

一 Ajax技术与原理

1.1 Ajax简介

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML）。

AJAX 不是新的编程语言，而是一种使用现有标准的新方法。

AJAX 是与服务器交换数据并更新部分网页的艺术，在不重新加载整个页面的情况下。

1.2 Ajax所包含的技术

大家都知道ajax并非一种新的技术，而是几种原有技术的结合体。它由下列技术组合而成。

- 1.使用CSS和XHTML来表示。
- 2.使用DOM模型来交互和动态显示。
- 3.使用XMLHttpRequest来和服务器进行异步通信。
- 4.使用javascript来绑定和调用。

AJAX 的核心是 XMLHttpRequest 对象。

不同的浏览器创建 XMLHttpRequest 对象的方法是有差异的。

IE 浏览器使用 ActiveXObject，而其他的浏览器使用名为 XMLHttpRequest 的 JavaScript 内建对象

1.3 Ajax的工作原理

Ajax的工作原理相当于在用户和服务器之间加了一个中间层(AJAX引擎)，使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器。像一些数据验证和数据处理等都交给Ajax引擎自己来做,,只有确定需要从服务器读取新数据时再由Ajax引擎代为向服务器提交请求。

来看看和传统方式的区别

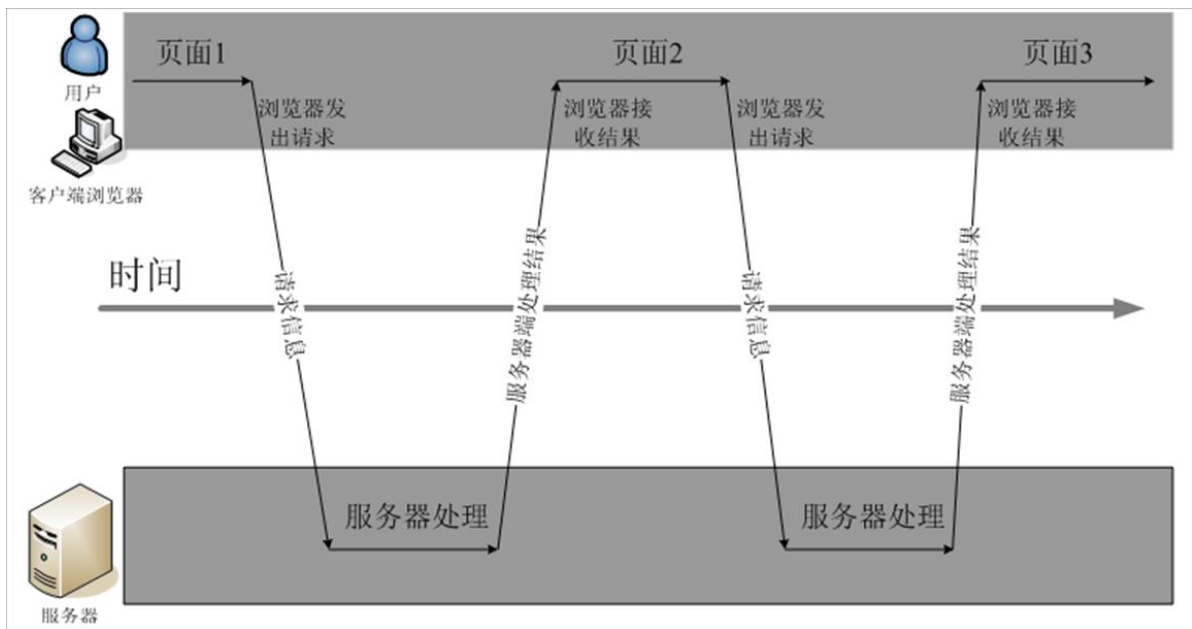


图 1 传统的 Web 应用模型

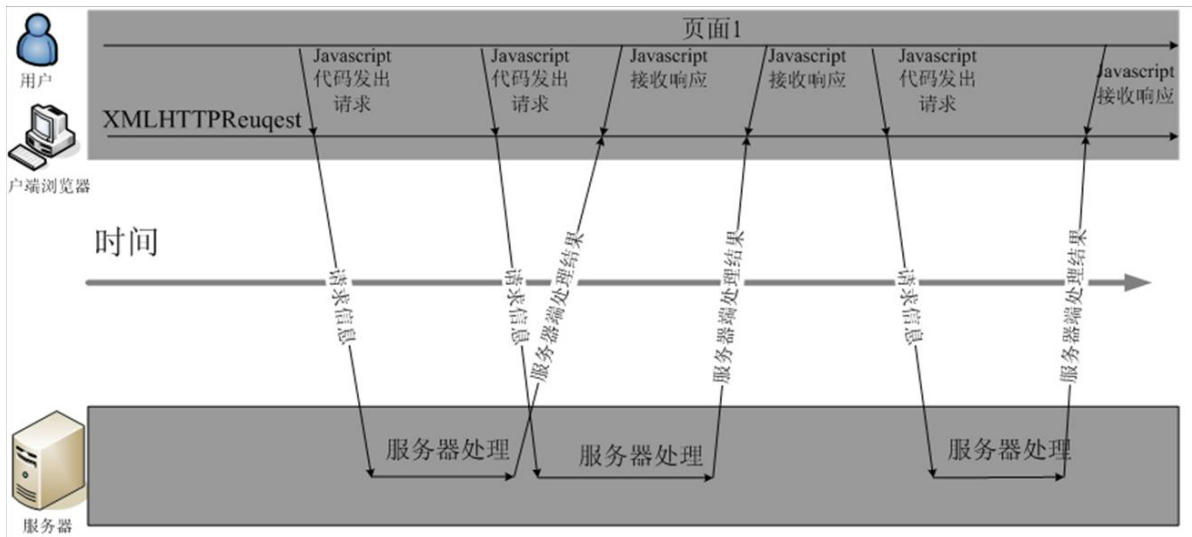


再看看它们各自的交互

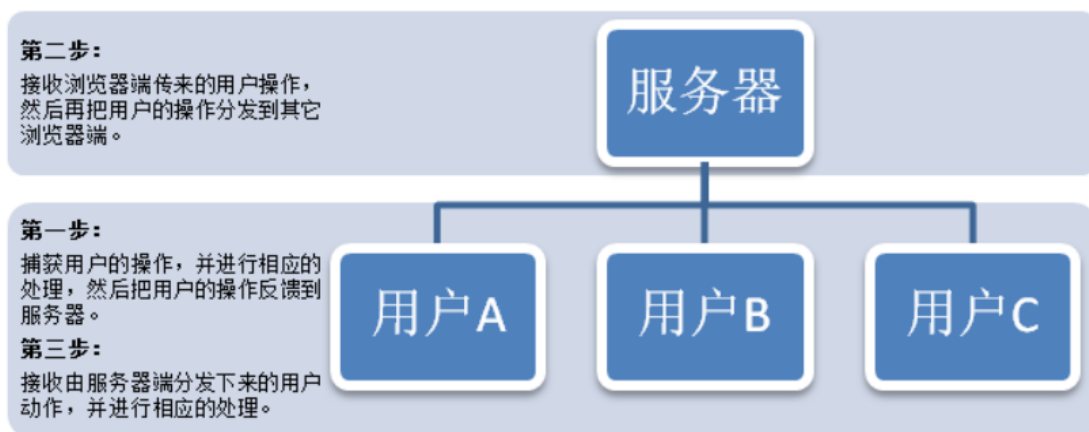
浏览器的普通交互方式



浏览器的Ajax交互方式



在创建Web站点时，在客户端执行屏幕更新为用户提供了很大的灵活性。下面是使用Ajax可以完成的功能：动态更新购物车的物品总数，无需用户单击Update并等待服务器重新发送整个页面。提升站点的性能，这是通过减少从服务器下载的数据量而实现的。例如，在Amazon的购物车页面，当更新篮子中的一项物品的数量时，会重新载入整个页面，这必须下载32K的数据。如果使用Ajax计算新的总量，服务器只会返回新的总量值，因此所需的带宽仅为原来的百分之一。消除了每次用户输入时的页面刷新。例如，在Ajax中，如果用户在分页列表上单击Next，则服务器数据只刷新列表而不是整个页面。直接编辑表格数据，而不是要求用户导航到新的页面来编辑数据。对于Ajax，当用户单击Edit时，可以将静态表格刷新为内容可编辑的表格。用户单击Done之后，就可以发出一个Ajax请求来更新服务器，并刷新表格，使其包含静态、只读的数据。



1.4 XMLHttpRequest常用属性

1. onreadystatechange 属性

onreadystatechange 属性存有处理服务器响应的函数。下面的代码定义一个空的函数, 可同时对 onreadystatechange 属性进行设置:

```
xmlHttpRequest.onreadystatechange = function() { //我们需要在这写一些代码}
```

2. readyState 属性

readyState 属性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。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)

我们要向这个 onreadystatechange 函数添加一条 If 语句, 来测试我们的响应是否已完成(意味着可获得数据):

```
xmlHttpRequest.onreadystatechange = function() {  
    if (xmlHttpRequest.readyState == 4) {  
        //从服务器的response获得数据  
    }  
}
```

3..responseText 属性

可以通过 responseText 属性来取回由服务器返回的数据。在我们的代码中, 我们将把时间文本框的值设置为等于 responseText:

```
xmlHttpRequest.onreadystatechange = function() {
    if (xmlHttpRequest.readyState == 4) {
        document.myForm.time.value = xmlHttpRequest.responseText;
    }
}
```

其它属性如下：

属 性	描 述
onreadystatechange	状态改变的事件触发器，每个状态改变时都会触发这个事件处理器，通常会调用一个JavaScript函数
readyState	请求的状态。有5个可取值：0 = 未初始化，1 = 正在加载，2 = 已加载，3 = 交互中，4 = 完成
responseText	服务器的响应，返回数据的文本。
responseXML	服务器的响应，返回数据的兼容DOM的XML文档对象，这个对象可以解析为一个DOM对象。
responseBody	服务器返回的主题（非文本格式）
responseStream	服务器返回的数据流
status	服务器的HTTP状态码（如：404 = "文件未找到"、200 = "成功"，等等）
statusText	服务器返回的状态文本信息，HTTP状态码的相应文本（OK或Not Found（未找到）等等）

1.5 XMLHttpRequest方法

1. open() 方法

open() 有三个参数。第一个参数定义发送请求所使用的方法，第二个参数规定服务器端脚本的URL，第三个参数规定应当对请求进行异步地处理。

```
xmlHttpRequest.open("GET", "test.php", true);
```

2. send() 方法

send() 方法将请求送往服务器。如果我们假设 HTML 文件和 PHP 文件位于相同的目录，那么代码是这样的：

```
xmlHttpRequest.send(null);
```

其它方法如下：

方 法	描 述
abort()	停止当前请求
getAllResponseHeaders()	把HTTP请求的所有响应首部作为键/值对返回
getResponseHeader("header")	返回指定首部的串值
open("method", "URL", [asyncFlag], ["userName"], ["password"])	建立对服务器的调用。method参数可以是GET、POST或PUT。url参数可以是相对URL或绝对URL。这个方法还包括3个可选的参数，是否异步，用户名，密码
send(content)	向服务器发送请求
setRequestHeader("header", "value")	把指定首部设置为所提供的值。在设置任何首部之前必须先调用open()。设置header并和请求一起发送 ('post'方法一定要)

二 Ajax编程步骤

为了方便理解，我给AJAX统一了一个流程，要想实现AJAX，就要按照以后步骤走：

1. 创建XMLHttpRequest对象。

2. 设置请求方式。
3. 调用回调函数。
4. 发送请求。

下面来看下具体步骤：

2.1 创建XMLHttpRequest对象

创建XMLHttpRequest对象的语法是：

```
var xmlHttp=new XMLHttpRequest();
```

如果是IE5或者IE6浏览器，则使用ActiveX对象，创建方法是：

```
var xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
```

一般我们手写AJAX的时候，首先要判断该浏览器是否支持XMLHttpRequest对象，如果支持则创建该对象，如果不支持则创建ActiveX对象。JS代码如下：

```
//第一步：创建XMLHttpRequest对象
var xmlHttp;
if (window.XMLHttpRequest) {
    //非IE
    xmlHttp = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    //IE
    xmlHttp = new ActiveXObject("Microsoft.XMLHTTP")
}
```

2.2 设置请求方式

在WEB开发中，请求有两种形式，一个是get，一个是post，所以在这里需要设置一下具体使用哪个请求，XMLHttpRequest对象的open()方法就是来设置请求方式的。open()方法

功能	参数
规定请求的类型、URL 以及是否异步处理请求	参数1：设置请求类型（GET 或 POST），GET与POST的区别请自己百度一下，这里不做解释； 参数2：文件在服务器上的位置； 参数3：是否异步处理请求，是为true，否为false。

如下：

```
//第二步：设置和服务端交互的相应参数，向路径http://localhost:8080/JsLearning3/getAjax
准备发送数据
var url = "http://localhost:8080/JsLearning3/getAjax";
xmlHttp.open("POST", url, true);
```

open方法如下：

方法	描述
<code>open(method,url,async)</code>	<p>规定请求的类型、URL 以及是否异步处理请求。</p> <ul style="list-style-type: none"> • <i>method</i>: 请求的类型: GET 或 POST • <i>url</i>: 文件在服务器上的位置 • <i>async</i>: true (异步) 或 false (同步)
<code>send(string)</code>	<p>将请求发送到服务器。</p> <ul style="list-style-type: none"> • <i>string</i>: 仅用于 POST 请求

GET 还是 POST? 与 POST 相比, GET 更简单也更快, 并且在大部分情况下都能用。然而, 在以下情况中, 请使用 POST 请求:

无法使用缓存文件 (更新服务器上的文件或数据库) 向服务器发送大量数据 (POST 没有数据量限制) 发送包含未知字符的用户输入时, POST 比 GET 更稳定也更可靠

异步 - True 或 False? AJAX 指的是异步 JavaScript 和 XML (Asynchronous JavaScript and XML) 。XMLHttpRequest 对象如果要用于 AJAX 的话, 其 `open()` 方法的 `async` 参数必须设置为 `true`: 对于 web 开发人员来说, 发送异步请求是一个巨大的进步。很多在服务器执行的任务都相当费时。AJAX 出现之前, 这可能会引起应用程序挂起或停止。通过 AJAX, JavaScript 无需等待服务器的响应, 而是: 在等待服务器响应时执行其他脚本 当响应就绪后对响应进行处理

2.3 调用回调函数

如果在上一步中 `open` 方法的第三个参数选择的是 `true`, 那么当前就是异步请求, 这个时候需要写一个回调函数, XMLHttpRequest 对象有一个 `onreadystatechange` 属性, 这个属性返回的是一个匿名的方法, 所以回调函数就在这里写 `xmlHttp.onreadystatechange=function()`, `function()` 内部就是回调函数的内容。所谓回调函数, 就是请求在后台处理完, 再返回到前台所实现的功能。在这个例子里, 我们的回调函数要实现的功能就是接收后台处理后反馈给前台的数据, 然后将这个数据显示到指定的 `div` 上。因为从后台返回的数据可能是错误的, 所以在回调函数中首先要判断后台返回的信息是否正确, 如果正确才可以继续执行。代码如下:

```
//第三步: 注册回调函数
xmlHttp.onreadystatechange = function() {
    if (xmlHttp.readyState == 4) {
        if (xmlHttp.status == 200) {
            var obj = document.getElementById(id);
            obj.innerHTML = xmlHttp.responseText;
        } else {
            alert("AJAX服务器返回错误!");
        }
    }
}
```

在上面代码中, `xmlHttp.readyState` 是存有 XMLHttpRequest 的状态。从 0 到 4 发生变化。0: 请求未初始化。1: 服务器连接已建立。2: 请求已接收。3: 请求处理中。4: 请求已完成, 且响应已就绪。所以这里我们判断只有当 `xmlHttp.readyState` 为 4 的时候才可以继续执行。

`xmlHttp.status` 是服务器返回的结果, 其中 200 代表正确。404 代表未找到页面, 所以这里我们判断只有当 `xmlHttp.status` 等于 200 的时候才可以继续执行。

```
var obj = document.getElementById(id);obj.innerHTML = xmlHttp.responseText;
```


这段代码就是回调函数的核心内容，就是获取后台返回的数据，然后将这个数据赋值给id为testid的div。xmlHttpRequest对象有两个属性都可以获取后台返回的数据，分别是：responseText和responseXML，其中responseText是用来获得**字符串形式**的响应数据，responseXML是用来获得**XML形式**的响应数据。至于选择哪一个是取决于后台给返回的数据的，这个例子里我们只是显示一条字符串数据所以选择的是responseText。responseXML将会在以后的例子中介绍。

AJAX状态值与状态码区别 **AJAX状态值是指**，运行AJAX所经历过的几种状态，无论访问是否成功都将响应的步骤，可以理解成为AJAX运行步骤。如：正在发送，正在响应等，由AJAX对象与服务器交互时所得；使用“ajax.readyState”获得。（由数字1~4单位数字组成）**AJAX状态码是指**，无论AJAX访问是否成功，由HTTP协议根据所提交的信息，服务器所返回的HTTP头信息代码，该信息使用“ajax.status”所获得；（由数字1XX,2XX三位数字组成，详细查看RFC）这就是我们在使用AJAX时为什么采用下面的方式判断所获得的信息是否正确的原因。

```
if (ajax.readyState == 4 && ajax.status == 200) {。。。};}
```

AJAX运行步骤与状态值说明 在AJAX实际运行当中，对于访问XMLHttpRequest (XHR) 时并不是一次完成的，而是分别经历了多种状态后取得的结果，对于这种状态在AJAX中共有5种，分别是：0 - (未初始化)还没有调用send()方法 1 - (载入)已调用send()方法，正在发送请求 2 - (载入完成)send()方法执行完成， 3 - (交互)正在解析响应内容 4 - (完成)响应内容解析完成，可以在客户端调用了 对于上面的状态，其中“0”状态是在定义后自动具有的状态值，而对于成功访问的状态（得到信息）我们大多数采用“4”进行判断。

AJAX状态码说明 1：请求收到，继续处理 2：操作成功收到，分析、接受 3：完成此请求必须进一步处理 4：请求包含一个错误语法或不能完成 5：服务器执行一个完全有效请求失败

再具体就如下：

100——客户必须继续发出请求 101——客户要求服务器根据请求转换HTTP协议版本 200——交易成功 201——提示知道新文件的URL 202——接受和处理、但处理未完成 203——返回信息不确定或不完整 204——请求收到，但返回信息为空 205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件 206——服务器已经完成了部分用户的GET请求 300——请求的资源可在多处得到 301——删除请求数据 302——在其他地址发现了请求数据 303——建议客户访问其他URL或访问方式 304——客户端已经执行了GET，但文件未变化 305——请求的资源必须从服务器指定的地址得到 306——前一版本HTTP中使用的代码，现行版本中不再使用 307——申明请求的资源临时性删除 400——错误请求，如语法错误 401——请求授权失败 402——保留有效ChargeTo头响应 403——请求不允许 404——没有发现文件、查询或URI 405——用户在Request-Line字段定义的方法不允许 406——根据用户发送的Accept拖，请求资源不可访问 407——类似401，用户必须首先在代理服务器上得到授权 408——客户端没有在用户指定的饿时间内完成请求 409——对当前资源状态，请求不能完成 410——服务器上不再有此资源且无进一步的参考地址 411——服务器拒绝用户定义的Content-Length属性请求 412——一个或多个请求头字段在当前请求中错误 413——请求的资源大于服务器允许的大小 414——请求的资源URL长于服务器允许的长度 415——请求资源不支持请求项目格式 416——请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If-Range请求头字段 417——服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求 500——服务器产生内部错误 501——服务器不支持请求的函数 502——服务器暂时不可用，有时是为了防止发生系统过载 503——服务器过载或暂停维修 504——关口过载，服务器使用另一个关口或服务来响应用户，等待时间设定值较长 505——服务器不支持或拒绝支请求头中指定的HTTP版本

2.4 发送请求


```
//第四步：设置发送请求的内容和发送报送。然后发送请求
var uname= document.getElementsByName("userName")[0].value;
var upass= document.getElementsByName("userPass")[0].value ;
var params = "userName=" + uname+ "&userPass=" +upass+ "&time=" + Math.random();

// 增加time随机参数，防止读取缓存
xmlHttpRequest.setRequestHeader("Content-type", "application/x-www-form-
urlencoded;charset=UTF-8");
// 向请求添加 HTTP 头，POST如果有数据一定加!!!
xmlHttpRequest.send(params);
```

如果需要像 HTML 表单那样 POST 数据，请使用 `setRequestHeader()` 来添加 HTTP 头。然后在 `send()` 方法中规定您希望发送的数据。

三 jquery的ajax操作

3.1传统方式实现Ajax的不足

步骤繁琐

方法、属性、常用值较多不好记忆

3.2 ajax()方法

可以通过发送 HTTP 请求加载远程数据，是 jQuery 最底层的 Ajax 实现，具有较高灵活性。

`$.ajax([settings]);` //参数 settings 是 `$.ajax ()` 方法的参数列表，用于配置 Ajax 请求的键值对集合;

```
$.ajax({
    url:请求地址
    type:"get | post | put | delete " 默认是get,
    data:请求参数 {"id":"123","pwd":"123456"},
    dataType:请求数据类型"html | text | json | xml | script | jsonp ",
    success:function(data,dataTextStatus,jqxhr){ },//请求成功时
    error:function(jqxhr,textStatus,error)//请求失败时
})
```

3.3 get() 方法通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 `$.ajax` 。

```
$.get(url,data,function(result) {
    //省略将服务器返回的数据显示到页面的代码
});
```

url:请求的路径

data:发送的数据

success:成功函数

datatype 返回的数据

3.4 post() 方法通过远程 HTTP GET 请求载入信息。

```
$.post(url,data,function(result) {  
    //省略将服务器返回的数据显示到页面的代码  
});
```

url:请求的路径

data:发送的数据

success:成功函数

datatype 返回的数据

四 JSON

4.1、什么是JSON

JSON (JavaScript Object Notation) 是一种轻量级的数据交换格式。易于人阅读和编写。同时也易于机器解析和生成。它基于JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999的一个子集。JSON采用完全独立于语言的文本格式，但是也使用了类似于C语言家族的习惯（包括C, C++, C#, Java, JavaScript, Perl, Python等）。这些特性使JSON成为理想的数据交换语言。

4.2、JSON对象定义和基本使用

在标准的json格式中，json对象由在括号括起来，对象中的属性也就是json的key是一个字符串，所以一定要使用双引号引起来。每组key之间使用逗号进行分隔。

4.2.1、JSON的定义

Json定义格式：

```
var 变量名 = {  
    "key" : value ,           // Number类型  
    "key2" : "value" ,       // 字符串类型  
    "key3" : [] ,            // 数组类型  
    "key4" : {} ,            // json 对象类型  
    "key5" : [{},{},{}]      // json 数组  
};
```

4.2.2、JSON对象的访问

json对象，顾名思义，就知道它是一个对象。里面的key就是对象的属性。我们要访问一个对象的属性，只需要使用【对象名.属性名】的方式访问即可。

```
<script type="text/javascript">  
  
    // json的定义  
  
    var jsons = {  
  
        "key1": "abc", // 字符串类型
```

```

        "key2":1234,    // Number

        "key3":[1234,"21341","53"], // 数组

        "key4":{           // json类型

            "key4_1" : 12,

            "key4_2" : "kkk"

        },

        "key5":[{           // json数组

            "key5_1_1" : 12,

            "key5_1_2" : "abc"

        },{

            "key5_2_1" : 41,

            "key5_2_2" : "bbj"

        }]

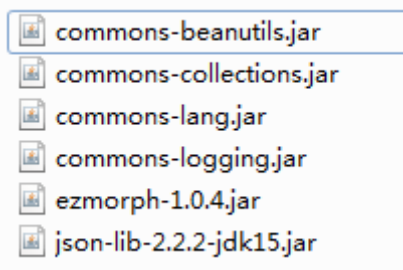
    };

    // 访问json的属性
    alert(jsons.key1); // "abc"
    // 访问json的数组属性
    alert(jsons.key3[1]); // "21341"
    // 访问json的json属性
    alert(jsons.key4.key4_1); //12
    // 访问json的json数组
    alert(jsons.key5[0].key5_1_2); // "abc"
</script>

```

4.3、JSON在java中的使用(重要)

我们要使用json和java中使用，我们需要使用到一个第三方的包。它就是



java对象和json之间的转换

《1》单个对象或map集合

java->json:

```

Users user2=new Users();
user2.setUsername("李四");
user2.setPassword("abc");
user2.setAge(20);
JSONObject obj=JSONObject.fromObject(user);//obj就是json格式的

```

json->java:

```

String str="{ 'username': '李四', 'password': 'admin', 'age': 19 }";
JSONObject json=JSONObject.fromObject(str);
Users user=(Users)JSONObject.toBean(json,Users.class);

```

《2》对象集合和json的转换

java集合->json数组:

```

List list=new ArrayList();
list.add("dd");
list.add("aa");
JSONArray obj=JSONArray.fromObject(list);//set也是这么转

```

json数组->java集合:

方式1:

```

String str2="[{ 'age': 20, 'password': 'abc', 'username': '李四' },
{ 'age': 10, 'password': 'adb', 'username': '张三' }]";
JSONArray json2=JSONArray.fromObject(str2);
Object[] obj=(Object[])JSONArray.toArray(json2,Users.class);

```

方式2:

```

String str3="[{ 'age': 20, 'password': 'abc', 'username': '李四' },
{ 'age': 10, 'password': 'adb', 'username': '展示干' }]";

JSONArray json3=JSONArray.fromObject(str3);
//默认转换成ArrayList
List<Users> list=(List<Users>) JSONArray.toCollection(json3,Users.class);

```

ajax实例2-实现数据的自动填充:

页面:

```

<script type="text/javascript" src="js/jquery-1.8.3.min.js"></script>
<script>
    $(function(){
        $("[name='uid']").blur(function(){
            var uid=$(this).val();
            $.ajax({
                url:"getuser",
                data:"userid="+uid,
                type:"post",
                dataType:"JSON",//设置服务器端响应回来的格式
                success:function(rs){
                    //alert(rs.username);
                }
            });
        });
    });

```

```

        $("[name='uname']").val(rs.username);
        $("[name='address']").val(rs.address);
        if(rs.sex=='男'){
            $("#boy")[0].checked=true;
        }else{
            $("#girl")[0].checked=true;
        }
    }
}

});
});
})

</script>
</head>
<body>
userid:<input type="text" name="uid"><br>
username <input type="text" id="uname" name="uname"><br>
address <input type="text" id="address" name="address"><br>
sex:<input type="radio" name="sex" id="boy">男
<input type="radio" name="sex" id="girl">女
</body>

```

后台:

实体类

```

public class Users {
    private String username;
    private String password;
    private String address;
    private String sex;
}

```

处理类:

```

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    String userid=req.getParameter("userid");

    Users users=new Users();
    switch (Integer.parseInt(userid)){
        case 1:
            users.setUsername("张安");
            users.setAddress("北京");
            users.setSex("男");
            break;
        case 2:
            users.setUsername("张安1");
            users.setAddress("北京2");
            users.setSex("女");
            break;
        case 3:
            users.setUsername("张安3");
            users.setAddress("北京3");
            users.setSex("男");
            break;
    }
}

```

```
        default:
            users.setUsername("");
            users.setAddress("");
            users.setSex("");
    }
    //将users对象响应给客户端，使用PrintWriter来实现
    resp.setContentType("text/html;charset=utf-8");
    PrintWriter writer = resp.getWriter();

    //将对象转换成json格式
    JSONObject jsonObject = JSONObject.fromObject(users);
    writer.print(jsonObject);

}
```