# JavaScript基础入门

第10天课堂笔记

# 目录

JavaScript基础入门 1

目录 2

一、上次课知识复习 3

1.1 作用域 3

1.2 闭包 3

1.3 数组 3

1. 字符串的属性和方法 5

2.1 属性 5

2.2 方法 5

1. 正则表达式 7

3.1 概述 8

3.2 正则用到的方法 8

3.2.1 字符串方法 8

3.2.2 正则的方法 9

1. 正则表达式的术语和操作符 10

4.1 精确匹配 10

4.2 预定义特殊字符 10

4.3 字符集 10

4.4 修饰符 11

4.5 边界 12

4.6 预定义类 12

4.7 量词 13

4.8 分组 15

4.9 或操作符 15

4.10 分组的反向引用 15

4.11 中文 16

# 一、上次课知识复习

## 1.1 作用域

JS没有块级作用域，只有函数能关住作用域。

在作用域里面定义的变量叫做局部变量。在函数外面是不能被访问到的。

定义在全局就是我们的全局变量。在任何地方都能被访问。

变量没有定义直接赋值，计算机会自动在全局。一定要写var关键字。

函数的形式参数也是函数内部的局部变量，在外面不能被访问。

作用域链：如果一个函数内部调用一个变量，从本层函数开始一层一层往外找，直到找到一个匹配的变量，就直接使用，一直找不到，报错，变量未定义。

遮蔽现象：如果有同名的变量分布在不同的层面，我们使用距离本层最近的。

函数也有作用域，它定义在哪儿，就在哪个作用域里使用。出了这个作用域，函数不能被访问到。

## 1.2 闭包

每一个函数都是一个闭包。它天生定义的时候就能记住内部的语句，还有当时所处的作用域，封闭成了一个密闭的环境。每次重新使用这个函数的时候，都能找到原来的作用域。

想观察闭包的现象，用一种特殊的方法。将内层的函数作为外层函数的返回值。将外层函数执行后赋值给一个新的变量，用这个变量调用内层函数，我们就能够观察闭包。

|  |
| --- |
| 1. function waiceng(){ 2. var a = 1; 3. runction neiceng(){ 4. Console.log(a); 5. } 6. return neiceng; 7. } 8. var i = waiceng(); 9. i(); |

每一个新的闭包，都会访问它定义时的那个作用域。

## 1.3 数组

数组是一个有序的数据集合。存了一组数据。

数据可以是任意类型，我们习惯一个数组存放同一类型的数据。

数组用它的数组字面量的方法定义。

|  |
| --- |
| 1. var arr = [12,”12”,null]; |

我们可以通过下标（索引值）来访问数组里面的某一项，下标从0开始的。

也可以通过下标来进行修改值。

|  |
| --- |
| 1. arr[1]; //访问的是第二项，索引值是1 2. arr[1] = 23; //更改值 |

数组是object类型，是一个对象。有属性和方法。

数组有一个长度属性。使用数组名.length，得到数组里面数据的长度。数据的个数。

我们数组最大的索引值是数组的长度减1.

|  |
| --- |
| 1. 最大的索引值 = arr.length - 1 |

我们可以对数组进行遍历，可以对每一个数组的数据进行一些操作。

|  |
| --- |
| 1. for(var i = 0; i <= arr.length - 1;i++){ 2. 取每一项用arr[i]; 3. 可以进行其他的操作； 4. } |

方法：

首尾操作：

push() 在数组的尾部加入一个或多个元素，返回数组新的长度。

pop() 删除数组的最后一项元素，返回这个删除的元素。

unshift() 在数组的首位添加一个或多个元素，返回的是新的长度。

shift() 删除数组首位的元素，返回这个删除的元素。

|  |
| --- |
| 1. var arr = [1,3,4,6,8]; 2. arr.push(arr.shift()); //[3,4,6,8,1] |

合并和拆分

concat() 将两个数组合并并且返回一个新的数组。原来的两个数组不变。

slice() 截取数组的一部分，返回一个新的数组。参数都是索引值。

|  |
| --- |
| 1. Var arr = [1,3,4,5,6,7]; 2. Var a = arr.slice(2,4); //开始位置的元素截取，结束位置的元素不包括。 |

如果只有一个参数，是开始位置，从这个位置开始截取到数组的结尾。

参数可以是负值。

|  |
| --- |
| 1. Var b = arr.slice(-1,-4); //取不出值，负值不是索引值，直接从后面数位置 |

删除、添加、替换：

splice() 找到指定位置的元素删除，替换成新的元素。

参数只有两个：删除，第一个参数是起始位置的索引值，第二个参数是删除的长度。

添加：第二个参数值是0，后面的参数要有其他的新元素。

替换：把指定的删除，在原位置添加新的元素。

倒序和排序：

倒序：reverse() 将数组里面的数据的顺序完全颠倒。

排序：sort() 会按照某种顺序来给我们排序。

默认的情况：用字符编码顺序，来排序。

有对比函数参数：

|  |
| --- |
| 1. var a = [1,35,67,12,79,10,30]; 2. a.sort(function(a,b){ 3. if(a > b){ 4. return -1; 5. }else if(a < b){ 6. return 1; 7. }else{ 8. return 0; 9. } 10. }); |

数组转字符串：

join() 有一个参数，如果传递就用它做连字符，如果没有默认用逗号链接。

# 字符串的属性和方法

## 2.1 属性

Length 返回字符串的长度

|  |
| --- |
| 1. var a = "我们想要学习更多 的Word啊！" 2. console.log(a.length); |



字符串的长度包括：汉字、英文字母、空格、和特殊符号，每一个都占了一个长度。

## 2.2 方法

charAt() 返回在指定位置的字符。

Character 字符的意思

这个参数就是字符串的下标。索引值也是从0开始。

|  |
| --- |
| 1. console.log(a.charAt(5)); |



Concat() 连接字符串。

可以连接两个或多个字符串，并且返回一个新的字符串，原来的字符串不受影响。

|  |
| --- |
| 1. var b = a.concat("得自己总结",",或者从别人那里得到"); 2. console.log(a); 3. console.log(b); |



indexOf() 方法可返回某个指定的字符串值在字符串中首次出现的位置。

从字符串前面开始找，找到第一个符合的字符串，输出它的索引值。

|  |
| --- |
| 1. console.log(b.indexOf("得")); |



lastIndexOf() 方法可返回一个指定的字符串值最后出现的位置，在一个字符串中的指定位置从后向前搜索。

注意：输出的是最后一个出现的位置的索引值，不是从后面数第几个。

|  |
| --- |
| 1. console.log(b.lastIndexOf("得")); //返回索引值 |



slice() 方法可提取字符串的某个部分，并以新的字符串返回被提取的部分。

Start：截取开始的位置，包括。

End：结束的索引值位置，不包括这个位置。

|  |
| --- |
| 1. var c = a.slice(3,7);//包含起始位置，不包含结束位置 2. console.log(c); |



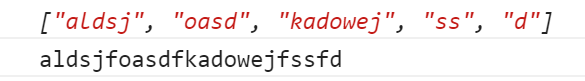
也可以只传一个参数：截取到最后。

可以传递负的参数。

split() 方法用于把一个字符串分割成字符串数组。

参数：就是我们的分隔符。

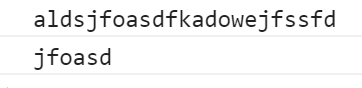
|  |
| --- |
| 1. var a = "aldsjfoasdfkadowejfssfd"; 2. var arr = a.split("f"); //必须加引号 3. var b = arr.join("f"); 4. console.log(arr); 5. console.log(b); |



substr() 方法可在字符串中抽取从 start 下标开始的指定数目的字符。

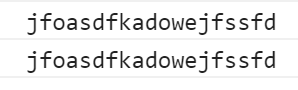
截取从指定位置开始的指定长度的字符串片段。

|  |
| --- |
| 1. var b = a.substr(4,6);//从下标为4的开始，截取6个字符 2. console.log(a); 3. console.log(b); |



也可以只输入一个参数：从起始位置截取到最后。它可以等价于slice方法只传一个数值。

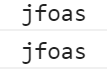
|  |
| --- |
| 1. var b = a.substr(4);//从下标为4的开始，截取6个字符 2. var c = a.slice(4); |



第一个参数可以传负值。

substring() 方法用于提取字符串中介于两个指定下标之间的字符。

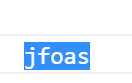
|  |
| --- |
| 1. var d = a.substring(4,9); |



从起始位置开始截取到结束位置，不包括结束位置。

substring与slice和substr不同：可以颠倒参数的顺序。

|  |
| --- |
| 1. var c = a.slice(9,4); //起始位置不能大于结束位置 2. var d = a.substring(9,4); //提前将两个数值调换，从比较小的值开始取，取到9的前面 |



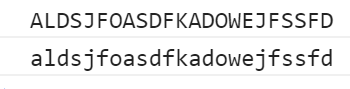
不同点：substring不能传负值。

转换大小写：

toLowerCase() 方法用于把字符串转换为小写。

toUpperCase() 方法用于把字符串转换为大写。

|  |
| --- |
| 1. var u = a.toUpperCase(); 2. console.log(u); 3. var l = u.toLowerCase(); 4. console.log(l); |



练习：把字符串“i love kaola very much”每个单词的首字母大写。

|  |
| --- |
| 1. //首字母大写 2. var a = "i love kaola very much"; 3. //找到所有的单词，转成数组，每一个数据就是一个单词 4. var arr = a.split(" "); //["i", "love", "kaola", "very", "much"] 5. //console.log(arr); 6. //对每一个单词进行操作，遍历数组 7. for(var i = 0 ; i <= arr.length - 1; i++){ 8. //首字母大写 9. //字符串找首位的字符 10. arr[i] = arr[i].charAt(0).toUpperCase() + arr[i].slice(1); 11. } 12. //console.log(arr); 13. //数组转字符串 14. var b = arr.join(" "); 15. console.log(b); |



# 正则表达式

正则表达式是被用来匹配字符串中的字符组合的模式，常用来做表单验证。在JavaScript中，正则表达式也是对象，是一种引用类型。

简单的感受一下

|  |
| --- |
| 1. //用户输入一个电话号码 2. //000-88888888 3. //输入 4. var a = prompt("请输入8位电话号码"); 5. var reg = /^\d{3}-\d{8}$/; 6. if(reg.test(a)){ 7. alert("输入正确"); 8. }else{ 9. alert("输入不正确"); 10. } |

## 3.1 概述

正则表达式（regular expression），简写成RegExp。是一种简单匹配字符串的模式。

可以通过正则表达式对一些字符串和某些符合特定语句规范的字符串进行控制。

正则的定义方法，最简单的方法用正则字面量来定义。

正则字面量：//就是我们正则的限定符，内部书写我们正则术语。类似于字符串的引号。

|  |
| --- |
| 1. var reg = /\s+/; //\s表示空格类的字符 2. var reg = /abc/ |

a/b/c都是我们正则的一个单独的术语。匹配的时候，必须全部都满足，abc必须都存在，并且排列顺序必须与正则规定的一样。

## 3.2 正则用到的方法

一种是字符串方法，一种是正则方法。

### 3.2.1 字符串方法

split() 根据匹配字符串切割父字符串

match() 使用正则表达式与字符串相比较，返回一个包含匹配结果的数组。

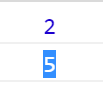
search() 对正则表达式或指定字符串进行搜索，返回第一个出现的匹配项的下标。

replace() 用正则表达式和字符串直接比较，然后用新的子串来替换被匹配的子串。

|  |
| --- |
| 1. //split方法，用空白区域切割字符串 2. var a = "aa bbb c dd eeeeee"; 3. var arr = a.split(/\s+/); 4. console.log(arr); |



|  |
| --- |
| 1. //search方法，检索子字符串在父字符串中第一次出现的位置 2. var a = "abcdefg"; 3. //console.log(a.search("cd")); 4. console.log(a.search(/cd/)); 5. var b = "aaaa o o bbb a o o aa"; 6. //console.log(b.search("o o")); //没法匹配到合适的值返回-1 7. console.log(b.search(/o\s+o/)); |



|  |
| --- |
| 1. //match() 使用正则表达式与字符串相比较，返回一个包含匹配结果的数组。 2. var a = "asdfjsidddasddkdfw"; 3. //var arr = a.match("d"); 4. var arr = a.match(/d+/g); //如果不加g，只匹配到第一个就结束了 5. console.log(arr); |



|  |
| --- |
| 1. var a = "aaaa o o bbb o o aaa"; 2. var arr = a.match(/o\s+o/g); 3. console.log(arr); |



将'www.baidu.com'替换成'www.163.com'

|  |
| --- |
| 1. var a = "www.baidu.com"; 2. var b = a.replace(/baidu/,"163"); //返回一个新的字符串 3. console.log(b); |

删除所有的空格

|  |
| --- |
| 1. var a = "aa b c d e f "; 2. var b = a.replace(/\s+/g,""); //匹配全局，不然只匹配第一个 3. console.log(b); |



### 3.2.2 正则的方法

exec() 在目标字符串中执行一次正则匹配操作。

test() 测试当前正则是否能匹配目标字符串。

字符串的方法：字符串.方法();

正则的方法：正则表达式.正则方法();

练习：在'aaaabccccbacabc'中查找'abc'字符串

Exec方法：只能执行一次匹配。

|  |
| --- |
| 1. var str = "aaaabccccbacabc"; 2. //var arr = str.match(/abc/); 3. var arr = /abc/.exec(str); //正则表达式后面调用正则方法 4. console.log(arr); |



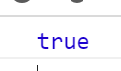
|  |
| --- |
| 1. console.log(arr.index); |

可以直接对数组进行一个index属性的调用，返回的是匹配的字符串在源字符串中的索引值。

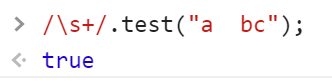
Test方法：返回值是true或false。

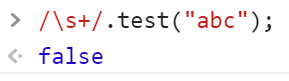
练习：判断'aaddccddabcddeeddfff'是否包含'abc'字符串

|  |
| --- |
| 1. var str = "aaddccddabcddeeddfff"; 2. console.log(/abc/.test(str)); |



练习：检测'a bc'和'abc'是否包含空格





# 正则表达式的术语和操作符

由一些普通字符和一些特殊字符（又叫元字符--metacharacters）组成。普通字符包括大小写的字母和数字，而元字符则具有特殊的含义。

特殊字符：

( ) [ ] { } \ ^ $ | ? \* + .

如果你想在字符里面匹配这些特殊的字符，用到一个转义字符”\”.

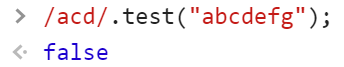
\( \) \[ \.

## 4.1 精确匹配

如果正则表达式要匹配的值不是特殊字符或者是操作符，那么他就需要进行精确匹配，这些字符必须出现在正则表达式里面。

/abc/进行匹配的时候，必须将“a”、“b”、“c”这三个术语直接写在正则表达式里面。它匹配的字符串必须包含这三个字符，而且顺序必须是abc，三个字母必须是紧挨的。

|  |
| --- |
| 1. /acd/.test(“abcdefg”); |



## 4.2 预定义特殊字符

\t 制表符 tab键打出来的空白字符

\n 回车符 enter键打出来空白字符

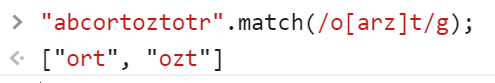
\r 换行

\f 翻页

\b 空格 空格键打出来的空白字符

## 4.3 字符集

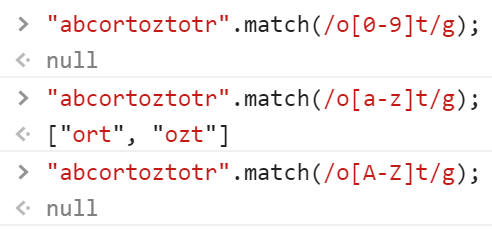
我们可以用一类字符来匹配一个字符，只要这一类里面包含了这个字符就能够匹配成功。这个类，我们用[]包裹起来。里面罗列了所有的可能性，只要满足一种可能性就能够正确匹配。

简单类:正则的一个字符对应一个字符，我们可以用[]把它们括起来，让[]这个整体对应一个字符。

范围类：有时候它的可能性特别多，在一个范围之内，我们可以用一个-表示范围。在范围内的都能够进行匹配。

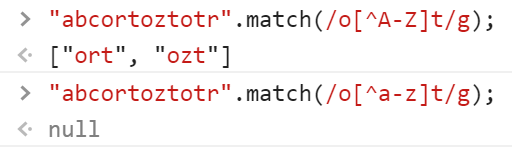
一般有：[0-9]、[a-z]、[A-Z]。

id[0-9]——id0、id5，包含了所有的0到9的数字的可能性，0和5都在这个范围内，能够进行正确匹配。



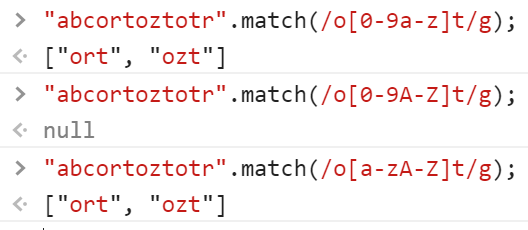
负向类:[]里面的前面加个元字符（^）进行取反，表示匹配不能为括号里面的字符。

可以对简单类取反，也可以对范围类取反，对组合类也可以取反。



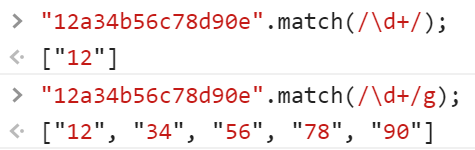
组合类：允许匹配不同类型的。

取从0-9或者从a-z都可以：[0-9a-zA-Z];

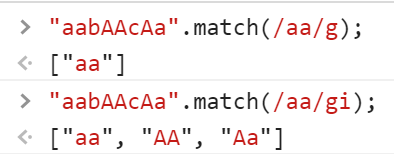


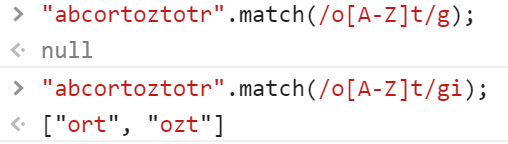
## 4.4 修饰符

g 修饰符用于执行全局匹配（查找所有匹配而非在找到第一个匹配后停止）。



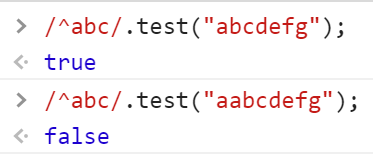
i 修饰符用于执行对大小写不敏感的匹配。



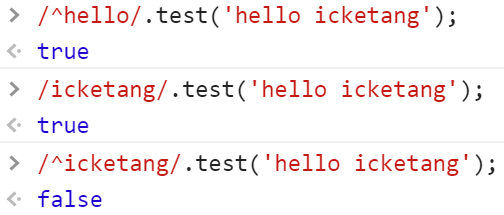


## 4.5 边界

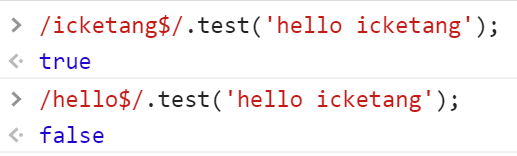
^ 表示开头 注意不能写在[的后面。



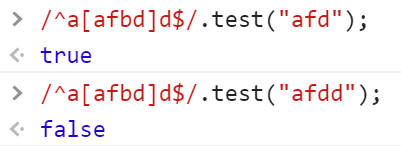
界定我们的字符串必须以正则里面的规律书写开头。如果不是以正则匹配的开头那么测试会返回false。



$ 表示结尾

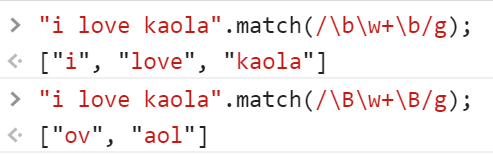


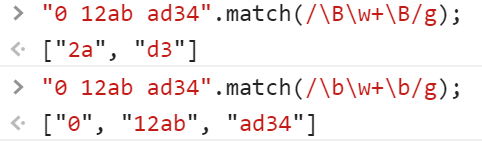
开头和结尾的界定符号，可以组合使用：



\b 单词边界 指[a-zA-Z\_0-9]之外的字符

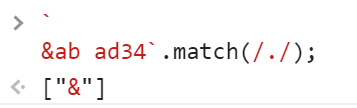
\B 非单词边界



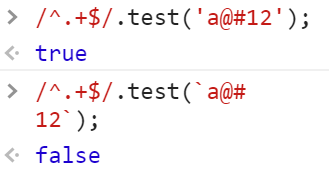


## 4.6 预定义类

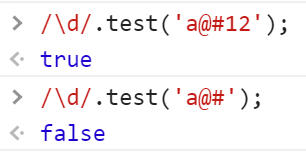
. [^\n\r] 除了换行和回车之外的任意字符



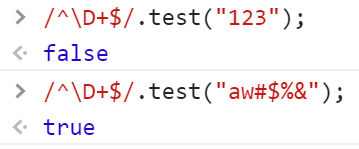
练习：



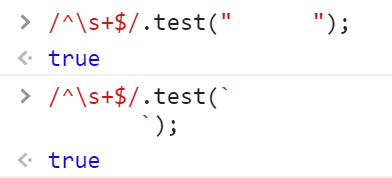
\d [0-9] 数字字符



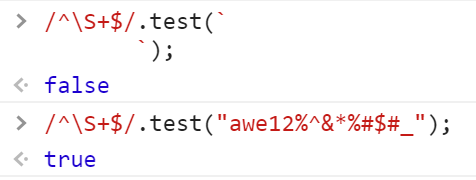
\D [^0-9] 非数字字符



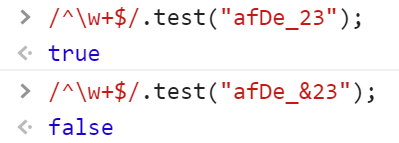
\s [ \t\n\x0B\f\r] 空白字符



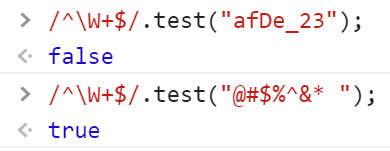
\S [^ \t\n\x0B\f\r] 非空白字符



\w [a-zA-Z\_0-9] 单词字符(所有的字母、下划线、数字)



\W [^a-zA-Z\_0-9] 非单词字符



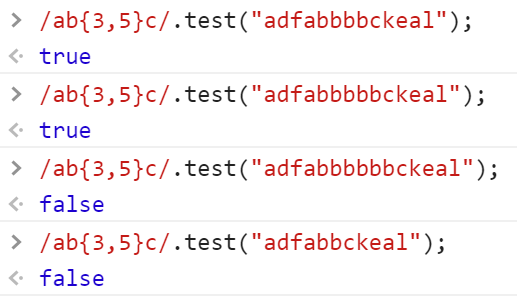
## 4.7 量词

写在一对儿大括号里面{}。可以规定前面的术语出现几次。

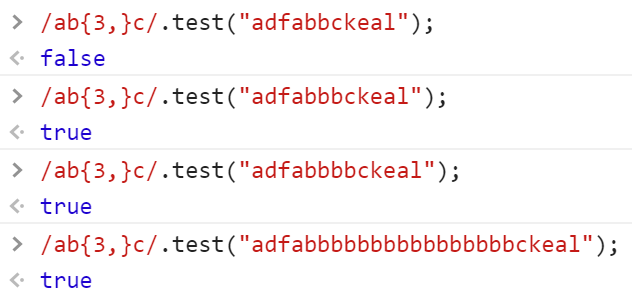
{n} 硬性量词 对应零次或者n次



{m,n} 对应前面的术语可以出现大于等于m次，小于等于n次。



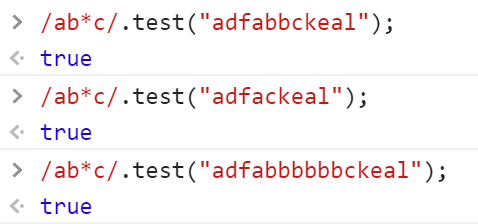
{m,} 对应前面的术语最少出现m次，多于m次是允许的。



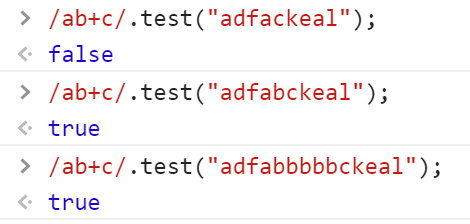
? 表示出现0次或1次 相当于{0,1}



\* 表示出现0次或以上 相当于{0,}



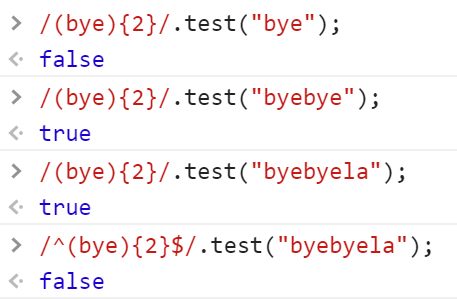
+ 表示出现1次或以上 相当于{1,}



## 4.8 分组

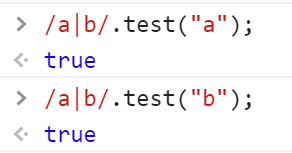
虽然量词的出现，能帮助我们处理一排密紧密相连的同类型字符。但这是不够的，我们用中括号表示范围内选择，大括号表示重复次数。如果想获取重复多个字符，我们就要用小括号进行分组了。

分组的字符用一对小括号包裹，可以同时被操作。

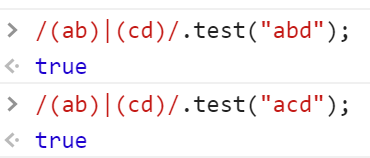


## 4.9 或操作符

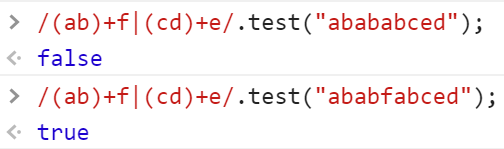
我们可以用一个竖线“|”表示或者的意思。要么出现前面的，要么出现后面。



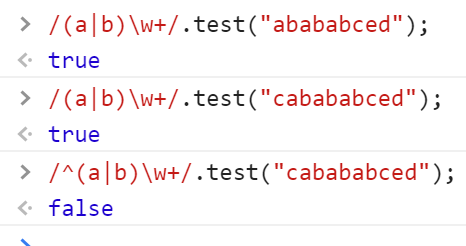
字符串里面或者有a，或者有b，都可以正确匹配。



字符里面必须有连在一起书写的ab或者cd，才能正确匹配。



说明：或操作符可以将它前面和后面区分成两个大的部分，要么匹配前面要么匹配后面。

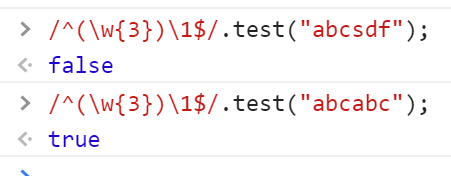
说明：如果只需要一部分二选一，这一部分需要再写在一个分组小括号里。

## 4.10 分组的反向引用

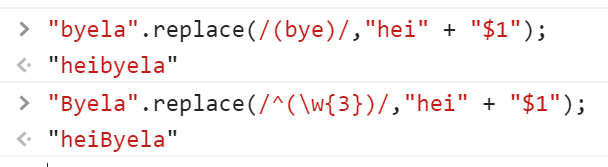
反向引用标识是对正则表达式中的匹配组捕获的子字符串进行编号。

正则里的引用：用一个“\编号”，编号从1开始。

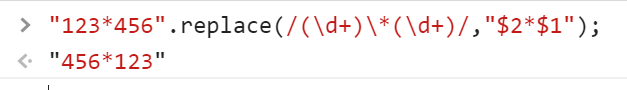
字符串方法里的引用：用一个“$编号”，编号从1开始。

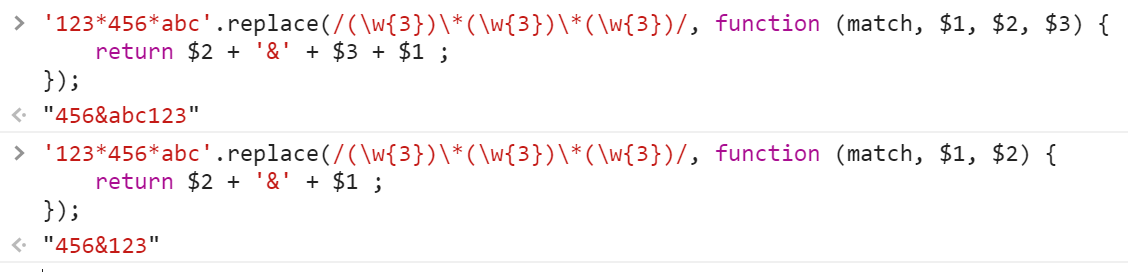


注意：反向引用的值是分组的匹配之后的子字符串，并不是又用分组里面的正则再次去匹配一次。反向引用的值必须和前面分组匹配的值完全相同。



两个分组的反向引用案例：





## 4.11 中文

匹配中文：[\u4e00-\u9fa5]

匹配范围类，包含了我们所有的中文。



|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |