

Ajax

初始Ajax

局部刷新页面



The screenshot shows a simple registration form with a green '立即注册' (Register Now) button.

- 什么是Ajax
 - Ajax: 一种不用刷新整个页面便可与服务器通讯的办法
 - Web的传统模型。客户端向服务器发送一个请求，服务器返回整个页面，如此反复
 - 在Ajax模型中，数据在客户端与服务器之间独立传输。服务器不再返回整个页面
- XMLHttpRequest
 - 最早最重要的Ajax谜题是 XMLHttpRequest (XHR) API。XHR是一种用于在Web浏览器和Web服务器间传输数据消息的JavaScript API。它支持浏览器使用HTTP POST (将数据传到服务器) 或 GET 请求 (从后台服务器访问数据)。该API是大多数Ajax交互的核心，也是现代Web开发的一项基本技术。
 - Ajax并不是一项新技术，它实际上是几种技术，每种技术各尽其职，以一种全新的方式聚合在一起
 - 服务器端语言：服务器需要具备向浏览器发送特定信息的能力。Ajax与服务器端语言无关。

XMLHttpRequest

使浏览器发出HTTP请求与接收HTTP响应

使用XMLHttpRequest对象发送一个Ajzx请求



The screenshot shows a browser developer tools panel with the XMLHttpRequest object highlighted.

- XMLHttpRequest是一个浏览器接口，使得JavaScript可以进行HTTP(S)通信。
- 最早，微软在IE 5引进了这个接口。因为它太有用，其他浏览器也模仿部署了，ajax操作因此得以诞生。
 - Internet Explorer把XMLHttpRequest实现为一个ActiveX对象
 - 其他浏览器(Firefox、Safari、Opera...)把它实现为一个本地的JavaScript对象。
 - XMLHttpRequest在不同浏览器上的实现是兼容的，所以可以用同样的方式访问XMLHttpRequest实例的属性和方法，而不论这个实例创建的方法是什么。
- 但是，这个接口一直没有标准化，每家浏览器的实现或多或少有点不同。HTML 5的概念形成后，W3C开始考虑标准化这个接口。2008年2月，就提出了 XMLHttpRequest Level 2草案。
- 这个XMLHttpRequest的新版本，提出了很多有用的新功能，将大大推动互联网革新。

ajax是一种技术方案，但并不是一种新技术。它依赖的是现有的CSS/HTML/Javascript，而其中最核心的依赖是浏览器提供的XMLHttpRequest对象，是这个对象使得浏览器可以发出HTTP请求与接收HTTP响应。

所以我用一句话来总结两者的关系：我们使用XMLHttpRequest对象来发送一个Ajax请求。

XMLHttpRequest对象

Ajax.jsp

```

<script type="text/javascript">
    // 产生XMLHttpRequest对象
    function createXMLHttpRequest() {
        var xmlreq = false;
        if(window.ActiveXObject){
            // for IE6 IE5
            xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
        }else if (window.XMLHttpRequest){
            xmlreq = new XMLHttpRequest();
        }
        return xmlreq;
    }
</script>

```

XMLHttpRequest对象

新建一个XMLHttpRequest的实例。

```
var xht = new XMLHttpRequest();
```

- 为了应对所有的现代浏览器，包括IE5 和 IE6，请检查浏览器是否支持 XMLHttpRequest 对象。如果支持，则创建 XMLHttpRequest 对象。如果不支持，则创建 ActiveXObject：为了每次写Ajax的时候都节省一点时间，可以把对象检测的内容打包成一个可复用的函数：

```

function createXMLHttpRequest() {
    var xmlreq = false;
    if(window.ActiveXObject) {
        /*for IE6, IE5*/
        xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
    } else if(window.XMLHttpRequest) {
        /*for IE7+, Firefox, Chrome, Opera, Safari*/
        xmlreq = new XMLHttpRequest();
    }
}

```

XMLHttpRequest的方法

XMLHttpRequest的属性

响应的个属性和方法：

status 服务器返回的http状态码。200表示“成功”，404表示“未找到”，500表示“服务器内部错误”等。

onreadystatechange 请求状态改变的事件触发器（readyState变化时会调用这个属性上注册的javascript函数）。

statusText 包括 Web 服务器返回的完整响应字符串，其中包括响应文本（例如，304 Not Modified）

responseText 响应文本的字符串表示

responseXML 响应文本的 XML 表示，一个包含 DOM 和所有相关 DOM 方法的文档片段

readyState 实例化完成后，XMLHttpRequest 对象有 5 种状态，使用以下值表示：

- 0:未初始化。对象已创建，未调用open;
- 1: open方法成功调用，但Send方法未调用；
- 2: send方法已经调用，尚未开始接受数据；
- 3: 正在接受数据。Http响应头信息已经接受，但尚未接收完成；

XMLHttpRequest的属性

Ajax开发步骤

创建XMLHttpRequest对象

Ajax.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<script type="text/javascript">
    // 产生XMLHttpRequest对象
    function createXMLHttpRequest() {
        var xmlreq = false;
        if(window.ActiveXObject){
            // for IE6 IE5
            xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
        }else if (window.XMLHttpRequest){
            xmlreq = new XMLHttpRequest();
        }
        return xmlreq;
    }

    window.onload = function () {
        var btn = document.getElementById("btn");
        //给btn绑定点击事件
        btn.onclick = function (){

            //1、创建XMLHttpRequest对象，使用open()方法，打开与服务器链接
            var xmlreq = createXMLHttpRequest();
            //xmlreq.open("GET", "/ajaxServlet?id=1"); //发送的是get请求，数据通过url地址传递

            xmlreq.open("POST", "/ajaxServlet");
            //标识此次请求的请求体格式为： urlencode 以便于服务端接受数据
            xmlreq.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

            var id = 1;
            var data ="id =" + id;

            // 3、发送请求到服务端
            xmlreq.send(data);

            //4、相当于请求完成之后的回调处理函数
            xmlreq.onreadystatechange = function () {
                if (xmlreq.readyState === 4){//当响应完成之后 再处理响应数据
                    console.log(xmlreq.readyState)//接收整个响应的数据
                    document.getElementById("main").innerHTML =
                        <h1>"+xmlreq.responseText+"</h1>";

                }
            }
        }
    }
</script>
<html>
<head>
    <title>Ajax的学习</title>
</head>
```

```

<body>
    <input type="button" id="btn" value="AJax请求">
    <div id="main">
        </div>
    </body>
</html>

```

ajaxServlet

```

@WebServlet( "/ajaxServlet")
public class ajaxServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");
        PrintWriter printwriter = response.getWriter();
        printwriter.print("Hello Ajax");
        // ..... .
        String id = request.getParameter("id");
        System.out.println(id);
    }
}

```

打开与服务器的连接，并设置链接信息

打开与服务器的连接，并设置链接信息

当得到 XMLHttpRequest 对象后，就可以调用该对象的 open() 方法打开与服务器的连接了。open()方法的参数如下：

open(method, url, async):

- method: 请求方式，通常为 GET 或 POST;
 - url: 请求的服务器地址，例如：/ajaxdemo1/AServlet，若为 GET 请求，还可以在 URL 后追加参数；
 - async: 这个参数可以不给，默认值为 true，表示异步请求；
- GET 和 POST 请求方式的差异
- 1、GET 没有请求主体，使用 xhr.send(null)
 - 2、GET 可以通过在请求 URL 上添加请求参数
 - 3、POST 可以通过 xhr.send('name=itcast&age=10')
 - 4、POST 需要设置
 - 5、GET 效率更好（应用多）
 - 6、GET 大小限制约 4K，POST 则没有限制

发送数据

发送数据。

- `send(data):`

- `open` 方法定义了 Ajax 请求的一些细节。`send` 方法可为已经待命的请求发送指令
- `data`: 将要传递给服务器的字符串。
- 若选用的是 GET 请求，则不会发送任何数据，给 `send` 方法传递 `null` 即可: `request.send(null);`
- 当向 `send()` 方法提供参数时，要确保 `open()` 中指定的方法是 POST，如果没有数据作为请求体的一部分发送，则使用 `null`。
- 一定在发送请求 `send()` 之前注册 `readystatechange` (不管同步或者异步)

通常在一次 GET 请求过程中，参数传递都是通过 URL 地址中的 `?参数` 传递。

```
var xhr = new XMLHttpRequest()  
  
// GET 请求传递参数通常使用的是问号传参  
// 这里可以在请求地址后面加上参数，从而传递数据到服务端  
xhr.open('GET', './delete.php?id=1')
```

发送数据。

```
var xhr = new XMLHttpRequest()  
  
// open 方法的第一个参数的作用就是设置请求的 method  
xhr.open('POST', './add.php')  
  
// 设置请求头中的 Content-Type 为 application/x-www-form-urlencoded  
// 标识此次请求的请求体格式为 urlencoded 以便于服务端接收数据  
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded')  
  
// 需要提交到服务端的数据可以通过 send 方法的参数传递  
// 格式: key1=value1&key2=value2  
xhr.send('key1=value1&key2=value2')  
xhr.onreadystatechange = function () {  
    if (this.readyState === 4) {  
        console.log(this.responseText)  
    }  
}
```

接收响应数据

readyState

readyState

- `readyState`

- `readyState` 属性表示 Ajax 请求的当前状态。它的值用数字代表。
 - 0 表示未初始化。还没有调用 `open` 方法
 - 1 表示正在加载。`open` 方法已被调用，但 `send` 方法还没有被调用
 - 2 表示已加载完毕。`send` 已被调用。请求已经开始
 - 3 表示交互中。服务器正在发送响应
 - 4 代表完成。响应发送完毕
- 每次 `readyState` 值的改变，都会触发 `readystatechange` 事件。如果把 `onreadystatechange` 事件处理函数赋给一个函数，那么每次 `readyState` 值的改变都会引发该函数的执行。
- `readyState` 值的变化会因浏览器的不同而有所差异。但是，当请求结束的时候，每个浏览器都会把 `readyState` 的值统一设为 4

status

responseText

responseText

- **responseText**

- XMLHttpRequest 的 **responseText** 属性包含了从服务器发送的数据。它是一个HTML, XML或普通文本，这取决于服务器发送的内容。当 **readyState** 属性值变成 4 时, **responseText** 属性才可用，表明 Ajax 请求已经结束

```
function doSomething() {
    if(request.readyState == 4) {
        if(request.status == 200 || request.status == 304) {
            alert(request.responseText);
        }
    }
}
```

reponseXML

responseXML

- **responseXML**

- 如果服务器返回的是 XML，那么数据将储存在 **responseXML** 属性中。
- 只用服务器发送了带有正确首部信息的数据时，**responseXML** 属性才是可用的。MIME 类型必须为 **text/xml**

写服务器端的响应

写服务器端的响应

```
// 服务器端设置响应内容形式为纯文本形式
response.setContentType("text/plain;charset=utf-8");
// 服务器端创建输出流
PrintWriter out = response.getWriter();
// 向浏览器端发送数据
out.print(data);
out.flush();
out.close();
```

数据格式

数据格式

- 在服务器端 **AJAX** 是一门与语言无关的技术。在业务逻辑层使用何种服务器端语言都可以。
- 从服务器端接收数据的时候，那些数据必须以浏览器能够理解的格式来发送。服务器端的编程语言只能以如下 3 种格式返回数据：
 - XML
 - JSON
 - HTML

HTML

HTML

```
}
```

```
</script>
```

- HTML 由一些普通文本组成。如果服务器通过 XMLHttpRequest 发送 HTML，文本将存储在 responseText 属性中。
- 不必从 responseText 属性中读取数据。它已经是希望的格式，可以直接将它插入到页面中。
- 插入 HTML 代码最简单的方法是更新这个元素的 innerHTML 属性。
- 优点：
 - 从服务器端发送的 HTML 代码在浏览器端不需要用 JavaScript 进行解析。
 - HTML 的可读性好。
 - HTML 代码块与 innerHTML 属性搭配，效率高。
- 缺点：
 - 若需要通过 AJAX 更新一篇文档的多个部分，HTML 不合适

XML

XML

一种数据描述手段

老掉牙的东西，简单演示一下，不在这里浪费时间，基本现在的项目不用了。

淘汰的原因：数据冗余太多

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>People at Clearleft</title>

  <head>
    <link rel="stylesheet" href="../css/clearleft.css" />

```

JSON

JSON简介

JSON

- JSON (JavaScript Object Notation) 一种简单的数据格式，比 XML 更轻巧。JSON 是 JavaScript 原生格式，这意味着在 JavaScript 中处理 JSON 数据不需要任何特殊的 API 或工具包。
- JSON 的规则很简单：对象是一个无序的“‘名称/值’对”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“名称”后跟一个“：“（冒号）；“‘名称/值’对”之间使用“，”（逗号）分隔。

```
var jsonObject = {
  "name": "langqiao",
  "age": 30,
  "address": {"country": "china", "city": "beijing"}, I
  "teacher": function(){
    alert("javaee, UI");
  }
};
```

JSON

```
var jsonObject = {
    "name": "langiao",
    "age": 30,
    "address": {"country": "china", "city": "beijing"},
    "teacher": function(){
        alert("javaee,UI");
    }
};

window.onload = function() {
    alert(jsonObject.name);
    alert(jsonObject.address.country);
    jsonObject.teacher();
}
```

JSON.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>JSON数据格式</title>
    <script type="text/javascript">
        var jsonObject = {
            "name": "二狗",
            "age": 25,
            "address": {"county": "China", "city": "北京"},
            "teacher": function () {
                alert("JavaEE,UI");
            }
        };

        window.onload = function () {
            alert(jsonObject.name);
            alert(jsonObject.age);
            alert(jsonObject.address.county);
            alert(jsonObject.address.city);
            jsonObject.teacher();
        }
    </script>
</head>
<body>

</body>
</html>
```

JSON解析

JSON解析

- JSON只是一种文本字符串。它被存储在`responseText`属性中
- 为了读取存储在`responseText`属性中的JSON数据，需要根据JavaScript的`eval`语句。函数`eval`会把一个字符串当作它的参数。然后这个字符串会被当作JavaScript代码来执行。因为JSON的字符串就是由JavaScript代码构成的，所以它本身是可执行的

```
var jsonResponse = xhr.responseText;
var personObject = eval("(" + jsonResponse + ")");
var name = personObject.person.name;
var website = personObject.person.website;
var email = personObject.person.email;

• JSON提供了json.js包，下载http://www.json.org/json.js后，使用parseJSON()方法将字符串解析成JS对象

var jsonResponse = xhr.responseText;
var personObject = jsonResponse.parseJSON();
var name = personObject.person.name;
var website = personObject.person.website;
var email = personObject.person.email;
```

JSON事例

当请求数据时，返回一个json对象

JSON示例

```
if(request.status == 200 || request.status == 304) {
    var detailNode = document.getElementById("details");
    detailNode.innerHTML = "";
    // 获取返回的文本值
    var jsonResponse = request.responseText;
    // 使用eval()函数解析返回的json数据
    var parseJsonObj = eval("(" + jsonResponse + ")")
    // 获取json中的元素
    var name = parseJsonObj.language.name;
    var desc = parseJsonObj.language.desc;
    detailNode.innerHTML += "<h3>" + name + "</h3>";
    detailNode.innerHTML += "<p>" + desc + "</p>";
}
}
```

对比总结

- 若应用程序不需要与其他应用程序共享数据的时候，使用HTML片段来返回数据时最简单的
- 如果数据需要重用，JSON文件是个不错的选择，其在性能和文件大小方面有优势
- XML淘汰：数据冗余太多

JsonAjax.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>JsonAjax页面</title>
    <script type="text/javascript">
        // 产生XMLHttpRequest对象
        function createXMLHttpRequest() {
            var xmlreq = false;
            if(window.ActiveXObject){
                // for IE6 IE5
                xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
            }else if (window.XMLHttpRequest){
```

```

        xmlhttp = new XMLHttpRequest();
    }
    return xmlhttp;
}

window.onload = function () {

    //为button绑定事件
    document.getElementById("btn").onclick = function () {
        var xhr = createXMLHttpRequest();
        xhr.open("get","/jsonServlet");
        //相当于请求完成之后的回调处理函数
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4){
                //服务端响应数据(因后台传过来的是一个json字符串, 可以使用 eval 或 JSON.parse
                //进行解析; 如果后台传过来的是一个对象, 转换之后, 就是一个json对象; 如果是一个List, 转换之后, 是一个json数
                //组
                var respText = xhr.responseText;
                //对接收到的数据进行解析
                var jsonObject = eval("(" + respText + ")"); //得到的是一个数组
                var goodsName = jsonObject[0].goodsName;
                //将其放到 div 中
                document.getElementById("data").innerHTML = goodsName;
            }
        }
        xhr.send(null);
    }
}
</script>
</head>
<body>
    <input type="button" id="btn" value="获取json数据">
    <div id="data">

    </div>
</body>
</html>

```

jsonServlet类

```

@WebServlet( "/jsonServlet")
public class jsonServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/json");
        response.setCharacterEncoding("utf-8");
        PrintWriter out = response.getWriter();
    }
}

```

```

goodsservice goodsservice = new goodsserviceImpl();
//获取商品列表
List<Goods> goodsList = goodsservice.findAllGoods();
//将其转化为JSON字符串
String strJson = JSON.toJSONString(goodsList);
//将其输出到jsp页面
out.print(strJson);
//关闭流
out.close();

}
}

```

JsonAjax.jsp

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
<title>JsonAjax页面</title>
<script type="text/javascript">
// 产生XMLHttpRequest对象
function createXMLHttpRequest() {
    var xmlreq = false;
    if(window.ActiveXObject){
        // for IE6 IE5
        xmlreq = new ActiveXObject("Microsoft.XMLHTTP");
    }else if (window.XMLHttpRequest){
        xmlreq = new XMLHttpRequest();
    }
    return xmlreq;
}

window.onload = function () {

    //为button绑定事件
    document.getElementById("btn").onclick = function () {
        var xhr = createXMLHttpRequest();
        xhr.open("get", "/jsonServlet");
        //相当于请求完成之后的回调处理函数
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4){
                //服务端响应数据
                var respText = xhr.responseText;
                //因后台传过来的是一个json字符串，可以使用 eval 或 JSON.parse 进行
                //解析；如果后台传过来是一个对象，转换之后，就是一个json对象；如果是一个List，转换之后，是一个json数组
                var jsonObj = JSON.parse(respText);
                //对接收到的数据进行解析
                //var jsonObject = eval("(" + respText + ")"); //得到的是一个数组
                var inner = "";
                for (var i = 0 ;i < jsonObj.length ;i++){
                    inner += jsonObj[i].goodsName;
                }
            }
        }
    }
}

```

```

        document.getElementById("data").innerHTML = inner;
        /* var goodsName = jsonObject[0].goodsName;
        //将其放到 div 中
        document.getElementById("data").innerHTML = goodsName;*/
    }
}
xhr.send(null);
}
}
</script>
</head>
<body>
<input type="button" id="btn" value="获取json数据">
<div id="data">

</div>
</body>
</html>

```

Jquery之Ajax

异步请求，局部刷新

JQUERY之AJAX

- jQuery 中有一套专门针对 AJAX 的封装，功能十分完善，经常使用，需要着重注意。
- jQuery 对 Ajax 操作进行了封装，
 - 在 jQuery 中最底层的方法是 \$.ajax()
 - 第二层是 load(), \$.get() 和 \$.post()
 - 第三层是 \$.getScript() 和 \$.getJSON()

Ajax常用方法

load()方法

jsonServlet类

JSONload.jsp

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
<title>Jquery之Ajax</title>
<script src="js/jquery-1.12.4.js"></script>
<script type="text/javascript">
$(function () {
    //绑定事件 url地址/ 传递参数 / 回掉函数 : 回传参数
    $("#btn").load("/jsonServlet", function (data) {
        //alert(data);
    })
})

```

```
//将响应数据进行解析
var respData = eval("(" + data + ")");
//进行循环展示 两个参数

/*$(respData).each(function (index,item) {
    //将其在div中进行展示 追加
    $("#show").append(index + "---" + item.goodsName +<br>)
})*/
$.each(respData,function (index,item) {
    //将其在div中进行展示 追加
    $("#show").append(index + "---" + item.goodsName +<br>)
})

})
}

</script>
</head>
<body>
<input type="button" id="btn" value="load方式">
<div id="show">

</div>
</body>
</html>
```

\$.ajax

```
$.ajax

$.ajax({
    url: './get.php',
    type: 'get',
    dataType: 'json',
    data: { id: 1 },
    beforeSend: function (xhr) {
        console.log('before send')
    },
    success: function (data) {
        console.log(data)
    },
    error: function (err) {
        console.log(err)
    },
})
```

```
$.ajax  
  console.log('request completed')  
}  
})
```

常用选项参数介绍：

url: 请求地址

type: 请求方法，默认为 get

dataType: 服务端响应数据类型

contentType: 请求体内容类型，默认 application/x-www-form-urlencoded

data: 需要传递到服务端的数据，如果 GET 则通过 URL 传递，如果 POST 则通过请求体传递

timeout: 请求超时时间

beforeSend: 请求发起之前触发

success: 请求成功之后触发（响应状态码 200）

error: 请求失败触发

complete: 请求完成触发（不管成功与否）

.get()方法 (或 .post() 方法)

jQuery.get(url, [data], [callback], [type])

返回值:XMLHttpRequest

概述

通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 `$.ajax`。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 `$.ajax`。

参数

url,[data],[callback],[type]

String,Map,Function,String

V1.0

url:待载入页面的URL地址

data:待发送 Key/value 参数。

callback:载入成功时回调函数。

type:返回内容格式, `xml`, `html`, `script`, `json`, `text`, `_default`.

示例



描述:

请求 test.php 网页，忽略返回值。

jQuery 代码:

```
$.get("test.php");
```

描述:

请求 test.php 网页，传递2个参数，忽略返回值。

jQuery 代码:

```
$.get("test.php", { name: "John", time: "2pm" } );
```

`$.get()`

```
<html>
  <head>
    <title>Get</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script type="text/javascript">
      $(function () {
        $("#btn").click(function () {
          $.get(
            url,
            data,
            type,
            callback
          )
        })
      })
    </script>
  </head>
  <body>
```

get.jsp(发送get请求)

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>发送get请求</title>
  <script src="js/jquery-1.12.4.js"></script>
  <script type="text/javascript">
    /*
    $(function () {
      $("#btn").click(function () {
        $.get(
          url,地址
          data,传递参数
          type,返回方式
          callback;回掉函数
        )
      })
    })
    */
    $(function () {
      //给button绑定点击事件
    })
  </script>
</head>
<body>
```

```

$( "#btn" ).click(function () {
    //在每次发送请求之前先进行清空
    $("#show").empty();
    //当点击的时候触发get请求，请求到Servlet，然后将响应的数据在div(其他容器)中循环展示
    $.get(
        // url地址
        "jsonServlet",
        //请求参数
        {id:"123"},
        //回调函数
        function (data) { //回调函数中携带参数data，本身为一个json数组，无需处理直接
            输出
            $.each(data,function (index,item) {
                //将其在div中进行展示 追加
                $("#show").append(index + "---" + item.goodsName + "<br>");
            })
        },
        "json" //作用：将回调函数中数据：data 进行解析
    )
})

```

</script>

</head>

<body>

<input type="button" id="btn" value="发送get请求">

<div id="show">

</div>

</body>

</html>

\$post()

异步提交；局部刷新，页面不会刷新

get.jsp(发送post请求)

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>发送get请求</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script type="text/javascript">

        $(function () {
            $("#btn").click(function () {
                $.post(
                    "jsonServlet",
                    {id:"123"},
                    function (data) {
                        $.each(data,function (index,item) {
                            $("#show").append(index + "---" + item.goodsName + "<br>");
                        })
                    }
                )
            })
        })
    </script>
</head>
<body>
    <input type="button" id="btn" value="发送post请求">
    <div id="show">
        </div>
    </body>
</html>

```

```
        })
    },
    "json"
)
})
}
</script>
</head>
<body>
<input type="button" id="btn" value="发送get请求">
<div id="show">

</div>
</body>
</html>
```

应用

应用

The screenshot illustrates a comment submission process. On the left, a form has '姓名' (Name) and '内容' (Content) fields, and a '提交' (Submit) button. A yellow circle highlights the '提交' button. An arrow points to the right, leading to the results on the right. The results show a '评论' (Comment) section with '姓名: 小芳' (Name: Xiao Fang), '内容: 我是小芳' (Content: I am Xiao Fang), and a '提交' (Submit) button. Below this is a '已有评论:' (Existing Comments:) section containing 'Tom:' and 'I am Tom'. At the bottom left, the rendered HTML code is shown:

```
<div class="comment">
<h6>Tom: </h6>
```

serialize()方法

应用

验证日期

验证email

动态加载列表框

跨域

