部：【可选参数】窗体顶部与显示区域顶部的距离（单位：像素） 默认值为-1， 默认值时窗体在显示区域内上下居中。</li><li>样式：窗体的主要显示样式（见下文说明），默认值为-1</li><li>扩展样式：窗体的其他显示样式（见下文说明），默认值为-1</li><li>父窗体句柄：多窗体时，如果当前窗体为子窗体，则应在创建时写上主窗体的句柄，以表示当前窗体为哪个窗体的子窗体。</li></ul> <p>返回值：</p> <ul style="list-style-type: none"><li>成功：返回窗体句柄（Handle）</li><li>失败：返回 0，并将@error 设置为 1</li></ul>

### 创建一个窗体：

```
$Form = GUICreate("Form", 300, 200, -1, -1, -1, -1)  
GUISetState(@SW_SHOW, $Form)
```

### 图例 ·



(图 3-18)

## 2、样式

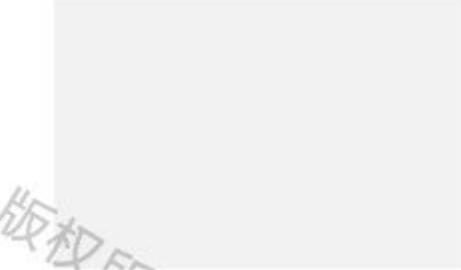
窗体的样式种类很多，这里介绍常用的几种，更多样式请参见帮助文档。另外，由于本章的目的是 GUI 设计入门，所以关于多窗体设计的样式值本章内不做介绍，后续章节中再行学习。

GUI 的样式是一系列的十六进制值，如 0x00C00000 表示窗体具有标题栏，这样的十六进制值难以记忆和应用，所以 Au3 中将这些值转化为了常量，保存于 WindowsConstants.au3 文件中，使用时直接将其 #include 下即可。

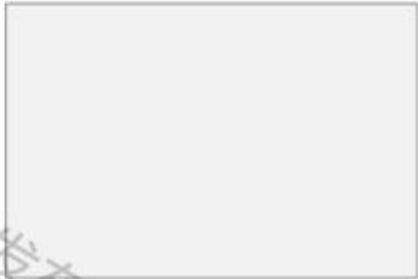
额外的，因样式有多种，当需要同时使用多种样式时，可使用 BitOR 函数进行合并。暂不多介绍 BitOR 函数的实际功能，目前只要会用即可。使用 BitOR 函数合并多个样式的方法：

```
BitOR(样式 1, 样式 2, ..., 样式 n)
```

### (1) \$WS\_POPUP

样式常量 (值)	说明
\$WS_POPUP (值：0x00800000)	<p>样式： 弹出式窗口（GUI 的默认样式之一），与子窗体进行区分。</p> <p>代码：  <code>#include &lt;WindowsConstants.au3&gt;</code>  <code>\$Form = GUICreate("Form", 300, 200, -1, -1, \$WS_POPUP)</code>  <code>GUISetState(@SW_SHOW, \$Form)</code></p> <p>图例：</p> 
	<p>备注：</p> <p>可能会有人有疑问，这也算个窗体？其实这才是真正的窗体，平时我们常见的各种窗体类型都是在基本窗体上做的衍生，例如增加标题栏、最小化按钮、最大化按钮、退出按钮等。我们会在本节逐步学到这些知识。</p>

## (2) \$WS\_BORDER

样式常量(值)	说明
\$WS_BORDER (值: 0x00800000)	<p>样式： 使窗体周围带有边框样式。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_BORDER)) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p>  <p>备注： 窗体周围多了一圈黑色边框。值得注意的是，当窗体带有标题栏等属性时，本样式的效果就基本看不出来了。</p>

(3) \$WS\_DLGFRA  
ME

样式常量(值)	说明
\$WS_DLGFRA ME (值: 0x00400000)	<p>样式： 对话框边框样式。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_DLGFRA ME)) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> 

	备注： 窗体周围有一定的立体效果。
--	----------------------

### ( 4 ) \$WS\_SIZEBOX/\$WS\_THICKFRAME

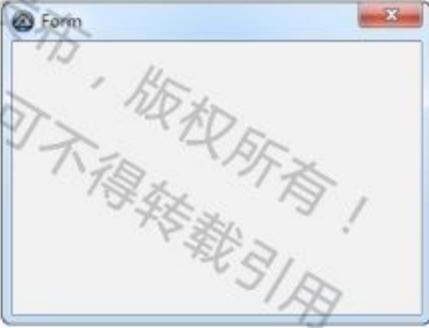
样式常量(值)	说明
\$WS_SIZEBOX 或 \$WS_THICKFRAME ( 值 : 0x00400000 )	<p>样式： 样式：窗口具有可调大小的边框。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_SIZEBOX)) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> 
	<p>备注：</p> <p>窗体建立后，窗体周围会出现视觉效果边条，拖动边条可调整窗体大小。</p>

### ( 5 ) \$WS\_CAPTION

样式常量(值)	说明
\$WS_CAPTION ( 值 : 0x00C00000 )	<p>样式： 样式：窗体带有标题栏。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_CAPTION)) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> 

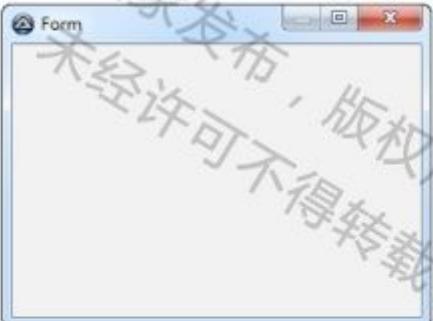
	<p>备注：</p> <p>窗体建立后，窗体具有标题栏，可设置窗体标题名（如 Form）。</p>

## (6) \$WS\_SYSMENU

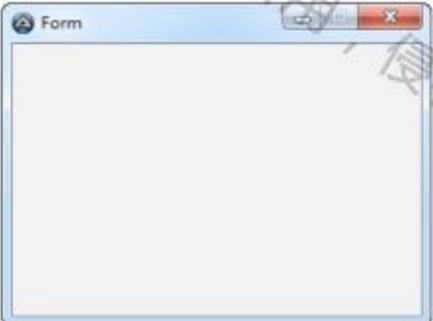
样式常量(值)	说明
\$WS_SYSMENU (值：0x00080000)	<p>样式： 样式：窗体标题栏具有系统菜单。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU)) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p>  <p>备注：</p> <p>窗体标题栏单击鼠标右键可看到系统菜单，同时标题栏右上角出现退出按钮。</p>

## (7) \$WS\_MAXIMIZEBOX

样式常量(值)	说明
\$WS_MAXIMIZEBOX (值：0x00010000)	<p>样式： 样式：窗体标题栏带有最大化按钮。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU,         \$WS_MAXIMIZEBOX)) GUISetState(@SW_SHOW, \$Form)</pre>

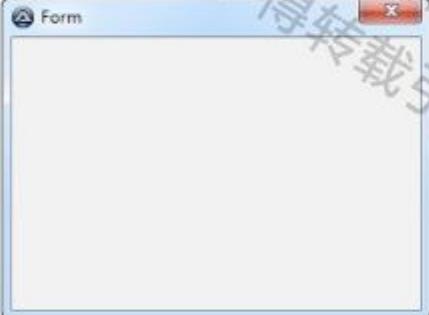
	<p>图例：</p> 
	<p>备注：</p> <p>必须与\$WS_SYSMENU 样式同时使用，不能与\$WS_EX_CONTEXTHELP( 扩展样式，稍后介绍 ) 同时使用。</p>

#### ( 8 ) \$WS\_MINIMIZEBOX

样式常量 ( 值 )	说明
\$WS_MINIMIZEBOX ( 值 : 0x00020000 )	<p>样式：</p> <p>样式：窗体标题栏出现最小化按钮。</p>
	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU,     \$WS_MINIMIZEBOX)) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p> 
	<p>备注：</p> <p>必须与\$WS_SYSMENU 样式同时使用，不能与\$WS_EX_CONTEXTHELP( 扩展样式，稍后介绍 ) 同时使用。</p>

#### ( 9 ) \$WS\_POPUPWINDOW

样式常量 ( 值 )	说明
\$WS_POPUPWINDOW ( 值 : 0x80880000 )	样式： 等同于同时使用\$WS_BORDER、\$WS_POPUP 和\$WS_SYSMENU。

<p>代码 :</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_CAPTION, \$WS_POPUPWINDOW)) GUISetState(@SW_SHOW, \$Form)</pre>
<p>图例 :</p> 
<p>备注 :</p> <p>等同于代码 : BitOR(\$WS_BORDER, \$WS_POPUP, \$WS_SYSMENU)。虽然有这类整合型样式，但仍旧建议将各个样式分开书写，便于维护和修改。</p>

## ( 10 ) \$WS\_DISABLED

样式常量 ( 值 )	说明
<p>\$WS_DISABLED ( 值 : 0x08000000 )</p>	<p>样式 :</p> <p>窗体初始状态为禁用。</p> <p>代码 :</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU, _          \$WS_DISABLED)) GUISetState(@SW_SHOW, \$Form)</pre> <p>备注 :</p> <p>这种状态下窗体不响应用户操作，如果窗体中包含控件，则控件同样不响应用户操作。因窗体为禁用状态，右上角的退出键也会变得无效。如果需要退出，可单击窗体后使用 ESC 键退出，或鼠标单击脚本任务栏图标，在弹出菜单中选择退出脚本。</p>

## ( 11 ) \$WS\_VISIBLE

样式常量 ( 值 )	说明
<p>\$WS_VISIBLE ( 值 : 0x10000000 )</p>	<p>样式 :</p> <p>窗体的初始状态为可见。</p> <p>代码 :</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU, _</pre>

	\$WS_VISIBLE))
	<p><b>备注：</b> 如果包含此样式，则创建窗体后不需要使用 GUISetState(@SW_SHOW)来显示窗体。但仍旧建议创建窗体时使窗体隐藏，待所有控件、数据等都创建完毕后，再显示窗体。</p>

### ( 12 ) \$WS\_MAXIMIZE

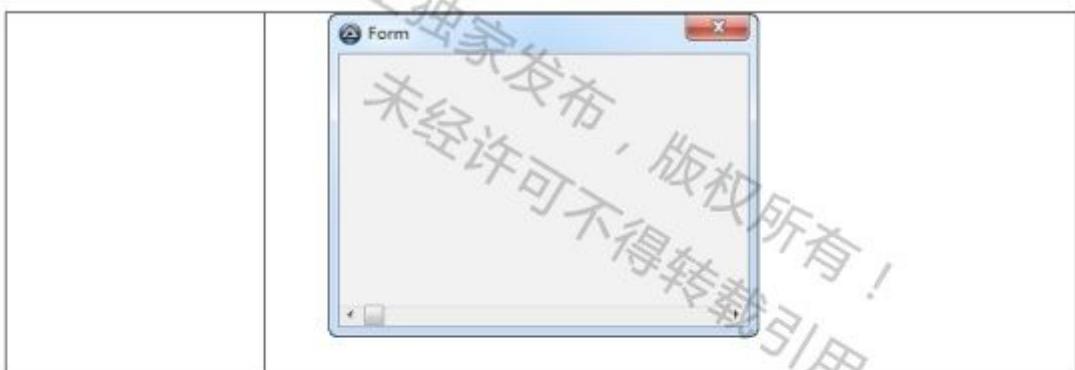
样式常量(值)	说明
\$WS_MAXIMIZE ( 值 : 0x01000000 )	<p><b>样式：</b> 窗体的初始状态为最大化。</p> <p><b>代码：</b></p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU, _          \$WS_MAXIMIZE)) GUISetState(@SW_SHOW, \$Form)</pre> <p><b>备注：</b> 窗体在创建后自动最大化，充满整个显示区域。</p>

### ( 13 ) \$WS\_MINIMIZE

样式常量(值)	说明
\$WS_MINIMIZE ( 值 : 0x20000000 )	<p><b>样式：</b> 窗体的初始状态为最小化。</p> <p><b>代码：</b></p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU, _          \$WS_MINIMIZE)) GUISetState(@SW_SHOW, \$Form)</pre> <p><b>备注：</b> 窗体在创建后会自动最小化至任务栏。</p>

### ( 14 ) \$WS\_HSCROLL

样式常量(值)	说明
\$WS_HSCROLL ( 值 : 0x00100000 )	<p><b>样式：</b> 窗体具有水平滚动条。</p> <p><b>代码：</b></p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU, _          \$WS_HSCROLL)) GUISetState(@SW_SHOW, \$Form)</pre> <p><b>图例：</b></p>



## (15) \$WS\_VSCROLL

样式常量(值)	说明
\$WS_VSCROLL (值: 0x00200000)	<p>样式： 窗体具有垂直滚动条。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU, \$WS_VSCROLL)) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p>

至此，我们已经掌握了绝大多数基本样式了。需要说明的是：

- 很多样式都是可以混合使用的，请根据 GUI 的实际需求选择合适的样式。例如\$WS\_POPUP、\$WS\_CAPTION、\$WS\_SYSMENU 就经常一起使用。
- 但也有些样式是相互冲突的，或者当某个样式生效时，某些其他样式就不会生效。例如窗体有标题栏时，窗体周围的一些属性就会没有效果；又例如不可能要求窗体的初始状态既为最大化又为最小化。

· 如果未指定样式，样式的默认值为：

```
BitOR($WS_MINIMIZEBOX, $WS_CAPTION, $WS_POPUP, $WS_SYSMENU)
```

### 3、扩展样式

窗体的扩展样式种类很多，这里介绍最常用的几种，更多样式请参见帮助文档。另外，由于本章的目的是 GUI 设计入门，所以关于多窗体的样式值，本章内不做介绍，后续章节中再行学习。

GUI 的扩展样式是一系列的十六进制值，如 0x00000200 表示窗体具有凹陷的边缘，这样的十六进制值难于记忆和应用，所以 Au3 中将这些值转化为了常量，保存于 WindowsConstants.au3 文件中，使用时直接将其 #include 下即可。

### ( 1 ) \$WS\_EX\_CLIENTEDGE

样式常量(值)	说明
\$WS_EX_CLIENTEDGE ( 值 : 0x00000200 )	<p>样式： 窗体具有凹陷的边缘。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, \$WS_POPUP,_ \$WS_EX_CLIENTEDGE) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> 

### ( 2 ) \$WS\_EX\_WINDOWEDGE

样式常量(值)	说明
\$WS_EX_WINDOWEDGE ( 值 : 0x00000100 )	<p>样式： 窗体具有凸起的边缘。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, \$WS_POPUP,_ \$WS_EX_WINDOWEDGE) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> 

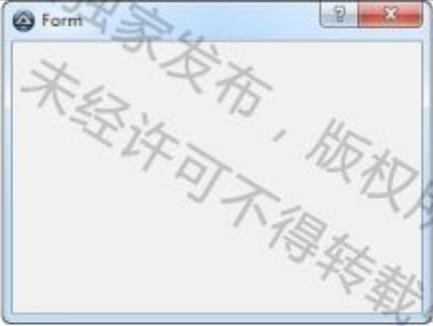
备注： 这也是扩展样式的默认值	

## (3) \$WS\_EX\_STATICEDGE

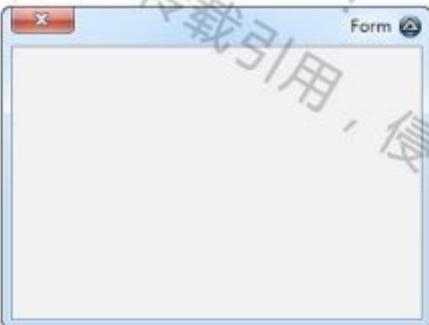
样式常量(值)	说明
\$WS_EX_STATICEDGE (值：)	<p>样式： 窗体带有立体的边缘</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, \$WS_POPUP,_ \$WS_EX_STATICEDGE) GUISetState(@SW_SHOW, \$Form)</pre>
图例：	
	<p>备注： 一般用于不允许用户输入任何数据的窗体</p>

## (4) \$WS\_EX\_CONTEXTHELP

样式常量(值)	说明
\$WS_EX_CONTEXTHELP (值：0x00000400 )	<p>样式： 窗体标题栏包括一个问号按钮。</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,_ BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU),_ \$WS_EX_CONTEXTHELP) GUISetState(@SW_SHOW, \$Form)</pre>
图例：	

	
	<p>备注：</p> <p>不能与\$WS_MAXIMIZEBOX 或\$WS_MINIMIZEBOX 样式同时使用。</p>

#### ( 5 ) \$WS\_EX\_LAYOUTRTL

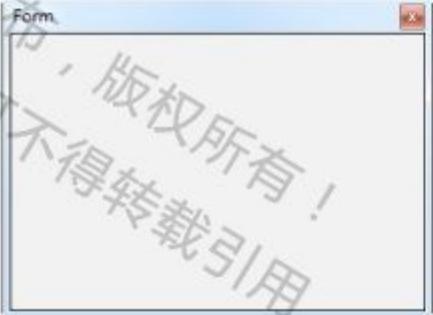
样式常量 ( 值 )	说明
\$WS_EX_LAYOUTRTL ( 值 : 0x400000 )	<p>样式：</p> <p>窗体布局从右至左</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU),_ \$WS_EX_LAYOUTRTL) GUICreate(@SW_SHOW, \$Form)</pre>
	<p>图例：</p> 
	<p>备注：</p> <p>一般用于一些特殊语言习惯的操作系统，或者纯粹为了搞怪.....</p>

#### ( 6 ) \$WS\_EX\_TOPMOST

样式常量 ( 值 )	说明
\$WS_EX_TOPMOST ( 值 : 0x00000008 )	<p>样式：</p> <p>窗体置于顶层</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU),_</pre>

	<p><code>\$WS_EX_TOPMOST)</code>  <code>GUISetState(@SW_SHOW, \$Form)</code></p>
	<p>备注：</p> <p>窗体将位于显示区域顶层，不会被其他窗口遮挡。</p>

## (7) \$WS\_EX\_TOOLWINDOW

样式常量(值)	说明
\$WS_EX_TOOLWINDOW (值：0x00000080)	<p>样式： 窗体作为浮动工具栏样式</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU), _     \$WS_EX_TOOLWINDOW) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p>  <p>备注： 标题栏比正常标题栏要短，并使用更小的字体，程序图标也不会显示于标题栏。窗体不会出现在任务栏，Alt+Tab 组合键也无法切换到此窗体。</p>

## (8) \$WS\_EX\_ACCEPTFILES

样式常量(值)	说明
\$WS_EX_ACCEPTFILES (值：0x00000010)	<p>样式： 允许窗体的输入类控件接受文件拖放</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1,     BitOR(\$WS_POPUP, \$WS_CAPTION, \$WS_SYSMENU), _     \$WS_EX_ACCEPTFILES) GUISetState(@SW_SHOW, \$Form)</pre> <p>备注： 此样式需要结合相应代码才能实现，将在后续章节中介绍。</p>

至此，我们已经掌握了绝大多数扩展样式了。需要说明的是：

- 很多扩展样式都是可以混合使用的，请根据 GUI 的实际需求选择合适的样式。例如 \$WS\_EX\_WINDOWEDGE、\$WS\_EX\_TOPMOST、\$WS\_EX\_ACCEPTFILES。
- 但也有些样式是相互冲突的，或者当某个样式生效时，某些其他样式就不会生效。例如窗体有标题栏时，窗体周围的一些属性就会没有效果；有最大最小化按钮时，问号按钮就不会生效。
- 如果未指定扩展样式，样式的默认值为：

\$WS_EX_WINDOWEDGE
--------------------

#### 4. GUIDelete

与其字面意思相同，GUIDelete 与 GUICreate 具有相反的功能，用于删除窗体。语法：

函数	说明
GUIDelete 删除窗体	<p>语法： GUIDelete([窗体句柄])</p> <p>参数： 窗体句柄：【可选参数】将要删除的窗体的句柄，如不设置则删除最近创建的一个窗体。</p> <p>返回值： 成功：返回 1 失败：返回 0</p>

那 GUIDelete 需要在什么情况下使用？到目前为止我们所学习的 GUI 设计都只有一个窗体，当程序退出时，窗体就自动被删除，这其实与在程序退出时执行一次 GUIDelete 的效果是相同的。在某些条件下，我们需要删除已创建的窗体，如多窗体应用程序，或多个窗体交替运行。

例如万能驱动助理 v6，在启动时，先启动一个窗体用于检测硬件：



(图 3-19)

当硬件检测完成时，GUIDelete 此检测窗体，然后 GUICreate 主窗体：



(图 3-20)

上例就是一个创建->删除->创建窗体的例子。

在这里还要强调的一个概念是，删除窗体并不代表程序退出，只是不再提供与用户的 GUI 交互罢了。虽然绝大多数应用程序关闭窗体相当于退出了程序，但概念认识上，一定要将“删除窗体”与“退出程序”分开来理解。很多程序在退出 GUI 后都会执行一些其他动作才真正退出，例如在删除窗体后删除临时文件等。

举个创建与删除窗体的例子：

```
$Form1 = GUICreate("Form1", 300, 200, -1, -1)
GUISetState(@SW_SHOW, $Form1)
Sleep(2000)
GUIDelete($Form1)

$Form2 = GUICreate("Form2", 200, 400, -1, -1)
GUISetState(@SW_SHOW, $Form2)
Sleep(2000)
GUIDelete($Form2)

Exit
```

创建窗体 1，令其显示，显示 2 秒后，将窗体 1 删除；

创建窗体 2，令其显示，显示 2 秒后，将窗体 2 删除。

GUIDelete 的用法，将在今后与大家探讨多窗体时更为详细的研究，本章仅作了解。

### 3.5 标签 (Label)、文本框 (Input)、按钮 (Button)

标签、文本框与按钮，可以说是最最基本的三个控件，有着极高的使用频率。在 3.2.2 节为大家

演示基本的控件创建法时，已经初步学习过这三个控件，而本节将要更为深入的研究一下。

### 3.5.1 标签控件

#### 1、GUICtrlCreateLabel

标签 ( Label ) 控件，一般而言是一块不可由用户直接编辑的文本区域，主要用于显示说明类文本，有时也用于输出任务执行结果。

标签控件一般由函数 GUICtrlCreateLabel 函数进行创建。

函数	说明
GUICtrlCreateLabel 创建标签控件	<p>语法： GUICtrlCreateLabel("文本", 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</p> <p>参数：</p> <ul style="list-style-type: none"><li>文本：控件的文本</li><li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li><li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li><li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li><li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li><li>样式：标签的主要显示样式（下文详解），默认值为-1</li><li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li></ul> <p>返回值：</p> <ul style="list-style-type: none"><li>成功：返回控件 ID</li><li>失败：返回 0</li></ul>

在窗体中创建标签：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Label1 = GUICtrlCreateLabel("Label1", 16, 16, 140, 17, -1, -1)
$Label2 = GUICtrlCreateLabel("Label2", 16, 48, 140, 17, -1, -1)
GUISetState(@SW_SHOW, $Form)
```

图例：



( 图 3-21 )

## 2. 样式

Label 的样式种类很多，这里介绍最常用的几种，更多样式请参见帮助文档。

Label 的样式是一系列的十六进制值，为了便于记忆，Au3 中将这些值转化为了常量，保存于 StaticConstants.au3 文件中，使用时直接将其 #include 下即可。

### (1) \$WS\_BORDER、\$SS\_SUNKEN

样式常量(值)	说明
\$WS_BORDER (值 : 0x00800000)	样式： \$WS_BORDER，使 Label 带有边框 (Label 默认无边框) \$SS_SUNKEN，使 Label 带有半凹陷效果 (Label 默认为平面效果)
\$SS_SUNKEN (值 : 0x1000)	代码： <pre>#include &lt;StaticConstants.au3&gt; #include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, -1, -1) \$Label1 = GUICtrlCreateLabel("Label1", 16, 16, 140, 17) \$Label2 = GUICtrlCreateLabel("Label2", 16, 48, 140, 17, \$WS_BORDER) \$Label3 = GUICtrlCreateLabel("Label3", 16, 80, 140, 17, \$SS_SUNKEN) GUISetState(@SW_SHOW, \$Form)</pre>
	图例： 

#### 备注：

其中，Label1 是默认样式，Label2 是带有边框的样式 (\$WS\_BORDER)，Label3 是半凹陷样式 (\$SS\_SUNKEN)。值得注意的是，\$WS\_BORDER 这个公共样式是保存于 WindowsConstants.au3 中的，而不是 StaticConstants.au3 中。

### (2) \$SS\_LEFT、\$SS\_RIGHT、\$SS\_CENTER

样式常量(值)	说明
\$SS_LEFT (值 : 0x0000)	样式： \$SS_LEFT，文本左对齐 (这也是默认样式) \$SS_RIGHT，文本右对齐 \$SS_CENTER，文本居中
\$SS_RIGHT (值 : 0x0002)	代码：(为了使效果更明显，笔者为 Label 额外添加了边框样式) <pre>#include &lt;StaticConstants.au3&gt; #include &lt;WindowsConstants.au3&gt;</pre>

( 值 : 0x01 )	<pre>\$Form = GUICreate("Form", 300, 200, -1, -1, -1) \$Label1 = GUICtrlCreateLabel("Label1", 16, 16, 140, 17) \$Label2 = GUICtrlCreateLabel("Label2", 16, 48, 140, 17, _     BitOR(\$WS_BORDER, \$SS_LEFT)) \$Label3 = GUICtrlCreateLabel("Label3", 16, 80, 140, 17, _     BitOR(\$WS_BORDER, \$SS_RIGHT)) \$Label4 = GUICtrlCreateLabel("Label4", 16, 112, 140, 17, _     BitOR(\$WS_BORDER, \$SS_CENTER)) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例 :</p>
	<p>备注 :</p> <p>其中 Label1 是默认样式 , Label2 是左对齐 , Label3 是右对齐 , Label4 是居中。另外 , 如果 Label 中的文本长度大于 Label 本身的长度 , 文本会被换行 , 而如果不想被换行 ( 超出部分被截断 ) , 可使用 \$SS_LEFTNOWORDWRAP 样式 , 即左对齐、超出字符被截断。</p>

### ( 3 ) \$SSETCHEDHORZ、\$SSETCHEDVERT

样式常量 ( 值 )	说明
\$SSETCHEDHORZ ( 值 : 0x10 )	<p>样式 :</p> <p>\$SSETCHEDHORZ , 横向蚀刻线 \$SSETCHEDVERT , 纵向蚀刻线</p>
\$SSETCHEDVERT ( 值 : 0x11 )	<p>代码 :</p> <pre>#include &lt;StaticConstants.au3&gt; #include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, -1) \$Label1 = GUICtrlCreateLabel("Label1", 16, 16, 140, 17, _     \$SSETCHEDHORZ) \$Label2 = GUICtrlCreateLabel("Label2", 16, 48, 17, 140, _     \$SSETCHEDVERT) GUISetState(@SW_SHOW, \$Form)</pre>

<p><b>备注：</b> 如果 Label 具有蚀刻线样式，则其文本不会再被显示。蚀刻线一般用于分隔一个控件区域与另一个控件区域，例如当两个控件区域功能不同，或为了使控件整体效果更为整洁时使用。上例中 Label1 为横向蚀刻线，Label2 为纵向蚀刻线。</p>	

### 3、扩展样式

Label 的扩展样式种类不多，且均为公共样式（非专有样式）。公共样式的常量保存于 WindowsConstants.au3、GUIConstantsEx.au3 文件中，使用时直接将其#include 下即可。

#### (1) \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量(值)	说明
\$WS_EX_CLIENTEDGE (值：0x00000200)	<p>样式： \$WS_EX_CLIENTEDGE，Label 边缘带有凹陷样式 \$WS_EX_STATICEDGE，Label 边缘带有三维边框样式</p>
\$WS_EX_STATICEDGE (值：0x00020000)	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, -1, -1) \$Label1 = GUICtrlCreateLabel("Label1", 16, 16, 140, 17) \$Label2 = GUICtrlCreateLabel("Label2", 16, 48, 140, 17, -1, -1, -1, -1, \$WS_EX_CLIENTEDGE) \$Label3 = GUICtrlCreateLabel("Label3", 16, 80, 140, 17, -1, -1, -1, -1, \$WS_EX_STATICEDGE) GUISetState(@SW_SHOW, \$Form)</pre>
<p><b>图例：</b></p>	

	<p><b>备注：</b></p> <p>其中 Label1 为默认样式，Label2 带有凹陷样式，Label3 带有三维边框样式。但实际上，如果不从官方解释上看，而从实际视觉效果上观察，凹陷样式（\$WS_EX_CLIENTEDGE）要比半凹陷样式（\$SS_SUNKEN）的凹陷程度深，而半凹陷样式（\$SS_SUNKEN）与三维边框样式（\$WS_EX_STATICEDGE）基本相同。</p>
--	---

## (2) \$GUI\_WS\_EX\_PARENTDRAG

样式常量(值)	说明
\$GUI_WS_EX_PARENTDRAG (值：0x00100000)	<p><b>样式：</b> 允许使用此 Label 控件拖动整个窗体</p> <p><b>代码：</b></p> <pre>#include &lt;GUIConstantsEx.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1, -1, -1) \$Label1 = GUICtrlCreateLabel("Label1", 16, 16, 140, 17, -1, _\$GUI_WS_EX_PARENTDRAG)</pre> <p><b>备注：</b> \$GUI_WS_EX_PARENTDRAG 样式一般不需要设置，但可用其灵活的解决一些 GUI 设计中的问题。例如使用图片制作更为个性化的标题栏时，由于原本标题栏被隐藏，窗体无法被拖动，可以使用一个无内容的 Label 放置于“标题栏”位置，实现窗体拖动。</p>

## 3.5.2 文本框控件

## 1、GUICtrlCreateInput

文本框 ( Input ) 控件，一般而言是一块可由用户直接编辑的文本区域，主要用于用户输入特定数据，有时也用于输出任务执行结果（便于用户复制或修改）。

文本框控件一般由函数 GUICtrlCreateInput 函数进行创建。

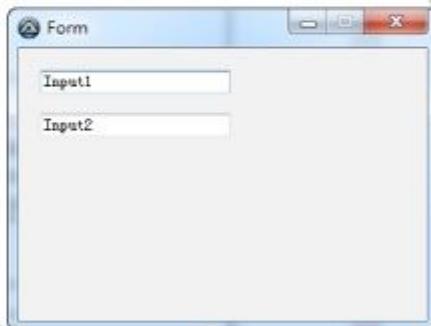
函数	说明
GUICtrlCreateInput 创建文本框控件	<p><b>语法：</b></p> <pre>GUICtrlCreateInput("文本", 左侧, 顶部, 宽度[, 高度[, 样式[, 扩展样式]]])</pre> <p><b>参数：</b></p> <ul style="list-style-type: none"> <li>文本：控件的文本</li> <li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li> <li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li> <li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li> <li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li> <li>样式：标签的主要显示样式（下文详解），默认值为-1</li> <li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li> </ul> <p><b>返回值：</b></p>

	成功：返回控件 ID 失败：返回 0
--	-----------------------

在窗体中创建文本框：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Input1 = GUICtrlCreateInput("Input1", 16, 16, 140, 17, -1, -1)
$Input2 = GUICtrlCreateInput("Input2", 16, 48, 140, 17, -1, -1)
GUISetState(@SW_SHOW, $Form)
```

图例：



( 图 3-22 )

## 2. 样式

Input 的样式种类很多，这里介绍最常用的几种，更多样式请参见帮助文档。

Input 的样式是一系列的十六进制值，为了便于记忆，Au3 中将这些值转化为了常量，保存于 EditConstants.au3 文件中，使用时直接将其#include 下即可。

### ( 1 ) \$ES\_LEFT、\$ES\_CENTER、\$ES\_RIGHT

样式常量 ( 值 )	说明
\$ES_LEFT ( 值 : 0x0000 )	样式： \$ES_LEFT : 文本左对齐 ( 这也是默认样式 ) \$ES_CENTER : 文本居中 \$ES_RIGHT : 文本右对齐
\$ES_CENTER ( 值 : 0x0001 )	代码： <pre>#include &lt;EditConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$input1 = GUICtrlCreateInput("Input1", 16, 16, 265, 21) \$input2 = GUICtrlCreateInput("Input2", 16, 48, 265, 21, \$ES_LEFT) \$input3 = GUICtrlCreateInput("Input3", 16, 80, 265, 21, \$ES_CENTER) \$input4 = GUICtrlCreateInput("Input4", 16, 112, 265, 21, \$ES_RIGHT) GUISetState(@SW_SHOW, \$Form)</pre>
\$ES_RIGHT ( 值 : 0x0002 )	图例：

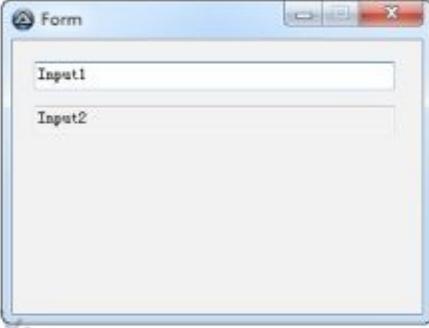


## (2) \$ES\_LOWERCASE、\$ES\_UPPERCASE、\$ES\_NUMBER、\$ES\_PASSWORD

样式常量(值)	说明
\$ES_LOWERCASE (值：0x0010)	样式： \$ES_LOWERCASE，将文本数据中的大写字母转化为小写字母 \$ES_UPPERCASE，将文本数据中的小写字母转化为大写字母 \$ES_NUMBER，仅允许数字数据 \$ES_PASSWORD，密码样式，所有文本数据转化为密码字符（如*）
\$ES_NUMBER (值：0x2000)	代码： <pre>#include &lt;EditConstants.au3&gt;  \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Input1 = GUICtrlCreateInput("", 16, 16, 265, 21, \$ES_LOWERCASE) \$Input2 = GUICtrlCreateInput("", 16, 48, 265, 21, \$ES_UPPERCASE) \$Input3 = GUICtrlCreateInput("", 16, 80, 265, 21, \$ES_NUMBER) \$Input4 = GUICtrlCreateInput("", 16, 112, 265, 21, \$ES_PASSWORD)  GUICtrlSetData(\$Input1, "abcDEF") GUICtrlSetData(\$Input2, "abcDEF") GUICtrlSetData(\$Input3, "12345678901234567890") GUICtrlSetData(\$Input4, "abc123")  GUISetState(@SW_SHOW, \$Form)</pre>
\$ES_PASSWORD (值：0x0020)	图例：

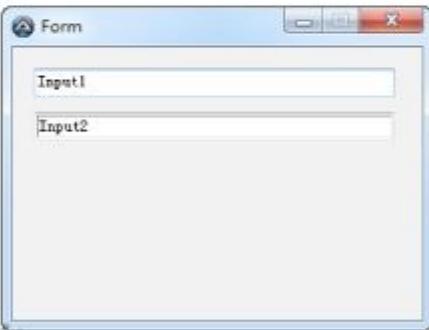
	
<p><b>备注：</b></p> <p>其中，Input1 会将大写字母转化为小写，所以即使为 Input1 设置数据“abcDEF”，显示数据却为“abcdef”；Input2 会将小写字母转化为大写，所以即使为 Input2 设置数据“abcDEF”，显示数据却为“ABCDEF”；Input3 仅允许输入数字，所以当输入非数字时会出现上图中提示；Input4 为密码输入框，所输入的任何数据都会被转化为密码字符（不同系统默认的密码字符不同）。</p>	

## (3) \$ES\_READONLY

样式常量(值)	说明
\$ES_READONLY (值：0x0800)	<p><b>样式：</b> 使文本框只读，用户不可直接编辑（但可以通过 GUICtrlSetData 设置其值）</p> <p><b>代码：</b></p> <pre>#include &lt;EditConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Input1 = GUICtrlCreateInput("Input1", 16, 16, 265, 21) \$Input2 = GUICtrlCreateInput("Input2", 16, 48, 265, 21, _      \$ES_READONLY) GUISetState(@SW_SHOW, \$Form)</pre> <p><b>图例：</b></p> 
<p><b>备注：</b></p> <p>其中，Input1 是正常样式，Input2 则为只读。Input2 中无法直接输入任何数据，同时底色会改变为窗体颜色，以示无法输入。</p>	

## (4) \$WS\_BORDER

样式常量(值)	说明

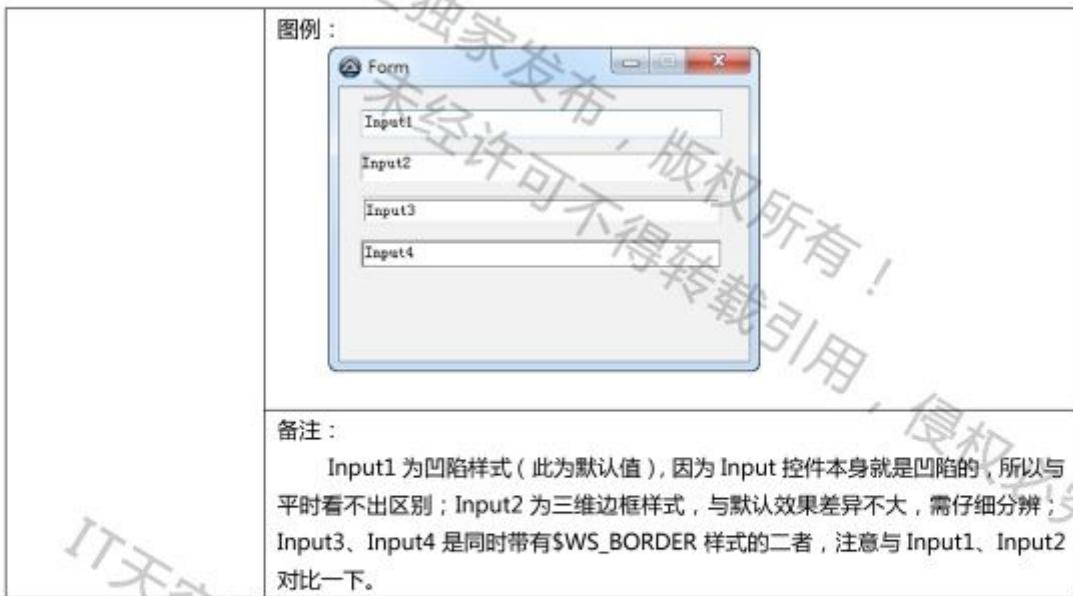
\$WS_BORDER ( 值 : 0x00800000 )	样式： 使控件带有边框
	代码： <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Input1 = GUICtrlCreateInput("Input1", 16, 16, 265, 21) \$Input2 = GUICtrlCreateInput("Input2", 16, 48, 265, 21, \$WS_BORDER) GUISetState(@SW_SHOW, \$Form)</pre>
	图例：
	
备注： Input1 为默认样式，Input2 为带有边框的样式。注意，\$WS_BORDER 是一个公共属性，包含于 WindowsConstants.au3 中。	

### 3、扩展样式

Input 控件的扩展样式种类不多，且均为公共样式（非专有样式）。公共样式的常量保存于 WindowsConstants.au3 文件中，使用时直接将其 #include 下即可。

#### \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量 (值)	说明
\$WS_EX_CLIENTEDGE ( 值 : 0x00000200 )	样式： \$WS_EX_CLIENTEDGE，Input 边缘带有凹陷样式 \$WS_EX_STATICEDGE，Input 边缘带有三维边框样式
\$WS_EX_STATICEDGE ( 值 : 0x00020000 )	代码： <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Input1 = GUICtrlCreateInput("Input1", 16, 16, 265, 21, _      - 1, \$WS_EX_CLIENTEDGE) \$Input2 = GUICtrlCreateInput("Input2", 16, 48, 265, 21, _      - 1, \$WS_EX_STATICEDGE) \$Input3 = GUICtrlCreateInput("Input3", 16, 80, 265, 21, _      \$WS_BORDER, \$WS_EX_CLIENTEDGE) \$Input4 = GUICtrlCreateInput("Input4", 16, 112, 265, 21, _      \$WS_BORDER, \$WS_EX_STATICEDGE) GUISetState(@SW_SHOW, \$Form)</pre>



### 3.5.3 按钮控件

#### 1、GUICtrlCreateButton

按钮 ( Button ) 控件，一般用于用户决定使程序执行某任务 ( 以单击按钮为触发条件 )。

按钮控件一般由函数 GUICtrlCreateButton 函数进行创建。

函数	说明
GUICtrlCreateButton 创建按钮控件	<p>语法： GUICtrlCreateButton("文本", 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>文本：控件的文本</li> <li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li> <li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li> <li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li> <li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li> <li>样式：标签的主要显示样式（下文详解），默认值为-1</li> <li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回控件 ID</li> <li>失败：返回 0</li> </ul>

在窗体中创建按钮：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Button1 = GUICtrlCreateButton("Button1", 16, 16, 57, 25)
$Button2 = GUICtrlCreateButton("Button2", 16, 48, 57, 25)
```

```
GUISetState(@SW_SHOW, $Form)
```

图例：



(图 3-23)

## 2. 样式

Button 控件的样式种类很多，这里介绍最常用的几种，更多样式请参见帮助文档。

Button 控件的样式是一系列的十六进制值，为了便于记忆，Au3 中将这些值转化为了常量，保存于 ButtonConstants.au3 文件中，使用时直接将其#include 下即可。

(1) \$BS\_LEFT、\$BS\_RIGHT、\$BS\_CENTER、\$BS\_VCENTER、\$BS\_TOP、\$BS\_BOTTOM

样式常量(值)	说明
\$BS_LEFT (值：0x0100)	样式： \$BS_LEFT，文本左对齐 \$BS_RIGHT，文本右对齐
\$BS_RIGHT (值：0x0200)	\$BS_CENTER，文本水平居中 \$BS_VCENTER，文本垂直居中
\$BS_CENTER (值：0x0300)	\$BS_TOP，文本贴近顶部 \$BS_BOTTOM，文本贴近底部
\$BS_VCENTER (值：0xC00)	代码： <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Button1 = GUICtrlCreateButton("Button1", 32, 88, 65, 33,     \$BS_LEFT) \$Button2 = GUICtrlCreateButton("Button2", 120, 88, 65, 33,     \$BS_RIGHT) \$Button3 = GUICtrlCreateButton("Button3", 208, 88, 65, 33,     \$BS_CENTER) \$Button4 = GUICtrlCreateButton("Button4", 32, 136, 65, 33,     \$BS_VCENTER) \$Button5 = GUICtrlCreateButton("Button5", 120, 136, 65, 33,     \$BS_TOP) \$Button6 = GUICtrlCreateButton("Button6", 208, 136, 65, 33,     \$BS_BOTTOM)</pre>
\$BS_TOP (值：0x400)	
\$BS_BOTTOM (值：0x800)	

	<p><code>GUISetState(@SW_SHOW, \$Form)</code></p> <p>图例：</p> <p>备注：</p> <p>Button1~6 以此是文本左对齐、右对齐、水平居中、垂直居中、靠近顶部、靠近底部。</p>
--	--

## (2) \$BS\_DEFPUSHBUTTON

样式常量(值)	说明
\$BS_DEFPUSHBUTTON (值：0x0001)	<p>样式：</p> <p>按钮样式为点击后的样式，而不是默认的等待点击的样式。</p> <p>代码：</p> <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Button1 = GUICtrlCreateButton("Button1", 32, 88, 65, 33, _      \$BS_DEFPUSHBUTTON) \$Button2 = GUICtrlCreateButton("Button2", 120, 88, 65, 33) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> <p>备注：</p> <p>当具有此样式时，无论按钮是否具有焦点，都可以使用回车点击这个按钮，适用于一些默认项目或需要快速点击的项目。</p>

( 3 ) \$BS\_MULTILINE

样式常量(值)	说明
\$BS_MULTILINE ( 值 : 0x2000 )	<p>样式： 当按钮的文本过长时自动换行</p> <p>代码：</p> <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Button1 = GUICtrlCreateButton("Button1 Button1", 32, 88, 65, 33) \$Button2 = GUICtrlCreateButton("Button2 Button2", 120, 88, 65, 33, _\$BS_MULTILINE) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p> 
	<p>备注：</p> <p>默认条件下按钮的描述文本并不会自动换行，多余文本会无法显示，如 Button1；而具有\$BS_MULTILINE 样式后，过多的文本将分行显示，如 Button2。</p>

( 4 ) \$WS\_BORDER

样式常量(值)	说明
\$WS_BORDER ( 值 : 0x00800000 )	<p>样式： 使控件带有边框</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Button1 = GUICtrlCreateButton("Button1", 32, 88, 65, 33) \$Button2 = GUICtrlCreateButton("Button2", 120, 88, 65, 33, _\$WS_BORDER) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p>



### 3、扩展样式

Button 控件的扩展样式种类不多，且均为公共样式（非专有样式）。公共样式的常量保存于 WindowsConstants.au3 文件中，使用时直接将其 #include 下即可。

#### \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量（值）	说明
\$WS_EX_CLIENTEDGE (值：0x00000200)	样式： \$WS_EX_CLIENTEDGE，Button 控件边缘带有凹陷样式 \$WS_EX_STATICEDGE，Button 控件边缘带有三维边框样式
\$WS_EX_STATICEDGE (值：0x00020000)	代码： <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 301, 201, -1, -1, -1, -1) \$Button1 = GUICtrlCreateButton("Button1", 32, 88, 65, 33, -1, \$WS_EX_CLIENTEDGE) \$Button2 = GUICtrlCreateButton("Button2", 120, 88, 65, 33, -1, \$WS_EX_STATICEDGE) \$Button3 = GUICtrlCreateButton("Button3", 208, 88, 65, 33) GUISetState(@SW_SHOW, \$Form)</pre>
	图例： 

	备注： Button1 带有凹陷样式，Button2 带有三维边框样式，Button3 为默认状态， 注意对比。
--	--

### 3.5.4 标签、文本框、按钮控件应用举例

#### 1、输入长方形的长和宽，计算长方形的面积。

##### GUI 设计与分析：

输入长、宽，输出面积，需要 3 个 Input 控件。同时，需要 3 个 Label 控件去分别解释这 3 个 Input 控件的作用。额外的，需要 1 个按钮来确认执行计算，并需要 1 个按钮来清空当前数据以方便再次计算。综上，GUI 设计为：



( 图 3-24 )

##### 代码：

```
#include <GUIContstantsEx.au3>
#include <WindowsConstants.au3>

AutoItSetOption("MustDeclareVars", 1)

Global $gu_Form = GUICreate("长方形面积计算", 275, 137, -1, -1,
                           BitOR($WS_POPUP, $WS_CAPTION, $WS_SYSMENU))
Global $gu_Label1 = GUICtrlCreateLabel("长方形的长", 16, 16, 64, 17)
Global $gu_Label2 = GUICtrlCreateLabel("长方形的宽", 16, 46, 64, 17)
Global $gu_Label3 = GUICtrlCreateLabel("长方形的面积", 16, 76, 76, 17)
Global $gu_Input1 = GUICtrlCreateInput("", 96, 12, 169, 21)
Global $gu_Input2 = GUICtrlCreateInput("", 96, 42, 169, 21)
Global $gu_Input3 = GUICtrlCreateInput("", 96, 72, 169, 21)
Global $gu_Button1 = GUICtrlCreateButton("计算", 144, 104, 57, 25)
Global $gu_Button2 = GUICtrlCreateButton("清空", 208, 104, 57, 25)

_Main()
Exit

Func _Main()
    GUISetState(@SW_SHOW, $gu_Form)
```

```

;定义变量用于存储长、宽、面积
Local $a, $b, $s

Local $nMsg
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit
        Case $gu_Button1 ;按钮 1 被按下
            ;读取文本框 1 和文本框 2 获取长和宽，并将读取的字符串转化为数值
            $a = Number(GUICtrlRead($gu_Input1))
            $b = Number(GUICtrlRead($gu_Input2))
            ;当长和宽都大于 0 时，执行计算，否则报错
            If $a > 0 And $b > 0 Then
                $s = $a * $b
                GUICtrlSetData($gu_Input3, $s)
            Else
                MsgBox(0, "错误", "长或宽存在错误，请重新输入")
            Endif
        Case $gu_Button2 ;按钮 2 被按下
            ;将文本框 1~3 的数据设置为空字符串，等同于清空数据
            GUICtrlSetData($gu_Input1, "")
            GUICtrlSetData($gu_Input2, "")
            GUICtrlSetData($gu_Input3, "")
    EndSwitch
WEnd
EndFunc

```

运行图：



(图 3-25)

## 2、输入用户名和密码，判断是否正确

### GUI 设计与分析：

输入用户名、密码，需要 2 个 Input 控件，同时需要 2 个 Label 控件去分别描述二者功能。需要一个“登录”按钮，额外需要一个“退出”按钮。

预先制定一份用户名、密码表，存储于二维数组中。当用户单击“登录”按钮后，分别获取用户

输入的用户名和密码，使用循环遍历的方式去二维数组中查找用户名、密码的匹配性。存在匹配，则登录成功，否则登录失败。



(图 3-26)

代码：

```
#include <EditConstants.au3>
#include <GUIConstantsEx.au3>
#include <WindowsConstants.au3>

AutoItSetOption("MustDeclareVars", 1)

Global $gu_Form1 = GUICreate("用户登录", 243, 115, -1, -1)
Global $gu_Label1 = GUICtrlCreateLabel("用户名", 8, 16 + 3, 40, 17)
Global $gu_Label2 = GUICtrlCreateLabel("密码", 8, 48 + 3, 28, 17)
Global $gu_Input1 = GUICtrlCreateInput("", 56, 16, 177, 21)
Global $gu_Input2 = GUICtrlCreateInput("", 56, 48, 177, 21, $ES_PASSWORD)
Global $gu_Button1 = GUICtrlCreateButton("登录", 120, 80, 57, 25)
Global $gu_Button2 = GUICtrlCreateButton("退出", 184, 80, 49, 25)

_Main()
Exit

Func _Main()
    GUISetState(@SW_SHOW)

    Local $User, $Psw

    Local $nMsg
    While 1
        $nMsg = GUIGetMsg()
        Switch $nMsg
            Case $GUI_EVENT_CLOSE
                Exit
            Case $gu_Button1
                ;按下 Button2 时，检测用户名密码匹配性
                ;分别获取用户输入的用户名密码
                $User = GUICtrlRead($gu_Input1)
                $Psw = GUICtrlRead($gu_Input2)
```

```

;用户名不可为空，但密码可为空
If $User <> "" Then
    If _CheckUserPsw($User, $Psw) Then
        ;用户名存在，且密码匹配
        MsgBox(0, "信息", "用户名和密码正确！")
    Else
        ;用户名或密码有误
        MsgBox(0, "信息", "无效的用户名或密码！")
    EndIf
Else
    ;用户没有输入用户名
    MsgBox(0, "信息", "请输入用户名")
EndIf

Case $gu_Button2
    ;按下 Button2 时，退出程序
    Exit
EndSwitch
WEnd
EndFunc  ;==> Main

;生成用户名密码列表
Func _ GetUserAndPswList()
    ;二维数组，第一列为 UID，第二列为用户名，第三列为密码
    Local $a_UserPsw[3] = [{"UID": "User", "Psw": "Psw"}, _ 
        [1, "Una", "abc123"], _ 
        [2, "Skye", "123abc"], _ 
        [3, "GMan", ""]]
    Return $a_UserPsw
EndFunc  ;==> _ GetUserAndPswList

;检查用户名密码正确性
Func _CheckUserPsw($User, $Psw)
    ;获取用户名密码数组
    Local $a_UserPsw = _ GetUserAndPswList()

    Local $i, $flg = 0
    ;以循环遍历数组，查找用户名密码的匹配性
    For $i = 1 To UBound($a_UserPsw, 1) - 1
        ;当用户名存在，且完全匹配（区分大小写）时，则匹配成功，跳出循环
        If $a_UserPsw[$i][1] = $User And
            $a_UserPsw[$i][2] == $Psw Then
            $flg = 1
            ExitLoop
        EndIf
    Next

```

```
If $flg = 1 Then  
    Return True  
Else  
    Return False  
EndIf  
EndFunc ;==>_CheckUserPsw
```

运行图：



( 图 3-27 )

其他说明：

( 1 ) 由 KODA 设计的 GUI，有时不能满足我们对细节的追求，例如即便是顶部相同的 Label 控件和 Input 控件，其文本内容也会存在一定的高低差别。这时其实没有必要再回到 KODA 中调整，而是直接在代码中调整即可。上例代码中的：

```
Global $gu_Label1 = GUICtrlCreateLabel("用户名", 8, 16 + 3, 40, 17)  
Global $gu_Label2 = GUICtrlCreateLabel("密码", 8, 48 + 3, 28, 17)
```

这两个 Label 控件的“顶部”参数都增加了 3，笔者特意留下了修改方式，供大家参阅。

( 2 ) 出于安全考虑，输入密码的文本框不能是明文的，所以用于输入密码的 Input 控件带有 \$ES\_PASSWORD 样式。

```
Global $gu_Input2 = GUICtrlCreateInput("", 56, 48, 177, 21, $ES_PASSWORD)
```

但无论密码是否明文显示，使用 GUICtrlRead 获取控件数据是没有影响的。另外必须指出的是，本例代码是比较简单的，所以预置的账号密码列表中密码是明文存储的，但在实际应用中，密码一般是加密存储的，不应明文存储。至于怎么加密解密，今后我们会共同学习。

( 3 ) 本例代码只列举了一个简单的用户名密码登录机制，但实际上在验证成功后并没有执行什么功能代码。一般而言，登录成功后会进入特定的操作界面或执行特定任务。

## 3.6 复选框 ( CheckBox )、单选框 ( Radio )、组 ( Group )

### 3.6.1 复选框控件

#### 1、GUICtrlCreateCheckbox

复选框 ( CheckBox ) 控件，一般用于使用户在多个既定条件中进行选择。多个选项一般不具有互斥性，可一个都不选，可选中一个，亦可同时选中多个。

复选框控件一般由 GUICtrlCreateCheckbox 函数进行创建。

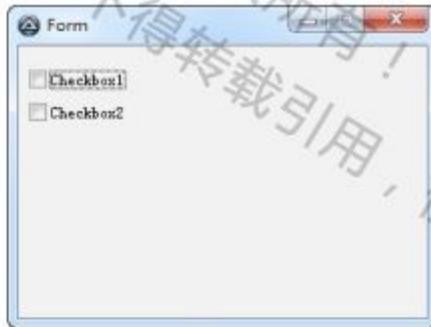
函数	说明
----	----

<b>GUICtrlCreateCheckbox</b> 创建复选框控件	<p>语法：</p> <pre>GUICtrlCreateCheckbox("文本", 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>文本：控件的文本</li> <li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li> <li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li> <li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li> <li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li> <li>样式：标签的主要显示样式（下文详解），默认值为-1</li> <li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回控件 ID</li> <li>失败：返回 0</li> </ul>
---	---

在窗体中创建复选框：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 8, 16, 137, 17)
$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 8, 40, 137, 17)
GUISetState(@SW_SHOW, $Form)
```

图例：



( 图 3-28 )

## 2、样式

CheckBox 控件的样式种类很多，这里介绍最常用的几种，更多样式请参见帮助文档。

CheckBox 控件的样式是一系列的十六进制值，为了便于记忆，Au3 中将这些值转化为了常量，保存于 ButtonConstants.au3 文件中，使用时直接将其#include 下即可。

### ( 1 ) \$BS\_AUTOCHECKBOX、\$BS\_AUTO3STATE

样式常量(值)	说明
\$BS_AUTOCHECKBOX ( 值 : 0x0003 )	样式： \$BS_AUTOCHECKBOX，CheckBox 控件具有选中、不选中两种状态，这也是默认样式。一般用于 2 种选择。

<b>\$BS_AUTO3STATE</b> ( 值 : 0x0006 )	<p><b>\$BS_AUTO3STATE</b>, CheckBox 控件具有选中、不确定、不选中三种状态( 三态 )。一般用于 3 种选择。</p> <p>代码 :</p> <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 16, 16, 89, 17) \$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 16, 40, 89, 17, _      \$BS_AUTOCHECKBOX) \$Checkbox3 = GUICtrlCreateCheckbox("Checkbox3", 16, 64, 89, 17, _      \$BS_AUTO3STATE) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例 :</p>
	<p>备注 :</p> <p>CheckBox1 为默认样式 ( 即 \$BS_AUTOCHECKBOX ), 未选中状态 ( 空 );          CheckBox2 为 \$BS_AUTOCHECKBOX 样式 , 选中状态 ( 对号 );          CheckBox3 为 \$BS_AUTO3STATE 样式 , 不确定状态 ( 方块 )。</p>

### ( 2 ) \$BS\_LEFT、\$BS\_RIGHT、\$BS\_CENTER

样式常量 ( 值 )	说明
\$BS_LEFT ( 值 : 0x0100 )	<p>样式 :</p> <p>\$BS_LEFT , 文本内容左对齐          \$BS_RIGHT , 文本内容右对齐          \$BS_CENTER , 文本内容居中</p>
\$BS_RIGHT ( 值 : 0x0200 )	<p>代码 :</p> <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 16, 16, 89, 17, _      \$BS_LEFT) \$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 16, 40, 89, 17, _      \$BS_RIGHT) \$Checkbox3 = GUICtrlCreateCheckbox("Checkbox3", 16, 64, 89, 17, _      \$BS_CENTER) GUISetState(@SW_SHOW, \$Form)</pre>
\$BS_CENTER ( 值 : 0x0300 )	

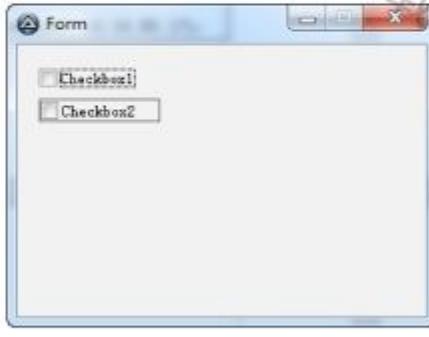
	<p>图例：</p> 
	<p>备注：</p> <p>一般情况下，文本均为左对齐。</p>

## (3) \$BS\_RIGHTBUTTON

样式常量(值)	说明
\$BS_RIGHTBUTTON (值：0x0020)	<p>样式：</p> <p>复选框的空心方框位于右侧，而不是一般情况下的左侧</p> <p>代码：</p> <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 16, 16, 73, 17) \$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 16, 40, 73, 17, _  \$BS_RIGHTBUTTON) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p> 

## (4) \$WS\_BORDER

样式常量(值)	说明
\$WS_BORDER (值：0x00800000)	使控件带有边框

	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 16, 16, 89, 17) \$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 16, 40, 89, 17, _      \$WS_BORDER) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p>  <p>备注：</p> <p>CheckBox1 为默认样式，CheckBox2 为带有边框的样式。注意，\$WS_BORDER 是一个公共属性，包含于 WindowsConstants.au3 中。</p>

### 3、扩展样式

CheckBox 控件的扩展样式种类不多，且均为公共样式（非专有样式）。

公共样式的常量保存于 WindowsConstants.au3 文件中，使用时直接将其 #include 下即可。

#### \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量(值)	说明
\$WS_EX_CLIENTEDGE (值：0x00000200)	<p>样式：</p> <p>\$WS_EX_CLIENTEDGE，控件边缘带有凹陷样式 \$WS_EX_STATICEDGE，控件边缘带有三维边框样式</p>
\$WS_EX_STATICEDGE (值：0x00020000)	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 16, 16, 89, 17, _      -1, \$WS_EX_CLIENTEDGE) \$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 16, 40, 89, 17, _      -1, \$WS_EX_STATICEDGE) \$Checkbox3 = GUICtrlCreateCheckbox("Checkbox3", 16, 64, 89, 17) GUISetState(@SW_SHOW, \$Form)</pre>



### 3.6.2 单选框控件

#### 1、GUICtrlCreateRadio

单选框 ( Radio ) 控件，一般用于使用户在多个既定条件中选择其中一个。同一组单选框之间具有互斥性，一次至多只能选中其中一个。

单选框控件一般由 GUICtrlCreateRadio 函数进行创建。

函数	说明
GUICtrlCreateRadio 创建单选框控件	<p>语法： GUICtrlCreateRadio("文本", 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>文本：控件的文本</li> <li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li> <li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li> <li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li> <li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li> <li>样式：标签的主要显示样式（下文详解），默认值为-1</li> <li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回控件 ID</li> <li>失败：返回 0</li> </ul>

在窗体中创建单选框：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Radio1 = GUICtrlCreateRadio("Radio1", 16, 16, 73, 17)
$Radio2 = GUICtrlCreateRadio("Radio2", 16, 48, 73, 17)
$Radio3 = GUICtrlCreateRadio("Radio3", 16, 80, 73, 17)
```

```
GUISetState(@SW_SHOW, $Form)
```

图例：



(图 3-29)

## 2. 样式

Radio 控件的样式种类很多，这里介绍最常用的几种，更多样式请参见帮助文档。

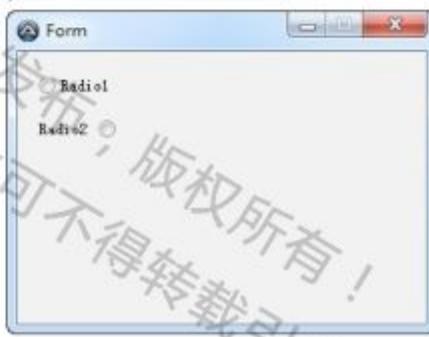
Radio 控件的样式是一系列的十六进制值，为了便于记忆，Au3 中将这些值转化为了常量，保存于 ButtonConstants.au3 文件中，使用时直接将其#include 下即可。

### (1) \$BS\_LEFT、\$BS\_RIGHT、\$BS\_CENTER

样式常量(值)	说明
\$BS_LEFT (值：0x0100)	样式： \$BS_LEFT，文本内容左对齐 \$BS_RIGHT，文本内容右对齐 \$BS_CENTER，文本内容居中
\$BS_RIGHT (值：0x0200)	代码： <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Radio1 = GUICtrlCreateRadio("Radio1", 16, 16, 73, 17, _\$BS_LEFT) \$Radio2 = GUICtrlCreateRadio("Radio2", 16, 48, 73, 17, _\$BS_RIGHT) \$Radio3 = GUICtrlCreateRadio("Radio3", 16, 80, 73, 17, _\$BS_CENTER) GUISetState(@SW_SHOW, \$Form)</pre>
\$BS_CENTER (值：0x0300)	图例：

	备注： 一般情况下，文本均为左对齐。
--	-----------------------

## (2) \$BS\_RIGHTBUTTON

样式常量(值)	说明
\$BS_RIGHTBUTTON (值：0x0020)	<p>样式： 单选框的空心圆形在右侧，而不是一般情况下的左侧</p> <p>代码：</p> <pre>#include &lt;ButtonConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Radio1 = GUICtrlCreateRadio("Radio1", 16, 16, 57, 17) \$Radio2 = GUICtrlCreateRadio("Radio2", 16, 48, 57, 17, _      \$BS_RIGHTBUTTON) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p>  <p>备注： 根据用户习惯和 GUI 整体布局进行调整，但一般仍建议遵守默认布局</p>

## (3) \$WS\_BORDER

样式常量(值)	说明
\$WS_BORDER (值：0x00800000)	<p>样式： 使控件带有边框</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Radio1 = GUICtrlCreateRadio("Radio1", 16, 16, 57, 17) \$Radio2 = GUICtrlCreateRadio("Radio2", 16, 48, 57, 17, _      \$WS_BORDER) GUISetState(@SW_SHOW, \$Form)</pre>

	<p>图例：</p> 
	<p>备注：</p> <p>Radio1 为默认样式，Radio2 为带有边框的样式。注意，\$WS_BORDER 是一个公共属性，包含于 WindowsConstants.au3 中。</p>

### 3、扩展样式

Radio 控件的扩展样式种类不多，且均为公共样式（非专有样式）。公共样式的常量保存于 WindowsConstants.au3 文件中，使用时直接将其#include 下即可。

#### \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量(值)	说明
\$WS_EX_CLIENTEDGE (值：0x00000200)	<p>样式：</p> <p>\$WS_EX_CLIENTEDGE，控件边缘带有凹陷样式 \$WS_EX_STATICEDGE，控件边缘带有三维边框样式</p>
\$WS_EX_STATICEDGE (值：0x00020000)	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Radio1 = GUICtrlCreateRadio("Radio1", 16, 16, 57, 17_ - 1, \$WS_EX_CLIENTEDGE) \$Radio2 = GUICtrlCreateRadio("Radio2", 16, 48, 57, 17_ - 1, \$WS_EX_STATICEDGE) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p> 

	备注： Radio1 带有凹陷样式，Radio2 带有三维边框样式。
--	---------------------------------------

### 3.6.3 组控件

#### 1、GUICtrlCreateGroup

组 ( Group ) 控件，一般用于将相关功能划分在一起，便于用户理解与操作。Group 控件还能将相关联的单选框划分为一组，当选中其中一个时，其他单选框自动取消选定。

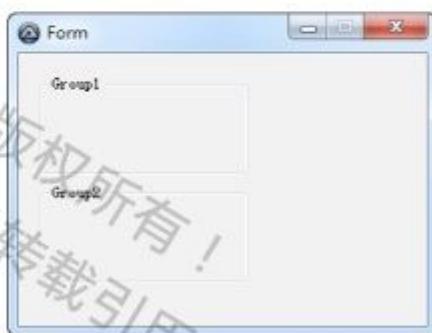
组控件一般由 GUICtrlCreateGroup 函数进行创建。

函数	说明
GUICtrlCreateGroup 创建组控件	<p>语法：</p> <pre>GUICtrlCreateGroup("文本", 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展 样式]]]])</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>文本：控件的文本</li> <li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li> <li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li> <li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li> <li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li> <li>样式：标签的主要显示样式（下文详解），默认值为-1</li> <li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回控件 ID</li> <li>失败：返回 0</li> </ul>

在窗体中创建标签：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Group1 = GUICtrlCreateGroup("Group1", 16, 16, 153, 73)
$Group2 = GUICtrlCreateGroup("Group2", 16, 96, 153, 73)
GUISetState(@SW_SHOW, $Form)
```

图例：



(图 3-30)

## 2、样式、扩展样式

由于 Group 控件主要功能就是将其他控件进行分组，所以基本没有专有的样式和扩展样式，本文不再赘述，有兴趣可以参见帮助文档中的“GUI 控件样式”表。

## 3、特殊注意

既然是 Group 控件用于将控件标记为一组，那么“组”由什么开始，又至什么结束？

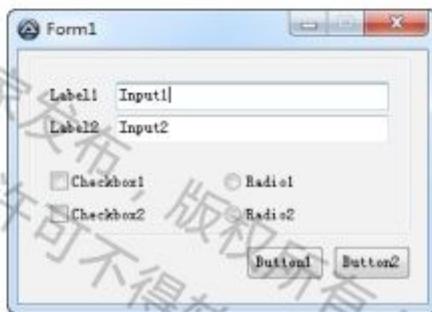
一般而言，自 GUICtrlCreateGroup 创建语句之后的控件，会被认为是同一组。那么组创建语句即是组的开始，亦被称为“组开始语句”。

而组一般是以下述语句为结束的：

```
GUICtrlCreateGroup("", -99, -99, 1, 1)
```

此语句被称之为“组结束语句”。

例如：



(图 3-31)

GUI 创建代码为：

```
$Form = GUICreate("Form", 301, 187, 192, 124)

$Group1 = GUICtrlCreateGroup("", 8, 0, 281, 137) ;组开始
$Label1 = GUICtrlCreateLabel("Label1", 24, 27, 36, 17)
$Label2 = GUICtrlCreateLabel("Label2", 24, 51, 36, 17)
$Input1 = GUICtrlCreateInput("Input1", 72, 24, 201, 21)
$Input2 = GUICtrlCreateInput("Input2", 72, 48, 201, 21)
$Checkbox1 = GUICtrlCreateCheckbox("Checkbox1", 24, 88, 89, 17)
$Checkbox2 = GUICtrlCreateCheckbox("Checkbox2", 24, 112, 89, 17)
$Radio1 = GUICtrlCreateRadio("Radio1", 152, 88, 57, 17)
$Radio2 = GUICtrlCreateRadio("Radio2", 152, 112, 57, 17)
GUICtrlCreateGroup("", -99, -99, 1, 1) ;组结束

$Button1 = GUICtrlCreateButton("Button1", 168, 144, 57, 25)
$Button2 = GUICtrlCreateButton("Button2", 232, 144, 57, 25)

GUISetState(@SW_SHOW, $Form)
```

当有多个组时，每个组均以此方法为结束：

```
$Form = GUICreate("Form", 300, 200, -1, -1)

$Group1 = GUICtrlCreateGroup("Group1", 16, 16, 153, 73);组 1 开始
GUICtrlCreateGroup("", -99, -99, 1, 1);组 1 结束

$Group2 = GUICtrlCreateGroup("Group2", 16, 96, 153, 73);组 2 开始
GUICtrlCreateGroup("", -99, -99, 1, 1);组 2 结束

GUISetState(@SW_SHOW, $Form)
```

值得注意的是，组开始语句与结束语句的匹配方式，类似于 If 与 EndIf 的匹配方式，即组开始语句总是与最近的未匹配的组结束语句进行匹配。例如当组包含组时：



(图 3-32)

```
$Form = GUICreate("Form", 300, 200, -1, -1)

$Group1 = GUICtrlCreateGroup("Group1", 8, 8, 225, 169);组 1 开始

$Group2 = GUICtrlCreateGroup("Group2", 16, 32, 201, 57);组 2 开始
GUICtrlCreateGroup("", -99, -99, 1, 1);组 2 结束

$Group3 = GUICtrlCreateGroup("Group3", 16, 104, 201, 57);组 3 开始
GUICtrlCreateGroup("", -99, -99, 1, 1);组 3 结束

GUICtrlCreateGroup("", -99, -99, 1, 1);组 1 结束

GUISetState(@SW_SHOW, $Form)
```

代码中组 1 的开始语句与最外侧的组结束语句匹配，从而形成了组 1 包含组 2、3。

### 3.6.4 复选框、单选框、组控件应用举例

在接触实例之前，我们还有些事情需要先解决。对于复选框和单选框控件而言，与标签、文本框控件最大的区别在于，前者主要表达的是状态（选中、非选中等），而后者表达的是数据。那么我们要怎样得知选中与否？又怎样通过代码设置其是否选中？

首先，获取复选框、单选框控件是否选中很简单，使用我们已经学过的 GUICtrlRead 即可。GUICtrlRead 函数具有良好的自适应性，对于不同种类的控件自动获取不同类型的数据。例如对于标签、文本框控件，GUICtrlRead 获取的是数据；而对复选框、单选框控件，GUICtrlRead 获取的是状态。

那么状态的值是什么？状态值也是一系列十六进制数，为方便记忆已被定义为常量并存储于 GUIConstantsEx.au3 文件中，使用时包含以下即可。

选中状态 : \$GUI_CHECKED
非选中状态 : \$GUI_UNCHECKED
不确定状态 ( 复选框三态样式 ) : \$GUI_INDETERMINATE

其次，设置控件的状态也不复杂，需要使用到 GUICtrlSetState 函数。

函数	说明
GUICtrlSetState 设置控件状态	<p>语法： GUICtrlSetState(控件 ID, 状态)</p> <p>参数：</p> <p>控件 ID：控件创建函数所返回的 ID</p> <p>状态（常用值）：</p> <p>\$GUI_UNCHECKED，非选中</p> <p>\$GUI_CHECKED，选中</p> <p>\$GUI_INDETERMINATE，不确定（复选框三态）</p> <p>\$GUI_SHOW，显示控件</p> <p>\$GUI_HIDE，隐藏控件</p> <p>\$GUI_ENABLE，启用控件</p> <p>\$GUI_DISABLE，禁用控件</p> <p>\$GUI_FOCUS，使控件获取焦点</p> <p>\$GUI_NOFOCUS，使控件放弃焦点</p> <p>返回值：</p> <p>成功：返回 1</p> <p>失败：返回 0</p>

额外的，仍可使用 GUICtrlSetData 来设置复选框、单选框、组控件的文本部分。这方面使用方法与对文本框、标签、按钮设置文本没有区别。

好了，了解了这些后，我们来看个实例。

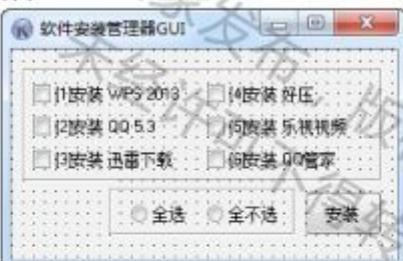
### 设计一个简单的软件安装管理器 GUI

#### GUI 设计分析：

软件安装管理器，即将数个软件集合在一起，等待用户选择需要安装的软件，当用户单击“安装”按钮时，所有软件一同安装。当然，在此我们只是做个 GUI，并不具有实际功能。（软件安装功能需要学习更多知识，并配以大量的实践才能够制作出来，努力学习吧！）

多个软件之间，一般没有互斥关系，要选择一个或多个，需要使用复选框。除此之外，还需要“全

选”和“全不选”功能，便于操作。



(图 3-33)

**代码：**

```
#include <GUIConstantsEx.au3>

AutoItSetOption("MustDeclareVars", 1)

Global $gu_Form1 = GUICreate("软件安装管理器 GUI", 285, 156, -1, -1)

Global $gu_Group1 = GUICtrlCreateGroup("", 8, 8, 265, 89)
Global $gua_Checkbox[7] ;将复选框控件创建为一个数组，便于程序遍历
$gua_Checkbox[1] = GUICtrlCreateCheckbox("[1]安装 WPS 2013", 16, 24, 121, 17)
$gua_Checkbox[2] = GUICtrlCreateCheckbox("[2]安装 QQ 5.3", 16, 48, 121, 17)
$gua_Checkbox[3] = GUICtrlCreateCheckbox("[3]安装 迅雷下载", 16, 72, 121, 17)
$gua_Checkbox[4] = GUICtrlCreateCheckbox("[4]安装 好压", 144, 24, 121, 17)
$gua_Checkbox[5] = GUICtrlCreateCheckbox("[5]安装 乐视视频", 144, 48, 121, 17)
$gua_Checkbox[6] = GUICtrlCreateCheckbox("[6]安装 QQ 管家", 144, 72, 121, 17)
GUICtrlCreateGroup("", -99, -99, 1, 1)

Global $gu_Group2 = GUICtrlCreateGroup("", 72, 96 + 1, 137, 41)
Global $gu_Radio1 = GUICtrlCreateRadio("全选", 88, 112, 49, 17)
Global $gu_Radio2 = GUICtrlCreateRadio("全不选", 144, 112, 57, 17)
GUICtrlCreateGroup("", -99, -99, 1, 1)

Global $gu_Button1 = GUICtrlCreateButton("安装", 216, 104, 57, 33)

_Main()
Exit

Func _Main()

;将单选框 1 设置为选中（即全选选项被选中）
GUICtrlSetState($gu_Radio1, $GUIT_CHECKED)
;全选所有复选框
_Select(1)

GUISetState(@SW_SHOW)
```

```
Local $i, $s = ""

Local $nMsg
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit
        Case $gu_Radio1 ;全选
            _Select(1)
        Case $gu_Radio2 ;全不选
            _Select(0)
        Case $gu_Button1
            $s = ""
            ;遍历复选框数组
            For $i = 1 To UBound($gua_Checkbox, 1) - 1
                ;如果复选框为选中，则记录其序号到$s 变量中
                If GUICtrlRead($gua_Checkbox[$i]) = $GUI_CHECKED Then
                    $s &= $i & " "
                EndIf
            Next
            If $s <> "" Then
                ;$s 变量不为空时，则有复选框被选中
                ;去掉$s 中字符串两端的空格，并将字符串间空格转化为顿号
                $s = StringReplace(StringStripWS($s, 3), " ", ", ")
                ;展示被选中的复选框序号
                MsgBox(0, "提示", "复选框：" & $s & " 被选中")
            Else
                ;$s 变量为空时，则没有复选框被选中
                MsgBox(0, "提示", "没有复选框被选中")
            EndIf
        EndSwitch
    WEnd

EndFunc ;==>_Main

;复选框全选、全部选函数
Func _Select($Mode)
    Local $i
    Switch $Mode
        Case 1 ;参数$Mode 为 1 时，全选
            ;遍历复选框数组，将所有复选框状态设置为选中
            For $i = 1 To UBound($gua_Checkbox, 1) - 1
                GUICtrlSetState($gua_Checkbox[$i], $GUI_CHECKED)
            Next
    EndFunc
```

```

Case 0 ;参数$Mode 为 0 时，全不选
;遍历复选框数组，将所有复选框状态设置为不选中
For $i = 1 To UBound($gua_Checkbox, 1) - 1
    GUICtrlSetState($gua_Checkbox[$i], $GUI_UNCHECKED)
Next
EndSwitch
EndFunc  ;==>_Select

```

运行图：



(图 3-34)

(图 3-35)

## 3.7 图像 (Pic)、图标 (Icon)

### 3.7.1 图片控件

#### 1、GUICtrlCreatePic

图像 (Pic) 控件，一般用于静态的展示图片文件 ( bmp、jpg、非动态 gif 等 )，通过合理的规划使用，可使 GUI 更为丰富多样。值得注意的是，一定要将图像控件与图形控件区分开，两者功能并不相同。

图像控件一般由 GUICtrlCreatePic 函数进行创建。

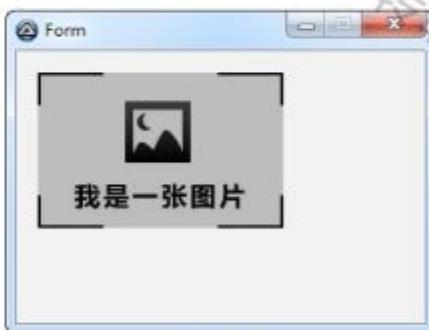
函数	说明
GUICtrlCreatePic 创建图像控件	<p>语法：</p> <p>GUICtrlCreatePic("图片文件", 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>图片文件：将要加载的图片文件路径（相对路径或绝对路径）</li> <li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li> <li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li> <li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li> <li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li> <li>样式：标签的主要显示样式（下文详解），默认值为-1</li> <li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li> </ul> <p>返回值：</p>

	成功：返回控件 ID 失败：返回 0
--	-----------------------

在窗体中创建：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Pic1 = GUICtrlCreatePic("D:\Temp\demo.jpg", 16, 16, 180, 114)
GUISetState(@SW_SHOW, $Form)
```

图例：



( 图 3-36 )

## 2、样式

Pic 控件一般仅用于展示图片文件，没有太多的样式。常用样式均为公共样式，存储于 WindowsConstants.au3 和 GUIConstantsEx.au3 中，使用时直接将其#include 下即可。

### \$WS\_BORDER

样式常量(值)	说明
\$WS_BORDER (值：0x00800000)	<p>样式： 使控件带有边框</p> <p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Pic1 = GUICtrlCreatePic("D:\Temp\demo2.jpg", _     16, 11, 120, 80) \$Pic2 = GUICtrlCreatePic("D:\Temp\demo2.jpg", _     16, 101, 120, 80,_     \$WS_BORDER) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p>



### 3. 扩展样式

Pic 控件一般仅用于展示图片文件，没有太多的扩展样式。常用扩展样式均为公共扩展样式，存储于 WindowsConstants.au3 和 GUIConstantsEx.au3 中，使用时直接将其#include 下即可。

#### (1) \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量(值)	说明
\$WS_EX_CLIENTEDGE (值 : 0x00000200)	样式： \$WS_EX_CLIENTEDGE , Pic 控件边缘带有凹陷样式 \$WS_EX_STATICEDGE , Pic 控件边缘带有三维边框样式
\$WS_EX_STATICEDGE (值 : 0x00020000)	代码： <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Pic1 = GUICtrlCreatePic("D:\Temp\demo2.jpg", _      16, 11, 120, 80, _      -1,\$WS_EX_CLIENTEDGE) \$Pic1 = GUICtrlCreatePic("D:\Temp\demo2.jpg", _      16, 101, 120, 80, _      -1,\$WS_EX_STATICEDGE) GUISetState(@SW_SHOW, \$Form)</pre>
图例：	

	<p>备注：</p> <p>Pic1 带有凹陷样式，Pic2 带有三维边框样式。</p>
<b>( 2 ) \$GUI_WS_EX_PARENTDRAG</b>	
样式常量 ( 值 )	说明
\$GUI_WS_EX_PARENTDRAG ( 值 : 0x00800000 )	<p>样式：</p> <p>通过拖动图像控件而拖动整个窗体</p> <p>代码：</p> <pre>#include &lt;GUIConstantsEx.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Pic1 = GUICtrlCreatePic("D:\Temp\demo.jpg",     16, 16, 180, 114,     -1, \$GUI_WS_EX_PARENTDRAG) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> <p>备注：</p> <p>此 GUI 中，使鼠标位于图片控件上方，按住鼠标左键，当移动鼠标时即可拖动整个窗体。</p>

### 3.7.2 图标框控件

#### 1、GUICtrlCreateIcon

图标 ( Icon ) 控件，一般用于静态的展示图标文件，通过合理的规划使用，可使 GUI 更为丰富多样。

图标控件一般由 GUICtrlCreateIcon 函数进行创建。

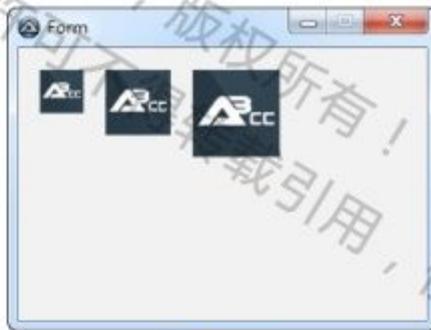
函数	说明
GUICtrlCreateIcon 创建图标控件	<p>语法：</p> <pre>GUICtrlCreateIcon("图标文件", 图标序号, 左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</pre> <p>参数：</p> <p>图标文件：将要加载的图标文件路径（相对路径或绝对路径）</p>

	<p>图标序号：如果图标文件中包含多个图标，则必须制定其序号，否则使用值-1代替</p> <p>左侧：控件左侧距离窗体左侧的距离（单位：像素）</p> <p>顶部：控件顶部距离窗体顶部的距离（单位：像素）</p> <p>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</p> <p>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</p> <p>样式：标签的主要显示样式（下文详解），默认值为-1</p> <p>扩展样式：标签的其他显示样式（下文详解），默认值为-1</p> <p>返回值：</p> <p>成功：返回控件 ID</p> <p>失败：返回 0</p>
--	---

在窗体中创建：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Icon1 = GUICtrlCreateIcon("D:\Temp\demo1.ico", -1, 16, 16, 32, 32)
$Icon2 = GUICtrlCreateIcon("D:\Temp\demo2.ico", -1, 64, 16, 48, 48)
$Icon3 = GUICtrlCreateIcon("D:\Temp\demo3.ico", -1, 128, 16, 64, 64)
GUISetState(@SW_SHOW, $Form)
```

图例：



(图 3-37)

顺道一提的是，图标的常见尺寸一般有 16×16、32×32、48×48、64×64、128×128 等，一般以 16 作为步进值，通过网络收集来的图标大多提供这些标准尺寸。如非特殊需要，不要自定义一些特殊尺寸。

## 2、样式

Icon 控件一般仅用于展示图标文件，没有太多的样式。常用样式均为公共样式，存储于 WindowsConstants.au3 和 GUIConstantsEx.au3 中，使用时直接将其#include 下即可。

### \$WS\_BORDER

样式常量（值）	说明
\$WS_BORDER (值：0x00800000)	样式： 使控件带有边框

	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Icon1 = GUICtrlCreateIcon("D:\Temp\test1.ico", -1,_ 16, 16, 48, 48) \$Icon2 = GUICtrlCreateIcon("D:\Temp\test1.ico", -1,_ 88, 16, 48, 48, \$WS_BORDER) GUISetState(@SW_SHOW, \$Form)</pre>
	<p>图例：</p> 
	<p>备注：</p> <p>\$Icon1 为默认样式，\$Icon2 带有边框</p>

### 3、扩展样式

Icon 控件一般仅用于展示图标文件，没有太多的扩展样式。常用扩展样式均为公共扩展样式，存储于 WindowsConstants.au3 和 GUIConstantsEx.au3 中，使用时直接将其 #include 下即可。

#### \$WS\_EX\_CLIENTEDGE、\$WS\_EX\_STATICEDGE

样式常量(值)	说明
\$WS_EX_CLIENTEDGE (值：0x00000200)	<p>样式：</p> <p>\$WS_EX_CLIENTEDGE，Icon 控件边缘带有凹陷样式 \$WS_EX_STATICEDGE，Icon 控件边缘带有三维边框样式</p>
\$WS_EX_STATICEDGE (值：0x00020000)	<p>代码：</p> <pre>#include &lt;WindowsConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Icon1 = GUICtrlCreateIcon("D:\Temp\test1.ico", -1,_ 16, 16, 48, 48, -1, \$WS_EX_CLIENTEDGE) \$Icon2 = GUICtrlCreateIcon("D:\Temp\test1.ico", -1,_ 88, 16, 48, 48, -1, \$WS_EX_STATICEDGE) GUISetState(@SW_SHOW, \$Form)</pre>



### 3.7.3 图片框、图标框控件应用举例

对于 Label、Input 等控件而言，我们使用 GUICtrlSetData 改变其文本数据，对于 CheckBox、Radio 等控件而言，我们使用 GUICtrlSetState 改变其状态，那么对于 Pic、Icon 等控件，我们应该怎样改变他们的图像、图标？

使用 GUICtrlSetImage 就可以了：

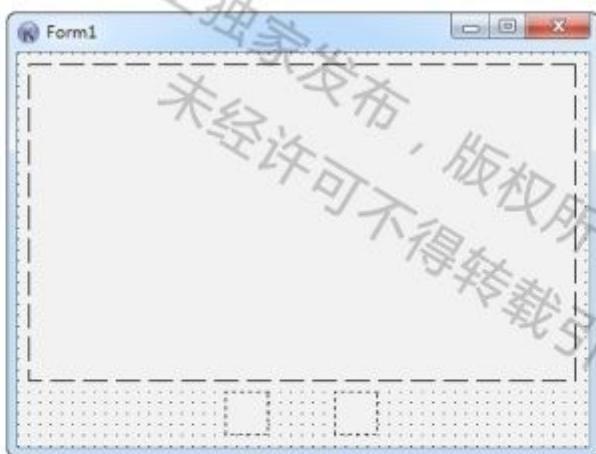
函数	说明（简单语法）
GUICtrlSetImage 设置控件的图像	<p>语法： GUICtrlSetImage(控件 ID, 图像/图标文件[, 图标序号])</p> <p>参数： 控件 ID：控件创建函数所返回的 ID 图像/图标文件：图像/图标文件的相对路径或绝对路径 图标序号：如果参数 2 是图标文件且包含多个图标，需设置图标编号</p> <p>返回值： 成功：返回 1 失败：返回 0</p>

#### 制作一个简单的图片查看器 GUI

##### GUI 设计分析：

图片查看器，需要一个图像控件来展示图片，并需要“上一张”和“下一张”两个按钮来切换图片，而为了按钮更漂亮，我们用图标来代替按钮（图标控件是可以响应鼠标单击的）。

其实一般而言，图片查看器是可以浏览路径以选择图片的，但为了便于初学者学习，暂时先预置 5 张图片，不给初学者带来太多学习负担。



(图 3-38)

代码：

```
#include <GUIConstantsEx.au3>

AutoItSetOption("MustDeclareVars", 1)

Global $gu_Form1 = GUICreate("图片查看器", 420, 290, -1, -1)
Global $gu_Pic1 = GUICtrlCreatePic("", 8, 8, 401, 233)
Global $gu_Icon1 = GUICtrlCreateIcon(@ScriptDir & '\img\left.ico', -1, 152, 248, 32, 32)
Global $gu_Icon2 = GUICtrlCreateIcon(@ScriptDir & '\img\right.ico', -1, 232, 248, 32, 32)

_Main()
Exit

Func _Main()

;图片索引(因为我们只是用了5张图做例子,所以索引的范围为1~5)
Local $Index = 1

;创建图片数组(数组元素存储图片的路径)
Local $a_PicList = _CreatePictureList()
;将图像控件内设置为第1张图片
GUICtrlSetImage($gu_Pic1, $a_PicList[$Index])

GUISetState(@SW_SHOW)

Local $nMsg
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit
        Case $gu_Icon1
```

```

If $Index = 1 Then
    ;如果索引为 1，则已是第一张图片
    MsgBox(0, '信息', '已经是第一张图了')
Else
    ;如果索引不为 1，则索引自减 1
    ;自减后为上一张图的索引
    $Index -= 1
    ;清空图像控件内的图片
    GUICtrlSetImage($gu_Pic1, "")
    ;将图像控件内图片设置为上一张图
    GUICtrlSetImage($gu_Pic1, $a_PicList[$Index])
EndIf

Case $gu_Icon2
If $Index = UBound($a_PicList, 1) - 1 Then
    ;如果索引为图片数组元素索引的最大值，则已是最后一张图片
    MsgBox(0, '信息', '已经是最后一张图了')
Else
    ;如果索引不为图片数组元素索引的最大值，则索引自增 1
    ;自增后为下一张图的索引
    $Index += 1
    ;清空图像控件内的图片
    GUICtrlSetImage($gu_Pic1, "")
    ;将图像控件内图片设置为下一张图
    GUICtrlSetImage($gu_Pic1, $a_PicList[$Index])
EndIf
EndSwitch
WEnd

EndFunc  ;==>_Main

Func _CreatePictureList()
    ;图片存储路径为当前目录下 img 目录
    Local $PicDir = @ScriptDir & '\img'
    ;创建图片数组，创建时只包含图片文件名
    Local $a_PicList[] = [
        '1.jpg',
        '2.jpg',
        '3.jpg',
        '4.jpg',
        '5.jpg'
    ]
    ;通过循环将数组内的每个文件名改为文件的绝对路径
    Local $i
    For $i = 1 To UBound($a_PicList, 1) - 1
        $a_PicList[$i] = $PicDir & '\' & $a_PicList[$i]
    Next

```

```
;返回数组  
Return $a_PicList  
EndFunc ;==>_CreatePictureList
```

运行图：



( 图 3-39 )

## 3.8 进度条 ( Progress )

### 3.8.1 进度条控件

#### 1、GUICtrlCreateProgress

进度条 ( Progress ) 控件，一般用于显示任务进度状况。当一个任务的执行需要较多步骤和较长时时间时，进度条能让用户更加明确任务的实际执行状况，以不至于误认为程序假死。

进度条控件一般由 GUICtrlCreateProgress 函数进行创建。

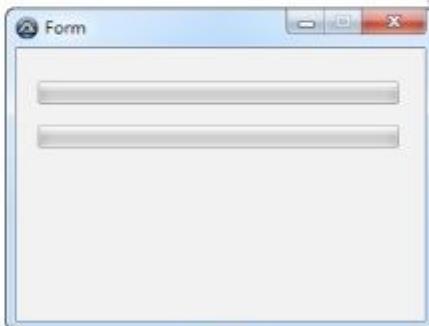
函数	说明
GUICtrlCreateProgress 创建进度条控件	<p>语法： GUICtrlCreateProgress(左侧, 顶部[, 宽度[, 高度[, 样式[, 扩展样式]]]])</p> <p>参数：</p> <ul style="list-style-type: none"><li>左侧：控件左侧距离窗体左侧的距离（单位：像素）</li><li>顶部：控件顶部距离窗体顶部的距离（单位：像素）</li><li>宽度：【可选参数】控件的宽（单位：像素） (不指定时根据文本内容自动调整)</li><li>高度：【可选参数】控件的高（单位：像素） (不指定时根据文本内容自动调整)</li><li>样式：标签的主要显示样式（下文详解），默认值为-1</li><li>扩展样式：标签的其他显示样式（下文详解），默认值为-1</li></ul> <p>返回值：</p>

	成功：返回控件 ID 失败：返回 0
--	-----------------------

在窗体中创建进度条：

```
$Form = GUICreate("Form", 300, 200, -1, -1)
$Progress1 = GUICtrlCreateProgress(16, 24, 265, 17)
$Progress2 = GUICtrlCreateProgress(16, 56, 265, 17)
GUISetState(@SW_SHOW, $Form)
```

图例：



( 图 3-40 )

值得一提的是，虽然 Progress 中的数据并非文本，但同样可以使用 GUICtrlSetData 为其设置数据。Progress 控件的数据是百分比，使用 GUICtrlSetData 为其设置百分比时，设置的值等同于百分比值（如设置值为“50”则进度条展示进度为“50%”）。

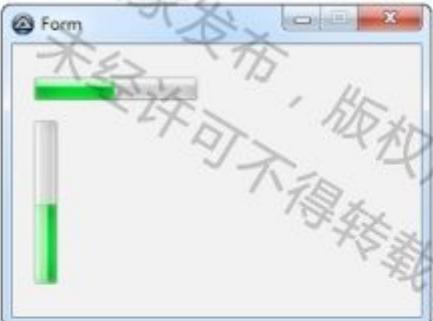
## 2、样式

Progress 的样式种类不多，这里介绍最常用的几种，更多样式请参见帮助文档。

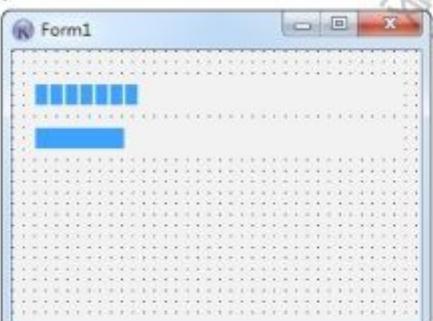
Progress 的样式是一系列的十六进制值，为了便于记忆，Au3 中将这些值转化为了常量，保存于 ProgressConstants.au3 文件中，使用时直接将其#include 下即可。

### ( 1 ) \$PBS\_VERTICAL

样式常量 ( 值 )	说明
\$PBS_VERTICAL ( 值 : 0x04 )	<p>样式： 进度条为垂直显示，而非默认的水平显示。</p> <p>代码：</p> <pre>#include &lt;ProgressConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Progress1 = GUICtrlCreateProgress(16, 24, 120, 17) \$Progress2 = GUICtrlCreateProgress(16, 56, 17, 120, _      \$PBS_VERTICAL) GUICtrlSetData(\$Progress1, 50) GUICtrlSetData(\$Progress2, 50) GUISetState(@SW_SHOW, \$Form)</pre>

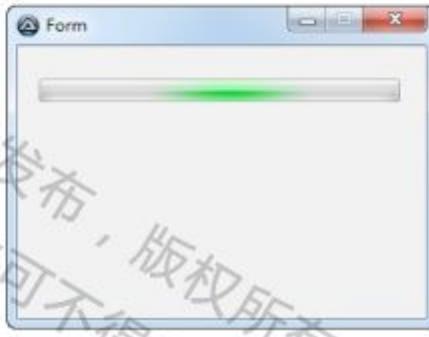
	<p>图例：</p> 
	<p>备注：</p> <p>需要注意的是，进度条垂直显示时，百分比是从底部到顶端显示进度的。</p>

### ( 2 ) \$PBS\_SMOOTH

样式常量 ( 值 )	说明
\$PBS_SMOOTH ( 值 : 0x01 )	<p>样式：</p> <p>使进度条平滑显示，而非默认的分段显示</p> <p>代码：</p> <pre>#include &lt;ProgressConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Progress1 = GUICtrlCreateProgress(16, 24, 265, 17) \$Progress2 = GUICtrlCreateProgress(16, 56, 265, 17,_ \$PBS_SMOOTH) GUICtrlSetData(\$Progress1, 50) GUICtrlSetData(\$Progress2, 50) GUISetState(@SW_SHOW, \$Form)</pre> <p>图例：</p> 
	<p>备注：</p> <p>在某些主题效果下，可能会出现即便没有设置平滑样式，也出现了平滑样式的情况（例如 Win7 默认主题）；同样可能会出现即便设置了平滑样式，但却没有平滑样式的情况（例如 WinXP 默认主题）。</p>

### ( 3 ) \$PBS\_MARQUEE

样式常量 ( 值 )	说明

\$PBS_MARQUEE ( 值 : 0x00000008 )	样式： 进度条显示为“跑马灯”效果，即并不显示真正的进度，而是一直在来回滚动。
	代码： <pre>#include &lt;ProgressConstants.au3&gt; \$Form = GUICreate("Form", 300, 200, -1, -1) \$Progress1 = GUICtrlCreateProgress(16, 24, 265, 17, \$PBS_MARQUEE) GUISetState(@SW_SHOW, \$Form)  For \$i=1 To 100     GUICtrlSetData(\$Progress1,\$i)     Sleep(50) Next</pre>
	图例： 
	备注： 范例代码中，即便为进度条以此设置 1~100 的值，进度条仍不显示实际进度，只是在反复滚动。其实这类滚动条一般用于进度结束时间不太明确的条件下，例如获取文件列表时。

### 3、扩展样式

Progress 控件没有太值得一用的扩展样式，有兴趣可以参见帮助文档，本文不再赘述。

#### 3.8.2 进度条控件应用举例

##### 制作一个简单的心理压力测试软件

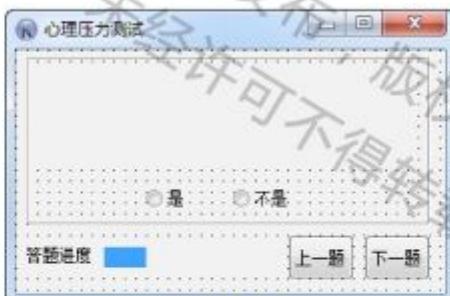
###### GUI 设计与分析

心理压力测试，即令用户对一系列的题目进行选择（本例我们只简单的选是/否），最后根据用户的整体选择得出分数，将分数对应不同的压力等级。

GUI 方面，我们需要一个 Label 控件去显示题目，需要两个 Radio 控件去分别代表是和否，需要一个 Progress 控件去显示答题进度，并需要两个 Button 控件分别用于选择上一题和下一题。

代码方面，我们创建一个二维数组，第一列用于记录用户答题结果，第二列为题目内容，这样，

将题目与答题结果对应起来。最后，遍历数组，根据用户的答题结果计算分数，从而得出压力状况并展示给用户。



(图 3-41)

代码：

```
#include <GUIConstantsEx.au3>
#include <ProgressConstants.au3>

AutoItSetOption("MustDeclareVars", 1)

Global $gu_Form1 = GUICreate("心理压力测试", 317, 180, -1, -1)

Global $gu_Group1 = GUICtrlCreateGroup("", 8, 0, 297, 129)
Global $gu_Label1 = GUICtrlCreateLabel("", 16, 16, 276, 65)
Global $gu_Radio1 = GUICtrlCreateRadio('是', 96, 96, 57, 25)
Global $gu_Radio2 = GUICtrlCreateRadio('不是', 160, 96, 57, 25)
GUICtrlCreateGroup(1, -99, -99, 1, 1)

Global $gu_Label2 = GUICtrlCreateLabel('答题进度', 8, 144 + 3, 52, 17)
Global $gu_Progress1 = GUICtrlCreateProgress(64, 144, 129, 17, $PBS_SMOOTH)
Global $gu_Button1 = GUICtrlCreateButton('上一题', 200, 136, 49, 33)
Global $gu_Button2 = GUICtrlCreateButton('下一题', 256, 136, 49, 33)

_Main()
Exit

Func _Main()

;问题索引，初始为 1
Local $Index = 1
;生成问题列表
Local $a_QuestionList = _CreateQuestionList()
;展示第 1 个问题
_DisplayQuestion($Index, $a_QuestionList)

GUISetState(@SW_SHOW)
```

```
;计分变量
Local $Score = 0

Local $nMsg
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit
        Case $gu_Button1
            ;上一题
            If $Index > 1 Then
                ;当不是第一道题时（非第一题才能查看前一道题）

                ;记录当前题的答案
                If GUICtrlRead($gu_Radio1) = $GUI_CHECKED Then
                    ;当选择“是”时，问题对应的0元素值为1
                    $a_QuestionList[$Index][0] = 1
                Else
                    ;当选择“否”时，问题对应的0元素值为0
                    $a_QuestionList[$Index][0] = 0
                EndIf

                ;索引自减1，即前一题
                $Index -= 1
                ;显示前一题
                _DisplayQuestion($Index, $a_QuestionList)
            EndIf
        Case $gu_Button2
            ;下一题
            If $Index = $a_QuestionList[0][0] Then
                ;当索引为最后一题时
                ;开始计算分数
                $Score = _CalcScore($a_QuestionList)
                ;根据分数显示不同的压力等级
                Select
                    Case $Score >= 0 And $Score <= 30
                        MsgBox(0, '压力测试', _
                            '心理压力值为' & $Score & ', 您的心理压力较小！')
                    Case $Score > 30 And $Score <= 60
                        MsgBox(0, '压力测试', _
                            '心理压力值为' & $Score & ', 您的心理压力一般！')
                    Case $Score > 60 And $Score <= 100
                        MsgBox(0, '压力测试', _
                            '心理压力值为' & $Score & ', 您的心理压力较大！')
                EndSelect
            EndIf
    EndSwitch
EndWhile
```

```
EndSelect
;分数显示完毕，退出
Exit
Else
;当索引不是最后一题时

;记录当前题的答案
If GUICtrlRead($gu_Radio1) = $GUI_CHECKED Then
;当选择“是”时，问题对应的0元素值为1
$a_QuestionList[$Index][0] = 1
Else
;当选择“否”时，问题对应的1元素值为0
$a_QuestionList[$Index][0] = 0
Endif

;索引自增1，即下一题
$Index += 1
;显示下一题
_DisplayQuestion($Index, $a_QuestionList)
EndIf
EndSwitch
WEnd

EndFunc ;==>_Main

;创建问题列表
Func _CreateQuestionList()
;创建一个二维数组
;第一列为问题的答案（是/否 1/0，默认为1）
;第二列为问题的内容
Local $a_QuestionList[0][2] = [[1, '_',
[1, '在与陌生人交流时，经常会手足无措，不知道应该要表达什么，或刻意寻找话题？'],
[1, '精神紧张时，头脑不清醒，几乎无法做出正确的判断？'],
[1, '经常为处境困难而感到沮丧，抱怨周围环境给自己带来的不公，头脑中充满了不满？'],
[1, '稍不如意就会怒气冲冲？'],
[1, '被他人指责时非常愤怒，不承认自己的错误，或认为他人的指责是无端的？'],
[1, '当别人请求帮助时，会不耐烦？'],
[1, '不能宽容他人？'],
[1, '被别人认为是挑剔的人，过于追求完美？'],
[1, '经常办错他人交办的事情？'],
[1, '当不愉快的事情缠身时，需要较长的时间才能解脱？']])
;使用二维数组的[0][0]元素记录二维数组的行数（即第一维的大小，亦即问题总数）
$a_QuestionList[0][0] = UBound($a_QuestionList, 1) - 1

Return $a_QuestionList
```

```

EndFunc  ;==>_CreateQuestionList

;显示问题
Func _DisplayQuestion($Index, $a_QuestionList)
    ;使用 Label1 显示问题
    GUICtrlSetData($gu_Label1, $Index & ' ' & $a_QuestionList[$Index][1])

    ;显示答题进度 (当前答题数/题目总数*100 )
    GUICtrlSetData($gu_Progress1, $Index / $a_QuestionList[0][0] * 100)

    ;根据答案设置 Radio 的选择
    If $a_QuestionList[$Index][0] = 0 Then
        ;答案为 0，即“不是”，则 Radio1 为不选中，Radio2 为选中
        GUICtrlSetState($gu_Radio1, $GUI_UNCHECKED)
        GUICtrlSetState($gu_Radio2, $GUI_CHECKED)
    Else
        ;答案为 1，即“是”，则 Radio1 为选中，Radio2 为不选中
        GUICtrlSetState($gu_Radio1, $GUI_CHECKED)
        GUICtrlSetState($gu_Radio2, $GUI_UNCHECKED)
    EndIf

    If $Index = 1 Then
        ;当是第一题时，隐藏“上一题”按钮
        GUICtrlSetState($gu_Button1, $GUI_HIDE)
    Else
        ;否则显示“上一题”按钮
        GUICtrlSetState($gu_Button1, $GUI_SHOW)
    EndIf

    If $Index = UBound($a_QuestionList, 1) - 1 Then
        ;当是最后一题时，将“下一题”按钮的文本改成“完成”
        GUICtrlSetData($gu_Button2, '完成')
    Else
        ;否则维持“下一题”文本
        GUICtrlSetData($gu_Button2, '下一题')
    EndIf
EndFunc  ;==>_DisplayQuestion

;分数计算函数
Func _CalcScore($a_QuestionList)
    ;将进度条提示文字改为“正在评分”
    GUICtrlSetData($gu_Label2, '正在评分')
    ;将进度条值归零
    GUICtrlSetData($gu_Progress1, 0)

```

```
;将单选按钮、按钮等都设置为不可用（防止计算分数时用户的误操作）
GUICtrlSetState($gu_Radio1, $GUI_DISABLE)
GUICtrlSetState($gu_Radio2, $GUI_DISABLE)
GUICtrlSetState($gu_Button1, $GUI_DISABLE)
GUICtrlSetState($gu_Button2, $GUI_DISABLE)

;积分变量归零
Local $Score = 0

;循环遍历题目数组
Local $i
For $i = 1 To $a_QuestionList[0][0]

    If $a_QuestionList[$i][0] = 1 Then
        ;当题目答案为1（是）时，计分+10
        $Score += 10
    EndIf

    ;每次计分后等待50毫秒，为的是让进度条递增更为明显
    ;此不是必要的代码行，只是为了显示进度效果，否则会因计算很快，进度条一闪而过
    Sleep(50)

    ;按照计分进度，更新进度条百分比
    GUICtrlSetData($gu_Progress1, $i / $a_QuestionList[0][0] * 100)
Next

Return $Score
EndFunc  ;==>_CalcScore
```

运行图例：



(图 3-42)



(图 3-43)



(图 3-44)

## 3.9 Level3 总结

Level3 作为对 GUI 的入门章节，目的是令大家对 GUI 设计有一个整体的感悟，并能创建和使用基本的 GUI。

3.1 节中，我们首先对于什么是 GUI、GUI 的基本设计观形成了整体观念。这个整体观念对于今后的 GUI 设计有着极为重要的指导意义，无论多么复杂丰富的 GUI，万变不离其宗的只有“怎样做好与用户的交互”。

3.2 节中，我们快速的对 GUI 设计形成了入门。本着循序渐进的教学观念，遵循螺旋上升的学习方式，本节（乃至本章）并没有一下子将太多的相关知识一股脑的推给大家，而是先引导大家学习最为核心的部分，理解最为重要的框架和方法。学习 GUI 要先理解重点知识，再逐步发散和完善。抓不住重点，一下吞的过多过猛，只会令学习事倍功半。

3.3 节中，我们学习了 GUI 设计辅助工具：KODA。KODA 设计器令 GUI 设计更为直观化，便于调整各控件，并可以快速的查看 GUI 效果。KODA 设计器的设计窗口，利于设计者快速调整控件属性、样式等，并方便设计者在多个窗体、控件间切换。但仍要记住的是，KODA 再便捷，也只是 GUI 设计辅助工具，不要依赖 KODA，因为最终控制 GUI 的还是代码，代码才是最终的王道。

3.4 节中，我们学习了窗体相关知识。窗体是控件的容器，是 GUI 设计的基本平台。掌握窗体的创建，理解窗体的含义，并熟悉窗体的各种样式、扩展样式，可令 GUI 设计更为灵活多变。

3.5~3.8 节，我们开始更为详细的了解各类控件，当然，本章作为入门只讲述的几个基本控件，它们是：Label（标签）、Input（文本框）、Button（按钮）、CheckBox（复选框）、Radio（单选框）、Group（组）、Pic（图像）、Icon（图标）、Progress（进度条）。我们详细的了解了每个控件的创建方法，并了解了它们的各种样式和扩展样式。对于控件而言，关键点是掌握在什么条件下使用什么控

件，这个在每节最初都有描述，不要不顾功能的乱使用控件，更不要过多的追求控件样式而忽略了功能才是重点。

Level3 整章是以窗体、控件为索引进行描述的，有些重点方法和函数零散分布在各节中。为了便于大家归纳，这里帮大家总结一个列表：

消息循环模式	3.2.3 小节，第 1 部分	这个是当前阶段 GUI 学习的难点和重点，不易理解，但作用影响广泛，请深刻理解，杜绝生搬硬套。
GUICreate	3.2.1 小节，第 2 部分 3.4 节，第 1 部分	窗体创建函数
GUIDelete	3.4 节，第 4 部分	窗体删除函数，对包含多个单窗体切换、多窗体等有着重要意义，不要忽视
GUICtrlRead	3.2.3 小节，第 2 部分	读取控件数据
GUICtrlSetData	3.2.3 小节，第 2 部分	设置控件数据
GUICtrlSetState	3.6.4 小节	设置控件状态
GUICtrlCreateLabel	3.5.1 小节	创建标签控件
GUICtrlCreateInput	3.5.2 小节	创建文本框控件
GUICtrlCreateButton	3.5.3 小节	创建按钮控件
GUICtrlCreateCheckbox	3.6.1 小节	创建复选框控件
GUICtrlCreateRadio	3.6.2 小节	创建单选框控件
GUICtrlCreateGroup	3.6.3 小节	创建组控件
GUICtrlCreatePic	3.7.1 小节	创建图像控件
GUICtrlCreateIcon	3.7.2 小节	创建图标控件
GUICtrlCreateProgress	3.8.1 小节	创建进度条控件

好了，至此我们顺利完成了 Level3 的学习，对于 GUI，您入门了吗？

## Level 4

## 4.1 输入框与消息框

在 Level1 中，我们首次学习了 InputBox 与 MsgBox 函数，用于基本输入与输出。由于当时大家是首次接触编程，为了减少学习负担，只介绍了这两个函数的简单语法，而本节中，我们将学习它们的详细语法。

### 4.1.1 输入框 (InputBox)

#### 1、语法

函数	说明
InputBox 输入框	<p>语法：</p> <pre>InputBox ( "标题", "提示" [, "默认" [, "密码" [, 宽度 = -1 [, 高度 = -1 [, 左距 = Default, [ 顶距 = Default [, 超时 = 0 [, 句柄]]]]]]])</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>标题：输入框的标题文本</li> <li>提示：用于“提示用户输入什么样的数据”的提示文本</li> <li>默认：[可选]数据框的默认数据</li> <li>密码：[可选]以特定字符替代输入的字符，输入密文时使用</li> <li>宽度：[可选]输入框的宽度，默认值约为 250 (像素)，最小值为 190 (像素)</li> <li>高度：[可选]输入框的高度，默认值约为 190 (像素)，最小值为 115 (像素)</li> <li>左距：[可选]输入框左侧与显示区域左侧的距离，默认居中</li> <li>顶距：[可选]输入框顶部与显示区域顶部的距离，默认居中</li> <li>超时：[可选]输入框的等待时间 (秒)，超时后自动关闭，默认为 0 (永久等待)</li> <li>句柄：[可选]父窗体句柄，用于指定此输入框属于哪个窗体，无父窗体则无需设置</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回输入的字符串 (最大返回 254 个字符)</li> <li>失败：返回空字符串，并将@error 设为非 0 值</li> </ul> <p>@error 值：</p> <ul style="list-style-type: none"> <li>1：用户单击了“取消”按钮</li> <li>2：已超时</li> <li>3：输入框显示失败 (通常由于参数设置错误造成)</li> <li>4：输入框无法显示在任何屏幕上</li> <li>5：“宽度”、“高度”、“左距”或“顶距”参数设置有误</li> </ul> <p>备注：</p> <ul style="list-style-type: none"> <li>(1) 如果输入的内容带有回车等换行符，则返回值为第一个换行符之前的内容；</li> <li>(2) 一般而言密码字符只有 1 个字符，如果是多个字符，首字符仍代表密码字符，之后字符中的数字代表限制密码输入的位数，之后的字符中有“M”则表示必须输入字符后才可单击“确定”按钮。</li> </ul>

#### 2、范例

## 例 1：一般方式

```
Local $r

;最简方式
$r = InputBox("InputBox - 最简方式", "请输入数据:")
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)

;有默认数据，默认数据为 abc123
$r = InputBox("InputBox - 有默认数据", "请输入数据:", "abc123")
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)
```

这是 InputBox 最常见、最简单的两种使用方式。



(图 4-1)

## 例 2：密码相关

```
Local $psw

;将输入的数据以 "*" 进行显示
$psw = InputBox("InputBox - 密码", "请输入数据:", "*", "*")
MsgBox(0, '输出', '刚才输入的密码是: ' & $psw)

;将输入的数据以 "*" 进行显示，并限制最长只能输入 6 位
$psw = InputBox("InputBox - 密码(限定位数)", "请输入数据:", "", "*6")
MsgBox(0, '输出', '刚才输入的密码是: ' & $psw)

;将输入的数据以 "*" 进行显示，且必须输入字符才能单击确定按钮
$psw = InputBox("InputBox - 密码(必须输入)", "请输入数据:", "", "+M")
MsgBox(0, '输出', '刚才输入的密码是: ' & $psw)

;将输入的数据以 "*" 进行显示，并限制最长只能输入 8 位，且必须输入字符才能单击确定按钮
$psw = InputBox("InputBox - 密码(限定+必须)", "请输入数据:", "", "+8M")
MsgBox(0, '输出', '刚才输入的密码是: ' & $psw)
```

密码字符样式：



(图 4-2)

说明：

- (1) 密码字符可以是各种字符，通常使用“\*”，当然大家也可以试试其他字符；
- (2) 数值用于限定可输入的最大字符数，M 用于限定必须输入字符，而数值和 M 的顺序是没有先后的，写成“\*8M”和“\*M8”没有区别；
- (3) 如果只想限定输入，而不想要以密码形式显示输入的字符，可以变通处理。将密码字符设置为空格，然后写限定值就可以了，例如“8M”表示限定输入 8 字符且必须输入（注意 8 前有个空格），同时并不以任何字符替代原字符（输入字符明文显示）。

#### 例 3：更改宽度、高度、左距、顶距

```
Local $r  
  
;将 InputBox 的宽改为 300 (像素)，高改为 150 (像素)  
$r = InputBox("InputBox - 更改宽高", "请输入数据:", "", "", 300, 150)  
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)  
  
;将 InputBox 左侧与显示区域左侧距离设置为 50 (像素)，并将 InputBox 顶部与显示区域顶部距离设置为  
30 (像素)  
$r = InputBox("InputBox - 更改左顶", "请输入数据:", "", "", -1, -1, 50, 30)  
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)  
  
;综合上述二者，宽 300 (像素)，高 150 (像素)，左距 50 (像素)，顶距 30 (像素)  
$r = InputBox("InputBox - 更改宽高左顶", "请输入数据:", "", "", 300, 150, 50, 30)  
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)
```

一般维持默认即可，实际应用中并不经常改变 InputBox 的大小或位置。



(图 4-3)

#### 例 4：超时

```
Local $r
```

```
;如果用户 5 秒内没有单击确定，则自动关闭输入框
$r = InputBox("InputBox - 5 秒超时", "请输入数据:", "", "", -1, -1, Default, Default, 5)
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)
```

说明：

(1) 未设置的参数，请根据语法写入默认值，但并不是所有默认值都是空字符串、0 或 -1，如上述例子中“顶距”和“左距”的默认值就是“Default”；

(2) 值得注意的是，超时后 InputBox 会自动关闭，而非自动单击“确定”按钮，所以超时后的返回值并非输入的值，而是空字符串。另一层意义上讲，只有单击“确定”按钮 InputBox 才会返回输入的值。



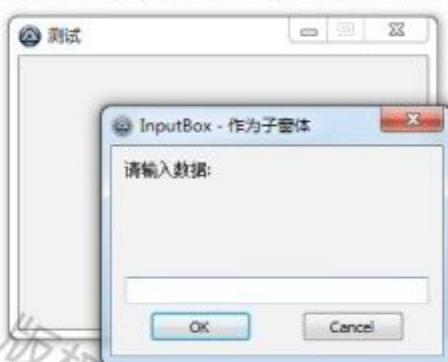
(图 4-4)

#### 例 5：作为子窗体

```
;创建窗体 ($Form) 并显示
Local $Form = GUICreate("测试", 300, 200, -1, -1)
GUISetState(@SW_SHOW, $Form)

;使用 InputBox，并将父窗体设置为$Form
Local $r = InputBox("InputBox - 作为子窗体", "请输入数据:", "", "", -1, -1, Default, Default, 0, $Form)
MsgBox(0, '输出', '刚才输入的数据是: ' & $r)
```

当 InputBox 作为某窗体的子窗体时，InputBox 弹出时其父窗体将不能被操作。



(图 4-5)

## 4.1.2 消息框 ( MsgBox )

## 1、语法

函数	说明
MsgBox 消息框	<p>语法： MsgBox ( 标志, "标题", "文本" [, 超时 = 0 [, 句柄]])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>标志：用于指定消息框按钮类型、图标、默认按钮、样式等（下文详解）</li> <li>标题：消息框的标题文本</li> <li>文本：消息框所显示的消息文本</li> <li>超时：[可选]等待多少时间（秒）后，自动关闭消息框，默认为 0（永久等待）</li> <li>句柄：[可选]父窗体句柄，用于指定此消息框属于哪个窗体。无父窗体则无需设置</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：返回被按下按钮的 ID（按钮 ID 下文详解）</li> <li>失败：返回\$IDTIMEOUT (-1)，表示消息框超时自动关闭</li> </ul> <p>备注：</p> <ul style="list-style-type: none"> <li>(1) 标志，见下文</li> <li>(2) 按钮 ID，见下文</li> </ul>

## 2、“标志”相关解释

MsgBox 的第一个参数“标志”，是一个复合型的值，标志类型比较多，这里介绍一些常用值，其余生僻值的还请参见帮助文档。

类型	常量	值	功能
按钮	\$MB_OK	0	“确定”
	\$MB_OKCANCEL	1	“确定” + “取消”
	\$MB_ABORTRETRYIGNORE	2	“终止” + “重试” + “忽略”
	\$MB_YESNOCANCEL	3	“是” + “否” + “取消”
	\$MB_YESNO	4	“是” + “否”
	\$MB_RETRYCANCEL	5	“重试” + “取消”
	\$MB_CANCELTRYCONTINUE	6	“取消” + “重试” + “继续”
图标		0	无图标
	\$MB_ICONERROR	16	错误图标
	\$MB_ICONQUESTION	32	问号图标
	\$MB_ICONWARNING	48	警告图标
	\$MB_ICONINFORMATION	64	信息图标
默认按钮	\$MB_DEFBUTTON1	0	第一按钮为默认按钮
	\$MB_DEFBUTTON2	256	第二按钮为默认按钮
	\$MB_DEFBUTTON3	512	第三按钮为默认按钮
特殊		0	无特殊
	\$MB_RIGHT	524288	标题、文本为右对齐
	\$MB_RTLREADING	1048576	从右至左方式显示信息
	\$MB_SETFOREGROUND	65536	多个窗体层叠时，使消息框在最前（最顶）
	\$MB_TOPMOST	262144	将消息框置于顶层，不会被其他窗体覆盖

说明：

(1) 如果使用常量表示标志，则必须包含 MsgBoxConstants.au3，而使用标志值时无需包含，例如：

```
#include <MsgBoxConstants.au3>
MsgBox($MB_OKCANCEL, "测试标题", "测试文本")
```

与

```
MsgBox(1, "测试标题", "测试文本")
```

二者的意义、效果是相同的。至于读者是喜欢记忆常量或是值，完全看个人喜好，英文常量便于记忆，数值则书写简便。

(2) 多个不同类型标志可以单独使用（如上例），也可以叠加使用（标志值相加即可），例如：

```
#include <MsgBoxConstants.au3>
MsgBox($MB_OKCANCEL + $MB_ICONINFORMATION, "测试标题", "测试文本")
```

或

```
MsgBox(1 + 64, "测试标题", "测试文本")
```

(3) \$MB\_SETFOREGROUND，使消息框位于最前，意思是说，当产生消息框的程序不是当前多个窗体最前的那个时，将消息框置于最前，以便于用户看到消息框，以免被其他窗体遮挡。但这个“使最前”只是“一时的”，而不是“永久的”，当激活其他窗体时，其他窗体可以遮挡这个消息框。

(4) 而\$MB\_TOPMOST，相当于给予消息框一个“永久最顶（最前）”的属性，其他窗体（哪怕是被激活的）无法遮挡这个消息框，直到这个消息框的按钮被点击、超时、或关闭。

### 3、“按钮 ID”相关解释

当用户点击 MsgBox 中的一个按钮时，MsgBox 函数会返回按钮的 ID。不同按钮分别对应不同的 ID 值，这样程序即可得知用户按下了哪个按钮。按钮 ID 如下表

按钮	按钮 ID（常量）	值
确定	\$IDOK	1
取消	\$IDCANCEL	2
终止	\$IDABORT	3
重试	\$IDRETRY	4
忽略	\$IDIGNORE	5
是	\$IDYES	6
否	\$IDNO	7
再重试	\$IDTRYAGAIN	10
继续	\$IDCONTINUE	11

说明：

(1) 如果使用常量表示按钮 ID，则必须包含 MsgBoxConstants.au3，而直接使用值时无需包含，例如：

```
#include <MsgBoxConstants.au3>

Local $r = MsgBox($MB_OKCANCEL, '按钮测试', '请单击“确定”或“取消”')
Switch $r
    Case $IDOK
        MsgBox($MB_OK, '按钮测试', '用户按下了“确定”按钮')
```

```
Case $IDCANCEL  
    MsgBox($MB_OK, '按钮测试', '用户按下了“取消”按钮')  
EndSwitch
```

与

```
Local $r = MsgBox(1, '按钮测试', '请单击“确定”或“取消”')  
Switch $r  
    Case 1  
        MsgBox(0, '按钮测试', '用户按下了“确定”按钮')  
    Case 2  
        MsgBox(0, '按钮测试', '用户按下了“取消”按钮')  
EndSwitch
```

二者的意义、效果是相同的。至于读者是喜欢记忆常量或是值，完全看个人喜好，英文常量便于记忆，数值则书写简便。

### (2) “重试”与“再重试”的区别：

重试 (\$IDRETRY)，对应按钮标志为\$MB\_RETRYCANCEL 时的“重试”按钮；

再重试 (\$IDTRYAGAIN)，对应按钮标志为\$MB\_CANCELTRYCONTINUE 时的“重试”按钮。

造成这个问题的原因，可能是由于中英文系统在翻译时的偏差造成的，大家知道这是两个不同的按钮就可以了。

## 4、范例

### 例 1：用户选择不同按钮

```
#include <MsgBoxConstants.au3>  
  
;对不同按钮的响应  
Local $r = MsgBox($MB_OKCANCEL, ' MsgBox - 不同按钮响应', '请单击“确定”或“取消”')  
Switch $r  
    Case $IDOK ;返回值为“确定”按钮的 ID，则用户按下了“确定”按钮  
        MsgBox($MB_OK, ' MsgBox - 不同按钮响应', '用户按下了“确定”按钮')  
    Case $IDCANCEL ;返回值为“取消”按钮的 ID，则用户按下了“取消”按钮  
        MsgBox($MB_OK, ' MsgBox - 不同按钮响应', '用户按下了“取消”按钮')  
EndSwitch
```

MsgBox 的返回值为按钮 ID，通过 Switch...Case...EndSwitch 多分支语句对返回值进行判定，即可得知用户按下了哪个按钮。



(图 4-6)

**例 2：不同图标用于不同提醒**

```
#include <MsgBoxConstants.au3>

;不同图标用于不同类的提示
;1 - 错误提示
MsgBox($MB_OK + $MB_ICONERROR, _
        ' MsgBox - 错误', _
        'XX 数据发生错误，无法继续执行！')

;2 - 警告提示
MsgBox($MB_OKCANCEL + $MB_ICONWARNING, _
        ' MsgBox - 警告', _
        '发现 XX 数据异常，确定要继续吗？')
```

图标的用途是直观的使用户了解到这是一个怎样类型的提醒，图标分类与消息类型对应：

- “错误”：一般用于提示当前程序出现了某些运行故障，需要终止或退出
- “问号”：一般用于询问用户以做出某些选择，或以帮助的方式描述某些功能的含义
- “警告”：一般用于警告用户当前的操作可能带有某些风险
- “信息”：一般用于友好的提示用户当前程序的运行状况信息



(图 4-7)



(图 4-8)

**例 3：更改默认按钮，引导用户倾向性**

```
#include <MsgBoxConstants.au3>

;更改默认按钮，引导用户倾向性
MsgBox($MB_OKCANCEL + $MB_ICONWARNING + $MB_DEFBUTTON2, _
        ' MsgBox - 警告', _
        '发现 XX 数据异常，确定要继续吗？')
```

一般而言，`MsgBox`的第一个按钮为默认按钮，但有些时候如果更倾向于用户单击其他按钮，则建议更改默认按钮。如上例是发出了一个警告，出现警告时更希望用户单击取消按钮，所以默认按钮为第二按钮。



(图 4-9)

另一方面讲，可以通过回车、空格等直接完成对默认按钮的点击，所以更改默认按钮有时也是为了用户操作更加便捷。

### 例 4：超时

```
#include <MsgBoxConstants.au3>

;超时
Local $r = MsgBox($MB_OKCANCEL, 'MsgBox - 5 秒超时', '请单击"确定"或"取消"', 5)
Switch $r
    Case $IDOK ;返回值为“确定”按钮的 ID，则用户按下了“确定”按钮
        MsgBox($MB_OK, 'MsgBox - 5 秒超时', '用户按下了“确定”按钮')
    Case $IDCANCEL ;返回值为“取消”按钮的 ID，则用户按下了“取消”按钮
        MsgBox($MB_OK, 'MsgBox - 5 秒超时', '用户按下了“取消”按钮')
    Case $IDTIMEOUT ;超时，用户未作选择
        MsgBox($MB_OK, 'MsgBox - 5 秒超时', '超时，用户未作选择')
EndSwitch
```

与 InputBox 相同，MsgBox 也有超时自动关闭的功能。MsgBox 超时自动关闭，不等同于按下默认按钮，超时自动关闭后 MsgBox 的返回值为 \$IDTIMEOUT ( 值为 -1 )。可通过对返回值的判定得知是用户按下了哪个按钮或是超时，从而针对不同情况作出不同的程序执行。



( 图 4-10 )

### 例 5：作为子窗体

```
#include <MsgBoxConstants.au3>

;创建窗体（$Form）并显示
Local $Form = GUICreate("测试", 300, 200, -1, -1)
GUICreate(@SW_SHOW, $Form)

;使用 InputBox，并将父窗体设置为$Form
MsgBox($MB_OK + $MB_ICONINFORMATION, "MsgBox - 子窗体", "测试数据", 0, $Form)
```

当 MsgBox 作为某窗体的子窗体时，MsgBox 弹出时其父窗体将不能被操作。



(图 4-11)

## 4.2 运行外部程序

### 4.2.1 运行外部程序 ( Run、RunWait )

有时候，我们需要调用外部程序运行以完成某些特定工作。例如运行软件安装包完成软件的自动安装、运行 7za.exe 完成对文件的压缩或解压缩等。调用外部程序，由函数 Run 与 RunWait 来完成，此二者在日常编程中有着很高的使用率。

#### 1、语法

在 Run 与 RunWait 的语法中，涉及复杂概念的部分（如输入输出流、子主进程互动）暂不介绍，这些知识将在后续章节中学习了其他知识后再逐步学习，就不在这里对大家形成学习负担了。学以致用才是我们秉承的学习理念，先会用，后完善。

函数	说明
Run 调用外部程序运行，不等待程序运行完成，即执行下一条语句。	语法： <code>Run ( "程序" [, "工作目录" [, 显示 ]])</code> 参数： 程序：所运行程序的相对路径或绝对路径 工作目录：[可选]所运行程序的工作目录（不设置可留空） 显示：[可选]程序运行时的显示状态 <code>@SW_HIDE</code> ：隐藏窗口 <code>@SW_MINIMIZE</code> ：最小化窗口 <code>@SW_MAXIMIZE</code> ：最大化窗口 返回值： 成功：返回所运行程序的进程标识符（PID） 失败：返回 0，且@error 被设置为非 0 值
RunWait 调用外部程序运行，等待程序运行完成。	语法： <code>RunWait ( "程序" [, "工作目录" [, 显示 ]])</code>

行，等待直至程序运行完成后，才执行下一条语句。	<p>参数：</p> <p>程序：所运行程序的相对路径或绝对路径 工作目录：[可选]所运行程序的工作目录（不设置可留空） 显示：[可选]程序运行时的显示状态     @SW_HIDE：隐藏窗口     @SW_MINIMIZE：最小化窗口     @SW_MAXIMIZE：最大化窗口</p> <p>返回值：</p> <p>成功：返回所运行程序的退出代码 失败：返回 0，且@error 被设置为非 0 值</p>
-------------------------	---

从语法上看，Run 与 RunWait 没有太大区别，甚至连参数含义都基本一样。主要的不同有两点，还请注意：

(1) Run 在调用外部程序运行后，并不等待外部程序的运行结束，就继续执行之后的程序语句；而 RunWait 在调用外部程序运行后，将等待外部程序运行结束后，才继续执行之后的程序语句。这是二者最大的不同。

(2) 在调用外部程序运行成功的条件下，Run 的返回值是所调用程序的进程标识符（PID），而 RunWait 的返回值是所调用程序的退出代码。

其中，PID 是系统用于标注进程的 ID，打开 Windows 任务管理器，选中进程选项卡，菜单栏->查看->选择列，勾选 PID，即可看到当前系统每个进程的 PID。关于 PID 的应用后续章节讲解进程类的内容时会更为深入的学习，这里不再赘述。

而程序的退出代码，是指程序在退出时的一个返回值。当程序正常执行、非正常执行，或遇到运行环境影响等时，退出代码可能是不同的，但至于返回什么值就要看程序开发者如何指定了。Au3 中可以使用“Exit <值>”的方式来指定程序的退出代码（如“Exit 1”）。程序的退出代码可以类比函数的返回值，用于将程序的运行信息反馈给主调用程序。例如调用外部程序进行压缩时，退出代码为 1 则压缩成功、为 0 则压缩失败，这样主调用程序就可以以此为判定标志，通知用户压缩操作是成功了或是失败了。

## 2、范例

### 例 1：“不等待运行”与“等待运行”

以 Run 方式运行任务管理器（如果任务管理器已经在运行了，请关闭）：

```
Run('C:\Windows\System32\TaskMgr.exe')
MsgBox(0, 'Run 测试', '任务管理器没有退出，就已经执行 MsgBox 语句了')
```

运行上述代码，大家会发现，当任务管理器被运行后，立刻就执行了之后的 MsgBox 语句。这符合 Run 函数的“只运行、不等待”特性。

以 RunWait 方式运行任务管理器（如果任务管理器已经在运行了，请关闭）：

```
RunWait('C:\Windows\System32\TaskMgr.exe')
MsgBox(0, 'RunWait 测试', '任务管理器退出后，才执行 MsgBox 语句')
```

运行上述代码，大家会发现，当任务管理器被运行后，不会立刻执行 MsgBox 语句，脚本会进行等待，直到用户手动关闭了任务管理器，之后的 MsgBox 语句才会被执行，这符合 RunWait 函数的

“运行并等待”特性。

#### 例 2：运行后最大化，运行后最小化

当运行某程序后，需要程序 UI 最大化，可以指定显示参数为 @SW\_MAXIMIZE：

```
Run('C:\Windows\RegEdit.exe', "", @SW_MAXIMIZE)
```

相应的，当运行某程序后，需要程序 UI 最小化，可以指定显示参数为 @SW\_MINIMIZE：

```
Run('C:\Windows\RegEdit.exe', "", @SW_MINIMIZE)
```

RunWait 在此参数方面与 Run 相同，就不再重复举例了。

需要注意的是，如果某些程序不支持最大化，最大化将会失败。

#### 例 3：隐藏运行

如果希望隐藏运行某些程序，可以将显示参数设置为 @SW\_HIDE。

为了测试这个参数，我们首先编写一个简单的批处理 test.cmd，并将其放置在 D 盘根目录下，批处理内容为：

```
md d:\xxx
```

意思是创建 d:\xxx 文件夹（目录）。

而后编写 Au3 代码隐藏调用这个批处理：

```
RunWait('d:\test.cmd', "", @SW_HIDE)
```

运行后，我们发现 d:\xxx 目录已被正常创建，而我们并没有看到批处理的“黑框”。

注意，如果被隐藏的程序出现运行停止、运行错误、运行未响应等问题，由于是隐藏运行所以不会被发觉且不能被直接操作，所以在以隐藏方式运行程序时一定要先行确定被调用程序的稳定性，不要乱用隐藏方式，以免埋下隐患。

### 3、“绝对路径”、“相对路径”的相关解释

#### (1) 绝对路径

“绝对路径”是文件或文件夹在整个文件系统中的绝对位置，例如上例中注册表编辑器的路径 “C:\Windows\RegEdit.exe”，就是一种绝对路径写法。另一种层面上理解，可以将绝对目录理解为“完整路径”。

绝对路径可以更为完整的表达一个文件的所在位置，在被读写、被调用时不易出现错误，一般建议使用绝对路径。而对于某些可能存在变化的路径，Au3 中都有对应的目录宏来表示，例如当前目录下的 test.txt 可以表示为 “@ScriptDir & "\test.txt"”，Windows 目录下的注册表编辑器可以表示为 “@WindowsDir & '\RegEdit.exe'”，这使得绝对路径也可以相当灵活。

#### (2) 相对路径

“相对路径”是文件相对于当前程序工作目录的路径，其中，可以使用 “.” 表示当前路径，“..” 表示上层路径，使用当前路径时也可以省略 “.”。例如当前脚本路径为 “d:\xxx\test.au3”，则 “.\t1.txt” 或 “t1.txt” 表示 “d:\xxx\demo.txt”，而 “..\t2.txt” 表示 “d:\t2.txt”。

相对路径有着很高的灵活性，可以简洁的表达文件相对于当前文件的位置信息，这一点大家会在今后书写程序时逐步体会出来。但相对于使用绝对路径，使用相对路径的出错率会稍高（特别对于新手），而且会因为工作目录的变更而产生一些连带问题。所以，推荐大家更多的使用绝对路径。

## 4、工作目录

“目录”，目录是文件夹的另外一种称呼，其实“目录”这个称呼比“文件夹”更早，目录源于Linux的目录结构，而文件夹更多见于Windows的相关描述。广义上讲二者的意义相同，不必过于纠结。

而“工作目录”概念与“相对路径”息息相关。首先要明确的是程序的“工作目录”并不等同于程序的“当前目录”，如果手动双击运行程序，则二者是相同的，但如果程序被调用，则二者可能是不同的，此时程序的工作目录为主调程序的工作目录。

举个例子来解释一下，我们来编写a、b、c三个程序。

目录结构为：

```
d:\xxx\aa.exe  
d:\xxx\test\bb.exe  
d:\xxx\test\cc.exe
```

程序代码为：

```
c.exe :  
    MsgBox(0, 'c', '我是 c')  
  
b.exe :  
    RunWait('.\c.exe')  
  
a.exe :  
    Run('.\test\bb.exe')
```

从目录结构和程序代码来看，c会产生一个消息框，b调用了当前目录下的c.exe，而a.exe调用了其目录下test目录中的b.exe。

直接运行b.exe，会正常调用c.exe，c.exe会弹出消息框。这一点是正确无误的。

由此推断，运行a.exe，将调用b.exe，而b.exe应当会调用c.exe，从而c.exe会弹出消息框，但实际运行结果，b.exe并没有成功调用其目录下的c.exe。这就奇怪了，明明直接运行b.exe会正常调用c.exe，那为什么用a.exe调用b.exe时，b.exe就没有成功调用c.exe呢？

这里有个疏漏，即“相对目录是相对于当前程序的工作目录的，而不是相对于当前程序的”。运行a.exe，则工作目录为“d:\xxx”，而b.exe被a.exe调用，则b.exe的工作目录与主调程序相同，同样为“d:\xxx”。然而b.exe中使用了相对目录，那么此时相对于“d:\xxx”而言“.\c.exe”的路径其实是“d:\xxx\c.exe”，而不是“d:\xxx\test\c.exe”，所以b.exe调用c.exe失败了！

要解决这个问题，需要为b.exe指定工作目录，将a.exe的代码改为：

```
Run('.\test\bb.exe', '\test')
```

即将b.exe的工作目录指定为b.exe的所在目录，此时再调用c.exe就正常了。

a、b、c的这个例子，是相对目录造成调用问题的一个经典例子，很多新手为了方便喜欢写相对目录，殊不知相对目录里的“坑”相当多，所以如果不是对工作目录理解很透彻，或不希望因为相对目录出现不必要的问题，还是请使用绝对目录更加稳妥。

## 5、包含变量路径

很多时候，我们需要使用变量来表示路径的一部分，这在日常程序书写中非常常见。但很多朋友

在书写此类代码时遇到困难，这里我们就来专门探讨一下。

首先我们来回顾一下字符串的概念，对于字符串而言，引号内的部分即为字符串的内容，所以即便有类似变量的写法出现在引号内，那它也是字符串的一部分，而不能被称作变量。例如：

```
Local $t = 'abc'
Local $s = '$t123'
MsgBox(0, "", $s)
```

\$s 的值就是 “\$t123”，而绝非 “abc123”。只要被引号包含，无论\$t 再怎么像个变量，它也就是字符串的一部分，而绝非变量。那么如何让\$s 的值为 “abc123” 呢？需要这样书写：

```
Local $t = 'abc'
Local $s = $t & '123'
MsgBox(0, "", $s)
```

这时\$s 的值就为 “abc123” 了。

这里我们要明确的是，变量不可写在字符串内，而需要以连接运算符与字符串进行连接。有些朋友有将变量写在字符串内的习惯，可能是受了批处理书写规则的影响，这里一定要改正。

然后，我们言归正传。如果已定义变量\$WinDir，且赋值为 C:\Windows，那么怎样表达 C:\Windows\RegEdit.exe？写法如下：

```
Local $WinDir = 'C:\Windows'
Local $Path = $WinDir & '\RegEdit.exe'
MsgBox(0, "", $Path)
```

其实只要理解了，书写就简单了，不易出错。

除了变量以外，宏、常量表达路径与变量相同，例如对于上例而言，不是所有用户的系统都在 C 盘，所以使用@WindowsDir 宏来表示 Windows 目录更为稳妥。改写后如下：

```
Local $Path = @WindowsDir & '\RegEdit.exe'
MsgBox(0, "", $Path)
```

## 6、执行命令提示符 ( cmd ) 命令

命令提示符 ( Win+R，输入 cmd，回车)，是计算机维护人员常用的一种命令行环境。很多人将它与 DOS 搞混，甚至直接称其为 DOS，这是不正确的。DOS 是 16 位环境，而命令提示符则是 32 位环境，命令提示符虽然兼容绝大多数 DOS 命令，但这只是一种出于兼容和保持用户操作习惯的设计。所以必须明确，命令提示符绝不等同于 DOS。

命令提示符程序一般位于 C:\Windows\System32\cmd.exe，可以运行它，而后在黑框中输入命令，这也是一般的使用方法。另一种方法，可以通过使用 cmd.exe 的参数完成对命令的执行。例如 “cmd.exe /c md d:\xxx”，即使用 md 命令创建 d:\xxx 目录，并在完成后退出 cmd.exe ( cmd.exe 的/c 参数，意思是使用 cmd.exe 执行命令后退出 cmd.exe )。

像 “md” 这类命令还有很多，如 “rd、del、move、copy” 等，这些都是命令提示符的内置命令，不能被直接调用。但我们可以通过上一段中的方法间接实现。例如，使用 rd 命令删除 d:\xxx 目录：

```
RunWait('c:\windows\system32\cmd.exe /c rd /s /q d:\xxx', "", @SW_HIDE)
```

即调用 cmd.exe 执行 “rd /s /q d:\xxx” 命令，为了不显示黑框将显示参数设置为@SW\_HIDE。而为了使调用 cmd.exe 更为通用化，可以使用 Au3 的宏@ComSpec 替代 cmd.exe 的绝对路径。上

述代码可以修改为：

```
RunWait(@ComSpec & '/c rd /s /q d:\xxx', "", @SW_HIDE)
```

这样，我们就可以很方便的使用 Run 或 RunWait 来执行一些批处理命令了。

### 7、关于引号的问题

对于批处理而言，当某个路径存在空格时，需要使用双引号对路径进行包含。例如：

```
rd /s /q "d:\xxx yyy"
```

这个如果要使用 Au3 来表达，则书写为：

```
RunWait(@ComSpec & '/c rd /s /q "d:\xxx yyy"', "", @SW_HIDE)
```

注意上述代码字符串部分使用了单双引号混合的方式。这是个简单问题，但曾经不少学习者在这里遇到了问题。

这种路径带有空格的路径，在很多程序参数中很常见，例如调用 7za.exe 对特定目录进行压缩：

```
RunWait(@ScriptDir & '\7za.exe' & _  
        '-t7z -a "' & @ScriptDir & '\Test.7z' & '" "C:\Program Files\2345pic\" -r')
```

其中：

- 压缩文件路径可能存在空格，则使用双引号进行了包含；
- 源目录 “C:\Program Files\2345pic\” 中带有引号，同样使用双引号进行包含。

这样书写，才保证了 7za.exe 的正常执行。

不过对于 Run 和 RunWait 所直接调用程序的路径，其中即便带有空格，也是不需要使用双引号进行包含，毕竟 Run 和 RunWait 是 Au3 函数，与批处理运行的方式并不相同。例如启动 IE：

```
Run('C:\Program Files\Internet Explorer\iexplore.exe')
```

虽然 IE 的绝对路径中带有数个空格，但丝毫没影响 Run 的调用（RunWait 同理，不再赘述）。

### 4.2.2 关联打开 ( ShellExecute、ShellExecuteWait )

Run 与 RunWait 一般用于运行.exe、.cmd、.bat 等可执行程序，而如果我们想“打开”一个其他类型的文件，要怎么做？

在 Windows 操作系统中，根据后缀名的不同，系统自动将某个类型的文件与可以将其打开的应用程序关联起来，例如.txt 文档默认与记事本进行关联，当尝试打开一个.txt 文件时，记事本程序 notepad.exe 会自动启动并将其打开。后缀名与对应程序的关联，一般记录于注册表中。

下面我们要介绍的 ShellExecute 与 ShellExecuteWait 函数，会自动寻找所打开文件与其关联程序的对应关系，并将目标文件以其关联程序打开，是两个非常方便的函数。

#### 1、语法

函数	说明
ShellExecute 使用关联程序打开 目标文件，不等待 文件关闭，继续执行之后的其他代码	语法： <pre>ShellExecute ("文件" [, "参数" [, "工作目录" [, "打开方式" [, 显示]]]])</pre> 参数： 文件：将要打开的文件的相对路径或绝对路径 参数：[可选]所运行程序的参数

	<p>工作目录 : [可选]所运行程序的参数          打开方式 : [可选]指定打开方式              如不指定，则使用注册表内所规定的默认打开方式              open, 打开目标文件              edit, 编辑目标文件              print, 打印目标文件              properties, 显示目标文件/目录的属性          显示 : [可选]显示效果              @SW_HIDE : 隐藏窗口              @SW_MINIMIZE : 最小化窗口              @SW_MAXIMIZE : 最大化窗口</p>
	<p>返回值 :              成功 : 返回关联程序的进程标识符 ( PID )              失败 : 返回 0，并将 @error 设置为非 0</p>
ShellExecuteWait 使用关联程序打开目标文件，等待直至文件关闭，而后执行之后的其他代码	<p>语法 :  <code>ShellExecuteWait ( "文件" [, "参数" [, "工作目录" [, "打开方式" [, 显示]]]] )</code></p> <p>参数 :              文件 : 将要打开的文件的相对路径或绝对路径              参数 : [可选]所运行程序的参数              工作目录 : [可选]所运行程序的参数              打开方式 : [可选]指定打开方式                  如不指定，则使用注册表内所规定的默认打开方式                  open, 打开目标文件                  edit, 编辑目标文件                  print, 打印目标文件                  properties, 显示目标文件/目录的属性          显示 : [可选]显示效果                  @SW_HIDE : 隐藏窗口                  @SW_MINIMIZE : 最小化窗口                  @SW_MAXIMIZE : 最大化窗口</p>
	<p>返回值 :              成功 : 返回关联程序的退出代码              失败 : 返回 0，并将 @error 设置为非 0</p>

**说明**

( 1 ) 二者的函数形式、参数基本相同，区别有以下两点：

- ShellExecute 在打开目标文件后，并不等待文件的关闭，就继续执行之后的代码；而 ShellExecuteWait 会在打开目标文件后，等待目标文件关闭，才继续执行之后的代码。
- 在打开目标文件成功的前提下，ShellExecute 的返回值为关联程序的进程标识符，而 ShellExecuteWait 的返回值为关联程序的退出代码。

( 2 ) 其他说明：

- ShellExecute 与 ShellExecuteWait 的关系，与 Run 与 RunWait 的关系类似。
- 除非关联程序有特定的参数设定时，才需要指定参数。
- 工作目录的相关注意事项请参见 Run 或 RunWait。

· 一般无需指定“打开方式”，使用注册表内所关联的默认方式即可。

### 2、范例

打开纯文本文档

```
ShellExecute(@ScriptDir & "\test.txt")
```

运行后，自动使用记事本（notepad.exe）打开了 test.txt。

注意，程序同目录下 test.txt 必须存在，否则将报错。

## 4.3 驱动器

驱动器（Drive）是 Windows 操作系统中的重要概念，驱动器分为物理驱动器和逻辑驱动器两大类。物理驱动器一般指硬件设备，例如一个硬盘、一个光驱、一个 U 盘等；而逻辑驱动器一般指基于前者的逻辑划分，例如将一个硬盘划分为不同分区，每个逻辑分区即为一个逻辑驱动器，而像 U 盘、光盘通常只有一个逻辑分区，所以 U 盘、光盘整体被认为是一个逻辑驱动器。

Windows 中通常以盘符表示驱动器，这也是 Windows 操作系统划分文件系统的一个特色，盘符通常以英文字母表示，A 和 B 被预留给软驱，而软驱在现行计算机中已不属于常见设备，所以盘符一般是由 C 开始，依次向后排列的。

好，介绍完驱动器的相关知识，本节我们来学习与驱动器相关的函数。

### 4.3.1 获取所有驱动器

#### 1、获取驱动器列表（DriveGetDrive）

函数	说明
DriveGetDrive 获取驱动器列表	语法： <pre>DriveGetDrive ("类型")</pre>
	参数： 类型： <ul style="list-style-type: none"><li>ALL：全部驱动器</li><li>CDROM：光盘驱动器</li><li>REMOVABLE：可移动驱动器（一般指 U 盘）</li><li>FIXED：固定驱动器（一般指硬盘）</li><li>NETWORK：网络驱动器</li><li>RAMDISK：内存虚拟硬盘</li><li>UNKNOWN：未知的驱动器类型</li></ul>
	返回值： 成功：返回一个数组，0 元素为驱动器数量，1~n 元素为驱动器盘符（C: D: ...） 失败：设置\$error 为 1
	备注： 一般而言，如果要获取所有驱动器，则设置类型参数为“ALL”；如果要获取特定类

	型的驱动器，则设置类型为要获取的驱动器类型。值得注意的是，移动硬盘会被认为是“FIXED”类型，而不是“REMOVABLE”，这一点可能是由Windows驱动器分类机制制造的。
--	--

## 2、范例

获取所有类型驱动器的列表：

```
#include <Array.au3>
#include <MsgBoxConstants.au3>

;获取所有驱动器列表
Local $a_Drive = DriveGetDrive("ALL")
If @error Then
    ;如果发生错误，则提示用户获取时出错
    MsgBox($MB_OK + $MB_ICONERROR, "错误", "获取驱动器列表时发生错误！")
Else
    ;如果没有发生错误，则展示驱动器列表数组
    _ArrayDisplay($a_Drive, "$a_Drive")
EndIf
```

驱动器列表数组效果图：



(图 4-12)

### 4.3.2 获取驱动器信息

除了可以获得驱动器列表，我们还可以更为详细的获取每个驱动器的信息。

#### 1、获取驱动器状态、类型 ( DriveStatus、DriveGetType )

函数	说明
DriveStatus 获取驱动器状态	语法： DriveStatus ("驱动器") 参数： 驱动器：驱动器的盘符，英文字母+英文冒号（如 C:）
	返回值： UNKNOWN：驱动器状态未知，可能是由于驱动器未格式化等原因

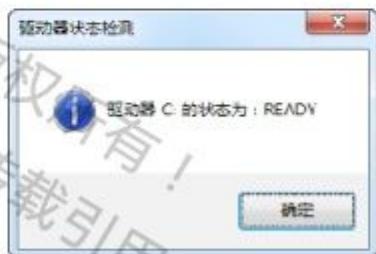
	<p>READY : 驱动器状态正常 NOTREADY : 软驱没有插入软盘，或光驱没有插入光盘 INVALID : 驱动器不存在，或网络驱动器无法正常访问</p>
DriveGetType 获取驱动器类型	<p>语法： DriveGetType ("驱动器" [, 选项 = 1])</p> <p>参数：</p> <p>驱动器：驱动器的盘符，英文字母+英文冒号（如 C:） 选项：[可选] \$DT_DRIVETYPE ( 值为 1 ) : 返回值为驱动器类型（默认） \$DT_SSDSTATUS ( 值为 2 ) : 返回值为驱动器是否为 SSD \$DT_BUSTYPE ( 值为 3 ) : 返回值为驱动器的总线类型</p> <p>返回值：</p> <p>成功：</p> <p>选项为\$DT_DRIVETYPE 时，返回： "Unknown" ( 未知 ), "Removable" ( 移动 ), "Fixed" ( 固定 ) "Network" ( 网络 ), "CDROM" ( 光驱 ), "RAMDisk" ( 内存盘 )</p> <p>选项为\$DT_SSDSTATUS 时，返回： 当驱动器是 SSD ( 固态硬盘 ) 时，返回字符串 "SSD" 当驱动器不是 SSD 时，返回空字符串</p> <p>选项为\$DT_BUSTYPE 时，返回： "Unknown" , "SCSI" , "ATAPI" , "ATA" , "1394" , "SSA" , "Fibre" , "USB" , "RAID" , "iSCSI" , "SAS" , "SATA" , "SD" , "MMC" , "Virtual" , "FileBackedVirtual"</p> <p>失败：返回空字符串，并将@error 设置为 1</p> <p>备注：</p> <p>对于选项，如果使用常量定义，则必须包含 Constants.au3，而如果直接使用值，则不必包含。</p>

## 例 1：检测驱动器状态

```
#include <MsgBoxConstants.au3>

Local $Drive = "C:"
Local $DriveStatus = DriveStatus($Drive)
MsgBox($MB_OK + $MB_ICONINFORMATION, _
"驱动器状态检测",_
"驱动器 " & $Drive & " 的状态为：" & $DriveStatus)
```

## 例 1 效果图：



( 图 4-13 )

## 例 2：检测驱动器类型

```
#include <Constants.au3>
#include <MsgBoxConstants.au3>

Local $Drive = "C:"
Local $DriveType = DriveGetType($Drive, $DT_DRIVETYPE)
Local $DriveSsdStatus = DriveGetType($Drive, $DT_SSDSTATUS)
Local $DriveBusType = DriveGetType($Drive, $DT_BUSTYPE)

Local $s = ""
If $DriveType <> "" Then
    $s &= $Drive & " 的驱动器类型为：" & $DriveType & @CRLF
EndIf

If $DriveSsdStatus <> "" Then
    $s &= $Drive & " 为固态硬盘" & @CRLF
Else
    $s &= $Drive & " 不是固态硬盘" & @CRLF
EndIf

If $DriveBusType <> "" Then
    $s &= $Drive & " 的驱动器总线为：" & $DriveBusType & @CRLF
EndIf

If $s = "" Then
    MsgBox($MB_OK + $MB_ICONERROR, "错误", "获取驱动器类型时发生错误！")
Else
    MsgBox($MB_OK + $MB_ICONINFORMATION, "信息", $s)
EndIf
```

例 2 效果图：



(图 4-14)

## 2、获取驱动器空间 (DriveSpaceFree、DriveSpaceTotal)

函数	说明
DriveSpaceFree	语法： DriveSpaceFree ("驱动器")

获取驱动器的剩余空间大小	参数： 驱动器：驱动器的盘符，英文字母+英文冒号（如 C:）
	返回值： 成功：返回驱动器的剩余空间大小（单位 MB） 失败：设置@error 为 1
DriveSpaceTotal 获取驱动器的总空间大小	语法： DriveSpaceTotal ("驱动器")
	参数： 驱动器：驱动器的盘符，英文字母+英文冒号（如 C:）

返回值：  
成功：返回驱动器的总空间大小（单位 MB）  
失败：设置@error 为 1

### 说明：

- 返回的空间大小单位为 MB，如果要以 GB 作为单位需要将返回值除以 1024
- 可以使用 Round 函数对返回值四舍五入，Round 的语法很简单：“Round(数值, 位数)”，数值就是将要被四舍五入的数值，位数是保留几位（如 2 表示小数点后 2 位），此函数后续章节介绍数学函数时会更加详细的探讨。

### 例：

```
#include <MsgBoxConstants.au3>

Local $Drive = "C:"

Local $FreeSpace = DriveSpaceFree($Drive)
$FreeSpace = Round($FreeSpace / 1024, 2)

Local $TotalSpace = DriveSpaceTotal($Drive)
$TotalSpace = Round($TotalSpace / 1024, 2)

MsgBox($MB_OK + $MB_ICONINFORMATION, _ 
    "驱动器空间检测", _ 
    "驱动器 " & $Drive & " 的剩余空间为：" & $FreeSpace & "G" & @CRLF & _ 
    "驱动器 " & $Drive & " 的总空间为：" & $TotalSpace & "G")
```

### 效果图：



### 3、获取驱动器文件系统、卷标 ( DriveGetFileSystem、DriveGetLabel )

函数	说明
DriveGetFileSystem 获取驱动器的文件系统类型	<p>语法： DriveGetFileSystem ( "驱动器" )</p> <p>参数： 驱动器：驱动器的盘符，英文字母+英文冒号（如 C:）</p> <p>返回值： 成功：返回驱动器文件系统类型，返回值有： "FAT"、"FAT32"、"NTFS"、"exFAT"、"NWFS"、"CDFS"、"UDF" 等 注意，返回值为 1 时，可能是光驱没有插入光盘，或驱动器未格式化 失败：设置@error 为 1</p>
DriveGetLabel 获取驱动器的卷标	<p>语法： DriveGetLabel ( "驱动器" )</p> <p>参数： 驱动器：驱动器的盘符，英文字母+英文冒号（如 C:）</p> <p>返回值： 成功：返回驱动器的卷标字符串 失败：设置@error 为 1</p>

例：

```
#include <MsgBoxConstants.au3>

Local $Drive = "C:"

Local $FileSystem=DriveGetFileSystem($Drive)
Local $Label=DriveGetLabel($Drive)

MsgBox($MB_OK + $MB_ICONINFORMATION, _ 
    "驱动器信息检测", _ 
    "驱动器 " & $Drive & " 的文件系统为：" & $FileSystem & @CRLF & _ 
    "驱动器 " & $Drive & " 的卷标为：" & $Label)
```

效果图：



(图 4-16)

#### 4、获取驱动器序列号 ( DriveGetSerial )

函数	说明
DriveGetSerial 返回驱动器的序列号	<p>语法： DriveGetSerial ( "驱动器" )</p> <p>参数：</p>

	驱动器：驱动器的盘符，英文字母+英文冒号（如 C:）
	返回值： 成功：返回驱动器的序列号字符串 失败：设置@error 为 1
	备注： 此处的序列号并不是硬件序列号，而是驱动器的 Windows 卷 ID。

例：

```
#include <MsgBoxConstants.au3>

Local $Drive = "C:"
Local $VolumeID = DriveGetSerial($Drive)
MsgBox($MB_OK + $MB_ICONINFORMATION, _
    "驱动器信息检测", _
    "驱动器 " & $Drive & " 的序列号为：" & $VolumeID)
```

效果图：



(图 4-17)

## 5、范例

使用本节所学函数，获取一份详细的驱动器信息列表

```
#include <Array.au3>
#include <Constants.au3>
#include <MsgBoxConstants.au3>

Opt('MustDeclareVars', 1)

_Main()
Exit

Func _Main()
; 获取所有驱动器列表
Local $a_Drive = DriveGetDrive('ALL')
If @error Then
    ; 如果获取驱动器列表失败，则提示用户
    MsgBox($MB_OK + $MB_ICONERROR, '错误', '获取驱动器列表时发生错误！')
Else
    ; 如果获取驱动器列表正常，则继续获取详细信息
```

```

;~ _ArrayDisplay($a_Drive, '$a_Drive') ;调试信息，查看驱动器列表

;创建驱动器信息数组
;数组的行数=驱动器数+1
;数组的列数根据第 0 行的赋值自动指定
Local $a_DriveInfo[$a_Drive[0] + 1][0] = [
    ['盘符', '状态', '类型', 'SSD 状态', '总线', '剩余空间', '总空间', '文件系统', '卷标', '卷 ID']
]

Local $i, $Drive
;遍历驱动器列表数组
For $i = 1 To $a_Drive[0]
    $Drive = $a_Drive[$i]

    ;驱动器盘符
    $a_DriveInfo[$i][0] = $Drive
    ;驱动器状态
    $a_DriveInfo[$i][1] = DriveStatus($Drive)
    ;驱动器类型
    $a_DriveInfo[$i][2] = DriveGetType($Drive, $DT_DRIVETYPE)
    ;驱动器的 SSD 状态
    $a_DriveInfo[$i][3] = DriveGetType($Drive, $DT_SSDSTATUS)
    ;驱动器的总线类型
    $a_DriveInfo[$i][4] = DriveGetType($Drive, $DT_BUSTYPE)
    ;驱动器剩余空间 ( 单位 GB )
    $a_DriveInfo[$i][5] = Round(DriveSpaceFree($Drive) / 1024, 2)
    ;驱动器总空间 ( 单位 GB )
    $a_DriveInfo[$i][6] = Round(DriveSpaceTotal($Drive) / 1024, 2)
    ;驱动器的文件系统
    $a_DriveInfo[$i][7] = DriveGetFileSystem($Drive)
    ;驱动器的卷标
    $a_DriveInfo[$i][8] = DriveGetLabel($Drive)
    ;驱动器的卷 ID
    $a_DriveInfo[$i][9] = DriveGetSerial($Drive)

Next

_ArrayDisplay($a_DriveInfo, '$a_DriveInfo')

EndIf
EndFunc  ;==>_Main

```

效果图：

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9
[0]	盘符	状态	类型	SSD状态	包线	剩余空间	总空间	文件系统	卷标	卷ID
[1]	c:	READY	Fixed	SSD	RAID	73.42	111.69	NTFS	OS	2053011616
[2]	d:	READY	Fixed		RAID	798.1	800	NTFS	Data1	1689525056
[3]	e:	READY	Fixed		RAID	430.87	800	NTFS	Data2	1253705871
[4]	f:	READY	Fixed		RAID	143.48	203	NTFS	Temp	3022041019
[5]	g:	READY	Removable		USB	14.64	14.64	exFAT	SKJ3-16G	4235299343

Copy Data & Mbr/Res      Copy Data Only  
.6J [10]      Exit Script

(图 4-18)

### 4.3.3 设置驱动器信息

#### 1、设置驱动器卷标 (DrivesetLabel)

函数	说明
DrivesetLabel 设置驱动器卷标	语法： DrivesetLabel ("驱动器", "卷标") 参数： 驱动器：驱动器的盘符，英文字母+英文冒号（如 C:） 卷标：即将设置的卷标 返回值： 成功：返回 1 失败：返回 0

#### 2、范例

```
Local $r = DrivesetLabel("C:", "OS")
If $r Then
    MsgBox(0 + 64, "设置卷标", "卷标设置成功！")
Else
    MsgBox(0 + 16, "设置卷标", "卷标设置失败！")
Endif
```

效果图：



(图 4-19)

#### 4.3.4 网络驱动器

##### 1、映射网络驱动器 ( DriveMapAdd、DriveMapDel )

函数	说明
DriveMapAdd 映射网络驱动器	<p>语法：  <code>DriveMapAdd ( "设备", "远程共享" [, 标志 = 0 [, "用户名" [, "密码"]]] )</code></p> <p>参数：</p> <p>设备：  一般为盘符 ( 如 X: ), 如果使用 "*" 则自动分配当前未分配的盘符 ;  也可以为设备 ( 如 LPT1: ), 不过不常用</p> <p>远程共享：  远程共享地址 ( 如 "\\\\" 远程服务器\共享目录" )</p> <p>标志 : [可选] ( 可组合使用 , 相加即可 )</p> <p>0 : 默认  1 : 持续映射  8 : 如果需要 , 弹出身份验证对话框</p> <p>用户名 : [可选] 连接所需的用户名 ( "用户名" 或 "域\用户名" )</p> <p>密码 : [可选] 连接所需的密码</p>
	<p>返回值：</p> <p>成功 : 1  失败 : 0</p>
	<p>备注：</p> <p>注意 , 当 "设备" 为 "*" 时 , 成功时的返回值为自动分配的盘符 , 如 "X."</p>
DriveMapDel 断开已映射的网 络驱动器	<p>语法：  <code>DriveMapDel ( "设备" )</code></p> <p>参数：</p> <p>设备 : 已映射的盘符 ( X: ) 或设备 ( LPT1: )</p> <p>返回值：</p> <p>成功 : 1  失败 : 0</p> <p>备注：</p> <p>如果映射的连接没有使用盘符 , "设备" 可写成 "\\\\" 远程服务器\共享目录" 的形式</p>

##### 2、获取网络驱动器信息 ( DriveMapGet )

函数	说明
DriveMapGet 获取已映射的网 络驱动器信息	<p>语法：  <code>DriveMapGet ( "设备" )</code></p> <p>参数：</p> <p>设备 : 已映射的盘符 ( X: ) 或设备 ( LPT1: )</p>
	<p>返回值：</p> <p>成功 : 返回指定网络驱动器信息  失败 : 返回空字符串 , 并将 @error 设置为 1</p>

### 3、范例

例 1：映射位于 192.168.0.232 的主机上的 data1 目录到本地 Y 驱动器，并设置为持续映射，共享账号为“Skye”，密码为“abc123”。

```
DriveMapAdd("Y:", "\\\\"192.168.0.232\\data1", 1, "Skye", "abc123")
```

效果图：



(图 4-20)

例 2：获取网络映射驱动器 Y 的信息

```
Local $r = DriveMapGet("Y:")
MsgBox(0, "", $r)
```

上例返回值为：\\192.168.0.232\data1，即映射的源位置。

例 3：断开已映射的网络驱动器 Y

```
DriveMapDel("Y:")
```

### 4.3.5 关于光驱

#### 1、弹出、关闭光驱 ( CDTray )

函数	说明
CDTray 弹出或关闭光驱	<p>语法： CDTray ( "驱动器", "操作" )</p> <p>参数： 驱动器：光盘驱动器的盘符 操作：“open” 弹出光驱，“close” 关闭光驱</p> <p>返回值： 成功：1 失败：0</p> <p>备注： ( 1 ) 本函数也可以操作一些虚拟光驱卸载其中光盘映像； ( 2 ) 本函数只能用于本地光驱，而对远程映射的光驱无效； ( 3 ) 对于一些必须手动关闭的光驱（如笔记本），本函数的“close” 操作无效</p>

### 2、范例

例 1：弹出、关闭光驱

```
;设定光驱盘符为 G:
Local $CDrom = 'G:'

;弹出光驱
```

```
CDTray($CDrom, 'Open')
```

;等待 3 秒

```
Sleep(3000)
```

;关闭光驱

```
CDTray($CDrom, 'Close')
```

#### 例 2：智能弹出有光盘的光驱

```
;获取本机所有光驱的盘符 (有些 PC 上不只一个光驱)
Local $a_CDRom = DriveGetDrive("CDRom")

Local $i
;遍历所有光驱
For $i = 1 To $a_CDRom[0]
    If DriveStatus($a_CDRom[$i]) = "READY" Then
        ;如果光驱的状态为 READY (即内部有光盘), 则弹出光盘
        CDTray($a_CDRom[$i], 'Open')
    EndIf
Next
```

## 4.4 目录

### 4.4.1 目录操作

#### 1、创建目录 ( DirCreate )

函数	说明
DirCreate	语法： DirCreate ( "路径" )
创建目录	参数： 路径：将要创建目录的路径
	返回值： 成功：1 失败：0
	备注： 如果即将创建目录的父目录不存在，父目录也将一同被创建。例如在没有 D:\XXX 的条件下创建 D:\XXX\YYY，函数会自动先创建 D:\XXX，再创建 D:\XXX\YYY。这是一个很实用的功能，可以有效避免因父目录不存在而造成的目录创建失败。

#### 2、复制、移动目录 ( DirCopy、DirMove )

函数	说明
DirCopy 复制目录(包括其中子目录和所有文件)	<p>语法： DirCopy ("源目录", "目标目录" [, 标志 = 0])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>源目录：即将被复制的目录(形如 d:\xxx，结尾无反斜杠)</li> <li>目标目录：即将被复制到的位置(形如 d:\yyy，结尾无反斜杠)</li> <li>标志：[可选] <ul style="list-style-type: none"> <li>\$FC_NOOVERWRITE (值为 0)：不覆盖已存在的文件，这也是默认值</li> <li>\$FC_OVERWRITE (值为 1)：覆盖已存在的文件</li> </ul> </li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：1</li> <li>失败：0</li> </ul> <p>备注：</p> <ul style="list-style-type: none"> <li>(1) 如果“标志”使用常量，则必须事先包含 FileConstants.au3，而如果直接使用值则没必要包含；</li> <li>(2) 当目标目录结构不存在时，将自动创建。</li> </ul>
DirMove 移动目录(包括其中子目录和所有文件)	<p>语法： DirMove ("源目录", "目的目录" [, 标志 = 0])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>源目录：即将被复制的目录(形如 d:\xxx，结尾无反斜杠)</li> <li>目标目录：即将被复制到的位置(形如 d:\yyy，结尾无反斜杠)</li> <li>标志：[可选] <ul style="list-style-type: none"> <li>\$FC_NOOVERWRITE (值为 0)：不覆盖已存在的文件，这也是默认值</li> <li>\$FC_OVERWRITE (值为 1)：覆盖已存在的文件</li> </ul> </li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：1</li> <li>失败：0</li> </ul> <p>备注：</p> <ul style="list-style-type: none"> <li>(1) 如果“标志”使用常量，则必须事先包含 FileConstants.au3，而如果直接使用值则没必要包含；</li> <li>(2) 当目标目录结构不存在时，将自动创建；</li> <li>(3) 如果源目录和目标目录不在同一个驱动器上，函数执行的是“先将源目录复制到目标位置，再将源目录删除”操作，而不是移动；</li> <li>(4) 可使用 DirMove 变通的为文件夹重命名，例如移动 d:\xxx 到 d:\yyy，相当于将目录 xxx 重命名为 yyy。</li> </ul>

### 3、删除目录 (DirRemove)

函数	说明
DirRemove 删除目录	<p>语法： DirRemove ("目标目录" [, 递归子目录 = 0])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>目标目录：即将被删除的目录</li> <li>递归子目录、子文件： <ul style="list-style-type: none"> <li>0：不递归，只有当目标目录为空时才会被删除</li> </ul> </li> </ul>

	1 : 递归，目标目录内所有子目录、子文件都将被删除，目标目录也将被删除
	返回值：
	成功：1
	失败：0

#### 4、范例

```
#include <FileConstants.au3>

;源目录
Local $SrcDir = 'D:\SrcDir'

;目标目录
Local $DestDir = 'D:\DestDir'

;创建目录 D:\SrcDir
DirCreate($SrcDir)
Sleep(3000)

;复制 D:\SrcDir 为 D:\DestDir
DirCopy($SrcDir, $DestDir, $FC_NOOVERWRITE)
Sleep(3000)

;删除刚才复制形成的 D:\DestDir
DirRemove($DestDir, 0)
Sleep(3000)

;移动 D:\SrcDir 到 D:\DestDir
DirMove($SrcDir, $DestDir, $FC_OVERWRITE)
Sleep(3000)

;删除刚才移动形成的 D:\DestDir
DirRemove($DestDir, 1)
```

运行此代码时注意观察 D 盘，会清晰的看到目录被创建、被复制、被移动和被删除。

#### 4.4.2 获取目录信息

##### 1、获取目录大小 ( DirGetSize )

函数	说明
DirGetSize 获取目录的大小	<p>语法： DirGetSize ( "路径" [, 标志 = 0] )</p> <p>参数：</p> <ul style="list-style-type: none"> <li>路径：即将被获取大小的目录路径</li> <li>标志：[可选] ( 可组合使用，值相加即可 )</li> <li>0 : ( 默认 ) 计算目标目录的总体积，其中子目录体积一并被计算在内</li> </ul>

	<p>1 : 拓展模式，返回一个数组 0 元素为目标目录体积 1 元素为目标目录中文件数 2 元素为目标目录中目录数 2 : 不计算目标目录中子目录的体积</p>
	<p>返回值： 成功：返回目录体积值，启用拓展模式时返回数组 失败：返回-1，并将@error 值设置为 1</p>

注意：

- 如果目标目录比较大，或目录结构复杂，目录大小的获取需要一定时间；
- 体积值除以 1024 后的单位为 KB，再除以 1024 为 MB，再除以 1024 才为 GB；
- 由于驱动器也是一种特殊的目录，所以也可以使用本函数获取驱动器的已用空间。

## 2、范例

例 1：获取 Windows 目录的体积

```
Local $Size = DirGetSize('C:\Windows')
$Size = Round($Size / 1024 / 1024 / 1024, 2)
MsgBox(0 + 64, '获取目录体积', 'C:\Windows 的体积为：' & $Size & 'GB')
```

例 1 效果图：



(图 4-21)

例 2：获取 Windows 目录的体积，以及其中文件数和其中目录数

```
Local $a_Size = DirGetSize('C:\Windows', 1)
MsgBox(0 + 64, '获取目录体积', 'C:\Windows 目录' & @CRLF & @CRLF &_
'体积为 ' & Round($a_Size[0] / 1024 / 1024 / 1024, 2) & 'GB' & @CRLF &_
'文件 ' & $a_Size[1] & '个' & @CRLF &_
'目录 ' & $a_Size[2] & '个')
```

例 2 效果图：



(图 4-22)

## 4.5 文件

### 4.5.1 文件操作

#### 1、复制、移动文件 (FileCopy、FileMove)

函数	说明
FileCopy 复制文件	<p>语法： FileCopy ("源文件", "目的路径" [, 标志 = 0])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>源文件：即将被复制的文件的路径（支持通配符）</li> <li>目的路径：即将被复制到的目的路径</li> <li>标志：[可选]（可组合使用，相加即可）           <ul style="list-style-type: none"> <li>\$FC_NOOVERWRITE（值为 0）：（默认）不覆盖已存在文件</li> <li>\$FC_OVERWRITE（值为 1）：覆盖已存在文件</li> <li>\$FC_CREATEPATH（值为 8）：自动创建不存在的目录结构</li> </ul> </li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：1</li> <li>失败：0</li> </ul> <p>备注：</p> <p>如果“标志”使用常量，则必须事先包含 FileConstants.au3，而如果直接使用值则没必要包含。</p>
FileMove 移动文件	<p>语法： FileMove ("源文件", "目的路径" [, 标志 = 0])</p> <p>参数：</p> <ul style="list-style-type: none"> <li>源文件：即将被移动的文件的路径（支持通配符）</li> <li>目的路径：即将被移动到的目的路径</li> <li>标志：           <ul style="list-style-type: none"> <li>\$FC_NOOVERWRITE（值为 0）：（默认）不覆盖已存在文件</li> <li>\$FC_OVERWRITE（值为 1）：覆盖已存在文件</li> <li>\$FC_CREATEPATH（值为 8）：自动创建不存在的目录结构</li> </ul> </li> </ul>

返回值：
成功：1
失败：0
备注：
(1) 如果“标志”使用常量，则必须事先包含 FileConstants.au3，而如果直接使用值则没必要包含；
(2) 可使用 FileMove 变通的为文件重命名，例如移动 d:\1.txt 到 d:\2.txt，相当于将文件 1.txt 重命名为 2.txt。

## 2、删除文件 ( FileDelete )

函数	说明
FileDelete 删除文件	语法： FileDelete ("文件") 参数： 文件：即将被删除的文件（支持通配符） 返回值： 成功：1 失败：0

## 3、范例

### 例 1：复制文件

```
#include <FileConstants.au3>

;一般写法（复制文件，覆盖已存在文件，若目标目录结构不存在则自动创建）
FileCopy('d:\test1\1.txt', 'd:\test2\1.txt', $FC_OVERWRITE + $FC_CREATEPATH)

;不指定目标文件名，自动以原文件名复制到目标目录，效果与上一条语句相同
FileCopy('d:\test1\1.txt', 'd:\test2\', $FC_OVERWRITE + $FC_CREATEPATH)

;复制的同时重命名文件
FileCopy('d:\test1\1.txt', 'd:\test2\2.txt', $FC_OVERWRITE + $FC_CREATEPATH)
```

### 例 2：使用通配符复制文件

```
#include <FileConstants.au3>

;复制 test1 目录下所有文件到 test2 目录
FileCopy('d:\test1\*', 'd:\test2\', $FC_OVERWRITE + $FC_CREATEPATH)

;复制 test1 目录下所有 txt 文件到 test2 目录
FileCopy('d:\test1\*.txt', 'd:\test2\', $FC_OVERWRITE + $FC_CREATEPATH)
```

### 例 3：移动文件

```
#include <FileConstants.au3>
```

```
;一般写法(移动文件, 覆盖已存在文件, 若目标目录结构不存在则自动创建)
FileMove('d:\test1\aa.txt', 'd:\test2\aa.txt', $FC_OVERWRITE + $FC_CREATEPATH)

;不指定目标文件名, 自动以原文件名移动到目标目录, 效果与上一条语句相同
FileMove('d:\test1\aa.txt', 'd:\test2\', $FC_OVERWRITE + $FC_CREATEPATH)

;移动的同时重命名文件
FileMove('d:\test1\aa.txt', 'd:\test2\bb.txt', $FC_OVERWRITE + $FC_CREATEPATH)

;重命名当前文件
FileMove('d:\test1\aa.txt', 'd:\test1\cc.txt', $FC_OVERWRITE + $FC_CREATEPATH)
```

**例 4：使用通配符移动文件**

```
#include <FileConstants.au3>

;移动 test1 目录下所有文件到 test2 目录
FileMove('d:\test1\*.*', 'd:\test2\', $FC_OVERWRITE + $FC_CREATEPATH)

;移动 test1 目录下所有 txt 文件到 test2 目录
FileMove('d:\test1\*.txt', 'd:\test2\', $FC_OVERWRITE + $FC_CREATEPATH)
```

**例 5：删除文件**

```
;删除 test1 目录下的 aa.txt
FileDelete('d:\test1\aa.txt')
```

**例 6：使用通配符删除文件**

```
;删除 test2 目录下的所有文件
FileDelete('d:\test2\*.*')

;删除 test2 目录下的所有文本文件
FileDelete('d:\test2\*.txt')
```

**4.5.2 存在性检查****1、检查文件存在性 ( FileExists )**

函数	说明
FileExists 检查文件的存在 性(亦可检查目录 的存在性)	语法： FileExists ("路径") 参数： 路径：即将被检查存在与否的文件、目录的路径 返回值：

	成功 : 1 , 目标文件或目录存在 失败 : 0 , 目标文件或目录不存在
	备注 : 由于驱动器是一种特殊的目录，所以本函数亦可用于检查驱动器的存在性。但对于没有插入光盘的光驱、没有插入软盘的软驱等，函数的返回值为 0。

### 2、范例

#### 例 1 : 检测文件存在性

```
Local $r = FileExists('C:\Windows\RegEdit.exe')
Switch $r
    Case 1
        MsgBox(0 + 64, '文件存在性检测', '目标文件存在！')
    Case 0
        MsgBox(0 + 16, '文件存在性检测', '目标文件不存在！')
EndSwitch
```

#### 例 2 : 检测目录存在性

```
Local $r = FileExists('C:\Program Files')
Switch $r
    Case 1
        MsgBox(0 + 64, '目录存在性检测', '目标目录存在！')
    Case 0
        MsgBox(0 + 16, '目录存在性检测', '目标目录不存在！')
EndSwitch
```

#### 例 3 : 检测盘符存在性

```
Local $r = FileExists('C:')
Switch $r
    Case 1
        MsgBox(0 + 64, '驱动器存在性检测', '目标驱动器存在！')
    Case 0
        MsgBox(0 + 16, '驱动器存在性检测', '目标驱动器不存在！')
EndSwitch
```

### 4.5.3 获取文件信息

#### 1、获取文件的体积、属性 ( **FileGetSize**、**FileGetAttrib** )

函数	说明
FileGetSize 获取文件的大小	语法 : FileGetSize ("文件") 参数 : 文件 : 目标文件的路径

	<p><b>返回值：</b> 成功：返回文件的字节大小 失败：返回 0，并将@error 设置为 1</p> <p><b>备注：</b> 所返回的文件大小，除以 1024 后体积单位为 KB，再除以 1024 后体积单位为 MB</p>
FileGetAttrib 获取文件属性	<p><b>语法：</b> <code>FileGetAttrib ("文件")</code></p> <p><b>参数：</b> 文件：目标文件的路径</p> <p><b>返回值：</b> 成功：返回文件属性代码字符串 失败：返回空字符串，并将@error 设置为 1</p> <p><b>备注：</b> 属性字符串中，各字符代表的含义 “R”：只读文件 “A”：存档文件 “S”：系统文件 “H”：隐藏文件 “N”：普通文件 “D”：目录文件 “O”：脱机文件 “C”：压缩文件（NTFS 压缩，非其他压缩） “T”：临时文件</p>

## 2、获取文件的版本、时间（FileGetVersion、FileGetTime）

函数	说明
FileGetVersion 获取文件的版本信息	<p><b>语法：</b> <code>FileGetVersion ("文件" [, "版本信息字段"])</code></p> <p><b>参数：</b> 文件：目标文件的路径 版本信息字段：[可选]指定获取特定字段的版本信息</p> <p><b>返回值：</b> 成功：返回版本信息字符串（形如 3.3.12.0） 失败：返回空字符串，并将@error 设置为 1</p> <p><b>备注：</b> 可用的版本信息字段： <code>Comments</code>（注释） <code>InternalName</code>（内部名称） <code>ProductName</code>（产品名称） <code>CompanyName</code>（公司名称） <code>LegalCopyright</code>（法律版权） <code>ProductVersion</code>（产品版本） <code>FileDescription</code>（文件说明） <code>LegalTrademarks</code>（法律商标）</p>

	PrivateBuild ( 专用编译 ) FileVersion ( 文件版本 ) OriginalFilename ( 原始文件名 ) SpecialBuild ( 特别编译 )
<b>FileGetTime</b> 获取文件的时间信息	<p>语法：  <b>FileGetTime ( "文件" [, 选项 = 0 [, 格式 = 0]] )</b></p> <p>参数：</p> <p>文件：目标文件的路径          选项：[可选]指定获取哪种时间信息              \$FT_MODIFIED ( 值为 0 )：修改时间 ( 默认 )              \$FT_CREATED ( 值为 1 )：创建时间              \$FT_ACCESSED ( 值为 2 )：访问时间          格式：[可选]指定返回值格式              0：返回数组 ( 默认 )                  0 元素：年 ( 四位数 )，如 2014                  1 元素：月 ( 01~12 )，如 07                  2 元素：日 ( 01~31 )，如 25                  3 元素：时 ( 00~23 )，如 08                  4 元素：分 ( 00~59 )，如 25                  5 元素：秒 ( 00~59 )，如 17              1：返回字符串                  格式为 YYYYMMDDHHMMSS ( 年月日时分秒 )                  如 20140725082517，单数前会自动补 0</p> <p>返回值：</p> <p>成功：返回字符串或数组 ( 由 “格式” 参数决定 )          失败：将 @error 设置为非 0 值</p> <p>备注：</p> <p>选项使用常量时，需事先包含 FileConstants.au3，直接使用值则无需包含</p>

### 3、范例

#### 例 1，获取文件体积

```

;以 IE 浏览器主程序文件为例
Local $TagFile = 'C:\Program Files\Internet Explorer\iexplore.exe'

;获取目标文件体积
Local $Size = FileGetSize($TagFile)
;将目标文件体积单位折算为 MB，并四舍五入
$Size = Round($Size / 1024 / 1024, 2)

MsgBox(0 + 64, "目标文件体积为：" & $Size & "MB")

```

例 1 效果图



(图 4-23)

**例 2，获取文件属性**

;以 IE 浏览器主程序文件为例

Local \$TagFile = 'C:\Program Files\Internet Explorer\iexplore.exe'

;获取目标文件属性

Local \$Attrib = FileGetAttrib(\$TagFile)

MsgBox(0 + 64, "目标文件的属性为：" &amp; \$Attrib)

**例 2 效果图**

(图 4-24)

**例 3，获取文件版本信息**

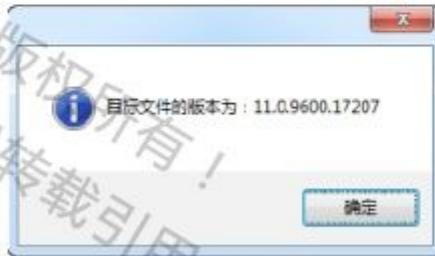
;以 IE 浏览器主程序文件为例

Local \$TagFile = 'C:\Program Files\Internet Explorer\iexplore.exe'

;获取文件版本信息

Local \$Ver = FileGetVersion(\$TagFile)

MsgBox(0 + 64, "目标文件的版本为：" &amp; \$Ver)

**例 3 效果图**

(图 4-25)

### 例 4，获取文件的详细版本信息

```
#include <Array.au3>

;以 IE 浏览器主程序文件为例
Local $TagFile = 'C:\Program Files\Internet Explorer\iexplore.exe'

;创建版本信息数组，第一列为版本字段名，第二列用于之后存储对应的版本信息
Local $a_Ver[][] = [[], []]
    ['Comments', ""], _
    ['InternalName', ""], _
    ['ProductName', ""], _
    ['CompanyName', ""], _
    ['LegalCopyright', ""], _
    ['ProductVersion', ""], _
    ['FileVersion', ""], _
    ['OriginalFilename', ""], _
    ['SpecialBuild', ""]]

;以循环方式，逐个获取各版本字段对应的版本信息
Local $i
For $i = 1 To UBound($a_Ver, 1) - 1
    $a_Ver[$i][1] = FileGetVersion($TagFile, $a_Ver[$i][0])
Next

_ArrayDisplay($a_Ver, '$a_Ver')
```

例 4 效果图

Row	Col 0	Col 1
[0]		
[1]	Comments	iexplore
[2]	InternalName	Internet Explorer
[3]	ProductName	Microsoft Corporation
[4]	CompanyName	© Microsoft Corporation. All rights reserved.
[5]	LegalCopyright	
[6]	ProductVersion	11.00.9600.16428
[7]	FileVersion	Internet Explorer
[8]	FileDescription	
[9]	LegalTrademarks	
[10]	PrivateBuild	
[11]	FileVersion	11.00.9600.16428 (winblue_qdr.131013-1700)
[12]	OriginalFilename	IEXPLORE.EXE.MJL
[13]	SpecialBuild	

( 图 4-26 )

## 例 5，获取文件的时间信息（数组方式）

```
#include <Array.au3>

;以 IE 浏览器主程序文件为例
Local $TagFile = 'C:\Program Files\Internet Explorer\iexplore.exe'

;获取目标文件的时间信息数组
Local $a_Time = FileGetTime($TagFile)

_ArrayDisplay($a_Time, '$a_Time')
```

例 5 效果图

Row	Col 0
[0]	2014
[1]	06
[2]	21
[3]	04
[4]	14
[5]	31

Copy Data & Hide Row | Copy Data Only  
[6] Exit Script

( 图 4-27 )

## 例 6，获取文件的时间信息（字符串方式）

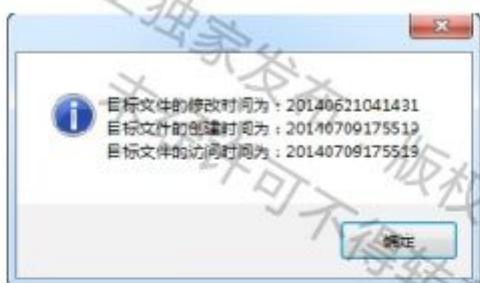
```
#include <FileConstants.au3>

;以 IE 浏览器主程序文件为例
Local $TagFile = 'C:\Program Files\Internet Explorer\iexplore.exe'

Local $Time1 = FileGetTime($TagFile, $FT_MODIFIED, 1)
Local $Time2 = FileGetTime($TagFile, $FT_CREATED, 1)
Local $Time3 = FileGetTime($TagFile, $FT_ACCESSED, 1)

MsgBox(0 + 64, "目标文件的修改时间为：" & $Time1 & @CRLF &_
"目标文件的创建时间为：" & $Time2 & @CRLF &_
"目标文件的访问时间为：" & $Time3)
```

例 6 效果图



(图 4-28)

#### 4.5.4 设置文件信息

除了能够获取文件的信息外，还可以对文件的部分信息进行设置，例如设置文件的属性信息和时间信息。

##### 1、设置文件的属性 ( FileSetAttrib )

函数	说明
FileSetAttrib 设置文件的属性	<p>语法：</p> <pre>FileSetAttrib ( "文件", "+-属性代码" [, 递归 = 0] )</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>文件：目标文件路径（可使用通配符）</li> <li>属性代码：</li> <ul style="list-style-type: none"> <li>属性代码解释见备注</li> <li>+：增加属性，如 "+RH" 即增加只读、隐藏属性</li> <li>-：取消属性，如 "-SH" 即取消系统、隐藏属性</li> </ul> <li>递归：</li> <ul style="list-style-type: none"> <li>0：（默认）不递归子目录、子文件</li> <li>1：递归子目录、子文件</li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：1</li> <li>失败：0</li> </ul> <p>备注：</p> <p>属性代码：</p> <ul style="list-style-type: none"> <li>"R"：只读文件</li> <li>"A"：存档文件</li> <li>"S"：系统文件</li> <li>"H"：隐藏文件</li> <li>"N"：普通文件</li> <li>"O"：脱机文件</li> <li>"T"：临时文件</li> </ul> </ul>

##### 2、设置文件的时间 ( FileSetTime )

函数	说明
----	----

FileSetTime 设置文件的时间 信息	语法： FileSetTime ("文件", "时间" [, 类型 = 0 [, 递归 = 0]])
	参数： 文件：目标文件路径（可使用通配符） 时间：格式为 YYYYMMDDHHMMSS（年月日时分秒），留空则为当前时间 （如 20140725094316，表示 2014 年 7 月 25 日 9 点 43 分 16 秒） 类型：[可选]修改哪种时间信息 \$FT_MODIFIED (值为 0)：修改时间（默认） \$FT_CREATED (值为 1)：创建时间 \$FT_ACCESSED (值为 2)：访问时间
	递归： 0：（默认）不递归子目录、子文件 1：递归子目录、子文件
	返回值： 成功：1 失败：0
备注： （1）如果所设置时间早于 1980 年 1 月 1 日，设置将无效，返回值为 0； （2）如果文件为只读文件，设置将无效，返回值为 0。	

### 3、范例

#### 例 1：设置文件属性

```
;以当前目录下 test.txt 举例
Local $File = @ScriptDir & '\test.txt'

;设置目标文件属性为只读、隐藏
FileSetAttrib($File, '+RH')
;此时目标文件会不可写、不可见

Sleep(3000)

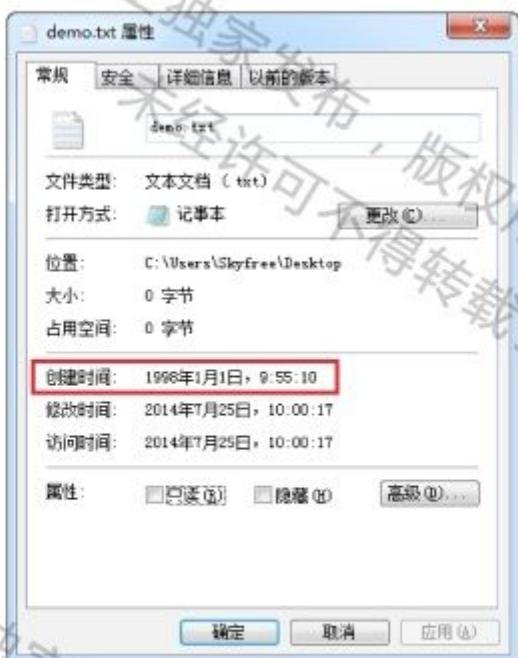
;取消目标文件的只读、隐藏属性
FileSetAttrib($File, '-RH')
;此时目标文件将恢复正常状态，可写、可见
```

#### 例 2：设置文件时间

```
#include <FileConstants.au3>

;以当前目录下 demo.txt 举例
Local $File = @ScriptDir & '\demo.txt'

;设置目标文件的创建时间为 1998 年 1 月 1 日 8 点 55 分 10 秒
FileSetTime($File, '19980101095510', $FT_CREATED)
```



(图 4-29)

#### 4.5.5 快捷方式

##### 1、创建文件的快捷方式 ( FileCreateShortcut )

函数	说明
FileCreateShortcut 创建快捷方式	<p>语法：</p> <pre>FileCreateShortcut ( "文件", "快捷方式" [, "工作目录" [, "参数" [, "描述" [, "图标文件" [, "快捷键" [, 图标索引 [, 状态]]]]]] ] )</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>文件：即将被创建快捷方式的文件路径</li> <li>快捷方式：快捷方式的路径（注意，快捷方式以.lnk为后缀名）</li> <li>工作目录：[可选]程序文件的工作目录</li> <li>参数：[可选]程序文件的运行参数</li> <li>描述：[可选]对快捷方式的描述（鼠标点击快捷方式时以 Tip 方式显示的说明）</li> <li>图标文件：[可选]包含图标的文件（可以是.ico文件，也可以是.dll、.exe等）</li> <li>快捷键：[可选]快捷方式的快捷键</li> <li>图标索引：[可选]图标在图标文件中的索引</li> <li>状态：[可选] <ul style="list-style-type: none"> <li>@SW_SHOWNORMAL，激活并显示一个窗体</li> <li>@SW_SHOWMINNOACTIVE，最小化方式显示窗体，且不激活</li> <li>@SW_SHOWMAXIMIZED，最大化方式显示窗体，激活</li> </ul> </li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功：1</li> <li>失败：0</li> </ul> <p>备注：</p>

	(1) 无需设置的参数，留空（空字符串）即为默认； (2) 快捷键相关知识在后续学习 Send 函数时会详细说明，在此不多赘述，感兴趣可以查看下帮助文档内相关内容。
--	---

## 2、获取文件的快捷方式信息 ( FileGetShortcut )

函数	说明
FileGetShortcut 获取快捷方式信息	<p>语法：  <code>FileGetShortcut ( "快捷方式" )</code></p> <p>参数：          快捷方式：快捷方式的路径（注意，快捷方式以.lnk 为后缀名）</p> <p>返回值：          成功：返回一个数组，包含快捷方式的信息（对应参考 FileCreateShortcut）          0 元素：快捷方式所指向的文件的路径          1 元素：工作目录          2 元素：程序运行参数          3 元素：快捷方式描述          4 元素：图标文件路径          5 元素：图标在图标文件中的索引          6 元素：程序运行状态值          值 1 对应：@SW_SHOWNORMAL          值 7 对应：@SW_SHOWMINNOACTIVE          值 3 对应：@SW_SHOWMAXIMIZED          失败：设置@error 为 1       </p>

## 3、范例

### 例 1：创建一个快捷方式

```
Local $File = 'C:\Program Files\Internet Explorer\iexplore.exe' ;<-- 以 IE 浏览器为例
Local $Link = @DesktopDir & '\IE.lnk' ;<-- 创建桌面快捷方式 "IE.lnk"

FileCreateShortcut($File, $Link)
```

### 例 2：指定快捷方式的工作目录、参数、描述

```
Local $WorkDir = 'C:\Program Files\Internet Explorer' ;<-- 设置工作目录为 IE 所在目录
Local $File = $WorkDir & '\iexplore.exe'
Local $Link = @DesktopDir & '\IE.lnk'

Local $Parm = 'http://www.itiankong.com/' ;<-- 设置运行参数为 URL，对 IE 而言即运行时打开此 URL
Local $Desc = 'IT 天空' ;<-- 设置快捷方式描述

FileCreateShortcut($File, $Link, $WorkDir, $Parm, $Desc)
```

### 例 3：指定快捷方式的图标文件

```
Local $WorkDir = 'C:\Program Files\Internet Explorer'
Local $File = $WorkDir & '\iexplore.exe'
```

## Let's AutoIt Plus

```
Local $Link = @DesktopDir & '\IE.lnk'  
Local $Parm = 'http://www.itiankong.com/'  
Local $Desc = 'IT 天空'  
Local $IconFile = @ScriptDir & '\acc.ico' ; <-- 图标文件为.ico 文件  
  
FileCreateShortcut($File, $Link, $WorkDir, $Parm, $Desc, $IconFile)
```

### 例 4：指定快捷方式的图标文件、图标索引

```
Local $WorkDir = 'C:\Program Files\Internet Explorer'  
Local $File = $WorkDir & '\iexplore.exe'  
Local $Link = @DesktopDir & '\IE.lnk'  
Local $Parm = 'http://www.itiankong.com/'  
Local $Desc = 'IT 天空'  
Local $IconFile = 'C:\Windows\system32\imageres.dll' ; <-- 图标文件为 dll 文件  
Local $IconIndex = 15 ; <-- 指定图标索引  
  
FileCreateShortcut($File, $Link, $WorkDir, $Parm, $Desc, $IconFile, ", $IconIndex)
```

### 例 5：指定快捷方式的快捷键

```
Local $WorkDir = 'C:\Program Files\Internet Explorer'  
Local $File = $WorkDir & '\iexplore.exe'  
Local $Link = @DesktopDir & '\IE.lnk'  
Local $Parm = 'http://www.itiankong.com/'  
Local $Desc = 'IT 天空'  
Local $IconFile = 'C:\Windows\system32\imageres.dll'  
Local $IconIndex = 15  
Local $Hotkey = '^!i' ; <-- 指定快捷键为 Ctrl+Alt+I  
  
FileCreateShortcut($File, $Link, $WorkDir, $Parm, $Desc, $IconFile, $Hotkey, $IconIndex)
```

### 例 6：指定目标程序的运行状态

```
Local $WorkDir = 'C:\Program Files\Internet Explorer'  
Local $File = $WorkDir & '\iexplore.exe'  
Local $Link = @DesktopDir & '\IE.lnk'  
Local $Parm = 'http://www.itiankong.com/'  
Local $Desc = 'IT 天空'  
Local $IconFile = 'C:\Windows\system32\imageres.dll'  
Local $IconIndex = 15  
Local $Hotkey = '^!i'  
  
FileCreateShortcut($File, $Link, $WorkDir, $Parm, $Desc, _  
    $IconFile, $Hotkey, $IconIndex, @SW_SHOWMAXIMIZED) ; <-- 最大化并激活
```

所创建快捷方式的属性：



(图 4-30)

## 例 7：获取快捷方式的信息

```

Local $Link = @DesktopDir & '\IE.lnk'

Local $a_LinkInfo = FileGetShortcut($Link)
MsgBox(0 + 64, '快捷方式信息', _
    '源文件路径 : ' & $a_LinkInfo[0] & @CRLF & _
    '工作目录 : ' & $a_LinkInfo[1] & @CRLF & _
    '程序运行参数 : ' & $a_LinkInfo[2] & @CRLF & _
    '快捷方式描述 : ' & $a_LinkInfo[3] & @CRLF & _
    '图标文件 : ' & $a_LinkInfo[4] & @CRLF & _
    '图标索引 : ' & $a_LinkInfo[5] & @CRLF & _
    '程序运行状态 : ' & $a_LinkInfo[6] & @CRLF)

```

所获取的快捷方式信息：



(图 4-31)

## 4.5.6 更改工作目录

### 1、变更工作目录 ( FileChangeDir )

函数	说明
FileChangeDir 改变当前程序的工作目录	语法： <code>FileChangeDir ( "工作目录" )</code> 参数： 工作目录：即将被变更为的工作目录 返回值： 成功：1 失败：0 备注： 这个函数经常被用来解决“因工作目录的变更而导致的相对目录不正确问题”

### 2、范例

```
;当工作目录不等于当前目录时，变更工作目录为当前目录
If @WorkingDir <> @ScriptDir Then FileChangeDir(@ScriptDir)
```

上例是用法，在使用相对路径较多的程序中较常用。

此时回忆 4.2.1 节第 4 小节，若在 b.exe 的代码头部添加上述代码，则在 a.exe 中不用指定 b.exe 的工作目录，b.exe 也能够正常调用 c.exe。

## 4.5.7 回收站

### 1、删除文件至回收站 ( FileRecycle、FileRecycleEmpty )

函数	说明
FileRecycle 删除文件至回收站	语法： <code>FileRecycle ( "目标文件" )</code> 参数： 目标文件：目标文件或目录（支持通配符） 返回值： 成功：1 失败：0 备注： 删除目录时，不需要以反斜杠结尾
FileRecycleEmpty 清空回收站	语法： <code>FileRecycleEmpty ( ["驱动器"] )</code> 参数： 驱动器：[可选]用于指定清空哪个驱动器上的回收站，当不指定时，清空所有回收站内文件或目录 返回值： 成功：1

	失败 : 0
--	--------

## 2、范例

FileRecycle 和 FileRecycleEmpty 是两个非常简便易用的函数，当考虑删除文件又不想彻底删除时，配合使用很愉快！

### 例 1：删除文件至回收站

```
;删除当前目录下的 test.txt 文件
FileRecycle(@ScriptDir & '\test.txt')
```

```
;删除当前目录下的 xxx 目录
FileRecycle(@ScriptDir & '\xxx')
```

### 例 2：清空回收站

```
;清空 D 盘的回收站
FileRecycleEmpty('D:')
```

```
;清空整个回收站
FileRecycleEmpty()
```

## 4.6 文件的装载

将“文件的装载”单独列为一节来讲，是因为文件的装载是一个重要且常见的功能。我们所书写的程序有时会依赖其他文件（如.exe、.dll）以完成特定的功能，而目标系统中可能并不包含这些文件，这时我们需要将这些文件“包含”进我们所编写的程序内，待需要时将它们“装载”到特定位置。

### 1、装载文件 ( FileInstall )

函数	说明
FileInstall 装载文件到特定 位置	<p>语法：</p> <pre>FileInstall ( "文件", "释放路径" [, 标志 = 0] )</pre> <p>参数：</p> <ul style="list-style-type: none"> <li>文件：即将被释放的文件（必须是字符串，可以是相对路径）</li> <li>释放路径：文件被释放的位置</li> <li>标志：[可选]是否覆盖已存在的文件           <ul style="list-style-type: none"> <li>\$FC_NOOVERWRITE ( 值为 0 )：( 默认 ) 不覆盖</li> <li>\$FC_OVERWRITE ( 值为 1 )：覆盖</li> </ul> </li> </ul> <p>返回值：</p> <ul style="list-style-type: none"> <li>成功 : 1</li> <li>失败 : 0</li> </ul>

	<p>备注：</p> <p>( 1 ) “文件”必须是字符串，不得包含变量、函数调用等非字符串内容；</p> <p>( 2 ) 如果标志使用常量，则必须事先包含 FileConstants.au3；如果直接使用值，则无需包含；</p> <p>( 3 ) 注意，如果文件被释放到特定目录中，目录必须存在，否则释放将失败，为了避免这种情况，可事先使用 DirCreate 函数创建释放目录；</p> <p>( 4 ) 如果脚本没有被编译，则 FileInstall 的执行方式与 FileCopy 基本相同。</p>
--	--

## 2、范例

装载 7za.exe 到系统临时目录：

```
FileInstall('.\7za.exe', @TempDir & '\7za.exe', 1)
```

FileInstall 函数应该做多层意义上的理解：

( 1 ) 对于未编译的脚本 (.au3) , FileInstall 的功能仅仅是简单复制，功能类似于 FileCopy。上述代码对于未编译脚本的意义应该理解为“将当前目录下的 7za.exe 复制到系统临时目录下”。

( 2 ) 在编译脚本时，对于编译器而言，FileInstall 的文件参数相当于告之编译器将哪个文件包含进当前程序内。上述代码对于编译器的意义应该理解为“将当前目录下的 7za.exe 包含至当前程序内”。

( 3 ) 对于已编译的脚本 (.exe) , 文件参数则代表被包含进来的文件资源（不再是其路径下的文件），释放路径参数则代表将已包含的文件所要释放到的位置。上述代码对于已编译的程序的意义应该理解为“将包含的 7za.exe 文件释放到系统临时目录下”。

值得注意的是，包含的文件越多，则程序体积就会越大，所以不要包含过多文件到程序中。

FileInstall 函数在实践应用中会经常用到，本节先做了解，在今后实践中逐步熟练。

## 4.7 注册表

注册表包含大量 Windows 操作系统的配置信息，对系统配置的修改大多数就是对注册表的修改。在日常工作中，需要用到注册表修改的情况非常多，熟练掌握注册表的读、写、删，对于驾驭系统有着极大的帮助。本节，我们就来学习一下怎样使用 Au3 函数修改注册表信息。

在学习之前，我们先来了解一些注册表的常识。

### ( 1 ) 注册表的 5 大根键：

HKEY\_LOCAL\_MACHINE，可简写为 HKLM，这是最重要的注册表键（没有之一），当前计算机几乎所有的系统配置信息都在此注册表键内。

HKEY\_USERS，可简写为 HKU，用户配置信息均保存在此注册表键内。

HKEY\_CURRENT\_USER，可简写为 HKCU，当前用户配置信息。其实这个键是 HKEY\_USERS 中的一个用户子键的映射，自动将当前登录的用户注册表键映射至此。

HKEY\_CLASSES\_ROOT，可简写为 HKCR，主要保存了文件与应用程序的对应关系，包括对于某些特定文件使用怎样的右键菜单等。其实这个键是 HKLM\SOFTWARE\Classes 的映射。

HKEY\_CURRENT\_CONFIG，可简写为 HKCC，当前配置信息。

### (2) 值类型

REG\_BINARY：二进制值

REG\_SZ：字符串值

REG\_MULTI\_SZ：多字符串值

REG\_EXPAND\_SZ：可扩充字符串值

REG\_QWORD：QWORD (64位) 值

REG\_DWORD：DWORD (32位) 值

### (3) 关于 64 位注册表值

如果使用 32 位程序读写 64 位 Windows 系统注册表值时，无论是读取或是写入位置都会被重定向，这属于经典的“64 位重定向问题”，将在 Level5 中讲述 x86、x64 问题时详解，这里仅做了解。为了使 32 位程序可以正常读写 64 位注册表项，可在 32 位程序中将注册表根键书为形如“HKLM64”、“HKCU64”的形式。

## 1. 注册表的读、写、删 (RegRead、RegWrite、RegDelete)

函数	说明
RegRead 读注册表键值	<p>语法： RegRead ("键", "值")</p> <p>参数： 键：注册表键项 值：注册表值项</p> <p>返回值： 成功：返回读取到的值，并将@extened 设置为值类型 失败：设置@error 为非 0 值 @error 值： 1：无法打开请求的键 2：无法打开请求的主键 3：无法连接远程注册表 -1：无法打开请求的值 -2：不支持的值类型</p> <p>备注： (1)当读取的值为 REG_MULTI_SZ 类型时，多个字符串间是使用@LF 作为分隔的，可使用 StringSplit 函数进行分割。 (2)访问远程注册表，键形如 "\计算机名\键"，但前提是具有访问权限。</p>
RegWrite 写注册表键值	<p>语法： RegWrite ("键" [, "值" [, "类型" [, 值]]])</p> <p>参数： 键：注册表键项 值：[可选]注册表值项 类型：[可选]写入值的类型 (REG_BINARY、REG_SZ、REG_MULTI_SZ、REG_EXPAND_SZ、REG_QWORD、REG_DWORD) 值：[可选]写入的值</p>

	<p><b>返回值：</b></p> <p>成功：1 失败：0，并设置@error 为非 0 值</p> <p><b>@error 值：</b></p> <ul style="list-style-type: none"> <li>1：无法打开请求的键</li> <li>2：无法打开请求的主键</li> <li>3：无法连接远程注册表</li> <li>-1：无法打开请求的值</li> <li>-2：不支持的值类型</li> </ul> <p><b>备注：</b></p> <ul style="list-style-type: none"> <li>(1) 当写入 REG_MULTI_SZ 类型的值时，多个字符串间使用@LF 进行分隔；</li> <li>(2) 访问远程注册表，键形如 "\\\计算机名\键"，但前提是具有访问权限；</li> <li>(3) 当写入默认值项 (Default) 时，将值参数写为空字符串即可。</li> </ul>
RegDelete 删除注册表键值	<p><b>语法：</b></p> <pre>RegDelete ( "键" [, "值"] )</pre> <p><b>参数：</b></p> <ul style="list-style-type: none"> <li>键：注册表键项</li> <li>值：[可选]注册表值项</li> </ul> <p><b>返回值：</b></p> <p>成功：1 专用：0，键/项不存在 失败：2</p> <p><b>@error 值：</b></p> <ul style="list-style-type: none"> <li>1：无法打开请求的键</li> <li>2：无法打开请求的主键</li> <li>3：无法连接远程注册表</li> <li>-1：无法打开请求的值</li> <li>-2：不支持的值类型</li> </ul> <p><b>备注：</b></p> <ul style="list-style-type: none"> <li>(1) 不写值时，将删除整个注册表键；</li> <li>(2) 值为空字符串时，将删除默认值项 (Default)；</li> <li>(3) 访问远程注册表，键形如 "\\\计算机名\键"，但前提是具有访问权限；</li> <li>(4) 删除注册表项目有风险，请务必谨慎操作。</li> </ul>

## 2、注册表遍历 (RegEnumKey、RegEnumVal)

函数	说明
RegEnumKey 枚举注册表键项	<p><b>语法：</b></p> <pre>RegEnumKey ( "键", 索引 )</pre> <p><b>参数：</b></p> <ul style="list-style-type: none"> <li>键：注册表键项</li> <li>索引：子键索引（从 1 开始，1、2、……表示第 1 个子键、第 2 个子键、……）</li> </ul> <p><b>返回值：</b></p> <p>成功：返回所枚举的子键 失败：返回空字符串，并设置@error 为非 0 值</p>

	<p>@error 值：</p> <ul style="list-style-type: none"> <li>1：无法打开请求的键</li> <li>2：无法打开请求的主键</li> <li>3：无法连接远程注册表</li> <li>-1：无法枚举请求的子键（一般为超出了枚举范围）</li> </ul>
RegEnumVal 枚举注册表值项	<p>语法： RegEnumVal( "键", 索引 )</p> <p>参数：</p> <p>键：注册表值项</p> <p>索引：值项索引（从 1 开始，1、2、……表示第 1 个值项、第 2 个值项、……）</p> <p>返回值：</p> <p>成功：返回所枚举值，并将@extened 设置为值的类型</p> <p>失败：返回空字符串，并设置@error 为非 0 值</p> <p>@error 值：</p> <ul style="list-style-type: none"> <li>1：无法打开请求的键</li> <li>2：无法打开请求的主键</li> <li>3：无法连接远程注册表</li> <li>-1：无法枚举请求的子键（一般为超出了枚举范围）</li> </ul> <p>备注：</p>

### 3、范例

（若已打开注册表编辑器，在使用 Au3 代码修改注册表后，需刷新注册表编辑器才能看到变化）

#### 例 1：写入键值

```
RegWrite('HKEY_LOCAL_MACHINE\SOFTWARE\itk', 'Web', 'REG_SZ', 'http://www.itankong.com/')
```

#### 例 1 效果图：

名称	类型	数据
(默认)	REG_SZ	(数值未设置)
Web	REG_SZ	http://www.itankong.com/

( 图 4-32 )

#### 例 2：读取键值

```
Local $Val = RegRead('HKEY_LOCAL_MACHINE\SOFTWARE\ACC', 'Web')
MsgBox(0, '读注册表', '读到的值为 ' & $Val)
```

#### 例 3：删除键值

```
;删除 ACC 子键下的 Web 值项
RegDelete('HKEY_LOCAL_MACHINE\SOFTWARE\ACC', 'Web')

;删除整个 ACC 子键
RegDelete('HKEY_LOCAL_MACHINE\SOFTWARE\ACC')
```

### 例 4：枚举键

```
#include <Array.au3>

Local $Key
Local $a_Key[1], $p = 1

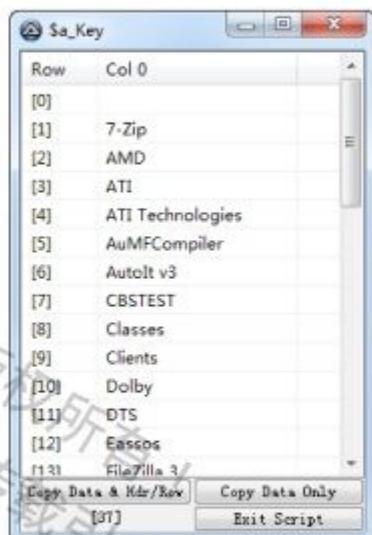
Local $i = 1
While 1
    ;枚举 SOFTWARE 键下的键项
    $Key = RegEnumKey('HKEY_LOCAL_MACHINE\SOFTWARE', $i)

    If @error = -1 Then
        ;如果超出了枚举范围（已枚举完 SOFTWARE 键下的所有键项），则退出循环
        ExitLoop
    Else
        ;否则，将新枚举的子键加入数组
        ReDim $a_Key[$p + 1]
        $a_Key[$p] = $Key
        $p += 1

        ;索引值自增 1，准备枚举下一个
        $i += 1
    EndIf
WEnd

;显示 SOFTWARE 键下的所有键项
_ArrayDisplay($a_Key, '$a_Key')
```

效果图：



(图 4-33)

## 例 5：枚举值

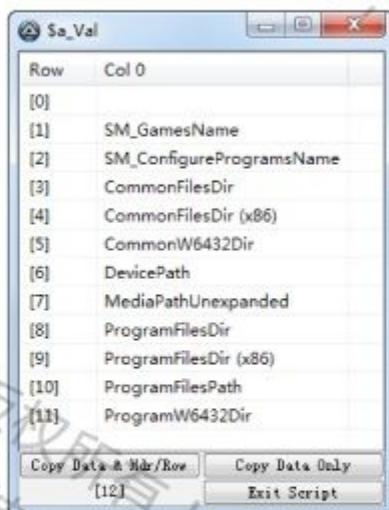
```
#include <Array.au3>

Local $Val
Local $a_Val[1], $p = 1

Local $i = 1
While 1
    ;枚举 CurrentVersion 子键下的所有值项
    $Val = RegEnumVal('HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion', $i)
    If @error = -1 Then
        ;如果超出了枚举范围（已枚举完 CurrentVersion 键下的所有值项），则退出循环
        ExitLoop
    Else
        ;否则，将新枚举的值项加入数组
        ReDim $a_Val[$p + 1]
        $a_Val[$p] = $Val
        $p += 1
    EndIf
    $i += 1
WEnd

;显示 CurrentVersion 子键下的所有值项
_ArrayDisplay($a_Val, '$a_Val')
```

效果图：



( 图 4-34 )

## 4.8 延时计时

### 1、延时等待 ( Sleep )

函数	说明
Sleep 暂停程序的执行	<p>语法： Sleep ( 时间 )</p> <p>参数： 时间：暂停的时间，单位为毫秒 ( 1000 毫秒=1 秒 )</p> <p>返回值： ( 无返回值 )</p> <p>备注： ( 1 ) 最大暂停时间为 2147483647 毫秒 ( 24 天 )； ( 2 ) 最小值为 10 毫秒，设置小于 10 毫秒的值将自动默认为 10 毫秒； ( 3 ) 暂停时间并不是非常精准，有细微误差。</p>

### 2、计时 ( TimerInit、TimerDiff )

函数	说明
TimerInit 初始化计时器 ( 计时起点 )	<p>语法： TimerInit()</p> <p>参数： ( 无参数 )</p> <p>返回值： 返回一个计时起点，用于 TimerDiff 函数进行计时计算</p> <p>备注： “计时起点”是一种特殊的句柄，一般仅用于 TimerDiff 函数，不应作为其他用途，否则将可能存在潜在的错误</p>
TimerDiff 计算与计时起点间的时间间隔	<p>语法： TimerDiff ( 计时起点 )</p> <p>参数： 计时起点：由 TimerInit 函数生成的计时起点</p> <p>返回值： 返回与计时起点间的时间间隔，时间单位为毫秒 ( 1000 毫秒=1 秒 )</p> <p>备注： ( 1 ) TimerDiff 的时间是比较精准的； ( 2 ) 个别 CPU 可能会导致计时错误，但不常见。</p>

### 3、范例

```
; 创建时钟 ( 计时器起点 )
```

```
Local $Timer = TimerInit()
```

```
; 暂停脚本 3000 毫秒 ( 3 秒 )
```

```
Sleep(3000)
```

```
;计算与计时器起点间的时间间隔
Local $Diff = TimerDiff($Timer)
MsgBox(0 + 64, "计时", "脚本实际暂停时间为：" & $Diff & "毫秒")
```

效果图：



(图 4-35)

由此可见，Sleep 的精度比 TimerDiff 稍低。不过上例的计算也不是个精准的例子，因为从脚本脱离暂停状态，到定义\$Diff 变量、获取时间间隔、变量赋值、使用 MsgBox 显示的过程，也是需要一定时间的。

在绝大多数实际操作中，并不在意这零点几毫秒的精度，而且 Sleep 的用途远比计时器更多。

## 4.9 内存信息

### 1、获取内存信息 ( MemGetStats )

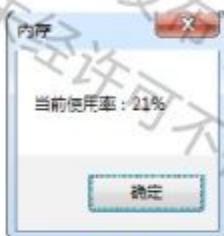
函数	说明
MemGetStats	语法： MemGetStats()
获取内存相关信息	参数： (无参数)
	返回值： 返回一个包含当前内存信息的数组 0 元素：内存使用率 (百分比) 1 元素：物理内存总数 2 元素：物理内存可用数 3 元素：页面文件总数 4 元素：页面文件可用数 5 元素：虚拟内存总数 6 元素：虚拟内存可用数
	备注： 32 位系统 (在不借助拓展寻址的条件下) 最大可识别物理内存数约为 3.25GB，如果有显卡等设备“借”内存为显存的话，则物理内存可用数会更小。

### 2、范例

```
Local $a_MemInfo = MemGetStats()
```

```
 MsgBox(0, '内存', '当前使用率 : ' & $a_MemInfo[0] & '%')
```

效果图：



(图 4-36)

## 4.10 剪切板控制

剪切板是 Windows 系统下的一个缓冲区域，当我们“复制”、“剪切”某段文本时，文本内容就会暂存至剪切板，而之后当我们“粘贴”时，剪切板内的文本内容就会释放到目标编辑器中。值得注意的是，“粘贴”后剪切板中的内容不会被清空，直到下一次再执行“复制”或“剪切”时被自动覆盖。

### 1、写入剪切板、提取剪切板 ( ClipPut、ClipGet )

函数	说明
ClipPut 将文本写入剪切板	语法： <code>ClipPut ( "文本" )</code> 参数： 文本：即将写入剪切板的文本 返回值： 成功：1 失败：0 备注： ( 1 ) 剪切板已有内容将被覆盖； ( 2 ) 内容为空字符串时剪切板内容将被清空。
ClipGet 读取剪切板内的文本内容	语法： <code>ClipGet ()</code> 参数： ( 无 ) 返回值： 成功：返回剪切板的文本内容 失败：将@error 设置为非 0 值 @error 值： 1：剪切板为空 2：剪切板包含非文本条目 3 或 4：无法访问剪切板

## 2、范例

例 1：将文本写入剪切板

```
ClipPut('http://www.itiankong.com/')
```

可以在执行上述代码后，在文本编辑器中选择“粘贴”，即可将文本粘贴至文本编辑器内。

例 2：读取剪切板内的文本

```
Local $r = ClipGet()
MsgBox(0, '剪切板内的文本', $r)
```

MsgBox 会显示剪切板内的文本内容。

## 4.11 系统环境变量

系统环境变量是 Windows 系统的一种全局指代变量，通常用于批处理中指代特定目录。例如 %systemdrive% 指代系统盘（如 C:），%windir% 指代 Windows 安装目录（如 C:\Windows）。我们可以通过 Au3 代码直接获得系统环境变量所指代的值。

### 1、获取环境变量 ( EnvGet )

函数	说明
EnvGet 获取环境变量值	<p>语法： EnvGet ( "环境变量" )</p> <p>参数： 环境变量：系统环境变量的名称（注意参考备注）</p> <p>返回值： 成功：系统环境变量所指代的值 失败：空字符串</p> <p>备注： 注意，“环境变量”参数只需要系统环境变量名即可，例如 "%systemdrive%" 的系统环境变量名为 “systemdrive”，而不包含两侧百分号。</p>

## 2、范例

```
Local $SystemDrive = EnvGet('systemdrive')

MsgBox(0 + 64, '系统环境变量',_
"%systemdrive% 的值为 ' & $SystemDrive")
```

效果图：



(图 4-37)

## 4.12 Level4 总结

Level4 章节，主要以介绍 Au3 的基本函数为主。本章分 11 个小节介绍了各类 Au3 常用函数，这些函数涉及最为常见的 Windows 系统操作。Au3 的函数库包含了各类常用功能，学通这些功能函数会令我们在解决问题时如虎添翼。

4.1 节，我们完整的学习了之前已经接触过的 MsgBox 与 InputBox 函数，此二者是最为基本的数据输入输出方式。其中 MsgBox 有着极高的使用频率，程序运行中的各类提示、报错均会使用到它。MsgBox 的状态参数很多，但不要机械记忆，只记录几个常用的，其余的到用时再查也不迟。

4.2 节，我们学习了对外部程序的调用。有些时候我们的程序不能独立解决一些问题，但这是很正常的情况，我们不可能为了实现压缩解压功能而独自开发一套压缩算法，这就像为了喝牛奶而养头奶牛一样。外部程序可能解决我们的很多问题、实现很多功能，例如文件压缩解压、硬件信息获取等等，Run 和 RunWait 函数有着较高的使用频率，灵活使用它们会令我们的程序更加多样化。（除此之外，DllCall 也是一个调用外部功能的方法，今后我们会着重学习）

4.3 节，我们接触到了驱动器相关函数。驱动器是 Windows 操作系统中的重要概念，Windows 操作系统的目录结构就是基于驱动器的（而不像 Linux 是基于目录的）。本节中我们学会了如何获取驱动器的信息，如状态、类型、卷标、总大小、可用空间等等，这些知识会在我们操作 Windows 文件系统时提供很大帮助。

4.4 节，我们学习了目录（文件夹）相关函数。创建目录、复制目录、删除目录，是我们平时操作目录时的重要过程。结合 4.3 节中的相关支持，我们可以更好的理解“相对路径”、“绝对路径”、“工作目录”等知识。良好的目录操作，是文件操作的基础。（广义上讲，目录也是一种特殊的“文件”）

4.5 节，我们学习了各类对文件操作的方法。这是比较长的一节，列举了文件的复制、移动、删除、存在性检测、属性获取等方法。日常程序中少不了对文件的操作，很多操作也依赖于对文件的操作，所以学好本节是尤其重要的。不过本节并未列举所有文件操作函数，文件的搜索、文件的读写、配置文件的操作等复杂知识，将在后续章节逐步解析。总而言之，文件的操作是复杂而常用的。

4.6 节，我们学习了文件的装载，这是一个初学者经常搞不明白的知识内容，虽然同属于文件的范畴，但单拿出来作为重点进行了讲解。将外部文件包含至当前程序内，在使用时装载至目标位置，是 FileInstall 的重点功能。要理解这个函数的意义，一定要理解在未编译、编译时和编译后的函数概念，这部分已作为重点在 4.6 节中讲述过，如果不熟悉还请回忆并理解消化。

4.7 节，我们学习了注册表的相关操作。注册表是 Windows 系统配置信息的简单数据库，对系

统设置的修改，即是对注册表的修改。注册表的修改最常用的莫过于读、写、删，值得注意的是注册表内的值分为多种不同类型，一定不要搞混类型。另外特别提醒，注册表修改有风险，请谨慎操作。

4.8~4.11节，都是一些简单的却比较好玩的函数，延时、计时、内存信息、剪切板、系统环境变量等等，都是一些和系统息息相关的功能函数，绝大多数使用简单，没有复杂参数。今后关于函数的章，每章最后几节都会介绍一些有意思简单的函数，供大家修正一下因前期学习而过热的头脑！

功能函数会令我们编程更加容易，我们甚至不必了解功能函数的执行过程，即可使用函数实现需要的功能。学好函数，是令程序实现更多功能的重要基石！

# **Level 5**

经过 Level1~Level4 的学习，大家已经顺利的成长为一个 Au3er，能够理解代码基本语法、明确算法思路，并能够书写简单的应用程序。在 Level5，我们将暂停急促的脚步，将之前的知识加以梳理和补充，做到稳扎稳打、循序渐进。

## 5.1 关于数组和循环结构的一些补充

### 5.1.1 数组内再存储数组

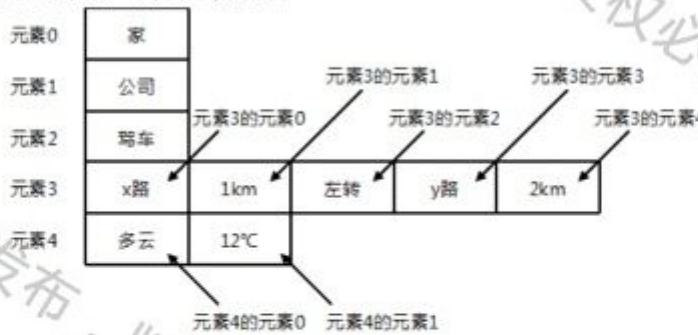
早在 Level2 中就已经学习过数组，与普通变量最大的区别在于数组是一个“数据集合”，数组中通常包含多个数据。一般而言，我们使用数组存储数值型、字符串型、布尔型等数据，但你知道数组内还可以再存储数组吗？

“数组内再存储数组”与“多维数组”是两个不同的概念。以二维数组为例，二维数组更像一张表，通常有 X 行 Y 列，形如：

	列0	列1	列2	列3	列4
行0	ID	UID	金币	技术	互助
行1	Skye	I010	215	51	137
行2	Una	1011	337	85	91
行3	Cman	1012	150	121	19
行4	Sa	1013	271	15	201

(图 5-1)

“数组内再存储数组”则相当于将一个数组放进另一个数组中，使一个数组成为另一个数组的元素（尽管这个元素内包含了多个元素），形如：



(图 5-2)

如上图的一维数组中，元素 0、1、2 均为一般数据，而元素 3 是一个拥有 5 个元素的一维数组、元素 4 是一个拥有 2 个元素的一维数组。

这种在数组内再存储数组的方式，令 Au3 的数组变的更加灵活，可用于存储复杂数据，但这种存储方式也会令程序更易出错、维护更加困难。所以，除非对代码逻辑的把握非常到位，一般不建议采用这种方式。

代码举例：

```
Local $a_Path[] = ["x 路", "1km", "左转", "y 路", "2km"]
Local $a_Weather[] = ["多云", "12°C"]

Local $a_Navigate[] = ["家", "公司", "驾车", $a_Path, $a_Weather]

MsgBox(0, "", ($a_Navigate[3])[4]) ; -> 输出：“2km”
MsgBox(0, "", ($a_Navigate[4])[1]) ; -> 输出：“12°C”
```

运行结果：



特别注意：

(1) “(\$a\_Navigate[3])[4]” 的含义是一维数组 \$a\_Navigate 的元素[3]的元素[4]；

(2) “\$a\_Navigate[3][4]” 的含义是二维数组 \$a\_Navigate 的元素[3][4]。

二者的含义不同。如果在上例中采用(2)的写法，那么 \$a\_Navigate 会被认为是一个二维数组，但是实际上它是一个一维数组，此时程序会报错，不能正常运行。

提醒一点，这种特殊的数组形式，\_ArrayDisplay 函数不能展示其子数组元素值。

“数组内再存储数组”与“多维数组”虽是两个不同的概念，但殊途同归，“数组内再存储数组”恰恰在一定程度上解释了“多维数组”的真正含义。

假设建立一个一维数组，它的每一个元素又都是一个一维数组，每个成为其元素的一维数组元素数又相同，那么，从数据的角度讲，这个“大”一维数组的含义与二维数组其实并没有不同。

以二维数组存储数据：

```
Local $a_Array[][] = [[{"ID": "UID", "金币": "技术", "技术": "互助"}, _  
    ["Skye", 1010, 215, 51, 137], _  
    ["Una", 1011, 337, 85, 91], _  
    ["Gman", 1012, 150, 121, 19], _  
    ["Sa", 1013, 271, 15, 201]]  
  
MsgBox(0, "", $a_Array[2][0] & "的" & $a_Array[0][2] & "为" & $a_Array[2][2])
```

在一维数组内存储多个一维数组数据：

```
Local $a_SubArray0[] = ["ID", "UID", "金币", "技术", "互助"]  
Local $a_SubArray1[] = ["Skye", 1010, 215, 51, 137]  
Local $a_SubArray2[] = ["Una", 1011, 337, 85, 91]  
Local $a_SubArray3[] = ["Gman", 1012, 150, 121, 19]  
Local $a_SubArray4[] = ["Sa", 1013, 271, 15, 201]  
  
Local $a_Array[] = [$a_SubArray0, $a_SubArray1, $a_SubArray2, $a_SubArray3, $a_SubArray4]  
MsgBox(0, "", ($a_Array[2])[0] & "的" & ($a_Array[0])[2] & "为" & ($a_Array[2])[2])
```

运行结果均为：



(图 5-5)

从结构的角度讲，一个一维数组，如果每个元素都是一个等长的一维数组，那么它就是“二维数组”，类推，如果其每个元素又是一个等长的一维数组，那么它就是“三维数组”，继续类推，就形成了更高维数的数组。

不过仍旧提醒大家的是，“数组内再存储数组”所形成的“多维数组”与直接声明的多维数组在数据调用方面是不同的，即刚才讲到的“`($a_Navigate[3])[4]`”与“`$a_Navigate[3][4]`”的区别，这点请务必牢记于心。

另外，如果一个数组内存储大量的其他数组，会影响执行效率，请谨慎使用。

### 5.1.2 For.....In.....Next 循环

我们在 Level1 中学习过 For...To...Step...Next 循环，并在之后章节中频繁使用，这种循环方式使用频率极高、使用场景广泛。但它并非是唯一的一种 For 循环。

下面我们来学习一下 For.....In.....Next 循环。

**语法：**

```
For <变量> In <数组/对象>
    <语句段>
Next
```

**功能：**

每次循环，变量将被赋值为数组（或对象）中的一个元素值，直至全部遍历完毕。

（对象也是一种多数据集合，但不同于数组的数据集合方式，更高 Level 中将进行学习）

**说明：**

（1）被遍历目标必须是至少具有一个元素的数组（或对象），否则循环将被跳过不执行；

（2）被遍历目标是多维数组时，循环将被跳过不执行。

（换句话说，For.....In.....Next 循环只能用于一维数组的遍历）

**举例：**

```
Local $a_Array[] = [0, 1, 2, 3, 4, 5]
Local $Value
For $Value In $a_Array
    MsgBox(0, "", $Value); <- 依次使用 MsgBox 输出 0、1、2、……、5
Next
```

一般而言，For.....In.....Next 主要被用于遍历对象，而不是数组。但 For.....In.....Next 的确可

以更为方便的遍历一维数组，灵活使用还是非常方便的。

`For.....In.....Next` 与 `For...To...Step...Next` 最大的区别在于对于数组元素的引用，前者是通过一个变量获取了数组元素的值，而后者则是直接使用的数组元素。前者对于引用后的值的变化并不会直接修改原数组。

举例说明，刚才的例子如果使用 `For...To...Step...Next` 来实现是这样的：

```
Local $a_Array[] = [0, 1, 2, 3, 4, 5]
Local $Value
Local $i
For $i = 0 To UBound($a_Array, 1) - 1
    $Value = $a_Array[$i]
    MsgBox(0, "", $Value); <- 依次使用 MsgBox 输出 0、1、2、……、5
Next
```

仔细观察，每次循环 `$Value` 只是被赋予了一个元素的值，而 `$Value` 并非是数组元素本身，数组元素本身是 `$a_Array[$i]`。换句话说，`$a_Array[$i]` 只是把自己的值给了 `$Value`，所以 `$Value` 的任何变更并不会影响到 `$a_Array[$i]`，这就像函数的值传递一样。

对于 `For.....In.....Next` 与 `For...To...Step...Next` 的区别，大家应对比上述两例，加深理解，避免出错。

### 5.1.3 ExitLoop

有时，我们需要在遇到某些条件时退出当前循环，这种情况使用 `ExitLoop` 来实现。

**语法：**

```
ExitLoop [循环等级]
```

**功能：**

退出循环

**说明：**

“循环等级”为可选参数，默认为 1（即当前循环）。如果循环有多层嵌套，例如一个循环内嵌套一个循环（即双层循环），可设置 `ExitLoop` 的循环等级为 2，就可以一次性跳出这个双层循环了。但仍建议保持“循环等级”为默认值，一次只跳出一层循环，从而避免不必要的麻烦。

**举例：**

```
Local $sum = 0
Local $i
For $i = 1 To 100
    $sum += 1
    If $sum >= 50 Then ExitLoop
Next
MsgBox(0, "", $sum); <- 输出 50
```

上例中，按照设定 `For` 循环应该执行 100 次，而 `$sum` 的最终值应该为 100。但因为在循环内进行了判定，当 `$sum` 的值大于等于 50 时就会跳出循环，所以循环实际上只执行了 50 次，而 `$sum` 的最终值也只有 50。

特别注意一定不要滥用 ExitLoop，尤其是不要随意跳出多重循环，否则将造成代码查错困难。

## 5.2 数组的常见算法

数组是最为常见的数据集合，实际应用中很多数据都是以数组的方式进行存储和调用的。本节将对数种常用的数组算法进行解析，让大家更为深刻的体会一下数组的操控方式。

当然，本节中很多方法都可以通过 UDF 数组相关函数来实现，但大家不要知其然却不知其所以然，认真学习本节，理解的过程就是进步的过程。

### 5.2.1 插入

#### 1、向一维数组插入一个数据

实现“将数据插入数组”，以简单的一维数组为例进行分析。

(1) 将一个数据插入数组，那么现有的数组必须要增加一个元素的空间。拓展数组大小在 Au3 中是比较简单的，只要使用 ReDim 重新定义数组就可以了。

(2) 但 ReDim 只能在数组的尾部增加元素空间，而要插入数据的位置却不一定在尾部，还可能在数组的头部、数组的中部。所以，使用 ReDim 在数组尾部增加元素空间后，还需要“移动”现有元素的数据，从而为即将插入的数据“留空”。

通过图示来更好的理解一下：

建立一个有 5 个元素的一维数组，并为元素依次赋值为 A、B、C、D、E

A	B	C	D	E	
---	---	---	---	---	--

(图 5-6)

要插入一个数据，需要先将数组拓展出一个元素空间。使用 ReDim 拓展数组空间时，所拓展的空间在数组的尾部。

A	B	C	D	E	
---	---	---	---	---	--

(图 5-7)

如果将数据插入数组尾部，直接将待插入数据放入尾部元素内即可：

A	B	C	D	E	X
---	---	---	---	---	---

(图 5-8)

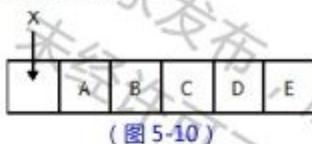
如果将数据插入数组头部，则需：

(1) 先将所有数据后移一个元素位置：

	A	B	C	D	E
--	---	---	---	---	---

(图 5-9)

(2) 再将待插入数据放入头部元素内：

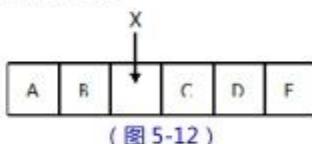


如果将数据插入数组中部，则需：

(1) 先将目标位置及之后的数据后移一个元素位置：



(2) 再将待插入数据放入目标元素内：



综上，在数组中插入数据的算法分三步：

增加元素位置，移动已有数据（如果需要），将待插入数据放至目标位置。

接下来，我们用代码实现向一维数组中插入一个数据：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

; 向元素 5 插入数据 1
Func _Main()
Local $a_Array[] = [5, 7, 0, 6, 3, 9, 2, 4, 8]; <- 一维数组

Local $Range = 5; <- 插入位置为 5
Local $Value = 1; <- 插入数据为 1
_ArrayDisplay($a_Array, "$a_Array - 原始")
_SK_ArrayInsert_1D($a_Array, $Range, $Value)
Switch @error
Case 0
_ArrayDisplay($a_Array, "$a_Array - 插入")
Case 1
MsgBox(0 + 16, "错误", "目标数组并非数组")
Case 2
MsgBox(0 + 16, "错误", "目标数组并非一维数组")
Case 3
MsgBox(0 + 16, "错误", "目标位置不是数字")
Case 4
```

```
        MsgBox(0 + 16, "错误", "目标位置超出了数组范围")
EndSwitch
EndFunc  ;==>_Main

;
; 函数：
; _SK_ArrayInsert_1D
; 功能：
; 向一维数组中插入一个数据
; 参数：
; $a_Array，被插入的数组
; $Range，插入的位置
; $Value，插入的数据
; 返回值：
; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error：
;     1，目标数组并非数组
;     2，目标数组并非一维数组
;     3，目标位置不是数字
;     4，目标位置超出了数组范围
;

Func _SK_ArrayInsert_1D(ByRef $a_Array, $Range, $Value)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, 0)
    If Not (IsNumber($Range)) Then Return SetError(3, 0, 0)
    If Not ($Range >= 0 And $Range <= UBound($a_Array, 1)) Then Return SetError(4, 0, 0)

    ;使数组增加一个元素
    ReDim $a_Array[UBound($a_Array, 1) + 1]

    ;自目标位置开始，所有数据向后移动一个元素
    Local $i
    For $i = UBound($a_Array, 1) - 1 To $Range + 1 Step -1
        $a_Array[$i] = $a_Array[$i - 1]
    Next

    ;将待插入数据插入目标位置
    $a_Array[$Range] = $Value

    Return SetError(0, 0, 1)
EndFunc  ;==>_SK_ArrayInsert_1D
```

运行图：



(图 5-13)

(图 5-14)

如上图所示，我们成功的向一个一维数组的第 5 个位置插入了数据 1。

\_SK\_ArrayInsert\_1D 是一个自定义函数，功能是向一维数组指定位置插入一个数据。这个函数可以满足向数组头部、中部、尾部插入数据的要求，上例只举例中部，有兴趣可以改改代码测试向头部或尾部插入一个数据。

本例除了和大家讨论“向一维数组插入数据”，还示例了如何书写一个良好的自定义函数：

- (1) 良好的自定义函数应具有带有明确含义的函数名，且函数名具有公共性，不易与代码内的其他函数重名或混淆；
- (2) 良好的自定义函数应具有完善的函数功能说明、参数说明和返回值说明，以备在今后的使用中能快速的了解函数意义和使用方法；
- (3) 良好的自定义函数应具有充分的检测机制，能尽可能周全的考虑到用户在使用函数时可能遇到的各种错误，并将错误以返回值或@error 值的方式传递给主调用函数；
- (4) 良好的自定义函数应具有高效的执行效率，优化逻辑、减少冗余。

大家在制作自定义公共函数时务必遵循上述几点要求。

## 2、向二维数组插入“一行”数据

在学习了如何向一维数组插入一个数据后，下面来扩展怎么向二维数组插入“一行”数据。

就“插入”这个算法而言，一维数组和二维数组的算法基本相同，均遵循“增加元素位置，移动已有数据（如果需要），将待插入数据放至目标位置”的三步骤。而与一维数组不同的是，二维数组的数据是按行进行移动或插入的，这一点只需稍加改动就可以实现。

向二维数组插入一行数据：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit
```

```
; 向二维数组的第 6 行 (0 基, 行号为 5) 插入一行数据
Func _Main()
    Local $a_Array[] = [{"编号", "姓名", "性别", "年龄", "表达", "逻辑", "专业"},_
        {"06", "李峰", "男", "25", "81", "85", "79"},_
        {"07", "张硕", "男", "27", "75", "93", "91"},_
        {"04", "赵明", "男", "21", "93", "73", "85"},_
        {"05", "孙伟", "男", "26", "90", "82", "67"},_
        {"10", "谢娜", "女", "25", "87", "71", "83"},_
        {"03", "顾岚", "女", "22", "97", "87", "95"},_
        {"09", "马丽", "女", "20", "91", "67", "72"},_
        {"01", "王杨", "男", "26", "83", "91", "93"},_
        {"08", "刘超", "男", "23", "90", "93", "97"}] ; <- 二维数组

    Local $Range = 5 ; <- 插入的位置
    Local $Value = "02|韩雪|女|20|86|75|91" ; <- 插入的数据
    _ArrayDisplay($a_Array, "$a_Array - 原始")
    _SK_ArrayInsert_2D($a_Array, $Range, $Value)
    Switch @error
        Case 0
            _ArrayDisplay($a_Array, "$a_Array - 插入")
        Case 1
            MsgBox(0 + 16, "错误", "目标数组并非数组")
        Case 2
            MsgBox(0 + 16, "错误", "目标数组并非二维数组")
        Case 3
            MsgBox(0 + 16, "错误", "目标位置不是数字")
        Case 4
            MsgBox(0 + 16, "错误", "目标位置超出了数组范围")
        Case 5
            MsgBox(0 + 16, "错误", "要插入的数据与目标数组列数不同")
    EndSwitch
EndFunc ;==> _Main

;
; 函数：
; _SK_ArrayInsert_2D
; 功能：
; 向二维数组中插入一行数据
; 参数：
; $a_Array，被插入的数组
; $Range，插入的位置
; $Value，插入的数据（字符串，以“|”进行分隔）
; 返回值：
; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error :
```

```

; 1, 目标数组并非数组
; 2, 目标数组并非二维数组
; 3, 目标位置不是数字
; 4, 目标位置超出了数组范围
; 5, 要插入的数据与目标数组列数不同
;

Func _SK_ArrayInsert_2D(ByRef $a_Array, $Range, $Value)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 2 Then Return SetError(2, 0, 0)
    If Not (IsNumber($Range)) Then Return SetError(3, 0, 0)
    If Not ($Range >= 0 And $Range <= UBound($a_Array, 1)) Then Return SetError(4, 0, 0)

    Local $a_InsertArray = StringSplit($Value, "|", 2)
    If UBound($a_InsertArray, 1) <> UBound($a_Array, 2) Then Return SetError(5, 0, 0)

    ;使数组增加一行
    ReDim $a_Array[UBound($a_Array, 1) + 1][UBound($a_Array, 2)]

    ;自目标行开始，所有数据向后移动一行
    Local $i, $j
    For $i = UBound($a_Array, 1) - 1 To $Range + 1 Step -1
        ;整行移动
        For $j = 0 To UBound($a_Array, 2) - 1
            $a_Array[$i][$j] = $a_Array[$i - 1][$j]
        Next
    Next

    ;将待插入数据插入目标行
    For $j = 0 To UBound($a_Array, 2) - 1
        $a_Array[$Range][$j] = $a_InsertArray[$j]
    Next

    Return SetError(0, 0, 1)
EndFunc  ;==> _SK_ArrayInsert_2D

```

运行图：

## Let's AutoIt Plus

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
[0]	编号	姓名	性别	年龄	表达	逻辑	专业
[1]	06	李峰	男	25	81	85	79
[2]	07	张硕	男	27	75	93	91
[3]	04	赵明	男	21	93	73	85
[4]	05	孙伟	男	26	90	82	67
[5]	10	谢娜	女	25	87	71	83
[6]	03	顾嵩	女	22	97	87	95
[7]	09	马丽	女	20	91	67	72
[8]	01	王杨	男	26	83	91	93
[9]	08	刘超	男	23	90	93	97

( 图 5-15 )

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
[0]	编号	姓名	性别	年龄	表达	逻辑	专业
[1]	06	李峰	男	25	81	85	79
[2]	07	张硕	男	27	75	93	91
[3]	04	赵明	男	21	93	73	85
[4]	05	孙伟	男	26	90	82	67
[5]	02	韩雪	女	20	86	75	91
[6]	10	谢娜	女	25	87	71	83
[7]	03	顾嵩	女	22	97	87	95
[8]	09	马丽	女	20	91	67	72
[9]	01	王杨	男	26	83	91	93
[10]	08	刘超	男	23	90	93	97

( 图 5-16 )

如上图所示，“02|韩雪|女|20|86|75|91”这行数据被成功的插入了二维数组第 6 行（行号为 5）。

仔细观察代码，与一维数组插入数据最大的不同点在于：

( 1 ) 一维数组是一次移动一个数据：

```
Local $i  
For $i = UBound($a_Array, 1) - 1 To $Range + 1 Step -1  
    $a_Array[$i] = $a_Array[$i - 1]  
Next
```

而二维数组是一次移动一整行数据：

```
Local $i, $j  
For $i = UBound($a_Array, 1) - 1 To $Range + 1 Step -1  
    For $j = 0 To UBound($a_Array, 2) - 1  
        $a_Array[$i][$j] = $a_Array[$i - 1][$j]  
    Next  
Next
```

( 2 ) 一维数组是插入一个数据：

```
$a_Array[$Range] = $Value
```

而二维数组则是插入一行数据：

```
For $j = 0 To UBound($a_Array, 2) - 1
    $a_Array[$Range][$j] = $a_InsertArray[$j]
Next
```

从本例大家可以看出，算法具有极大的通用性，虽然数组性质不同、插入的数据量不同，但算法相通。根据基本算法灵活衍生出应对问题的新算法，才能将算法真正为我所用。

## 5.2.2 删除

### 1、从一维数组中删除一个数据

实现“将数据从数组删除”，以简单的一维数组为例进行分析。

- (1) 抹除一个数据容易，将这个元素赋值为空字符串，就相当于“消灭”了元素中存储的数据；
- (2) 数据被抹除了还不算完成任务，数组还得减少一个数组元素，可以通过 ReDim 重定义数组以减少元素数量，但 ReDim 只能从数组尾部减少元素，而我们要删除的数据不一定位于数组尾部，还可能在头部、中部；
- (3) 如果数据不位于尾部，则必须先移动现有数据，而后才能重定义数组，否则数组尾部的数据将会丢失。

通过图示来更好的理解一下：

建立一个有 5 个元素的一维数组，并为元素依次赋值为 A、B、C、D、E

A	B	C	D	E
---	---	---	---	---

( 图 5-17 )

如果要删除的数据位于数组的尾部，直接使用 ReDim 减少数组尾部的一个元素位置即可：

A	B	C	D
---	---	---	---

( 图 5-18 )

如果要删除的数据位于数组的中部（例如 C），则需要：

- (1) 将 C 之后的数据向前移动一个元素位置：

A	B	D	E	F
---	---	---	---	---

( 图 5-19 )

- (2) 使用 ReDim 减少数组尾部的一个元素位置：

A	B	D	E
---	---	---	---

( 图 5-20 )

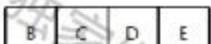
如果要删除的数据位于数组的头部（例如 A），则需要：

- (1) 将 A 之后的数据向前移动一个元素位置：

B	C	D	E	F
---	---	---	---	---

( 图 5-21 )

- (2) 使用 ReDim 减少数组尾部的一个元素位置：



(图 5-22)

综上，删除数组中数据的算法分两步：

移动已有数据（如果需要），减少元素位置。

接下来，我们用代码实现删除一维数组中的一个数据：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

; 删除一维数组中元素 5 的数据
Func _Main()
    Local $a_Array[] = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]; <- 一维数组

    Local $Range = 5; <- 删除位置为 5

    _ArrayDisplay($a_Array, "$a_Array - 原始")
    _SK_ArrayDelete_1D($a_Array, $Range)
    Switch @error
        Case 0
            _ArrayDisplay($a_Array, "$a_Array - 删除")
        Case 1
            MsgBox(0 + 16, "错误", "目标数组并非数组")
        Case 2
            MsgBox(0 + 16, "错误", "目标数组并非一维数组")
        Case 3
            MsgBox(0 + 16, "错误", "目标位置不是数字")
        Case 4
            MsgBox(0 + 16, "错误", "目标位置超出了数组范围")
    EndSwitch
EndFunc ;==> _Main

; -----
; 函数：
;   _SK_ArrayDelete_1D
; 功能：
;   删除一维数组中的一个数据
; 参数：
;   $a_Array, 目标数组
;   $Range, 要删除的位置
; 返回值：
```

```

; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error :
;   1，目标数组并非数组
;   2，目标数组并非一维数组
;   3，目标位置不是数字
;   4，目标位置超出了数组范围
;-----

Func _SK_ArrayDelete_1D(ByRef $a_Array, $Range)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, 0)
    If Not (IsNumber($Range)) Then Return SetError(3, 0, 0)
    If Not ($Range >= 0 And $Range < UBound($a_Array, 1)) Then Return SetError(4, 0, 0)

    ;自目标位置之后的一个数据开始，所有数据向前移动一个元素
    Local $i
    For $i = $Range + 1 To UBound($a_Array, 1) - 1
        $a_Array[$i - 1] = $a_Array[$i]
    Next

    ;将数组缩减一个元素
    ReDim $a_Array[UBound($a_Array, 1) - 1]

    Return SetError(0, 0, 1)
EndFunc  ;==> _SK_ArrayDelete_1D

```

运行图：

Sa_Array - 原始	
Row	Col 0
[0]	0
[1]	1
[2]	2
[3]	3
[4]	4
[5]	5
[6]	6
[7]	7
[8]	8
[9]	9
<input type="button" value="Copy Data &amp; Hdr/Row"/> <input type="button" value="Copy Data Only"/>	
<input type="button" value="Exit Script"/> [10]	

( 图 5-23 )

Sa_Array - 剔除	
Row	Col 0
[0]	0
[1]	1
[2]	2
[3]	3
[4]	4
[5]	6
[6]	7
[7]	8
[8]	9
<input type="button" value="Copy Data &amp; Hdr/Row"/> <input type="button" value="Copy Data Only"/>	
<input type="button" value="Exit Script"/> [9]	

( 图 5-24 )

如上图所示，我们成功的将一维数组元素 5 的数据删除掉了。

## 2、从二维数组中删除“一行”数据

在学习了如何从一维数组删除一个数据后，下面来扩展如何从二维数组中删除“一行”数据。

就“删除”这个算法而言，一维数组和二维数组的算法基本相同，均遵循“移动已有数据（如果需要），减少元素位置”的两步骤。而与一维数组不同的是，二维数组的数据是按行进行移动或删除的，

## Let's AutoIt Plus

这一点只需稍加改动就可以实现。

从二维数组删除一行数据：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

; 删除二维数组第 6 行 (0 基, 行号为 5) 一行数据
Func _Main()
Local $a_Array[] = [{"序号", "时间", "IP 来源", "行为", "处理"},_
["01", "2014-11-5 17:35", "10.3.15.247", "SQL 注入", "阻断"],_
["02", "2014-11-6 02:17", "10.36.185.2", "文件注入", "阻断"],_
["03", "2014-11-6 09:01", "10.201.1.233", "扫描", "忽略"],_
["04", "2014-11-6 16:31", "10.0.0.254", "扫描", "忽略"],_
["05", "2014-11-7 05:22", "10.61.17.189", "DDoS", "阻断"],_
["06", "2014-11-7 23:51", "10.221.37.196", "文件注入", "阻断"]]; <- 二维数组

Local $Range = 5; 删除的位置
_ArrayDisplay($a_Array, "$a_Array - 原始")
_SK_ArrayDelete_2D($a_Array, $Range)
Switch @error
Case 0
_ArrayDisplay($a_Array, "$a_Array - 删除")
Case 1
MsgBox(0 + 16, "错误", "目标数组并非数组")
Case 2
MsgBox(0 + 16, "错误", "目标数组并非二维数组")
Case 3
MsgBox(0 + 16, "错误", "目标位置不是数字")
Case 4
MsgBox(0 + 16, "错误", "目标位置超出了数组范围")
EndSwitch
EndFunc ;==> _Main

; -----
; 函数：
; _SK_ArrayDelete_2D
; 功能：
;   删除二维数组中的一行数据
; 参数：
;   $a_Array，目标数组
;   $Range，要删除的位置
; 返回值：
```

```

; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error :
;   1，目标数组并非数组
;   2，目标数组并非二维数组
;   3，目标位置不是数字
;   4，目标位置超出了数组范围
;-----

Func _SK_ArrayDelete_2D(ByRef $a_Array, $Range)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 2 Then Return SetError(2, 0, 0)
    If Not (IsNumber($Range)) Then Return SetError(3, 0, 0)
    If Not ($Range >= 0 And $Range < UBound($a_Array, 1)) Then Return SetError(4, 0, 0)

    ;自目标位置之后的一个行开始，所有数据向前移动一行
    Local $i, $j
    For $i = $Range + 1 To UBound($a_Array, 1) - 1
        ;整行移动
        For $j = 0 To UBound($a_Array, 2) - 1
            $a_Array[$i - 1][$j] = $a_Array[$i][$j]
        Next
    Next

    ;将数组缩减一行
    ReDim $a_Array[UBound($a_Array, 1) - 1][UBound($a_Array, 2)]

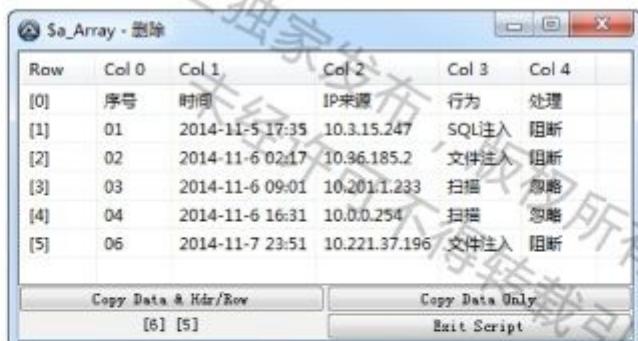
    Return SetError(0, 0, 1)
EndFunc  ;==> _SK_ArrayDelete_2D

```

运行图：

Row	Col 0	Col 1	Col 2	Col 3	Col 4
[0]	序号	时间	IP来源	行为	处理
[1]	01	2014-11-5 17:35	10.3.15.247	SQL注入	阻断
[2]	02	2014-11-6 02:17	10.36.185.2	文件注入	阻断
[3]	03	2014-11-6 09:01	10.201.1.233	扫描	忽略
[4]	04	2014-11-6 16:31	10.0.0.254	扫描	忽略
[5]	05	2014-11-7 05:22	10.61.17.189	DDoS	阻断
[6]	06	2014-11-7 23:51	10.221.37.196	文件注入	阻断

( 图 5-25 )



(图 5-26)

如上图所示，我们成功的删除了二维数组的第 6 行（行号为 5）一行数据。

请大家对比对一维和二维数组处理的不同，删除单个数据与删除一行数据的不同。与 5.2.1 节的对比类似，此处不再赘述。

### 5.2.3 查找

“查找”是数组的重要算法之一，一般用于在数组中查询特定元素的索引。本节介绍两种常见的数组查找算法，供大家学习参考。

#### 1、顺序查找法

在一个数组中查找某个数据，最简单的方法就是从头到尾的将这个数组遍历一遍，如果某个元素值等于被查找的数据，则这个数据存在于这个数组中，并由此可得到数据的元素索引。

依照这个算法，我们通过代码实现在一个一维数组中查找某个数据：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

; 查找一个一维数组中是否有 6 这个数据，并得到其元素索引
Func _Main()
    Local $a_Array[] = [5, 7, 0, 6, 3, 9, 2, 4, 8]; <- 一维数组
    _ArrayDisplay($a_Array, "$a_Array")

    Local $Value = 6; <- 要查找的数据
    Local $Index = _SK_ArraySearch_1D($a_Array, $Value)
    Switch @error
        Case 0
            MsgBox(0 + 64, "信息", "在数组中找到目标值，索引为：" & $Index)
        Case 1
            MsgBox(0 + 16, "错误", "目标数组并非数组")
```

```

Case 2
    MsgBox(0 + 16, "错误", "目标数组并非一维数组")
Case 3
    MsgBox(0 + 16, "错误", "起始位置超出数组范围")
Case 4
    MsgBox(0 + 16, "错误", "结束位置超出数组范围")
Case 5
    MsgBox(0 + 16, "错误", "起始位置大于结束位置")
Case 6
    MsgBox(0 + 64, "信息", "没有在数组内找到目标值")
EndSwitch
EndFunc  ;==>_Main

; -----
; 函数：
; _SK_ArraySearch_1D
; 功能：
; 在一个一维数组中查找目标数据
; 参数：
; $a_Array，目标数组
; $Value，目标数据
; $Start，指定从哪个索引开始向后查找，默认值为 0
; $End，指定查找到哪个索引位置，默认值为 0
; (当$End 参数为 0 时，$End 的值将自动等于数组的最终索引)
; 返回值：
; 成功返回元素索引，失败返回-1
; 失败时设置$error 值为非 0，$error：
; 1，目标数组并非数组
; 2，目标数组并非一维数组
; 3，起始位置超出数组范围
; 4，结束位置超出数组范围
; 5，起始位置大于结束位置
; 6，没有在数组内找到目标值
; -----

Func _SK_ArraySearch_1D(Const ByRef $a_Array, $Value, $Start = 0, $End = 0)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, -1)
    If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, -1)
    If $End = 0 Then $End = UBound($a_Array, 1) - 1
    If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, -1)
    If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, -1)
    If Not ($Start <= $End) Then Return SetError(5, 0, -1)

    Local $i
    Local $Index = -1
    ;遍历数组，逐个查找

```

```

For $i = 1 To UBound($a_Array, 1) - 1
    If $a_Array[$i] = $Value Then
        ;当找到匹配元素后，$Index 被赋值为目标元素索引
        $Index = $i
        ;退出循环，结束查找
        ExitLoop
    EndIf
Next

If $Index >= 0 Then
    ;找到
    Return SetError(0, 0, $Index)
Else
    ;未找到
    Return SetError(6, 0, -1)
EndIf
EndFunc  ;==> _SK_ArraySearch_1D

```

运行图：

Row	Col 0
[0]	5
[1]	7
[2]	0
[3]	6
[4]	3
[5]	9
[6]	2
[7]	4
[8]	8

(图 5-27)



(图 5-28)

如上图所示，我们成功的在数组中找到了值为 6 的元素，并输出其索引为 3。

顺序查找法，是一种最为简单的查找算法，最大的好处就是实现简单，而最大问题是查找效率偏低。由于顺序查找法需要从头至尾的遍历整个数组，如果数组包含 1000 个元素，而目标是第 900 个元素，则要查找 900 次才能找到结果，类推，则数组越大顺序查找法的效率越低。另外，顺序查找法的平均性能差，假设第 1 个数据就是，则只需要查找一次，而假设第 999 个数据才是，则需要查找 999 次，效率靠运气，忽高忽低。所以，一般顺序查找法只适用于数据量较少的数组。

## 2、折半查找法（二分查找法）

折半查找法，又称为二分查找法，优点是效率高、查找次数少、平均性能优，缺点是要求目标数组必须为有序数组。什么是有序数组？按照一定规则排序的数组称为有序数组，例如数字从小到大、字母从 A 到 Z、人名按字母先后、日期时间递增等。在有序数组中，折半查找法的效率是非常高的。

那什么叫“折半”？

假设已一个有序的一维数组：`$a_Array`；

定义`$Low`，初值为数组最小索引；

定义`$High`，初值为数组最大索引；

定义`$Mid`；

在`$Low <= $High`的前提下，

令`$Mid=Int(($Low+$High)/2)`；（`$Mid`如果带有小数则忽略小数部分，它是索引必须为整数）

如果`$a_Array[$Mid]`=目标值，那么

    目标值找到，跳出循环，`$Mid`为目标值索引；

否则，如果 目标值>`$a_Array[$Mid]`，那么

`$Low=$Mid+1`；（即查找索引的下限提升为当前的中间值+1）

否则（即 目标值<`$a_Array[$Mid]`）

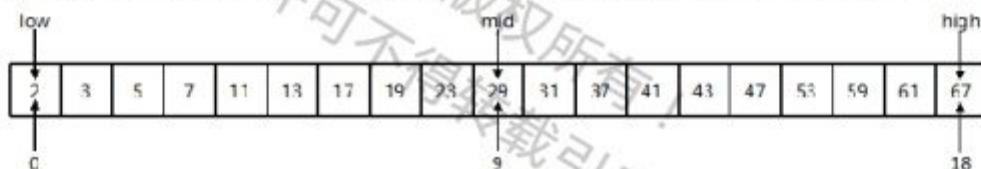
`$High=$Mid-1`；（即查找索引的上限降低为当前的中间值-1）

继续循环，直到找到目标值，或打破循环的前提

依据上述算法，反复折半后，最终查找到目标值。

下面我们更为直观的观察一下折半查找法的过程。下图是一个包含 19 个数值元素的有序一维数组，目标数据为“19”，利用折半查找法查找其索引。

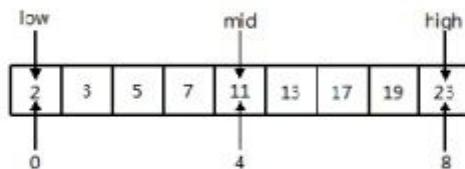
令`$Low` 为最小元素索引 0，令`$High` 为最大元素索引 18，`$Mid` 为 $(0+18)/2$  即 9。



（图 5-29）

由于`$a_Array[$Mid]`即`$a_Array[9]`为 29，而目标值 19<29，则`$High=$Mid-1=9-1=8`，即上限索引降低为 8。

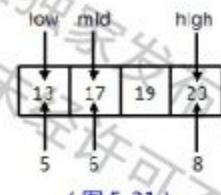
那么数组的查找范围就被缩小到了索引 0~索引 8 之间，此时`$Low` 为 0，`$High` 为 8，`$Mid` 为 $(0+8)/2$  即 4。



（图 5-30）

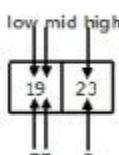
由于`$a_Array[$Mid]`即`$a_Array[4]`为 11，而目标值 19>11，则`$Low=$Mid+1=4+1=5`，即下限索引提升为 5。

那么数组的查找范围被进一步缩小到了索引 5~索引 8 之间，此时`$Low` 为 5，`$High` 为 8，`$Mid` 为 $(5+8)/2$  即 6.5，出现小数时舍弃小数部分，即`$Mid` 的值为 6。



由于 \$a\_Array[\$Mid] 即 \$a\_Array[6] 为 17，而目标值 19 > 17，则 \$Low = \$Mid + 1 = 6 + 1 = 7，即下限索引提升为 7。

那么数组的查找范围被进一步缩小到了索引 7~索引 8 之间，此时 \$Low 为 7，\$High 为 8，\$Mid 为  $(7+8)/2$  即 7.5，出现小数时舍弃小数部分，即 \$Mid 的值为 7。



(图 5-32)

此时 \$a\_Array[\$Mid] 即 \$a\_Array[7] 为 19，与目标值相等，所以成功查找到了目标值，其索引为 7（即 \$Mid 当前的索引）。至此，折半查找法结束。

由此可见，折半查找法在逐步缩小查找范围，随着查找范围的缩小逐渐靠近目标值。很明显，折半查找法的查找效率远高于顺序查找法。

将刚才的例子用代码实现：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

;使用折半查找法在有序数组中找到目标值 19
Func _Main()
Local $a_Array[] = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29,_
31, 37, 41, 43, 47, 53, 59, 61, 67]; <- 一维有序数组
_ArrayDisplay($a_Array, "$a_Array")

Local $Value = 19; <- 目标值
Local $Index = _SK_ArrayBinarySearch_1D($a_Array, $Value)
Switch @error
Case 0
    MsgBox(0 + 64, "信息", "在数组中找到目标值，索引为：" & $Index)
Case 1
    MsgBox(0 + 16, "错误", "目标数组并非数组")
Case 2
    MsgBox(0 + 16, "错误", "目标数组并非一维数组")
```

```

Case 3
    MsgBox(0 + 16, "错误", "起始位置超出数组范围")
Case 4
    MsgBox(0 + 16, "错误", "结束位置超出数组范围")
Case 5
    MsgBox(0 + 16, "错误", "起始位置大于结束位置")
Case 6
    MsgBox(0 + 64, "信息", "没有在数组内找到目标值")
EndSwitch
EndFunc  ;==>_Main

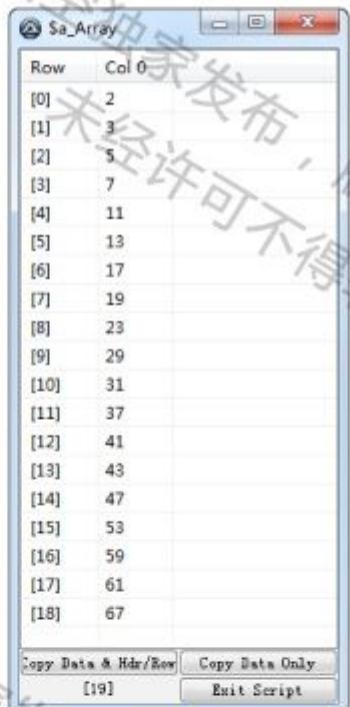
; -----
; 函数：
; _SK_ArrayBinarySearch_1D
; 功能：
; 在一个一维数组中查找目标数据
; 参数：
; $a_Array，目标数组
; $Value，目标数据
; $Start，指定从哪个索引开始向后查找，默认值为 0
; $End，指定查找到哪个索引位置，默认值为 0
; (当$End 参数为 0 时，$End 的值将自动等于数组的最终索引)
; 返回值：
; 成功返回元素索引，失败返回-1
; 失败时设置@error 值为非 0，@error：
; 1，目标数组并非数组
; 2，目标数组并非一维数组
; 3，起始位置超出数组范围
; 4，结束位置超出数组范围
; 5，起始位置大于结束位置
; 6，没有在数组内找到目标值
; -----
Func _SK_ArrayBinarySearch_1D(Const ByRef $a_Array, $Value, $Start = 0, $End = 0)
If Not (IsArray($a_Array)) Then Return SetError(1, 0, -1)
If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, -1)
If $End = 0 Then $End = UBound($a_Array, 1) - 1
If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, -1)
If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, -1)
If Not ($Start <= $End) Then Return SetError(5, 0, -1)

Local $Low = $Start, $Mid, $High = $End
Local $Index = -1
While ($Low <= $High)
    $Mid = Int(($Low + $High) / 2)
    If $Value = $a_Array[$Mid] Then

```

```
$Index = $Mid  
ExitLoop  
Elseif $Value > $a_Array[$Mid] Then  
    $Low = $Mid + 1  
Else  
    $High = $Mid - 1  
EndIf  
WEnd  
  
If $Index >= 0 Then  
    Return SetError(0, 0, $Index)  
Else  
    Return SetError(6, 0, -1)  
EndIf  
EndFunc ;==> _SK_ArrayBinarySearch_1D
```

运行图：



Row	Col 0
[0]	2
[1]	3
[2]	5
[3]	7
[4]	11
[5]	13
[6]	17
[7]	19
[8]	23
[9]	29
[10]	31
[11]	37
[12]	41
[13]	43
[14]	47
[15]	53
[16]	59
[17]	61
[18]	67

( 图 5-33 )



( 图 5-34 )

由图可见，成功的在一个一维有序数组中找到了目标值 19，其索引为 7。

折半查找法的算法相对复杂，建议大家反复阅读算法和代码，先理解、后使用。

### 3、拓展

在一个无序成绩表中查询特定人的特定某项成绩。

分析：

( 1 ) 无序表，需要使用顺序查找法；

- (2) 成绩表一般是由“人名->成绩”组成的二维数组；  
 (3) 只要查询到人名，同一行内即可找到其成绩；  
 (4) 相当于，将一维数组的顺序查找算法扩展到二维数组的某列。

**代码：**

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

;使用顺序查找法，在一个无序成绩表中查询特定人的某项成绩
Func _Main()
Local $a_Array[10][6] = [{"编号", "姓名", "性别", "年龄", "表达", "逻辑", "专业"},_
["06", "李峰", "男", "25", "81", "85", "79"],_
["07", "张硕", "男", "27", "75", "93", "91"],_
["04", "赵明", "男", "21", "93", "73", "85"],_
["05", "孙伟", "男", "26", "90", "82", "67"],_
["10", "谢娜", "女", "25", "87", "71", "83"],_
["03", "顾岚", "女", "22", "97", "87", "95"],_
["09", "马丽", "女", "20", "91", "67", "72"],_
["01", "王杨", "男", "26", "83", "91", "93"],_
["08", "刘超", "男", "23", "90", "93", "97"]]; <- 二维数组
_ArrayDisplay($a_Array, "$a_Array")

Local $Value = "顾岚", $SubItem = 1; <- 查询人名“顾岚”，在第2列（0基，列索引为1）查找
Local $Index = _SK_ArraySearch_2D($a_Array, $Value, 0, 0, $SubItem)
Switch @error
Case 0
  MsgBox(0 + 64, "信息", $Value & "的" & $a_Array[0][5] & "得分为：" &
$a_Array[$Index][5])
Case 1
  MsgBox(0 + 16, "错误", "目标数组并非数组")
Case 2
  MsgBox(0 + 16, "错误", "目标数组并非二维数组")
Case 3
  MsgBox(0 + 16, "错误", "起始位置超出数组范围")
Case 4
  MsgBox(0 + 16, "错误", "结束位置超出数组范围")
Case 5
  MsgBox(0 + 16, "错误", "起始位置大于结束位置")
Case 6
  MsgBox(0 + 16, "错误", "子索引超界")
Case 7
```

```
        MsgBox(0 + 64, "信息", "没有在数组内找到目标值")
EndSwitch
EndFunc ;==>_Main

;
; 函数：
; _SK_ArraySearch_2D
; 功能：
; 在一个二维数组中查找目标数据
; 参数：
; $a_Array，目标数组
; $Value，目标数据
; $Start，指定从哪个索引开始向后查找，默认值为 0
; $End，指定查找到哪个索引位置，默认值为 0
;           （当$End 参数为 0 时，$End 的值将自动等于数组的最终索引）
; $SubItem，列索引
; 返回值：
; 成功返回元素索引，失败返回-1
; 失败时设置@error 值为非 0，@error：
;   1，目标数组并非数组
;   2，目标数组并非二维数组
;   3，起始位置超出数组范围
;   4，结束位置超出数组范围
;   5，起始位置大于结束位置
;   6，子索引超界
;   7，没有在数组内找到目标值
;

Func _SK_ArraySearch_2D(Const ByRef $a_Array, $Value, $Start = 0, $End = 0, $SubItem = 0)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, -1)
    If UBound($a_Array, 0) <> 2 Then Return SetError(2, 0, -1)
    If $End = 0 Then $End = UBound($a_Array, 1) - 1
    If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, -1)
    If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, -1)
    If Not ($Start <= $End) Then Return SetError(5, 0, -1)
    If Not ($SubItem >= 0 And $SubItem < UBound($a_Array, 2)) Then Return SetError(6, 0, -1)

    Local $i
    Local $Index = -1
    For $i = 1 To UBound($a_Array, 1) - 1
        If $a_Array[$i][$SubItem] = $Value Then
            $Index = $i
            ExitLoop
        EndIf
    Next
```

```

If $Index >= 0 Then
    Return SetError(0, 0, $Index)
Else
    Return SetError(7, 0, -1)
Endif
EndFunc ;==> _SK_ArraySearch_2D

```

运行图：

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
[0]	编号	姓名	性别	年龄	表达	逻辑	专业
[1]	06	李峰	男	25	81	85	79
[2]	07	张硕	男	27	75	93	91
[3]	04	赵明	男	21	93	73	85
[4]	05	孙伟	男	26	90	82	67
[5]	10	谢娜	女	25	87	71	83
[6]	03	顾岚	女	22	97	87	95
[7]	09	马丽	女	20	91	67	72
[8]	01	王杨	男	26	83	91	93
[9]	08	刘超	男	23	90	93	97

( 图 5-35 )



( 图 5-36 )

## 5.2.4 排序

“排序”是数组算法中尤为重要的部分，在很多实际应用中我们都需要将数组数据按照一定的规则进行排序，进而有利于数据处理工作。本节介绍两种常见的数组排序算法，供大家学习参考。

### 1、插入排序法

插入排序法的思路，是每次将一个待排序数据按照其大小插入到已经有序的数据中。

插入排序法的执行过程：

将数组的第 2 个元素与第 1 个元素对比，使第 1、2 个元素变得有序；

将数组的第 3 个元素与第 1、2 个元素对比，将第 3 个元素按照大小插入到第 1、2 个元素中，使第 1、2、3 元素变得有序；

将数组的第 4 个元素与第 1、2、3 个元素对比，将第 4 个元素按照大小插入到第 1、2、3 个元素中，使第 1、2、3、4 元素变得有序；

.....  
将数组的第 n 个元素与第 1~(n-1)个元素对比，将第 n 个元素按照大小插入到第 1~(n-1)个元素中，使第 1~n 个元素变得有序。

从而，整个数组的排序完成。

下面我们来一起推演一下插入排序法的实现过程：

假设一个一维数组，元素值依次为 5、1、4、2、3：

5	1	4	2	3
---	---	---	---	---

( 图 5-37 )

### ( 1 ) i 指向值为 1 的元素

t=数组[i]=1

j=i-1，即指向 i 前面的一个元素

j	i			
5	1	4	2	3

( 图 5-38 )

因为 数组[j]>t，所以 数组[j+1]=数组[j]

j	i			
5	5	4	2	3

( 图 5-39 )

j 自减 1 后，指向了前一个位置

j	i			
5	5	4	2	3

( 图 5-40 )

j<数组下限，数据对比与移动结束

插入有序数据：数组[j+1]=t=1

j	i			
1	5	4	2	3

( 图 5-41 )

### ( 2 ) i 指向值为 4 的元素

t=数组[i]=4

j=i-1，即指向 i 前面的一个元素

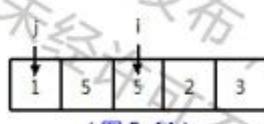
j	i			
1	5	4	2	3

( 图 5-42 )

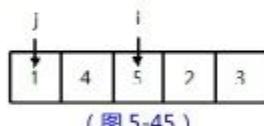
因为 数组[j]>t，所以 数组[j+1]=数组[j]

j	i			
1	5	5	2	3

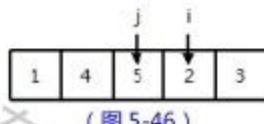
(图 5-43)

 $j=1$ , 指向前面一个位置

(图 5-44)

数组 $[j] < t$ , 数据对比与移动结束插入有序数据: 数组 $[j+1]=t=4$ 

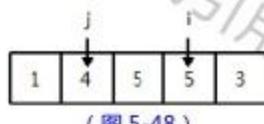
(图 5-45)

**(3) i 指向值为 2 的元素** $t=\text{数组}[i]=2$  $j=i-1$ , 即指向 i 前面的一个元素

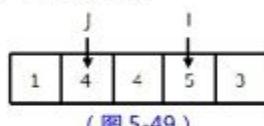
(图 5-46)

因为 数组 $[j]>t$ , 所以 数组 $[j+1]=\text{数组}[j]$ 

(图 5-47)

 $j=1$ , 指向前面一个位置

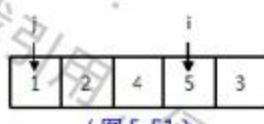
(图 5-48)

因为 数组 $[j]>t$ , 所以 数组 $[j+1]=\text{数组}[j]$ 

(图 5-49)

 $j=1$ , 指向前面一个位置

(图 5-50)

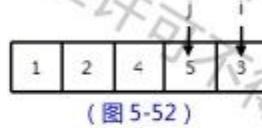
因为 数组 $[j]<t$ , 所以 数据对比与移动结束插入有序数据: 数组 $[j+1]=t=2$ 

(图 5-51)

## (4) i 指向值为 3 的元素

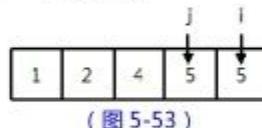
t=数组[i]=3

j=i-1，即指向 i 前面的一个元素



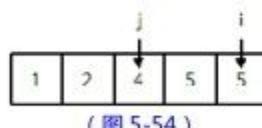
(图 5-52)

因为 数组[j]&gt;t，所以 数组[j+1]=数组[j]



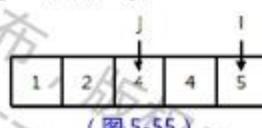
(图 5-53)

j=1，指向前一个位置



(图 5-54)

因为 数组[j]&gt;t，所以 数组[j+1]=数组[j]



(图 5-55)

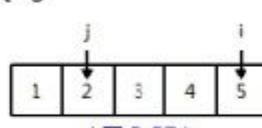
j=1，指向前一个位置



(图 5-56)

因为 数组[j]&lt;t，所以 数据对比与移动结束

插入有序数据： 数组[j+1]=t=3



(图 5-57)

*i* 的数据遍历结束，最终排序完成

(图 5-58)

总结一下上述过程，每次排序数组的数据为：

循环	执行	[0]	[1]	[2]	[3]	[4]
--	(原始数组)	5	1	4	2	3
第 1 次	[0][1]对比，使[0]~[1]有序	1	5	4	2	3
第 2 次	[2]插入[0]~[1]，使[0]~[2]有序	1	4	5	2	3
第 3 次	[3]插入[0]~[2]，使[0]~[3]有序	1	2	4	5	3
第 4 次	[4]插入[0]~[3]，使[0]~[4]有序	1	2	3	4	5

插入排序法的代码实现：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

; 使用插入排序法，使一个无序一维数组变得有序
Func _Main()
Local $a_Array[] = [5, 1, 4, 2, 3]
_ArrayDisplay($a_Array, '$a_Array - 原始')

_SK_ArraySort_InsertionSort_1D($a_Array)
Switch @error
Case 0
    _ArrayDisplay($a_Array, '$a_Array - 排序')
Case 1
    MsgBox(0 + 16, '错误', '目标数组并非数组')
Case 2
    MsgBox(0 + 16, '错误', '目标数组并非一维数组')
Case 3
    MsgBox(0 + 16, '错误', '起始位置超出数组范围')
Case 4
    MsgBox(0 + 16, '错误', '结束位置超出数组范围')
Case 5
    MsgBox(0 + 16, '错误', '起始位置大于结束位置')
EndSwitch
EndFunc ;==>_Main

; -----
; 函数：
; _SK_ArraySort_InsertionSort_1D
; 功能：
; 将一个一维数组按照从小到大的顺序排序
; 参数：
; $a_Array，目标数组
; $Start，指定从哪个索引开始排序，默认值为 0
; $End，指定排序到哪个索引为止，默认值为 0
; ( 当$End 参数为 0 时，$End 的值将自动等于数组的最终索引 )
; 返回值：
; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error :
; 1，目标数组并非数组
```

```

; 2, 目标数组并非一维数组
; 3, 起始位置超出数组范围
; 4, 结束位置超出数组范围
; 5, 起始位置大于结束位置
; -----
Func _SK_ArraySort_InsertionSort_1D(ByRef $a_Array, $Start = 0, $End = 0)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, 0)
    If $End = 0 Then $End = UBound($a_Array, 1) - 1
    If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, 0)
    If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, 0)
    If Not ($Start <= $End) Then Return SetError(5, 0, 0)

    Local $t
    Local $i, $j
    For $i = $Start + 1 To $End
        $t = $a_Array[$i]
        $j = $i - 1
        While ($j >= $Start And $a_Array[$j] > $t)
            $a_Array[$j + 1] = $a_Array[$j]
            $j -= 1
        WEnd
        If $j <> $i - 1 Then
            $a_Array[$j + 1] = $t
        EndIf
    Next

    Return SetError(0, 0, 1)
EndFunc  ;==> _SK_ArraySort_InsertionSort_1D

```

运行图：

Row	Col 0
[0]	5
[1]	1
[2]	4
[3]	2
[4]	3

Copy Data & Hdr/Bow    Copy Data Only  
[5]                      Exit Script

( 图 5-59 )

Row	Col 0
[0]	1
[1]	2
[2]	3
[3]	4
[4]	5

Copy Data & Hdr/Bow    Copy Data Only  
[5]                      Exit Script

( 图 5-60 )

## 2、冒泡排序法

冒泡排序法的思路，即逐个走访要排序的元素，每次对两个元素的数据进行比较，并将其按照大小顺序进行交换，走访会一直持续下去，直至不再有需要交换的数据，继而排序完成。

冒泡排序法的执行过程：

将元素 0 与元素 1~n 进行对比，哪个元素的值比元素 0 小，则与元素 0 进行数据交换，从而第 1 次走访结束后元素 0 的值为元素 0~n 中最小；

将元素 1 与元素 2~n 进行对比，哪个元素的值比元素 1 小，则与元素 1 进行数据交换，从而第 2 次走访结束后元素 1 的值为元素 1~n 中最小；

将元素 x 与元素(x+1)~n 进行对比，哪个元素的值比元素 x 小，则与元素 n 进行数据交换，从而第(x+1)次走访结束后元素 x 的值为元素 x~n 中最小；

.....

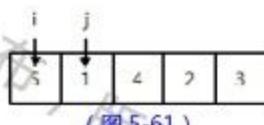
将元素(n-1)与 n 进行对比，如果元素(n-1)小于元素 n 则进行值交换，从而第 n 次对比结束后，所有元素值即已按照从小到大的顺序排序完成。

下面我们来举个实际例子推演一下冒泡排序法的执行过程：

假设一个一维数组，元素值依次为 5、1、4、2、3。

### (1) i 指向元素 0

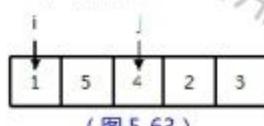
i 指向数据 5，j 指向数据 1



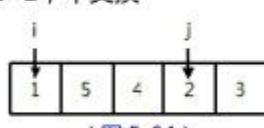
因为  $5 > 1$ ，所以将 1 与 5 交换



$j+1=1$ ，指向下一个数据 4， $1 < 4$ ，不交换



$j+1=2$ ，指向下一个数据 2， $1 < 2$ ，不交换

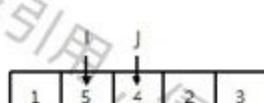


$j+1=3$ ，指向下一个数据 3， $1 < 3$ ，不交换



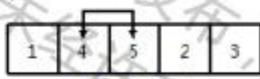
### (2) i 指向元素 1

i 指向数据 5，j 指向数据 4



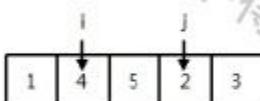
(图 5-66)

因为  $5 > 4$ , 所以将 5 与 4 交换



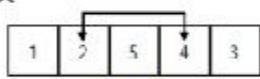
(图 5-67)

$j = 1$ , 指向下一个数据 2



(图 5-68)

因为  $4 > 2$ , 所以将 4 与 2 交换



(图 5-69)

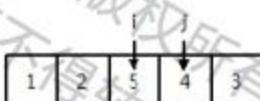
$j = 1$ , 指向下一个数据 3,  $2 < 3$ , 不交换



(图 5-70)

### (3) i 指向元素 2

i 指向数据 5, j 指向数据 4



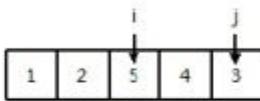
(图 5-71)

因为  $5 > 4$ , 所以将 5 与 4 交换



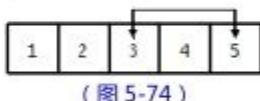
(图 5-72)

$j = 1$ , 指向下一个数据 3



(图 5-73)

因为  $5 > 3$ , 所以将 5 与 3 交换



(图 5-74)

### (4) i 指向元素 4

i 指向数据 4, j 指向数据 5, 因为  $4 < 5$ , 不交换



(图 5-75)

排序结束

有序数组：

1	2	3	4	5
---	---	---	---	---

(图 5-76)

总结一下上述过程，每次排序数组的数据为：

循环	执行	[0]	[1]	[2]	[3]	[4]
--	(原始数组)	5	1	4	2	3
第 1 次	将[0]~[4]的最小值放在[0]上	1	5	4	2	3
第 2 次	将[1]~[4]的最小值放在[1]上	1	2	5	4	3
第 3 次	将[2]~[4]的最小值放在[2]上	1	2	3	5	4
第 4 次	将[3]~[4]的最小值放在[3]上	1	2	3	4	5
--	(排序结束)	1	2	3	4	5

冒泡排序法的代码实现：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

;使用冒泡排序法，使一个无序一维数组变得有序
Func _Main()
    Local $a_Array[] = [5, 1, 4, 2, 3]
    _ArrayDisplay($a_Array, '$a_Array - 原始')

    _SK_ArraySort_BubbleSort_1D($a_Array)
    Switch @error
        Case 0
            _ArrayDisplay($a_Array, '$a_Array - 排序')
        Case 1
            MsgBox(0 + 16, '错误', '目标数组并非数组')
        Case 2
            MsgBox(0 + 16, '错误', '目标数组并非一维数组')
        Case 3
            MsgBox(0 + 16, '错误', '起始位置超出数组范围')
        Case 4
            MsgBox(0 + 16, '错误', '结束位置超出数组范围')
        Case 5
            MsgBox(0 + 16, '错误', '起始位置大于结束位置')
    EndSwitch
EndFunc   ==> _Main
```

```
; -----
; 函数：
; _SK_ArraySort_BubbleSort_1D
; 功能：
; 将一个一维数组按照从小到大的顺序排序
; 参数：
; $a_Array，目标数组
; $Start，指定从哪个索引开始排序，默认值为 0
; $End，指定排序到哪个索引为止，默认值为 0
; (当$End 参数为 0 时，$End 的值将自动等于数组的最终索引)
; 返回值：
; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error：
; 1，目标数组并非数组
; 2，目标数组并非一维数组
; 3，起始位置超出数组范围
; 4，结束位置超出数组范围
; 5，起始位置大于结束位置
; -----
Func _SK_ArraySort_BubbleSort_1D(ByRef $a_Array, $Start = 0, $End = 0)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, 0)
    If $End = 0 Then $End = UBound($a_Array, 1) - 1
    If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, 0)
    If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, 0)
    If Not ($Start <= $End) Then Return SetError(5, 0, 0)

    Local $t
    Local $i, $j
    For $i = $Start To $End - 1
        For $j = $i + 1 To $End
            If $a_Array[$j] < $a_Array[$i] Then
                $t = $a_Array[$i]
                $a_Array[$i] = $a_Array[$j]
                $a_Array[$j] = $t
            EndIf
        Next
    Next

    Return SetError(0, 0, 1)
EndFunc  ;==> _SK_ArraySort_BubbleSort_1D
```

运行图：

Row	Col 0
[0]	5
[1]	1
[2]	4
[3]	2
[4]	3

(图 5-77)

Row	Col 0
[0]	1
[1]	2
[2]	3
[3]	4
[4]	5

(图 5-78)

### 3、拓展

将成绩表按照“编号”从小到大的顺序进行排序

**分析：**

- (1) 成绩表一般是一种“人名->成绩”的二维数组；
- (2) 按照编号排序，即使用冒泡排序算法对编号进行排序即可；
- (3) 由于是二维数组，每个编号对应后面人名、成绩等多个信息，而不是一个；
- (4) 排序中交换数据时，需要将整行进行交换，而不能只交换被排序的那一个数据。

**代码：**

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
Exit

; 使用冒泡排序法，将成绩单按照“编号”从小到大的顺序排列
Func _Main()
    Local $a_Array[] = [['编号', '姓名', '性别', '年龄', '表达', '逻辑', '专业'], ...
        ['06', '李峰', '男', '25', '81', '85', '79'], ...
        ['07', '张硕', '男', '27', '75', '93', '91'], ...
        ['04', '赵明', '男', '21', '93', '73', '85'], ...
        ['05', '孙伟', '男', '26', '90', '82', '67'], ...
        ['10', '谢娜', '女', '25', '87', '71', '83'], ...
        ['03', '顾嵐', '女', '22', '97', '87', '95'], ...
        ['09', '马丽', '女', '20', '91', '67', '72'], ...
        ['01', '王杨', '男', '26', '83', '91', '93'], ...
        ['02', '韩雪', '女', '20', '86', '75', '91'], ...
        ['08', '刘超', '男', '23', '90', '93', '97']]
    _ArrayDisplay($a_Array, '$a_Array - 原始')

    _SK_ArraySort_BubbleSort_2D($a_Array, 1, 0, 0)
    Switch @error

```

```
Case 0
    _ArrayDisplay($a_Array, '$a_Array - 排序')
Case 1
    MsgBox(0 + 16, '错误', '目标数组并非数组')
Case 2
    MsgBox(0 + 16, '错误', '目标数组并非二维数组')
Case 3
    MsgBox(0 + 16, '错误', '起始位置超出数组范围')
Case 4
    MsgBox(0 + 16, '错误', '结束位置超出数组范围')
Case 5
    MsgBox(0 + 16, '错误', '起始位置大于结束位置')
Case 6
    MsgBox(0 + 16, '错误', '子索引超界')
EndSwitch
EndFunc ;=> Main

; -----
; 函数：
; _SK_ArraySort_BubbleSort_2D
; 功能：
; 将一个二维数组按照从小到大的顺序排序
; 参数：
; $a_Array，目标数组
; $Start，指定从哪个索引开始排序，默认值为 0
; $End，指定排序到哪个索引为止，默认值为 0
;           (当$End 参数为 0 时，$End 的值将自动等于数组的最终索引)
; $SubItem，列索引
; 返回值：
; 成功返回 1，失败返回 0
; 失败时设置@error 值为非 0，@error：
;   1，目标数组并非数组
;   2，目标数组并非二维数组
;   3，起始位置超出数组范围
;   4，结束位置超出数组范围
;   5，起始位置大于结束位置
;   6，子索引超界
;

Func _SK_ArraySort_BubbleSort_2D(ByRef $a_Array, $Start = 0, $End = 0, $SubItem = 0)
    If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
    If UBound($a_Array, 0) <> 2 Then Return SetError(2, 0, -1)
    If $End = 0 Then $End = UBound($a_Array, 1) - 1
    If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, 0)
    If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, 0)
    If Not ($Start <= $End) Then Return SetError(5, 0, 0)
```

```

If Not ($SubItem >= 0 And $SubItem < UBound($a_Array, 2)) Then Return SetError(6, 0, 0)

Local $t
Local $i, $j, $k
For $i = $Start To $End - 1
    For $j = $i + 1 To $End
        If $a_Array[$j][$SubItem] < $a_Array[$i][$SubItem] Then
            For $k = 0 To UBound($a_Array, 2) - 1
                $t = $a_Array[$i][$k]
                $a_Array[$i][$k] = $a_Array[$j][$k]
                $a_Array[$j][$k] = $t
            Next
        EndIf
    Next
Next

Return SetError(0, 0, 1)
EndFunc ;==> _SK_ArraySort_BubbleSort_2D

```

运行图：

IT天空独家发布，版权所有！未经许可不得转载引用，侵权必究！

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
[0]	编号	姓名	性别	年龄	表达	逻辑	专业
[1]	06	李峰	男	25	81	85	79
[2]	07	张硕	男	27	75	93	91
[3]	04	赵明	男	21	93	73	85
[4]	05	孙伟	男	26	90	82	67
[5]	10	谢娜	女	25	87	71	83
[6]	03	顾嵩	女	22	97	87	95
[7]	09	马丽	女	20	91	67	72
[8]	01	王杨	男	26	83	91	93
[9]	02	魏雷	女	20	86	75	91
[10]	08	刘超	男	23	90	93	97

( 图 5-79 )

Row	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6
[0]	编号	姓名	性别	年龄	球达	逻辑	专业
[1]	01	王杨	男	26	83	91	93
[2]	02	韩雪	女	20	86	75	91
[3]	03	顾嵩	女	22	97	87	95
[4]	04	赵明	男	21	93	73	85
[5]	05	孙伟	男	26	90	82	67
[6]	06	李峰	男	25	81	85	79
[7]	07	张硕	男	27	75	93	91
[8]	08	刘超	男	23	90	93	97
[9]	09	马丽	女	20	91	67	72
[10]	10	谢娜	女	25	87	71	83

(图 5-80)

仔细观察代码发现，冒泡排序法对于一维和二维数组的最大不同在于数据的对比与交换。

处理一维数组时，对比的是二维数组内的两个数据，交换的是一维数组内的两个元素值：

```
For $i = $Start To $End - 1
    For $j = $i + 1 To $End
        If $a_Array[$j] < $a_Array[$i] Then
            $t = $a_Array[$i]
            $a_Array[$i] = $a_Array[$j]
            $a_Array[$j] = $t
        EndIf
    Next
Next
```

处理二维数组时，对比的是二维数组特定列的两个数据，交换的是二维数组内的两整行值：

```
For $i = $Start To $End - 1
    For $j = $i + 1 To $End
        If $a_Array[$j][$SubItem] < $a_Array[$i][$SubItem] Then
            For $k = 0 To UBound($a_Array, 2) - 1
                $t = $a_Array[$i][$k]
                $a_Array[$i][$k] = $a_Array[$j][$k]
                $a_Array[$j][$k] = $t
            Next
        EndIf
    Next
Next
```

而就冒泡排序的算法而言，一维或二维数组没有什么区别，只是数据内容有所变化罢了。由此，请大家进一步通过类比的思想进一步体会算法的通用性。

### 5.2.5 去重复

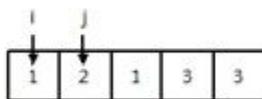
去掉数组中的重复值，是日常处理数据时的一种常见操作。去重复的算法相对简单，即遍历整个数组，令每个元素与其之后的所有元素进行对比，删除与之相等的元素。这其中包含了“在数组中删除元素”的算法，而删除元素后数组大小会产生改变，也是本算法的难点之一。

下面我们来举个简单例子推演一下去重复算法的执行过程：

假设一个一维数组，元素值依次为 1、2、1、3、3。

#### (1) i 指向元素 0

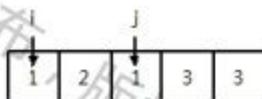
i 指向数据 1, j=i+1 指向数据 2



(图 5-81)

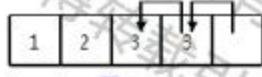
1 与 2 不重复， $j+=1$  指向下一个元素

i 指向数据 1, j 指向数据 1



(图 5-82)

1 与 1 重复，删除 j 指向的数据 1，并将之后的元素前移

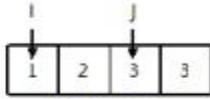


(图 5-83)

数组最大索引减少 1

注意，由于数据的前移，此时 j 指向了数据 3，即之前的( $j+1$ )元素，所以此时 j 无需自增 1

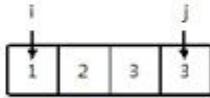
i 指向数据 1, j 指向数据 3



(图 5-84)

1 与 3 不重复， $j+=1$  指向下一个元素

i 指向数据 1, j 指向数据 3



(图 5-85)

1 与 3 不重复， $j+=1$  已超出数组范围，则第一波“去重复”结束

#### (2) i 指向元素 1

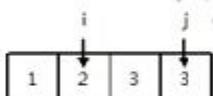
i 指向数据 2, j=i+1 指向数据 3



(图 5-86)

2与3不重复， $j+1$ 指向下一个元素

i指向数据2，j指向数据3

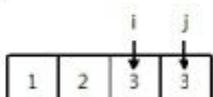


(图 5-87)

2与3不重复， $j+1$ 已超出数组范围，则第二波“去重复”结束

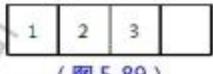
### (3) i指向元素2

i指向数据3， $j=i+1$ 指向数据3



(图 5-88)

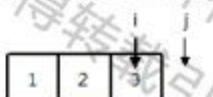
3和3重复，删除j指向的数据3



(图 5-89)

数组最大索引减少1

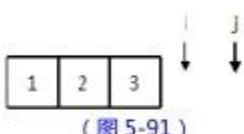
因数组最大索引减少1后，此时j的指向已超出数组范围，则第三波“去重复”结束



(图 5-90)

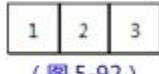
### (4) i指向元素3

由于之前的删除元素、缩减数组的操作，此时i已超出数组范围， $j=i+1$ 同样超出数组范围  
整个“去重复”结束



(图 5-91)

去重复结束：



(图 5-92)

去重复的代码实现：

```
#include <Array.au3>

Opt("MustDeclareVars", 1)

_Main()
```

```

Exit

;去掉一维数组中的重复元素
Func _Main()
    Local $a_Array[] = [1, 2, 1, 3, 3]
    _ArrayDisplay($a_Array, '$a_Array - 原始')

    _SK_ArrayUnique($a_Array)

    Switch @error
        Case 0
            _ArrayDisplay($a_Array, '$a_Array - 去重复')
        Case 1
            MsgBox(0 + 16, '错误', '目标数组并非数组')
        Case 2
            MsgBox(0 + 16, '错误', '目标数组并非一维数组')
        Case 3
            MsgBox(0 + 16, '错误', '起始位置超出数组范围')
        Case 4
            MsgBox(0 + 16, '错误', '结束位置超出数组范围')
        Case 5
            MsgBox(0 + 16, '错误', '起始位置大于结束位置')
    EndSwitch
EndFunc   ;==>_Main

; -----
; 函数：
;   _SK_ArrayUnique
; 功能：
;   去掉一维数组中的重复元素
; 参数：
;   $a_Array , 目标数组
;   $Start , 指定从哪个索引开始排序，默认值为 0
;   $End , 指定排序到哪个索引为止，默认值为 0
;       ( 当$End 参数为 0 时，$End 的值将自动等于数组的最终索引 )
; 返回值：
;   成功返回 1，失败返回 0
;   失败时设置@error 值为非 0，@error :
;       1，目标数组并非数组
;       2，目标数组并非一维数组
;       3，起始位置超出数组范围
;       4，结束位置超出数组范围
;       5，起始位置大于结束位置
; -----
Func _SK_ArrayUnique(ByRef $a_Array, $Start = 0, $End = 0)

```

```
If Not (IsArray($a_Array)) Then Return SetError(1, 0, 0)
If UBound($a_Array, 0) <> 1 Then Return SetError(2, 0, 0)
If $End = 0 Then $End = UBound($a_Array, 1) - 1
If Not ($Start >= 0 And $Start < UBound($a_Array, 1)) Then Return SetError(3, 0, 0)
If Not ($End >= 0 And $End < UBound($a_Array, 1)) Then Return SetError(4, 0, 0)
If Not ($Start <= $End) Then Return SetError(5, 0, 0)

Local $i, $j, $k
Local $flg
$i = $Start
While $i <= $End - 1
    $j = $i + 1
    While $j <= $End
        $flg = 1
        If $a_Array[$i] = $a_Array[$j] Then
            For $k = $j + 1 To UBound($a_Array, 1) - 1
                $a_Array[$k - 1] = $a_Array[$k]
            Next
            ReDim $a_Array[UBound($a_Array, 1) - 1]
            $flg = 0
            $End -= 1
        EndIf
        If $flg Then $j += 1
    WEnd
    $i += 1
WEnd

Return SetError(0, 0, 1)
EndFunc ;==> _SK_ArrayUnique
```

运行图：

Row	Col 0
[0]	1
[1]	2
[2]	1
[3]	3
[4]	3

( 图 5-93 )

Row	Col 0
[0]	1
[1]	2
[2]	3

( 图 5-94 )

## 5.3 编译参数

很多 Au3 代码的开头，都有一些以“#”开头的段，形如：

```
#Region  
#AutoIt3Wrapper_icon=MyExe.ico  
#AutoIt3Wrapper_Outfile=MyExe.exe  
#AutoIt3Wrapper_UseUpx=n  
#AutoIt3Wrapper_UseX64=n  
#AutoIt3Wrapper_Res_Comment=MyExe - IT 天空  
#AutoIt3Wrapper_Res_Description=我的程序 v3.7  
#AutoIt3Wrapper_Res_Fileversion=1.3.24.115  
#AutoIt3Wrapper_Res_ProductVersion=1.3.0.0  
#AutoIt3Wrapper_Res_Language=2052  
#AutoIt3Wrapper_Res_LegalCopyright=Copyright 2014 Skyfree.  
#AutoIt3Wrapper_Res_requestedExecutionLevel=requireAdministrator  
#AutoIt3Wrapper_Run_Tidy=y  
#AutoIt3Wrapper_Run_Obfuscator=y  
#Obfuscator_Parameters=/cs=0 /cn=0 /cf=0 /cv=0 /sf=1 /sv=1  
#EndRegion
```

像这样的段，一般被称为“编译参数”。编译参数的作用是什么？常用的编译参数有哪些？本节我们将针对这些问题答疑解惑。

### 5.3.1 AutoIt3 与 SciTE4AutoIt3

通常所使用的 Au3 编程环境一般由三部分组成：AutoIt3、SciTE4AutoIt3 和第三方修改。其中，“第三方修改”是高手们为了使 Au3 编程环境更加实用，而对 Au3 编辑器、设计器、帮助文档等进行的汉化与修改。AutoIt3 和 SciTE4AutoIt3 则是 Au3 编程环境的核心组成部分。

#### 1、AutoIt3

AutoIt3 是 Au3 编程环境最核心的部分，换句话说，它才是 Au3 的真身。

##### (1) 安装

下载地址：<https://www.autoitscript.com/cgi-bin/getfile.pl?autoit3/autoit-v3-setup.exe>

下载后是一个.exe 安装包，双击运行，按照提示安装即可：



(图 5-95)

AutoIt3 安装完毕后，我们就有了一个基本的 Au3 编程环境。

### (2) SciTE

AutoIt3 自带 SciTE 编辑器作为代码编辑工具，只是功能比较简单。使用 SciTE 创建一段代码，可以正常运行和编译。

```
#cs -----
# AutoIt Version: 3.3.12.0
# Author: myName

Script Function:
Template AutoIt script.

#ce

; Script Start - Add your code below here

MsgBox(0,'','Hello World')
```

(图 5-96)

在 SciTE 编辑器内，按 F5 运行脚本，实际上是运行：

```
'<Au3 安装目录>\SciTE\..\autoit3.exe" /ErrorStdOut "<Au3 脚本路径>"'
```

在 SciTE 编辑器内，按 F7 编译脚本，实际上是运行：

```
'<Au3 安装目录>\SciTE\..\aut2exe\aut2exe.exe" /in "<Au3 脚本路径>"'
```

### (3) 右键菜单

在 au3 脚本上单击鼠标右键时，弹出菜单如下图：



(图 5-97)

说明：

- Run Script->运行脚本
- Run Script (x64)->以 64 位方式运行脚本
- Run Script (x86)->以 32 位方式运行脚本
- Compile Script->编译脚本
- Compile Script (x64)->以 64 位方式编译脚本
- Compile Script (x86)->以 32 位方式编译脚本
- Edit Script->编辑脚本
- Open->打开脚本

注意：

- 32 位系统仅能以 x86 方式运行或编译脚本，无法使用 x64 方式
- Run Script (x64) 和 Run Script (x86)，仅在 x64 系统下才会出现
- Compile Script (x64) 和 Compile Script (x86)，仅在 x64 系统下才会出现
- x64 系统在安装时会提示默认使用 x86 还是 x64

若选择 x86，则 Run Script 与 Compile Script 默认为 32 位方式

若选择 x64，则 Run Script 与 Compile Script 默认为 64 位方式

## 2、SciTE4AutoIt3

SciTE4AutoIt3 (SciTE for AutoIt3)，是为 AutoIt3 量身定做的增强包。虽然名为 SciTE 增强包，但其实除了 SciTE 的增强之外，还包含了很多其他组件，如代码整理 (Tidy)、GUI 设计 (Koda)、代码精简 (Au3Stripper)、代码混淆 (Obfuscator)、编译增强 (AutoIt3Wrapper) 等等。

AutoIt3 与 SciTE4AutoIt3 的关系，就像咖啡和牛奶一样，虽然喝咖啡不是必须加牛奶，但牛奶却可以有效的改善咖啡的口感。

### (1) 安装

下载地址：<http://www.autoitscript.com/autoit3/scite/download/SciTE4AutoIt3.exe>

注意，安装 SciTE4AutoIt3 的前提是已经安装了 AutoIt3。

下载后是一个.exe 安装包，双击运行，按照提示操作即可：



(图 5-98)

### (2) SciTE

此时的 SciTE 已经可以识别并高亮编译参数了：

A screenshot of the SciTE4AutoIt3 editor window. The title bar says "D:\au3\test.au3 - SciTE". The menu bar includes File, Edit, Search, View, Tools, Options, Language, Buffers, Help. The code area contains the following AutoIt script:

```
1 #Region ***** Directives curated by AutoIt3Wrapper_GUI *****
2 #AutoIt3Wrapper_UseUpx=y
3 #AutoIt3Wrapper_UseX64=n
4 #AutoIt3Wrapper_Res_Description=Test
5 #AutoIt3Wrapper_Res_Fileversion=1.0.0.1
6 #AutoIt3Wrapper_Res_LegalCopyright=Copyright 2014
7 #AutoIt3Wrapper_Res_Language=1052
8 #AutoIt3Wrapper_Run_Tidy=y
9 #AutoIt3Wrapper_Run_Au3Stripper=y
10 #EndRegion ***** Directives created by AutoIt3Wrapper_GUI *****
11
12 #cs
13
14     AutoIt Version: 3.3.12.0
15     Author: myName
16
17     Script Function:
18     Template AutoIt script.
19
20 #ce
21
22 ; Script Start - Add your code below here
23
24 MsgBox(0, "", "Hello World")
25
```

The code is color-coded: directives like #Region, #EndRegion, #cs, and #ce are in blue; comments are in green; and the MsgBox command is highlighted in yellow.

(图 5-99)

在 SciTE 编辑器内，按 F5 运行脚本，实质上是运行：

```
"<Au3 安装目录>\SciTE\AutoIt3Wrapper\AutoIt3Wrapper.exe" /run /prod /ErrorStdOut /in "<Au3 脚本路径>" /UserParams
```

在 SciTE 编辑器内，按 F7 编译脚本，实质上是运行：

```
"<Au3 安装目录>\SciTE\AutoIt3Wrapper\AutoIt3Wrapper.exe" /prod /in "<Au3 脚本路径>"
```

值得注意的是，在安装 SciTE 增强包后，在 SciTE 内运行、编译脚本所使用的命令已经与仅安装 AutoIt3 时截然不同。无论是运行还是安装，都是通过一个叫做 AutoIt3Wrapper.exe 的程序在代理执行。什么是“代理执行”？AutoIt3Wrapper.exe 并不具备 autoit3.exe ( autoit3\_x64.exe ) 运行

脚本的能力，也并不具备 aut2exe.exe（aut2exe\_x64.exe）编译脚本的能力，但却能够调用了二者来运行或编译。为什么要代理执行而不是直接执行？5.3.2 节中揭晓。

### （3）右键菜单

在 au3 脚本上单击鼠标右键时，弹出菜单如下图：



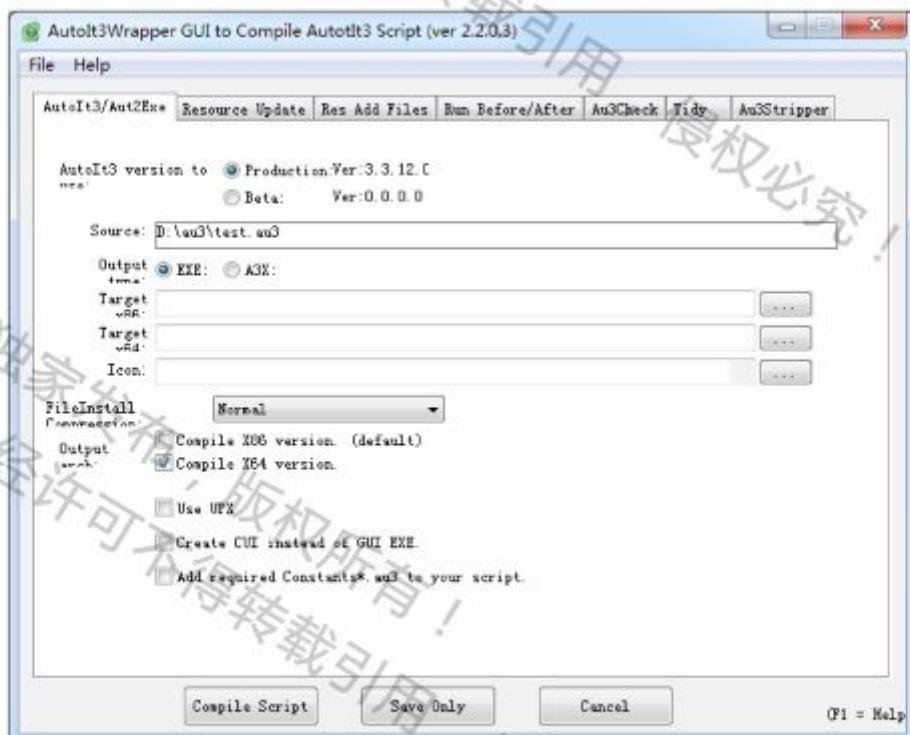
（图 5-100）

与仅安装 AutoIt3 时相比，多出了“Compile with Options”和“Tidy”两个项目，而其余项目无论是描述还是功能，均与仅安装 AutoIt3 时相同。

使用“Compile with Options”时，将调出 AutoIt3Wrapper.exe 的图形界面，可在图形界面中设置编译选项，对应到脚本中，就是编译参数。

使用“Tidy”时，将对脚本进行整理，自动调整代码缩进，使代码更加条理。

## 5.3.2 AutoIt3Wrapper



（图 5-101）

笔者个人认为，用其貌不扬来形容 AutoIt3Wrapper.exe 并不为过，不过这并不影响它强大的功能，或许原作者就没打算在 UI 上多费心呢。AutoIt3Wrapper.exe 的主要功能分为三个方面：

### (1) 设置编译参数

在 SciTE 编辑器中使用 Ctrl+F7 快捷键，或在脚本右键菜单中单击“Compile with Options”，都可以调出 AutoIt3Wrapper 的图形界面。在图形界面中填入对应的设置信息，单击“Save Only”按钮会将设置信息以编译参数的方式保存至脚本内，而单击“Compile Script”按钮则会在保存编译参数后执行编译工作。

AutoIt3Wrapper 中各选项最终都会以编译参数的方式保存在脚本中，所以不再就各选项做一一介绍，而在 5.3.4 章节中直接学习各编译参数的含义。

### (2) 根据编译参数运行脚本

有些编译参数不仅在编译时有效，在直接运行时也会对脚本产生影响。例如当设置脚本使用 x64 方式运行时，AutoIt3Wrapper 会自动调用 autoit3\_x64.exe 去运行脚本，而当设置脚本使用 x86 方式运行时，AutoIt3Wrapper 会自动调用 autoit3.exe 去运行脚本。

值得注意的是，仅限在 SciTE 内使用 F5 快捷键运行脚本时才有这样的效果，因为只有在这种条件下运行脚本才是使用 AutoIt3Wrapper 代理的，而直接双击运行脚本、右键菜单运行脚本的方式都不是通过 AutoIt3Wrapper 代理的，不通过代理则不会去应用任何编译参数的设定。

### (3) 根据编译参数编译脚本

使用 AutoIt3Wrapper 编译时，不仅仅是简单调用了 aut2exe.exe 进行编译，而是在编译时将一些编译参数所设定的信息一同“包装”(wrapper)进了程序中。例如文件说明、文件版本、产品名称、产品版本、版权信息等。AutoIt3Wrapper 还可以根据参数设定进行一些特殊的编译处理。例如当设置整理脚本时，会自动先调用 Tidy.exe 对脚本进行整理，再进行编译；当设置程序以 x64 方式运行时，编译时会自动调用 aut2exe\_x64.exe 进行编译，当设置程序以 x86 方式运行时，编译时会自动调用 aut2exe.exe 进行编译；当设置程序使用 Upx 压缩节省体积时，编译后会自动调用 upx.exe 对编译后的程序进行压缩。

值得注意的是，仅限在 SciTE 内使用 F7 快捷键、右键菜单使用“Compile with Options”编译脚本时才有这样的效果，因为只有在这种条件下编译脚本才是使用 AutoIt3Wrapper 代理的，而其他编译脚本的方式不是通过 AutoIt3Wrapper 代理的，不通过代理则不会去应用任何编译参数的设定。

其他说明，SciTE4AutoIt3 自 2014 年 5 月 4 日的版本起终止了对 Obfuscator.exe 的支持，继而使用 Au3Stripper.exe 进行替代。AutoIt3Wrapper 也不再带有 Obfuscator 的设置选项，全部以 Au3Stripper 设置选项代替。

Obfuscator.exe 的功能包括代码精简和代码迷惑两部分，前者旨在对未使用的函数、未使用的变量进行精简，后者用于对变量、函数、字符串、数字进行加密迷惑。代码迷惑功能，可以使程序即便被反编成源代码，也很难在短时间内读懂，可以一定程度上对代码起到保护作用。而 Au3Stripper.exe 则只有代码精简功能，并不具备代码迷惑功能。

针对这一问题，国内各路高手给出了各种解决方案，5.3.5 节将要介绍的 AccAu3Wrapper 就是其中一种出色的解决方案，所以解决代码混淆问题不只有依赖 AutoIt3Wrapper.exe 这一个思路。

### 5.3.4 常用的编译参数

#### 1、编译

##### ( 1 ) #AutoIt3Wrapper\_UseX64

是否使用 x64 模式。等于 y 时为 x64 模式，等于 n 时为 x86 模式。

```
#AutoIt3Wrapper_UseX64=n
```

( 注意，只在 x64 系统下运行或编译脚本时可用此选项 )

##### ( 2 ) #AutoIt3Wrapper\_OutFile、#AutoIt3Wrapper\_OutFile\_x64

设置编译后程序的输出路径，可以是相对或绝对路径。

#AutoIt3Wrapper\_OutFile 为 x86 模式的输出，前提是#AutoIt3Wrapper\_UseX64=n。

#AutoIt3Wrapper\_OutFile\_x64 为 x64 模式的输出，前提是#AutoIt3Wrapper\_UseX64=y。

```
#AutoIt3Wrapper_OutFile=test.exe
```

```
#AutoIt3Wrapper_OutFile=d:\au3\test.exe
```

```
#AutoIt3Wrapper_OutFile_x64=d:\au3\test.exe
```

##### ( 3 ) #AutoIt3Wrapper\_Icon

指定编译后程序所使用的图标，可以是相对或绝对路径。

```
#AutoIt3Wrapper_Icon=test.ico
```

```
#AutoIt3Wrapper_Icon=d:\au3\test.ico
```

##### ( 4 ) #AutoIt3Wrapper\_UseUpx、#AutoIt3Wrapper\_Compression

使用 Upx 压缩编译后的程序，减少程序体积。

#AutoIt3Wrapper\_UseUpx 等于 y 时启用压缩，等于 n 时不启用压缩。

```
#AutoIt3Wrapper_UseUpx=y
```

#AutoIt3Wrapper\_Compression 用于设置压缩等级，值可设置为 0 到 4。其中 4 是最高压缩，  
默认压缩等级为 2。

```
#AutoIt3Wrapper_Compression=4
```

#### 2、信息

##### ( 1 ) #AutoIt3Wrapper\_Res\_Comment

设置程序说明。

```
#AutoIt3Wrapper_Res_Comment=我是程序说明
```

##### ( 2 ) #AutoIt3Wrapper\_Res\_Description

设置程序描述。

```
#AutoIt3Wrapper_Res_Description=我是程序描述
```

##### ( 3 ) #AutoIt3Wrapper\_Res\_Fileversion

设置文件版本。

```
#AutoIt3Wrapper_Res_Fileversion=3.0.15.311
```

##### ( 4 ) #AutoIt3Wrapper\_ResFileVersion\_AutoIncrement

设置文件版本递增方式。等于 y 时自动递增文件版本，等于 p 时每次编译询问是否递增，等于 n  
时不自动递增。

## Let's AutoIt Plus

```
#AutoIt3Wrapper_Res_FileVersion_AutoIncrement=p
```

### ( 5 ) #AutoIt3Wrapper\_Res\_ProductVersion

设置产品版本。

```
#AutoIt3Wrapper_Res_ProductVersion=3.0.0.1
```

### ( 6 ) #AutoIt3Wrapper\_Res\_Language

设置程序语言。用于告之系统程序的语言类型，中文为 2052。其实即使这里设置成了英文，程序中的中文内容还是可以正确显示的，这只是一个属性值，但建议不要写错。

```
#AutoIt3Wrapper_Res_Language=2052
```

### ( 7 ) #AutoIt3Wrapper\_Res\_LegalCopyright

设置版权信息。

```
#AutoIt3Wrapper_Res_LegalCopyright=Copyright (C) 2006-2014 itiankong.net, All rights reserved
```

### ( 8 ) #AutoIt3Wrapper\_Res\_RequestedExecutionLevel

设置运行等级请求，主要用于对 Windows UAC 的权限请求。有 asInvoker、highestAvailable、requireAdministrator、None 四种值。

asInvoker，运行时进行请求（默认值）；

highestAvailable，请求最高可用权限；

requireAdministrator，请求管理员权限；

None，不请求。

如果程序执行了一些系统权限要求较高的操作，可尝试设置请求等级为 requireAdministrator。

```
#AutoIt3Wrapper_Res_RequestedExecutionLevel=requireAdministrator
```

## 3、整理

### #AutoIt3Wrapper\_Run\_Tidy

是否启用代码整理功能。等于 y 时启用，等于 n 时不启用（默认）。

```
#AutoIt3Wrapper_Run_Tidy=y
```

## 4、精简

（注意，仅限 2014 年 5 月 4 日之后的 AutoIt3Wrapper 支持这两个编译参数）

### ( 1 ) #AutoIt3Wrapper\_Run\_Au3Stripper

是否启用代码精简功能。等于 y 时启用，等于 n 时不启用（默认）。

```
#AutoIt3Wrapper_Run_Au3Stripper=y
```

### ( 2 ) #Au3Stripper\_Parameters

精简参数，仅当#AutoIt3Wrapper\_Run\_Au3Stripper=y 时有效。常用参数：

/sf，精简未使用的函数

/sv，精简未使用的变量

/so，等同于同时使用/sf 和/sv

/soi，等同于/so，但会保留主脚本不精简

/mo，整合脚本并删除脚本注释

```
#Au3Stripper_Parameters=/sf /sv
```

## 5、加密

(注意，仅限 2014 年 5 月 4 日之前的 AutoIt3Wrapper 支持这两个编译参数)

### (1) #AutoIt3Wrapper\_Run\_Obfuscator

是否启用代码精简、混淆功能。等于 y 时启用，等于 n 时不启用（默认）。

```
#AutoIt3Wrapper_Run_Obfuscator=y
```

### (2) #Obfuscator\_parameters

精简、混淆参数，仅当#AutoIt3Wrapper\_Run\_Obfuscator=y 时有效。常用参数：

/cs，加密字符串，0 为不加密、1 为加密（默认）

/cn，加密数字，0 为不加密、1 为加密（默认）

/cf，加密函数名，0 为不加密、1 为加密（默认）

/cv，加密变量名，0 为不加密、1 为加密（默认）

/sf，精简未使用的函数，0 为不精简（默认）、1 为精简

/sv，精简未使用的变量，0 为不精简（默认）、1 为精简

```
#Obfuscator_parameters=/cs=0 /cn=0 /cf=0 /cv=0 /sf=1 /sv=1
```

## 6、运行

### #AutoIt3Wrapper\_Run\_Before、#AutoIt3Wrapper\_Run\_After

#AutoIt3Wrapper\_Run\_Before，编译前执行某些命令

#AutoIt3Wrapper\_Run\_After，编译后执行某些命令

均可以有多行，但每行只能有一个命令。命令方式类似批处理，但可以使用下列特殊指代变量：

变量	功能
%in%	当前脚本
%out%	同#AutoIt3Wrapper_OutFile 指定的值
%out64%	同#AutoIt3Wrapper_OutFile_x64 指定的值
%icon%	同#AutoIt3Wrapper_Icon 指定的值
%scriptdir%	当前脚本目录
%scriptfile%	当前脚本文件名（无后缀名）
%fileversion%	同#AutoIt3Wrapper_Res_Fileversion 指定的值
%scitedir%	SciTE 编辑器所在目录

例如：编译前删除此前编译遗留的文件

```
#AutoIt3Wrapper_Run_Before=del %scriptdir%\my.ex~  
#AutoIt3Wrapper_Run_Before=del %scriptdir%\my.exe
```

例如：编译后将程序复制到特定位置

```
#AutoIt3Wrapper_Run_After=copy "%out%" "d:\test\test.exe"
```

### 5.3.5 AccAu3Wrapper 与其专有编译参数

AccAu3Wrapper 是基于 AutoIt3Wrapper 源代码开发的新一代 Wrapper，但不同于一般简单

的汉化修改，AccAu3Wrapper 几乎全部重写了整个软件构架，并重构了更为人性化的 UI。



(图 5-102)

### (1) 更友好的 UI

与 AutoIt3Wrapper 不同，AccAu3Wrapper 的 UI 更加直观、人性化，并便于操作。

### (2) 与 AutoIt3Wrapper 兼容

AccAu3Wrapper 兼容 AutoIt3Wrapper 的所有编译参数，并支持 AutoIt3Wrapper 的所有命令行参数，可以使用 AccAu3Wrapper 完全替代 AutoIt3Wrapper。

AccAu3Wrapper 的编译参数前缀为 "#AccAu3Wrapper\_"，但兼容 AutoIt3Wrapper 原版的 "#AutoIt3Wrapper\_" 前缀。不过对于 AccAu3Wrapper 特有的功能必须使用 "#AccAu3Wrapper\_" 前缀。

### (3) 同时兼容 Obfuscator.exe 和 Au3Stripper.exe

SciTE4AutoIt3 无论是带有 Obfuscator.exe 还是带有 Au3Stripper.exe，AccAu3Wrapper 均可自适应。甚至如果用户可以在不带 Obfuscator.exe 的版本里手动添加 Obfuscator.exe，即可实现对代码混淆功能的支持。

### (4) 代码混淆的中文支持

Obfuscator.exe 的加密混淆功能，本身并不支持中文，中文经过加密混淆会产生乱码。

专有编译参数 "#AccAu3Wrapper\_DBSSupport"，等于 y 时启用中文支持，等于 n 时关闭。

```
#AccAu3Wrapper_DBSSupport=y
```

**(5) 防傻瓜式反编译**

本功能主要用于应对一些傻瓜型反编译工具。

专有编译参数 "#AccAu3Wrapper\_Antidecompile"，等于 y 时启用防反，等于 n 时关闭。

```
#AccAu3Wrapper_Antidecompile=y
```

### 5.3.6 脚本右键菜单优化

通过前几节的学习，我们学习了 Au3 编程环境的组成与诸多编译参数。本节用于补充一个细节问题，以避免大家在运行或编译脚本时遇到一些奇怪问题。

在 5.3.1 节中，我们了解到 Au3 编程环境的基础部分由 AutoIt3 和 SciTE4AutoIt3 两部分组成。AutoIt3 是真正的 Au3 编程环境核心，SciTE4AutoIt3 是一个强大而实用的增强包，但由于二者是附加关系，所以 SciTE4AutoIt3 在安装后仍保留了 AutoIt3 的一些原本特性，例如 AutoIt3 原本右键菜单中的所有项目。

在 5.3.2 节中，我们了解到在安装了 SciTE4AutoIt3 增强包后，在 SciTE 内运行或编译脚本变为由 AutoIt3Wrapper “代理”。通过 AutoIt3Wrapper 的代理，可以在运行或编译时附加编译参数所指定的设置，例如运行方式是 x86 还是 x64。

那么，问题来了。由于右键菜单保留了 AutoIt3 的原本项目，致使直接通过右键菜单来运行脚本（“Run Script”、“Run Script (x64)” 或 “Run Script (x86)”），或直接使用右键菜单来编译脚本（“Compile Script”、“Compile Script (x64)” 或 “Compile Script (x86)”），实际上是不通过 AutoIt3Wrapper 代理的，而是直接运行或编译了程序，这将导致所有编译参数无效。

而右键菜单编译脚本，除非选择 “Compile with Options” 才是通过 AutoIt3Wrapper 代理编译，虽然可行，但也令右键菜单编译变得麻烦。

由此，如果已经决定了同时使用 AutoIt3 和 SciTE4AutoIt3，不再单独卸载 SciTE4AutoIt3，可以对脚本的右键菜单进行优化，令右键菜单的运行和编译与在 SciTE 内的运行和编译相同，均通过 AutoIt3Wrapper 代理。

安装 AutoIt3 和 SciTE4AutoIt3，假设安装路径为 D:\AutoIt3，安装选项全默认。

**(1) 找到 Au3 脚本右键菜单项目**

打开注册表：

```
HKEY_CLASSES_ROOT\AutoIt3Script\Shell
```

Au3 脚本右键菜单项目全在这里。

**(2) 修改运行方式**

找到注册表项：

```
HKEY_CLASSES_ROOT\AutoIt3Script\Shell\Run\Command
```

修改默认键值为：

```
"D:\AutoIt3\SciTE\AutoIt3Wrapper\AutoIt3Wrapper.exe" /run /prod /ErrorStdOut /in "%1"  
/UserParams
```

即使用 AutoIt3Wrapper 代理运行脚本。

### (3) 修改编译方式

找到注册表项：

HKEY\_CLASSES\_ROOT\AutoIt3Script\Shell\Compile\Command

修改默认值为：

"D:\AutoIt3\SciTE\AutoIt3Wrapper\AutoIt3Wrapper.exe" /in "%l"

### (4) 删除一些不用的项目：

注意，x86 系统下安装后没有这些项目。

x64 方式编译

HKEY\_CLASSES\_ROOT\AutoIt3Script\Shell\CompileX64

x86 方式编译

HKEY\_CLASSES\_ROOT\AutoIt3Script\Shell\CompileX86

x64 方式运行

HKEY\_CLASSES\_ROOT\AutoIt3Script\Shell\RunX64

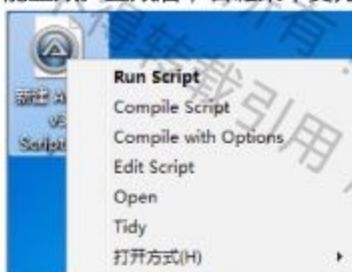
x86 方式运行

HKEY\_CLASSES\_ROOT\AutoIt3Script\Shell\RunX86

为什么可以删除 x86、x64 运行或编译模式？这个问题很简单，刚才我们已经将运行和编译的默认方法都改为了由 AutoIt3Wrapper 代理，由 AutoIt3Wrapper 代理则使用 x86 或 x64 均由编译参数来控制，无需人为指定。

### (5) 优化完毕

优化完毕后，可能需要重启才能生效。生效后，右键菜单变为：



(图 5-103)

此时的“Run Script”（运行）与“Compile Script”（编译）已与在 SciTE 内使用 F5 运行、F7 编译完全相同了。换句话说，直接运行脚本的效果与编译后运行脚本的效果完全一致，在 SciTE 内与在 SciTE 外运行或编译脚本的效果完全一致。

由此，可以充分避免由于方式不一致所造成各种问题。

额外说一点的是，如果您目前使用的是 IT 天空提供的 Au3 编程环境，开发者已经做好了上述修改，不再劳驾使用者亲自动手。IT 天空所提供的 Au3 编程环境还整合了 AccAu3Wrapper，并做好了充分的汉化工作，推荐使用。

下载地址：<http://www.itiankong.com/download/au3.html>

## 5.4 x86 与 x64

一般意义上讲，x86 即 32 位，x64 即 64 位。随着 PC 内存越来越大，x64 的 Windows 操作系统越来越普及，从而使应用程序的“x86 or x64”问题被推上台面。

Au3 原生支持将脚本编译为 x64 应用程序，所以编译虽是重点却不是难点，难点在于 x86、x64 所产生的一些现象和处理方法，本节将对此展开讨论。

### 5.4.1 x86 与 x64 的基本知识

x86 是 Intel 首先开发出的一种处理器架构，由于早期 Intel 的 CPU 命名方式为 8086、80186、80286、80386 等，所以此构架得名“80x86”，简称为“x86”。随着 PC 的发展，x86 逐渐成为了 PC 的标准构架，被广泛应用于 Intel 和 AMD 的 CPU 中。x86 亦被称之为“x86-32”，还被 Intel 称之为“IA32”(Intel Architecture, 32-bit)。

后来由于内存容量的迅速发展，突飞猛进至 G 时代，但受制于 x86 (32 位) 构架，内存最大寻址被限制在约 3.25G。AMD 抢先为 x86 提供了 64 位拓展，支持了大内存寻址，并将此构架抢先应用在自家 CPU 中推向市场，AMD 将此构架称之为“AMD64”。

受制于操作系统的市场需求，Intel 被迫接受了“AMD64”，并在扩展了一些新指令集后融入了自家产品，Intel 把这种“接受并扩展”的构架称之为 EMT64，EMT64 后又被更名为“Intel 64”。这种由 x86 拓展的 x64 (无论是 AMD64 还是 Intel 64)，后被统一称为“x86-64”，简称为“x64”。

综上：

x86，即“x86-32”，又称之为“IA32”、“Intel 32”；

x64，即“x86-64”，被 AMD 称之为“AMD64”，被 Intel 称之为“Intel 64 (EMT64)”。

说白了 x64 就是 x86 构架的一种拓展，所以对 x86 构架下的应用程序具有兼容性，这也从另一个层面上解释了为什么绝大多数 x86 应用程序可以在 x64 平台上正常运行。而由于 x64 兼容 x86 应用程序、支持大内存、节省部署与维护成本等诸多优势，x64 这种方式被广泛应用于各个方面，不仅在民用级市场，在服务器市场方面也占了非常大的比例。

那有没有一种“纯 64 位”架构？有的，Intel 安腾系列处理器支持一种叫做“IA64”的架构，与 IA32 不同，IA64 主要面向服务器市场。IA64 有别于 x86 (IA32)，与 x86 架构完全不同，所以应用程序互不兼容，IA64 同样有别于 x64，x64 说到底只是 x86 (IA32) 的一种拓展。IA64 以优异的性能在服务器方面曾活跃过一段时期，但随着 x64 的兴起，无论是硬件还是软件开发商更倾向于这种成本更低、兼容程度更好的 64 位方式，IA64 就逐步淡出了市场。

### 5.4.2 指定 x86 或 x64 方式

注意，在 x64 系统下才能指定以 x86 或 x64 方式运行或编译脚本，x86 系统下只能运行或编译

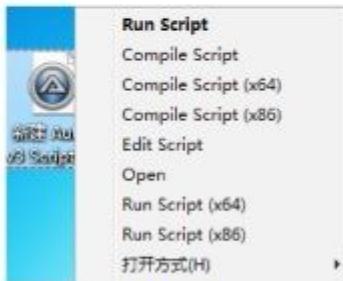
x86 脚本，切记。

### 1、AutoIt3

#### (1) SciTE 中

回顾 5.3.1 节第 1 小节，仅安装 AutoIt3 时，无论在安装时选择默认以 x86 方式还是 x64 方式，在 SciTE 中 F5 运行脚本、F7 编译脚本都是以 x86 方式进行的。若要编译 x64 应用程序，必须以右键菜单方式编译。

#### (2) 右键菜单



(图 5-104)

在右键菜单中：

“Run Script (x86)”，以 x86 方式运行脚本；

“Run Script (x64)”，以 x64 方式运行脚本。

“Compile Script (x86)”，以 x86 方式编译脚本；

“Compile Script (x64)”，以 x64 方式编译脚本。

如果在安装时，选择默认以 x86 方式，则：

“Run Script”，以 x86 方式运行脚本；

“Compile Script”，以 x86 方式编译脚本。

如果在安装时，选择默认以 x64 方式，则：

“Run Script”，以 x64 方式运行脚本；

“Compile Script”，以 x64 方式编译脚本。

### 2、AutoIt3+SciTE4AutoIt3

#### (1) SciTE 中

回顾 5.3.1 节第 2 小节，安装 AutoIt3 与 SciTE4AutoIt3 后，在 SciTE 中 F5 运行脚本、F7 编译脚本均是由 AutoIt3Wrapper 代理运行或编译的。结合 5.3.2 节中对于编译参数的探讨，可以直接由编译参数来决定是 x86 还是 x64。

当 "#AutoIt3Wrapper\_UseX64=n" 时，SciTE 内 F5 运行脚本、F7 编译脚本均是以 x86 方式；

当 "#AutoIt3Wrapper\_UseX64=y" 时，SciTE 内 F5 运行脚本、F7 编译脚本均是以 x64 方式；

#### (2) 右键菜单



(图 5-105)

安装 SciTE4AutoIt3 后右键菜单并没有太大变化，除了多出来的两个选项，其余选项与仅安装 AutoIt3 时一致。但需要特别指出的是，直接使用右键菜单运行或编译脚本会忽略掉编译参数，因为右键菜单的编译或运行并非由 AutoIt3Wrapper 代理。除非使用“Compile with Options”调出 AutoIt3Wrapper 的 UI 后执行编译，编译参数才会有效。

在安装 SciTE4AutoIt3 后，统一 SciTE 内和右键菜单的运行或编译方法参见 5.3.6 节，或者直接使用 IT 天空提供的 Au3 编程环境。如果不打算统一，那请尽可能在 SciTE 内运行或编译，这样不易出错，不会造成直接运行与编译后的运行方式不一致。

### 5.4.3 x64 系统的目录重定向

所谓 x64 系统的目录重定向，是指在 x64 系统下使用 x86 程序对 System32 目录进行访问时，会被自动重定向到 SysWOW64 目录。

假设 Windows 目录为“C:\Windows”，对于下述目录的访问：

C:\Windows\System32

会被重定向至：

C:\Windows\SysWOW64

举个例子，假设使用 x86 程序访问文件：

C:\Windows\System32\itk.txt

则会被自动变更为访问：

C:\Windows\SysWOW64\itk.txt

我们做个简单的测试（系统环境为 x64，编程环境为 AutoIt3+SciTE4AutoIt3，运行、编译脚本均在 SciTE 内以 F5、F7 快捷键进行）。

首先创建一个文本文档“itk.txt”，然后将 itk.txt 放入 C:\Windows\System32 中，最后书写下述代码并测试：

```
#Region
#AutoIt3Wrapper_UseX64=n
#AutoIt3Wrapper_Res_Language=2052
#EndRegion
```

```
_Test_Main()
Exit

Func _Test_Main()
    Local $File = "C:\Windows\System32\itk.txt"
    If FileExists($File) Then
        MsgBox(0, "", "存在")
    Else
        MsgBox(0, "", "不存在")
    EndIf
EndFunc ;==>_Test_Main
```

运行结果：



(图 5-106)

奇怪，明明 C:\Windows\System32\itk.txt 是存在的，但运行结果却是不存在？这就是重定向功能的问题，实际上程序的访问已被重定向至 C:\Windows\SysWOW64\itk.txt，当然会提示不存在。那么，怎么解决这个问题？

### 解决方法 1，以 x64 方式运行

用心的同学可能已经注意到了刚才的脚本中 "#AutoIt3Wrapper\_UseX64=n"，即指定脚本以 x86 方式运行。如果设置为 "#AutoIt3Wrapper\_UseX64=y"，则脚本将以 x64 方式运行。而重定向问题仅发生在 "x86 程序+x64 系统" 时，所以将程序改为 x64 方式运行后重定向问题将不再产生。

更改刚才的代码：

```
#Region
#AutoIt3Wrapper_UseX64=y
#AutoIt3Wrapper_Res_Language=2052
#EndRegion

_Test_Main()
Exit

Func _Test_Main()
    Local $File = "C:\Windows\System32\itk.txt"
    If FileExists($File) Then
        MsgBox(0, "", "存在")
    Else
        MsgBox(0, "", "不存在")
```

```

EndIf
EndFunc  ;==>_Test_Main

```

运行结果：



(图 5-107)

### 解决方法 2，关闭重定向

重定向功能可以通过一个 API 函数来关闭。什么是 API ? Application Programming Interface 即应用程序编程接口，是 Windows 预留给开发者的用于调用系统函数（功能）的开放式接口。关于 API 和 API 的使用方法将在后续章节进行学习，本节只提供一个写好的函数，大家直接调用即可。

```

;关闭重定向
Func _SK_Wow64FsRedirection_Disable()
    DllCall("kernel32.dll", "int", "Wow64DisableWow64FsRedirection", "int", 1)
EndFunc  ;==>_SK_Wow64FsRedirection_Disable

```

值得注意的是，这里的“关闭”重定向，只针对当前进程关闭重定向功能，而并不是关闭系统的重定向功能。从执行这个函数开始，当前进程不再受重定向功能影响，直至当前进程结束，其间其他进程仍旧受到重定向功能的制约。

明确了这些后，再次更改刚才的代码：

```

#Region
#AutoIt3Wrapper_UseX64=n
#AutoIt3Wrapper_Res_Language=2052
#EndRegion

_Test_Main()
Exit

Func _Test_Main()
    _SK_Wow64FsRedirection_Disable()

    Local $File = "C:\Windows\System32\itk.txt"
    If FileExists($File) Then
        MsgBox(0, "", "存在")
    Else
        MsgBox(0, "", "不存在")
    Endif

EndFunc  ;==>_Test_Main

```

```
Func _SK_Wow64FsRedirection_Disable()
    DllCall("kernel32.dll", "int", "Wow64DisableWow64FsRedirection", "int", 1)
EndFunc  ;==>_SK_Wow64FsRedirection_Disable
```

运行结果：



(图 5-108)

可以发现，以 x86 方式运行并关闭重定向后，可以正常访问 system32 目录。

### 其他问题：关于@SystemDir

在 x64 系统下（假设 Windows 目录为 “C:\Windows”）：

当脚本以 x86 方式运行时，@SystemDir 的值为 “C:\Windows\SysWOW64”；

当脚本以 x64 方式运行时，@SystemDir 的值为 “C:\Windows\system32”。

而值得注意的是，当脚本以 x86 方式运行，并已关闭了重定向功能后，@SystemDir 的值仍为 “C:\Windows\SysWOW64”，并不会变为 “C:\Windows\system32”。很多 Au3er 在这个问题上出现错误，所以如果是“x86 方式+关闭重定向”的模式，System32 的路径建议写死（形如“解决方法 2”中的那段代码），而不是使用宏。

### 5.4.4 x64 系统的注册表重定向

所谓 x64 系统的注册表重定向，是指在 x64 系统下使用 x86 程序对注册表进行操作时，会被自动重定向到 x86 结点上。

举个例子，我们要在 “HKEY\_LOCAL\_MACHINE\SOFTWARE” 下创建一个名为 “Test” 的项，并设置其默认键值为 “Test”。回顾一下 RegWrite 函数的语法，很容易可以写出下面代码：

```
#Region
#AutoIt3Wrapper_UseX64=n
#AutoIt3Wrapper_Res_Language=2052
#EndRegion

_Test_Main()
Exit

Func _Test_Main()
    RegWrite("HKLM\SOFTWARE\Test", "", "REG_SZ", "Test")
EndFunc  ;==>_Test_Main
```

（其中 “HKLM” 是 HKEY\_LOCAL\_MACHINE”的简写，如果忘记了，参见 4.7 节）

运行上述脚本，然后打开注册表编辑器，按照注册表路径进行查找，奇怪的是竟然无法找到：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Test
```

看来，果真遇到重定向问题了，那 Test 项究竟被写到哪去了？莫急，仔细查找会发现：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Test
```

原来是被重定向到 Wow6432Node 去了。

那么，怎么解决注册表的重定向问题？

### 解决方法 1，以 x64 方式运行

这个解决方法与解决目录重定向问题时一致，设置 "#AutoIt3Wrapper\_UseX64=y" 使脚本以 x64 方式运行。

```
#Region
#AutoIt3Wrapper_UseX64=y
#AutoIt3Wrapper_Res_Language=2052
#EndRegion

_Test_Main()
Exit

Func _Test_Main()
    RegWrite("HKLM\SOFTWARE\Test", "", "REG_SZ", "Test")
EndFunc  ;==>_Test_Main
```

运行后下述键、键值已被正确写入：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Test
```

### 解决方法 2，为根键添加 64 后缀

x86 方式运行时解决方法也很简单，只需要在根键后增加 64 后缀即可。例如将 HKLM 写为 HKLM64 即可解决 HKEY\_LOCAL\_MACHINE 根键下的重定向问题。

```
#Region
#AutoIt3Wrapper_UseX64=n
#AutoIt3Wrapper_Res_Language=2052
#EndRegion

_Test_Main()
Exit

Func _Test_Main()
    RegWrite("HKLM64\SOFTWARE\Test", "", "REG_SZ", "Test")
EndFunc  ;==>_Test_Main
```

同样的，HKU、HKCU、HKCR、HKCC 写成 HKU64、HKCU64、HKCR64、HKCC64 即可。

注意：解决方法 1、解决方法 2 对于 RegRead、RegDelete 等注册表相关函数均有效。

### 其他拓展：x86 程序自适应 x86、x64 系统的注册表

写一段程序，声明几个全局常量用于代表注册表根键，运行一个函数根据系统位宽来决定根键后是否有“64”后缀。

```
#include-once

;-----  
; 根键全局常量：  
;   $_HKLM，替代 HKLM 或 HKLM64  
;   $_HKU，替代 HKU 或 HKU64  
;   $_HKCU，替代 HKCU 或 HKCU64  
;   $_HKCR，替代 HKCR 或 HKCR64  
;   $_HKCC，替代 HKCC 或 HKCC64  
;  
Global Const $_HKLM = _SK_RegRootKey("HKLM")  
Global Const $_HKU = _SK_RegRootKey("HKU")  
Global Const $_HKCU = _SK_RegRootKey("HKCU")  
Global Const $_HKCR = _SK_RegRootKey("HKCR")  
Global Const $_HKCC = _SK_RegRootKey("HKCC")  
  
;  
; 函数：  
;   _SK_GetRegRootKey  
; 功能：  
;   根据系统位宽决定注册表根键是否有“64”后缀  
; 参数：  
;   $RootKey，指定根键名（合法范围：HKLM、HKU、HKCU、HKCR、HKCC）  
; 返回值：  
;   成功返回根键名，失败返回空  
;   成功时设置@error 值为 0，@extended：  
;     1，系统为 x64，根键有 64 后缀  
;     2，系统为 x86，根键无 64 后缀  
;   失败时设置@error 值为非 0，@error：  
;     1，根键简写不在合法范围  
;     2，无法判定系统位宽  
;  
Func _SK_GetRegRootKey($RootKey)  
  If $RootKey = "HKLM" Or_  
    $RootKey = "HKU" Or_  
    $RootKey = "HKCU" Or_  
    $RootKey = "HKCR" Or_  
    $RootKey = "HKCC" Then  
      Switch @OSArch  
        Case "x64"  
          Return SetError(0, 1, $RootKey & "64")  
        Case "x86"  
          Return SetError(0, 2, $RootKey)
```

```

Case Else
    Return SetError(2, 0, "")
EndSwitch
Else
    Return SetError(1, 0, "")
EndIf
EndFunc  ;==>_SK_GetRegRootKey

```

将上述代码保存为一个 au3 文件，待使用时 Include 这个 au3 文件，根键全局常量会自动根据系统位宽取值。之后，使用根键全局常量来替代注册表根键即可。如：

```
RegWrite("HKLM\SOFTWARE\Test", "", "REG_SZ", "Test")
```

可以写成：

```
RegWrite($_HKLM & "\SOFTWARE\Test", "", "REG_SZ", "Test")
```

这样即完成了 x86 程序自适应 x86、x64 系统的注册表。

## 5.5 获取运行参数

代码在编译为应用程序后，不能直接通过修改代码的方式来修改程序的运行方式。对于编译后的程序，通过外部手段使其以不同运行方式或实现不同的功能，需要对程序发送一个“通知”。通知的方法很多，最直接的方法就是使用运行参数。

运行参数其实并不陌生，很多程序都有运行参数。例如一个软件安装包，可以使用 “/s” 参数来通知其以静默方式安装；7za.exe 可以通过参数来控制是压缩还是解压、压缩哪个文件、解压到哪里去；Ghost.exe 可以通过参数来控制是备份还是还原、备份哪个分区、还原哪个映像。

怎么为我们自己的程序添加运行参数？本节一起来学习学习。

### 5.5.1 关于运行参数

关于运行参数，有一些固定书写的习惯：

- ( 1 ) 参数一般以 “/” 或 “-” 开头，例如 “/silent”、“-auto”；
- ( 2 ) 一个程序的运行参数一般只采用 “/” 方式或只采用 “-” 方式，或同时兼容二者，但绝不会混用二者（一个参数用 “/”，另一个参数用 “-”）；
- ( 3 ) 虽然参数可采用的写法比较宽松，但一般仅使用英文或数字，且一般情况下 Windows 应用程序的参数并不区分大小写；
- ( 4 ) 同时使用多个参数时，参数之间以空格作为间隔，如 “/auto /clean”；
- ( 5 ) 如果某个参数的值包含空格，则一般采用英文引号进行包含，否则空格会认为是参数间隔。

```
D:\Install\Test.exe /s /d="d:\program files\"
```

作为初学者，先以这些习惯约束自己。

### 5.5.2 获取运行参数

程序是怎样获取运行参数的呢？例如一个程序 Test.exe，当运行 “Test.exe /a /c” 时，Test.exe 是怎么“知道”给了它 “/a /c” 两个参数？

Au3 中有一个特殊变量 \$Cmdline，是脚本默认声明的，也就是说只要脚本在运行这个变量就已经被声明了，无需人为操作。而变量 \$Cmdline 的作用，就是用来存储运行参数。

\$Cmdline 是一个数组变量，\$Cmdline[0] 用于保存参数的数量，\$Cmdline[1] 是第 1 个参数，\$Cmdline[2] 是第 2 个参数……\$Cmdline[n] 是第 n 个参数。假设程序运行时没有运行参数，则 \$Cmdline[0] 的值为 0，数组的大小为 1；假设程序运行时参数为 “/a /c”，则 \$Cmdline[0] 的值为 2，\$Cmdline[1] 的值为 “/a”，\$Cmdline[2] 的值为 “/c”，数组的大小为 3。

可以书写一个名为 Test.au3 的脚本：

```
#include <Array.au3>
_ArrayDisplay($Cmdline)
```

将此脚本编译为 Test.exe，直接运行或运行 “Test.exe /a /c” 分别观察 \$Cmdline 数组。

运用 \$Cmdline 数组，就可以易如反掌的获取运行参数了。

另外，\$Cmdline 数组还有一个特殊的性质，即在获取参数时会自动忽略参数内的英文引号，例如参数为下述形式时：

D:\Install\Test.exe /s /d="d:\program files\"

\$Cmdline[0] -> 2

\$Cmdline[1] -> /s

\$Cmdline[2] -> /d=d:\program files\

可见，参数 2 中路径两侧的英文引号已经在获取时被自动忽略掉了。

另外，为了测试运行参数更为方便（总不能每次都编译了再测试），SciTE 提供参数测试功能，但最多只能设置 4 个参数。在 SciTE 中使用 Shift+F8 快捷键，调出参数设置框，填入参数即可。



(图 5-109)

### 5.5.3 使用运行参数

#### 1、单参数

使用一个参数来决定脚本的不同运行方式，这是最基本的参数控制模式，举例：

```

_Test_Main()
Exit

Func _Test_Main()
If $Cmdline[0] = 1 Then
    Switch $Cmdline[1]
        Case "/auto"
            MsgBox(0, "", "自动")
        Case "/manual"
            MsgBox(0, "", "手动")
        Case "/disable"
            MsgBox(0, "", "禁用")
    EndSwitch
Endif
EndFunc  ;==> _Test_Main

```

当运行参数依次为 /auto、/manual、/disable 时，运行依次为下图从左至右：



(图 5-110)

(图 5-111)

(图 5-112)

## 2、多参数

有时多个参数是配合起来使用的，例如一个参数指定运行方式，另一个参数指定路径，举例：

```

_Test_Main()
Exit

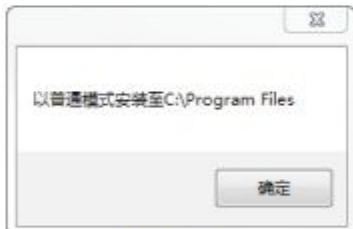
Func _Test_Main()
Local $RunMode = 1
Local $Path = "C:\Program Files"

If $Cmdline[0] > 0 Then
    Local $i
    For $i = 1 To $Cmdline[0]
        Select
            Case $Cmdline[$i] = "/s"
                $RunMode = 0
            Case StringLeft($Cmdline[$i], 3) = "/d="
                $Path = StringTrimLeft($Cmdline[$i], 3)
        EndSelect
    Next
EndIf

```

```
If $RunMode = 1 Then  
    MsgBox(0, "", "以普通模式安装至" & $Path)  
ElseIf $RunMode = 0 Then  
    MsgBox(0, "", "以静默模式安装至" & $Path)  
EndIf  
EndFunc ;==>_Test_Main
```

直接运行无参数



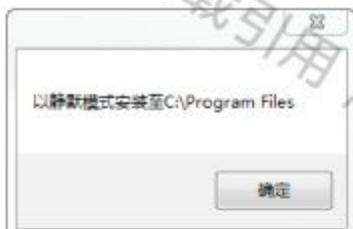
(图 5-113)

运行参数 : /d="D:\My Softwares"



(图 5-114)

运行参数 : /s



(图 5-115)

运行参数 : /s /d="D:\My Softwares"

或 : /d="D:\My Softwares" /s



(图 5-116)

注意 :

(1) 用户在书写参数时是不按顺序的，所以必须——检测参数；

(2) 对于 “/d=d:\softwares” 这种参数，使用正则表达式来检测会更加准确，有关正则表达式

的知识将在之后章节讲解；

- (3) 参数越多，所能配合出的模式越多，注意考虑每种情况；
- (4) 如果参数没有指定某个项目，这个项目必须要有默认值。

## 5.6 文件拖拽至 UI

将文件直接拖拽进 UI 是一种可以增进用户体验的操作方式。例如转换视频时，要求用户浏览目录选中源视频，或用户直接可以将视频拖拽进源视频的输入框，明显后者的体验要好很多。而在 Au3 中怎样实现文件拖拽呢？

### 1、关于文件拖拽的几个关键点

文件拖拽在 Au3 中不需要复杂的代码即可实现，大家只需牢记几个关键点：

(1) 窗体必须能够“接受文件”。窗体能够接受文件是将文件拖拽至 UI 的大前提，这就像要把货搬进屋里得先打开门一样。令窗体接受文件在 3.4 节中简要讲解过，只需要将窗体的扩展样式设置为“\$WS\_EX\_ACCEPTFILES”就可以了。

(2) 控件必须能够“接受拖拽”。被拖拽的文件需要拖拽至某控件才能完成拖拽，这就像把货搬进屋总得找个货架子放，总不能直接扔地上。为控件加个状态“\$GUI\_DROPACCEPTED”就可以了。

(3) 满足(1)(2)后，拖拽文件至 UI 会产生一个系统事件，利用消息循环模式可以获取到该事件。就像关闭窗体时会触发“\$GUI\_EVENT\_CLOSE”一样，拖拽会触发“\$GUI\_EVENT\_DROPPED”事件。

(4) 如果有多个控件接受拖拽，那么怎么判断文件被拖拽到了哪个控件？不要担心，当拖拽发生时，文件被拖拽至哪个控件，Au3 会自动令宏“@GUI\_DropId”的值等于那个控件的 ID。

(5) 最后一个问题，怎么获取被拖拽文件的路径？这一点在 Au3 下也很智能，当拖拽发生时，Au3 会自动令宏“@GUI\_DragFile”的值等于被拖拽文件路径，读取这个宏就可以了。

综上，实现文件拖拽的所有关键点 Au3 已经帮我们准备好了，我们只需要把它们利用和整合起来就可以了。不过本节只和大家讨论单文件的拖拽，令大家对文件拖拽有个基本的认识，多文件拖拽需要高级控件支持，我们将在之后章节再行学习。

### 2、拖拽文件到 UI

先看一段简单的代码，然后一同分析与学习。

```
#include <ButtonConstants.au3>
#include <EditConstants.au3>
#include <GUIConstantsEx.au3>
#include <StaticConstants.au3>
#include <WindowsConstants.au3>

Opt("MustDeclareVars", 1)
```

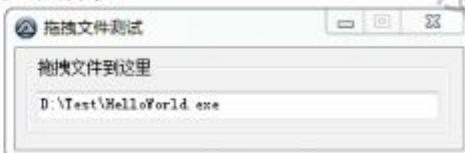
```
Global $gu_Form1 = GUICreate("拖拽文件测试", 331, 75, -1, -1, -1, $WS_EX_ACCEPTFILES)
Global $gu_Group1 = GUICtrlCreateGroup("拖拽文件到这里", 8, 8, 313, 57)
Global $gu_Input1 = GUICtrlCreateInput("", 16, 32, 297, 21)
GUICtrlCreateGroup("", -99, -99, 1, 1)

_Demo_Main()
Exit

Func _Demo_Main()
    GUICtrlSetState($gu_Input1, $GUI_DROPACCEPTED)
    GUISetState(@SW_SHOW)

    Local $a_Time, $Size, $Version
    Local $nMsg
    While 1
        $nMsg = GUIGetMsg()
        Switch $nMsg
            Case $GUI_EVENT_CLOSE
                Exit
            Case $GUI_EVENT_DROPPED
                If @GUI_DropId = $gu_Input1 Then
                    GUICtrlSetData($gu_Input1, @GUI_DragFile)
                Endif
        EndSwitch
    WEnd
EndFunc ;==>_Demo_Main
```

运行图 ( 将文件拖拽至输入框后 ) :



( 图 5-117 )

在这段代码中，首先为窗体设置了可接受文件：

```
GUICreate("拖拽文件测试", 331, 75, -1, -1, -1, $WS_EX_ACCEPTFILES)
```

而后令文本框接受拖拽：

```
GUICtrlSetState($gu_Input1, $GUI_DROPACCEPTED)
```

完成了接受文件拖拽的基本支持，然后判断拖拽事件的发生：

```
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        ...
        Case $GUI_EVENT_DROPPED
        ...

```

```
EndSwitch
WEnd
```

当拖拽发生后，判断文件被拖拽至了哪个控件，并将控件值设置为被拖拽的文件路径：

```
Case $GUI_EVENT_DROPPED
If @GUI_DropId = $gu_Input1 Then
    GUICtrlSetData($gu_Input1, @GUI_DragFile)
EndIf
```

整体一气呵成，非常流畅。如果没有理解，请对比第1小节内容、第2小节代码和上述代码分析反复揣摩，拖拽虽然是一个很“酷”的功能，但却非常容易实现。

这里有个小细节，细心观察的朋友会发现，当拖拽完成时文本框控件的值已经被设置了被拖拽文件的完成路径，为什么还要使用下列代码来设置一次文本框的内容？

```
GUICtrlSetData($gu_Input1, @GUI_DragFile)
```

有兴趣的读者可以去掉这行代码试试，不要只拖拽一次，要多次拖拽不同文件试一试，你会发现后拖拽的内容并不会覆盖先拖拽的内容，而是一次次的被叠加写入文本框。

### 3、拖拽后触发更多操作

在拖拽完成后，有时需要立刻执行某些代码，例如拖拽一个视频则立即检查视频的大小、格式、编码等，拖拽一个文件则立刻检查文件的大小、版本、SHA1值等。下面我们举一个简单的例子来描述一下这种“拖拽即触发”的形式。

```
#include <ButtonConstants.au3>
#include <EditConstants.au3>
#include <GUIConstantsEx.au3>
#include <StaticConstants.au3>
#include <WindowsConstants.au3>

Opt("MustDeclareVars", 1)

Global $gu_Form1 = GUICreate("拖拽文件测试", 331, 196, -1, -1, -1, $WS_EX_ACCEPTFILES)
Global $gu_Group1 = GUICtrlCreateGroup("拖拽文件到这里", 8, 8, 313, 57)
Global $gu_Input1 = GUICtrlCreateInput("", 16, 32, 297, 21)
GUICtrlCreateGroup("", -99, -99, 1, 1)
Global $gu_Group2 = GUICtrlCreateGroup("文件信息", 8, 72, 313, 105)
Global $gu_Label1 = GUICtrlCreateLabel("修改时间", 16, 99, 52, 17)
Global $gu_Input2 = GUICtrlCreateInput("", 72, 96, 241, 21)
Global $gu_Label2 = GUICtrlCreateLabel("文件大小", 16, 123, 52, 17)
Global $gu_Input3 = GUICtrlCreateInput("", 72, 120, 241, 21)
Global $gu_Label3 = GUICtrlCreateLabel("文件版本", 16, 147, 52, 17)
Global $gu_Input4 = GUICtrlCreateInput("", 72, 144, 241, 21)
GUICtrlCreateGroup("", -99, -99, 1, 1)

_Demo_Main()
```

```
Exit

Func _Demo_Main()
    GUICtrlSetState($gu_Input1, $GUI_DROPACCEPTED)
    GUISetState(@SW_SHOW)

    Local $a_Time, $Size, $Version
    Local $nMsg
    While 1
        $nMsg = GUIGetMsg()
        Switch $nMsg
            Case $GUI_EVENT_CLOSE
                Exit
            Case $GUI_EVENT_DROPPED
                If @GUI_DropId = $gu_Input1 Then
                    GUICtrlSetData($gu_Input1, @GUI_DragFile)

                    $a_Time = FileGetTime(@GUI_DragFile, 0)
                    GUICtrlSetData($gu_Input2, _
                        $a_Time[0] & "年" & $a_Time[1] & "月" & $a_Time[2] & "日" & _
                        $a_Time[3] & ":" & $a_Time[4] & ":" & $a_Time[5])

                    $Size = FileGetSize(@GUI_DragFile)
                    GUICtrlSetData($gu_Input3, $Size / 1024 & " KB")

                    $Version = FileGetVersion(@GUI_DragFile)
                    GUICtrlSetData($gu_Input4, $Version)
                EndIf
            EndSwitch
        WEnd
    EndFunc ;==>_Demo_Main
```

运行图(将文件拖拽至输入框后):



(图 5-118)

这段代码是在第 2 小节的代码基础上加以修改实现的。本例增加了获取文件修改时间、文件大小以及文件版本的功能。

对于“拖拽即触发”的问题，关键是要理解触发点是在哪里。平时我们按一个按钮实现一个功能，

触发点是按钮的单击动作，而拖拽的触发点则是文件拖拽完成。判断拖拽事件、被拖拽目标控件 ID、被拖拽文件的方法我们已经熟悉，所以在被拖拽完成时执行相应命令即可：

```

Case $GUI_EVENT_DROPPED
    If @GUI_DropId = $gu_Input1_Then
        GUICtrlSetData($gu_Input1, @GUI_DragFile)

        $a_Time = FileGetTime(@GUI_DragFile, 0)
        GUICtrlSetData($gu_Input2, _
            $a_Time[0] & "年" & $a_Time[1] & "月" & $a_Time[2] & "日" & _
            $a_Time[3] & ":" & $a_Time[4] & ":" & $a_Time[5])

        $Size = FileGetSize(@GUI_DragFile)
        GUICtrlSetData($gu_Input3, $Size / 1024 & " KB")

        $Version = FileGetVersion(@GUI_DragFile)
        GUICtrlSetData($gu_Input4, $Version)
    EndIf

```

上述代码是在拖拽事件发生、判定被拖拽目标控件为文本框后，使用 FileGetTime 函数获取了被拖拽文件修改时间，使用 FileGetSize 函数获取了文件体积，使用 FileGetVersion 函数获取了文件版本，并将这些信息展示出来，从而实现了“拖拽即触发”。

关于文件的拖拽就先讲这么多，希望能给大家带来抛砖引玉的效果，大家利用拖拽方法发挥自己的创造力吧！

## 5.7 实例练习：我的 7z

7z ( 7-zip ) 是一种主流的高效压缩格式，具有相当高的压缩比，7z 在 GNU 协议下开放源代码，日常应用十分广泛。7za.exe 是 7z 官方提供的独立程序，可通过命令行方式调用，以实现压缩、解压工作。这种命令行方式便于我们使用 Au3 来调用，以实现我们所需的功能。另一种调用 7z 的方式是通过 dll ( 动态链接库 )，这种方式我们将在之后章节学习，本节的重点是为 7za.exe 设计一个简洁的 GUI 来完成压缩工作。

### 分析

( 1 ) 7za.exe 的命令很多，用于压缩的一般命令是：

```
7za.exe a <压缩包> <源文件> [-mx=<压缩等级>]
```

其中

- “a” 命令用于标注当前执行的是压缩工作；
- “<压缩包>”，即压缩后将压缩包保存至什么位置，相对路径或绝对路径均可；

- “<源文件>”，即被压缩的文件或目录的路径，相对路径或绝对路径均可；
- “-mx” 压缩等级（可选参数），有 1、3、5、7、9 共五个等级，默认为 5。

说明：

- 路径中如果带有空格，一定切记两侧用英文引号进行包含；
- 有意思的是，即便路径中没有空格，路径两侧也是可以使用英文引号的；
- 压缩等级是可选参数，类似的可选参数有很多，有兴趣可参见 7z 帮助文档。

(2) 根据(1)中的参数，我们的 UI 应该有“压缩包”、“源文件”两个输入路径的地方，输入路径使用文本框；“压缩等级”是固定值，令用户选择即可，使用单选框。一般用户的习惯是先设定“源文件”，所以“源文件”先于“压缩包”进行设定，且由于一般用户不用选择太多压缩等级，提供三个压缩等级(1、5、9)即可。

(3) 如果需要浏览选择源文件，再手动设定保存到什么位置、压缩包名称是什么，可以说已经不符合我们“便捷”的初衷了。便捷选中源文件，可以利用文件拖拽的形式，而当拖拽发生时根据所拖拽文件的路径，自动创建位于源文件同目录下的以源文件名命名的压缩包，这样就一气呵成的完成了选中源文件和设定压缩包保存位置。

(4) 源文件有文件和目录两种形式，这两种形式在生成默认的压缩包名称时有一定区别：

如果是文件：

D:\Test\Test.exe

则压缩包形式为：

D:\Test\Test.7z

而如果是目录：

D:\Test

则压缩包形式为：

D:\Test.7z

对于这两种情况，需要判断源文件到底是文件还是目录，然后根据判断生成不同的压缩包名。判断源文件是文件或是目录，使用函数 FileGetAttrib(4.5.3节)即可。

(5) 运行参数。为了进一步拓展使用，我们可以为自己的程序设定运行参数，以便于调用。

/src=<源文件>，用于设定源文件路径；

/dest=<压缩包>，用于设定压缩包保存位置；

/lv=<压缩等级>，用于设定压缩等级；

/auto，使用此参数时程序自动进行，不再等用户确认。

(6) 7za.exe 分为 x86 与 x64 两种，虽然在 x64 系统下可以兼容使用 x86 的程序，但我们仍希望在 x64 系统下使用 x64 的 7za.exe 提高压缩效率。在装载 7za.exe 时要对系统位宽进行一个判定，从而装载不同的 7za.exe。

代码

```
#NoTrayIcon  
  
#Region  
#AutoIt3Wrapper_OutFile=My7z.exe
```

```
#AutoIt3Wrapper_UseX64=n
#AutoIt3Wrapper_Res_Comment=我的 7z
#AutoIt3Wrapper_Res_Description=我自制的 7z 压缩程序
#AutoIt3Wrapper_Res_Fileversion=1.0.0.1
#AutoIt3Wrapper_Res_LegalCopyright=Copyright 2014 Skyfree
#AutoIt3Wrapper_Res_Language=2052
#AutoIt3Wrapper_Res_requestedExecutionLevel=None
#AutoIt3Wrapper_Run_Tidy=Y
#EndRegion

#include <ButtonConstants.au3>
#include <EditConstants.au3>
#include <GUIConstantsEx.au3>
#include <StaticConstants.au3>
#include <WindowsConstants.au3>

#include <Array.au3>

Opt("MustDeclareVars", 1)

Global $gu_My7z_Form1
Global $gu_My7z_Group1, $gu_My7z_Label1, $gu_My7z_Label2, $gu_My7z_Input1,
$gu_My7z_Input2
Global $gu_My7z_Group2, $gu_My7z_Radio1, $gu_My7z_Radio2, $gu_My7z_Radio3
Global $gu_My7z_Button1, $gu_My7z_Button2

_My7z_Main()
Exit

Func _My7z_Main()
;获取运行参数
Local $a_Parm = _My7z_GetParm()

;如果运行参数设置了自动运行，则不创建 UI，直接压缩
Local $Error
If $a_Parm[0] Then
;根据运行参数进行压缩操作
_My7z_Compress($a_Parm[1], $a_Parm[2], $a_Parm[3])
;根据返回的错误值提示用户
$Error = @error
_My7z_ErrorMessage($Error)
;自动运行后直接退出
Exit
Endif
```

```
;正常模式
;启动 UI
_My7z_GuiCreate()
;设置 UI (部分值根据运行参数进行设定)
_My7z_GuiSet($a_Parm)
;显示 UI
GUISetState(@SW_SHOW, $gu_My7z_Form1)

Local $r
Local $Src, $Dest, $Lv
Local $nMsg
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE, $gu_My7z_Button2
            ;关闭窗体或单击退出按钮，准备退出程序
            ;询问用户是否真的要退出，防止误关闭
            $r = MsgBox(1 + 64, "信息", "确定要退出吗？")
            If $r=1 Then Exit
        Case $gu_My7z_Button1
            ;单击“压缩”按钮，准备进行压缩

            ;从文本框 1 读取源文件路径
            $Src = GUICtrlRead($gu_My7z_Input1)

            ;从文本框 2 读取压缩包保存位置
            $Dest = GUICtrlRead($gu_My7z_Input2)

            ;根据单选框的选择决定压缩等级
            Select
                Case GUICtrlRead($gu_My7z_Radio1) = $GUIL_CHECKED
                    $Lv = 1
                Case GUICtrlRead($gu_My7z_Radio2) = $GUIL_CHECKED
                    $Lv = 5
                Case GUICtrlRead($gu_My7z_Radio3) = $GUIL_CHECKED
                    $Lv = 9
            EndSelect
            ;开始压缩，隐藏主窗体
            GUISetState(@SW_HIDE, $gu_My7z_Form1)
            ;进行压缩
            _My7z_Compress($Src, $Dest, $Lv)
            ;利用变量保存 _My7z_Compress 函数运行的错误值
            $Error = @error
            ;压缩结束，显示主窗体
```

```

GUISetState(@SW_SHOW, $gu_My7z_Form1)
;根据 _My7z_Compress 的错误值提示用户可能发生的错误
(My7z_ErrorMessage($Error)
; _My7z_Compress 的错误值为 0 时证明压缩没有遇到错误，则压缩完成
If Not ($Error) Then MsgBox(0 + 64, "信息", "压缩完成!")

Case $GUI_EVENT_DROPPED
;响应拖拽事件
If @GUI_DropId = $gu_My7z_Input1 Then
;拖拽发生，且文件拖拽至文本框 1

;设置文本框 1 的值为被拖拽的文件
GUICtrlSetData($gu_My7z_Input1, @GUI_DragFile)

;判断被拖拽文件是否是文件夹
If StringInStr(FileGetAttrib(@GUI_DragFile), "D") Then
;如果是文件夹，则压缩包名为 文件夹.7z
GUICtrlSetData($gu_My7z_Input2, @GUI_DragFile & ".7z")
Else
;如果是文件，则压缩包名为 文件.7z
GUICtrlSetData($gu_My7z_Input2, _
StringMid(@GUI_DragFile, 1, _
StringInStr(@GUI_DragFile, ".", 0, -1) - 1) & ".7z")
EndIf
EndIf
EndSwitch
WEnd
EndFunc ;==>_My7z_Main

;创建 UI
Func _My7z_GuiCreate()
;创建窗体与控件
$gu_My7z_Form1 = GUICreate("我的 7z - by Skyfree", 427, 122, -1, -1, -1, $WS_EX_ACCEPTFILES)
$gu_My7z_Group1 = GUICtrlCreateGroup("", 8, 0, 409, 73)
$gu_My7z_Label1 = GUICtrlCreateLabel("要压缩什么拖到这里", 16, 19, 112, 17)
$gu_My7z_Label2 = GUICtrlCreateLabel("压缩后将保存到这里", 16, 44, 112, 17)
$gu_My7z_Input1 = GUICtrlCreateInput("", 136, 16, 273, 21)
$gu_My7z_Input2 = GUICtrlCreateInput("", 136, 40, 273, 21)
GUICtrlCreateGroup("", -99, -99, 1, 1)
$gu_My7z_Group2 = GUICtrlCreateGroup("", 8, 72, 265, 41)
$gu_My7z_Radio1 = GUICtrlCreateRadio("速度最快", 16, 87, 73, 17)
$gu_My7z_Radio2 = GUICtrlCreateRadio("中规中矩", 104, 87, 73, 17)
$gu_My7z_Radio3 = GUICtrlCreateRadio("体积最小", 192, 87, 73, 17)
GUICtrlCreateGroup("", -99, -99, 1, 1)
$gu_My7z_Button1 = GUICtrlCreateButton("压缩", 312, 79, 49, 33)

```

```
$gu_My7z_Button2 = GUICtrlCreateButton("退出", 368, 79, 49, 33)
EndFunc ;==>_My7z_GuiCreate

;设置 UI
Func _My7z_GuiSet($a_Parm)

;参数 1 即源文件路径，如果这个参数有值则设定文本框 1 的值
If $a_Parm[1] <> "" Then GUICtrlSetData($gu_My7z_Input1, $a_Parm[1])

;参数 2 即压缩包路径，如果这个参数有值则设定文本框 2 的值
If $a_Parm[2] <> "" Then GUICtrlSetData($gu_My7z_Input2, $a_Parm[2])

;参数 3 即压缩等级，如果这个参数有值则设定对应的单选框为选中
Switch $a_Parm[3]
    Case 1
        GUICtrlSetState($gu_My7z_Radio1, $GUI_CHECKED)
    Case 5
        GUICtrlSetState($gu_My7z_Radio2, $GUI_CHECKED)
    Case 9
        GUICtrlSetState($gu_My7z_Radio3, $GUI_CHECKED)
    Case Else
        ;如果参数设置有问题，或没有设置此参数，则默认单选框 2 为选中
        GUICtrlSetState($gu_My7z_Radio2, $GUI_CHECKED)
EndSwitch

;设置文本框 1 接受拖放
GUICtrlSetState($gu_My7z_Input1, $GUI_DROPACCEPTED)
EndFunc ;==>_My7z_GuiSet

;压缩
Func _My7z_Compress($Src, $Dest, $Lv)
    ;如果源文件为空或不存在，则返回 0 并设置错误值为 1
    If $Src = "" Or Not (FileExists($Src)) Then Return SetError(1, 0, 0)

    ;如果压缩包为空，则返回 0 并设置错误值为 2
    If $Dest = "" Then Return SetError(2, 0, 0)

    ;如果压缩等级设置有误，则返回 0 并设置错误值为 3
    If Not ($Lv >= 0 And $Lv <= 9) Then Return SetError(3, 0, 0)

    ;装载 7za.exe
    Local $7zaExe = @TempDir & "\7za.exe"
    _My7z_FileInstall_7z($7zaExe)
    ;装载失败，则返回 0 并设置错误值为 4
    If @error Then Return SetError(4, 0, 0)
```

```

;如果压缩包已存在，则删除压缩包
If FileExists($Dest) Then FileDelete($Dest)

;开始压缩
Local $Cmd = $7zaExe & ' a "' & $Dest & '" "' & $Src & '" -mx=' & $Lv
RunWait($Cmd)

;压缩完成后删除 7za.exe
FileDelete($7zaExe)

If FileExists($Dest) Then
    ;如果压缩包存在，则基本判定压缩完成，返回 1 并设置错误值为 0
    Return SetError(0, 0, 1)
Else
    ;如果压缩包不存在，则压缩失败，返回 0 并设置错误值为 5
    Return SetError(5, 0, 0)
EndIf
EndFunc  ;==> My7z_Compress

;装载 7z.exe 文件
Func _My7z_FileInstall_7z($7zaExe)
    Local $r
    ;根据系统位宽判断释放 x86 的 7za.exe 还是 x64 的 7za.exe
    Switch @OSArch
        Case "x86"
            ;系统为 x86
            $r = FileInstall(".\bin\7za.x86.exe", $7zaExe, 1)
            If $r Then
                ;释放成功，返回 1 并设置错误值为 0
                Return SetError(0, 0, 1)
            Else
                ;释放失败，返回 0 并设置错误值为 1、扩展值为 1
                Return SetError(1, 1, 0)
            EndIf
        Case "x64"
            ;系统为 x64
            $r = FileInstall(".\bin\7za.x64.exe", $7zaExe, 1)
            If $r Then
                ;释放成功，返回 1 并设置错误值为 0
                Return SetError(0, 0, 1)
            Else
                ;释放失败，返回 0 并设置错误值为 1、扩展值为 2
                Return SetError(1, 2, 0)
            EndIf
    EndSwitch
EndFunc

```

```
Case Else
    ;系统位宽判定失败，返回 0 并设置错误值为 2
    Return SetError(2, 0, 0)
EndSwitch
EndFunc  ;==>_My7z_FileInstall_7z

;获取参数
Func _My7z_GetParm()
Local $a_Parm[4]
If $Cmdline[0] > 0 Then
    ;如果有运行参数
    Local $i
    For $i = 1 To $Cmdline[0]
        ;逐个检索运行参数，有符合要求的则保存
        Select
            Case $Cmdline[$i] = '/auto'
                ;自动运行
                $a_Parm[0] = 1
            Case StringLeft($Cmdline[$i], 5) = "/src="
                ;源文件的完整路径
                $a_Parm[1] = StringTrimLeft($Cmdline[$i], 5)
            Case StringLeft($Cmdline[$i], 6) = "/dest="
                ;保存压缩包的完整路径
                $a_Parm[2] = StringTrimLeft($Cmdline[$i], 6)
            Case StringLeft($Cmdline[$i], 4) = "/lv="
                ;压缩等级
                ;将压缩等级的英文转化为实际压缩等级
                Switch StringTrimLeft($Cmdline[$i], 4)
                    Case "fastest"
                        $a_Parm[3] = 1
                    Case "normal"
                        $a_Parm[3] = 5
                    Case "ultra"
                        $a_Parm[3] = 9
                    Case Else
                        $a_Parm[3] = 5
                EndSwitch
            EndSelect
        Next
        ;返回运行参数数组，并将错误值设置为 0
        Return SetError(0, 0, $a_Parm)
    Else
        ;如果没有运行参数
        ;返回运行参数数组，并将错误值设置为 1
        Return SetError(1, 0, $a_Parm)
    EndIf
EndFunc
```

```

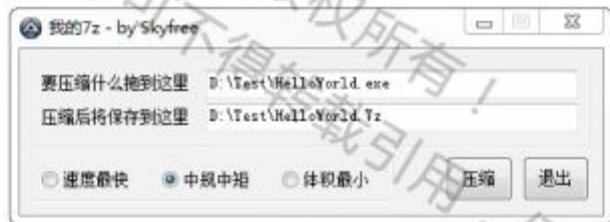
EndIf
EndFunc ;==>_My7z_GetParm

;根据 _My7z_Compress 函数的错误值提示用户运行中发生的错误
Func _My7z_ErrorMessage($Error)
    Switch $Error
        Case 1
            MsgBox(0 + 16, "错误", "源文件为空或不存在", 0, $gu_My7z_Form1)
        Case 2
            MsgBox(0 + 16, "错误", "目标文件为空", 0, $gu_My7z_Form1)
        Case 3
            MsgBox(0 + 16, "错误", "压缩级别设置有误", 0, $gu_My7z_Form1)
        Case 4
            MsgBox(0 + 16, "错误", "7za.exe 装载失败", 0, $gu_My7z_Form1)
        Case 5
            MsgBox(0 + 16, "错误", "压缩失败！", 0, $gu_My7z_Form1)
    EndSwitch
EndFunc ;==>_My7z_ErrorMessage

```

### 效果图

( 1 ) 拖拽入文件时 :



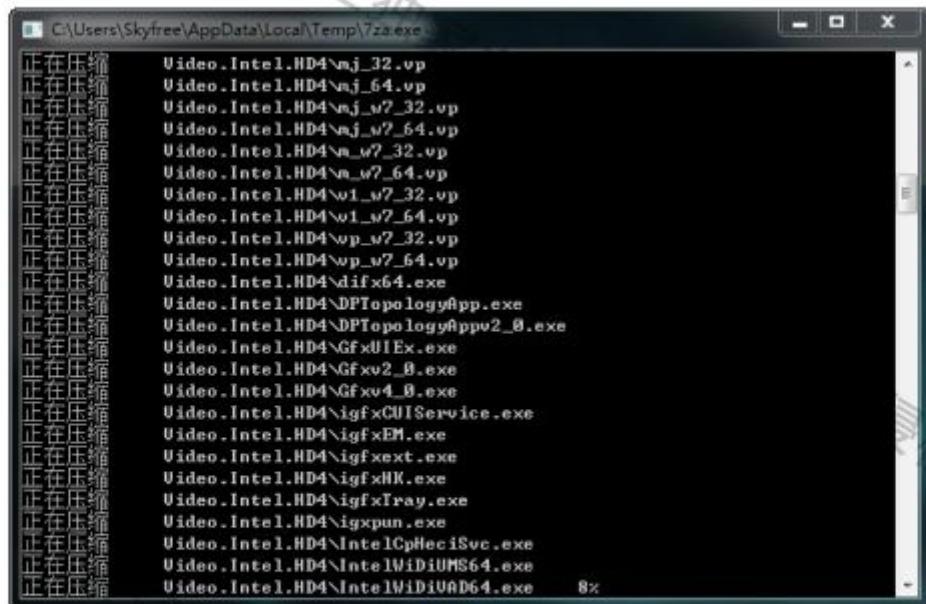
( 图 5-119 )

拖拽入目录时 :



( 图 5-120 )

运行时 ( 压缩大目录比较明显 , 小文件会一闪而过 ) :



(图 5-121)

如果不喜欢黑框，也可以调整 RunWait 参数进行隐藏，但会导致压缩大文件时看不到进度。

### 拓展：使用运行参数

运行：

```
D:\Test\My7z.exe /src="D:\Test\HelloWorld.exe" /dest="D:\Test\HelloWorld.7z" /lv=ultra /auto
```

将自动调用“D:\Test\My7z.exe”将 D:\Test 下的“HelloWorld.exe”压缩成 HelloWorld.7z。

而如果不使用/auto 参数，则：



(图 5-122)

程序会将参数所指定的源文件、压缩包、压缩等级等设置至 GUI 中对应控件，等待用户操作。

## 5.8 Level5 总结

Level5 作为“休息、总结、拓展、练习”的一章，将大家之前所学的内容融会贯通，拓展新知识、巩固旧知识，做到稳扎稳打、循序渐进。

5.1 节，拓展的数组和循环的相关知识，这些知识放在本章讲解是为了避免对之前的知识学习形成干扰，从而影响对基础知识的学习。例如数组内存储数组就对一维、二维数组的学习容易形成干扰，而在理解和实践过一维、二维数组后再学习数组内存储数组就更容易理解并难以混淆了。

5.2 节，是非常“涨姿势、增智商”的一节，本节讲解了数组相关的经典算法，这些算法都十分巧妙，而对于算法逻辑的理解则是重中之重。对于这一节的算法一定要逐步推敲、逐渐消化，虽然这些算法中的大多数可以通过 UDF 来实现，但对于算法的理解过程不仅仅只有“理解了算法”这一个结果，重点在于理解过程中对逻辑思维的锻炼。强大的逻辑+巧妙的算法=高效的应用程序。

5.3 节，编译参数是增强编译效果的捷径，但本节并不是只讲解了编译参数的作用，而是从 Au3 编程环境的组成开始，逐步分析编译参数的形成和使用，并明确指出了一些容易由编译参数而引发的错误，对于右键菜单与 SciTE 内运行、编译不一致的问题也提出了解决方案，这些内容的价值比对编译参数的功能解释更有价值。

5.4 节，随着 x64 系统越来越普及，程序对于 x86、x64 环境的兼容被提上台面。由于我们经常说的 x64 ( x86-64 ) 系统其实是 x86 ( x86-32 ) 系统的一种拓展，所以 x86 程序可以兼容在两个平台上，为程序的通用化提供了基本平台。而在通用兼容的过程中重定向问题带来的困扰尤其突出，本节在讲解了一些基础后着重讲解了如何解决文件和注册表的重定向问题，为 x86 程序在 x86、x64 双平台上的兼容运行铺平了道路。

5.5 节，程序运行参数，可以通过不同的参数来控制程序以不同的方式运行。虽然程序的代码在编译后已经固定，但可以根据参数的不同来执行不同的模块，达到单程序多用途化。而通过运行参数又可以使程序可以很方便的被其他程序调用而实现特定功能，也算是初步体会了一下“接口”概念。

5.6 节，文件拖拽是一种能有效增加用户好感的交互方式，而实现文件拖拽的基本条件 Au3 已经为大家准备好了，所以只要明确实现拖拽的几个步骤，则实现拖拽只需要很简单的代码。不要忽略用户体验，好的程序就是“不需要让用户去思考”。

5.7 节，作为一个练习节，集合了需求分析、外部程序调用、GUI 设计、文件拖拽、运行参数等多种之前学过的知识，将这些知识进行有机整合。本节代码做了足够多的注释，希望大家能够边读边理解，读懂程序只是目的之一，最重要的收获往往来自读的过程！

好的，本章学习到此为止，希望大家都得到了自己所需要的收获！