JS正则表达式

# 一，初识正则表达式

## ，什么是正则表达式

正则表达式，又称规则表达式。（英语：Regular Expression，在代码中常简写为regex、regexp或RE），计算机科学的一个概念。正则表通常被用来检索、替换那些符合某个模式(规则)的文本。

**理解**：

1，正则表达式是一个字符串。它表示了一种规则。

2，正则表达式有它自己特定的语法。

3，它有两个主要的功能：**验证**，**匹配**。

## ，正则表达式的作用

**验证：**

比如说当你注册一个网站时，有的网站让提供手机号，邮箱，性名，身份证号.....等等， 这些操作就是验证。

**匹配：**

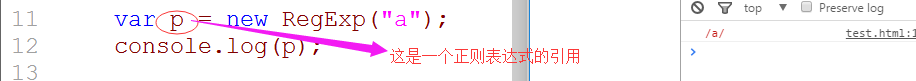
把一个字符串中，我们想要的一部分找出来。如：”今天是2017年4月22号...”

## ，如何创建正则表达式

有两种方式去创建一个正则表达式：

* **1，使用new**
* **2，字面量**

**1，使用new**



**2，字面量**

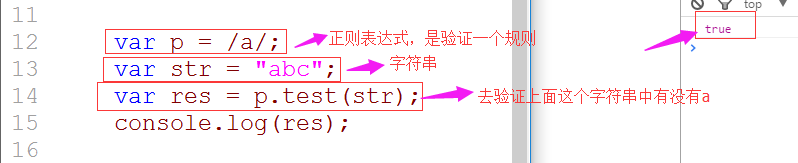
## ，正则表达式的使用

正则表达式是一个对象，这个对象中常见的有三个方法：compile(), **exec(), test();** 其中complile用的不多，test()它是用来验证， exec()是用来匹配的。我们先学习test()。

**test的用法：**

正则表达式.test(字符串); 如果验证成功就返回true, 否则返回false。

**代码如下：**



# 二，正则表达式深入

## （1），正则表达式的组成

前面说过一个正则表达式： var p = /a/;

**三个组成部分：**

* 原子： 必选，不能少。 最基本的组成单位。
* 元字符： 可选， 用来修饰原子。
* 模式修正符：可选，共有三个 i , g, m

## （2），模式修正符

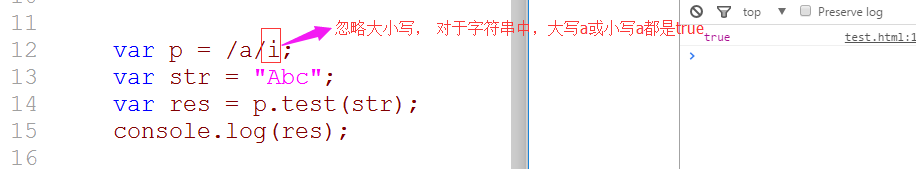
模式修正符是写在正则表达式最后的/ 的后面。

**I：忽略大小写**

**G：全局匹配**

M：多行匹配

**代码如下：**



对于G， Ｍ后面讲。

## （3），原子

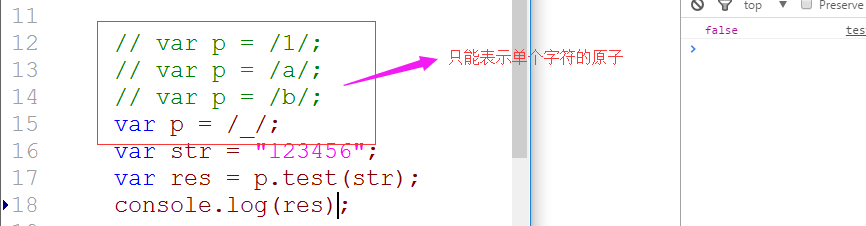
最基本的组成单位，不能少。

**按照原子的表现能力，可以为分两类：**

* １，只能表示一个字符的原子
* ２，可以表示一类字符的原子

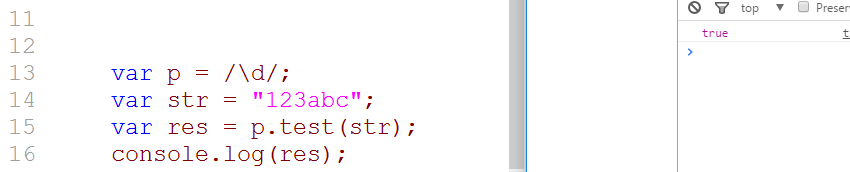
### １，只能表示一个字符的原子

例如：大小写字母，数据，下划线，符号．．．．

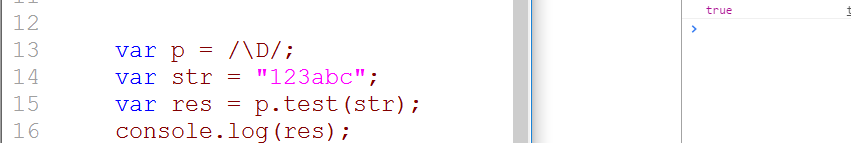


### ２，可以表示一类字符的原子

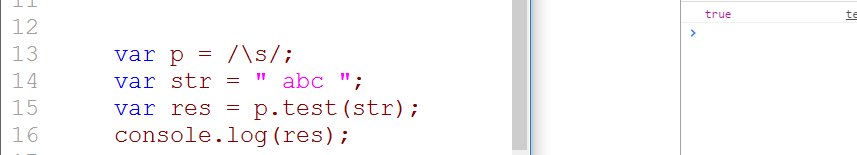
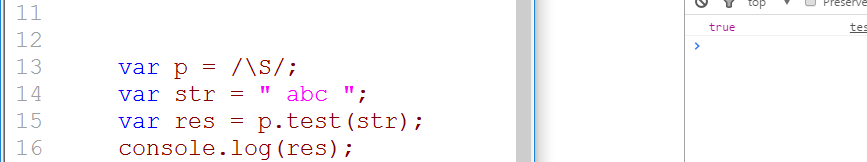
**ａ， \d 表示十进制的数字[0-9], 只要包含数字就是true**



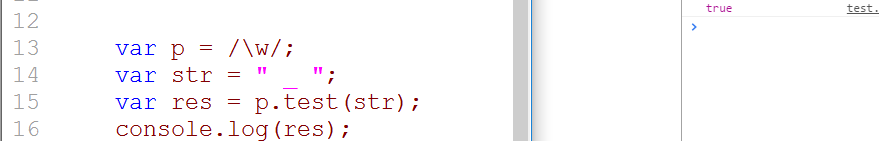
**b， \D 表示只要包含非数字就是true**



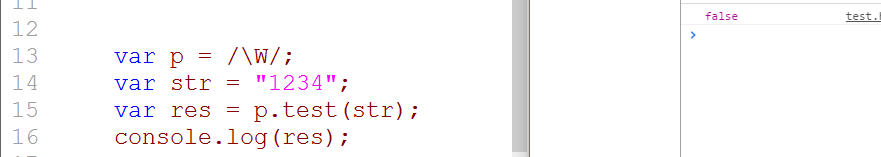
**c， \s 表示空白字符（空格， tab, 回车键）只要有空白就是**

**true****d， \S 表示非空白字符（空格， tab, 回车键） 只要有非空白就是 true**

**e， \w 表示一个字 0~9 a~z A~Z \_**



**f， \W 表示除了 0~9 a~z A~Z \_ 以外的字符**



**注意的问题：**

* 1，对于test()来说，只是检测有还是没有。 不管字符串中的个数。
* 2，对于test()来说，不管字符出现在字符串中什么位置。
* 3，对于test()来说，正则中的字符串顺序要与字符串中子串的顺序一致。
* 4，使用模式修正符，可以不考虑大小写问题。

## （4），元字符

是用来修饰原子，原子有两类。

### 1，元字符的分类

元字符有很多，不好记，我们把它们分类：

**分类如下：**

* 开始元字符和结束元字符
* 数量相关的元字符
* 逻辑相关的元字符
* 转义符
* 分组组合

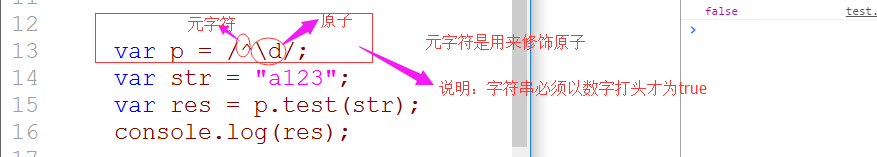
### 2，开始元字符 ^ 与 结束元字符 $

**开始元字符： ^**

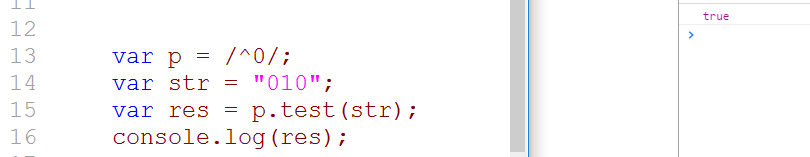
出现位置：一般是在正则表达式的最前面出现

作用：表示必须要以它后面紧贴的那个原子作为开始

代码如下：



小练习：电话都有区号，区号一般是以0打头

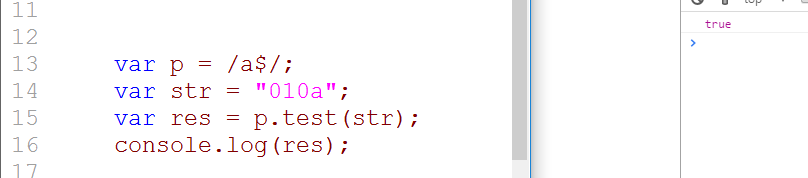


**结束元字符： $**

出现位置：一般是在正则表达式的最后面出现

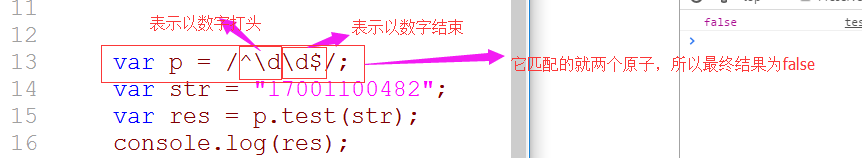
作用：表示必须要以它前面紧贴的那个原子作为结束

代码如下：

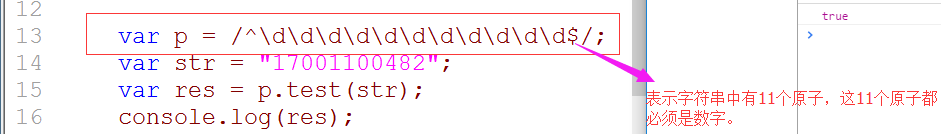


小练习：

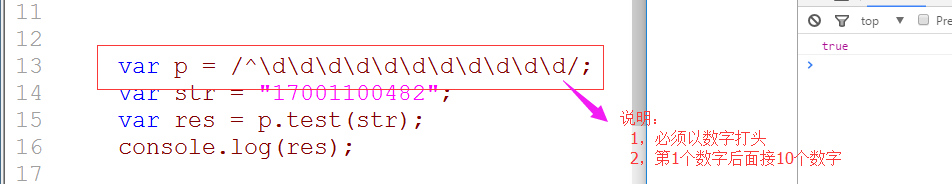
对于开始元字符与结束元字符一起使用时，一定要注意：



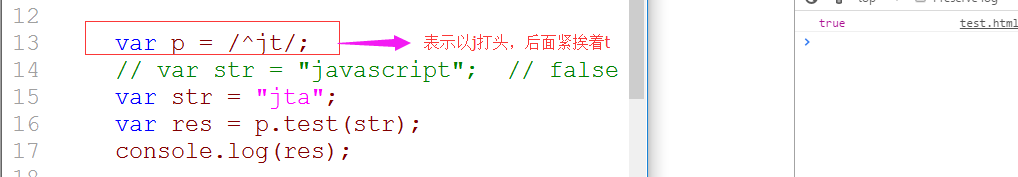
如何修改让它变成true:



再说明一下：



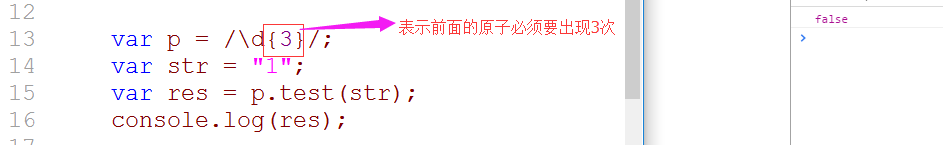
再练习一下：



### 3，数量相关的元字符 {} \* + ？

**a, {m} : 前面的原子必须要出现m次。**

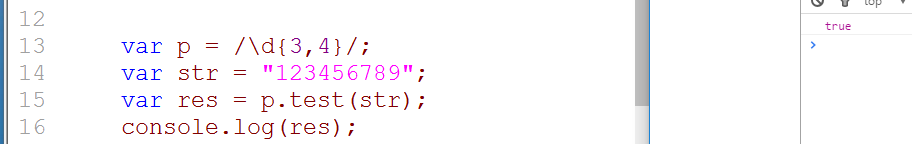
代码如下：



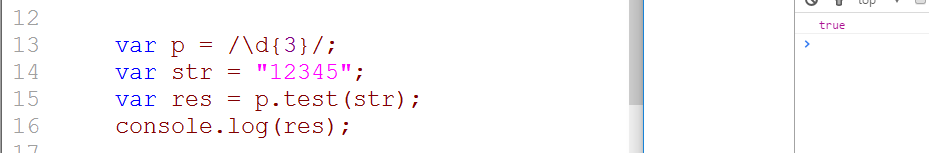
**b, {m,n} : 前面的原子必须至少要出现m次，最多出现n次。**



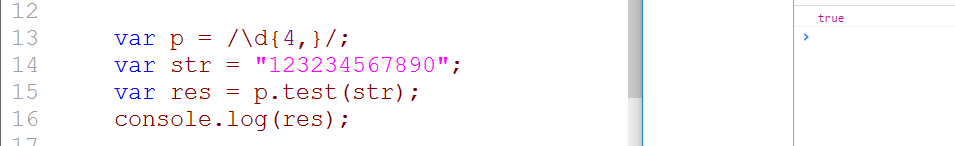
问题：{3，5} ==》 {3} 或 {4} 或 {5}



为什么？

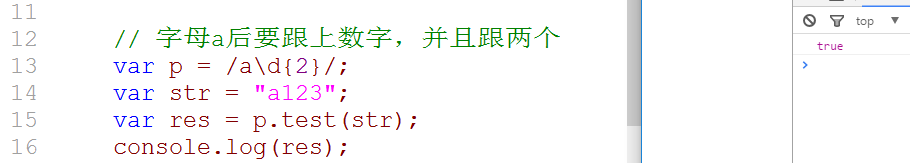


**c, {m,} : 前面的原子必须至少要出现m次，最多不限**

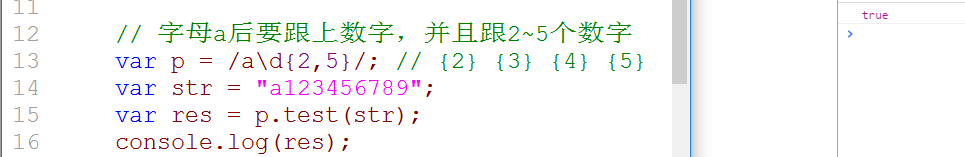


**练习：**

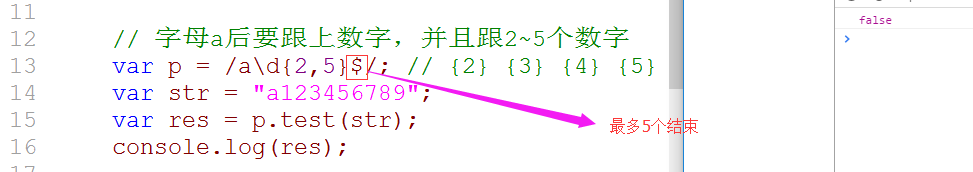
练习一：

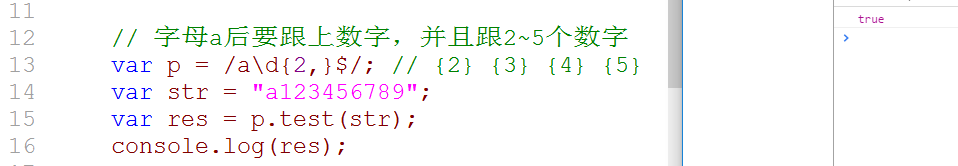


练习二：



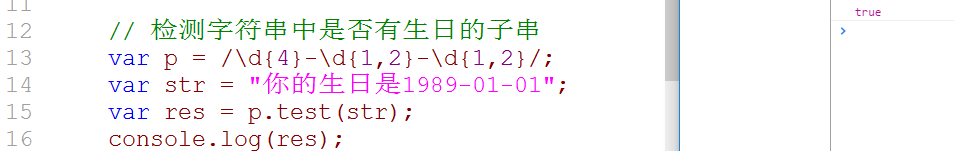
练习三：



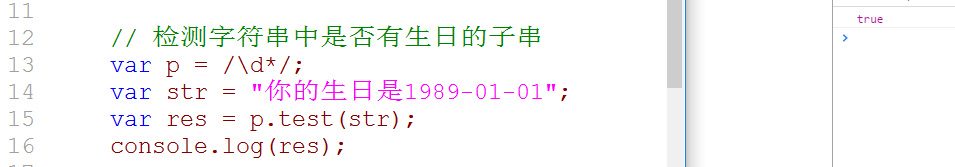
练习四： 

**d, \* : 前面的原子可以出现0次或很多次 相当于{0,}**

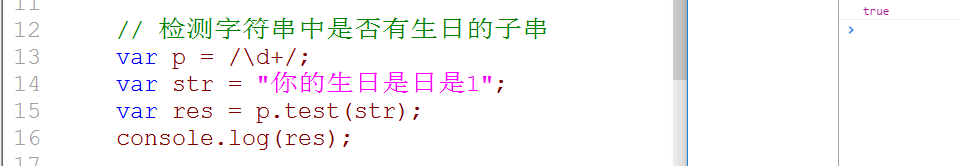
复习{}的用法，如下：



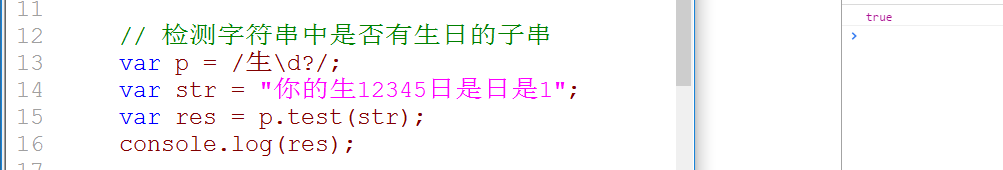
\*的用法：



**e, + : 前面的原子至少出现1次 相当于{1,}**



**f, ? : 前面的原子至少出现0次 或者是1次 相当于{0，1}**



**复习：**

**1，什么是正则表达式？**

a， 正则表达式就是带着某种规则的字符串，只是它是一个特殊的字符串。

b， 正则表达式有它自己的语法

c， 正则表达式的功能是用来进行验证与匹配和替换

**2，正则表达式有什么作用？**

功能是用来进行验证与匹配

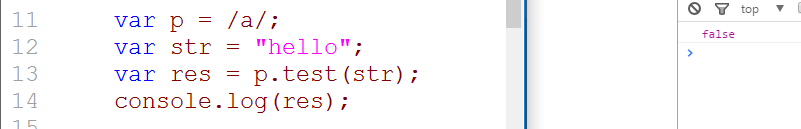
**3，创建正则表达式的两种方式？**

**new**

**字面量**



**4，正则表达式的test方法？**

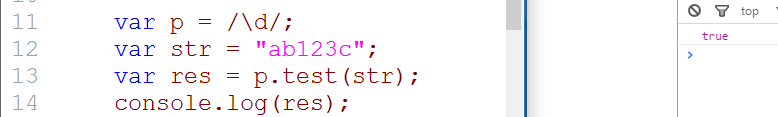


**5，正则表达式的三个组成部分？**

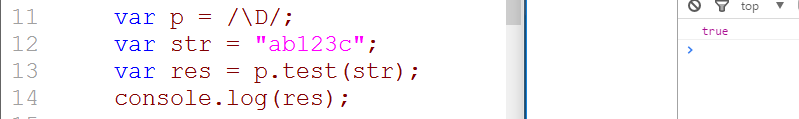
* 原子 必须 是正则最基本的组成单位
* 元字符 可选 用来修饰原子
* 模式修正符 可选 i g m

**6，表示一类的原子？**

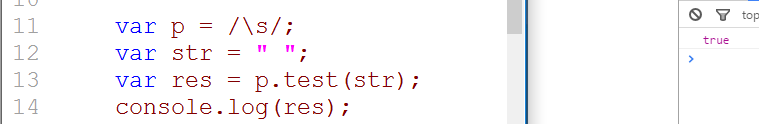
\d 表示十进制的数字 [0-9]



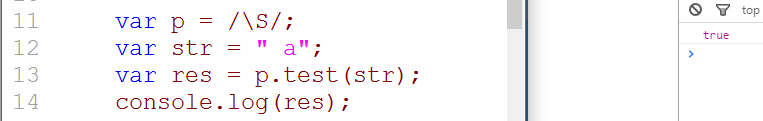
\D 表示非数字



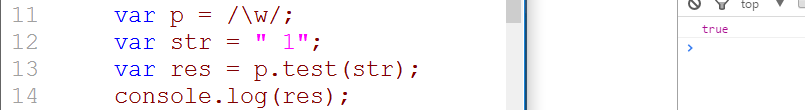
\s 表示空白字符



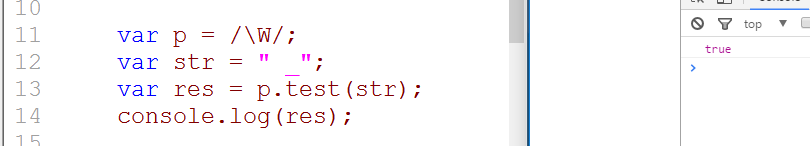
\S 表示非空白字符



\w 表示 [0-9a-zA-Z\_]



\W 表示除了[0-9a-zA-Z\_]之外

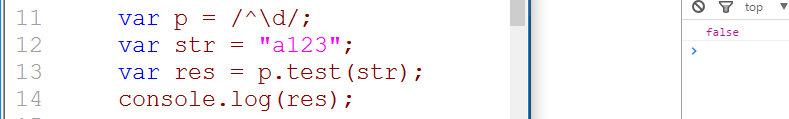


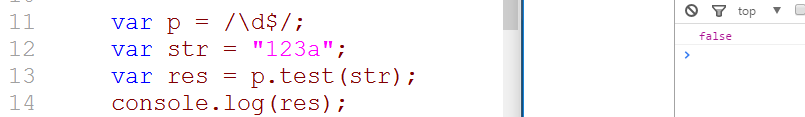
**7，^ $**

它们是元字符，元字符有很多，我们把它们进行了分类：

* 表示开始与结束的
* 表示数量
* 表示逻辑
* 转义的
* 分组组合

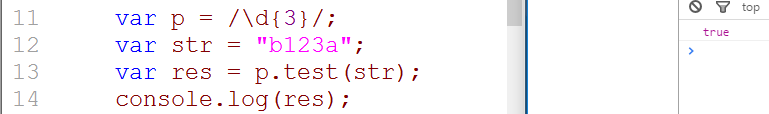
**^ 表示开始，以它后面紧挨着的原子作为开始， 通常出现在正则表达式的最前面**

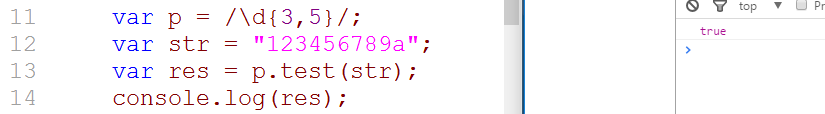


**$ 表示结束 以它前面紧挨着的原子作为结束，通常出现在正则表达的最后面**

8，{}， \*， +， ？

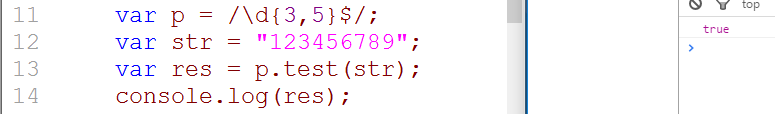
{}分三种：

**{m} 表示前面的原子必须要出现m次**

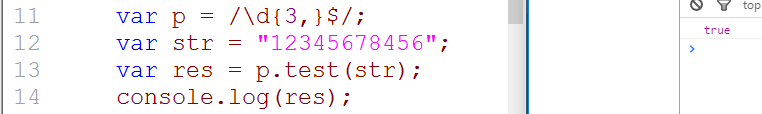
**{m,n} 表示前面的原子至少要出现m次， 最多出现n次** 

上面的{3,5} 它就是{3}或{4}或{5}

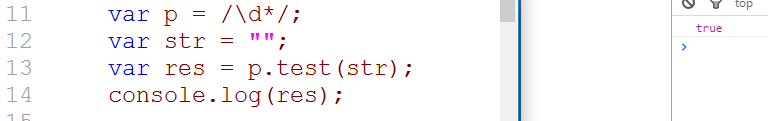
下面写法：以{3}或{4}或{5}作为结束



**{m,} 表示前面的原子至少要出现m次， 最多不限**



**\* 表示前面的原子可以出现0或很多次，类似于{0,}**



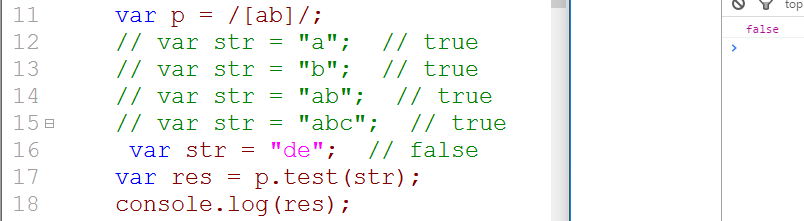
**? 表示前面的原子可以出现0或1次，类似于{0,1}**

**+ 表示前面的原子至少要出现1次**

### 4，逻辑相关的元字符 [ ] |

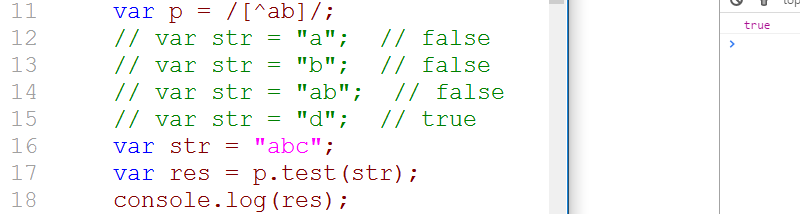
**[] ： 表示多个原子，只要有一个在目标字符串中出现就行。**

代码如下：



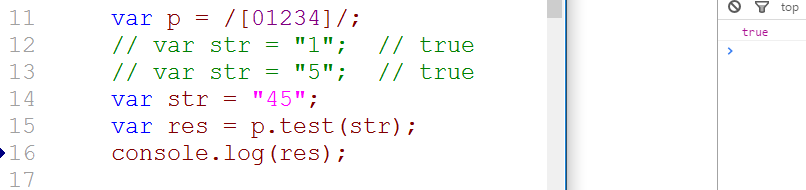
在［］中还可以使用两个额外的符号：　＾　-

* ^ 表示取反
* - 表示区间

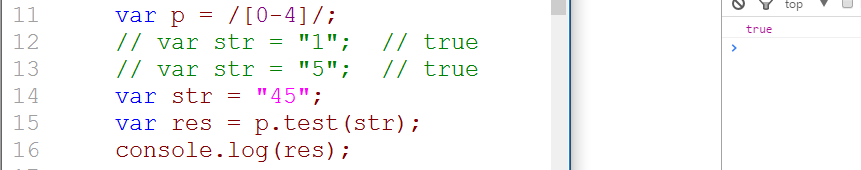
**^ 表示取反**

**- 表示区间**

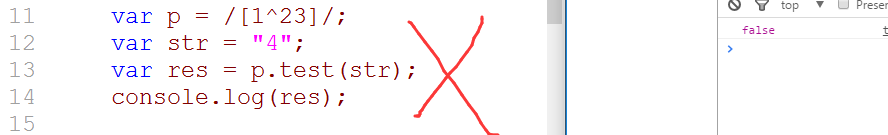
**对于下面的写法，我们可以使用- 表示区间**



使用-后的写法：

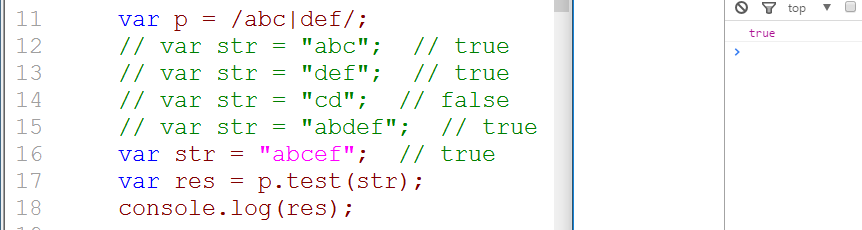


下面的写法，是什么意思 ？下面的写法是错误的，我们一定不能将^写在原子中间，只能写在开头。



**| 或**

**代码如下：**



说明：对于 /abc|def/ 表示或者abc, 或者是def，或者是abcef, 或者abdef

**注意下面的区别：**

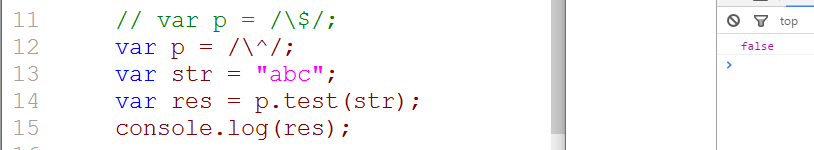
/cat|dog/ 表示：cat 或 dog 或catog 或 cadog

/[catdog]/ 表示：对于里面的6个字符出一个就行

/catdog/ 表示：六个字符都必须出现，顺序也不变

### 5，转义符 \

我们前面学习了很多的元字符 $ ^ {} [] ? \* | 等等，当你想要这些元字符当作一个普通的原子使用时，这个时候你需要用到转义符 /

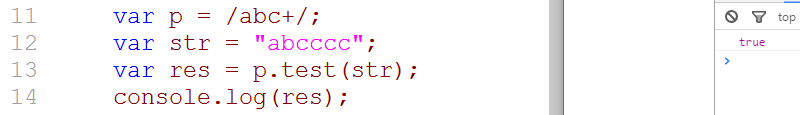


### 6，分组组合 （）

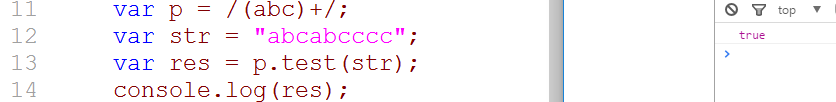
它可以把多个原子组合成一个大原子。

**比如：**

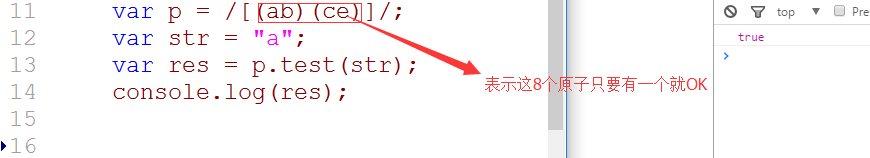
/abc+/ 表示ab后面加1个以上的c, 代码如下：



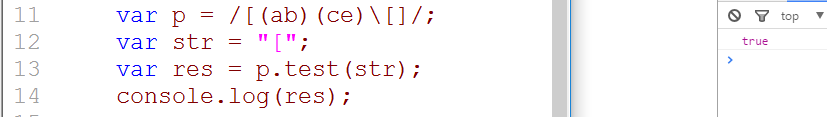
/（abc）+/ 表示abc这个整体要出现一次以上：



练习：



看下面的练习： \[ 表示转义



**总结：**

1， $ 如果它出在正则表达式的最后，表示以它紧挨着前面的原子结尾。 如果想看一个目标字符串中没有没$本身。 要使用转义后的：\$

2， ^ 如果它出在正则表达式的最前面，表示以它紧挨着后面的原子作为开始。看一个目标字符串中没有没^本身。 要使用转义后的：\^

3，+ 表示前面的原子要连续的出现1次或多次，看一个目标字符串中没有没+本身。 要使用转义后的：\+

4， \* 表示前面的原子要连续的出现0次或多次，看一个目标字符串中没有没\*本身。 要使用转义后的：\\*

5， ？示前面的原子要的出现0次或1次，看一个目标字符串中没有没？本身。 要使用转义后的：\？

6，[] 表示一个原子表的开始与结束， 看一个目标字符串中没有没[, ]本身。 要使用转义后的：\[ \]

7，() 表示分组， ()里面的内容我们可以叫它为子表达式，[]里面的内容我们只可以叫它为原子表。()表示子表达式的开始与结束。看一个目标字符串中没有没（, ）本身。 要使用转义后的：\（ \）

8，| 指两项之间的选择。看一个目标字符串中没有没|本身。 要使用转义后的：\|

9，{} 表示数量相关的一个元字符， 看一个目标字符串中没有没{}本身。 要使用转义后的：\{ \}

10， 看一个目标字符串中没有没\本身。 要使用转义后的：\\

### 7，优先级

# 三，典型应用之验证

## ，编写一个正则表达式的流程

**分三步：**

1，分析：把正确的与错误的列举出来

2，提炼：把规则提出来

3，书写：用原子与元字符来组合，表达第2步中的规则。

## ，手机号码验证

**第一步：分析**

正确：17001100482， 13312345678......

错误：01234321445, a123445342....

**第二步：提炼**

1，必须是数字

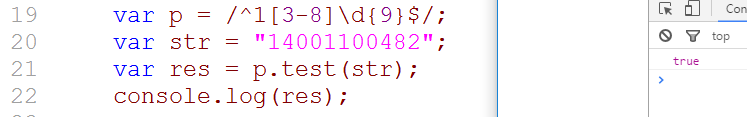
2，必须是11位

3，必须是1打头

4，第二位必须是3，4，5，6，7，8

5，必须是数字结尾

**第三步：书写：**



**说明：有时候，你看到别人写的正则表达式，和你的不一样，原因是你们的提炼的规则不一样。**

## ，身份证号验证

**第一步：分析**

正确：410883198904156553 .....

错误：a345678956789 1234 234567893456789034567894567

**第二步：提炼**

1，必须是18位

2，以数字开头

3，前17位必须是数字

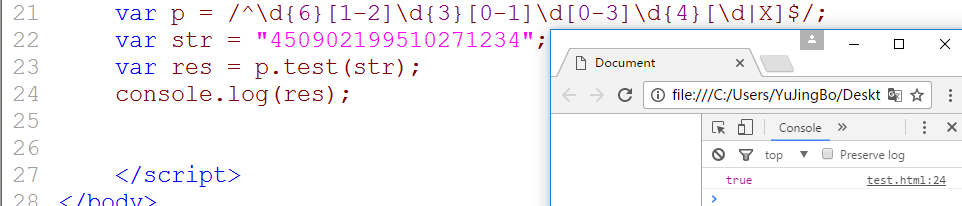
4，最后一位可能是数字也可能是X

5，第7位必须是1或2

6，第11位必须是0或1

7，第13位必须是0或1，或2，或3

**第三步：书写**



## ，正整数验证

**第一步分析：**

正确： 123， 456....

错误： 1.2 0123 -123 a123

**第二步提炼：**

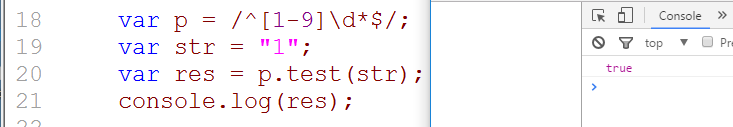
1，全部是数字

2，第1位不能是0， 不能是-， 说白了第一位就是[1-9]

3，第二位开始是数字，个数是0个及以上 \*

4，数字开始，数字结束

**第三步书写：**



## ，电子邮箱验证

**第一步分析：**

正确： [2345@qq.com](mailto:2345@qq.com) [abc@sina.cn](mailto:abc@sina.cn) [6789@163.com](mailto:6789@163.com) \_[45678@qq.net](mailto:45678@qq.net)

错误： 234qq.com [sdf@.com](mailto:sdf@.com) 2345.com

**第二步提炼：**

1，在@前面必须有内容

2，只能有一个@

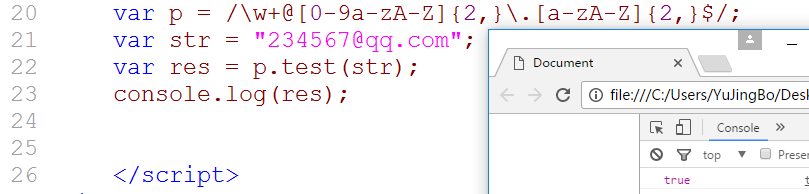
3，有一个. 并且这个.要出现在@的后面

4，所有字符都必须是0-1， a-z, A-Z, \_ ---> \w

5，@和. 之间必须要有至少要有2个以上的数字，或字母（不分大小写）

6，.后面有2个以上的大小字母

**第三步书写：**



## ，十六进制颜色

**第一步分析：**

正确： #123456 #123 #abcdef #ABC

错误： #qwerta #1234567 #12 #abcd1234

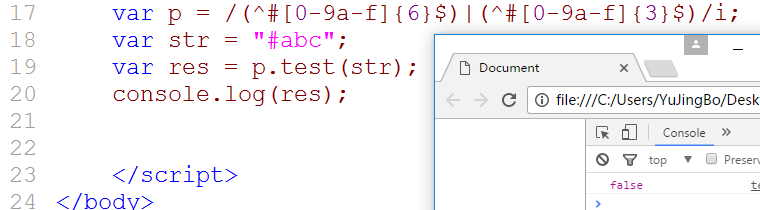
**第二步提炼：**

1，位数只能有3位或6位 |

2，元素由[0-9a-fA-F]

3，以#打头

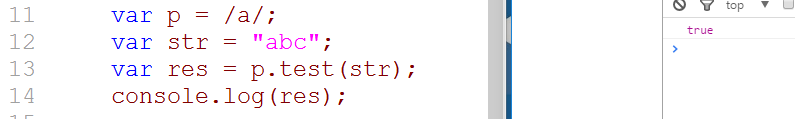
**第三步书写：**



# 四，典型应用之匹配

## exec()方法

test()是来验证的。代码如下：



**exec()格式：**

返回值 = 正则表达式.exec(目标字符串);

**返回值：**

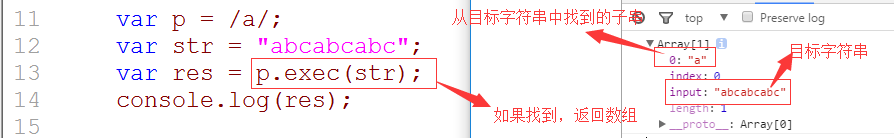
如果找到，返回数组

如果找不到，返回null

**功能：**

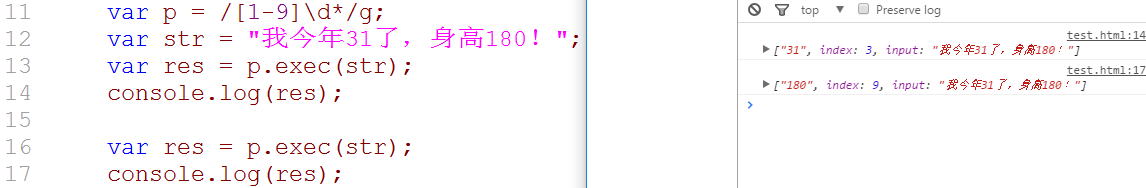
在目标字符串中找出符合正则表达式规则的子串。

**代码如下：**

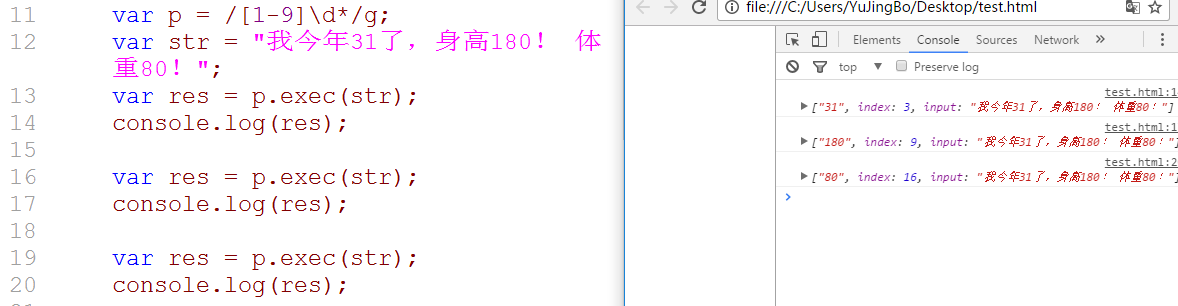


说明，如果按照上面的写法，只会找到第一个。

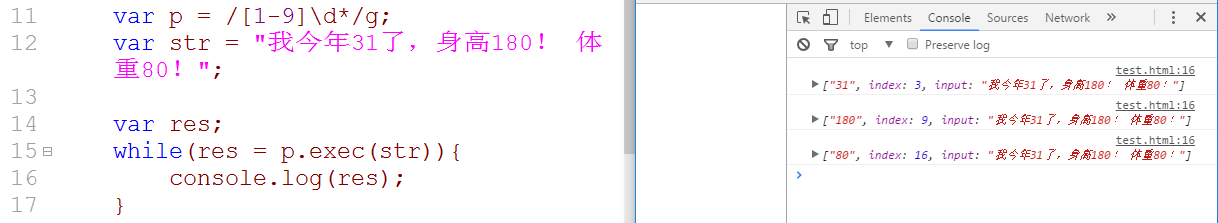
**如果找到所有呢？ 模式修正符g 全局匹配！**



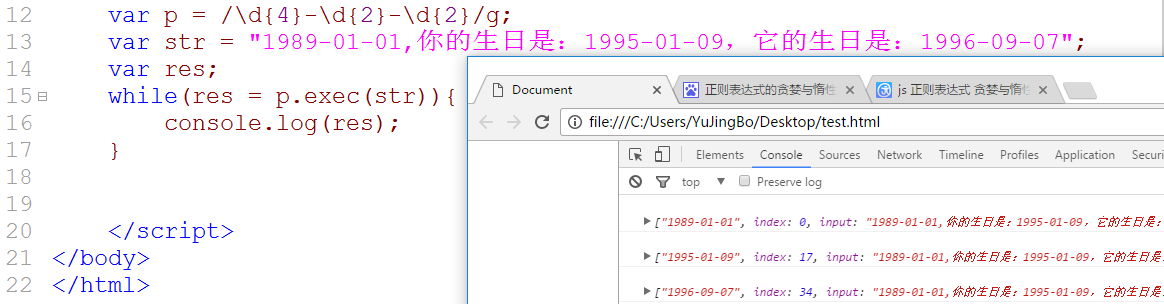
**如果目标字符串中有多个符号要求的子串，我们需要写多份结果：**



**如果我们不知道，目标字符串中有多少个符合要求？ 怎么做？**

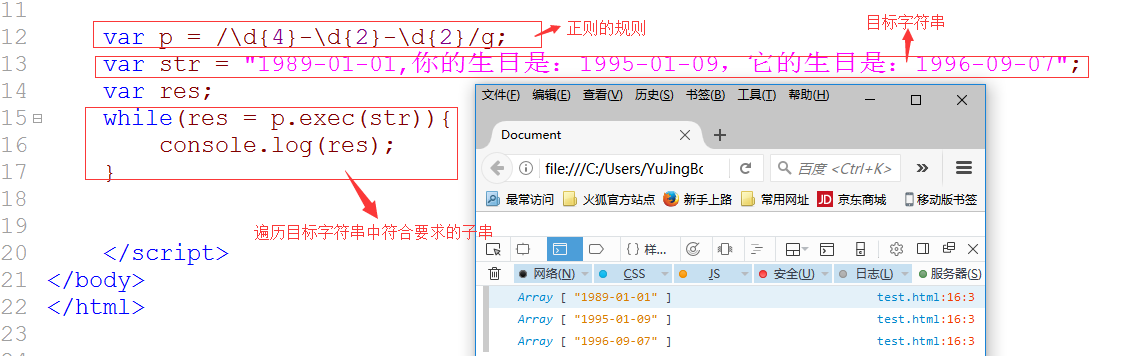


**小练习：**

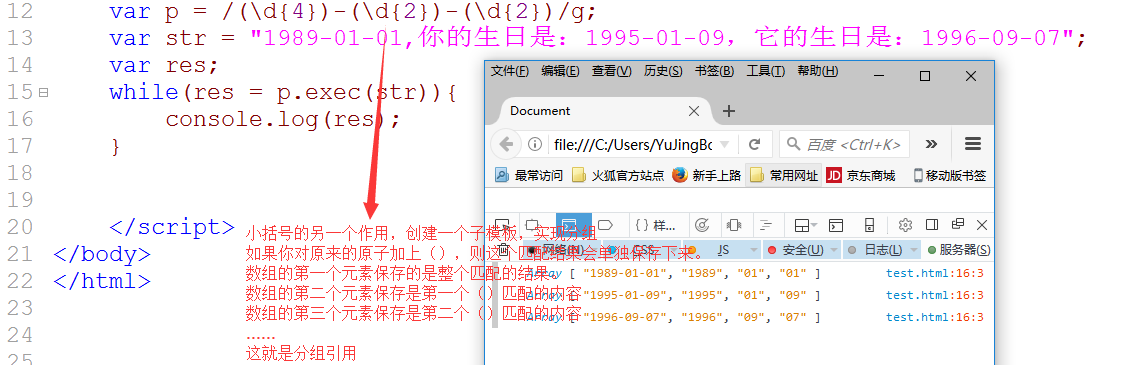


## 分组引用 （）

没有分组的情况如下：

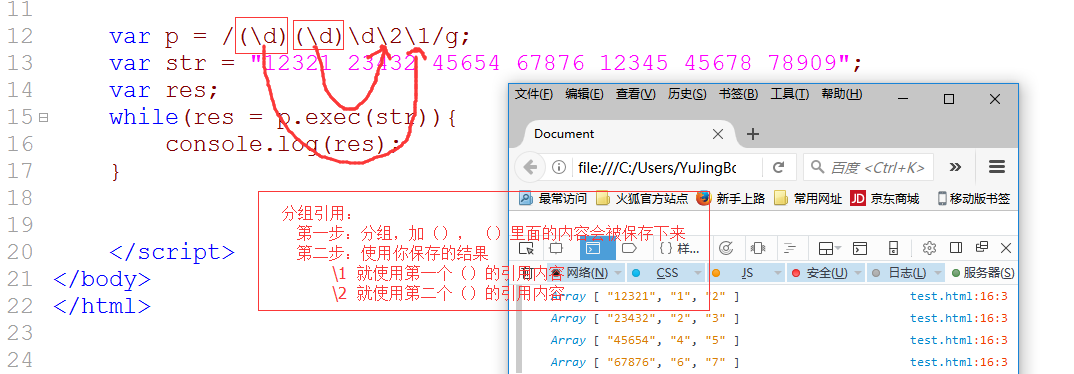


**使用分组：**



## 向后引用

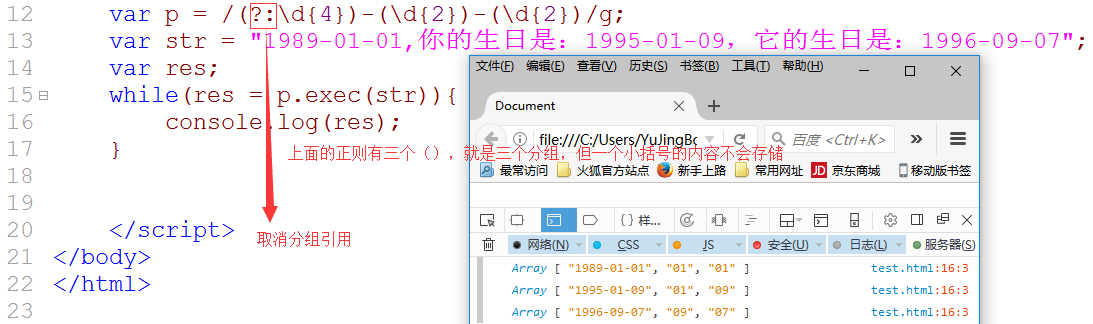
**比如： 12321 23432 67876**



## 取消分组引用

分组引用需要额外的空间来存储。当我们只是需要分组，不需要引用时，可以取消引用。

做法：在()里面前加上？：

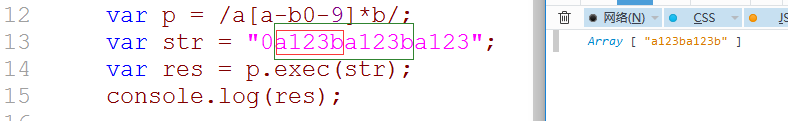


# 五，贪婪模式与惰性模式

## 贪婪模式

贪婪模式：匹配尽可能多的字符

当匹配到字符有多有少时，首先匹配多的。

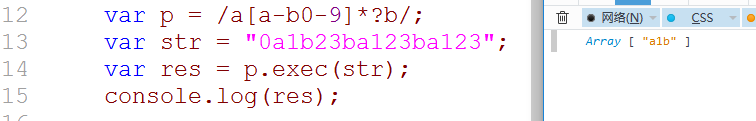


因为这里面有个\*， 所以才会存在贪婪模式。思考？除了\* ，哪些还有贪婪模式：

**\* + ？ {}**

## 惰性模式

尽可能少的去匹配，默认是贪婪模式，如何变成惰性模式？**需要在重复内容的后面加上？**

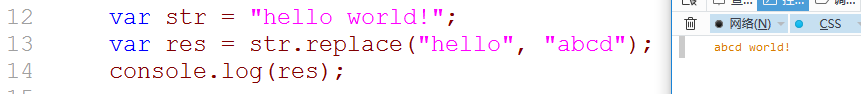


# 六，正则表达式综合运用

## 复习replace方法

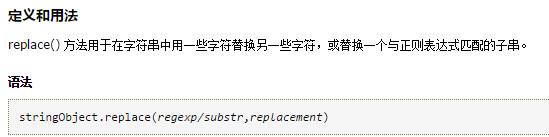
学习string时，有些方法可以配合正则使用。 replace()方法就是string中的一个方法， replace这个方法的作用：替换。

**代码如下：**

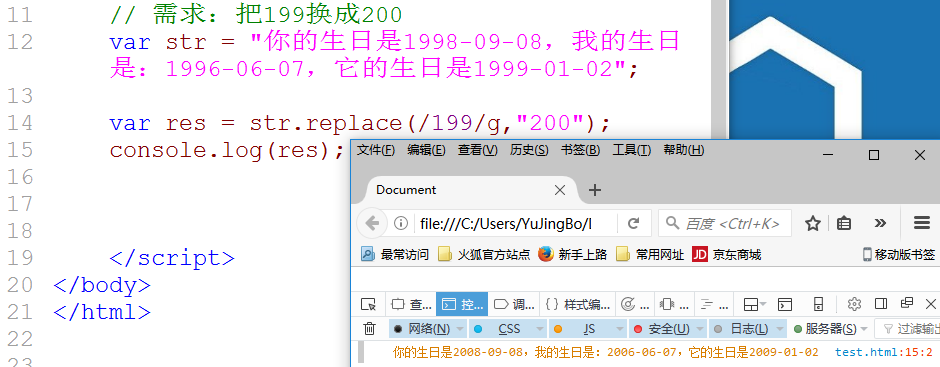


## 练习一

replace方法第一个参数可以使用正则：

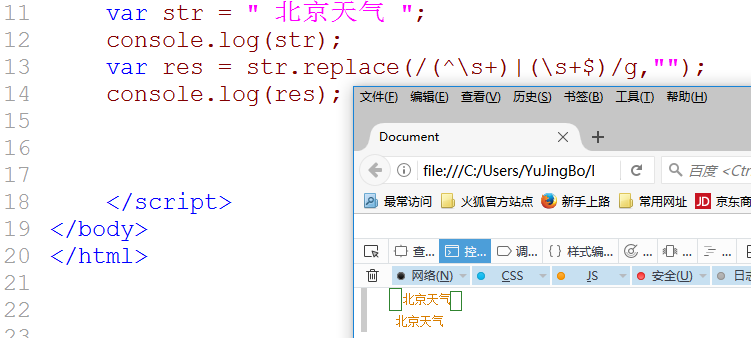


练习代码如下：



## 练习二

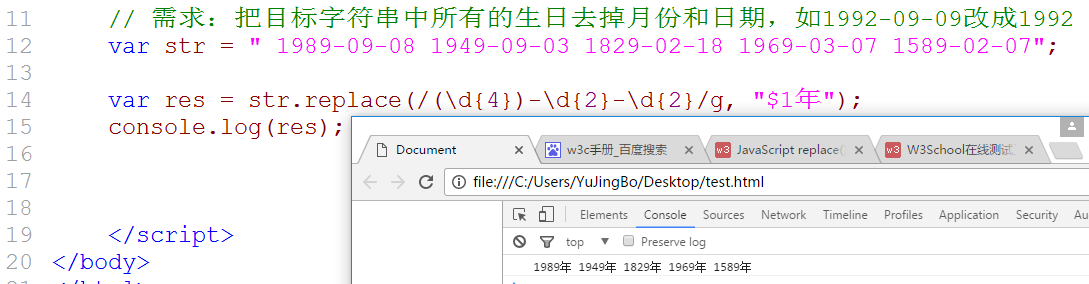
去除字符串前后的空格，代码如下：



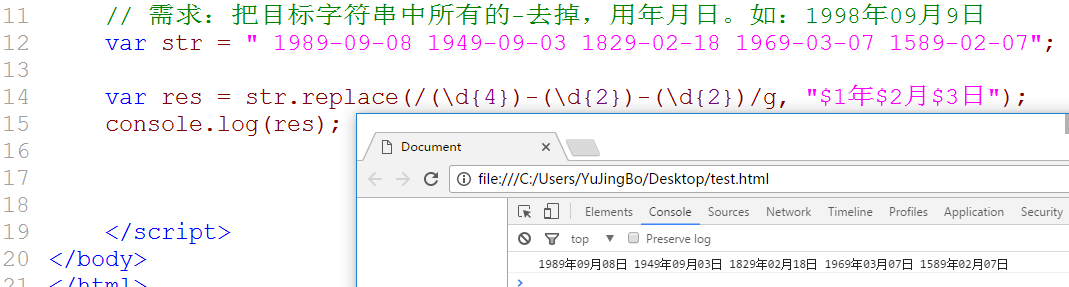
## （4）练习三

**$1, $2, $3**

把匹配到的内容，用分组引用的部分来代替。在正则中第一个（）,对应$1, 第二个（）,就对应$2.....

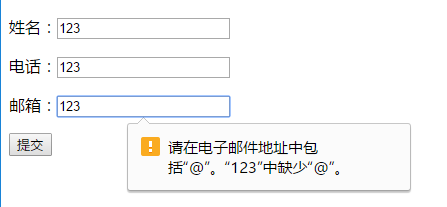


加强1：



# 七，正则表达式与html5表单

当我们使用type=”email”时，如果我们输入的格式不对，它会给我们一个提示：



如果说我想输入一个生日，能否也有这样类似的提示：

