批处理之家论坛（BBS.BATHOME.NET）

批处理 For 语句
从入门到精通

兼谈变量延迟

作者：namejm

初稿：2008-10-26
定稿：2010-12-25

本文作者 namejm，长年混迹于中国 DOS 联盟论坛（bbs.cn-dos.net），并曾长期担任批处理之家论坛（bbs.bathome.net）站长，以其多年的批处理代码编写经验，以“但求最好”的心态，两年磨一剑，以通俗易懂、风趣生动的语言，由浅入深地介绍了批处理中最为强大的 for 语句，有别于网上粗制滥造的教程，是广大批处理爱好者不可多得的教材。

目录

批处理 For 语句从入门到精通

首发地址：http://bbs.bathome.net/thread-2189-1-1.html

一、前言

在批处理中，for 是最为强大的命令语句，它的出现，使得解析文本内容、遍历文件路径、数值递增/递减等操作成为可能；配合 if、call、goto 等流程控制语句，更是可以实现脚本复杂的自动化、智能化操作；合理使用 for 语句，还能使代码大为简化，免除各位编写大量重复语句之苦。而能否熟练使用 for 语句，已经成为衡量一个人批处理水平高低最主要的标准。

在这个系列教程中，我将通过实际应用中频繁出现的例子，带领大家步入 for 语句的神奇之门，一步步迈向 for 语句的魔幻殿堂，使得大家在实际的应用中，能独立写出简洁高效的代码，在批处理的世界里自由驰骋。

注意：以下的讲解，都是基于简体中文 Windows XP Pro SP3 的操作系统环境

二、for 语句的基本用法

正如色彩缤纷的七彩光芒是由红绿蓝三原色构成的一样，最复杂的 for 语句，也有其基本形态，它的模样是这样的：

在 cmd 窗口中：for %I in (command1) do command2
在批处理文件中：for %%I in (command1) do command2

之所以要区分 cmd 窗口和批处理文件两种环境，是因为在这两种环境下，命令语句表现出来的行为虽然基本一样，但是在细节上还是稍有不同，最明显的一个差异就是：在 cmd 窗口中，for 之后的形式变量 I 必须使用单百分号引用，即%I；而在批处理文件中，引用形式变量 I 必须使用双百分号，即%%I。为了方便起见，若不是特别强调，以下的讲解都以批处理文件环境为例。

我们先来看一下 for 语句的基本要素都有些什么：

1、for、in 和 do 是 for 语句的关键字，它们三个缺一不可；
2、%%I 是 for 语句中对形式变量的引用，就算它在 do 后的语句中没有参与语句的执行，也是必须出现的；
3、in 之后，do 之前的括号不能省略；
4、command1 表示字符串或变量，command2 表示字符串、变量或命令语句；

现在，你可能已经会写一个简单的 for 语句了，比如：

[code1]

```
@echo off
for %%I in (bbs.bathome.cn) do echo %%I
pause
```

保存为批处理文件并执行，将会在弹出的批处理窗口中看到这样的信息：

[result1]

```
bbs.bathome.cn
请按任意键继续...
```

很快地，你会觉得这个 for 语句是如此的简单，简单到你丝毫感受不出它的强大：这个 for 语句，和我直接用 echo 语句没什么两样啊！

是的，演示代码永远都只是演示而已，就像大多数高级语言的教科书一样，

在引导新手学习的时候，基本上都是千篇一律地告诉大家如何编写一个能显示 hello world! 的窗口，从这些演示代码中，你看不到它们具有多少实用性，你只是感到有点好奇：咦，居然弹出了一个窗口？片刻之后，你就会觉得索然无味。

那好吧，为了让大家对 for 更加感兴趣，我们先来分析一下 for 语句的一些注意事项，之后，再让大家看看更为强大的 for 语句实例。

1、for 语句的形式变量 I，可以换成 26 个字母中的任意一个，这些字母会区分大小写，也就是说，%%I 和%%i 会被认为不是同一个变量；形式变量 I 还可以换成其他的字符，但是，为了不与批处理中的%0～%9 这 10 个形式变量发生冲突，请不要随意把%%I 替换为%%0 ～%%9 中的任意一个；

2、in 和 do 之间的 command1 表示的字符串或变量可以是一个，也可以是多个，每一个字符串或变量，我们称之为一个元素，每个元素之间，用空格键、跳格键、逗号、分号或等号分隔；

3、for 语句依次提取 command1 中的每一个元素，把它的值赋予形式变量 I,带到 do 后的 command2 中参与命令的执行；并且每次只提取一个元素，然后执行一次 do 后的命令语句，而无论这个元素是否被带到 command2 中参与了 command2 的运行；当执行完一次 do 后的语句之后，再提取 command1 中的下一个元素，再执行一次 command2，如此循环，直到 command1 中的所有元素都已经被提取完毕，该 for 语句才宣告执行结束；

其中，第 3 点是最为关键的，它描述了 for 语句的执行过程，是 for 语句的精髓所在，大家一定要牢记这一条，才能深刻理解更为复杂的 for 流程。

有了以上的基础，我们再来看一个例子，这个例子修改了 code1 的部分内容，结果将大不一样：

[code2]

```
@echo off
for %%I in (bbs,bathome,cn) do echo %%I
pause
```

和 code1 的执行结果相比，code2 的执行结果发生了如下变化：

1、显示结果分成了 3 行（不算最后一行中文提示）；

2、每一行都从逗号处被切分；

如果把 bbs.bathome.cn 这个字符串中的点号换为空格、跳格或等号，执行结果将和 code2 的执行结果别无二致。

现在，我们来分析一下 code2 代码中 for 语句的执行过程：

首先，for 语句以逗号为分隔符，把 bbs,bathome,cn 这个字符串切分成三个元素：bbs、bathome 和 cn，由此决定了 do 后的语句将会被执行 3 次；

然后，第一次执行过程是这样的：先把 bbs 这个字符串作为形式变量 I 的

值，带入 do 后的语句中加以执行，也就是执行 echo %%I 语句，此时的 I 值为 bbs，因此，第一次执行的结果，将会在屏幕上显示 bbs 这个字符串；第二次执行和第一次执行的过程是一样的，只不过此时 I 的值已经被替换为 command1 中的第二个元素了，也就是 bathome 这个字符串；如此循环，当第三次 echo 执行完毕之后，整条 for 语句才算执行完毕，此时，将执行下一条语句，也就是 pause 命令。

其实，这个例子只比上一个例子多了一点花样，有趣了那么一点点：一条 for 语句的执行结果居然被分成了 3 行！

为了让大家见识一下 for 的真正威力，本人绞尽脑汁，翻帖无数，不得要领，万般无奈之下，只好亮出了尘封在箱底多年的一段代码：检测当前硬盘都有哪些分区^_^

[code3]

```
@echo off
set str=c d e f g h i j k l m n o p q r s t u v w x y z
echo 当前硬盘的分区有：
for %%i in (%str%) do if exist %%i: echo %%i:
pause
```

这段代码能检测硬盘都有哪些分区，包括 U 盘和移动硬盘的分区，但是，当光驱中有盘的时候，也会被列出来，这是本代码的一个缺憾，在以后的讲解中，我将向大家讲述如何消除这个瑕疵，敬请关注本系列的后续章节。

高级应用：

想知道当前目录下都有哪些文件吗？请用下面的代码：

```
@echo off
for %%i in (*.*) do echo "%%i"
pause
```

想列出当前目录下所有的文本文件吗？请用下面的代码：

```
@echo off
for %%i in (*.txt) do echo "%%i"
pause
```

想列出只用两个字符作为文件名的文本文件吗？请用下面的代码：

```
@echo off
for %%i in (??.txt) do echo "%%i"
pause
```

题外话：

1、列出当前目录下各种文件的方法，最简单的还是用 dir 命令，但是，从以上代码中，各位可以加深对 for 语句执行流程的理解（用到了通配符*和?）；
2、注意：以上代码不能列出含有隐藏或系统属性的文件；

练习：用 for 语句建立 test1.txt、test2.txt 和 test3.txt 三个文本文件。

更全面的练习请看这个帖子：for 语句从入门到精通配套练习题 http://bbs.bathome.cn/thread-2336-1-1.html

三、文本解析显神威：for /f 用法详解

前言

for /f 是个十分强大的家伙。

如果说，for 语句是批处理中最强大的语句的话，那么，for /f 就是精华中的精华。

for /f 的强大，和它拥有众多的开关密切相关。因为开关众多，所以用法复杂，本章将分成若干小节，为大家逐一介绍强大的 for /f 语句。

（一） 为解析文本而生：for /f 的基本用法

所有的对象，无论是文件、窗体、还是控件，在所有的非机器语言看来，无外乎都是形如"c:\test.txt"、"CWnd"之类的文本信息；而所有的对象，具体的如 ini 文件中的某条配置信息、注册表中的某个键值、数据库中的某条记录……都只有转化为具有一定格式的文本信息，方可被代码识别、操控。可以说，编程的很大一部分工作，都是在绞尽脑汁想方设法如何提取这些文本信息。

而提取文本信息，则是 for /f 的拿手好戏：读取文件内容；提取某几行字符；截取某个字符片段；对提取到的内容再切分、打乱、杂糅……只要你所能想到的花样，for /f 都会想方设法帮你办到，因为，for /f 就是被设计成专门用于解析文本的。

先来看个例子。

假如有个文本文件 test.txt，内容如下：

[txt1]
论坛的目标是：不求最大，但求最好，做最实用的批处理论坛。
论坛地址：bbs.bathome.cn。
这里是：新手晋级的福地，高手论剑的天堂。

那么，将如下代码保存为 test.cmd，并放在 test.txt 同一目录下运行，将会在屏幕上原样显示 test.txt 的内容：

[code4]

```
@echo off
for /f %%i in (test.txt) do echo %%i
pause
```

这段代码，主要是让你树立这样一种观念：读取文本文件的内容，请使用 for /f 语句！

进阶话题：for /f 语句是把整个 test.txt 一次性显示出来的？

在这段代码中，虽然执行结果是把 test.txt 中的所有内容都显示出来了，貌似 for /f 语句是把整个 test.txt 一次性显示到屏幕上，实际上并非如此。

无论 for 语句做何种变化，它的执行过程仍然遵循基本的 for 流程：依次处理每个元素，直到所有的元素都被处理为止。只不过在 for /f 语句中，这里的元素是指文件中的每一行，也就是说，for /f 语句是以行为单位处理文本文件的。这是一条极为重要的规则，在上一章中也强调过它的重要性，希望在接下来的学习过程中，你能时刻牢记这一原则，那么，很多问题将会迎刃而解。以下是验证这一说法的演示代码（在[code4]的基础上添加了&pause 语句）：

[code5]

```
@echo off
for /f %%i in (test.txt) do echo %%i&pause
pause
```

（二） 切分字符串的利器：delims=

也许你对[code4]这段代码不屑一顾：不就是把 test.txt 的内容显示出来了么？好像用处不大啊。

好吧，我们来玩个魔术。

还是[txt1]这段文本，把[code4]改造一下：

[code6]

```
@echo off
```

```
for /f "delims=，" %%i in (test.txt) do echo %%i
pause
```

再次运行 test.cmd，看到什么变化了吗？！

```
[result2]
论坛的目标是：不求最大
论坛地址：bbs.bathome.cn。
这里是：新手晋级的福地
请按任意键继续...
```

结果，你惊奇地发现，每行第一个逗号之后的所有内容都不见了（如果有不存在逗号的行，则保留原样），也就说，你成功地提取到了每行第一个逗号之前的所有内容！

试想一下，这段代码会有什么用呢？

如果别人给了你一个软件清单，每行都是"英文软件名（逗号）中文软件名"的格式，而你却只想保留英文名的时候，这段代码将是多么有用啊！再假设，有这么一个 IP 文件，第一列是数字格式的 IP 地址，第二列是具体的空间地址，列与列之间用逗号分隔，而你想提取其中数字格式的 IP，呵呵，我不说你也知道该怎么办了吧？

要是文本内容不是以逗号分隔，而是以其他符号分隔，那么，把"delims=,"的逗号换成相应的符号就可以了。

在这里，我们引入了一个新的开关："delims=，"，它的含义是：以逗号作为被处理的字符串的分隔符号。

在批处理中，指定分隔符号的方法是：添加一个形如 "delims=符号列表" 的开关，这样，被处理的每行字符串都会被符号列表中罗列出来的符号切分开来。

需要注意的是：如果没有指定"delims=符号列表"这个开关，那么，for /f 语句默认以空格键或跳格键作为分隔符号。请把[txt1]中不同位置上的标点符号改为空格或跳格，再运行[code4]试试。

进阶话题：如果我要指定的符号不止一个，该怎么办？

在上面的讲解中，我提到了指定分隔符号的方法：添加一个形如"delims=符号列表"的开关。不知道你注意到没有，我的说法是"符号列表"而非"符号"，这是大有讲究的，因为，你可以一次性指定多个分隔符号！

还是以[txt1]为例，把[code6]再改造一下：

[code7]

```
@echo off
for /f "delims=. ，" %%i in (test.txt) do echo %%i
pause
```

结果显示：

[result3]

```
论坛的目标是：不求最大
论坛地址：bbs
这里是：新手晋级的福地
请按任意键继续...
```

这样，第一个点号或第一个逗号之前的内容都被提取出来了。

[code7]的执行过程是：逐行读取 test.txt 中的内容，以点号和逗号切分每一行的内容（不存在点号和逗号的行，则不再切分，为了描述的方便，我们把被点号或逗号切分的一个一个的字符串片段，称之为节），然后，for /f 会提取第一节的内容作为最终结果，显示在屏幕上。需要注意的是，在这里，所有行的字符串被切分成了两个以上的节，但是，[code7]的代码只会提取第一节字符串的内容，因为 for /f 语句默认只提取第一节的符串。

（三） 定点提取：tokens=

上一节在讲解 delims= 的时候，我一再强调 for /f 默认只能提取到第一节的内容，现在我们来思考一个问题：如果我要提取的内容不在第一节上，那怎么办？

这回，就该轮到 tokens= 出马了。

tokens= 后面一般跟的是数字，如 tokens=2，也可以跟多个，但是每个数字之间用逗号分隔，如 tokens=3,5,8，它们的含义分别是：提取第 2 节字符串、提取第 3、第 5 和第 8 节字符串。注意，这里所说的“节”，是由 delims= 这一开关划分的，它的内容并不是一成不变的。

下面来看一个例子：

[txt2]

尺有所短，寸有所长，学好批处理没商量，考虑问题复杂化，解决问题简洁化。

对[txt2]这段文本，假设它们保存在文件 test.txt 中，如果我想提取“学好批处理没商量”这句话，该如何写代码呢？

我们稍微观察一下[txt2]就会发现，如果以逗号作为切分符号，就正好可以把“学好批处理没商量”化为单独的一“节”，结合上一节的讲解，我们知道，"delims=，" 这个开关是不可缺少的，而要提取的内容在以逗号切分的第 3 节上，那么，tokens= 后面的数字就应该是 3 了，最终的代码如下：

[code8]

```
@echo off
for /f "delims= ，  tokens=3" %%i in (test.txt) do echo %%i
pause
```

如果我们现在要提取的不只一个“节”，而是多个，那又怎么办呢？比如，要提取以逗号切分的第 2 节和第 5 节字符串，是写成这样吗？

[code9]

```
@echo off
for /f "delims= ，  tokens=2,5" %%i in (test.txt) do echo %%i
pause
```

运行批处理后发现，执行结果只显示了第 2 节的内容。

原来，echo 后面的 %%i 只接收到了 tokens=2,5 中第一个数值 2 所代表的那个字符串，而第二个数值 5 所代表的字符串因为没有变量来接收，所以就无法在执行结果中显示出来了。

那么，要如何接收 tokens= 后面多个数值所指代的内容呢？

for /f 语句对这种情况做如下规定：

如果 tokens= 后面指定了多个数字，如果形式变量为%%i，那么，第一个数字指代的内容用第一个形式变量%%i 来接收，第二个数字指代的内容用第二个形式变量%%j 来接收，第三个数字指代的内容用第三个形式变量%%k 来接收……第 N 个数字指代的内容用第 N 个形式变量来接收，其中，形式变量遵循字母的排序，第 N 个形式变量具体是什么符号，由第一个形式变量来决定：如果第一个形式变量是%%i，那么，第二个形式变量就是%%j；如果第一个形式变量用的是%%x，那么，第二个形式变量就是%%y。

现在回头去看[code9]，你应该知道如何修改才能满足题目的要求了吧？修改

结果如下：

[code10]
```
@echo off
for /f "delims= ， tokens=2,5" %%i in (test.txt) do echo %%i %%j
pause
```

如果有这样一个要求：显示[txt2]中的内容，但是逗号要替换成空格，如何编写代码？

结合上面所学的内容，稍加思索，你可能很快就得出了答案：

[code11]
```
@echo off
for /f "delims= , tokens=1,2,3,4,5" %%i in (test.txt) do echo %%i %%j %%k %%l %%m
pause
```

写完之后，你可能意识到这样一个问题：假如要提取的“节”数不是 5，而是 10，或者 20，或者更多，难道我也得从 1 写到 10、20 或者更多吗？有没有更简洁的写法呢？

答案是有的，那就是：如果要提取的内容是连续的多“节”的话，那么，连续的数字可以只写最小值和最大值，中间用短横连接起来即可，比如 tokens=1,2,3,4,5 可以简写为 tokens=1-5 。

还可以把这个表达式写得更复杂一点：tokens=1,2-5，tokens=1-3,4,5，tokens=1-4,5……怎么方便就怎么写吧。

大家可能还看到一种比较怪异的写法：

[code12]
```
@echo off
for /f "delims= ， tokens=1,*" %%i in (test.txt) do echo %%i %%j
pause
```

结果，第一个逗号不见了，取代它的是一个空格符号，其余部分保持不变。

其中奥妙就在这个星号上面。

tokens=后面所接的星号具备这样的功能：字符串从左往右被切分成紧跟在*之前的数值所表示的节数之后，字符串的其余部分保持不变，整体被*所表示的一个变量接收。

理论讲解是比较枯燥的，特别是为了严密起见，还使用了很多限定性的修饰词，导致句子很长，增加了理解的难度，我们还是结合[code12]来讲解一下吧。

[txt2] 的内容被切分，切分符号为逗号，当切分完第一节之后，切分动作不再继续下去，因为 tokens=1,* 中，星号前面紧跟的是数字 1；第一节字符串被切分完之后，其余部分字符串不做任何切分，整体作为第二节字符串，这样，[txt2] 就被切分成了两节，分别被变量%%i 和变量%%j 接收。

以上几种切分方式可以结合在一起使用。不知道下面这段代码的含义你是否看得懂，如果看不懂的话，那就运行一下代码，然后反复揣摩，你一定会更加深刻地理解本节所讲解的内容的：

[code13]

```
@echo off
for /f "delims= ,  tokens=1,3-4,*" %%i in (test.txt) do echo %%i %%j %%k %%l
pause
```

（四） 跳过无关内容，直奔主题：skip=n

很多时候，有用的信息并不是贯穿文本内容的始终，而是位于第 N 行之后的行内，为了提高文本处理的效率，或者不受多余信息的干扰，for /f 允许你跳过这些无用的行，直接从第 N+1 行开始处理，这个时候，就需要使用参数 skip=n，其中，n 是一个正整数，表示要跳过的行数。例如：

[code14]

```
@echo off
for /f "skip=2" %%i in (test.txt) do echo %%i
pause
```

这段代码将跳过头两行内容，从第 3 行起显示 test.txt 中的信息。

（五） 忽略以指定字符打头的行：eol=

在 cmd 窗口中敲入：for /?，相关的解释为：

[quote]
eol=c　　- 指一个行注释字符的结尾(就一个)
[/quote]

[quote]
FOR /F "eol=; tokens=2,3* delims=, " %i in (myfile.txt) do @echo %i %j %k

会分析 myfile.txt 中的每一行，忽略以分号打头的那些行……
[/quote]

第一条解释狗屁不通，颇为费解：行注释字符的结尾是什么意思？“(就一个)”怎么回事？结合第二条解释，才知道 eol 有忽略指定行的功能。但是，这两条解释是互相矛盾的：到底是忽略以指定字符打头的行，还是忽略以指定字符结尾的行？

实践是检验真理的唯一标准，还是用代码来检验一下 eol 的作用吧：

[code15]

```
@echo off
for /f "eol=;" %%i in (test.txt) do echo %%i
pause
```

结果，那些以分号打头的行没有显示出来。

由此可见，第二条解释是正确的，**eol= 的准确含义是：忽略以指定字符打头的行。**而第一条的“结尾”纯属微软在信口开河。

那么，“(就一个)”又作何解释呢？

试试这个代码：

[code16]

```
@echo off
for /f "eol=,;" %%i in (test.txt) do echo %%i
pause
```

此时，屏幕上出现 此时不应有 ;"。 的报错信息。可见，在指定字符的时候，只能指定 1 个——在很多时候，我对这样的设计颇有微词而又无可奈何：为什么只能指定 1 个而不是多个？要忽略多个还得又是 if 又是 findstr 加管道来多次过滤，那效率实在太低下了——能用到的功能基本上都提供，但是却又做不到更好，批处理，你的功能为什么那么弱？

不知道大家注意到没有，如果 test.txt 中有以分号打头的行，那么，这些行在代码[code14]的执行结果中将凭空消失。

原来，**for /f 语句是默认忽略以分号打头的行内容的，正如它默认以空格键或跳格键作为字符串的切分字符一样。**

很多时候，我们可以充分利用这个特点，比如，在设计即将用 for 读取的配置文件的时候，可以在注释文字的行首加上分号，例如在编写病毒文件查杀代码

的时候，可以通过 for 语句来读取病毒文件列表，那么，病毒文件列表.ini 这个配置文件可以这样写：

```
;以下是常见的病毒文件，请见一个杀一个^_^
;copyleft:没有
qq.exe
msn.exe
iexplore.exe
```

如果要取消这个默认设置，可选择的办法是：

1、为 eol=指定另外一个字符；
2、使用 for /f "eol=" 语句，也就是说，强制指定字符为空，就像对付 delims=一样。

（六）如何决定该使用 for /f 的哪种句式？(兼谈 usebackq 的使用)

for /f %%i in (……) do (……) 语句有好几种变形语句，不同之处在于第一个括号里的内容：有的是用单引号括起来，有的是用双引号包住，有的不用任何符号包裹，具体格式为：

1、for /f %%i in (文件名) do (……)
2、for /f %%i in ('命令语句') do (……)
3、for /f %%i in ("字符串") do (……)

看到这里，我想很多人可能已经开始犯了迷糊了：如果要解决一个具体问题，面对这么多的选择，如何决定该使用哪一条呢？

实际上，当我在上面罗列这些语句的时候，已经有所提示了，不知道你是否注意到了。

如果你一时无法参透其中奥妙，那也无妨，请听我一一道来便是。

1、当你希望读取文本文件中的内容的话，第一个括号中不用任何符号包裹，应该使用的是第 1 条语句；例如：你想显示 test.txt 中的内容，那么，就使用 for /f %%i in (test.txt) do echo %%i；

2、当你读取的是命令语句执行结果中的内容的话，第一个括号中的命令语句必须使用单引号包裹，应该使用的是第 2 条语句；例如：你想显示当前目录下

文件名中含有 test 字符串的文本文件的时候，应该使用 for /f %%i in ('dir /a-d /b *test*.txt') do echo %%i 这样的语句；

3、当你要处理的是一个字符串的时候，第一个括号中的内容必须用双引号括起来，应该是用的是第 3 条语句；例如：当你想把 bbs.bathome.cn 这串字符中的点号换为短横线并显示出来的话，可以使用 for /f "delims=. tokens=1-3" %%i in ("bbs.bathome.cn") do echo %%i-%%j-%%k 这样的语句。

很显然，第一个括号里是否需要用符号包裹起来，以及使用什么样的符号包裹，取决于要处理的对象属于什么类型：如果是文件，则无需包裹；如果是命令语句，则用单引号包裹；如果是字符串，则使用双引号括起来。

当然，事情并不是绝对如此，如果细心的你想到了批处理中难缠的特殊字符，你肯定会头大如斗。

或许你头脑中灵光一闪，已经想到了一个十分头痛的问题：在第 1 条语句中，如果文件名中含有空格或&，该怎么办？

照旧吗？

拿个叫 test 1.txt 的文件来试试。

你很快写好了代码，新建文件-->码字-->保存为批处理，前后费时不到 1 分钟：

[code17]

```
@echo off
for /f %%i in (test 1.txt) do echo %%i
pause
```

你兴冲冲地双击批处理，运行后，屏幕上出现了可耻的报错信息：系统找不到文件 test 。

当你把 test 1.txt 换成 test&1.txt 后，更怪异的事情发生了：CMD 窗口在你眼前一闪而过，然后，优雅地消失了。

你可能觉得自己的代码写错了某些符号，你再仔细的检查了一次，确认没有笔误，然后，你再次双击批处理，结果问题照旧；你开始怀疑其他程序对它可能有影响，于是关掉其他窗口，再运行了一次，问题依旧；你不服气地连续运行了好几次，还是同样的结果。

怪哉！

你一拍大腿，猛然想起了一件事：当路径中含有特殊字符的时候，应该使用引号把路径括起来。对，就是它了！

但是，当你把代码写出来之后，你很快就焉了：for /f %%i in ("test 1.txt") do echo %%i，这不就是上面提到的第 3 条 for /f 命令的格式吗？批处理会把 test 1.txt 这个文件名识别为字符串啊！

你百无聊赖地在 CMD 窗口中输入 for /? ，并重重地敲下了回车，漫无目的地在帮助信息中寻找，希望能找到点什么。

结果还真让你到了点什么。

你看到了这样的描述：

usebackq - 指定新语法已在下类情况中使用:
在作为命令执行一个后引号的字符串并且一个单引号字符为文字字符串命令并允许在 filenameset 中使用双引号扩起文件名称。

但是，通读一遍之后，你却如坠五里雾中，不知所云。

还好，下面有个例子，并配有简单的说明：

```
FOR /F "usebackq delims==" %i IN (`set`) DO @echo %i
```

会枚举当前环境中的环境变量名称。

你仔细对比了 for /f 语句使用 usebackq 和不使用 usebackq 时在写法上的差别，很快就找到了答案：当使用了 usebackq 之后，如果第一个括号中是一条命令语句，那么，就要把单引号'改成后引号`（键盘左上角 esc 键下面的那个按键，与~在同一键位上）。

回过头去再看那段关于 usebackq 的描述，字斟句酌，反复揣摩，终于被你破译了天机：**usebackq 是一个增强型参数，当使用了这个参数之后，原来的 for 语句中第一个括号内的写法要做如下变动：如果第一个括号里的对象是一条命令语句的话，原来的单引号'要改为后引号`；如果第一个括号里的对象是字符串的话，原来的双引号"要改为单引号'；如果第一个括号里的对象是文件名的话，要用双引号"括起来。**

验证一下，把[code17]改写成如下代码：

[code18]

```
@echo off
for /f "usebackq" %%i in ("test 1.txt") do echo %%i
pause
```

测试通过！

此时，你很可能会仰天长叹：Shit，微软这该死的机器翻译！

至于把[code17]代码中的空格换成&后，CMD 窗口会直接退出，那是因为&是复合语句的连接符，CMD 在预处理的时候，会优先把&前后两部分作为两条语句来解析，而不是大家想象中的一条完整的 for 语句，从而产生了严重的语法错误。因为牵涉到预处理机制问题，不属于本节要讨论的内容，在此不做详细讲解。

这个时候，我们会吃惊地发现，区区一条 for 语句，竟然有多达 6 种句型：

1、for /f %%i in (文件名) do (……)
2、for /f %%i in ('命令语句') do (……)
3、for /f %%i in ("字符串") do (……)
4、for /f "usebackq" %%i in ("文件名") do (……)
5、for /f "usebackq" %%i in (`命令语句`) do (……)
6、for /f "usebackq" %%i in ('字符串') do (……)

其中，4、5、6 由 1、2、3 发展而来，他们有这样的对应关系：1-->4、2-->5、3-->6。

好在后 3 种情形并不常用，所以，牢牢掌握好前三种句型的适用情形就可以了，否则，要在这么多句型中确定选择哪一条语句来使用，还真有点让人头脑发懵。

至于 for /f 为什么要增加 usebacq 参数，我只为第 4 条语句找到了合理的解释：为了兼容文件名中所带的空格或&。它在第 5、6 条语句中为什么还有存在的必要，我也不是很明白，这有待于各位去慢慢发现。

（七）变量延迟详解

变量延迟在 for 语句中起着至关重要的作用，不只是在 for 语句中，在其他的复合语句中，它也在幕后默默地工作着，为了突出它的重要性，本节内容在单独的楼层中发出来，希望引起大家的重视。

对于批处理新手而言，“变量延迟”这个概念很可能闻所未闻，但是，它却像一堵横亘在你前进道路上的无形高墙，你感受不到它的存在，但当你试图往前冲时，它会把你狠狠地弹回来，让你无法逾越、无功而返；而一旦找到了越过它的方法，你就会发现，在 for 的世界里，前面已经是一片坦途，而你对批处理的理解，又上升到了一个新的境界。

例如，你编写了这样一个代码：

[code19]
```
@echo off
set num=0&&echo %num%
pause
```

你的本意是想对变量 num 赋值之后，再把这个值显示出来，结果，显示出来的并不是 0，而是显示：ECHO 处于关闭状态。

之所以会出错，是因为“变量延迟”这个家伙在作怪。

在讲解变量延迟之前，我们需要了解一下批处理的执行过程，它将有助于我们深入理解变量延迟。

批处理的执行过程是怎样的呢？

“自上而下，逐条执行”，我想，这个经典的说法大家都已经耳熟能详了，没事的时候倒着念，也还别有一番古韵呢^_^，但是，我想问大家的是，大家真的深刻地理解了这句话的含义了吗？

“自上而下”，这一条和我们本节的讲解关系不大，暂时略过不说，后一条，“逐条执行”和变量延迟有着莫大的干系，它是我们本节要关注的重点。

很多人往往认为一行代码就是一条语句，从而把“逐条执行”与“逐行执行”等同起来，这就大错特错了。

莫非“逐条执行”里暗藏着玄机？

正是如此。

“逐条”并不等同于“逐行”。这个“条”，是“一条完整的语句”的意思，并不是指“一行代码”。在批处理中，是不是一条完整的语句，并不是以行来论的，而是要看它的作用范围。

什么样的语句才算“一条完整的语句”呢？

1、在复合语句中，整个复合语句是一条完整的语句，而无论这个复合语句占用了多少行的位置。常见的复合语句有：for 语句、if……else 语句、用连接符&、||和&&连接的语句，用管道符号|连接的语句，以及用括号括起来的、由多条语句组合而成的语句块；
2、在非复合语句中，如果该语句占据了一行的位置，则该行代码为一条完整的语句。

例如：

[code20]
```
@echo off
set num=0
for /f %%i in ('dir /a-d /b *.exe') do (
    set /a num+=1
    echo num 当前的值是 %num%
)
echo 当前目录下共有 %num% 个 exe 文件
dir /a-d /b *.txt|findstr "test">nul&&(
    echo 存在含有 test 字符串的文本本件
)||echo 不存在含有 test 字符串的文本文件
if exist test.ini (
    echo 存在 test.ini 文件
) else echo 不存在 test.ini 文件
pause
```

上面的代码共有 14 行，但是只有完整的语句只有 7 条，它们分别是：
第 1 条：第 1 行的 echo 语句；
第 2 条：第 2 行的 set 语句；
第 3 条：第 3、4、5、6 行上的 for 复合语句；
第 4 条：第 7 行的 echo 语句；
第 5 条：第 8、9、10 行上用&&和||连接的复合语句；
第 6 条：第 11、12、13 行上的 if……else 复合语句；
第 7 条：第 14 行上的 pause 语句。

在这里，我之所以要花这么长的篇幅来说明一行代码并不见得就是一条语句，是因为批处理的执行特点是“逐条”执行而不是“逐行”执行，澄清了这个误解，将会更加理解批处理的预处理机制。

在代码“逐条”执行的过程中，cmd.exe 这个批处理解释器会对每条语句做一些预处理工作，这就是批处理中大名鼎鼎的“预处理机制”。预处理的大致情形是这样的：首先，把一条完整的语句读入内存中（不管这条语句有多少行，它们都会被一起读入），然后，识别出哪些部分是命令关键字，哪些是开关、哪些是参数，哪些是变量引用……如果代码语法有误，则给出错误提示或退出批处理环境；如果顺利通过，接下来，就把该条语句中所有被引用的变量及变量两边的百分号对，用这条语句被读入内存之就已经赋予该变量的具体值来替换……当所有的预处理工作完成之后，批处理才会执行每条完整语句内部每个命令的原有功能。也就是说，如果命令语句中含有变量引用（变量及紧邻它左右的百分号对），并且某个变量的值在命令的执行过程中被改变了，即使该条语句内部的其他地方也用到了这个变量，也不会用最新的值去替换它们，因为某条语句在被预处理的时候，所有的变量引用都已经被替换成字符串常量了，变量值在复合语句内部被改变，不会影响到语句内部的其他任何地方。

顺便说一下，运行代码[code20]之后，将在屏幕上显示当前目录下有多少个 exe 文件，是否存在含有 test 字符串的文本文件，以及是否存在 test.ini 这个文件等信息。让很多人百思不得其解的是：如果当前目录下存在 exe 文件，那么，有多少个 exe 文件，屏幕上就会提示多少次 "num 当前的值是 0" ，而不是显示 1 到 N（N 是 exe 文件的个数）。

结合上面两个例子，我们再来分析一下，为什么这两段代码的执行结果和我们的期望有一些差距。

在[code19]中，set num=0&&echo %num%是一条复合语句，它的含义是：把 0 赋予变量 num，成功后，显示变量 num 的值。

虽然是在变量 num 被赋值成功后才显示变量 num 的值，但是，因为这是一条复合语句，在预处理的时候，&&后的%num%只能被 set 语句之前的语句赋予变量 num 的具体值来替换，而不能被复合语句内部、&&之前的 set 语句对 num 所赋予的值来替换，可见，此 num 非彼 num。可是，在这条复合语句之前，我们并没有对变量 num 赋值，所以，&&之后的%num%是空值，相当于在&&之后只执行了 echo 这一命令，所以，会显示 echo 命令的当前状态，而不是显示变量 num 的值（虽然该变量的值被 set 语句改变了）。

在[code20]中，for 语句的含义是：列举当前目录下的 exe 文件，每发现一个 exe 文件，变量 num 的值就累加 1，并显示变量 num 的值。

看了对[code19]的分析之后，再来分析[code20]就不再那么困难了：第 3、4、5 行上的代码共同构成了一条完整的 for 语句，而语句"echo num 当前的值是 %num%"与"set /a num+=1"同处复合语句 for 的内部，那么，第 4 行上 set 改变了 num 的值之后，并不能对第 5 行上的变量 num 有任何影响，因为在预处理阶段，第 5 行上的变量引用%num%已经被在 for 之前就赋予变量 num 的具体值替换掉了，它被替换成了 0（是被第 2 行上的 set 语句赋予的）。

如果想让代码[code19]的执行结果中显示&&之前赋予 num 的值，让代码[code20]在列举 exe 文件的时候，从 1 到 N 地显示 exe 文件的数量，那又该怎么办呢？

对代码[code19]，可以把用&&连接复合语句拆分为两条单独的语句，写成：

```
@echo off
set num=0
echo %num%
pause
```

但是，这不是我们这次想要的结果。

对这两段代码都适用的办法是：使用变量延迟扩展语句，让变量的扩展行为延迟一下，从而获取我们想要的值。

在这里，我们先来充下电，看看“变量扩展”有是怎么一回事。

用 CN-DOS 里批处理达人 willsort 的原话，那就是：“在许多可见的官方文档中，均将使用一对百分号闭合环境变量以完成对其值的替换行为称之为“扩展（expansion）”，这其实是一个第一方的概念，是从命令解释器的角度进行称谓的，而从我们使用者的角度来看，则可以将它看作是引用（Reference）、调用（Call）或者获取（Get）。”（见：什么情况下该使用变量延迟？http://www.cn-dos.net/forum/viewthread.php?tid=20733）说得直白一点，所谓的“变量扩展”，实际上就是很简单的这么一件事情：用具体的值去替换被引用的变量及紧贴在它左右的那对百分号。

既然只要延迟变量的扩展行为，就可以获得我们想要的结果，那么，具体的做法又是怎样的呢？

一般说来，延迟变量的扩展行为，可以有如下选择：

1、在适当位置使用 setlocal enabledelayedexpansion 语句；
2、在适当的位置使用 call 语句。

使用 setlocal enabledelayedexpansion 语句，那么，[code19]和[code20]可以分别修改为：

```
@echo off
setlocal enabledelayedexpansion
set num=0&&echo !num!
pause

@echo off
set num=0
setlocal enabledelayedexpansion
for /f %%i in ('dir /a-d /b *.exe') do (
    set /a num+=1
    echo num 当前的值是 !num!
)
echo 当前目录下共有 %num% 个 exe 文件
dir /a-d /b *.txt|findstr "test">nul&&(
    echo 存在含有 test 字符串的文本本件
)||echo 不存在含有 test 字符串的文本文件
if exist test.ini (
    echo 存在 test.ini 文件
) else 不存在 test.ini 文件
```

```
pause
```

使用第 call 语句，那么，[code19]和[code20]可以分别修改为：

```
@echo off
set num=0&&call echo %%num%%
pause
```

```
@echo off
set num=0
for /f %%i in ('dir /a-d /b *.exe') do (
    set /a num+=1
    call echo num 当前的值是 %%num%%
)
echo 当前目录下共有 %num% 个 exe 文件
dir /a-d /b *.txt|findstr "test">nul&&(
    echo 存在含有 test 字符串的文本本件
)||echo 不存在含有 test 字符串的文本文件
if exist test.ini (
    echo 存在 test.ini 文件
) else 不存在 test.ini 文件
pause
```

由此可见，如果使用 setlocal enabledelayedexpansion 语句来延迟变量，就要把原本使用百分号对闭合的变量引用改为使用感叹号对来闭合；如果使用 call 语句，就要在原来命令的前部加上 call 命令，并把变量引用的单层百分号对改为双层。 其中，因为 call 语句使用的是双层百分号对，容易使人犯迷糊，所以用得较少，常用的是使用 setlocal enabledelayedexpansion 语句（set 是设置的意思，local 是本地的意思，enable 是能够的意思，delayed 是延迟的意思，expansion 是扩展的意思，合起来，就是：让变量成为局部变量，并延迟它的扩展行为）。

通过上面的分析，我们可以知道：

1、为什么要使用变量延迟？因为要让复合语句内部的变量实时感知到变量值的变化。

2、在哪些场合需要使用变量延迟语句？在复合语句内部，如果某个变量的值发生了改变，并且改变后的值需要在复合语句内部的其他地方被用到，那么，就需要使用变量延迟语句。而复合语句有：for 语句、if……else 语句、用连接符&、||和&&连接的语句、用管道符号|连接的语句，以及用括号括起来的、由多条语句组合而成的语句块。最常见的场合，则是 for 语句和 if……else 语句。

3、怎样使用变量延迟？

方法有两种：

① 使用 setlocal enabledelayedexpansion 语句：在获取变化的变量值语句

之前使用 setlocal enabledelayedexpansion，并把原本使用百分号对闭合的变量引用改为使用感叹号对来闭合；

② 使用 call 语句：在原来命令的前部加上 call 命令，并把变量引用的单层百分号对改为双层。

“变量延迟”是批处理中一个十分重要的机制，它因预处理机制而生，用于复合语句，特别是大量使用于强大的 for 语句中。只有熟练地使用这一机制，才能在 for 的世界中如鱼得水，让自己的批处理水平更上一层楼。很多时候，对 for 的处理机制，我们一直是雾里看花，即使偶有所得，也只是只可意会难以言传。希望大家反复揣摩，多加练习，很多细节上的经验，是只有通过大量的摸索才能得到的。Good Luck！

本节内容在原理上参考了这篇文章：什么情况下该使用变量延迟？http://www.cn-dos.net/forum/viewthread.php?tid=20733，在本论坛中的地址是：http://bbs.bathome.cn/viewthread.php?tid=2899

特别鸣谢：willsort。

四、翻箱倒柜遍历文件夹：for /r

（一）for /r 的作用及用法

按照帮助信息里文绉绉的说法，for /r 的作用是“递归”，我们换一个通俗一点的，叫“遍历文件夹”。

更详细的解释就是：在下面的语句中，如果“元素集合”中只是一个点号，那么，这条语句的作用就是：列举“目录”及其之下的所有子目录，对这些文件夹都执行“命令语句集合”中的命令语句。其作用与嵌套进 for /f 复合语句的 "dir /ad /b /s 路径" 功能类似。如果省略了“目录”，将在当前目录下执行前面描述的操作。

[quote]
for /r 目录 %%i in (元素集合) do 命令语句集合
[/quote]

先来个代码增强一下印象：

[code21]

```
@echo off
for /r d:\test %%i in (.) do echo %%i
pause
```

执行的结果如下所示：

[quote]
d:\test\.
d:\test\1\.
d:\test\2\.
d:\test\3\.
[/quote]

效果就是显示 d:\test 目录及其之下是所有子目录的路径，其效果与 dir /ad /b /s d:\test 类似。若要说到两者的区别，可以归纳出 3 点：

1、for /r 列举出来的路径最后都带有斜杠和点号，而 dir 语句则没有，会对获取到的路径进行进一步加工产生影响；

2、for /r 不能列举带隐藏属性的目录，而 dir 语句则可以通过指定 /a 后面紧跟的参数来获取带指定属性的目录，更加灵活；

3、若要对获取到的路径进行进一步处理，则需要把 dir 语句放入 for /f 语句中进行分析，写成 for /f %%i in ('dir /ad /b /s') do …… 的形式；由于 for /r 语句是边列举路径边进行处理，所以，在处理大量路径的时候，前期不会感到有停顿，而 for /f 语句则需要等到 dir /ad /b /s 语句把所有路径都列举完之后，再读入内存进行处理，所以，在处理大量路径的时候，前期会感到有明显的停顿。

第 2 点差别很容易被大家忽视，导致用 for /r 列举路径的时候会造成遗漏；而第 3 点则会让大家有更直观的感受，很容易感觉到两者之间的差别。

要是“元素集合”不是点号呢？那又如何？

来看看这个代码：

[code22]

```
@echo off
for /r d:\test %%i in (a b c) do echo %%i
pause
```

运行的结果是：

[quote]

```
D:\test\1\a
D:\test\1\b
D:\test\1\c
D:\test\2\a
D:\test\2\b
D:\test\2\c
D:\test\3\a
D:\test\3\b
D:\test\3\c
```

[/quote]

原来，它的含义是：列举 d:\test 及其所有的子目录，对所有的目录路径都分别添加 a、b、c 之后再显示出来。

再来看一个代码：

[code23]

```
@echo off
for /r d:\test %%i in (*.txt) do echo %%i
pause
```

运行结果是：

[quote]
D:\test\test.txt
D:\test\1\1.txt
D:\test\1\2.txt
D:\test\2\a.txt
D:\test\2\b.txt
D:\test\3\1.txt
[/quote]

这段代码的含义是：列举 d:\test 及其所有子目录下的 txt 文本文件（以.txt 结尾的文件夹不会被列出来）。

我们再回过头来归纳一下这个语句的作用：

[quote]
for /r 目录 %%i in (元素集合) do 命令语句集合
[/quote]

上面语句的作用是：

1、列举“目录”及该目录路径下所有子目录，并把列举出来的目录路径和元素集合中的每一个元素拼接成形如“目录路径\元素”格式的新字符串，然后，对每一条这样的新字符串执行“命令语句集合”中的每一条命令；
特别的是：当“元素集合”带以点号分隔的通配符?或*的时候，把“元素集合”视为文件（不视为文件夹），整条语句的作用是匹配“目录”所指文件夹及其所有子文件夹下匹配的文件；若不以点号分隔，则把“元素集合”视为文件夹（不视为文件）；
2、当省略掉“目录”时，则针对当前目录；
3、当元素集合中仅仅是一个点号的时候，将只列举目录路径；

（二）for /r 还是 dir /ad /b /s？列举目录时该如何选择

前面已经说过，当列举目录时，for /r 和 dir /ad /b /s 的效果是非常类似的，这就产生了一个问题：当我要获取目录路径并进行进一步处理的时候，两者之间，我该如何选择？

这个问题，前面其实已经有过一些讨论，现在我们再来作详细的分析。

我们来看一下两者各自的优缺点：

1、for /r：

1）优点：

① 只通过 1 条语句就可以同时实现获取目录路径和处理目录路径的操作；
② 遍历文件夹的时候，是边列举边处理的，获取到一条路径就处理一条路径，内存占用小，处理大量路径的时候不会产生停顿感；

2）缺点：
① 不能获取到带隐藏属性的目录，会产生遗漏；
② 不能获取带指定属性的目录

2、dir /ad /s：

1）优点：

① 能一次性获取带任意属性的目录，不会产生遗漏；
② 能通过指定不同的参数获取带任意属性的目录，更具灵活性。

2）缺点：
① dir /ad /s 语句仅能获取到目录路径，若要实现进一步的处理，还需要嵌入 for /f 语句中才能实现，写法不够简洁；
② 嵌入 for /f 语句之后，需要写成 for /f "delims=" %%i in ('dir /ad /b /s') do …… 的格式，受 for /f 语句运行机制的制约，需要先列举完所有的路径放入内存之后，才能对每一条路径进行进一步的处理，处理大量路径时，内存占用量偏大，并且在前期会产生明显的停顿感，用户体验度不够好；

综合上述分析，可以做出如下选择：

1、若仅仅是为了获取某文件夹及其所有子文件夹的路径的话，请选择 dir /ad /b /s 语句；
2、若需要过滤带隐藏属性的文件夹的话，for /r 和 dir 语句都可以实现，但 for /r 内存占用小，处理速度快，是上上之选；
3、若需要获取所有文件夹，则除了 dir /ad /b /s 外，别无选择，因为 for /r 语句会遗漏带隐藏属性的文件夹；

在实际的使用中，我更喜欢使用 for /f 和 dir 的组合，因为它不会产生遗漏，并能给我带来更灵活的处理方式，唯一需要忍受的，就是它在处理大量路径时前期的停顿感，以及在这背后稍微有点偏高的内存占用；在我追求速度且可以忽略带隐藏属性的目录的时候，我会换用 for /r 的方案，不过这样的情形不多—

—有谁会愿意为了追求速度而容忍遗漏呢？

五、仅仅为了匹配第一层目录而存在：for /d

for /d 中 /d ，完整的含义是 /directory，本意是为了处理文件夹，它的完整语句应该是这样的：

[quote]
for /d %%i in (元素集合) do 命令语句集合
[/quote]

当“元素集合”中包含有通配符?或*时，它会匹配文件夹，但是，相比 for /r 而言，这个时候的 for /d，其作用就小得可怜了：它仅能匹配当前目录下的第一级文件夹，或是指定位置上的文件夹，而不能匹配更深层次的子文件夹。

例如：for /d %%i in (d:\test*) do echo %%i 这样的语句 ，会匹配到形如：d:\test、d:\test1、d:\test2 之类的文件夹，若不存在这样的路径，将不会有任何回显。

当“元素集合”中不包含任何的通配符时，它的作用和 "for %%i in (元素集合) do 命令语句集合" 这样的语句别无二致。

因此，for /d 的角色就变得很微妙了：当“元素集合”中包含通配符?或*时，它的作用就是匹配文件夹，此时，它仅能匹配当前目录下的第一级文件夹，或是指定位置上的文件夹，在层次深度上不及 for /r，但和 for /r 一样的坏脾气：不能匹配带隐藏属性的文件夹；在灵活性上不及 for /f 和 dir 的组合；当“元素集合”中不包含任何统配符的时候，它完全是 "for %%i in (元素集合) do ……" 语句的翻版，但是又稍显复杂。

for /d 的作用是如此有限，我使用的次数是如此之少，以至于我一度找不到它的用武之地，认为它食之无味，弃之可惜，完全是鸡肋一块。

某年某月，我在 cmd 窗口里写下了这样的代码：

[code24]
```
for /d %i in (test*) do @echo %i
```

我的本意是想查看在我的临时目录下，长年累月的测试工作到底建立了多少测试文件夹，以便我随后把 echo 换成 rd 删除之。这个时候，我发现这条代码是

如此的简洁，是 for /r 或 for 和 dir /ad /b 的组合所无法替代的（echo 换成 rd 就可以直接删除掉这些测试目录）。

简洁的代码给我带来的喜悦仅仅持续了短短 10 几秒的时间，我便开始了迷惘——能用到 for /d 的类似情形，貌似少之又少且乏善可陈啊。

六、计数循环：for /l

/l 者，/loop 的缩写是也，从鸟语翻译过来，loop 就是循环的意思。实际上，所有的 for 语句，都可以看成是一种“循环”，只是在/l 中，特指按照指定次数进行循环罢了。

for /l 语句的完整格式是这样的：for /l %%i in (x,y,z) do (……)，在这个语句中，x、y 和 z 都只能取整数，正负皆可，x 指代起始值，y 指代步长，z 为终止值，具体含义为：从 x 开始计数，以 y 为步长，直至最接近 z 的那个整数值为止，这之间有多少个数，do 后的语句就执行多少次。

举个具体的例子：

[code25]

```
for /l %%i in (1,2,10) do echo bathome
```

在以上的代码中，初始值是 1，步长为 2，终止值为 10，表明计数从 1 开始，每隔 2 个数计算一次，直至最接近 10 的那个整数，罗列出来，就是 1,3,5,7,9，再下一个就是 11，超过 10 了，不再计算在内，所以，do 后的语句只执行 5 次，将连续显示 5 个 bathome。

实际上，x，y 和 z 的值可正可负，甚至为 0，限制非常宽松：

1、步长 y 的值不能为 0；
2、当步长 y 的值为正整数时，终止值 z 不能小于初始值 x；
3、当步长 y 的值为负整数的时候，终止值 z 不能大于初始值 x。

换而言之，必须保证 in 和 do 之间能取到一个有效的数组序列。

例如：

[code26]

```
for /l %%i in (-1,2,5) do echo bathome
```

[code27]

```
for /l %%i in (5,-2,-1) do echo bathome
```

以上两条代码的功能完全一样，都将显示 4 次 bathome，区别就在于[code26]是正序计算，而[code27]是逆序计数而已。

以下几条代码都是有问题的：

[code28]

```
for /l %%i in (1,0,1) do echo bathome
```

[code29]

```
for /l %%i in (2,1,1) do echo bathome
```

[code30]

```
for /l %%i in (1,-1,2) do echo bathome
```

其中，[code28]违反了步长不能为 0 的限制，将陷入无限循环中；[code29]和[code30]都犯了同样的错误：无法获得有效的数列元素，导致 in 和 do 之间取到的值为空元素，从而使得整条 for 语句无从执行。

当大家明白了 for /l 的具体功能之后，是否会想到了与它有异曲同工之妙的 goto 循环语句呢？似乎，for /l 和 goto 循环语句可以相互替换？

一般而言，for /l 语句可以换成 goto 循环，但是，goto 循环并不一定能被 for /l 语句替换掉。具体原因，请大家仔细想想，我在此不再详细解说，只是就大家非常关心的一个问题提供一个简洁的答案，那就是：什么时候该用 for /l 计数循环，而什么时候又该用 goto 条件循环？

答案非常简单：**当循环次数确定的时候，首选 for /l 语句，也可使用 goto 语句但不推荐；当循环次数不确定的时候，用 goto 语句将是唯一的选择，因为，这个时候需要用 if 之类的条件语句来判断何时结束 goto 跳转。**

后记：

当 Windows 为我们打开了五彩缤纷的图形窗口的时候
DOS 命中注定会陨落
CMD 毫无悬念将萎缩
批处理逐渐趋向无声无息
而 powershell 的到来，无疑会让更多的人忘记批处理
这是一门即将失传的技艺
这是一块行将就木的领域
然而，命令行工具仍然具有批量处理一切的巨大威力
字符界面仍然是高效操作的代名词
曾为批处理的方便灵活而击节赞赏
曾被批处理的简洁快速深深折服
一直以来，总想为批处理的推广做些什么
于是，从在 CN-DOS 里尽职尽责地为大家解答疑问，到创办了自己的论坛专职答疑解惑，再到无怨无悔地码字写教程，一步步走来，喜怒哀愁，五味杂陈
直至如今辞去站长等一切管理职务，逐渐淡出批处理圈子
梦依旧在，只是，心有余而力渐有不足
这篇从入门到精通的教学帖，从 2008 年 10 月开贴到现在，不知不觉拖拖拉拉竟然过去了两年有余
每每看到跟帖的会员在问什么时候有更新
心中总有一丝愧疚
今天，终于抽空对它做个了断
只是，年年岁岁花相似，岁岁年年人不同
繁杂的事务使我已不再有当初的心境
for /l 部分总有虎头蛇尾的感觉
只能向各位说声抱歉了
在我彻底淡出批处理圈子之前
我只能尽我所能地向各位倾我所学了
最后，我希望论坛的管理人员能按照顶楼的管理提示经常为这个帖子抽抽水
或者是为了大家阅览的方便而永久锁定这个帖子