Javascript正则表达式

1.直接量语法：

/pattern/attributes;

2. 创建RegExp对象的语法

new RegExp(pattern,attributes);

参数：参数pattern是一个字符串，指定了正则表达式的模式；参数attributes是一个可选的参数，包含属性 g，i，m，分别使用与全局匹配，不区分大小写匹配，多行匹配；

* 1. **支持正则表达式的String对象的方法**

1.search()方法；该方法用于检索字符串中指定的子字符串，或检索与正则表达式相匹配的字符串。

基本语法：stringObject.search(regexp);

@param 参数regexp可以需要在stringObject中检索的字符串，也可以是需要检索的RegExp对象。

@return(返回值) stringObject中第一个与regexp对象相匹配的子串的起始位置。如果没有找到任何匹配的子串，则返回-1；

注意：search()方法不执行全局匹配，它将忽略标志g，同时它也没有regexp对象的lastIndex的属性，且总是从字符串开始位置进行查找，总是返回的是stringObject匹配的第一个位置。

2.match()方法；该方法用于在字符串内检索指定的值，或找到一个或者多个正则表达式的匹配。该方法类似于indexOf()或者lastIndexOf(); 但是它返回的是指定的值，而不是字符串的位置；

基本语法：

stringObject.match(searchValue) 或者stringObject.match(regexp)

@param(参数)

searchValue 需要检索字符串的值；

regexp: 需要匹配模式的RegExp对象；

@return(返回值) 存放匹配成功的数组;

它可以全局匹配模式，全局匹配的话，它返回的是一个数组。

如果 regexp 没有标志 g，那么 match() 方法就只能在 stringObject 中执行一次匹配。如果没有找到任何匹配的文本， match() 将返回 null。否则，它将返回一个数组，其中存放了与它找到的匹配文本有关的信息。该数组的第 0 个元素存放的是匹配文本，而其余的元素存放的是与正则表达式的子表达式匹配的文本。除了这些常规的数组元素之外，返回的数组还含有两个对象属性。index 属性声明的是匹配文本的起始字符在 stringObject 中的位置，input 属性声明的是对 stringObject 的引用。

如果 regexp 具有标志 g，则 match() 方法将执行全局检索，找到 stringObject 中的所有匹配子字符串。若没有找到任何匹配的子串，则返回 null。如果找到了一个或多个匹配子串，则返回一个数组。不过全局匹配返回的数组的内容与前者大不相同，它的数组元素中存放的是 stringObject 中所有的匹配子串，而且也没有 index 属性或 input 属性。

3. replace()方法：该方法用于在字符串中使用一些字符替换另一些字符，或者替换一个与正则表达式匹配的子字符串；

基本语法：stringObject.replace(regexp/substr,replacement);

@param(参数)

regexp/substr; 字符串或者需要替换模式的RegExp对象。

replacement：一个字符串的值，被替换的文本或者生成替换文本的函数。

@return(返回值) 返回替换后的新字符串

注意：字符串的stringObject的replace()方法执行的是查找和替换操作，替换的模式有2种，既可以是字符串，也可以是正则匹配模式，如果是正则匹配模式的话，那么它可以加修饰符g,代表全局替换，否则的话，它只替换第一个匹配的字符串；

replacement 既可以是字符串，也可以是函数，如果它是字符串的话，那么匹配的将与字符串替换，replacement中的$有具体的含义，如下：

1,2,3.....99 含义是：与regexp中的第1到第99个子表达式相匹配的文本。

4. split()方法: 该方法把一个字符串分割成字符串数组。

基本语法如：stringObject.split(separator,howmany);

@param(参数)

1. separator[必填项]，字符串或正则表达式，该参数指定的地方分割stringObject;

2. howmany[可选] 该参数指定返回的数组的最大长度，如果设置了该参数，返回的子字符串不会多于这个参数指定的数组。如果没有设置该参数的话，整个字符串都会被分割，不考虑他的长度。

@return(返回值) 一个字符串数组。该数组通过在separator指定的边界处将字符串stringObject分割成子字符串。

**1-2 RegExp对象方法**

1. test()方法：该方法用于检测一个字符串是否匹配某个模式；

基本语法：RegExpObject.test(str);

@param(参数) str是需要检测的字符串；

@return (返回值) 如果字符串str中含有与RegExpObject匹配的文本的话，返回true，否则返回false；

2. exec()方法: 该方法用于检索字符串中的正则表达式的匹配。

基本语法：RegExpObject.exec(string)

@param(参数)：string【必填项】要检索的字符串。

@return(返回值)：

如果 exec() 找到了匹配的文本，则返回一个结果数组。否则，返回 null。此数组的第 0 个元素是与正则表达式相匹配的文本，第 1 个元素是与 RegExpObject 的第 1 个子表达式相匹配的文本（如果有的话），第 2 个元素是与 RegExpObject 的第 2 个子表达式相匹配的文本（如果有的话），以此类推。除了数组元素和 length 属性之外，exec() 方法还返回两个属性。index 属性声明的是匹配文本的第一个字符的位置。input 属性则存放的是被检索的字符串 string。我们可以看得出，在调用非全局的 RegExp 对象的 exec() 方法时，返回的数组与调用方法 String.match() 返回的数组是相同的。

但是，当 RegExpObject 是一个全局正则表达式时，exec() 的行为就稍微复杂一些。它会在 RegExpObject 的 lastIndex 属性指定的字符处开始检索字符串 string。当 exec() 找到了与表达式相匹配的文本时，在匹配后，它将把 RegExpObject 的 lastIndex 属性设置为匹配文本的最后一个字符的下一个位置。这就是说，您可以通过反复调用 exec() 方法来遍历字符串中的所有匹配文本。当 exec() 再也找不到匹配的文本时，它将返回 null，并把 lastIndex 属性重置为 0。

了解正则中的方括号[]的含义

[abc]: 查找在方括号中的任意一个字符；

[^abc]: 查找不在方括号中的任意一个字符；

[0-9]: 查找0-9中的任意一个数字；

[a-z]: 查找从小写a到z中的任意一个字符；

(red|blue|green); 查找小括号中的任意一项，小括号中的 | 是或者的意思；

理解javascript中的元字符

1. 元字符. 用于匹配任何单个字符(除了换行符以外)；

基本语法：new RegExp(“regexp.”) 或者直接量语法 /regexp./

比如代码如下：

var str = "abcde";

console.log(str.match(/a.c/)); // ["abc", index: 0, input: "abcde"]

2. \w; 查找任意一个字母或数字或下划线，等价于A\_Za\_z0\_9,\_

基本语法：new RegExp(“\w”); 或 直接量语法：/\w/

比如代码如下：

var str = "abcde";

// 匹配单个字符，找到一个直接返回

console.log(str.match(/\w/)); // ["a", index: 0, input: "abcde"]

// 匹配所有字符

console.log(str.match(/\w+/)); //["abcde", index: 0, input: "abcde"]

3. \W; 查找非单词的字符，等价于[^A\_Za\_z0\_9\_]

基本语法：new RegExp(“\W”) 或直接量 /\W/

var str = "abcde";

// 匹配单个字符，没有找到返回null

console.log(str.match(/\W/)); // null

4. \d；匹配与一个数字字符，等价于[0-9];

基本语法：new RegExp(“\d”); 或 直接量语法：/\d/

代码如下：

var str = "abcde111";

console.log(/\d/g.exec(str)); // ["1", index: 5, input: "abcde111"]

5. \D; 匹配一个非数字字符，等价于[^0-9]

基本语法：new RegExp(“\D”) 或直接量 /\D/

如下测试代码：

var str = "abcde111";

console.log(/\D+/g.exec(str)); // ["abcde", index: 0, input: "abcde111"]

6. \s；匹配任何空白字符，包括空格，制表符，换行符等等。等价于[\f\n\r\t\v]

基本语法：new RegExp(“\s”) 或直接量 /\s/

如下测试代码：

var str="Is this all there is?";

console.log(/\s/g.exec(str)); // [" ", index: 2, input: "Is this all there is?"]

7. \S；匹配任何非空白字符，等价于[^\f\n\r\t\v]

基本语法：new RegExp(“\S”) 或直接量 /\S/

如下测试代码：

var str="Is this all there is?";

console.log(/\S+/g.exec(str)); // ["Is", index: 0, input: "Is this all there is?"]

8. \b; 匹配一个单词边界，也就是指单词和空格间的位置，比如’er\b’可以匹配”never”中的”er”,但是不能匹配”verb”中的”er”

基本语法：new RegExp(“\bregexp”) 或直接量 /\bregexp/

如下测试代码：

var str="Is this all there is?";

console.log(/\bthis\b/g.exec(str)); // ["this", index: 3, input: "Is this all there is?"]

9. \B; 匹配非单词边界,’er\B’能匹配’verb’中的’er’,但不能匹配’never’中的’er’

基本语法：new RegExp(“\Bregexp”) 或直接量 /\Bregexp/

测试代码如下：

var str="Is this all there is?";

console.log(/\Bhi/g.exec(str)); // ["hi", index: 4, input: "Is this all there is?"]

10. \n; 匹配一个换行符;返回换行符被找到的位置。如果未找到匹配，则返回 -1。

基本语法：new RegExp(“\n”) 或直接量 /\n/

如下测试代码：

var str="Is this all \nthere is?";

console.log(/\n/g.exec(str)); // ["换行符", index: 12, input: "Is this all ↵there is?"]

11. \xxx; 查找一个以八进制数xxx规定的字符,如果未找到匹配，则返回 null。

基本语法：new RegExp(“\xxx”) 或直接量 /\xxx/

如下测试代码：

var str="Visit W3School. Hello World!";

console.log(/\127/g.exec(str)); // ["W", index: 6, input: "Visit W3School. Hello World!"]

如上代码分析：127的八进制转换为10进制的值等于 1\*8的二次方 + 2\*8的一次方 + 7\*8的0次方 = 64 + 16 + 7 = 87 而W的ASCLL编码转换为10进制也是87，因此打印W

12. \xdd；查找以16进制数dd规定的字符。如果未找到匹配，则返回 null。

基本语法：new RegExp(“\xdd”) 或直接量 /\xdd/

如下测试代码：

var str="Visit W3School. Hello World!";

console.log(/\x57/g.exec(str)); // ["W", index: 6, input: "Visit W3School. Hello World!"]

W的16进制数等于57；

13.\uxxxx; 查找以16进制数的xxxx规定的Unicode字符。

基本语法：new RegExp(“\uxxx”) 或直接量 /\uxxx/

如下测试代码：

var str="Visit W3School. Hello World!";

console.log(/\u0057/g.exec(str)); // ["W", index: 6, input: "Visit W3School. Hello World!"]

**了解量词**

**1. n+ 匹配至少包含一个或者多个n的字符串。**

**2. n\* 匹配零个或者多个n的字符串。**

**3. n？匹配零个或者1个n的字符串**

**4. n{x}  匹配包含x个的n的序列字符串。X必须是数字。**

**5. n{x,y} 匹配包含至少x个的n字符串，最多y个n字符串**。

**6. n{x,} 匹配至少包含x个n序列的字符串；**

**7. n$ 匹配任何以n结尾的字符串；**

**8. ^n 匹配任何以n开头的字符串；**

**9. ?=n 匹配任何其后紧接指定字符串n的字符串；**

**10 ?!n 匹配任何其后不紧接n的字符串**

**11. ^ 以字符串开始的地方匹配，不匹配任何字符；**

**12. $ 以字符串结束的地方匹配，不匹配任何字符；**

**13. \b 匹配一个单词边界，也就是单词与空格之间的位置，不匹配任何字符；**

**14. |  左右两边表达式之间 “或” 关系，匹配左边或者右边。**

**15.？<=n  匹配任何其前是指定字符串n的字符串；(es6)**

**16.?<!n匹配任何其前不是字符串n的字符串；(es6)**

**实例**

**22. 验证邮箱的正则表达式**

 一般域名的规律为“[N级域名][三级域名.]二级域名.顶级域名”，比如“qq.com”、“www.qq.com”、“mp.weixin.qq.com”、“12-34.com.cn”，分析可得域名类似“\*\* .\*\* .\*\* .\*\*”组成。

* “\*\*”部分可以表示为[a-zA-Z0-9\_-]+
* “.\*\*”部分可以表示为\.[a-zA-Z0-9\_-]+
* 多个“.\*\*”可以表示为(\.[a-zA-Z0-9\_-]+)+

 综上所述，域名部分可以表示为[a-zA-Z0-9\_-]+(\.[a-zA-Z0-9\_-]+)+

Var reg=/^[a-zA-Z0-9\_-][+@[a-zA-Z0-9\_-]+(\.[a-zA-Z0-9\_-]+)+$/](mailto:+@[a-zA-Z0-9_-]+(\.%5ba-zA-Z0-9_-%5d+)+$/)

**23.验证QQ号的正则表达式**

Var reg=/^[1-9]\d{4,10}$/

**24.验证手机号的正则表达式**

Var reg=/^1[34578]\d{9}$/

**25.去除字符串中的标点符号**

Var str=str.replace(/\W+|\_/,’’)

26.支持中文、数字、字母、”\_”、”-”的组合

var pat=/^(\w|[\u4e00-\u9fa5]|-){4,20}$/;