# AJAX&JSON

## 学习目标：

1. 能够理解异步的概念
2. 能够了解原生js的ajax
3. 能够使用jQuery的$.get()进行访问
4. 能够使用jQuery的$.post()进行访问
5. 能够使用jQuery的$.ajax()进行访问
6. 能够使用jQuery3.0的$.get()新增签名进行访问
7. 能够使用jQuery3.0的$.post()新增签名进行访问
8. 能够掌握json的三种数据格式
9. 能够使用json转换工具Jackson进行json格式字符串的转换
10. 能够完成用户名是否存在的查重案例
11. 能够完成自动补全的案例

# 反馈

|  |
| --- |
| 对于路径和加不加斜杆全靠猜  全路径=“/工程名字/资源路径”  相对路径=”资源路径”  浏览器执行的路径，第一个“/”是访问webapps发布资源的根目录  Location.href=/工程名字/资源路径;前端js代码，浏览器执行  a标签，href=/工程名字/资源路径，静态代码，浏览器执行  <form action=” /工程名字/资源路径”>,静态代码，浏览器执行  response.sendRedirect(/工程名字/资源路径)，第二次请求是由浏览器执行  cookie.setPath(/工程名字/资源路径),浏览器使用这个路径判断是否携带cookie带到服务器端  response.setHead(“refresh”,”3;url=/工程名字/资源路径”);浏览器执行定时跳转，底层原理重定向    服务器执行的路径，第一个“/”是访问webapps目录里面当前工程的根目录  request.getRequestDispatcher(/资源路径).forward(request,response); 服务器执行当前路径跳转  以上都可以写相对路径（相对当前资源的路径），就是不要写“/”，直接写资源路径 |
| 基本上是不看笔记就做不出案例，这样子就慌得不行了~  非常正常，笔记上的案例要学习解决问题的思想，如果不看笔记就能写出来那就是天才（只有过目不忘才能做到）  **学习方法**  看笔记---》学习思想----》关闭笔记敲出案例-----》写自己笔记进行总结-----》以后复习思想 |
| 怎么扩充大脑内存  参考上面的学习方法 |
| jQuery.fn.extend(object) 中的fn指的是什么?  fn代表对jq对象扩展，这是固定写法代表对jq对象 |
| servlet的线程安全吗?  Servlet创建是单例模式，只要是单例模式成员变量就会有线程安全问题。  解决方法：   * 1. 尽量不要在成员变量位置定义变量   2. 加锁解决（同步代码块，乐观锁，悲观锁等）   乐观锁，多个任务同时操作，但是最后只有一个最快完成的操作成功，java乐观锁有兴趣同学了解AtomicInteger等类  悲观锁，就是同步代码块解决，多个任务一个一个进行排队 |
| 好想项目实战和期待项目实战  马上就到了。。。 |

# ajax概述

## 什么是ajax

Ajax 即"Asynchronous Javascript And XML"（异步 JavaScript 和 XML），是指一种创建交互式网页应用的网页开发技术。Ajax = 异步 JavaScript 和 XML（标准通用标记语言的子集）。通过在后台与服务器进行少量数据交换，Ajax 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页（不使用 Ajax）如果需要更新内容，必须重载整个网页页面。

## ajax的应用场景

ajax的应用场景很多，常见的应用场景如下：

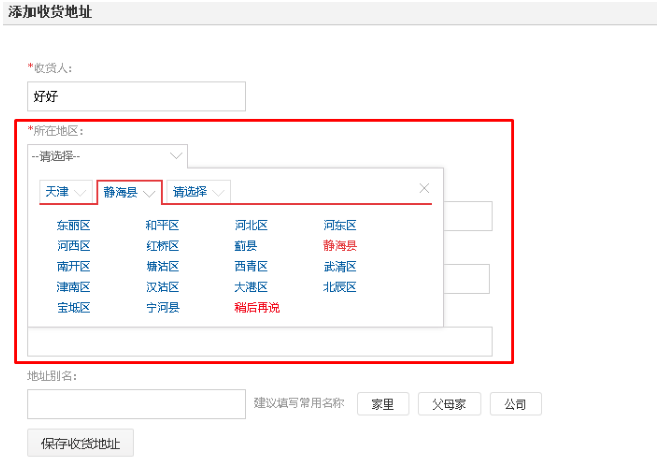
### 检查用户名是否已经被注册

很多站点的注册页面都具备自动检测用户名是否存在的友好提示，该功能整体页面并没有刷新，但仍然可以异步与服务器端进行数据交换，查询用户的输入的用户名是否在数据库中已经存在。



### 省市二联下拉框联动

很多站点都存在输入用户地址的操作，在完成地址输入时，用户所在的省份是下拉框，当选择不同的省份时会出现不同的市区的选择，这就是最常见的省市联动效果。



### 内容自动补全

不管是专注于搜索的百度，还是站点内商品搜索的京东，都有搜索功能，在i搜索框输入查询关键字时，整个页面没有刷新，但会根据关键字显示相关查询字条，这个过程是异步的。

**百度的搜索补全功能：**



**京东的搜索补全功能：**



## 同步方式与异步方式的区别

**同步方式发送请求**

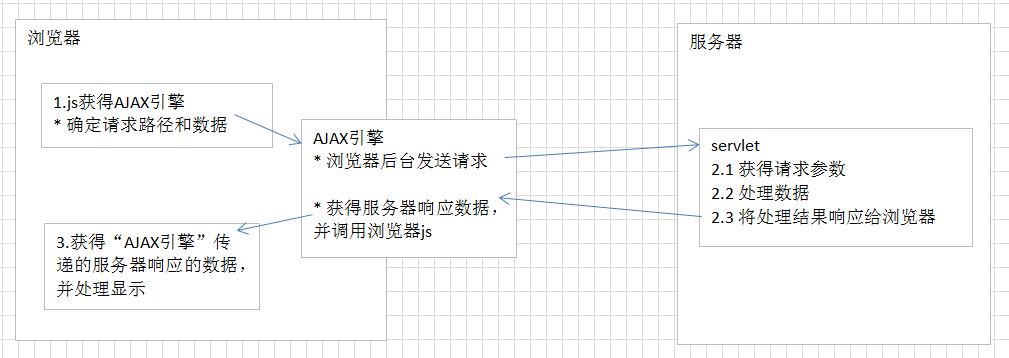
发送一个请求，需要等待响应返回，然后才能够发送下一个请求，如果该请求没有响应，不能发送下一个请求，客户端会处于一直等待过程中。

**异步方式发送请求**

发送一个请求，不需要等待响应返回，随时可以再发送下一个请求，即不需要等待。

|  |
| --- |
|  |

## AJAX原理分析



**AJAX引擎会在不刷新浏览器地址栏的情况下，发送异步请求**

1.1使用JavaScript获得浏览器内置的AJAX引擎（XMLHttpRequest对象）

1.2使用js确定请求路径和请求参数

1.3 AJAX引擎对象根据请求路径和请求参数进行发送请求

**服务器接收到ajax引擎的请求进行处理**

2.1 服务器获得请求参数数据

2.2 服务器处理请求业务（调用业务层代码）

2.3 服务器响应数据给ajax引擎

**AJAX引擎获得服务器响应的数据，通过执行JavaScript的回调函数将数据更新到浏览器页面具体位置。**

3.1 通过设置给AJAX引擎的回调函数获得服务器响应的数据

3.2 使用JavaScript在指定的位置，显示响应数据，从而局部修改页面的数据，达到局部刷新目的。

# js原生的ajax（了解）

## js原生的ajax的开发步骤

1）创建Ajax引擎对象

2）为Ajax引擎对象绑定监听（监听服务器已将数据响应给引擎）

3）绑定提交地址

4）发送请求

5）监听里面处理响应数据

## js原生的ajax的代码实现

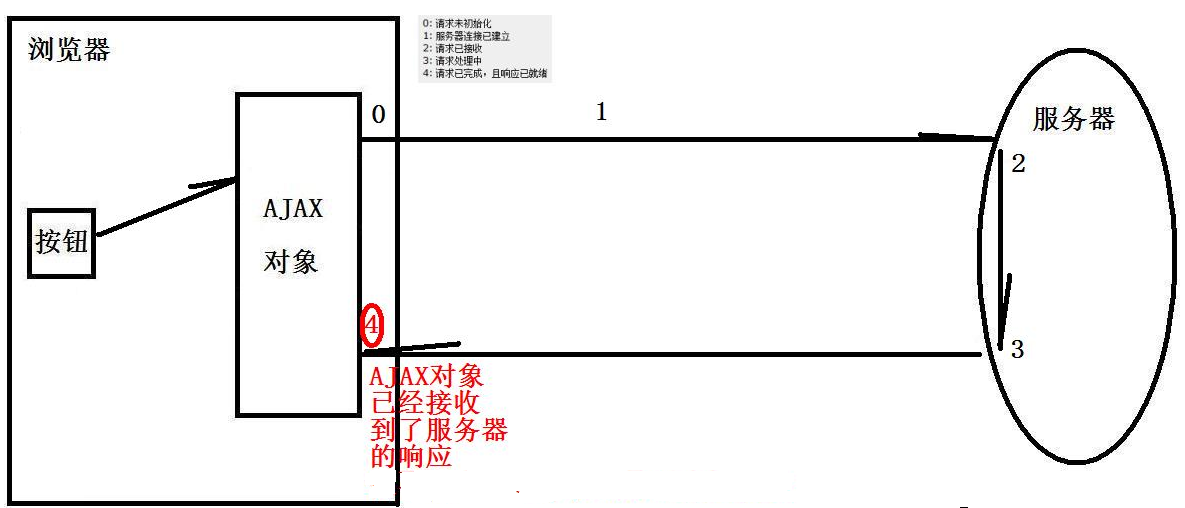
### js代码

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script type="text/javascript"**>  *//同步请求servlet* **function** *sendRequest*() {  ***location***.href=**"Ajax1Servlet"**;  }    *//异步get请求（了解）* **function** *sendSyncGet*() {   *//让ajax引擎对象发送异步请求  //1.创建ajax引擎对象* **var** xmlHttp = **new *XMLHttpRequest***();  *//2.设置回调函数（就是服务器响应回来后触发的函数）* xmlHttp.onreadystatechange=**function** () {   *//onreadystatechange,readyState是ajax引擎与服务器通信过程状态码，0~4  // 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  //ajax引擎与服务器通信过程状态码改变的时候，每次改变都会调用这个函数   //回调函数的目的：处理服务器响应回来的数据  //不仅要获取服务器响应的结果，还要保证通信过程正常，就是保证http通信状态码200* **if**(xmlHttp.readyState==4 && xmlHttp.status==200){  *//来到这里，说明通信正常，并且接收到了服务器响应的数据  //获取服务器返回的数据* **var** content =xmlHttp.responseText;  *alert*(content);  }   };   *//3.初始化请求（设置请求路径，请求方法类型）  //xmlHttp.open(method,url)  // method,请求方法类型，get或post  // url，请求服务器的资源地址* xmlHttp.open(**"get"**,**"Ajax1Servlet?name=admin&pwd=123"**);  *//4.发送异步请求* xmlHttp.send();     }  </**script**> </**head**> <**body**>   <**input type="button" value="同步请求" onclick="***sendRequest*();**"**>**&nbsp;** <**input type="button" value="异步get请求" onclick="***sendSyncGet*();**"**> </**body**> </**html**> |

### Servlet代码

|  |
| --- |
| **package** com.itheima.web;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"Ajax1Servlet"**, urlPatterns = **"/Ajax1Servlet"**) **public class** Ajax1Servlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取get请求数据* String name = request.getParameter(**"name"**);  String pwd = request.getParameter(**"pwd"**);  System.***out***.println(**"name:"**+name);  System.***out***.println(**"pwd:"**+pwd);  *//输出数据给浏览器* response.getWriter().write(**"hello post ajax"**);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取get请求数据* String name = request.getParameter(**"name"**);  String pwd = request.getParameter(**"pwd"**);  System.***out***.println(**"name:"**+name);  System.***out***.println(**"pwd:"**+pwd);  *//输出数据给浏览器* response.getWriter().write(**"hello get ajax"**);  } } |

### ajax引擎连接状态readyState值0~4变化过程



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

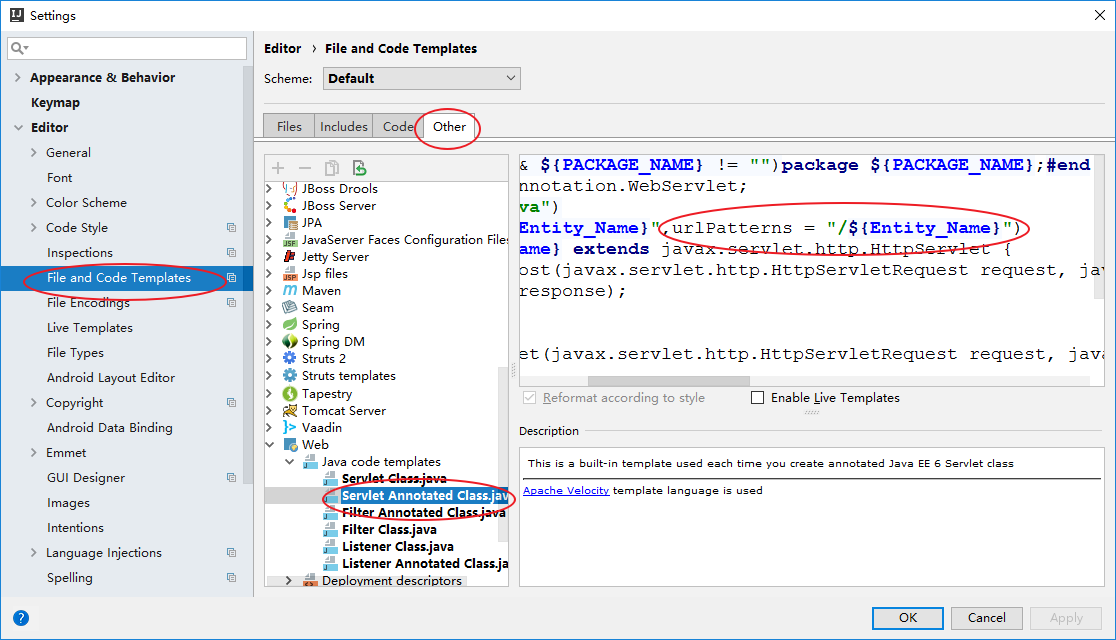
3: 请求处理中

4: 请求已完成，且响应已就绪

这里状态值4只能说明接收到了服务器的响应服务器处理ajax请求结束，但是不能代表正确的获取了服务器的响应，需要配合http状态码200两个条件就可以说明正确的获取了服务器响应。只有这两个条件满足，xmlhttp.responseText才可以获取到正确的响应数据。

|  |
| --- |
| xmlhttp.onreadystatechange = function(){  if(xmlhttp.readyState == 4){  if(xmlhttp.status == 200){  alert("响应数据" + xmlhttp.responseText);  }  }  }; |

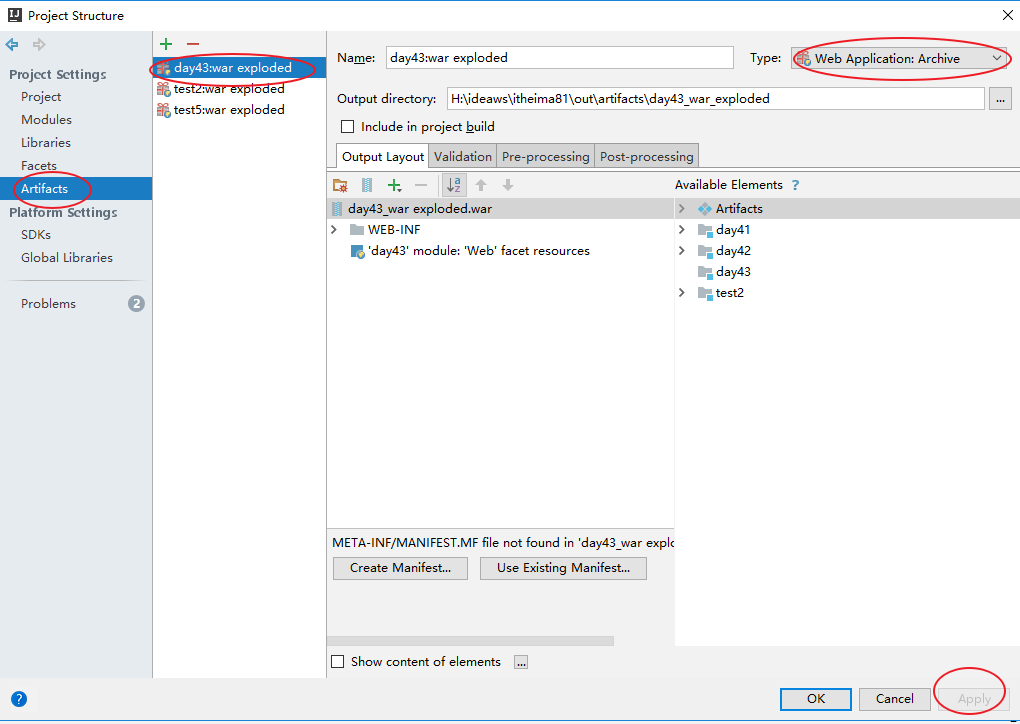
### 设置servlet代码生成模板

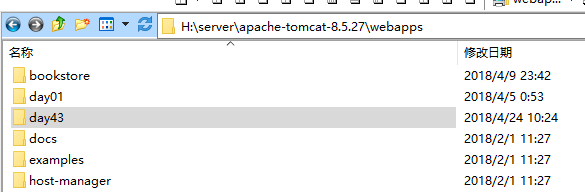


自动生成如下

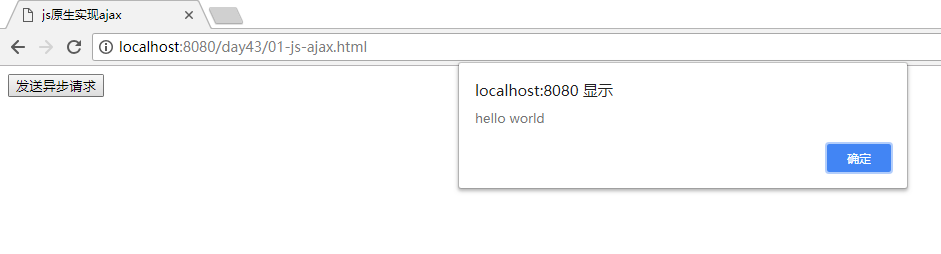


### 部署web项目到服务器webapps里面





### 代码实现效果



# jQuery框架的ajax（重点）

## jQuery框架的ajax简介

jquery是一个优秀的js框架，自然对js原生的ajax进行了封装，封装后的ajax的操 作方法更简洁，功能更强大，与ajax操作相关的jquery方法有如下几种，但开发中 经常使用的有三种：

| **请求方式** | **语法** |
| --- | --- |
| GET请求 | $.get(url, [data], [callback], [type]) |
| POST请求 | $.post(url, [data], [callback], [type]) |
| AJAX请求 | $.ajax([settings]) |
| GET请求 | $.get([settings]) |
| POST请求 | $.post([settings]) |

## GET请求方式

### GET请求方式概述

通过远程 HTTP GET 请求载入信息。这是一个简单的 GET 请求功能，如需复杂的ajax参数设置请使用$.ajax。

### GET请求方式语法

jQuery.get(url, [data], [callback], [type])

其中，参数说明如下：

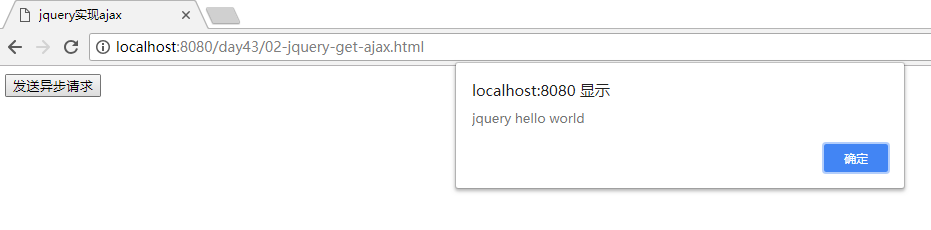
| **参数名称** | **解释** |
| --- | --- |
| url | 请求的服务器端url地址 |
| data | 发送给服务器端的请求参数，格式可以是key=value，也可以是js对象 |
| callback | 当请求成功后的回掉函数，可以在函数体中编写我们的逻辑代码 |
| type | 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等 |

### GET请求方式代码

#### js代码

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**>  <**script type="text/javascript"**>   *//jquery异步get请求* **function** *sendSyncGet*() {   *//语法  /\*  \* $.get(url[,data][,callback][,type]);  \* 参数1：url，服务器资源地址  \* 参数2：data,请求参数数据，支持2种格式  \* 第一种，key和value格式，key1=value1&key2=value2...  \* 第二种，js对象格式，{key1:value1,key2:value2...}  \* key可以将双引号也可以不加  \* value是字符串就加双引号，不是可以不加  \* js对象就是json对象  \* 参数3：callback，回调函数  \* 格式： function(result){ .... }  \* result，服务器返回的数据  \* 这个函数都已经判断好4和200，不用我们判断  \* 参数4：type,设置服务器返回的数据类型  \* text,不设置默认这个，服务器返回的是文本字符串  \* json,将服务器返回的json字符串转换为json对象(js对象)  \* json对象，{name:"zhangsan",age:20};  \* json字符串,"{\"name\":\"zhangsan\",\"age\":20}"  \* json字符串是有要求的，key必须使用双引号括起来  \* value是字符串就是用双引号括起来，不是可以不使用  \* xml，返回的数据类型为xml格式数据（已经被淘汰），因为解析麻烦  \*  \* \*/* **var** url = **"Ajax1Servlet"**;  **var** data = **"name=admin&pwd=123"**;  **var** *callback* = **function** (result) {  *alert*(result);  };  **$**.get(url,data,*callback*);    }    </**script**> </**head**> <**body**>   <**input type="button" value="异步get请求" onclick="***sendSyncGet*();**"**> </**body**> </**html**> |

#### 代码实现效果



## POST请求方式

### POST请求方式概述

通过远程 HTTP POST 请求载入信息。这是一个简单的 POST 请求功能，如需复杂的ajax参数设置请使用$.ajax。

### POST请求方式语法

jQuery.post(url, [data], [callback], [type])

其中，参数说明如下：

| **参数名称** | **解释** |
| --- | --- |
| url | 请求的服务器端url地址 |
| data | 发送给服务器端的请求参数，格式可以是key=value，也可以是js对象 |
| callback | 当请求成功后的回掉函数，可以在函数体中编写我们的逻辑代码 |
| type | 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等 |

### POST请求方式代码

#### js代码

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**>  <**script type="text/javascript"**>   *//jquery异步post请求(推荐使用这个)* **function** *sendSyncPost*() {   *//语法  /\*  \* $.post(url[,data][,callback][,type]);  \* 参数1：url，服务器资源地址  \* 参数2：data,请求参数数据，支持2种格式  \* 第一种，key和value格式，key1=value1&key2=value2...  \* 第二种，js对象格式，{key1:value1,key2:value2...}  \* key可以将双引号也可以不加  \* value是字符串就加双引号，不是可以不加  \* js对象就是json对象  \* 参数3：callback，回调函数  \* 格式： function(result){ .... }  \* result，服务器返回的数据  \* 这个函数都已经判断好4和200，不用我们判断  \* 参数4：type,设置服务器返回的数据类型  \* text,不设置默认这个，服务器返回的是文本字符串  \* json,将服务器返回的json字符串转换为json对象(js对象)  \* json对象，{name:"zhangsan",age:20};  \* json字符串,"{\"name\":\"zhangsan\",\"age\":20}"  \* json字符串是有要求的，key必须使用双引号括起来  \* value是字符串就是用双引号括起来，不是可以不使用  \* xml，返回的数据类型为xml格式数据（已经被淘汰），因为解析麻烦  \*  \* \*/* **var** url = **"Ajax1Servlet"**;  **var** data = {**name**:**"admin"**,**pwd**:**"123"**};  **var** *callback* = **function** (result) {  *alert*(result);  };  **$**.post(url,data,*callback*);    }    </**script**> </**head**> <**body**>   <**input type="button" value="异步get请求" onclick="***sendSyncPost*();**"**> </**body**> </**html**> |

#### 代码实现效果



## AJAX请求方式

### AJAX请求方式概述

通过 HTTP 请求加载远程数据。jQuery 底层 AJAX 实现。简单易用的高层实现见get和post方法。$.ajax()方法可以更加详细的设置底层的参数。

### AJAX请求方式语法

jQuery.ajax([settings])

其中，settings是一个js字面量形式的对象，格式是{name:value,name:value... ...}，常用的name属性名如下：

| **属性名称** | **解释** |
| --- | --- |
| url | 请求的服务器端url地址 |
| async | (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false |
| data | 发送到服务器的数据，可以是键值对形式，也可以是js对象形式 |
| type | (默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET" |
| dataType | 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等 |
| success | 请求成功后的回调函数 |
| error | 请求失败时调用此函数 |

### AJAX请求方式代码

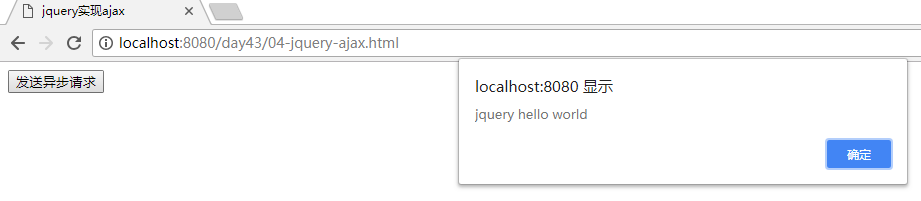
#### js代码

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**>  <**script type="text/javascript"**>   *//jquery异步ajax请求* **function** *sendSyncAjax*() {   *//语法  /\*  \* $.ajax({js对象});  \*  \* url 请求的服务器端url地址  async (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false  data 发送到服务器的数据，可以是键值对形式，也可以是js对象形式  type (默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET"  dataType 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等  success 请求成功后的回调函数  error 请求失败时调用此函数  \*  \*  \* \*/* **$**.ajax({  **url**:**"Ajax2Servlet"**,  **data**:{**name**:**"admin"**,**pwd**:**"1234"**},  **type**:**"post"**,  */\* dataType:"text",\*/* success:**function** (result) {  *alert*(result)  },  error:**function** (result) {  *//result，是js对象，可以获取到相关信息  alert*(**"服务器忙。。。请求稍后再试"**);  }  });     }    </**script**> </**head**> <**body**>   <**input type="button" value="异步ajax请求" onclick="***sendSyncAjax*();**"**> </**body**> </**html**> |

#### Servlet代码

|  |
| --- |
| **package** com.itheima.web;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"Ajax2Servlet"**, urlPatterns = **"/Ajax2Servlet"**) **public class** Ajax2Servlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取get请求数据* String name = request.getParameter(**"name"**);  String pwd = request.getParameter(**"pwd"**);  System.***out***.println(**"name:"**+name);  System.***out***.println(**"pwd:"**+pwd);  *//输出数据给浏览器* response.getWriter().write(**"hello post ajax"**);  **throw new** RuntimeException(**"error post"**);  }   **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取get请求数据* String name = request.getParameter(**"name"**);  String pwd = request.getParameter(**"pwd"**);  System.***out***.println(**"name:"**+name);  System.***out***.println(**"pwd:"**+pwd);  *//输出数据给浏览器* response.getWriter().write(**"hello get ajax"**);  } } |

#### 代码实现效果



## jQuery3.0 的GET新增签名方式

### jQuery3.0 的GET新增签名概述

jQuery 3 为jQuery.get()和 jQuery.post() 这两个工具函数增加了新签名，从而使得它们和 $.ajax() 的接口风格保持一致。

### jQuery3.0 的GET新增签名方式语法

jQuery.get([settings])

其中，settings是一个js字面量形式的对象，格式是{name:value,name:value... ...}，常用的name属性名如下：

| **属性名称** | **解释** |
| --- | --- |
| url | 请求的服务器端url地址 |
| async | (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false |
| data | 发送到服务器的数据，可以是键值对形式，也可以是js对象形式 |
| type | (默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET" |
| dataType | 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等 |
| success | 请求成功后的回调函数 |
| error | 请求失败时调用此函数 |

### jQuery3.0 的GET新增签名方式代码

#### js代码

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**>  <**script type="text/javascript"**>   *//jquery3.0新特性对get异步请求新增了与ajax方法一样的方法签名* **function** *sendSyncAjax*() {   *//语法  /\*  \* jquery旧方式：$.get(url,data,callbcak,type);  \* jquery新特性方式：$.get({js对象});，与ajax方法一样了  \*  \* url 请求的服务器端url地址  async (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false  data 发送到服务器的数据，可以是键值对形式，也可以是js对象形式  dataType 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等  success 请求成功后的回调函数  error 请求失败时调用此函数  \*  \*  \* \*/* **$**.get({  **url**:**"Ajax2Servlet"**,  **data**:{**name**:**"admin"**,**pwd**:**"1234"**},  */\* dataType:"text",\*/* success:**function** (result) {  *alert*(result)  },  error:**function** (result) {  *//result，是js对象，可以获取到相关信息  alert*(**"服务器忙。。。请求稍后再试"**);  }  });     }    </**script**> </**head**> <**body**>   <**input type="button" value="异步ajax请求" onclick="***sendSyncAjax*();**"**> </**body**> </**html**> |

## jQuery3.0 的POST新增签名方式

### jQuery3.0 的POST新增签名概述

jQuery 3 为jQuery.get()和 jQuery.post() 这两个工具函数增加了新签名，从而使得它们和 $.ajax() 的接口风格保持一致。

### jQuery3.0 的POST新增签名方式语法

jQuery.post([settings])

其中，settings是一个js字面量形式的对象，格式是{name:value,name:value... ...}，常用的name属性名如下：

| **属性名称** | **解释** |
| --- | --- |
| url | 请求的服务器端url地址 |
| async | (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false |
| data | 发送到服务器的数据，可以是键值对形式，也可以是js对象形式 |
| type | (默认: "GET") 请求方式 ("POST" 或 "GET")， 默认为 "GET" |
| dataType | 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等 |
| success | 请求成功后的回调函数 |
| error | 请求失败时调用此函数 |

### jQuery3.0 的GET新增签名方式代码

#### js代码​

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**>  <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**>  <**script type="text/javascript"**>   *//jquery3.0新特性对post异步请求新增了与ajax方法一样的方法签名* **function** *sendSyncAjax*() {   *//语法  /\*  \* jquery旧方式：$.post(url,data,callbcak,type);  \* jquery新特性方式：$.post({js对象});，与ajax方法一样了  \*  \* url 请求的服务器端url地址  async (默认: true) 默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false  data 发送到服务器的数据，可以是键值对形式，也可以是js对象形式  dataType 预期的返回数据的类型，取值可以是 xml, html, script, json, text, \_defaul等  success 请求成功后的回调函数  error 请求失败时调用此函数  \*  \*  \* \*/* **$**.post({  **url**:**"Ajax2Servlet"**,  **data**:{**name**:**"admin"**,**pwd**:**"1234"**},  */\* dataType:"text",\*/* success:**function** (result) {  *alert*(result)  },  error:**function** (result) {  *//result，是js对象，可以获取到相关信息  alert*(**"服务器忙。。。请求稍后再试"**);  }  });     }    </**script**> </**head**> <**body**>   <**input type="button" value="异步ajax请求" onclick="***sendSyncAjax*();**"**> </**body**> </**html**> |

# 综合案例 ：检测用户名是否已经被注册

## 需求

在用户注册页面，输入用户名，当用户名输入框失去焦点时，发送异步请求，将输入框的用户名传递给服务器端进行是否存在的校验。服务器返回json数据。

## 代码效果



## 布局代码

|  |
| --- |
| <**body**> <**div**>  <**font**>会员注册</**font**>USER REGISTER  <**form class="form-horizontal" style="margin-top**: 5**px**;**"**>  <**table**>  <**tr**>  <**td**>用户名</**td**>  <**td**>  <**input type="text" name="username" placeholder="请输入用户名"**>  <**span id="usernameInfo" style="color**:**red"**></**span**>  </**td**>  </**tr**>  <**tr**>  <**td**>密码</**td**>  <**td**>  <**input type="password" placeholder="请输入密码"**>  </**td**>  </**tr**>  </**table**>  <**input type="submit" value="注册"**/>  </**form**> </**div**> </**body**> |

## 代码实现

### jsp代码

|  |
| --- |
| <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>Title</**title**>  <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**>  <**script type="text/javascript"**>   $(**function** () {  *//给username注册失去焦点事件，异步请求检查用户名是否被注册，要求服务器返回json字符串数据* $(**"#username"**).blur(**function** () {  *//校验用户名有效性* **var** username = $(**this**).val();  *//如果有效，发送异步校验请求* **if**(username){  *//发送异步请求* **var** url=**"CheckUsernameServlet"**;*//处理注册的请求Servlet* **var** data={**name**:username};  **var** *callback* =**function** (result) {  *//result，{flag:true,msg:"xxx"};* **if**(result.flag){  *//可以注册  // alert();* $(**"#usernameInfo"**).css(**"color"**,**"green"**);  }**else**{  *//已被占用或者发生异常* $(**"#usernameInfo"**).css(**"color"**,**"red"**);  }  $(**"#usernameInfo"**).html(result.msg);  };  **var** type=**"json"**;*//将服务器返回的json字符串转换为js对象* $.post(url,data,*callback*,type);  }  });  });    </**script**> </**head**> <**body**> <**div**>  <**font**>会员注册</**font**>USER REGISTER  <**form class="form-horizontal" style="margin-top**: 5**px**;**"**>  <**table**>  <**tr**>  <**td**>用户名</**td**>  <**td**>  <**input type="text" id="username" name="username" placeholder="请输入用户名"**>  <**span id="usernameInfo" style="color**:**red"**></**span**>  </**td**>  </**tr**>  <**tr**>  <**td**>密码</**td**>  <**td**>  <**input type="password" placeholder="请输入密码"**>  </**td**>  </**tr**>  </**table**>  <**input type="submit" value="注册"**/>  </**form**> </**div**> </**body**>  </**html**> |

​

### Servlet代码

|  |
| --- |
| **package** com.itheima.servlet;  **import** com.itheima.model.User; **import** com.itheima.service.UserService;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException;  @WebServlet(name = **"CheckUsernameServlet"**, urlPatterns = **"/CheckUsernameServlet"**) **public class** CheckUsernameServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   *//实例用户业务类* **private** UserService **userService** = **new** UserService();  **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  *//获取post提交的数据* request.setCharacterEncoding(**"utf-8"**);  response.setContentType(**"text/html;charset=utf-8"**);   *//定义json* String jsonData = **null**;   **try** {  *//获取用户输入的用户名数据* String username = request.getParameter(**"name"**);  *//校验用户名是否存在，可以根据用户名查找用户对象* User user = **userService**.getUserByUserName(username);  *//判断user有效性* **if**(user==**null**){  *//可以注册* jsonData=**"{\"flag\":true,\"msg\":\"可以注册\"}"**;  }**else**{  *//已被占用* jsonData=**"{\"flag\":false,\"msg\":\"已被占用\"}"**;  }   } **catch** (Exception e) {  e.printStackTrace();  jsonData=**"{\"flag\":false,\"msg\":\"服务器正忙，请求稍后再试\"}"**;  }   *//返回json字符串给前端* response.getWriter().write(jsonData);   } } |

### service代码

|  |
| --- |
| **package** com.itheima.service;  **import** com.itheima.dao.UserDao; **import** com.itheima.model.User;  */\*\*  \* 用户业务类  \*/* **public class** UserService {   *//实例userDao* **private** UserDao **userDao** = **new** UserDao();  */\*\*  \* 根据用户名查找用户对象  \** ***@param username*** *\** ***@return*** *User  \** ***@throws*** *Exception  \*/* **public** User getUserByUserName(String username)**throws** Exception {  **return userDao**.getUserByUserName(username);  } } |

### dao代码

|  |
| --- |
| **package** com.itheima.dao;  **import** com.itheima.model.User; **import** com.itheima.util.JdbcUtils; **import** org.springframework.dao.DataAccessException; **import** org.springframework.dao.EmptyResultDataAccessException; **import** org.springframework.jdbc.core.BeanPropertyRowMapper; **import** org.springframework.jdbc.core.JdbcTemplate;  **import** java.sql.SQLException;  */\*\*  \* 用户数据库访问类  \*/* **public class** UserDao {  *//核心类* **private** JdbcTemplate **jdbcTemplate** = **new** JdbcTemplate(JdbcUtils.*getDataSource*());   **public** User login(User user)**throws** SQLException {  *//定义sql* String sql=**"select** *\** **from user where username=? and password=?"**;  **try** {  *//执行sql* **return jdbcTemplate**.queryForObject(  sql,**new** BeanPropertyRowMapper<User>(User.**class**),  user.getUsername(),user.getPassword()  );  *//jdbcTemplate如果查不到数据会抛出异常,我们需要抓到* } **catch** (EmptyResultDataAccessException e) {  e.printStackTrace();  **return null**;  }  }   */\*\*  \* 根据用户名查找用户对象  \** ***@param username*** *\** ***@return*** *User  \** ***@throws*** *SQLException  \*/* **public** User getUserByUserName(String username)**throws** SQLException {   **try** {  String sql = **"SELECT** *\** **FROM USER WHERE username=?"**;  **return jdbcTemplate**.queryForObject(sql,**new** BeanPropertyRowMapper<User>(User.**class**),username);  } **catch** (EmptyResultDataAccessException e) {  e.printStackTrace();  **return null**;  }   } } |

# json数据格式

## 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。它基于ECMAScript的一个子集，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。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。

json包含2种概念

第一个概念：json字符串，json格式字符串

第二个概念：json对象，js对象

## json对象的语法格式

json对象有三种数据格式，分别如下：

| **类型** | **语法** | **解释** |
| --- | --- | --- |
| 对象类型 | {name:value,name:value...} | 其中name是字符串类型，而value是任意类型 |
| 数组/集合类型 | [value,value,value...]或[{},{}... ...] | 其中value是任何类型（比如js对象，或数组） |
| 混合类型 | {name:[]... ...} | 合理包裹嵌套对象类型和数组类型 |

## json对象解析练习

### 练习1

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html>  
  <head>  
    <title>json01</title>  
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">  
    <meta http-equiv="description" content="this is my page">  
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">  
  </head>  
  <body>  
​  
  </body>  
    
  <script language="JavaScript">  
 /\*\*  
 \* 案例一  
 \* var person={key:value,key:value}  
 \*    
 \* class Person{  
 \* String firstname = "张";  
 \*   String lastname = "三丰";  
 \*   Integer age = 100;  
 \* }  
 \*   
 \* Person p = new Person();  
 \* System.out.println(p.firstname);  
 \*/  
   
 //json的定义  
 var person = {"firstname":"张","lastname":"三丰","age":100};  
   
 //json解析  
 alert(person.firstname);  
 alert(person.lastname);  
 alert(person.age);  
   
  </script>  
</html>  
​  
​

### 练习2

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html>  
<head>  
<title>json02</title>  
<meta http-equiv="keywords" content="keyword1,keyword2,keyword3">  
<meta http-equiv="description" content="this is my page">  
<meta http-equiv="content-type" content="text/html; charset=UTF-8">  
</head>  
<body>  
​  
</body>  
​  
<script language="JavaScript">  
 /\*\*  
 \* 案例二  
 \* [{key:value,key:value},{key:value,key:value}]  
 \*    
 \*/  
 var json = [  
 {"firstname":"张","lastname":"三丰","age":100},  
 {"firstname":"张","lastname":"翠山","age":58},  
 {"firstname":"张","lastname":"无忌","age":23}  
             ];  
   
 for(var i=0;i<json.length;i++){  
 alert(json[i].lastname);  
 }  
   
  </script>  
</html>  
​

### 练习3

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html>  
  <head>  
    <title>json03</title>  
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">  
    <meta http-equiv="description" content="this is my page">  
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">  
  </head>  
  <body>  
​  
  </body>  
    
  <script language="JavaScript">  
   /\*\*  
 \* 案例三  
 \* {  
 \*   "param":[{key:value,key:value},{key:value,key:value}]  
 \* }  
 \*    
 \*    
 \*/  
   
 var json = {  
 "baobao":[  
         {"name":"小双","age":18,"addr":"扬州"},  
         {"name":"建宁","age":18,"addr":"北京海淀"},  
         {"name":"龙儿","age":38,"addr":"岛国"},  
         {"name":"阿珂","age":17,"addr":"台湾"}  
         ]  
 }  
   
 //全取  
 for(var i=0;i<json.baobao.length;i++){  
 alert(json.baobao[i].name);  
 }  
   
  </script>  
</html>  
​  
​

### 练习4

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  
<html>  
  <head>  
    <title>insertBefore.html</title>  
    <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">  
    <meta http-equiv="description" content="this is my page">  
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">  
  </head>  
  <body>  
​  
  </body>  
    
  <script language="JavaScript">  
   /\*\*  
 \* 案例四  
 \* {  
 \*   "param1":[{key:value,key:value},{key:value,key:value}],  
 \*   "param2":[{key:value,key:value},{key:value,key:value}],  
 \*   "param3":[{key:value,key:value},{key:value,key:value}]  
 \* }  
 \*    
 \*    
 \*/  
 var json = {  
 "baobao":[  
         {"name":"小双","age":18,"addr":"扬州"},  
         {"name":"建宁","age":18,"addr":"北京海淀"},  
         {"name":"龙儿","age":38,"addr":"岛国"},  
         {"name":"阿珂","age":17,"addr":"台湾"}  
         ],  
 "haohao":[  
 {"name":"楠楠","age":23,"addr":"北京昌平修正"},  
 {"name":"倩倩","age":18,"addr":"上海"}  
         ]  
 }  
   
   
 //取倩倩  
 alert(json.haohao[1].name);  
​  
  </script>  
</html>  
​

## json的转换工具

### 思考

**问题1：**服务器端返回如下数据库表数据给到ajax引擎异步请求，服务器传递的数据肯定是字符串类型,那什么格式字符串方便前端解析数据？



答案：服务器返回json字符串给到ajax引擎，$.post(url,data,callback,type),type=”json”，将服务器返回的字符串转换为json对象，前端得到json对象，可以解析数据。能转换为json对象的数据必须是json格式字符串。

结论：

服务器必须返回json格式字符串

**问题2：**服务器端操作数据库表数据可以获取java对象，那么如何返回json字符串呢？

答：需要将java对象转换为json格式字符串

“{\“id\”:1,\”name\”:\”张三\”,\”password\”:\”123\”}”,json格式字符串要求属性名必须使用双引号

### json转换工具的概述

json的转换工具是通过java封装好的一些jar工具包，直接将java对象或集合转换成json格式的字符串。

### 常见的json转换工具

| **工具名称** | **介绍** |
| --- | --- |
| Jsonlib | Java 类库，需要导入的jar包较多 |
| Gson | google提供的一个简单的json转换工具 |
| Fastjson | alibaba技术团队提供的一个高性能的json转换工具 |
| Jackson | 开源免费的json转换工具，springmvc转换默认使用jackson |

### 开发步骤

1）导入json相关jar包

2）创建java对象或集合

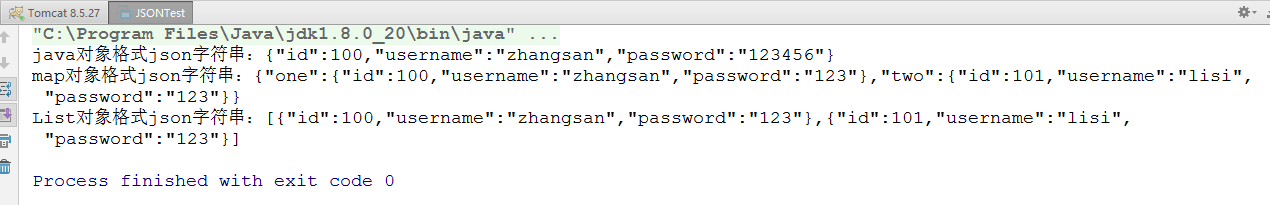
3) 使用jackson的ObjectMapper对象的writeValueAsString方法进行转换

### 转换代码实现

|  |
| --- |
| **package** com.itheima.json;  **import** com.fasterxml.jackson.core.JsonProcessingException; **import** com.fasterxml.jackson.databind.ObjectMapper; **import** com.itheima.model.User;  **import** java.util.ArrayList; **import** java.util.HashMap; **import** java.util.List; **import** java.util.Map;  **public class** JSONTest {  **public static void** main(String[] args) {   **try** {  *//将java对象转换为json字符串* User user = **new** User();  user.setId(100);  user.setUsername(**"zhangsan"**);  user.setPassword(**"123456"**);  *//ObjectMapper(),jackson的核心类* String jsonData = **new** ObjectMapper().writeValueAsString(user);  System.***out***.println(**"java对象格式json字符串："**+jsonData);   *//将Map对象转换为json字符串* Map<String,User> map = **new** HashMap<String,User>();  map.put(**"one"**,**new** User(100,**"zhangsan"**,**"123"**));  map.put(**"two"**,**new** User(101,**"lisi"**,**"123"**));  jsonData = **new** ObjectMapper().writeValueAsString(map);  System.***out***.println(**"map对象格式json字符串："**+jsonData);   *//将List集合转换为json字符串* List<User> userList = **new** ArrayList<User>();  userList.add(**new** User(100,**"zhangsan"**,**"123"**));  userList.add(**new** User(101,**"lisi"**,**"123"**));  jsonData = **new** ObjectMapper().writeValueAsString(userList);  System.***out***.println(**"List对象格式json字符串："**+jsonData);   } **catch** (JsonProcessingException e) {  e.printStackTrace();  }    } } |

​

### 转换控制台效果



# 综合案例

## 综合案例 ：内容自动补全

### 效果



### 需求

在输入框输入关键字，下拉框中异步显示与该关键字相关的商品的名称

### 代码实现

#### 导入数据库脚本

导入素材文件夹中的demo.sql



#### 编写实体模型

|  |
| --- |
| **package** com.itheima.model;  **import** java.io.Serializable;  **public class** User2 **implements** Serializable {   **private int id**;  **private** String **name**;  **private** String **password**;   **public int** getId() {  **return id**;  }   **public void** setId(**int** id) {  **this**.**id** = id;  }   **public** String getName() {  **return name**;  }   **public void** setName(String name) {  **this**.**name** = name;  }   **public** String getPassword() {  **return password**;  }   **public void** setPassword(String password) {  **this**.**password** = password;  }   **public** User2() {  }   **public** User2(**int** id, String name, String password) {  **this**.**id** = id;  **this**.**name** = name;  **this**.**password** = password;  } } |

#### 编写jsp页面

|  |
| --- |
| <%@ **page language**="**java**" **contentType**="**text/html; charset=UTF-8**"  **pageEncoding**="**UTF-8**"%> <!DOCTYPE **html** PUBLIC **"-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd"**> <**html**> <**head**> <**meta http-equiv="Content-Type" content="text/html; charset=UTF-8"**> <**title**>Insert title here</**title**> <**style type="text/css"**>  .**content**{  **width**:643**px**;  **margin**:200**px auto**;  **text-align**: **center**;  }  **input**[**type**=**'text'**]{  **width**:530**px**;  **height**:40**px**;  **font-size**: 14**px**;  }  **input**[**type**=**'button'**]{  **width**:100**px**;  **height**:46**px**;  **background**: **#38f**;  **border**: 0;  **color**: **#fff**;  **font-size**: 15**px** }  .**show**{  **position**: **absolute**;  **width**: 535**px**;  **height**:100**px**;  **border**: 1**px solid #999**;  **border-top**: 0;  **display**: **none**;  } </**style**> <**script type="text/javascript" src="js/jquery-3.3.1.min.js"**></**script**> <**script type="text/javascript"**>  *//加载事件* $(**function** () {   *//给搜索框注册键盘弹起keyup事件* $(**"#word"**).keyup(**function** () {  *//获取搜索内容，判断有效性* **var** word = $(**this**).val();  *//有效，发送异步请求* **if**(word){  $.post({  **url**:**"SearchServlet"**,*//提交请求servlet，获取搜索列表* **data**:{**word**:word},  **dataType**:**"json"**,  success:**function** (result) {  *//result,json对象，[{id:xx,username:xxx,password:xxx},{...}]* **if**(result.**length**>0){  *//数组长度大于0，证明返回数据有效  //拼接每个用户名的html代码，最后将内容设置到.show里面* **var** html = **""**;*//用来拼接用户名* **for**(**var** user **of** result){  html+=**"<div>"**+user.username+**"</div>"**;  }  *//将html设置到.show里面* $(**".show"**).html(html);  $(**".show"**).show();  }  },  error:**function** () {  *alert*(**"服务器忙。。。"**);  *//location.href="index.jsp";* }  });  }  });  }); </**script**> </**head**> <**body**>  <**div class="content"**>  <**img alt="" src="img/logo.png"**><**br**/><**br**/>  <**input type="text" name="word" id="word"**>  <**input type="button" value="搜索一下"**>  <**div class="show"**></**div**>  </**div**> </**body**> </**html**> |

#### 编写Servlet

|  |
| --- |
| **package** com.itheima.web;  **import** com.fasterxml.jackson.databind.ObjectMapper; **import** com.itheima.model.User; **import** com.itheima.service.UserService;  **import** javax.servlet.ServletException; **import** javax.servlet.annotation.WebServlet; **import** javax.servlet.http.HttpServlet; **import** javax.servlet.http.HttpServletRequest; **import** javax.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.util.List;  @WebServlet(name = **"SearchServlet"**, urlPatterns = **"/SearchServlet"**) **public class** SearchServlet **extends** HttpServlet {  **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }   *//实例业务类* **private** UserService **userService** = **new** UserService();  **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {   *//处理post乱码* request.setCharacterEncoding(**"utf-8"**);  response.setContentType(**"text/html;charset=utf-8"**);   **try** {  *//获取搜索条件内容* String word = request.getParameter(**"word"**);  *//调用业务处理搜索商品* List<User> userList = **userService**.getUserListByWord(word);  *//将userList转换为json字符串* String jsonData = **new** ObjectMapper().writeValueAsString(userList);  *//输出给前端* response.getWriter().write(jsonData);  } **catch** (Exception e) {  e.printStackTrace();  *// throw new RuntimeException(e);* response.sendRedirect(request.getContextPath()+**"/index.jsp"**);*//无效果  //由于目前前端异步请求来的，所有的响应都是给ajax引擎对象，目前ajax直接收json字符串。* }   } } |

#### 编写Service

|  |
| --- |
| **package** com.itheima.service;  **import** com.itheima.dao.UserDao; **import** com.itheima.model.User;  **import** java.util.List;  **public class** UserService {   *//实例userDao* **private** UserDao **userDao** = **new** UserDao();  */\*\*  \* 根据搜索条件查找用户列表  \** ***@param word*** *\** ***@return*** *List*<*User*>  *\*/* **public** List<User> getUserListByWord(String word)**throws** Exception {  **return userDao**.getUserListByWord(word);  } } |

#### 编写Dao

|  |
| --- |
| **package** com.itheima.dao;  **import** com.itheima.model.User; **import** com.itheima.util.JdbcUtils; **import** org.springframework.dao.DataAccessException; **import** org.springframework.dao.EmptyResultDataAccessException; **import** org.springframework.jdbc.core.BeanPropertyRowMapper; **import** org.springframework.jdbc.core.JdbcTemplate;  **import** java.sql.SQLException; **import** java.util.List;  */\*\*  \* 用户数据访问  \*/* **public class** UserDao {  *//核心类* **private** JdbcTemplate **jdbcTemplate** = **new** JdbcTemplate(JdbcUtils.*getDataSource*());   */\*\*  \* 根据搜索条件查询用户列表  \** ***@param word*** *\** ***@return*** *List*<*User*>  *\** ***@throws*** *SQLException  \*/* **public** List<User> getUserListByWord(String word)**throws** SQLException {  **try** {  String sql=**"select** *\** **from user where username like ? limit 0,3"**;  **return jdbcTemplate**.query(sql,**new** BeanPropertyRowMapper<User>(User.**class**),word+**"%"**);  } **catch** (EmptyResultDataAccessException e) {  e.printStackTrace();  **return null**;  }   } } |

#### JdbcUtils工具类

package com.itheima.util;  
​  
import java.io.IOException;  
import java.io.InputStream;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Properties;  
​  
import javax.sql.DataSource;  
​  
import com.alibaba.druid.pool.DruidDataSourceFactory;  
​  
/\*  
 1. 声明静态数据源成员变量  
 2. 创建连接池对象  
 3. 定义公有的得到数据源的方法  
 4. 定义得到连接对象的方法  
 5. 定义关闭资源的方法  
 \*/  
public class JdbcUtils {  
 // 1. 声明静态数据源成员变量  
 private static DataSource ds;  
​  
 // 2. 创建连接池对象  
 static {  
 // 加载配置文件中的数据  
 InputStream is = JdbcUtils.class.getResourceAsStream("/druid.properties");  
 Properties pp = new Properties();  
 try {  
 pp.load(is);  
 // 创建连接池，使用配置文件中的参数  
 ds = DruidDataSourceFactory.createDataSource(pp);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
​  
 // 3. 定义公有的得到数据源的方法  
 public static DataSource getDataSource() {  
 return ds;  
 }  
​  
 // 4. 定义得到连接对象的方法  
 public static Connection getConnection() throws SQLException {  
 return ds.getConnection();  
 }  
​  
 // 5.定义关闭资源的方法  
 public static void close(Connection conn, Statement stmt, ResultSet rs) {  
 if (rs != null) {  
 try {  
 rs.close();  
 } catch (SQLException e) {}  
 }  
​  
 if (stmt != null) {  
 try {  
 stmt.close();  
 } catch (SQLException e) {}  
 }  
​  
 if (conn != null) {  
 try {  
 conn.close();  
 } catch (SQLException e) {}  
 }  
 }  
​  
 // 6.重载关闭方法  
 public static void close(Connection conn, Statement stmt) {  
 close(conn, stmt, null);  
 }  
}  
​

#### 配置文件

driverClassName=com.mysql.jdbc.Driver  
url=jdbc:mysql://127.0.0.1:3306/demo  
username=root  
password=root  
initialSize=5  
maxActive=50  
maxWait=3000  
maxIdle=6  
minIdle=3

### 代码效果

