
RegExp


> console.dir(new RegExp())
▼ /(?:)/ i
dotAll: false
flags: ""
global: false
ignoreCase: false
lastIndex: 0
multiline: false
source: "(?:)"
sticky: false
unicode: false
▼ __proto__:
▶ compile: ƒ compile()
▶ constructor: ƒ RegExp()
dotAll: (...)
▶ exec: ƒ exec()
flags: (...)
global: (...)
ignoreCase: (...)
multiline: (...)
source: (...)
sticky: (...)
▶ test: ƒ test()
▶ toString: ƒ toString()
unicode: (...)
▶ Symbol(Symbol.match): ƒ [Symbol.match]()
▶ Symbol(Symbol.replace): ƒ [Symbol.replace]()

正则的元字符

Character Classes	
.	Any character
[abc]	Any of the characters a, b, or c (same as a|b|c)
[^abc]	Any character except a, b, and c (negation)
[a-zA-Z]	Any character a through z or A through Z (range)
[abc[hij]]	Any of a,b,c,h,I,j (same as a|b|c|h|i|j) (union)
[a-z&&[hij]]	Either h, i, or j (intersection)
\s	A whitespace character (space, tab, newline, form feed, carriage return)
\S	A non-whitespace character ([^\s])
\d	A numeric digit [0-9]
\D	A non-digit [^0-9]
\w	A word character [a-zA-Z_0-9]
\W	A non-word character [^\w]

正则的元字符和修饰符

> 任何一个正则都是由 元字符 和 修饰符 组成的

修饰符

> g(global)：全局匹配
> i(ignoreCase)：忽略大小写匹配
> m(multiline)：多行匹配

元字符

> [量词元字符]
> +：让前面的元字符出现一到多次
> ?：出现零到一次
> *：出现零到多次
> {n}：出现n次
> {n,}：出现n到多次
> {n,m}：出现n到m次
>
> [特殊意义的元字符]
> \：转义字符（把一个普通字符转变为有特殊意义的字符，或者把一个有意义字符转换为普通的字符）

.：除了\n（换行符）以外的任意字符
\d：匹配一个0~9之间的数字
\D：匹配任意一个非0~9之间的数字（大写字母和小写字母的组合正好是反向的）
\w：匹配一个 `0~9或字母或_` 之间的字符
\s：匹配一个任意空白字符
\b：匹配一个边界符
x|y：匹配x或者y中的一个
[a-z]：匹配a-z中的任意一个字符
[^a-z]：和上面的相反，匹配任意一个非a-z的字符
[xyz]：匹配x或者y或者z中的一个字符
[^xyz]：匹配除了xyz以外的任意字符
()：正则的小分组，匹配一个小分组（小分组可以理解为大正则中的一个小正则）
^：以某一个元字符开始
$：以某一个元字符结束
?:：只匹配不捕获
?=：正向预查
?!：负向预查
……

除了以上特殊元字符和量词元字符，其余的都叫做普通元字符：代表本身意义的元字符

1 创建方式

1.1.1 字面量方式创建——不支持变量的匹配

```
1. //方式1：字面量方式创建------不支持变量的匹配
2.     let reg =/^\d+$/;
3.     console.log(reg.test(1234567890));//true
4.
5.     //不支持变量的匹配
6.     let reg2 = /^\d+"name"+$/;//这个的结果是： \d+出现一次或多次，“name 出现
   一次 ，”+出现一次或多次
7.     let name = "liukai";
8.     console.log(reg2.test("2liukai6"));//false
9.     console.log(reg2.test('22"name""'));//true
```

1.1.2 实例创建的方式new RegExp(‘正则字符串’)

```
1. //方式2：实例创建方式创建------注意转义字符  \\d
2.     let reg1 = new RegExp("^\\d+$");
3.     console.log(reg1.test("123123"));//true
4.     console.log(reg1.test(123123));//true
```

```
5.
6.      //支持变量的匹配
7.      let name = "liukai";
8.      let reg2 = new RegExp("^\\d+"+name+"+$");
9.      console.log(reg2.test("2liukai"));//true
10.     console.log(reg2.test('22"name""'));//false
```

2 正则中的方法

2.1 正则捕获`exec()`

2.1.1 正则的懒惰性

出现的情况

> 每次执行exec的时候，正则只捕获第一个匹配的内容，在不进行任何处理的情况下，
> 执行多次exec，都捕获的是第一次捕获的内容

```
1.      let reg =/\d+/;
2.
3.      //第1次执行exec
4.      let arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
5.      console.log(arr);//["1993", index: 5, input: "时间如流水1993我不能着急
    0515欲速则不达", groups: undefined]
6.      console.log(arr[0]);//1993
7.      console.log(reg.lastIndex);//0
8.
9.      //第2次执行exec
10.     arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
11.     console.log(arr[0]);//1993
12.     console.log(reg.lastIndex);//0
13.
14.     //第3次执行exec
15.     arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
16.     console.log(arr[0]);//1993
17.     console.log(reg.lastIndex);//0
```

正则懒惰性的原因：

> lastIndex是正则每次捕获的字符串中开始查找的位置
> 懒惰模式下，这个lastIndex的值是不发生变化的
> 所以：每次查找的都是相同的位置，结果也就相同了

解决正则的懒惰性：

> 正则末尾加g(全局修饰符)

```
//解决正则的懒惰性
    let reg =/\d+/g;
    //第1次执行exec
    let arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
    //console.log(arr);//["1993", index: 5, input: "时间如流水1993我不能着急0515欲速则不达", groups: undefined]
    console.log(arr[0]);//1993
    console.log(reg.lastIndex);//0

    //第2次执行exec
    arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
    console.log(arr[0]);//0515
    console.log(reg.lastIndex);//18

    //第3次执行exec
    arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
    console.log(arr);//null
    console.log(reg.lastIndex);//0

    //第4次执行exec  返回null之后，再次执行捕获的话，又会重新开始捕获
    /*
        arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
        console.log(arr[0]);//null
        console.log(reg.lastIndex);//0
    */
```

一次性捕获所有的方法

```
function getAllReg(reg,str){
    //捕获一次
    let result=[];
    let arr =[];
    arr=reg.exec(str);//
    while(arr){//如果不为null，就继续
        result.push(arr[0]);
        arr=reg.exec(str);
    }
    return result;
}

let reg =/\d+/g;
let str="时间如流水1993我不能着急0515欲速则不达";
//测试...
console.log(getAllReg(reg,str));//["1993", "0515"]
```

```
17.
```

2.1.2 正则的贪婪性

> 正则的每一次捕获都是按照匹配的最长的结果去捕获的
> 解决贪婪性——在量词元素后边加上一个 `?` 即可

正则的贪婪性

```
1.  console.log("---------正则的贪婪性------------");
2.      {
3.          let reg = /\d+/g;
4.          let arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
5.          console.log(arr[0]); //1993
6.          //贪婪：这样的正则匹配到1 ， 19 ， 199，1993都可以，但是它匹配的是1993，
    最长的那一个
7.      }
8.  **********************************************************************
    **************
9.  console.log("---------解决正则的贪婪性-----量词元素后加？-------");
10.     {
11.         //解决正则的贪婪
12.         let reg = /\d+?/g;
13.         let arr = reg.exec("时间如流水1993我不能着急0515欲速则不达");
14.         console.log(arr[0]); //1
15.     }
16.
17. **********************************************************************
    **************
18.     console.log("---------自定义方法解决贪婪，匹配所有-------");
19.     {
20.         function getNoGreedReg(reg, str) { //greed-贪婪
21.             let arr = [];
22.             let result = [];
23.             arr = reg.exec(str);
24.             while(arr) {
25.                 result.push(arr[0]);
26.                 arr = reg.exec(str);
27.             }
28.             return result;
29.         }
30.
31.         //测试
32.         let reg = /\d+?/g;
33.         let str= "时间如流水1993我不能着急0515欲速则不达";
34.         console.log(getNoGreedReg(reg,str));//["1", "9", "9", "3", "0",
     "5", "1", "5"]
35.     }
```

```
36.
```

2.1.3 字符串的`match`方法实现捕获

> `str.match(reg)` ——>把所有的和正则匹配的内容都捕获到
> **缺点：**match只能捕获到大正则匹配的内容，小正则的内容捕获不到

``

```
1. console.log("----------字符串的match方法捕获------------");
2. {
3.     let reg = /\d+/g;
4.     let str = "时间如流水1993我不能着急0515欲速则不达";
5. 
6.     let result = str.match(reg);
7.     console.log(result); //Array["1993","0515"]
8. 
9. }
```

2.1.4 正则的分组捕获

```
1.     /**
2.          * 正则分组的作用：
3.          * 1.改变优先级
4.          * 2.分组的引用
5.          *       \2:代表的是和第2个分组出现的字符一模一样
6.          *       \1:代表的是和第1个分组出现的字符一模一样
7.          *       \n:代表的是和第n个分组出现的字符一模一样
8.          * 3.分组的捕获：正则在捕获的时候，不仅仅把大正则的内容捕获到，而且还会把每一个小分组的内容捕获到
9.          *
10.      */
11.         //身份证号码
12.         let reg = /^(\d{6})(\d{4})(\d{2})(\d{2})\d{2}(\d)(X|\d)$/;
13.         //             610126--1234---22-----22---55---5---x或2
14.         //               区号---年-----月-----日---不知道-男女--不知道
15.         let str = "12345612342222555X";
16.         let str2 = "2345612342222555X2";
17. 
18.         let arr = reg.exec(str);
19.         let arr1 = reg.exec(str2);
20.         console.log(arr);
21.         //["12345612342222555X", "123456", "1234", "22", "22", "5", "X", index: 0, input: "12345612342222555X", groups: undefined]
22.         console.log(arr1); //null
23. 
```

```
        //使用match
        let arr3 = str.match(reg);
        console.log(arr3);
        //["12345612342222555X", "123456", "1234", "22", "22", "5", "X", index: 0, input: "12345612342222555X", groups: undefined]
        //发现match跟reg.xexc捕获到的一模一样
        //区别在哪？------- 倒数第三个小正则和倒数第一个我们不想要
        //不要匹配它们  --（?:\d）小分组中加上？：   只匹配，不捕获
        let reg2 = /^(\d{6})(\d{4})(\d{2})(?:\d{2})\d{2}(\d)(?:X|\d)$/;
        let arr4 = reg2.exec(str);
        //(5) ["12345612342222555X", "123456", "1234", "22", "5", index: 0, input: "12345612342222555X", groups: undefined]
        console.log(arr4);
```

2.1.5 exec 和 match`的区别

`exec` 和 `match` 的区别

> 在正则分组，并且字符串有多个匹配结果的情况下：
>
> - match捕获的结果只是大正则所匹配的所有结果，
> - exec捕获的是大正则和小正则的结果

```
let reg =/liukai(\d+)/g;
let str = "liukai123liukai456liukai789"

//使用match捕获
let arr=str.match(reg);
console.log(arr);//["liukai123", "liukai456", "liukai789"]

//使用exec捕获
let arr1= reg.exec(str);
console.log(arr1);// ["liukai123", "123", index: 0, input: "liukai123liukai456liukai789", groups: undefined]
arr1= reg.exec(str);
console.log(arr1);// ["liukai456", "456", index: 9, input: "liukai123liukai456liukai789", groups: undefined]
arr1= reg.exec(str);
console.log(arr1);//["liukai789", "789", index: 18, input: "liukai123liukai456liukai789", groups: undefined]

```

2.2 正则捕获 `replace`

2.2.1 `str.replach(a,b)` 的原理

```
    //str.replach(a,b)的原理
    //把字符串str中a样式的字符串用b替换掉-------他只替换一次
    let str ="BigSpinach love program";
    //需求把所有的i替换成 “哈哈”
    let result= str.replace("i","哈哈");
    console.log(result);//B哈哈gSpinach love program

    //replace他是懒惰的，不会搞后边的
    let result2= str.replace("i","哈哈");
    console.log(result2);//B哈哈gSpinach love program
```

2.2.2 `replace` 中使用正则

```
//replace中使用正则
let str ="BigSpinach love program";
let reg = /i/g;
let result = str.replace(reg,"哈哈");
console.log(result);//B哈哈gSp哈哈nach love program

```

2.2.3 `replace` 中使用匿名函数操作 `str`

`replace的关键点`

> 1.function()函数执行几次取决于字符串中的字符跟正则匹配成功了几次
> 2.function()中的arguments输出的结果跟 reg.exec捕获到的结果非常的相似，
> 即使正则分组，我们也可以通过arguments获取到分组捕获的内容
> 3.在function中的return，return的是啥，就表示用啥替换掉当前大正则所捕获到的内容

```
let str ="BigSpinach love program";
let reg = /i/g;
let result = str.replace(reg,function(){
    console.log(arguments);//这里会执行两次是因为正则匹配到了两次 i
    //Arguments(3) ["i", 1, "BigSpinach love program", callee: ƒ, Symbo
l(Symbol.iterator): ƒ]
    //Arguments(3) ["i", 5, "BigSpinach love program", callee: ƒ, Symbo
l(Symbol.iterator): ƒ]
    return "123";
});
console.log(result);//B123gSp123nach love program
    /*replace的关键点
        1.function()函数执行几次取决于字符串中的字符跟正则匹配成功了几次
        2.function()中的arguments输出的结果跟 reg.exec捕获到的结果非常的相似，
            即使正则分组，我们也可以通过arguments获取到分组捕获的内容
```

```
        3.在function中的return，return的是啥，就表示用啥替换掉当前大正则所捕获到的内容
    */
```

2.2.4 `replace`练习——获取字符串中看每一个字符出现的次数以及那个字符出现的次数最多

```
let str="I'm BigSpinach,BigSpinach love program";
    //let reg = /(')\{1\}|(\D)\{1\}|(\s)\{1\}/;
    let reg = /(\D)\{1\}|(\s)\{1\}/g;
    var obj={};
    let result = str.replace(reg,function(largeReg,smallReg,index,input){
    //获取到每一个字符
        //console.log(arguments);
        //["I", "I", undefined, 0, "I'm BigSpinach,BigSpinach love program", callee: ƒ, Symbol(Symbol.iterator): ƒ]
        var character = arguments[0];
        //console.log(smallReg);===arguments[1]
        //将每一个字符当做obj的一个属性添加进去
        /*
        if(obj[character]>=1){
                obj[character]+=1;
            }else{
                obj[character]=1;
        }
        */
        obj[character]>=1?obj[character]+=1:obj[character]=1;

    });
    //console.log(obj);//{I: 1, ': 1, m: 2, " ": 3, B: 2, …}
    //console.log(result);

    //用假设法找出最大的value
    let maxNum =0;
    for(let key in obj){
        obj[key]>maxNum?maxNum=obj[key]:null;
    }
    //console.log(maxNum);
    //把所有符合maxNum的key都找到
    let arr = [];
    for (let key in obj) {
        obj[key]==maxNum? arr.push(key):null
    }
    //console.log(arr);
    console.log("最多出现的字符是："+arr.toString()+";次数是："+maxNum+"。");
```

2.2.5 replace 实现模板引擎的初步思想

```
let str ="My name is {0},我今年{1}岁,i love {2}。";
let reg = /{(\d+)}/g;
let arr=["刘凯", 26, "Program"];
str = str.replace(reg,function(){
    return arr[arguments[1]];
    //return arr[RegExp.$1];//IE不兼容+Chrom也不行了
});

console.log(str);
//My name is Program,我今年Program岁,i love Program。
//My name is 刘凯,我今年26岁,i love Program。
```

2.2.6 使用replace拆分url

```
let url = "http://202.110.112.57/wgdcnccdn.inter.qiyi.com/videos/v0/20180606/c4/4c/a30dc18f27dfa6132bd8fe1dcf26d094.f4v?key=02e6e69ca23f0701c3dabd879dee04bdd&dis_k=bbeb790ff8c36f16b4137fc06d1a0376&dis_t=1528293384&src=iqiyi.com&uuid=7b8b65d4-5b17e808-45&rn=1528293380858&qd_tm=1528293380953&qd_tvid=1078238100&qd_vipdyn=0&qd_k=6828f2fd943ec1c1048d9e790e692b33&cross-domain=1&qd_aid=206231501&qd_uid=&qd_stert=0&qypid=1078238100_020200310100000000000&qd_p=7b8b65d4&qd_src=01010031010000000000&qd_index=1&qd_vip=0&qyid=25b1a7a60a4371a0f72548d731796b3f&pv=0.1&qd_vipres=0&range=1461248-22526975&wshc_tag=0&wsts_tag=5b17e833&wsid_tag=7b8b65d4&wsiphost=ipdbm";

//使用正则
let reg = /([^?=&]+)=([^?=&]+)/g;
let obj={};
url.replace(reg,function(){
        obj[arguments[1]]=  arguments[2];
});
console.log(obj);
```