正则表达式整理

## 正则表达式的介绍

正则表达式：是用来匹配和处理文本的字符串；是用正则表达式语言创建的；用于解决文本替代问题。

正则表达式的作用：搜索和替换。

## 匹配单个字符

纯文本正则表达式匹配原始文本内对应字符。匹配单个任意字符：使用.（大多数不支持换行符）

多个结果匹配（如javascript，可以使用-g）

正则表达式区分大小写（在javascript中使用-i,忽略区分大小写）

对于很多元字符：\* . ? + $ ^ [ ] ( ) { } | \ /，他们具有特殊的含义，在使用的时候要是用转义字符。例如\\代表\，\|代表|

## 匹配一组字符中的一个

[ ]：匹配括号中所有字符中的一个。另外，在[ ]中使用元字符不要进行转义。[123+]表示匹配123+中的一个，且+不要转义。

对于连续的字符我们可以在[ ]中使用元字符-。例如，[1-9]：表示1,2,…,9中的一个；A-Z表示A,B,…,Z中的一个。

我们也可以对[ ]中的数据取非^，表示不匹配这组字符中任意一个。例如，[^1-9]表示匹配项不包含任意一个

## 匹配元字符

匹配转义字符：\* . ? + $ ^ [ ] ( ) { } | \ / （用\转义）

匹配空白字符：\b(回退)\f(换页)\n(换行)\r(回车)\t(制表)\v(垂直制表)

Windows使用\r\n文本行结束标签。Unix和Linux只用一个\n。

匹配特定的字符类别：

* 元字符\d等价[0-9]；元字符\D等价[^0-9]
* 元字符\w等价[a-zA-Z0-9]；\W等价[^a-zA-Z0-9]
* 元字符\s等价[\f\n\r\t\v]；\S等价[^\f\n\r\t\v]
* 十六进制\x ,八进制\0匹配其10数值对应的字符

POSIX字符类：

* [:alnum:]等价[a-zA-Z0-9]
* [:alpha:]等价[a-zA-Z]
* [:blank:]等价[\t ]
* [:cntrl:]等价ASCII控制字符，即0-31，127
* [:digit:]等价[0-9]
* [:graph:]等价[:print:]-空格
* [:lower:]等价[a-z]
* [:print:]等价任一可打印字符
* [:punct:]等价不是①④的
* [:space:]等价[^\f\n\r\t\v]
* [:upper:]等价[A-Z]
* [:xdigit:]等价[a-fA-F0-9]

## 重复匹配

匹配一个或多个：元字符+ ，用于字符和集合之后，（至少一个；不匹配零个字符的情况）。匹配+本身，用\+。在字符集合里，像.和+这样的元字符将被解释为普通字符，不需要转义。

匹配零个或多个：元字符\*

匹配零个或者一个：元字符？

使用{ }进行重复匹配：

* 精确匹配次数：{5}连续出现5次
* 重复匹配区间：{3,5}重复出现3-5次
* 至少出现次数：{n,}至少出现n次

注意：\*和+都是“贪婪型”元字符，匹配行为多多益善。此时，可使用?后缀将其转化为“懒惰型”版本。如“贪婪型”：\* + {n,} ，“懒惰型”：\*? +? {n,}?

文本：<b>hello</b><b>world</b>

正则表达式：<b>.\*</b>---------->匹配到： hello</b><b>world

正则表达式：<b>.\*?</b>--------->匹配到：hello

解析：贪婪型会一直匹配到文本末尾，而懒惰型碰到匹配项就停止。

## 位置匹配

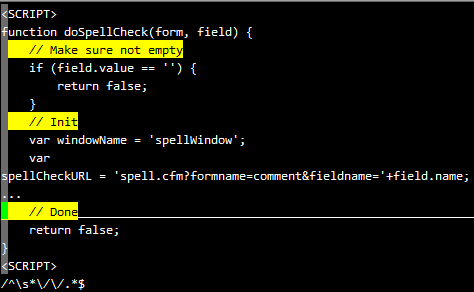
单词边界：\b，匹配的是位置。匹配一个这样的位置：\w和\W之间。另外\B的含义正好和\b相反。\bcat\b匹配单词cat

\< 匹配一个单词的开头 不是所有的编辑器都支持

\> 匹配一个单词的结束 不是所有的编辑器都支持

字符串边界字符：：①^定义字符串开头②$定义字符串结尾。

分行匹配模式：在分行匹配模式下，^ 不仅匹配正常的字符串开头，还将匹配行分隔符（换行符）后的开始位置（这个位置不可见）；类似地，$ 不仅匹配正常的字符串结尾，还将匹配行分隔符（换行符）后面的结束位置。



## 子表达式的使用()

子表达式必须用两个元字符和(和)括起来，称为一个分组。

子表达式的作用把某些元素独立起来使用。

|字符是正则表达式的或操作符，左右两边分别作以整体看待。

字表达式允许嵌套。

Ip地址：((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5])\.){3}((\d{1,2})|(1\d{2})|(2[0-4]\d)|(25[0-5])

## 回溯引用

回溯引用：模式的后半部分引用在前半部分中定义的子表达式。使用\1\2\3代表回溯引用第123个表达式（不同环境符号可能不同）。C#中match.Groups[1]对应着第一个匹配。  
回溯引用只能用来引用模式里的子表达式。  
回溯引用匹配通常从1开始计数，第0个匹配代表整个正则表达式。

例如：

<h1>Welcome to my page</H1>

Content is divided into twosections:<br>

<h2>Introduction</h2>

正则表达式：<[hH]([1-6])>.\*?</[hH]\1>

正则表达式适合用于复杂的替换，尤其是需要使用回溯引用的场合。  
大小写转换元字符： \E（结束\L或\U转换）  
\l（把下一字符转小写）  \L（把\L到\E间转小写）  
\u（把下一字符转大写）  \U（把\U到\E间转大写）

例如：

正则表达式：(<[hH]1>)(.\*?)(</[hH]1>) 替换：$1\U$2\E$3

## 前后查找

前后查找（lookaround），对某一位置的前后内容进行查找。其中的前后指模式与被查找文本的相对位置而言，左为前。

***向前查找***制定了一个必须匹配但不在结果中返回的模式。使用方式(?=)，向前（左）匹配，返回匹配值不含=后内容。例如：

* 文本：http:www.baidu.com
* 正则表达式： .+(:) //匹配http:但是不返回:

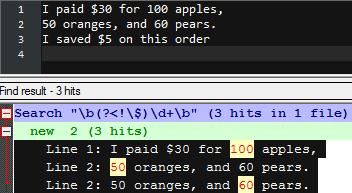
***向后查找***操作符：?<=,不消费?<=后面的内容。

* 向前向后的例子。截取title中的内容<title>hello</title>
* 正则表达式：(?<=<title>).\*(?=<title>)

负向前查找：向前查找不予给定模式匹配的文本，负向后查找：向后查找不予给定模式匹配文本。

***负向前查找***操作符：(?!)

***负向后查找***操作符：(?<!)



## 嵌入条件

回溯引用条件： 只在一个前面的表达式搜索取得成功的情况下才允许使用一个表达式。

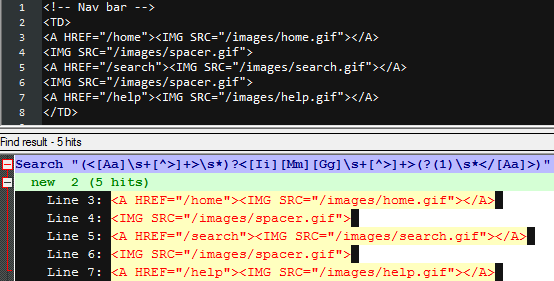
?(backreference)true-regex|false-regex

// ?，表明这是一个条件

//括号里的backreference是一个回溯引用，是子表达式标号

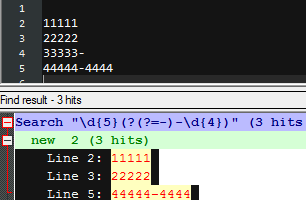
//true-regex是一个只有在backreference存在时才会被执行的子表达式

//false-regex是一个只有在backreference不存在时才会被执行的子表达式



**前后查找**条件：前后查找条件只在一个向前查找或向后查找操作取得成功的情况下才允许一个表达式被使用 。

语法：?(condition)true-regex|false-regex



只有?=向前匹配成功-，才会继续匹配\d{4}