正则

正则表达式

正则的写法

转义符 \

正则身上的方法 exec test

test（字符串）查看字符串中是否有规则匹配项，如果有就返回true，否则就是false

match 是字符串的方法

string.replace( ) 替换 过滤敏感词

[ ]

{ }代表量词

^ 开头

？ 一系列

边界符 \b

\w： 代表 数字 字母 下划线

m : 多行匹配

重复子项 \数字

千分符，

把网址用正则变成对象

正则

正则表达式

- 定义：专门用来检索字符串的一种规则（更擅长处理模糊范围的字符串）
- 懒惰：你让他找一个，绝对不会找第二个，你让他找一堆，不会找第二堆
- 贪婪：只要符合这个规则就不停的找，找不到为止

正则的写法

- new RegExp（/规则/修饰符） 规则可以为字符串,不带引号的，也可以为字符串拼接
- new RegExp（字符串或者变量 ， 修饰符）

```
let str = 'abc';
let aa = 'a';
//什么时候用new RegExp
```

```
console.log(/aa/); //找的是字符aa
console.log(new RegExp(aa)); //找的是变量aa，其实就是找字符串‘a’
```

转义符 \

```
let str = 'a\da'
console.log(\d);   //  /d/
console.log(\\d);  //  \d
```

正则身上的方法 exec test

- exec（字符串）找到正则匹配的字符串，（首次出现的字符（就一次），并且放在数组中）
- exec返回的是数组

```
0: "8"    字符
index: 2
input: "dh819dx1"
groups: undefined
length: 1

let str ='dh819dx1';
console.log(/1/.exec（str）); // 查看字符串中有没有正则匹配项
```

```
let str ='dh819dx1';
console.log(/\d/.exec（str）)  //  \d  找到正则字符串中首次出现的数字，只找一次 ['8']
console.log(/\d+/.exec（str）)  //\d+ 找到正则字符串中首次出现连续的数字，只找一次 ['819']
onsole.log(/\d+/g.exec（str）) // /\d+/g 找到正则字符串中首次出现连续的数字，只找一次，并且最重要的是x后面的1中不到，有局限性  ['819']
```

test（字符串）查看字符串中是否有规则匹配项，如果有就返回true，否则就是false

- test返回的是布尔值

```
let str ='dh819dx1';
```

```
console.log(/z/.test（str）); // 查看字符串中有没有z 有就是true 没有就是false
```

match 是字符串的方法

- string.match (/ /) 是字符串的方法
- 找到正则匹配的字符并且把他们放在数组中（返回值是数组）跟exec类似
- 找不到返回null

```
let str ='819xxx16641';
console.log(string.macth(/\d/)); //找到字符串第一个数字
console.log(string.macth(/\d+/)); //找到字符串多个数字,注意xxx后面的找不到
console.log(string.macth(/\d+/g)); //找到字符串多个数字,xxx后面的也可以找的到
```

```
let str ='819xxx166Xx41';
console.log(string.macth(/z/ig)); //全局查找，忽略大小写
```

string.replace( ) 替换 过滤敏感词

- 括号里可以放 （字符串 正则 ，‘替换字符’ 函数）
- 注意： replace 必须配合分组使用（）
- 功能：
  - 可以把要替换的东西替换
  - 可以把元素包起来，然后在前后面加东西
- replace细节问题：默认情况下，函数形参，第一个参数是每次匹配的字符，第二个参数是匹配的索引，第三个参数是整个字符串，第三个参数是undefined
- replace细节问题：分组（）情况下， 第一个参数是每次匹配的字符，第二个参数是第一个分组，第三个参数是第二个分组，第三个参数是第三个分组，依次类推，分组结束以后还有&按默认情况下运算，从第二项开始
- 分组从左往右数

```
let str = '2019/11/12';
let s= str.replace(/(((\d)+)\D+(\d+)\D+(\d+)/,function(&0,&1,&2,&3){
return &1+'年'+&2+'月'+&3+'日';
})
console.log(s)

let str = '2019/11/12';
(((\d)+)  //9
(\d+)   //2
```

```
分组中的提权功能
（1+1）*5 //先计算括号里面的
```

```
let str = '珠枫'
console.log(string.replace('枫','峰'));
```

```
let str = '1珠226枫3';
console.log(string.replace(/\d+/g,''));
```

```
let str = '1珠226枫枫枫3';
console.log(string.replace(/珠|枫枫枫/g,'*')); //也可以用来过滤敏感词,敏感词不管有几个字都是一颗*，因为写的是一颗星
```

```
let str = '1珠226枫枫枫3';
console.log(string.replace(/珠|枫枫枫/g,function($0){
    let temp = '';
    for( let i=0;i<$0.length; i++){
     temp +='*';
    }
     return temp
     }));     // 过滤敏感词,敏感词有几个字就是几颗*
```

[ ]

- 中文的区间范围 [\u4e00 - \u9fa5]
- []中的字符在正则中是找任意一个字符
- [123] -> 要么找1要么找2要么找3
- 也可以使用多少 - 多少的写法来写
- [0-9] -> \d ascll码来编排的
- 小写英文 : [a-z]
- 大写英文 : [A-Z]
- 如果要拿到大写和小写的字符[A-z] × 因为acsll码的91-96是别的字符不算字母 要[A-Za-z]写 √
-
- $:从字符串末尾进行匹配

```
let str = 'a1ca2ca3ca4c'
console.log(/a\dc/g)  //找a开头c结尾的数字
```

```
console.log(/a(1|2|3)c/g) //找a开头c结尾的数字
console.log(/a[1|2|3])c/g) //找a开头c结尾的数字
console.log(/a[123])c/g) //找a开头c结尾的数字
console.log(/a[1-3])c/g) //找a开头c结尾的数字
```

```
let str2 = 'a1cA2ca3cb4c'
需求：找a-z之间的的数字
console.log(str2.match(/[]\d[]/g)); //错的
console.log(str2.match(/[A-Za-z]\d[A-Za-z]/g)); //对的
\d中间是一个数字
```

```
16-108才能注册,是就是true不是false
let str = '108'
16-19=>1[0-9]
20-99=>[2-9][0-9]
100-108=>10[0-8]
console.log(str2.match(/^(1[0-9]|[2-9][0-9]|10[0-8])$/.test(str));

^ 开头
$ 结尾
| 或者
（）加括号是一个整体
1[0-9] 0-9的任意数 //10 11 12 13...19
 ^ + $ = 整个字符串都要匹配我的规则
```

```
计算出下列字符字节，假设英文是1个字节，中文是2个字节
let str = '大家好，是兄弟就来砍我，come.on!'; //30个字节
let num =0; //计数
for( let i=0;i<str.length;i++){
 if(/[a-z\.,!]/i.test(str[i])){
 num++;
 }else{
 num +=2;
 }
}
console.log(num);
```

{ }代表量词

- 描述{}前面字符的数量
- {m,} 最少m个，最多不限
-
  - -> {1,} 最少出现 1或多次

- {m,n} -> {2,5} 最少出现2次。最多出现5次
- {n} 最少出现n次，最多也是出现n次或者 最少有n位数，最多有n位数
- ? 最少可以没有，最多出现1次 -> {0,1}
- ◦ 最少可以没有，最多无限 -> {0,}

```
let str = 'a12ca456ca6789ca123456caca1ca78c';
 console.log(str.match(/a\d{0,3}c/g)) //a开头c结尾，中间数字最少出现0次，最多出现三次
 console.log(str.match(/a\d{2}c/g)) //a开头c结尾，中间数字最少出现2次，最多出现2次
 console.log(str.match(/a\d?c/g))  //a开头c结尾，最少可以没有，最多出现1次
 console.log(str.match(/a\d{0,1}c/g)) //a开头c结尾，最少可以没有，最多出现1次
 console.log(str.match(/a\d*c/g))  //a开头c结尾，最少可以没有，最多不限
 console.log(str.match(/a\d{0,}c/g)) //a开头c结尾，最少可以没有，最多不限
```

^ 开头

- ^如果在中括号里面，那么就代表排除，不是代表开头
- ^从字符串开头进行匹配

```
let str = 'a12ca456ca6789ca123456caca1ca78c'
console.log( str.match(/a[^123]+c/g)); //a开头c结尾，中间排除1 2 3
```

```
把标签删除，最后留下了文字 ，  也称洗数据
let str2 = '<span class="money clone">北京19990/月</span>';
let s = str.replace(/<[^>]+>/g,'').replace(/\n/g,'');
console.log(s.match(/\d+\//g).join('').split('/'));

(?=(?:\/) 获取但不显示/

整个字符串都要符合123才被匹配，但是只要12
 console.log(/12(?=(?:3))/.exec('123'));  //12
```

? 一系列

- ? <= 后瞻仰 (找括号后面的字符)
- ? = 前瞻仰 （找括号前面的字符）
- ? ！ 负前瞻仰 （找括号前面的字符）
- ? <! 负后瞻仰 (找括号后面的字符)

```
let str ='中国人美国人德国人英国人'
console.log(str.replace(/(?=<中国人)/g,'神'))
console.log(str.match(/(?=<中国)人/g))
console.log(str.replace(/中国人/g,（$0)=>$0.substr(0,2)+'神'))
```

```
正则验证手机号
console.log(/^1[3-9]\d{9}$/.test())
正则验证QQ号
let str=2212802552;
console.log(/^[1-9]\d{5,10}$/.test())
```

```
去掉前后空格
let str = ' d n s ';
console.log(str.repalc(/^\s+|\s+$/g,''));

把所有不是空格的变成*号
console.log(str.repalc(/^\S+/g,'*'));
```

边界符 \b

- 英文字符串开头有一个边界符，字母和字母之间是没有边界符的
- 中文左边有边界符，右边没有边界符（）
- 整个字符串都是中文，是没有边界符的

- 中英文结合的情况下中文的左右边有边界符，右边没有边界符
-

```
let str = ' d n s ';
```

\w： 代表 数字 字母 下划线

```
6-18个字符，可以使用字母，数字，下划线，需要以字母开头
console.log(/^[a-zA-z]\w{5,17}$/.test())
console.log(/^[a-zA-z][0-9A-Za-z_]{5,17}$/.test())

验证邮箱
let str = 'xxx-love@163.com'
console.log(/^[a-zA-z][\w.-]{5,17}@$[0-9]{0,3}\.c(om|n)/.test())
```

m：多行匹配

定义：配合^和$使用，只想要开头结尾的元素，中间相同的元素不要

```
let str='某th 某is某
         某th 某is某
         某th 某is某
'
console.log（/^某/）//第一行开头的某
console.log（/某$/） //第三行结束的某
console.log（/^某$/g） //第一行开头的某，第三行结束的某
console.log（/^某$/m） //第一行开头的某，第一行结束的某
console.log（/^某$/gm）//三行开头的某，三行结束的某
```

重复子项 \数字

- \数字 （子项的个数）-> \1 \2 数字代表第一个子项 和第二子项，这个\数字一定是和子项内容一致的
- 重复子项只能用在匹配子项之后 aa-> (a)\1 对的 aba-> (a)\1 错的
  - \1 \2 数字代表第一个子项和一个重复子项 和第二个子项和一个重复子项
  - \1+ \2+ 数字代表第一个子项和无限重复子项 和第二个子项和无限重复子项

```
let ary='aaabbbccc'
console.log(/ (a)\1(b)\2(c)\3/)  //aabbcc
// \1  \2 数字代表第一个子项和一个重复子项 和第二个子项和一个重复子项
console.log(/ (a)\1+(b)\2+(c)\3+/)  //aaabbbccc
// \1+  \2+ 数字代表第一个子项和无限重复子项 和第二个子项和无限重复子项

let ary='aaacccbbb'
console.log(/ (a)\1(c)\2(b)\3/)  //aabbcc
console.log(/ (a)\1+(c)\2+(b)\3+/)  //aaabbbccc
```

千分符，

```
let str = '1000000';
console.log(/\d(?=\d{3})/.exec(str));  // 1  从左往右说，三个数字前面的数，也就是1

要求：从后往前找，把3个数字前面的数字后加上一个，号
let str = '1000000';
console.log(str.replace(/(\d)(?=(\d{3})+$)/g,'$1,')); //  +$ 用加号说明只要遇到三个数字前面的都加逗号，要是不加+号，只匹配一次， $用数字结尾，$1，在每个分组后面加逗号 （\d） 不加分组不能加东西或者替换东西，加了$的意思是从结尾开始找(从字符串的最后开始查找)

要求：从后往前找，把3个数字前面的数字后加上一个，号
```

```
      let temp = '';
      let num = 0;
let str = '1000000';
    for(let i=str.length-1;i>=0;i--){
        if((temp.length-num)%3){
            temp += str[i];
        }else{
            temp += ',' +str[i];
            num++;
        }
    }
    console.log(temp);
    temp = temp.substring(1);
    temp = temp.split('').reverse().join('');
    console.log(temp);
```

把网址用正则变成对象

```
let str2 = 'ie=utf-8&f=8&rsv_bp=1&rsv_idx=1&tn=baidu&wd=js%20全角转半角%20正则&oq=%25E5%2585%25A8%25E8%25A7%2592%25E8%25BD%25AC%25E5%258D%258A%25E8%25A7%2592%2520%25E6%25AD%25A3%25E5%2588%2599&rsv_pq=ed9277970000f747&rsv_t=719cvLoB2Cy7pOW4%2F1hf3xOXnlRxJsSEPLA5xAfCoQ5RkeA6ruGTQ0OPjo4';

console.log(str2.match(/([a-z]\w{0,10}=[\w%-]+&?)+/))

思路：因为对象是有属性名和属性值，可以把等号前面的当属性名，等号后面的当属性值，然后用分组，循环赋值
let re = /^(http)s?:\/\/w{3}\.[a-z]{2,18}(\.[a-z]{2,3}){1,2}\/[a-z]{1,10}\?(([a-z]\w{0,10})=([\w%-]+)&?)+$/;

let re2 = /([a-z]\w{0,10})=([\w%-]+)/g;  //用分组把等号前面的和等号后面的括起来，正则里面替换必须先要分组

    let obj = {};
         str2.replace(re2,(...arg)=>{
             let a = arg[1];
             let b = arg[2];
             obj[a] = b;
         });
         console.log(obj);
```

- \：转义字符 把正则中有特殊含义的字符 转成字符本身(不再有任何特殊的含义)，转义符转的是后面的字符

- \ :第二个\代表本身
  \d： 代表了 0-9之间的数字
  \D： 代表除了 0-9 的任意字符;
  \w： 代表 数字 字母 下划线;
  \W： 代表除了 数字 字母 下划线 之外的任意字符
  \s : 一个空格
  \S： 一个非空格
  \b： 一个边界符
  \B： 一个非边界符
  ^： 代表以什么 字符 开头
  $： 代表以什么 字符 结尾
  .： 代表除了换行以外的所有字符
  \n： 代表换行
  x|y： 代表 x 或者 y
  [ab]： 代表a或者b
  [^ab]： 代表非ab
  [a-z]： 代表 a-z之间的任意字符
  [^a-z]： 代表除了 小写字母
  ()： 分组和提升优先级的意思
  (?:)： 非捕获 匹配
  (?=)
  (?!)
- 量词元字符：一般都是用在了其他元字符的后边
  ？： 代表 前边的字符出现 0或1次
  + : 代表 前边的字符出现 1或多次
  * : 代表 前边的字符出现 0或多次
  {n}： 代表 前边的字符出现 n次
  {n,m}： 代表 前边的字符出现 n到m次
  {n,}： 代表 那边的字符出现 n到多次
- 修饰符： 有多个修饰符，顺序无所谓
  i : 忽略大小写 ignoreCase
  m : 多行匹配 mutiline
  g : 全局匹配 global