**Ajax && 数据交换语言**

# 1.AJAX概述

## 1.1　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）。

与传统Web应用不同的是，Ajax将请求与页面分离：在传统的Web应用中，每个请求即对应一个页面，不管客户端以POST还是GET方式提交请求，每次请求都会丢弃当前页面，等待服务器生成一个全新的页面。在等待期间，旧的页面已经丢弃，新的页面尚未生成，整个浏览器一片空白，而用户在此期间除了等待，不能进行任何操作，这种不连续的体验，给用户带来了很不好的体验。这也是传统BS结构的缺点（独占式的请求，频繁的页面刷新，简陋的页面）。Ajax的出现弥补了以上不足，Ajax使用XMLHttpRequest对象异步发送请求,Ajax应用不采用请求对应页面的模式,发送请求不要求重新加载页面。浏览器发送请求后无需等待浏览器响应，而是可以继续原来的操作。在服务器响应完成后，客户端使用JavaScript函数将响应数据加载到浏览器中，实现局部刷新。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，用户的界面以连续的方式运行。使用Ajax能够避免频繁刷新页面，服务器响应的是数据，而不再是页面内容。Ajax负责获取服务器数据，然后将服务器数据动态加载到浏览器中。

Ajax是由已有的几种技术组合而成，通过在浏览器和服务器之间添加Ajax中间层，运行浏览器异步发送数据，同时JavaScript通过DOM动态加载响应内容局部更新HTML页面。用户的请求不再直接向服务器提交，而是使用XMLHttpRequest异步的向服务器发送,从而避免了丢弃当前页面。

* 与服务器异步交互，避免阻塞用户操作；
* 浏览器页面局部刷新，无需频繁重新加载页面；

Ajax技术基于JavaScript，却有加强了JavaScript。JavaScript创建并使用XMLHttpRequest向服务器发送请求和解析数据，同时又通过DOM动态更新HTML页面。

Ajax技术甚至催生了一种新的网络游戏平台:网页游戏--游戏玩家无需下载任何客户端,直接打开网页就可开始游戏。Ajax的出现让B/S结构的应用更加完善。

## 1.2　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。

## 1.3　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技术向服务器发出异步请求，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；

## 1.4　AJAX的优缺点

优点：

* AJAX使用Javascript技术向服务器发送异步请求,JavaScript负责解析服务器的响应结果和动态刷新页面；
* 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，减轻了服务器和带宽的负担,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；

# 2.AJAX技术

Ajax技术的灵魂是异步发送请求的对象XMLHttpRequest，若没有这个对象Ajax将失去原有的意义。Ajax最早应用于IE5。JavaScript是Ajax技术的另一个重要部分。主要完成以下事情：

* 创建XMLHttpRequest对象；
* 通过XMLHttpRequest向服务器发送请求；
* 创建回调函数，监视服务器响应状态，在服务器响应完成后，回调函数启动；
* 回调函数通过DOM动态更新HTML页面；

|  |
| --- |
| **小知识点：什么是回调函数**  **回调函数就是一个通过[函数指针](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0%E6%8C%87%E9%92%88/2674905" \t "https://baike.baidu.com/item/%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0/_blank)调用的函数。如果你把函数的[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88/2878304" \t "https://baike.baidu.com/item/%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0/_blank)（地址）作为[参数传递](https://baike.baidu.com/item/%E5%8F%82%E6%95%B0%E4%BC%A0%E9%80%92/9019335" \t "https://baike.baidu.com/item/%E5%9B%9E%E8%B0%83%E5%87%BD%E6%95%B0/_blank)给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。** |

## 2.1　AJAX第一例(get请求)

### 2.1.1 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!");  }  } |

### 2.1.2　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XMLHttpRequest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。根据MSDN的解释，XMLHttpRequest提供客户端同HTTP服务器通信的协议，客户端可以通过该对象向服务器发送请求，并使用微软XML文档模型DOM来处理服务器的响应。

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：

|  |
| --- |
| **小知识点：什么是DOM2**  **DOM 是用来访问或操作HTML文档、XHTML文档、XML文档中的节点元素。现在基本上所有的浏览器都都执行了W3C发布的DOM规范，所以在浏览器上就可以用DOM的这些API。 DOM的级别Level：  DOM0：不是W3C规范。  DOM1：开始是W3C规范。专注于HTML文档和XML文档。  DOM2：对DOM1增加了样式表对象模型  DOM3：对DOM2增加了内容模型 (DTD 、Schemas) 和文档验证。** |
| **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){  alert（“对不起您使用的浏览器版本暂不支持Ajax，请您更换浏览器重试，或升级浏览器！”）；  throw e;  }  }  }  **return** xmlHttp;  } |

### 2.1.3　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：<c:url value=’/ajaxdemo1/AServlet’/>，若为GET请求，还可以在URL后追加参数；<c:url value=’/ajaxdemo1/AServlet’/>?username=’张三’
* 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", "<c:url value = '/AServlet'/>", **true**); |

### 2.1.4　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**);get请求中若send中有参数，那么不会报错，但是get请求也不会处理send中的参数 |

### 2.1.5　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(监听服务器响应)，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2、3、4时引发。

　　下面代码会被执行3次！对应XMLHttpRequest的后3种状态！

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

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。**实际应用中往往需要对服务器响应不正常的情况进行处理，如弹出错误弹窗等，告诉浏览者服务器响应出错。如果服务器响应出现了错误，但页面没有输出任何提示，则对浏览器来说是一个巨大的困惑。我们可以通过下面代码来实现，我们在学习中会简化这种操作，这里要注意。**

|  |
| --- |
| 简化版本：  xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {  **var** text = xmlHttp.responseText;  **// 获取h1元素**  **var** h1 = document.getElementById("h1");  h1.innerHTML = text;  }  }; |
| xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4) {  if(&& xmlHttp.status == 200){  **var** text = xmlHttp.responseText;  **// 获取h1元素**  **var** h1 = document.getElementById("h1");  h1.innerHTML = text;  }else{  Window.alert(“你请求的页面有异常,服务器不能正确处理!”);  }  }  }; |

### 2.1.6　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！Ajax1.jsp

|  |
| --- |
| 完整代码:ajax1.jsp  <body>  <button id=*"btn"*>点击这里</button>  <h1 id=*"h1"*></h1>  </body> |
| **package** cn.ajax\_basic.get;  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println("Hello Ajax!");  resp.getWriter().print("Hello Ajax\_Get!!!");  }  } |



## 2.2　AJAX第二例（发送POST请求）

### 2.2.1　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。

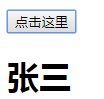
|  |
| --- |
| xmlHttp.send("b=B"); |

这时Servlet就可以获取到这个参数！！！

|  |
| --- |
| package cn.ajax\_basic.post;  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  resp.setContentType("text/html;charset=utf-8");  req.setCharacterEncoding("utf-8");  String username = req.getParameter("username");  System.out.println(req.getParameter("a"));  System.out.println("POST： Hello AJAX!" + username);  resp.getWriter().print(username);  }  } |

ajax2.jsp

|  |
| --- |
| <script type=*"text/javascript"*>  **// 创建异步对象---略**  window.onload = **function**() {**//文档加载完毕后执行**  **var** btn = document.getElementById("btn");  btn.onclick = **function**() {**//给按钮的点击事件注册监听**  **var** xmlHttp = createXMLHttpRequest();  xmlHttp.open("POST", "<c:url value = '/APostServlet'/>?a=A", **true**);  **//设置POST请求头**  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  xmlHttp.send("username=张三&password=123");**//POST请求参数**  xmlHttp.onreadystatechange = **function**() {**//当xmlHttp的状态发生变化时执行**  **// 双重判断：xmlHttp的状态为4（服务器响应结束），以及服务器响应的状态码为200（响应成功）**  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {  **// 获取服务器的响应结束**  **var** text = xmlHttp.responseText;  **// 获取h1元素**  **var** h1 = document.getElementById("h1");  h1.innerHTML = text;  }  };  };  };  </script> |
| <body>  <button id=*"btn"*>点击这里</button>  <h1 id=*"h1"*></h1>  </body> |



通过上面我们发现POST请求中在URL后面添加参数是可以被获取到的,但是如果没有请求头,那么会默认作为get请求处理,此时send参数不会别获取;

## 2.3　AJAX第三例（用户名是否已被注册）

用户注册是十分常见的功能,而怎样给用户更好的体验,则是作为程序员的我们应该考虑的,其中很重要的一点就是怎么子啊用户注册的时候动态的验证用户注册的内容。

### 2.3.1　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！

### 2.3.2　案例分析

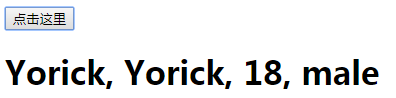
* Ajax3.jsp页面中给出注册表单；
* 在username表单字段中添加onblur事件；
* 异步方法获取username表单字段的内容，向服务器发送异步请求，参数为username；
* ValidateUsernameServlet：获取username参数，判断是否为“Yorick”，如果是响应true，否则响应false；

### 2.3.3　代码

|  |
| --- |
| window.onload = **function**() {  **// 获取文本框，给它的失去焦点事件注册监听**  **var** userEle = document.getElementById("usernameEle");  **//onblur事件,当用户离开输入字段时对其进行验证**  userEle.onblur = **function**() {  **//1.得到异步对象**  **var** xmlHttp = createXMLHttpRequest();  **//2.打开连接**  xmlHttp.open("POST", "<c:url value='/ValidateUsernameServlet'/>", **true**);  **//3.设置请求头：Content-Type**  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  **//4.发送请求，给出请求体**  xmlHttp.send("username=" + userEle.value);    **//5.给xmlHttp的onreadystatechange事件注册监听**  xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {**//双重判断**  **//获取服务器的响应，判断是否为1**  **// 是：获取span，添加内容：“用户名已被注册”**  **var** text = xmlHttp.responseText;  **var** span = document.getElementById("errorSpan");  **if**(text == "1") {  **//得到span元素**  span.innerHTML = "用户名已被注册！";  } **else** {  span.innerHTML = "OK";  }  }  };  };  };  </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> |
| **package** cn.ajax\_basic.post;  **/\***  **\* 1. 获取参数username**  **\* 2. 判断是否为Yorick**  **\* 3. 如果是：响应1**  **\* 4. 如果不是：响应0**  **\*/** @Override  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  resp.setContentType("text/html;charset=utf-8");  String username = req.getParameter("username");  **if**(username.equalsIgnoreCase("Yorick")) {  resp.getWriter().print("1");  }**else** {  resp.getWriter().print("0");  }  }  } |

## 2.4 服务器返回xml数据的处理

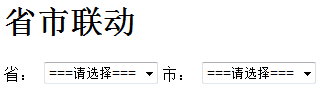
|  |
| --- |
| **package** cn.ajax\_basic.get;  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  resp.setContentType("text/xml;charset=utf-8");  String xml = "<students>" + "<student number='Yorick'>" +  "<name>Yorick</name>" + "<age>18</age>" +  "<sex>male</sex>" +  "</student>" +  "</students>";  resp.getWriter().print(xml);  }  } |
| <script type=*"text/javascript"*>  **// 创建异步对象略**  window.onload = **function**(){  **//获取点击按钮**  **var** btn = document.getElementById("btn");  btn.onclick = **function**(){  **//获取异步对象**  **var** xmlHttp = createXMLHttpRequest();  **//连接服务器**  xmlHttp.open("GET", "<c:url value='/XMLServlet'/>", **true**);  **//发送请求**  xmlHttp.send(**null**);  **//4. 给异步对象的onreadystatechange事件注册监听器**  xmlHttp.onreadystatechange = **function**(){  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200){**//双重判断**  **// 获取服务器的响应结果（xml）**  **var** doc = xmlHttp.**responseXML**;  **// 查询文档下名为student的所有元素，得到数组，再取下标0元素**  **var** ele = doc.getElementsByTagName("student")[0];  **var** number = ele.getAttribute("number");**//获取元素名为number的属性值**  **var** name;  **var** age;  **var** sex;  **//处理浏览器的差异--window.addEventListener为true说明不是IE**  **if**(window.addEventListener){  name = ele.getElementsByTagName("name")[0].textContent;**//其它浏览器**  }**else**{  name = ele.getElementsByTagName("name")[0].text;**//IE浏览器**  }  **if**(window.addEventListener){  age = ele.getElementsByTagName("age")[0].textContent;  }**else**{  age = ele.getElementsByTagName("age")[0].text;  }  **if**(window.addEventListener){  sex = ele.getElementsByTagName("sex")[0].textContent;  }**else** {  sex = ele.getElementsByTagName("sex")[0].text;  }  **var** text = number + ", " + name + ", " + age + ", " + sex;  document.getElementById("h1").innerHTML = text;  }  }  }  }  </script>  </head>  <body>  <button id=*"btn"*>点击这里</button>  <h1 id=*"h1"*></h1>  </body> |



## ？？2.5　AJAX第四例（省市二级联动）

我们在开发中也经常会遇到联动的问题，这也是通过异步交互完成的，下面我们来看一下；【第一次开发过程中onchange事件不能如期发生，使用selectSingleNode解析xml时，返回null，为题未解！！180923】

### 2.5.1　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ajax5.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。

### 2.4.2　代码实现

服务器端：使用china.xml保存所有省份和城市名称,这也意味着我们要解析xml获取所有的请求数据

|  |
| --- |
| <?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

|  |
| --- |
| **package cn.ajax\_basic.get;**  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  **//设置响应类型和编码格式**  resp.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 document = reader.read(input);  **// xpath查询所有province元素的name属性,得到一个由属性名和属性值构成的属性的集合**  List<Attribute> provinceNameAttributeList = document.selectNodes("//province/@name");  **// 用来装载所有name属性值**  List<String> provinceNames = **new** ArrayList<>();  **// 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**  resp.getWriter().print(str.substring(1, str.length()-1));  } **catch** (DocumentException e) {  **throw** **new** RuntimeException();  }  } |

CityServlet.java

|  |
| --- |
| @Override  **public** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  **//设置请求编码**  req.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**  resp.setContentType("text/xml;charset=utf-8");  String provinceName = req.getParameter("pname");**//获取省份名称**  System.***out***.println(provinceName);  **try** {  InputStream input = **this**.getClass().getResourceAsStream("src/china.xml");  SAXReader reader = **new** SAXReader();  Document document = reader.read(input);  Element root = document.getRootElement();  System.***out***.println(root);  **// 查询指定省份名称的<province>元素**  Element node = (Element) document.selectSingleNode("//province[@name='" + provinceName + "']");  System.***out***.println(node);  String xml = node.asXML();  **// 把元素转换成字符串发送给客户端**  resp.getWriter().print(xml);  } **catch** (DocumentException e) {  **throw** **new** RuntimeException();  }  } |

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。

Ajax5.jsp

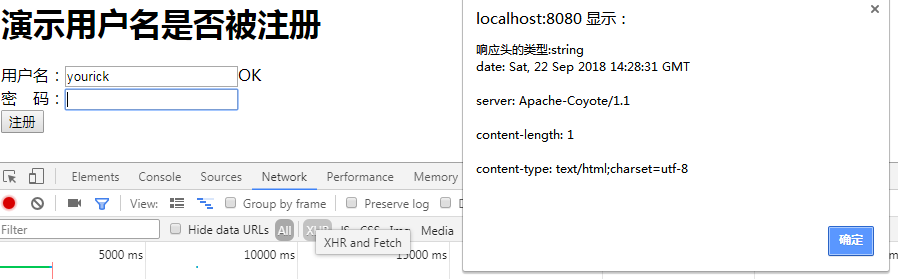
|  |
| --- |
| <script type=*"text/javascript"*>  **var** xmlHttp;  **function** createXMLHttpRequest() {  **if**(window.XMLHttpRequest){  **//适用于DOM2浏览器**  xmlHttp = **new** XMLHttpRequest();  }**else** **if** (window.ActiveXObject) {  **try** {  xmlHttp = **new** ActiveXObject("Msxml2.XMLHTTP");  } **catch** (e) {  **// 适用于IE5.5，以及IE更早版本**  **try**{  xmlHttp = **new** ActiveXObject("Microsoft.XMLHTTP");  } **catch** (e){  alert("对不起您使用的浏览器版本暂不支持Ajax，请您更换浏览器重试，或升级浏览器！");  **throw** e;  }  }  }  } |
| **//文档加载完成后,加载所有的省份信息**  window.onload = **function** () {  createXMLHttpRequest();  xmlHttp.open("GET","<c:url value = '/ProvinceServlet'/>",**true**);  xmlHttp.send(**null**);  xmlHttp.onreadystatechange = province;  } |
| **//本方法获取服务器响应的所有省份的名称**  **function** province() {  **if**(xmlHttp.readyState == 4){  **if**(xmlHttp.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 = xmlHttp.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++) {  **//alert("-"+provinceNameArray[i]+"-");**  addProvinceOption(provinceNameArray[i]);  }  **// 为province的<select>元素添加onchange事件监听**  document.getElementById("province").onchange = loadCities;  }**else** {  window.alert("你请求的页面有异常,服务器不能正确处理!");  }  }  } |
| **//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节点**  **var** text = document.createTextNode(proName);**//使用省份名称创建文本节点**  option.appendChild(text);**//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**  createXMLHttpRequest();  xmlHttp.open("POST", "<c:url value='/CityServlet'/>", **true**);  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  alert(proName);  alert(xmlHttp);  xmlHttp.send("pname=" + proName);**//把下拉列表中选择的值发送给服务器！**  xmlHttp.onreadystatechange = cities;  } |
| **//获取响应的所有城市**  **function** cities() {  **if**(xmlHttp.readyState == 4){  **if** (xmlHttp.status == 200) {  **var** doc = xmlHttp.responseXML;**//获取服务器响应的xml文档**  **// 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);  }  } **else** {  window.alert("你请求的页面有异常,服务器不能正确处理!!");  }  }  } |
| **//删除city的<select>元素的所有子元素**  **function** removeChildNodes(citySelect) {  **var** nodes = citySelect.childNodes;**//获取当前元素的所有子元素集合**  **while** (nodes.length > 0) {**//遍历所有的子元素**  citySelect.removeChild(nodes[0]);**//删除子元素**  }  } |
| **//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** cityTextNode = document.createElement(cityName);**//使用cityName创建文本节点**  cityOption.appendChild(cityTextNode);  cityOption.setAttribute("value",cityName);**//设置option的属性**  citySelect.appendChild(cityOption);**//将option节点添加到select节点**  }  </script> |

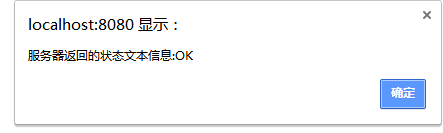
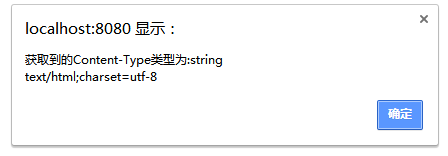
## 2.6 XMLHttpRequest的其它方法和属性

XMLHttpRequest除了提供以上的几个方法外,还提供了一些其它方法,下面我们来认识一下:

* abort():停止发送当前请求。
* getAllResponseHeaders():获取服务器返回的全部响应头，返回值是一个由所有响应头的“名：值”对所组成的字符串。
* getResponseHeader（“headerLabel”）：根据响应头的名字，获取对应的响应头。
* statusText：该属性是服务器返回的状态文本信息，只有当服务器的响应已经完成时，才会有该状态文本信息。

|  |
| --- |
| **var** text = xmlHttp.responseText;  **var** headers = xmlHttp.getAllResponseHeaders();  **var** header = xmlHttp.getResponseHeader("Content-Type");  **var** statusText = xmlHttp.statusText;  alert("响应头的类型:" + **typeof** headers + "\n" + headers);  alert("获取到的Content-Type类型为:" + **typeof** header + "\n" + header)  alert("服务器返回的状态文本信息:" + statusText); |

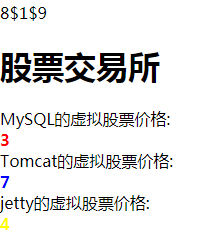




## 2.7 其它练习

### 2.7.1 简易版股票交易所

|  |
| --- |
| <script type=*"text/javascript"*>  **var** xmlHttp;  **function** createXMLHttpRequest() {  **if** (window.XMLHttpRequest) {  **//适用于DOM2浏览器**  xmlHttp = **new** XMLHttpRequest();  } **else** **if**(window.ActiveXObject){  **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){  alert("对不起您使用的浏览器版本暂不支持Ajax，请您更换浏览器重试，或升级浏览器！");  **throw** e;  }  }  }  }  **function** getPrice() {  createXMLHttpRequest();  **var** uri = "ajax6.jsp";  xmlHttp.open("GET",uri,**true**);  xmlHttp.onreadystatechange = processResponse;  xmlHttp.send(**null**);  }  **function** processResponse(){  **if**(xmlHttp.readyState == 4){  **if**(xmlHttp.status == 200){  **var** prices = xmlHttp.responseText.split("$");  document.getElementById("mysql").innerHTML = prices[0];  document.getElementById("tomcat").innerHTML = prices[1];  document.getElementById("jetty").innerHTML = prices[2];  **//设置一秒钟后再次发送请求**  setTimeout("getPrice()",1000);  }else{  window.alert(“你请求的页面有异常,服务器不能正确处理!”);  }  }  }  document.body.onload = getPrice;  </script>  </head>  <body>  <h1>股票交易所</h1>  MySQL的虚拟股票价格:<div id = *"mysql"* style="color:*red*;font-weight:*bold*;"></div>  Tomcat的虚拟股票价格:<div id = *"tomcat"* style="color:*blue*;font-weight:*bold*"></div>  jetty的虚拟股票价格:<div id = *"jetty"* style = "color:*yellow*;font-weight:*bold*"></div>  </body>  </html> |



# 3.Ajax小工具

通过上面的学习,我们的ajax已经入门了,但是我们是不是发现这么一个问题,那就是不管什么时候,我们都需要把ajax的所有步骤的操作一遍,这样既繁琐又容易出错,那么可不可以把ajax中重复的代码写在一个js上,然后每次在script中导入就行了呢,答案是可以的,下面我们就来亲自动手做做!

## 步骤1:创建js

## 步骤2:在js中书写重复代码-创建XMLHttpRequest对象

|  |
| --- |
| **function** createXMLHttpRequest() {  **try** {  **return** **new** XMLHttpRequest();**//大多数浏览器**  } **catch** (e) {  **try** {  **return** ActvieXObject("Msxml2.XMLHTTP");**//IE6.0**  } **catch** (e) {  **try** {  **return** ActvieXObject("Microsoft.XMLHTTP");**//IE5.5及更早版本**  } **catch** (e) {  alert("哥们儿，您用的是什么浏览器啊？");  **throw** e;  }  }  }  } |

## 步骤3:书写重复代码--打开连接及其它

|  |
| --- |
| **function** ajax(){  **//得到xmlhttp对象**  **var** xmlHttp = createXMLHttpRequest();  **//打开连接**  xmlHttp.open(method,url,asyn);  **//判断是否为POST**  **if**("POST" == method){  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  }  **//发送请求--如果是post请求发送参数,如果守GET请求,参数几遍书写,也发布出去**  xmlHttp.send(params);  **//注册监听**  xmlHttp.onreadystatechange = **function**() {  **if**(xmlHttp.readyState == 4 && xmlHttp.status == 200) {**//双重判断**  **var** data;**//服务器相应数据**  **// 获取服务器的响应数据，进行转换！**  **if**(!type) {**//如果type没有赋值，那么默认为文本**  data = xmlHttp.responseText;  } **else** **if**(type == "xml") {  data = xmlHttp.responseXML;  } **else** **if**(type == "text") {  data = xmlHttp.responseText;  } **else** **if**(type == "json") {  **var** text = xmlHttp.responseText;  data = eval("(" + text + ")");  }    **// 调用回调方法**  callback(data);  }  };  } |

初步分析书写如上,这是我们发现在ajax中有几个参数是需要在使用的时候手动传参的,那么可不可以把这些参数封装成一个对象呢?答案是肯定的,那么我们再分析,有哪些参数是可以有默认值的,不用外部传参呢?下面我们来分析

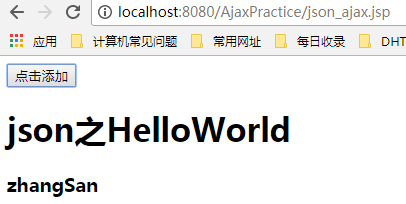
* Method:当方法外部不写的时候,默认是false,所以!method就是true,默认get方法
* Asyn:当异步连接不书写,那么默认是undefined,我们只需要判断其为undefined,那么就视其为默认异步
* **URL:这个是必传**
* Params:get请求,不传,默认为null
* Callback:回调,服务器响应的数据是要被使用的,所以回调是必须的
* Type:如果type没有赋值，那么默认为文本

|  |
| --- |
| /\*  \* option对象有如下属性  \*/  /\*请求方式\*/method,  /\*请求的url\*/ url,  /\*是否异步\*/asyn,  /\*请求体\*/params,  /\*回调方法\*/callback,  /\*服务器响应数据转换成什么类型\*/type  function ajax(option) {  /\*  \* 1. 得到xmlHttp  \*/  var xmlHttp = createXMLHttpRequest();  /\*  \* 2. 打开连接  \*/  if(!option.method) {//默认为GET请求  option.method = "GET";  }  if(option.asyn == undefined) {//默认为异步处理  option.asyn = true;  }  xmlHttp.open(option.method, option.url, option.asyn);  /\*  \* 3. 判断是否为POST  \*/  if("POST" == option.method) {  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  }  /\*  \* 4. 发送请求  \*/  xmlHttp.send(option.params);    /\*  \* 5. 注册监听  \*/  xmlHttp.onreadystatechange = function() {  if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {//双重判断  var data;  // 获取服务器的响应数据，进行转换！  if(!option.type) {//如果type没有赋值，那么默认为文本  data = xmlHttp.responseText;  } else if(option.type == "xml") {  data = xmlHttp.responseXML;  } else if(option.type == "text") {  data = xmlHttp.responseText;  } else if(option.type == "json") {  var text = xmlHttp.responseText;  data = eval("(" + text + ")");  }    // 调用回调方法  option.callback(data);  }  };  }; |

## 步骤4:实验可用性

我们使用JSON中的JSON与XML比较的JSON代码来实验一下

|  |
| --- |
| <script type=*"text/javascript"* **src=*"*<c:url value=*'/ajax-lib/ajaxutils.js'***/>*"*></script>  <script type=*"text/javascript"*>  window.onload = **function**(){  **var** btn = document.getElementById("btn");  btn.onclick = **function**(){  ajax({  url:"<c:url value='/JSONServlet'/>",  type:"json",  callback:**function**(data){  document.getElementById("h3").innerHTML = data.students[0].name;  }  });  };  };  </script>  </head>  <body>  <button id=*"btn"*>点击添加</button>  <h1>json之HelloWorld</h1>  <h3 id=*"h3"*>点击替换</h3>  </body> |
| /\*\*  \* Servlet implementation class JSONServlet  \*/  @WebServlet("/JSONServlet")  **public** **class** JSONServlet **extends** HttpServlet {    **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  response.setContentType("text/html;charset=utf-8");  **//向客户端发送JSON对象**  String students ="{\"students\":[{\"number\":\"ITCAST\_1001\",\"name\":\"zhangSan\",\"age\":18,\"sex\":\"male\"}]}";  response.getWriter().print(students);  System.*out*.println(students);  }  } |



事实证明了可行性;源代码如下

|  |
| --- |
| // 创建request对象  function createXMLHttpRequest() {  try {  return new XMLHttpRequest();//大多数浏览器  } catch (e) {  try {  return ActvieXObject("Msxml2.XMLHTTP");//IE6.0  } catch (e) {  try {  return ActvieXObject("Microsoft.XMLHTTP");//IE5.5及更早版本  } catch (e) {  alert("哥们儿，您用的是什么浏览器啊？");  throw e;  }  }  }  }  function ajax(option) {  var xmlHttp = createXMLHttpRequest();  if(!option.method) {//默认为GET请求  option.method = "GET";  }  if(option.asyn == undefined) {//默认为异步处理  option.asyn = true;  }  xmlHttp.open(option.method, option.url, option.asyn);  if("POST" == option.method) {  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");  }  xmlHttp.send(option.params);  xmlHttp.onreadystatechange = function() {  if(xmlHttp.readyState == 4 && xmlHttp.status == 200) {//双重判断  var data;  // 获取服务器的响应数据，进行转换！  if(!option.type) {//如果type没有赋值，那么默认为文本  data = xmlHttp.responseText;  } else if(option.type == "xml") {  data = xmlHttp.responseXML;  } else if(option.type == "text") {  data = xmlHttp.responseText;  } else if(option.type == "json") {  var text = xmlHttp.responseText;  data = eval("(" + text + ")");  }  option.callback(data);  }  };  };  ajaxutils.js |

# 4.XStream--将JavaBean转换为XML的小伎俩

## 4.1　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。

## 4.2　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); |

### 4.2.1　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）；

### 4.2.2　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：

|  |
| --- |
| **public** List<Province> getProvinceList(){  Province p1 = **new** Province();  p1.setName("北京");  p1.addCity(**new** City("东城区", "DongChengQu"));  p1.addCity(**new** City("昌平区", "ChangPingQu"));    Province p2 = **new** Province();  p2.setName("辽宁");  p2.addCity(**new** City("沈阳", "shenYang"));  p2.addCity(**new** City("葫芦岛", "huLuDao"));    **//创建集合存放JavaBean对象**  List<Province> list = **new** ArrayList<>();  list.add(p1);  list.add(p2);    **return** list;  } |
| @Test  **public** **void** listToXML() {  List<Province> proList = getProvinceList();  XStream xStream = **new** XStream();  String xml = xStream.toXML(proList);  System.***out***.println(xml);  } |
| 打印结果：  <list>//与对象名对应！  <cn.ajax\_basic.xstream.Province>//与对象名对应！与Province类名对应  <name>辽宁省</name>//与Province类的name属性对应  <cities>//与Province类的cities属性对应  <cn.ajax\_basic.xstream.City>//与City类名对应  <name>沈阳</name>//与City类的name属性对应  <description>shenyang</description>  </cn.ajax\_basic.xstream.City>  <cn.ajax\_basic.xstream.City>  <name>大连</name>  <description>dalian</description>  </cn.ajax\_basic.xstream.City>  </cities>  </cn.ajax\_basic.xstream.Province>  <cn.ajax\_basic.xstream.Province>  <name>吉林省</name>  <cities>  <cn.ajax\_basic.xstream.City>  <name>长春</name>  <description>changchen</description>  </cn.ajax\_basic.xstream.City>  <cn.ajax\_basic.xstream.City>  <name>白城</name>  <description>baicheng</description>  </cn.ajax\_basic.xstream.City>  </cities>  </cn.ajax\_basic.xstream.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！

### 4.2.3　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：

|  |
| --- |
| @Test  **public** **void** alias() {  List<Province> proList = getProvinceList();  XStream xStream = **new** XStream();  xStream.alias("china", List.**class**);  xStream.alias("province", Province.**class**);  xStream.alias("city", City.**class**);  String xml = xStream.toXML(proList);  System.***out***.println(xml);  } |
| <china>  <province>  <name>辽宁省</name>  <cities>  <city>//把City类对象的元素名称命名为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> |

### 4.2.4　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");//意思是:让Province类的name成员以属性形式出现！ |
| <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> |

### 4.2.5　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");//意思是:隐藏Province类的名为cities的集合元素 |
| <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> |

### 4.2.6　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"); //在生成的XML文档中省略City类的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> |

# 数据交换语言

在实际的开发过程中我们经常用到数据交换,那么什么是数据交换语言呢?

如同人类通过交流相互合作，计算机网络中计算机的相互合作也依赖于彼此的信息交流。人类间的相互交流可以通过当面交流，电话，短信，邮件，IM应用来实现，但这建立在两人使用共同语言的基础上。计算机间可以以TCP等方式传输数据，但我们需要预定义一种为双方计算机所接受的“语言“，方能使数据交流正常进行，这种语言我们称之为”数据交换语言“。

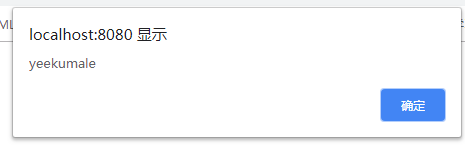
基于数据交互的基础性，各种开发语言基本都提供了自身的数据交换功能或者对通用交换语言标准的接口。但在不同的系统不同的语言间交换数据时，我们一般倾向于使用无关于平台及语言的数据交换语言。此类语言主要包括XML，JSON，YAML，Protobuf等，常用于接口调用，配置文件，数据存储等场景。我们以前最长使用的是xml，今天我们要认识一下使用最多的JSON。

## 没有JSON的时候怎么创建对象

我们在学习一样新知识的时候,经常问自己一下,如果没有出现这种新知识,那会是怎样的呢?理解了这个,有助于我们更深刻的理解新知识。同样在这里问自己一些JSON没出现之前js如何创建对象呢?数据交换是怎么进行的呢?

后一点实际上前面我们已经讲解过了,没有JSON时,数据交换主要依靠XML来进行,但是XML文件庞大,占用流量较多,在大流量访问的时候,这种弊端无法容忍的。那么JSON没有出现的时候，js创建对象又是如何进行的呢?

|  |
| --- |
| window.onload = **function**() {  **var** p = **new** Person("yeeku","male");  alert(p.name + p.sex);  }  //定义一个函数作为构造器  **function** Person(name,sex) {  **this**.name = name;  **this**.sex = sex;  } |



这种创建方法,每次要创建一个对象,先要定义一个该对象的构造函数,这种方法和Java及其类似,但是放在网页中,这种方式却很不合时宜,以为网页是要作为响应通过网络发送给客户端的,这种方式,无疑加大了网络的压力,所以更快捷,更简单,数据量夜更小的JSON就应运而出了。Json不仅在响应页面中有力压传统方式的优点,同时服务端的代码中,也因其简单和比XML数据格式小的优点得到了广大开发者的喜爱.

## 5.2 JSON

### 5.2.1 什么是JSON

JSON全称为JavaScript Object Notation（ JavaScript 对象表示法）， 是基于JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999的一个子集。是一种轻量级的数据交换格式。

JSON早在2000年就已经在使用了。2006年Douglas Crockford把JSON提交给IETF后，JSON开始作为Javascript的一个严格的子集得到推广。JSON用Javascript中的一些模式来表示**结构化数据**，这使得JSON对于Javascript开发者来说极为易读易写，极大地推动了JSON在web开发界的应用。目前JSON已被越来越多的API及Database选为输出的首选格式，在轻量级数据交换语言界独占鳌头。

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

### 5.2.2 JSON的特点

* JSON是一种轻量级的数据交换语言而非编程语言，其语法只支持字符串，数值，布尔值及null以及在此基础上的对象和数组。JSON轻量级的一部分体现即来源于此，因为JSON只处理了非常有限的数据类型，甚至没有一个日期数据类型。与之相对的重量级数据交换语言如XML则拥有更为复杂的数据类型及规则。有限的数据类型固然一定程度上限制了JSON在功能上的实现，却也使得JSON字符使用量极少，极大地降低了读写成本，特别适合互联网的数据传递。
* JSON采用与编程语言无关的文本格式，任何编程语言只要按JSON的规则都能解析JSON，这使得JSON的跨平台传递效率极高。JSON同时采用了大量类C语言（C，C++，C#，Java，JavaScript，Perl，Python等）的语言习惯，使得其可读性极高，容易编写编译。
* JSON是Javascript的一个子集，因而具有Javascript的原生支持。JSON可以用Javascript内建的方法直接进行解析，转换成Javascript也非常方便。
* JSON不是完整的标记语言，这使得其拓展性及可检索性较低。

### 5.2.3 对比XML和JSON

扩展标记语言 (Extensible Markup Language, XML) ，用于标记电子文件使其具有结构性的标记语言，可以用来标记数据、定义数据类型，是一种允许用户对自己的标记语言进行定义的源语言。 XML使用DTD(document type definition)文档类型定义来组织数据;格式统一，跨平台和语言，早已成为业界公认的标准。XML是标准通用标记语言 (SGML) 的子集，非常适合 Web 传输。XML 提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。  
 JSON(JavaScript Object Notation)一种轻量级的数据交换格式，具有良好的可读和便于快速编写的特性。可在不同平台之间进行数据交换。JSON采用兼容性很高的、完全独立于语言文本格式，同时也具备类似于C语言的习惯(包括C, C++, C#, Java, JavaScript, Perl, Python等)体系的行为。这些特性使JSON成为理想的数据交换语言。

* XML的优缺点  
   XML的优点  
  　　A.格式统一，符合标准；  
  　　B.容易与其他系统进行远程交互，数据共享比较方便。  
   XML的缺点  
  　　A.XML文件庞大，文件格式复杂，传输占带宽；  
  　　B.服务器端和客户端都需要花费大量代码来解析XML，导致服务器端和客户端代码变得异常复杂且不易维护；  
  　　C.客户端不同浏览器之间解析XML的方式不一致，需要重复编写很多代码；  
  　　D.服务器端和客户端解析XML花费较多的资源和时间。
* JSON的优缺点  
   JSON的优点：  
  　　A.数据格式比较简单，易于读写，格式都是压缩的，占用带宽小；  
  　　B.易于解析，客户端JavaScript可以简单的通过eval()进行JSON数据的读取；  
  　　C.支持多种语言，包括ActionScript, C, C#, ColdFusion, Java, JavaScript, Perl, PHP, Python, Ruby等服务器端语言，便于服务器端的解析；  
  　　D.在PHP世界，已经有PHP-JSON和JSON-PHP出现了，偏于PHP序列化后的程序直接调用，PHP服务器端的对象、数组等能直接生成JSON格式，便于客户端的访问提取；  
  　　E.因为JSON格式能直接为服务器端代码使用，大大简化了服务器端和客户端的代码开发量，且完成任务不变，并且易于维护。  
   JSON的缺点  
  　　A.没有XML格式这么推广的深入人心和喜用广泛，没有XML那么通用性；  
  　　B.JSON格式目前在Web Service中推广还属于初级阶段。

### 5.2.4 JSON对象的语法

我们在上文提过， JSON其语法只支持字符串，数值，布尔值及null以及在此基础上的对象和数组。下面我们分别介绍其在JSON中的表达形式。

JSON的语法

* 数据在名称/值对中
* 数据由逗号分隔
* 花括号保存对象
* 方括号保存数组

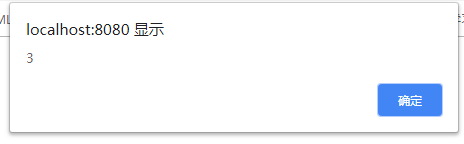
JSON能够表达的值

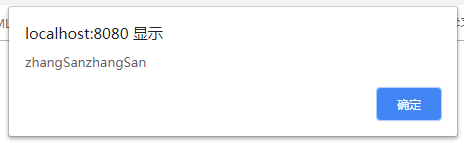
* [数字](http://baike.baidu.com/view/37407.htm" \t "_blank)（整数或浮点数）
* 字符串（在双引号中）
* 逻辑值（true 或 false）
* 数组（数组属性在方括号中）
* 对象（在花括号中）
* null

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

注意，key也要在双引号中！Js中使用eval()来解析JSON数据对象。当然除了上面的写法，json也有其他的写法，下面我们测试一下几种写法；

|  |
| --- |
| window.onload = **function** () {  **var** id = document.getElementById("h1");  id.onclick = **function**(){  **var** str = "1 + 2";  **var** pStr = eval("(" + str + ")");  alert(pStr);  }  } |
| id.onclick = **function**() {  **var** str = {"name":"zhangSan", "age":"18", "sex":"male", "hobby":["cf", "sj", "ddm"]};  alert(str.hobby);  } |
| id.onclick = **function**(){//数组  **var** str = "{\"name\":\"zhangSan\", \"age\":\"18\", \"sex\":\"male\", \"hobby\":[\"cf\", \"sj\", \"ddm\"]}";  alert(str.hobby);  } |
| id.onclick = **function** () {//对象  **var** person = {"name":"zhangSan","getName":**function**(){  **return** **this**.name;  }};  alert(person.name + person.getName());  } |
| id.onclick = **function**() {  **var** person = {"name":"zhangSan","zhangSan":{"sex":"male","age":18}};  alert(person.zhangSan.age);  } |



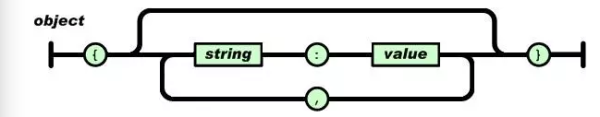


从上面可以看出在js中JSON中的写法和对应解析的方法有两种,json中元素的属性还可以是函数,json字符串中可以是一个运算.

### 5.2.5 JSON对数据格式的表达

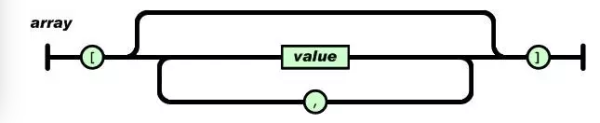
#### 1 对象

对象是一个无序的“键－值”集合。一个对象以“{”（左括号）开始，“}”（右括号）结束。每个“键”后跟一个“:”（冒号）；“键－值”之间使用“,”（逗号）分隔。



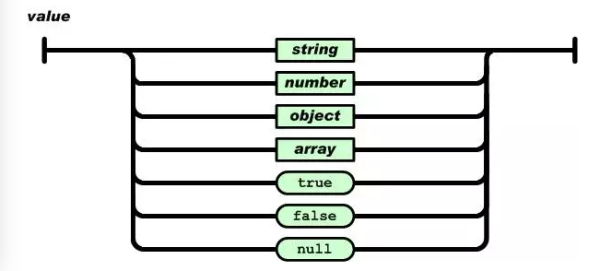
#### 2. 数组

数组是值（value）的有序集合。一个数组以“[”（左中括号）开始，“]”（右中括号）结束。值之间使用“,”（逗号）分隔。



#### 3. 值

值（value）可以是双引号括起来的字符串（string）、数值(number)、true、false、 null、对象（object）或者数组（array）。这些结构可以嵌套。



熟悉JSON后我们能很轻松地写出json对象和提取出其中的信息。JSON的解析工具浩如繁星，如JavaScript中的eval()函数，这里也不详细展开了。下面再介绍.

额外安利一个检验JSON格式的网址[JSON在线解析及格式化验证 - JSON.cn](http://link.zhihu.com/?target=http://www.json.cn" \t "https://zhuanlan.zhihu.com/p/_blank)，不仅能在线检验JSON代码格式的正确，同时能够给出相应的修改意见。如果你的代码没有问题，还会自动帮你排版为更易读的版式，强迫症福音！



## 5.3 JSON与XML的对比

XML全称为extensible markup language，是一种类似于HTML的语言。XML没有预先定义的标签，使用DTD(document type definition)文档类型定义来组织数据，是业界公认的标准数据交换语言之一。

### 5.3.1 JSON与XML的区别主要包括：

#### 1. 标记语言

JSON与XML最大的不同在于XML是一个完整的标记语言，而JSON不是。这使得XML在程序判读上需要比较多的功夫。XML利用标记语言的特性提供了绝佳的延展性（如XPath），在复杂数据存储，扩展及高级检索方面具备对JSON的优势，但同时也意味着冗余信息较多。对于需要高性能交互的应用，过多的冗余信息则成了缺点。

#### 2. 字符占用量

XML由于其标记语言的特性具有大量的冗余信息，这使得其字符占用量超过同等信息量的JSON文件。以下我们提供一个例子：

为了提供一个人的相关信息，我们先以XML语言编写相关文件：



然后我们再以JSON文件编写一次：



可以看出，字符数仅为XML编码所用约1/2。

#### 3. 编译方式

XML主流的编译方式包括DOM, SAX,JDOM 和DOM4J四种，DOM拥有较强的遍历能力和检索能力，但需要将文件整体读入内存进行处理，对计算机性能和内存的要求较高。SAX处理的优点非常类似于流媒体的优点。分析能够立即开始，而不是等待所有的数据被处理。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中。这对于大型文档来说是个巨大的优点。 而目前JSON的编译方式还比较单一，与DOM较为类似。因而在大规模数据的处理上有一定劣势。

### 5.3.2 JSON vs XML -- 应用场景分析

从以上分析我们可以得出，JSON由于其非标记语言的设计思路具有字符利用率高，解析速度快的优点，而XML作为一种完整的标记语言虽然继承了标记语言的冗余度，在读写成本上逊色于JSON，但在数据的检索与复用上对JSON有较大的优势。下面我们就不同的应用场景分析对JSON和XML的选择。

#### 1. 开放式平台

此类应用场景实例包括微博，Facebook等。此类场景的特点是：

1. 调用方不完全可控，而且是针对公网的，你可能不知道是谁、是什么语言、是什么方式在调用你提供的数据接口；

2. 接口访问量一般都非常大，要求具有很高的性能和吞吐量；

3. 需要考虑安全问题，外部提交的数据可能不是合法的。

所以在这种情况下，需要考虑数据传输的带宽消耗和数据交换协议的易用性，以及多语言支持程度。以前对于html页面使用的JavaScript接口调用一般都使用XML格式，最近几年几乎都转成了JSON格式了，因为JSON传输量更小，比XML更加容易使用。 而对于开放平台，由于使用的场景很多，所以需要提供多种交换协议格式。基本上都会提供XML和JSON。

大型互联网公司像Twitter， Facebook和LinkedIn的很多原始数据仍然使用XML而不是JSON存储，像可编程Web和其他数据表明，**XML仍然是API的主要数据格式**，但是“大JSON”正在快速上升。 Twitter的API大约两年前开始就只支持JSON了。Foursquare也跟进了。

有业界人士表示：“当涉及到数据API服务时， XML仍然是最常用的格式，但JSON是的增长更快。尽管还有很多XML格式的API，但最近的API ，越来越倾向于使用JSON格式。这样的例子还有很多…… 企业正在迅速从XML迁移到JSON。”

#### 2. 内部服务

对于一个大型系统来说，内部服务的数据交换无处不在。从最基本和常见的数据库数据交换、memcached缓存数据交换、消息队列的数据交换到系统之间使用的RPC服务框架等等，都可以算作内部服务的数据交换。

内部服务的特点是不用考虑防火墙，不对外开放，速度快（基本无带宽成本）。

内部服务的数据交换协议的选择空间非常大，一般需要考虑：

1. 数据交换语言的性能

2. 是否需要跨语言支持

3. 数据交换协议的消息体大小

以数据库为例，传统数据库如Oracle,Postgres等均支持XML，因为在数据量的吞吐上数据交换语言的性能及消息体大小并不会成为瓶颈，同时传统关系型数据库对数据的可检索性及复用性要求较高，这使得XML成为一种理想的选择。

而随着大数据和NoSQL型数据库的发展，JSON在数据库上的应用越来越普遍，因为

1. JSON的设计思路就是从存储无关系数据的角度出发的，这与NoSQL型数据库的设计出发点相符；

2. 大数据时代数据规模的激增及分布式存储使得数据交换语言的性能及消息体大小成为制约数据库性能的极大瓶颈。

这是有很多知名的例子，像MongoDB，CouchDB，和Riak。这三种数据库都基于JSON，横向可扩展，由Web驱动。

其他的例子比比皆是：亚马逊DynamoDB的架构是完全基于REST/JSON的。 Neo4j，图形数据库，有一个REST/JSON API，没有对应XML的支持。 HBase的的REST架构目前支持XML，但这种支持正走在被废弃的路上。

结语

JSON作为一种轻量级的数据交换语言，因其较低的读写成本，较好的跨平台性以及其数据结构与当前数据交流要求的契合在web大数据时代得到了越来越广泛的应用。但JSON并不是尽善尽美的，从其语法结构的设计我们可以看出JSON是基于”键－值“型数据的一种较为简单的数据交换语言，单一的数据结构在提供了较低读写成本的同时也使得JSON的检索效率及可复用性较差，使其在复杂关系性数据的存储上具有一定劣势。在实际应用中我们仍需根据数据类型，硬件性能及数据交流实际需求在以JSON为代表的轻量级数据交换语言和以XML为代表的重量级数据交换语言间做出合理的选择。

* 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。

### 5.3.3 代码分析

在上面我们从理论上分析了一下XML和JSON的优劣,下面我们还是通过代码来说明一下;

通过Servlet传输一个Student对象到前台,对比XML和JSON的优劣,json2与ajax7页面

|  |
| --- |
| **public** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  resp.setContentType("text/xml;charset=utf-8");  String xml = "<students>" +  "<student number='1001'>" +  "<name>zhangSan</name>" +  "<age>18</age>" +  "<sex>male</sex>" +  "</student>" +  "</students>";  System.***out***.println(xml);  System.***out***.println(req.getParameter("name"));  resp.getWriter().print(xml);  } |
| <script type=*"text/javascript"* src=*"*<c:url value=*'/ajax-lib/ajaxutils.js'*/>*"*></script>  <script type=*"text/javascript"*>  window.onload = **function**() {//文档加载完毕后执行  **var** btn1 = document.getElementById("xml");  **var** btn2 = document.getElementById("json");  btn1.onclick = **function**() {//给按钮的点击事件注册监听  ajax({  url:"<c:url value='/XMLServlet'/>",  params:"name=Yorick",  method:"POST",  type:"xml",  callback:**function**(data){  **var** ele = data.getElementsByTagName("student")[0];  **var** number = ele.getAttribute("number");//获取元素名为number的属性值  **var** name;  **var** age;  **var** sex;  // 处理浏览器的差异  **if**(window.addEventListener) {  name = ele.getElementsByTagName("name")[0].textContent;//其他浏览器  } **else** {  name = ele.getElementsByTagName("name")[0].text;//IE支持  }  **if**(window.addEventListener) {  age = ele.getElementsByTagName("age")[0].textContent;//其他浏览器  } **else** {  age = ele.getElementsByTagName("age")[0].text;//IE支持  }  **if**(window.addEventListener) {  sex = ele.getElementsByTagName("sex")[0].textContent;//其他浏览器  } **else** {  sex = ele.getElementsByTagName("sex")[0].text;//IE支持  }  **var** text = number + ", " + name + ", " + age + ", " + sex;  document.getElementById("h1").innerHTML = text;  }  });    };  };  </script>  </head>  <body>  <h1 id=*"h1"*></h1>  <button id=*"xml"*>单击1</button>  </body> |
| @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  req.setCharacterEncoding("utf-8");  resp.setContentType("text/html;charset=utf-8");  String json = "{\"students\":[{\"number\":\"1001\",\"name\":\"zhangSan\",\"age\":18,\"sex\":\"male\"}]}";  System.***out***.println(req.getParameter("a"));  resp.getWriter().print(json);  } |
| <script type=*"text/javascript"* src=*"*<c:url value=*'/ajax-lib/ajaxutils.js'*/>*"*></script>  <script type=*"text/javascript"*>  window.onload = **function**() {  document.getElementById("btn").onclick = **function**() {  alert("btn");  ajax({  url:"<c:url value='/JSONServlet'/>?a=A",  type:"json",  callback:**function**(data){  document.getElementById("h1").innerHTML = data.students[0].name;  }  });  }  }  </script> |



通过上面的代码,我们知道

* + 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。

## 5.4 把Java对象转换成JSON

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。

### 5.4.1　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

### 5.4.2　json-lib中的核心类

在json-lib中只有两个核心类：

* + - JSONObject；
    - JSONArray；

#### 5.4.2.1　JSONObject

JSONObject类本身是一个Map，所以学习它很方便。

|  |
| --- |
| @Test  **public** **void** testJSONObject() {  JSONObject jo = **new** JSONObject();  jo.put("name", "Yorick");  jo.put("age", 18);  jo.put("sex", "male");  System.***out***.println(jo.toString());  } |
| @Test  **public** **void** fromObject() {  //把JavaBean对象转换成json  JSONObject jo = JSONObject.*fromObject*(**new** Person("Yorick", 19, "male"));  System.***out***.println(jo.toString());  } |
| @Test  **public** **void** fromObjectMap() {  Map<String, String> map = **new** HashMap<>();  map.put("name", "aa");  map.put("sex", "female");  //把map装换为json  JSONObject jo = JSONObject.*fromObject*(map);  System.***out***.println(jo.toString());  } |
| @Test  **public** **void** xmlToJson() {  String xml = "<person><name>zhaoLiu</name><age>59</age><sex>female</sex></person>";  XMLSerializer xs = **new** XMLSerializer();  //把xml转换成JSON，这需要导入xom-1.1.jar  JSONObject jo = (JSONObject) xs.read(xml);  System.***out***.println(jo.toString());  } |

#### 5.4.2.2　JSONArray

JSONArray本身是一个List，所以使用起来很方便。

|  |
| --- |
| @Test  **public** **void** objectJson() {  JSONArray ja = **new** JSONArray();  Person p1 = **new** Person("Yorick1", 26, "male");  Person p2 = **new** Person("Yorick2", 27, "female");  ja.add(p1);  ja.add(p2);  System.***out***.println(ja.toString());  } |
| @Test  **public** **void** listJson() {  Person p1 = **new** Person("zhangSan", 18, "male");  Person p2 = **new** Person("liSi", 23, "female");  List<Person> list = **new** ArrayList<>();  list.add(p1);  list.add(p2);  JSONArray ja = JSONArray.*fromObject*(list);  System.***out***.println(ja.toString());  } |
| @Test  **public** **void** arrayJson() {  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());  } |

### 5.4.3　JS解释服务器发送过来的JSON字符串

前面介绍了各种数据类型转换为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。详见代码5.3.3

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