# 1. 正则的使用

## 1.1 01-正则简介

**1、创建正则的两种方法**

a)、通过构造函数创建 var regEx=new RegExp( /\d/ )

b)、通过字面量 var regEx2=/\d/

**2、正则表达式的语法**

a) // 所有的正则格式都写在//中间 即：/ \d/

b) \d 预定义类 表示 ：数字

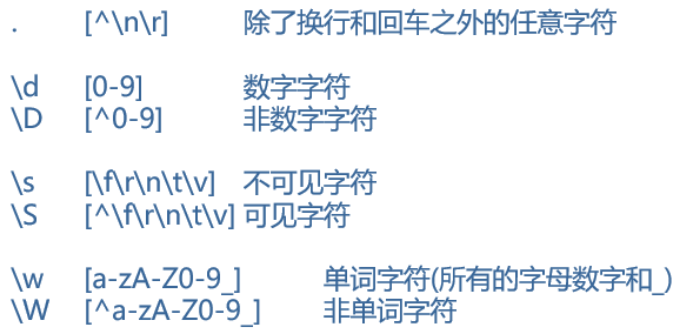
**3、判断正则是否正确**

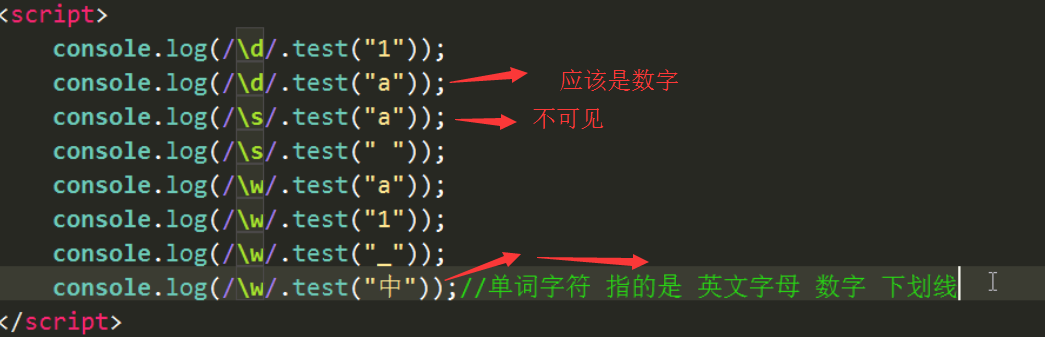
* 1. regEx.test(“1”) 返回值：true
  2. regEx.test(“a”) 返回值：false

c) regEx.test(“a1”) 返回值：true 说明只要有数字就返回true

## 1.2 02-预定义类

**1、其他预定义类**





## 1.3 03-基础语法补充

**1、test()输入的和正则规定的语法存在的关系**

**只能多，不能少**

console.log(/chuan/.test("chuan"));  
console.log(/chuan/.test("chuang"));  
console.log(/chuan/.test("chua")); 可以多 不能少 只要有就行

**2、|和()和[]语法的使用区别**

a) | 表示”或” 优先级别最低

b) () 可以提升优先级别

c) [abc] 表示“或”即：a或b或c

console.log(/foot|boot/.test("foot")); 表示foot或者boot  
console.log(/foot|boot/.test("boot")); 表示foot或者boot  
console.log(/(foot)|(boot)/.test("boot"));表示foot或者boot 级别要高点  
console.log(/f|boot/.test("boot")); 表示f或者boot  
console.log(/(f|b)oot/.test("boot")); 表示foot或者boot  
console.log(/a1|b1|c1/.test("boot")); 表示a1或b1或c1  
console.log(/a|b|c1/.test("boot")); 表示a或b或c1  
console.log(/(a|b|c)1/.test("boot")); 表示a1或b1或c1  
console.log(/[abc]1/.test("boot")); 表示a1或b1或c1

## 1.4 04-字符类

**1、几种类别及语法**

a) 简单类

console.log(/[abc]/.test("a"));  
console.log(/[abc]/.test("b"));  
console.log(/[abc]/.test("c"));  
console.log(/[abc]/.test("d")); false a或者b或者c都可以 其他的不行  
console.log(/[abc]/.test("da")); true 只要有我想要的就可以

b) 负向类 ^表示相反

console.log(/[^abc]/.test("a")); 不能是a或b或c  
console.log(/[^abc]/.test("b")); 不能是a或b或c  
console.log(/[^abc]/.test("c")); 不能是a或b或c  
console.log(/[^abc]/.test("d")); true 不能是a或b或c d可以  
console.log(/[^abc]/.test("da")); true 只要有我想要的就可以

c) 范围类

console.log(/[abcdefghi]/.test("a"));   
console.log(/[a-i]/.test("a")); 表示范围a-i  
console.log(/[a-z]/.test("a")); 表示所有小写字母  
console.log(/[A-Z]/.test("a")); 表示所有大写字母  
console.log(/[0-9]/.test("a")); 表示所有数字

d) 组合类

console.log(/[a-zA-Z0-9]/.test(""));表示所有的大小写字母还有数字  
console.log(/[a-kA-N0-5]/.test(""));组合类中的范围自定义

## 1.5 05-验证密码强度

**步骤分析：**

1、动态输入动态获取内容，判断其级别 自定义一个级别arr=[“ ”,”低”,”中”,”高”]

2、键盘弹起触发事件

3、判断级别

* 1. 先申明一个安全级别初始为level=0
  2. 如果有a-z就让级别自增
  3. 如果有0-9就让级别自增
  4. 如果有^a-z0-9就在自增
  5. 判断输入的长度<6级别就还是level=0

4、把输入的值以及级别渲染在html中

* 1. 级别通过数组arr[] ,值通过level自增，下标来获取
  2. 进度通过字符串拼接

**var** inp1 = document.getElementById("inp1");  
**var** strength = document.getElementById("strength");  
**var** strengthLevel = document.getElementById("strengthLevel");  
**var** arr = ["", "低", "中", "高"];  
inp1.onkeyup = **function** () {  
 **var** level = 0;//记录安全级别  
 **if** (/[a-z]/.test(**this**.value)) {  
 level++;//如果有小写字母级别+1  
 }  
 **if** (/[0-9]/.test(**this**.value)) {  
 level++;//如果有数字级别+1  
 }  
 **if** (/[^a-z0-9]/.test(**this**.value)) {  
 level++;//如果有其他的字符级别+1  
 }  
 **if** (**this**.value.length < 6) {  
 level = 0;  
 }  
 strength.innerHTML = arr[level];  
 strengthLevel.className = "strengthLv" + level;  
};

## 1.6 06-边界

**1、/^a/ 和 [^a] 的区别**

a) /^a/ 表示以什么开头 表示：a开头

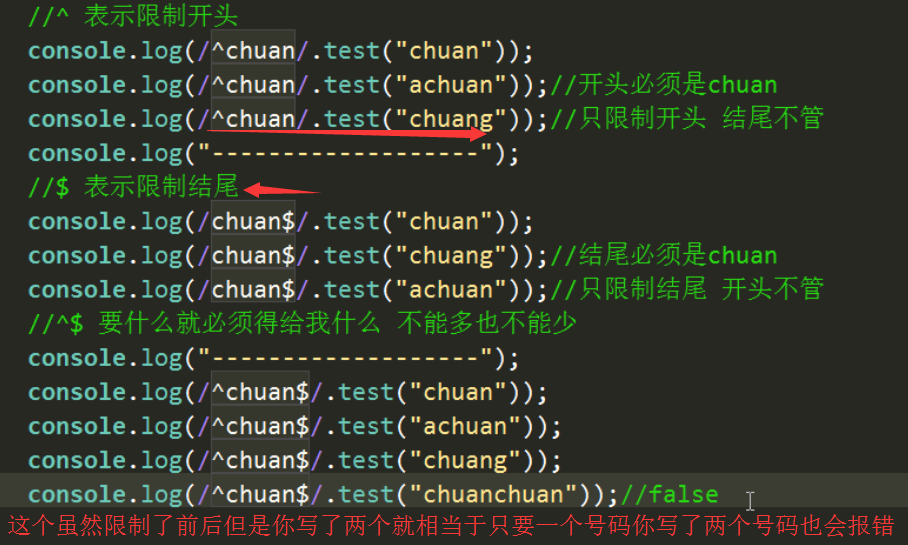
b) [^a] 表示非 表示：不是a

**2、/^a/ 、/a$/ 、/^a$/ 的区别**

a) /^a/ 表示以a开头不管结尾

b) /a$/ 表示以a结尾不管开头

c) /^a$/ 表示以a结尾并且以a结尾



## 1.7 07-量词

**1、量词的定义**

a) "\*" 重复零次或更多 x>=0

b) "+" 重复一次或更多次 x>=1

c) "?" 重复零次或一次 x=(0||1)

d) {n} n次 x=n

e) {n,} 重复n次或更多 x>=n

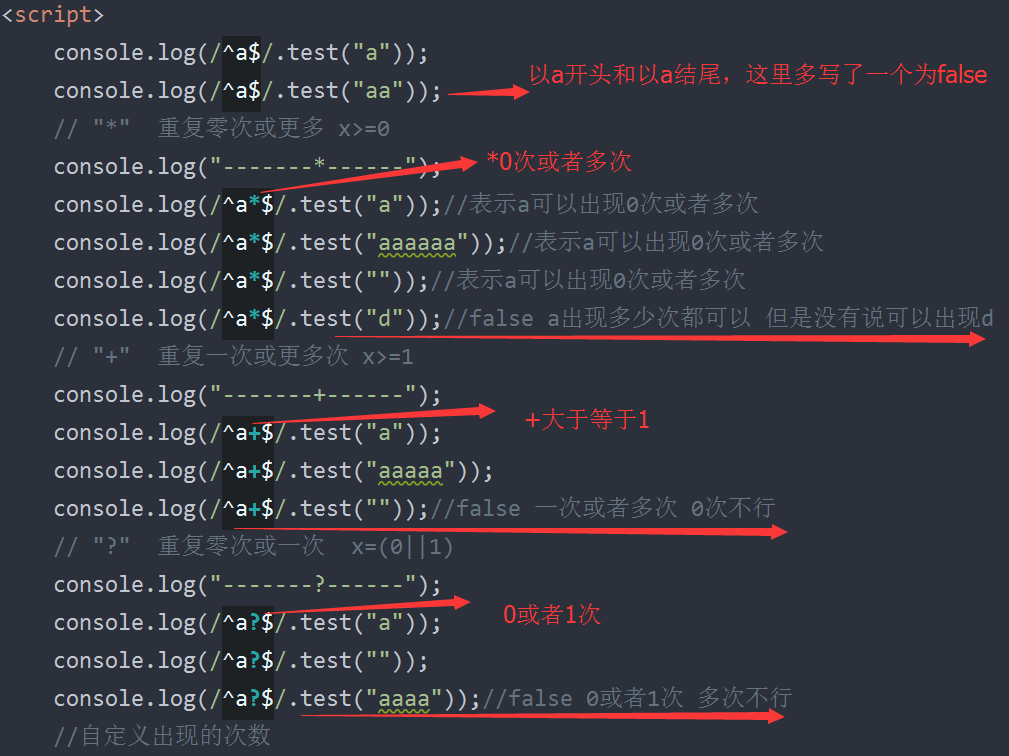
f) {n,m} 重复出现的次数比n多但比m少 n<=x<=m

**2、\*,+,?分别用{}如何表示**

a) \* = {0,}

b) + = {1,}

c) ? = {0,1}



## 1.8 08-括号总结

**总结：**

a) {}表示次数

b) []表示一个字符的位置

c) ()表示一组



# 2. 正则匹配案例

## 2.1 09-验证座机

**1、验证注意事项**

a) 开头和结尾都需要严格的限制 即开头/^ $/

b) 判断 如果输入的值和规定的正则匹配就弹出正确，否则就弹出错误

c) 一个/d只能代表一个数字

**var** inp1=document.getElementById("inp1");  
inp1.onblur=**function**(){  
 **var** regEX=/^0\d{2,3}-\d{7,8}$/;  
 **if**(regEX.test(**this**.value)){  
 alert("正确");  
 }**else**{  
 alert("错误");  
 }  
};

## 2.2 10-验证姓名

**1、所有中文字的范围**

a) [\u4e00-\u9fa5]

b) 他是nuicode编码，我们要识别可通过 unescape('\u4e00')来转码

**2、姓名的验证**

a) 申明是文字以及字数多少 要有开头和结尾

b) 判断当前的值和正则匹配，成功就显示正确，当前的兄弟元素添加类名

**var** inp1 = document.getElementById("inp1");  
inp1.onblur = **function** () {  
 **var** regName = /^[\u4e00-\u9fa5]{2,6}$/;  
 //根据姓名的规律去判断是否符合要求 如果符合显示正确 否则显示错误  
 **if** (regName.test(**this**.value)) {  
 **this**.nextSibling.innerHTML = "正确";  
 **this**.nextSibling.className = "right";  
 } **else** {  
 **this**.nextSibling.innerHTML = "错误";  
 **this**.nextSibling.className = "wrong";  
 }  
};

## 2.3 11-完整版表单验证

1. **验证判断的封装**

a) 自定义一个方法，对象和规则是需要变换的，所以需传入这个两个参数

b) 对象失去焦点和当前的值都是和当前的规则判断，

c) 调用时传入当前的对象和当前的规则

**function** check(inp,reg){  
 inp.onblur=**function**(){  
 **if**(reg.test(**this**.value)){  
 **this**.nextSibling.innerHTML="正确";  
 **this**.nextSibling.className="right";  
 }**else**{  
 **this**.nextSibling.innerHTML="错误";  
 **this**.nextSibling.className="wrong";  
 }  
 }  
}

1. **qq判断应注意**

a) 第一个不为0，总数为5-10位，第一个非0也算一个数，所以\d{4,10}



## 2.4 12-完整版表单验证

**1、限制号码的输入格式**

a) 13[0-9] , 14[5-7] , 15[0-9] , 17[1456789] , 18[0-9] 限制这几个字段前三位数只能输入规定的数

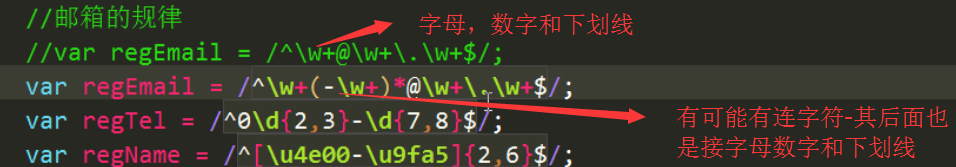
## 2.5 13-完整版表单验证

**1、连接符 . 的使用**

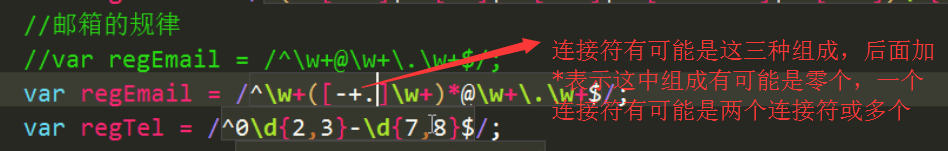
a) . 代表除回车和空格之外的所有字符使用 . 需要转义符 \. 后方可使用

**2、中划线 - + . \* 的使用**

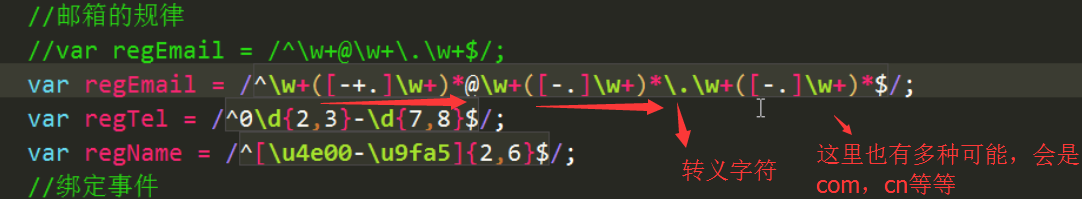
a) ([-.+]\w+)\* 表示 - . +中的一个 后面有文字数字下划线 而且出现的 次数是大于等于0



多个连接符组成用\*



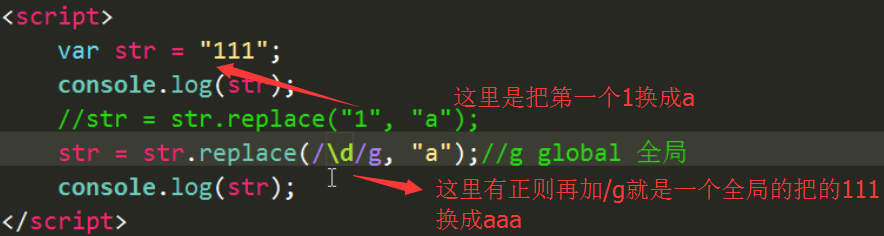
最终：



## 2.6 14-封装trim方法

**1、/\d/和/\d+/和 /\d/g的区别 str=”111”;**

* 1. /\d/ 表示第一个1替换成a 显示a11
  2. /\d+/ 表示整个都替换了成a 显示a
  3. /\d/g 表示全局的都替换成a 显示aaa



1. **trim的封装**

**function** trim(str) {  
 **return** str.replace(/^\s+|\s+$/g, "");  
}

1. \s开始和结尾都是不可见的字符，通过替换成 ”” 不占位
2. 获取当前输入的值，通过trim方法替换掉输入值中的空字符串

inp1.onblur = **function** () {  
 **if** (trim(**this**.value) === "") {  
 alert("输入不能为空");  
 } **else** {  
 alert("正确");  
 }  
};

# 3. 递归

## 3.1 15-递归

**1、递归累加原理**

a) 1+2+3=3+(1+2)

b) 1+2+3+4=4+(1+2+3)

c) 1+2+3+4+5=5+(1+2+3+4)

d) 1+2+3+……+n=n+(1+2+3……+n-1)

e) n个数的和=n+ (n-1的和)

**function** sub(n){  
 **if**(n===1){  
 **return** 1;  
 };  
 **var** s=n+sub(n-1);  
 **return** s;  
};  
console.log(sub(5));

**2、sub方法被持续调用**

a) N=5,会调用sub(4),

b) N=4,会调用sub(3)...

## 3.2 16-递归案例

**1、什么是阶乘,如何求阶乘**

a) 1！=1

b) 2!=2\*1

c) 3!=3\*2\*1

d) 4!=4\*3\*2\*1

e) n!=n\*(n-1)!

**function** getJC(n) {  
 **if** (n === 1) {  
 **return** 1;  
 }  
 **var** jc = n \* getJC(n - 1);  
 **return** jc;  
}

1. fibonacci的求和公**式**

a) getFib(3)=getFib(1)+getFib(2)

b) getFib(4)=getFib(2)+getFib(3)

c) getFib(n)=getFib(n-1)+getFib(n-2)

d) 第n项的值等于n的前两项的和

**function** getFib(n) {  
 **if** (n === 1 || n === 2) {  
 **return** 1;  
 }  
 **var** fib = getFib(n - 2) + getFib(n - 1);  
 **return** fib;  
}  
console.log(getFib(12));//144