正则表达式从入门到精通

什么是正则表达式

如果原来没有使用过正则表达式，那么可能对这个术语和概念会不太熟悉。不过，它们并不是您想象的那么新奇。

请回想一下在硬盘上是如何查找文件的。您肯定会使用 ? 和 * 字符来帮助查找您正寻找的文件。? 字符匹配文件名中的单个字符，而 * 则匹配一个或多个字符。

一个如 'data?.dat' 的模式可以找到下述文件：

data1.dat
data2.dat
datax.dat
dataN.dat

如果使用 * 字符代替 ? 字符，则将扩大找到的文件数量。'data*.dat' 可以匹配下述所有文件名：

data.dat
data1.dat
data2.dat
data12.dat
datax.dat
dataXYZ.dat

尽管这种搜索文件的方法肯定很有用，但也十分有限。? 和 * 通配符的有限能力可以使你对正则表达式能做什么有一个概念，不过正则表达式的功能更强大，也更灵活。

早期起源

正则表达式的“祖先”可以一直上溯至对人类神经系统如何工作的早期研究。Warren McCulloch 和 Walter Pitts 这两位神经生理学家研究出一种数学方式来描述这些神经网络。

1956 年，一位叫 Stephen Kleene 的数学家在 McCulloch 和 Pitts 早期工作的基础上，发表了一篇标题为“神经网事件的表示法”的论文，引入了正则表达式的概念。正则表达式就是用来描述他称为“正则集的代数”的表达式，因此采用“正则表达式”这个术语。

随后，发现可以将这一工作应用于使用 Ken Thompson 的计算搜索算法的一些早期研究，Ken Thompson 是 Unix 的主要发明人。正则表达式的第一个实用应用程序就是 Unix 中的 qed 编辑器。

如他们所说，剩下的就是众所周知的历史了。从那时起直至现在正则表达式都是基于文本的编辑器和搜索工具中的一个重要部分。

使用正则表达式

在典型的搜索和替换操作中，必须提供要查找的确切文字。这种技术对于静态文本中的简单搜索和替换任务可能足够了，但是由于它缺乏灵活性，因此在搜索动态文本时就有困难了，甚至是不可能的。

使用正则表达式，就可以：

- ✧ 测试字符串的某个模式。例如，可以对一个输入字符串进行测试，看在该字符串是否存在一个电话号码模式或一个信用卡号码模式。这称为数据有效性验证。
- ✧ 替换文本。可以在文档中使用一个正则表达式来标识特定文字，然后可以全部将其删除，或者替换为别的文字。
- ✧ 根据模式匹配从字符串中提取一个子字符串。可以用来在文本或输入字段中查找特定文字。

例如，如果需要搜索整个 web 站点来删除某些过时的材料并替换某些 HTML 格式化标记，则可以使用正则表达式对每个文件进行测试，看在该文件中是否存在所要查找的材料或 HTML 格式化标记。用这个方法，就可以将受影响的文件范围缩小到包含要删除或更改的材料的那些文件。然后可以使用正则表达式来删除过时的材料，最后，可以再次使用正则表达式来查找并替换那些需要替换的标记。

另一个说明正则表达式非常有用的示例是一种其字符串处理能力还不为人所知的语言。VBScript 是 Visual Basic 的一个子集，具有丰富的字符串处理功能。与 C 类似的 Jscript 则没有这一能力。正则表达式给 JScript 的字符串处理能力带来了明显改善。不过，可能还是在 VBScript 中使用正则表达式的效率更高，它允许在单个表达式中执行多个字符串操作。

正则表达式语法

一个正则表达式就是由普通字符（例如字符 a 到 z）以及特殊字符（称为*元字符*）组成的文字模式。该模式描述在查找文字主体时待匹配的一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

这里有一些可能会遇到的正则表达式示例：

JScript	**VBScript**	**匹配**
/^\[\t]*$/	"^\[\t]*$"	匹配一个空白行。
/\d{2}-\d{5}/	"\d{2}-\d{5}"	验证一个 ID 号码是否由一个 2 位数字，一个连字符以及一个 5 位数字组成。
/<(.*)>.*<\/\1>/	"<(.*)>.*<\/\1>"	匹配一个 HTML 标记。

下表是元字符及其在正则表达式上下文中的行为的一个完整列表：

字符	**描述**
\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个 向后引用、或一个八进制转义符。例如，'n' 匹配字符 "n"。'\n' 匹配一个换行符。序列 '\\' 匹配 "\" 而 "\(" 则匹配 "("。
^	匹配输入字符串的开始位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。

*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于 {0,1}。
{*n*}	*n* 是一个非负整数。匹配确定的 *n* 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{*n*,}	*n* 是一个非负整数。至少匹配 *n* 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{*n*,*m*}	*m* 和 *n* 均为非负整数，其中 *n* <= *m*。最少匹配 *n* 次且最多匹配 *m* 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。
?	当该字符紧跟在任何一个其他限制符 (*，+，?，{*n*}，{*n*,}，{*n*,*m*}) 后面时，匹配模式是非贪婪的。非贪婪模式尽可能少的匹配所搜索的字符串，而默认的贪婪模式则尽可能多的匹配所搜索的字符串。例如，对于字符串 "oooo"，'o+?' 将匹配单个 "o"，而 'o+' 将匹配所有 'o'。
.	匹配除 "\n" 之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用象 '[.\n]' 的模式。
(*pattern*)	匹配 *pattern* 并获取这一匹配。所获取的匹配可以从产生的 Matches 集合得到，在 VBScript 中使用 **SubMatches** 集合，在 JScript 中则使用 **$0**…**$9** 属性。要匹配圆括号字符，请使用 '\(' 或 '\)'。
(?:*pattern*)	匹配 *pattern* 但不获取匹配结果，也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符 (|) 来组合一个模式的各个部分是很有用。例如， 'industr(?:y|ies) 就是一个比 'industry|industries' 更简略的表达式。
(?=*pattern*)	正向预查，在任何匹配 *pattern* 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如，'Windows (?=95|98|NT|2000)' 能匹配 "Windows 2000" 中的 "Windows" ，但不能匹配 "Windows 3.1" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。
(?!*pattern*)	负向预查，在任何不匹配 *pattern* 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如'Windows (?!95|98|NT|2000)' 能匹配 "Windows 3.1" 中的 "Windows"，但不能匹配 "Windows 2000" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
x|*y*	匹配 *x* 或 *y*。例如，'z|food' 能匹配 "z" 或 "food"。'(z|f)ood' 则匹配 "zood" 或 "food"。
[*xyz*]	字符集合。匹配所包含的任意一个字符。例如， '[abc]' 可以匹配 "plain" 中的 'a'。
[^*xyz*]	负值字符集合。匹配未包含的任意字符。例如， '[^abc]' 可以匹配 "plain" 中的'p'。
[*a*-*z*]	字符范围。匹配指定范围内的任意字符。例如，'[a-z]' 可以匹配 'a' 到 'z' 范围内的任意小写字母字符。
[^*a*-*z*]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，'[^a-z]' 可以匹配任何不在 'a' 到 'z' 范围内的任意字符。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的

	'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。
\c*x*	匹配由 *x* 指明的控制字符。例如，\cM 匹配一个 Control-M 或回车符。*x* 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\d	匹配一个数字字符。等价于 [0-9]。
\D	匹配一个非数字字符。等价于 [^0-9]。
\f	匹配一个换页符。等价于 \x0c 和 \cL。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。
\w	匹配包括下划线的任何单词字符。等价于'[A-Za-z0-9_]'。
\W	匹配任何非单词字符。等价于 '[^A-Za-z0-9_]'。
\x*n*	匹配 *n*，其中 *n* 为十六进制转义值。十六进制转义值必须为确定的两个数字长。例如，'\x41' 匹配 "A"。'\x041' 则等价于 '\x04' & "1"。正则表达式中可以使用 ASCII 编码。.
num	匹配 *num*，其中 *num* 是一个正整数。对所获取的匹配的引用。例如，'(.)\1' 匹配两个连续的相同字符。
n	标识一个八进制转义值或一个向后引用。如果 *n* 之前至少 *n* 个获取的子表达式，则 *n* 为向后引用。否则，如果 *n* 为八进制数字 (0-7)，则 *n* 为一个八进制转义值。
nm	标识一个八进制转义值或一个向后引用。如果 *nm* 之前至少有 *nm* 个获得子表达式，则 *nm* 为向后引用。如果 *nm* 之前至少有 *n* 个获取，则 *n* 为一个后跟文字 *m* 的向后引用。如果前面的条件都不满足，若 *n* 和 *m* 均为八进制数字 (0-7)，则 *nm* 将匹配八进制转义值 *nm*。
nml	如果 *n* 为八进制数字 (0-3)，且 *m* 和 *l* 均为八进制数字 (0-7)，则匹配八进制转义值 *nml*。
\u*n*	匹配 *n*，其中 *n* 是一个用四个十六进制数字表示的 Unicode 字符。例如， \u00A9 匹配版权符号 (©)。

建立正则表达式

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与操作符将小的表达式结合在一起来创建更大的表达式。

可以通过在一对分隔符之间放入表达式模式的各种组件来构造一个正则表达式。对 JScript 而言，分隔符为一对正斜杠 (/) 字符。例如：/expression/

对 VBScript 而言，则采用一对引号 ("") 来确定正则表达式的边界。例如："expression"

在上面所示的两个示例中，正则表达式模式 (expression) 均存储在 RegExp 对象的 Pattern 属性中。

正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择或者所有这些组件的任意组合。

优先级顺序

在构造正则表达式之后，就可以象数学表达式一样来求值，也就是说，可以从左至右并按照一个优先级顺序来求值。

下表从最高优先级到最低优先级列出各种正则表达式操作符的优先级顺序：

操作符	描述
\	转义符
()，(?:)，(?=)，[]	圆括号和方括号
*，+，?，{n}，{n,}，{n,m}	限定符
^，$，*\anymetacharacter*	位置和顺序
|	“或”操作

普通字符

普通字符由所有那些未显式指定为元字符的打印和非打印字符组成。这包括所有的大写和小写字母字符，所有数字，所有标点符号以及一些符号。

最简单的正则表达式是一个单独的普通字符，可以匹配所搜索字符串中的该字符本身。例如，单字符模式 'A' 可以匹配所搜索字符串中任何位置出现的字母 'A'。这里有一些单字符正则表达式模式的示例：

```
/a/
/7/
/M/
```

等价的 VBScript 单字符正则表达式为：

```
"a"
"7"
"M"
```

可以将多个单字符组合在一起得到一个较大的表达式。例如，下面的 JScript 正则表达式不是别的，就是通过组合单字符表达式 'a'、'7'以及 'M' 所创建出来的一个表达式。

```
/a7M/
```

等价的 VBScript 表达式为：

```
"a7M"
```

请注意这里没有连接操作符。所需要做的就是将一个字符放在了另一个字符后面。

特殊字符

有不少元字符在试图对其进行匹配时需要进行特殊的处理。要匹配这些特殊字符，必须首先将这些字符转义，也就是在前面使用一个反斜杠 (\)。下表给出了这些特殊字符及其含义：

特殊字符	说明
$	匹配输入字符串的结尾位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，则 $ 也匹配 '\n' 或 '\r'。要匹配 $ 字符本身，请使用 \$。
( )	标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 \(和 \)。
*	匹配前面的子表达式零次或多次。要匹配 * 字符，请使用 *。
+	匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 \+。
.	匹配除换行符 \n 之外的任何单字符。要匹配 .，请使用 \。
[	标记一个中括号表达式的开始。要匹配 [，请使用 \[。
?	匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。要匹配 ? 字符，请使用 \?。
\	将下一个字符标记为或特殊字符、或原义字符、或向后引用、或八进制转义符。例如， 'n' 匹配字符 'n'。'\n' 匹配换行符。序列 '\\' 匹配 "\"，而 '\(' 则匹配 "("。
^	匹配输入字符串的开始位置，除非在方括号表达式中使用，此时它表示不接受该字符集合。要匹配 ^ 字符本身，请使用 \^。
{	标记限定符表达式的开始。要匹配 {，请使用 \{。
|	指明两项之间的一个选择。要匹配 |，请使用 \\|。

非打印字符

有不少很有用的非打印字符，偶尔必须使用。下表显示了用来表示这些非打印字符的转义序列：

字符	含义
\c*x*	匹配由 *x* 指明的控制字符。例如， \cM 匹配一个 Control-M 或回车符。*x* 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的 'c' 字符。
\f	匹配一个换页符。等价于 \x0c 和 \cL。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。

字符匹配

句点 (.) 匹配一个字符串中任何单个的打印或非打印字符，除了换行符 (\n) 之外。下面的 JScript 正则表达式可以匹配 'aac'、'abc'、'acc'、'adc'如此等等，同样也可以匹配 'a1c'、'a2c'、a-c'以及 a#c'：

```
/a.c/
```

等价的 VBScript 正则表达式为：

```
"a.c"
```

如果试图匹配一个包含文件名的字符串，其中句点 (.) 是输入字符串的一部分，则可以在正则表达式中的句点前面加上一个反斜杠 (\) 字符来实现这一要求。举例来说，下面的 JScript 正则表达式就能匹配 'filename.ext'：

```
/filename\.ext/
```

对 VBScript 而言，等价的表达式如下所示：

```
"filename\.ext"
```

这些表达式仍然是相当有限的。它们只允许匹配*任何*单字符。很多情况下，对从列表中匹配特殊字符十分有用。例如，如果输入文字中包含用数字表示为 Chapter 1，Chapter 2 诸如此类的章节标题，你可能需要找到这些章节标题。

括号表达式

可以在一个方括号 ([和]) 中放入一个或多个单字符，来创建一个待匹配的列表。如果字符被放入括号中括起来，则该列表称为*括号表达式*。括号内和其他任何地方一样，普通字符代表其本身，也就是说，它们匹配输入文字中出现的一处自己。大多数特殊字符在位于括号表达式中时都将失去其含义。这里有一些例外：

- ']' 字符如果不是第一项，则将结束一个列表。要在列表中匹配 ']' 字符，请将其放在第一项，紧跟在开始的 '[' 后面。
- '\' 仍然作为转义符。要匹配 '\' 字符，请使用 '\\'。

括号表达式中所包含的字符只匹配该括号表达式在正则表达式中所处位置的一个单字符。下面的 JScript 正则表达式可以匹配 'Chapter 1'、'Chapter 2'、'Chapter 3'、'Chapter 4' 以及 'Chapter 5'：

```
/Chapter [12345]/
```

在 VBScript 中要匹配同样的章节标题，请使用下面的表达式：

```
"Chapter [12345]"
```

请注意单词 'Chapter' 及后面的空格与括号内的字符的位置关系是固定的。因此，括号表达式只用来指定满足紧跟在单词 'Chapter' 和一个空格之后的单字符位置的字符集合。这里是第九个字符位置。

如果希望使用范围而不是字符本身来表示待匹配的字符，则可以使用连字符将该范围的开始和结束字符分开。每个字符的字符值将决定其在一个范围内的相对顺序。下面的 JScript 正则表达式包含了一个等价于上面所示的括号列表的范围表达式。

```
/Chapter [1-5]/
```

VBScript 中相同功能的表达式如下所示：

```
"Chapter [1-5]"
```

如果以这种方式指定范围，则开始和结束值都包括在该范围内。有一点特别需要注意的是，在 Unicode 排序中起始值一定要在结束值之前。

如果想在括号表达式中包括连字符，则必须使用下述方法之一：

- 使用反斜杠将其转义：[\-]
- 将连字符放在括号列表的开始和结束位置。下面的表达式能匹配所有的小写字母和连字符：[-a-z][a-z-]
- 创建一个范围，其中开始字符的值小于连字符，而结束字符的值等于或大于连字符。下面两个正则表达式都满足这一要求：[!--][!-~]

同样，通过在列表开始处放置一个插入符(^)，就可以查找所有不在列表或范围中的字符。如果该插入符出现在列表的其他位置，则匹配其本身，没有任何特殊含义。下面的 JScript 正则表达式匹配章节号大于 5 的章节标题：

```
/Chapter [^12345]/
```

对 VBScript 则使用：

```
"Chapter [^12345]"
```

在上面所示的示例中，表达式将匹配第九个位置处除 1，2，3，4，or 5 之外的任何数字字符。因此，'Chapter 7' 为一个匹配，同样 'Chapter 9' 也是如此。

上面的表达式可以使用连字符 (-) 表示。对 JScript 为：

```
/Chapter [^1-5]/
```

或者，对 VBScript 为：

```
"Chapter [^1-5]"
```

括号表达式的典型用法是指定对任何大写或小写字母字符或任何数字的匹配。下面的 JScript 表达式给出了这一匹配：

```
/[A-Za-z0-9]/
```

等价的 VBScript 表达式为：

″[A-Za-z0-9]″

限定符

有时候不知道要匹配多少字符。为了能适应这种不确定性，正则表达式支持限定符的概念。这些限定符可以指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。

下表给出了各种限定符及其含义的说明：

字符	描述
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 中的"do" 。? 等价于 {0,1}。
{*n*}	*n* 是一个非负整数。匹配确定的 *n* 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{*n*,}	*n* 是一个非负整数。至少匹配 *n* 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{*n*,*m*}	*m* 和 *n* 均为非负整数，其中 *n* <= *m*。最少匹配 *n* 次且最多匹配 *m* 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。

对一个很大的输入文档而言，章节数很轻易就超过九章，因此需要有一种方法来处理两位数或者三位数的章节号。限定符就提供了这个功能。下面的 JScript 正则表达式可以匹配具有任何位数的章节标题：

/Chapter [1-9][0-9]*/

下面的 VBScript 正则表达式执行同样的匹配：

″Chapter [1-9][0-9]*″

请注意限定符出现在范围表达式之后。因此，它将应用于所包含的整个范围表达式，在本例中，只指定了从 0 到 9 的数字。

这里没有使用 '+' 限定符，因为第二位或后续位置上并不一定需要一个数字。同样也没有使用 '?' 字符，因为这将把章节数限制为只有两位数字。在 'Chapter' 和空格字符之后至少要匹配一个数字。

如果已知章节数限制只有 99 章，则可以使用下面的 JScript 表达式来指定至少有一位数字，但不超过两个数字。

/Chapter [0-9]{1,2}/

对 VBScript 可以使用下述正则表达式：

"Chapter [0-9]{1,2}"

上述表达式的缺点是如果有一个章节号大于 99，它仍只会匹配前两位数字。另一个缺点是某些人可以创建一个 Chapter 0，而且仍能匹配。一个更好的用来匹配两位数的 JScript 表达式如下：

/Chapter [1-9][0-9]?/

或者

/Chapter [1-9][0-9]{0,1}/

对 VBScript 而言，下述表达式与上面等价：

"Chapter [1-9][0-9]?"

或者

"Chapter [1-9][0-9]{0,1}"

'*'、'+' 和 '?' 限定符都称之为*贪婪的*，也就是说，他们尽可能多地匹配文字。有时这根本就不是所希望发生的情况。有时则正好希望最小匹配。

例如，你可能要搜索一个 HTML 文档来查找一处包含在 H1 标记中的章节标题。在文档中该文字可能具有如下形式：

<H1>Chapter 1 - Introduction to Regular Expressions</H1>

下面的表达式匹配从开始的小于号 (<) 到 H1 标记结束处的大于号之间的所有内容。

/<.*>/

VBScript 的正则表达式为：

"<.*>"

如果所要匹配的就是开始的 H1 标记，则下述非贪婪地表达式就只匹配。

/<.*?>/

或者

"<.*?>"

通过在 '*'、 '+' 或 '?' 限定符后放置 '?'，该表达式就从贪婪匹配转为了非贪婪或最小匹配。

定位符

到现在为止，所看到的示例都只考虑查找任何地方出现的章节标题。出现的任何一个字符串 'Chapter' 后跟一个空格和一个数字可能是一个真正的章节标题，也可能是对其他章节的交叉引用。由于真正的章节标题总是出现在一行的开始，因此需要设计一个方法只查找标题而不查找交叉引用。

定位符提供了这个功能。定位符可以将一个正则表达式固定在一行的开始或结束。也可以创建只在单词内或只在单词的开始或结尾处出现的正则表达式。下表包含了正则表达式及其含义的列表：

字符	描述
^	匹配输入字符串的开始位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了 **RegExp** 对象的 **Multiline** 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
\b	匹配一个单词边界，也就是指单词和空格间的位置。
\B	匹配非单词边界。

不能对定位符使用限定符。因为在一个换行符或者单词边界的前面或后面不会有连续多个位置，因此诸如 '^*' 的表达式是不允许的。

要匹配一行文字开始位置的文字，请在正则表达式的开始处使用 '^' 字符。不要把 '^' 的这个语法与其在括号表达式中的语法弄混。它们的语法根本不同。

要匹配一行文字结束位置的文字，请在正则表达式的结束处使用 '$' 字符。

要在查找章节标题时使用定位符，下面的 JScript 正则表达式将匹配位于一行的开始处最多有两个数字的章节标题：

```
/^Chapter [1-9][0-9]{0,1}/
```

VBScript 中相同功能的正则表达式如下：

```
"^Chapter [1-9][0-9]{0,1}"
```

一个真正的章节标题不仅出现在一行的开始，而且这一行中也仅有这一个内容，因此，它必然也位于一行的结束。下面的表达式确保所指定的匹配只匹配章节而不会匹配交叉引用。它是通过创建一个只匹配一行文字的开始和结束位置的正则表达式来实现的。

```
/^Chapter [1-9][0-9]{0,1}$/
```

对 VBScript 则使用：

```
"^Chapter [1-9][0-9]{0,1}$"
```

匹配单词边界有少许不同，但却给正则表达式增加了一个非常重要的功能。单词边界就是单词和空格之间的位置。非单词边界就是其他任何位置。下面的 JScript 表达式将匹配单词 'Chapter' 的前三个字符，因为它们出现在单词边界后：

/\bCha/

对 VBScript 为：

″\bCha″

这里 '\b' 操作符的位置很关键。如果它位于要匹配的字符串的开始，则将查找位于单词开头处的匹配；如果它位于改字符串的末尾，则查找位于单词结束处的匹配。例如，下面的表达式将匹配单词 'Chapter' 中的 'ter'，因为它出现在单词边界之前：

/ter\b/

以及

″ter\b″

下面的表达式将匹配 'apt'，因为它位于 'Chapter' 中间，但不会匹配 'aptitude' 中的'apt'：

/\Bapt/

以及

″\Bapt″

这是因为在单词 'Chapter' 中 'apt' 出现在非单词边界位置，而在单词 'aptitude' 中位于单词边界位置。非单词边界操作符的位置不重要，因为匹配与一个单词的开头或结尾无关。

选择与编组

选择允许使用 '|' 字符来在两个或多个候选项中进行选择。通过扩展章节标题的正则表达式，可以将其扩充为不仅仅适用于章节标题的表达式。不过，这可没有想象的那么直接。在使用选择时，将匹配'|' 字符每边最可能的表达式。你可能认为下面的 JScript 和 VBScript 表达式将匹配位于一行的开始和结束位置且后跟一个或两个数字的 'Chapter' 或 'Section'：

- /^Chapter|Section [1-9][0-9]{0,1}$/
- "^Chapter|Section [1-9][0-9]{0,1}$"

不幸的是，真正的情况是上面所示的正则表达式要么匹配位于一行开始处的单词 'Chapter'，要么匹配一行结束处的后跟任何数字的 'Section'。如果输入字符串为 'Chapter 22'，上面的表达式将只匹配单词 'Chapter'。如果输入字符串为 'Section 22'，则该表达式将匹配 'Section 22'。但这种结果不是我们此处的目的，因此必须有一种办法来使正则表达式对于所要做的更易于响应，而且确实也有这种方法。

可以使用圆括号来限制选择的范围，也就是说明确该选择只适用于这两个单词 'Chapter' 和 'Section'。不过，圆括号同样也是难处理的，因为它们也用来创建子表达式，有些内容将在后面关于子表达式的部分介绍。通过采用上面所示的正则表达式并在适当位置添加圆括号，就可以使该正则表达式既可以匹配 'Chapter 1'，也可以匹配 'Section 3'。

下面的正则表达式使用圆括号将 'Chapter' 和 'Section' 组成一组，所以该表达式才能正确工作。对 JScript 为：

/^(Chapter|Section) [1-9][0-9]{0,1}$/

对 VBScript 为：

"^(Chapter|Section) [1-9][0-9]{0,1}$"

这些表达式工作正确，只是产生了一个有趣的副产品。在 'Chapter|Section' 两边放置圆括号建立了适当的编组，但也导致两个待匹配单词之一都被捕获供今后使用。由于在上面所示的表达式中只有一组圆括号，因此只能有一个捕获的 submatch。可以使用 VBScript 的 Submatches 集合或者 JScript 中 RegExp 对象的 $1-$9 属性来引用这个子匹配。

有时捕获一个子匹配是所希望的，有时则是不希望的。在说明所示的示例中，真正想做的就是使用圆括号对单词 'Chapter' 或 'Section' 之间的选择编组。并不希望在后面再引用该匹配。实际上，除非真的是需要捕获子匹配，否则请不要使用。由于不需要花时间和内存来存储那些子匹配，这种正则表达式的效率将更高。

可以在正则表达式模式圆括号内部的前面使用 '?:'来防止存储该匹配供今后使用。对上面所示正则表达式的下述修改提供了免除子匹配存储的相同功能。对 JScript：

/^(?:Chapter|Section) [1-9][0-9]{0,1}$/

对 VBScript：

"^(?:Chapter|Section) [1-9][0-9]{0,1}$"

除了 '?:' 元字符，还有两个非捕获元字符用于称之为预查的匹配。一个为正向预查，用 ?= 表示， 在任何开始匹配圆括号内的正则表达式模式的位置来匹配搜索字符串。一个为负向预查，用 '?!' 表示，在任何开始不匹配该正则表达式模式的位置来匹配搜索字符串。

例如，假定有一个包含引用有 Windows 3.1、Windows 95、Windows 98 以及 Windows NT 的文档。进一步假设需要更新该文档，方法是查找所有对 Windows 95、Windows 98 以及 Windows NT 的引用，并将这些引用更改为 Windows 2000。可以使用下面的 JScript 正则表达式，这是一个正向预查，来匹配 Windows 95、Windows 98 以及 Windows NT：

/Windows(?=95 |98 |NT)/

在 VBScript 要进行同样的匹配可以使用下述表达式：

"Windows(?=95 |98 |NT)"

找到一个匹配后，紧接匹配到的文字（而不包括预查中使用的字符）就开始对下一次匹配的搜索。例如，如果上面所示的表达式匹配到 'Windows 98'，则将从 'Windows' 而不是 '98' 之后继续查找。

向后引用

正则表达式一个最重要的特性就是将匹配成功的模式的某部分进行存储供以后使用这一能力。请回想一下，对一个正则表达式模式或部分模式两边添加圆括号将导致这部分表达式存储到一个临时缓冲区中。可以使用非捕获元字符 '?:'，'?='，or '?!' 来忽略对这部分正则表达式的保存。

所捕获的每个子匹配都按照在正则表达式模式中从左至右所遇到的内容存储。存储子匹配的缓冲区编号从 1 开始，连续编号直至最大 99 个子表达式。每个缓冲区都可以使用 '*n*' 访问，其中 *n* 为一个标识特定缓冲区的一位或两位十进制数。

向后引用一个最简单，最有用的应用是提供了确定文字中连续出现两个相同单词的位置的能力。请看下面的句子：

```
Is is the cost of of gasoline going up up?
```

根据所写内容，上面的句子明显存在单词多次重复的问题。如果能有一种方法无需查找每个单词的重复现象就能修改该句子就好了。下面的 JScript 正则表达式使用一个子表达式就可以实现这一功能。

```
/\b([a-z]+) \1\b/gi
```

等价的 VBScript 表达式为：

```
"\b([a-z]+) \1\b"
```

在这个示例中，子表达式就是圆括号之间的每一项。所捕获的表达式包括一个或多个字母字符，即由 '[a-z]+' 所指定的。该正则表达式的第二部分是对前面所捕获的子匹配的引用，也就是由附加表达式所匹配的第二次出现的单词。'\1'用来指定第一个子匹配。单词边界元字符确保只检测单独的单词。如果不这样，则诸如 "is issued" 或 "this is" 这样的短语都会被该表达式不正确地识别。

在 JScript 表达式中，正则表达式后面的全局标志 ('g') 表示该表达式将用来在输入字符串中查找尽可能多的匹配。大小写敏感性由表达式结束处的大小写敏感性标记 ('i') 指定。多行标记指定可能出现在换行符的两端的潜在匹配。对 VBScript 而言，在表达式中不能设置各种标记，但必须使用 **RegExp** 对象的属性来显式设置。

使用上面所示的正则表达式，下面的 JScript 代码可以使用子匹配信息，在一个文字字符串中将连续出现两次的相同单词替换为一个相同的单词：

```
var ss = "Is is the cost of of gasoline going up up?.\n";
var re = /\b([a-z]+) \1\b/gim;   //创建正则表达式样式。
var rv = ss.replace(re,"$1");   //用一个单词替代两个单词。
```

最接近的等价 VBScript 代码如下：

```
Dim ss, re, rv
ss = "Is is the cost of of gasoline going up up?." & vbNewLine
Set re = New RegExp
re.Pattern = "\b([a-z]+) \1\b"
re.Global = True
re.IgnoreCase = True
```

```
re.MultiLine = True
rv = re.Replace(ss,"$1")
```

请注意在 VBScript 代码中，全局、大小写敏感性以及多行标记都是使用 **RegExp** 对象的适当属性来设置的。

在 **replace** 方法中使用 **$1** 来引用所保存的第一个子匹配。如果有多个子匹配，则可以用 **$2**，**$3** 等继续引用。

向后引用的另一个用途是将一个通用资源指示符 (URI) 分解为组件部分。假定希望将下述的 URI 分解为协议 (ftp，http，etc)，域名地址以及页面/路径：

```
http://msdn.microsoft.com:80/scripting/default.htm
```

下面的正则表达式可以提供这个功能。对 JScript，为：

```
/(\w+):\/\/([^/:]+)(:\d*)?([^# ]*)/
```

对 VBScript 为：

```
"(\w+):\/\/([^/:]+)(:\d*)?([^# ]*)"
```

第一个附加子表达式是用来捕获该 web 地址的协议部分。该子表达式匹配位于一个冒号和两个正斜杠之前的任何单词。第二个附加子表达式捕获该地址的域名地址。该子表达式匹配不包括 '^'、'/' 或 ':' 字符的任何字符序列。第三个附加子表达式捕获网站端口号码，如果指定了该端口号。该子表达式匹配后跟一个冒号的零或多个数字。最后，第四个附加子表达式捕获由该 web 地址指定的路径以及\或者页面信息。该子表达式匹配一个和多个除'#' 或空格之外的字符。

将该正则表达式应用于上面所示的 URI 后，子匹配包含下述内容：

RegExp.$1 包含 "http"
RegExp.$2 包含 "msdn.microsoft.com"
RegExp.$3 包含 ":80"
RegExp.$4 包含 "/scripting/default.htm"