JS 正则表达式

方式

- 字面量、直接量
- 构造函数

写法

`/匹配内容/`

`new RegExp()`

```
var str = 'I love js';
var pattern = new RegExp('js');
```

```
var str = 'I love js';
var pattern = /js/;

console.log(pattern.test(str)); // 找到返回true 找不到返回false
console.log(pattern.exec(str));
// 找不到返回null
/*
["js", index: 7, input: "I love js", groups: undefined]
*/
```

模式修饰符
* i:ignoreCase
* g:global
* m:multiline

`/js/i`
`/js/igm`
`/js/im`
`/js/gm`

```
var str = 'I love JS';
```

```
var pattern = /js/i;
```

```
var str = 'I love js';
var pattern = new RegExp('js', 'i');
```

创建正则

```
var str = 'I love JS';
var userInput = 'love';
var pattern = new RegExp(userInput, 'i');

console.log(pattern);
console.log(pattern.exec(userInput));
```

转义字符

```
var str = '//我是注释';
var pattern = /\//;
console.log(pattern.exec(str));
```

```
var str = '\\\\';
var pattern = /\\\\/;
console.log(pattern.exec(str));
```

- \n
- \t
- 中文Unicode范围\u4e00-\u9fa5

字符类 [ ]

介绍

`[] 字符类 中括号内的字符 先匹配到哪个就算匹配成功`

```
var str = 'javascript';
var pattern = /[js]/
console.log(pattern.exec(str)); // j
```

```
var str = '对javascript啊';
var pattern = /[\u4e00-\u9fa5]/
console.log(pattern.exec(str)); // 对
```

- [a-zA-Z] a到Z的英文字母
- [\u4e00-\u9fa5] 中文匹配
- [0-9] 匹配数字

常用字符类与别名

- /[a-zA-Z0-9_]/ 别名 /\w/
- /[0-9]/ 别名 /\d/
- / / 别名 /\s/ 匹配空白（空格 tab unitcode空白）
- /\S/ 除了匹配空白（空格 tab unitcode空白）

```
var str = '3.14';
var pattern = /\./
console.log(pattern.exec(str)); // .
```

```
var str = '\n';
var pattern = /./
console.log(pattern.exec(str)); // null
```

```
var str = '_@';
var pattern = /[a-zA-Z0-9_]/
console.log(pattern.exec(str)); // _
```

```
var str = '_@';
var pattern = /\w/
console.log(pattern.exec(str)); // _
```

重复（量词）

具体几次{n}

```
var str = '110';
var pattern = /\d{2}/
console.log(pattern.exec(str)); // 11
```

范围{n,m} n次到m次

```
var str = '110';
var pattern = /\d{1,4}/
console.log(pattern.exec(str)); // 110
```

大于n次 {n,}

```
var str = '110';
var pattern = /\d{1,}/
console.log(pattern.exec(str)); // 110
```

0次或1次 {0,1} 也可以用 ?

```
var str = '110';
var pattern = /\d{0,1}/
console.log(pattern.exec(str)); // 1
```

0次或1次 ?

```
var str = '110';
var pattern = /\d?/
console.log(pattern.exec(str)); // 1
```

若干次 *

```
var str = '110';
var pattern = /\d*/
console.log(pattern.exec(str)); // 110
```

至少一次 +

```
var str = 'fff1';
var pattern = /[a-z]+/
console.log(pattern.exec(str)); // fff
```

```
var str = 'fff1';
var pattern = /\d+/
console.log(pattern.exec(str)); // 1
```

示例

```
var str = '肯德基豪华午餐：￥15.5! ';
// var pattern = /\d+\.?\d*/;
var pattern = /\d{1,}\.{0,1}\d{0,}/
console.log(pattern.exec(str));
```

贪婪匹配/非贪婪匹配

贪婪匹配

```
var str = 'aaab';
var pattern = /a+/
console.log(pattern.exec(str)); // aaa
```

非贪婪模式

001

```
var str = 'aaab';
var pattern = /a+?/
console.log(pattern.exec(str)); // a
```

002

```
var str = 'aaab';
var pattern = /a+b/
console.log(pattern.exec(str)); // aaab
```

003

```
var str = 'aaab';
var pattern = /a+?b/
console.log(pattern.exec(str)); // aaab
```

后面跟字符了非贪婪不起作用

004

```
var str = 'aaab';
var pattern = /a+b/
console.log(pattern.exec(str)); // aaab
```

示例

- 贪婪

```
var str = '<td><p>a</p></td><td><p>b</p></td>';
var pattern = /<td>.*<\/td>/
console.log(pattern.exec(str));
```

`打印结果：<td><p>a</p></td><td><p>b</p></td>`

- 非贪婪

```
var str = '<td><p>a</p></td><td><p>b</p></td>';
var pattern = /<td>.*?<\/td>/
console.log(pattern.exec(str));
```

`打印结果：  <td><p>a</p></td>`

选择、分组、引用

选择

选择是用符号 |

001

```
var str = 'css html js';
var pattern = /html|js|css/
console.log(pattern.exec(str)); // css
```

pattern中包含的字符，在字符串str中先出现哪个就匹配哪个

002

```
var str = 'ab';
var pattern = /ab|a/
console.log(pattern.exec(str)); // ab
```

003

```
var str = 'ab';
var pattern = /a|ba/
console.log(pattern.exec(str)); // a
```

分组和引用

001

```
var str = 'abababab';
var pattern = /ab+/
console.log(pattern.exec(str)); // ab
```

注：ab+ 匹配一个a多个b

002

```
var str = 'abababab';
var pattern = /(ab)+/
console.log(pattern.exec(str)); // abababab ab
```

匹配多个ab，并且将ab分组

003

```
var str = 'abcd';
var pattern = /(ab)c/
console.log(pattern.exec(str)); // abc ab
```

匹配abc 并且将ab分组

004

```
var str = 'abcd';
var pattern = /(?:ab)c/
console.log(pattern.exec(str)); // abc
```

非贪婪，值匹配abc

005

```
var str = 'abcd';
var pattern = /(ab)(c)/
console.log(pattern.exec(str)); // abc ab c
```

匹配abc 分组ab 分组c

006

```
var str = 'abcd';
var pattern = /(abc)/
console.log(pattern.exec(str)); // abc abc
```

匹配abc 分组abc

007

```
var str = 'abcd';
var pattern = /(a(b(c)))/
console.log(pattern.exec(str)); // abc abc bc c
```

匹配abc 分组abc 分组bc 分组c

008

```
var str = 'ab cd ab';
var pattern = /(ab) cd \1/
console.log(pattern.exec(str)); // "ab cd ab", "ab"
```

匹配ab cd ab 分组ab \1意思是取第一分组

009

```
var str = '<p><a>这段链接</a></p>';
var pattern = /<([a-zA-Z]+)>(.*?)<\/\1>/;
console.log(pattern.exec(str));

// "<p><a>这段链接</a></p>", "p", "<a>这段链接</a>"
```

匹配一个标签并分组 (.*?)分组中间所有内容，包括a标签 \1取第一分组

位置

- ^ 以什么开头
- $ 以什么结尾

001

```
var str = 'js';
var pattern = /^js/; // 以js开头
console.log(pattern.exec(str)); // js
```

002

```
var str = 'html js';
var pattern = /^js/
console.log(pattern.exec(str)); // null
```

/[^0-9]/ 除了数字

003

```
var str = 'html js';
var pattern = /js$/ // 以js结尾
console.log(pattern.exec(str)); // js
```

004

```
var str = 'html js css';
var pattern = /js$/
console.log(pattern.exec(str)); // null
```

005

```
var str = 'html js css';
var pattern = /js$/
console.log(pattern.exec(str)); // null
```

006

```
var str = 'a1b3c12345550';
var pattern = /\d+/; // 至少一个数字
console.log(pattern.exec(str)); // 1
```

007

```
var str = '131234555a0';
var pattern = /^\d+$/ // 全部是数字
console.log(pattern.exec(str)); // null
```

008

```
var str = '1312345550';
var pattern = /^\d+$/
console.log(pattern.exec(str)); // 1312345550
```

008

```
var str = '1312345550';
var pattern = /\D/;  // 非数字
console.log(pattern.exec(str)); // null
```

009

```
var str = '131234555a0';
var pattern = /\D/ // 非数字
console.log(pattern.exec(str)); // a
```

010

```
var str = '1312345550aaa';
var pattern = /^\d+[a-z]+$/  // 数字开头 字母结尾
console.log(pattern.exec(str)); // 1312345550aaa
```

边界匹配 \b单词边界

001

```
var str = 'js';
// var pattern = /js/;
var pattern = /\bjs\b/; // 以js开头以js结尾
console.log(pattern.exec(str)); // js
```

002

```
var str = 'js html';
// var pattern = /js/;
var pattern = /\bjs/;  // 以js开头
console.log(pattern.exec(str)); // js
```

003

过滤出js，前后不能是数字、_、字母

```
var str = '@@@@·+js@@@@@';
// var pattern = /js/;
var pattern = /\bjs\b/;
console.log(pattern.exec(str)); // js
```

004

```
var str = '@@@@js@@@@@';
// var pattern = /js/;
var pattern = /\bjs\b/;
console.log(pattern.exec(str)); // js
```

005

`<p class = "odd2 odd odd1"> </p>`

`var pattern = new RegExp('^|\\s+' + className + '$|\\s+');`

要么开头，要么前面有若干个空格符
中间是className
要么结尾，要么后面跟若干个空格符

可以直接使用/b

`var pattern = new RegExp('\\b' + className + '\b');`

```
<p class = "odd2 odd odd1"> </p>

function getByClassName(className, parentNode) {
    if (!document.getElementsByClassName) {
        return document.getElementsByClassName(className);
    } else {
        parentNode.parentNode || document;
        var nodeList = [];
        var allNodes = parentNode.getElementsByTagName('*');
        var pattern = new RegExp('^|\\s+' + className + '$|\\s+');

        for (let i = 0; i < allNodes.length; i++) {
            if (pattern.test(allNodes[i].className)) {
                nodeList.push(allNodes[i]);
            }
        }
    }
}
```

前瞻性匹配 (?=) (?!)

(?=)

```
// 前瞻性匹配
var str = 'javascript';
var pattern = /java(?=script)/;  // java后面接的是script匹配成功
console.log(pattern.exec(str)); // java
```

(?!)

```
var str = 'javascript';
var pattern = /java(?!script)/;  // java后面不是script匹配成功
console.log(pattern.exec(str)); // null
```

RegExp对象

双重转义

`new RegExp("\\b");`

创建对象

```
var pattern = /js/;
```

```
var pattern = new RegExp('js');
```

对象属性 ignoreCase、global、multiline、source

```
var str = 'js js js';
var pattern = /js/g;
console.log(pattern.ignoreCase); // false
console.log(pattern.global); // true
console.log(pattern.multiline); // false
console.log(pattern.source); // js
```

对象属性lastIndex

```
console.log(pattern.lastIndex); // 0
console.log(pattern.exec(str)); // js 0
console.log(pattern.lastIndex); // 2
console.log(pattern.exec(str)); // js 3
console.log(pattern.lastIndex); // 5
console.log(pattern.exec(str)); // js 6
console.log(pattern.lastIndex); // 8
console.log(pattern.exec(str)); // null
console.log(pattern.lastIndex); // 0
console.log(pattern.exec(str)); // js 0
console.log(pattern.lastIndex); // 2
```

```
var str = 'js js js';
var pattern = /js/g;
console.log(pattern.lastIndex); // 0
console.log(pattern.test(str)); // true
console.log(pattern.lastIndex); // 2
console.log(pattern.test(str)); // true
console.log(pattern.lastIndex); // 5
console.log(pattern.test(str)); // true
console.log(pattern.lastIndex); // 8
console.log(pattern.test(str)); // false
console.log(pattern.lastIndex); // 0
console.log(pattern.test(str)); // true
console.log(pattern.lastIndex); // 2
```

```
var str = 'js js js';
var pattern = /(j)s/g;
console.log(pattern.exec(str)); // j
```

```
console.log(pattern.exec(str)); // j
console.log(pattern.exec(str)); // j
console.log(pattern.exec(str)); // null
```

示例

```
var str = '1.js 2.js 3.js';
var pattern = /js/g;
var total = 0,
    match = '',
    reuslt;
while ((result = pattern.exec(str)) != null) {
    total++;
    match += '第' + total + '个匹配到的结果是：' + result[0] + ',它的位置是：' + resul
t.index + '\n';
}
console.log('总共：' + total + '处');
console.log(match);
```

toString()、toLocaleString()、valueOf()

```
var pattern = new RegExp('a\\nb');
console.log(pattern.toString()); // /a\nb/
console.log(pattern.toLocaleString()); // /a\nb/
console.log(pattern.valueOf()); // /a\nb/
```

构造函数属性

可以直接点语法 也可以用别名

```
var str = 'js js js';
var pattern = /(j)s/;
pattern.exec(str);
console.log(RegExp.input); // js js js
console.log(RegExp.$_); // js js js
console.log(RegExp["$_"]); // js js js


console.log(RegExp.lastMatch); // js
console.log(RegExp["$&"]); // js

console.log(RegExp.leftContext); // 空格
console.log(RegExp["$`"]); // 空格
console.log(RegExp.rightContext); // 空格 js js
```

```
console.log(RegExp["$'"]); // 空格 js js

console.log(RegExp.lastParen); // j
console.log(RegExp["$+"]); // j

console.log(RegExp.$1); // j
console.log(RegExp.$2); // 空格
```

search 查找匹配内容的位置

```
var str = 'html js js js';
var pattern = /js/;
console.log(str.search(pattern)); // 5

var str = 'html js js js';
var pattern = /js/g;
console.log(str.search(pattern)); // 5

var str = 'html js js js';
var pattern = /js/g;
console.log(str.search('js')); // 5
```

match

match VS exec

> match: 非全局的情况下才会返回分组中匹配到的内容，全局匹配只能匹配到所有匹配到的字符
> // Exec: 无论是否全局匹配都会返回分组中匹配到的内容，都只会返回当前匹配到的一个内容，而不是全部返回。
>
> 多行匹配(g和m联合使用)(m配合$和^使用)

1、要配合g和m使用

```
var str = 'js js js';
var pattern = /js/;
console.log(str.match(pattern)); // js
```

2、配合g使用

```
var str = 'js js js';
var pattern = /js/g;
```

```
console.log(str.match(pattern)); // ["js", "js", "js"]
```

3、配合g、m使用

```
var str = '1.js\n2.js\n3.js';
var pattern = /js/gm;
console.log(str);
console.log(str.match(pattern)); // ["js", "js", "js"]
```

4、配合g使用

```
var str = '1.js\n2.js\n3.js';
var pattern = /js/g;
console.log(str);
console.log(str.match(pattern)); // ["js", "js", "js"]
```

5、配合g和$使用

```
var str = '1.js\n2.js\n3.js';
var pattern = /js$/g;
console.log(str);
console.log(str.match(pattern)); // ["js"]
```

6、配合g、m和$使用

```
var str = '1.js\n2.js\n3.js';
var pattern = /js$/gm;
console.log(str);
console.log(str.match(pattern)); // ["js", "js", "js"]
```

7、配合g和使用

```
var str = 'js1\njs2\njs3';
var pattern = /^js/g;
console.log(str);
console.log(str.match(pattern)); // ["js"]
```

8、配合g、m和使用

```
var str = 'js1\njs2\njs3';
var pattern = /^js/gm;
console.log(str);
console.log(str.match(pattern)); // ["js", "js", "js"]
```

split方法 字符串组成数组

```
var str = 'html,css,js';
console.log(str.split(',')); // ["html", "css", "js"]
```

要配合g使用

```
var str = 'html,css,js';
var pattern = /,/;
// var pattern = /,/g;
console.log(str.split(pattern)); // ["html", "css", "js"]
```

空格处理?

```
var str = 'html,  css,  js';
var pattern = /,/;
console.log(str.split(pattern)); // ["html", "  css", "  js"]
```

前后有空格或者无空格 /\s,\s/

```
var str = 'html,  css,  js';
var pattern = /\s*,\s*/;
console.log(str.split(pattern)); // ["html", "css", "js"]
```

replace方法

替换js

```
var str = 'I love js js';
console.log(str.replace('js', 'swift')); // I love swift js
```

和g配合使用 匹配所有js

```
var str = 'I love js js';
var pattern = /js/g;
console.log(str.replace(pattern, 'swift')); // I love swift swift
```

```
var str = '1111-11-11';
var pattern = /-/g;
console.log(str.replace(pattern, '.')); // 1111.11.11
```

示例：给js加粗并且渲染红色，使用到分组$1

```
var str = 'I love js';
var pattern = /(js)/;
console.log(str.replace(pattern, '<strong style="color:red">$1</strong>')); // I lo
ve <strong style="color:red">js</strong>
document.write(str.replace(pattern, '<strong style="color:red">$1</strong>'));
```

示例：过滤敏感词，几个字几个*

```
var str = '敏感词过滤中国军队和啊好一起办证';
var pattern = /(中国军队)|啊好|办证/g;
console.log(str.replace(pattern, function($0, $1) {
    console.log($0);
    /*
    中国军队
    啊好
    办证
    */
   var result = '';
   for (let i = 0; i < $0.length; i++) {
    result += "*";
   }
   return result;
})); // index.js:472 敏感词过滤****和**一起**
```