RegExp

> Java RegExp https://www.runoob.com/java/java-regular-expressions.html
> RegExp语法 https://www.runoob.com/regexp/regexp-syntax.html

语法

字符	说明
\	将下一字符标记为特殊字符、文本、反向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"\"，"\\("匹配"("。
^	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。
.	匹配除"\r\n"之外的任何单个字符。若要匹配包括"\r\n"在内的任意字符，请使用诸如"[\s\S]"之类的模式。
*	{0,}
+	{1,}

?	{0,1} 当此字符紧随任何其他限定符（*、+、?、{*n*}、{*n*,}、{*n*,*m*}）之后时，匹配模式是"非贪心的"。"非贪心的"模式匹配搜索到的、尽可能短的字符串，而默认的"贪心的"模式匹配搜索到的、尽可能长的字符串。例如，在字符串"oooo"中，"o+?"只匹配单个"o"，而"o+"匹配所有"o" **`*` `+` 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** 通过在 `*` `+` `?` 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
{	标记**限定符**表达式的开始。要匹配 {，请使用 \{。
{*n*}	n个
{*n*,}	>=n个
{*n*,*m*}	n~m个 不能有空格
( )	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 \(和 \)。
(*pattern*)	匹配 *pattern* 并捕获该匹配的子表达式。可以使用 **$0...$9** 属性从结果"匹配"集合中检索捕获的匹配。若要匹配括号字符 ( )，请使用"\("或者"\)"。
(?:*pattern*)	匹配 *pattern* 但不捕获该匹配的子表达式，即它是一个非捕获匹配，不存储供以后使用的匹配。这对于用"or"字符 (|) 组合模式部件的情况很有用。例如，'industr(?:y|ies) 是比 'industry|industries' 更经济的表达式。
(?=*pattern*)	执行正向预测先行搜索的子表达式，该表达式匹配处于匹配 *pattern* 的字符串的起始点的字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?=95|98|NT|2000)' 匹配"Windows 2000"中的"Windows"，但不匹配"Windows 3.1"中的"Windows"。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。
(?!*pattern*)	执行反向预测先行搜索的子表达式，该表达式匹配不处于匹配 *pattern* 的字符串的起始点的搜索字符串。它是一个非捕获匹配，即不能捕获供以后使用的匹配。例如，'Windows (?!95|98|NT|2000)' 匹配"Windows 3.1"中的 "Windows"，但不匹配"Windows 2000"中的"Windows"。预测先行不占用字符，即发生匹配后，下一匹配的搜索紧随上一匹配之后，而不是在组成预测先行的字符后。
x|*y*	匹配 *x* 或 *y*。例如，'z|food' 匹配"z"或"food"。'(z|f)ood' 匹配"zood"或"food"。
[	标记一个**中括号表达式**的开始。要匹配 [，请使用 \[。
[*xyz*]	字符集。匹配包含的任一字符。例如，"[abc]"匹配"plain"中的"a"。
[^*xyz*]	反向字符集。匹配未包含的任何字符。例如，"[^abc]"匹配"plain"中"p"，"l"，"i"，"n"。

[*a–z*]	字符范围。匹配指定范围内的任何字符。例如，"[a–z]"匹配"a"到"z"范围内的任何小写字母。
[^*a–z*]	反向范围字符。匹配不在指定的范围内的任何字符。例如，"[^a–z]"匹配任何不在"a"到"z"范围内的任何字符。
\b	匹配一个字边界，即字与空格间的位置。例如，"er\b"匹配"never"中的"er"，但不匹配"verb"中的"er"。
\B	非字边界匹配。"er\B"匹配"verb"中的"er"，但不匹配"never"中的"er"。
\c*x*	匹配 *x* 指示的控制字符。例如，\cM 匹配 Control–M 或回车符。*x* 的值必须在 A–Z 或 a–z 之间。如果不是这样，则假定 c 就是"c"字符本身。
\d	数字字符匹配。等效于 [0–9]。
\D	非数字字符匹配。等效于 [^0–9]。
\f	换页符匹配。等效于 \x0c 和 \cL。
\n	换行符匹配。等效于 \x0a 和 \cJ。
\r	匹配一个回车符。等效于 \x0d 和 \cM。
\s	匹配任何空白字符，包括空格、制表符、换页符等。与 [\f\n\r\t\v] 等效。
\S	匹配任何非空白字符。与 [^ \f\n\r\t\v] 等效。
\t	制表符匹配。与 \x09 和 \cI 等效。
\v	垂直制表符匹配。与 \x0b 和 \cK 等效。
\w	匹配任何字类字符，包括下划线。与"[A–Za–z0–9_]"等效。
\W	与任何非单词字符匹配。与"[^A–Za–z0–9_]"等效。
\x*n*	匹配 *n*，此处的 *n* 是一个十六进制转义码。十六进制转义码必须正好是两位数长。例如，"\x41"匹配"A"。"\x041"与"\x04"&"1"等效。允许在正则表达式中使用 ASCII 代码。
num	匹配 *num*，此处的 *num* 是一个正整数。到捕获匹配的反向引用。例如，"(.)\1"匹配两个连续的相同字符。
n	标识一个八进制转义码或反向引用。如果 *n* 前面至少有 *n* 个捕获子表达式，那么 *n* 是反向引用。否则，如果 *n* 是八进制数 (0–7)，那么 *n* 是八进制转义码。
nm	标识一个八进制转义码或反向引用。如果 *nm* 前面至少有 *nm* 个捕获子表达式，那么 *nm* 是反向引用。如果 *nm*前面至少有 *n* 个捕获，则 *n* 是反向引用，后面跟有字符 *m*。如果两种前面的情况都不存在，则 *nm* 匹配八进制值 *nm*，其中 *n* 和 *m* 是八进制数字 (0–7)。

\nml	当 *n* 是八进制数 (0–3)，*m* 和 *l* 是八进制数 (0–7) 时，匹配八进制转义码 *nml*。
\u*n*	匹配 *n*，其中 *n* 是以四位十六进制数表示的 Unicode 字符。例如，\u00A9 匹配版权符号 (©)。

普通字符

字符	描述
[ABC]	匹配所有ABC
[^ABC]	匹配所有除ABC
[A-Z]	匹配A-Z区间
.	匹配除换行符（\n、\r）之外的任何单个字符
[\s\S]	匹配所有 \s 匹配所有空白符(含换行) \S 非空白符(不含换行)
\w	匹配字母、数字、下划线 = [A-Za-z0-9_]

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 \(和 \)。
[	标记一个中括号表达式的开始。要匹配 [，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

符号	匹配次数
*	{0,}
+	{1,}
?	{0,1}
{n}	n个
{n,}	>=n个
{n,m}	n~m个

`*` `+` 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

通过在 `*` `+` `?` 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

eg.

从字符串 str 中提取数字部分的内容(匹配一次):

```
var str = "abc123def";
var patt1 = /[0-9]+/;
document.write(str.match(patt1));
```

以下标记的文本是获得的匹配的表达式:

`123`

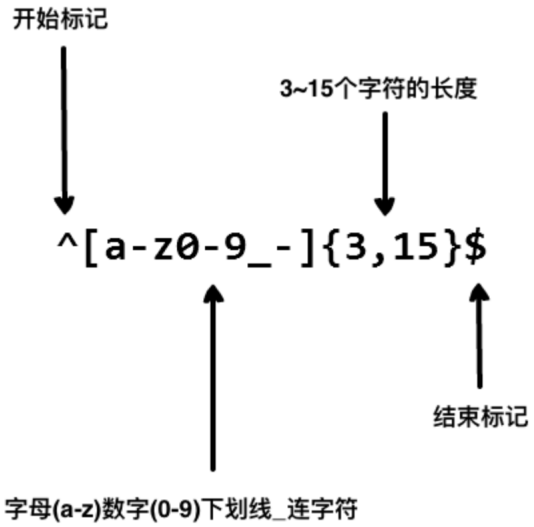
匹配以数字开头，并以 abc 结尾的字符串。

```
var str = "123abc";
var patt1 = /^[0-9]+abc$/;
document.write(str.match(patt1));
```

以下标记的文本是获得的匹配的表达式:

`123abc`

我们在写用户注册表单时，只允许用户名包含字符、数字、下划线和连接字符(-)，并设置用户名的长度，我们就可以使用以下正则表达式来设定。

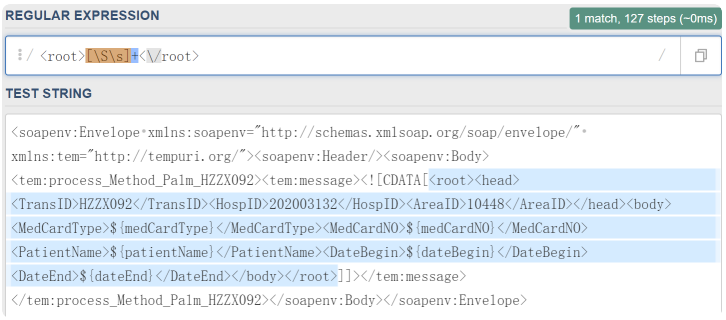


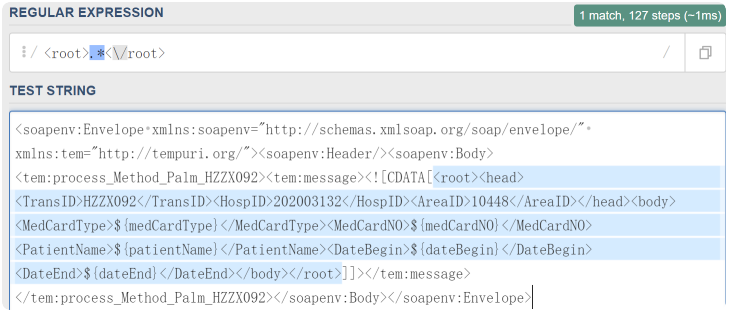
以上的正则表达式可以匹配 **runoob、runoob1、run-oob、run_oob**， 但不匹配 **ru**，因为它包含的字母太短了，小于 3 个无法匹配。也不匹配 **runoob$**, 因为它包含特殊字符。

```
//先判断用户名是否是合法的表达式;
String regx = "(^[a-zA-Z0-9_-]{6,16}$)|(^[\u2E80-\u9FFF]{2,5})";
if(!empName.matches(regx)){
    return MsgDTO.fail().add("va_msg", "用户名必须是6-16位数字和字母的组合或者2-5位中文");
}
```

```
//允许数字字母以及_-，6-16位或者中文2-5位
var regName = /(^[a-zA-Z0-9_-]{6,16}$)|(^[\u2E80-\u9FFF]{2,5})/;
//1、校验用户名
```

```
//2、校验邮箱
var email = $("#email_add_input").val();
var regEmail = /^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/;
if(!regEmail.test(email)){
```


REGULAR EXPRESSION
1 match, 127 steps (~0ms)
/ <root>[\S\s]+<\/root> /
TEST STRING
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tem="http://tempuri.org/"><soapenv:Header/><soapenv:Body>
<tem:process_Method_Palm_HZZX092><tem:message><![CDATA[<root><head>
<TransID>HZZX092</TransID><HospID>202003132</HospID><AreaID>10448</AreaID></head><body>
<MedCardType>${medCardType}</MedCardType><MedCardNO>${medCardNO}</MedCardNO>
<PatientName>${patientName}</PatientName><DateBegin>${dateBegin}</DateBegin>
<DateEnd>${dateEnd}</DateEnd></body></root>]]></tem:message>
</tem:process_Method_Palm_HZZX092></soapenv:Body></soapenv:Envelope>


REGULAR EXPRESSION
1 match, 127 steps (~1ms)
/ <root>.*<\/root> /
TEST STRING
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tem="http://tempuri.org/"><soapenv:Header/><soapenv:Body>
<tem:process_Method_Palm_HZZX092><tem:message><![CDATA[<root><head>
<TransID>HZZX092</TransID><HospID>202003132</HospID><AreaID>10448</AreaID></head><body>
<MedCardType>${medCardType}</MedCardType><MedCardNO>${medCardNO}</MedCardNO>
<PatientName>${patientName}</PatientName><DateBegin>${dateBegin}</DateBegin>
<DateEnd>${dateEnd}</DateEnd></body></root>]]></tem:message>
</tem:process_Method_Palm_HZZX092></soapenv:Body></soapenv:Envelope>