**正则表达式**

首先：正则表达式可以看做是一个独立的部分。在我们的前端和后端其实都可以用到。作用也是一样的。

一、通过一个例子演示一下正则表达式与我们传统的js有哪些不同的地方。

例：找出字符串中的所有数字

1、用传统字符串操作完成

2、用正则表达式完成

首先复习一下字符串的方法：

indexOf、substring、charAt、split、a<=9 && a>=0

先写传统字符串的方式，再用正则的方式，可以看出正则的方式要比传统字符串的方式性能要好。

var str = 'haj123sdk54hask33dkhalsd879';

console.log(findNum(str));

function findNum(str){

return str.match(/\d+/g);

}

**二、什么是正则？**

写几个东西，猜猜是什么

13756789988

now777@qq.com

我们人是认识，但是电脑认不认识这几个东西呢？

**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

给定一个正则表达式和另一个字符串，我们可以达到如下的目的：1. 给定的字符串是否符合正则表达式的过滤逻辑（称作“匹配”）；2. 可以通过正则表达式，从字符串中获取我们想要的特定部分。

正则表达式的特点是：1. 灵活性、逻辑性和功能性非常的强；2. 可以迅速地用极简单的方式达到字符串的复杂控制。

**三、正则的写法：**

复习数组和对象的写法，比如：

var arr = [];

var arr = new Array();

var obj = {};

var obj = new Object();

正则的写法，正则其实也是一个系统对象。

var re = //; //简写

var re = new RegExp(); // 全称，Reg是正则的意思，Exp是表达式的意思。

注意：平时推荐大家使用简写的形式。因为简写的方式比全称的写法性能要好。但是有一种形式（正则需要传参的时候），必须用全称的写法。

但是我们发现一个细节，var re = //;这里只写两个//，浏览器会把这里认为是注释。所以我们尽量不要让这里为空。可以给它添加点东西。

var re = /a/;

var re = new RegExp('a');

正则简写时，/a/千万不要加引号。

但在全称写法时，一定要加上引号。代表a是字符串，如果不写引号，a就代表是变量了。

**四、正则表达式的常用方法**

正则下面的方法：exec、test

字符串下面的方法：search、match、replace、split

**五、test方法**

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

例一：

var str = 'abcdef';

var re = /b/;

alert( re.test(str) ); // 真

解释：为什么弹出的是真？

首先，正则去匹配字符串，正则就是b，去在str里面找b，哪么找到了没有，肯定是找到了，所以返回true。

如果改成var re = /w/;呢? 这就找不到，这就返回假。

如果改成var re = /bc/;呢?

如果改成var re = /bd/;呢?

例二：判断一个字符串里是否全是数字

var str = '3748t29348791';

想法：

var re = /t/;

这样合不合适呢，如果里面不是t了呢，而是r了呢。会走什么呢？

这个时候var re = /t/;这里应该匹配的是非数字，哪怎么做到呢？

首先我们要接触一个叫转义字符的这么一个东西。

参看下面的转义字符。

。。。。。。

看了转义字符，我们发现这里就应该写一个var re = /\D/;就可以了。（非数字）

var re = /\D/;

if( re.test(str) ){

alert('不全是数字');

}else{

alert('全是数字');

}

**六、search**

概念：正则去匹配字符串 , 如果匹配成功，就返回匹配成功的位置，如果匹配失败就返回 -1。（很像字符串中的indexOf()一样）

写法：字符串.search(正则)。注意它是字符串下面的方法

注意：它会忽略g标识

例一：

var str = 'abcdef';

var re = /b/;

alert(str.search(re)); // 1，因为找到了b，它就返回b在这个字符串中的位置。

如果写了一个var re = /w/;就返回-1。

如果写了一个var re = /bcd/;就返回1。

如果写了一个var re = /B/;就返回-1。说明这个B在这里面没有找到，说明:正则默认是区分大小写的。如果写了一个大写B，就是没有找到的。

怎么样不区分大小写呢？

如果不区分大小写的话，在正则的最后加标识 i

var re = /B/i;这个时候就是不管你字符串中是大写的B还是小写的b，它都能匹配成功。

如果后面多了i，用全称怎么写呢？即加一个参数。

var re = new RegExp('B','i');

**七、match**

概念：正则去匹配字符串，如果匹配成功，就返回匹配成功的数组，如果匹配不成功，就返回null

写法：字符串.match(正则)

例一：打开看最开始的哪个例子

var str = 'haj123sdk54hask33dkhalsd879';

var re = /\d/;

console.log( str.match(re) ); // Array [ "1" ]

这是为什么呢？\d为什么只找到第一个，而后面的没有找呢？

这涉及到正则的另一个默认行为：正则匹配成功就会结束，不会继续匹配了。

如果想全部查找，就要加标识 g(全局匹配)

再次看这个例子：

var str = 'haj123sdk54hask33dkhalsd879';

var re = /\d/g;

console.log( str.match(re) ); // Array [ "1", "2", "3", "5", "4", "3", "3", "8", "7", "9" ]

这又是为什么呢？123没有连成一个整体？

因为我们这里写的是一个\d，它每次查找只找一位，哪如果我们想找两位的话，就要写两个\d\d，

再看这个例子：

var str = 'haj123sdk54hask33dkhalsd879';

var re = /\d\d/g;

console.log( str.match(re) ); // Array [ "12", "54", "33", "87" ]

思考：这样是不是很不智能，再写一个var re = /\d\d\d/g;就只能返回Array [ "123", "879" ]，如果写成var re = /\d\d\d\d/g;这里面没有4个的数字，哪就匹配不成功，返回null

这样我们了解到位数很重要，但是我们这里的位数又不确定，哪怎么解决呢？

这里就又涉及到一个新的概念：量词。这里说说量词+

它代表至少出现一次。可以看做是大于等于1

var str = 'haj123sdk54hask33dkhalsd879';

var re = /\d+/g;

console.log( str.match(re) ); // Array [ "123", "54", "33", "879" ]

var re = /\d+/g;这里的意思就是\d至少出现一次，或者多次。

**八、replace**

概念：正则去匹配字符串，匹配成功的字符去替换成新的字符串，返回替换过以后的字符串。

写法：字符串.replace(正则, 新的字符串)

注意：第二个参数：可以是字符串，也可以是一个回调函数（函数的第一个参数：就是匹配成功的字符）

例一：

var str = 'aaa';

var re = /a/;

str = str.replace(re,'b');

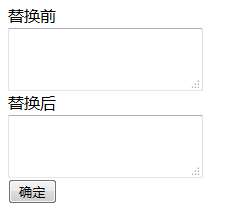
alert(str); // baa

思考：为什么是baa？为什么没有都替换呢？

如果这时改成var re = /a/g;就都替换成bbb了。

如果这时改成var re = /a+/g;就都替换成b了。

例二：敏感词过滤.html



比如：

人民日报：美国减税就是在挑起“税务战”

要替换成

\*\*日报：美国\*\*就是在挑起“\*\*\*”

即人民、减税、税务战要替换成\*，一个汉字变成一个\*。

思考：

一、首先，做这个效果我们要想一下用哪个方法。（replace）

难点：这个正则怎么写

var re = /人民减税税务战/; 我们这样写，能匹配成功吗？

（这里我们就又要涉及到一个新的东西：|）

|：或的意思。

二、进一步，用上竖杠

var re = /人民|减税|税务战/;

oInput.onclick = function() {

aT[1].value = aT[0].value.replace(re, '\*');

};

这时，我们运行，发现第一个确实变成了星号，但是后面的没有替换

原因，我们没有加g，加全局匹配。加上以后，对应的词就替换成了\*。但是这样，跟我们想要的效果还是有差别，我们想要一个字对应一个星。

三、怎么样做到一个汉字对应一个星呢？

这就要用到replace的第二个参数。让它变成一个回调函数。

如：

var re = /人民|减税|税务战/g;

oInput.onclick = function() {

// aT[1].value = aT[0].value.replace(re, '\*');

aT[1].value = aT[0].value.replace(re, function (){

return '\*'

});

};

函数返回一个'\*'，同上面的写法完全一样。

至于有几个字符，就要用到函数的参数。（参数其实是匹配成功的字符串）

如：

oInput.onclick = function() {

// aT[1].value = aT[0].value.replace(re, '\*');

aT[1].value = aT[0].value.replace(re, function (str){

console.log(str); //人民 减税 税务战

return '\*'

});

};

思考：即然我们已经能得到这个匹配成功的字符，我们是不是就能根据它的长度加上几个星了。

如：

oInput.onclick = function() {

aT[1].value = aT[0].value.replace(re, function (str){

var result = '';

for(var i=0; i<str.length; i++){

result += '\*';

}

return result

});

};

**九、split**

概念：把字符串拆分成数组

写法：字符串.split(正则，长度)

注意：第一个参数可以是一个字符串，也可以是一个正则。第二个参数是可选，即返回数组的长度。

如：

var str = '2017-6-14 15:23:40';

var arr = str.split(/-|:|\s/g);

**十、()，小括号有两个意思：**

一、分组操作

二、匹配子项

三、优选级

**一、分组操作：**

例：匹配日期，2017-4-29。我们要匹配一个数字加一个-，即匹配到2017-和4-，后面的29不匹配上。

var str = '2017-4-29';

var re = /\d+-/g;

str.replace(re, function ($0){

console.log($0); // 2017- 4-

})

说明：$0就是匹配成功的字符

这样调整一下正则：

var re = /\d-+/g;

就返回：7- 4-

再调整一下正则：

var re = /(\d-)+/g;

就返回：7-4-

结论：

没有分组，量词+仅仅针对它前面的哪个-，

有了分组，量词+就是针对它前面的这个整体(\d-)。

**二、匹配子项**

定义：

把正则的整体叫做（母亲），然后把左边第一个小括号里面的正则，叫做这个正则的第一个子项(即母亲的第一个孩子)，第二个小括号就是这个母亲的第二个孩子。

再看这个例子：

var str = '2017-4-29';

var re = /\d+-/g;

str.replace(re, function ($0){

console.log($0); // 2017- 4-

})

这里的$0即代表这个母亲整体。

再改一下

var str = '2017-4-29';

var re = /(\d+)(-)/g;

str.replace(re, function ($0){

console.log($0); // 2017- 4-

})

(\d+)(-) 这个整体我们把它叫做母亲。

(\d+)即这个母亲的第一个孩子。

(-)即这个母亲的第二个孩子。

讲这么多它有什么用呢？

首先，我们的正则可能会写得非常之复杂，如果直接去操作一个母亲的话，操作就不太容易，如果能通过操作子项，来操作正则的一部分，就比较容易点了。

很多方法都可以来匹配子项的，像这个replace就可以。

str.replace(re, function ($0, $1, $2){

console.log($0);

console.log($1);

console.log($2);

})

第一个参数：$0（整体母亲）

第二个参数 : $1(第一个孩子)

第三个参数 : $2(第二个孩子)

参数名我们是可以随便写的，但是这个顺序就是这样的。

案例使用：

我想把这个时间“2017-4-29”变成2017.4.29，这个正则不变，我们如何写出。

方法一：

return $0.substring(0, $0.length-1) + '.';

方法二：

return $1 + '.';

**match**中的匹配子项

例如：

var str = 'abc';

var re = /abc/;

console.log( str.match(re) ); // Array [ "abc" ]

改一下：

var str = 'abc';

var re = /(a)(b)(c)/;

console.log( str.match(re) ); // Array [ "abc", "a", "b", "c" ]

注意这是不加g的情况下，才可以获取到子项的集合。

如果加g，则只返回整体

var str = 'abc';

var re = /(a)(b)(c)/g;

console.log( str.match(re) ); // Array [ "abc" ]

**十一、[]** 即字符类。

概念：一组相似的元素，用[]来表示。[] 中括号的整体代表一个字符

例如：

var str = 'abc';

var re = /a[bde]c/;

console.log(re.test(str)); // true，匹配成功

[bde]这三个是或的关系，可以出现b，也可以出现d，也可以出现e

再改改：

var str = 'abdc';

var re = /a[bde]c/;

console.log(re.test(str)); // false

**中括号里的 ^ 排除**

如果^写在[]里面的话，就代表排除的意思

如：

var str = 'abc';

var re = /a[^bde]c/;

console.log(re.test(str)); // false

再改改：

var str = 'awc';

var re = /a[^bde]c/;

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

**中括号里的 范围**

有时我们需要写一个a到z的，如：

[abcdefghi....z]这样一直写到z是可以的，但是比较麻烦。

简写：

[a-z]这就代表a到z这26个英文字母。

注意：

1、一定要是从小到大，而不能写成从z-a。

2、也不一定必须是a-z，也可以是e-j

[a-z0-9A-Z]这样即小写a-z，0-9，大写A-Z，但是不管你写多少，这个中括号里面的都只代表一个字符。

字符类案例：

过滤html标签：

<h3>标题：</h3>文字内容

我想得到：标题：文字内容

思考：<h3>标题：</h3>这个是以<开始，以>结束，里面的字符个数不确定，可能是数字，字母，也有/，

var re = /<\w+>/g;这样可不可以？

前面的替换了，但是</h3>没有被替换。如果再替换一个/呢，哪再有class有引号呢？这么多特殊字符怎么办？

其实，我们这样想，不管它中间是什么，只要不是>号，就OK。

var re = /<[^>]+>/g;

<[^>]+>说明：<开头，[^>]+只要不是>都可以，且排除多个，最后以>结束。

**十二、量词**

概念：不确定字符的个数

写法：{}

如：

{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。

案例：判断输入的是否是QQ号

QQ号规则：5-12位，不以0开头。

var re = /[1-9]\d{4,11}/;这样可不可以呢？

用123456aa来测试看看。

**就要另外看两个东西：**

^ : 正则的最开始位置，就代表起始的意思

$ : 正则的最后位置 , 就代表结束的意思

var re = /^[1-9]\d{4,11}$/;这就是说我们必须以1-9的数字开头，以4-11位的数字结束。

**再来一个例子：**

去掉字符串前后空格：

var re = /^\s+|\s+$/g;

^\s+代表起始的空格，可以有多个

\s+$代表结束的空格，可以有多个

中间的竖线，代表或者

g代表全局匹配。

**转义字符：**

什么叫转义字符：本来这个字符它有自身的意思，但是加上反斜扛就成了另外一个意思。

比如：n，它就是n

alert('anb');

alert('a\nb');加上反斜扛，哪么\n就是换行的意思了。

\s : 空格

\S : 非空格

\d : 数字

\D : 非数字

\w : 字符 ( 字母 ，数字，下划线\_ )

\W : 非字符

. : 除了换行符以外的任意字符

\. : 真正的点

\b : 独立的部分 （ 起始，结束，空格 ）

\B : 非独立的部分

\1 : 重复的第一个子项

\2 : 重复的第二个子项

转义字符：将特殊符号恢复成它字面本身意思

通过：\ 这个符号来进行转义

需要转义的符号 ： [ ] { } ( ) . ? \* + ^ $ \ / |

**一、**例：真正的点和任意字符的例子：

var str = 'a.c';

var re = /a\.c/;

alert(re.test(str)); // true

**二、**例：独立的部分和非独立的部分的例子：

var str = 'onetwo';

var re = /one/;

alert( re.test(str) ); // true

例：再改改

var str = 'onetwo';

var re = /\bone/;

alert( re.test(str) ); // true

例：再改改

var str = 'onetwo';

var re = /one\b/;

alert( re.test(str) ); // false

**三、**例：独立的部分和非独立的部分的例子：通过class获取元素

试试这个正则：

var re = new RegExp(sClass);

这时我们就可以试试独立的部分。

var re = new RegExp('\\b'+sClass+'\\b');

哪为什么[\\b](file:///\\\\b)这里要写两个反斜扛呢？

看看alert('\');我们想输出一个反斜扛，这样能输出吗？反倒会报错。

alert('\\');这才能真正输出一个反斜扛

alert('\'');这样输出一个单引号

同样的道理， 我们要输出\b，前面也要加上一个反斜扛。

**四、**例：复重的子项：

var str = 'abca';

var re = /(a)(b)(c)\1/;

alert( re.test(str) ); // true

这里的\1代表着第一个子项，即（a）。

var str = 'abca';

var re = /(a)(b)(c)\2/;

alert( re.test(str) ); // false

这里的\2代表着第二个子项，即（b）。所以不匹配。

**五、**例：

var re = /\w\w/;

var re = /(\w)\1/;

这两正则是一样的吗？

\w\w可以匹配上ab,cd,w9

(\w)\1只可以匹配cc,99

所以，重复的子项针对前面哪项是一个不确定的字符的时候使用。

**六、**重复的子项的使用场景：找重复项最多的字符和个数

var str = 'assssjdssskssalsssdkjsssdss';

思考：首先要找重复的字符，必须要把它们排在一起。

所以首先把它转成数组再排序，排序之后再转成字符串。

var arr = str.split('');

str = arr.sort().join('');

正则：var re = /(\w)\1+/g;

这里我们用replace方法，但是我们这里不是要替换，只是用它来修改我们的value和index。

**常用正则：**

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

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

**工作中的建议**：所有的正则放在一个对象中，用时直接调用。方便管理

var re = {

qq : /^[1-9][0-9]{4,9}$/,

email : /^\w+@[a-z0-9]+(\.[a-z]+){1,3}$/,

number : /\d+/

};

比如：要查找qq。直接调用re.qq即可以了。

综合案例：

对用户名、手机号码、email、url地址等的检测

字符串转驼峰形式、去掉字符串前后空格