正则表达式简介

正则表达式是由一些具有特殊含义的字符组成的字符串，多用于查找、替换符合规则的字符串。在表单验证、Url映射等处都会经常用到。

一、元字符

元字符：即为有特定含义的字符，常见的元字符如下

常用的元字符

代码	说明
.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线或汉字
\s	匹配任意的空白符
\d	匹配数字
\b	匹配单词的开始或结束
^	匹配字符串的开始（在集合字符里[^a]表示非（不匹配）的意思
$	匹配字符串的结束

详解和示例：

（1）. 匹配任何任意字符　例如 .　可以匹配 1，n，*，+，- ,等

（2）\d\w\s　匹配第一个字符为数字，第二个字符为字母或数字、或下划线或汉字，第三字符为空格的字符串 例如：11　，2a　，1_

（3）^\d\d\d$ 匹配三个全部都为数字的字符串　例如：　123,456,789

还可以用于验证输入的字符串是否符合qq（身份证号）的验证：

例如：^\d{8}$ 匹配8位数字的qq号，^\d{15}&匹配15位均为数字的身份证号

（4）\bOlive\b 匹配单词Olive 例如：　I Love Oliver and Olive .这个时候返回的是Olive 而不是Oliver,因为\b....\b返回的匹配的单词

二、反义字符

反义字符：多用于查找除某个字符以外其他任意字符均可以的情况

常用的反义字符如下：

常用的反义字符

代码/语法	说明
\W	匹配任意不是字母，数字，下划线，汉字的字符
\S	匹配任意不是空白符的字符
\D	匹配任意非数字的字符
\B	匹配不是单词开头或结束的位置
[^x]	匹配除了x以外的任意字符
[^aeiou]	匹配除了aeiou这几个字母以外的任意字符

详解和示例：

（1）\W　匹配除字母、数字、下划线、汉字以为的字符形如 +，-，*

（2）\S　匹配除空格以外的任意字符形如：1，*，）

（3）[^abcde]匹配除abcde以为的其他字符 如 e，f，g，h

三、限定字符

限定字符多用于重复匹配次数

常用的限定字符如下

常用的限定符

代码/语法	**说明**
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次
{n,m}	重复n到m次

详解和示例：

（1）\d*　匹配重复0次或多次数字　例如:可能为空 或 任意数字 （2,3。。。。）

（2）\d+ 匹配重复1次或多次数字　例如:可能为1个或多个数字 1,23,234,2345，.....

（3）\d? 匹配重复次个或者一次数字 例如：可能为空或者任意的一个数字（1,2，。。。）

（4）\d{8}匹配重复8次数字　例如：123456768

（5）\d{4,}匹配重复至少4次数字 例如：1234,12345,124244,。。。。。

（6）^\d{8,11}$ 匹配重复8-11次数字 例如：12345678,123456789,1234567890,12345678901

四、转义字符

在实际的开发中，可能会遇到要比配元字符的情况，这个时候就需要进行字符转义，如元字符 . * \ 需要转换为\. * \\

例如： 需要匹配qq邮箱 \d{8,}+qq+\.+com 在这里的. 就需要加斜杠

五、字符分枝

字符分枝多用于满足不同情况的选择，用“|”将不同的条件分割开来，比如有些固定电话区号有三位，有些有四位，这个时候可以采用字符分枝

例如：\d{3}-\d{8}|\d{4}-\d{8} 可以匹配两种不同长度区号的固定电话

下边的IP地址正则表达式也有用到字符分枝

六、字符分组

字符分组多用于将多个字符重复，主要通过使用小括号()来进行分组

形如： (\d\w){3} 重复匹配3次（\d\w)

常用于表示IP地址 形如：((25[0-5]|2[0-4][0-9]|[0-1]\d\d)\.){3}(25[0-5]|2[0-4][0-9]|[0-1]\d\d)

解析：先把IP地址分为两部分一部分是123.123.123. 另一部分是123，又因Ip最大值为255，所以先使用分组，然后在组里边再进行选择，组里也有三部分，0-199,200-249,250-255，分别和上述的表达是对应，最后还要注意分组之后还要加上一个.，因为是元字符所以要转义故加上\. 然后再把这部分整体看做是一个组，重复三次，再加上仅有数字的一组也就是不带\.的那一组即可完成IP地址的校验

常用分组语法

用分组语法

分类	代码/语法	说明
捕获	(exp)	匹配exp,并捕获文本到自动命名的组里
	(?<name>exp)	匹配exp,并捕获文本到名称为name的组里，也可以写成(?'name'exp)
	(?:exp)	匹配exp,不捕获匹配的文本，也不给此分组分配组号
零宽断言	(?=exp)	匹配exp前面的位置
	(?<=exp)	匹配exp后面的位置
	(?!exp)	匹配后面跟的不是exp的位置
	(?<!exp)	匹配前面不是exp的位置
注释	(?#comment)	这种类型的分组不对正则表达式的处理产生任何影响，用于提供注释让人阅读

七、懒惰匹配和贪婪匹配

贪婪匹配：正则表达式中包含重复的限定符时，通常的行为是匹配**尽可能多**的字符。

懒惰匹配，有时候需要匹配**尽可能少**的字符。

例如： a.*b，它将会匹配最长的以a开始，以b结束的字符串。如果用它来搜索aabab的话，它会匹配整个字符串aabab。但是我们此时可能需要匹配的是ab这样的话就需要用到懒惰匹配了。懒惰匹配会匹配尽可能少的字符

常用的懒惰匹配限定符如下

懒惰限定符

代码/语法	说明
*?	重复任意次，但尽可能少重复
+?	重复1次或更多次，但尽可能少重复
??	重复0次或1次，但尽可能少重复
{n,m}?	重复n到m次，但尽可能少重复
{n,}?	重复n次以上，但尽可能少重复

八、后向引用

后向引用用于重复搜索前面某个分组匹配的文本。

使用小括号指定一个子表达式后，**匹配这个子表达式的文本**(也就是此分组捕获的内容)可以在表达式或其它程序中作进一步的处理。默认情况下，每个分组会自动拥有一个组号，规则是：从左向右，以分组的左括号为标志，第一个出现的分组的组号为1，第二个为2，以此类推

示例：\b(\w+)\b\s+\1\b可以用来匹配重复的单词，像go go, 或者kitty kitty。

这个表达式首先是一个单词，也就是单词开始处和结束处之间的多于一个的字母或数字(\b(\w+)\b)，这个单词会被捕获到编号为1的分组中，然后是1个或几个空白符(\s+)，最后是分组1中捕获的内容（也就是前面匹配的那个单词）(\1)。

你也可以自己指定子表达式的组名。要指定一个子表达式的组名，请使用这样的语法：(?<Word>\w+)(或者把尖括号换成'也行：(?'Word'\w+)),这样就把\w+的组名指定为Word了。要反向引用这个分组捕获的内容，你可以使用\k<Word>,所以上一个例子也可以写成这样：\b(?<Word>\w+)\b\s+\k<Word>\b

九、零宽断言

有时候需要查找某些匹配之前或之后的东西，这个时候就需要用到们像\b,^,$那样用于指定一个位置，这个位置应该满足一定的条件(即断言)，因此它们也被称为零宽断言

(?=exp)也叫零宽度正预测先行断言，它断言自身出现的位置的后面能匹配表达式exp。比如\b\w+(?=ing\b)，匹配以ing结尾的单词的前面部分(除了ing以外的部分)，如查找I'm singing while you're dancing.时，它会匹配sing和danc。

(?<=exp)也叫零宽度正回顾后发断言，它断言自身出现的位置的前面能匹配表达式exp。比如(?<=\bre)\w+\b会匹配以re开头的单词的后半部分(除了re以外的部分)，例如在查找reading a book时，它匹配ading。

十、其他语法

.NET常用的处理选项

名称	说明
IgnoreCase(忽略大小写)	匹配时不区分大小写。
Multiline(多行模式)	更改^和$的含义，使它们分别在任意一行的行首和行尾匹配，而不仅仅在整个字符串的开头和结尾匹配。(在此模式下,$的精确含意是:匹配\n之前的位置以及字符串结束前的位置.)
Singleline(单行模式)	更改.的含义，使它与每一个字符匹配（包括换行符\n）。
IgnorePatternWhitespace(忽略空白)	忽略表达式中的非转义空白并启用由#标记的注释。
ExplicitCapture(显式捕获)	仅捕获已被显式命名的组。

其他语法

代码/语法	说明
\a	报警字符(打印它的效果是电脑嘀一声)
\b	通常是单词分界位置，但如果在字符类里使用代表退格
\t	制表符，Tab
\r	回车
\v	竖向制表符
\f	换页符
\n	换行符
\e	Escape
\0nn	ASCII代码中八进制代码为nn的字符
\xnn	ASCII代码中十六进制代码为nn的字符
\unnnn	Unicode代码中十六进制代码为nnnn的字符
\cN	ASCII控制字符。比如\cC代表Ctrl+C

代码/语法	说明
\A	字符串开头(类似^，但不受处理多行选项的影响)
\Z	字符串结尾或行尾(不受处理多行选项的影响)
\z	字符串结尾(类似$，但不受处理多行选项的影响)
\G	当前搜索的开头
\p{name}	Unicode中命名为name的字符类，例如\p{IsGreek}
(?>exp)	贪婪子表达式
(?<x>-<y>exp)	平衡组
(?im-nsx:exp)	在子表达式exp中改变处理选项
(?im-nsx)	为表达式后面的部分改变处理选项
(?(exp)yes|no)	把exp当作零宽正向先行断言，如果在这个位置能匹配，使用yes作为此组的表达式；否则使用no
(?(exp)yes)	同上，只是使用空表达式作为no
(?(name)yes|no)	如果命名为name的组捕获到了内容，使用yes作为表达式；否则使用no
(?(name)yes)	同上，只是使用空表达式作为no

十一、常用的实用正则表达式整理（摘自自学编程网）

只能输入数字："^[0-9]*$"。

只能输入n位的数字："^"d{n}$"。

只能输入至少n位的数字："^"d{n,}$"。

只能输入m~n位的数字：。"^"d{m,n}$"

只能输入零和非零开头的数字："^(0|[1-9][0-9]*)$"。

只能输入有两位小数的正实数："^[0-9]+(.[0-9]{2})?$"。

只能输入有1~3位小数的正实数："^[0-9]+(.[0-9]{1,3})?$"。

只能输入非零的正整数："^"+?[1-9][0-9]*$"。

只能输入非零的负整数："^"-[1-9][]0-9"*$。

只能输入长度为3的字符："^.{3}$"。

只能输入由26个英文字母组成的字符串："^[A-Za-z]+$"。

只能输入由26个大写英文字母组成的字符串："^[A-Z]+$"。

只能输入由26个小写英文字母组成的字符串："^[a-z]+$"。

只能输入由数字和26个英文字母组成的字符串："^[A-Za-z0-9]+$"。

只能输入由数字、26个英文字母或者下划线组成的字符串："^"w+$"。

验证用户密码："^[a-zA-Z]"w{5,17}$"正确格式为：以字母开头，长度在6~18之间，只能包含字符、数字和下划线。

验证是否含有^%&',;=?$""等字符："[^%&',;=?$"x22]+"。

只能输入汉字："^["u4e00-"u9fa5]{0,}$"

验证Email地址："/^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(\.[a-zA-Z0-9_-])+/"。

验证InternetURL："^http://(["w-]+".)+["w-]+(/["w-./?%&=]*)?$"。

验证电话号码："^("(d{3,4}-)|"d{3.4}-)?"d{7,8}$"正确格式为："XXX-XXXXXXX"、"XXXX-XXXXXXXX"、"XXX-XXXXXXX"、"XXX-XXXXXXXX"、"XXXXXXX"和"XXXXXXXX"。

验证身份证号(15位或18位数字)："^"d{15}|"d{18}$"。

验证一年的12个月："^(0?[1-9]|1[0-2])$"正确格式为："01"～"09"和"1"～"12"。

验证一个月的31天："^((0?[1-9])|((1|2)[0-9])|30|31)$"正确格式为;"01"～"09"和"1"～"31"。

利用正则表达式限制网页表单里的文本框输入内容：

用正则表达式限制只能输入中文：onkeyup="value=value.replace(/[^"u4E00-"u9FA5] /g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^"u4E00-"u9FA5]/g,''))"

用正则表达式限制只能输入全角字符： onkeyup="value=value.replace(/[^"uFF00-"uFFFF]/g,'')" onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^"uFF00-"uFFFF]/g,''))"

用正则表达式限制只能输入数字：onkeyup="value=value.replace(/[^"d]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^"d]/g,''))"

用正则表达式限制只能输入数字和英文：onkeyup="value=value.replace(/["W]/g,'') "onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^"d]/g,''))"

得用正则表达式从URL地址中提取文件名的javascript程序，如下结果为page1

以下是引用片段：

```
s="http://www.9499.net/page1.htm"
s=s.replace(/(.*"/){0,}([^".]+).*/ig,"$2")
alert(s)
```

匹配双字节字符(包括汉字在内)：[^"x00-"xff]

应用：计算字符串的长度(一个双字节字符长度计2，ASCII字符计1)

以下是引用片段：

```
String.prototype.len=function(){return this.replace([^"x00-"xff]/g,"aa").length;}
```

匹配空行的正则表达式："n["s|]*"r

匹配HTML标记的正则表达式：/<(.*)>.*<"/"1>|<(.*) "/>/

匹配首尾空格的正则表达式：(^"s*)|("s*$)

以下是引用片段：

```
String.prototype.trim = function()
{
return this.replace(/(^"s*)|("s*$)/g, "");
}
```

利用正则表达式分解和转换IP地址：

下面是利用正则表达式匹配IP地址，并将IP地址转换成对应数值的Javascript程序：

以下是引用片段：

```
function IP2V(ip)
{
re=/("d+)".("d+)".("d+)".("d+)/g //匹配IP地址的正则表达式
if(re.test(ip))
{
return RegExp.$1*Math.pow(255,3))+RegExp.$2*Math.pow(255,2))+RegExp.$3*255+RegExp.$4*1
}
else
{
throw new Error("Not a valid IP address!")
}
}
```

不过上面的程序如果不用正则表达式，而直接用split函数来分解可能更简单，程序如下：

以下是引用片段：

```
var ip="10.100.20.168"
ip=ip.split(".")
alert("IP值是："+(ip[0]*255*255*255+ip[1]*255*255+ip[2]*255+ip[3]*1))
```