# 1. AJAX的引入

## 1.1 传统的Web应用

之前我们所做的开发都是传统Web应用，基本的流程如下：

客户端先向服务器请求网页资源，服务器接到请求后，根据请求的不同将带有不同数据信息的页面响应给客户端，最后客户端就能看到所请求资源。

但是这样有一个很突出的问题，就是每进行一次请求都将会重新刷新页面。即使两次请求返回的页面只有很小的改动（有时只需获得一些验证数据等），服务器也会相应全新的内容并且页面会刷新。这不利于用户体验，而且消耗网络资源，特别是网络情况不好时尤为突出。

## 1.2 AJAX的引入

为了解决传统Web应用的上述缺点，人们开始使用AJAX技术。AJAX（Asynchronous JavaScript And XML）意思是“异步的JS和XML”，这是2005年Google推出的。

AJAX使得浏览器可与服务器进行异步通信，即可实现在不刷新页面的情况下，请求并获得服务器的响应数据，再处理返回的结果按需操作DOM从而实现“局部刷新”网页内容。

AJAX提高了效率并提供了更友好的交互方式，因此现在的浏览器均内置了异步通信对象（XmlHttpRequest）以便开发者使用。但我们要知道，并不是学习了AJAX后，项目中就要一律使用AJAX，这是根据需要决定的。

现在AJAX使用得非常广泛，例如注册账号时，输入用户名后，应该异步请求服务器查看此用户名是否被占用，提前通知。再比如输入验证码时也应该提前告知用户输入是否正确，还有多级菜单联动也可使用AJAX技术。

## 1.3 理解同步和异步

我们说AJAX是异步的，传统方式是同步的。在Java集合中，我们也提到了同步和异步，那么同步和异步到底是什么概念呢？

传统方式的同步交互和AJAX异步交互可以这样理解：

同步交互：指发送一个请求，需要等待返回，然后才能够发送下一个请求，有个等待过程；

异步交互：指发送一个请求，不需要等待返回，随时可以再发送下一个请求，即不需要等待。浏览器的异步交互是由AJAX引擎来实现的。

因此，同步和异步的区别是一个按序同步执行，需要等待，一个异步执行，无需等待。实际上在JS中，也有许多异步方法（例如setTimeout等），这是需要注意的，由于他们的异步执行，因此有时无法确定那部分代码先执行，有可能需要修改我们预想的代码。

关于同步和异步，还可参见博客“http://blog.csdn.net/cqkxboy168/article/details/9026205/”，讲述了底层的操作系统进程和同步、异步相关知识，概念大部分是相通的（可以说最初的同步异步概念是从操作系统中来的）。此博客内容已经被我优化显示在附录1中，可以学习。

# 2. AJAX开发实例

## 2.1 AJAX开发步骤

一般采用以下步骤来实现AJAX请求：

（1）创建异步请求对象XMLHttpRequest，调用open(method, url)方法设置请求的方式和请求路径；对于POST方式请求，还需调用setRequestHeader()设置“content-type”请求头为“application/x-www-form-urlencoded”；

（2）调用send(请求参数)方法请求服务器资源。例如send("name=zs")。若无请求参数，则使用send(null)。

（3）在onreadystatechange事件中可监听服务器响应的状态变化，其中可根据响应的状态得到服务器返回的数据结果。可根据结果通过JS按需操作DOM实现页面变化，即“局部刷新”。

注意：AJAX只负责请求服务器并获得响应资源；服务器端负责处理AJAX请求的Servlet或Action不能转发或重定向到web页面，因为这样还是会使页面全部刷新，而只能以流的方式响应数据给浏览器。

## 2.2 案例一：异步获得text/html数据（GET请求）

要求：页面点击一次“显示服务器时间”按钮，就会触发一次AJAX请求服务器时间，并将结果显示在页面上，要实现局部刷新。

首先完成服务器端代码（基于Servlet。对于Struts，也可通过调用Servlet的API实现。但是需要注意的是，Struts的Action方法虽然返回值类型是String类型，但是只能返回null，并且不需配置result节点。否则若返回有值的字符串，就会刷新整个页面）：

|  |
| --- |
| @WebServlet(urlPatterns = **"/queryDate"**) **public class** DateServlet **extends** HttpServlet {  @Override  **protected void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  PrintWriter pw = resp.getWriter();  pw.write(**new** Date().toString());  pw.close();  } } |

HTML页面如下（页面现无需使用JSP）：

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="zh-CN"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>获取服务器时间</**title**> </**head**> <**body**>  *<!-- 该p标签中就用于显示服务器时间 -->* <**p id="serverTime"**></**p**>  *<!-- 由于并非表单提交，所以不必使用form标签 -->* <**button id="showTimeBtn" type="button"**>显示服务器时间</**button**> </**body**> <**script**>  *// 按钮的点击事件中进行AJAX请求* **document**.querySelector(**"#showTimeBtn"**).**onclick** = **function**() {  *// 1. 创建异步对象* **var** xhr = **new** XMLHttpRequest();  *// 2.设置请求方式和路径* xhr.open(**'GET'**, **'http://localhost:8080/demo/queryDate'**);  *// 3. 发送请求* xhr.send(**null**);  *// 4.在onreadystatechange事件中监听服务器返回状态。* xhr.**onreadystatechange** = **function**() {  *// 若状态码为4，并且响应码为200，说明返回数据成功* **if** (4 == xhr.**readyState** && 200 == xhr.**status**) {  *// Servlet默认响应的数据类型是"text/html"* **var** data = xhr.**responseText**;  *// DOM操作，将数据显示在"serverTime"的标签中* **document**.querySelector(**"#serverTime"**).**innerText** = data;  }  }  } </**script**> </**html**> |

上述代码中，判断了AJAX的“状态码”是否等于4，这个状态码一共有0~4五种数值：

|  |  |
| --- | --- |
| AJAX返回的状态码 | 含义 |
| 0 | 表示AJAX对象已经创建完毕，但还未调用open()设置参数 |
| 1 | 表示AJAX对象调用了open方法，但还未发送请求 |
| 2 | 表示已调用send()方法，但请求还没到达服务器 |
| 3 | 表示正在等待服务器返回数据 |
| 4 | 表示AJAX对象已经收到服务器响应信息 |

因此重点是我们要先判断状态码是否为4。但状态码为4只代表服务器返回了数据，我们还要知道服务器返回的本次请求的响应码，响应码为200就表示成功，而如果是404则表示未找到页面等。

有时，还要在请求路径上加上一个值是随机数的参数，例如："time=" + Math.random()。这是因为在微软的IE和Edge浏览器中，如果本次ajax请求和上次请求的路径一致，则不会重新请求服务器，这样就导致第一次请求以后不能“局部刷新”了。因此我们给路径加一个参数使得每次请求路径都会变化。而在Chrome和Firefox中则没有这样的问题。

当然，我们也可在Servlet中直接返回一个jsp页面，然后AJAX拿到页面数据否，可把整个页面数据加载到一个div中显示，这样相当于无刷新地请求了一个页面。

## 2.3 案例二：异步获得XML数据（POST请求）

还能使用异步获得XML格式数据。本次使用POST方式请求。

案例要求：服务器返回的XML数据中描述了学生及其状态（正常/异常），若是正常状态，文字显示绿色，否则显示红色。

（1）服务器端只返回需要的XML数据（不包含样式信息）

|  |
| --- |
| **package** com.ssh.action;  @WebServlet(urlPatterns = **"/queryStudent"**) **public class** StudentServlet **extends** HttpServlet {  @Override  **protected void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  *// 可接收到param参数* req.setCharacterEncoding(**"utf-8"**);  System.***out***.println(req.getParameter(**"param"**));  *// 设置响应的格式为xml* resp.setContentType(**"text/xml;charset=utf-8"**);  *// 创建DOM对象* Document dom = DocumentHelper.*createDocument*();  Element root = dom.addElement(**"students"**);  *// 第一个学生* Element stu1 = root.addElement(**"student"**);  stu1.addAttribute(**"name"**, **"张三"**);  stu1.addAttribute(**"status"**, **"正常"**);  *// 第二个学生* Element stu2 = root.addElement(**"student"**);  stu2.addAttribute(**"name"**, **"李四"**);  stu2.addAttribute(**"status"**, **"异常"**);  *// 输出。* XMLWriter writer = **new** XMLWriter(resp.getWriter());  writer.write(root);  writer.close();  } } |

（2）HTML页面渲染数据：

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="zh-CN"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>获取服务器时间</**title**>  <**style**>  .**normal** {  **color**: **red**;  }  .**abnormal** {  **color**: **green**;  }  </**style**> </**head**> <**body**>  *<!-- div中显示学生信息 -->* <**p id="stuInfo"**></**p**>  <**button id="showInfoBtn" type="button"**>显示学生</**button**> </**body**> <**script**>  **document**.querySelector(**"#showInfoBtn"**).**onclick** = **function**() {  **var** xhr = **new** XMLHttpRequest();  *// 这个路径加上了随机的参数* xhr.open(**'POST'**, **'http://localhost:8080/demo/queryStudent?time='** + ***Math***.random());  *// POST请求需要设置请求头* xhr.setRequestHeader(**"content-type"**, **"application/x-www-form-urlencoded"**);  *// 若需要传递参数，则在send中写明(这是一个无用参数，只是例子)* **var** param = **'张三'**;  xhr.send(**'param='** + param);  xhr.**onreadystatechange** = **function**() {  **if** (4 == xhr.**readyState** && 200 == xhr.**status**) {  *// 用responseXML接收XML数据* **var** xml = xhr.**responseXML**;  *// 同样可用DOM方法操作xml中的数据。  // 获得student节点* **var** studentElems = xml.querySelectorAll(**"student"**);  *// 遍历student* **for**(**var** i = 0; i < studentElems.**length**; i++) {  *// 创建一个p标签存放学生信息* **var** student = **document**.createElement(**"p"**);  *// 添加姓名信息* student.append(studentElems[i].getAttribute(**'name'**));  *// 创建span标签存放状态信息* **var** status = **document**.createElement(**"span"**);  status.**innerText** = studentElems[i].getAttribute(**'status'**);  *// 根据状态加上不同的class属性以显示颜色* **if** (status.**innerText** == **'正常'**) {  status.setAttribute(**"class"**, **"normal"**);  } **else** {  status.setAttribute(**"class"**, **"abnormal"**);  }  *// 再给student节点添加状态信息* student.append(status);  *// 在页面中添加* **document**.querySelector(**"#stuInfo"**).append(student);  }  }  }  } </**script**> </**html**> |

这里解释一下为什么要为POST请求设置Content-Type请求头。这是因为对于GET请求，参数就在URL中，Servlet通过request.getParameter()拿到参数值是没问题的。而对于POST请求，response.getParameter方法只能拿到表单类型为application/x-www-form-urlencoded的参数值，AJAX不是这样的表单类型，但是可以通过设置Content-Type请求头达到相同的效果，以此让POST请求能够通过getParameter方法拿到请求参数（可参考文件上传一章对表单类型的解释）。

## 2.4 案例三：异步获得JSON数据

最近在Web开发中，JSON作为数据传输格式也变得十分流行。我们也可使用AJAX来请求处理JSON数据。

首先需要服务器返回JSON格式的字符串。在Java中，有专门的第三方jar包可直接把对象转换成JSON字符串（或相反），方便开发者传递JSON数据。常用的jar包有Jackson、Gson（Google开发的）和fastjson（alibaba开发的）。这里我们在项目中使用fastjson将数据对象转成JSON字符串以流的方式响应给客户端。fastjson的github地址：<https://github.com/alibaba/fastjson> 。

说明：JSON中常使用键值对形式，Java中对象的属性名和属性值就会转换成键值对形式。有时，也常用Map集合封装数据，因为Map对象也会转换为JSON的键值对形式。

另外，本案例采用Struts作为示例。实现功能：省市区选择的三级联动，比如选择了A省，就会异步请求A省下城市数据，当选择一个城市时，又会请求该城市下的区信息。

Struts代码：

|  |
| --- |
| **package** com.ssh.action;  **import** com.alibaba.fastjson.JSON; **import** com.opensymphony.xwork2.ActionSupport; **import** org.apache.struts2.ServletActionContext; **import** org.springframework.context.annotation.Scope; **import** org.springframework.stereotype.Controller;  **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.util.ArrayList;  @Controller *// Action也得加入容器中* @Scope(**"prototype"**) **public class** DemoAction **extends** ActionSupport {  *// 接收的参数。省市名* **private** String **province**;  **private** String **city**;   *// 查询某省城市* **public** String queryCity() **throws** IOException {  ArrayList<String> cityList = **new** ArrayList<>();  **if** (**province**.equals(**"A省"**)) {  cityList.add(**"甲市"**);  cityList.add(**"乙市"**);  }  **if** (**province**.equals(**"B省"**)) {  cityList.add(**"丙市"**);  }  *// 将对象转成JSON字符串* String json = JSON.*toJSONString*(cityList);  *// 获得response对象输出* HttpServletResponse response = ServletActionContext.*getResponse*();  *// 设置编码并输出* response.setContentType(**"text/html;charset=UTF-8"**);  response.getWriter().write(json);  *// 为了使页面不跳转，返回null。* **return null**;  }   *// 查询区信息* **public** String queryDistrict() **throws** IOException {  ArrayList<String> district = **new** ArrayList<>();  **if**(**city**.equals(**"甲市"**)) {  district.add(**"一区"**);  district.add(**"二区"**);  }  **if** (**city**.equals(**"乙市"**)) {  district.add(**"三区"**);  district.add(**"四区"**);  }  **if** (**city**.equals(**"丙市"**)) {  district.add(**"五区"**);  district.add(**"六区"**);  }  String json = JSON.*toJSONString*(district);  HttpServletResponse response = ServletActionContext.*getResponse*();  response.setContentType(**"text/html;charset=UTF-8"**);  response.getWriter().write(json);  **return null**;  }   **public** String getProvince() {  **return province**;  }   **public void** setProvince(String province) {  **this**.**province** = province;  }   **public** String getCity() {  **return city**;  }   **public void** setCity(String city) {  **this**.**city** = city;  } } |

配置（基于Spring）：

|  |
| --- |
| *<!-- 不需写result节点 -->* <**action name="queryCity" class="demoAction" method="queryCity"** /> <**action name="queryDistrict" class="demoAction" method="queryDistrict"** /> |

HTML页面：

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="zh-CN"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>三级联动</**title**> </**head**> <**body**>  <**select id="provinceSelect"**>  <**option**>选择省份</**option**>  <**option**>A省</**option**>  <**option**>B省</**option**>  </**select**>  <**select id="citySelect"**>  <**option**>选择城市</**option**>  </**select**>  <**select id="districtSelect"**>  <**option**>选择区</**option**>  </**select**> </**body**> <**script**>  **var *provinceSelect*** = **document**.querySelector(**"#provinceSelect"**);  **var *citySelect*** = **document**.querySelector(**"#citySelect"**);  **var *districtSelect*** = **document**.querySelector(**"#districtSelect"**);  *// 省份选择框的改变事件* ***provinceSelect***.onchange = **function**() {  *// 首先把下两级菜单清空。只保留一项提示选择的菜单。  // 因此这里只要设置长度为1即可* ***citySelect***.**options**.length = 1;  ***districtSelect***.**options**.length = 1;  *// 获得选中的文本* **var** province = **this**.**selectedOptions**[0].**innerText**;  **console**.log(province);  **if**(**"选择省份"** != province) {  *// 进行AJAX请求城市* **var** xhr = **new** XMLHttpRequest();  xhr.open(**'POST'**, **'queryCity.action?time='** + ***Math***.random());  xhr.setRequestHeader(**"content-type"**, **"application/x-www-form-urlencoded"**);  xhr.send(**'province='** + province);  xhr.**onreadystatechange** = **function**() {  **if** (xhr.**readyState** == 4 && xhr.**status** == 200) {  *// 返回的JSON也是文本。  // 但是需要将文本解析成JSON对象，要调用浏览器提供的JSON.parse()方法  // 得到的就是数组对象* **var** cityArray = ***JSON***.parse(xhr.**responseText**);  *// 遍历数组，将结果添加到城市列表中* **for** (**var** index in cityArray) {  *// 创建option元素并设置值* **var** option = **document**.createElement(**"option"**);  option.**innerText** = cityArray[index];  *// 在城市选择框中加入这个option。* ***citySelect***.append(option);  }  }  }  }  }   *// 城市选择框的改变事件* ***citySelect***.onchange = **function** () {  *// 先清空区选择框，只保留一项* ***districtSelect***.**options**.length = 1;  *// 获得选中的文本* **var** city = **this**.**selectedOptions**[0].**innerText**;  *// AJAX请求* **var** xhr = **new** XMLHttpRequest();  xhr.open(**'POST'**, **'queryDistrict.action?time='** + ***Math***.random());  xhr.setRequestHeader(**"content-type"**, **"application/x-www-form-urlencoded"**);  xhr.send(**"city="** + city);  xhr.**onreadystatechange** = **function** () {  **if** (xhr.**readyState** == 4 && xhr.**status** == 200) {  *// 得到并解析成JSON对象* **var** district = ***JSON***.parse(xhr.**responseText**);  **for** (**var** index in district) {  **var** option = **document**.createElement(**"option"**);  option.**innerText** = district[index];  ***districtSelect***.append(option);  }  }  }  } </**script**> </**html**> |

## 2.5 XMLHttpRequest总结

（1）xhr中的方法：open()、setRequestHeader()和send()。其中open方法原型如下：

open(method, url, 可选的boolean)。最后一个参数默认为true，表示AJAX为异步请求，但是也可设置成false，这时AJAX就为同步请求。页面也不会刷新，只是该请求操作会立即执行。但是不推荐这样做。

异步就是该代码可以和其他代码同时执行，而同步就是该代码不能与其他代码同时执行，必须先执行完本代码后，其他代码才能执行。

比如我们在100行进行异步请求B数据，在洗异步请求的onreadystatechange事件中是能拿到B数据的，但是若我们在此事件范围之外的200行想拿到这个B数据，可能是拿不到的。这是因为异步代码并不是按顺序执行的，这是需要注意的。若确有此需求，就要使用同步请求了。

setRequestHeader()方法其实就用于设置XHR请求头的。例如POST中设置的Content-Type。

（2）xhr中的事件：onreadystatechange。onreadystatechange可指向一个函数，用于做事件处理（即注册事件处理）。XHR对象的readyState的每次变化都会触发onreadystatechange指向的事件处理器。

（3）xhr中的属性：

readyState：标识着XMLHttpRequest对象的当前状态。

responseText：接收服务器返回的文本类型的正文数据。

responseXML：接收服务器返回的XML类型的正文数据（返回的是DOM对象）。

status：接收服务器返回的响应状态码

statusText：接收服务器返回的响应码描述

常用AJAX传输HTML、XML和JSON数据。需要注意的是，尽量只用AJAX传递有用的业务数据，然后JavaScript根据数据操作DOM来显示。而不要传输一些与数据无关的东西，比如传递HTML标签元素、样式信息等，这些应该是在前端JS和CSS做的。

# 3. 在jQuery中使用AJAX

jQuery中封装了许多JS常用功能，简化了程序开发。jQuery中也对AJAX操作进行了封装，使用jQuery可提高效率。

## 3.1 使用$.get和$.post方法

$.get和$.post分别用于以GET和POST方式发送AJAX请求，其一般格式是：

$.get/post(url, sendData, function(backData, textStatus, ajax) {

...

})

参数解释如下：

（1）url和sendData分别表示请求路径和要上传的参数，注意sendData的数据格式是JSON格式。

（2）参数3是一个回调函数，该函数其中可接收三个参数，参数1接收到AJAX返回的数据，参数2接收到AJAX返回状态的文本描述，如“success”、“error”等，参数3就是接收到XMLHttpRequest对象，也可直接使用该对象获取数据。根据JS特性，若只要使用参数1，那么只接收参数1即可，但是若只要使用参数3，则前两个参数也要写上，因为是按照顺序的，另外，参数名称是不重要的。

说明：不需用随机参数解决IE的问题，因为jQuery自动处理兼容性问题。

例如：

|  |
| --- |
| **$**.post(**'queryCity.action'**, {  **province**: **'A省'** }, **function** (backData, textStatus, ajax) {  **var** data = ***JSON***.parse(backData);  **for** (**var** index in data) {  **console**.log(data[index]);  }  **console**.log(typeof data);  **console**.log(textStatus); *// 显示success* **console**.log(ajax.**readyState**); *// 显示响应码为4* }); |

## 3.1 使用load方法

由元素调用load()方法，可将服务器返回结果直接添加到该对象的标签中。

一般格式：

jQuery对象.load(url, sendData, function(backData, textStatus, ajax) {

...

})

实例代码：获得服务器时间：

|  |
| --- |
| $(**"#showTime"**).load(**'getTime.action'**); |

就是简单的一句话，让showTime元素中显示获得数据，这里没有特殊需要，就没有使用其他参数，这是load()最简单的形式。

## 3.2 使用ajax方法

一般使用ajax()方法对请求进行较多的控制。ajax中所有的参数都是JSON的格式，较完整的格式是：

$.ajax({

type: '请求方式',

url: '请求路径,'

data: JSON格式数据, // 要上传的参数

success: function(backData, textStatus, ajax) {

成功返回后执行的方法。

}

});

示例代码：请求XML数据

|  |
| --- |
| $.ajax({  **type**: **'POST'**,  **url**: **'queryXML.action'**,  **data**: {  **school**: **'A校'** },  success: **function**(backData) {  **var** $xml = $(backData);  **console**.log($xml.find(**'student'**));  } }); |

# 附录1 “并发 并行 同步 异步 多线程的区别”

（本章整理自博客http://blog.csdn.net/cqkxboy168/article/details/9026205/）

1. 并发：在操作系统中，是指一个时间段中有几个程序都处于已启动运行到运行完毕之间，且这几个程序都是在同一个处理机上运行。其中两种并发关系分别是同步和互斥。

2. 互斥：进程间相互排斥的使用临界资源的现象，就叫互斥。

3. 同步：进程之间的关系不是相互排斥临界资源的关系，而是相互依赖的关系。进一步的说明：就是前一个进程的输出作为后一个进程的输入，当第一个进程没有输出时第二个进程必须等待。具有同步关系的一组并发进程相互发送的信息称为消息或事件。

其中并发又有伪并发和真并发，伪并发是指单核处理器的并发，真并发是指多核处理器的并发。

4. 并行：在单处理器中多道程序设计系统中，进程被交替执行，表现出一种并发的外部特种；在多处理器系统中，进程不仅可以交替执行，而且可以重叠执行。在多处理器上的程序才可实现并行处理。从而可知，并行是针对多处理器而言的。并行是同时发生的多个并发事件，具有并发的含义，但并发不一定并行，也亦是说并发事件之间不一定要同一时刻发生。

5. 多线程：多线程是程序设计的逻辑层概念，它是进程中并发运行的一段代码。多线程可以实现线程间的切换执行。

6. 异步：异步和同步是相对的，同步就是顺序执行，执行完一个再执行下一个，需要等待、协调运行。异步就是彼此独立,在等待某事件的过程中继续做自己的事，不需要等待这一事件完成后再工作。线程就是实现异步的一个方式。异步是让调用方法的主线程不需要同步等待另一线程的完成，从而可以让主线程干其它的事情。

异步和多线程并不是一个同等关系,异步是最终目的,多线程只是我们实现异步的一种手段。异步是当一个调用请求发送给被调用者,而调用者不用等待其结果的返回而可以做其它的事情。实现异步可以采用多线程技术或则交给另外的进程来处理。