# 一 Ajax原理

## 1 同步与异步

同步：发送完一个请求，只有收到响应后才能发送另外的请求

异步：发送完一个请求，不用等收到该请求的 响应，就可以直接发送另外的请求

## 2什么是AJAX

Asynchronous JavaScript And Xml的缩写

浏览器在不刷新全部页面的情况下，和服务端进行通信的过程

Ajax就是实现局部刷新的异步通信技术

## 3 AJAX工作原理

通过XmlHttpRequest对象和服务端进行通信，浏览器收到服务端的返回值后，对局部区域的内容进行设置

# 二 js Ajax

## 1 AJAX开发步骤

1)创建AJAX异步对象，createAJAX()

2)准备发送异步请求， ajax.open(method,url)

3)如果是POST请求，一定要设置AJAX请求头:

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

GET请求无需设置

4)真正发送请求体中的数据到服务器，ajax.send()

如果是get请求，ajax.send(null);

如果是post请求，ajax.send(“name=zhangsan”)

5)监听服务端响应的状态变化，ajax.onreadystatechange(状态改变的事件触发器)

6)获取AJAX的数据后，执行相关操作

<script type=*"text/javascript"*>

//1 创建ajax对象

**function** createAjax() {

**var** ajax = **null**;

**try** {

//针对IE浏览器

ajax = **new** ActiveXObject("microsoft.xmlhttp");

} **catch** (e) {

//针对其他浏览器

ajax = **new** XMLHttpRequest();

}

**return** ajax;

}

</script>

<script type=*"text/javascript"*>

**function** btnClick() {

//1 创建ajax对象

**var** ajax = createAjax();

//method 数据提交方式

//url 请求的资源

**var** method = "GET";

**var** url = "${pageContext.request.contextPath}/AjaxTimeServlet";

//2 对ajax进行预处理

ajax.open(method, url);

//3 发送ajax请求

ajax.send(**null**);

//4 监听ajax状态的变化

//0-4 状态

/\*0 表示创建ajax对象，还没有调用open()方法

1 表示调用open方法，还没有调用send方法

2 表示调用send方法，但是ajax请求还没发送到服务端

3 表示请求发送到服务端，但是没有开始处理

4 表示处理完成ajax请求\*/

ajax.onreadystatechange = **function**() {

// 4表明通过ajax请求进行通信完成

**if** (ajax.readyState == 4) {

//判断通信状态，200表明正常返回

**if** (ajax.status == 200) {

//获取响应的数据

**var** v = ajax.responseText;

//alert(v);

//给元素设置值

document.getElementById("dt").innerHTML = v;

}

}

};

}

</script>

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

// **TODO** Auto-generated method stub

Date date = **new** Date();

SimpleDateFormat sdf = **new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String time = sdf.format(date);

//针对ajax请求，通过PrintWriter返回数据

response.getWriter().write(time);

}

## 2 状态

0 表示创建ajax对象，还没有调用open()方法

1 表示调用open方法，还没有调用send方法

2 表示调用send方法，但是ajax请求还没发送到服务端

3 表示请求发送到服务端，但是没有开始处理

4 表示处理完成ajax请求

# 三 JSON

## 1什么是JSON

JSON(JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。它采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

Js中任何支持的类型都可以通过 JSON 来表示，例如字符串、数字、对象、数组等。其中：

1）对象表示为键值对

2）数据由逗号分隔

3）花括号{}保存对象

4）方括号[]保存数组

注意：JSON 是 JS 对象的字符串表示法，它使用文本表示一个 JS 对象的信息，本质是一个字符串。

## 2 fastjson

Fastjson是alibaba提供的第三方组件，用来处理json格式数据

本例导入jar包：fastjson-1.2.2.jar

测试代码如下：

ArrayList<Student> list=**new** ArrayList<>();

**for**(**int** i=1;i<3;i++){

list.add(**new** Student(101+i, "hoho", 20+i));

}

//对象转为json格式字符串

String json = JSON.*toJSONString*(list);

System.***out***.println(json);

Student s = **new** Student(1, "haha", 3);

String json2 = JSON.*toJSONString*(s);

System.***out***.println(json2);

String json3 = "{\"name\":\"张三\",\"age\":24}";

//json格式字符串转为对象

UserInfo userInfo=JSON.*parseObject*(json3, UserInfo.**class**);

System.***out***.println("name:" + userInfo.getName()+ ", age:"+userInfo.getAge());

String json6="[{\"name\":\"zhangsan\",\"age\":25}]";

// 转为数组

List<UserInfo> ll2 = JSON.*parseArray*(json6, UserInfo.**class**);

System.***out***.println(ll2.get(0).getName());

String json4="{\"name\":\"zhangsan\",\"age\":25}";

Map<String, Object> map = JSON.*parseObject*(json4, **new** TypeReference<Map<String, Object>>(){});

System.***out***.println(map.get("name"));

String json5="[{\"name\":\"zhangsan\",\"age\":25}]";

List<UserInfo> ll = JSON.*parseObject*(json5, **new** TypeReference<List<UserInfo>>(){});

System.***out***.println(ll.get(0).getName());

Person p = **new** Person();

p.setId(1);

p.setName("haha");

p.setDate(**new** Date());

// 日期处理

// 时间戳

String info = JSON.*toJSONString*(p);

System.***out***.println(info);

//默认格式为yyyy-MM-dd HH:mm:ss

String info2 = JSON.*toJSONString*(p, SerializerFeature.***WriteDateUseDateFormat***);

System.***out***.println(info2);

// 根据自定义格式输出日期

System.***out***.println(JSON.*toJSONStringWithDateFormat*(p, "yyyy-MM-dd", SerializerFeature.***WriteDateUseDateFormat***));

## 3 jackson

ArrayList<Student> list=**new** ArrayList<>();

**for**(**int** i=1;i<3;i++){

list.add(**new** Student(101+i, "码子", 20+i));

}

//ObjectMapper对象，用于序列化和反序列化

ObjectMapper mapper=**new** ObjectMapper();

String json = **null**;

**try** {

//将对象转换为JSON格式字符串

json = mapper.writeValueAsString(list);

} **catch** (JsonProcessingException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(json);

Student s = **new** Student(1, "haha", 3);

String json2 = **null**;

**try** {

json2 = mapper.writeValueAsString(s);

} **catch** (JsonProcessingException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(json2);

String json3 = "{\"name\":\"张三\",\"age\":24}";

//json格式字符串转为对象

UserInfo userInfo = **null**;

**try** {

userInfo = mapper.readValue(json3, UserInfo.**class**);

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("name:" + userInfo.getName()+ ", age:"+userInfo.getAge());

String json6="[{\"name\":\"zhangsan\",\"age\":25}]";

// 转为数组

List<UserInfo> ll2 = **null**;

**try** {

ll2 = mapper.readValue(json6, **new** TypeReference<List<UserInfo>>() {

});

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println(ll2.get(0).getName());

Person p = **new** Person();

p.setId(1);

p.setName("haha");

p.setDate(**new** Date());

// 设置日期格式

mapper.setDateFormat(**new** SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));

// 不设置格式，时间戳

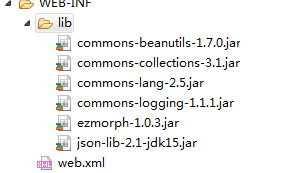
String info = mapper.writeValueAsString(p);

System.***out***.println(info);

## 4 Json-lib

Json-lib，是第三方组件，可以用来生成json格式数据

可以将Java对象转成json格式的字符串，也可以将Java对象转换成xml格式的文档，同样可以将json字符串转换成Java对象或是将xml字符串转换成Java对象



常用方法：

将数据转为json格式对象：

JSONObject.fromObject()

将数据转为json格式数组

JSONArray.fromObject()

通过JsonConfig的对象注册处理器：

registerJsonValueProcessor()

**public** **static** **void** bean2Json(){

User u = **new** User();

u.setId(10);

u.setName("zhagnsan");

u.setAge(20);

//将实体类对象转为JSONObject对象

JSONObject obj = JSONObject.*fromObject*(u);

//转为json格式的字符串

System.***out***.println(obj.toString());//{"name":"zhagnsan","id":10,"age":20}

JSONArray arr = JSONArray.*fromObject*(u);

System.***out***.println(arr.toString());//[{"name":"zhagnsan","id":10,"age":20}]

}

**public** **static** **void** list2Json(){

List<User> l = **new** ArrayList<>();

User u = **new** User();

u.setId(10);

u.setName("zhagnsan");

u.setAge(20);

l.add(u);

u = **new** User();

u.setId(11);

u.setName("lisi");

u.setAge(20);

l.add(u);

u = **new** User();

u.setId(12);

u.setName("wangwu");

u.setAge(20);

l.add(u);

JSONArray arr = JSONArray.*fromObject*(l);

System.***out***.println(arr.toString());

//异常

// JSONObject obj = JSONObject.fromObject(l);

// System.out.println(obj.toString());

}

**public** **static** **void** map2Json(){

Map<Integer, User> map = **new** HashMap<>();

User u = **new** User();

u.setId(10);

u.setName("zhagnsan");

u.setAge(20);

map.put(10, u);

u = **new** User();

u.setId(11);

u.setName("lisi");

u.setAge(20);

map.put(11, u);

u = **new** User();

u.setId(12);

u.setName("wangwu");

u.setAge(20);

map.put(12, u);

JSONObject obj = JSONObject.*fromObject*(map);

System.***out***.println(obj.toString());

JSONArray arr = JSONArray.*fromObject*(map);

System.***out***.println(arr.toString());

}

@Test

**public** **void** date2json(){

Map<String, Object> map = **new** HashMap<>();

map.put("name", "lisi");

map.put("birth", **new** Date());

JSONObject object = JSONObject.*fromObject*(map);

System.***out***.println(object.toString());

JsonConfig config = **new** JsonConfig();

//注册转换器,第一个参数表示要转换的类型；第二个参数，对应的转换的对象

config.registerJsonValueProcessor(Date.**class**, **new** DateJson("yyyy-MM-dd HH:mm:ss"));

JSONObject obj1 = JSONObject.*fromObject*(map, config);

System.***out***.println(obj1.toString());

}

//将日期转为指定的格式

**public** **class** DateJson **implements** JsonValueProcessor{

**private** SimpleDateFormat sdf;

**public** DateJson(String format){

sdf = **new** SimpleDateFormat(format);

}

@Override

**public** Object processArrayValue(Object value, JsonConfig arg1) {

// **TODO** Auto-generated method stub

**return** sdf.format((Date)value);

}

@Override

**public** Object processObjectValue(String key, Object value, JsonConfig arg2) {

// **TODO** Auto-generated method stub

**return** sdf.format((Date)value);

}

}

# 四 jQuery Ajax

## 1 load方法

返回结果自动添加到jQuery对象代表的标签中间

<script type=*"text/javascript"*>

$("button").click(**function**(){

**var** url = "${pageContext.request.contextPath}/AjaxTimeServlet";

//将获取的数据直接绑定到对应的元素

//$("#did").load(url);//使用get提交方式

**var** sendData = {name:"zhangsan", age:12};

//第二个参数 sendData 提交给服务器的数据,数据要求是json对象格式

//第三个参数 function 回调函数

// backData 返回的数据

// textStatus 运行状态的文本表示

// ajax ajax对象

//使用第二参数发送数据，load方法使用post提交方式

/\* $("#did").load(url, sendData, function(backData,textStatus, ajax){

alert(backData);

alert(textStatus);

alert(ajax.responseText);

}); \*/

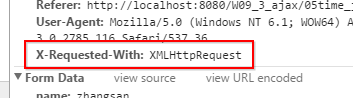
$("#did").load(url, sendData, **function**(backData){

alert(backData);

});

});

</script>



注意：使用jQuery会在请求头中增加图中所示的内容，js原生写法不会增加该内容

## 2 $.get()/$.post()

<script type=*"text/javascript"*>

$("#uin").blur(**function**(){

**var** url = "${pageContext.request.contextPath}/UserExistServlet";

**var** sendData = {name: $("#uin").val()};

//以get方式发送ajax请求

/\* $.get(url, sendData, function(backData){

$("#tip").html(backData);

}); \*/

//以post方式发送ajax请求

$.post(url, sendData, **function**(backData){

$("#tip").html("<img width='20' src='" + backData + "'/>");

});

});

</script>

3 serialize()

序列化表单数据为字符串

**var** tt = $("form").serialize();

alert(**typeof** tt);//string

alert(tt);//name=zhangsan&pwd=123

//data:{name:$(":text").val(), password:$(":password").val()},

data:$("form").serialize(),//将表单中的数据序列化

//data:"name=zhagnsan&password=123",

## 4 $.ajax()

<script type=*"text/javascript"*>

**function** btnClick(){

//type 提交方式

//url 请求的资源

//data 提交给资源的参数，json格式

//dataType 返回的数据的类型

//success ajax请求成功对应的函数

//error ajax请求异常对应的函数

$.ajax({

type:"post",

url:"${pageContext.request.contextPath}/UserListServlet",

data:{name:"zhangsan"},

dataType:"json",//预期返回的是json对象格式的数据

success:**function**(backData){

//alert(backData);

$("#tid").empty();

//遍历返回的数组

$(backData).each(**function**(){

**var** html = "<tr>";

html += "<td>" + **this**.id + "</td>";

html += "<td>" + **this**.name + "</td>";

html += "<td>" + **this**.age + "</td>";

html += "</tr>";

$("#tid").append($(html));

});

},

error:**function**(backData){

alert(backData);

}

});

}

</script>