← 正则表达式 – 运算符优先级 正则表达式 – 示例 →

正则表达式 - 匹配规则

基本模式匹配

一切从最基本的开始。模式，是正则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。

字符簇

在INTERNET的程序中，正则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)在正则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的正则表达式有一些内置的通用字符簇，列表如下：

字符簇	描述
[[:alpha:]]	任何字母

[[:digit:]]	任何数字
[[:alnum:]]	任何字母和数字
[[:space:]]	任何空白字符
[[: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]

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	描述
^[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** ；两个数字用逗号分隔的数字 `{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
^[1-9][0-9]{0,}$        // 所有的正整数
^\-{0,1}[0-9]{1,}$      // 所有的整数
^[-]?[0-9]+\.?[0-9]+$   // 所有的浮点数
```

最后一个例子不太好理解，是吗？这么看吧：以一个可选的负号（`[-]?`）开头（`^`）、跟着1个或更多的数字（`[0-9]+`）、和一个小数点（`\.`）再跟上1个或多个数字（`[0-9]+`），并且后面没有其他任何东西（`$`）。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]{1,}\.?[0-9]{1,}$
```

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
^[1-9][0-9]*$        // 所有的正整数
^\-?[0-9]+$          // 所有的整数
^[-]?[0-9]+(\.[0-9]+)?$ // 所有的浮点数
```

当然这并不能从技术上降低正则表达式的复杂性，但可以使它们更容易阅读。

← 正则表达式 – 运算符优先级　　正则表达式 – 示例 →