Regular Expression

正则表达式是由普通字符（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。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。

3.1 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。

3.2 非打印字符

字符	含义
/cx	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。

3.3 特殊字符

所谓特殊字符，就是一些有特殊含义的字符，如上面说的"*.txt"中的*，简单的说就是表示任何字符串的意思。如果要查找文件名中有＊的文件，则需要对＊进行转义，即在其前加一个/。ls /*.txt。正则表达式有以下特殊字符。

特别字符	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 /|。

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。

3.4 限定符

限定符用来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。有*或+或?或{n}或{n,}或{n,m}共 6 种。

*、+和?限定符都是贪婪的，因为它们会尽可能多的匹配文字，只有在它们的后面加上一个?就可以实现非贪婪或最小匹配。

正则表达式的限定符有：

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。

3.5 定位符

用来描述字符串或单词的边界，^和$分别指字符串的开始与结束，/b 描述单词的前或后边界，/B 表示非单词边界。不能对定位符使用限定符。

3.6 选择

用圆括号将所有选择项括起来，相邻的选择项之间用|分隔。但用圆括号会有一个副作用，是相关的匹配会被缓存，此时可用?:放在第一个选项前来消除这种副

作用。

其中?:是非捕获元之一，还有两个非捕获元是?=和?!，这两个还有更多的含义，前者为正向预查，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，后者为负向预查，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。

3.7 后向引用

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相关匹配存储到一个临时缓冲区中，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。

可以使用非捕获元字符 '?:', '?=', or '?!' 来忽略对相关匹配的保存。

4. 各种操作符的运算优先级

相同优先级的从左到右进行运算，不同优先级的运算先高后低。各种操作符的优先级从高到低如下：

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

5. 全部符号解释

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'。
/cx	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_]'。
/xn	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。
/un	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 (?)。

6. 部分例子

正则表达式	说明
//b([a-z]+) /1/b/gi	一个单词连续出现的位置
/(/w+):////([^/:]+)(:/d*)?([^#]*)/	将一个 URL 解析为协议、域、端口及相对路径
/^(?:Chapter|Section) [1-9][0-9]{0,1}$/	定位章节的位置
/[-a-z]/	A 至 z 共 26 个字母再加一个-号。
/ter/b/	可匹配 chapter，而不能 terminal
//Bapt/	可匹配 chapter，而不能 aptitude
/Windows(?=95 |98 |NT)/	可匹配 Windows95 或 Windows98 或 WindowsNT,当找到一个匹配后，从 Windows 后面开始进行下一次的检索匹配。

7. 正则表达式匹配规则

7.1 基本模式匹配

一切从最基本的开始。模式，是正规表达式最基本的元素，它们是一组描述字符串特征的字符。模式可以很简单，由普通的字符串组成，也可以非常复杂，往往用特殊的字符表示一个范围内的字符、重复出现，或表示上下文。例如：

^once

这个模式包含一个特殊的字符^，表示该模式只匹配那些以 once 开头的字符串。例如该模式与字符串"once upon a time"匹配，与"There once was a man from NewYork"不匹配。正如如^符号表示开头一样，$符号用来匹配那些以给定模式结

尾的字符串。

bucket$

这个模式与"Who kept all of this cash in a bucket"匹配，与"buckets"不匹配。字符^和$同时使用时，表示精确匹配（字符串与模式一样）。例如：

^bucket$

只匹配字符串"bucket"。如果一个模式不包括^和$，那么它与任何包含该模式的字符串匹配。例如：模式

once

与字符串

There once was a man from NewYork
Who kept all of his cash in a bucket.

是匹配的。

在该模式中的字母(o-n-c-e)是字面的字符，也就是说，他们表示该字母本身，数字也是一样的。其他一些稍微复杂的字符，如标点符号和白字符（空 格、制表符等），要用到转义序列。所有的转义序列都用反斜杠(/)打头。制表符的转义序列是：/t。所以如果我们要检测一个字符串是否以制表符开头，可以 用这个模式：

^/t

类似的，用/n 表示“新行”，/r 表示回车。其他的特殊符号，可以用在前面加上反斜杠，如反斜杠本身用//表示，句号.用/.表示，以此类推。

7.2 字符簇

在 INTERNET 的程序中，正规表达式通常用来验证用户的输入。当用户提交一个 FORM 以后，要判断输入的电话号码、地址、EMAIL 地址、信用卡号码等是否有效，用普通的基于字面的字符是不够的。

所以要用一种更自由的描述我们要的模式的办法，它就是字符簇。要建立一个

表示所有元音字符的字符簇，就把所有的元音字符放在一个方括号里：

[AaEeIiOoUu]

这个模式与任何元音字符匹配，但只能表示一个字符。用连字号可以表示一个字符的范围，如：

[a-z] //匹配所有的小写字母
[A-Z] //匹配所有的大写字母
[a-zA-Z] //匹配所有的字母
[0-9] //匹配所有的数字
[0-9/./-] //匹配所有的数字，句号和减号
[/f/r/t/n] //匹配所有的白字符

同样的，这些也只表示一个字符，这是一个非常重要的。如果要匹配一个由一个小写字母和一位数字组成的字符串，比如"z2"、"t6"或"g7"，但不是"ab2"、"r2d3"或"b52"的话，用这个模式：

^[a-z][0-9]$

尽管[a-z]代表 26 个字母的范围，但在这里它只能与第一个字符是小写字母的字符串匹配。

前面曾经提到^表示字符串的开头，但它还有另外一个含义。当在一组方括号里使用^是，它表示“非”或“排除”的意思，常常用来剔除某个字符。还用前面的例子，我们要求第一个字符不能是数字：

^[^0-9][0-9]$

这个模式与"&5"、"g7"及"-2"是匹配的，但与"12"、"66"是不匹配的。下面是几个排除特定字符的例子：

[^a-z] //除了小写字母以外的所有字符
[^/////^] //除了(/)(/)(^)之外的所有字符
[^/"/'] //除了双引号(")和单引号(')之外的所有字符

特殊字符"."(点，句号)在正规表达式中用来表示除了“新行”之外的所有字符。

所以模式"^.5$"与任何两个字符的、以数字 5 结尾和以其他非“新行”字符开头的字符串匹配。模式"."可以匹配任何字符串，除了空串和只包括一个“新行”的字符串。

PHP 的正规表达式有一些内置的通用字符簇，列表如下：

字符簇含义

[[:alpha:]] 任何字母
[[:digit:]] 任何数字
[[:alnum:]] 任何字母和数字
[[:space:]] 任何白字符
[[:upper:]] 任何大写字母
[[:lower:]] 任何小写字母
[[:punct:]] 任何标点符号
[[:xdigit:]] 任何 16 进制的数字，相当于[0-9a-fA-F]

7.3 确定重复出现

到现在为止，你已经知道如何去匹配一个字母或数字，但更多的情况下，可能要匹配一个单词或一组数字。一个单词有若干个字母组成，一组数字有若干个单数组成。跟在字符或字符簇后面的花括号({})用来确定前面的内容的重复出现的次数。

字符簇 含义
^[a-zA-Z_]$ 所有的字母和下划线
^[[:alpha:]]{3}$ 所有的 3 个字母的单词
^a$ 字母 a
^a{4}$ aaaa
^a{2,4}$ aa,aaa 或 aaaa
^a{1,3}$ a,aa 或 aaa
^a{2,}$ 包含多于两个 a 的字符串
^a{2,} 如：aardvark 和 aaab，但 apple 不行
a{2,} 如：baad 和 aaa，但 Nantucket 不行
/t{2} 两个制表符
.{2} 所有的两个字符

这些例子描述了花括号的三种不同的用法。一个数字，{x}的意思是“前面的字符或字符簇只出现 x 次”；一个数字加逗号，{x,}的意思是“前面的内容出 现 x 或

更多的次数”；两个用逗号分隔的数字，{x,y}表示“前面的内容至少出现 x 次，但不超过 y 次”。我们可以把模式扩展到更多的单词或数字：

^[a-zA-Z0-9_]{1,}$ //所有包含一个以上的字母、数字或下划线的字符串
^[0-9]{1,}$ //所有的正数
^/-{0,1}[0-9]{1,}$ //所有的整数
^/-{0,1}[0-9]{0,}/.{0,1}[0-9]{0,}$ //所有的小数

最后一个例子不太好理解，是吗？这么看吧：与所有以一个可选的负号(/-{0,1})开头(^)、跟着 0 个或更多的数字([0-9]{0,})、和一个 可选的小数点(/.{0,1})再跟上 0 个或多个数字([0-9]{0,})，并且没有其他任何东西($)。下面你将知道能够使用的更为简单的方法。

特殊字符"?"与{0,1}是相等的，它们都代表着：“0 个或 1 个前面的内容”或“前面的内容是可选的”。所以刚才的例子可以简化为：

^/-?[0-9]{0,}/.?[0-9]{0,}$

特殊字符"*"与{0,}是相等的，它们都代表着“0 个或多个前面的内容”。最后，字符"+"与 {1,}是相等的，表示“1 个或多个前面的内容”，所以上面的 4 个例子可以写成：

^[a-zA-Z0-9_]+$ //所有包含一个以上的字母、数字或下划线的字符串
^[0-9]+$ //所有的正数
^/-?[0-9]+$ //所有的整数
^/-?[0-9]*/.?[0-9]*$ //所有的小数

当然这并不能从技术上降低正规表达式的复杂性，但可以使它们更容易阅读。