# 正则表达式专题

2012年10月4日 星期四 12:00

## (一). 基础语法

**创建一个正则表达式**

第一种方法：

```
var reg = /pattern/;
```

第二种方法：

```
var reg = new RegExp('pattern');
```

**正则表达式的exec方法简介**

语法：

```
reg.exec(str);
```

其中str为要执行正则表达式的目标字符串。

例如：

```
<script type="text/javascript">
  var reg = /test/;
  var str = 'testString';
  var result = reg.exec(str);
  alert(result);
</script>
```

将会输出test，因为正则表达式reg会匹配str（‘testString’）中的’test’子字符串，并且将其返回。

我们使用下面的函数来做匹配正则的练习：

```
function execReg(reg,str){
  var result = reg.exec(str);
  alert(result);
}
```

函数接受一个正则表达式参数reg和一个目标字符串参数str，执行之后会alert出正则表达式与字符串的匹配结果。

用这个函数测试上面的例子就是：

```
<script type="text/javascript">
  function execReg(reg,str){
    var result = reg.exec(str);
    alert(result);
  }
  var reg = /test/;
  var str = 'testString';
  execReg(reg,str);
</script>
```

上面的例子用正则里的test去匹配字符串里的test，实在是很无聊，同样的任务用indexOf方法就可以完成了。用正则，自然是要完成更强大的功能：

**一片两片三四片，落尽正则全不见**

上面的小标题翻译成正则就是{1}，{2}，{3,4}，{1,}。

**c{n}**

{1}表示一个的意思。

/c{1}/只能匹配一个c。

/c{2}/则会匹配两个连续的c。

以此类推，

/c{n}/则会匹配n个连续的c。

看下面的例子：

```
reg = /c{1}/;
str='cainiao';
execReg(reg,str);
```

返回结果c

```
reg = /c{2}/;
str='cainiao';
```

```
execReg(reg,str);
```
返回结果null，表示没有匹配成功。
```
reg = /c{2}/;
str='ccVC果冻爽';
execReg(reg,str);
```
返回结果cc。

**c{m,n}**

c{3,4}的意思是，连续的3个c或者4个c。

例如
```
reg = /c{3,4}/;
str='ccVC果冻爽';
execReg(reg,str);
```
返回结果null，表示没有匹配成功。
```
reg = /c{3,4}/;
str='cccTest';
execReg(reg,str);
```
结果返回ccc。
```
reg = /c{3,4}/;
str='ccccTest';
execReg(reg,str);
```
结果返回cccc，这表明正则会尽量多品牌，可3可4的时候它会选择多匹配一个。
```
reg = /c{3,4}/;
str='cccccTest';
execReg(reg,str);
```
仍然只匹配4个c。

由以上例子可以推断出，c{m,n}表示m个到n个c，且m小于等于n。

**c{n,}**

c{1,}表示1个以上的c。例如：
```
reg = /c{1,}/;
str='cainiao';
execReg(reg,str);
```
结果返回c。
```
reg = /c{1,}/;
str='cccccTest';
execReg(reg,str);
```
返回ccccc，再次说明了正则表达式会尽量多地匹配。
```
reg = /c{2,}/;
str='cainiao';
execReg(reg,str);
```
结果返回null，c{2,}表示2个以上的c，而cainiao中只有1个c。

由以上例子可知，c{n,}表示最少n个c，最多则不限个数。

***, +, ?**

*表示0次或者多次，等同于{0,}，即

c* 和 c{0,} 是一个意思。

+表示一次或者多次，等同于{1,}，即

c+ 和 c{1,} 是一个意思。

最后，?表示0次或者1次，等同于{0,1}，即

c? 和 c{0,1} 是一个意思。

**贪心(贪婪)与非贪心(非贪婪)**

人都是贪婪的，正则也是如此。我们在例子reg = /c{3,4}/;str='ccccTest';的例子中已经看到了，能匹配四个的时候，正则绝对不会去匹配三个。

上面所介绍的所有的正则都是这样，只要在合法的情况下，它们会尽量多去匹配字符，这就叫做贪心(贪婪)模式。

如果我们希望正则尽量少地匹配字符，那么就可以在表示数字的符号后面加上一个?。组成如下的形式：

{n,}?, *?, +?, ??, {m,n}?

同样来看一个例子：
```
reg = /c{1,}?/;
str='ccccc';
execReg(reg,str);
```
返回的结果只有1个c，尽管有5个c可以匹配，但是由于正则表达式是非贪心模式，所以只会匹配一个。

**/^开头,结尾$/**

^表示只匹配字符串的开头。看下面的例子：

```
reg = /^c/;
str='维生素c';
execReg(reg,str);
```

结果为null，因为字符串‘维生素c’的开头并不是c，所以匹配失败。

```
reg = /^c/;
str='cainiao';
execReg(reg,str);
```

这次则返回c，匹配成功，因为cainiao恰恰是以c开头的。

与^相反，$则只匹配字符串结尾的字符，同样，看例子：

```
reg = /c$/;
str='cainiao';
execReg(reg,str);
```

返回null，表示正则表达式没能在字符串的结尾找到c这个字符。

```
reg = /c$/;
str='维生素c';
execReg(reg,str);
```

这次返回的结果是c，表明匹配成功。

**点“.”**

‘.’会匹配字符串中除了换行符\n之外的所有字符，例如

```
reg = /./;
str='cainiao';
execReg(reg,str);
```

结果显示，正则匹配到了字符c。

```
reg = /./;
str='blueidea';
execReg(reg,str);
```

这次是b。

```
reg = /.+/;
str='blueidea——经典论坛 好_。';
execReg(reg,str);
```

结果是“blueidea——经典论坛 好_。“也就是说所有的字符都被匹配掉了，包括一个空格，一个下滑线，和一个破折号。

```
reg = /.+/;
str='bbs.blueidea.com';
execReg(reg,str);
```

同样，直接返回整个字符串——bbs.blueidea.com，可见”.”也匹配”.”本身。

```
reg = /^./;
str='\ncainiao';
execReg(reg,str);
```

结果是null，终于失败了，正则要求字符串的第一个字符不是换行，但是恰恰字符是以\n开始的。

**二选一，正则表达式中的或，“|“**

b|c表示，匹配b或者c。

例如：

```
reg = /b|c/;
str='blueidea';
execReg(reg,str);
```

结果是b。

```
reg = /b|c/;
str='cainiao';
execReg(reg,str);
```

结果是c。

```
reg = /^b|c.+/;
str='cainiao';
```

```
execReg(reg,str);
```
匹配掉整个cainiao。
```
reg = /^b|c.+/;
str='bbs.blueidea.com';
execReg(reg,str);
```
结果只有一个b，而不是整个字符串。因为上面正则表达式的意思是，匹配开头的b或者是c.+。

**括号**
```
reg = /^(b|c).+/;
str='bbs.blueidea.com';
execReg(reg,str);
```
这次的结果是整个串bbs.blueidea.com，机上上面的括号这后，这个正则的意思是，如果字符串的开头是b或者c，那么匹配开头的b或者c以及其后的所有的非换行字符。
如果你也实验了的话，会发现返回的结果后面多出来一个“,b“，这是()内的b|c所匹配的内容。我们在正则表达式内括号里写的内容会被认为是子正则表达式，所匹配的结果也会被记录下来供后面使用。我们暂且不去理会这个特性。

**字符集合[abc]**

[abc]表示a或者b或者c中的任意一个字符。例如：
```
reg = /^[abc]/;
str='bbs.blueidea.com';
execReg(reg,str);
```
返回结果是b。
```
reg = /^[abc]/;
str='test';
execReg(reg,str);
```
这次的结果就是null了。
我们在字字符集合中使用如下的表示方式:[a-z],[A-Z],[0-9]，分别表示小写字母，大写字母，数字。例如：
```
reg =  /^[a-zA-Z][a-zA-Z0-9_]+/;
str='test';
execReg(reg,str);
```
结果是整个test，正则的意思是开头必须是英文字母，后面可以是英文字母或者数字以及下划线。

**反字符集合[^abc]**

^在正则表达式开始部分的时候表示开头的意思，例如/^c/表示开头是c；但是在字符集和中，它表示的是类似“非“的意思，例如[^abc]就表示不能是a，b或者c中的任何一个。例如：
```
reg = /[^abc]/;
str='blueidea';
execReg(reg,str);
```
返回的结果是l，因为它是第一个非abc的字符（即第一个b没有匹配）。同样：
```
reg = /[^abc]/;
str='cainiao';
execReg(reg,str);
```
则返回i，前两个字符都是[abc]集合中的。
由此我们可知：[^0-9]表示非数字，[^a-z]表示非小写字母，一次类推。

**边界与非边界**

\b表示的边界的意思，也就是说，只有字符串的开头和结尾才算数。例如/\bc/就表示字符串开始的c或者是结尾的c。看下面的例子：
```
reg = /\bc/;
str='cainiao';
execReg(reg,str);
```
返回结果c。匹配到了左边界的c字符。
```
reg = /\bc/;
str='维生素c';
execReg(reg,str);
```
仍然返回c，不过这次返回的是右侧边界的c。
```
reg = /\bc/;
str='bcb';
execReg(reg,str);
```
这次匹配失败，因为bcb字符串中的c被夹在中间，既不在左边界也不再右边界。

与\b对应\B表示非边界。例如：

```
reg = /\Bc/;
str='bcb';
execReg(reg,str);
```

这次会成功地匹配到bcb中的c，。然而

```
reg = /\Bc/;
str='cainiao';
execReg(reg,str);
```

则会返回null。因为\B告诉正则，只匹配非边界的c。

**数字与非数字**

\d表示数字的意思，相反，\D表示非数字。例如：

```
reg = /\d/;
str='cainiao8';
execReg(reg,str);
```

返回的匹配结果为8，因为它是第一个数字字符。

```
reg = /\D/;
str='cainiao8';
execReg(reg,str);
```

返回c，第一个非数字字符。

**空白**

\f匹配换页符，\n匹配换行符，\r匹配回车，\t匹配制表符，\v匹配垂直制表符。
\s匹配单个空格，等同于[\f\n\r\t\v]。例如：

```
reg = /\s.+/;
str='This is a test  String.';
execReg(reg,str);
```

返回“is a test String.”，正则的意思是匹配第一个空格以及其后的所有非换行字符。
同样，\S表示非空格字符。

```
reg = /\S+/;
str='This is a test  String.';
execReg(reg,str);
```

匹配结果为This，当遇到第一个空格之后，正则就停止匹配了。

**单词字符**

\w表示单词字符，等同于字符集合[a-zA-Z0-9_]。例如：

```
reg = /\w+/;
str='blueidea';
execReg(reg,str);
```

返回完整的blueidea字符串，因为所有字符都是单词字符。

```
reg = /\w+/;
str='.className';
execReg(reg,str);
```

结果显示匹配了字符串中的className，只有第一个“.”——唯一的非单词字符没有匹配。

```
reg = /\w+/;
str='中文如何？';
execReg(reg,str);
```

试图用单词字符去匹配中文自然行不通了，返回null。
\W表示非单词字符，等效于[^a-zA-Z0-9_]

```
reg = /\W+/;
str='中文如何？';
execReg(reg,str);
```

返回完整的字符串，因为，无论是中文和“？”都算作是非单词字符。

**反向引用**

形式如下：/(子正则表达式)\1/
依旧用例子来说明：
1.

```
reg = /\w/;
```

```
str='blueidea';
execReg(reg,str);
```
返回b。
2.
```
reg = /(\w)(\w)/;
str='blueidea';
execReg(reg,str);
```
返回bl,b,l
bl是整个正则匹配的内容，b是第一个括号里的子正则表达式匹配的内容，l是第二个括号匹配的内容。
3.
```
reg = /(\w)\1/;
str='blueidea';
execReg(reg,str);
```
则会返回null。这里的“\1”就叫做反向引用，它表示的是第一个括号内的字正则表达式匹配的内容。在上面的例子中，第一个括号里的(\w)匹配了b，因此“\1”就同样表示b了，在余下的字符串里自然找不到b了。
与第二个例子对比就可以发现，“\1”是等同于“第1个括号匹配的内容”，而不是“第一个括号的内容”。
```
reg = /(\w)\1/;
str='bbs.blueidea.com';
execReg(reg,str);
```
这个正则则会匹配到bb。
同样，前面有几个子正则表达式我们就可以使用几个反向引用。例如：
```
reg = /(\w)(\w)\2\1/;
str='woow';
execReg(reg,str);
```
会匹配成功，因为第一个括号匹配到w，第二个括号匹配到o，而\2\1则表示ow，恰好匹配了字符串的最后两个字符。

**括号（2）**

前面我们曾经讨论过一次括号的问题，见下面这个例子：
```
reg = /^(b|c).+/;
str='bbs.blueidea.com';
execReg(reg,str);
```
这个正则是为了实现只匹配以b或者c开头的字符串，一直匹配到换行字符，但是。上面我们已经看到了，可以使用“\1”来反向引用这个括号里的子正则表达式所匹配的内容。而且exec方法也会将这个字正则表达式的匹配结果保存到返回的结果中。

**不记录子正则表达式的匹配结果**

使用形如(?:pattern)的正则就可以避免保存括号内的匹配结果。例如：
```
reg = /^(?:b|c).+/;
str='bbs.blueidea.com';
execReg(reg,str);
```
可以看到返回的结果不再包括那个括号内的字正则表达式多匹配的内容。
同理，反向引用也不好使了：
```
reg = /^(b|c)\1/;
str='bbs.blueidea.com';
execReg(reg,str);
```
返回bb,b。bb是整个正则表达式匹配的内容，而b是第一个子正则表达式匹配的内容。
```
reg = /^(?:b|c)\1/;
str='bbs.blueidea.com';
execReg(reg,str);
```
返回null。由于根本就没有记录括号内匹配的内容，自然没有办法反向引用了。

**正向预查**

形式：(?=pattern)
所谓正向预查，意思就是：要匹配的字符串，后面必须紧跟着pattern!
我们知道正则表达式/cainiao/会匹配cainiao。同样，也会匹配cainiao9中的cainiao。但是我们可能希望，cainiao只能匹配cainiao8中的菜鸟。这时候就可以像下面这样写：/cainiao(?=8)/，看两个实例：
```
reg = /cainiao(?=8)/;
str='cainiao9';
execReg(reg,str);
```
返回null。
```
reg = /cainiao(?=8)/;
```

```
str='cainiao8';
execReg(reg,str);
```
匹配**cainiao**。
需要注意的是，括号里的内容并不参与真正的匹配，只是检查一下后面的字符是否符合要求而已，例如上面的正则，返回的是cainiao，而不是cainiao8。
再来看两个例子：
```
reg = /blue(?=idea)/;
str='blueidea';
execReg(reg,str);
```
匹配到**blue**，而不是blueidea。
```
reg = /blue(?=idea)/;
str='bluetooth';
execReg(reg,str);
```
返回**null**，因为blue后面不是idea。
```
reg = /blue(?=idea)/;
str='bluetoothidea';
execReg(reg,str);
```
同样返回**null**。
**?!**
形式(?!pattern)和?=恰好相反，要求字符串的后面不能紧跟着某个pattern，还拿上面的例子：
```
reg = /blue(?!idea)/;
str='blueidea';
execReg(reg,str);
```
返回**null**，因为正则要求，blue的后面不能是idea。
```
reg = /blue(?!idea)/;
str='bluetooth';
execReg(reg,str);
```
则成功返回**blue**。
**匹配元字符**
首先要搞清楚什么是元字符呢？我们之前用过*,+,?之类的符号，它们在正则表达式中都有一定的特殊含义，类似这些有特殊功能的字符都叫做元字符。例如
```
reg = /c*/;
```
表示有任意个c，但是如果我们真的想匹配’c*’这个字符串的时候怎么办呢？只要将*转义了就可以了，如下：
```
reg = /c\*/;
str='c*';
execReg(reg,str);
```
返回匹配的字符串：**c***。
同理，要匹配其他元字符，只要在前面加上一个“\”就可以了。
**正则表达式的修饰符**
**全局匹配，修饰符g**
形式：/pattern/g
例子：reg = /b/g;
后面再说这个g的作用。先看后面的两个修饰符。
**不区分大小写，修饰符i**
形式：/pattern/i
例子：
```
var reg = /b/;
var str = 'BBS';
execReg(reg,str);
```
返回**null**，因为大小写不符合。
```
var reg = /b/i;
var str = 'BBS';
execReg(reg,str);
```
匹配到B，这个就是i修饰符的作用了。
**行首行尾，修饰符m**
形式：/pattern/m
m修饰符的作用是修改^和$在正则表达式中的作用，让它们分别表示行首和行尾。例如：
```
var reg = /^b/;
```

```
var str = 'test\nbbs';
execReg(reg,str);
```

匹配失败，因为字符串的开头没有b字符。但是加上m修饰符之后：

```
var reg = /^b/m;
var str = 'test\nbbs';
execReg(reg,str);
```

匹配到b，因为加了m修饰符之后，^已经表示行首，由于bbs在字符串第二行的行首，所以可以成功地匹配。

**exec方法详解**

**exec方法的返回值**

exec方法返回的其实并不是匹配结果字符串，而是一个对象，简单地修改一下execReg函数，来做一个实验就可以印证这一点：

```
function  execReg(reg,str){
  var result =  reg.exec(str);
 alert(typeof result);
}
var reg = /b/;
var  str='bbs.bblueidea.com';
execReg(reg,str);
```

结果显示result的类型是object。而且是一个类似数组的对象。使用for in可以知道它的属性：index input 0。其中index是表示匹配在原字符串中的索引；而input则是表示输入的字符串；

至于0则是表示只有一个匹配结果，可以用下标0来引用这个匹配结果，这个数量可能改变。我们可以通过返回值的length属性来得知匹配结果的总数量。

根据以上对返回值的分析，修改execReg函数如下：

```
function  execReg(reg,str){
  var result =  reg.exec(str);
  document.write('index:'+result.index+'<br />' + 'input:'+result.input+'<br />');
  for(i=0;i<result.length;i++){
    document.write('result['+i+']:'+result[i]+'<br />')
  }
}
```

马上来实验一下：

```
var reg = /\w/;
var  str='bbs.bblueidea.com';
execReg(reg,str);
```

结果如下：

- index:0
- input:bbs.bblueidea.com
- result[0]:b

输入字符串为bbs.bblueidea.com；

匹配的b在原字符串的索引是0。

正则的匹配结果为一个，b；

```
var reg =  /(\w)(\w)(.+)/;
var  str='bbs.bblueidea.com';
execReg(reg,str);
```

结果为：

- index:0
- input:bbs.bblueidea.com
- result[0]:bbs.bblueidea.com
- result[1]:b
- result[2]:b
- result[3]:s.bblueidea.com

由上面两个例子可见，返回对象[0]就是整个正则表达式所匹配的内容。后续的元素则是各个子正则表达式的匹配内容。

**exec方法对正则表达式的更新**

exec方法在返回结果对象的同时，还可能会更新原来的正则表达式，这就要看正则表达式是否设置了g修饰符。先来看两个例子吧：

```
var reg = /b/;
var str =  'bbs.blueidea.com';
execReg(reg,str);
execReg(reg,str);
```

结果如下：

- index:0
- input:bbs.blueidea.com
- result[0]:b
- index:0
- input:bbs.blueidea.com
- result[0]:b

也就是说，两次匹配的结果完全一样，从索引可以看出来，匹配的都是字符串首的b字符。
下面看看设置了g的正则表达式表现如何：

```
var reg = /b/g;
var str =  'bbs.blueidea.com';
execReg(reg,str);
execReg(reg,str);
```

结果如下：

- index:0
- input:bbs.blueidea.com
- result[0]:b
- index:1
- input:bbs.blueidea.com
- result[0]:b

可以看得出来，第二次匹配的是字符串的字符串的第二个b。这也就是g修饰符的作用了，下面来看exec是如何区别对待g和非g正则表达式的。
如果正则表达式没有设置g，那么exec方法不会对正则表达式有任何的影响，如果设置了g，那么exec执行之后会更新正则表达式的lastIndex属性，表示本次匹配后，所匹配字符串的下一个字符的索引，下一次再用这个正则表达式匹配字符串的时候就会从上次的lastIndex属性开始匹配，也就是上面两个例子结果不同的原因了。

**test方法**

test方法仅仅检查是否能够匹配str，并且返回布尔值以表示是否成功。同样建立一个简单的测试函数：

```
function  testReg(reg,str){
 alert(reg.test(str));
}
```

实例1

```
var reg = /b/;
var str =  'bbs.blueidea.com';
testReg(reg,str);
```

成功，输出`true`。
实例2

```
var reg = /9/;
var str =  'bbs.blueidea.com';
testReg(reg,str);
```

失败，返回`false`。

**使用字符串的方法执行正则表达式**

**match方法**

形式：str.match(reg);
与正则表达式的exec方法类似，该方法同样返回一个类似数组的对象，也有input和index属性。我们定义如下一个函数用来测试：

```
function  matchReg(reg,str){
  var result =  str.match(reg);
  if(result ){
    document.write('index:'+result.index+'<br />'+'input:'+result.input+'<br />');
    for(i=0;i<result.length;i++){
      document.write('result['+i+']:'+result[i]+'<br />')
    }
  }else{
    alert('null：匹配失败！')
  }
}
```

例如：

```
var reg = /b/;
var str =  'bbs.blueidea.com';
matchReg(reg,str);
```

结果如下：

- index:0
- input:bbs.blueidea.com
- result[0]:b

可见，和exec的结果一样。
但是如果正则表达式设置了g修饰符，exec和match的行为可就不一样了，见下例：

- index:undefined
- input:undefined
- result[0]:b
- result[1]:b
- result[2]:b

设置了g修饰符的正则表达式在完成一次成功匹配后不会停止，而是继续找到所有可以匹配到的字符。返回的结果包括了三个b。不过没有提供input和index这些信息。

**replace方法**

形式：str. replace (reg,’ new str’);
它的作用是将str字符串中匹配reg的部分用’ ’ new str”部分代码，值得注意的是原字符串并不会被修改，而是作为返回值被返回。例子：

```
var reg = /b/;
var str = 'bbs.blueidea.com';
var newStr = str.replace(reg,'c');
document.write(newStr);
```

结果为cbs.blueidea.com，只有第一个b被替换为c。

```
var reg = /b/g;
var str = 'bbs.blueidea.com';
var newStr = str.replace(reg,'c');
document.write(newStr);
```

输出ccs.clueidea.com
由于，设置了g修饰符，所以会替换掉所有的b。

```
var reg = /\w+/g;
var str = 'bbs.blueidea.com';
var newStr = str.replace(reg,'word');
document.write(newStr);
```

输出：

word.word.word。

**在replace函数中使用$引用子正则表达式匹配内容**

就像在正则里我们可以使用\1来引用第一个子正则表达式所匹配的内容一样，我们在replace函数的替换字符里也可以使用$1来引用相同的内容。
还是来看一个例子吧：

```
var reg = /(\w+).(\w+).(\w+)/;
var str = 'bbs.blueidea.com';
var newStr = str.replace(reg,'$1.$1.$1');
document.write(newStr);
```

输出的结果为：

bbs.bbs.bbs

首先，我们知道第一个子正则表达式匹配到了bbs，那么$1也就代表bbs了。其后我们把替换字符串设置为’$1.$1.$1’,其实也就是“bbs.bbs.bbs”。
同理，$2就是blueidea，$3就是com。
在来看一个例子，颠倒空格前后两个单词的顺序。

```
var reg = /(\w+)\s(\w+)/;
var str = 'cainiao gaoshou';
var newStr = str.replace(reg,'$2 $1');
document.write(newStr);
```

结果为：gaoshou cainiao，也就是空格前后的单词被调换顺序了。
由于在替换文本里$有了特殊的含义，所以我们如果想要是用$这个字符的话，需要写成$$，例如：

```
var reg = /(\w+)\s(\w+)/;
var str = 'cainiao gaoshou';
var newStr = str.replace(reg,'$$ $$');
document.write(newStr);
```

结果为：$ $。

**search方法和split方法**

同样，字符串的search方法和split方法中也可以使用正则表达式，形式如下：
str.search(reg);
search返回正则表达式第一次匹配的位置。例子：

```
var reg = /idea/;
var str = 'blueidea';
var pos =  str.search(reg);
document.write(pos);
```

结果为4。

下面的例子找出第一个非单词字符：

```
var reg = /\W/;
var str =  'bbs.blueidea.com';
var pos =  str.search(reg);
document.write(pos);
```

结果为3，也就是那个点“.”的位置。

```
str.split(reg,’ seprator’ );
split返回分割后的数组，例如：
var reg = /\W/;
var str =  'bbs.blueidea.com';
var arr =  str.split(reg);
document.write(arr);
```

结果为：bbs,blueidea,com，可见数组被非单词字符分为了有三个元素的数组。

```
var  reg = /\W/;
var  str = 'http://www.baidu.com/';
var  arr = str.split(reg);
document.write(arr.length+'<br  />');
document.write(arr);
```

结果为：

```
7
http,,,www,baidu,com,
```

可见字符串被分为了有7个元素的数组，其中包括了三个为空字符串的元素。

## ㈡. 经典JavaScript正则表达式实战

**关于测试代码**

本文不是使用Dreamweaver编辑，以下测试代码可能已经在赋值粘贴的过程做了一些调整，可能执行失效。

**匹配结尾的数字**

如

```
30CAC0040 取出40
3SFASDF92 取出92
正则如下：/\d+$/g
```

**统一空格个数**

字符串内如有空格，但是空格的数量可能不一致，通过正则将空格的个数统一变为一个。

例如：蓝 色  理    想

变成：蓝 色 理 想

```
<script type="text/javascript">
  var str="蓝 色 理 想"
  var reg=/\s+/g
  str = str.replace(reg," ")
  document.write(str)
</script>
```

**判断字符串是不是由数字组成**

这个正则比较简单，写了一个测试

```
<script type="text/javascript">
  function isDigit(str){
    var reg = /^\d*$/;
    return reg.test(str);
  }
```

```
  var str = "7654321";
  document.write(isDigit(str));
   var str = "test";
  document.write(isDigit(str));
</script>
```

**电话号码正则**

```
/^\d{3,4}-\d{7,8}(-\d{3,4})?$/
```

区号必填为3-4位的数字，区号之后用“-”与电话号码连接

```
^\d{3,4}-
```

电话号码为7-8位的数字

```
\d{7,8}
```

分机号码为3-4位的数字，非必填，但若填写则以“-”与电话号码相连接

```
(-\d{3,4})?
```

**手机号码正则表达式**

正则验证手机号，忽略前面的0，支持130-139，150-159。忽略前面0之后判断它是11位的。

```
/^0*(13|15)\d{9}$/
```

^0*匹配掉开头任意数量的0。
由于手机号码是13任意数字9位，和15任意数字9位，所以可以用(13|15)\d{9}匹配。
测试代码如下：

```
function testReg(reg,str){
 return reg.test(str);
}
var reg = /^0*(13|15)\d{9}$/;
var str = '13889294444';
var str2 = '12889293333';
var str3 = '23445567';
document.write(testReg(reg,str)+'<br />');
document.write(testReg(reg,str2)+'<br />');
document.write(testReg(reg,str3)+'<br />');
```

**使用正则表达式实现删除字符串中的空格：**

代码以及测试代码如下：

```
<script type="text/javascript">
//删除字符串两侧的空白字符。
function trim(str){
 return str.replace(/^\s+|\s+$/g,'');
}
//删除字符串左侧的空白字符。
function ltrim(str){
 return str.replace(/^\s+/g,'');
}
//删除字符串右侧的空白字符。
function rtrim(str){
 return str.replace(/\s+$/g,'');
}
//以下为测试代码
var trimTest = " 123456789 ";
//前后各有一个空格。
document.write('length:'+trimTest.length+'<br />');
//使用前
document.write('ltrim length:'+ltrim(trimTest).length+'<br />');
//使用ltrim后
```

```
document.write('rtrim length:'+rtrim(trimTest).length+'<br />');
//使用rtrim后
document.write('trim length:'+trim(trimTest).length+'<br />');
//使用trim后
</script>
```

测试的结果如下：

```
length:11
ltrim length:10
rtrim length:10
trim length:9
```

**限制文本框只能输入数字和小数点等等**

只能输入数字和小数点

```
var reg = /^\d*\.?\d{0,2}$/
```

开头有若干个数字，中间有0个或者一个小数点，结尾有0到2个数字。

只能输入小写的英文字母和小数点，和冒号，正反斜杠(：./\)

```
var reg = /[a-z\.\/\\:]+/;
```

a-z包括了小写的英文字母，\.是小数点，\/和\\分别是左右反斜线，最后是冒号。整个组成一个字符集和代码任一均可，最后在加上+，1或者多个。

**替换小数点前内容为指定内容**

请问 怎么把这个字符串的小数点前面的字符替换为我自定义的字符串啊？

例如：infomarket.php?id=197 替换为 test.php?id=197

应该可以把第一个点“.”之前的所有单词字符替换为test就可以了。我写的正则如下：

```
<script type="text/javascript">
var str = "infomarket.php?id=197";
var reg = /^\w*/ig;
//匹配字符串开头的任意个单词字符
str = str.replace(reg,'test');
document.write(str);
</script>
```

**只匹配中文的正则表达式**

前两天看的《JavaScript开发王》里恰好有中文的unicode范围，正则如下：

```
/[\u4E00-\u9FA5\uf900-\ufa2d]/
```

写了一个简单的测试，会把所有的中文替换成“哦”。

```
<script type="text/javascript">
var str = "有中文?and English.";
var reg = /[\u4E00-\u9FA5\uf900-\ufa2d]/ig;
str = str.replace(reg,'哦');
document.write(str);
</script>
```

**返回字符串的中文字符个数**

一般的字符长度对中文和英文都是不分别的 如JS里的length，那么如何返回字符串中中文字符的个数呢？guoshuang老师在原帖中给出了解决方案，我又没看懂……

不过我自己也想到了一个办法：先去掉非中文字符，再返回length属性。函数以及测试代码如下：

```
<script type="text/javascript">
function cLength(str){
 var reg = /[^\u4E00-\u9FA5\uf900-\ufa2d]/g;
 //匹配非中文的正则表达式
 var temp = str.replace(reg,'');
 return temp.length;
}
var str = "中文123";
document.write(str.length+'<br />');
document.write(cLength(str));
</script>
```

结果：

```
5
2
```

中文两个，数字三个，正确。
下面的测试也正确。

```
var str = "中文123tets@#!#%$#[][{}";
document.write(str.length+'<br />');
document.write(cLength(str));
```

**正则表达式取得匹配IP地址前三段**

192.168.118.101，192.168.118.72，192.168.118.1都替换成：192.168.118
只要匹配掉最后一段并且替换为空字符串就行了，正则如下：

```
/\.\d{1,3}$/
```

匹配结尾的.n,.nn或者.nnn。
测试代码如下：

```
function replaceReg(reg,str){
 return str.replace(reg,'')
}
var reg = /\.\d{1,3}$/;
var str = '192.168.118.101';
var str2 = '192.168.118.72';
var str3 = '192.168.118.1';
document.write(replaceReg(reg,str)+'<br />');
document.write(replaceReg(reg,str2)+'<br />');
document.write(replaceReg(reg,str3)+'<br />');
```

相似的有，这个帖子里有一个验证IP地址的方法：求检验MAC地址的正则表达例子

**匹配<ul>与<ul>之间的内容**

<ul>safsf<ul>safsf</ul><ul>safsf</ul></ul>
用正则可以得到 <ul>起到下个<ul> 之间的内容。
正则如下：

```
/<ul>[\s\S]+?<ul>/i
```

首先匹配两侧的ul标签，中间的[\s\S]+?可以匹配一个或者多个任意字符，一定要非贪婪，否则会匹配<ul>safsf<ul>safsf</ul><ul>。

**用正则表达式获得文件名**

c:\images\tupian\006.jpg
可能是直接在盘符根目录下，也可能在好几层目录下，要求替换到只剩文件名。
xlez的正则如下：

```
/[^\\\/]*[\\\/]+/g
```

首先匹配非左右斜线字符0或多个，然后是左右斜线一个或者多个。形如“xxx/”或者“xxx\”或者“/”或者“\”
函数以及测试代码：

```
<script type="text/javascript">
function getFileName(str){
 var reg = /[^\\\/]*[\\\/]+/g;
 //xxx\或者是xxx/
 str = str.replace(reg,'');
 return str;
}
var str = "c:\\images\\tupian\\006.jpg";
document.write(getFileName(str)+'<br />');
var str2 = "c:/images/tupian/test2.jpg";
document.write(getFileName(str2));
</script>
```

注意，\需要转义。

**绝对路径变相对路径**

将<IMG height="120" width="800" src="http://23.123.22.12/image/somepic.gif">转换为：<IMG height="120" width="800" src="/image/somepic.gif">。
其中网址可能改变，例如http://localhost等等。
cloudchen的正则：

```
/http:\/\/[^\/]+/
```
首先是http://，然后[^\/]+找过1个或者多个非/字符，因为遇到第一个/表示已经到目录了，停止匹配。
测试代码如下：
```
<script type="text/javascript">
  var str = '<IMG height="120" width="800" \
  src="http://23.123.22.12/image/somepic.gif">';
  var reg = /http:\/\/[^\/]+/;
  str = str.replace(reg,"");
  alert(str)
</script>
```
**用户名正则**
用于用户名注册，，用户名只 能用 中文、英文、数字、下划线、4-16个字符。
hansir和解决方案弄成正则：
```
/^[\u4E00-\u9FA5\uf900-\ufa2d\w]{4,16}$/
```
中文字符或者单词字符，4到16个。实现4到16结成到正则里的关键就是开始^和结束$，这就等于整个字符串只能有这些匹配的内容，不能有多余的。
函数和测试代码如下：
```
<script type="text/javascript">
  function isEmail(str){
    var reg = /^[\u4E00-\u9FA5\uf900-\ufa2d\w]{4,16}$/;
    return reg.test(str);
  }
  var str = '超级无敌用户名regExp';
  var str2 = '捣乱的@';
  var str3 = '太短'
  var str4 = '太长longlonglonglonglonglonglong'
  document.write(isEmail(str)+'<br />');
  document.write(isEmail(str2)+'<br />');
  document.write(isEmail(str3)+'<br />');
  document.write(isEmail(str4)+'<br />');
</script>
```
**匹配英文地址**
规则如下：
包含 ″点″, ″字母″,″空格″,″逗号″,″数字″，但开头和结尾不能是除字母外任何字符。
[\.a-zA-Z\s,0-9]这个字符集就实现了字母，空格，逗号和数字。最终正则如下：
```
/^[a-zA-Z][\.a-zA-Z\s,0-9]*?[a-zA-Z]+$/
```
开头必须有字母，结束也必须是一个以上字母。测试代码如下：
```
<script type="text/javascript">
  function testReg(reg,str){
    return reg.test(str);
  }
  var reg = /^[a-zA-Z][\.a-zA-Z\s,0-9]*?[a-zA-Z]+$/;
  var str = 'No.8,ChangAn Street,BeiJing,China';
  var str2 = '8.No,ChangAn Street,BeiJing,China';
  var str3 = 'No.8,ChangAn Street,BeiJing,China88';
  document.write(testReg(reg,str)+'<br />')
  document.write(testReg(reg,str2)+'<br />')
  document.write(testReg(reg,str3)+'<br />')
</script>
```
**正则匹配价格**
价格的格式应该如下：
开头数字若干位，可能有一个小数点，小数点后面可以有两位数字。hansir给出的对应正则如下：
```
/^(\d*\.\d{0,2}|\d+).*$/
```
hansir给出的测试代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>无标题文档</title>
<script type="text/javascript">
  function checkPrice(me){
    if(!(/^(?:\d+|\d+\.\d{0,2})$/.test(me.value))){
      me.value = me.value.replace(/^(\d*\.\d{0,2}|\d+).*$/,'$1');
    }
  }
</script>
</head>
<body>
<input type="text" onkeyup="checkPrice(this);"/>
</body>
</html>
```

**身份证号码的匹配**

身份证号码可以是15位或者是18位，其中最后一位可以是X。其它全是数字，正则如下：

```
/^(\d{14}|\d{17})(\d|[xX])$/
```

开头是14位或者17位数字，结尾可以是数字或者是x或者是X。

测试代码如下：

```
<script type="text/javascript">
  function testReg(reg,str){
    return reg.test(str);
  }
  var reg = /^(\d{14}|\d{17})(\d|[xX])$/;
  var str = '123456789012345';//15位
  var str2 = '123456789012345678';//18位
  var str3 = '1234567890123456X';//最后一位是X
  var str4 = '1234';//位数不对
  document.write(testReg(reg,str)+'<br />');
  document.write(testReg(reg,str2)+'<br />');
  document.write(testReg(reg,str3)+'<br />');
  document.write(testReg(reg,str4)+'<br />');
</script>
```

**要求文本有指定行数**

匹配至少两行的字符串，每行都为非空字符。

只要匹配到[\n\r]就表示有换行了，再保证换行的两段都不是空字符就可以了。正则如下：

```
/\S+?[\n\r]\S+?/i
```

这个正则的应用应该是用在textarea里，如果是如下要求：可以支持所有字符，中间可带空格，可以包括英文、数字、中文、标点这样的话，只要针对空格再改一下就行了。（按照非空的要求，上面有不能匹配“字符+空格+换行+字符”的字符串）。修改如下：

```
/\S+?\s*?[\n\r]\s*?\S+?/i
```

**单词首字母大写**

每单词首字大写，其他小写。如blue idea转换为Blue Idea，BLUE IDEA也转换为Blue Idea

cloeft的正则：

```
/\b(\w)|\s(\w)/g
```

所谓“首字母”包括两种情况：第一种是边界(开头)的单词字符，一种是空格之后的新单词的第一个字母。测试代码如下：

```
<script type="text/javascript">
  function replaceReg(reg,str){
    str = str.toLowerCase();
```

```
    return str.replace(reg,function(m){return m.toUpperCase()})
  }
  var reg = /\b(\w)|\s(\w)/g;
  var str = 'blue idea';
  var str2 = 'BLUE IDEA';
  var str3 = 'Test \n str is no good!';
  var str4 = 'final test';
  document.write(replaceReg(reg,str)+'<br />');
  document.write(replaceReg(reg,str2)+'<br />');
  document.write(replaceReg(reg,str3)+'<br />');
  document.write(replaceReg(reg,str4)+'<br />');
</script>
```

**正则验证日期格式**

yyyy-mm-dd格式

正则如下：

```
/^\d{4}-\d{1,2}-\d{1,2}$/
```

4位数字，横线，1或者2位数字，再横线，最后又是1或者2位数字。

测试代码如下：

```
<script type="text/javascript">
  function testReg(reg,str){
    return reg.test(str);
  }
  var reg = /^\d{4}-\d{1,2}-\d{1,2}$/;
  var str = '2008-8-8';
  var str2 = '2008-08-08';
  var str3 = '08-08-2008';
  var str4 = '2008 08 08';
  document.write(testReg(reg,str)+'<br />');
  document.write(testReg(reg,str2)+'<br />');
  document.write(testReg(reg,str3)+'<br />');
  document.write(testReg(reg,str4)+'<br />');
</script>
```

第二种格式：

yyyy-mm-dd

或

yyyy/mm/dd

用“或”简单地修改一下就行了。

```
/^\d{4}(-|\/)\d{1,2}(-|\/)\d{1,2}$/
```

**去掉文件的后缀名**

www.abc.com/dc/fda.asp 变为 www.abc.com/dc/fda

如果文件后缀已知的话这个问题就非常简单了，正则如下：

```
/\.asp$/
```

匹配最后的.asp而已，测试代码如下：

```
<script type="text/javascript">
  function delAspExtension(str){
    var reg = /\.asp$/;
    return str.replace(reg,'');
  }
  var str = 'www.abc.com/dc/fda.asp';
  document.write(delAspExtension(str)+'<br />');
</script>
```

如果文件名未知的话就用这个正则：/\.\w+$/，测试代码如下：

```
<script type="text/javascript">
    function delExtension(str){
        var reg = /\.\w+$/;
        return str.replace(reg,'');
    }
    var str = 'example.com/dc/fda.asp';
    document.write(delExtension(str)+'<br />');
    var str2 = 'test/regular/fda.do';
    document.write(delExtension(str2)+'<br />');
    var str3 = 'example.com/dc/fda.strange_extension';
    document.write(delExtension(str3)+'<br />');
</script>
```

**验证邮箱的正则表达式**

正则：

```
/^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(\.[a-zA-Z0-9_-])+/
```

开始必须是一个或者多个单词字符或者是-，加上@，然后又是一个或者多个单词字符或者是-。然后是点“.”和单词字符和-的组合，可以有一个或者多个组合。

```
<script type="text/javascript">
    function isEmail(str){
        var reg = /^([a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(\.[a-zA-Z0-9_-])+/;
        return reg.test(str);
    }
    var str = 'test@hotmail.com';
    document.write(isEmail(str)+'<br />');
    var str2 = 'test@sima.vip.com';
    document.write(isEmail(str2)+'<br />');
    var str3 = 'te-st@qq.com.cn';
    document.write(isEmail(str3)+'<br />');
    var str4 = 'te_st@sima.vip.com';
    document.write(isEmail(str4)+'<br />');
    var str5 = 'te._st@sima.vip.com';
    document.write(isEmail(str5)+'<br />');
</script>
```

我不太了解邮箱的具体规则。感觉这个正则比较简单，EMAIL校验 正则 讨论 求解里有比较详细的邮箱正则讨论。

**匹配源代码中的链接**

能够匹配HTML代码中链接的正则。

原帖正则：

```
/<a href=".+?">.+?<\/a>/g
```

感觉有点严格，首先要<a href=””.+?”>有，而且href属性可以是一个或者多个除换行外任意字符(非贪婪)。后面是.+?<\/a>，一个或者多个除换行外任意字符（非贪婪），再加上结束标签。

有个问题，如果a的起始标签最后有空格，或者除了href还有其它属性的话，上面的正则就不能匹配这个链接了。

例如：

<a href=”asdfs” >……多了个空格。

<a id=”xx” href=””asdfs”>……前面有属性。

……

重写正则：

```
/<a\s(\s*\w*?=".+?")*(\s*href=".+?")(\s*\w*?=".+?")*\s*>[\s\S]*?<\/a>/
```

思路如下：首先要有<a和一个空格。/<a\s/

第一个(\s*\w*?=”.+?”)*

可以匹配一个属性，属性前面可能有或者没有多余的空格，用\s*匹配；属性名肯定是单词字符，用\w*?匹配；=”.+?”就是匹配属性值了非换行字符若干个；整个括号外面加个*表示可能有任意多个属性。

(\s*href=”.+?”)

匹配href，它也是一个属性，所以只要把上面子正则表达式中的\w修改为href=就行了。

(\s*\w*?=”.+?”)*重复第一个子正则表达式，再次接受任意个属性。

\s*>，属性最后再加上若干个空格和>。

[\s\S]*?，链接的文字，可能有任何字符组成，若干个，非贪婪。

<\/a>最后是结束标签。
补充：属性名和=之间，以及=和属性值之间也可能有空格。所以要再加上几个\s*。
最后的实例代码如下：

```
<script type="text/javascript">
function findLinks(str){
    var reg = /<a\s(\s*\w*?\s*=\s*".+?")*(\s*href\s*=\s*".+?")(\s*\w*?\s*=\s*".+?")*\s*>[\s\S]*?<\/a>/g;
    var arr = str.match(reg);
    for(var i=0;i<arr.length;i++){
       //alert(arr[i]);
      document.write('link:'+arr[i]+'<br />');
    }
}
var str = '<p>测试链接：<a id = "test" href="http://bbs.blueidea.com" title="无敌">经典论坛
</a></p><a? href = "http://www.blueidea.com/"? >蓝色理想</a>';
var arr = findLinks(str);
</script>
```

会把所有的链接在页面直接显示出来。注意，
**本帖遗留问题：如何执行从右到左的匹配。貌似JS或者VBS没有提供这个功能2、JS或者VBS不支持后行断言。。用什么方法实现这个功能。**

**匹配链接的文字**

代码：<a href="#>这里要保存</a>，只保存链接的文本内容，标签信息删掉。
前面写过一个匹配链接的正则：

```
/<a\s(\s*\w*?=".+?")*(\s*href=".+?")(\s*\w*?=".+?")*\s*>[\s\S]*?<\/a>/
```

不过我们需要捕获的是文字内容，所以需要做一定的修改。第一步就是在所有的括号内都加上?:表示不捕获。第二步就是再多加一个括号放在[\s\S]*?两侧，这样就可以捕获到链接的文字内容了。最后正则如下：

```
/<a\s(?:\s*\w*?\s*=\s*".+?")*(?:\s*href\s*=\s*".+?")(?:\s*\w*?\s*=\s*".+?")*\s*>([\s\S]*?)<\/a>/
```

测试代码如下：

```
<script type="text/javascript">
function anchorText(str){
    var reg =/<a\s(?:\s*\w*?\s*=\s*".+?")*(?:\s*href\s*=\s*".+?")(?:\s*\w*?\s*=\s*".+?")*\s*>([\s\S]*?)<\/a>/;
    str = str.replace(reg,'$1');
    return str;
}
var str = '<a id = "test" href="http://bbs.blueidea.com" title="无敌">经典论坛</a>';
document.write(anchorText(str));
</script>
```

**正则判断标签是否闭合**

例如：<img xxx=" xxx" 就是没有闭合的标签；
p的内容，同样也是没闭合的标签。
从简单的正则开始，先匹配起始标签

```
/<[a-z]+/i
```

再加上若干属性：

```
/<[a-z]+(\s*\w*?\s*=\s*".+?")*/i
```

下面就到关键点了，标签的闭合。标签可能有两种方式闭合，<img xxx=" xxx" />
或者是 xxx 。
(\s*\/>)
匹配img类的结束，即/>。
(\s*?>[\s\S]*?<\/\1>)
匹配\p类标签的结束标签。>是其实标签末尾，之后是标签内容若干个任意字符，最后的<\/\1>就是结束标签了。
加上一个或就可以解决了，最后的完整正则表达式：
整个正则：

```
/<([a-z]+)(\s*\w*?\s*=\s*".+?")*(\s*?>[\s\S]*?<\/\1>|\s*\/>)/i
```

拿这个正则，只要匹配到了就表示闭合，没匹配到则没有闭合。不过没有考虑相同标签嵌套的问题，例如
aaaaaa test
也被判断为合格，可以通过把最后的匹配p类结束标签写成子正则表达式，并且更改为非贪心，然后在匹配结果中检查是否成对。正则如下：

```
/<([a-z]+)(\s*\w*?\s*=\s*".+?")*(\s*?>[\s\S]*?(<\/\1>)+|\s*\/>)/i
```

**用正则获得指定标签的内容**

有如下代码：

```
<channel>
  <title>蓝色理想</title>
</channel>
<item>
  <title>界面设计测试规范</title>
</item>
<item>
    <title>《古典写实美女》漫画教程</title>
</item>
<item>
    <title>安远¯消失的光年</title>
</item>
<item>
    <title>asp.net 2.0多语言网站解决方案</title>
</item>
```

要求匹配item里的title而不匹配channel里的title。
基本正则：

```
/<title>[\s\S]*?<\/title>/gi
```

首先是title标签，内容为任意字符若干个，然后是title结束标签。这个正则已经能匹配到所有的title标签。
首先，我简单地修改了一下原正则：

```
/<title>[^<>]*?<\/title>/gi ,
```

因为title里面不应该再嵌有其它标签，这个正则同样是匹配所有标题的内容，最后再加上不去匹配channel中的title。整个正则如下：

```
/<title>[^<>]*?<\/title>(?!\s*<\/channel>)/gi
```

(?!\s*<\/channel>)表示要匹配字符串的后面不能跟着若干个空格和一个channel的结束标签。
原帖里有很方便的测试工具，这里就不给测试代码了。

**正则判断是否为数字与字母的混合**

不能小于12位，且必须为字母和数字的混合。
验证字符串包含数字简单，验证字符串包含字母也简单，验证字符串不包含其它字符也简单，可以用这三个正则分别检查一次字符串，逻辑运算出最终结果。
但是怎么能把这些功能写进一个正则表达式里呢？这个问题真有点伤脑筋。
下面是lexrus的正则：

```
/^([a-z]+(?=[0-9])|[0-9]+(?=[a-z]))[a-z0-9]+$/ig
```

思路非常的清晰啊：
[a-z]+(?=[0-9])
字母开头，后面必须紧跟着数字。
[0-9]+(?=[a-z]
数字开头，后面必须紧跟着字母。
[a-z0-9]+
后面的字符只要是数字或者字母就可以了。经过测试，发现不好使，123dd会被识别为不合法，dd123则为合法，可见“数字开头，紧跟字母”的正则没有起作用。测试代码如下：

```
<script type="text/javascript">
function istrue(str){
    var reg=/^([a-z]+(?=[0-9])|[0-9]+(?=[a-z]))[a-z0-9]+$/ig;
    return reg.test(str);
}
var str? = 'AaBc';
var str2 = 'aaa123';
var str3 = '123dd';
var str4 = '1230923403982';
document.write(istrue(str)+'<br />');
document.write(istrue(str2)+'<br />');
document.write(istrue(str3)+'<br />');
document.write(istrue(str4)+'<br />');
</script>
```

结果为：

false，true，false，false
结果中的第三个，将'123dd'判断为非法是错误的。刚开始以为是g的问题，去掉了还是不好使。应该是浏览器bug，我认为lexrus的正则是正确的，可能是浏览器无法处理或”|”的两边都包含正向预查(?=)。
修改之后的正则如下：

```
/^(([a-z]+[0-9]+)|([0-9]+[a-z]+))[a-z0-9]*$/i
```

意思和上面差不多，但是没有使用正向预查，测试代码如下：

```
<script type="text/javascript">
function istrue(str){
var reg=/^(([a-z]+[0-9]+)|([0-9]+[a-z]+))[a-z0-9]*$/i;
return reg.test(str);
}
var str? = 'AaBc';
var str2 = 'aaa123';
var str3 = '123dd';
var str4 = '1230923403982';
document.write(istrue(str)+'<br />');
document.write(istrue(str2)+'<br />');
document.write(istrue(str3)+'<br />');
document.write(istrue(str4)+'<br />');
</script>
```

结果为
false，true，true，false
正确。

**空格与英文同时存在**

匹配英文以及空格，要求必须既有英文字母又有空格。
这个思路和上面的差不多，只要把数字改成空格就可以了。正则如下：

```
/^(([a-z]+\s+)|(\s+[a-z]+))[a-z\s]*$/i
```

英文开头加空格，或者是空格开头加英文，后面可以是英文或者空格。测试代码如下：

```
<script type="text/javascript">
function istrue(str){
var reg=/^(([a-z]+\s+)|(\s+[a-z]+))[a-z\s]*$/i;
return reg.test(str);
}
var str? = 'asdf';
var str2 = 'sadf sdf';
var str3 = 'asdf ';
document.write(istrue(str)+'<br />');
document.write(istrue(str2)+'<br />');
document.write(istrue(str3)+'<br />');
</script>
```

利用这个思路也可以实现英文空格英文，英文单词多于两个的匹配。同样，也可以把英文字母换成单词字符\w。

**显示或者保存正则表达式匹配的部分内容**

有如下电话号码：
13588888333
13658447322
13558885354
13587774654
13854554786
要求，要求只匹配135开头的电话，但是匹配结果只保留135后面的数字。
由于JavaScript里的正则不支持(?=xx)xxx的模式，只支持xxx(?=xx)的模式。所以只能将135后面的内容作为一个子正则表达式匹配的内容，然后再在后面引用。

```
Carl给出的函数如下：
function f(phoneNumber) {
  var pattern = /^(135)(\d{8})$/;
    if(pattern.test(phoneNumber))
```

```
 return phoneNumber.replace(pattern,"$2");
  else
 return "不是135打头的手机号码！";
}
/^(135)(\d{8})$/
```

正则中，135作为开头表示第一个子正则表达式，第二个括号内的子正则表达式则匹配后面的8个数字，然后在replace中使用$2就可以引用这个子正则表达式匹配的内容了。测试代码如下：

```
<script type="text/javascript">
function f(phoneNumber) {
  var pattern = /^(135)(\d{8})$/;
  if(pattern.test(phoneNumber))
 return phoneNumber.replace(pattern,"$2");
  else
 return "不是135打头的手机号码！";
}
var arr = new Array(
  "13588888333",
  "13658447322",
  "13558885354",
  "13587774654",
  "13854554786"
);
for(var i = 0; i < arr.length; i++)
  document.write(f(arr[i])+'<br />');
</script>
```

**正则表达式替换变量**

有一个数组：

```
var _A = ['A','B','C','D'];
```

有一个有“变量”的字符串。

```
var _B = '<ul><li>$0$</li><li>$1$</li><li>$2$</li><li>$3$</li></ul>';
```

说是变量，其实只是字符串中的特殊字符，例如$0$，就称这个为伪变量吧。

最后的要求就是使用正则获得下面这样一个字符串：

```
_C = '<ul><li>A</li><li>B</li><li>C</li><li>D</li></ul>';
```

IamUE给出了代码：

```
<script type="text/javascript">
var _A = ['A','B','C','D'];
var _B = '<ul><li>$0$</li><li>$1$</li><li>$2$</li><li>$3$</li></ul>';
var reg=/\$\d+\$/ig;
C=_B.replace(reg,function($1){
var indexnum=$1.replace(/\$/ig,"");
if (indexnum<_A.length)
{return _A[indexnum];}
else{return ""}
});
alert(C);
</script>
```

代码分析：看到代码之后感觉有点晕，首先，正则reg中没有任何的括号，应该是没有捕获内容的，那么后面怎么又使用$1了引用了呢？通过alert测试，发现它是整个正则匹配的内容，而且不一定要写作$1，可以写为$0，甚至是写为x都没关系，它总是整个匹配。

第一次，$1匹配到_B中的“$0$”，匿名函数中将它的$去掉，变成了0，检查是否越界之后，用这个0作为下标去访问数组_A。

由于正则reg定义了g属性，所以会继续替换$1$、$2$等等。步骤都和上面一样。

**正则替换指定属性中的文本**

有如下代码：

```
<td align="center"><img src="../photo/ccg/thumbnails/O'Malley's West_jpg.gif" border="0"
onClick="MM_openBrWindow('../photo/ccg/pages/O'Malley's West_jpg.htm','BE','width=386,height=306')"><br>
```

O'Malley's West</td>
要求将所有onclick属性中的'替换成\'，也就是将单引号转义。
首先，需要匹配onclick属性：

```
/onclick\s*=\s*".+?"/ig
```

然后再将所有的'都替换成\'就可以了。

**将阿拉伯数字替换为中文大写形式**

将123替换成壹贰叁。
只要匹配一个数字就可以了，测试代码如下（如果显示为乱码就调整一下浏览器的字符编码）：

```
function replaceReg(reg,str){
 return str.replace(reg,function(m){return arr[m];})
}
arr=new Array("零","壹","贰","叁","肆","伍","陆","柒","捌","玖");
var reg = /\d/g;
var str = '13889294444';
var str2 = '12889293333';
var str3 = '23445567';
document.write(replaceReg(reg,str)+'<br />');
document.write(replaceReg(reg,str2)+'<br />');
document.write(replaceReg(reg,str3)+'<br />');
```

**替换文本中的URL为链接**

将一个用户输入的一段文字中的url替换成可以点击的link地址。例如：http://www.blueidea.com可以替换成 [url]http://www.cctv.com[/url]或<a href="http://www.cctv.com">http://www. blueidea.com</a>.
这个正则的关键就在于匹配链接，匹配之后，在两边加上a标签和属性不是问题。

```
/http:\/\/[\w-]*(\.[\w-]*)+/ig
```

首先匹配http://。
[\w-]*是可能的www和bbs等。
\.[\w-]*匹配.xxx形式，至少有一个。
测试代码如下：

```
<script type="text/javascript">
function replaceReg(reg,str){
 return str.replace(reg,function(m){return '<a href="'+m+'">'+m+'</a>';})
}
var reg = /http:\/\/[\w-]*(\.[\w-]*)+/ig;
var str = '将一个用户输入的一段文字中的url替换成可以点击的link地址。\
测试一下：http://www.blueidea.com紧接着中文，还有http://bbs.blueidea.com \
is very good!http://blueidea.com!最后在看看带.cn的：http://www.sina.com.cn呵呵。';
document.write(replaceReg(reg,str)+'<br />');
</script>
```

**从HTML代码段删除指定标签极其内容**

在一段代码中去除<script ..... /script>, <head>...</head>,<%.....%>等代码块
隆的正则：

```
/<(script|meta|%)[\s\S]*?\/(script|meta|%)>/
```

试了一下，匹配如下文本正常：

```
<script type="text/javascript">
我是要被删除的脚本
</script>
哎。就剩下我了。
```

但是，如果使用类似的正则：

```
/<(script|head|%)[\s\S]*?\/(script|head|%)>/ig
```

匹配有嵌套的标签：

```
<head>
<script type="text/javascript">
我是要被删除的脚本
</script>
```

```
</head>
哎。就剩下我了。
```
实际匹配的内容是：
```
<head>
<script type="text/javascript">
我是要被删除的脚本
</script>
```
这是因为[\s\S]*?里的非贪婪造成的。可以使用JavaScript正则里的反向引用来解决这个问题，如果起始标签匹配了head，那么结束标签也必须是head。最后的正则如下：
```
/<(script|head|%)[\s\S]*?\/\1>/ig
```
**用正则给文本分段**

原代码：[title]标题一[/title]内容一232323sdfga [title]标题二[/title]内容二2232323 [title]标题三[/title]内容三2232323
要把文本分段成如下格式：
一、[title]标题一[/title]内容一232323sdfga
二、[title]标题二[/title]内容二2232323
三、[title]标题三[/title]内容三2232323
只要用正则匹配title就可以了，所以正则比较简单
```
/\[title\]/ig
```
至于开始的的汉字序号，只要一个数组就解决了，最终代码如下：
```
<script type="text/javascript">
function replaceReg(reg,str){
 var mark =0;
 return str.replace(reg,function(m){mark++;return '<br />'+arr[mark]+'、'+m;})
}
var arr = ["零","壹","贰","叁","肆","伍","陆","柒","捌","玖"];
var reg = /\[title\]/ig;
var str = '[title]标题一[/title]内容一232323sdfga \
[title]标题二[/title]内容二2232323 [title]标题三[/title]内容三2232323';
document.write(replaceReg(reg,str)+'<br />');
</script>
```
**转换源代码中的标签**

将代码中的HTML标签img转换为[img]url[/img]。
```
/<img(?:\s*\w*?\s*=\s*".+?")*?\s*src\s*=\s*"(.+?)"(?:\s*\w*?\s*=\s*".+?")*\s*>/ig
```
这段正则和匹配链接标签的正则基本一样，修改如下,标签名img，没有结束标签而是>结束。
测试代码如下：
```
<script type="text/javascript">
function replaceReg(reg,str){
 return str.replace(reg,'[img]$1[/img]')
}
var reg =
/<img(?:\s*\w*?\s*=\s*".+?")*?\s*src\s*=\s*"(.+?)"(?:\s*\w*?\s*=\s*".+?")*\s*>/ig;
var str = '我就是传说中的图片了<img src="URL">哎。';
document.write(replaceReg(reg,str)+'<br />');
</script>
```
第二个是替换object代码嵌入的flash代码替换为[swf]url[/swf]。
针对原文的正则如下：
```
/<object[\s\S]*?src=([\s\S]+?)(?=\s)[\s\S]*<\/object>/i
```
如果是所有的属性都有双引号的话正则也需要修改。
测试如下：
```
<script type="text/javascript">
function replaceReg(reg,str){
 return str.replace(reg,'[swf]$1[/swf]')
}
var reg = /<object[\s\S]*?src=([\s\S]+?)(?=\s)[\s\S]*<\/object>/i;
```

```
var str = '<object classid=clsid:D27CDB6E-AE6D-11cf-96B8-444553540000 \
codebase=http://download.macromedia.com/pub/shockwave/cabs/flash/\
swflash.cab#version=5,0,0,0 width=255 height=250><param name=movie \
value=url><param name=quality value=high><embed src=url quality=high \
pluginspage=http://www.macromedia.com/shockwave/download/index.cgi?\
P1_Prod_Version=ShockwaveFlash type=application/x-shockwave-flash \
width=255 height=250></embed></object>';
document.write(replaceReg(reg,str)+'<br />');
</script>
```

**给属性添加双引号**

给HTML标签中的属性添加双引号。

<a href=xxx>改为：<a href="xxx">

LeXRus的第一个正则如下：

```
/(?!\<\w+)(\s+\w+)\=([^>\"\s]+)/ig
```

第一个括号没看明白，JS应该是不支持。所以我擅自给删掉了，剩下的正则如下：

```
/(\s+\w+)\=([^>\"\s]+)/ig
```

第一个括号里的\s+\w+匹配的是属性名。

然后是=，不用转义。

第二个括号里的[^>\"\s]+匹配属性值。不匹配>”和空格。这里的引号不用转义。在意思不改变的情况下，稍微改了改，正则如下：

```
/(\s+\w+)=([^>"\s]+)/ig
```

需要注意的是这个正则不匹配=两边有空格的属性，例如href = xxx。相匹配的话就改成：

```
/(\s+\w+)\s*=\s*([^>"\s]+)/ig
```

代码：

```
str=str.replace(/(?!\<\w+)(\s+\w+)\=([^>\"\s]+)/ig,'$1="$2"');
```

其中’ $1=”$2”’ 就实现了给属性值添加上双引号。不过ncs指出了这个正则替换的几个问题，一是上面的空格问题，二是如果非标签内部有等号，且前面又恰巧有空白字符的话，它将会被误识别为属性，例如：

<a href=xxx target=yyy title = asdfasf> test=sd

里面的test=sd也会被匹配。三是如果属性原来使用了单引号，会被再包上一层双引号……

来看看LeXRus前辈的新正则替换方法：

```
str=str.replace(/(?!<\w+)(\s+\w+)\s*=\s*([^>\"\s]+)(?=[^>]*>)/ig,'$1="$2"')
.replace(/\"\'([^\'\"]+)\'\"/ig,'\"$1\"');
```

先来看第一个正则：

```
/ (\s+\w+)\s*=\s*([^>"\s]+)(?=[^>]*>)/ig
```

结尾新添的(?=[^>]*>)意在解决普通文本中有等号被误识别为属性的问题：

<a href=xxx target=yyy title = asdfasf> test=sd

就没问题了，但是

<a href=xxx target=yyy title = asdfasf> test=sd<tag>又一个标签</tag>

中的test=sd<tag>又会被识别为属性。

我觉得改成下面的正则就没问题了：

```
/(\s+\w+)\s*=\s*([^<>"\s]+)(?=[^<>]*>)/ig
```

分别在第二个括号的字符集合和最后的反向预查的字符集合中添加了一个<。

下面再来分析第二个正则，

```
/\"\'([^\'\"]+)\'\"/ig
```

这个正则用于匹配双引号，单引号多层嵌套的情况，同样，不用转义，修改正则如下：

```
/"'([^'"]*)'"/ig
```

这样基本任务就完成了。测试代码如下：

```
<script type="text/javascript">
function rp(str,trg){
 var reg1 = /(\s+\w+)\s*=\s*([^<>"\s]+)(?=[^<>]*>)/ig
 var reg2 = /"'([^'"]*)'"/ig;
 str=str.replace(reg1,'$1="$2"').replace(reg2,'\"$1\"');
 trg.value=str;
}
</script>
<textarea id="sou" style="width:100%">
<a href = xxx name=aaa target=_blank title='asdfasf'
onclick=alert('blueidea')> asfd=asfd
```

```
</textarea>
<input type="button" onclick="rp(sou.value,sou)" value="replace"/>
```

原帖里LeXRus又提出了新问题：

**hint=i am lexrus**

**这样的属性会有问题，不过我感觉不加引号的话，属性值里就不可能有空格，否则会被识别为多个属性了。不过看到最后ncs的回帖我就哭了：**

onclick=if(document.forms.length>0)

这样的属性怎么办？大于号会被识别为标签结束……还是分离行为与文档吧。补充一下，其实修补一下正则也可以解决，只要改成如下正则即可：

```
/(\s+\w+)\s*=\s*([^"\s]+)(?=[^<>]*>)/ig
```

就是去掉第二个括号内字符集合里的<>。最后这个问题也解决。

**给table加上tbody**

有若干table，但是没有tbody。现在需要用正则批量加上。

匹配table结束标签</table>比较简单，在前面加上一个</tbody>就行了。

但是，匹配table的起始标签有点难度，因为可能有属性。不过之前匹配过链接了，这个也大同小异。

实例table代码如下：

```
<table width="100%" border="0" cellpadding="2" cellspacing="3">
<table width="100%">
```

正则：

```
/<table\s(\s*\w*?\s*=\s*".+?")*?\s*?>/g
```

匹配一个<table，在匹配若干个属性，最后只要再找到>就代表标签结束。

之后再replace一下，加上<tbody>就可以了。

**去掉标签的所有属性**

```
<td style="width: 23px; height: 26px;" align="left">***</td>
```

变成没有任何属性的

```
<td>***</td>
```

思路：非捕获匹配属性，捕获匹配标签，使用捕获结果替换掉字符串。正则如下：

```
/(<td)\s(?:\s*\w*?\s*=\s*".+?")*?\s*?(>)/
```

首先，td匹配掉了标签，后面可以用$1引用，后面的若干属性被(?:)匹配掉，而最后匹配的>则可以在后面用$2引用。

示意代码：

```
str = str.replace(reg,' $1$2' );
```

**正则替换特定单词**

要求禁止输入某几个单词，如果拒绝red,yellow,white。这个帖子到时不难，但是让我弄清楚了好几个概念。

第一个，小心字符集合里的“或”

```
/[^red|yellow|white]/
```

这个正则里的所有或都没有意义，等同于：

```
/[^redyellowwhite]/
```

意思就是不能含有以下列出的所有字母。

正解：

```
/red|yellow|white/
```

第二个概念：

只要整个正则匹配成功，无论子正则表达式是否匹配成功，括号都会捕捉。例如

```
/(red)|(yellow)|(white)/
```

会捕捉到三个结果，尽管实际上最多只可能有一个括号匹配成功。但是只要有一个匹配到了，两外两个也会记录空串。

**指定文字高亮显示**

请教正则表达式：如何替换搜索结果中的关键字为高亮显示？

不劳而获一次，这个**子虚乌有**前辈已经给出了非常好的解决方案：我直接把代码贴出来了：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<META name="Author" content="Sheneyan" />
<script type="text/javascript">
function encode(s){
  return  s.replace(/&/g,"&amp;").replace(/</g,"&lt;").replace(/>/g,"&gt;").replace(/([\\\.\*\[\]\(\)\$\^])/g,"\\$1");
}
function decode(s){
  return  s.replace(/\\([\\\.\*\[\]\(\)\$\^])/g,"$1").replace(/&gt;/g,">").replace(/&lt;/g,"<").replace(/&amp;/g,"&");
}
function highlight(s){
```

```
 if (s.length==0){
alert('搜索关键词未填写！');
return false;
 }
 s=encode(s);
 var obj=document.getElementsByTagName("body")[0];
 var t=obj.innerHTML.replace(/<span\s+class=.?highlight.?>([^<>]*)<\/span>/gi,"$1");
 obj.innerHTML=t;
 var cnt=loopSearch(s,obj);
 t=obj.innerHTML
 var r=/{searchHL}((((?!\/searchHL})|[^{])*){\/searchHL}/g
 t=t.replace(r,"<span class='highlight'>$1</span>");
 obj.innerHTML=t;
 alert("搜索到关键词"+cnt+"处")
}
function loopSearch(s,obj){
 var cnt=0;
 if (obj.nodeType==3){
cnt=replace(s,obj);
return cnt;
 }
 for (var i=0,c;c=obj.childNodes[i];i++){
if (!c.className||c.className!="highlight")
cnt+=loopSearch(s,c);
 }
 return cnt;
}
function replace(s,dest){
 var r=new RegExp(s,"g");
 var tm=null;
 var t=dest.nodeValue;
 var cnt=0;
 if (tm=t.match(r)){
cnt=tm.length;
t=t.replace(r,"{searchHL}"+decode(s)+"{/searchHL}")
dest.nodeValue=t;
 }
 return cnt;
}
</script>
<style type="text/css">
.highlight{background:green;font-weight:bold;color:white;}
</style>
</head>
<body>
<form onsubmit="highlight(this.s.value);return false;">
<p><input name="s" id="s" title="搜索内容："/><input type="submit" value="搜索"/></p>
</form>
```

```
<div id="content">
测试高亮的代码。很长很长的代码.....
</div>
</body>
</html>
```

**删除标签**

删除除了<img>、<br>、<p>之外所有的标签。子虚乌有给出代码中关键的一句：

```
o.innerHTML.replace(/(<\/?(?!br|p|img)[^>\/]*)\/?>/gi,'');
```

刚开始没反应过来，后来才想起来，这个正则不用区分起始和结束标签。

<\/?(?!br|p|img)

匹配除了保护标签外标签的起始标签或者是结束标签的一部分。

[^>\/]*

匹配到>或者/就结束。

\/?>

起始标签或者结束标签的结尾。

源文档 <http://www.cainiao8.com/web/js_note/js_regular_expression_blueidea.html#_Toc213927708>