1 JavaScript正则

1.1 使用步骤

（1）定义正则表达式对象，有2种方式，3种匹配模式。
（2）和待校验字符串进行校验。

```
var reg = /abc/gim;
var reg2 = new RegExp("abc", "gim");
var str = "abcdefg";
var flag = reg.test(str);
var flag2 = reg2.test(str);
console.log(flag);
console.log(flag2);
```

如上定义了两种正则表达式，使用 `//` 定义或创建一个 `RegExp` 对象。
`gim` 表示3中匹配模式，`g` 表示全局匹配，即符合条件的每一处都会匹配，`i` 表示忽略大小写，`m` 表示多行匹配，即如果字符中有 `\n` 会作为多行对每一行进行匹配。

调用 `test()` 方法进行匹配，正确返回true，错误返回false。

1.2 元字符

`.` ： 除换行外的任意字符
`\w` ： 字母、数字、下划线
`\W` ： 非单词字符
`\s` ： 空白符，包括空格、制表符、换页符
`\S` ： 非空白符
`\d` ： 数字
`\D` ： 非数字
`\b` ： 单词的开始或结束
`^` ： 字符串开始
`$` ： 字符串结束

1.3 集合表示

使用 `[]` 表示。`[abc]` 表示包含 `abc` 的任一字符。
`[^abc]` 表示除 `abc` 的任一字符。
`[a-z]` 表示所有小写英文字符。

1.4 出现次数

`*` 表示出现0次或多次
`+` 表示出现1次或多次
`?` 表示出现0次或1次
`{n}` 表示出现n次
`{n,}` 表示出现n次或多次
`{n,m}` 表示出现n次到m次

1.5 或者

使用 `|` 表示或者，如 `/World|Java/` 和World、Java都匹配。

1.6 案例

在书城项目中，比如注册的邮箱，使用JavaScript的正则表达式判断，如果格式不正确，返回false，不进行表单提交。

```
<form th:action="@{/user.do}" method="post" onsubmit="return check();">
```

在regist.html中的form表单的onsubmit，根据函数check()的返回值决定是否发送HTTP请求给服务器。

```
var email = $("emailTxt").value;
var emailSpan = $("emailSpan");
reg = /^[a-zA-Z0-9_\.-]+@([a-zA-Z0-9-]+[\.]{1})+[a-zA-Z]+$/;
if (!reg.test(email)) {
    emailSpan.style.visibility = "visible";
    return false;
}
emailSpan.style.visibility = "hidden";
```

上面是regist.js文件中check()函数判断邮箱的部分，如果邮箱不匹配正则表达式，返回false，不进行提交，并且显示提示，即html页面的提示span可见。

2 Ajax

一般的请求是同步的，用户点击网页按钮，浏览器页面变为空白（一般服务器响应较快，时间短），等待服务器响应进行页面刷新。异步请求则是，发送请求后，浏览器的内容不变，直到服务器响应完成后，浏览器收到响应结果再进行处理。异步请求的好处是用户可以继续浏览网页内容。

原生Ajax的实现如下，以注册页面判断用户名是否已经注册为例。
原理：当注册页面的用户名文本框失去焦点后，浏览器会访问服务器判断用户名是否已经注册，但此时用户还能继续在网页上填写其他文本框，根据服务器返回的json数据，如果已经注册，页面上有提示。

```
<input type="text" placeholder="请输入用户名" id="unameTxt" name="uname"
onblur="checkUname()"/>
```

在regist.html的用户名文本框中，添加onblur事件，失去焦点时执行checkUname()函数。

```
function checkUname() {
    createXMLHttpRequest();
    var uname = $("unameTxt").value;
    var url = "user.do?operate=checkUname&uname=" + uname;
    xmlHttpRequest.open("GET", url, true);
    xmlHttpRequest.onreadystatechange = ckUnameCB;
    xmlHttpRequest.send();
}
```

checkUname()函数创建一个XMLHttpRequest对象，使用open()和send()方法给服务器发送请求，onreadystatechange设置回调函数，即收到服务器端响应后的处理。

```
var xmlHttpRequest;
function createXMLHttpRequest() {
    if (window.XMLHttpRequest) {
        xmlHttpRequest = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        try {
            xmlHttpRequest = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e) {
            xmlHttpRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
    }
}
```

创建XMLHttpRequest对象的代码如上，要判断浏览器类型。

回调函数ckUnameCB的代码如下

```
function ckUnameCB() {
    if (xmlHttpRequest.readyState === 4 && xmlHttpRequest.status === 200) {
        var responseText = xmlHttpRequest.responseText;
        console.log(responseText);
        if (responseText === "{'uname':'1'}") {
            alert("用户名已经被注册！");
        }
    }
}
```

根据readyState和status判断如何处理，readyState有0-4四种状态，status200是正常响应，404等是出现相应的问题。再对返回的json数据进行处理。

在DispatcherServlet中使用如下方式返回json数据。

```
if (methodReturnStr.startsWith("json:")) {
    String jsonStr = methodReturnStr.substring("json:".length());
    PrintWriter out = response.getWriter();
    out.print(jsonStr);
    out.flush();
}
```

3 Vue

```
<script language="JavaScript" src="../js/vue.js"></script>
<script language="JavaScript">
    window.onload = function () {
        var vue = new Vue({
            "el": "#div0",
            data: {
                msg: "hello!!!",
                uname: "橘右京"
            }
        })
        }
</script>
```

在html中引入 `vue.js` 文件，在onload方法中创建一个Vue对象，`el` 表示作用域，`#div0` 表示作用域的id是div0。`data` 中定义的是变量名称和对应的值。

3.1 {{}}

相当于innerText，显示对应区域data内同名变量的值。

```
<div id="div0">
    <span>{{msg}}</span>
</div>
```

打开网页，上面显示msg对应的值，即 `hello!!!` 。

3.2 v-bind

绑定属性值。

```
<div id="div0">
    <input type="text" v-bind:value="uname"/>
    <input type="text" :value="uname"/>
</div>
```

文本框中的value属性和data中的uname绑定，显示uname的值。
两种方式均可。

3.3 v-model

```
<div id="div0">
    <span>{{uname}}</span>
    <input type="text" v-model:value="uname"/>
    <input type="text" v-model="uname"/>
</div>
```

双向绑定，修改文本框中uname的值，span中显示的uname值也会同时更改。两种方式均可。

3.4 trim

去除首尾空格。

```
<input type="text" v-model.trim="uname"/>
```

会自动去除文本框的首尾空格，即，即使用户在文本框中增加首尾空格，uname的值没有首尾空格。

3.5 v-if

```
<div v-if="num%2==0" class="div1"> </div>
<div v-else="num%2==0" class="div2"> </div>
```

`v-if` 和 `v-else` 之间不能有其他元素，根据条件判断显示哪个div，不显示的不会出现在网页源码。

3.6 v-show

```
<div v-show="num%2==0" class="div3"> </div>
```

根据条件判断是否显示，观察网页源码，通过增加了 `display: none;` 的样式控制不显示。

3.7 v-for

迭代。

```
<script language="JavaScript">
    window.onload = function () {
        var vue = new Vue({
            "el": "#div0",
            data: {
                fruitList: [
                    {fname: "苹果", price: 5, fcount: 88, remark: "苹果好吃"},
                    {fname: "西瓜", price: 9, fcount: 77, remark: "西瓜很甜"},
                    {fname: "菠萝", price: 5, fcount: 55, remark: "菠萝不好吃"}
                ]
            }
        })
    }
</script>
```

有如上vue对象，其中data有一个集合列表fruitList。

```
<div id="div0">
    <table>
        <tr>
            <th>名称</th>
            <th>单价</th>
            <th>库存</th>
            <th>评价</th>
        </tr>
        <tr v-for="fruit in fruitList">
            <td>{{fruit.fname}}</td>
            <td>{{fruit.price}}</td>
            <td>{{fruit.fcount}}</td>
            <td>{{fruit.remark}}</td>
        </tr>
    </table>
</div>
```

在div0中的table中使用v-for迭代展示fruitList数据。

3.8 v-on

绑定事件

```
var vue = new Vue({
    "el": "#div0",
    data: {
        msg: "hello world!"
    },
    methods: {
        myReverse: function () {
            this.msg = this.msg.split("").reverse().join("");
        }
    }
})
```

在onload中定义如上Vue对象，其中 `methods` 中有一个 `myReverse` 方法用于将变量 `msg` 字符串反转。

```
<div id="div0">
    <input type="text" v-bind:value="msg"/>
    <input type="button" value="反转" v-on:click="myReverse"/>
    <input type="button" value="反转" @click="myReverse"/>
</div>
```

使用 `v-on:click` 或 `@click` 绑定button的点击事件为调用 `myReverse` 方法，点击后可以看到文本框中的msg的值发生反转。

3.9 监听

```
var vue = new Vue({
    "el": "#div0",
    data: {
        num1: 1,
        num2: 2,
        num3: 3
    },
    watch: {
        num1: function (newValue) {
            this.num3 = parseInt(newValue) + parseInt(this.num2);
        },
        num2: function (newValue) {
            this.num3 = parseInt(this.num1) + parseInt(newValue);
        }
    }
})
```

使用 `watch` 关键字，对于变量num1和num2进行监听，如果监听到变量的值发生改变，调用对应的函数修改num3的值。

```
<div id="div0">
    <input type="text" v-model:value="num1"/>
    +
    <input type="text" v-model:value="num2"/>
    =
    <span>{{num3}}</span>
</div>
```

使用v-model双向绑定后即可实现简单加法功能。

3.10 生命周期

生命周期涉及的方法包括beforeCreate、created、beforeMount、mounted、beforeUpdate、updated。

```
var vue = new Vue({
    "el": "#div0",
    data: {
        msg: 1
    },
    methods: {
        changeMsg: function () {
            this.msg = this.msg + 1;
        }
    },
    beforeCreate: function () {
        console.log("beforeCreate:vue对象创建之前");
        console.log("msg:" + this.msg);
    },
    created: function () {
        console.log("created:vue对象创建之后");
        console.log("msg:" + this.msg);
    },
    beforeMount: function () {
        console.log("beforeMount:数据装载之前");
        console.log("span:" + $("span").innerText);
    },
    mounted: function () {
        console.log("mounted:数据装载之后");
        console.log("span:" + $("span").innerText);
    },
    beforeUpdate: function () {
        console.log("beforeUpdate:数据更新之前");
        console.log("msg:" + this.msg);
        console.log("span:" + $("span").innerText);
    },
    updated: function () {
        console.log("Updated:数据更新之后");
        console.log("msg:" + this.msg);
        console.log("span:" + $("span").innerText);
    }
});
```

使用下面的html页面测试

```
<div id="div0">
    <span id="span">{{msg}}</span><br/>
    <input type="button" value="改变msg的值" @click="changeMsg"/>
</div>
```

在Vue对象创建前（beforeCreate）msg是undefined，在数据装载之前（beforemounted）span中的内容是{{msg}}，点击修改button调用changeMsg方法，在数据更新前（beforeUpdate）msg的值已经修改了，但span中的值还未修改。

4 Axios

一个封装Ajax的框架。

需要先引入vue和axios的js文件

```
<script language="JavaScript" src="../js/vue.js"></script>
<script language="JavaScript" src="../js/axios.min.js"></script>
```

4.1 普通参数

```
var vue = new Vue({
    "el": "#div0",
    data: {
        uname: "lina",
        pwd: "ok"
    },
    methods: {
        axios01: function () {
            axios({
                method: "POST",
                url: "/pro07/axios01.do",
                params: {
                    uname: vue.uname,
                    pwd: vue.pwd
                }
            }).then(function (value) {
                console.log(value)
            }).catch(function (reason) {
                console.log(reason)
            });
        }
    }
})
```

在onload中定义如上Vue对象，在关键字methods中定义一个名为 `axios01` 的方法，使用 `axios({}).then().catch();` 格式，第一个括号中包括请求的method、url、参数等，请求参数使用params关键字。第二个括号中是成功后运行的函数，其中的参数value包括服务器响应内容等。第三个参数是出现异常后调用的函数，参数reason包括服务器响应内容、异常信息等。
成功响应后响应内容可以通过 `value.data` 获得。
有异常的响应内容通过 `reason.response.data` 获得， `reason.message` 和 `reason.stack` 可以获得错误信息。

在后端实现一个 `/axios01.do` Servlet。

```
resp.setCharacterEncoding("utf-8");
resp.setContentType("text/html;charset=utf-8");
PrintWriter out = resp.getWriter();
out.write("服务器收到的是" + uname + "_" + pwd);
```

使用 `out.write()` 方法返回给浏览器响应内容。

4.2 JSON格式

浏览器发送JSON数据

```
var vue = new Vue({
    "el": "#div0",
    data: {
        uname: "lina",
        pwd: "ok"
    },
    methods: {
        axios02: function () {
            axios({
                method: "POST",
                url: "/pro07/axios02.do",
                data: {
                    uname: vue.uname,
                    pwd: vue.pwd
                }
            }).then(function (value) {
                console.log(value)
            }).catch(function (reason) {
                console.log(reason)
            });
        }
    }
})
```

与发送普通参数类似，区别在于不是使用params关键字，而是使用data关键字。

服务器把JSON数据转换成Object

(1) 导入 `gson-2.2.4.jar`
(2) 从 `request.getReader()` 中读取JSON数据的字符串
(3) 创建一个Gson对象，使用 `fromJson()` 方法把接收到的解析成User类的对象。

```
StringBuilder sb = new StringBuilder();
BufferedReader br = req.getReader();
String str = null;
while ((str = br.readLine()) != null) {
    sb.append(str);
}
str = sb.toString();
Gson gson = new Gson();
User user = gson.fromJson(str, User.class);
System.out.println("user = " + user);
```

服务器把响应以JSON格式发给浏览器

```
String userJsonStr = gson.toJson(user);
resp.setCharacterEncoding("utf-8");
resp.setContentType("application/json;charset=utf-8");
resp.getWriter().write(userJsonStr);
```

使用Gson对象的 `toJson()` 方法把Java对象转换成JSON字符串，通过 `response.getWriter()` 响应给浏览器。

浏览器收到服务器的JSON数据并处理

```
then(function (value) {
    console.log(value)
    var data = value.data;
    vue.uname = data.uname;
    vue.pwd = data.pwd;
})
```

在axios方法的then中的函数进行处理，接收到的参数value的data已经是JavaScript的对象，可以直接获取其属性值。

JavaScript把JSON数据转换成对象

JavaScript语言中提供了对象和字符串相互转换的API

```
var str = "{\"uname\":\"lina\",\"pwd\":\"ok\"}";
var obj = JSON.parse(str);
console.log(obj.uname);
console.log(obj.pwd);
```

使用 `JSON.parse()` 方法可以把JSON字符串转换成JS对象。

```
var user = {"uname": "lina", "age": 20}
console.log(typeof user) // object
var str2 = JSON.stringify(user);
console.log(typeof str2) // string
console.log(str2)
```

使用 `JSON.stringify()` 方法可以把JS对象转换成字符串。