Javascript正则表达式总结

作用：正则表达式在字符串处理方面发挥着巨大的作用，掌握了它，就能针对某一特定模式的字符串或者批量对字符串进行处理。 虽然各个编程语言对正则的支持不同，但是差异并不大。

在js中创建正则的两种方式：

1. 使用字面量。

/pattern/flags, 这就是正则表达式的字面量语法，pattern表示正则表达式的模式，flags为正则表达式的标志。比如： /a[bc]\d{2}$/gi。

2. 使用构造函数。

除了使用字面量外，还可以使用构造函数生成正则表达式。

new RegExp(pattern [, flags]), 比如:var reg= new RegExp('[a-zA-Z]+\\d$', 'gi')

这里构造函数的两个参数，第一个参数可以是字符串（es5及以前）或正则表达式（es6），第二个可选参数为字符串。

当我们事前不能确定正则表达式的值时，或者正则依赖第三方输入来动态生成正则时，需要使用这种方式。

这里要注意的一点是：第一个参数为一个字符串时，当我们使用正则里的特殊字符，比如\d或\b等时，需要对\进行转义，也就是写成\\d或\\b的形式，因为\在字符串里有特殊含义。

正则表达式的语法：

**flags - 标识符**

g - global,全局匹配，找到所有的匹配，而不是在第一个匹配后停止

i - ignoreCase 忽略大小写

m - multiline 多行搜索，不使用'm'模式时，整个字符串会被当做一行，换行符也被当做正则里的一个字符，使用'm'模式时，字符串换行后的内容会被当做新的一行。

**元字符**

正则表达式有两种基本字符类型组成：

原义字符 - 代表他们本身含义的字符

元字符 - 在正则表达式中有特殊含义的非字母字符

比如： . \* + ? () [] {} ^ $ | \

**字符类**

. - 匹配任意单个字符，行结束符除外

\d - 匹配任意阿拉伯数字，相当于[0-9]

\D - 匹配任意一个不是阿拉伯数字的字符，相当于1, 这里^用在[]中表示取反的意思。

\w - 匹配任意阿拉伯字母，数字和下划线, 相当于[0-9a-zA-Z\_]

\W - 与w相反， 相当于2

\s - 匹配一个空白符，包括空格符，制表符，换行符，换页符和其他空格字符。

\S - 匹配一个非空白符，与上面相反

\t - 匹配一个水平制表符

\r - 匹配一个回车符

\n - 匹配一个换行符

\v - 匹配一个垂直制表符

等等

**字符集合**

[abc] - 又叫字符组，表示匹配集合中的任意一个字符，可以用-来指定范围

[^abc] - 反义字符组，匹配不是集合中字符的一个字符，可以用-来指定范围

**边界**

^ - 不使用在[]中时，表示匹配输入开始

$ - 匹配输入结尾

\b - 匹配零宽单词边界

\B - 匹配零宽非单词边界，与上面b含义相反。

**量词**

？ - 匹配0次或1次

+ - 匹配一次或多次

\* - 匹配0次或多次

{n} - 匹配n次

{m,n} - 匹配最少m次，最多n次

{m,} - 匹配最少m次

**贪婪模式和非贪婪模式**

举个例子， /\d{3,6}/g，这样一个正则表达式如果匹配字符串'234955033'，是匹配3或4，5，6个数字呢？

js中正则表达式会默认尽可能多的匹配，也就是匹配6个数字，即贪婪模式。

非贪婪模式则是让正则表达式尽可能少的匹配，也就是只匹配3个数字，正则要使用非贪婪模式，只需要在量词后面加上?即可。

**分组**

() - 将正则表达式分组，以控制量词的作用范围，比如：baron{3} 和 (baron){3}，二者是有区别的。

| - 表示 或，也可以配合()进行使用，比如: foo|bar 和 fo(o|b)ar

**反向引用和忽略分组**

有时，我们在匹配到正则对应的字符串时，还想要使用匹配到的部分内容，这时候可以使用$1,$2,$3等符号引用分组捕获的内容。

**前瞻，后顾和断言**

这里出现了三个新的概念，但其实他们也很简单。

首先，我们要分清何为前，何为后。

正则匹配字符串是从前往后解析，所以，往字符串尾部方向叫前瞻，往字符串头部方向叫后顾。

js中的正则表达式支持前瞻不支持后顾，所以这里只介绍前瞻。

前瞻一般结合断言使用，其实断言(assert)就相当于给正则加的一个限制条件 -- 表示匹配项不仅要符合正则表达式，而且要符合断言给出的条件。

正向前瞻 - exp(?=assert)，比如/\d{2}(?=a)/ 表示匹配两个数字且后面必须有字母a跟随

负向前瞻 - exp(?!assert)，比如/\d{2}(?!a)/ 表示匹配两个数字且后面没有字母a跟随

注意： 断言只是相当于正则的条件，并不会真正的匹配相应的字符。

正则的实例属性

一个正则表达式的实例具有以下一些实例属性

multiline - 布尔值，表示正则是否多行匹配，对应于标志中的'm'

global - 布尔值，表示是否全局匹配，对应于标志中的'g'

ignoreCase - 布尔值，表示是否忽略大小写，对应于标志中的'i'

source - 字符串，是正则表达式模式的字符串表示，比如/\d{2}ac/gi的source属性就是'\d{2}ac', 该值不会包含正则两边的//和标志gi，也不会像构造函数参数那样对\进行转义。

lastIndex - 整数，表示下一次匹配的起始位置，该值只有在'g'模式下才有用，在不全局匹配时始终是0

正则的实例方法

正则表达式主要有两个实例方法test和exec。

假设regexp表示一个正则的实例，str表示一个字符串，那么

regexp.test(str) - 返回一个布尔值，如果找到匹配项，返回true，否则false。

reg.exex(str) - 返回一个数组， 数组第一项为匹配到的字符串，后面项依次为分组捕获到的信息，该数组还具有input和index两个属性，input表示进行匹配的字符串 - 即参数str，index表示匹配到的字符串的起始位置。

这里要注意的是：应用了全局'g'标志和没有全局'g'标志的同一个模式的正则，对于同一个字符串，两种方法的表现可能是不同的。 因为如果没有全局标识，每次都会从字符串起始位置0开始匹配，而应用了'g'，会从上次匹配的结束位的下一位开始匹配。

正则的静态属性

正则的构造函数RegExp具有一些静态属性，这些属性中保存着最新一次正则匹配的信息。

比如RegExp.$1-$9中，保存着最近一次匹配中分组捕获到的信息。

还有一些其他的静态属性。

**JavaScript正则表达式函数总结（常用）**

JS中正则对象修饰符可选值为："i" "g" "m"，即忽略大小写 进行全局匹配 多行模式

JS中正则表达式支持的元字符：

1：枚举方括号表达式，范围方括号表达式，枚举取反方括号表达式，范围取反方括号表达式

2：. \w \W \d \D \s \S \b \B \n \r \f \v \888 \uffff \xff

3：n? n+ n\* n{a} n{a,} n{a,b}及这些量词的非贪心模式

4：^ $ (?!) (?=) (?:)

其他：

1：JS中引用分组请用\number，不支持''和<>这样的命名分组

2：不支持量词的占有模式

3：不支持逆向环视

4：方括号表达式中复合表达式支持如下：

[abc0-9] [abc\d] [\d\w^&]等等类似的

5：支持修饰符组合，比如gim mig gi ig等等，无顺序要求

JS正则对象属性：

global：该对象是否开启全局匹配，一个布尔值

ignoreCase：该对象是否开启忽略大小写，一个布尔值

multiline：该对象是否开启多行模式，一个布尔值

lastIndex：该对象下一次开始匹配的位置，即上一次匹配结束的位置，一个正整数

source：正则表达式的源文本，即/regexp/flag的regexp，不包含修饰符，一个字符串

创建正则表达式：

1：myreg = new RegExp(patternString[,globalFlags]);

2：myreg = /yourRegexp/globalFlags;

JS中正则表达式常用方法：

在RegExp对象中：

1：test(yourString);

定义：检测一个字符串是否匹配某个模式。

其他说明：该方法将忽略"g"标志和正则对象的lastIndex属性（即总是从开头找起），只要查找到符合模式的字符串即匹配成功。

2：exec(yourString);

定义：检索字符串中的正则表达式的匹配。

其他说明：若没开启"g"标志，只查找一次，不修改正则对象的lastIndex属性，返回一个对象，包含下标0、index、input，这时属性和match等价；开启"g"，将修改正则对象的lastIndex属性为本次找到的子字符串的末尾位置，返回一个和没开启"g"一样的对象。（你可以自行修改正则对象的lastIndex属性来规定exec开始查找的位置。）

在String对象中：

1：search(yourRegexpOrString);

定义：检索字符串中与指定的子字符串或正则表达式相匹配的子字符串。

其他说明：该方法将忽略"g"标志和正则对象的lastIndex属性（即总是从开头找起），返回找到的第一个字符的位置，如果未找到返回-1。

2：match(yourRegexpOrString);

定义：在字符串内查找一个或多个与正则表达式匹配的字符串，返回一个对象。（忽略laseIndex属性）

其他说明：若没开启"g"标志，将只查找第一个匹配的字符串，返回一个对象，包含下标0、index、input，其中下标0等价于index，input是String的引用；开启"g"，返回一个数组，数组的length是匹配的字符串个数，每个元素是每个匹配的起始字符位置。

3：replace(yourRegexpOrString,placementString);

定义：用于在字符串中用一些字符替换另一些字符，或替换一个与正则表达式匹配的子串。

其他说明：如果没有"g"标志只会替换一次，并且如果参数是字符串而不是正则对象始终替换一次。返回一个字符串，为用placementString替换yourRegexpOrString后的结果。placementString中$有特殊用途，见下表：

|  |  |
| --- | --- |
| $1、$2、...、$99 | regexp中捕获的引用分组。 |
| $& | 本次匹配的文本。 |
| $` | 本次匹配到的子串左侧的文本。 |
| $' | 本次匹配到的子串右侧的文本。 |
| $$ | $本身。 |

该方法的第二个参数可以为一个函数，每次匹配都会调用该函数，以函数返回值作为placement，一共传入4个参数，第一个参数为本次匹配的子文本（$&），中间参数为子表达式匹配字符串，个数不限（$i），倒数第二个参数为本次匹配到的子文本下标位置，最后一个参数表示执行place方法的字符串本身。

4：split(yourRegexpOrString[,howMany]);

定义：把一个字符串分割成字符串值的数组。

其他说明：是arr.join(separatorString)反操作，如果传入""，则每个字符都将会被分隔到数组。howMany指定返回数组的个数。