

Ajax 学习目标

- ☐ 了解Ajax的概念和作用
- ☐ 了解同步和异步的区别
- ☐ 能够使用jQuery的 `$.get()` 和 `$.post()` 发送Ajax请求
- ☐ 能够使用jQuery的 `$.ajax()` 方法发送Ajax请求
- ☐ 能够定义和解析json
- ☐ 能够使用Jackson转换json格式
- ☐ 能够使用fastjson转换json格式

1-JS的AJAX【了解】

知识点- AJAX的概述

1.目标

- 能够理解异步的概念

2.路径

- 什么是Ajax
- 什么是异步
- 为什么要学习Ajax

3.讲解

3.1 什么是AJAX

什么是 AJAX ?

AJAX = 异步 JavaScript 和 XML。

AJAX 是一种用于创建快速动态网页的技术。

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，必需重载整个网页面。

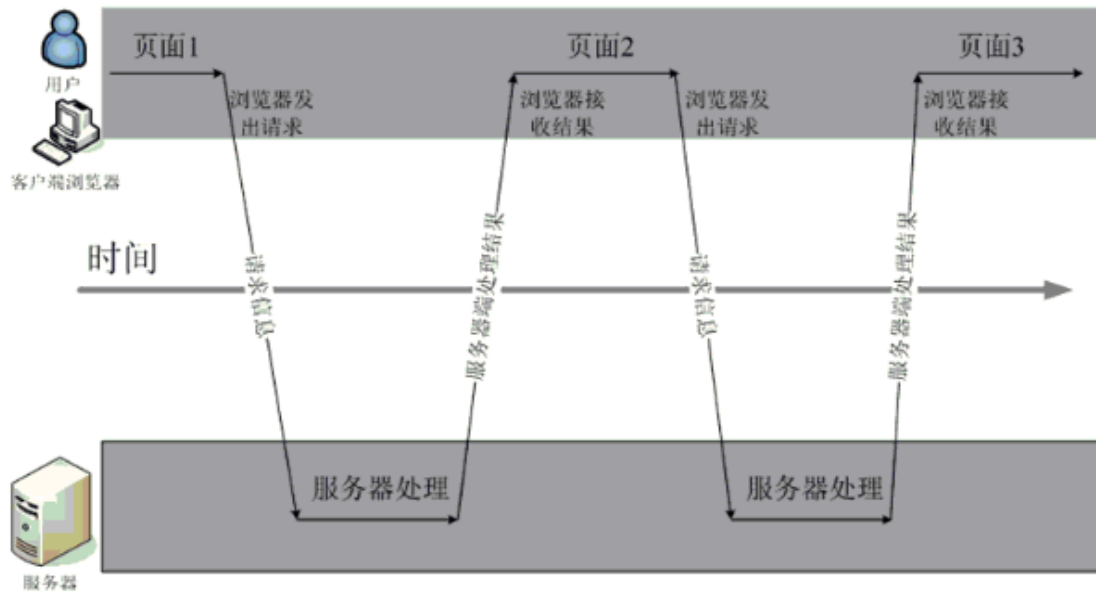
有很多使用 AJAX 的应用程序案例：新浪微博、Google 地图、开心网等等。

<https://www.w3school.com.cn/ajax/index.asp>

简单来说：**Ajax是一门动态网页技术，发送异步请求，实现在不重新加载整个页面的情况下，实现页面局部刷新，提高用户体验。**

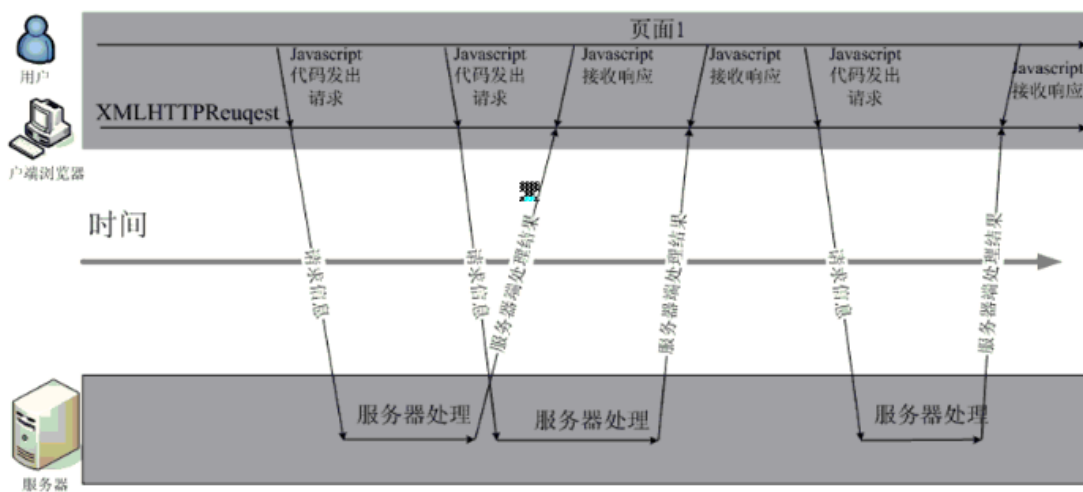
3.2什么是异步

- 同步



同步：就是事情要一件一件的做，做完上一件，才能继续做下一件事。

- 异步



异步：两个不同的对象在做事，(多线程)

使用Ajax发送异步请求，就需要额外创建一个对象 XMLHttpRequest 异步请求对象

3.3为什么要学习AJAX

实现页面局部刷新。

提升用户的体验。(异步)

4.小结

1. Ajax: Ajax是一门动态网页技术, 用于发送异步请求, 在不重新加载整个页面的情况下, 实现页面局部刷新, 提高用户体验。

知识点-JS的Ajax入门【了解】

1.目标

在网页上点击按钮, 点击按钮, 发送Ajax请求到服务器, 响应hello Ajax

2.步骤

第一步: 创建异步请求对象。

第二步: 打开连接。

第三步: 发送请求。

第四步: 设置监听异步请求对象状态改变所触发的函数, 处理结果

3.讲解

3.1GET请求方式的入门

ServletDemo01.java:

```
@WebServlet(value = "/demo01")
public class ServletDemo01 extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

        String name = request.getParameter("name");
        System.out.println("name = " + name);
        //进行后台数据响应
        response.getWriter().print("Hello Ajax");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        doPost(request, response);
    }
}
```

demo01.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <span id="spanId"></span><br>
```

```

<button id="btn01" onclick="sendAjax01()">js发送ajax请求【get】</button>
<button id="btn02" onclick="sendAjax02()">js发送ajax请求【post】</button>

<script>
    //1.js发送ajax请求【get】
    function sendAjax01() {
        //1.创建异步请求对象
        var xmlhttp;
        if (window.XMLHttpRequest)
        {
            // code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp=new XMLHttpRequest();
        }
        else
        {
            // code for IE6, IE5
            xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
        //2.打开连接 参数1: 请求方式 参数2: 请求地址 参数3: 是否发送异步请求 默认true
        //get方式传递参数 一般直接拼接在url后面
        xmlhttp.open("GET","demo01?name=zs");
        //3.发送请求
        xmlhttp.send();
        //4.监听请求对象状态 进行结果处理
        xmlhttp.onreadystatechange=function()
        {
            //readyState==4 表示异步发送对象已经接收完后台响应的数据    status==200:
            http状态码 表示请求处理成功
            if (xmlhttp.readyState==4 && xmlhttp.status==200)
            {
                //xmlhttp.responseText:表示接收后台响应的文本数据
                document.getElementById("spanId").innerText =
                xmlhttp.responseText;
            }
        }
    }
</script>
</body>
</html>

```

3.2POST请求方式的入门

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <span id="spanId"></span><br>
    <button id="btn01" onclick="sendAjax01()">js发送ajax请求【get】</button>
    <button id="btn02" onclick="sendAjax02()">js发送ajax请求【post】</button>

    <script>

        //2.js发送ajax请求【post】
        function sendAjax02() {
            //1.创建异步请求对象
            var xmlhttp;

```

```

        if (window.XMLHttpRequest)
        {
            // code for IE7+, Firefox, Chrome, Opera, Safari
            xmlhttp=new XMLHttpRequest();
        }
        else
        {
            // code for IE6, IE5
            xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
        }
        //2.打开连接  参数1: 请求方式  参数2: 请求地址  参数3: 是否发送异步请求  默认true
        xmlhttp.open("POST","demo01");
        //3.发送请求
        //post方式发送ajax请求时  需要设置请求头  并且参数一般写在send方法中  key=value
        xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
        xmlhttp.send("name=ls");
        //4.监听请求对象状态  进行结果处理
        xmlhttp.onreadystatechange=function()
        {
            //readyState==4 表示异步发送对象已经接收完后台响应的数据  status==200:
            http状态码 表示请求处理成功
            if (xmlhttp.readyState==4 && xmlhttp.status==200)
            {
                //xmlhttp.responseText:表示接收后台响应的文本数据
                document.getElementById("spanId").innerText = xmlhttp.responseText;
            }
        }
    }
</script>
</body>
</html>

```

3.3涉及到的API

3.3.1XMLHttpRequest: 异步请求发送对象

不同的浏览器对该对象的创建的方式不一样，MSIE浏览器，比较早的浏览器，创建这个对象的时候将这个对象封装到ActiveXObject的插件中。像火狐或者谷歌浏览器则直接new出来。

```

function createXmlHttp(){
    var xmlhttp;
    try{ // Firefox, Opera 8.0+, Safari
        xmlhttp=new XMLHttpRequest();
    }catch (e){
        try{// Internet Explorer
            xmlhttp=new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e){
            try{
                xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }
            catch (e){}
        }
    }
    return xmlhttp;
}

```

3.3.XMLHttpRequest的对象的API

- 方法

open(): 打开连接。传递三个参数。第一个是请求方式(GET/POST), 第二个是请求路径, 第三个是否是异步的(默认就是异步,不需要这个参数)。

send([post请求的参数]): 发送请求。

setRequestHeader(): 解决POST请求参数的问题。 key和值 content-type

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

- 属性

onreadystatechange: 监听异步请求发送对象的状态的改变,需要一个函数响应它

readyState: 该属性就记录这个对象的状态。

0 (未初始化)	对象已建立, 但是尚未初始化 (尚未调用open方法)
1 (初始化)	对象已建立, 尚未调用send方法
2 (发送数据)	send方法已调用, 但是当前的状态及http头未知
3 (数据传送中)	已接收部分数据, 因为响应及http头不全, 这时通过responseBody和responseText获取部分数据会出现错误,
4 (完成)	数据接收完毕,此时可以通过通过responseBody和responseText获取完整的回应数据

status: 状态码。

responseText:获得字符串形式的响应数据(响应体)。

responseXML :获得 XML 形式的响应数据(响应体)

4.小结

1. 响应结果

```
response.getWriter().print("Hello Ajax");
```



2. js实现ajax的步骤

1. 创建异步请求发送对象
2. 打开连接
3. 发送请求
4. 监听异步请求发送对象状态变化 进行处理

3. ajax请求和普通web请求的区别

1. ajax请求发送的是异步请求 普通web请求发送的是同步请求
2. ajax请求 服务器响应的是数据, 普通web请求 服务器响应的是整个页面

4. 使用js实现ajax请求，步骤繁琐，需要注意浏览器的兼容性问题，开发效率低，所以，实际开发中，我们一般都会使用jQuery的ajax或vue的axios实现发送异步请求。

案例-使用JS的AJAX完成用户名的异步校验

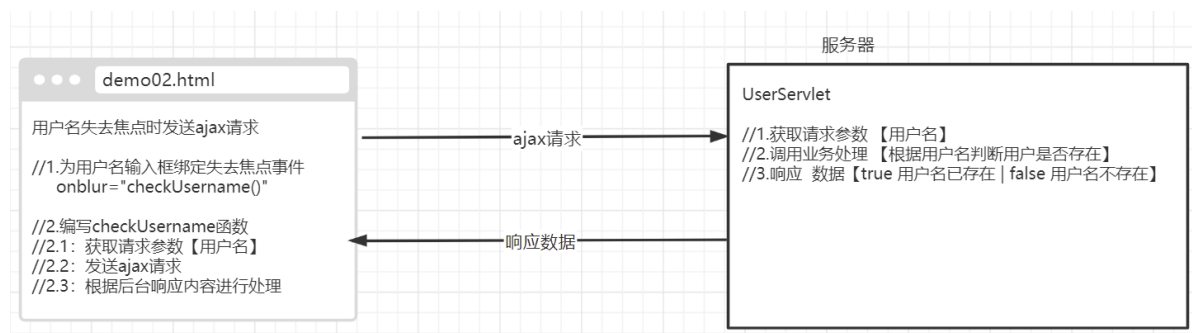
1.需求

我们有一个网站，网站中都有注册的页面，当我们在注册的页面中输入用户名的时候(失去焦点的时候)，这个时候会提示，用户名是否存在。

用户注册的页面

用户名	<input type="text" value="aaa"/>	用户名已经被占用!
密码	<input type="password"/>	
邮箱	<input type="text"/>	
电话	<input type="text"/>	
<input type="button" value="注册"/>		

2.思路分析



1. 准备数据库、创建项目、添加页面、工具类、配置文件、jar包...
2. 在页面上找到用户名输入框，为其绑定失去焦点事件 `onblur="checkUsername()"`
3. 编写checkUsername函数

```
function chekUsername(){
    //1.获取请求参数 【用户名】
    //2.发送ajax请求
    //3.根据后台响应数据 进行处理
}
```

4. 编写Servlet

```
//1.获取请求参数
//2.调用业务处理 【根据用户名查询用户】
//3.响应 true【用户名存在】|false【用户名不存在】
```

5. 编写Service

3.代码实现

3.1.环境的准备

- 创建数据库和表

```
create database day29;
use day29;
create table user(
    id int primary key auto_increment,
    username varchar(20),
    password varchar(20),
    email varchar(50),
    phone varchar(20)
);
insert into user values (null, 'aaa', '123', 'aaa@163.com', '15845612356');
insert into user values (null, 'bbb', '123', 'bbb@qq.com', '15845612356');
insert into user values (null, 'ccc', '123', 'ccc@163.com', '15845612356');
```

- 创建实体类

```
public class User implements Serializable{

    private int id;
    private String username;
    private String password;
    private String email;
    private String phone;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }
}
```



```

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    @Override
    public String toString() {
        return "User{" +
            "id=" + id +
            ", username='" + username + '\'' +
            ", password='" + password + '\'' +
            ", email='" + email + '\'' +
            ", phone='" + phone + '\'' +
            '}';
    }
}

```

- 导入jar包 (驱动,c3p0,DBUtils)
- 引入工具类(C3P0Utils),配置文件(c3p0-config.xml)
- 页面的准备

3.2代码实现

- 页面

```

<input type="text" id="username" onblur="checkUsername()" />

<script>
    function checkUsername() {
        //1.获取请求参数 用户名
        var username = document.getElementById("username").value;
        //2.发起ajax请求
        var xhr = new XMLHttpRequest();
        xhr.open("GET","user?username="+username);
        xhr.send();
        xhr.onreadystatechange = function () {
            if(xhr.readyState==4 && xhr.status==200){
                //3.根据响应数据 处理
                //获取后台响应数据 响应的是"true"或"false"
                var result = xhr.responseText;
                if(result=="true"){
                    document.getElementById("usernamespan").innerHTML = "<font
color='red'>用户名已经被占用! </font>";
                }else if (result=="false"){
                    document.getElementById("usernamespan").innerHTML = "<font
color='green'>用户名可以使用! </font>";
                }
            }
        }
    }

```

```

    }

}
</script>

```

- UserServlet

```

package com.itheima.web;

import com.itheima.bean.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(value = "/user")
public class UserServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        try {
            //1.获取请求参数 【用户名】
            String username = request.getParameter("username");
            // 2.调用业务处理 【根据用户名判断用户是否存在】
            UserService userService = new UserService();
            User user = userService.existsUser(username);
            // 3.响应 数据【true 用户名已存在 | false 用户名不存在】
            if(user==null){
                //响应给浏览器的是 "false"
                response.getWriter().print(false);
            }else{
                //响应给浏览器的是 "true"
                response.getWriter().print(true);
            }
        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().print("服务器异常!");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doPost(request, response);
    }
}

```

- UserService

```

package com.itheima.service;

import com.itheima.bean.User;
import com.itheima.dao.UserDao;

```

```
import java.sql.SQLException;

public class UserService {

    /**
     * 根据用户名判断用户是否存在
     * @param username 用户名
     * @return user对象
     */
    public User existsUser(String username) throws SQLException {
        //1.调用dao
        UserDao userDao = new UserDao();
        return userDao.findUserByName(username);
    }
}
```

- UserDao

```
package com.itheima.dao;

import com.itheima.bean.User;
import com.itheima.utils.C3P0Utils;
import org.apache.commons.dbutils.QueryRunner;
import org.apache.commons.dbutils.handlers.BeanHandler;

import java.sql.SQLException;

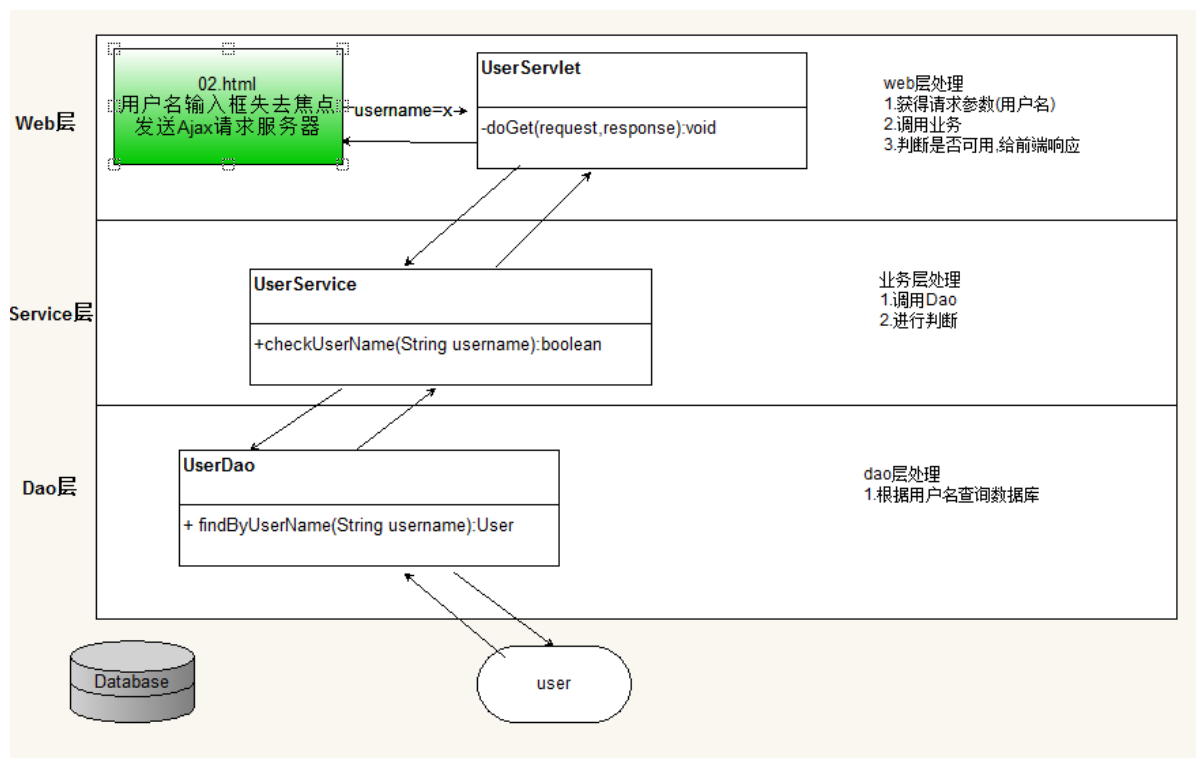
public class UserDao {

    /**
     * 根据用户名查找用户
     * @param username
     * @return
     */
    public User findUserByName(String username) throws SQLException {
        //操作数据库
        QueryRunner queryRunner = new QueryRunner(C3P0Utils.getDataSource());
        String sql = "select * from user where username=?";
        return queryRunner.query(sql, new BeanHandler<>(User.class), username);
    }
}
```

4.小结



4.1 思路



4.2注意事项

1. 根据用户名查询user表., 返回user对象
 - 如果user!=null, 证明存在
 - 如果user==null 证明不存在
2. 响应给前端的true或者false 是**字符串**

```
if (result=="true") {
```

3. 响应时使用 `response.getWriter().print(true);`

不要使用 `response.getWriter().println(true);` eg: "true\n" **"true" 或 "false\n"** "true"

平常在页面输出时使用 `println` 没啥效果, 但是在 **ajax** 请求响应时, 就不要再使用 `println` 了, 否则会在响应的内容后面添加换行空白符。

2-JQ的AJAX【重点】

知识点-JQ的AJAX介绍

1. 目标

- 知道为什么要学习JQ的AJAX原因

2.讲解

js发送ajax请求

- 1.处理浏览器兼容
- 2.步骤繁琐 需要去使用属太多

2.1 为什么要使用jQuery的AJAX

jQuery中的ajax实现对js中的ajax实现做了封装，帮助我们处理好了浏览器兼容性，以及将模板代码封装了，我们只需要简单 **调用一个方法就可以实现发送ajax请求** 了，简单方便！

2.2 jQuery的Ajax的API

url：请求地址，必选项

data：请求参数，可选项 eg: {name:value}

callback：回调函数【请求成功处理函数】，可选项 function(data){ 请求成功页面处理 }

参数data表示的是后台响应的数据 等价于 xhr.responseText

dataType：响应数据类型，可选项 默认是text，开发中常用json

请求方式	语法
GET请求	<code>\$.get(url, [data], [callback], [dataType])</code>
POST请求	<code>\$.post(url, [data], [callback], [dataType])</code>
AJAX请求	<code>\$.ajax([settings])</code>
GET请求(3.0新特性)	<code>\$.get([settings])</code>
POST请求(3.0新特性)	<code>\$.post([settings])</code>

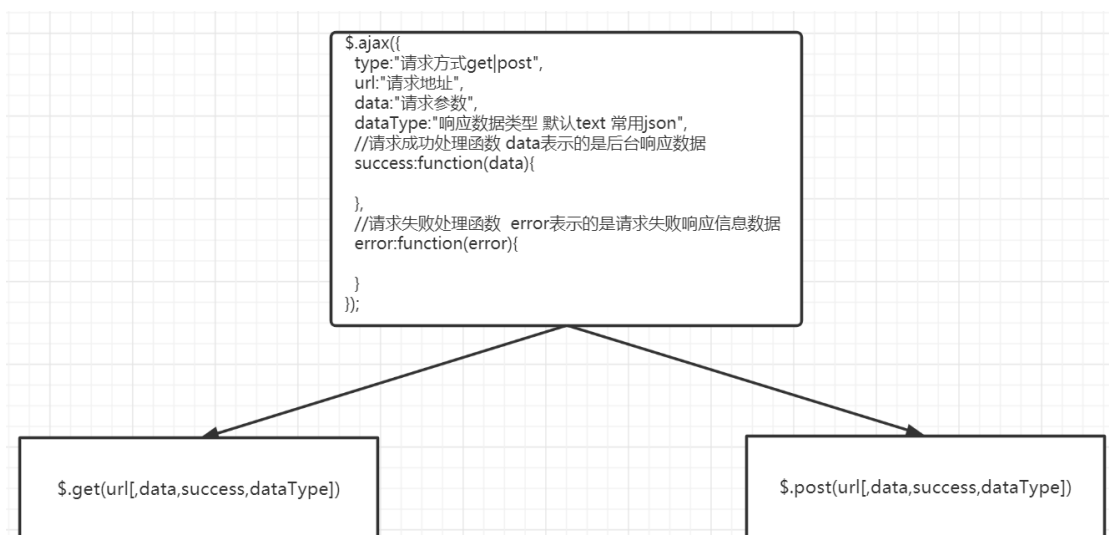
3.小结

1. 为什么要用JQ的Ajax?

js实现ajax存在弊端：①需要解决浏览器兼容性 ②步骤繁琐 代码冗余

实际开发中，一般会使用jQuery实现ajax或使用vue的axios实现ajax请求

2. 掌握的方法



知识点-JQ的AJAX入门【重点】

1.目标

在网页上点击按钮, 发送Ajax请求服务器, 响应hello ajax...

分别使用get, post, ajax三种方法

2.讲解2.1

get()

- get方式, 语法 `$.get(url, [data], [callback], [type]);`

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	当请求成功后的回调函数, 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, _default等

- 实例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="js/jquery-3.3.1.min.js"></script>
</head>
<body>
  <span id="spanId"></span><br>
  <button id="btn01" >jQuery发送ajax请求【$.get】</button>
  <button id="btn02" >jQuery发送ajax请求【$.post】</button>
  <button id="btn03" >jQuery发送ajax请求【$.ajax】</button>

  <script>
    //1. jQuery发送get方式的ajax请求
    $("#btn01").click(function () {
      //发送ajax请求 语法: $.get(请求地址[, 请求参数, 成功处理函数, 响应数据类型])
      //1.get方式请求传递参数 直接在url后面写即可
      $.get("demo01?name=zs", function (data) {
        $("#spanId").text(data);
      });
    });
  </script>
</body>
```

```
</html>
```

2.2post()

- post方式, 语法 `$.post(url, [data], [callback], [type])`

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, _default等

- 实例

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="js/jquery-3.3.1.min.js"></script>
</head>
<body>
  <span id="spanId"></span><br>
  <button id="btn01" >jQuery发送ajax请求【$.get】</button>
  <button id="btn02" >jQuery发送ajax请求【$.post】</button>
  <button id="btn03" >jQuery发送ajax请求【$.ajax】</button>

  <script>

    //2. jQuery发送post方式的ajax请求
    $("#btn02").click(function () {
      //语法: $.post(请求地址[, 请求参数, 成功处理函数, 响应数据类型]);
      //{name:"ls"} : {key1:value1, key2:value2}   js对象: 属性-值的键值对形式   也类似与
      后面的json格式数据
      //1.post方式请求参数传递   {k1:v1, k2:v2}
      $.post("demo01", {name:"ls"}, function (data) {
        $("#spanId").text(data);
      });
    });
  </script>
</body>
</html>
```

2.3ajax()

- 语法 `$.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, _default等
success	请求成功后的回调函数
error	请求失败时调用此函数

- 实例

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="js/jquery-3.3.1.min.js"></script>
</head>
<body>
  <span id="spanId"></span><br>
  <button id="btn01" >jQuery发送ajax请求【$.get】</button>
  <button id="btn02" >jQuery发送ajax请求【$.post】</button>
  <button id="btn03" >jQuery发送ajax请求【$.ajax】</button>

  <script>
    //3. jQuery使用$.ajax()方法发送ajax请求
    $("#btn03").click(function () {
      //发送ajax请求
      $.ajax({
        type: "GET",
        url: "demo02",
        success: function (data) {
          $("#spanId").text(data);
        },
        error: function (err) {
          console.log(err);
          alert("服务器异常! ")
        }
      });
    });
  </script>
</body>
</html>

```


3.小结

- 1. \$.get|post(请求地址url,请求参数data,请求成功处理函数callback,响应数据类型dataType);
- 2. \$.ajax({}): 好处是方便捕获错误处理

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ajax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, _default等
success	请求成功后的回调函数
error	请求失败时调用此函数

案例-使用JQ的Ajax完成用户名异步校验

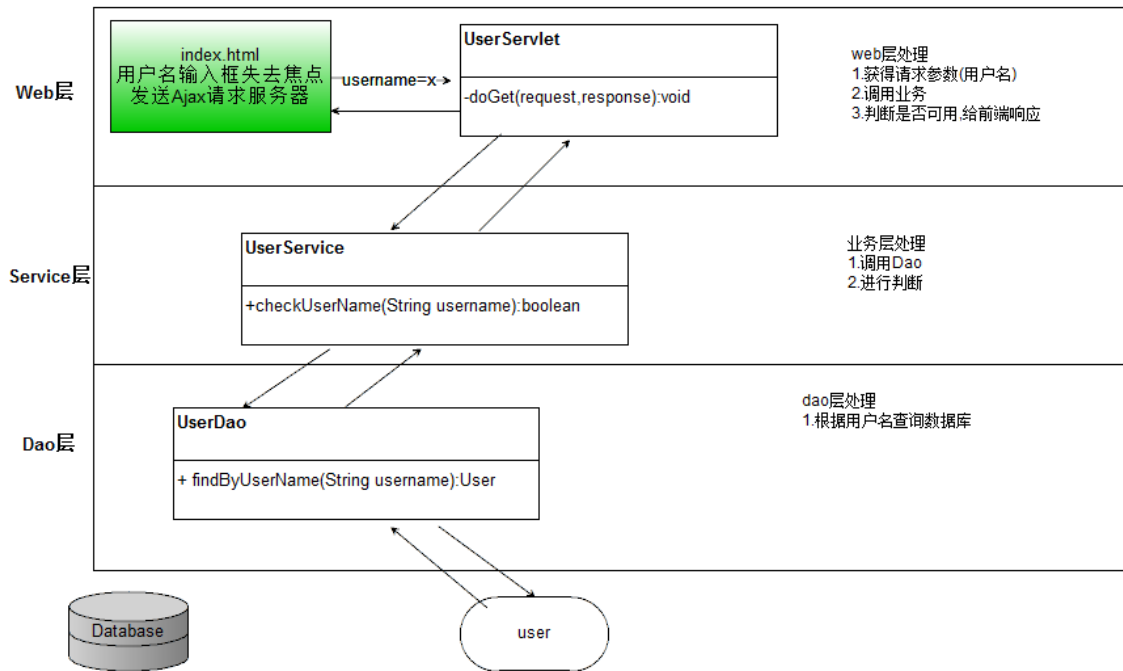
1.需求分析

我们有一个网站, 网站中都有注册的页面, 当我们在注册的页面中输入用户名的时候, 这个时候会提示, 用户名是否存在。

用户注册的页面

用户名	<input type="text" value="aaa"/>	用户名已经被占用!
密码	<input type="password"/>	
邮箱	<input type="text"/>	
电话	<input type="text"/>	
<input type="button" value="注册"/>		

2,思路分析



1. 为注册页面的用户名输入框绑定失去焦点事件, 发送ajax请求

```

//注意: 找到在什么时候发送ajax请求
$("#username").blur(function(){
    //1. 获取请求参数
    var username = $(this).val();
    //2. 发送ajax请求
    $.get("user?username="+username, function(data){
        //3. 根据响应数据 进行页面处理
    });
});

```

2. 编写Servlet

```

//1. 获取请求参数
//2. 调用业务处理 返回true|false
//3. 响应数据

```

3. 编写Service

```

boolean existsUser(String username){
    //调用dao 根据用户名查询用户 user==null: 用户不存在 user!=null: 用户存在
}

```

4. 编写dao

```

//根据用户名查询用户
User findUserByName(String username){
}

```

3.代码实现

- 前端

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="js/jquery-3.3.1.min.js"></script>
</head>
<body>
<center>
  <h1>用户注册页面</h1>
  <table border="1px" width="500px">
    <tr>
      <td>用户名:</td>
      <td><input type="text" id="username" />
        <span id="usernamespan"></span>
      </td>
    </tr>

    <tr>
      <td>密码:</td>
      <td><input type="password" />
      </td>
    </tr>

    <tr>
      <td>邮箱:</td>
      <td><input type="email" />
      </td>
    </tr>

    <tr>
      <td>电话:</td>
      <td><input type="phone" />
      </td>
    </tr>

    <tr>
      <td><input id="submitId" type="button" value="注册" /></td>
    </tr>

  </table>
</center>
<script>
  //为用户名输入框绑定失去焦点事件 发送ajax请求 验证用户名是否可用
  $("#username").blur(function () {
    //0. 获取要验证的用户名
    var username = $(this).val();
    //1.使用$.get
    /*$.get("user?username="+username,function (data) {
      if(data=="true"){
        $("#usernamespan").html("<font color='red'>用户名已经被占用!
</font>");
      }else if(data=="false"){
```

```

        $("#username span").html("<font color='green'>用户名可以使用!");
    </font>");
    }
    });*/
    //2. 使用$.post
    /*$.post("user", {username:username}, function (data) {
        if(data=="true"){
            $("#username span").html("<font color='red'>用户名已经被占用!");
        </font>");
        }else if(data=="false"){
            $("#username span").html("<font color='green'>用户名可以使用!");
        </font>");
        }
    });*/
    //3. 使用$.ajax
    $.ajax({
        type:"post",
        url:"user",
        data:{username:username},
        dataType:"text",
        success:function (data) {
            if(data=="true"){
                $("#username span").html("<font color='red'>用户名已经被占用!");
            </font>");
            }else if(data=="false"){
                $("#username span").html("<font color='green'>用户名可以使用!");
            </font>");
            }
        }
    });
});
</script>
</body>

</html>

```

4.小结

1. JQ的Ajax

```

$.get("请求地址 url" [, "请求参数 data", function(data){}, "响应数据类型 dataType"]);
$.post("请求地址 url" [, "请求参数 data", function(data){}, "响应数据类型 dataType"]);
$.ajax({
    type:"请求方式get|post",
    url:"请求地址",
    data:"请求参数 {name1:value1,name2:value2...}",
    dataType:"响应数据类型 默认text可以省略 常用json不能省略",
    success:funtion(data){

    },
    error:function(e){

    }
});

```

知识点-json的定义和解析【重点】

1.目标

- ☐ 掌握json定义和解析的语法

2.路径

1. json简介
2. 语法介绍
3. 使用示例

3.讲解

3.1.json简介

JSON

编辑

JSON(JavaScript Object Notation) 是一种轻量级的数据交换格式。它基于ECMAScript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。易于人阅读和编写，同时也易于机器解析和生成（一般用于提升网络传输速率）。

- json： **是一种轻量级的数据交换格式**。理想的数据交互语言，方便进行客户端和服务端进行数据的交换传输，易于解析和生成，结构简单，节省网络传输流量。

3.2. 语法介绍

- 定义方式：
 - 对象形式： **{key:value,key:value...}**
 - key是字符串
 - value是任意的合法数据类型
 - 多个之间使用,隔开,最后一个,不写
 - key和value之间使用:连接
 - 数组形式： **[element1, element2, ...]**
 - JSON对象数组： **[{}, {}, {} ...]**
 - 混合(嵌套)形式： 以上两种类型任意混合
 - json对象的value，可以是任意类型，当然也可以是json数组
 - 数组里的element，可以是任意类型，当然也可以是json对象
- 解析语法：
 - 获取json对象里的value值： **json对象.key**
 - 获取数组里指定索引的元素： **json数组[索引]**

3.3. 使用示例

- 对象形式

```
//1.json对象 解析: json对象.key
var user1 = {"username":"张三","age":18,"address":"深圳"};
console.log("user1对象的用户名: "+user1.username);
```

- 数组形式

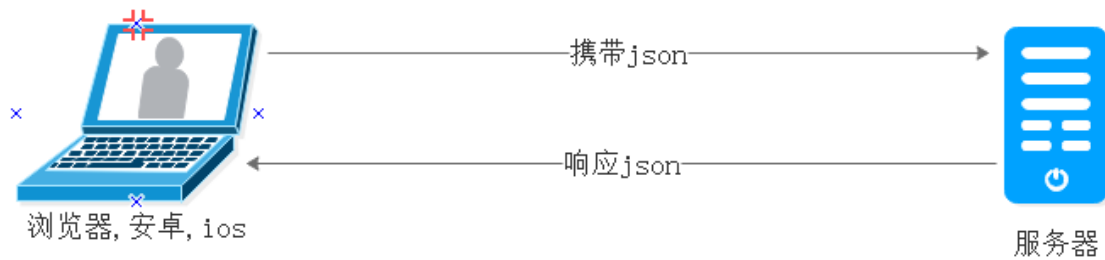
```
//2.json简单数组 解析: json数组[下标]
var citys = ["深圳","广州","东莞"];
console.log("citys数组中的第二个元素: "+citys[1])
//3.json对象数组
var users = [{"username":"张三","age":18,"address":"深圳"}, {"username":"李四","age":19,"address":"深圳"}, {"username":"王五","age":20,"address":"深圳"}];
console.log("获取json对象数组中的第三个元素: "+users[2]);
console.log("获取json对象数组中的第三个元素的username: "+users[2].username);
```

- 混合形式

```
//4.混合模式 解析: 采用剥洋葱方式 从外到内一层层解析
//统一响应数据格式: flag:true|false 请求处理是否成功 message: " " 请求处理信息 list: [user1,user2,user2] 响应的数据
var result = {flag:true,message:"查询成功",data:[{"username":"张三","age":18,"address":"深圳"}, {"username":"李四","age":19,"address":"深圳"}, {"username":"王五","age":20,"address":"深圳"}]}
//需求: 要求获取result json对象中数据里面第三个元素的姓名
console.log(result.data[2].username);
```

4.小结

1. JSON: 一种容易生成和解析的数据格式, 通常用作数据的交换(客户端和服务端之间数据的交换)



2. JSON数据格式

- json对象
 - key一定是String类型
 - value可以是任意类型
 - key和value之间使用:隔开 每一对键值直接使用,隔开
 - 解析: json对象.key
- json数组
 - 解析: json数组[下标]
- 混合形式
 - 解析: 剥洋葱 从外到内一层层解析

知识点-Jackson转换工具

1. 目标

- 能够使用转换工具Jackson，转换Java对象和json格式

2. 路径

1. 常见工具类介绍
2. Jackson的API介绍
3. Jackson的使用示例

3. 讲解

3.1. 常见工具类

- 在Ajax使用过程中，服务端返回的数据可能比较复杂，比如 `List<User>`；这些数据通常要转换成json格式，把json格式字符串返回客户端
- 常见的转换工具有：
 - **Jackson**：SpringMVC内置的转换工具 无需依赖外部jar包，直接在JDK上运行，大数据量json转换速度快，对于复杂数据类型转换没有问题
 - **fastjson**：Alibaba提供的转换工具
 - **gson**：google提供的转换工具
- 转换工具作用：可以让json对象和java对象互相转换

3.2. Jackson的API介绍

java：是一门面向对象的语言，使用java编程：找对象帮你做事

- Jackson提供了转换的核心类：**ObjectMapper**
- **ObjectMapper**的构造方法：无参构造
- **ObjectMapper**的常用方法：

方法	说明
<code>writeValueAsString(Object obj)</code>	把obj对象里的数据转换成json格式
<code>readValue(String json, Class type)</code>	把json字符串，还原成type类型的Java对象
<code>readValue(String json, TypeReference reference)</code>	把json字符串，还原成带泛型的复杂Java对象

- 其中 **TypeReference**，**com.fasterxml.jackson.core.type.TypeReference**
 - 是一个抽象类，用于配置完整的泛型映射信息，避免泛型丢失的问题。用法示例：

```
// List<Integer> 类型的映射信息
TypeReference ref1 = new TypeReference<List<Integer>>() {};
```

```
// List<User> 类型的映射信息
TypeReference ref2 = new TypeReference<List<User>>() {};
```

```
// Map<String,User> 类型的映射信息
TypeReference ref3 = new TypeReference<Map<String,User>>() {};
```

抽象类：具有构造方法，不能实例化，加{}创建对象使用的是匿名内部类。

接口：通过匿名内部类创建实例




普通类：使用protected修饰构造方法的普通类，如果不在同一个包中，也需要通过匿名内部类创建实例

3.3.Jackson使用示例

3.3.1java对象转成JSON

步骤:

1. 导入jar包

 jackson-annotations-2.2.3.jar	2018/2/20 21:45	JAR 文件	33 KB
 jackson-core-2.2.3.jar	2018/2/20 21:44	JAR 文件	189 KB
 jackson-databind-2.2.3.jar	2018/2/20 21:45	JAR 文件	846 KB

2. 创建ObjectMapper对象
3. 调用 `writeValueAsString(Object obj)`

实现

```
package com.itheima.test;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.itheima.bean.User;
import org.junit.Test;

import java.sql.ClientInfoStatus;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class JacksonTest {

    /*

    {"id":1,"username":"zs","password":"123456","email":"zs@qq.com","phone":"13112345678"}

    */
    //1.将java对象转成json数据
    @Test
    public void fun01() throws JsonProcessingException {
        //1.创建ObjectMapper对象
        ObjectMapper om = new ObjectMapper();
        //2.使用writeValueAsString方法
        //准备一个java对象
        User user = new User(1,"zs","123456","zs@qq.com","13112345678");
        String jsonStr = om.writeValueAsString(user);
        System.out.println("jsonStr = " + jsonStr);
    }

    /*
```



```

    [{"id":1,"username":"zs","password":"123456","email":"zs@qq.com","phone":"13112345678"},
    {"id":2,"username":"ls","password":"123456","email":"zs@qq.com","phone":"13112345678"}
    ],
    {"id":3,"username":"ww","password":"123456","email":"zs@qq.com","phone":"13112345678"}
    ]

    */
    //2.将list集合转成json数据
    @Test
    public void fun02() throws JsonProcessingException {
        //1.创建ObjectMapper对象
        ObjectMapper om = new ObjectMapper();
        //2.使用writeValueAsString方法
        //准备一个list集合
        User user1 = new User(1,"zs","123456","zs@qq.com","13112345678");
        User user2 = new User(2,"ls","123456","zs@qq.com","13112345678");
        User user3 = new User(3,"ww","123456","zs@qq.com","13112345678");
        List<User> list = new ArrayList<>();
        list.add(user1);
        list.add(user2);
        list.add(user3);

        String jsonStr = om.writeValueAsString(list);
        System.out.println("jsonStr = " + jsonStr);
    }




    //3.将map集合转为json数据
    @Test
    public void fun03() throws JsonProcessingException {
        //1.创建一个ObjectMapper对象
        ObjectMapper om= new ObjectMapper();
        //2.准备一个map集合
        Map<String,Object> map = new HashMap<>();
        map.put("flag",true);
        map.put("message","查询成功!");
        User user1 = new User(1,"zs","123456","zs@qq.com","13112345678");
        User user2 = new User(2,"ls","123456","zs@qq.com","13112345678");
        User user3 = new User(3,"ww","123456","zs@qq.com","13112345678");
        List<User> list = new ArrayList<>();
        list.add(user1);
        list.add(user2);
        list.add(user3);
        map.put("data", list);
        //3.调用writeValueAsString方法
        String jsonStr = om.writeValueAsString(map);
        System.out.println("jsonStr = " + jsonStr);
    }
}

```

3.3.2json转成Java对象

步骤:

1. 导入jar包

 jackson-annotations-2.2.3.jar	2018/2/20 21:45	JAR 文件	33 KB
 jackson-core-2.2.3.jar	2018/2/20 21:44	JAR 文件	189 KB
 jackson-databind-2.2.3.jar	2018/2/20 21:45	JAR 文件	846 KB

2. 创建ObjectMapper对象
3. 调用readValue(String json, Class type)或者readValue(String json, TypeReference reference)

实现

```
package com.itheima.test;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.itheima.bean.User;
import org.junit.Test;

import java.io.IOException;
import java.sql.ClientInfoStatus;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class JacksonTest {

    //4.将json数据转为java对象
    @Test
    public void fun04() throws IOException {
        //1.创建ObjectMapper对象
        ObjectMapper om = new ObjectMapper();
        //2.调用readValue()方法
        //准备一个json字符串
        String jsonStr = "{\n\"id\":1,\n\"username\": \"zs\",\n\"password\": \"123456\",\n\"email\": \"zs@qq.com\",\n\"phone\": \"13112345678\"}";
        User user = om.readValue(jsonStr, User.class);
        System.out.println("user = " + user);
    }

    //5.将json数据转为list集合对象
    @Test
    public void fun05() throws IOException {
        //1.创建ObjectMapper对象
        ObjectMapper om = new ObjectMapper();
        //2.调用readValue方法
        String jsonStr = "[{\n\"id\":1,\n\"username\": \"zs\",\n\"password\": \"123456\",\n\"email\": \"zs@qq.com\",\n\"phone\": \"13112345678\"},\n{\n\"id\":2,\n\"username\": \"ls\",\n\"password\": \"123456\",\n\"email\": \"zs@qq.com\",\n\"phone\": \"13112345678\"},\n{\n\"id\":3,\n\"username\": \"ww\",\n\"password\": \"123456\",\n\"email\": \"zs@qq.com\",\n\"phone\": \"13112345678\"}]";
        List list = om.readValue(jsonStr, List.class);
        System.out.println("list = " + list);
    }
}
```

```

//针对泛型丢失，可以自己指定要转换的数据类型 通过TypeReference实现
TypeReference<List<User>> reference = new TypeReference<List<User>>() {};
List<User> users = om.readValue(jsonStr, reference);
System.out.println("users = " + users);
}

//6.将json数据转为map集合
@Test
public void fun06() throws IOException {
    //1.创建ObjectMapper对象
    ObjectMapper om = new ObjectMapper();
    //2.调用readValue方法
    //准备一个json字符串
    String jsonStr = "{\"flag\":true,\"data\":[";
    [{"id":1,\"username\":\"zs\",\"password\":\"123456\",\"email\":\"zs@qq.com\",\"phone\":\"13112345678\"},
    {"id":2,\"username\":\"ls\",\"password\":\"123456\",\"email\":\"zs@qq.com\",\"phone\":\"13112345678\"},
    {"id":3,\"username\":\"ww\",\"password\":\"123456\",\"email\":\"zs@qq.com\",\"phone\":\"13112345678\"}],\"message\":\"查询成功!\"}";
    Map map = om.readValue(jsonStr, Map.class);
    System.out.println("map = " + map);
}
}

```

4. 小结

1. JackSon：json转换工具包，是SpringMVC内置的转换器，功能强大稳定
2. 使用
 1. 导入jar包
 2. 创建ObjectMapper对象
 3. **java对象-->json数据：writeValueAsString(Object obj);**
 4. **json数据-->java对象：readValue(String json,Class clazz);**

知识点-fastjson转换工具

1. 目标

- ☐ 能够使用转换工具Fastjson，转换Java对象和json格式

2. 路径

1. Fastjson的API介绍
2. Fastjson的使用示例

3.讲解

3.1. fastjson的API介绍

- fastjson提供了核心类：**JSON**
- JSON** 提供了一些常用的 **静态** 方法：

方法	说明
<code>toJSONString(Object obj)</code>	把obj对象里的数据转换成json格式
<code>parseObject(String json, Class type)</code>	把json字符串，还原成type类型的Java对象
<code>parseObject(String json, TypeReference reference)</code>	把json字符串，还原成带泛型的复杂Java对象

- 其中 **TypeReference**：`com.alibaba.fastjson.TypeReference`
 - 构造函数使用**protected**修饰，用于配置完整的泛型映射信息，避免泛型丢失的问题。用法示例：

```
// List<Integer> 类型的映射信息
TypeReference ref1 = new TypeReference<List<Integer>>() {};

// List<User> 类型的映射信息
TypeReference ref2 = new TypeReference<List<User>>() {};

// Map<String,User> 类型的映射信息
TypeReference ref3 = new TypeReference<Map<String,User>>() {};
```

3.2. fastjson的使用示例

3.2.1java对象转成json

步骤

1. 导入jar包

 fastjson-1.2.39.jar	2017/10/22 20:53	JAR 文件	517 KB
---	------------------	--------	--------

2. 调用JSON.toJSONString(Object obj);

实现

```
package com.itheima.test;

import com.alibaba.fastjson.JSON;
import com.itheima.bean.User;
import org.junit.Test;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class FastjsonTest {
```

```

    /*
    {"email":"zs@qq.com","id":1,"password":"123456","phone":"13112345678","username":"zs"}
    */
    //1.将java对象转成json数据
    @Test
    public void fun01(){
        //准备一个java对象
        User user = new User(1,"zs","123456","zs@qq.com","13112345678");
        //使用JSON.toJSONString()方法即可
        String jsonStr = JSON.toJSONString(user);
        System.out.println("jsonStr = " + jsonStr);
    }

    /*
    [{"email":"zs@qq.com","id":1,"password":"123456","phone":"13112345678","username":"zs"},
    {"email":"zs@qq.com","id":2,"password":"123456","phone":"13112345678","username":"ls"}
    ,
    {"email":"zs@qq.com","id":3,"password":"123456","phone":"13112345678","username":"ww"}
    ]
    */
    //2.将list集合转成json数据
    @Test
    public void fun02(){
        //准备一个list集合
        User user1 = new User(1,"zs","123456","zs@qq.com","13112345678");
        User user2 = new User(2,"ls","123456","zs@qq.com","13112345678");
        User user3 = new User(3,"ww","123456","zs@qq.com","13112345678");
        List<User> list = new ArrayList<>();
        list.add(user1);
        list.add(user2);
        list.add(user3);
        System.out.println("list = " + list);
        //使用JSON.toJSONString()方法即可
        String jsonStr = JSON.toJSONString(list);
        System.out.println("jsonStr = " + jsonStr);
    }

    /*
    {"flag":true,"data":
    [{"email":"zs@qq.com","id":1,"password":"123456","phone":"13112345678","username":"zs"},
    {"email":"zs@qq.com","id":2,"password":"123456","phone":"13112345678","username":"ls"}
    ,
    {"email":"zs@qq.com","id":3,"password":"123456","phone":"13112345678","username":"ww"}
    ],"message":"查询成功"}
    */
    //3.将map集合转成json数据
    @Test
    public void fun03(){
        //准备一个map集合
        Map<String,Object> map = new HashMap<>();
        map.put("flag",true);
        map.put("message","查询成功");
    }

```

```


        User user1 = new User(1, "zs", "123456", "zs@qq.com", "13112345678");
        User user2 = new User(2, "ls", "123456", "zs@qq.com", "13112345678");
        User user3 = new User(3, "ww", "123456", "zs@qq.com", "13112345678");
        List<User> list = new ArrayList<>();
        list.add(user1);
        list.add(user2);
        list.add(user3);
        map.put("data", list);
        //使用JSON.toJSONString()方法即可
        String jsonStr = JSON.toJSONString(map);
        System.out.println("jsonStr = " + jsonStr);
    }
}

```

3.2.2 json转成java对象

步骤

1. 导入jar包

 fastjson-1.2.39.jar	2017/10/22 20:53	JAR 文件	517 KB
---	------------------	--------	--------

2. 调用JSON.parseObject(String json, Class clazz);

实现

```

package com.itheima.test;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.TypeReference;
import com.itheima.bean.User;
import org.junit.Test;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class FastjsonTest {

    //4.将json转为java对象
    @Test
    public void fun04(){
        //1.准备一个json数据
        String json = "{\n"
        + "\"email\": \"zs@qq.com\", \"id\": 1, \"password\": \"123456\", \"phone\": \"13112345678\", \"\n"
        + \"username\": \"zs\"}";

        //2.使用JSON.parseObject方法
        User user = JSON.parseObject(json, User.class);
        System.out.println("user = " + user);
    }

    //5.将json转为list集合
    @Test
    public void fun05(){
        //1.准备一个json数据

```

```

String json = "[{"email\":\"zs@qq.com\",\"id\":1,\"password\":\"123456\",\"phone\":\"13112345678\",\"username\":\"zs\"},{\"email\":\"zs@qq.com\",\"id\":2,\"password\":\"123456\",\"phone\":\"13112345678\",\"username\":\"ls\"},{\"email\":\"zs@qq.com\",\"id\":3,\"password\":\"123456\",\"phone\":\"13112345678\",\"username\":\"ww\"}]";

//2.使用JSON.parseObject方法
//Object obj = JSON.parse(json);
List list = JSON.parseObject(json, List.class);
System.out.println("list = " + list);

//不想泛型丢失 使用TypeReference自己指定要转换的类型
TypeReference<List<User>> reference = new TypeReference<List<User>>(){};
List<User> users = JSON.parseObject(json, reference);
System.out.println("users = " + users);
}

//6.将json转为map集合
@Test
public void fun06(){
    //1.准备一个json数据
    String jsonStr="{\"flag\":true,\"data\":[" +
    [{"email\":\"zs@qq.com\",\"id\":1,\"password\":\"123456\",\"phone\":\"13112345678\",\"username\":\"zs\"},
    [{"email\":\"zs@qq.com\",\"id\":2,\"password\":\"123456\",\"phone\":\"13112345678\",\"username\":\"ls\"},
    [{"email\":\"zs@qq.com\",\"id\":3,\"password\":\"123456\",\"phone\":\"13112345678\",\"username\":\"ww\"}],\"message\":\"查询成功\"]}";
    //2.使用JSON.parseObject方法
    Map map = JSON.parseObject(jsonStr, Map.class);
    System.out.println("map = " + map);
}
}

```

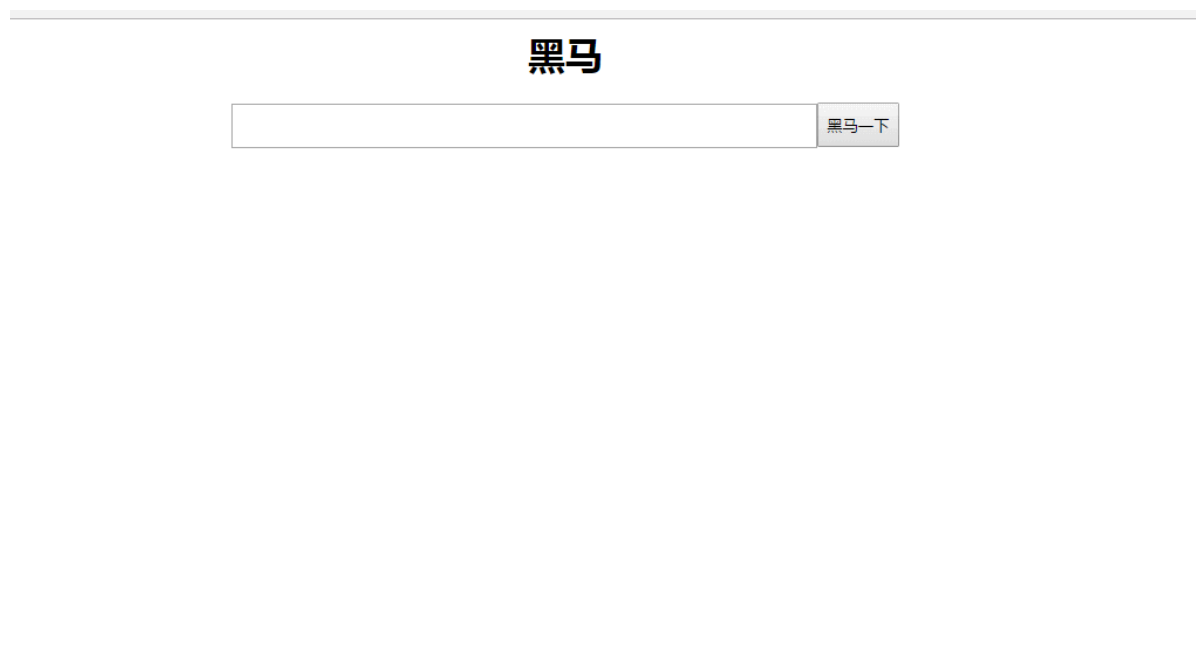
4.小结

1. fastJSON: 阿里巴巴提供的json和java对象互转工具,速度比较快.但是在进行复杂数据类型转换时偶尔有问题。
2. java对象转成json
 - `JSON.toJSONString(Object obj)`
3. json转成java对象
 - `JSON.parseObject(String json,Class clazz)`

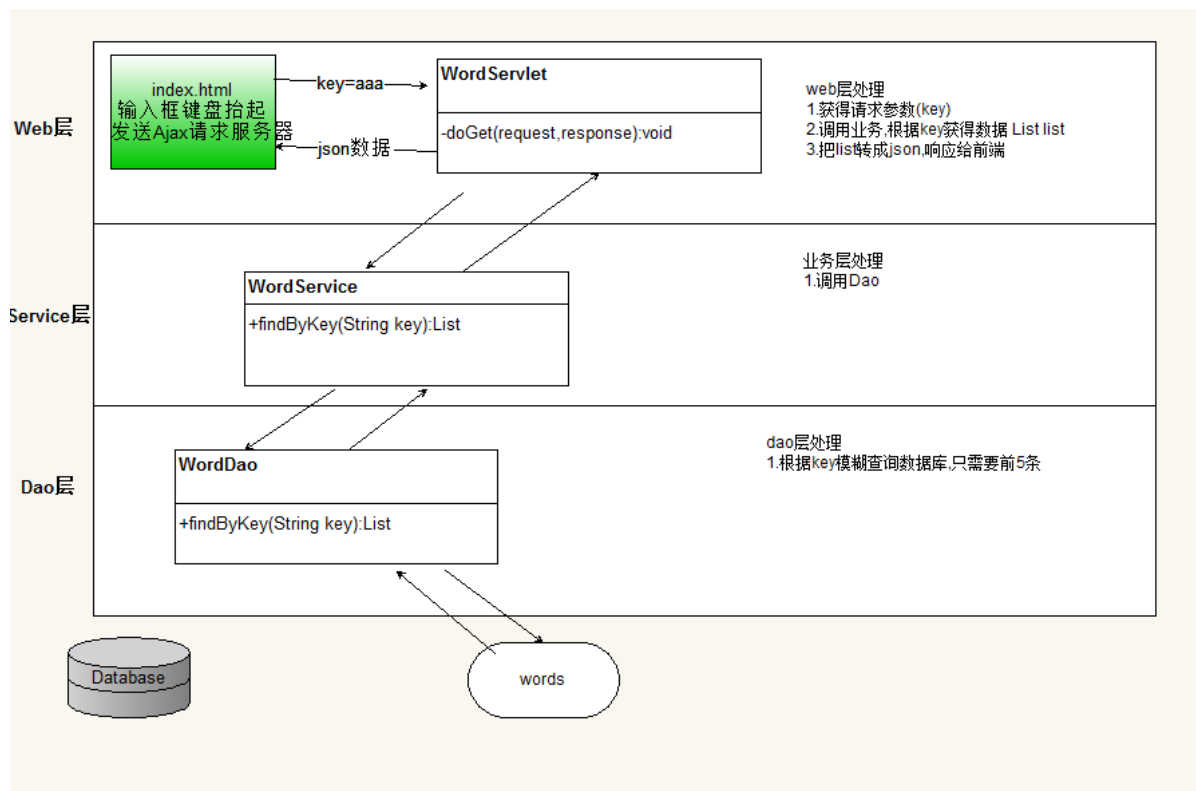
案例三：能够完成自动补全的案例(返回JSON数据)

1.需求

实现一个搜索页面，在文本框中输入一个值以后(键盘抬起的时候)，给出一些提示



2.思路分析



1. 创建数据库插入数据及准备页面
2. 找到输入框绑定键盘松开keyup事件，在键盘松开之后发送ajax请求，获取通过关键字查询到的词汇填充到提示框中，显示在页面。


```
$("#inputId").keyup(function(){
    //1.获取用户输入的内容 关键字: word
    //2.发起ajax请求
    //3.请求成功处理,将后台响应的json数据填充到提示框 显示在页面
});
```

3. 编写Servlet

```
//1.获取请求参数 word
//2.调用业务处理 返回List<Words> 5个元素
//3.将List<Words>转为json数据响应给前台
```

4. 编写Service

5. 编写dao

```
public List<Words> getWordsList(String word){
    //select * from words where word like concat('%',?,'%') limit 0,5
}
```

3.代码实现

3.1.环境的准备

- 创建数据库

```
create table words(
    id int primary key auto_increment,
    word varchar(50)
);
insert into words values (null, 'all');

insert into words values (null, 'after');

insert into words values (null, 'app');

insert into words values (null, 'apple');

insert into words values (null, 'application');

insert into words values (null, 'applet');

insert into words values (null, 'and');

insert into words values (null, 'animal');

insert into words values (null, 'back');

insert into words values (null, 'bad');

insert into words values (null, 'bag');

insert into words values (null, 'ball');

insert into words values (null, 'banana');

insert into words values (null, 'bear');
```

```
insert into words values (null, 'bike');

insert into words values (null, 'car');

insert into words values (null, 'card');

insert into words values (null, 'careful');

insert into words values (null, 'cheese');

insert into words values (null, 'come');

insert into words values (null, 'cool');

insert into words values (null, 'dance');

insert into words values (null, 'day');

insert into words values (null, 'dirty');

insert into words values (null, 'duck');

insert into words values (null, 'east');

insert into words values (null, 'egg');

insert into words values (null, 'every');

insert into words values (null, 'example');
```

- 创建JavaBean

```
package com.itheima.bean;

import java.io.Serializable;

public class Words implements Serializable{

    private int id;
    private String word;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getWord() {
        return word;
    }

    public void setWord(String word) {
        this.word = word;
    }
}
```

```

@Override
public String toString() {
    return "Words{" +
        "id=" + id +
        ", word='" + word + '\'' +
        '}';
}
}

```

- 导入jar,工具类, 配置文件
- 创建页面,demo06.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<center>

    <h1>黑马</h1>

    <input id="inputId" type="text" style="width: 500px; height: 38px;" /><input
        type="button" style="height: 38px;" value="黑马一下" />
    <div id="divId"
        style="width: 500px; border: 1px red solid; height: 300px; position:
absolute; left: 394px;display:none">
        <table id="tabId" width="100%" height="100%" border="1px">
            <tr><td>aaaa</td></tr>
            <tr><td>bbbb</td></tr>
            <tr><td>cccc</td></tr>
            <tr><td>dddd</td></tr>
            <tr><td>eeee</td></tr>
        </table>
    </div>
</center>
</body>
</html>

```

3.2实现

- demo06.html

```

<script src="js/jquery-3.3.1.min.js"></script>
<script>
    //为输入框绑定键盘松开keyup事件, 根据输入的关键字向后台发起ajax请求, 获取词汇集合的json数据, 遍
    历填充到提示框中, 显示到页面
    $("#inputId").keyup(function(){
        //1. 获取用户输入的内容 关键字: word
        var word = $(this).val();

        //优化1: 如果输入框没有内容 不需要向后台发起ajax请求了 直接跳出 $.trim(): 去除前后空格
        if(word==" "||$.trim(word)== ""){
            $("#divId").hide();

```

```

        return;
    }

    //2.发起ajax请求
    //语法: $.get(url,data,success,dataType); 由于dataType后台响应数据类型默认是text, 而
    现在这里是json, 所以需要手动设置
    $.get("WordsServlet?word="+word,function (data) {

        //优化2: 当后台没有响应数据时 就没有必要进行遍历以及提示框的显示 隐藏提示框并退出
        if(data==" "||$.trim(data)==""){
            $("#divId").hide();
            return;
        }

        //3.请求成功处理, 将后台响应的json数据填充到提示框 显示在页面
        //3.1:遍历data 后台响应的json数据 每遍历一次 就输出一行
        //声明一个变量 接收遍历的内容
        var tr = "";
        $(data).each(function(index,element){
            tr+="|<td>"+element.word+"</td></tr>";
        });
        $("#tabId").html(tr);
        //让提示框显示
        $("#divId").show();
    }, "json");

});
</script>

|  |

```

- WordsServlet

```

@WebServlet(value = "/WordsServlet")
public class WordsServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        try {
            //1.获取请求参数 word
            String word = request.getParameter("word");
            //2.调用业务处理 返回List<Words> 5个元素
            WordsService wordsService = new WordsService();
            List<Words> list = wordsService.getWordsList(word);
            System.out.println("list = " + list);
            //3.将List<Words>转为json数据响应给前台
            String json = JSON.toJSONString(list);
            response.getWriter().print(json);
        } catch (Exception e) {
            e.printStackTrace();
            response.getWriter().print("服务器异常! ");
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        doPost(request, response);
    }
}

```

```
}
```

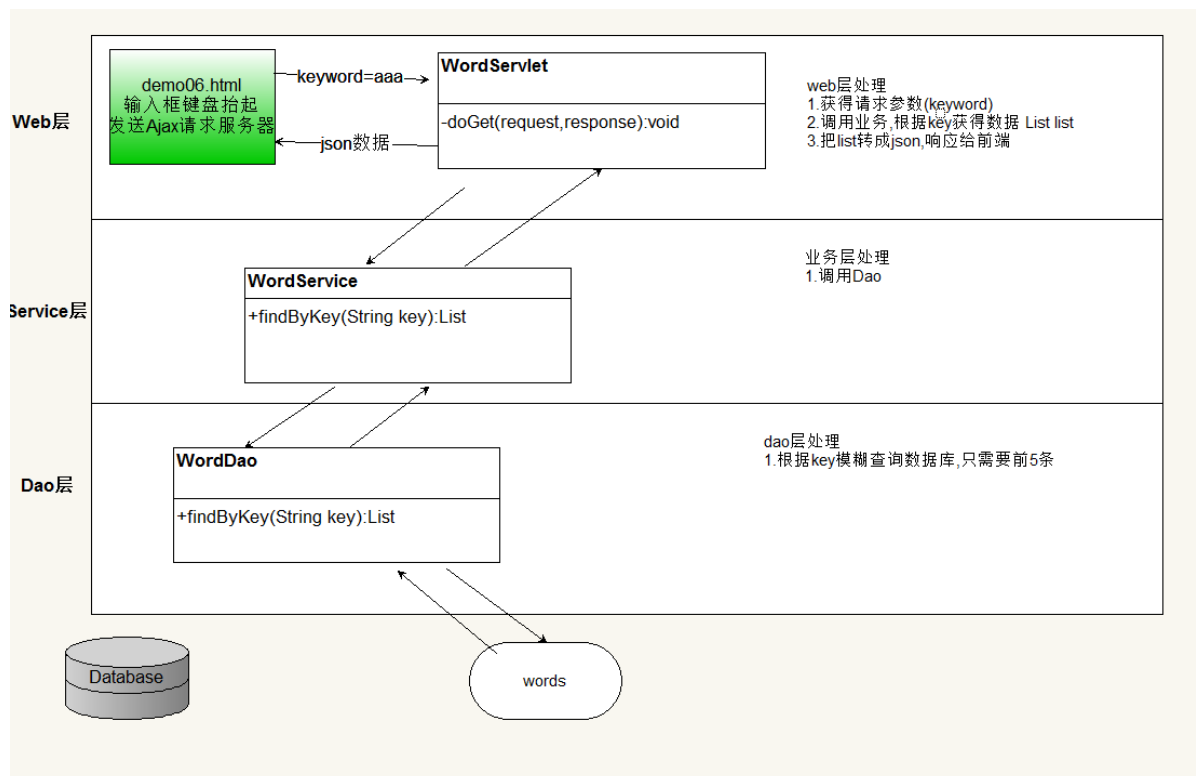
- WordsService

```
public class WordsService {  
    /**  
     * 根据关键字word获取词汇集合  
     * @param word 关键字  
     * @return 词汇集合 5个元素  
     */  
    public List<Words> getWordsList(String word) throws SQLException {  
        //1.调用dao  
        WordsDao wordsDao = new WordsDao();  
        return wordsDao.getWordsList(word);  
    }  
}
```

- WordsDao

```
public class WordsDao {  
  
    //根据关键字word获取词汇集合 只保留5个  
    public List<Words> getWordsList(String word) throws SQLException {  
        //操作数据库  
        QueryRunner queryRunner = new QueryRunner(C3P0Utils.getDataSource());  
        String sql = "select * from words where word like concat('%',?, '%') limit  
0,5";  
        return queryRunner.query(sql, new BeanListHandler<>(Words.class), word);  
    }  
}
```

4. 小结



注意:

- sql编写

```
String sql = "select * from words where word like concat('%',?, '%') limit 0,5";
```

- 细节处理
 - 自动补充div的隐藏显示
 - 内容填充

总结

1. Ajax概念：是一门动态网页技术，可以在不重载整个页面的情况下，实现局部刷新。
2. js实现Ajax 【了解】
 1. 创建XMLHttpRequest对象
 2. 打开连接 open
 3. 发送请求 send
 4. 监听异步请求对象状态变化，进行处理
3. JQ实现Ajax 【重点】
 1. `$.get("请求地址url","请求参数data",成功处理函数success,"响应数据类型dataType");`
 2. `$.post("请求地址url","请求参数data",成功处理函数success,"响应数据类型dataType");`
 3. `$.ajax({});`
4. JSON
 1. 概念：json是一个轻量级的数据交互格式，实现前后台的数据交互，结构简单，易于解析，是一个完美的数据交互格式。
 2. json定义和解析
 1. json对象：{"key":value,"key":value ...} 解析：json对象.key
 2. json对象数组：[{},{},{} ...] 解析：json数组[下标]
 3. 混合形式：剥洋葱的方式解析，从外到内一层层解析
5. JSON转换工具
 1. Jackson：SpringMVC内置，转换速度比较快，并且在进行复制数据类型转换时没有问题
 1. 导入jar包
 2. 创建ObjectMapper对象
 3. 调用方法
 1. `java对象-->json: writeValueAsString(Object obj);`
 2. `json-->java对象: readValue(String json,Class clazz);`
 2. fastjson：转换速度更快，但是在转换复杂数据类型时偶尔会出问题
 1. 导入jar包
 2. 调用方法
 1. `java对象-->json: JSON.toJSONString(Object obj);`
 2. `json-->java对象: JSON.parseObject(String json,Class clazz);`
6. 练习：
 1. 案例一【写一遍就行 js实现ajax复制就可以了】
 2. **案例二【分别使用\$.get \$.post \$.ajax实现】**
 3. 使用Jackson和fastjson完成java对象和json的转换
 4. 案例三 专业信息添加和修改的实现

