# 一、今日目标

1、能够掌握三种方式创建正则表达式

2、能够掌握正则语法中的量词

3、能够掌握正则语法中的元字符

4、能够掌握正则语法中的修饰符

5、能够掌握捕获和反向引用的用法

6、能够匹配中文

7、能够掌握正则断言的用法

8、能够掌握正则对象的常用属性和方法

9、能够掌握符合正则表达式的字符串方法

# 二、正则表达式概述

w3c手册位置：

上面的JavaScript --- 左侧的JavaScript --- 右侧的参考书 --- 左侧的JS RegExp

或者：

上面的JavaScript --- 右侧的JavaScript --- 左侧的JS RegExp

1、概述

正则表达式(regular expression)描述了一种字符串匹配的模式，可以用来检查一个字符串中是否含有某种子串、将匹配的子串做替换或者从某个字符串中取出符合某个条件的子串等。

匹配就是在某个字符串中查找你想要查找的内容（由正则表达式决定），如果查找到了，就表示匹配。没有查找到就表示不匹配。

正则表达式本身是一门独立的技术，可以被大多数编程语言所使用，如Javascript、PHP、Java…

正则表达式也是用于字符串查找的，只是更加高级的查找，查找之后可以按需直接使用或替换等。

比如在字符串“abacadabvcababacad”中查找a：

* JavaScript语法：'abacad'.search('a');

如果查询条件是下面的条件，用普通的查询就不能完成了。

要求查找a，什么样的a呢？后面必须是b的a。这正是正则表达式的用武之地。

2、应用场景

（1）表单验证里面，验证字符的合法性，如邮箱是否合法，手机号是否合法等等。

（2）信息过滤，如论坛帖子或者评论中的非法字符，例如 sql注入、js脚本注入、煽动性的言论。

（3）信息采集，采集别人网站上面的内容，例如整页采集时，筛选出需求的部分。

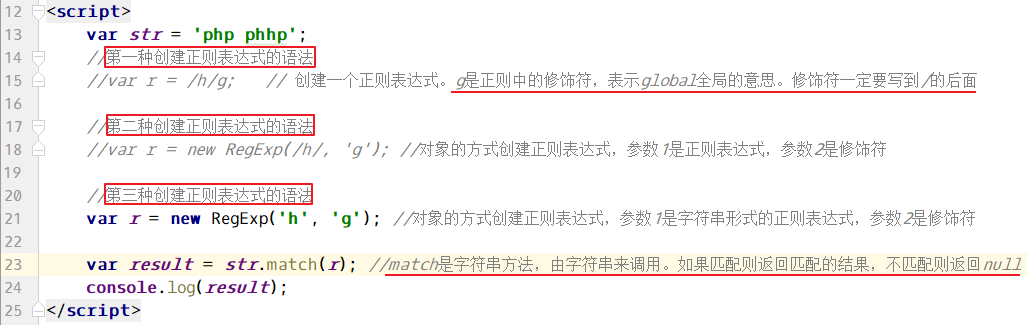
（4）信息替换。

（5）页面伪静态的规则。

# 三、入门案例

1、正则用法说明

小例子：在字符串“php phhp”中查找h。



如果没有修饰符g，则匹配一次之后，将停止向后继续查找。加入全局修饰符g，表示全局匹配。

所以上面的结果是：



2、全局修饰符g

g是global，表示全局的意思，它是正则表达式语法中的修饰符，**修饰符应该放到定界符(/)的后面**。

# 四、贪婪模式

在字符串 “aaabbbaaabbb” 中查找b

代码

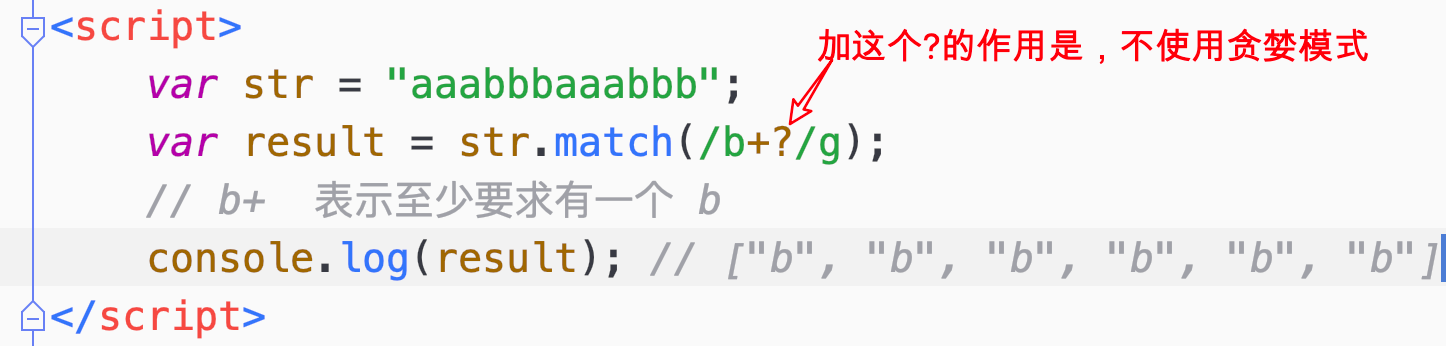


问题来了，我们要匹配至少一个b，那么当匹配到 b 之后，为什么没有停止，而是继续又匹配了两个b。换句话说，一个 b，也满足至少一个b这个条件呀，为什么结果不是 [“b”, “b”, “b”, “b”, “b”, “b”] ？

这就涉及到正则表达式默认的匹配模式了，正则表达式默认使用贪婪模式，理解为，它匹配到结果后，并不甘心停止，而是贪婪的继续向后匹配，直到不能匹配为止。

比如，上述代码，当匹配到一个b的时候，明显是符合条件的，但是并没有算做一次匹配，而是贪婪的继续向后匹配，直到发现下一个字母是a，才终止了本次匹配。

如果不想使用贪婪模式，则需要在正则表达式 **贪婪的位置** 加一个“?”，比如



小结：

* 正则表达式默认使用贪婪模式
* 如果不想使用贪婪模式，需要在正则表达式后加一个 “?”

# 五、正则语法

1、量词

n+ 匹配任何包含至少一个 n 的字符串。匹配一个或多个前导字符（**前导字符就是紧挨着+的前一个字符**）

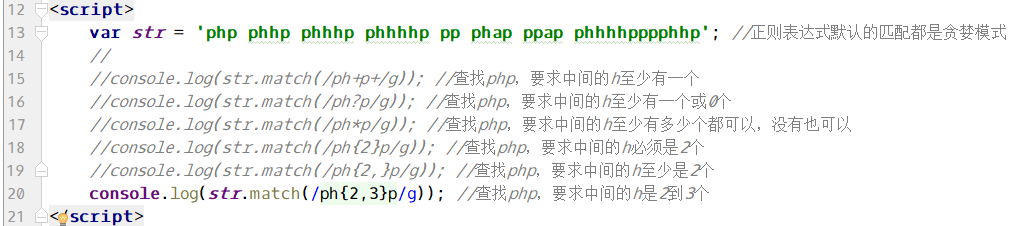
n\* 匹配任何包含零个或多个 n 的字符串。前导字符有没有都行

n? 匹配任何包含零个或一个 n 的字符串。

n{X} 匹配包含 X 个 n 的序列的字符串。

n{X,Y} 匹配包含 X 到 Y 个 n 的序列的字符串，包括X和Y个

n{X,} 匹配包含至少 X 个 n 的序列的字符串。



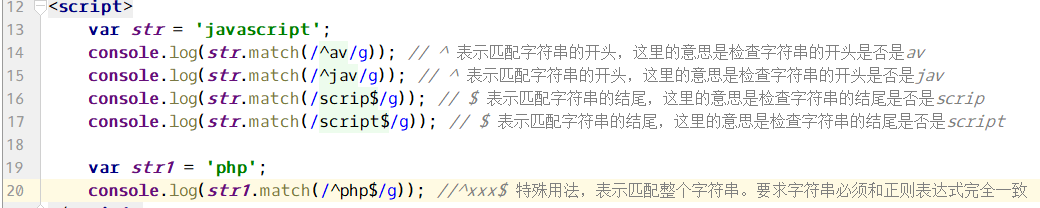
2、定界符

表示位置的符号

**n$** 匹配任何结尾为 n 的字符串。从末尾开始匹配

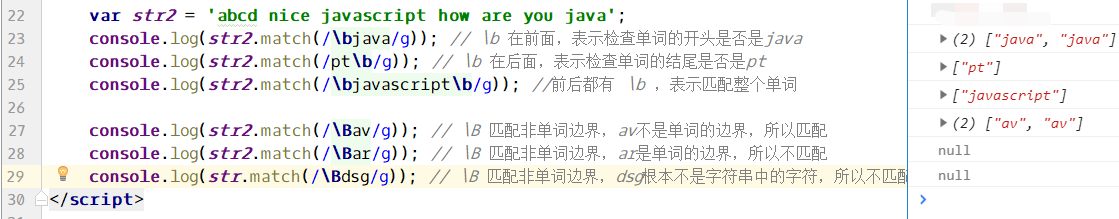
**^n** 匹配任何开头为 n 的字符串。（注意位置在前面）

**^n$** 表示只匹配字符串n。也就是匹配整个字符串。



**\b** 匹配单词边界。（边界字母个数没有限制；\b放前面表示匹配单词前面的边界，\b表示匹配单词后面的边界）

**\B** 匹配非单词边界。



3、元字符

**.** 表示单个字符，表示任意字符。除了换行和行结束符。

**.\*** 表示任意字符串（0个或多个任意字符，除了换行和行结束符）**.**是任意一个字符，\*表示任意个前导字符。二者组合表示任意字符串

**\w** 匹配任何数字、字母、下划线。一个\w表示**一个**字符。

**\W** 匹配任何非数字、非字母、非下划线。

**\d** 查找数字。

**\D** 查找非数字字符。

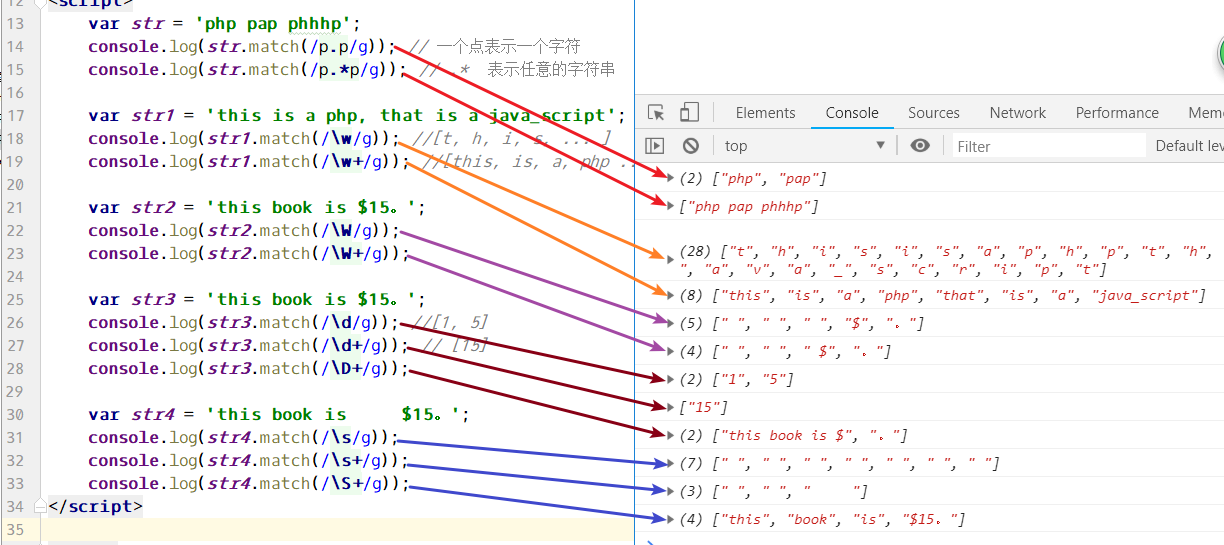
**\s** 查找空白字符。

**\S** 查找非空白字符。

**\n** 查找换行符。

**\r** 查找回车符。

**\t** 查找制表符。



4、方括号

方括号表示一个范围，也称为**字符簇**。

[abc] 查找方括号之内的任何字符。

[^abc] 查找任何不在方括号之间的字符。（方括号中的^表示取反）

[0-9] 查找任何从 0 至 9 的数字。表示一个字符

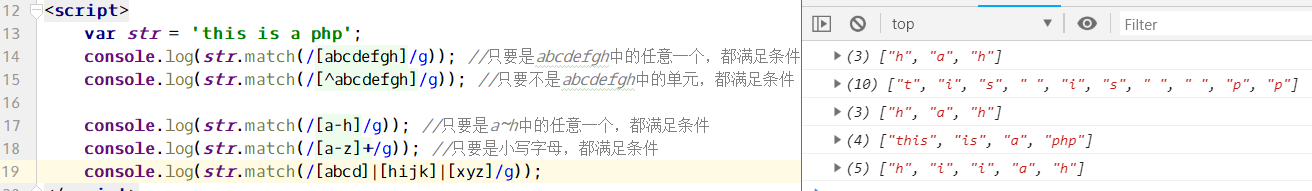
[a-z] 查找任何从小写 a 到小写 z 的字符。

[A-Z] 查找任何从大写 A 到大写 Z 的字符。

[A-z] 查找任何从大写 A 到小写 z 的字符。**包括下划线**。

[5-8] 查找5 <= 目标 <= 8的字符

**(**red|blue|green**)** 查找任何指定的选项。（竖线|表示或者）



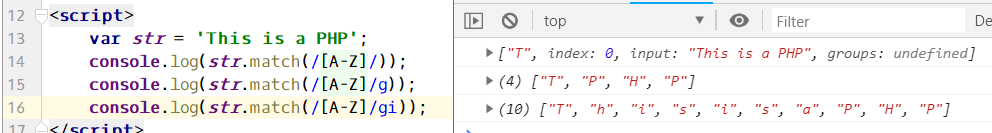
5、修饰符

直接量语法形式的正则表达式，要把修饰符放到两条斜线的后面，比如 “/正则表达式/修饰符”

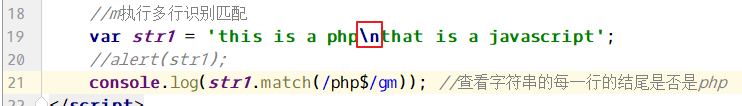
对象形式的正则表达式，修饰符是构造器的第二个参数，比如“var r = new RegExp(正则表达式, 修饰符);”

**一次可以使用多个修饰符**。

i 执行对大小写不敏感的匹配。实际上就是不区分大小写的匹配

g 执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。

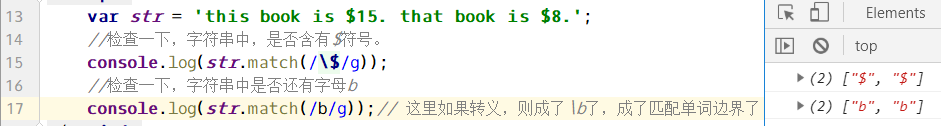
m 执行多行匹配。



6、转义字符\

如果匹配的字符串在正则中有特殊含义的都必须加转义字符。如[]$.\*?+|^{}()

但是不要乱加转义（bBwWdDsSnrt）。



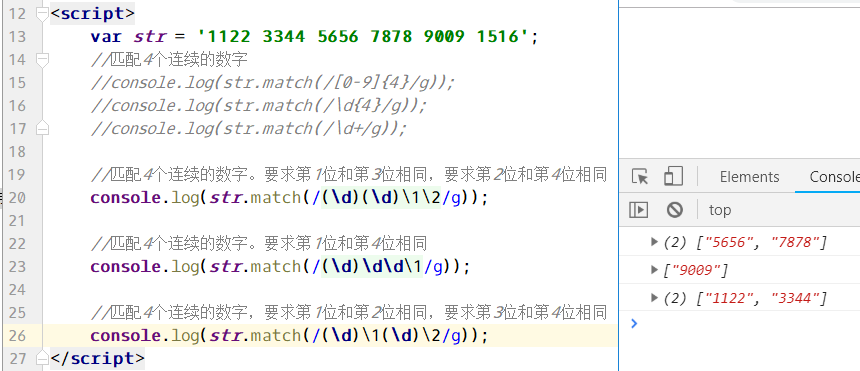
# 六、分组/捕获和反向引用

**捕获和反向引用的语法的解释**：

正则中出现的小括号，就叫捕获或者分组

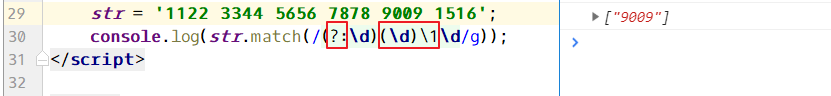
在正则语法中（在/…/内），在捕获的后面，用“\1”来引用前面的捕获。用\2表示第二个捕获的内容….

在正则语法外（如replace时），用“$1”来引用前面的捕获，用$2来引用第2个捕获的内容…。



**禁止引用**

**(?:正则) 这个小括号中的内容不能够被引用**



# 七、匹配中文（utf-8编码）

每个字符（中文、英文字母、数字、各种符号、拉丁文、韩文、日文等）都对应着一个Unicode编码。

查看Unicode编码，找到中文的部分，然后获取中文的Unicode编码的区间，就可以用正则匹配了。

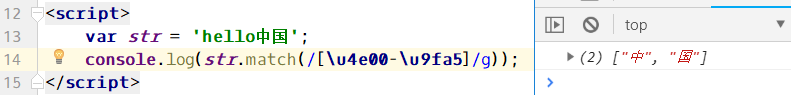
前面我们用[a-z]表示小写字母，[0-9]表示数字，这就是一个范围表示，如果有一个数x能够表示第一个中文，有一个数y能够表示最后一个中文，那么[x-y]就可以表示所有的中文了。

**中文的Unicode编码从4E00开始，到9FA5结束**。

**[4E00-9FA5]这个区间就能够表示中文**。

JS语法：**[\u4e00-\u9fa5]**

**完整的Unicode编码表**：http://blog.csdn.net/hherima/article/details/9045861

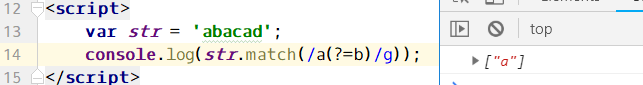


# 八、环视（断言/零宽断言/正向预测/负向预测）

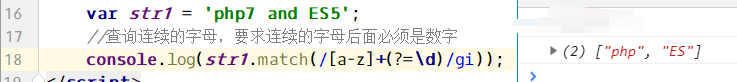
every(?=n) 匹配任何其后紧接指定字符串 n 的字符串。

有一个字符串是“abacad”，从里面查找a，什么样的a呢？后面必须紧接b的a。

正则语法是：/a(?=b)/g



延伸一下，案例中的a、b都是正则表达式，实际上换一个复杂一点的正则表达式也是可以的。



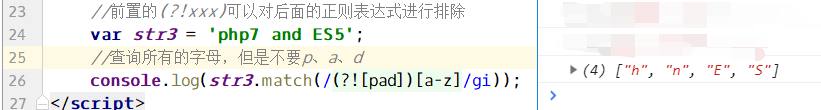
every(?!n) 匹配任何其后没有紧接指定字符串 n 的字符串。

有一个字符串是“abacad”，从里面查找a，什么样的a呢？后面**不能**紧接b的a。

正则语法是：/a(?!b)/g



另外，还会看到(?!B)[A-Z]这种写法，其实它是[A-Z]范围里，排除B的意思，前置的(?!B)只是对后面数据的一个限定，从而达到过滤匹配的效果。



# 九、**正则对象**的属性和方法

**正则对象的属性和方法，由正则表达式来调用，千万不要用字符串去调用**。

得到正则表达式的方式有下面三种：

* + var r = /正则表达式/; //直接量语法
  + var r = new RegExp(/正则表达式/, 'g'); //创建对象方法
  + var r = new RegExp('正则表达式', 'g'); //创建对象方法

1、exec方法和lastIndex属性

1112 3344 4455

* exec：

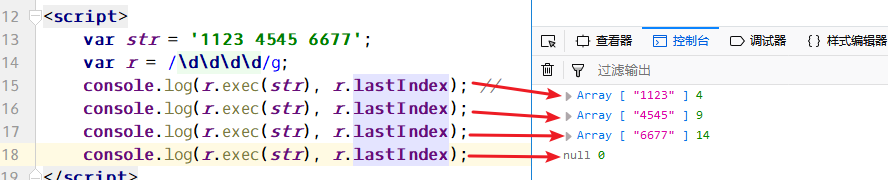
exec方法执行一个正则匹配，**只匹配一次**，匹配到结果就返回一个数组类型的结果，匹配不到就返回null。并将表示匹配的位置 置为下一个匹配的位置。

* lastIndex：

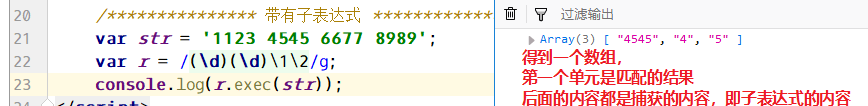
一个整数，标示开始下一次匹配的字符位置。没有更多匹配重置lastIndex为0.

**依次调用exec匹配下一个的例子**：

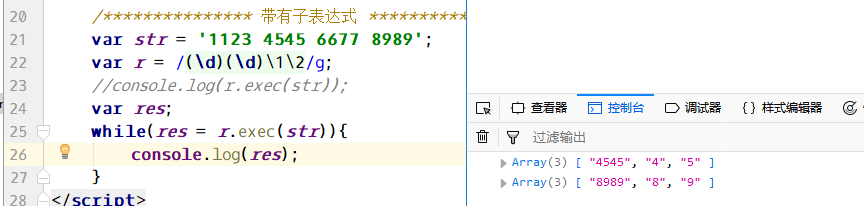
依次调用exec，会将匹配的位置不断的后移，直至结尾。



匹配带子表达式的例子，带子表达式并且依次调用的例子：

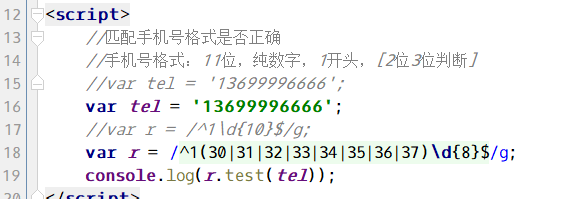


依次调用exec，循环的方式，匹配所有的结果：



2、test方法

test方法检测目标字符串和正则表达式是否匹配，如果匹配返回true，不匹配返回false。



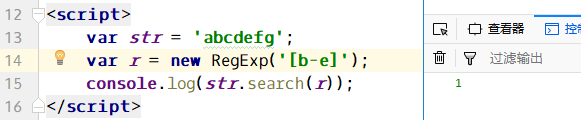
# 十、支持正则表达式的 **String 对象**的方法

**这些方法一定要用字符串去调用。正则表达式当做他们的参数**。

1、search()

在字符串中搜索符合正则表达式的结果。如果找到结果返回结果的位置，停止向后检索，也就是说忽略全局标识符g；如果没有匹配结果，返回-1.

下面的例子是在字符串中查找bcde中的任意一个字符，如果发现了，则返回它的位置。



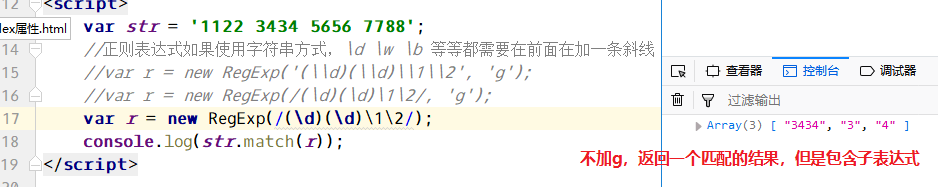
2、match()

在字符串中检索匹配正则表达式的子串；如果匹配，返回包含匹配结果的一个数组；不匹配返回null。

**不加全局g的情况**：

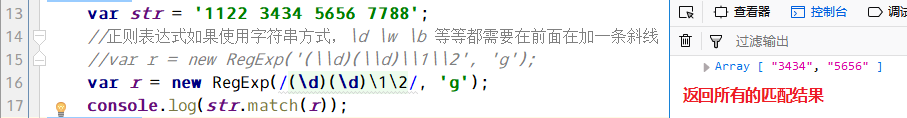
获取的结果只是第一个匹配的内容，匹配的内容中的第一个单元是匹配的结果，后面的单元是子表达式

带子表达式的情况：



**加全局的g的情况**：

获取的结果是所有匹配的内容，但是不包含子表达式。

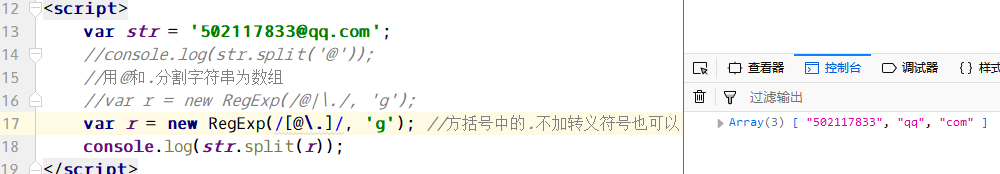


**多次调用和调用一次效果一样**。

3、split()

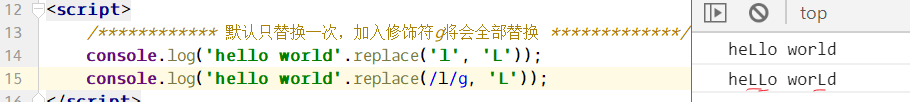
将字符串分割成数组：

特点是可以用正则表达式来当分隔符。



4、replace()替换

**默认只替换一次，加入修饰符g将会全部替换**。

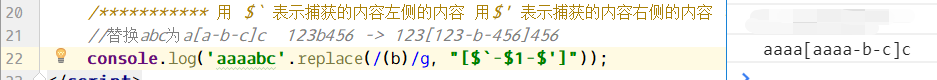


**替换的时候，使用$1表示第一个子表达式，用$2表示第二个子表达式，以此类推**。



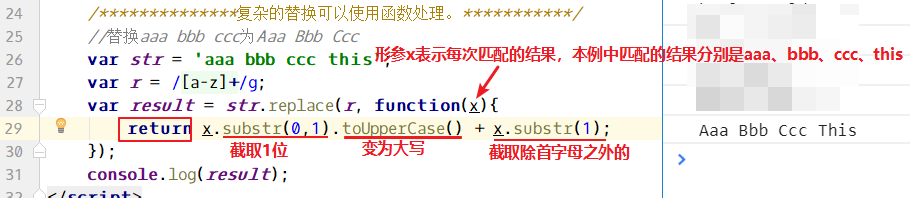
**用 $`表示捕获的内容左侧的内容 用$'表示捕获的内容右侧的内容**

替换abc为a[a-b-c]c ghk 🡪 g[g-h-k]k



**复杂的替换可以使用函数处理**。

替换aaa bbb ccc为Aaa Bbb Ccc



# 十一、案例

1、匹配手机号格式是否正确

/^1[3-9][0-9]{9}$/

2、匹配邮箱格式是否正确

@ 前面，只能写数字、字母、下划线，长度1,

3、解决结巴程序

把“今今今天晚晚晚晚晚晚上吃吃吃吃吃吃鸡”字符串换成单字的形式，即“今天晚上吃鸡”；

4、字符串整理

现有一个字符串， [b]JavaScript[/b] is better than [b]java[/b] ，其实本意是想使用加粗标签，但是误把尖括号写成了中括号，我们最后需要替换成 <b>JavaScript</b> is better than <b>java</b>

5、用户名验证

注册的时候，要求用户名：长度4~10位，不能是纯数字，不能是纯字母，不能有特殊符号，不能以数字打头。

# 十二、案例答案

1、略

2、略

3、解决结巴程序



4、字符串整理



5、用户名验证

