《RegExp正则表达式》Notes

@Seymour0314 来源菜鸟教程 2023-04-05

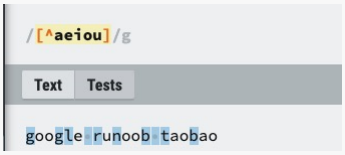
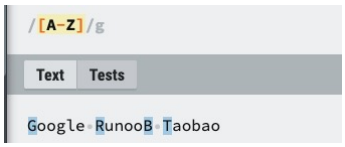
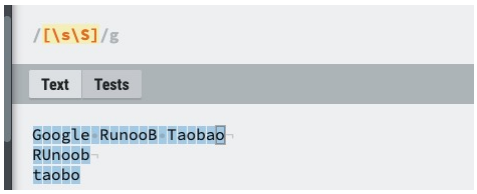
第一章 语法

普通字符

> 普通字符包括没有显式指定为元字符的所有可打印和不可打印字符。这包括所有大写和小写字母、所有数字、所有标点符号和一些其他符号。

- []
- [^]
- [-]
- .
- \s 与 \S
- \w

字符	描述
[ABC]	匹配 **[...]** 中的所有字符，例如 **[aeiou]** 匹配字符串 "google runoob taobao" 中所有的 e o u a 字母。 

字符	描述
[^ABC]	匹配除了 **[...]** 中字符的所有字符，例如 **[^aeiou]** 匹配字符串 "google runoob taobao" 中除了 e o u a 字母的所有字母。 
[A-Z]	[A-Z] 表示一个区间，匹配所有大写字母，[a-z] 表示所有小写字母。 
.	匹配除换行符（\n、\r）之外的任何单个字符，相等于 `[^\n\r]`。 
[\s\S]	匹配所有。\s 是匹配所有空白符，包括换行，\S 非空白符，不包括换行。 
\w	匹配字母、数字、下划线。等价于 [A-Za-z0-9_] 

非打印字符

非打印字符也可以是正则表达式的组成部分。下表列出了表示非打印字符的转义序列：

字符	描述
\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]。注意 Unicode 正则表达式会匹配全角空格符。
\S	匹配任何非空白字符。等价于 `[^ \f\n\r\t\v]`。

字符	描述
\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。

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 **runoo*b** 中的 *****，简单的说就是表示任何字符串的意思。如果要查找字符串中的 ***** 符号，则需要对 ***** 进行转义，即在其前加一个 `\`，**runo*ob** 匹配字符串 **runo*ob**。

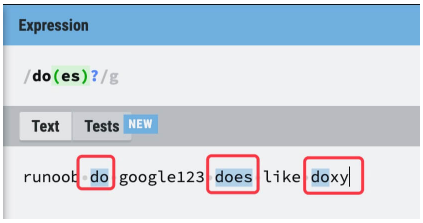
许多元字符要求在试图匹配它们时特别对待。若要匹配这些特殊字符，必须首先使字符"转义"，即，将反斜杠字符** 放在它们前面。下表列出了正则表达式中的特殊字符：

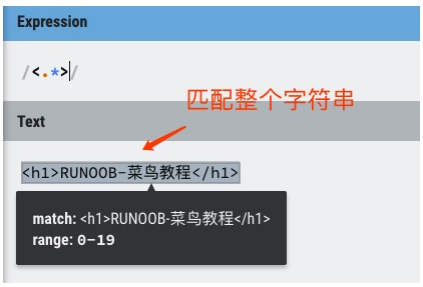
特别字符	描述
$	匹配输入字符串的结尾位置。如果设置了 RegExp 对象的 Multiline 属性，则 `$` 也匹配 '\n' 或 '\r'。要匹配 `$` 字符本身，请使用 $。
( )	标记一个子表达式的开始和结束位置。子表达式可以获取供以后使用。要匹配这些字符，请使用 (和)。
*	匹配前面的子表达式零次或多次。要匹配 * 字符，请使用 *。
+	匹配前面的子表达式一次或多次。要匹配 + 字符，请使用 +。
.	匹配除换行符 \n 之外的任何单字符。要匹配 . ，请使用 . 。
[	标记一个中括号表达式的开始。要匹配 [，请使用 [。
?	匹配前面的子表达式零次或一次，或指明一个非贪婪限定符。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，当该符号在方括号表达式中使用时，表示不接受该方括号表达式中的字符集合。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 |。

限定符

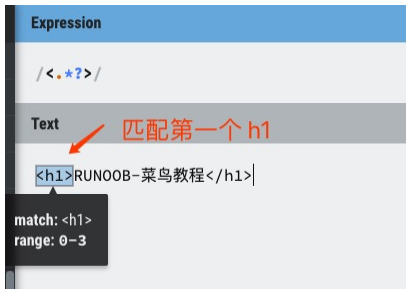
限定符用来指定正则表达式的一个给定组件必须要出现多少次才能满足匹配。有 ***** 或 **+** 或 **?** 或 **{n}** 或 **{n,}** 或 **{n,m}** 共6种。

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"** 、 **"does"**、 **"doxy"** 中的 **"do"** 和 **"does"**。? 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。

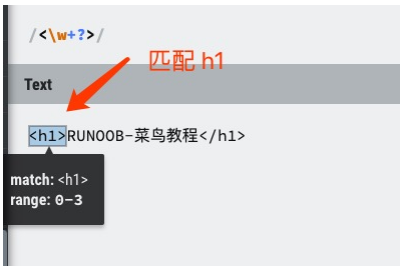
> `*` 和 `+` 限定符都是贪婪的，因为它们会尽可能多的匹配文字，只有在它们的后面加上一个 `?` 就可以实现非贪婪或最小匹配。
>
> 例如，您可能搜索 HTML 文档，以查找在 **h1** 标签内的内容。HTML 代码如下：
>
> ```
> 1 <h1>RUNOOB-菜鸟教程</h1>
> ```
>
> **贪婪：**下面的表达式匹配从开始小于符号 (<) 到关闭 h1 标记的大于符号 (>) 之间的所有内容。
>
> ```
> 1 /<.*>/
> ```
>
> 
>
>
> **非贪婪：**如果您只需要匹配开始和结束 h1 标签，下面的非贪婪表达式只匹配

```
/<.*?>/
```



也可以使用以下正则表达式来匹配 h1 标签，表达式则是：

```
/<\w+?>/
```



通过在 *、+ 或 ? 限定符之后放置 ?，该表达式从"贪婪"表达式转换为"非贪婪"表达式或者最小匹配。

定位符

定位符使您能够将正则表达式固定到行首或行尾。

它们还使您能够创建这样的正则表达式，这些正则表达式出现在一个单词内、在一个单词的开头或者一个单词的结尾。

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。

正则表达式的定位符有：

字符	描述
^	匹配输入字符串开始的位置。如果设置了 RegExp 对象的 Multiline 属性，^ 还会与 \n 或 \r 之后的位置匹配。
$	匹配输入字符串结尾的位置。如果设置了 RegExp 对象的 Multiline 属性，$ 还会与 \n 或 \r 之前的位置匹配。
\b	匹配一个单词边界，即字与空格间的位置。
\B	非单词边界匹配。

注意：不能将限定符与定位符一起使用。由于在紧靠换行或者单词边界的前面或后面不能有一个以上位置，因此不允许诸如 **^*** 之类的表达式。

若要匹配一行文本开始处的文本，请在正则表达式的开始使用 **^** 字符。不要将 **^** 的这种用法与中括号表达式内的用法混淆。

若要匹配一行文本的结束处的文本，请在正则表达式的结束处使用 **$** 字符。

若要在搜索章节标题时使用定位点，下面的正则表达式匹配一个章节标题，该标题只包含两个尾随数字，并且出现在行首：

```
/^Chapter [1-9][0-9]{0,1}/
```

真正的章节标题不仅出现行的开始处，而且它还是该行中仅有的文本。它既出现在行首又出现在同一行的结尾。下面的表达式能确保指定的匹配只匹配章节而不匹配交叉引用。通过创建只匹配一行文本的开始和结尾的正则表达式，就可做到这一点。

```
/^Chapter [1-9][0-9]{0,1}$/
```

匹配单词边界稍有不同，但向正则表达式添加了很重要的能力。单词边界是单词和空格之间的位置。非单词边界是任何其他位置。下面的表达式匹配单词 Chapter 的开头三个字符，因为这三个字符出现在单词边界后面：

```
/\bCha/
```

\b 字符的位置是非常重要的。如果它位于要匹配的字符串的开始，它在单词的开始处查找匹配项。如果它位于字符串的结尾，它在单词的结尾处查找匹配项。例如，下面的表达式匹配单词 Chapter 中的字符串 ter，因为它出现在单词边界的前面：

```
/ter\b/
```

下面的表达式匹配 Chapter 中的字符串 apt，但不匹配 aptitude 中的字符串 apt：

```
/\Bapt/
```

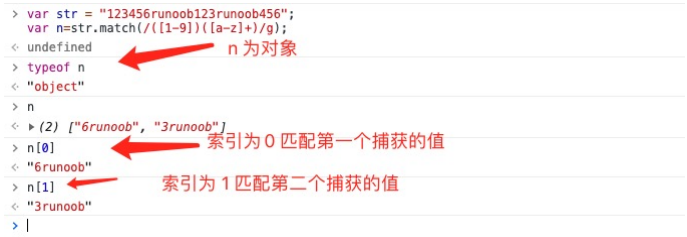
字符串 apt 出现在单词 Chapter 中的非单词边界处，但出现在单词 aptitude 中的单词边界处。对于 **\B** 非单词边界运算符，不可以匹配单词的开头或结尾，如果是下面的表达式，就不匹配 Chapter 中的 Cha：

```
\BCha
```

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。

() 表示捕获分组，**()** 会把每个分组里的匹配的值保存起来， 多个匹配值可以通过数字 n 来查看(**n** 是一个数字，表示第 n 个捕获组的内容)。



> 但用圆括号会有一个副作用，使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。

?=、?<=、?!、?<! 区别

exp1(?=exp2)：查找 exp2 前面的 exp1。

(?<=exp2)exp1：查找 exp2 后面的 exp1。

exp1(?!exp2)：查找后面不是 exp2 的 exp1。

(?<!exp2)exp1：查找前面不是 exp2 的 exp1。

反向引用

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到右出现的顺序存储。缓冲区编号从 1 开始，最多可存储 99 个捕获的子表达式。

- 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 **?:**、**?=** 或 **?!** 来重写捕获，忽略对相关匹配的保存。

反向引用的最简单的、最有用的应用之一，是提供查找文本中两个相同的相邻单词的匹配项的能力。以下面的句子为例：

```
Is is the cost of of gasoline going up up?
```

上面的句子很显然有多个重复的单词。如果能设计一种方法定位该句子，而不必查找每个单词的重复出现，那该有多好。下面的正则表达式使用单个子表达式来实现这一点：

查找重复的单词：

```
var str = "Is is the cost of of gasoline going up up";
var patt1 = /\b([a-z]+) \1\b/igm;
document.write(str.match(patt1));//Is is,of of,up up
```

> 捕获的表达式，正如 **[a-z]+** 指定的，包括一个或多个字母。正则表达式的第二部分是对以前捕获的子匹配项的引用，即，单词的第二个匹配项正好由括号表达式匹配。**\1** 指定第一个子匹配项。
>
> 单词边界元字符确保只检测整个单词。否则，诸如 "is issued" 或 "this is" 之类的词组将不能正确地被此表达式识别。

> 正则表达式后面的全局标记 **g** 指定将该表达式应用到输入字符串中能够查找到的尽可能多的匹配。
>
> 表达式的结尾处的不区分大小写 **i** 标记指定不区分大小写。
>
> 多行标记 **m** 指定换行符的两边可能出现潜在的匹配。

反向引用还可以将通用资源指示符 (URI) 分解为其组件。假定您想将下面的 URI 分解为协议（ftp、http 等等）、域地址和页/路径：

```
https://www.runoob.com:80/html/html-tutorial.html
```

输出所有匹配的数据：

```
var str = "https://www.runoob.com:80/html/html-tutorial.html";
var patt1 = /(\w+):\/\/([^/:]+)(:\d*)?([^# ]*)/;
arr = str.match(patt1);
for (var i = 0; i < arr.length ; i++) {
    document.write(arr[i]);
    document.write("<br>");
}
```

https://www.runoob.com:80/html/html-tutorial.html
https
www.runoob.com
:80
/html/html-tutorial.html

> 第一个括号子表达式捕获 Web 地址的协议部分。该子表达式匹配在冒号和两个正斜杠前面的任何单词。
>
> 第二个括号子表达式捕获地址的域地址部分。子表达式匹配非 **:** 和 **/** 之后的一个或多个字符。
>
> 第三个括号子表达式捕获端口号（如果指定了的话）。该子表达式匹配冒号后面的零个或多个数字。只能重复一次该子表达式。
>
> 最后，第四个括号子表达式捕获 Web 地址指定的路径和 / 或页信息。该子表达式能匹配不包括 # 或空格字符的任何字符序列。
>
> 将正则表达式应用到上面的 URI，各子匹配项包含下面的内容：
>
> - 第一个括号子表达式包含 `https`
> - 第二个括号子表达式包含 `www.runoob.com`
> - 第三个括号子表达式包含 `:80`
> - 第四个括号子表达式包含 `/html/html-tutorial.html`

运算符优先级

正则表达式从左到右进行计算，并遵循优先级顺序，这与算术表达式非常类似。

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。下表从最高到最低说明了各种正则表达式运算符的优先级顺序：

运算符	描述
\	转义符
(), (?:), (?=), []	圆括号和方括号
*, +, ?, {n}, {n,}, {n,m}	限定符
^, $, \任何元字符、任何字符	定位点和序列（即：位置和顺序）
|	替换，"或"操作 字符具有高于替换运算符的优先级，使得"m|food"匹配"m"或"food"。若要匹配"mood"或"food"，请使用括号创建子表达式，从而产生"(m|f)ood"。

第二章 修饰符（标记）

标记也称为修饰符，正则表达式的标记用于指定额外的匹配策略。

标记不写在正则表达式里，标记位于表达式之外，格式如下：

```
/pattern/flags
```

下表列出了正则表达式常用的修饰符：

修饰符	含义	描述
i	ignore - 不区分大小写	将匹配设置为不区分大小写，搜索时不区分大小写: A 和 a 没有区别。
g	global - 全局匹配	查找所有的匹配项。
m	multi line - 多行匹配	使边界字符 ^ 和 $ 匹配每一行的开头和结尾，记住是多行，而不是整个字符串的开头和结尾。

修饰符	含义	描述
s	特殊字符圆点 . 中包含换行符 **\n**	默认情况下的圆点 . 是匹配除换行符 **\n** 之外的任何字符，加上 **s** 修饰符之后, . 中包含换行符 \n。

第三章 元字符（总览）

下表包含了元字符的完整列表以及它们在正则表达式上下文中的行为：

字符	描述

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" 。? 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'。
.	匹配除换行符（\n、\r）之外的任何单个字符。要匹配包括 '\n' 在内的任何字符，请使用像"**(.|\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。

字符	描述
(?=pattern)	正向肯定预查（look ahead positive assert），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"Windows2000"中的"Windows"，但不能匹配"Windows3.1"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)	正向否定预查(negative assert)，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"Windows3.1"中的"Windows"，但不能匹配"Windows2000"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)	反向(look behind)肯定预查，与正向肯定预查类似，只是方向相反。例如，"`(?<=95|98|NT|2000)Windows`"能匹配"`2000Windows`"中的"`Windows`"，但不能匹配"`3.1Windows`"中的"`Windows`"。
(?<!pattern)	反向否定预查，与正向否定预查类似，只是方向相反。例如"`(?<!95|98|NT|2000)Windows`"能匹配"`3.1Windows`"中的"`Windows`"，但不能匹配"`2000Windows`"中的"`Windows`"。
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'、'l'、'i'、'n'。
[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。

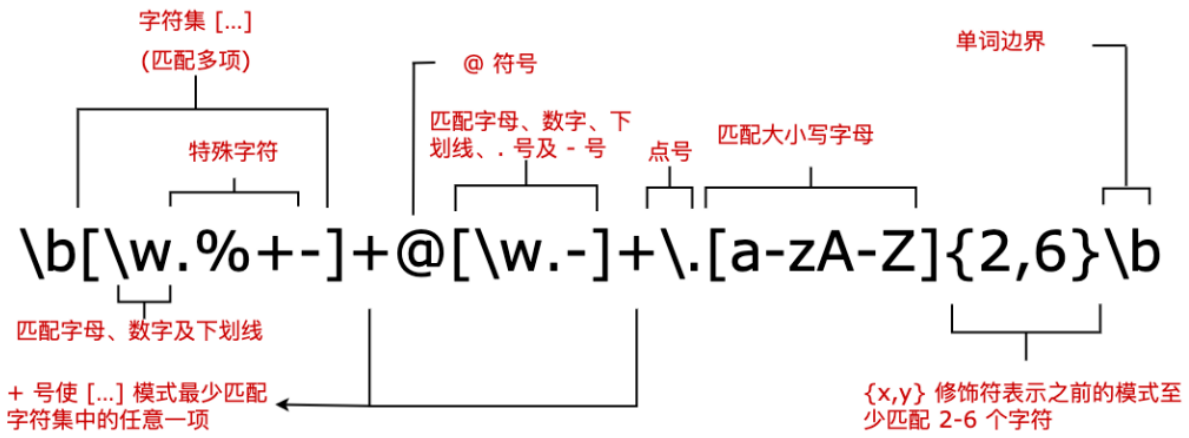
字符	描述
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\r	匹配一个回车符。等价于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\v	匹配一个垂直制表符。等价于 \x0b 和 \cK。
\w	匹配字母、数字、下划线。等价于'[A-Za-z0-9_]'。
\W	匹配非字母、数字、下划线。等价于 '[^A-Za-z0-9_]'。
\xn	匹配 n，其中 n 为十六进制转义值。十六进制转义值必须为确定的两个数字长。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 (?)。

分析一个匹配邮箱的正则表达式

```
var str = "abcd test@runoob.com 1234";
var patt1 = /\b[\w.%+-]+@[\w.-]+\.[a-zA-Z]{2,6}\b/g;
document.write(str.match(patt1));
```

test@runoob.com

匹配邮箱的正则表达式解析



测试匹配：abcd test@runoob.com 1234

第四章 示例

正则表达式	描述
`hello`	匹配 {hello}
`gray|grey`	匹配 {gray, grey}
`gr(a|e)y`	匹配 {gray, grey}
`gr[ae]y`	匹配 {gray, grey}
`b[aeiou]bble`	匹配 {babble, bebble, bibble, bobble, bubble}
`[b-chm-pP]at|ot`	匹配 {bat, cat, hat, mat, nat, oat, pat, Pat, ot}

正则表达式	描述
`colou?r`	匹配 {color, colour}
`rege(x(es)?|xps?)`	匹配 {regex, regexes, regexp, regexps}
`go*gle`	匹配 {ggle, gogle, google, gooogle, goooogle, ...}
`go+gle`	匹配 {gogle, google, gooogle, goooogle, ...}
`g(oog)+le`	匹配 {google, googoogle, googoogoogle, googoogoogoogle, ...}
`z{3}`	匹配 {zzz}
`z{3,6}`	匹配 {zzz, zzzz, zzzzz, zzzzzz}
`z{3,}`	匹配 {zzz, zzzz, zzzzz, ...}
`[Bb]rainf\*\*k`	匹配 {Brainf**k, brainf**k}
`\d`	匹配 {0,1,2,3,4,5,6,7,8,9}
`1\d{10}`	匹配 11 个数字，以 1 开头
`[2-9]|[12]\d|3[0-6]`	匹配 2 到 36 范围内的整数
`Hello\nworld`	匹配 Hello 后跟换行符，后跟 world
`\d+(\.\d\d)?`	包含一个正整数或包含两位小数位的浮点数。
`[^*@#]`	排除 *、@ 、# 三个特色符号
`//[^\r\n]*[\r\n]`	匹配 **//** 开头的注释
`^dog`	匹配以 "dog" 开始
`dog$`	匹配以 "dog" 结尾
`^dog$`	is exactly "dog"

正则表达式	描述
`/\b([a-z]+) \1\b/gi`	一个单词连续出现的位置。
`/(\w+):\/\/([^/:]+)(:\d*)?([^# ]*)/`	匹配一个 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，而不能匹配 terminal。

正则表达式	描述
`/\Bapt/`	可匹配 chapter，而不能匹配 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。
`/^\s*$/`	匹配空行。
`/\d{2}-\d{5}/`	验证由两位数字、一个连字符再加 5 位数字组成的 ID 号。
`<[a-zA-Z]+.*?>([\s\S]*?)</[a-zA-Z]*?>`	匹配 HTML 标记。

在线测试网站

https://c.runoob.com/front-end/854/