正则表达式

2019年8月22日

10:29

一、什么是正则表达式

正则表达式就是指定一套规则，使用这套规则来匹配字符串

常见规则

\d：表示一个十进制的数字 [0-9]

\D：表示非数字

\w：表示一个字 [0－9a-zA-Z\_]

\W：表示除[0－9a-zA-Z\_]之外的字符

\s：表示一个空白字符（空格，tab，换页符等）

\S：表示一个非空白字符

二、定义正则表达式

1、通过两个斜杠，把表达式的内容写在斜杠内

var reg = /a/gi；

a:要匹配的内容

g:全文查找

i:忽略大小写来查找

2、通过构造方法

var reg = new RegExp("a","gi")；//匹配所有的a和A

三、使用正则表达式验证手机号码

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12343554133 |  | 1234323144 |  | 1234934334 |

var reg = /^1[0-9]{10}$/;

^:匹配字符串的开始

1:此位置必须是1

[0-9]:此位置可以是0-9的任意数字

\d:任意数字，等价于[0-9]

{10}:前一项匹配10次

{n,}:前一项至少匹配n次

{n,m}:前一项匹配n到m次

$:匹配字符串的结束

. : 匹配除回车以外的任意字符

四、使用正则表达式验证固话

|  |  |  |
| --- | --- | --- |
| 010-23423 |  | 0531-1234234 |

var reg = /^0\d{2,3}-\d{7,8}$/;

var reg = /^0\d{2}-\d{8}$|^0\d{3}-\d{8}$/;

五、使用正则表达式验证邮箱

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 123432@qq.com |  | hvgu@163.com |  | hvgu@edu.com.cn |

var reg = /^\w+@\w+(\.\w+){1,2}$/;

[\u4e00-\u9fa5]:匹配中文字符

/\p{Unified\_Ideograph}/u:匹配所有汉字

\w:匹配任意字符（数字、字母、下划线、汉字）

+:前一项出现1次或多次，等价于{1，}

\*:前一项不出现或出现任意次，等价于{0，}

?:前一项出现0次或1次，等价于{0，1}

/g 表示全局模式

/u 表示按unicode(utf-8)匹配（主要针对多字节比如汉字）

/i 表示不区分大小写（如果表达式里面有 a， 那么 A 也是匹配对象）

/s 表示将字符串视为单行来匹配

六、使用正则表达式验证身份证号码(15位或18位)

var reg = /^\d{15}$|^\d{18}$/

var reg = /^\d{14}(\d|X|x)$|^\d{17}(\d|X|x)$/

var str="123,123,123123,cat,javascript,cat";

var reg=/,/;

//alert(reg.test(str);//检查reg是否存在，返回true或false

//alert(reg.exec(str));//返回匹配的字符串或null

//alert(str.search(reg));//返回匹配字符串出现的位置或-1

var reg=/12/g

var arr=str.match(reg);//以数组的形式返回所有匹配的字符串

for(vari=0;i<arr.length;i++){

//alert(arr[i]);

}

alert(str.replace(reg,"dog"));//替换匹配的字符串

var arr=str.split(reg);//利用匹配的字符串将原始字符串分割为数组

for(vari=0;i<arr.length;i++){

alert(arr[i]);

}

七、贪婪模式

正则匹配默认贪婪模式,例子 :

var re = /^(\d+)(0\*)$/;

re.exec('102300'); // ['102300', '102300', '']

为了让\d+使用非贪婪匹配,可以加个"?",例子

var re = /^(\d+?)(0\*)$/;

re.exec('102300'); // ['102300', '1023', '00']

八、匹配结果

使用reg.exec(String)获取String.match(reg)来匹配时,返回值包括了每一个括号的内容.

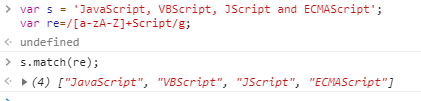
九、全局匹配

全局匹配标识 : "g"

全局匹配特征 : 全局匹配能够执行多次正则匹配,每次匹配都会更新正则表达式的lastIndex 属性,例子:



对于字符串的方法match,全局匹配能够返回多次正则匹配的返回值所组成的数组



断言 :

* (?=pattern) 正向先行断言,紧接该位置之后的字符序列能够匹配pattern
* (?!pattern) 负向先行断言,紧接该位置之后的字符序列不能够匹配pattern
* (?<=pattern) 正向后行断言
* (?<!pattern) 负向后行断言

注意 : 在JS中,后行断言不被支持

字符组 [\*] :

* 字符组的匹配结果只能是单个字符,[hello]匹配的不是hello而是h或e或l或o
* 字符组内特殊字符无需转义,[\^\*\-+|(a)]表示^ \* - + | ( a )
* 连字符[a-z],要求后面字符字符码要大于前面的字符码
* 排除型[^a-z],匹配排除了a-z的所有字符
* \b,字符组内表示退格符,字符组外表示单词边界

捕获组 /(a)(b)(c)/ :

* 组0是正则表达式整体匹配结果，组123才是子表达式匹配结果
* 在replace中，JS通过$number引用捕获组内容,不要引用$0
* 反向引用 :在正则表达式内对捕获组进行引用,/(boom)==\1/,等价于(boom)==boom

非捕获组 (?:[a-z]+)

通配符 . \s \S

* .定义是除了\n以外的所有字符
* \s是所有的空白,包括空白,换行,tab缩进
* \S是除了空白以外的任何字符
* 匹配所有字符 - (.|\n)或者是[\s\S](推荐用法)

单词边界 \b

* \b匹配单词边界，不匹配任何字符,以\w来界定单词
* 在字符组中[\b]表示的是退格符

var str = 'abc\_d=efg+hij哈opq%';

var reg = /.\b./g/

// ["d=", "g+", "j哈", "q%"]

str.match(reg);

* 可见单词的结束点处具有一个分界