**RegExp 对象**

正则表达式描述了字符的模式对象。

当您检索某个文本时，可以使用一种模式来描述要检索的内容。RegExp 就是这种模式。

简单的模式可以是一个单独的字符。

更复杂的模式包括了更多的字符，并可用于解析、格式检查、替换等等。

您可以规定字符串中的检索位置，以及要检索的字符类型，等等。

正则表达式是描述字符模式的对象。

正则表达式用于对字符串模式匹配及检索替换，是对字符串执行模式匹配的强大工具。

**语法**

var patt=new RegExp(pattern,modifiers);  
  
或者更简单的方式:  
  
var patt=/pattern/modifiers;

* pattern（模式） 描述了表达式的模式
* modifiers(修饰符) 用于指定全局匹配、区分大小写的匹配和多行匹配

***注意：****当使用构造函数创造正则对象时，需要常规的字符转义规则（在前面加反斜杠 \）。比如，以下是等价的：*

var re = new RegExp("\\w+");

var re = /\w+/;

更多关于 RegExp 对象请阅读我们的 [JavaScript RegExp 对象教程](http://www.runoob.com/js/js-obj-regexp.html)。

**修饰符**

修饰符用于执行区分大小写和全局匹配:

|  |  |
| --- | --- |
| **修饰符** | **描述** |
| [i](http://www.runoob.com/js/jsref-regexp-i.html) | 执行对大小写不敏感的匹配。 |
| [g](http://www.runoob.com/js/jsref-regexp-g.html) | 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。 |
| m | 执行多行匹配。 |

**方括号**

方括号用于查找某个范围内的字符：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| [[abc]](http://www.runoob.com/jsref/jsref-regexp-charset.html) | 查找方括号之间的任何字符。 |
| [[^abc]](http://www.runoob.com/jsref/jsref-regexp-charset-not.html) | 查找任何不在方括号之间的字符。 |
| [0-9] | 查找任何从 0 至 9 的数字。 |
| [a-z] | 查找任何从小写 a 到小写 z 的字符。 |
| [A-Z] | 查找任何从大写 A 到大写 Z 的字符。 |
| [A-z] | 查找任何从大写 A 到小写 z 的字符。 |
| [adgk] | 查找给定集合内的任何字符。 |
| [^adgk] | 查找给定集合外的任何字符。 |
| (red|blue|green) | 查找任何指定的选项。 |

**元字符**

元字符（Metacharacter）是拥有特殊含义的字符：

|  |  |
| --- | --- |
| **元字符** | **描述** |
| [.](http://www.runoob.com/jsref/jsref-regexp-dot.html) | 查找单个字符，除了换行和行结束符。 |
| [\w](http://www.runoob.com/jsref/jsref-regexp-wordchar.html) | 查找单词字符。 |
| [\W](http://www.runoob.com/jsref/jsref-regexp-wordchar-non.html) | 查找非单词字符。 |
| [\d](http://www.runoob.com/jsref/jsref-regexp-digit.html) | 查找数字。 |
| [\D](http://www.runoob.com/jsref/jsref-regexp-digit-non.html) | 查找非数字字符。 |
| [\s](http://www.runoob.com/jsref/jsref-regexp-whitespace.html) | 查找空白字符。 |
| [\S](http://www.runoob.com/jsref/jsref-regexp-whitespace-non.html) | 查找非空白字符。 |
| [\b](http://www.runoob.com/jsref/jsref-regexp-begin.html) | 匹配单词边界。 |
| [\B](http://www.runoob.com/jsref/jsref-regexp-begin-not.html) | 匹配非单词边界。 |
| \0 | 查找 NULL 字符。 |
| [\n](http://www.runoob.com/jsref/jsref-regexp-newline.html) | 查找换行符。 |
| \f | 查找换页符。 |
| \r | 查找回车符。 |
| \t | 查找制表符。 |
| \v | 查找垂直制表符。 |
| [\xxx](http://www.runoob.com/jsref/jsref-regexp-octal.html) | 查找以八进制数 xxx 规定的字符。 |
| [\xdd](http://www.runoob.com/jsref/jsref-regexp-hex.html) | 查找以十六进制数 dd 规定的字符。 |
| [\uxxxx](http://www.runoob.com/jsref/jsref-regexp-unicode-hex.html) | 查找以十六进制数 xxxx 规定的 Unicode 字符。 |

**量词**

|  |  |
| --- | --- |
| **量词** | **描述** |
| [n+](http://www.runoob.com/jsref/jsref-regexp-onemore.html) | 匹配任何包含至少一个 n 的字符串。  例如，/a+/ 匹配 "candy" 中的 "a"，"caaaaaaandy" 中所有的 "a"。 |
| [n\*](http://www.runoob.com/jsref/jsref-regexp-zeromore.html) | 匹配任何包含零个或多个 n 的字符串。  例如，/bo\*/ 匹配 "A ghost booooed" 中的 "boooo"，"A bird warbled" 中的 "b"，但是不匹配 "A goat grunted"。 |
| [n?](http://www.runoob.com/jsref/jsref-regexp-zeroone.html) | 匹配任何包含零个或一个 n 的字符串。  例如，/e?le?/ 匹配 "angel" 中的 "el"，"angle" 中的 "le"。 |
| [n{X}](http://www.runoob.com/jsref/jsref-regexp-nx.html) | 匹配包含 X 个 n 的序列的字符串。  例如，/a{2}/ 不匹配 "candy," 中的 "a"，但是匹配 "caandy," 中的两个 "a"，且匹配 "caaandy." 中的前两个 "a"。 |
| [n{X,}](http://www.runoob.com/jsref/jsref-regexp-nxcomma.html) | X 是一个正整数。前面的模式 n 连续出现至少 X 次时匹配。  例如，/a{2,}/ 不匹配 "candy" 中的 "a"，但是匹配 "caandy" 和 "caaaaaaandy." 中所有的 "a"。 |
| [n{X,Y}](http://www.runoob.com/jsref/jsref-regexp-nxy.html) | X 和 Y 为正整数。前面的模式 n 连续出现至少 X 次，至多 Y 次时匹配。  例如，/a{1,3}/ 不匹配 "cndy"，匹配 "candy," 中的 "a"，"caandy," 中的两个 "a"，匹配 "caaaaaaandy" 中的前面三个 "a"。注意，当匹配 "caaaaaaandy" 时，即使原始字符串拥有更多的 "a"，匹配项也是 "aaa"。 |
| [n$](http://www.runoob.com/jsref/jsref-regexp-ndollar.html) | 匹配任何结尾为 n 的字符串。 |
| [^n](http://www.runoob.com/jsref/jsref-regexp-ncaret.html) | 匹配任何开头为 n 的字符串。 |
| [?=n](http://www.runoob.com/jsref/jsref-regexp-nfollow.html) | 匹配任何其后紧接指定字符串 n 的字符串。 |
| [?!n](http://www.runoob.com/jsref/jsref-regexp-nfollow-not.html) | 匹配任何其后没有紧接指定字符串 n 的字符串。 |

**RegExp 对象方法**

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [compile](http://www.runoob.com/jsref/jsref-regexp-compile.html) | 编译正则表达式。 | 1 | 4 |
| [exec](http://www.runoob.com/jsref/jsref-exec-regexp.html) | 检索字符串中指定的值。返回找到的值，并确定其位置。 | 1 | 4 |
| [test](http://www.runoob.com/jsref/jsref-test-regexp.html) | 检索字符串中指定的值。返回 true 或 false。 | 1 | 4 |

**支持正则表达式的 String 对象的方法**

|  |  |  |  |
| --- | --- | --- | --- |
| **方法** | **描述** | **FF** | **IE** |
| [search](http://www.runoob.com/js/jsref-search.html) | 检索与正则表达式相匹配的值。 | 1 | 4 |
| [match](http://www.runoob.com/js/jsref-match.html) | 找到一个或多个正则表达式的匹配。 | 1 | 4 |
| [replace](http://www.runoob.com/js/jsref-replace.html) | 替换与正则表达式匹配的子串。 | 1 | 4 |
| [split](http://www.runoob.com/js/jsref-split.html) | 把字符串分割为字符串数组。 | 1 | 4 |

在正则表达式中，如果直接给出字符，就是精确匹配。用\d可以匹配一个数字，\w可以匹配一个字母或数字，所以：

* '00\d'可以匹配'007'，但无法匹配'00A'；
* '\d\d\d'可以匹配'010'；
* '\w\w'可以匹配'js'；

.可以匹配任意字符，所以：

* 'js.'可以匹配'jsp'、'jss'、'js!'等等。

要匹配变长的字符，在正则表达式中，用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，用{n}表示n个字符，用{n,m}表示n-m个字符：

来看一个复杂的例子：\d{3}\s+\d{3,8}。

我们来从左到右解读一下：

1. \d{3}表示匹配3个数字，例如'010'；
2. \s可以匹配一个空格（也包括Tab等空白符），所以\s+表示至少有一个空格，例如匹配' '，'\t\t'等；
3. \d{3,8}表示3-8个数字，例如'1234567'。

综合起来，上面的正则表达式可以匹配以任意个空格隔开的带区号的电话号码。

如果要匹配'010-12345'这样的号码呢？由于'-'是特殊字符，在正则表达式中，要用'\'转义，所以，上面的正则是\d{3}\-\d{3,8}。

但是，仍然无法匹配'010 - 12345'，因为带有空格。所以我们需要更复杂的匹配方式。

进阶

要做更精确地匹配，可以用[]表示范围，比如：

* [0-9a-zA-Z\\_]可以匹配一个数字、字母或者下划线；
* [0-9a-zA-Z\\_]+可以匹配至少由一个数字、字母或者下划线组成的字符串，比如'a100'，'0\_Z'，'js2015'等等；
* [a-zA-Z\\_\$][0-9a-zA-Z\\_\$]\*可以匹配由字母或下划线、$开头，后接任意个由一个数字、字母或者下划线、$组成的字符串，也就是JavaScript允许的变量名；
* [a-zA-Z\\_\$][0-9a-zA-Z\\_\$]{0, 19}更精确地限制了变量的长度是1-20个字符（前面1个字符+后面最多19个字符）。

A|B可以匹配A或B，所以(J|j)ava(S|s)cript可以匹配'JavaScript'、'Javascript'、'javaScript'或者'javascript'。

^表示行的开头，^\d表示必须以数字开头。

$表示行的结束，\d$表示必须以数字结束。

你可能注意到了，js也可以匹配'jsp'，但是加上^js$就变成了整行匹配，就只能匹配'js'了。

RegExp

有了准备知识，我们就可以在JavaScript中使用正则表达式了。

JavaScript有两种方式创建一个正则表达式：

第一种方式是直接通过/正则表达式/写出来，第二种方式是通过new RegExp('正则表达式')创建一个RegExp对象。

两种写法是一样的：

**var** re1 = /ABC\-001/;

**var** re2 = **new** RegExp('ABC\\-001');

re1; *// /ABC\-001/*

re2; *// /ABC\-001/*

注意，如果使用第二种写法，因为字符串的转义问题，字符串的两个\\实际上是一个\。

先看看如何判断正则表达式是否匹配：

var re = /^\d{3}\-\d{3,8}$/;

re.test('010-12345'); // **true**

re.test('010-1234x'); // **false**

re.test('010 12345'); // **false**

RegExp对象的test()方法用于测试给定的字符串是否符合条件。

切分字符串

用正则表达式切分字符串比用固定的字符更灵活，请看正常的切分代码：

'a b c'.split(' '); // ['a', 'b', '', '', 'c']

嗯，无法识别连续的空格，用正则表达式试试：

'a b c'.split(/\s+/); // ['a', 'b', 'c']

无论多少个空格都可以正常分割。加入,试试：

'a,b, c d'.split(/[\s\,]+/); // ['a', 'b', 'c', 'd']

再加入;试试：

'a,b;; c d'.split(/[\s\,\;]+/); // ['a', 'b', 'c', 'd']

如果用户输入了一组标签，下次记得用正则表达式来把不规范的输入转化成正确的数组。

分组

除了简单地判断是否匹配之外，正则表达式还有提取子串的强大功能。用()表示的就是要提取的分组（Group）。比如：

^(\d{3})-(\d{3,8})$分别定义了两个组，可以直接从匹配的字符串中提取出区号和本地号码：

**var** re = /^(\d{3})-(\d{3,8})$/;

re.exec('010-12345'); *// ['010-12345', '010', '12345']*

re.exec('010 12345'); *// null*

如果正则表达式中定义了组，就可以在RegExp对象上用exec()方法提取出子串来。

exec()方法在匹配成功后，会返回一个Array，第一个元素是正则表达式匹配到的整个字符串，后面的字符串表示匹配成功的子串。

exec()方法在匹配失败时返回null。

提取子串非常有用。来看一个更凶残的例子：

**var** re = /^(0[0-9]|1[0-9]|2[0-3]|[0-9])\:(0[0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9]|[0-9])\:(0[0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9]|[0-9])$/;

re.exec('19:05:30'); *// ['19:05:30', '19', '05', '30']*

这个正则表达式可以直接识别合法的时间。但是有些时候，用正则表达式也无法做到完全验证，比如识别日期：

**var** re = /^(0[1-9]|1[0-2]|[0-9])-(0[1-9]|1[0-9]|2[0-9]|3[0-1]|[0-9])$/;

对于'2-30'，'4-31'这样的非法日期，用正则还是识别不了，或者说写出来非常困难，这时就需要程序配合识别了。

贪婪匹配

需要特别指出的是，正则匹配默认是贪婪匹配，也就是匹配尽可能多的字符。举例如下，匹配出数字后面的0：

**var** re = /^(\d+)(0\*)$/;

re.exec('102300'); *// ['102300', '102300', '']*

由于\d+采用贪婪匹配，直接把后面的0全部匹配了，结果0\*只能匹配空字符串了。

必须让\d+采用非贪婪匹配（也就是尽可能少匹配），才能把后面的0匹配出来，加个?就可以让\d+采用非贪婪匹配：

**var** re = /^(\d+?)(0\*)$/;

re.exec('102300'); *// ['102300', '1023', '00']*

全局搜索

JavaScript的正则表达式还有几个特殊的标志，最常用的是g，表示全局匹配：

**var** r1 = /test/g;

*// 等价于:*

**var** r2 = **new** RegExp('test', 'g');

全局匹配可以多次执行exec()方法来搜索一个匹配的字符串。当我们指定g标志后，每次运行exec()，正则表达式本身会更新lastIndex属性，表示上次匹配到的最后索引：

**var** s = 'JavaScript, VBScript, JScript and ECMAScript';

**var** re=/[a-zA-Z]+Script/g;

*// 使用全局匹配:*

re.exec(s); *// ['JavaScript']*

re.lastIndex; *// 10*

re.exec(s); *// ['VBScript']*

re.lastIndex; *// 20*

re.exec(s); *// ['JScript']*

re.lastIndex; *// 29*

re.exec(s); *// ['ECMAScript']*

re.lastIndex; *// 44*

re.exec(s); *// null，直到结束仍没有匹配到*

全局匹配类似搜索，因此不能使用/^...$/，那样只会最多匹配一次。

正则表达式还可以指定i标志，表示忽略大小写，m标志，表示执行多行匹配