[Ajax的组成部分 2](#_Toc260184364)

[各种Ajax框架概述 2](#_Toc260184365)

[Ajax基本流程 3](#_Toc260184366)

[XMLHttpRequest 对象的三个重要的属性 3](#_Toc260184367)

[readyState 属性可能的值： 4](#_Toc260184368)

[Ajax 同步 和 异步 的区别 4](#_Toc260184369)

[Ajax基本代码片段(常用于拷贝) 4](#_Toc260184370)

[最基本的Ajax代码 4](#_Toc260184371)

[Servlet的doGet() 或 doPost()方法, 基本代码 7](#_Toc260184372)

[实例 7](#_Toc260184373)

[jQuery实现Ajax: 7](#_Toc260184374)

[JSON 8](#_Toc260184375)

[Prototype框架 8](#_Toc260184376)

[用传统方式校验用户名是否重复 9](#_Toc260184377)

[用Ajax方式校验用户名是否重复 10](#_Toc260184378)

[使用超连接的技巧 13](#_Toc260184379)

[封装Formbean数据时的问题 13](#_Toc260184380)

[防止表单重复提交 13](#_Toc260184381)

[使用复选框的技巧 13](#_Toc260184382)

[使用下拉列表框的技巧 13](#_Toc260184383)

[联动下拉列表框(静态) 13](#_Toc260184384)

[文件上传 17](#_Toc260184385)

[DispatchAction的应用 17](#_Toc260184386)

[Validator框架 17](#_Toc260184387)



## Ajax的组成部分

|  |
| --- |
| JavaScript  CSS  DOM  XMLHttpRequest对象 |

# 各种Ajax框架概述

|  |  |  |
| --- | --- | --- |
| 富客户端 | dojo [兜机油] | 纯粹页面端框架 |
| Ext JS | 纯粹页面端框架, 由于大, 慢, 所以更多的用于内网 |
| 与服务器远程调用 | DWR | 反向Ajax, 和 js调用java代码, 不提供页面效果 |
| prototype | js库, 变得不太好用了, 人们都转向jquery |
| JSON-RPC-java |  |
| 其它类 | GWT(Google Web Toolkit) |  |
|  |  |  |
|  |  |  |
|  | jquery | 轻量级, 前台 |
|  | mootools | 国外的 |
|  | script.aculo.us | 国外的, 提供了很多特效效果 |
|  | rico | 国外的 |

# Ajax基本流程

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * 异步（**A**synchronous） * XMLHttpRequest的属性和方法  |  |  | | --- | --- | | onreadystatechange | 设置回调函数 | | open(方式，URL，异步方式） | 发送数据方式 的设置 | | setRequestHeader(“content-type”,“application/x-www-form-urlencoded”); | 设置请求头 | | send(参数表） | 正式开始发送 | | readyState与status | readyState:交付状态 status:服务器的状态 200:准备 | | responseText，responseXML | 接收服务器返回的数据 |  * 局部刷新 |

## XMLHttpRequest 对象的三个重要的属性

|  |  |
| --- | --- |
| **onreadystatechange** | 存有处理服务器响应的函数 |
| **readyState** | 存有服务器响应的状态信息。每当 readyState 改变时，**onreadystatechange**函数就会被执行。 |
| **responseText** | 得到服务器返回的数据 |

## readyState 属性可能的值：

|  |  |
| --- | --- |
| **状态** | **描述** |
| 0 | 请求未初始化（在调用 open() 之前） |
| 1 | 请求已提出（调用 send() 之前） |
| 2 | 请求已发送（这里通常可以从响应得到内容头部） |
| 3 | 请求处理中（响应中 通常有部分数据可用，但是服务器还没有完成响应） |
| 4 | 请求已完成（可以访问服务器响应并使用它） |

## Ajax 同步 和 异步 的区别

|  |
| --- |
| //同步方式下, send这句话会等待服务器端数据回来后才执行完  //异步方式下, send这句话会立即完成执行. |

# Ajax基本代码片段(常用于拷贝)

## 最基本的Ajax代码

|  |  |
| --- | --- |
| 获取XMLHttpRequest对象 | **var** xmlHttp; //声明一个保存 XMLHttpRequest 对象的 xmlHttp 变量。  /\*\*  \* 返回XMLHttpRequest对象实例xmlHttp  \*/  **function** getXMLHttpRequest () {  **var** xmlHttp;  **try** {  xmlHttp = **new** XMLHttpRequest(); //使用 XMLHttp=new XMLHttpRequest() 来创建此对象。这条语句针对 Firefox、Opera 8.0+以及 Safari 浏览器。  **return** xmlHttp;  } **catch** (e) {  **try** {  xmlHttp = **new** ActiveXObject("Msxml2.XMLHTTP");//假如失败，则尝试针对 Internet Explorer 6.0+ 的 xmlHttp=new ActiveXObject("Msxml2.XMLHTTP")，  **return** xmlHttp;  } **catch** (e) {  **try** {  xmlHttp = **new** ActiveXObject("Microsoft.XMLHTTP");//假如也不成功，则尝试针对 Internet Explorer 5.5+ 的 xmlHttp=new ActiveXObject("Microsoft.XMLHTTP")。  **return** xmlHttp;  } **catch** (e) {  alert("您的浏览器不支持AJAX！"); //假如这三种方法都不起作用，那么这个用户所使用的浏览器已经太过时了，他或她会看到一个声明此浏览器不支持 AJAX 的提示。  **return** **false**;  }  }  }  }  xmlHttp = getXMLHttpRequest(); //getXMLHttpRequest方法返回XMLHttpRequest对象实例, 赋值给xmlHttp变量 |
| 注册回调函数(接收的是text) | **function** **handleStateChange**() { //onreadystatechange 属性 定义回调函数  **if**(xmlHttp.readyState==4){ //readyState 属性存有服务器响应的状态码。每当 readyState 改变时，onreadystatechange函数就会被执行。  document.myForm.time.value = xmlHttp.responseText; //服务器返回的数据会存储在xmlHttp.responseText里面 。  }  } |
| 注册回调函数(接收的是xml) | **function** **handleStateChange**() { //onreadystatechange 属性 定义回调函数  **if** (xmlHttp.readyState == 4) {  **var** xmlDoc = xmlHttp.responseXML.documentElement;  document.getElementById("companyname").innerHTML = xmlDoc.getElementsByTagName("compname")[0].childNodes[0].nodeValue;  document.getElementById("contactname").innerHTML = xmlDoc.getElementsByTagName("contname")[0].childNodes[0].nodeValue;  document.getElementById("address").innerHTML = xmlDoc.getElementsByTagName("address")[0].childNodes[0].nodeValue;  document.getElementById("city").innerHTML = xmlDoc.getElementsByTagName("city")[0].childNodes[0].nodeValue;  document.getElementById("country").innerHTML = xmlDoc.getElementsByTagName("country")[0].childNodes[0].nodeValue;  }  } |
| get方式, 向服务器发出请求  **url含: 请求地址, 所有参数, 时间戳** | //**get方式: url含: 请求地址, 所有参数, 时间戳**  **var** queryString = "username=youxuan&password=xiaoguo";  **var** url = "http://202.202.96.53/login/login1.jsp?" + queryString + "&timestamp=" + **new** Date().openTime();  xmlHttp.open("GET", url, "true"); //建立请求, 第三个参数: 规定应当对请求进行异步地处理  xmlHttp.onreadystatechange = **handleStateChange**; //注册回调函数, 一定要放在xmlHttp.open("GET", url);后面  xmlHttp.send(**null**); //执行将请求送往服务器。get方式, 则参数为null |
| **post方式:**  queryString携带所有参数,  url只有请求地址和时间戳 | //**post方式:** queryString携带所有参数, url只有请求地址和时间戳  **var** queryString = "username=youxuan&password=xiaoguo";  **var** url = "http://202.202.96.53/login/login1.jsp?timestamp=" + **new** Date.openTime();  xmlHttp.open("POST", url);  xmlHttp.onreadystatechange = **handleStateChange**; //注册回调函数, 一定要放在xmlHttp.open("POST", url);后面  xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded"); //设置同步  xmlHttp.send(queryString); //执行将请求送往服务器。post方式, 则参数为 参数和参数值 |

## Servlet的doGet() 或 doPost()方法, 基本代码

|  |
| --- |
| **public** **void** doGet(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.setContentType("text/html");  PrintWriter out = response.getWriter();    out.print("--------");  out.flush();  out.close();  } |

# 实例

## jQuery实现Ajax:

1. 获取文本框中的内容

2. 将文本框中的数据发送给服务器的servlet

3. 接收服务器返回的数据

4. 将服务器端返回的数据动态的现实在页面上.

|  |
| --- |
| function verify(){  // alert("我被点击了！")    // 1. 获取文本框中的内容  // var username = document.getElementById("username").value; // dom方式  var jqueryObj = $("#username") //Jquery 的查找节点的方式， 参数中#加上id属性值可以找到一个节点。返回的都是jquery的对象， 可以继续在上面执行其他的jquery方法。  var name = jqueryObj.val();  alert("username=" + username);  // 2. 将文本框中的数据发送给服务器的servlet  $.get("AJAXServer?name=" + username, null, callback ); // 第一个参数，服务器端的URL地址；第二个参数，要发送的名值对；弟三个参数，设置回调函数。callback：代码继续往后运行。  }  function callback(data){ //服务器传回来的数据作为回调方法参数.  // 3. 接收服务器返回的数据  // 4. 将服务器端返回的数据动态的现实在页面上.  var resultObj = $("#result"); //拿到节点,  resultObj.html(data); //改变节点的value值.  } |

如何获取文本框当中的内容:

1.

## JSON

|  |
| --- |
| * 概念：   + 全称JavaScript Object Notation，是JavaScript的一种轻量级的数据交换格式，易于人阅读和编写。   + JSON定义了表示对象（或哈希表）和数组的格式，注意：Javascript中用同一种数据结构来表示对象和哈希表。 * 对象（或哈希表）的构建格式：   + 用一对{}表示，其中可以包含多对“名称和值”，名称和值之间用“:”分隔，每对“名称和值”之间用“,”分隔。名称可以用单引号或双引号引起来，如果名称中包含了空格字符，则必须用引号引起来。   + 举例：{name:”zxx” , weight:120 , ”birthday” : new Date(1973,3,8)} * 数组的构建格式：   + 用一对[]表示，每个元素值之间用用“,”分隔。 * 对象和数组中的值可以还是JSON格式的对象或数组，即对象和数组可以相互嵌套。 |

## Prototype框架

|  |
| --- |
| * 概念：   + Prototype是一个Javascript工具库，与服务器端的编程语言没有关系，可用于java ee和.net项目中。 * 三个主要的功能：   + 为DOM编程操作提供了诸多便利和加强的API类。   + 提供了对XMLHttpRequest对象进行封装的Ajax API类，简化了XMLHttpRequest对象创建和调用细节。   + 提供了构造和解析JSON对象的扩展方法。 * 技巧：   + 在实际项目中最好是使用压缩版本的Prototype库。 |
| new Ajax.Updater('msg','<html:rewrite action="/RegUserUI"/>', {method: 'get', parameters: {'uname': username} } ); |
| 将Element.update方法改写为$函数的形式， $(‘msg’).update(transport.responseText) |

## 用传统方式校验用户名是否重复

|  |
| --- |
| **HTML代码**  <%@ page language=*"java"* pageEncoding=*"utf-8"*%>  <%@ taglib uri=*"http://struts.apache.org/tags-bean"* prefix=*"bean"* %>  <%@ taglib uri=*"http://struts.apache.org/tags-html"* prefix=*"html"* %>  <%@ taglib uri=*"http://struts.apache.org/tags-logic"* prefix=*"logic"* %>  <%@ taglib uri=*"http://struts.apache.org/tags-tiles"* prefix=*"tiles"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html:html lang=*"true"*>  <head>    <title>RegUser.jsp</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  <script type=*"text/javascript"*>  **function** verfiyUserName(){  **var** userName = document.getElementById("username").value;  **var** url = '<html:rewrite action="/VerifyUserName"/>?userName=' + userName;  window.location.href = url; //让窗口地址等于这个url  //window.location.replace(url); //换地址; //window.location.reload(true);reload是刷新  **return** **fals**e; //返回false, 后面的所有代码将不再执行, 包括调用这个请求的超链接的href目标地址也不会访问.  }  </script>    </head>    <body>  <form>  用户名：<input type=*"text"* id=*"username"* name=*"username"* /><a href=*"#"* onclick="return verfiyUserName()"/>" 用户名是否存在</a><br/>  <div>${msg}</div>  密码：<input type=*"password"* name=*"password"* /><br>  确认密码：<input type=*"password"* name=*"password2"* /><br>  <input type=*"submit"* value=*"注册"*/>  </form>  </body>  </html:html> |

## 用Ajax方式校验用户名是否重复

|  |
| --- |
| **HTML代码**  <%@ page language=*"java"* pageEncoding=*"utf-8"*%>  <%@ taglib uri=*"http://struts.apache.org/tags-bean"* prefix=*"bean"* %>  <%@ taglib uri=*"http://struts.apache.org/tags-html"* prefix=*"html"* %>  <%@ taglib uri=*"http://struts.apache.org/tags-logic"* prefix=*"logic"* %>  <%@ taglib uri=*"http://struts.apache.org/tags-tiles"* prefix=*"tiles"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html:html lang=*"true"*>  <head>    <title>RegUser.jsp</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  <script type=*"text/javascript"*>    //创建XMLHttpRequest对象函数  **function** createXHR(){  **var** xmlhttp = **null**;  **if**(window.XMLHttpRequest){ //IE7和FireForx, Mozillar, Opera等非IE浏览器创建XMLHttpRequest对象  **var** xmlhttp = **new** XMLHttpRequest();  **if**(window.overrideMimeType){//解决部分版本的Mozilla中服务器响应头信息没有XML时 不能正常工作的问题  xmlhttp.overrideMimeType('text/xml');  }  }**else** **if**(window.ActiveXOBject){//IE6, IE5.5, IE5创建XMLHttpRequest对象  **var** MSXML = ['MSXML2.XMLHTTP.6.0','MSXML2.XMLHTTP.5.0','MSXML2.XMLHTTP.4.0','MSXML2.XMLHTTP.3.0','MSXML2.XMLHTTP','Microsoft.XMLHTTP'];  **for**(**var** n = 0; n < MSXML.length; n ++){  **try**{  xmlhttp = **new** ActiveXObject(MSXML[n]);  **break**;  }**catch**(e){}  }  }**else**{  alert("cant't Create XMLHttpRequest Object");  }  //alert(xmlhttp);  **return** xmlhttp;  }    //超链接绑定的函数  **function** verfiyUserName(){    **var** userName = document.getElementById("username").value;  **var** url = '<html:rewrite action="/AjaxVerifyUserName"/>?userName=' + userName;    **var** xhr = createXHR();  xhr.open('GET', url, **false**);//组装发送的数据: get方式, 目标地址为url, false代表同步方式,数据返回来之后才执行读取结果那一行代码  //xhr.addRequestHeader(); //可以添加头文件  xhr.send(body);    //alert(xhr.responseText); //服务器返回结果文本  document.getElementById("result").innerHTML = xhr.responseText;  **return** **false**;  }    //失去焦点事件绑定的函数  **function** verfiyUserName2(userName){  **var** url = '<html:rewrite action="/AjaxVerifyUserName"/>?userName=' + userName;    **var** xhr = **new** createXHR();  xhr.onreadystatechange = **function**(){ //异步传输需要的回调函数,  //xhr.readyState等于0代表还未初始化xhr这个对象, 1代表调用open方法了, 2代表调用sent方法了, 3代表正在接收, 4代表接收数据完毕  //xhr.status等于200代表服务器送回来的结果正常  **if**(xhr.readyState==4 && xhr.status==200){  document.getElementById("result").innerHTML = xhr.responseText;  }  }  xhr.open('GET', url, **true**);//组装发送的数据: get方式, 目标地址为url, true代表异步方式  //xhr.addRequestHeader(); //可以添加头文件  xhr.send(body);    **return** **false**;  }  </script>    </head>    <body>  <form>  用户名：<input type=*"text"* id=*"username"* name=*"username"* onblur=*"verfiyUserName2(this.value)"*/><a href=*"#"* onclick="return verfiyUserName()"/>"用Ajax方式校验用户名是否重复</a><br/>  <div id=*"result"*>${msg}</div>  密码：<input type=*"password"* name=*"password"* /><br>  确认密码：<input type=*"password"* name=*"password2"* /><br>  <input type=*"submit"* value=*"注册"*/>  </form>  </body>  </html:html> |
| **Action代码**  **public** ActionForward execute(ActionMapping mapping, ActionForm form,  HttpServletRequest request, HttpServletResponse response) {  System.*out*.println("请求了AjaxVerifyUserNameAction");    String userName = request.getParameter("userName");  String msg = "恭喜你, 用户名可用!";  **if**("zxx".equals(userName)){ //userName写在后面的好处: 当userName==null时, 不会抛空指针异常.  msg = "很遗憾, 用户名已被占用!";  }    response.setContentType("text/plain;charset=utf-8");  **try** {  response.getWriter().print(msg);  } **catch** (IOException e) {  e.printStackTrace();  }    **return** **null**;  } |

## 使用超连接的技巧

## 封装Formbean数据时的问题

## 防止表单重复提交

## 使用复选框的技巧

## 使用下拉列表框的技巧

### 联动下拉列表框(静态)

|  |
| --- |
| **静态HTML代码**  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>MyHtml.html</title>    <script type=*"text/javascript"*>  **var** data = [['湖北','99','武汉','3','宜昌','9','荆州','6'],  ['湖南','77','长沙','4','岳阳','7','郴州','10'],  ['四川','77','成都','5','宜宾','11','锦阳','8','自贡','12']  ];  //填充所有的省份  **function** fillProvince(){  **for**(**var** i=0; i<data.length; i++){  **var** provincedData = data[i];  **var** optProvince = **new** Option(provincedData[0], provincedData[1]);  document.getElementById("provincId").add(optProvince);  }  }  //填充省份下的城市  **function** fillCity(provinceIndex){  document.getElementById("cityId").options.length=0; //简单的清空子元素办法  clearOptions(document.getElementById("cityId")); //用JS代码实现真正的清空子元素  **var** provinceData = data[provinceIndex];  **for**(**var** i=1; i<provinceData.length/2; i++){ //i从1开始的原因是:脚标为0的数据为省份  **var** optCity = **new** Option(provinceData[i\*2], provinceData[i+1]); //创建city节点, 第一个参数是value, 第二个参数是key  document.getElementById("cityId").add(optCity);  }  }  //清除options的内容  **function** clearOptions(selElement){  **var** options = selElement.options;  **var** len = options.length;  **for**(**var** i=0; i<len; i++){  //selElement.remove(i); //父元素的remove方法  options.remove(0); //子元素集合的remove方法  }  }  </script>  </head>  <body>  <select id=*"provincId"* name=*"provincId"* onchange=*"fillCity(this.selectedIndex)"*></select>  <select id=*"cityId"* name=*"cityId"*></select>  <script type=*"text/javascript"*>  fillProvince();//显示省份  fillCity(document.getElementById("provincId").selectedIndex); //页面载入时, 级联显示省份下的城市  </script>  </body>  </html> |
| **JSON格式**  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>MyHtml.html</title>  <script type=*"text/javascript"*>  **var** data = [{name:'湖北',id:99,cities:[{name:'武汉',id:3},{name:'宜昌',id:9},{name:'荆州',id:6}]},  {name:'湖南',id:77,cities:[{name:'长沙',id:4},{name:'岳阳',id:7},{name:'郴州',id:10}]},  {name:'四川',id:77,cities:[{name:'成都',id:5},{name:'宜宾',id:11},{name:'锦阳',id:8},{name:'自贡',id:12}]}];  //填充所有的省份  **function** fillProvince(){  **for**(**var** i=0; i<data.length; i++){  **var** provincedObj = data[i];  **var** optProvince = **new** Option(provincedObj.name, provincedObj.id);  document.getElementById("provincId").add(optProvince);  }  }  //填充省份下的城市  **function** fillCity(provinceIndex){  document.getElementById("cityId").options.length=0; //简单的清空子元素办法  //clearOptions(document.getElementById("cityId")); //用JS代码实现真正的清空子元素  **var** provinceObj = data[provinceIndex];  **for**(**var** i=0; i<provinceObj.cities.length; i++){ //i从1开始的原因是:脚标为0的数据为省份  **var** cityObj = provinceObj.cities[i];  **var** optCity = **new** Option(cityObj.name, cityObj.id); //创建city节点, 第一个参数是value, 第二个参数是key  document.getElementById("cityId").add(optCity);  }  }  //清除options的内容  **function** clearOptions(selElement){  **var** options = selElement.options;  **var** len = options.length;  **for**(**var** i=0; i<len; i++){  //selElement.remove(i); //父元素的remove方法  options.remove(0); //子元素集合的remove方法  }  }  </script>  </head>  <body>  <select id=*"provincId"* name=*"provincId"* onchange=*"fillCity(this.selectedIndex)"*></select>  <select id=*"cityId"* name=*"cityId"*></select>  <script type=*"text/javascript"*>  fillProvince();//显示省份  fillCity(document.getElementById("provincId").selectedIndex); //页面载入时, 级联显示省份下的城市  </script>  </body>  </html> |

## 文件上传

## DispatchAction的应用

## Validator框架

Struts请求处理过程的源码分析与digester