

正则表达式参考

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。

本文详细地列出了能在正则表达式中使用，以匹配文本的各种字符。当你需要解释一个现有的正则表达式时，可以作为一个快捷的参考。更多详细内容，请参考：Francois Liger,Craig McQueen,Pal Wilton[刘乐亭 译] C#字符串和正则表达式参考手册 北京：清华大学出版社 2003.2

一. 匹配字符

字符类	匹配的字符	举 例
\d	从 0 - 9 的任一数字	\d\d 匹配 72,但不匹配 aa 或 7a
\D	任一非数字字符	\D\D\D 匹配 abc,但不匹配 123
\w	任一单词字符，包括 A-Z,a-z,0-9 和下划线	\w\w\w\w 匹配 Ab-2，但不匹配 Σ £\$%*或 Ab_@
\W	任一非单词字符	\W 匹配@，但不匹配 a
\s	任一空白字符，包括制表符，换行符，回车符，换页符和垂直制表符	匹配在 HTML,XML 和其他标准定义中的所有传统空白字符
\S	任一非空白字符	空白字符以外的任意字符,如 A%&g3; 等
.	任一字符	匹配除换行符以外的任意字符除非设置了 MultiLine 先项
[...]	括号中的任一字符	[abc]将匹配一个单字符,a,b 或 c. [a-z]将匹配从 a 到 z 的任一字符
[^...]	不在括号中的任一字符	[^abc]将匹配一个 a、b、c 之外的单字符,可以 a,b 或 A、B、C [a-z]将匹配不属于 a-z 的任一字符,但可以匹配所有的大写字母

二. 重复字符

重复字符	含义	举例
{n}	匹配前面的字符 n 次	x{2} 匹配 xx, 但不匹配 x 或 xxx
{n,}	匹配前面的字符至少 n 次	x{2,} 匹配 2 个或多个的 x, 如 xxx, xxx..
{n,m}	匹配前面的字符至少 n 次, 至多 m 次。 如果 n 为 0, 此参数为可选参数	x{2,4} 匹配 xx, xxx, xxxx, 但不匹配 xxxxx
?	匹配前面的字符 0 次或 1 次, 实质上也是可选的	x? 匹配 x 或零个 x
+	匹配前面的字符 0 次或多次	x+ 匹配 x 或 xx 或大于 0 的任意多个 x
*	匹配前面的字符 0 次或更多次	x* 匹配 0, 1 或更多个 x

三. 定位字符

定位字符	描述
^	随后的模式必须位于字符串的开始位置, 如果是一个多行字符串, 则必须位于行首。对于多行文本 (包含回车符的一个字符串) 来说, 需要设置多行标志
\$	前面的模式必须位于字符串的末端, 如果是一个多行字符串, 必须位于行尾
\A	前面的模式必须位于字符串的开始位置, 忽略多行标志
\z	前面的模式必须位于字符串的末端, 忽略多行标志
\Z	前面的模式必须位于字符串的末端, 或者位于一个换行符前
\b	匹配一个单词边界, 也就是一个单词字符和非单词字符中间的点。要记住一个单词字符是[a-zA-Z0-9]中的一个字符。位于一个单词的词首
\B	匹配一个非单词字符边界位置, 不是一个单词的词首

注: 定位字符可以应用于字符或组合, 放在字符串的左端或右端

四. 分组字符

分组字符	定义	举例
()	<p>此字符可以组合括号内模式所匹配的字符，它是一个捕获组，也就是说模式匹配的字符作为最终设置了 <code>ExplicitCapture</code> 选项——默认状态下字符不是匹配的一部分</p>	<p>输入字符串为：ABC1DEF2XY</p> <p>匹配 3 个从 A 到 Z 的字符和 1 个数字的正则表达式：<code>([A-Z]{3}\d)</code></p> <p>将产生两次匹配：Match 1=ABC1; Match 2=DEF2</p> <p>每次匹配对应一个组：Match1 的第一个组=ABC; Match2 的第 1 个组=DEF</p> <p>有了反向引用，就可以通过它在正则表达式中的编号以及 C# 和类 <code>Group</code>, <code>GroupCollection</code> 来访问组。如果设置了 <code>ExplicitCapture</code> 选项，就不能使用组所捕获的内容</p>
(?:)	<p>此字符可以组合括号内模式所匹配的字符，它是一个非捕获组，这意味着模式所匹配的字符将不作为一个组来捕获，但它构成了最终匹配结果的一部分。它基本上与上面的组类型相同，但设定了选项 <code>ExplicitCapture</code></p>	<p>输入字符串为：1A BB SA 1 C</p> <p>匹配一个数字或一个 A 到 Z 的字母，接着是任意单词字符的正则表达式为：<code>(?:\d [A-Z]\w)</code></p> <p>它将产生 3 次匹配：每 1 次匹配=1A；每 2 次匹配=BB；每 3 次匹配=SA</p> <p>但是没有组被捕获</p>
(?<name>)	<p>此选项组合括号内模式所匹配的字符，并用尖括号中指定的值为组命名。在正则表达式中，可以使用名称进行反向引用，而不必使用编号。即使不设置 <code>ExplicitCapture</code> 选项，它也是一个捕获组。这意味着反向引用可以利用组内匹配的字符，或者通过 <code>Group</code> 类访问</p>	<p>输入字符串为：Characters in Sienfeld included Jerry Seinfeld, Elaine Benes, Cosmo Kramer and George Costanza 能够匹配它们的姓名，并在一个组 <code>lastName</code> 中捕获姓的正则表达式为：<code>\b[A-Z][a-z]*(?<lastName>[A-Z][a-z]*)\b</code></p> <p>它产生了 4 次匹配：First Match=Jerry Seinfeld; Second Match=Elaine Benes; Third Match=Cosmo Kramer; Fourth Match=George Costanza</p> <p>每一次匹配都对应了一个 <code>lastName</code> 组：</p> <p>第 1 次匹配：lastName group=Seinfeld</p> <p>第 2 次匹配：lastName group=Benes</p> <p>第 3 次匹配：lastName group=Kramer</p> <p>第 4 次匹配：lastName group=Costanza</p> <p>不管是否设置了选项 <code>ExplicitCapture</code>，组都将被捕获</p>
(?=)	<p>正声明。声明的右侧必须是括号中指定的模式。此模式不构成最终匹配的一部分</p>	<p>正则表达式 <code>\S+(?=\.NET)</code> 要匹配的输入字符串为：The languages were Java, C#.NET, VB.NET, C, Jscript.NET, Pascal</p> <p>将产生如下匹配：)</p>

		<p>C#</p> <p>VB</p> <p>JScript</p>
(?!)	<p>负声明。它规定模式不能紧临着声明的右侧。此模式不构成最终匹配的一部分</p>	<p>\d{3}(?![A-Z])要匹配的输入字符串为：123A 456 789 111C</p> <p>将产生如下匹配：</p> <p>456</p> <p>789</p>
(?<=)	<p>反向正声明。声明的左侧必须为括号内的指定模式。此模式不构成最终匹配的一部分</p>	<p>正则表达式(?<=New)([A-Z][a-z]+)要匹配的输入字符串为：The following states,New Mexico,West Virginia,Washington,New England</p> <p>它将产生如下匹配：</p> <p>Mexico</p> <p>England</p>
(?<!)	<p>反向正声明。声明的左侧必须不是括号内的指定模式。此模式不构成最终匹配的一部分</p>	<p>正则表达式(?<!1)\d{2}([A-Z])要匹配的输入字符串如下：123A 456F 789C 111A</p> <p>它将实现如下匹配：</p> <p>56F</p> <p>89C</p>
(?>)	<p>非回溯组。防止 Regex 引擎回溯并且防止实现一次匹配</p>	<p>假设要匹配所有以“ing”结尾的单词。输入字符串如下：He was very trusing</p> <p>正则表达式为：.*ing</p> <p>它将实现一次匹配——单词 trusting。“.”匹配任意字符，当然也匹配“ing”。所以，Regex 引擎回溯一位并在第 2 个“t”停止，然后匹配指定的模式“ing”。但是，如果禁用回溯操作：(?>.*ing)ing</p> <p>它将实现 0 次匹配。“.”能匹配所有的字符，包括“ing”——不能匹配，从而匹配失败</p>

五. 决策字符

字符	描述	举例
<code>(?(regex)yes_regex no_regex)</code>	如果表达式 <code>regex</code> 匹配, 那么将试图匹配表达式 <code>yes</code> 。否则匹配表达式 <code>no</code> 。正则表达式 <code>no</code> 是可先参数。注意, 作出决策的模式宽度为 0。这意味着表达式 <code>yes</code> 或 <code>no</code> 将从与 <code>regex</code> 表达式相同的位置开始匹配	正则表达式 <code>(?(\d)dA A-Z)B</code> 要匹配的输入字符串为: 1A CB 3A 5C 3B 它实现的匹配是: 1A CB 3A
<code>(?(group name or number)yes_regex no_regex)</code>	如果组中的正则表达式实现了匹配, 那么试图匹配 <code>yes</code> 正则表达式。否则, 试图匹配正则表达式 <code>no</code> 。 <code>no</code> 是可先的参数	正则表达式 <code>(\d7)?-(?(1)\d\d[A-Z] [A-Z][A-Z])</code> 要匹配的输入字符串为: 77-77A 69-AA 57-B 它实现的匹配为: 77-77A -AA

注: 上面表中列出的字符强迫处理器执行一次 if-else 决策

六. 替换字符

字符	描述
<code>\$group</code>	用 <code>group</code> 指定的组号替换
<code>\${name}</code>	替换被一个 <code>(?<name>)</code> 组匹配的最后子串
<code>\$\$</code>	替换一个字符 <code>\$</code>
<code>\$&</code>	替换整个的匹配
<code>\$\$^</code>	替换输入字符串匹配之前的所有文本
<code>\$\$'</code>	替换输入字符串匹配之后的所有文本
<code>\$\$+</code>	替换最后捕获的组
<code>\$\$_</code>	替换整个的输入字符串

注: 以上为常用替换字符, 不全

七. 转义序列

字 符	描 述
\\	匹配字符 “\”
\\.	匹配字符 “.”
*	匹配字符 “*”
\\+	匹配字符 “+”
\\?	匹配字符 “?”
\\	匹配字符 “ ”
\\(匹配字符 “(”
\\)	匹配字符 “)”
\\{	匹配字符 “{”
\\}	匹配字符 “}”
\\^	匹配字符 “^”
\\\$	匹配字符 “\$”
\\n	匹配换行符
\\r	匹配回车符
\\t	匹配制表符
\\v	匹配垂直制表符
\\f	匹配换面符
\\nnn	匹配一个 8 进数字，nnn 指定的 ASCII 字符。如\\103 匹配大写的 C
\\xnn	匹配一个 16 进数字，nn 指定的 ASCII 字符。如\\x43 匹配大写的 C
\\unnnn	匹配由 4 位 16 进数字（由 nnnn 表示）指定的 Unicode 字符
\\cV	匹配一个控制字符，如\\cV 匹配 Ctrl-V

八. 选项标志

选项标志	名称
I	IgnoreCase
M	Multiline
N	ExplicitCapture
S	SingleLine
X	IgnorePatternWhitespace

注：选项本身的信作含义如下表所示：

标志	名称
IgnoreCase	使模式匹配不区分大小写。默认的选项是匹配区分大小写
RightToLeft	从右到左搜索输入字符串。默认是从左到右以符合英语等的阅读习惯，但不符合阿拉伯语或希伯来语的阅读习惯
None	不设置标志。这是默认选项
Multiline	指定^和\$可以匹配行首和行尾，以及字符串的开始和结尾。这意味着可以匹配每个用换行符分隔的行。但是，字符“.”仍然不匹替换行符
SingleLine	规定特殊字符“.”匹配任意的字符，包括换行符。默认情况下，特殊字符“.”不匹替换行符。通常与MultiLine选项一起使用
ECMAScript	ECMA(European Computer Manufacturer's Association,欧洲计算机生产商协会)已经定义了正则表达式应该如何实现，而且已经在ECMAScript规范中实现，这是一个基于标准的JavaScript。这个选项只能与IgnoreCase和MultiLine标志一起使用。与其它任何标志一起使用，ECMAScript都将产生异常
IgnorePatternWhitespace	此选项从使用的正则表达式模式中删除所有非转义空白字符。它使表达式能跨越多行文本，但必须确保对模式中所有的空白进行转义。如果设置了此选项，还可以使用“#”字符来注释正则表达式
Complied	它把正则表达式编译为更接近机器代码的代码。这样速度快，但不允许对它进行任何修改