**正则 :** 也叫做规则，让计算机能够读懂人类的规则（正则都是操作字符串的）

用处：比如注册，用户名和邮箱的格式验证

**创建方式**：

1、构造函数new RegExp(pattern(检索的内容), options（修饰符）);

修饰词：i：不区分大小写 ； g：全局匹配

如：

var str = "hello word";

var reg = new RegExp("o","ig");

2、字面量方式创建

var reg1 = /o/ig;

**正则提供的方法，两种**

test()

概念：正则去匹配字符串，如果匹配成功就返回真，如果匹配失败就返回假

写法：正则.test(字符串)

解释：这个方法的最终返回结果就是一个布尔值。所以一般的用法就是用来做判断的。

注意：不要对这个方法使用g标识

如：console.log(reg1.test(str)); // true

exec()

具体执行检索匹配，返回数组

console.log(reg1.exec(str)); // ["o"]

console.log(reg1.lastIndex); // 下一次要查找的下标

console.log(reg1.exec(str)); // ["o"]

console.log(reg1.lastIndex); // 下一次要查找的下标

**字符串提供的方法，四种**

//1.match 返回匹配的结果，是一个数组，如果不成功，返回null

console.log(str.match(reg1)); //["o", "o"]

//2.replace 替换，有两个参数，会把匹配上的替换成第二个参数

console.log(str.replace(reg1, "\*\*\*\*\*")); // hell\*\*\*\*\* w\*\*\*\*\*rd

//3.split，以匹配上的字符拆分成数组

console.log(str.split(reg1)); // [ "hell", " w", "rd" ]

//4.search，返回找到元素的下标，只查找一次

console.log(str.search(reg1)); // 4

**转义字符**

. : 除了换行符以外的任意字符

var str = "wab";

var reg = /w.b/g;

console.log(reg.test(str)); // true

[] 字符类，匹配中括号范围内的任意字符

var str1 = " WEB";

var reg1 = /[a-zA-Z0-9]/g; //匹配a-zA-Z0-9之间任意一个字符

console.log(reg1.test(str1));

^ 在[]中表示取反

var str1 = " WEB";

var reg2 = /[^a-zA-Z0-9]/g; //匹配除了a-zA-Z0-9之间任意一个字符

console.log(reg2.test(str1)); // true，因为有空格

\d:匹配数字 \D:匹配非数字

var str2 = "1016";

var reg3 = /\D/g;

console.log(reg3.test(str2)); // false

\w:数字、字母、下划线 \W:非数字、字母、下划线

var str3 = "haha";

var reg4 = /\w/g;

console.log(reg4.test(str3)); // true

^ 行首 $行尾

var str5 = "194421";

var rge6 = /^1/ig;

// var rge6 = /2$/ig;

console.log(rge6.test(str5));

{4,7} : 最少出现4次，最多出现7次

{4,} : 最少出现4次，最多不限制

{4} : 正好出现4次

{1,} : 最少出现一次。简写为：+

{0,} : 至少出现0次。简写为：\*

{0,1} : 出现0次或者1次。简写为：？

+和\*的区别：+代表必须有，\*代表可以没有。

? 匹配前面的字符0次或一次

var str6 = "web";

var reg7 = /w?/ig;

console.log(reg7.test(str6));

\* 匹配0次或任意多次 尽量多的匹配

var str6 = "124309496hah958054fkjjfuio9080945"; //数字--\*\*

var reg7 = /\d\*/ig;

// console.log(reg7.test(str6));

console.log(str6.replace(reg7, "\*"));

+ 至少有一次

var str6 = "124309496hah958054fkjjfuio9080945"; //数字--\*\*

var reg7 = /\d+/ig;

console.log(reg7.test(str6));

console.log(str6.replace(reg7, "\*"));

{x,y} 至少匹配x次，至多y

var str7 = "12345678950";

var reg8 = /^1\d{10}$/ig; //以1开头，后面接10位数字

console.log(reg8.test(str7)); // true

| 或者

var str8 = "124@qq.com";

var reg9 = /.+@(qq.com)|(163.com)$/;

console.log(reg9.test(str8));

() 分组

var s = "17614930409";

var reg = /^1\d{2}(\d{4})(\d{4})$/;

s.replace(reg, function($0, $1, $2) {

// $0为正则整体

// $1为第一个小括号

// $2为第二个小括号

console.log($0, $1, $2);

})

匹配中文：[\u4e00-\u9fa5] 有事一00，有酒罚我

行首行尾空格：/^\s+|\s+$/g

Email：^\w+@[a-z0-9]+(\.[a-z]+){1,3}$

网址：[a-zA-Z]+://[^\s]\*

QQ号：^[1-9][0-9]{4,9}$

身份证：[1-9]\d{14}|[1-9]\d{17}|[1-9]\d{16}x

电话号码：/^0\d{2}-[^0]\d{6,7}$/i

邮箱：now999@qq.com.cn.net（用这个来说明怎么匹配上邮箱的）

案例：

判断输入的是否是QQ号

去掉字符串前后空格