web2.0

Ajax编程详解

刘新福
allanlxf@hotmail.com

Contents

1、Ajax概述

1. Ajax简介
2. Ajax的技术构成
3. Ajax核心编程

1.1、Ajax简介

什么是Ajax

Ajax并不是一种新的技术，而是一种新的Web应用开发模型。在Ajax应用中，客户端使用HTML、JavaScript、CSS及DOM技术现实同Web服务器的即时通信，并在不重新装载页面的前提下更新页面中的信息，即无抖动更新。Ajax使Web应用兼具B/S和C/S结构的应用的特点，也被视作基于web2.0的应用。

传统的Web应用流程

1. 客户端在浏览器上输入资源的URL。
2. 服务器解析URL，并执行某个服务器端程序，动态生成HTML页面。
3. 客户端在页面中输入信息，并提交表单，并等待服务器响应。
4. 服务器处理表单的数据，并根据处理结果向客户端发送下一个页面。
5. 客户端得到页面之后，继续输入信息...

在上述流程中，客户端必须等待当前请求被服务器处理完毕才可以发送下一个请求，并且在等待时，客户端将无事可做。因为浏览器进程被阻塞，在下一个页面生成之前，当前用户窗口中是一片空白。传统的Web应用的缺点在于：

1. 系统交互性比较差，不能即时得到响应。
2. 操作效率低下，因为用户在进行下一个请求之前必须等待响应。

1.1、Ajax简介

❖ Ajax Web应用流程

1. 客户端在浏览器上输入资源的URL。
2. 服务器解析URL，并执行某个服务器端程序，动态生成HTML页面。
3. 客户端在页面中输入信息，随时以异步的方式发送数据到服务器端处理，客户端可以继续执行其它的操作。
4. 服务器处理表单的数据，并根据处理结果向客户端发数据。
5. 客户端得到数据之后，更新页面中的局部内容，此时不会影响到客户端的正常操作。
6. 客户端继续输入信息…。

Ajax Web应用区别与传统的Web应用在于，用户在发出请求之后可以继续执行客户端的操作而不必等待服务器的响应。当服务器响应完成时会自动调用客户端程序完成页面的更新，这个操作也不会阻塞客户的当前操作。

比如在用户注册功能中，用户输入完用户名之后立刻发送请求到服务器端进行是否重复的验证，而用户不必等待验证结果可以继续输入密码、性别等其它信息，一旦服务器验证完毕后，会将验证结果自动显示到客户页面中。在整个过程中没有任何的页面刷新与阻塞。

1.2、Ajax的技术构成

- **HTML**

  用来构建Web表单并确定应用程序其它部分使用的元素。

- **JavaScript**

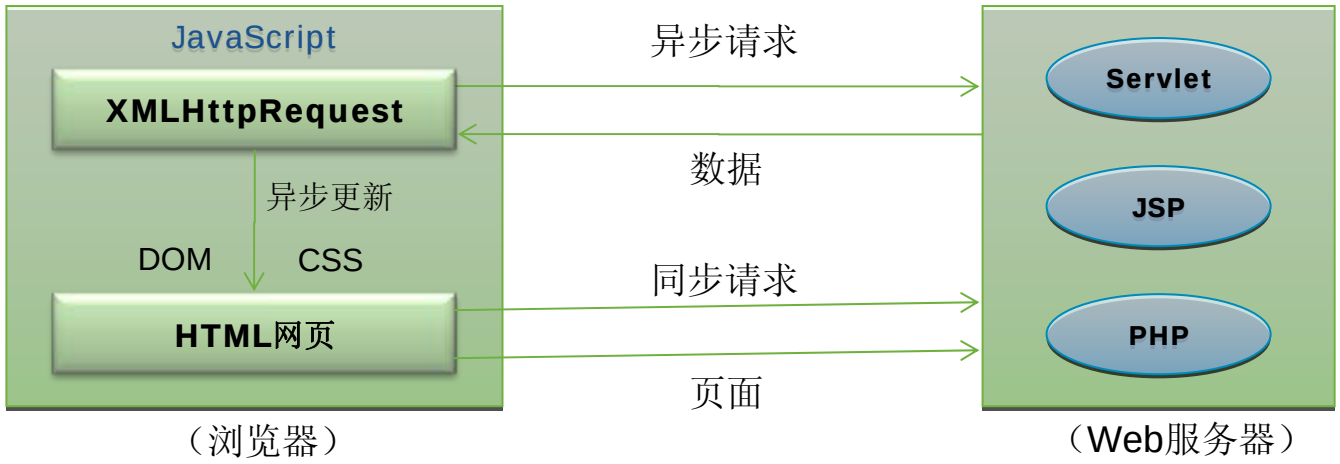
  Ajax 应用程序的核心代码，用来实现与服务器端应用程序的异步交互。

- **DOM**

  文档对象模型，用于（使用JavaScript技术）处理HTML结构和服务器返回的XML。

- **DHTML**

  用于在客户端动态更新表单的内容及样式(使用CSS)。



1.3、Ajax编程流程

- **Ajax编程的基本步骤如下：**

1. 创建XMLHttpRequest对象
2. 从Web页面中获取需要处理的数据
3. 打开到服务器的连接
4. 设置处理响应数据的JavaScript回调函数
5. 发送请求及请求数据

- **基于Ajax的即时验证程序**

- 主体功能及流程
  1. 用户访问服务器得到注册页面。
  2. 用户在注册页面中输入用户名、密码及电子邮件信息。
  3. 用户点击确定按钮，系统将注册信息保存到数据库中。
  4. 系统向用户显示注册的结果。
- 异常流程
  1. 在用户输入完用户名后，系统应该尽快提示该用户是否可用，而不必等到页面提交之后报错。

1.3、register.jsp(html)


```
<%@ page contentType="text/html;charset=gbk"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>用户注册</title>
    <link type="text/css" rel="stylesheet"
     href="${pageContext.request.contextPath}/style/register.css"></link>
    <script type="text/javascript“>
        //参见后面
    </script>
  </head>
  <body>
    <center>
    <div id="registerDiv">
    <h3>请输入您的注册信息</h3>
    <hr>
```

设置响应内容及页面编码格式

引入外部CSS，${pageContext.request.contextPath}为JSP EL，用来动态获取当前页面所在应用的路径。

1.3、register.jsp(html)


```
<form id="registerForm" name="registerForm">
<table>
   <tr>
     <td class="heading“><label for="userName">用户名</label></td>
     <td>
       <input type="text" name="userName" id="userName"
onblur="validateUserName(this.value)" onfocus="clearErrors()">
       <span id="message" class="hint"></span>
     </td>
   </tr>
   <tr>
     <td class="heading“><label for="password">密码</label></td>
     <td>
       <input type="password" name="password" id="password"
style="width:150">
     </td>
   </tr>
```

控件失去焦点时发送异步的请求来验证用户名是否可用。

显示用户名是否可用的提示区

控件获得焦点时，清空错误提示内容。

1.3、register.jsp(html)


```
      <tr>
        <td class="heading“><label for="email">电子邮件</label></td>
        <td>
          <input type="text" name="email" id="email">
        </td>
      </tr>
    </table>
    <br>
    <input type="button" value="注册" onclick="addUser(this.form)">
  </form>
  </div>
  <div id="successDiv" style="display:none;text-align:center">
    恭喜您，注册成功，请<a href="#">登录</a>!
  </div>
  </center>
 </body>
</html>
```

点击按钮，并不会提交表单，而是以异步的方式发送请求来增加用户的信息。

注册成功提示内容，只有注册成功之后才会显示。

1.3、register.jsp(JavaScript)


```
<script type="text/javascript">
    var client = null;
    function validateUserName(userName){
      client = getXMLHttpRequest();
      var requestURL = "${pageContext.request.contextPath}/user/
                                validateUserName.ajax?userName=" + userName;
      client.open("GET", requestURL, true);
      client.onreadystatechange = showMessage;
      client.send(null);
    }
```

XMLHttpRequest是一个JavaScript对象，是ajax编程体系中和服务器通信的核心客户端。遗憾的是不同的浏览器提供了不同的创建该对象的办法，主要可以分为IE和其它浏览器(比如firefox等)两种情况。

构造请求的URL，并附加参数信息，其中的EL表达式为动态获取应用的路径。

以GET方式打开请求，访问方式为异步模式，即发送请求后客户端不必等待响应，可以继续执行操作。

发送请求，由于GET请求参数信息包含到URL中，因此传入null。

设置处理响应的回调函数，值为一个JS函数对象。

1.3、register.jsp(JavaScript)


```
function getXMLHttpRequest(){
  var client = getXMLHttpRequestFromIE();
   if(client == null){
       client = new XMLHttpRequest();
   }
  return client;
}
function getXMLHttpRequestFromIE(){
  var namePrefixes = ["Msxml3","Msxml2","Msxml","Microsoft"];
  for(var i = 0; i < namePrefixes.length; i++){
      try{
         var name = namePrefixes[i] + ".XMLHTTP";
         return new ActiveXObject(name);
      }catch(e){
      }
  }
  return null;
}
```

在其它浏览器中，规定了类型为XMLHttpRequest的对象，因此可以直接使用new来创建对象。

在IE中，XMLHttpRequest的实现为一个ActiveX控件，该控件的名称在不同的版本中会有所不同，总结起来包括：Microsoft.XMLHTTP、 msxml.XMLHTTP、msxm2.XMLHTTP或msxml3.XMLHTTP

1.3、register.jsp(JavaScript)


```
function showMessage(){
  if(client.readyState == 4){
    if(client.status == 200){
      var result = client.responseText;
      if(result == "inuse"){
        var message = "用户名：";
        message += document.getElementById("userName").value;
        message += "已经被使用！";
        document.getElementById("message").innerHTML = message;
      }
    }else{
      showErrors(client.status);
    }
  }
}
```

验证用户名是否可用的回调方法，在另外的线程中被执行。

readyState=4表示响应已经完成，可以安全地得到响应的内容。
status=200表示服务器端程序成功执行，没有遇到错误的情况。

得到服务器端的响应文本，我们在应用中约定如果用户名在系统中已经存在，返回字符串inuse，否则返回其它信息。

调试功能，输出服务器端出错时返回的错误状态码。

1.3、register.jsp(JavaScript)


```
function addUser(form){
  client = getXMLHttpRequest();
  var requestURL = "${pageContext.request.contextPath}/user/add.ajax";
  client.open("POST", requestURL, true);
  client.onreadystatechange = showResult;
  var data = "";
  for(var i = 0; i < form.elements.length; i++){
    var element = form.elements[i];
    if(element.name.length > 0){
      if(data.indexOf("=") != -1){
        data += "&";
      }
      data += element.name + "=“ + element.value;
    }
  }
  client.send(data);
}
```

增加新用户信息的回调方法，在另外的线程中被执行。

将form表单中的有名称的控件和它们的值以参数的形式发往服务器端。这样在服务器端可以使用request.getParameter()或request.getReader()方法来获取这些信息。

POST请求中，数据通过请求的主体发送到服务器端。

1.3、register.jsp(JavaScript)


```
function showResult(){
  if(client.readyState == 4){
    if(client.status == 200){
      var successDiv = document.getElementById("successDiv");
      successDiv.style.display = "inline";
      var registerDiv = document.getElementById("registerDiv");
      registerDiv.style.display = "none";
    }else{
      showErrors(client.status);
    }
  }
}
```

显示successDiv的内容

隐藏registerDiv的内容

1.3、register.jsp(JavaScript)


```
function showErrors(responseStatus){
  var consoleDiv = document.getElementById("console");
  if(consoleDiv == null){
    consoleDiv = document.createElement("div");
    document.body.appendChild(consoleDiv);
    consoleDiv.id = "console";
    consoleDiv.className = "console";
    consoleDiv.style.width = "300px";
    consoleDiv.style.height = "50px";
    consoleDiv.style.margin = "0 auto";
    consoleDiv.style.border = "1px solid #DDD";
    consoleDiv.style.marginLeft = "20px";
  }
  consoleDiv.style.display = "inline";
  var errorMessage = "执行操作时发生错误，错误码为：";
  errorMessage += responseStatus;
  consoleDiv.innerHTML = errorMessage;
}
```

使用DHTML、DOM及CSS技术在页面的末尾动态追加div控件并在其中显示指定的错误提示信息。

1.3、register.jsp(JavaScript)


```
  function clearErrors(){
    var consoleDiv = document.getElementById("console");
    if(consoleDiv != null){
      consoleDiv.innerHTML = "";
      consoleDiv.style.display = "none";
    }
  }
</script>
```

清空错误提示控件的内容并将其从页面中隐藏。

1.3、web.xml


```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
          http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd“>
    <servlet>
        <servlet-name>AjaxHandlerServlet</servlet-name>
        <servlet-class>allan.ajax.action.AjaxHandlerServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AjaxHandlerServlet</servlet-name>
        <url-pattern>*.ajax</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

使AjaxHandlerServlet服务于所有以.ajax结尾的请求。

1.3、AjaxHandlerServlet.java


```
package allan.ajax.action;

import java.io.IOException;
import java.io.PrintWriter;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;
import javax.servlet.*;
import javax.servlet.http.*;
import allan.ajax.service.UserService;

public class AjaxHandlerServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException {
        execute(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException {
        execute(request, response);
    }
```

使AjaxHandlerServlet服务于所有以.ajax结尾的请求。并且每个请求由该Servlet的某个方法处理，这样做的好处在于，以后每增加一个新的功能，不必写新的Servlet，只需在本Servlet中增加新的处理方法即可。

1.3、Ajax编程流程


```
public void execute(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
    String path = request.getServletPath();
    path = path.substring(0, path.lastIndexOf("."));
    String handler = getHandlerMapping().get(path);
    if (handler == null) {
        response.sendError(HttpServletResponse.SC_NOT_FOUND, "Unknown path:" + path);
        return;
    }

    try {
        Class[] parameterTypes = new Class[] { HttpServletRequest.class,
                                                    HttpServletResponse.class };
        Method method = this.getClass().getDeclaredMethod(handler, parameterTypes);
        method.invoke(this, request, response);
    } catch (Exception e) {
        e.printStackTrace();
        throw new ServletException(e);
    }
}
```

得到请求路径中的Servlet路径部分并去掉后缀.ajax，然后根据结果从handlerMapping中查找处理方法

如果没有找到处理方法，向客户端报告错误码404，在客户端可以使用XMLHttpRequest的status属性得到该值。

使用反射机制调用方法，该方法的参数为HttpServletRequest和HttpServletResponse

如果方法调用失败，向客户端报告错误码500，在客户端可以使用XMLHttpRequest的status属性得到该值。

1.3、Ajax编程流程


```
protected Map<String, String> getHandlerMapping() {
    Map<String, String> handlers = new HashMap<String, String>();
    handlers.put("/user/validateUserName", "validateUserName");
    handlers.put("/user/add", "addUser");
    return handlers;
}
```

返回包含了请求路径和处理方法的对应关系的Map对象。每增加一个新的功能需要在该Map中增加一个键-值的对象关系。其中key为请求的路径，value为处理该请求的方法名称。

1.3、Ajax编程流程


```
protected void validateUserName(HttpServletRequest request,
                                HttpServletResponse response)  throws ServletException, IOException {
    response.setContentType("text/plain");
    PrintWriter out = response.getWriter();

    String userName = request.getParameter("userName");

    UserService userService = new UserService();
    boolean exist = false;
    try {
        exist = userService.isUserNameInUser(userName);
    } catch (Exception e) {
        e.printStackTrace();
        throw new ServletException(e);
    }

    if (exist) {
        out.print("inuse");
    } else {
        out.print("good name");
    }
    out.close();
}
```

设置响应内容为纯文本

读取请求的参数，取得需要验证的用户名

执行业务处理类验证用户名是否已经被使用

如果执行业务时发生错误，抛出ServletException，服务器会报告500状态码给客户端。

以文本形式返回的业务结果，inuse表示已经被使用，其它的表示尚未被使用，在客户端可以使用XMLHttpRequest的responseText属性来得到该信息。

1.3、Ajax编程流程


```
protected void addUser(HttpServletRequest request,
                       HttpServletResponse response)  throws ServletException, IOException {
    User user = createUserFromRequest(request);
    UserService userService = new UserService();
    try {
        userService.add(user);
    } catch (Exception e) {
        e.printStackTrace();
        throw new ServletException(e);
    }
}

protected User createUserFromRequest(HttpServletRequest request) {
    String userName = request.getParameter("userName");
    String password = request.getParameter("password");
    String email = request.getParameter("email");
    User user = new User();
    user.setUserName(userName);
    user.setPassword(password);
    user.setEmail(email);
    return user;
}
}
```

执行完操作之后，没有任何的响应主体内容，客户端可以通过响应的状态码来判断操作的是否成功。

在开发时，可以使用反射机制或apache common tookit或OGNL来编写通用的赋值方法。

2、发送请求与处理响应

1. 发送GET及POST请求
2. 生成响应头与响应主体内容

2.1、发送GET及POST请求

关于Ajax请求

Ajax请求，即使用XMLHttpRequest对象发送的请求与浏览器发送的请求并没有本质上的区别，都是基于HTTP协议的请求。在HTTP协议中规定了多种请求类型，包括：GET、POST、HEAD、PUT、DELETE等。从应用的角度来讲，比较常用的包括GET请求和POST请求。

GET请求

GET请求的主要用途是从指定的服务器中获取资源。在GET请求中，通常只需指定资源的路径即可。如果请求的是一个动态的资源，比如JSP、PHP、CGI等，可以在请求的路径后面附件查询的参数信息。以便程序可以根据该参数查询更为具体的信息。附件参数的办法如下：

请求的路径?名称1=值1&名称2=值2&名称3=值3...

在服务器端可以使用request.getQueryString()方法返回?后面的整个字符串，也可以使用request.getParameter(名称)返回某个值。

2.1、发送GET及POST请求

POST请求

POST请求的主要用途是向服务器发送信息。在POST请求中，参数信息并不是通过URL来传递的，而是在请求的主体中，这部分信息用户无法看见并且没有长度的限制。请求主体中的参数的格式一般为：

名称1=值1&名称2=值2&名称3=值3...

需要注意的是，为了通知服务器端请求主体内容为表单中的参数信息，需要调用XMLHttpRequest的方法来设置请求头，否则将无法取到参数。

```
setRequestHeader("Content-Type"," application/x-www-form-urlencoded");
```

在服务器端可以使用request.getReader()方法以流的形式得到这些信息，也可以使用request.getParameter(名称)返回某个值。

使用GET还是POST

GET请求和POST请求都可以访问服务器端的程序，并且都可以向程序传递信息。一般来讲，没有严格的规定某个请求必须使用GET或POST来完成。

如果考虑到信息的安全性，应该使用POST请求，因为GET请求所发送的信息显示在URL中；如果要发送的信息量比较大，应该使用POST请求，因为GET请求有长度的限制，而POST请求没有；如果要发送非文本内容，比如文件上传等必须使用POST请求。

2.1、发送GET及POST请求

发送Ajax请求

在Ajax应用中，请求的发送是通过XMLHttpRequest对象的open和send方法来协同完成的。通常在open方法中指定请求的类型，在send方法中发送数据。比如，下述代码向服务器发送GET请求：

```
var client = getXMLHttpRequest();
var url = “/ajax-app/request/get.ajax?name=管理员&grade=1”;
cient.open(“GET”, url, true);
client.send(null);
```

其中name=管理员&grade=1为请求的参数，send方法的参数为null是因为GET请求的数据都附加到了URL中。要发送POST请求，可以使用如下的代码：

```
var client = getXMLHttpRequest();
var url = “/ajax-app/request/post.ajax”;
cient.open(“POST”, url, true);
setRequestHeader(“Content-Type”,” application/x-www-form-urlencoded”);
client.send(“name=管理员&grade=1”);
```

在Ajax应用中，请求不是通过浏览器发送的，因此GET请求的参数不会显示到浏览器的地址栏上，开发者不必担心安全的问题。另外处于对本地系统的安全考虑JavaScript不能访问客户端本地的文件等资源，因此不能使用Ajax进行文件的上传。

2.1、发送GET及POST请求

因此，开发者只需考虑需要传递的信息量的大小来决定使用什么样的请求类型，即如果数据量不大使用GET请求，否则使用POST请求。还有，如果所有请求的服务器端程序只支持某种类型的请求，我们就没有选择的余地了。

- **关于请求数据的字符集**

如果在请求中包含非iso8859_1字符集中的字符，比如中文字符的话，开发者必须要考虑字符编码转换的问题。在具体处理时，分GET和POST两种情况。

- GET

GET请求的参数出现在URL中，需要在服务器端对URL进行字符集的转码处理，这个工作通常通过修改服务器的配置来完成。在Tomcat中，可以在其安装目录的conf下的server.xml中指定，做法如下：

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" URIEncoding="GBK"/>
```

- POST

POST请求的参数通过请求的主体来传送，编码转换的方式与GET有所不同。

2.1、发送GET及POST请求

在获取POST请求的参数之前，只要调用HttpServletRequest对象的setCharacterEncoding(charset)方法进行字符集转换即可。在Ajax请求中，使用的传输字符集为UTF-8，因此我们需要做如下的处理：

```
request.setCharacterEncoding("utf-8");
String name = request.getParameter("name");
String gender = request.getParameter("gender");
```

在开发时，我们可以将上述转换代码放在一个Servlet Filter中统一完成。示意代码如下：

```
package alan.web.util;
public class CharacterEncodingFilter implements Filter{
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws ServletException, IOException{
        request.setCharacterEncoding(“UTF-8”);
        chain.doFilter(request, response);
    }
    public void init(FilterConfig config){}
    public void destroy(){}
}
```

2.1、发送GET及POST请求

❖ 关于请求头和主体信息

在HTTP协议中规定客户端向服务器端发送的信息分为两个部分，请求头和请求的主体。其中主体信息通常是发给服务器端的处理程序处理的数据，这是请求的核心数据部分。而请求头部信息用来传递一些对服务器及处理程序有用的附加信息，比如请求的字符集、客户端的类型等，这有助于服务器及处理程序能更好地处理主体数据。

在Ajax应用中，使用XMLHttpRequest对象可以发送请求头及请求主体信息。头部信息使用它的setRequestHeader(name, value)方法发送。比如：

```
var client = getXMLHttpRequest();
client.open(method, url, true);
client.setRequestHeader(“Content-Type”,” application/x-www-form-urlencoded”);
client.send(data);
```

主体信息可以通过URL的附加参数或通过XMLHttpRequest对象的send方法发送，具体处理请参见前面的说明。

2.2、生成响应头与响应主体内容

关于Ajax响应

Ajax响应也是HTTP响应，开发者只需注意接收响应内容的客户端不是浏览器而是XMLHttpRequest对象即可。

在HTTP协议中规定服务器向客户端发送的信息分为两个部分，响应头和响应的主体。其中主体信息通常是发给客户端的处理程序处理的数据，这是响应的核心数据部分。而响应头部信息用来传递一些对客户端程序有用的附加信息，比如响应内容的字符集、服务器的类型等，这有助于客户端处理程序能更好地处理主体数据。

如果服务器端采用Servlet技术的话，可以使用HttpServletResponse对象来生成响应。要生成响应头信息，可以使用：

- setHeader(String name, String value)
- setIntHeader(String name, int value)
- setDateHeader(String name, long date)
- addHeader(String name, String value)
- setIntHeader(String name, int value)
- setDateHeader(String name, long date)

2.2、生成响应头与响应主体内容

在上述方法中，set(Int|Date)Header方法只设置一个值，如果给定名称的响应头已经存在，使用新的值覆盖原来的值；而add(Int|Date)Header方法运行时一个名称和多个值关联。下面的代码用来防止浏览器缓存响应的内容：

```
response.setHeader(“Cache-Control”, “no-cache”);
response.setHeader(“Pragma”, “no-cache”);
response.setDateHeader(“Expires”, 0);
```

要响应的主体内容可以使用如下的代码：

```
PrintWriter out = response.getWriter();
out.print(“内容”);
```

在Ajax客户端，可以使用XMLHttpRequest的如下方法获取响应头的信息：

- getAllRequestHeaders()
- getRequestHeader(name)

可以使用如下属性获取响应主体的信息：

- responseText
- responseXML

2.2、生成响应头与响应主体内容

❖ 响应的状态码

服务器端程序可以通过响应的状态码来通知客户端当前请求的处理情况。在HTTP协议中规定了如下一些常用的状态码：

200 – OK，一切顺利。

302 – 请求被重定义。

403 – 请求被禁止。

404 – 请求的资源不存在。

405 – 请求的方法不支持。

500 – 服务器内部错误，通常为程序执行过程中抛出异常。

在Servlet中，可以使用HttpServletResponse对象的setStatus(int code)或sendError(int code)来发送状态码。

当Servlet的服务器方法doGet和doPost运行时抛出异常时，服务器会自动在响应中设置值为500的状态码。

在Ajax客户端可以使用XMLHttpRequest对象的status属性得到响应的状态码的信息，比如：

2.2、生成响应头与响应主体内容

❖ 响应内容的字符集

服务器端程序可以通过响应的状态码来通知客户端当前请求的处理情况。在HTTP协议中规定了如下一些常用的状态码：

在Servlet中，可以使用HttpServletResponse对象的setStatus(int code)或sendError(int code)来发送状态码。

当Servlet的服务器方法doGet和doPost运行时抛出异常时，服务器会自动在响应中设置值为500的状态码。

在Ajax客户端可以使用XMLHttpRequest对象的status属性得到响应的状态码的信息，比如：

3、Ajax数据传输格式

1. 纯文本格式
2. XML格式
3. Json格式

4、XMLHttpRequest详解

1. 创建XMLHttpRequest对象
2. XMLHttpRequest的属性
3. XMLHttpRequest的方法

4.1、创建XMLHttpRequest对象

XMLHttpRequest

XMLHttpRequest是一个JavaScript对象，是ajax编程体系中和服务器通信的核心客户端。遗憾的是不同的浏览器提供了不同的创建该对象的办法，主要可以分为IE和其它浏览器(包括firefox,sof)两种情况。

- 在IE中，XMLHttpRequest的实现为一个ActiveX控件，该控件的名称在不同的版本中会有所不同，总结起来包括：Microsoft.XMLHTTP、msxml.XMLHTTP、msxm2.XMLHTTP或msxml3.XMLHTTP

```
function getXMLHttpRequestFromIE(){
    var namePrefixes = [“Msxml3”,”Msxml2”,”Msxml”,”Microsoft”];
    for(var i = 0; i < namePrefixes.length; i++){
        try{
            var name = namePrefixes[i] + “.XMLHTTP”;
            return new ActiveXObject(name);
        }catch(e){
        }
    }
    return null;
}
```

4.1、创建XMLHttpRequest对象

- 在其它浏览器中，规定了类型为XMLHttpRequest的对象，因此可以直接使用new来创建对象。

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

- 考虑所有浏览器，代码如下：

```
function getXMLHttpRequest(){
    var client = getXMLHttpRequestFromIE();
     if(client == null){
          client = new XMLHttpRequest();
     }
    return client;
}
```

4.2、XMLHttpRequest的属性

名称	类型	说明
onreadystatechange	function	用来处理响应的客户端函数对象
readyState	number	HTTP请求就绪状态值，代表请求的不同阶段 0：请求还没有发出，即调用open之前 1：请求以建立，但还没有发送，即调用send之前 2：请求已经发出，正在处理中 3：请求已经处理，但服务器还没有完成响应 4：请求已经处理完毕，全部响应内容都可用
status	number	HTTP响应的状态码
statusText	DOMString	HTTP响应的状态码的文本描述
responseText	DOMString	以文本格式得到响应的内容
responseXML	Document	以XML格式得到响应的内容，此时响应类型必须为text/xml,application/xml或以xml结尾。

4.3、XMLHttpRequest的方法

签名	返回值	说明
open(method,url,async)	void	打开请求，并使request对象处于未发送状态。
send(data)	void	发送请求及数据到服务器
setRequestHeader(name,value)	void	设置HTTP请求头信息，该方法必须在open及send之间调用。
getAllResponseHeaders()	DOMString	得到HTTP响应的所有头部信息。
getResponseHeader(name)	DOMString	得到HTTP响应的某个头部信息。
abort	void	终止请求

5、Ajax库及框架介绍

1. Prototype库
2. DWR框架
3. Dojo工具库