正则表达式

正则表达式

- 概述
- 相关方法
- 特殊字符
- 字符集

正则表达式概述

什么是正则表达式

- 正则表达式：（Regular Expression）用于匹配规律规则的表达式，正则表达式最初是科学家对人类神经系统的工作原理的早期研究，现在在编程语言中有广泛的应用。正则表通常被用来检索、替换那些符合某个模式(规则)的文本。
- 正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。

正则表达式的作用

1. 给定的字符串是否符合正则表达式的过滤逻辑(匹配)
2. 可以通过正则表达式，从字符串中获取我们想要的特定部分(提取)
3. 强大的字符串替换能力(替换)

正则表达式的测试

在线测试正则：https://c.runoob.com/front-end/854

正则表达式语法

语法

- 在 JavaScript 中，正则表达式也是对象，是一种索引类型。
- 使用一个正则表达式字面量是最简单的方式。两个 / 是正则表达式的定界符。
- 你可以通过下面两种方法创建一个正则表达式：
- 使用一个正则表达式字面量，如下所示：

  var **reg = /abc/**;

  正则表达式字面量在脚本加载后编译。若你的正则表达式是常量，使用这种方式可以获得更好的性能。
- 调用 RegExp 对象的构造函数，如下所示：

  var re = **new RegExp("abc");**

相关正则方法

相关正则方法

- 字符串的方法：

split()	根据匹配字符串切割父字符串
match()	使用正则表达式与字符串相比较，返回一个包含匹配结果的数组。
search()	对正则表达式或指定字符串进行搜索，返回第一个出现的匹配项的下标。
replace()	用正则表达式和字符串直接比较，然后用新的子串来替换被匹配的子串。

- 正则表达式方法：

exec()	在目标字符串中执行一次正则匹配操作。
test()	测试当前正则是否能匹配目标字符串。

String：split

- 根据匹配字符串切割父字符串

  eg: 根据空格切割字符串：'aa bbb c dd eeeeee'

  'aa bbb c dd eeeeee'.split(/\s+/)

  => ["aa", "bbb", "c", "dd", "eeeeee"]

  练习：根据字母a切割字符串'bbaacaaaadddaeeeeeaaf'

String：search

- 寻找匹配字符串在父字符串中位置

  eg：在‘abcdefg’中寻找‘cd’位置

  'abcdefg'.search(/cd/)

  => 2

  练习：在字符串'aaaa o o bbb aaa'中寻找‘o o’位置

String：match

- 在父字符串中寻找匹配字符串

  在 'abbcccbbbbbddbbbdabbb' 中查询重复 'b' 字符串

  'abbcccbbbbbddbbbdabbb'.match(/b+/g)

  => ["bb", "bbbbb", "bbb", "bbb"]

  练习：'aaaa o o bbb o o aaa'找寻所有 ‘o o’ 字符串

String：replace

- 替换父字符串中匹配字符串

  eg：将'www.hello.com'替换成'www.byebye.com'

  'www.hello.com'.replace(/hello/, 'byebye')

  => "www.byebye.com"

  练习：过滤字符串中空格：'aa　b　 c d e　 f　' => "aabcdef"

RegExp：exec

- 在字符串中寻找匹配字符串，该方法比其他正则方法或字符串方法支持的更复杂

  eg：在'aaaabccccbacabc'中查找'abc'字符串

  var result = /abc/.exec('aaaabccccbacabc')

  result => ["abc"]

  result.index => 3

  练习：在'ooooo 0 0 ooooooooooo'字符串中匹配["o 0 0 o"]

RegExp：test

- 检测字符串是否匹配正则表达式

  eg：判断'aaddccddabcddeeddfff'是否包含'abc'字符串

  /abc/.test('aaddccddabcddeeddfff')

  => true

  练习：检测'a bc'和'abc'是否包含空格

正则表达式的组成

正则表达式的组成

由一些普通字符和一些特殊字符（又叫元字符--metacharacters）组成。普通字符包括大小写的字母和数字，而元字符则具有特殊的含义。

特殊字符：javascript 中常用特殊字符有 ( ) [ ] { } \ ^ $ | ? * + .

若想匹配这类字符必须用转移符号 \ 如：\(,\^,\\

预定义特殊字符：

\t	/\t/	制表符	eg: console.log(/\t/.test('　　'))
\n	/\n	回车符	eg: console.log(/\n/.test(`aaa bbb`));
\f	/\f/	换页符	
\b	/\b/	空格	

正则的术语

字符集

简单类: 正则的多个字符对应一个字符，我们可以用 [] 把它们括起来，让 [] 这个整体对应一个字符[abc]

例子：o[usb]t——obt、ost、out

范围类: 有时匹配的东西过多，而且类型又相同，全部输入太麻烦，我们可以在中间加了个横线。[a-z]、[0-9]、[A-Z]

例子：id[0-9]——id0、id5

负向类: [] 前面加个元字符进行取反，表示匹配不能为括号里面的字符。[^a]

例子：o[^0-9]t——oat、o?t、o t

组合类: 允许用中括号匹配不同类型的单个字符。[0-9a-b]

例子：o[0-9a-b]t——oat、o?t、o

修饰符

g 修饰符用于执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。

```
'12a34b56c78d90e'.match(/\d+/)      => ["12"]
'12a34b56c78d90e'.match(/\d+/g)     => ["12", "34", "56", "78", "90"]
```

i 修饰符用于执行对大小写不敏感的匹配。

```
'aabAAcAa'.match(/aa/g)             => ["aa"]
'aabAAcAa'.match(/aa/gi)            => ["aa", "AA", "Aa"]
```

边界

^　　开头　　注意不能紧跟于左中括号的后面

```
/^hello/.test('hello javascript')          => true
/^javascript/.test('hello javascript')     => false
```

$　　结尾

```
/javascript$/.test('hello javascript')     => true
/hello$/.test('hello javascript')          => false
```

预定义类

.	[^\n\r]	除了换行和回车之外的任意字符
\d	[0-9]	数字字符
\D	[^0-9]	非数字字符
\s	[\t\n\xOB\f\r]	空白字符
\S	[^ \t\n\xOB\f\r]	非空白字符
\w	[a-zA-Z_0-9]	单词字符(所有的字母/数字/下划线)
\W	[^a-zA-Z_0-9]	非单词字符

量词

{n}	硬性量词	对应零次或者n次
{n, m}	软性量词	至少出现n次但不超过m次(中间不能有空格)
{n, }	软性量词	至少出现n次(+的升级版)
?	软性量词	出现零次或一次
*	软性量词	出现零次或多次(任意次)
+	软性量词	出现一次或多次（至少一次）

分组

虽然量词的出现，能帮助我们处理一排密紧密相连的同类型字符。但这是不够的，我们用中括号表示范围内选择，大括号表示重复次数。如果想获取重复多个字符，我们就要用小括号进行分组了。

```
/(bye){2}/.test('byebye')
=> true
/(bye){2}/.test('bye')
=> false
```

或操作符

可以使用竖线（|）字符表示或者的关系。

/a|bcd/　　匹配 a 或 bcd 字符。

/(ab)+|(cd)+/　　匹配出现一次或多次的 ab 或者 cd

分组的反向引用

反向引用标识是对正则表达式中的匹配组捕获的子字符串进行编号，通过“\编号(在表达式中)”，“$编号（在表达式外）”进行引用。从1开始计数。

```
/(bye)\1/.test('byebye')                    => true
/(bye)\1/.test('bye')              => false
'123*456'.replace(/(\d{3})\*(\d{3})/, '$2*$1')                  => "456*123"
'123*456'.replace(/(\d{3})\*(\d{3})/, function (match, $1, $2) {
        return $2 + '*' + $1
})                                                      => "456*123"
```

中文字符

匹配中文：[\u4e00-\u9fa5]

```
/[\u4e00-\u9fa5]+/.test('中文内容')
=> true
/[\u4e00-\u9fa5]+/.test('aaa')
=> false
```

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容