

写给前端的正则表达式入门

savokiss SegmentFault 今天

本文转载于 SegmentFault 社区专栏：码力全开

作者：savokiss

概览

在 **JavaScript** 中，使用 **//** 即可创建一个正则表达式对象，当然也可以使用 **new RegExp()**

常用的跟正则相关的方法有 **match**、**test** 和 **replace**。

其中 **match**，**replace** 都是字符串上的方法，**test** 是正则对象上的方法。

下面看具体的图示：

The diagram illustrates JavaScript regex syntax and usage with the following code and annotations:

```
const regexOne = /reg/i
const regexTwo = new RegExp('101', 'g')

const str = "regex101"

str.match(regexOne) // ["reg"]
str.match(regexTwo) // ["101"]
str.replace(/r/, 'R') // "Regex101"

const contains_r = /r/.test(str) // true
```

Annotations:

- Forward slashes**: Points to the slashes in `/reg/i`.
- Matching pattern**: Points to `reg` in `/reg/i`.
- Optional flag(s)**: Points to `i` in `/reg/i`.
- Call ``match`` on the string, and pass in a regex. Returns array of results**: Points to the `match` method calls.
- It's common to create the regex right in the method call**: Points to the inline regex `/r/` in `str.replace(/r/, 'R')`.
- ``test`` is the opposite: call method on the regex and pass in the string to test**: Points to the `test` method call in `/r/.test(str)`.

匹配单字符

- `/reg/` 可以直接匹配具体的字符串 `reg`。

- `/[arzy]/` 中的中括号 `[]` 则代表匹配 `arzy` 中的任意单个字符
- `/[f-h]/` 中的中杠 `-` 代表匹配字母表顺序中 `f` 到 `h` 中的任意单个字符
- `/[1-3]/` 则代表匹配数字 1 到 3

```
const str = "regex148"
```

```
str.match(/reg/) // ["reg"]
```

Find a three character
sequence that = "reg"

```
str.match(/[arzy]/) // ["r"]
```

Find a single character
that is any of: a, r, z, y

```
str.match(/[f-h]/) // ["g"]
```

Find a single character
that falls between
lowercase f and lowercase h

```
str.match(/[1-3]/) // ["1"]
```

Find a single number
between 1 and 3 (inclusive)

正则选项

正则对象的后面也可以跟选项，JavaScript 中常用的选项有：

- **i** - 代表忽略大小写
- **m** - 代表多行匹配
- **g** - 代表全局匹配（可以匹配多次）

```
const str = "regex148"
```

```
str.match(/e/) // ["e"]
```

Find the `_first_` match
for the letter 'e'

```
str.match(/e/g) // ["e", "e"]
```

g flag = 'global'
Search the entire string,
and return all matches

```
str.match(/E/i) // ["e"]
```

i flag = 'case insensitive'
Searching for capital 'E',
but finds lowercase 'e' also

```
str.match(/E/ig) // ["e", "e"]
```

Can stack multiple flags
here = case insensitive + global

边界匹配

- `^` - 代表匹配字符串的开头
- `$` - 代表匹配字符串的结尾

```
const str = "regex148"
```

```
str.match(/^r/) // ["r"]
```

Find the letter 'r' at the start of the string

```
str.match(/^x/) // null
```

Find the letter 'x' only at the start of the string won't find the other 'x'

```
str.match(/[0-9]$/) // ["8"]
```

Find any number at the end of the string

```
str.match(/^([a-z][a-z][a-z])/) // ["reg"]
```

Find three lowercase letters at the start of the string

```
str.match(/[a-z][0-9]/) // ["x1"]
```

Find any lowercase letter followed by any number

字符匹配

- `.` - 可以匹配除了换行符外的任意字符
- `\d` - 可以匹配任意数字
- `\D` - 可以匹配任意非数字
- `\s` - 匹配任意空白字符
- `\S` - 匹配任意非空白字符
- `\n` - 匹配换行
- `\w` - 它其实就等同于 `[A-Za-z0-9_]`，即匹配字母数字下划线

```
const str = "regex148"
```

```
str.match(/\d/g) // ["1", "4", "8"]
```

Find any digit, globally
(find all matches)

```
str.match(/\D\d/) // ["x1"]
```

Find any NON digit,
followed by any digit

```
str.match(/.\S./g) // ["reg", "ex1"]
```

wildcard, NON space, wildcard
this returns two matches, but
there aren't enough characters
left for a third match

量词匹配

- `*` - 匹配 0 次或多次
- `+` - 匹配 1 次或多次
- `?` - 匹配 0 次或 1 次
- `{3}` - 匹配 3 次
- `{2,4}` - 匹配 2、3 或 4 次
- `{2,}` - 匹配 2 次或多次

```
const str = "regex148"
```

```
str.match(/.*148/) // ["regex148"]
```

Find zero or more wildcards,
followed by '148'

```
str.match(/[a-z]+)/ // ["regex"]
```

Find one or more lowercase
letters (grouped together
into a single match)

```
str.match(/[0-9]{2}/) // ["14"]
```

Find exactly two numbers together
Notice: it gets '14' but doesn't
reuse the '4' for '48'

```
str.match(/[re]+z?[egr]*/) // ["rege"]
```

Find either
'r' or 'e'
once or more

Followed by any of the
letters 'e', 'g', or 'r'
zero or more times

Then find letter 'z'
zero or one times
(doesn't match in this case)

分组

() 小括号在正则中代表分组，一般在 `match` 方法中用来返回全匹配加上多个分组结果，如果使

用了 **g** 选项，则只返回全匹配。

在小括号中你可以使用管道符号 **|**，它代表**或**

```
const str = "regex148"

str.match(/.*(ex)(.+)/) // ["regex148", "ex", "148"]
str.match(/.*(ex)(.+)/g) // ["regex148"]
str.match(/(re|ex)[0-9]+/g) // ["ex148"]
```

Annotations:

- Arrow pointing to `(ex)` in the first line: ``match` returns the full match first, and then all the () groups as well`
- Arrow pointing to `(ex)` in the second line: `Find any wildcars, followed by the letters 'ex' together followed by any more wildcards`
- Arrow pointing to `/g` in the third line: `If the 'g' flag is set, `match` only returns the full match(es)`
- Arrow pointing to `(re|ex)` in the fourth line: `Find 're' or 'ex' followed by any count of numbers`

特殊符号

匹配特殊符号的时候需要加反斜杠 ****

JS 中的特殊字符有 **^ \$ \ . * + ? () [] {} |**

所以如果你需要匹配星号 `*`，就需要这样写： `*`

```
const str = "function(param1) { console.log('Testing!') }"
```

```
str.match(/(.*)/g) // ["function(param1) { console.log('Testing!') }", ""]
```

← If you don't escape the parens, this just matches any number of wildcards in a single group (which is the entire string)

```
str.match(/\(.*\) /) // ["(param1) { console.log('Testing!')"}]
```

← With backslash escapes, this matches the `_character_` (then any number of wildcards, and then the `_character_`) Notice! wildcard includes `()` so this matches more than just a single set of parens

```
str.match(/\([a-zA-Z0-9]*\) /) // ["(param1)"]
```

← Now, just match letters and numbers inside of the characters `()` - just returns the first paren group

取非匹配

匹配除了某个字符的任意字符，需要在中括号

`[]` 中使用 `^`

至此 `^` 具有两个含义：

- 如果用于正则表达式的开头，代表匹配字符串的开头
- 如果用于中括号 `[]` 内部，则代表匹配非此字符

```
const str = "function(param1) { console.log('Testing!') }"

str.match(/[^\0-9]+/g) // ["function(param", ") { console.log('Testing!') }"]
    Find anything BUT a number, one or more in a row
    The output is the whole string, but split where the
    number used to be

str.match(/[^\a-zA-Z]+/g) // ["(", "1) { ", ".", "('", "!'") }"]
    Find anything that isn't a lowercase letter
    OR an uppercase letter, one or more in a row

str.match(/\([^()]*\)/g) // ["(param1)", "('Testing!')"]
    Find ( followed by any number of NON close parens,
    followed by )
    This fixes the problem from the last example,
    and now returns both paren sets from the string
```

结语

上面只是介绍了正则表达式中的基本用法，在日常开发中基本就够用了。更深入的用法比如贪婪

和懒惰、零宽断言和捕获，感兴趣的小伙伴可以自行学习~

正则很强大，但是也不能滥用。如果你写出一个很复杂的只有你能看懂的正则，更好的做法是不使用正则去实现它~

参考链接：

Intro to Regex for Web Developers

SegmentFault 社区链接：

<https://segmentfault.com/a/1190000021145901>

- END -

与程序员相处的基本规范

Fasten Your Seatbelts



不会修电脑



不盗QQ号



搭网站请付费

.....

SegmentFault 「思否」
真正属于开发者的社区

长按图片
扫码关注

