Day 14---正则表达式

一、正则表达式概述

1.1 什么是正则表达式

正则表达式(regular expression)是一个描述字符模式的对象，ECMAScript的RegExp 类表示正则表达式，而String和RegExp都定义了使用正则表达式进行强大的模式匹配和文本检索与替换的函数。

正则表达式用于对字符串模式匹配及检索替换，是对字符串执行模式匹配的强大工具。

1.2 正则表达式的作用

正则表达式主要用来验证客户端的输入数据。

用户填写完表单单击按钮之后，表单就会被发送到服务器，在服务器端通常会用PHP、ASP.NET、JSP等服务器脚本对其进行进一步处理。因为客户端验证，可以节约大量的服务器端的系统资源，并且提供更好的用户体验。

二、创建正则表达式(123)

> 要使用正则表达式，必须先创建正则表达式对象，有2种创建对象的方式：

2.1 方式1：使用关键字new创建

> var patt = new RegExp(pattern,modifiers);
>
> 参数1：正则表达式的模式。字符串形式
>
> 参数2：模式修饰符。用于指定全局匹配、区分大小写的匹配和多行匹配

```
<script type="text/javascript">
    /*
        创建了一个正则表达式
        参数1：模式是：girl，意思是说可以匹配 "girl"这样的字符串
        参数2：模式修饰符：gi     g代表全局匹配  i代表不区分大小写
    */
    var pa = new RegExp("girl", "gi");
    //测试参数中的字符串"你好我的girl" 是否与匹配模式匹配。
    var isExist = pa.test("你好我的girl");  // 在本例中，是匹配的，这个字符串包含girl，所以返回true
    alert(isExist); //true
</script>
```

2.2 方式2：使用正则表达式直接量

> var pa = /pattern/modifiers;

> 两个/中间的表示正则表达式的模式，最后一个/的后面是模式修饰符
>
> 例如：上面的例子可以这样写 var pa = /girl/gi;
>
> 注意：这个时候模式和模式修饰符都能再添加双引号或单引号

```
<script type="text/javascript">
    var pa = /girl/gi;
    alert(pa.test("厉害了我的girl"));     //true
</script>
```

三、正则表达式模式修饰符(126)

> JavaScript中共有3种模式修饰符：g i u
>
> 1. g：表示全局。意思是说会对一个字符串进行多次匹配。如果不写g则只匹配一次，一旦匹配成功，则不会再次匹配
> 2. i：表示忽略大小写。意思是说在匹配的不区分大小写
> 3. u：表示可以多行匹配。

四、正则表达式方法详解(127)

> 经常用到的正则表达式方法有两个test() 和 exec()

4.1 test()方法

> test(字符串)
>
> - 参数：要匹配的字符串
> - 返回值：匹配成功返回true，失败返回false
>
> 在只想知道目标字符串与某个模式是否匹配，但不需要知道其文本内容的情况下，使用这个方法非常方便。因此， test() 方法经常被用在 if 语句中。

```
<script type="text/javascript">
    var pa = /girl/gi;
    if(pa.test("厉害了我的girl")){
        alert("这个女孩和你很配");
    }else {
        alert("你注定没有女孩去匹配");
    }
</script>
```

4.2 exec()方法

> exec(字符串)：该方法为专门为捕获组而设计的
>
> - 参数：要匹配的字符串
> - 返回值：返回的是一个数组。如果不匹配则返回null

> 关于返回值数组的说明：
>
> 1. 它确实是Array的实例。
> 2. 但是这个数组有两个额外的属性：index和input
>   - index：表示匹配的字符串在源字符串中的索引
>   - input：表示匹配的源字符串。
> 3. 数组的第一项目是与整个模式匹配的字符串，其他项是与模式中捕获组匹配的字符串
> 4. 如果没有捕获组，则数组中只有第一项。关于捕获组的概念以后再说

```
<script type="text/javascript">
    var pa = /girl/gi;
    var testStr = "myGirl, yourgirl, hisgIrl";
    var girls = pa.exec(testStr);  //捕获
    alert(girls.length + ":" + (girls instanceof Array));    //正则表达式没有捕获组，所以数组长度为1
    alert(girls[0]);     //第一次捕获的是 Girl
    //因为我们是用的全局匹配，所以此次匹配的时候从上次匹后的位置开始继续匹配
    alert(pa.exec(testStr)[0]);      // girl
    alert(pa.exec(testStr));     // gIrl
    alert(pa.exec(testStr)); //继续向后没有匹配的字符串，所以返回null
    // 返回null，如果继续再匹配，则会回到字符串的开始，重写开始匹配。
    alert(pa.exec(testStr));     // Girl
    // ...开启新一轮匹配
</script>
```

> 所以我们如果想找到全部匹配的字符串可以时候用循环，结束条件就是匹配结果为null

```
<script type="text/javascript">
    var pa = /girl/gi;
    var testStr = "myGirl, yourgirl, hisgIrl";
    var girls;
    while(girls = pa.exec(testStr)){     //如果等于null,会自动转成 false，结束。
        alert(girls);
    }
</script>
```

> 分组。在正则表达式中用()括起来任务是一组。组可以嵌套。

```
<script type="text/javascript">
    //()内的内容就是第1组(Girl)，其实我们完整真个表达式可以看出第0组 girl(Girl)
    // 将来对应着匹配结果数组的下标。
    var pa = /girl(Girl)/gi;
    var test = "girlGirl abdfjla Girlgirl fal girl";
    var girls;
    while(girls = pa.exec(test)){
      //匹配之后，数组的第0个元素对应的这第0组的匹配结果，第1个元素对应着第1组的匹配结果
        for (var i = 0; i < girls.length; i++) {
            console.log(girls[i]);
        }
        console.log("-------------");
    }
</script>
//最终运行结果：
girlGirl
Girl
-------------
Girlgirl
girl
------------
```

五、正则表达式规则(124)

表达式规则

正则表达式元字符是包含特殊含义的字符。它们有一些特殊功能，可以控制匹配模式的方式。反斜杠后的元字符将失去其特殊含义。

字符类：单个字符和数字

```
[0-9A-Za-z]
```

元字符/元符号	匹配情况
.	匹配除换行符外的任意字符
[a-z0-9]	匹配括号中的字符集中的任意字符
[^a-z0-9]	匹配任意不在括号中的字符集中的字符
\d ==[0-9]	匹配数字
\D ==[^0-9]	匹配非数字，同[^0-9]相同
\w [0-9A-Za-z_]	匹配字母和数字及_
\W	匹配非(字母和数字及_)

字符类：空白字符

元字符/元符号	匹配情况
\0	匹配null 字符
\b	匹配空格字符
\n	匹配换行符
\r	匹配回车字符
\t	匹配制表符
\s	匹配空白字符、空格、制表符和换行符
\S	匹配非空白字符

字符类：锚字符

元字符/元符号	匹配情况
^	行首匹配
$	行尾匹配

字符类：重复字符

元字符/元符号	匹配情况
? 例如（x?）	匹配0个或1 个x
* 例如（x*）	匹配0个或任意多个x
+ 例如（x+）	匹配至少一个x
(xyz)+	匹配至少一个(xyz)
{m,n} 例如x{m,n} n>=次数>=m	匹配最少m个、最多n个x
{n}	匹配前一项n次
{n,}	匹配前一项n次，或者多次

六、常用正则表示(128)

```
1、检查邮政编码
var pattern = /[1-9][0-9]{5}/; //共6位数字，第一位不能为0
var str = '224000';
alert(pattern.test(str));
2、检查文件压缩包
var pattern = /[\w]+\.zip|rar|gz/; //\w 表示所有数字和字母加下划线
var str = '123.zip'; //\.表示匹配.，后面是一个选择
alert(pattern.test(str));
3、删除多余空格
var pattern = /\s/g; //g 必须全局，才能全部匹配
var reg=new RegExp('\\s+','g');
var str = '111 222 333';
var result = str.replace(pattern,''); //把空格匹配成无空格
alert(result);
4、删除空格
var pattern = /^\s+/;
var str = ' goo gle ';
alert(str+"  "+str.length);
var result = str.replace(pattern, '');
alert(result+" "+result.length);
pattern = /\s+$/;
result = result.replace(pattern, '');
alert(result+" "+result.length);
pattern = /\s+/g;
result = result.replace(pattern, '');
alert(result+" "+result.length);
5、简单的电子邮件验证
var pattern = /^([a-zA-Z0-9_\.\-]+)@([a-zA-Z0-9_\.\-]+)\.([a-zA-Z]{2,4})$/;
var str = 'yc60.com@gmail.com';
alert(pattern.test(str));
var pattern = /^([\w\.\-]+)@([\w\.\-]+)\.([\w]{2,4})$/;
var str = 'yc60.com@gmail.com';
alert(pattern.test(str));
```

七、支持正则表达式的字符串方法

方法	描述
search	检索与正则表达式相匹配的第一个匹配项的索引。
match	找到一个或多个正则表达式的匹配。
replace	替换与正则表达式匹配的子串。
split	把字符串分割为字符串数组。

```
<script type="text/javascript">
    var s = "Abc123aBc";
    alert(s.search(/abc/gi));
    alert(s.search(/abc/gi)); // 即使设置的全局模式，每次search也是从开始向后查找

    //match方法和正则表达式的exec()方法的作用是一样的，但是match会一次性把所有的匹配放在一个数组中，全部返回
    alert(s.match(/abc/gi));    // Abc,aBc

    alert(s.replace(/[ab]/gi, "x"));        //把a或b替换成x
    var ss = s.split(/[0-9]+/gi);   //用1个或多个数字切割。    Abc,aBc
    alert(ss);
</script>
```