正则

正则：一个规则，它是用来处理字符串的，验证字符串是否符合某个规则（正则匹配），或者是把字符串中符合规则的内容取出来（正则捕获）

一个正则是由元字符组成的。

创建正则有两种方式(有一些区别)

var reg1=/\d/;//字面量方式 ，代表 包含0-9之间的任意一个数字即可

var reg2=new RegExp("\\d");//实例创建方式

区别在于：

1.实例创建方式需要多转译一次,把具有特殊意思，并且带\的都要多转译一次。2. 字面量方式无法识别变量，而实例创建方式可以，也就是说实例创建可以进行我们的字符串拼接（把一个变量代表的值放到正则中作为规则）。

var reg1=/\d/;

var reg2=new RegExp("\\d");

var reg=/zhufeng/;

console.log(reg.test("welcome zhufeng student"));//true

console.log(reg.test("welcome zhu feng student"));//false

var c="w100";

var reg=/^"+c+"$/;//以"开头，出现一到多次，然后是c出现一到多次，最后以"结尾，而不是我们认为的字符串拼接

var reg=new RegExp("^"+c+"$");//此时只能包含w100的

正则中还包含修饰符：i（ignoreCase 忽略大小写） , m（multiline 匹配换行） , g(global 全局匹配)

var reg=/^[a-z]$/i;

var reg=new RegExp("^[a-z]$","i");

console.log(reg.test("Z"));

具有特殊意义的元字符

\d :0-9之间任意一个数字

\ :转译字符

^ ：以某一个元字符开始

$ ：以某一个元字符结束

\n ：一个换行符

. :匹配除了\n以为的任何字符

x|y :x或者y

[xyz] :x y z 三个中一个

[^xyz] :除三个中的任一个

[a-z] :a--z之间任意一个

[^a-z] :除了a--z之间任意一个

\w :数字、字母、下划线 [a-zA-z\_0-9]

\b :匹配一个边界

\s :匹配任何的空白字符 空格 制表符 换页符

量词元字符（代表出现多少次）

\* ：出现零到多次

+ ：出现一到多次

？：出现零或一次

{n} :出现n次

{n,} :出现n到多次

{n,m} :出现n到m次

普通元字符 ： 所有的东西都可以作为我们的元字符，除了特殊的以外，都代表本身的意思

正则中有一个方法，

test 匹配字符串（验证某个字符串是否符合正则的规则的），返回true代表匹配 返回false代表不匹配。

例1：

var reg=/\d/;

var str1="zhufeng";

var str2="zhufeng2015";

var res1=reg.test(str1);//判断这个字符串是否符合我们的这个规则[包含0-9之间的任意一个数据即可]

console.log(res1);//false 不符合返回false 符合返回true

var res2=reg.test(str2);

console.log(res2);//true

例2

var reg=/^\d$/;//以0-9之间的数子开始和结束，并且中间只能包含一个数字

var str1="8";

var res1=reg.test(str1);

console.log(res1);//true

例3

var reg=/^\d+$/;//以0-9之间的数子开始和结束，并且中间只能包含一到多个数字

var str1="8253";

var res1=reg.test(str1);

console.log(res1);//true

例4

var reg=/^\d{11}$/;//简单的手机号码验证正则，只能是数字并且是11位

var reg=/^12.5$/;//注意：这里面的.代表是是除了换行外的任意字符，而不是小数点。

如果我想让他只代表小数点，var reg=/^12\.5$/;需要用转译字符来处理

console.log(reg.test("12.5"));

此时需要注意的是：

var reg=/^[12.5]$/;//[]包起来的里面都是代表本身意思的字符，没有特殊意义，.就是小数.

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

例5

var reg=/^wo(r|y)d$/;

//():正则的分组：大正则中的小括号是小正则，改变了默认的优先级

console.log(reg.test("word"));//true

console.log(reg.test("woyd"));//true

console.log(reg.test("wor"));//false 没有以d结束

console.log(reg.test("yd"));//false

console.log(reg.test("woryd"));//false

例6

var reg=/^[word]$/;//这个代表的是只能是以word中的一个开头结束

console.log(reg.test("w"));//true

console.log(reg.test("o"));//true

console.log(reg.test("wo"));//false

想反：考虑下这个问题，不会得再找我啊^\_^。

var reg=/^[^word]$/;

console.log(reg.test("w"));//

console.log(reg.test("o"));//

console.log(reg.test("wo"));//

exec : 正则捕获的方法（正则的捕获是懒惰的。）

1.首先拿我们的字符串和正则进行匹配，如果不匹配，捕获的结果是null

2.只有匹配，才按照规则进行捕获

通过exec如果可以获取到内容的话是一个数组（跟平时的长得不是很一样）

第一项是捕获到的内容，index是捕获到的索引，input我们捕获的那个原始的字符串

正则的捕获是懒惰的，默认只捕获第一个符合条件的，例如：只得到了2015，但是没有获取2014

正则的捕获是贪婪的，默认按照匹配最长的捕获。例如：捕获的是2015而不是2

如何取消正则的贪婪性，在量词元字符后面加？即可，出现在普通元字符后面代表出现0-1次，如果出现在量词元字符后面代表取消正则的贪婪性。

例子：

//1)可以加一个全局的修饰符g，需要捕获几次，我们就执行几次，这样就能都捕获到了

var reg=/\d+?/g;//var reg=/\d+/g;

var ary=[];

var str="zhufeng2015peixun2014";

var res=reg.exec(str);

while(res){

ary.push(res[0]);

res=reg.exec(str);

}

console.log(ary);

例子：

var reg=/\d+/g;

console.log(reg.lastIndex);

console.log(reg.exec("zhufeng2015peixun2014"));//["2015", index: 7, input: "zhufeng2015peixun2014"]

console.log(reg.lastIndex);

console.log(reg.exec("zhufeng2015peixun2014"));//["2014", index: 17, input: "zhufeng2015peixun2014"]

console.log(reg.lastIndex);

console.log(reg.exec("zhufeng2015peixun2014"));//null

console.log(reg.lastIndex);

分组：用()小括号包起来的就是一个分组，也是大正则中的一个小正则。

1.改变默认的优先级

2.可以进行分组捕获

例子：

捕获到的是一个数组，第一项是大正则捕获到的内容，接下来分别是每一个小分组捕获到的内容

index 是大正则捕获的开始索引

var reg=/(\d+)([a-z]+)/;

console.log(reg.exec("zhufeng2015peixun2014"));

["2015peixun", "2015", "peixun", index: 7, input: "zhufeng2015peixun2014"]

"2015peixun" 大正则捕获的内容 ary[0]

"2015" 第一个分组捕获的内容 ary[1]

"peixun" 第二个分组捕获的内容 ary[2]

....

注：如果不想捕获第二个分组中的内容，只需要再小括号中加?:即可

例子：

var reg=/(\d+)(?:[a-z]+)/;

console.log(reg.exec("zhufeng2015peixun2014"));

["2015peixun", "2015", index: 7, input: "zhufeng2015peixun2014"]

字符串中也有一个捕获的方法：match

不加全局匹配符g的时候，和我们的exec一样，大正则和分组的内容都可以捕获到

加g的时候，虽然执行一次就可以把所有的大正则匹配的内容都捕获到了（而exec需要多次操作才可以），

但是match加上g的时候不能捕获到小分组中的内容。

例子：

var reg=/(\d+)([a-z]+)/;

var str="zhufeng2015peixun2014";

console.log(str.match(reg));

["2015peixun", "2015", "peixun", index: 7, input: "zhufeng2015peixun2014"]

和下面例子进行比较，区分

var reg1=/(\d+)([a-z]+)/g;

var str1="zhufeng2015peixun2014";

console.log(str1.match(reg1));//[2015peixun]

分组的引用：

当我们发现正则中的某一部分需要和另外一部分匹配的内容一模一样才可以，这样的情况下我们用分组引用来解决。

如下例子：

var reg=/(\d+)zhufeng\1/;// \1就是对第一个分组的引用

\1代表的是和第一分组\d+一模一样的内容

console.log(reg.test("2015zhufeng2015"));//true

console.log(reg.test("2015zhufeng2014"));//false

下面是一些常用的正则整理：

1.匹配一个有效数字：正数、负数、0、小数

var reg=/^[+-]?(\d|[1-9]\d+)(\.\d+)?$/;

2.是正整数的

var reg=/^[1-9]\d\*$/;

3.年龄18-65： 18 19 2-5 0-9 60 61 62 63 64 65

var reg=/^(?:1[8-9])|(?:[2-9]\d)|(?:6[0-5])$/;//只匹配不捕获

4.中国标准真实姓名：最多四个 最少两个汉字

var reg=/^[\u4e00-\u9fa5]{2,4}/;

5.昵称：数字 字母 下划线 中线 汉字

var reg=/^(\w)-|[\u4e00-\u9fa5]+$}/;

6.邮箱 .cpm .cn xxx@xxx.xxx.xxx xxx@xxx.xxx

var reg=/^(?:[0-9\_a-z.A-Z-]+)@(?:[0-9a-zA-Z]+)(?:\.[a-zA-Z]{2,4}){1,2}$/;

7.身份证号

var reg=/^(\d{3})(\d{3})(\d{4})(\d{2})(\d{2})(?:\d{2})(\d)(?:\d|X)$/;

console.log(reg.exec("35032119910101002X"));

//["35032119910101002X ", "350", "321", "1991", "01", "01", "2", index: 0, input: "35032119910101002X "]

字符串中的replace方法，我们第一个参数如果写的是正则，没匹配一次replace就执行一次（不要忘记加g）

字符串中的replace方法，可以捕获正则匹配的内容，如果我们第二个参数传递的是一个function，

每捕获一次我们的函数就执行一次，并且不光光是执行，执行的时候浏览器还默认传递了参数

例子：判断字符出现的次数

var str="zhufengpeixunyangfengqihang";

//把每一个字母出现的次数进行统计

var obj={};

str.replace(/[a-zA-Z]/gi,function(){

var a=arguments[0];//当前捕获到的内容

if(obj[a]>=1){

obj[a]=obj[a]+1;

}else{

obj[a]=1;

}

});

console.log(obj);//Object {z: 1, h: 2, u: 2, f: 2, e: 3…}

例子：将小写的数字转大写

var str="8872840";

var ary=["零","壹","贰","叁","肆","伍","陆","柒","坝","九","十",];

str=str.replace(/\d/g,function(){

//var cur=arguments[0];

return ary[arguments[0]];

});

console.log(str);

以上是一些资料和个人理解的正则整理，欢迎大家更新,指正。

2015/08/15 by 林雅贞