目 录

第一章、AutoIt 初步 3
第一节、为什么要学习 AutoIt 4
第一个问题，为什么要学习编程？ 4
第二个问题，为什么要学习 AutoIt？ 4
第三个问题，AutoIt 的优势在哪里？ 4
第二节、AutoIt 的使用 5
1、AutoIt 的安装 5
2、我们的第一个 AutoIt 程序 5
3、编译 AutoIt 程序 6
第二章、AutoIt 基础 7
第一节、常量与变量 8
1、常量 8
2、变量 8
3、常量与变量共同的使用规则 9
第二节、数据类型 9
1、数值型数据 9
2、字符（串）型数据 10
3、布尔类型数据 10
第三节、算数运算相关 11
1、算数运算符 11
2、算数运算相关函数 11
第三章、顺序结构程序设计 13
第一节、AutoIt 中的数据输出 14
第二节、AutoIt 中的数据输入 16
第三节、AutoIt 中的赋值运算 18
第四节、顺序结构程序设计实例 18
第四章、选择结构程序设计 21
第一节、逻辑运算和关系运算 22
第二节、If…Then…Else 语句 23
第三节、Select...Case 与 Switch...Case 语句 25
1、“Select...Case”语句 25
2、“Switch...Case”语句 26
第四节、选择结构程序设计实例 27
第五章、循环结构程序设计 30
第一节、“While…WEnd”循环 31
第二节、“Do...Until”循环 32
第三节、“For...Next”循环 33
第四节、循环的嵌套 34
第五节、循环结构程序设计实例 35
第六章、字符串 39
第一节、字符串型数据 40

1、字符串变量的定义与赋值 40
2、字符串数据的连接 40
3、关于字符串的宏 41
第二节、ASCII 42
第三节、字符串相关函数 43
第七章、数组 46
第一节、一维数组 47
第二节、二维数组 49
第三节、数组应用实例 50
第八章、函数 58
第一节、函数的一般形式 59
1、函数的定义 59
2、函数的参数 59
3、数组做函数参数 60
4、参数的“值传递”与“址传递” 61
第二节、变量的作用域 63
1、局部变量 63
2、全局变量 63
3、变量的作用范围 64
第三节、函数的嵌套与递归 65
1、函数的嵌套 65
2、函数的递归 66

第一章、AutoIt 初步

在本章中，我们将与您一起初步接触 AutoIt。您将会获取一些新的概念，并写出您的第一个 AutoIt 程序，迈出您关于 AutoIt 的第一步！

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

第一节、为什么要学习 AutoIt

第一个问题，为什么要学习编程？

曾经有一句戏说，但颇为经典，“世界上只有两种人，懂二进制的与不懂二进制的”。对于很多从事 IT 业但非从事编程工作的人来说，编程到底有多重要？不少朋友认为并不重要，当然，包括我在起初时也有这个感觉。

人总有属于自己的各种想法，想在计算机中实现这些想法，就必须会编程。因为，只有编程可以最好的驾驭一台计算机，会编程可以让这个铁疙瘩死心塌地的为您做各种您想做的事。

而如果一个人有自己独特的想法，还能将这些独特的想法以编程的形式应用于计算机中，那么，这个人就会开始与众多“不懂二进制”的人渐渐的不同。而后，随着这种“不同”的慢慢深化，最终这个“懂二进制”的人的存在意义将与“不懂二进制”的人有很大不同。

在市场人才竞争日益激烈的今天，多一个存在的理由，就是多一份生存的机会。

第二个问题，为什么要学习 AutoIt？

我曾经见过不少朋友在选择编程语言时十分的慎重，经常耗费一星期甚至一个月来比较是 C++好还是 C#好，是 JAVA 好还是 VB.NET 好。当然，我并不是在否定这些朋友的慎重原则，慎重总是好的。

但是，对于广大非编程专业的 IT 人士而言，我们需要什么样的编程语言？由于我们非从事专业编程，我们在编程上的需要一般只有两点：

（1）编写各种满足我们工作需要的小程序，让我们的工作更加自动化、方便化、快捷化，我们很少编写各种大的应用程序，毕竟我们不想抢专业编程人员的饭碗……（^_^）

（2）从学习编程到实用编程越快越好，我们可不想学三个月的 C++才刚刚知道怎么写个循环语句，我们需要更快的从学习一种语言转换到可以在实践中运用这种语言。

而以上两点，是很多“大型”编程语言所不具备的，而却是 AutoIt 所具备的。

心理学上讲，学习具有迁移性。所谓迁移性，简单说，就是当您学会骑自行车后再学习骑摩托车不会很困难。也就是说，当学会一种技能后，再学习与其相似或原理上相近的技能时，会十分容易上手。不可想象一个不会骑自行车的人你忽然给他辆摩托车让他骑会怎么样？学习骑自行车我们不仅学会了怎么让自行车动起来，更重要的是我们学会了怎么保持这种二轮交通工具的平衡。这一点和编程语言的通用性是一样的。

我们无论学习哪种编程语言，或者以哪种编程语言开始，学习的都不仅仅是这种语言，而是一种编程习惯，一种编程算法。习惯与算法，可以在不同的编程语言之间通用，所以大可不必计较我们现在学的是什么语言，但我们一定要计较我们在学习这种语言时学会了什么算法与习惯！

第三个问题，AutoIt 的优势在哪里？

就我个人的理解，优势主要有如下几方面：

（1）较为宽松的语法

AutoIt 的语法较为宽松，虽然一个宽松的语法环境不太利于培养规整的编程习惯，但是宽松的语法环境可以有效的提高上手的速度，更可以让很多初学者在不必过于担心语法错误的条件下专心的学习各种编程基本知识。

（2）完善的帮助文档

AutoIt 有着完善的帮助文档，而且经由汉化工作者的大量努力，AutoIt 的中文帮助文档十分的实用。AutoIt 帮助文档中包含了大量 AutoIt 相关知识，当您忘记某一段知识时可以快速的查阅帮助

文档。有了这个帮助文档，您甚至可以不去记忆某些具体的语句内容，而是到需要时再去查阅。

（3）函数库十分丰富

AutoIt 有着大量的自带函数，并有着丰富的用户自定义函数（UDF）。如果您在这里还不理解什么是“函数”，不要紧，我们简单的说。例如您需要画一个三角形和一个圆形，最直接的办法是找纸和笔，依次画三角形和圆型，当然，如果您是领导，您也可以这样做：

“小张，帮我画个三角形，小李，帮我画个圆形！”

几秒种后将会有一张画着三角形和圆形的纸出现在您面前。这里的“小张”“小李”就是函数。

那么，在有着大量现成函数的 AutoIt 中，很多复杂的工作您只需要发号施令就可以了，不需要亲手去做。这一点也是 AutoIt 可以快速从学习转向实践的原因之一。

（4）广泛的使用

这里的广泛，当然不如 C++这种铁牌语言广泛，但是由于上文所述的 AutoIt 的诸多特点，让 AutoIt 在很多从事 IT 业却非从事专业编程的人员中有着十分广泛的应用。而且很多人乐于分享他们的源码，并提供帮助。网络中也有着各种以 AutoIt 为主的专业网站，例如：

http://Www.AutoIt.Net.Cn/

第二节、AutoIt 的使用

1、AutoIt 的安装

AutoIt 目前总版本为 v3，这也是为什么我们现在普遍把 AutoIt v3 称为 AU3 的原因。AU3 的最新版本您可以去 AutoIt 中文站（http://Www.AutoIt.Net.Cn/）中获取。AU3 相比其他编程语言轻则几百 M 重则几 GB 的大小可谓是轻盈许多，只有 30M 左右，十分方便下载与安装。本书全文将以 AutoIt v3.2.12.0.1 版本为基础进行讲解。

当您下载到最新版本的 AU3 后，双击即可开始安装。

AU3 安装完毕后会自动启动一个 AU3 工具箱并在桌面上创建一个 SCITE 编辑器的快捷方式。右键单击 AU3 工具箱图标，可以看到其中包含相当多的 AU3 相关工具和文档，我们在今后会经常使用到。SCITE 编辑器是我们用来书写 AU3 源代码的编辑器，SCITE 编辑器具有对 AU3 源代码中的语句进行各种加亮显示的功能，十分方便。

2、我们的第一个 AutoIt 程序

要书写一个程序，我们就必须先新建这个程序的源码文件。

在装好 AutoIt 程序包后，在桌面上单击鼠标右键→“新建”→“AutoIt v3 脚本”，这时桌面上会出现一个“新建 AutoIt v3 脚本.au3”，为了方便记忆，我们将它改名为“First.au3”。注意，如果您并未设置显示后缀名，“.au3”是不会显示的，当然，这不会造成什么影响。

鼠标右键单击“First.au3”→“编辑脚本（SCITE）”，这时会自动使用 SCITE 打开“First.au3”这个 AU3 脚本了。

凡是新建的 AU3 脚本文件，打开后都会有一些自带的内容，是一些提示或者常用的预编译等。对于初学者，可以把这些完全删除。

删除后，我们在脚本中写下如下代码：（注意，特别是对于初学者，请保证您处于英文输入状态下输入下面代码）

```
MsgBox(0,"First","Hello world !")
```

您现在不必强求自己了解这行代码的意思，随后我们会详细的进行了解。写完后，保存。关闭 SCITE 编辑器。双击我们桌面上的“First.au3”，这时会弹出一个带有“Hello world !”信息的消息框！

OK，我们的第一个 AU3 程序就这么完成了！虽然只有一行代码，但是这将是我们驾驭计算机的开始！

3、编译 AutoIt 程序

常常听高手说“编译”，听的多了，就会对这个词产生一种莫名的敬畏感。其实，编译是一件不那么神秘的事情。

说回我们刚才所做的，我们已经写了我们的第一个 AU3 脚本“First.au3”，在安装有 AU3 程序包的计算机中，如果我们双击“First.au3”会弹出“Hello world！”消息框，而如果我们把“First.au3”移动到一台没有安装 AU3 程序包的计算机中，再双击“First.au3”会怎么样？

答案是不会怎么样，最多弹出一个窗口是 Windows 问你用什么才能打开“First.au3”，而不会弹出“Hello world！”消息框，这要怎么办？难道 AU3 脚本只能运行于安装有 AU3 程序包的计算机中？别着急，如果我们要 AU3 脚本程序变的通用，我们要做的就是“编译”。

“编译”的意思，就是把一个源码文件转换成可执行文件，对我们的 AU3 来说，“编译”就是把 AU3 脚本转换成 EXE 可执行文件。

那要怎么编译呢？这个也很简单，右键单击我们的“First.au3”→“编译脚本（带进度）”，片刻之后当计算机屏幕左上角一个进度结束之后，桌面上会出现一个“First.exe”，OK，编译就是这么简单一件事。

现在，我们把“First.exe”无论移动到任何安装有 Windows 的计算机里，无论是这台计算机中有没有预先安装 AU3 程序包，都可以双击运行并显示我们的“Hello world！”

注：UNICODE 的 AU3 程序可能不能用于 Windows 9x 系列，不过这不是问题，毕竟没多少计算机还在使用 Windows 9x 了。

通过本章的学习，我们初步认识了 AutoIt 的相关知识，了解了为何要学习 AutoIt，并且我们使用 AutoIt v3 成功的编写并编译了我们的第一个 AutoIt 程序。虽然这些都比较简单，而且您可能还并不了解我们第一个程序里的那行语句的含义，不过不要着急，一切都还只是个开始，我们会一步步的将 AU3 的全部知识整合进我们的大脑知识库中！

《Let’s AutoIt》——第一章、AutoIt 初步，

完成于 2008-8-2，2008-8-14 第一次修正

作者：Skyfree
QQ：165718402
E-Mail:Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第二章、AutoIt 基础

在本章中，我们将和您一起学习 AutoIt 的一些基础概念。对基础概念的准确把握，将使您今后对 AutoIt 的学习更加得心应手！

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

第一节、常量与变量

1、常量

所谓常量，就是在程序运行过程中，其值不能被改变的量。在 AutoIt 中，常量的声明方法如下：

Const <常量名>=<数据>

例如：

Const $PRICE=10

“Const”为常量声明关键字，声明常量前必须使用此关键字。“$”是 AutoIt 的常变量前缀名，所有的 AutoIt 常变量名都必须以“$”作为前缀。“PRICE”为常量名，当然这个可以根据您的喜好而更改，但是最好这个名字是有意义的。“=10”是将“10”赋值给“PRICE”常量。

整体来讲，这行语句的意义是：声明一个名为 PRICE 的常量，并且为其赋值 10。

关于常量名的声明有各种习惯，这里 Skyfree 我推荐您使用 C 语言中常量名的书写习惯，即将常量名中的英文字母全部大写，这样便于与变量相互区分。当然，这仅仅是一个书写习惯而已，不按此习惯书写也不会出错。

声明多个常量时，可以分多行书写：

Const $PRICE1=10

Const $PRICE2=20

Const $PRICE3=50

也可以在同一行内书写多个：

Const $PRICE1=10,$PRICE2=20,$PRICE3=50

这里的逗号全部为英文逗号。

使用常量时要注意：

（1）常量一旦被声明则其值不能被改变，不要尝试在程序运行中以任何方法改变一个常量的值；

（2）常量声明时，注意不要与已存在的常量或变量的名字相重复；

（3）一个常量只能声明一次，不能多次声明。

声明常量最大的优点就是可以“一改全改”，例如程序中多处用到某物品的价格，如果价格用数值表示，当价格调整时，需要到程序全文中做多处修改，而如果用常量代表价格，则只需要修改常量的值即可。

2、变量

所谓变量，就是在程序运行过程中，其值可以被改变的量。在程序运行的过程中“可变”是变量与常量最大的区别。在 AutoIt 中，变量的声明方法如下：

Dim <变量名>

（还可以以“Local”或“Global”声明变量，由于牵涉其他相关内容，此后在相关章节会详细介绍，这里不加赘述）

例如：

Dim $Sky

“Dim”是变量声明关键字，声明变量前必须使用此关键字。“$”是 AutoIt 的常变量前缀名，所有的 AutoIt 常变量名都必须以“$”作为前缀。“Sky”为变量名，当然这个可以根据您的喜好而更改，但是最好这个名字是有意义的。

变量名声明也有一些习惯，如大写每个单词的首字母，例如$AutoIt，这样便于阅读与修改代码。当然，这仅仅是一个书写习惯而已，不按此习惯书写也不会出错。

声明多个变量，可以分多行书写：

Dim $Sky1
Dim $Sky2
Dim $Sky3
也可以在同一行内书写多个：
Dim $Sky1,$Sky2,$Sky3
为变量赋值的方法有多种，可以在声明变量时就为其赋值：
Dim $Sky=10
也可以在声明变量后为其赋值：
Dim $Sky
$Sky=10
使用常量时要注意：
（1）变量需要先声明后赋值，不要尝试在未声明变量前就为这个变量赋值；
（2）变量在声明时，注意不要与已存在的变量或常量的名字相重复；
（3）一个变量只能被声明一次，请勿多次声明同一个变量；
（4）AutoIt 中可以不预先声明某变量，而是到需要使用时直接使用并即时声明，强烈不推荐使用这个方法，虽然在某些小程序中会带来方便，但是十分不利于养成良好的编程习惯，且在出现问题时不容易查证。

3、常量与变量共同的使用规则

（1）常量与变量声明前，必须使用与其对应的关键字：Const 和 Dim；
（2）在 AutoIt 中，所有的常量与变量名前必须是用“$”前缀名；
（3）常量或变量名的命名规则为：以下划线（“_”）或英文字母开头的，下划线、英文字母与数字的组合，例如“$_FreeSkyCD”，“$Beijing_2008”都是正确的；
（4）在 AutoIt 中，其实是不区分变量中英文字母大小写的，例如$SKY 和$Sky 将会被认为是同一个变量，所以要严加注意不要重复使用同一个名字来声明常量或变量；
（5）常量与变量都遵循先声明后使用的原则，将“即时定义并使用”的方法抛之脑后吧，当您的编程习惯达到一定高度后再来驾驭这个方法也绝不迟；
（6）常量与变量名不宜过长；
（7）尽量使用有意义的名字作为常量或变量的名字，便于记忆与阅读。

第二节、数据类型

为了便于记忆与理解，我们将常见的数据分为不同的类型。

1、数值型数据

（1）普通的十进制数值型数据：5、3.1415926、-1.2、-30 等；
（2）指数型十进制数值型数据：1.2e3，这个等同于数学中的 1.2 乘以 10 的三次方，只不过我们在编程语言中的写法与平时不同了而已；
（3）十六进制数据：以“0x”开头，例如 0x312、0x4fff 等（如果您还不了解什么是二进制或者十六进制等，不用着急，因为我们只在特别的时候需要这种数据，需要时百度一下也不迟）。
声明数值型数据变量的一般形式：
Dim $t=10

2、字符（串）型数据

（1）字符串数据一般用“" "”（英文的双引号）或“' '”（英文的单引号）包含起来，例如：'a'、"b"、 'FreeSkyCD.Cn'、"AutoIt.Net.Cn"，英文单引号与英文双引号的使用意义是相同的，都用来说明引号中包含的是一个字符或字符串；

（2）一般情况下我们只使用双引号，而在特殊情况下，我们会将单双引号混用，例如字符串中包含英文引号，则需要这样写：

' 这个 "句子" 中包含了 "很多" 的 "双引号" ，留意到了吗？ '

也可以连续使用两个引号以表明这个引号是字符串中的，而非包含字符串所用的：

" 这个 ""句子"" 中包含了 ""很多"" 的 ""双引号"" ，看不到才怪！ "

不过为了便于阅读，建议使用单双引号混用的方法，这种方法更便于代码的阅读。

声明字符（串）型数据变量的一般形式：

Dim $t="Www. FreeSkyCD.Cn"

另，字符串一般使用“&”来联结，例如：

Dim $s1="Www."

Dim $s2="FreeSkyCD.Cn"

Dim $s

$s=$s1&$s2

则$s 的值为“Www. FreeSkyCD.Cn”

3、布尔类型数据

布尔类型值只有“true”（真）和“false”（假）两种值，是一种逻辑值，与此数据类型相关的内容会在此后讲解“选择结构程序设计”时详细讲解。

虽然我们人为的将数据类型分为多种，但是 AutoIt 的变量类型其实都是变体的，也就是说 AutoIt 的变量是没有固定类型的。在声明一个变量时无需同时声明它将要盛放数据的数据类型，例如声明一个变量：

Dim $s

我们可以为$s 赋数值类型的值，例如：

$s=1001

也可以为$s 赋字符串类型的值，例如：

$s="abc"

这为变量的使用提供了一定的方便，但也容易引起一些混乱，所以我们有一个建议：使用一个变量只存储一个数据类型的数据。也就是说如果这个瓶子里放过酱油了，就不要再放醋了。如果没有特殊需要，请遵循这个建议。

常量与变量一样，也是变体的，也就是说我们可以：

Const $ST=1023

也可以：

Const $ST="Miranda"

这个与变量的声明是基本相同的，声明常量时也无需声明常量的数据类型。

AutoIt 中，任何变量如果在声明后不为其赋值，那么其默认为空字符。

第三节、算数运算相关

日常生活中我们会遇到形形色色的算数运算，在 AutoIt 中也包含很多与算数运算相关的应用。

1、算数运算符

与数学中的类似，AutoIt 的算数运算符有如下几种，当然，可能会和数学书里的写法不一样。

运算符	名称	作用	范例
+	加法运算符	用于变量或数值之间的和运算	“10+20”，结果是 30
-	减法运算符	用于变量或数值之间的差运算	“20-10”，结果是 10
*	乘法运算符	用于变量或数值之间的积运算	“3*5”，结果是 15
/	除法运算符	用于变量或数值之间的商运算	“12/4”，结果是 3
^	冥运算符	用于计算某个变量或数值的几次方	“2^3”，结果是 8

如上所述为程序设计中常用的算数运算，算数表达式中变量或数值结合的顺序由算数运算符优先级决定，同优先级的按照从左至右的方式结合。

算数运算符的优先级与数学中的相同，从高到低依次为：“^”、“*”和“/”、“+”和“-”，例如：

1+2*3，由于“*”优先级高于“+”，所以先计算 2 和 3 的积，后计算 2 乘 3 的积与 1 的和；

1+2-3，由于“+”和“-”优先级相同，所以按照从左至右的结合方式，应先计算 1 与 2 的和，后计算 1 与 2 的和与 3 的差。

另，可以使用“()”来规定运算的优先顺序，这个也和我们在数学中是一样的，例如：

(1+2)*3，结果为 9。

2、算数运算相关函数

还有一些其他的算数计算，例如绝对值、正弦、余弦等，这些在程序设计中往往一函数的方式表达，下面介绍一些关于这些特殊运算的 AutoIt 函数：

函数名	函数语法	作用	范例
Abs	Abs(表达式)	求绝对值	“Abs(-3)”，结果为 3
Sin	Sin(表达式)	求正弦值，表达式为弧度值	预先将 π 的近似值赋值给变量$pi： $pi=3.1415926 则： “Sin($pi/6)”，值约为 0.5 “Cos($pi/3)”，值约为 0.5
ASin	ASin(表达式)	求反正弦值，表达式为弧度值	
Cos	Cos(表达式)	求余弦值，表达式为弧度值	
ACos	ACos(表达式)	求反余弦值，表达式为弧度值	
Tan	Tan(表达式)	求正切值，表达式为弧度值	
ATan	ATan(表达式)	求反正切值，表达式为弧度值	
Round	Round(表达式[,位数])	获取某数值精确到指定位数的结果，位数省略则四舍五入取整，位数为负数则精确至小数点前规定位数	“Round(-1.582, 1)”，结果为-1.6 “Round(123.5, -1)”，结果为 120
Ceiling	Ceiling(表达式)	舍弃小数部分后，取临近的最大整数（范例中 x 代表 1~9）	“Ceiling(1.x)”，结果为 2 “Ceiling(-1.x)”，结果为-1
Floor	Floor(表达式)	舍弃小数部分后，取临近的最小整数（范例中 x 代表 1~9）	“Floor(1.x)”，结果为 1 “Floor(-1.x)”，结果为-2
Log	Log(表达式)	求表达式的自然对数	“Log(10)”
Exp	Exp(表达式)	以 e 为底以表达式为幂的幂次方	“Exp(5)”
Mod	Mod(数值 1,数值 2)	求数值 1 除以 数值 2 后的余数	“Mod(7,4)”，结果是 3

Sqrt	Sqrt(表达式)	求表达式的平方根	“Sqrt(9)”，结果为 3

看了这么多关于算数运算的内容，是不是有一种又回到了学生时代的感觉呢？

通过本章的学习，我们了解了 AutoIt 中最基础的知识部分，学习了最基本的常量与变量知识、数据类型知识以及基本的算数运算方法。本章内容虽然简单，但不要小看这些简单的知识，任何复杂的程序都是由最简单的部分堆积而成的，所以一定要认真学习这些基础。

《Let’s AutoIt》——第二章、AutoIt 基础

完成于 2008-8-11 晨，2008-8-14 第一次修正

作者：Skyfree
QQ：165718402
E-Mail:Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第三章、顺序结构程序设计

在本章中，我们将和您一起学习 AutoIt 中最简单结构的程序设计——顺序结构程序设计。本章中您将学习到程序设计的一些基础知识与方法。

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

第一节、AutoIt 中的数据输出

AutoIt 是支持 GUI（Graphic User Interface，图形用户界面）程序设计的，所以可以在 GUI 界面中输出数据给用户。但为了我们能尽快学习 AutoIt 的程序设计方法，排除其他学习对当前学习的干扰，我们在现阶段仅使用最简单的数据输出方法——消息框，MsgBox。

其实我们在第一章中就已经接触了 MsgBox 的使用，还记得我们用 AutoIt 写的第一个程序“Hello World !”吗？

```
MsgBox(0,"First","Hello world !")
```

这个简单的程序就是通过 MsgBox 实现的。现在我们将详细的学习一下 MsgBox 的使用。

MsgBox 的语法：MsgBox (标志, "标题", "文本" [, 超时时间])

MsgBox 的参数说明：

（1）标志，标志规定着 MsgBox 中的按钮和提示图标类型，稍候会详细介绍。

（2）标题，MsgBox 的标题文字，例如“First”。

（3）文本，MsgBox 的文本内容（提示信息），例如“Hello world !”。

（4）超时时间，这是个可选参数，以秒为单位，在超过规定时间后会自动关闭 MsgBox。

MsgBox 中“标志”的详细说明：

（1）按钮相关标志值：

标志值（十进制）	标志值（十六进制）	相应按钮
0	0x0	确定
1	0x1	确定、取消
2	0x2	终止、重试、忽略
3	0x3	是、否、取消
4	0x4	是、否
5	0x5	重试、取消
6	0x6	取消、重试、继续

（2）图标相关标志值：

标志值（十进制）	标志值（十六进制）	相应图标
0	0x0	（无图标）
16	0x10	警告标志（一般用于错误提示）
32	0x20	问号图标
48	0x30	感叹号图标
64	0x40	由一个“i”和圆圈组成的图标（消息通知）

（3）默认按钮值：

标志值（十进制）	标志值（十六进制）	默认选中按钮
0	0x0	默认选中第一个按钮
256	0x100	默认选中第二个按钮
512	0x200	默认选中第三个按钮

（4）模式值：

标志值（十进制）	标志值（十六进制）	相应模式
0	0x0	应用程序模式
4096	0x1000	系统模式
8192	0x2000	任务模式

（5）其他值：

标志值（十进制）	标志值（十六进制）	其他
0	0x0	（无特别）
262144	0x40000	消息框将具有顶层窗口属性
524288	0x80000	标题文字及文本内容将右对齐

“标志”的可用值相当多，而且可以混合使用，不过不要为记忆这些而发愁，我们经常使用的只有其中的一小部分。“标志”的值也分为十进制和十六进制两种模式，这个可以完全根据您的个人喜好只记住其中一种值就可以了，总体来说十进制便于阅读，十六进制便于记忆。

下面我们看几个常用的例子：

（1）“错误”提示：

MsgBox(0+16,"错误提示范例","某某错误！")

运行这个 MsgBox 会出现一个“错误”提示框，并且 MsgBox 只有一个“确定”按钮。当然不是指这个程序是错的，而是对用户提示某某错误。标志我们使用了“0+16”，0 代表只有一个“确定”按钮，16 代表使用“叉号”图标，当然您可以直接写成：

MsgBox(16,"错误提示范例","某某错误！")

之所以把标志写成“0+16”而不直接写成 16，是因为“0+16”更便于修改与阅读。

（2）“确定与取消“提示：

MsgBox(1+48+256,"确定与取消提示范例","确定要继续么？")

运行这个 MsgBox 会出现一个提示“是否继续”的提示框，并且 MsgBox 有一个“确定”按钮与一个“取消”按钮。标志我们使用了“1+48+256”，1 代表使用“确定”和“取消”两个按钮，48 代表使用“叹号”图标，256 代表默认选中第二个按钮——也就是取消按钮。当然您也可以将“1+48+256”直接写成“305”，只不过“305”相对“1+48+256”的可阅读性要差很多。

（3）“完成”提示：

MsgBox(0+64+262144,"完成提示范例","任务已完成",3)

运行这个 MsgBox 会出现一个“完成”提示。标志我们使用了“0+64+262144”，0 代表只有一个“确定”按钮，64 代表使用“消息”图标，262144 表明将此提示框置于所有窗口最顶层，也就是说不会被其他窗口遮挡。值得注意的是这次我们使用了超时时间参数，我们将超时时间设置为 3，这说明如果 3 秒内不单击“确定”则会自动关闭 MsgBox。

MsgBox“标志”的使用是十分灵活多变的，这里不能一一给大家举例，还请大家多多尝试。

说到这里，可能有的朋友会提出这么一个问题，假设我们的 MsgBox 上有两个按钮“确定”和“取消”，那么怎么分辨用户是单击了“确定”还是单击了“取消”呢？要解决这个，我们必须要了解 MsgBox 的返回值。

所谓“返回值”，就是当一个函数运行完毕后回馈的值，在这里指的是 MsgBox 运行完毕后返回的值。MsgBox 的返回值如下：

按下的按钮	返回值
确定（OK）	1
取消（CANCEL）	2
终止（ABORT）	3
重试（RETRY）	4
忽略（IGNORE）	5
是（YES）	6
否（NO）	7
重试（TRY AGAIN）	10
继续（CONTINUE）	11

另外，如果是超过“超时时间”后退出的 MsgBox，返回值为“-1”。

为了详细的展示 MsgBox 的返回值，Skyfree 我在这里暂时借用一下将在下一章中学到的“Select…Case…”语句：

```
Dim $Flag
$Flag=MsgBox(1,"MsgBox 返回值范例","您按的什么按钮我知道！",5)
Select
    Case $Flag=1
        MsgBox(0,"MsgBox 返回值范例","您按的是“确定”按钮！")
    Case $Flag=2
        MsgBox(0,"MsgBox 返回值范例","您按的是“取消”按钮！")
    Case $Flag=-1
        MsgBox(0,"MsgBox 返回值范例","超时退出")
EndSelect
```

首先我们声明了一个$Flag。而后我们设置了一个具有“确定”和“取消”两个按钮并带有 5 秒超时闲置的 MsgBox，并使用之前声明的$Flag 来接收 MsgBox 的返回值。

下面到了“Select…Case…”语句，您这里无需知道具体的“Select…Case…”语句的语法和用法，这段“Select…Case…”语句的意思是：当$Flag 也就是 MsgBox 的返回值为 1 时，说明按下的是“确定”按钮；当$Flag 也就是 MsgBox 的返回值为 2 时，说明按下的是“取消”按钮；当$Flag 也就是 MsgBox 的返回值为-1 时，说明是超时退出。

通过这个例子，相信您已经对 MsgBox 的返回值和用法都有了一定的了解，如果现在您仍无法熟练使用 MsgBox 也没有关系，因为我们在后面会经常用到这个函数来输出数据给用户，用的次数多了，自然也就熟练了。

第二节、AutoIt 中的数据输入

AutoIt 是支持 GUI（Graphic User Interface，图形用户界面）程序设计的，所以可以在 GUI 界面中输入数据给程序。但为了我们能尽快学习 AutoIt 的程序设计方法，排除其他学习对当前学习的干扰，我们在现阶段仅使用最简单的数据输入方法——输入框，InputBox。

InputBox 的语法：InputBox ("标题", "提示信息" [, "默认数据" [, "密码字符" [, 宽度, 高度 [, 左边, 上边 [, 超时时间]]]]])

MsgBox 的参数说明：

（1）标题，输入框的标题文字，与 MsgBox 的标题类似。

（2）提示信息，提示用户需要输入什么数据的说明。

（3）默认数据，可选参数，用于设定在输入文本框中的默认文字。

（4）密码字符，可选参数，显示在输入文本框中用以代替用户输入字符的字符。如果要正常显示字符只需定义此参数为空字符串""（默认）或空格字符即可。 如果此参数被设为多字符的字符串则只有第一个字符才有效。第二个字符及后面的其它字符有其它特殊用途。密码字符 参数的第二个及后面的其它字符可被用来限制用户输入。如果第一个字符是空格则输入得字符将可见，若第二个字符是 M 则表示输入将是强制性（Mandatory）的，也就是说用户必须输入至少一个字符，如果在没有输入任何内容的情况下按下“确定”按钮则脚本不会有任何反应，输入框既不会消失也不会返回字符串。

（5）宽度，可选参数，用以设定窗口宽度。如有指定此参数则高度参数也必须指定。指定“-1”则表示使用默认宽度。

（6）高度，可选参数，用以设定窗口高度。如有指定此参数则宽度参数也必须指定。指定“-1”

则表示使用默认高度。

（7）左边，可选参数，输入框左边离屏幕左边的距离（单位：象素）。默认情况下，输入框是居中显示的，如有指定此参数则“上边”参数也必须指定。

（8）上边，可选参数，输入框上边离屏幕左边的距离（单位：象素）。默认情况下，输入框是居中显示的，如有指定此参数则“左边”参数也必须指定。

（9）超时时间，可选参数，以秒为单位。指定时间过后输入框将自动关闭。

InputBox 的参数要比 MsgBox 多，但是使用起来却并不复杂，InputBox 用于将用户输入的数据“返回”给 AutoIt 程序，InputBox 的返回值如下：

InputBox 运行成功时将会返回用户输入的字符串给 AutoIt 程序，但要注意返回的字符串不会超过 254 个字符，如果输入的内容中含有回车或换行符，那么返回的字符串将被这些字符的第一个断开。

InputBox 运行失败时会返回一个空字符串，并将@Error 的值设置为如下：（@error 是用来存放错误值的宏，您只需要了解它代表出错时的值即可，暂时不必深究什么是“宏”）

@error 为 1 时，说明用户按下了取消（Cancel）按钮；

@error 为 2 时，说明超过了“超时时间”；

@error 为 3 时，说明输入框显示失败，这通常是由参数无效引起的。

另外，InputBox 运行成功时也会将@Error 设置为 0.

使用 InputBox 时还有一些要注意的事项：

（1）用户可调整输入框的窗口大小，但有一个最小尺寸限制：大约 190 x 115（象素）。默认的大小是大约 250 x 190（象素）。

（2）您还可以在“密码字符”参数的后面加上一个数字以指定输入字符串的最大长度。例如：

```
Dim $Value
$Value=InputBox("测试", "请输入", "", " M2")
```

在密码字符参数中的 M 表示不接受空字符串，而数字 2 则表示最多只能输入两个字符。

下面我们举几个 InputBox 的应用实例：

（1）输入并输出某数据，例如我们要用 InputBox 输入一个数值，而用 MsgBox 输出经过处理的数值：

```
Dim $a,$b
$a=InputBox("输入并输出某数据范例","请输入一个正整数：")
$b=$a^3
MsgBox(0,"输入并输出某数据范例",$a&"的三次方是："&$b)
```

我们定义了$a 和$b 两个变量，并使用 InputBox 为$a 赋一个我们输入的正整数值，随后将$a 的三次方的结果赋值给$b，最后使用 MsgBox 显示$a 的三次方也就是$b 的值。

这里您可能会对“ $a&"的三次方是："&$b ”这段语句有所疑惑，“&”用于联结“&”左右两侧的字符串，这个会在后续章节中详细介绍，所以您只要对“&”有个大致了解即可。

在这个例子中我们每输入一个正整数并按下“确定”按钮后，程序就会返回这个正整数的三次方给我们。但是细心的您可能会发现，在这段程序中如果对 InpuBox 按下“取消”按钮会出现不正常的结果，这是因为我们并没有设定按下“取消”按钮的动作。

（2）输入后的返回值范例：

```
Dim $Value
$Value=InputBox("返回值范例","请输入一个数据：","","",-1,-1,0,0,5)
Select
    Case @error=0
        MsgBox(0,"返回值范例","您输入的数据为："&$Value)
    Case @error=1
```

```
        MsgBox(0,"返回值范例","您按下了“取消”按钮")
    Case @error=2
        MsgBox(0,"返回值范例","超时")
EndSelect
```

首先我们声明了$Value 变量，并将 InputBox 的返回值赋值给$Value。在这里我们还为 InputBox 设定了 5 秒超时时间。InputBox 运行后不仅会为$Value 赋值并会修改@error 的值，当@error 为 0 时，说明 InputBox 成功运行，$Value 的值也就是 InputBox 的返回值为用户输入的数据；当@error 为 1 时，说明用户按下了“取消”按钮；当@error 为 2 时，说明超过了“超时时间”。

关于 InputBox 的用法这里不能一一举例，InputBox 的用法十分灵活，这个还需要大家多多练习、多多体会。不过要切记一点，如果要使用 InputBox 一定不要忘记处理当用户按下“取消”按钮后要执行的动作。

第三节、AutoIt 中的赋值运算

赋值运算无时无刻都在伴随我们，说到赋值运算就不得不先说赋值运算符。

赋值运算符——“=”，这个和我们数学中等号一样的字符，作用是将赋值运算符右边表达式的值赋值给赋值运算符左边的变量，例如：

“$a=1+2”或者“$s=$a+$b”或者“$s=$a+2”等等。

所以，赋值运算符的语法为：

<变量>=<表达式>

要注意的是，赋值运算符左侧必须是变量。

由于在 AutoIt 中赋值运算符（“=”）与判断两个数据是否相等的等号运算符（“=”）是相同的，所以在使用时必须注意。这里，建议您在遇到例如“$s=$a+2”这种语句时，读作“将$a+2 的值赋给$s”，而不要读作“$s 等于$a+2”。

赋值运算符的使用十分简单，我们已经很多次的在使用它，不过还有一些特殊的赋值运算符需要我们了解：

运算符	名称	作用
+=	自增赋值	自身与运算符右侧的数据相加后赋值再赋值给自身
-=	自减赋值	自身与运算符右侧的数据相减后赋值再赋值给自身
*=	自乘赋值	自身与运算符右侧的数据相乘后赋值再赋值给自身
/=	自除赋值	自身与运算符右侧的数据相除后赋值再赋值给自身

总体来说，自赋值运算是将字符值运算符左侧的数据与右侧的数据做规定运算后再赋值给自身。假设我们已经声明了$a，并将$a 赋值为 3，那么：

“$a+=1”，$a 为 4，也就是说“$a+=1”等同于“$a=$a+1”。

另外，自赋值运算符右侧的数据是优先计算的，或者可以认为是默认带有“( )”的，例如：

“$a*=1+2”，$a 为 9，也就是说$a*=1+2”等同于“$a=$a*(1+2)”。

自赋值运算符虽然可以让我们少写几个字，但是如果滥用则会影响程序的可阅读性，所以仍旧推荐您只使用普通的赋值运算符（“=”），而不要使用这些自赋值运算符。

还有一点顺道说一下，自赋值运算符还包含一个“&=” 连续赋值运算符，这个是用于字符串的，我们会在后续专门的章节中详细学习到，这里不再赘述。

第四节、顺序结构程序设计实例

在顺序程序设计中，我们还有几点是需要额外知道的：

（1）Exit，这个语句用于马上退出 AutoIt 程序，正常情况下 AutoIt 在一个程序所有语句都执行完毕后会自动退出，所以一般不用书写 Exit，而如果您需要设定在特殊情况下退出 AutoIt 程序，则可以根据需要书写“Exit”。

（2）注释，在程序中写注释是一个优秀的习惯（Skyfree 我就未在学习程序设计时养成这个习惯），注释可以有效的增加程序的可读性，注释不被视作语句，程序执行时会被自动忽略。如果要加入注释，需要在注释语句前使用英文分号（“;”），如果使用 SCITE 编辑器编辑 AutoIt 代码，语句前加入“;”时这行语句会变成绿色，说明这行语句是注释，程序运行时自动略过。

（3）代码换行符（“ _”，空格+下划线），当某行语句太长不宜阅读时，可以使用代码换行符进行换行，要注意的是代码换行符不能在字符串中使用。

下面我们看两个顺序程序设计，以增强我们对本章知识的运用与理解。

1、输入圆的半径，返回圆的面积。

【代码】

```
Dim $pi=3.1415926
Dim $r,$s
$r=InputBox("圆的面积计算","请输入圆的半径：")
If @error=1 Then Exit
If $r="" Then Exit
$s=$pi*$r^2
MsgBox(0,"圆的面积计算",Round($s,2))
```

【解析】

根据已有数学知识经验，求圆的面积需要知道圆的半径，根据公式圆的面积等于π与半径平方的积，我们可以得到解决计算圆面积的思路。

这段代码中，我们首先定义了π，即“Dim $pi=3.1415926”，其实这个也可以定义为常量，因为π的值一般不会变化。随后我们定义了两个变量$s 和$r，分别代表圆的面积与半径。紧接着我们使用 InputBox 要求用户输入圆的半径，并将返回值复制给$r。

其后运用了两行判断语句（If…Then…），这个我们将在下一章学习，这里不加以详细说明。

“If @error=1 Then Exit”，如果@error=1，也就是当用户按下 InputBox 上的“取消”按钮后，本程序退出。

“If $r="" Then Exit”，则是一个纠错机制，即如果用户没有输入任何数据却按下了“确定”按钮或 InputBox 出现错误时，也退出本程序。其实这个纠错机制仍不完善，不能避免用户输入负数时怎么做，要完善纠错机制就要等我们学习了下一章的内容了，这里可以不必着急。

避过可能出现的特殊情况后，使用“$s=$pi*$r^2”将计算后得到的圆的面积赋值给“$s”，这里有一个细节，关于“$pi*$r^2”，由于“^”运算符的优先级高于“*”，所以会先计算半径的二次方，后计算π与圆半径二次方的乘积。

最后使用 MsgBox 将计算后的圆的面积输出，并使用 Round 取小数点后两位精度。

2、输入华氏温度，计算摄氏温度。提示：摄氏温度与华氏温度的转换关系为：

摄氏温度=5／9×(华氏温度-32)

【代码】

```
Dim $c,$f
$f=InputBox("华氏摄氏温度转换","请输入华氏温度：")
If @error=1 Then Exit
```

```
If $f="" Then Exit
$c=5/9*($f-32)
MsgBox(0,"摄氏温度为：",Round($c,2))
```

【解析】

首先声明变量$c 代表摄氏温度，$f 代表华氏温度。随后使用 InputBox 获取用户输入的华氏温度值，并赋值给$f。再通过简单的纠错机制后，根据公式运算摄氏温度“$c=5/9*($f-32)”。最后使用 MsgBox 输出求得的摄氏温度。

本章学习了程序设计中最基础的结构——顺序结构，顺序结构是一切程序设计结构的基础，牢固的把握顺序结构的写法，养成良好的程序书写习惯，这会为今后学习其他结构程序设计打下良好的基础！

《Let’s AutoIt》——第三章、顺序结构程序设计

完成于 2008-8-11 夜，2008-8-14 第一次修正

作者：Skyfree
QQ：165718402
E-Mail:Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第四章、选择结构程序设计

在本章中，我们将和您一起学习 AutoIt 中选择结构程序设计。您将会逐步掌握如何使用判断语句，并掌握根据不同情况运行不同代码的方法。

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

第一节、逻辑运算和关系运算

1、逻辑运算

说到逻辑运算，首先我们要回顾在第二章第二节中提到的布尔数据类型。布尔类型值只有“True”（真）和“False”（假）两种值，是一种逻辑值。布尔值之间有三种常见运算：与、或、非，在 AutoIt 中的运算符依次为“And”、“Or”、“Not”。

逻辑运算的真值表：

$a	$b	Not($a)	Not($b)	$a And $b	$a Or $b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

从上述列表可以看出：

（1）逻辑“与”（也就是“And”）运算中，两者都为真时结果才为真，两者有至少一者为假结果即为假；

（2）逻辑“或”（也就是“Or”）运算中，两者都为假是结果才为假，两者有至少一者为真结果即为真；

（3）逻辑“非”（也就是“Not”）运算中，“非”真为假，“非”假为真。

布尔类型数值和其他类型数值是可以相互转换的。

（1）数值型数据与布尔型数据的相互转换时，数值 0 转换为“False”，非 0 数值转换为“True”，所以我们也可以依照 C 语言的习惯，以 1 表示“真”，以 0 表示“假”，这种以 1 和 0 表示真假的方法要更为简洁。

（2）字符串型数据与布尔型数据相互转换时，只有空字符串（””）才会转换为“False”，其他字符串（哪怕是”0”）都会被转化为“True”。

个人建议直接以数值 1 代替“True”，以数值 0 代替“False”，这样在使用和记忆上都更为简便。

2、关系运算

关系运算十分常见，关系运算符的写法和我们数学中学习的略有差别，关系运算符列表如下：

关系运算符	名称	作用
=	等于	判断两个值是否相等（用于字符串时不区分大小写）
==	等于	判断两个值是否相等（用于字符串时区分大小写）
<>	不等于	判断两个值是否不相等
>	大于	判断第一个值（左边）是否大于第二个值（右边）
>=	大于等于	判断第一个值（左边）是否大于或等于第二个值（右边）
<	小于	判断第一个值（左边）是否小于第二个值（右边）
<=	小于等于	判断第一个值（左边）是否小于或等于第二个值（右边）

当关系运算符两侧的表达式满足关系运算所规定的条件时，结果为“真”，否则为“假”，例如：“6>5”，结果为“真”；“4<=7”，结果为“真”；“1=3”，结果为假。

请务必牢记这些关系与逻辑运算符的使用，在此后我们会经常用到这些关系与逻辑运算符。

第二节、If…Then…Else 语句

1、“If…Then…”语句

“If…Then…”语句用于判断当某条件为真时执行某个或某些语句。

“If…Then…”语句的语法：

（1）单行的“If…Then…”语句的语法：If <条件> Then <语句>，例如：

```
If $a>0 Then MsgBox(0,"范例","此数为正数")
```

（2）多行的“If…Then…”语句的语法：

```
If <条件> Then
    [语句或语句组]
EndIf
```

例如：

```
If $a>0 Then
    $b=$a+1
    $c=$a+$b
EndIf
```

也就是说，当“Then”后面只有一个语句时，可以将“If…Then…”语句写成单行形式，并且在结束时无需写“EndIf”；而如果“Then”后面有多个语句时，必须在“Then”后换行并书写多行语句，语句书写结束后，需要另起一行写上“EndIf”。

当“Then”后只有一个语句时也可以写成多行的形式，只是务必要写上“EndIf”，例如：

```
If $a>0 Then
    MsgBox(0,"范例","此数为正数")
EndIf
```

2、“If…Then…Else”语句

“If…Then…Else”语句用于判断当某条件为真时执行哪个或哪些语句，当某条件为假时又执行哪个或哪些语句。

“If…Then…Else”语句的语法：

```
If <条件> Then
    [语句或语句组 1]
Else
    [语句或语句组 2]
EndIf
```

即当“条件”为真时执行“语句或语句组 1”，否则执行“语句或语句组 2”。例如：

```
If $a>0 Then
    MsgBox(0,"范例","此数为正数")
Else
    MsgBox(0,"范例","此数为 0 或负数")
EndIf
```

3、“If…Then…Else”语句的嵌套与重复使用

“If…Then…Else”语句中可以包含“If…Then…Else”语句语句，例如：

```
If $a>0 Then
    MsgBox(0,"范例","此数为正数")
Else
    If $a<0 Then
        MsgBox(0,"范例","此数为负数")
    Else
        MsgBox(0,"范例","此数为 0")
    EndIf
EndIf
```

这就是一个简单的“If…Then…Else”语句嵌套的例子，嵌套时，“Else”总是去上文中寻找与它最近的“If”相匹配。

同样的，我们也可以多次使用“If…Then…Else”语句来实现上述程序段的功能：

```
If $a>0 Then
    MsgBox(0,"范例","此数为正数")
ElseIf $a<0 Then
    MsgBox(0,"范例","此数为负数")
Else
    MsgBox(0,"范例","此数为 0")
EndIf
```

其中，“ElseIf”是一个整体，中间无空格。

无论是使用嵌套还是重复使用，都请注意“If”、“Else”、“EndIf”的匹配关系，细心的朋友可能已经发现了，我在书写“If…Then…Else”语句会有一个规律性的“缩进”，有了代码的“缩进”后可以很容易的判明匹配关系。

“缩进”并不会对程序的执行造成什么影响，但是却可以有效的提高程序的可读性，方便查证错误与修改。

下面我们看一个简单的例子：

输入一个学生的成绩，60 分以下显示“不及格”，60~80 分显示“及格”，80~90 分显示“良好”，90~100 分显示“优秀”。

【代码】

```
Dim $s
$s=InputBox("学生成绩","请输入学生的成绩")
If @error=1 Or $s="" Then
    Exit
EndIf
If $s>=0 And $s<=100 Then
    If $s>90 Then
        MsgBox(0,"学生成绩","优秀！")
    ElseIf $s>80 Then
        MsgBox(0,"学生成绩","良好！")
    ElseIf $s>=60 Then
        MsgBox(0,"学生成绩","及格！")
    Else
        MsgBox(0,"学生成绩","不及格！")
    EndIf
Else
    MsgBox(0+16,"学生成绩","学生成绩输入错误！")
```

EndIf

【简析】

首先我们声明一个变量$s 来代表学生的成绩，随后使用 InputBox 要求用户输入学生的成绩并将 InputBox 的返回值赋值给$s，紧接着一个简单的判断当用户按下“取消”或者 InputBox 运行出错时退出程序，随后便进入了分数判定阶段。

分数判定阶段同时使用了“If…Then…Else”语句的嵌套与重复使用。首先判定输入的分数是否大于等于 0 且小于等于 100，否则判定为无效分数，发出错误提示。在分数大于等于 0 且小于等于 100 的范围内，详细判定输入的分数处于哪个分数段之间，并显示相关提示。

这个例子虽然并不复杂，但却将本节所学内容统统用上，所以请您仔细阅读这个例子，并能牢固把握“If…Then…”语句与“If…Then…Else”语句的使用方法。

第三节、Select...Case 与 Switch...Case 语句

在第二节中我们使用“If…Then…Else”语句解决了判定学生分数处于哪个水平的问题，但是大家可能都发现了，如果要判定多个分支，使用“If…Then…Else”语句是一种十分繁杂且不易阅读的方式。势必我们需要一种可以判定多个分支的语句。

1、“Select...Case”语句

“Select...Case”语句语法：

```
Select
    Case <条件 1>
        [语句或语句组 1]
    Case <条件 2>
        [语句或语句组 2]
    [Case Else]
        [语句或语句组 3]
EndSelect
```

其中，“Case Else”段并不是必须的，可以不写。如果用“Select...Case”语句来解决上一节中的判定学生分数处于哪个水平的问题就要简单多了。

【代码】

```
Dim $s
$s=InputBox("学生成绩","请输入学生的成绩")
If @error=1 Or $s="" Then
    Exit
EndIf
Select
    Case $s>90 And $s<=100
        MsgBox(0,"学生成绩","优秀！")
    Case $s>80 And $s<=90
        MsgBox(0,"学生成绩","良好！")
    Case $s>=60 And $s<=80
        MsgBox(0,"学生成绩","及格！")
    Case $s>=0 And $s<60
```

```
        MsgBox(0,"学生成绩","不及格！")
    Case Else
        MsgBox(0+16,"学生成绩","学生成绩输入错误！")
EndSelect
```

如上例所展现的，在判定多分支时，“Select...Case”语句比“If…Then…Else”语句更具有优势，而且程序本身更加规整、易阅读。

2、“Switch...Case”语句

“Switch...Case”语句与“Select...Case”语句十分相似，“Switch...Case”语句在判定同一个表达式的多种情况时比“Select...Case”语句更为简洁。

“Switch...Case”语句的语法：

```
Switch <表达式>
    Case <关于表达式的条件 1>
        [语句或语句组 1]
    Case <关于表达式的条件 2>
        [语句或语句组 2]
    Case Else
        [语句或语句组 3]
EndSwitch
```

其中，“Case Else”段不是必须的，可以不写。如果用“Switch...Case”语句来解决刚才的问题，那么会更加简单。

【代码】

```
Dim $s
$s=InputBox("学生成绩","请输入学生的成绩")
If @error=1 Or $s="" Then
    Exit
EndIf
Switch $s
    Case 91 To 100
        MsgBox(0,"学生成绩","优秀！")
    Case 81 To 90
        MsgBox(0,"学生成绩","良好！")
    Case 60 To 80
        MsgBox(0,"学生成绩","及格！")
    Case 0 To 59
        MsgBox(0,"学生成绩","不及格！")
    Case Else
        MsgBox(0+16,"学生成绩","学生成绩输入错误！")
EndSwitch
```

上例中，“Switch $s”规定了此后所有“Case”后的条件都是针对“$s”的，而像“91 To 100”代表着 91 到 100 之间的数值。

由此可见，在解决针对某一表达式（这里是变量“$s”）的问题时，“Switch...Case”语句相对“Select...Case”语句更具有优越性，而“Select...Case”语句在判定多分支时更具有灵活性。

第四节、选择结构程序设计实例

1、将三个数值从小到大排序。

【代码】

```
Dim $a,$b,$c
Dim $t
$a=6
$b=2
$c=7
If $a>$b Then
    $t=$a
    $a=$b
    $b=$t
EndIf
If $a>$c Then
    $t=$a
    $a=$c
    $c=$t
EndIf
If $b>$c Then
    $t=$b
    $b=$c
    $c=$t
EndIf
MsgBox(0,"三个数排序","从小到大依次为："&$a&"，"&$b&"，"&$c)
```

【简析】

这是个看似简单的问题，但是对于初学者不费一番脑筋也的确不好琢磨。

首先我们定义了三个变量“$a”、“$b”、“$c”，随后定义了一个用来做数据临时存放的变量“$t”。接着我们任意为“$a”，“$b”，“$c”赋三个数值型数据。

我们先来解释一下如下三行代码的含义：

“$t=$a”，将“$a”的数据存放到“$t”中。

“$a=$b”，用“$b”中的数据覆盖“$a”的原始数据。

“$b=$t”，将“$t”中存放的“$a”的原始数据覆盖“$b”的原始数据。

经过这三步，借助一个中间变量“$t”我们将“$a”与“$b”中存放的数据进行了交换。同样的，其他的类似语句也是将两个数据借助中间变量进行交换。

好，现在我们再回头看一下三个判断语句。

如果“$a>$b”则将“$a”与“$b”的数值进行交换，此时“$a”是“$a”与“$b”中最小的；

如果“$a>$c”则将“$a”与“$c”的数值进行交换，此时“$a”是“$a”与“$c”中最小的，同时“$a”也是“$a”、“$b”、“$c”中最小的；

如果“$b>$c”则将“$b”与“$c”的数值进行交换，此时“$b”是“$b”与“$c”中最小的，同时“$c”也成为了“$a”、“$b”、“$c”中最大的。

如此以来，“$a”存放的是最小数值，“$b”存放的是次最小数值，“$c”存放的是最大数值。这样，也就完成了对三个数值的排序。

2、输入一个四位以内的整数，例如 4580，12，986 等，要求：

（1）判明这个输入的数是几位数

（2）输入这个数的每一位数

（3）按照反向重组并输出一个新数

例如，输入的是 123，则输出：这是一个 3 位数，个位是 3，十位是 2，百位是 1，重组的新数为 321。

【代码】

```
Dim $Unit=0,$Ten=0,$Hundred=0,$Thousand=0
Dim $Digit=1
Dim $Num,$NewNum
$Num=InputBox("数字","请输入一个不大于 4 位数的整数")
If @error=1 Or $Num="" Then
    Exit
EndIf
Select
    Case $Num>=0 And $Num<10
        $Digit=1
        $Unit=Mod($Num,10)
        $NewNum=$Unit
    Case $Num>=10 And $Num<100
        $Digit=2
        $Unit=Mod($Num,10)
        $Ten=Mod(($Num-$Unit)/10,10)
        $NewNum=$Unit*10+$Ten
    Case $Num>=100 And $Num<1000
        $Digit=3
        $Unit=Mod($Num,10)
        $Ten=Mod(($Num-$Unit)/10,10)
        $Hundred=Mod(($Num-$Ten*10-$Unit)/100,10)
        $NewNum=$Unit*100+$Ten*10+$Hundred
    Case $Num>=1000 And $Num<10000
        $Digit=4
        $Unit=Mod($Num,10)
        $Ten=Mod(($Num-$Unit)/10,10)
        $Hundred=Mod(($Num-$Ten*10-$Unit)/100,10)
        $Thousand=Mod(($Num-$Hundred*100-$Ten*10-$Unit)/1000,10)
        $NewNum=$Unit*1000+$Ten*100+$Hundred*10+$Thousand
EndSelect
MsgBox(0,"数字","这是一个"&$Digit&"位数"&@CRLF _
    &"个位："&$Unit&@CRLF&"十位："&$Ten&@CRLF&"百位："&$Hundred&@CRLF _
    &"千位："&$Thousand&@CRLF&"重组的新数："&$NewNum)
```

【简析】

这是一个看似复杂的问题，但是千万不要把这个问题考虑的复杂了。对于复杂的问题，我们要做的是把它分解成简单的小问题。

（1）如何判断位数？

这个很简单，大于等于 0 小于 10 的肯定是一位数，大于等于 10 小于 100 的肯定是两位数，并依次类推三位数和四位数的取值区间。

（2）如何获得每位的数值？

一下子获取每一位似乎有点困难，那么我们就来看看怎么获取个位上的数值吧。假设数值 12，我们用 12/10=1.2，也就是说商 1 余 2，而这个余数恰恰就是个位的数字，所以要取得个位上的数值，只需要将整个数值除以 10 取余数就可以了。

那么如果十位、百位、千位的怎么取？我们可以想办法把十位、百位、千位都变成个位。

（3）怎么反向输出这个数？

如果得到了每位的值，这个就简单了。例如 123，个位是 3，十位是 2，百位是 1，则用个位*100+十位*10+百位，得到 321。

经过这么分析后，整个程序的编写思路就明确了。

另，@CRLF 是回车换行符，用于字符串的换行，后续讲解字符串的专门章节中会详细讲解，在此仅需了解它的作用即可。

本章我们学习了程序设计中最常使用的结构——选择结构，运用选择结构，可以让我们的程序在不同的条件下执行不同的动作，这可以让程序丰富起来。善用选择结构，可以让您的程序适应更多的使用环境，并且更为精细。

《Let’s AutoIt》——第四章、选择结构程序设计

完成于 2008-8-12 凌晨，2008-8-14 第一次修正

作者：Skyfree
QQ：165718402
E-Mail:Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第五章、循环结构程序设计

在本章中，我们将和您一起学习 AutoIt 中循环结构程序设计。循环结构在程序中的使用频率相当高，循环结构与顺序结构、选择结构一起，构造复杂程序的基本单元。

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

存在这么一类问题，需要将某段过程执行 N 次，或者每次执行有相似之处，例如我们要计算 1+2+3+……+10 的值，当然我们可以直接写：

$Sum=1+2+3+4+5+6+7+8+9+10

这个在数值较少时还可以使用，万一我们要计算 1+2+3+……+100000 的值怎么办？手工一个个的写不比数星星快多少。

面对这类问题，我们要使用一种新的程序设计结构来解决——循环结构。

所谓循环，就是将某一段程序按照某种规则执行一定次数。学习循环时脑中始终要警惕这么几件事：

（1）进入循环时或者说开始循环时各变量或数值的情况；

（2）以什么条件结束循环，如果进入循环后未设定在什么情况下结束或跳出循环，那么这将是一个“死循环”。大多数时候“死循环”是由于错误产生的，但并非“死循环”就是错误，某些情况下我们还要利用死循环；

（3）每次循环有哪些变量是在改变的，而又有哪些是不变的；

（4）知道（至少是大概知道）要循环多少次后才会结束。

循环结构是学习编程的第一个坎，很多人栽在这里，随后成了“学过编程的人”。Skyfree 我第一次学习 C 语言的循环结构时前后用了相当长的时间，最后才迈过了这道坎。如果前面的内容您都看懂了，在这一章里可能会出现您一眼看不懂的内容，所以一定要有耐心、有信心。

第一节、“While…WEnd”循环

“While…WEnd”语句语法：

```
While <表达式（条件）>
    [语句或语句组（循环体）]
WEnd
```

这里的表达式是一个返回布尔值的关系或逻辑表达式，这个表达式是“While…WEnd”正常运行的条件，如果不满足这个表达式规定的条件则循环结束。

“While…WEnd”循环可以理解为：当表达式要求的条件成立时，执行循环体。

例如我们要计算 1+2+3+……+100 的结果，我们要这么书写代码：

【代码】

```
Dim $Sum,$i
$Sum=0
$i=1
While $i<=100
    $Sum=$Sum+$i
    $i=$i+1
WEnd
MsgBox(0,"循环范例",$Sum)
```

【简析】

首先定义了“$Sum”和“$i”两个变量，“$Sum”用来存储加和，“$i”用来做渐变的加数，所以我们为“$Sum”赋初值 0，为“$i”赋初值 1。

随后我们进入了“While…WEnd”循环，循环正常进行的条件是“$i<=100”，即循环结束的条件是“$i>100”。循环中，每次循环我们都让当前的“$Sum”加当前的“$i”并再赋值给“$Sum”，这样“$Sum”就记录了每一次“$i”改变后的加和。同时，“$i”在每次循环后都增加 1。

仔细理解一下由两行语句组成的循环体，每一次“$i”都在改变，这样“$i”就依次代表 1、2、3、4……99、100，而“$Sum”则将每一次循环的“$i”值累加，这样也就实现了求 1+2+3+……+100 的值。

如果您还没看太明白，我将循环拆解一下：

（1）第一次循环（进入循环）

此时“$Sum”的值为 0，“$i”的值为 1；

循环正常执行的条件是“$i<=100”，此时的“$i” 值为 1，满足条件，第一次循环正式开始；

执行“$Sum=$Sum+$i”语句，相当于执行“$Sum=0+1”，执行结束后“$Sum”的值为 1；

执行“$i=$i+1”语句，相当于执行“$i=1+1”，执行后“$i”的值为 2；

第一次循环结束。

（2）第二次循环

此时“$Sum”的值为 1，“$i”的值为 2；

循环正常执行的条件是“$i<=100”，此时的“$i” 值为 2，满足条件，第二次循环正式开始；

执行“$Sum=$Sum+$i”语句，相当于执行“$Sum=1+2”，执行结束后“$Sum”的值为 3；

执行“$i=$i+1”语句，相当于执行“$i=2+1”，执行后“$i”的值为 3；

第二次循环结束。

（3）第三次循环

此时“$Sum”的值为 3，“$i”的值为 3；

循环正常执行的条件是“$i<=100”，此时的“$i” 值为 3，满足条件，第三次循环正式开始；

执行“$Sum=$Sum+$i”语句，相当于执行“$Sum=3+3”，执行结束后“$Sum”的值为 6；

执行“$i=$i+1”语句，相当于执行“$i=3+1”，执行后“$i”的值为 4；

第三次循环结束。

（4）第四次循环

……

或者，我们用更为直观的方式表达一下：

（1）第一次循环（进入循环）：$Sum=0+1，$i =1+1，则现在“$Sum”等于 1，“$i”等于 2；

（2）第二次循环：$Sum=1+2，$i =2+1，则现在“$Sum”等于 3，“$i”等于 3；

（3）第三次循环：$Sum=3+3，$i =3+1，则现在“$Sum”等于 6，“$i”等于 4；

（4）第四次循环：……

……

循环将会这么一直执行下去，直到“$i”的值为 101 时，由于循环正常执行的条件是“$i<=100”，此时将会退出循环，执行剩下的语句。

就经验来说，循环中总有一个变量在循环时改变，而这个变量与判断循环是否执行的表达式有一定的关系，当这个变量到达某个值时，打破了进行循环的条件，循环结束。

“While…WEnd”循环是最基本的循环，也是我们接触的第一个关于循环结构程序设计的语句。刚接触循环语句时可能会有一些困难和不好理解，但请务必一定要掌握这个最基本循环的用法。如果您在看完本节后仍无法理解 “While…WEnd”循环，那么请您将本节反复理解几次，不要急着看下几节的内容。

第二节、“Do...Until”循环

“Do...Until”语句语法：

```
Do
    [语句或语句组（循环体）]
```

Until <表达式（条件）>

这里的表达式是一个返回布尔值的关系或逻辑表达式，这个表达式是“Do...Until”结束运行的条件，如果不满足这个表达式规定的条件则正常循环。

“Do...Until”循环可以理解为：执行循环体，直到满足表达式规定的条件时结束循环。

“Do...Until”循环其实与“While…WEnd”循环的作用类似，差别主要有两点：

（1）“Do...Until”循环第一次执行循环体时并不检测表达式规定的条件，所以“Do...Until”循环至少会执行一次；

（2）“Do...Until”循环的表达式所规定的条件是达到这个条件后退出循环，这是与“While…WEnd”循环最为不同的地方。

很多人习惯把“While…WEnd”循环称为“当”循环，即“当”达到表达式规定的条件时执行循环；而把“Do...Until”循环称为“直到”循环，即执行循环体，“直到”满足退出条件时结束循环。

下面我们使用“Do...Until”循环实现计算 1+2+3+……+100 的和。

【代码】

```
Dim $Sum,$i
$Sum=0
$i=1
Do
    $Sum=$Sum+$i
    $i=$i+1
Until $i>100
MsgBox(0,"循环范例",$Sum)
```

一般情况下“Do...Until”循环可以实现的功能都可以用“While…WEnd”循环来完成，所以我们通常多用“While…WEnd”循环，而并不多用“Do...Until”循环。

第三节、“For...Next”循环

“For...Next”循环一定程度上讲比“While…WEnd”循环使用的频率要更高一些，尤其是在与后续几章将要学习的数组等相结合时，其作用显得十分重要。“For...Next”循环是继“While…WEnd”循环之后更为重要的一个循环结构。

“For...Next”循环语法：

```
For <变量> = <开始> To <停止> [Step <步进值>]
    [语句或语句组（循环体）]
Next
```

在“For...Next”循环开始时，我们首先为一个已经定义的变量赋一个初值并为其设置一个终值，而后每循环一次这个变量就以步进值为基数由初值靠近终值，步进值省略时默认步进值为 1。

下面我们使用“For...Next”循环实现计算 1+2+3+……+100 的和。

【代码】

```
Dim $Sum,$i
$Sum=0
For $i=1 To 100
    $Sum=$Sum+$i
Next
MsgBox(0,"循环范例",$Sum)
```

【简析】

首先，我们声明了变量“$Sum”和“$i”，其中“$Sum”用以盛放计算结果，并为其赋初值 0。

然后，进入“For...Next”循环。“For...Next”循环中，我们为“$i”赋初值 1，并为其设定终值 100，由于并未设定步进值，所以步进值默认为 1。也就是说，在循环时，“$i”将依次等于 1、2、3、……、99、100。那么，在循环体“$Sum=$Sum+$i”执行时，每次将当前“$Sum”与当前“$i”的和值赋值给“$Sum”，这样“$Sum”也就充当了盛放累加结果的作用。

随着“$i”的递增，最终会达到终值 100，此时循环结束，退出循环，继续执行剩下的语句。

为了便于理解，我们将这段循环做一下拆解：

（1）第一次循环（进入循环）

为“$i”赋初值 1，并为其设定终值 100，步长值未设定则步长值为 1；

第一次循环“$i”的值为 1，未超过终值，所以循环正常执行；

“$Sum”的值为 0，执行“$Sum=$Sum+$i”语句，相当于执行“$Sum=0+1”，执行结束后“$Sum”的值为 1；

第一次循环结束。

（2）第二次循环

第二次循环“$i”的值为 2，未超过终值，所以循环正常执行；

“$Sum”的值为 1，执行“$Sum=$Sum+$i”语句，相当于执行“$Sum=1+2”，执行结束后“$Sum”的值为 3；

第二次循环结束。

（3）第三次循环

第三次循环“$i”的值为 3，未超过终值，所以循环正常执行；

“$Sum”的值为 3，执行“$Sum=$Sum+$i”语句，相当于执行“$Sum=3+3”，执行结束后“$Sum”的值为 6；

第三次循环结束。

（4）第四次循环

……

或者，我们用更直观的方法表达一下：

（1）第一次循环（进入循环），$i=1，$Sum=0+1，现在的“$Sum”等于 1；

（2）第二次循环，$i=2，$Sum=1+2，现在的“$Sum”等于 3；

（3）第三次循环，$i=3，$Sum=3+3，现在的“$Sum”等于 6；

（4）第四次循环，……

……

循环将这样一直执行下去，直到“$i”的值到达其终值 100，循环将会退出，执行剩下的语句。

“For...Next”循环在某些条件下的使用要比“While…WEnd”循环更为简便。“For...Next”循环一般在下述几种情况时使用：

（1）做已知或大致已知次数的循环时；

（2）循环体中有与循环次数相同或相关的变量时。

在实际程序设计中，“For...Next”循环的使用频率相当高，所以“For...Next”循环也是极为重要的一种循环模式，请您务必理解“For...Next”循环的使用。

第四节、循环的嵌套

在上一章中我们用到了选择语句的嵌套，所谓嵌套，就是在一种模式中再使用相同或相似的模式。循环的嵌套十分常用，很多问题不可能靠一个循环语句来解决，这种时候我们就需要在一个循环里灵活的嵌套另一个循环。

我们来看这么一个题目，要求输出如下数字矩阵：

11　12　13　14　15

21　22　23　24　25

31　32　33　34　35

【代码】

```
Dim $i,$j
Dim $s
$s=""
For $i=1 To 3
    For $j=1 To 5
        $s=$s&" "&($i*10+$j)
    Next
    $s=$s&@CRLF
Next
MsgBox(0,"循环嵌套范例",$s)
```

【简析】

这个三行的数字矩阵其实是一个有换行的字符串，字符串的第一行为“11　12　13　14　15”，然后跟一个“@CRLF”（回车换行符），再紧跟下一行“21　22　23　24　25”，而后依次类推。所以，我们需要一个字符串变量来存放这个数字矩阵。

这个数字矩阵有如下规律：

（1）每行的个位数均为 1 到 5 递增，即列数是个位；

（2）从上到下，十位从 1 到 3 递增，即行数是十位。

那么如果用变量“$i”代表十位（行数），用变量“$j”作为个位（列数），那么第“$i”行第“$j”个数应该为“$i*10+$j”。这样我们就知道该怎么做了

首先定义了“$i”和“$j”，用来做两个循环中的递增变量。然后定义“$s”来盛放数字矩阵的的值，并把“$s”赋值为空字符串。

以下为描述方便，将“For $i=1 To 3”叫做第一个循环，把“For $j=1 To 5”叫做第二个循环。

进入第一个循环，设定“$i”的初值为 1，终值为 3，步长值省略即步长值为 1。

第一个循环每循环一次，第二个循环都要循环五次，而且每次都要将“$j”的值重新赋值为 1，并每次将其终值设定为 5。所以，“$s=$s&" "&($i*10+$j)”总共要执行 15 次，而“$s=$s&@CRLF”由于只属于第一个循环而不属于第二个循环，所以和第二个循环一样只执行 3 次。

在第二个循环中，“($i*10+$j)”的值会像我们在一开始分析的那样依次改变，而“$s=$s&" "&($i*10+$j)”则用来收集这个改变。并在第二个循环结束后，也就是“写”完数字矩阵一行的数字后，自动跟一个回车换行“$s=$s&@CRLF”。

在第一个循环执行 3 次后（此时第二个循环已经执行了 15 次），“$i”达到其终值 3，则第一个循环结束，跳出循环，执行剩下的语句。

循环嵌套在程序设计中十分常见，而且循环嵌套又牵涉到之后要学到的多维数组等，所以特别重要。不过不要为现在还不是很明白循环嵌套发愁，只要您能理解非嵌套的循环，理解循环嵌套就只是个时间与耐心的问题了。

第五节、循环结构程序设计实例

循环结构中还有一个语句，虽然使用起来会带来方便，但是不建议经常使用，只在不使用这个语句将使整个程序的复杂性增加、可阅读性降低时使用。

“ExitLoop”语句，作用是强制跳出当前循环。可以使用选择语句与之相结合，判定当满足某条件时强制跳出循环。要注意的是这个只会跳出当前循环，也就是说，像刚才第四节中的循环嵌套中，如果在第二个循环中有“ExitLoop”语句，那么只会跳出第二个循环，而不会跳出第一个循环。

“ExitLoop”语句只要知道即可，不建议经常使用。

1、计算 100 以内所有奇数的和

【代码】

```
Dim $i
Dim $Sum
$Sum=0
For $i=1 To 100 Step 2
    $Sum=$Sum+$i
Next
MsgBox(0,"循环结构实例",$Sum)
```

【简析】

这个其实和我们计算 1+2+3+……+100 是一样的，只不过这次我们只需要 1 到 100 中的所有奇数。100 以内的奇数，例如 1、3、5、7……等，是一个步进为 2 的等差数列，看到这个规律后，只需要将 1+2+3+……+100 的题目中的“For…Next”的步长值设为 2，让“$i”依次代表 1、3、5、7 等即可。

2、计算 5 的阶乘

【代码】

```
Dim $i,$r
$r=1
For $i=5 To 1 Step -1
    $r=$r*$i
Next
MsgBox(0,"循环结构实例",$r)
```

【简析】

所谓阶乘，例如要计算 5 的阶乘，就是计算 5×4×3×2×1 的值，那么这个程序的写法也就可想而之了。当然，您也可以计算 1×2×3×4×5 的值以达到相同的目的，不过本例题的作用是告诉您：步长可以为负值。

3、输出一个乘法口诀表，如：

1×1=1
2×1=2　2×2=4
3×1=3　3×2=6　3×3=9
4×1=4　4×2=8　4×3=12　4×4=16
……

【代码】

```
Dim $i,$j
Dim $s
$s=""
For $i=1 To 9
    For $j=1 To $i
        $s=$s&$i&"×"&$j&"="&($i*$j)&" "
```

```
        Next
        $s=$s&@CRLF
    Next
    MsgBox(0,"循环结构实例",$s)
```

【简析】

这个十分类似我们之前的输出数字矩阵的例题，但是有一点不同：每行的数据数量不同。

我们来找一下乘法口诀表的规律：

（1）每行的数被乘数与本行行数相同相同；

（2）每行的乘数最大值也与本行行数相同。

这样以来，第一个循环“For $i=1 To 9”的“$i”为从 1 到 9，而第二个循环“For $j=1 To $i”的“$j”则每次都是从 1 到当前的“$i”，这样才会出现类似乘法口诀表的阶梯状数据。

如果您没看很明白，请亲自手工展开这个嵌套循环，就像之前我为大家展开“While…WEnd”和“For…Next”循环一样。

另一个要解释的是“$s=$s&$i&"×"&$j&"="&($i*$j)&" "”，这行可能比较长，但不难理解，关键在于“$i&"×"&$j&"="&($i*$j)”，不过如果对字符串的操作还有疑惑，不用担心，下一章我们将彻彻底底的学习字符串的相关知识。

4、现有 10 元人民币、20 元人民币、50 元人民币若干张，问如果用其中 20 张组成 600 元，可以怎么组？

【代码】

```
Dim $Ten,$Twenty,$Fifty
Dim $s
$s=""
For $Ten=0 To 20
    For $Twenty=0 To 20
        For $Fifty=0 To 12
            If $Ten+$Twenty+$Fifty=20 _
                And $Ten*10+$Twenty*20+$Fifty*50=600 Then
                $s=$s&"10 元："&$Ten&"张    " _
                    &"20 元："&$Twenty&"张    " _
                    &"50 元："&$Fifty&"张    "&@CRLF
            EndIf
        Next
    Next
Next
MsgBox(0,"循环结构实例",$s)
```

【简析】

判定用 20 张组 600 元该怎么组，最简单的方法就是拿着一大堆 10 元、20 元、50 元的人民币一次次的尝试（如果我拿着一大堆人民币我想我是不会去琢磨怎么组 600 元……），一直尝试到可以用 20 张组 600 元为止。这种方法理论上要尝试上千次，这几乎是可以让一个人受不了的次数。

其实这种一次次尝试的方法就被叫做“枚举法”，所谓枚举，就是一次次的举例并验证这个举例是否正确。虽然尝试上千次会让一个普通人类受不了，但是对于计算机来说这就像人类呼吸一样简单，所以我们可以把这种工作放心的交给计算机去做。

这样，我们分析一下代码。

首先我们定义了“$Ten”、“$Twenty”、“$Fifty”三个变量，分别代表 10 元的张数、20 元的张数、50 元的张数。

而要组成 600 元，10 元最多需要 60 张，20 元最多需要 30 张，50 元最多需要 12 张。而既然已经有规定最多使用 20 张，那么 10 元、20 元最多也只需要 20 张，所以我们可以使用三层循环来枚举所要使用的张数。

每次枚举后，判定两个条件是否成立：

（1）使用钞票的总数是否为 20 张（“$Ten+$Twenty+$Fifty=20”）

（2）钞票的总面额数是否为 600 元（“$Ten*10+$Twenty*20+$Fifty*50=600”）

只有当这两个条件都成立时才是正确结果，所以这两个条件使用“And”联结。确定结果正确后，将结果写入“$s”中：

```
$s=$s&"10 元："&$Ten&"张  "&"20 元："&$Twenty&"张  "&"50 元："&$Fifty&"张  "&@CRLF
```

这个问题是对枚举法的使用范例，也是了解人类与计算机计算能力差别的开始。

循环问题相对之前所学的任何问题都要复杂一些，主要是在处理循环体以及循环体中每次均在变化的变量时容易出现问题。学习循环结构需要较大的耐心，解决循环结构问题的一般思路为：

（1）找到循环改变规律；

（2）认清每次循环时哪些量在变；

（3）明确循环退出的条件，避免死循环。

循环问题与之后的学习息息相关，请认真对待！

《Let’s AutoIt》——第五章、循环结构程序设计

完成于 2008-8-14 午，2008-8-14 第一次修正

作者：Skyfree
QQ：165718402
E-Mail:Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第六章、字符串

之前各章中，我们已经逐步了解了 AutoIt 中的各种数据类型。在本章中，我们将要接触一个不算新颖但是很讲究的数据类型——字符串。

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

第一节、字符串型数据

在第二章中我们已经初步提到了字符串型数据，只是由于当时并未接触选择、循环等程序设计结构，所以并未做更详细的使用说明。本章中我们将详细了解一下有关字符串型数据的方方面面。

1、字符串变量的定义与赋值

字符串变量的定义与赋值和其他变量相同，没有特殊之处，例如：

```
Dim $s
$s="Www.FreeSkyCD.Cn"
```

或者

```
Dim $s="Www.FreeSkyCD.Cn"
```

只不过所有的字符或字符串必须以英文双引号（" "）或英文单引号（' '）进行“包含”，即所有的字符或字符串都必须包含在英文引号中。一般情况下，我们只使用两种引号中的一种，但为了与其他编程语言的习惯相匹配，我建议大家只使用英文双引号来包含字符串。

而如果出现字符串中就包含了英文双引号，例如：

```
"Good morning!"
```

这句话中开头与结尾本身就包括了两个英文逗号，如果我们以如下方式输出：

```
Dim $s="Good morning!"
MsgBox(0,"范例",$s)
```

输出的结果是没有字符串两侧英文双引号的，那么要怎么才能把字符串中的引号也输出出来呢？方法有两种：

（1）单双引号混合使用法

```
Dim $s=' "Good morning!" '
MsgBox(0,"范例",$s)
```

我们将英文单引号使用在最外侧，这样就告诉程序将以单引号作为包含字符串用的标志，这样双引号就会被作为字符串中的普通字符来看待。所以，如果运行上述程序则会将字符串两侧的英文双引号输出。

（2）两次使用同一引号法

```
Dim $s="""Good morning!"""
MsgBox(0,"范例",$s)
```

这个就要好好的理解一下了，“"""Good morning!"""”中，最外侧的双引号用来规定以双引号作为包含字符串的标志，而后字符串中的连续使用两个双引号“""”将被作为一个双引号“"”的普通字符来计算。即，字符串中连续使用两个与包含字符串标志相同的引号将会被视作是一个引号的普通字符。

第二种方法不如第一种方法更为直观，所以建议使用第一种方法。

2、字符串数据的连接

字符串数据的连接我们已经在之前的章节中接触了很多，相信大家已经对字符串的连接有了一定的了解。

字符串连接运算符：&

作用：用于连接运算符左侧与右侧的字符串

范例：

```
Dim $s1="Www."
```

```
Dim $s2="FreeSkyCD.Cn"
Dim $s
$s=$s1&$s2
MsgBox(0,"范例",$s)
```

输出结果为“Www.FreeSkyCD.Cn”（两侧无英文引号）。

特殊情况，当“&”两侧有一侧为非字符串型数据时，例如

```
Dim $s1="Beijing"
Dim $s2=2008
Dim $s
$s=$s1&$s2
MsgBox(0,"范例",$s)
```

此时“$s2”为数值型数据，在“$s=$s1&$s2”时其实是先将数值型的“2008”转换为字符串型的“"2008"”，然后再执行“&”运算。所以这个例子的输出结果是“Beijing2008”。

最特殊的情况，当“&”两侧均为非字符串型数据时，例如

```
Dim $s1=123
Dim $s2=45
Dim $s
$s=$s1&$s2
MsgBox(0,"范例",$s)
```

其实这种情况与“&”两侧有一侧为非字符串型数据时相同，此时数值型的“123”和“45”都会被转化为字符串型的“"123"”和“"45"”。所以这个例子的输出结果是字符串“12345”。

由此可见，“&”运算时，如果“&”两侧中都是字符串型数据，则直接执行“&”运算；如果“&”两侧中存在非字符串型的数据，将会自动将非字符串型数据转换为字符串行数据，然后再做“&”运算。

另外，还有一个“&=”运算符，称为“连续赋值运算符”，其实就是将“&=”左侧的变量与右侧的数据做“&”运算，而后再赋值给“&=”左侧的变量，例如

```
Dim $s1="Beijing"
Dim $s2=2008
$s1&=$s2
MsgBox(0,"范例",$s1)
```

其中“$s1&=$s2”相当于“$s1=$s1&$s2”。

3、关于字符串的宏

AutoIt 中有一种被叫做“宏”的特殊数据，“宏”具有与常量类似的性质，普通语句无法更改它的值，但某些函数可以更改并设置其中某些“宏”的值。在这里，您只需要将“宏”理解为一些由 AutoIt 预先定义好的常量即可。

AutoIt 中的所有“宏”为了与常变量相区别，使用“@”作为前缀名。AutoIt 中的“宏”有很多，这里只介绍与字符串操作相关的宏：

宏名	作用	ACSII 值
@CR	回车符（可以用于换行）	13
@LF	换行符（可以用于换行）	10
@CRLF	回车换行符（可以用于换行），相当于@CR&@LF	
@TAB	制表符	9

说明：

（1）@CR、@LF、@CRLF 都可以用于换行，只是以不同方式编辑的文档可能会使用不同的换

行符，所以才有这么三种与之对应。不过通常用 Windows 记事本编辑的纯文本文档是默认以@CRLF 换行的，所以推荐只使用@CRLF 进行换行操作。

（2）使用“宏”就像使用一个普通常量一样，例如：

```
Dim $s1="Www.FreeSkyCD.Cn"
Dim $s2="Www.AutoIt.Net.Cn"
MsgBox(0,"范例",$s1&@CRLF&$s2)
```

这段程序的输出结果是将“Www.FreeSkyCD.Cn”和“Www.AutoIt.Net.Cn”分两行显示。

（3）每个字符都有其对应的 ASCII 值，计算机实质上是将所有字符都转化成 ASCII 值进行存储的，这个将在下一节中进行介绍。

第二节、ASCII

ASCII 列表是一份 ASCII 值与其相应字符对应关系的列表，ASCII 值供有 256 个（0~255），其中标准 ASCII 值有 128 个（0~127），IBM-PC 专用 ASCII 值 128 个（128~255）。一般情况下我们只使用标准 ASCII 值，不过这并不意味着我们一定要记忆这 128 个值，只要记住其中较为重要的几个即可。

这时我们要引入两个 AutoIt 的函数：Chr 和 Asc。

函数	作用	范例
Chr(ASCII 值)	将 ASCII 值转化为字符	Chr(97)，结果为“a”
Asc(字符)	将字符转化为 ASCII 值	Asc("a")，结果为 97

也就是说 Chr 和 Asc 是一组互逆的转化函数。

ASCII 常用值：

（1）0，ASCII 值为 0 的字符是“空字符”，这个字符不能直接显示。虽然我们没直接使用过，但是我们所使用的每个字符串均是以“空字符”结尾的，也就是说在每个字符串的末尾都有一个空字符。其实系统在读取一个字符串时是一个一个字符进行读取的，当读取到“空字符”时字符串读取就结束。我们来看下面的例子：

```
Dim $s1="Beijing"
Dim $s2=2008
Dim $s
$s=$s1&Chr(0)&$s2
MsgBox(0,"范例",$s)
```

这个例子的输出结果为“Beijing”而非“Beijing2008”。 在“$s=$s1&Chr(0)&$s2”执行完毕后，“$s”为“Beijing‘空字符’2008‘空字符’”，这样系统在输出这个字符串时在读取到第一个“空字符”时就会认为这个字符串结束了，所以只会输出第一个“空字符”之前的内容，即“Beijing”。

（2）10、13，ASCII 值为 10 的字符是换行符，值为 13 的字符是回车符，这两个字符都不能直接显示。Chr(10)对应宏@LF，Chr(13)对应宏@CR，这两个 ASCII 值在处理字符串时会经常使用。

（3）34，ASCII 值为 34 的字符是英文双引号（"），那么

```
Dim $s=' "Good morning!" '
MsgBox(0,"范例",$s)
```

这个例子我们可以写成：

```
Dim $s=Chr(34)&"Good morning!" &Chr(34)
MsgBox(0,"范例",$s)
```

输出结果同样为“"Good morning!"”。

（4）65~90、97~122，ASCII 值分别对应 A~Z、a~z。这个在有些时候也是常用的，例如用循环的方法输出 A~Z：

```
Dim $i,$s=""
For $i=65 To 90
    $s=$s&" "&Chr($i)
Next
MsgBox(0,"范例",$s)
```

或者用于判定一个字符是否为字母：

```
Dim $c
$c=InputBox("范例","输入任意字符")
If @error=1 Or $c="" Then
    Exit
EndIf
If (Asc($c)>=65 And Asc($c)<=90) Or (Asc($c)>=97 And Asc($c)<=122) Then
    MsgBox(0,"范例","是字母")
Else
    MsgBox(0,"范例","不是字母")
EndIf
```

本节只介绍了常用的 ASCII 值，其他 ASCII 值请百度一下。

第三节、字符串相关函数

针对字符串的操作是多种多样的，很多时候会有一些较为复杂的操作，为了实现这些操作，AutoIt 中提供了大量针对字符串操作的函数。

AutoIt 字符串操作常用函数如下：

函数名	语法	作用
StringLeft	StringLeft("字符串",数量)	返回字符串中从左开始指定数量的字符
StringRight	StringRight("字符串",数量)	返回字符串中从右开始指定数量的字符
StringMid	StringMid("字符串",起始位置[,数量])	取字符串从起始位置起指定数量个字符（数量省略则取从起始位置起所有字符）
StringLen	StringLen("字符串")	返回指定字符串的字符总数
StringLower	StringLower("字符串")	转换字符串为小写字母并返回
StringUpper	StringUpper ("字符串")	转换字符串为大写字母并返回
StringReplace	StringReplace("字符串","查找字符串", "替换字符串",[替换次数,[是否区分大小写]])	在字符串中查找指定字符串并替换，可以规定只替换其中几个并规定是否区分大小写（0 不区分，1 区分，2 使用基本/快速的比较方法）
StringSplit	StringSplit("字符串","分隔符"[,标志])	把字符串按照分隔符进行分割并返回，标志为 0 时把分隔符中的每个字符作为分隔标准，为 1 时把分隔符整体作为分隔标准。返回一个数组，第一个元素（$array[0]）保存拆分后子串的数量，其余元素（$array[1]、$array[2] 等等）则保存着拆分后的每个字符串。

		若（在目标字符串中）未发现分隔符则 @error 将被设为 1，子串数量（$array[0]）等于 1，而函数将返回整个字符串（$array[1]）
StringStripCR	StringStripCR("字符串")	去掉字符串中所有回车符
StringAddCR	StringAddCR("字符串")	字符串中所有换行符前添加回车符
StringStripWS	StringStripWS("字符串",标志)	去掉字符串中所有空格，标志：1 去掉左边空格，2 去掉右边空格，4 去掉双（或多）空格，8 去掉所有空格
StringTrimLeft	StringTrimLeft("字符串",数量)	删除字符串中从左开始指定数量的字符
StringTrimRight	StringTrimRight("字符串", 数量)	删除字符串中从右开始指定数量的字符
StringCompare	StringCompare("字符串 1","字符串 2" [,大小写区分])	比较字符串 1 和 2，可以规定只替换其中几个并规定是否区分大小写（0 不区分，1 区分，2 使用基本/快速的比较方法）。 返回值为 0 两字符串相等，大于 0 则字符串 1 大于 2，小于 0 则字符串 1 小于 2
StringInStr	StringInStr("字符串", "子字符串"[,区分大小写[,出现次序 [,开始[,数量]]]])	在字符串总查找子字符串，可以规定只替换其中几个并规定是否区分大小写（0 不区分，1 区分，2 使用基本/快速的比较方法）。“出现次序”规定提取出现的第几个子字符串，“开始”规定从第几个字符开始搜索，“数量”规定一共查找几个子字符串。搜索成功返回子字符串位置，否则返回 0。
StringIsAlNum	StringIsAlNum("字符串")	检查某个字符串是否仅含有字母或数字，是返回 1，否返回 0
StringIsAlpha	StringIsAlpha("字符串")	检查某个字符串是否仅含有字母，是返回 1，否返回 0
StringIsASCII	StringIsASCII("字符串")	检查某个字符串是否仅含有 ASCII 码值介于 0x00 到 0x7f (0 - 127)之间的字符，是返回 1，否返回 0
StringIsDigit	StringIsDigit("字符串")	检查某个字符串是否仅含有数字(0-9)字符，是返回 1，否返回 0
StringIsFloat	StringIsFloat("字符串")	检查某个字符串是否为浮点数（如 1.25），是返回 1，否返回 0
StringIsInt	StringIsInt("字符串")	检查某个字符串是否整型数（如 1、-3），是返回 1，否返回 0
StringIsLower	StringIsLower("字符串")	检查某个字符串是否仅含有小写字母，是返回 1，否返回 0
StringIsSpace	StringIsSpace("字符串")	检查某个字符串是否仅含有空格，是返回 1，否返回 0
StringIsXDigit	StringIsXDigit("字符串")	检查某个字符串是否仅含有十六进制字符(0-9，A-F) ，是返回 1，否返回 0

这里只简单介绍了这些函数的使用方法，更详细的内容请参阅 AutoIt 帮助文档。

AutoIt 中关于字符串的操作就讲到这里，关于字符串的操作比普通变量要复杂一些，但是您却没必要去记忆所有关于字符串操作的函数，只记住其中几个重要的，如 StringMid、StringLen、StringSplit、StringInStr 等即可，其他的可以在需要时再查阅。字符串的操作需要更多的是细心，请认真对待每一个字符串数据。

《Let's AutoIt》——第六章、字符串

完成于 2008-8-15 晨，2008-8-15 午第一次修正

作者：Skyfree
QQ：165718402
E-Mail：Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第七章、数组

本章中将接触一个新的变量类型——数组。数组型变量与之前学习的循环结构程序设计关系密切，在各类程序设计中应用十分广泛。

《Let's AutoIt》 （上）（AutoIt 编程基础篇）

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

数组是学习编程的第二道坎，之前我们已经越过了循环结构这第一道坎，而这第二道坎与第一道坎却息息相关。卡在第二道坎上的学习者十分多，很多人在这里变成了“学过一段时间编程的人”，所以还请大家一定要认真学习本章内容。

第一节、一维数组

所谓数组，即为一组变量的集合。其中每个变量称为一个“数组元素”，数组元素可以用相应的索引序号（下标）来访问。

一维数组的声明：

Dim 数组名[表达式]

说明：

（1）数组名的命名方式与变量的命名方式相同；

（2）数组名后是用方括弧（“[ ]”）括起来的表达式，不可用其他括弧替代；

（3）表达式表示数组中元素的个数，即数组长度；

（4）AutoIt 中表达式不仅允许是常量表达式，同样允许是变量表达式（这一点与有些只允许使用常量表达式的语言不同），但强烈建议在声明数组时仅使用常量表达式。

例如声明一个用来盛放学生成绩的数组：

```
Dim $Score[5]
```

数组“$Score[5]”包含 5 个变量，分别是“$Score[0]、$Score[1]、$Score[2]、$Score[3]、$Score[4]”。这里要注意，AutoIt 的数组与 C 语言中的数组十分相似，如果声明了“Dim 数组名[表达式]”，那么数组中的变量是从“数组名[0]”到“数组名[表达式-1]”。

声明数组后，我们就可以为这个数组赋值。要注意的是由于 AutoIt 的变量都是变体的，即变量没有固定数据类型，所以 AutoIt 数组中的每个元素也都是变体的。数组的赋值方式有两种：

（1）逐个赋值：

```
Dim $Score[5]
$Score[0]=90
$Score[1]=85
$Score[2]=71
$Score[3]=68
$Score[4]=97
```

这是一种最基本也是最常用的赋值方式，虽然有些麻烦，但是每个数组元素的值清晰明了，便于程序的阅读与修正。

（2）声明时赋值：

```
Dim $Score[5]=[90,85,71,68,97]
```

这种方法使用简便，可以一次将数组中的数据都赋值，这在数据较少时使用起来比逐个赋值更具有方便性。

特别的，在声明一个数组后如果不为其赋值，那么数组中的每个元素均为空字符。

数组中各元素的最大优势在于可以使用同一个名称而使用不同的索引序号来进行赋值、读取等操作，这就和我们之前所学习的循环结构有很大的关联。例如：

【代码】

```
Const $N=3
Dim $a[$N],$i,$s=""
For $i=0 To $N-1
    $a[$i]=$i
```

```
Next
For $i=0 To $N-1
    $s=$s&$a[$i]&"  "
Next
MsgBox(0,"范例",$s)
```

【简析】

首先，我们声明了一个常量$N 并为其赋值为 3，用以代表将要定义的数组的长度。

而后，我们声明了数组“$a[$N]”，并声明了变量“$i”和“$s”，并将“$s”赋值为空字符串。

然后我们使用

```
For $i=0 To $N-1
    $a[$i]=$i
Next
```

将数组“$a[$N]”中的每个元素赋值，等同于：

```
$a[0]=0
$a[1]=1
$a[2]=2
```

随后我们又使用了一次循环

```
For $i=0 To $N-1
    $s=$s&$a[$i]&"  "
Next
```

等同于：

```
$s=$s&$a[0] &"  "
$s=$s&$a[1] &"  "
$s=$s&$a[2] &"  "
```

将数组中的每个元素累加到变量“$s”中以便输出。

其实这两个循环可以写成一个：

```
For $i=0 To $N-1
    $a[$i]=$i
    $s=$s&$a[$i]&"  "
Next
```

只是为了向大家详细展示而刻意写成了两个。

最后我们使用我们已经熟悉的 MsgBox 将“$s”输出。输出结果为：

“0 1 2 ”

可见，数组其实是将一系列有一定关系的变量组成一个“组”，这些变量（又称数组元素）可以方便的依据序号来进行存储、读取工作。数组并不是一个很让人难以接受的事物，所以不需要对它有任何敬畏感。

另，在通常情况下，数组序号为 0 的元素一般是不使用的，原因有二：

（1）“第 0 个”容易引起混乱，例如声明了$a[3]，而要向其中存储 1、2、3，那么数组元素 0 存储 1，数组元素 1 存储 2，数组元素 2 存储 3，在读取时始终要计算该数组元素里存储的是比序号少 1 的数值，这在一定程度上是麻烦的。所以我们通常：

```
Dim $a[4],$i
For $i=1 To 3
    $a[$i]=$i
Next
```

这种做法就是将数组长度多定义一个，使用序号从“1“到“数组长度-1”的数组元素，而将序

号为 0 的元素空出。

（2）序号为 0 的元素经常用来存放一些特殊数据，例如我们定义一个存放学生姓名的数组：

Dim $StuName[5]=[4,"Skyfree","Miranda","Bell","Jiana"]

学生的姓名存放在序号为 1~4 的数组元素中，而序号为 0 的元素中存放的是学生数量。

类似的存放方法时长使用，序号为 0 的元素经常用来存放数组中实际元素的个数。有时候我们会定义一个较长的数组，但是并不会将其用尽，所以有一个用来存放实际数组元素数量的元素是十分重要的。

如上例可以接着写：

```
For $i=1 To $StuName[0]
    <各种关于$StuName 的操作>
Next
```

序号为 0 的元素存放数组元素实际数量这个应用在函数中也十分重要，因为我们的自定义函数通常不会知道参数中传过来的数组到底有多长、其中实际有多少元素。关于这一点将在下一章中详细介绍。

当然，这只是序号为 0 的元素使用的一个例子，序号为 0 的元素还可以存放其他特殊数据，这个就需要大家多多运用创造力了。

第二节、二维数组

第一节中我们已经学习了一维数组，一维数组中的每个元素都是基础变量，如数值型、字符型的变量，而如果一维数组中的每个变量又是一维数组，那么这个数组将会变的复杂起来，我们一般将这种数组称为二维数组。

理解二维数组也并不复杂。将数组变想象为空间，普通的一维数组可以认为是一条由一个个小方块组成的直线，每个小方块是一个基础变量：

□

□

□

□

□

二维数组则可以认为是将每个一维数组的小方块作为顶点向同一方向形成的射线，这条射线也可以认为是由一个个小方块组成的。

□■■■■■

□■■■■■

□■■■■■

□■■■■■

□■■■■■

这样二维数组就可以认为是一个平面，其中的每个小方块就是一个数组元素。由此，可以将二维数组认定为：一个每个元素都是一维数组的一维数组。

二维数组的声明：

Dim 数组名[表达式 1] [表达式 2]

二维数组中数组名、表达式的要求与一维数组相同。

例如定义一个存放学生姓名、语文成绩、数学成绩的二维数组：

Dim $NameScore [3][3]

二维数组赋值与一维数组相似：

（1）逐个赋值

```
Dim $NameScore[3][3]
$NameScore[0][0]="Skyfree"
$NameScore[0][1]=91
$NameScore[0][2]=83
$NameScore[1][0]="Tom"
$NameScore[1][1]=89
$NameScore[1][2]=97
$NameScore[2][0]="Jack"
$NameScore[2][1]=95
$NameScore[2][2]=92
```

声明了“$NameScore[3][3]”，其实相当于定义了三个一维数组，分别是数组 1：$NameScore[0][0]、$NameScore[0][1]、$NameScore[0][2]；数组 2：$NameScore[1][0]、$NameScore[1][1]、$NameScore[1][2]；数组 3：$NameScore[2][0]、$NameScore[2][1]、$NameScore[2][2]。那么针对它们每个元素赋值，就像针对这三个一维数组中的每个元素赋值一样。

（2）声明时赋值

```
Dim $NameScore[3][3]=[["Skyfree",91,83],["Tom",89,97],["Jack",95,92]]
```

这种赋值方式也与一维数组时类似，将这个二维数组理解为拥有三个一维数组的一维数组理解后，这个略显复杂的定义方式就明确了。

由二维数组概念衍生出“多维数组”，例如三维数组，三维数组就是二维数组的每个元素都为一维数组，转换为空间可以把这个想象为立方体。更多维的数组就要靠您更丰富的想象力了。AutoIt 最多支持六十四维数组。

在我们常用的程序设计中，一维与二维数组是最常用的，三维数组偶尔也会用到。所以一定要掌握最常用的一维与二维数组，对三维数组也应当了解。

第三节、数组应用实例

在我们平时书写关于数组的程序时，还有两点需要了解：

（1）关于数组的显示

某些时候我们想直接显示一个数组的全部，而不是通过中间变量去累加显示。这里我们可以使用一个 UDF（用户自定义函数）——“_ArrayDisplay”。

所谓 UDF 即不是 AutoIt 的自带函数，而是由许许多多 AutoIt 前辈向其中添加的函数。在使用“_ArrayDisplay”前必须在 AutoIt 代码的头部添加“#include <Array.au3>”，至于这个是做什么的、为什么要添加，这里暂时不做赘述，这个会在后续章节中详细介绍。

“_ArrayDisplay”函数语法：

```
_ArrayDisplay(一维或二维数组名,[ "标题名"])
```

注：这里并未介绍_ArrayDisplay 的全部函数参数，如有需要请详见 AutoIt 帮助文档。

例如我们要显示上节中学生成绩数组：

```
Dim $NameScore[3][3]=[["Skyfree",91,83],["Tom",89,97],["Jack",95,92]]
_ArrayDisplay($NameScore,"学生成绩")
```

运行这段程序后会出现一个列表以展示这个二维数组中的每一个元素。

（2）关于数组间的相互赋值

数组间是可以相互赋值的，我们可以将一个数组中的全部数值直接赋值给另一个数组，例如：

```
#include <Array.au3>
Dim $a[5]=[10,11,12,13,14]
Dim $b[5]
$b=$a
_ArrayDisplay($b)
```

这个例子中将“$a”数组中的所有值赋值给了“$b”数组，值得注意的是在数组赋值时只需要“$b=$a”，也就是“数组名 1=数组名 2”即可。

在数组与数组的相互赋值中，将会强制赋值号左侧的数组与赋值号右侧的数组完全相同，如：

```
#include <Array.au3>
Dim $a[5]=[10,11,12,13,14]
Dim $b[10]
$b=$a
_ArrayDisplay($b)
```

在“$b=$a”后，“$b”数组将与“$a”数组相同，也为一维的长度为 5 的数组，而不是在一开始定义时的长度为 8，这与其他编程语言不同。

即使出现：

```
#include <Array.au3>
Dim $a[5]=[10,11,12,13,14]
Dim $b[10][10]
$b=$a
_ArrayDisplay($b)
```

也会强制“$b”这个二维数组成为与“$a”数组相同的一维的长度为 5 的数组。甚至我们可以直接写成：

```
#include <Array.au3>
Dim $a[5]=[10,11,12,13,14]
Dim $b
$b=$a
_ArrayDisplay($b)
```

这种情况下也会强制“$b”称为与“$a”数组相同的一维的长度为 5 的数组。

那么，我们得出这样的结论：当赋值号右侧为数组时，无论赋值号左侧为何种数据类型的变量，均会强制与赋值号右侧数组完全一致。

这是 AutoIt 的灵活性，但不可否认在一定程度上可能会造成查错困难，所以请在数组赋值时格外小心。

下面我们来看一些关于数组的实例：

1、生成一个包含整数 1~10 的随机数组

【代码】

```
#include <Array.au3>
Const $N=11
Dim $i,$RandomNumArray[$N]
For $i=1 To $N-1
    $RandomNumArray[$i]=Random(1,10,1)
Next
```

```
_ArrayDisplay($RandomNumArray,"随机数组")
```

【简析】

在 AutoIt 中，生成随机数需要函数“Random”。

Random 函数语法：Random([最小值[,最大值[,标志]]])

参数含义：

（1）最小值，随机数的最小值，默认为 0；

（2）最大值，随机数的最大值，默认为 1；

（3）标志，设为 1 则返回整数，默认则返回一个浮点数（如 0.36、1.2）。

返回值：

成功：返回最小值与最大值之间的随机数

失败：返回 0，并把@Error 设置为 1（失败通常由参数错误引起）

那么我们来简单分析一下上述程序。

首先我们定义了一个常量“$N”并赋值为 11 以用来规定数组的长度，为了避免引起混乱，我这里避免使用序号为 0 的数组元素，只使用序号为 1 到 10 的数组元素。

然后我们声明了一个用来存放随机数的数组“$RandomNumArray”，并将长度定为“$N”。

随后进入循环，从 1 到 10 按照序号依次为数组中的每个元素赋一个随机整数值。

最后使用 UDF“_ArrayDisplay”将“$RandomNumArray”数组的全部值显示出来。

2、从一个包含整数 1~10 的随机数组中取出最小值与最大值

【代码】

```
#include <Array.au3>
Const $N=11
Dim $i,$RandomNumArray[$N],$Max,$Min
For $i=1 To $N-1
    $RandomNumArray[$i]=Random(1,10,1)
Next
_ArrayDisplay($RandomNumArray,"随机数组")
$Max=$RandomNumArray[1]
$Min=$RandomNumArray[1]
For $i=2 To $N-1
    If $RandomNumArray[$i]>$Max Then
        $Max=$RandomNumArray[$i]
    EndIf
    If $RandomNumArray[$i]<$Min Then
        $Min=$RandomNumArray[$i]
    EndIf
Next
MsgBox(0,"最大值","这个数组中的最大值为："&$Max _
    &@CRLF&"这个数组中的最小值为："&$Min)
```

【简析】

生成随机数数组的方法第 1 题中已经说明，这里不再赘述。

我们声明了变量“$Max”和“$Min”，首先假定数组的第一个值是数组的最大值且是数组的最小值（这种情况也的确存在，例如数组中的全部数值都相等时），然后进入循环，从数组元素的第二个值到最后一个值进行判定：

如果某数大于我们假定的最大值，那么就再假定最大值是这个“某数”，即“$Max=$RandomNumArray[$i]”；

如果某数小于我们假定的最小值，那么就再假定最小值是这个“某数”，即“$Mix=$RandomNumArray[$i]”。

当循环结束时，也就完成了最大值与最小值的选取。

3、生成一个包含整数 1~10 的随机数组，要求数组中的 10 个元素互不重复

【代码】

```
#include <Array.au3>
Const $N=11
Dim $RandomNumArray[$N]
Dim $Num,$i,$j,$Flag
$i=1
While $i<$N
    $Num=Random(1,10,1)
    $Flag=0
    $j=1
    While $j<$i
        If $Num=$RandomNumArray[$j] Then
            $Flag=1
            ExitLoop
        Else
            $j=$j+1
        EndIf
    WEnd
    If $Flag=0 Then
        $RandomNumArray[$i]=$Num
        $i=$i+1
    EndIf
WEnd
_ArrayDisplay($RandomNumArray,"随机不重复数组")
```

【简析】

生成数值不重复的随机数组，与我们之前的第一题不同。要生成不重复的随机数组，基本思路为：每生成一个数就要和之前已经生成的数组做比较，如果新生成的数存在于已生成的数组中，那么重新生成，否则就将这个数加入数组，准备生成下一个数。

变量声明部分不多说，与我们前几题中基本相同，只不过这次多声明了一个“$Num”变量和一个“$Flag”变量。

这次我们使用了两重“While…WEnd”循环，进入第一个“While…WEnd”循环前将“$i”的值赋为 1，第一个“While…WEnd”循环正常执行的条件是“$i<$N”。第一重循环开始处，我们为“$Num”赋一个随机整数值，随后将标志变量“$Flag”设置为 0，再将“$j”变量赋值为 1 后进入第二重“While…WEnd”循环。

第二重循环的正常执行条件是“$j<$i”，这个将在与已生成数组比较时起很大作用。第二重循环中，循环体判断新生成的“$Num”是否与已生成的数组中的值存在相同，只要存在一个相同值则将标志变量“$Flag”设置为 1 并直接跳出循环，否则将继续执行“$j=$j+1”。如果新生成的“$Num”在已生成数组中不存在相同，由于“$j”的累加最终会破坏正常执行循环“$j<$i”的条件，第二重循环也会因此而结束。

走出第二次循环后，无论是跳出、执行到最后破坏了循环条件结束或者根本初始“$j”就不是小于“$i”没执行第二重循环，最终标志“$Flag”只有 1 和 0 两种值。

“$Flag”为 1 时，说明新生成的“$Num”与已生成数组中的数值存在相同，不将“$Num”赋值给当前“$i”序号的数组元素，将重新循环并生成新的“$Num”再进行判定。

“$Flag”为 0 时，有两种情况：

第一种情况，为数组“$RandomNumArray”生成第一个元素的值时，此时由于“$i”和“$j”的值均为 1，所以“$j<$i”的条件不成立，也就是说第二重循环根本没执行，则“$Flag”的值肯定为 0，那么就将当前“$i”序号的数组元素也就是“$RandomNumArrayp[1]”赋值为“$Num”，并将“$i”自增 1 准备为下一个数组元素赋值。

第二种情况，经过第二重循环的判定后“$Flag”的值为 0，说明新生成的“$Num”与已生成数组中的数值不相同，那么就将当前“$i”序号的数组元素赋值为“$Num”，并将“$i”自增 1 准备为下一个数组元素赋值。

循环结束后，使用 UDF“_ArrayDisplay”将“$RandomNumArray”数组的全部值显示出来。

4、生成一个包含整数 1~10 的随机不重复数值数组，并未其按照从小到大的顺序排序

【代码】

```
#include <Array.au3>
Const $N=11
Dim $RandomNumArray[$N]
Dim $Num,$i,$j,$t,$Flag
$i=1
While $i<$N
    $Num=Random(1,10,1)
    $Flag=0
    $j=1
    While $j<$i
        If $Num=$RandomNumArray[$j] Then
            $Flag=1
            ExitLoop
        Else
            $j=$j+1
        EndIf
    WEnd
    If $Flag=0 Then
        $RandomNumArray[$i]=$Num
        $i=$i+1
    EndIf
WEnd
_ArrayDisplay($RandomNumArray,"随机不重复数组")
For $i=1 To $N-2
    For $j=$i+1 To $N-1
        If $RandomNumArray[$j]<$RandomNumArray[$i] Then
            $t=$RandomNumArray[$i]
            $RandomNumArray[$i]=$RandomNumArray[$j]
            $RandomNumArray[$j]=$t
        EndIf
    Next
Next
```

_ArrayDisplay($RandomNumArray," 排序后的随机不重复数组")

【简析】

排序问题一直是程序设计数据结构中的经典问题，如果今后有幸还能够为大家讲解 AutoIt 的数据结构，那么到时我会把这个问题单独作为一章来详细讲解。这里我们只讲解最简单的排序方法。

在之前的第四章第四节中，我们首次接触了三个数的排序，我们使用的方法是：首先让第一个数与第二个数比较并将第一个与第二个数按照从小到大到排列（通过交换法），然后比较第一个数与第三个数，最后比较第二个数与第三个数。

这也就为排序形成了一个最简单的规律：让第一个数与其后的每个数比较，通过交换让第一个数是从一到最后一个数中的最小，再让第二个数与其后的每个数比较，通过交换让第二个数是从二到最后一个数中的最小，依次类推，直到最后一个数，即完成了从小到大（升序）排序。

那么我们来看一下代码。随机不重复数组生成与第 3 题相同，不再赘述。

排序我们用了两个循环，第一个循环用来让第几个数与其后的其他数做比较，第二个循环用来让第一个循环规定的数与其后每个数做比较并根据判定做交换，这样就使第一个循环中规定的每个数都是从这个数到其后所有数中的最小。

两重循环完成后，升序排序完成。

5、将一个数据插入一个有序的数组中

【代码】

```
#include <Array.au3>
Const $N=12
Dim $i,$RandomNumArray[$N],$Num
For $i=1 To $N-2
    $RandomNumArray[$i]=Random(1,10,1)
Next
For $i=1 To $N-3
    For $j=$i+1 To $N-2
        If $RandomNumArray[$j]<$RandomNumArray[$i] Then
            $t=$RandomNumArray[$i]
            $RandomNumArray[$i]=$RandomNumArray[$j]
            $RandomNumArray[$j]=$t
        EndIf
    Next
Next
_ArrayDisplay($RandomNumArray,"随机有序数组")
$Num=Random(1,10,1)
MsgBox(0,"数组数据插入","将要插入数据"&$Num&"到刚才的数组中")
For $i=1 To $N-2
    If $i<>$N-2 Then
        If ($Num>=$RandomNumArray[$i] And $Num<$RandomNumArray[$i+1]) _
                Or ($i=1 And $Num<$RandomNumArray[$i+1]) Then
            For $j=$N-1 To $i+2 Step -1
                $RandomNumArray[$j]=$RandomNumArray[$j-1]
            Next
            $RandomNumArray[$i+1]=$Num
            ExitLoop
        EndIf
```

```
        Else
            $RandomNumArray[$i+1]=$Num
            ExitLoop
        EndIf
    Next
    _ArrayDisplay($RandomNumArray,"插入数据后的随机有序数组")
```

【简析】

有序数组的数据插入也是一个经典问题，虽然可以通过部分 UDF 函数直接实现，但是我仍旧建议大家详细了解数据插入的方法。这会使您麻烦几分钟，但会使您多学习一种方法。

数组生成与排序第 1 与第 4 题已经介绍，这里不再重复讲解。本题中使用的是随机数组，也就是说数组中的各元素可能会是重复的，这更贴近我们平时可能遇到的数组，毕竟谁也不能保证没有重复事件的发生。

我们从“$Num=Random(1,10,1)”这一句开始向下讲解代码。

为“$Num”赋一个 1~10 之间的随机值，用于插入到数组中。

“For…Next”循环开始，我们要逐一读取数组中的数据（或称“遍历”），并比较“$Num”与数组中的每个值。一般来说，数组中相邻的两个元素“$Num”只要大于等于其前者小于其后者则就可以将“$Num”插入到这里，而这里我们遇到两种特殊情况：

（1）“$Num”大于数组中所有的数

（2）“$Num”小于数组中所有的数

“$Num”大于数组中所有的数，即“$i”与“$N-2”相等时，所以判定后将“$Num”增补到数组最后即可。而“$Num”小于数组中所有的数的最前提条件是“$i”与“$N-2”不相等，这个与普通情况下的插入一致，所以我们只要在普通的插入条件后做追加判定即可。

一般情况下，即将“$Num”插入到两个相邻数组元素之间时，判定条件：

```
$Num>=$RandomNumArray[$i] And $Num<$RandomNumArray[$i+1]
```

增补一个在“$Num”小于数组中所有元素时的条件：

```
$i=1 And $Num<$RandomNumArray[$i+1]
```

这两种条件下，都是将数组中从最后一个元素到“$i+1”元素都像后“移动”，这也就是

```
For $j=$N-1 To $i+2 Step -1
    $RandomNumArray[$j]=$RandomNumArray[$j-1]
Next
```

这个循环的作用。

向后平移后，将“$Num”数值插入到之前“$i+1”的位置：

```
$RandomNumArray[$i+1]=$Num
```

数组数据插入完成。

6、将一个二维数组的数据沿对角线对换，如：

```
1  2  3
4  5  6
7  8  9
```

对换后：

```
1  4  7
2  5  8
3  6  9
```

【代码】

```
#include <Array.au3>
Dim $Array1[3][3]=[[1,2,3],[4,5,6],[7,8,9]]
```

```
Dim $Array2[3][3]
Dim $i,$j,$t
_ArrayDisplay($Array1)
For $i=0 To 2
    For $j=0 To 2
        $Array2[$i][$j]=$Array1[$j][$i]
    Next
Next
_ArrayDisplay($Array2)
```

【简析】

对换的关键在于：遍历整个二维数组，将序号为[x][y]的数组元素赋值给序号为[y][x]的数组元素。这个题目并不复杂，所以不多做解析，请大家自己理解。

数组是最基础的数据结构，也是在程序设计中极其常用的一种数据结构。数组与循环的关系密切，学好数组的关键在于学好循环，学好循环后只需要将数组理解为“一组可以通过序号访问的、序号不同而名字相同的变量”即可。

学习数组的关键在于勤于拆解代码与理解。

《Let’s AutoIt》——第七章、数组

完成于 2008-8-18 晨，2008-8-18 夜第二次修正

作者：Skyfree
QQ：165718402
E-Mail：Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS

第八章、函数

函数是程序中相对独立的功能模块，函数具有可移植性，可以用于不同的程序中。函数有利于加强程序的功能性与可读性，并加强程序代码的可移植性。

作者：Skyfree （隶属 自由天空技术论坛）

鸣谢：自由天空技术论坛（www.freeskycd.cn）
AutoIt 中文站（www.autoit.net.cn）

第一节、函数的一般形式

1、函数的定义

函数是一个独立的功能模块，我们在之前所使用如“MsgBox”、“InputBox”等就是函数，只不过我们之前所使用的函数大多数都是由系统预先定义的，称为内建函数，而今天我们要亲自定义一些函数。

函数的定义形式：

```
Func <函数名>([参数 1][,参数 2]…[,参数 n])
    [语句或语句组（函数体）]
    [Return  数据]
EndFunc
```

函数定义时必须以“Func”开头以“EndFunc”结尾。函数名的命名规则与变量命名规则相同。可以有参数，并可以设定多个参数，也可以无参数。函数体为语句或语句组，甚至可以无语句，无语句的函数称为空函数。“Return”语句用于设置函数的返回值，可以无“Return”语句，即函数无返回值。

2、函数的参数

首先说明，从这里开始，我们将使用“Local”彻底替代之前使用的“Dim”来定义变量，我们马上要接触到关于定义变量更深层的内容，所以暂时不要询问为什么要用“Local”来替代“Dim”，跟着我做就可以了。

我们以一个简单的例子来理解函数的参数。

```
Local $a,$b,$c
$a=5
$b=3
$c=Max($a,$b)
MsgBox(0,"函数范例","较大的数为："&$c)

Func Max($x,$y)
    Local $m
    If $x>=$y Then
        $m=$x
    Else
        $m=$y
    EndIf
    Return $m
EndFunc
```

这是一个我们在之前写过的，用于比较两个整数之间谁大谁小的程序，这里用函数来实现。

我们定义了函数“Max”，并为“Max”设定了两个参数“$x”和“$y”，函数定义时设定的函数参数我们称为形式参数。细看“Max”函数的代码，不难发现“Max”用来比较“$x”和“$y”哪个大，并把较大的值赋值给函数中定义的变量“$m”。最后使用“Return”语句将“$m”的值也就是“$x”和“$y”中的较大值返回。

回头看我们的主程序中，定义“$a”和“$b”并为这二者赋值后，语句“$c=Max($a,$b)”调用了函数“Max”，并为“Max”函数传递了两个参数“$a”和“$b”，调用函数时所使用的参数我们称为实际参数。“Max”函数在比较了两个传递过来的参数值后，将较大的值返回，在主程序中赋值给了“$c”。

最后使用 MsgBox 将“$c”的值显示出来，也就是将“$a”和“$b”中较大的值显示出来。

不难看出，一般情况下，函数的工作原理应当是这样的：

（1）定义一个函数，设置好函数名、形式参数等，并书写函数体；

（2）在主程序中调用函数，并在主程序中设定函数的实际参数值；

（3）实际参数值传递给函数的形式参数，函数对形式参数做一定的操作后返回一个值；

（4）主程序中调用函数的语句接收函数返回的值。

3、数组做函数参数

数组做函数参数与普通变量做函数参数是一样的，只是要注意将数组在主程序与函数间进行传递时只写数组名即可。

我们把上一章中的程序做成函数：生成一个不重复的由随机的 1~10 组成的数组，并为其排序。

```
#include <Array.au3>

Local $RandomArray
$RandomArray=CreatRandomArray()
_ArrayDisplay($RandomArray,"生成的随机数组")
$RandomArray=ArraySort($RandomArray)
_ArrayDisplay($RandomArray,"升序排序后的随机数组")

Func CreatRandomArray()
    Const $N=11
    Local $RandomNumArray[$N]
    Local $Num,$i,$j,$Flag
    $i=1
    While $i<$N
        $Num=Random(1,10,1)
        $Flag=0
        $j=1
        While $j<$i
            If $Num=$RandomNumArray[$j] Then
                $Flag=1
                ExitLoop
            Else
                $j=$j+1
            EndIf
        WEnd
        If $Flag=0 Then
            $RandomNumArray[$i]=$Num
            $i=$i+1
        EndIf
```

```
    WEnd
    $RandomNumArray[0]=10
    Return $RandomNumArray
EndFunc

Func ArraySort($Array)
    Local $i,$j
    For $i=1 To $Array[0]-1
        For $j=$i+1 To $Array[0]
            If $Array[$j]<$Array[$i] Then
                $t=$Array[$i]
                $Array[$i]=$Array[$j]
                $Array[$j]=$t
            EndIf
        Next
    Next
    $Array[0]=""
    Return $Array
EndFunc
```

我们定义了函数“CreatRandomArray”和函数“ArraySort”。

函数“CreatRandomArray” 用于生成一个不重复的由随机的 1~10 组成的数组，它没有参数，因为它不需要为其传递什么值以用来规定或运算什么，“CreatRandomArray”这种函数通常被叫做无参函数，一般被用作执行固定的工作。值得注意的是在“CreatRandomArray”函数的最后，有一行语句“$RandomNumArray[0]=10”，这句话是将数组中实际的元素数量存储在序号为 0 的元素中，以备之后该数组被调用时使用。

函数“ArraySort”用于将数组按照升序进行排序。“ArraySort”函数有一个参数“$Array”，这个形式参数用来接收由实际参数传来的数组。由于我们使用“CreatRandomArray”函数创建的数组的 0 元素中存储的是该数组中元素的实际数量，所以在“ArraySort”函数中对“$Array”进行排序时，两个循环分别为“For $i=1 To $Array[0]-1”和“For $j=$i+1 To $Array[0]”，这和我们在上一章中所使用的“$N”不同。

有了这两个函数我们的主程序变的出奇的简单：

```
$RandomArray=CreatRandomArray()
_ArrayDisplay($RandomArray,"生成的随机数组")
$RandomArray=ArraySort($RandomArray)
_ArrayDisplay($RandomArray,"升序排序后的随机数组")
```

创建数组，展示创建的数组。排序数组，展示排序后的数组。

使用函数后主程序可读性大大增强，而且使用函数最大的好处在于如果我们今后要使用类似的功能，例如我们可能会为一个包含 20 个数据的数组进行排序，我们可以直接将今天写好的代码复制过去，写一下函数的实际参数，然后仍旧是调用一下函数即可。

4、参数的“值传递”与“址传递”

我们来看一个简单的例子：

```
Local $a=1,$b=2
```

```
Exchange($a,$b)
MsgBox(0,"参数传递范例","$a="&$a&"，"&"$b="&$b)

Func Exchange($x,$y)
    Local $t
    $t=$x
    $x=$y
    $y=$t
EndFunc
```

我们定义了函数“Exchange”并为其设定了两个参数“$x”和“$y”。函数“Exchange”的作用一目了然——用于交换“$x”和“$y”的值。主函数中定义了“$a”和“$b”，并调用函数“Exchange”试图交换“$a”和“$b”的值。

不过如果试运行一下这个程序立刻就会知道，“$a”和“$b”的值并未实现交换，为什么？

这是因为一般情况下，实际参数（实参）与形式参数（形参）之间的数据传递为“值传递”，这就是说实参仅仅是把它的值传递给了形参，这也就意味着在上面的例子中“$a”和“$b”仅仅是把自己的数值传递给了函数“Exchange”中的“$x”和“$y”。这样以来，无论“$x”和“$y”如何变化，都与“$a”和“$b”无关。

那么有没有方法让实参与形参紧密的联系起来，形参改变实参也会改变呢？答案就是“址传递”。

我们在定义变量时，系统为变量分配一定的内存空间，而为了方便对这块内存空间进行读写操作，我们为这段内存空间取了一个别名，这个别名就是我们的变量名。那么何谓“址传递”，址传递就是将一个变量所代表内存空间的地址传给另一个变量。这样两个变量虽然名字不同，而实际上却操作的是同一块内存空间。简单说，就像你的朋友有个外号，你叫他的名字或者叫他的外号他都会答应一样。变量的“址传递”就是对一个已存在的变量取个外号。

懂得了什么叫做“址传递”，我们将上述程序做一定的修改：

```
Local $a=1,$b=2
Exchange($a,$b)
MsgBox(0,"参数传递范例","$a="&$a&"，"&"$b="&$b)

Func Exchange(ByRef $x,ByRef $y)
    Local $t
    $t=$x
    $x=$y
    $y=$t
EndFunc
```

乍一看似乎没什么变化，最重要的变化在于“$x”和“$y”变量前的“ByRef”，有了这个就证明实参在与形参传递时，实参传给形参的是实参变量的地址，而不是值。

在这个程序中，调用“Exchange($a,$b)”，将“$a”和“$b”的地址分别传给“$x”和“$y”，这样所有针对“$x”和“$y”的操作就与针对“$a”和“$b”操作无差别。

试运行一下上面的程序，“$a”和“$b”的值完美交换了。

函数的址传递的主要作用在于可以让一个函数返回一个以上的值，不过使用址传递时一定要小心，滥用址传递会造成难以查证的混乱。

建议：如果确实无特殊需要，不要使用址传递。

第二节、变量的作用域

在本书之初，我们在定义变量时只使用“Dim”，这是由于当时我们并没有函数的概念，所以不能强行的就开始学习所谓局部变量与全局变量，强行学习暂时无用的知识会对正在学习的知识造成干扰。

而在本章第一节中，我们就开始彻底抛弃“Dim”的使用，因为我们要开始规定变量的作用于了。

1、局部变量

所谓局部变量，即只在“本地”有效，例如函数中的各个以“Local”关键字命名的变量，只在这个函数中有效，而在这个函数外即为无效。

我们回头看一下第一节中的例子：

```
Local $a,$b,$c
$a=5
$b=3
$c=Max($a,$b)
MsgBox(0,"函数范例","较大的数为："&$c)

Func Max($x,$y)
    Local $m
    If $x>=$y Then
        $m=$x
    Else
        $m=$y
    EndIf
    Return $m
EndFunc
```

“$x”、“$y”、“$m”这些在函数“Max”中定义的变量则只在函数中有效，主程序中不能直接调用它们。

这种使用“Local”关键字定义的，只在“本地”有效的变量称为局部变量。

2、全局变量

某些时候我们需要一些变量在整个程序中都有效，这时候我们需要在主程序中使用“Global”关键字来定义变量。如：

```
Global $c
Sum()
MsgBox(0,"全局变量范例","$c="&$c)

Func Sum()
```

```
    Local $a=1,$b=2
    $c=$a+$b
EndFunc
```

这是个十分简单的例子，首先在主程序中定义了全局变量“$c”，而后定义了无参数的“Sum”函数。“Sum”函数中定义了$a”、“$b”两个局部变量，并将“$a+$b”的值赋给全局变量“$c”。由于“$c”是全局变量，在整个程序的任何部分它都有效且都可以对它进行各种更改，所以在函数“Sum”中对“$c”的更改也会生效。所以主程序运行的结果是“$c=3”

不推荐过多的使用全局变量，虽然全局变量使用起来比较方便，但是这会大大降低函数的通用性。并且如果在多处存在语句修改全局变量，势必会引起一定的混乱与难以查证的错误。

3、变量的作用范围

变量有其作用范围，在特殊情况下变量还会失去在某范围内的作用。下面我们来详细探讨一下变量的作用范围问题：

（1）不同范围内的同名局部变量

```
Local $a=2,$b=1
Local $c
$c=Addition($a,$b)
MsgBox(0,"变量作用域范例","$a 与$b 的和为："&$c)
$c=Subtraction($a,$b)
MsgBox(0,"变量作用域范例","$a 与$b 的差为："&$c)

Func Addition($x,$y)
    Local $c
    $c=$x+$y
    Return $c
EndFunc

Func Subtraction($x,$y)
    Local $c
    $c=$x-$y
    Return $c
EndFunc
```

在这个例子中，我们制作了两个简单的函数，分别用来做加法和减法运算。

两个函数中有相同的变量“$x”、“$y”和“$c”，这会不会引起问题，会不会造成变量重复定义的问题呢？答案是不会，因为这几个变量只在不同的区域内有效。

两个函数中的局部变量“$c”分别只在两个函数中有效。就像一班里有个叫 Jack 的同学，二班里也有一个叫 Jack 的同学，而如果在一班里叫“Jack”则只有一班的 Jack 会答应你，这是因为在一班这个范围内默认的 Jack 只有一个，一班的 Jack 在一班内是唯一的，不会引起混淆的。

局部变量也是一样，每个局部变量只在自己的作用域里有效，而不会迈出自己的作用范围生效。

（2）全局变量与局部变量重名

```
Global $c
$c=0
Sum()
MsgBox(0,"全局变量范例","$c="&$c)

Func Sum()
    Local $a=1,$b=2
    Local $c
    $c=$a+$b
EndFunc
```

我们将本节初的例子修改一下，这种情况下定义了全局变量“$c”并为其赋初值 0；而又在“Sum”函数中定义了局部变量“$c”，局部变量“$c”为已赋值的$a”与“$b”的和。运行这段代码后“$c”的值仍为 1，而不为 3.

这说明：在局部范围内，如果有全局变量与这个局部范围内的局部变量重名，则全局变量被屏蔽，只使用这个局部范围内的局部变量。简单说，强龙不压地头蛇（……）。

建议，尽量不要使局部变量与全局变量重名。

另，说两点其他的：

（1）其实 Local 与 Global 在被“标准翻译”后变的难以理解了。

Local：地方的，当地的，本地的，某一地方所特有的。这样很容易就可以理解成：以“Local”定义的变量只在“本地”也就是它的作用范围内有效。

Global：全世界的，总体的。很明显，使用“Global”定义的变量在整个“世界”里（这个程序的源码组成的世界里）都是有效的。

实在不愿意记住“局部变量”和“全局变量”，就记住一个是“本地的”一个是“世界的”。

（2）如果在函数中使用“Dim”声明变量，则这时的“Dim”与“Local”等同。而如果在主程序中使用“Dim”则与“Global”等同。值得注意的是，在 AutoIt 主程序中“Local”与“Global”的意义是相同的——至少在本书所用的版本中是相同的，不过仍推荐只在主程序中使用的变量以“Local”定义，而全局变量使用“Global”定义，以容易区分变量的作用。

第三节、函数的嵌套与递归

1、函数的嵌套

函数的嵌套很好理解，即在一个函数中调用另一个函数，这在程序设计中十分常用，下面我们看一个简单的例子：

```
Local $a=6,$b=2,$c=7
NumSort($a,$b,$c)
MsgBox(0,"三个数排序","从小到大依次为："&$a&"，"&$b&"，"&$c)

Func NumSort(ByRef $a,ByRef $b,ByRef $c)
    Select
```

```
            Case $a>$b
                Exchange($a,$b)
            Case $a>$c
                Exchange($a,$c)
            Case $b>$c
                Exchange($b,$c)
        EndSelect
    EndFunc

    Func Exchange(ByRef $x,ByRef $y)
        Local $t
        $t=$x
        $x=$y
        $y=$t
    EndFunc
```

这是我们在第四章中的一个将三个整数按升序排序的例子。这里我们定义了一个名为“NumSort”的函数用以将三个参数所规定的数进行升序排序。而在“NumSort”中为了方便交换数据，又调用了“Exchange”函数。

这样，在“NumSort”函数中调用“Exchange”函数，即是函数的嵌套。

特别要注意的是，我所使用的这两个函数均为址传递，正好可以加深一下大家对址传递的了解与应用。通常址传递被用于函数不只需要返回一个值时。

2、函数的递归

函数的递归与嵌套不同，嵌套是在函数中调用其他函数，而递归则很特别，递归是在自身的函数中调用自身。函数的递归在一般的程序设计中并不多使用，但是递归有时可以将程序设计的思路简化，方便创建算法。举个例子：

有五个人坐在一起，第五个人说他比第四个人大两岁，第四个人说他比第三个人大两岁，第三个人说他比第二个人大两岁，第二个人说他比第一个人大两岁，第一个人说他十岁，问第五个人几岁？

```
    MsgBox(0,"函数递归范例",Age(5))

    Func Age($n)
        Local $c
        If $n=1 Then
            $c=10
        Else
            $c=Age($n-1)+2
        EndIf
        Return $c
    EndFunc
```

我们定义了函数“Age”，“$n”代表这是第几个人，$c 作为存放函数返回值的变量。

函数“Age”在执行时，如果“$n”为 1，也就是说第一个人时，函数返回 10，否则就再次调用

自身先去获取下人个人的年龄。函数执行过程：

Age(5)→“$n=5，$c=Age(4)+2”→“$n=4，$c=Age(3)+2”→“$n=3，$c=Age(2)+2”→“$n=2，$c=Age(1)+2”→“$n=1，$c=10”→“Age(1)=10”→“Age(2)=12”→“Age(3)=14”→“Age(4)=16”→“Age(5)=18”

这就像一个人一个人的问下去，根据最后一个人回答的年龄，逆推每个人的年龄。

当然，在这个简单的例子里还看不出递归的优势所在，如果今后有幸可以为大家讲解 AutoIt 的数据结构，我们将详细的研究更为复杂实用的递归。

在这里，您仅需要知道递归是怎么个概念即可，并尽量理解这个例子。

函数的使用在程序设计时几乎是必须的，函数的功能独立性和便于移植性是它的最大特点。请大家一定掌握函数的用法，正确的使用函数可以帮大家节省大量的时间、提高编程效率。

到这里，《Let’s AutoIt》的第一部分（AutoIt 编程基础）共八章内容就全部写完了，希望大家喜欢。

可能看完这八章内容您仍旧写不出很实用的 AutoIt 程序，但是不要着急，如果您能完整的理解这八章中所有的内容，那么您的基础已经相当扎实了。有了扎实的基础，更深层的学习就仅仅是时间与经验的问题了！

《Let’s AutoIt》的第二部分（AutoIt 进阶）可能会在 10 月份才会动笔，所以您完全有时间将第一部分的八章基础内容完全掌握。

感谢您的观看！谢谢您的支持！

《Let’s AutoIt》——第八章、函数

完成于 2008-8-18 午，2008-8-18 夜第一次修正

作者：Skyfree
QQ：165718402
E-Mail：Skyfree@FreeSkyCD.Cn

自由天空技术论坛：
Www.FreeSkyCD.Cn\BBS