# AJAX

## AJAX概述

### 1　什么是AJAX

AJAX（Asynchronous Javascript And XML）翻译成中文就是“异步Javascript和XML”。即使用Javascript语言与服务器进行异步交互，传输的数据为XML（当然，传输的数据不只是XML）。

AJAX还有一个最大的特点就是，当服务器响应时，不用刷新整个浏览器页面，而是可以局部刷新。这一特点给用户的感受是在不知不觉中完成请求和响应过程。

* 与服务器异步交互；
* 浏览器页面局部刷新；

### 2.　同步交互与异步交互

* 同步交互：客户端发出一个请求后，需要等待服务器响应结束后，才能发出第二个请求；
* 异步交互：客户端发出一个请求后，无需等待服务器响应结束，就可以发出第二个请求。

### 3.　AJAX常见应用情景



当我们在百度中输入一个“传”字后，会马上出现一个下拉列表！列表中显示的是包含“传”字的10个关键字。

其实这里就使用了AJAX技术！当文件框发生了输入变化时，浏览器会使用AJAX技术向服务器发送一个请求，查询包含“传”字的前10个关键字，然后服务器会把查询到的结果响应给浏览器，最后浏览器把这10个关键字显示在下拉列表中。

* 整个过程中页面没有刷新，只是刷新页面中的局部位置而已！
* 当请求发出后，浏览器还可以进行其他操作，无需等待服务器的响应！



当输入用户名后，把光标移动到其他表单项上时，浏览器会使用AJAX技术向服务器发出请求，服务器会查询名为zhangSan的用户是否存在，最终服务器返回true表示名为zhangSan的用户已经存在了，浏览器在得到结果后显示“用户名已被注册！”。

* 整个过程中页面没有刷新，只是局部刷新了；
* 在请求发出后，浏览器不用等待服务器响应结果就可以进行其他操作；

### 4　AJAX的优缺点

优点：

* AJAX使用Javascript技术向服务器发送异步请求；
* AJAX无须刷新整个页面；
* 因为服务器响应内容不再是整个页面，而是页面中的局部，所以AJAX性能高；

缺点：

* AJAX并不适合所有场景，很多时候还是要使用同步交互；
* AJAX虽然提高了用户体验，但无形中向服务器发送的请求次数增多了，导致服务器压力增大；
* 因为AJAX是在浏览器中使用Javascript技术完成的，所以还需要处理浏览器兼容性问题；

## AJAX技术

四步操作：

* 创建核心对象；
* 使用核心对象打开与服务器的连接；
* 发送请求
* 注册监听，**监听服务器响应。**

XMLHTTPRequest

* open(请求方式, URL, 是否异步)
* send(请求体)
* onreadystatechange，指定监听函数，它会在xmlHttp对象的状态发生变化时被调用
* readyState，当前xmlHttp对象的状态，其中4状态表示服务器响应结束
* status：服务器响应的状态码，只有服务器响应结束时才有这个东东，200表示响应成功；
* responseText：获取服务器的响应体

### 1　AJAX第一例

#### 1.1 准备工作

因为AJAX也需要请求服务器，异步请求也是请求服务器，所以我们需要先写好服务器端代码，即编写一个Servlet！

这里，Servlet很简单，只需要输出“Hello AJAX!”。

|  |
| --- |
| **public** **class** AServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println("Hello AJAX!");  response.getWriter().print("Hello AJAX!");  }  } |

#### 1.2　AJAX核心（XMLHttpRequest）

其实AJAX就是在Javascript中多添加了一个对象：XMLHttpRequest对象。所有的异步交互都是使用XMLHttpServlet对象完成的。也就是说，我们只需要学习一个Javascript的新对象即可。

注意，各个浏览器对XMLHttpRequest的支持也是不同的！大多数浏览器都支持DOM2规范，都可以使用：**var xmlHttp = new XMLHttpRequest()**来创建对象；但IE有所不同，IE5.5以及更早版本需要：**var xmlHttp = new ActiveXObject(“Microsoft.XMLHTTP”)**来创建对象；而IE6中需要：**var xmlHttp = new ActiveXObject(“Msmxl2.XMLHTTP”)**来创建对象；而IE7以及更新版本也支持DOM2规范。

为了处理浏览器兼容问题，给出下面方法来创建XMLHttpRequest对象：

|  |
| --- |
| **function** createXMLHttpRequest() {  **var** xmlHttp;  // 适用于大多数浏览器，以及IE7和IE更高版本  **try**{  xmlHttp = **new** XMLHttpRequest();  } **catch** (e) {  // 适用于IE6  **try** {  xmlHttp = **new** ActiveXObject("Msxml2.XMLHTTP");  } **catch** (e) {  // 适用于IE5.5，以及IE更早版本  **try**{  xmlHttp = **new** ActiveXObject("Microsoft.XMLHTTP");  } **catch** (e){}  }  }  **return** xmlHttp;  } |

#### 1.3　打开与服务器的连接（open方法）

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；

|  |
| --- |
| **var** xmlHttp = createXMLHttpRequest();  xmlHttp.open("GET", "/ajaxdemo1/AServlet", **true**); |

#### 1.4　发送请求

当使用open打开连接后，就可以调用XMLHttpRequest对象的send()方法发送请求了。send()方法的参数为POST请求参数，即对应HTTP协议的请求体内容，若是GET请求，需要在URL后连接参数。

注意：若没有参数，需要给出null为参数！若不给出null为参数，可能会导致FireFox浏览器不能正常发送请求！

|  |
| --- |
| xmlHttp.send(**null**); |

#### 1.5　接收服务器响应

当请求发送出去后，服务器端Servlet就开始执行了，但服务器端的响应还没有接收到。接下来我们来接收服务器的响应。

XMLHttpRequest对象有一个onreadystatechange事件，它会在XMLHttpRequest对象的状态发生变化时被调用。下面介绍一下XMLHttpRequest对象的5种状态：

* 0：初始化未完成状态，只是创建了XMLHttpRequest对象，还未调用open()方法；
* 1：请求已开始，open()方法已调用，但还没调用send()方法；
* 2：请求发送完成状态，send()方法已调用；
* 3：开始读取服务器响应；
* 4：读取服务器响应结束。

onreadystatechange事件会在状态为1、2、3、4时引发。

　　下面代码会被执行四次！对应XMLHttpRequest的四种状态！

|  |
| --- |
| xmlHttp.onreadystatechange = **function**() {  alert('hello');  }; |

但通常我们只关心最后一种状态，即读取服务器响应结束时，客户端才会做出改变。我们可以通过XMLHttpRequest对象的readyState属性来得到XMLHttpRequest对象的状态。

|  |
| --- |
| xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4) {  alert('hello');  }  }; |

其实我们还要关心服务器响应的状态码是否为200，其服务器响应为404，或500，那么就表示请求失败了。我们可以通过XMLHttpRequest对象的status属性得到服务器的状态码。

最后，我们还需要获取到服务器响应的内容，可以通过XMLHttpRequest对象的responseText得到服务器响应内容。

|  |
| --- |
| xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {  alert(xmlHttp.responseText);  }  }; |

#### 1.6　AJAX第一例小结

* 创建XMLHttpRequest对象；
* 调用open()方法打开与服务器的连接；
* 调用send()方法发送请求；
* 为XMLHttpRequest对象指定onreadystatechange事件函数，这个函数会在XMLHttpRequest的1、2、3、4，四种状态时被调用；

XMLHttpRequest对象的5种状态：

* 0：初始化未完成状态，只是创建了XMLHttpRequest对象，还未调用open()方法；
* 1：请求已开始，open()方法已调用，但还没调用send()方法；
* 2：请求发送完成状态，send()方法已调用；
* 3：开始读取服务器响应；
* 4：读取服务器响应结束。

通常我们只关心4状态。

XMLHttpRequest对象的status属性表示服务器状态码，它只有在readyState为4时才能获取到。

XMLHttpRequest对象的responseText属性表示服务器响应内容，它只有在readyState为4时才能获取到！

### 2　AJAX第二例（发送POST请求）

发送POST请求

* 需要设置请求头：ContentType，其值为：application/x-www-form-urlencoded
* 使用xmlHttp对象的setRequestHeader(“Content-Type”, “application/x-www-form-urlencoded”)；
* 在发送时可以指定请求体了：xmlHttp.send(“username=张三&password=123”)

#### 2.1　发送POST请求注意事项

POST请求必须设置ContentType请求头的值为application/x-www.form-encoded。表单的enctype默认值就是为application/x-www.form-encoded！因为默认值就是这个，所以大家可能会忽略这个值！当设置了<form>的enctype=” application/x-www.form-encoded”时，等同与设置了Cotnent-Type请求头。

但在使用AJAX发送请求时，就没有默认值了，这需要我们自己来设置请求头：

|  |
| --- |
| xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); |

当没有设置Content-Type请求头为application/x-www-form-urlencoded时，Web容器会忽略请求体的内容。所以，在使用AJAX发送POST请求时，需要设置这一请求头，然后使用send()方法来设置请求体内容。

|  |
| --- |
| xmlHttp.send("b=B"); |

　　这时Servlet就可以获取到这个参数！！！

AServlet

|  |
| --- |
| **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  System.*out*.println(request.getParameter("b"));  System.*out*.println("Hello AJAX!");  response.getWriter().print("Hello AJAX!");  } |

ajax2.jsp

|  |
| --- |
| <script type=*"text/javascript"*>  **function** createXMLHttpRequest() {  **try** {  **return new** XMLHttpRequest();//大多数浏览器  } **catch** (e) {  **try** {  **return new** ActiveXObject("Msxml2.XMLHTTP");  } **catch** (e) {  **return new** ActiveXObject("Microsoft.XMLHTTP");  }  }  }  **function** send() {  **var** xmlHttp = createXMLHttpRequest();  xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {  **var** div = document.getElementById("div1");  div.innerText = xmlHttp.responseText;  div.textContent = xmlHttp.responseText;  }  };  xmlHttp.open("POST", "/ajaxdemo1/AServlet?a=A", **true**);  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  xmlHttp.send("b=B");  }  </script> |
| <h1>AJAX2</h1>  <button onclick="send()">测试</button>  <div id=*"div1"*></div> |

### 3　AJAX第三例（用户名是否已被注册）

#### 3.1　功能介绍

在注册表单中，当用户填写了用户名后，把光标移开后，会自动向服务器发送异步请求。服务器返回true或false，返回true表示这个用户名已经被注册过，返回false表示没有注册过。

客户端得到服务器返回的结果后，确定是否在用户名文本框后显示“用户名已被注册”的错误信息！

#### 3.2　案例分析

* regist.jsp页面中给出注册表单；
* 在username表单字段中添加onblur事件，调用send()方法；
* send()方法获取username表单字段的内容，向服务器发送异步请求，参数为username；
* RegistServlet：获取username参数，判断是否为“itcast”，如果是响应true，否则响应false；

#### 3.3　代码

regist.jsp

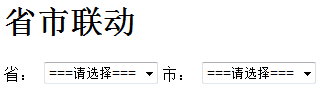
|  |
| --- |
| <script type=*"text/javascript"*>  **function** createXMLHttpRequest() {  **try** {  **return new** XMLHttpRequest();  } **catch** (e) {  **try** {  **return new** ActiveXObject("Msxml2.XMLHTTP");  } **catch** (e) {  **return new** ActiveXObject("Microsoft.XMLHTTP");  }  }  }  **function** send() {  **var** xmlHttp = createXMLHttpRequest();  xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {  **if**(xmlHttp.responseText == "true") {  document.getElementById("error").innerText = "用户名已被注册！";  document.getElementById("error").textContent = "用户名已被注册！";  } **else** {  document.getElementById("error").innerText = "";  document.getElementById("error").textContent = "";  }  }  };  xmlHttp.open("POST", "/ajaxdemo1/BServlet", **true**, "json");  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  **var** username = document.getElementById("username").value;  xmlHttp.send("username=" + username);  }  </script> |
| <h1>注册</h1>  <form action=*""* method=*"post"*>  用户名：<input id=*"username"* type=*"text"* name=*"username"* onblur="send()"/><span id=*"error"*></span><br/>  密　码：<input type=*"text"* name=*"password"*/><br/>  <input type=*"submit"* value=*"注册"*/>  </form> |

RegistServlet.java

|  |
| --- |
| **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.setCharacterEncoding("utf-8");  response.setContentType("text/html;charset=utf-8");    String username = request.getParameter("username");  **if**("itcast".equals(username)) {  response.getWriter().print(**true**);  } **else** {  response.getWriter().print(**false**);  }  } |

### 4　AJAX第四例（省市二级联动）

#### 4.1　功能介绍



select.jsp

|  |
| --- |
| <h1>省市联动</h1>  省：  <select name=*"province"* id=*"province"*>  <option>===请选择===</option>  </select>  市：  <select name=*"city"* id=*"city"*>  <option>===请选择===</option>  </select> |

　　当select.jsp页面打开时，向服务器发送异步请求，得到所有省份的名称（文本数据）。然后使用每个省份名称创建<option>，添加到<select name=”province”>中。

　　并且为<select name=”province”>元素添加onchange事件监听。当选择的省份发生变化时，再向服务器发送异常请求，得到当前选中的省份下所有城市（XML数据）。然后客户端解析XML文档，使用每个城市名称创建<option>，添加到<select name=”city”>元素中。

#### 4.2　代码实现

服务器端：使用china.xml保存所有省份和城市名称：

china.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <china>  <province name=*"北京"*>  <city>东城区</city>  <city>西城区</city>  ……  </province>  <province name=*"天津"*>  <city>和平区</city>  <city>河东区</city>  ……  </province>  <province name=*"河北"*>  <city>石家庄</city>  <city>衡水</city>  ……  </province>  ……  </china> |

* ProvinceServlet：负责把所有省份名称响应给客户端，这需要使用dom4j解析china.xml，得到所有<province>元素的name属性值，连接成一个字符串发送给客户端；
* CityServlet：负责得到某个省份元素，然后以字符串形式发送给客户端；

ProvinceServlet.java

|  |
| --- |
| **public** **class** ProvinceServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  // 注意设置编码  response.setContentType("text/html;charset=utf-8");    // 使用DOM4J解析xml文档  InputStream input = **this**.getClass().getClassLoader().getResourceAsStream("china.xml");  SAXReader reader = **new** SAXReader();  **try** {  Document doc = reader.read(input);  // xpath查询所有province元素的name属性  List<Attribute> provinceNameAttributeList = doc.selectNodes("//province/@name");  // 用来装载所有name属性值  List<String> provinceNames = **new** ArrayList<String>();  // 遍历每个属性，获取属性名称，添加到list中  **for**(Attribute proAttr : provinceNameAttributeList) {  provinceNames.add(proAttr.getValue());  }  // System.out.println(provinceNames);  // 把list转换成字符串  String str = provinceNames.toString();  // 把字符串前后中的[]去除发送给客户端  response.getWriter().print(str.substring(1, str.length()-1));  } **catch** (DocumentException e) {  **throw** **new** RuntimeException(e);  }  }  } |

CityServlet.java

|  |
| --- |
| **public** **class** CityServlet **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.setCharacterEncoding("utf-8");  // 注意，这里内容类型必须是text/xml，不然客户端得到的就不是xml文档对象，而是字符串了。  response.setContentType("text/xml;charset=utf-8");  // 获取省份参数  String provinceName = request.getParameter("provinceName");    InputStream input = **this**.getClass().getClassLoader().getResourceAsStream("china.xml");  SAXReader reader = **new** SAXReader();  **try** {  Document doc = reader.read(input);  // 查询指定省份名称的<province>元素  Element provinceElement = (Element)doc.selectSingleNode("//province[@name='" + provinceName + "']");  // System.out.println(provinceElement.asXML());  // 把元素转换成字符串发送给客户端  response.getWriter().print(provinceElement.asXML());  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  }  }  } |

客户端：

* 在打开select.jsp页面时就向服务器请求所有省份的名称，添加到<select name=”province”>元素中。
* 给<select name=”province”>元素添加onchange事件监听，内容为向服务器发送请求，得到XML文档：<province>元素，然后解析它，添加到<select name=”city”>中。

|  |
| --- |
| // 文档加载完成后  // 加载所有省份名称  window.onload = **function**() {  /\*  请求服务器，加载所有省名称到<select>中  \*/  /\*  1. ajax四步  \*/  **var** xmlHttp = createXMLHttpRequest();  xmlHttp.onreadystatechange = callback;//服务器响应完成后执行callback函数  xmlHttp.open("GET", "/ajaxdemo1/ProvinceServlet", **true**);//向服务器发送GET请求  xmlHttp.send(**null**);//发送请求  }; |
| // 本方法获取服务器响应的所有省份的名称  **function** callback() {  **if**(**this**.readyState == 4 && **this**.status == 200) {  // 把服务器响应的省份名称，使用逗号分割成字符串数组  **var** provinceNameArray = **this**.responseText.split(", ");  // 遍历每个省份名称，使用每个省份名称创建<option>元素，添加到province的<select>中  **for**(**var** i = 0; i < provinceNameArray.length; i++) {  addProvinceOption(provinceNameArray[i]);  }  // 为province的<select>元素添加onchange事件监听  document.getElementById("province").onchange = loadCities;  }  } |
| // 本函数在province的<select>元素发送变化时执行！  // 本函数会使用当前选中的省份名称为参数，向服务器发送请求，获取当前省份下的所有城市！  **function** loadCities() {  **var** proName = **this**.value;//获取<select>选择的省份名称  /\*  AJAX4步  \*/  **var** xmlHttp = createXMLHttpRequest();//创建异常对象  // 指定回调函数  xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {  // 得到服务器响应的xml文档对象  **var** doc = xmlHttp.responseXML;//注意，这里使用的是resopnseXML属性，不是resopnseText  // 获取文档中所有city元素  **var** cityElementList = doc.getElementsByTagName("city");  // 获取html元素：city的<select>  **var** citySelect = document.getElementById("city");  // 删除city的<select>元素的所有子元素  removeChildNodes(citySelect);    // 创建<option>元素，指定文本内容为“请选择”  **var** qxzOption = document.createElement("option");  **var** textNode = document.createTextNode("===请选择===");  qxzOption.appendChild(textNode);  // 把"请选择"这个<option>添加到<select>元素中  citySelect.appendChild(qxzOption);    // 循环遍历每个服务器端响应的每个<city>元素  **for**(**var** i = 0; i < cityElementList.length; i++) {  **var** cityEle = cityElementList[i];  **var** cityName = **null**;  // 获取<city>元素的文本内容！处理浏览器差异！  **if**(window.addEventListener) {  cityName = cityEle.textContent;  } **else** {  cityName = cityEle.text;  }  // 使用城市名称创建<option>，并添加到<select>元素中  addCityOption(cityName);  }  }  };  xmlHttp.open("POST", "/ajaxdemo1/CityServlet", **true**);//打开与服务器的连接  // 因为是POST请求，所以要设置Content-Type请求头  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  // 参数为当前选中的省份名称  xmlHttp.send("provinceName=" + proName);  } |
| // 使用proName创建<option>元素添加到<select>元素中  **function** addProvinceOption(proName) {  **var** option = document.createElement("option");//创建<option>元素  **var** textNode = document.createTextNode(proName);//使用省份名称创建文本节点  option.appendChild(textNode);//把省份名称的文本节点添加到<option>元素中  option.setAttribute("value", proName);//使用省份名称来设置<option>元素的value属性  document.getElementById("province").appendChild(option);//把<option>元素添加到<select>元素中  } |
| // 本函数用来创建城市的<option>，并添加到<select>元素中  **function** addCityOption(cityName) {  **var** citySelect = document.getElementById("city");//获取id为city的<select>  **var** cityOption = document.createElement("option");//创建<option>元素  **var** textNode = document.createTextNode(cityName);//使用城市名称创建文本节点  cityOption.appendChild(textNode);//把文本节点添加到<option>元素中  cityOption.setAttribute("value", cityName);//设置<option>元素的value属性为城市名称  citySelect.appendChild(cityOption);//把<option>元素添加到<select>元素中  } |
| //删除指定元素的所有子元素  **function** removeChildNodes(ele) {  **var** nodes = ele.childNodes;//获取当前元素的所有子元素集合  **while**(nodes.length > 0) {//遍历所有子元素  ele.removeChild(nodes[0]);//删除子元素  }  } |

### XStream

#### 1　XStream的作用

XStream可以把JavaBean对象转换成XML！

通常服务器向客户端响应的数据都是来自数据库的一组对象，而我们不能直接把对象响应给响应端，所以我们需要把对象转换成XML再响应给客户端，这时就需要使用XStream组合了。

#### 2　XStream入门

为了演示XStream的作用，我们需要先写两个类，Province和City。

City.java

|  |
| --- |
| **public** **class** City {  **private** String name;  **private** String description;  ……  } |

Province.java

|  |
| --- |
| **public** **class** Province {  **private** String name;  **private** List<City> cities = **new** ArrayList<City>();    **public** **void** addCity(City city) {  cities.add(city);  }  ……  } |

接下来，我们需要写一个main()，创建一个List，List中存放两个Province对象！最终我们把List转换成XML。

|  |
| --- |
| Province p1 = **new** Province("辽宁省");  p1.addCity(**new** City("沈阳", "shenyang"));  p1.addCity(**new** City("大连", "dalian"));    Province p2 = **new** Province("吉林省");  p2.addCity(**new** City("长春", "changchen"));  p2.addCity(**new** City("白城", "baicheng"));    List<Province> list = **new** ArrayList<Province>();  list.add(p1);  list.add(p2); |

##### 2.1　XStream相关JAR包

我们可以到http://xstream.codehaus.org/地址去下载XStream安装包！

XStream的必导JAR包：

* 核心JAR包：xstream-1.4.7.jar；
* 必须依赖包：xpp3\_min-1.1.4c（XML Pull Parser，一款速度很快的XML解析器）；

##### 2.2　使用XStream把Java对象转换成XML

下面是使用XStream转换list为XML的代码：

|  |
| --- |
| XStream xstream = **new** XStream();  String s = xstream.toXML(list);  System.*out*.println(s); |
| <list>  <cn.itcast.xstream.demo1.Province>  <name>辽宁省</name>  <cities>  <cn.itcast.xstream.demo1.City>  <name>沈阳</name>  <description>shenyang</description>  </cn.itcast.xstream.demo1.City>  <cn.itcast.xstream.demo1.City>  <name>大连</name>  <description>dalian</description>  </cn.itcast.xstream.demo1.City>  </cities>  </cn.itcast.xstream.demo1.Province>  <cn.itcast.xstream.demo1.Province>  <name>吉林省</name>  <cities>  <cn.itcast.xstream.demo1.City>  <name>长春</name>  <description>changchen</description>  </cn.itcast.xstream.demo1.City>  <cn.itcast.xstream.demo1.City>  <name>白城</name>  <description>baicheng</description>  </cn.itcast.xstream.demo1.City>  </cities>  </cn.itcast.xstream.demo1.Province>  </list> |

　　也就是说，XStream是根据对象名、类名、属性名来生成XML文档的！

##### 2.3　alias用法

大家也看到了，生成的XML中，与类名对应的元素名称包含了包名部分，这很不好看！若想自定义生成的元素名称，需要使用XStream为类名提供别名：

|  |
| --- |
| xstream.alias("province", Province.**class**);  xstream.alias("china", List.**class**);  xstream.alias("city", City.**class**); |
| <china>  <province>  <name>辽宁省</name>  <cities>  <city>  <name>沈阳</name>  <description>shenyang</description>  </city>  <city>  <name>大连</name>  <description>dalian</description>  </city>  </cities>  </province>  <province>  <name>吉林省</name>  <cities>  <city>  <name>长春</name>  <description>changchen</description>  </city>  <city>  <name>白城</name>  <description>baicheng</description>  </city>  </cities>  </province>  </china> |

##### 2.4　把子元素变为元素属性

例如我们需要把<province>子元素<name>变成：<province name=””>样式，那么需要调用如下方法：

|  |
| --- |
| xstream.useAttributeFor(Province.**class**, "name"); |
| <china>  <province name="辽宁省">  <cities>  <city>  <name>沈阳</name>  <description>shenyang</description>  </city>  <city>  <name>大连</name>  <description>dalian</description>  </city>  </cities>  </province>  <province name="吉林省">  <cities>  <city>  <name>长春</name>  <description>changchen</description>  </city>  <city>  <name>白城</name>  <description>baicheng</description>  </city>  </cities>  </province>  </china> |

##### 2.5　去除集合属性对应元素

大家可能已经发现了，因为Pronvice类有一个cities成员，所以生成了<cities>元素，但这个元素对XML文档而言没有什么意义，所以我们希望把它去除！

|  |
| --- |
| xstream.addImplicitCollection(Province.**class**, "cities"); |
| <china>  <province name="辽宁省">  <city>  <name>沈阳</name>  <description>shenyang</description>  </city>  <city>  <name>大连</name>  <description>dalian</description>  </city>  </province>  <province name="吉林省">  <city>  <name>长春</name>  <description>changchen</description>  </city>  <city>  <name>白城</name>  <description>baicheng</description>  </city>  </province>  </china> |

##### 2.6　让类的成员不生成对应XML元素

到现在为止，我们都是每个类，每个成员都有对应的元素（或属性）存在，但有时我们并不希望某些类的成员在对应的XML文档中出现，例如我们不希望City类的description成员出现在XML文档中，可以使用下面方法：

|  |
| --- |
| xstream.omitField(City.**class**, "description"); |
| <china>  <province name="辽宁省">  <city>  <name>沈阳</name>  </city>  <city>  <name>大连</name>  </city>  </province>  <province name="吉林省">  <city>  <name>长春</name>  </city>  <city>  <name>白城</name>  </city>  </province>  </china> |

### JSON

#### 1　什么是JSON

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。

JSON是用字符串来表示Javascript对象，例如可以在Servlet中发送一个JSON格式的字符串给客户端Javascript，Javascript可以执行这个字符串，得到一个Javascript对象。

XML也可以用来佟大为数据交换，前面已经学习过在Servlet中发送XML给Javascript，然后Javascript再去解析XML。

#### 2　JSON对象语法

JSON 语法：

* 数据在名称/值对中（与java的map相似）
* 数据由逗号分隔（多个键值对使用逗号分隔）
* 花括号保存对象（对象需要使用大括号括起来）
* 方括号保存数组（数组使用中括号括起来）

|  |
| --- |
| **var** person = {"name":"zhangSan", "age":"18", "sex":"male"};  alert(person.name + ", " + person.age + ", " + person.sex); |

　　注意，key也要在双引号中！

JSON值：

* [数字](http://baike.baidu.com/view/37407.htm" \t "_blank)（整数或浮点数）
* 字符串（在双引号中）
* 逻辑值（true 或 false）
* 数组（在方括号中）
* 对象（在花括号中）
* null

|  |
| --- |
| **var** person = {"name":"zhangSan", "age":"18", "sex":"male", "hobby":["cf", "sj", "ddm"]};  alert(person.name + ", " + person.age + ", " + person.sex + ", " + person.hobby); |

带有方法的JSON对象：

|  |
| --- |
| **var** person = {"name":"zhangSan", "getName":**function**() {**return this**.name;}};  alert(person.name);  alert(person.getName()); |

#### 3　JSON与XML比较

* 可读性：XML胜出；
* 解码难度：JSON本身就是JS对象（主场作战），所以简单很多；
* 流行度：XML已经流行好多年，但在AJAX领域，JSON更受欢迎。

#### 4　把Java对象转换成JSON对象

apache提供的json-lib小工具，它可以方便的使用Java语言来创建JSON字符串。也可以把JavaBean转换成JSON字符串。

##### 4.1　json-lib核心jar包

json-lib的核心jar包有：

* json-lib.jar

json-lib的依赖jar包有：

* commons-lang.jar
* commons-beanutils.jar
* commons-logging.jar
* commons-collections.jar
* ezmorph.jar
* xom-1.1.jar

##### 4.2　json-lib中的核心类

在json-lib中只有两个核心类：

* JSONObject；
* JSONArray；

##### 4.1　JSONObject

JSONObject类本身是一个Map，所以学习它很方便。

|  |
| --- |
| JSONObject jo = **new** JSONObject();  jo.put("name", "zhangSan");  jo.put("age", "18");  jo.put("sex", "male");  System.*out*.println(jo.toString()); |

|  |
| --- |
| Person person = **new** Person("liSi", 18, "female");  JSONObject jo = JSONObject.*fromObject*(person);  System.*out*.println(jo.toString()); |

|  |
| --- |
| Map map = **new** HashMap();  map.put("name", "wangWu");  map.put("age", "81");  map.put("sex", "male");    JSONObject jo = JSONObject.*fromObject*(map);  System.*out*.println(jo.toString()); |

|  |
| --- |
| String xml = "<person><name>zhaoLiu</name><age>59</age><sex>female</sex></person>";  XMLSerializer serial = **new** XMLSerializer();  JSONObject jo = (JSONObject)serial.read(xml);  System.*out*.println(jo.toString()); |

##### 4.2　JSONArray

JSONArray本身是一个List，所以使用起来很方便。

|  |
| --- |
| JSONArray ja = **new** JSONArray();  Person p1 = **new** Person("zhangSan", 18, "male");  Person p2 = **new** Person("liSi", 23, "female");  ja.add(p1);  ja.add(p2);    System.*out*.println(ja.toString()); |

|  |
| --- |
| Person p1 = **new** Person("zhangSan", 18, "male");  Person p2 = **new** Person("liSi", 23, "female");  List<Person> list = **new** ArrayList<Person>();  list.add(p1);  list.add(p2);    JSONArray ja = JSONArray.*fromObject*(list);    System.*out*.println(ja.toString()); |

|  |
| --- |
| Person p1 = **new** Person("zhangSan", 18, "male");  Person p2 = **new** Person("liSi", 23, "female");  Person[] persons = {p1, p2};    JSONArray ja = JSONArray.*fromObject*(persons);    System.*out*.println(ja.toString()); |

#### 5　JS解释服务器发送过来的JSON字符串

服务器发送过来JSON字符串后，客户端需要对其进行解析。这时客户端需要使用eval()方法对JSON字符串进行执行！但要注意，eval()方法在执行JSON时，必须把JSON字符串使用一对圆括号括起来。

|  |
| --- |
| **var** json = "{\"name\":\"zhangSan\", \"age\":\"18\", \"sex\":\"male\"}";  **var** person = eval("(" + json + ")");  alert(person.name + ", " + person.age + ", " + person.sex); |