# 构建方法

字面量, 构造函数和工厂符号都是可以的

new RegExp(pattern [, flags])

RegExp(pattern [, flags])

示例：

/ab+c/i;

new RegExp('ab+c', 'i');

new RegExp(/ab+c/, 'i');

参数

pattern 正则表达式的文本。

flags 如果指定，标志可以具有以下值的任意组合：

g 全局匹配;找到所有匹配，而不是在第一个匹配后停止

I 忽略大小写

m 多行; 将开始和结束字符（^和$）视为在多行上工作（也就是，分别匹配每一行的开始和结束（由 \n 或 \r 分割），而不只是只匹配整个输入字符串的最开始和最末尾处。

u Unicode; 将模式视为Unicode序列点的序列

y 粘性匹配; 仅匹配目标字符串中此正则表达式的lastIndex属性指示的索引(并且不尝试从任何后续的索引匹配)。

从ECMAScript 6开始，当第一个参数为正则表达式而第二个标志参数存在时，new RegExp(/ab+c/, 'i')不再抛出TypeError （“当从其他正则表达式进行构造时不支持标志”）的异常，取而代之，将使用这些参数创建一个新的正则表达式。ECMAScript 5中使用构造函数，且带标志符时会报错。

# 正则表达式中特殊字符的含义

# 字符类别（Character Classes）

字符 含义

**.** (点号，小数点) 匹配任意单个字符，但是行结束符除外：\n \r \u2028 或 \u2029。

在字符集中，点( . )失去其特殊含义，并匹配一个字面点( . )。

需要注意的是，m 多行（multiline）标志不会改变点号的表现。因此为了匹配多行中的字符集，可使用[^] （当然你不是打算用在旧版本 IE 中），它将会匹配任意字符，包括换行符。

例如，/.y/ 匹配 "yes make my day" 中的 "my" 和 "ay"，但是不匹配 "yes"。

**\d** 匹配任意阿拉伯数字。等价于[0-9]。

例如，/\d/ 或 /[0-9]/ 匹配 "B2 is the suite number." 中的 '2'。

**\D** 匹配任意一个不是阿拉伯数字的字符。等价于[^0-9]。

例如，/\D/ 或 /[^0-9]/ 匹配 "B2 is the suite number." 中的 'B'。

**\w** 匹配任意来自基本拉丁字母表中的字母数字字符，还包括下划线。等价于 [A-Za-z0-9\_]。

例如，/\w/ 匹配 "apple" 中的 'a'，"$5.28" 中的 '5' 和 "3D" 中的 '3'。

**\W**匹配任意不是基本拉丁字母表中单词（字母数字下划线）字符的字符。等价于 [^A-Za-z0-9\_]。

例如，/\W/ 或 /[^A-Za-z0-9\_]/ 匹配 "50%" 中的 '%'。

**\s**匹配一个空白符，包括空格、制表符、换页符、换行符和其他 Unicode 空格。

等价于 [ \f\n\r\t\v​\u00a0\u1680​\u180e\u2000​\u2001\u2002​\u2003\u2004​ \u2005\u2006​\u2007\u2008​\u2009\u200a​\u2028\u2029​​\u202f\u205f​ \u3000]。

例如 /\s\w\*/ 匹配 "foo bar" 中的 ' bar'。

**\S**匹配一个非空白符。等价于 [^ \f\n\r\t\v​\u00a0\u1680​\u180e\u2000​\u2001\u2002​\u2003\u2004​ \u2005\u2006​\u2007\u2008​\u2009\u200a​\u2028\u2029​\u202f\u205f​\u3000]。

例如，/\S\w\*/ 匹配 "foo bar" 中的 'foo'，和‘bar’。

**\t** 匹配一个水平制表符（tab）

**\r** 匹配一个回车符（carriage return）

\n 匹配一个换行符（linefeed）

**\v** 匹配一个垂直制表符（vertical tab）

**\f** 匹配一个换页符（form-feed）

**[\b]** 匹配一个退格符（backspace）（不要与 \b 混淆）

**\0** 匹配一个 NUL 字符。不要在此后面跟小数点。

**\cX** X 是 A - Z 的一个字母。匹配字符串中的一个控制字符。

例如，/\cM/ 匹配字符串中的 control-M。

**\xhh** 匹配编码为 hh （两个十六进制数字）的字符。

**\uhhhh** 匹配 Unicode 值为 hhhh （四个十六进制数字）的字符。

**\** 对于那些通常被认为字面意义的字符来说，表示下一个字符具有特殊用处，并且不会被按照字面意义解释。

例如 /b/ 匹配字符 'b'。在 b 前面加上一个反斜杠，即使用 /\b/，则该字符变得特殊，以为这匹配一个单词边界。或对于那些通常特殊对待的字符，表示下一个字符不具有特殊用途，会被按照字面意义解释。

例如，\* 是一个特殊字符，表示匹配某个字符 0 或多次，如 /a\*/ 意味着 0 或多个 "a"。 为了匹配字面意义上的 \* ，在它前面加上一个反斜杠，例如，/a\\*/匹配 'a\*'。

# 字符集合（Character Sets）

字符 含义

**[xyz]**

一个字符集合，也叫字符组。匹配集合中的任意一个字符。你可以使用连字符'-'指定一个范围。

例如，[abcd] 等价于 [a-d]，匹配"brisket"中的'b'和"chop"中的'c'。

**[^xyz]**

一个反义或补充字符集，也叫反义字符组。也就是说，它匹配任意不在括号内的字符。你也可以通过使用连字符 '-' 指定一个范围内的字符。

例如，[^abc] 等价于 [^a-c]。 第一个匹配的是 "bacon" 中的'o' 和 "chop" 中的 'h'。

# 边界（Boundaries）

字符 含义

**^**

匹配输入开始。如果多行（multiline）标志被设为 true，该字符也会匹配一个断行（line break）符后的开始处。

例如，/^A/ 不匹配 "an A" 中的 "A"，但匹配 "An A" 中的 "A"。

**$**

匹配输入结尾。如果多行（multiline）标志被设为 true，该字符也会匹配一个断行（line break）符的前的结尾处。

例如，/t$/ 不匹配 "eater" 中的 "t"，但匹配 "eat" 中的 "t"。

**\b**

匹配一个零宽单词边界（zero-width word boundary），如一个字母与一个空格之间。 （不要和 [\b] 混淆）

例如，/\bno/ 匹配 "at noon" 中的 "no"，/ly\b/ 匹配 "possibly yesterday." 中的 "ly"。

**\B**

匹配一个零宽非单词边界（zero-width non-word boundary），如两个字母之间或两个空格之间。

例如，/\Bon/ 匹配 "at noon" 中的 "on"，/ye\B/ 匹配 "possibly yesterday." 中的 "ye"。

# 分组（Grouping）与反向引用（back references）

字符 含义

**(x)**

匹配 x 并且捕获匹配项。 这被称为捕获括号（capturing parentheses）。

例如，/(foo)/ 匹配且捕获 "foo bar." 中的 "foo"。被匹配的子字符串可以在结果数组的元素 [1], ..., [n] 中找到，或在被定义的 RegExp 对象的属性 $1, ..., $9 中找到。

捕获组（Capturing groups）有性能惩罚。如果不需再次访问被匹配的子字符串，最好使用非捕获括号（non-capturing parentheses），见下面。

**\n**

n 是一个正整数。一个反向引用（back reference），指向正则表达式中第 n 个括号（从左开始数）中匹配的子字符串。

例如，/apple(,)\sorange\1/ 匹配 "apple, orange, cherry, peach." 中的 "apple,orange,"。一个更全面的例子在该表格下面。

**(?:x)** 匹配 x 不会捕获匹配项。这被称为非捕获括号（non-capturing parentheses）。匹配项不能够从结果数组的元素 [1], ..., [n] 或已被定义的 RegExp 对象的属性 $1, ..., $9 再次访问到。

# 数量词（Quantifiers）

字符 含义

**x\***

匹配前面的模式 x 0 或多次。

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"。

**x+**

匹配前面的模式 x 1 或多次。等价于 {1,}。

例如，/a+/ 匹配 "candy" 中的 "a"，"caaaaaaandy" 中所有的 "a"。

**x\*? x+?**

像上面的 \* 和 + 一样匹配前面的模式 x，然而匹配是最小可能匹配。

例如，/".\*?"/ 匹配 '"foo" "bar"' 中的 '"foo"'，而 \* 后面没有 ? 时匹配 '"foo" "bar"'。

**x?**

匹配前面的模式 x 0 或 1 次。

例如，/e?le?/ 匹配 "angel" 中的 "el"，"angle" 中的 "le"。

如果在数量词 \*、+、? 或 {}, 任意一个后面紧跟该符号（?），会使数量词变为非贪婪（ non-greedy） ，即匹配次数最小化。反之，默认情况下，是贪婪的（greedy），即匹配次数最大化。

在使用于向前断言（lookahead assertions）时，见该表格中 (?=)、(?!) 和 (?:) 的说明。

**x(?=y)**

只有当 x 后面紧跟着 y 时，才匹配 x。 例如，/Jack(?=Sprat)/ 只有在 'Jack' 后面紧跟着 'Sprat' 时，才会匹配它。/Jack(?=Sprat|Frost)/ 只有在 'Jack' 后面紧跟着 'Sprat' 或 'Frost' 时，才会匹配它。然而，'Sprat' 或 'Frost' 都不是匹配结果的一部分。

**x(?!y)**

只有当 x 后面不是紧跟着 y 时，才匹配 x。例如，/\d+(?!\.)/ 只有当一个数字后面没有紧跟着一个小数点时，才会匹配该数字。

/\d+(?!\.)/.exec("3.141") 匹配 141 而不是 3.141。

**x|y**

匹配 x 或 y

例如，/green|red/ 匹配 "green apple" 中的 ‘green'，"red apple." 中的 'red'。

**x{n}**

n 是一个正整数。前面的模式 x 连续出现 n 次时匹配。

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"。

**x{n,}**

n 是一个正整数。前面的模式 x 连续出现至少 n 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"。

**x{n,m}**

n 和 m 为正整数。前面的模式 x 连续出现至少 n 次，至多 m 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"。

# 断言（Assertions）

字符 含义

**x(?=y)**

仅匹配被y跟随的x。

举个例子，/Jack(?=Sprat)/，如果"Jack"后面跟着sprat，则匹配之。

/Jack(?=Sprat|Frost)/ ，如果"Jack"后面跟着"Sprat"或者"Frost"，则匹配之。但是，"Sprat" 和"Frost" 都不会在匹配结果中出现。

**x(?!y)**

仅匹配不被y跟随的x。

举个例子，/\d+(?!\.)/ 只会匹配不被点（.）跟随的数字。

/\d+(?!\.)/.exec('3.141') 匹配"141"，而不是"3.141"

# regExp 方法

RegExp.prototype.exec()

在目标字符串中执行一次正则匹配操作。

RegExp.prototype.test()

测试当前正则是否能匹配目标字符串。

RegExp.prototype.toString()

返回一个字符串，其值为该正则对象的字面量形式。覆盖了Object.prototype.toString() 方法。