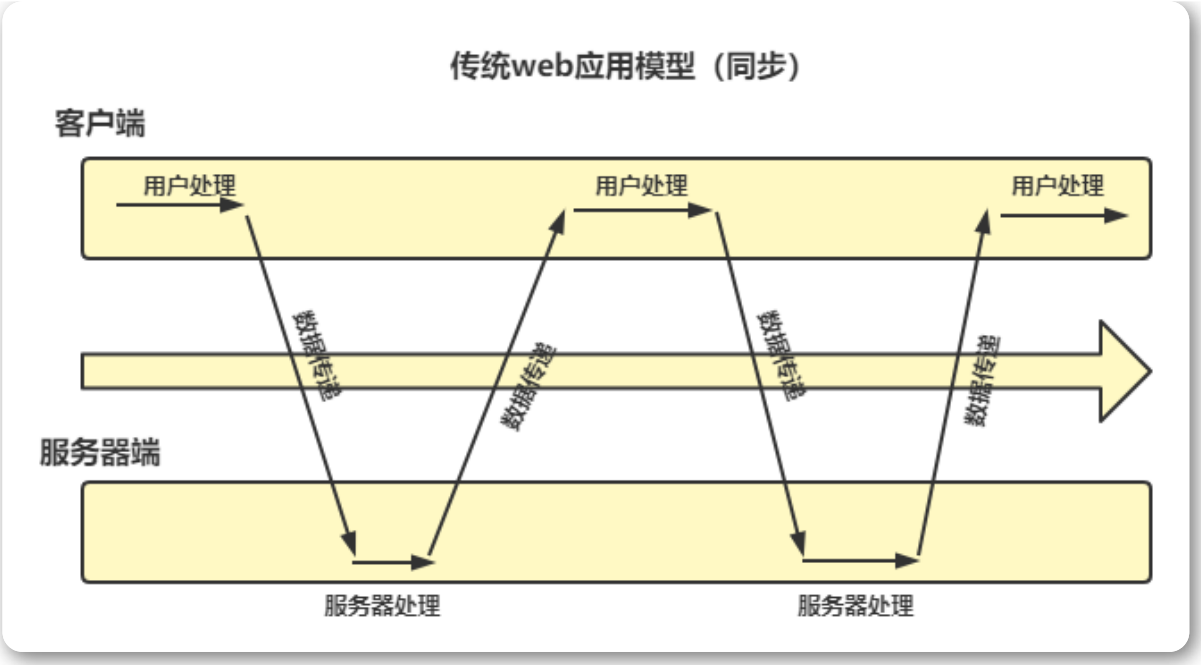
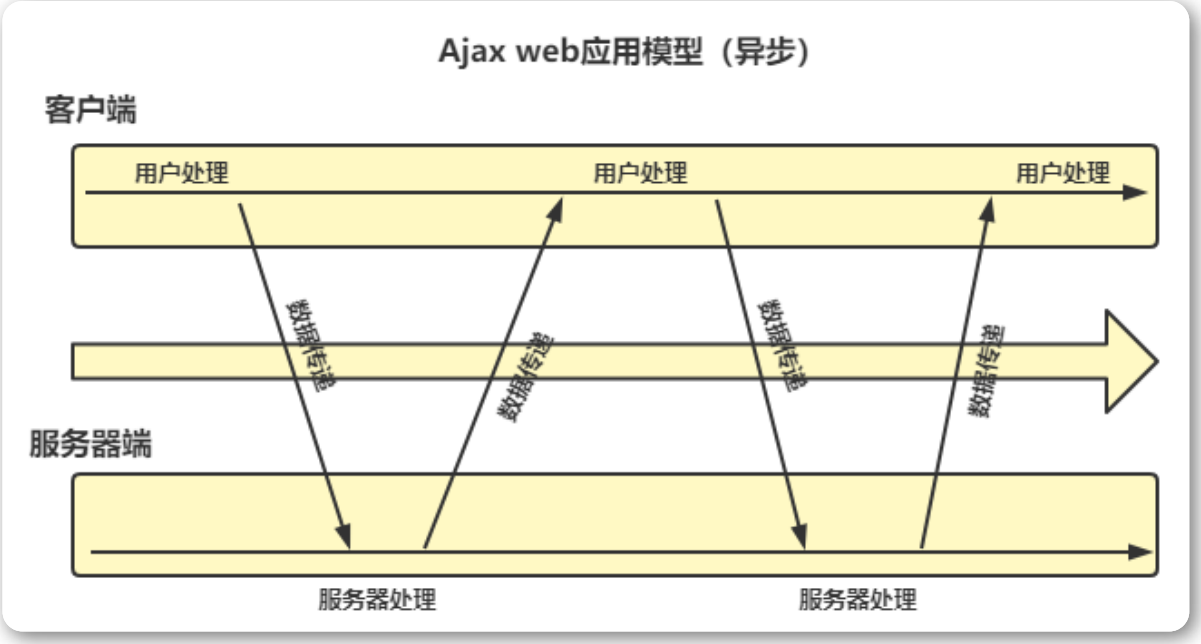
Ajax技术详解

Ajax简介

Ajax 即“Asynchronous Javascript And XML”（异步 JavaScript 和 XML），是指一种创建
交互式、快速动态应用的网页开发技术，无需重新加载整个网页的情况下，能够更新页面局
部数据的技术。通过在后台与服务器进行少量数据交换，Ajax 可以使页面实现异步更新。这意味着可以在不重新加载整个页面的情况下，对页面的某部分进行更新。





实时效果反馈

1.Ajax技术由哪些技术组成?

A xml

B javaScript

C html或xhtml

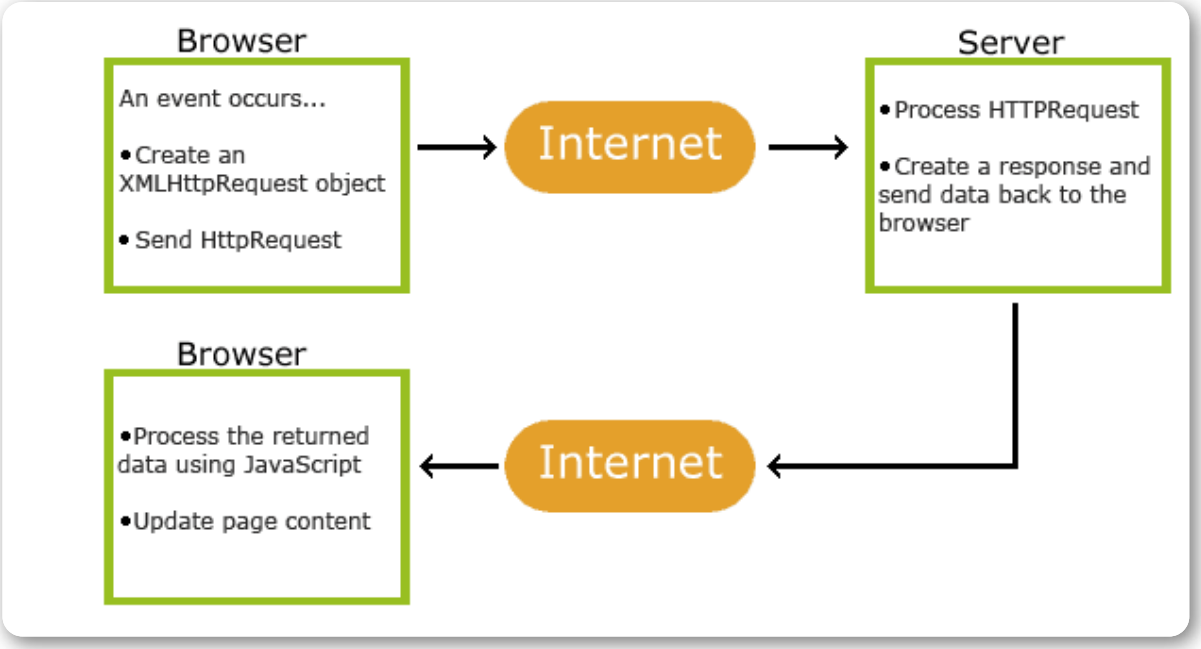
D 以上都是

答案

1=>D

Ajax 的使用

- Ajax 的运行原理



- XMLHttpRequest 对象

XMLHttpRequest 是浏览器接口对象，该对象的 API 可被 JavaScript、VBScript 以及其它web 浏览器内嵌的脚本语言调用，通过 HTTP 协议在浏览器和 web 服务器之间收发 XML 或其它数据。XMLHttpRequest 可以与服务器实现异步交互，而无需让整个页面刷新，因此成为 Ajax 编程的核心对象。

- Ajax 的使用步骤

创建 XMLHttpRequest 对象

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

给定请求方式以及请求地址

```
xhr.open("get","http://www.example.com");
```

发送请求

```
xhr.send()
```

获取服务器端给客户端的响应数据

```
xhr.onreadystatechange = function(){
    //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
    if(xhr.readyState == 4 && xhr.status == 200){
document.getElementById("span").innerHTML=xhr.responseText;
        alert(xhr.responseText);
    }
}
```

实时效果反馈

1.对XMLHttpRequest描述正确的是?

A XMLHttpRequest 是 AJAX 的基础

B XMLHttpRequest的对象用于客户端和服务器之间的异步通信

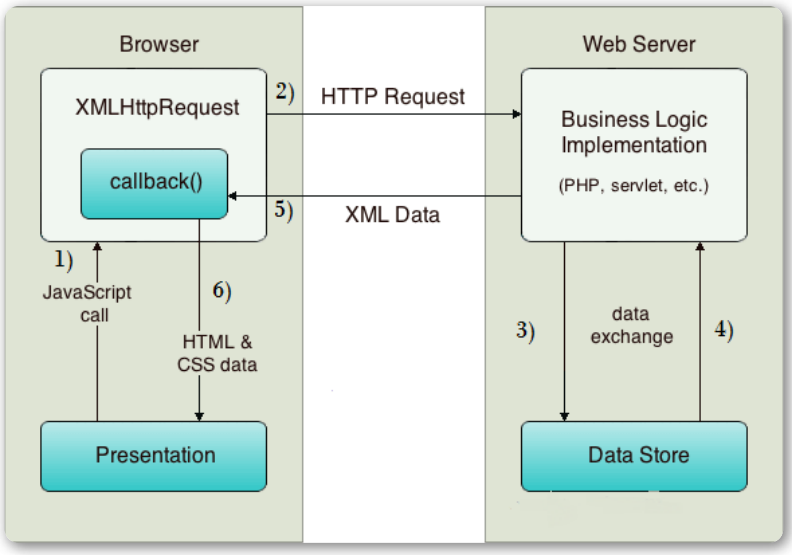
C XMLHttpRequest 术语缩写为XHR，中文可以解释为可扩展超文本传输请求

D 以上都是

答案

1=>D

Ajax请求



请求的步骤：

正如您在上面的示例中所看到的，XMLHttpRequest对象起着重要作用

1. 用户从 UI 发送请求，JavaScript 中调用 XMLHttpRequest对象。
2. HTTP请求由XMLHttpRequest 对象发送到服务器。
3. 服务器使用 JSP，PHP，Servlet，ASP.net 等与数据库交互。
4. 检索数据。
5. 服务器将 XML 数据或 JSON数据发送到 XMLHttpRequest 回调函数。

6. HTML和 CSS 数据显示在浏览器上。

方法	描述
open(*method*,*url*,*async*)	规定请求的类型、URL 以及是否异步处理请求。*method*：请求的类型；GET 或 POST*url*：文件在服务器上的位置*async*：true（异步）或 false（同步）
send(*string*)	将请求发送到服务器。*string*：仅用于 POST 请求

Get/Post请求

get 和 post请求是http协议中的两种请求方式。

- get请求一般用来请求获取数据，post请求一般作为发送数据到后台，传递数据，创建数据；
- get请求也可以传参到后台，但是传递的参数则显示在地址栏，安全性低，且参数的长度也有限制（2048字符），post请求则是将传递的参数放在request body中，不会在地址栏显示，安全性比get请求高，参数没有长度限制；
- get请求刷新浏览器或者回退没有影响，post请求则会重新请求一遍；
- get请求可以被缓存，也会保留在浏览器的历史记录中，post请求不会被缓存，也不会保留在浏览器的历史记录中；
- get请求通常是通过url地址请求，post常见的则是form表单请求

Get请求示例

```
xhr.open("GET",
"http://localhost:8080/get.txt?t=" +
Math.random(), true);
xhr.open("GET",
"http://localhost:8080/get.txt?
fname=zhang&lname=san", true);
```

Post请求示例

```
xhr.open("POST",
"http://localhost:8080/post.txt", true);
xhr.setRequestHeader("Content-
type","application/x-www-form-urlencoded");
xhr.send("fname=zhang&lname=san");
```

同步或异步

Async=true

当使用 async=true时，请规定在响应处于onreadystatechange事件中的就绪状态时执行的函数

```
xhr.onreadystatechange = function(){
if (xhr.readyState === 4 && xhr.status === 200) {

 document.getElementById("view").innerHTML = xhr.responseText;
    }
}
xmlhttp.open("GET","get.txt",true);
xmlhttp.send();
```

Async = false

我们不推荐使用 async=false，但是对于一些小型的请求，也是可以的。JavaScript 会等到服务器响应就绪才继续执行。如果服务器繁忙或缓慢，应用程序会挂起或停止。

```
xmlhttp.open("GET","get.txt",false);
xmlhttp.send();
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
```

Ajax服务器响应

状态行
xhr.status状态码，如200，304，404等；

响应主体
xhr.responseText与xhr.responseXML都表示响应主体。

如需获得来自服务器的响应，请使用 XMLHttpRequest 对象的 responseText或responseXML属性。

属性	描述
responseText	获得字符串形式的响应数据。
responseXML	获得 XML 形式的响应数据。

```
var xhr = new XMLHttpRequest();

xhr.open("GET", "http://localhost:8080/xmlTest.xml", true);
xhr.send();

xhr.onreadystatechange = function(){
if (xhr.readyState === 4 && xhr.status === 200) {
    //解析返回的xml文件
    xmlDoc = xhr.responseXML;
    txt = "";
    x = xmlDoc.getElementsByTagName("ARTIST");
    for (i=0;i<x.length;i++) {
        txt = txt + x[i].childNodes[0].nodeValue + "<br>";
    }

 document.getElementById("view").innerHTML = txt;
```

```
16     }
17 }
18
```

JSON详解

- JSON简介

JSON(JavaScript Object Notation) 是一种基于字符串的轻量级的数据交换格式。易于阅
读和编写，同时也易于机器解析和生成。JSON 是 JavaScript 数据类型的子集。

- 为什么要使用 JSON

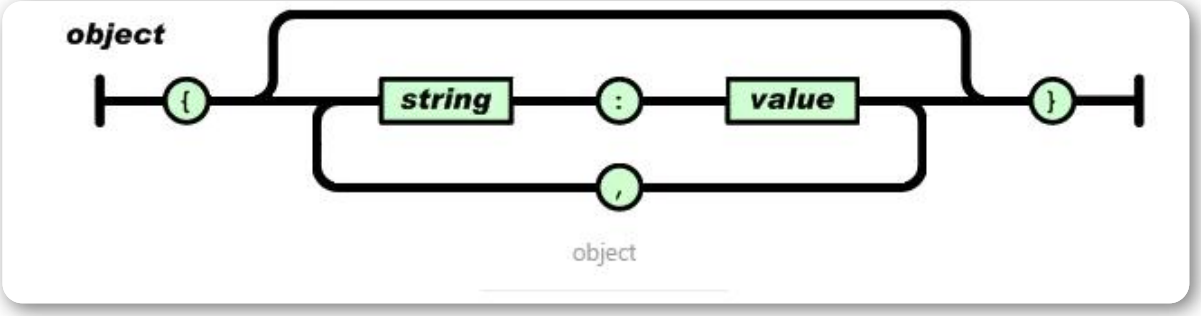
在 JSON 未出现之前在 Ajax 中对于数据传递方式，会使用 XML 作为主要数据格式来传输数据。直到 JSON 出现后逐渐放弃使用 XML 作为数据传输格式。JSON 比 XML 更小、更快、更易解析。

- JSON 格式的特征
- JSON的语法规则

JSON 是按照特定的语法规则所生成的字符串结构。

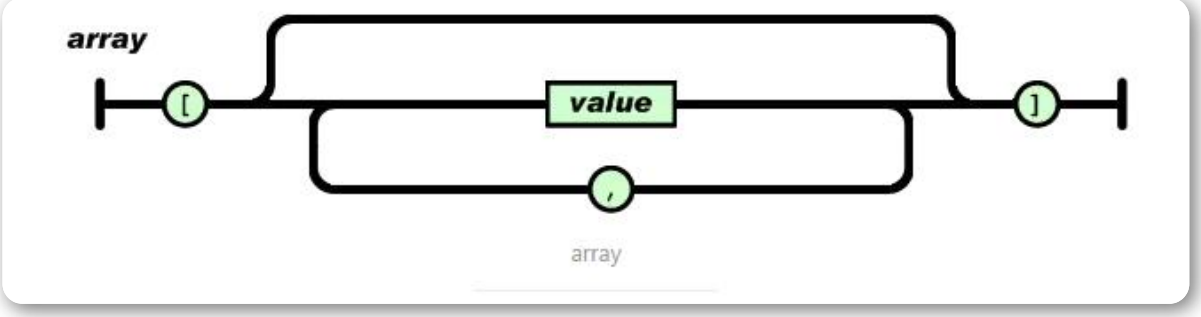
大括号表示 JSON 的字符串对象。{ }
属性和值用冒号分割。{"属性":"value"}
属性和属性之间用逗号分割。{"属性":"value","属性":"value",...}
中括号表示数组。[{"属性":"value"...},{"属性":"value"...}]

JSON 字符串对象：




```
{"userid":1,"username":"admin","sex":"male"}
```

数组：




```
[{"userid":1,"username":"admin"},
{"userid":2,"username":"zhangsan"}]
```

- JOSN的数据类型

string：字符串，必须要用双引号引起来。
number：数值，与 JavaScript 的 number 一致，
object：JavaScript 的对象形式，{ key:value }表示方式，可嵌套。
array：数组，JavaScript 的 Array 表示方式[value]，可嵌套。
true/false：布尔类型，JavaScript 的 boolean 类型。
null：空值，JavaScript 的 null。

实时效果反馈

1.下面是合法json格式的数据是?

A {"id":1,"result":"ok"}

B [{"id":1,"name":"Jack"},{"id":2,"name":"Bob"}]

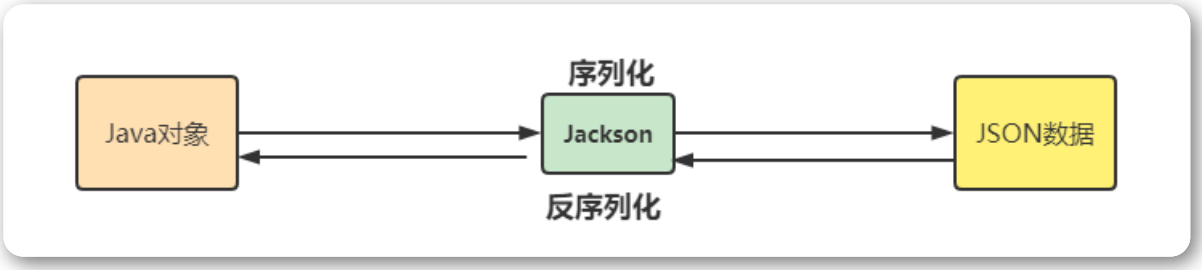
C [1,2,3,4,5,6]

D 以上都是

答案

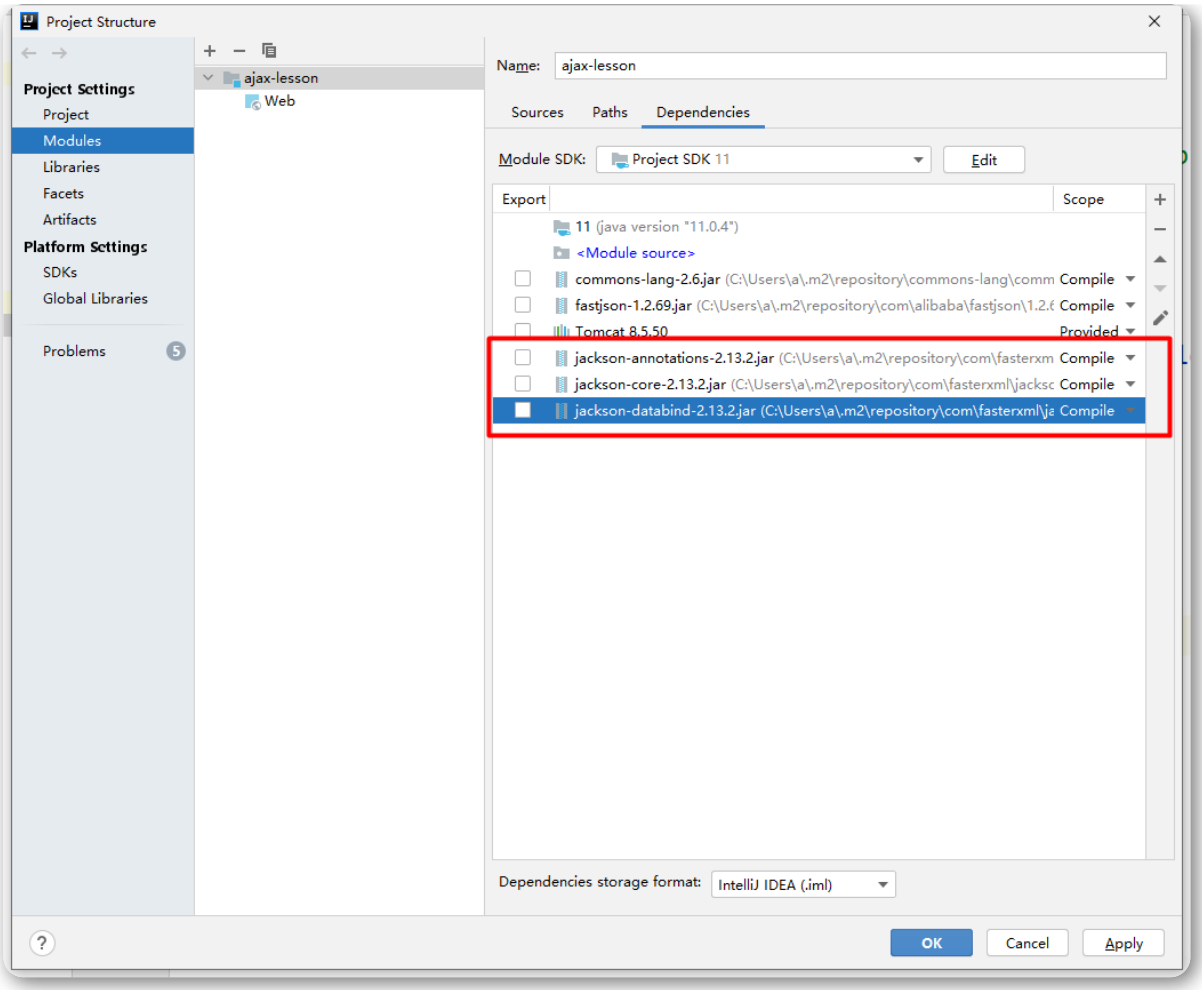
1=>D

JACKSON 的使用



在 JDK 中并没有内置操作 JSON 格式数据的 API，因此使用处理 JSON 格式的数据需要借
助第三方类库。几个常用的 JSON 解析类库：
Gson: 谷歌开发的 JSON 库，功能十分全面。
FastJson: 阿里巴巴开发的 JSON 库，性能十分优秀。
Jackson: 社区十分活跃且更新速度很快。被称为“最好的 Json 解析器”

- Jackson 简介
  Jackson 是一种解析 JSON 格式数据的 API，也是最流行，速度最快的 JSON API，最新版本是2.13.3，有3个jar包需要下载：
  jackson-core-2.13.3.jar（核心jar包）
  jackson-annotations-2.13.3.jar（提供Json注解支持）
  jackson-databind-2.13.3.jar（数据绑定，依赖于前两个包）
- 在项目中引入Jackson



- 序列化

  使用Jackson把java对象转换成Json数据。首先，创建TestBean.java

```java
public class TestBean {
    //id
    private String id;
    //姓名
    private String name;
    //嵌套对象
    private List<Element> elements;

    public String getId() {
        return id;
    }

    public void setId(String id) {
```

```
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Elements getElements() {
        return elements;
    }

    public void setElements(Elements elements) {
        this.elements = elements;
    }
}
```

再创建Element.java

```
public class Element {
    //年龄
    private Integer age;
    //呢称
    private String ename;

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
```

```
        this.age = age;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }
}
```

把Java对象转成Json

```
Element element = new Element();
element.setAge(23);
element.setEName("itbaizhan");
ObjectMapper objectMapper = new
ObjectMapper();
String elementStr =
objectMapper.writeValueAsString(element);
System.out.println(elementStr);
```

输出结果如下

```
{"age":23,"elName":"itbaizhan"}
```

- 反序列化

```
String str = "
{\"id\":1,\"name\":\"zhangsan\",\"elements
\":[{\"age\":22,\"elName\":\"xiaozhang\"},
{\"age\":26,\"elName\":\"xiaosan\"}]}";
ObjectMapper objectMapper = new
ObjectMapper();
TestBean testBean =
objectMapper.readValue(str,
TestBean.class);
System.out.println(testBean.toString());
```

输出结果如下：

```
TestBean(id=1, name=haha, elements=
[Element(age=22, elName=xiaozhang),
Element(age=26, elName=xiaosan)])
```

- 常用注解

将这个注解加载类上，不存在的字段将被忽略。

```
@JsonIgnoreProperties(ignoreUnknown =
true)
```

指定忽略字段

```
@JsonIgnoreProperties({ “password”, “secretKey” })
```

标在注解上，将忽略此字段

```
@JsonIgnore
```

标在时间字段上使用指定规则格式化（默认转化成时间戳）

```
@JsonFormat(timezone = "GMT+8", pattern = "yyyy-MM-dd")
```

是否参与序列化

```
@JsonInclude(参数)
```

JsonInclude.Include.NON_EMPTY：属性为空或者null都不参与序列化 JsonInclude.Include.NON_NULL：属性为null不参与序列化

标在字段上，指定序列化后的字段名

```
@JsonProperty("firstName")
```

自定义某些类型字段的序列化与反序列化规则

```
@JsonDeserialize(using= T extends JsonDeserializer.class)
@JsonSerialize(using= T extends JsonSerializer.class)
```

实时效果反馈

1.使用Jackson工具需要引入的包有?

A jackson-annotations.jar

B jackson-core.jar

C jackson-databind.jar

D 以上都是

答案

1=>D

Jquery 的 Ajax 使用

在 JQuery 中提供了对 Ajax 的封装，让我们在使用 Ajax 技术时变得更加容易。在 JQuery
中提供了很多的基于 Ajax 发送异步请求的方法，如：$.ajax()、$.get()、$.post()、$.getJSON()。

- $.ajax()在异步请求中提交数据

在$.ajax()方法中通过 data 属性来存放提交的数据，支持 JSON 格式的数据

提交普通格式数据

在 data 属性中我们可以通过两种方式来指定需要提交的数据。一种是通过
name=value&name=value 的结构。另一种是通过 JavaScript 对象来指定提交数据。无论使用哪种方式在Servlet中都是通过request.getParameter方法根据name获取value的。

通过 JavaScript 对象指定提交数据

```
data:{
    userid:100,
    username:"zhangsan"
}
```

提交 JSON 格式数据

在$.ajax()中提交 JSON 格式的数据需要使用 post 方式提交，通过 JSON.stringify()函数将
JavaScript 对象转换成 JSON 格式的字符串。在 Servlet 中通过字符输入获取提交的 JSON 格式的数据。

```
data:JSON.stringify({name:value,name:value.....})
```

在 Servlet 中通过 req.getReader().readLine()来获取提交的数据。

- $.ajax()处理响应中的 JSON 格式数据

$.ajax()方法会根据 dataType 属性中的值自动对响应的数据做类型处理。如果响应的是
一个 JSON 格式的数据，那么 dataType 的值为“JSON”,在回调函数中我们得到的直接就是
JSON 字符串转换完的 JavaScript 对象。不需要在使用
JSON.parse()做格式的转换处理。

- $.get()的使用

$.get()方法是$.ajax()方法基于 get 方式发送异步请求的简化版。

语法

$.get(url,function(result))

```
$.get(url,"name=value&name=value",function(result))
```

$.get(url,data,function(result))

```
$.get(url,
{userid:1,username:"zhangsan",...},function(result))
```

- $.post()的使用

$.post()方法是$.ajax()方法基于 post 方式发送异步请求的简化版。

语法

$.post(url,function(result))

```
$.post(url,"name=value&name=value",function(result))
```

$.post(url,data,function(result))

```
$.post(url,
{userid:1,username:"zhangsan",...},function(result))
```

- $.getJSON()的使用

$.getJSON()方法是$.ajax()方法基于get方式发送异步请求，并将响应结果中JSON格式
的字符串对象自动转换为 JavaScript 对象。在使用该方法时要求返回的数据必须是 JSON 格
式类型。$.getJSON()方法和
resp.setContentType("application/json")是一起使用的。

语法

$.getJSON(url,function(result))

```
$.getJSON(url,"name=value&name=value",function(result))
```

$.getJSON(url,data,function(result))

```
$.getJSON(url,
{userid:1,username:"zhangsan",...},function(result))
```

serialize()方法的使用

将form 表单中的数据自动拼接成 name=value&name=value 结构。

语法

```
var param = $("form").serialize();
```

param 的值为：name=value&name=value

1.ajax发送请求正确的方法是?

A $.get(url,function(result))

B $.post(url,function(result))

C $.getJSON(url,function(result))

D 以上都是

2. javaScript通过什么函数把对象数据转换为json格式?

A eval(xxx)

B JSON.Stringfy(xxx)

C JSON.parse()

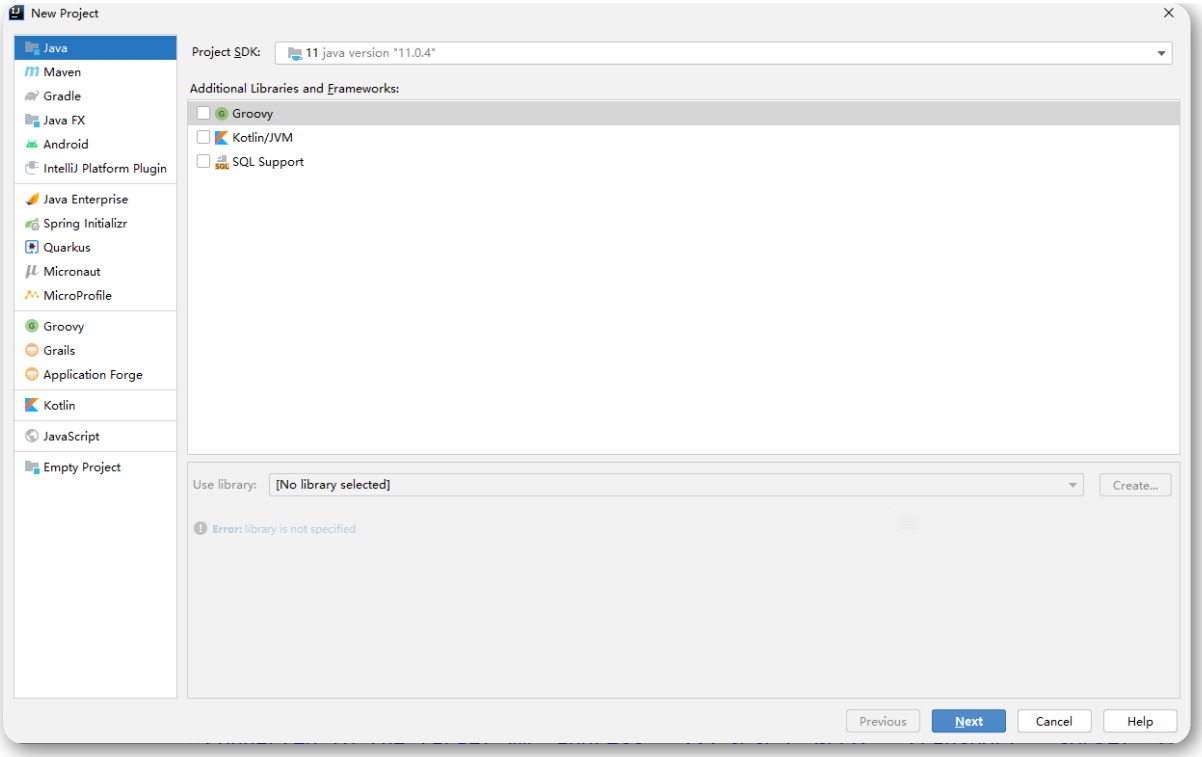
答案

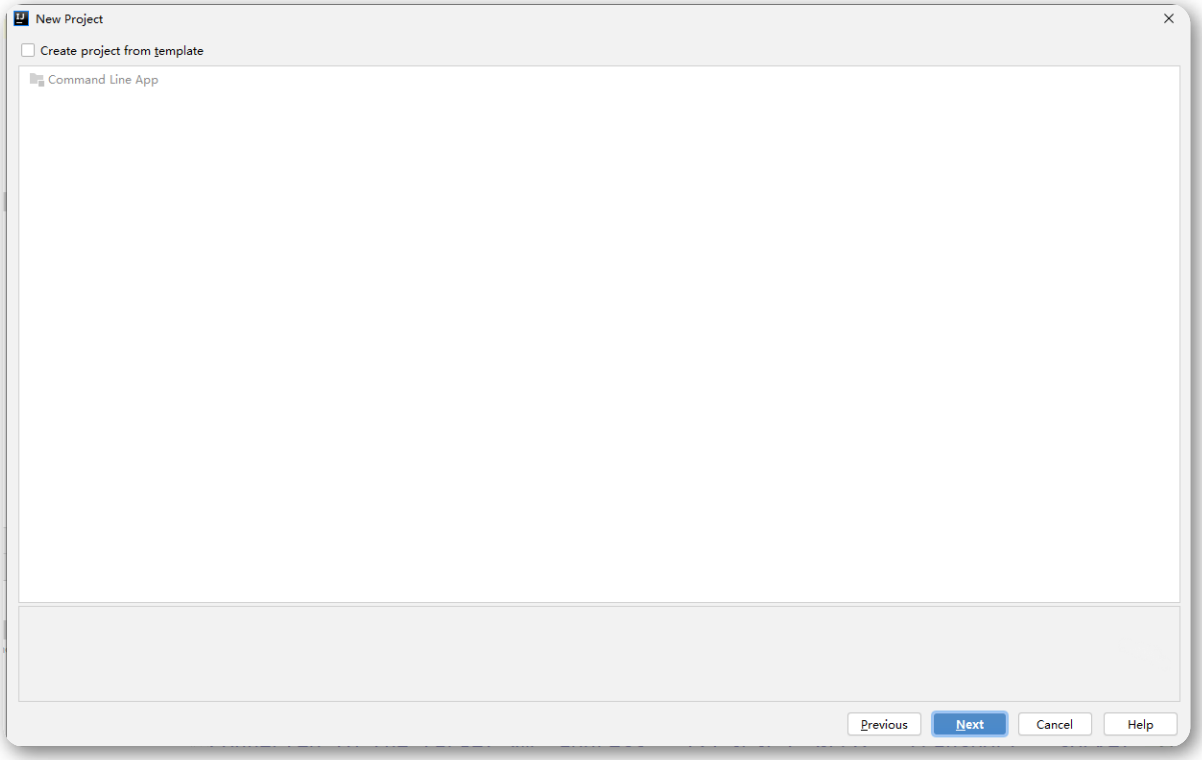
1=>D 2=>B

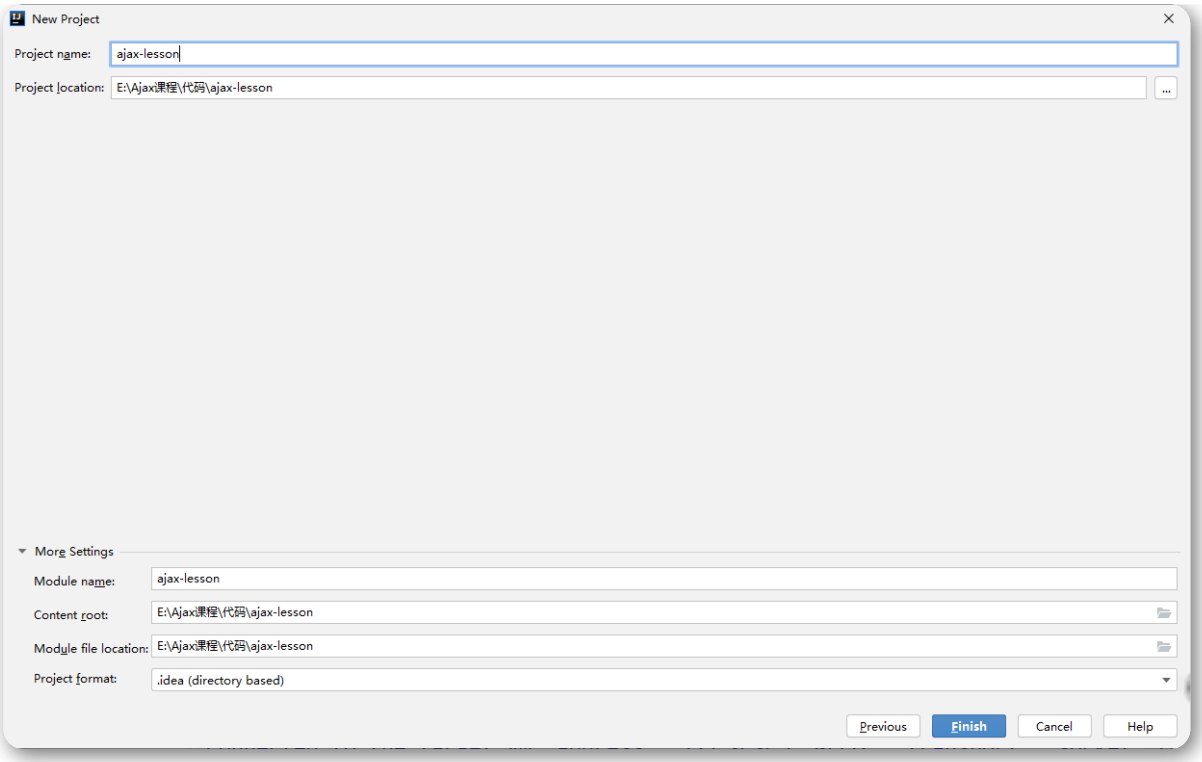
Ajax 实战

创建项目

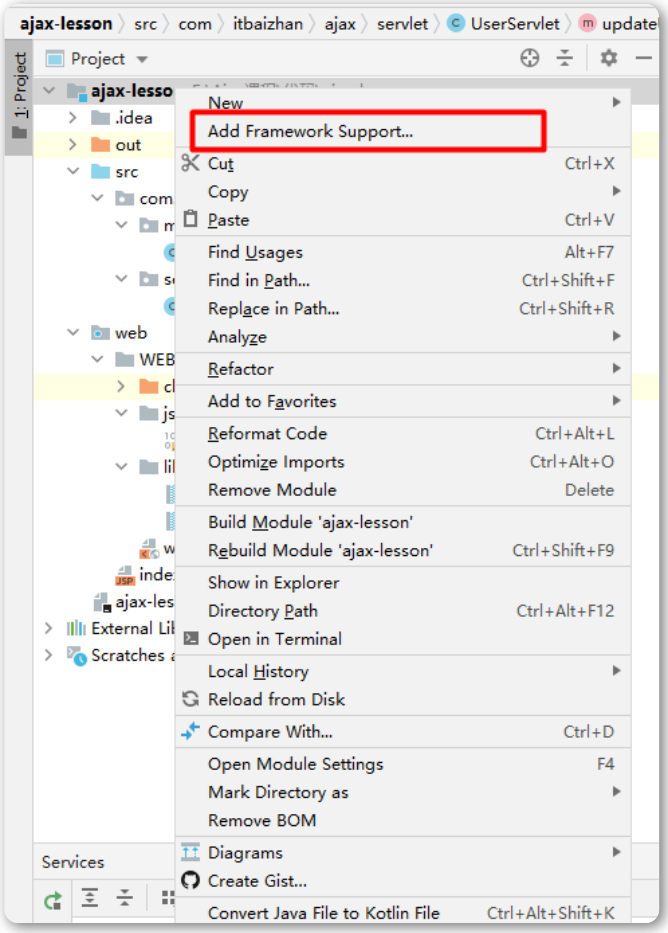
创建Java项目


New Project
Java
Maven
Gradle
Java FX
Android
IntelliJ Platform Plugin
Java Enterprise
Spring Initializr
Quarkus
Micronaut
MicroProfile
Groovy
Grails
Application Forge
Kotlin
JavaScript
Empty Project
Project SDK:
11 java version "11.0.4"
Additional Libraries and Frameworks:
Groovy
Kotlin/JVM
SQL Support
Use library:
[No library selected]
Create...
Error: library is not specified
Previous
Next
Cancel
Help

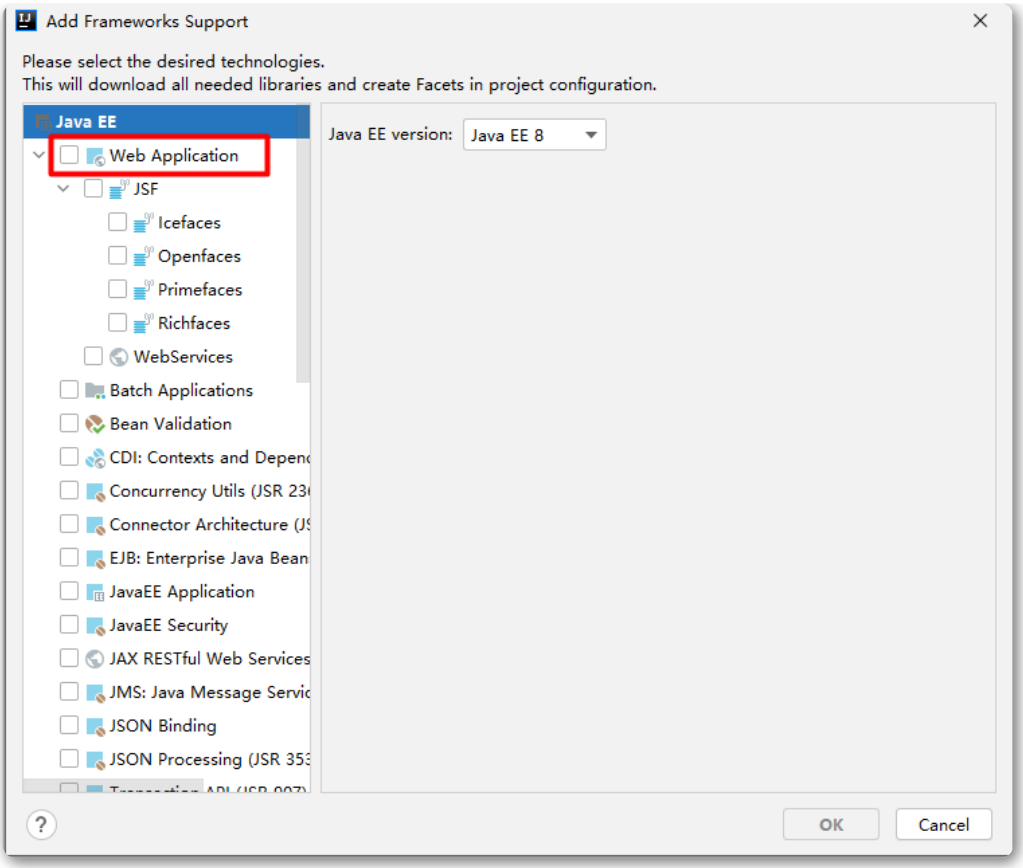

New Project
Create project from template
Command Line App
Previous
Next
Cancel
Help



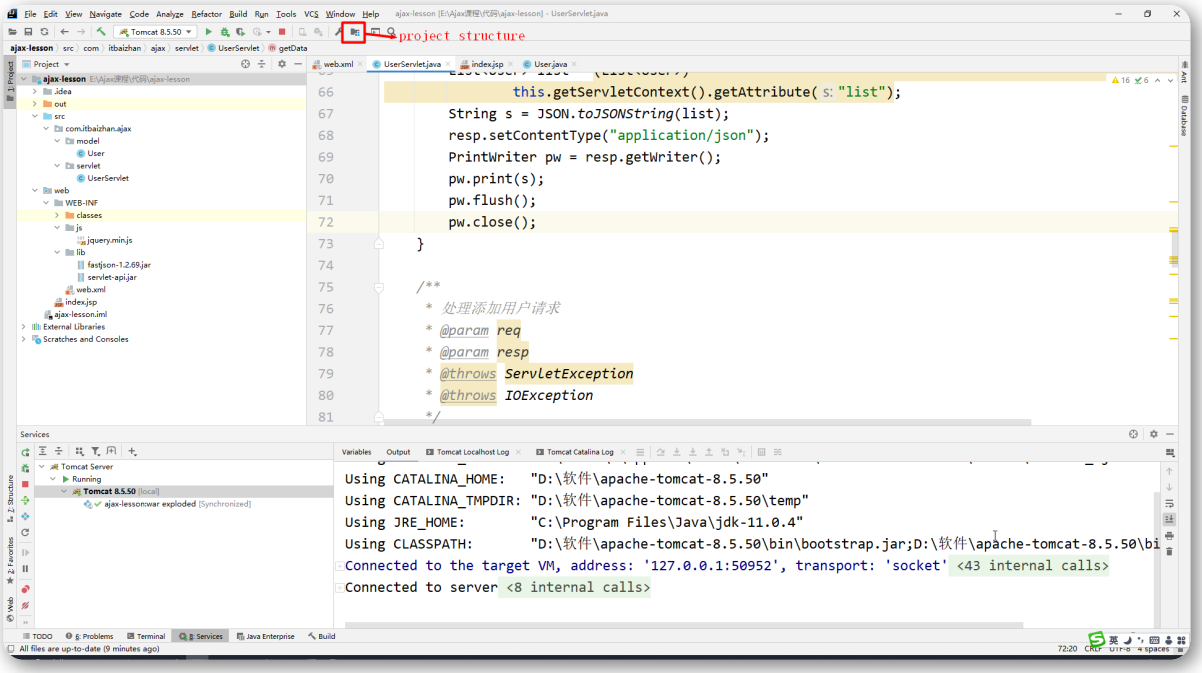
点击finish完成项目创建，然后右键选择"add Framework Support..."



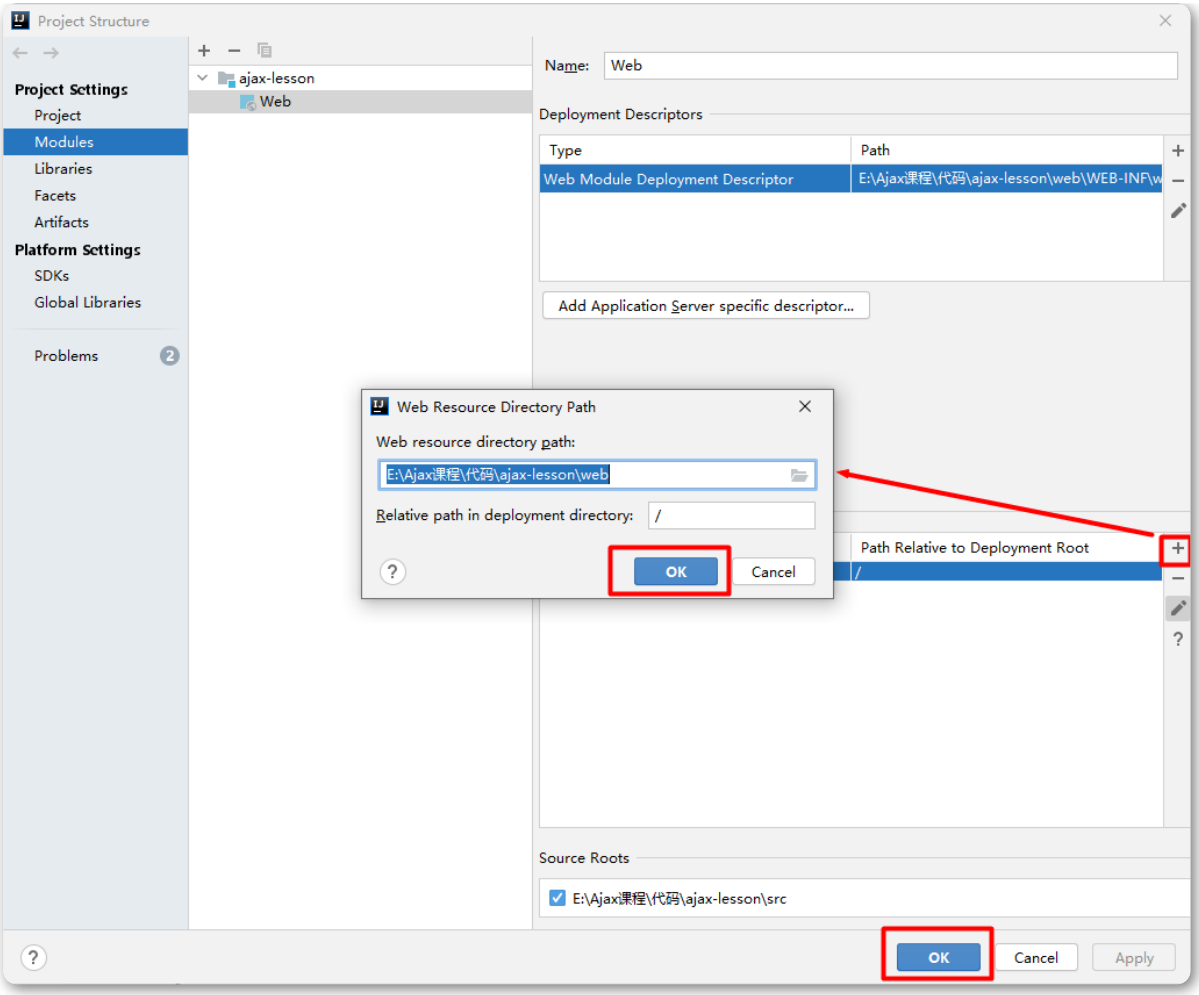
勾选红框中的选项，点击"OK"，完成后项目下出现web文件夹



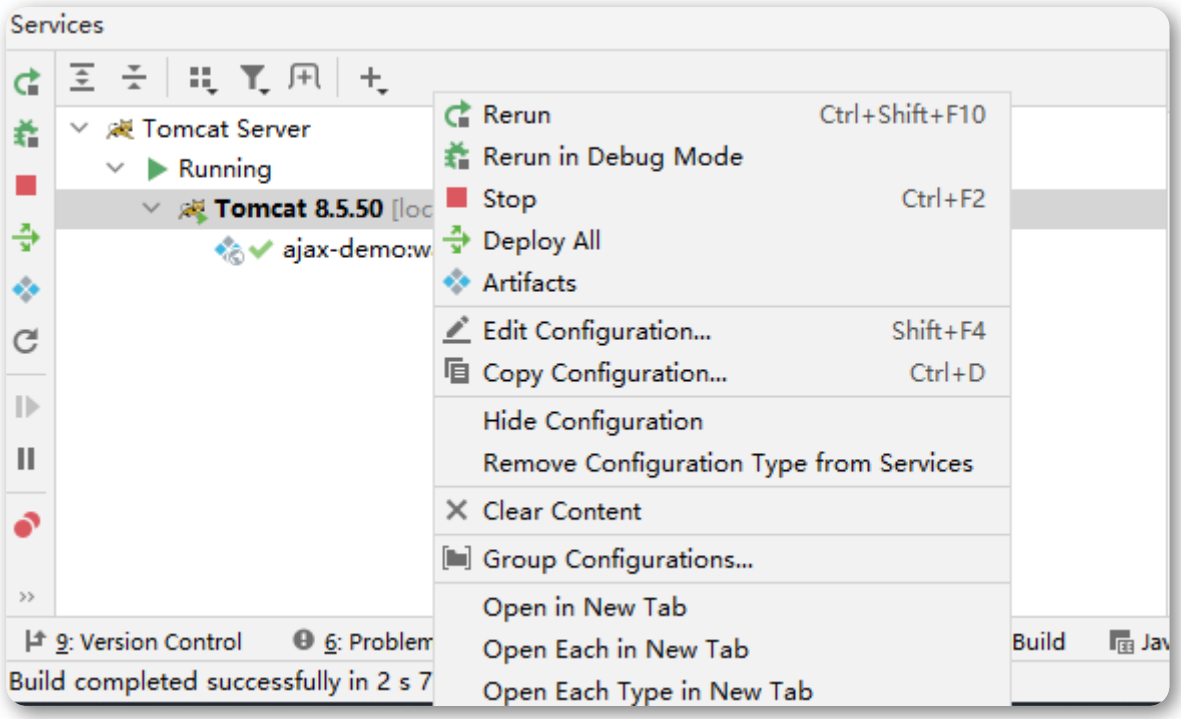
点击"project structure"



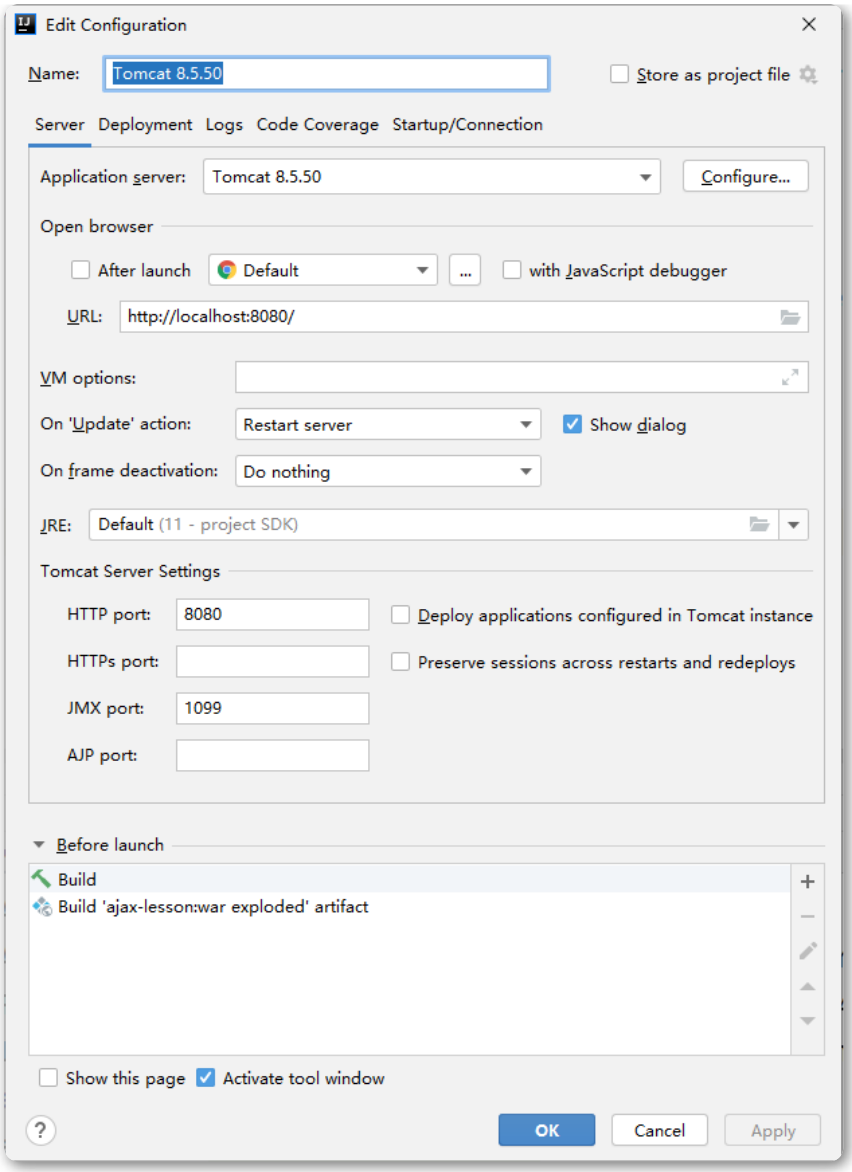
点击Modules，选择"Web"模块，再按图示点击"+"，按默认路径，再点击"OK"



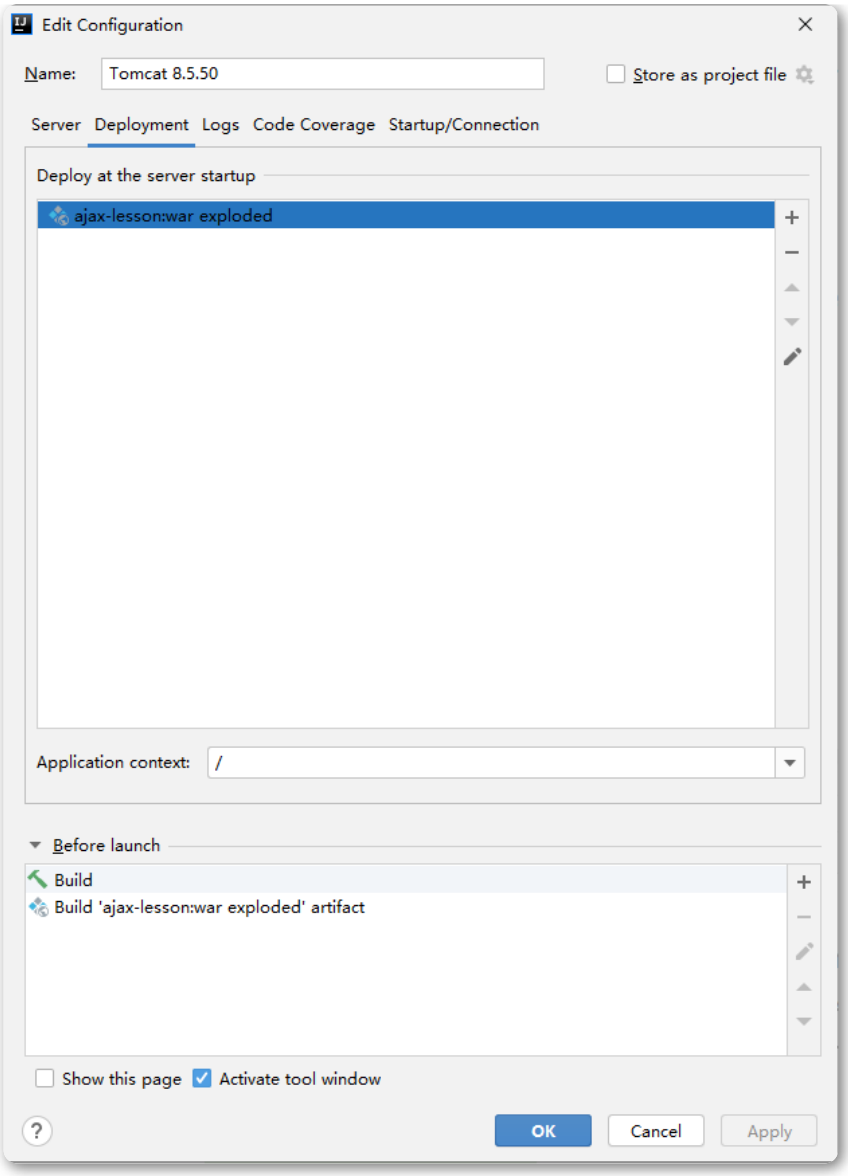
配置tomcat服务器



选择Edit Configuration...

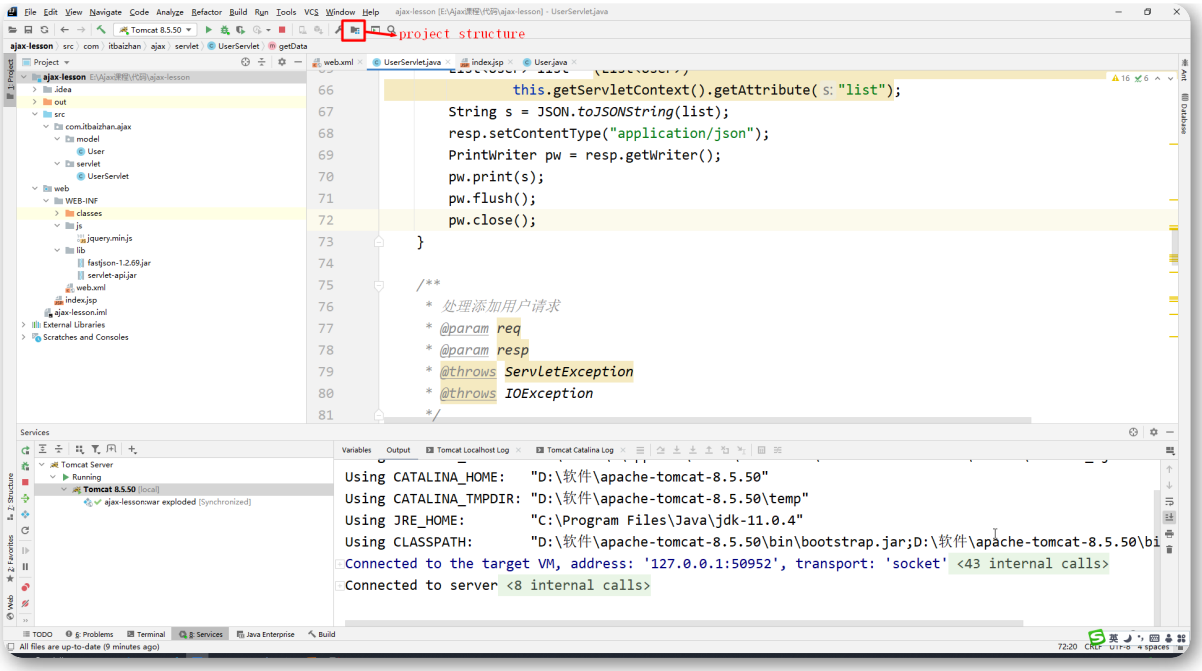


选择Deployment标签，点击右侧"+"，选择要部署的项目，按默认就可

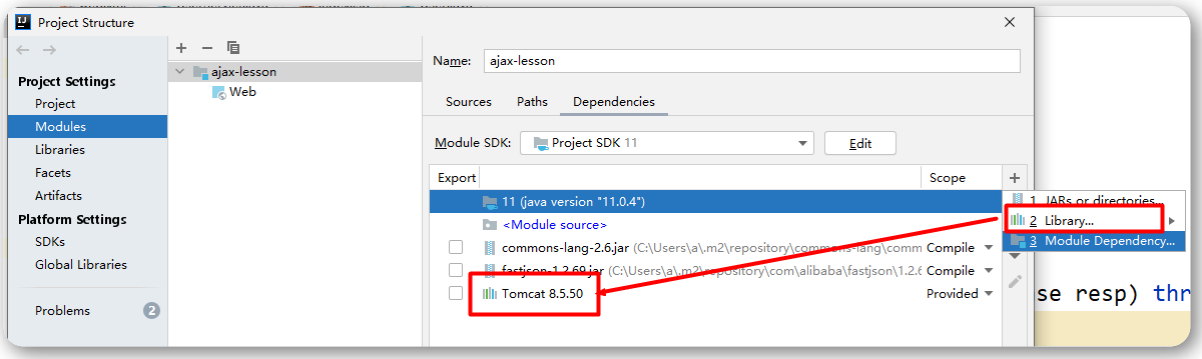


添加要部署的war文件，Application context设置为"/"，点击ok。

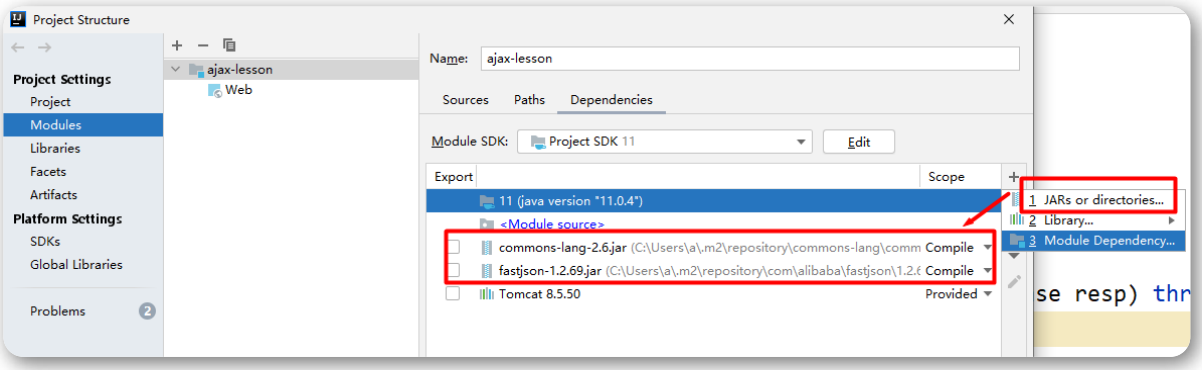
添加依赖包



在"dependencies"标签中，点击右侧"+"，选择Library...，添加Tomcat

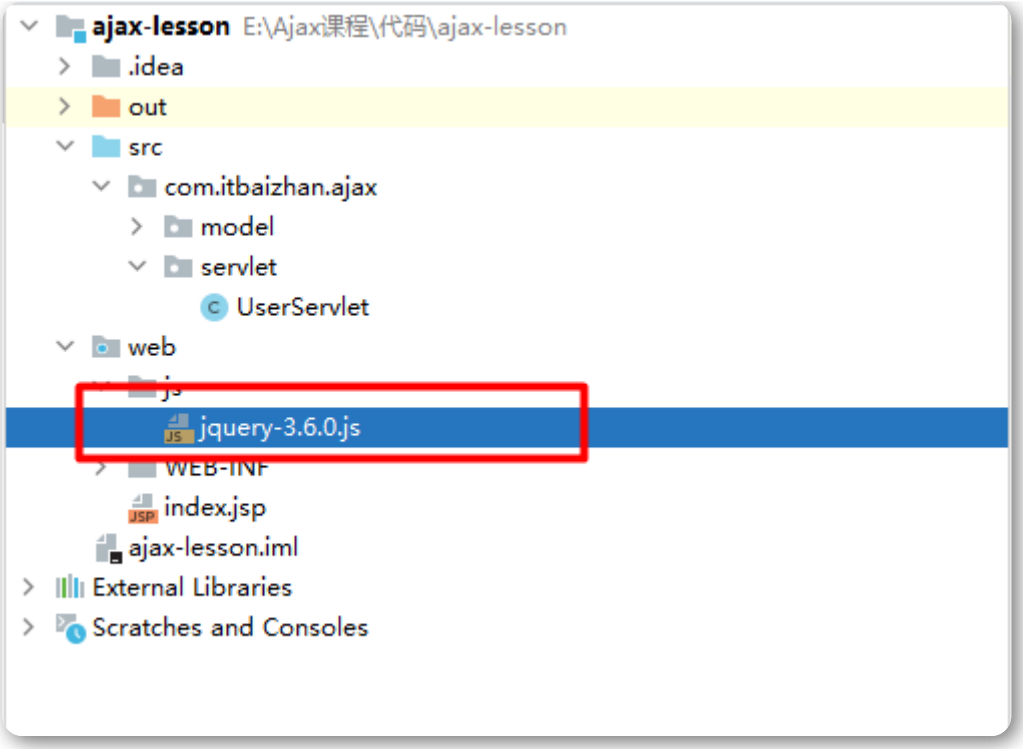


根据实际情况继续添加其它必要的依赖，如fastjson，commons-lang等。



导入jquery包

在web目录下，创建js文件夹，把jquery-3.6.0.js放到js目录下



创建实体类User

```java
package com.itbaizhan.ajax.pojo;

public class User {

    private Integer id;
    private String username;
    private String password;
    private Double salary;
    private String birthday;

    public Integer getId() {
        return id;
    }

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

    public String getUsername() {
```

```
20          return username;
21      }
22
23      public void setUsername(String username)
{
24          this.username = username;
25      }
26
27      public String getPassword() {
28          return password;
29      }
30
31      public void setPassword(String password)
{
32          this.password = password;
33      }
34
35      public Double getSalary() {
36          return salary;
37      }
38
39      public void setSalary(Double salary) {
40          this.salary = salary;
41      }
42
43      public String getBirthday() {
44          return birthday;
45      }
46
47      public void setBirthday(String birthday)
{
48          this.birthday = birthday;
```

```
    }

    public User(Integer id, String username, String password, Double salary, String birthday) {
        this.id = id;
        this.username = username;
        this.password = password;
        this.salary = salary;
        this.birthday = birthday;
    }

    public User() {
    }
}
```

创建页面

创建index.jsp中的静态部分

```
<body>
    <div>
        <table align="center" width="60%" border="1">
            <tr>
                <td>ID: </td>
                <td><input type="text" name="id" id="id"/></td>
                <td>姓名: </td>
                <td><input type="text" name="username" id="username"/></td>
                <td>密码: </td>
```

```
                <td><input type="text"
name="password" id="password"/></td>
            </tr>
            <tr>
                <td>收入: </td>
                <td><input type="text"
name="salary" id="salary"/></td>
                <td>出生日期: </td>
                <td><input type="text"
name="birthday" id="birthday"/></td>
                <td colspan="2"></td>
            </tr>
            <tr align="center">
                <td colspan="6">
                    <input type="button"
value="添加用户" id="add" />
                    <input type="button"
value="更新用户" id="update"/>
                </td>
            </tr>
        </table> <hr/>
        <table align="center" width="60%"
bgcolor="" border="1" id="myTable">
            <thead>
            <tr align="center">
                <td>ID</td>
                <td>姓名</td>
                <td>密码</td>
                <td>收入</td>
                <td>生日</td>
                <td>操作</td>
            </tr>
```

```
            </thead>
            <tbody id="tBody"></tbody>
        </table>
    </div>
</body>
```

创建UserServlet

```
@WebServlet("/user.do")
public class UserServlet extends HttpServlet
{
    @Override
    public void init() throws
ServletException {
    }

    @Override
    protected void doGet(HttpServletRequest
req, HttpServletResponse resp) throws
ServletException, IOException {
        this.doPost(req, resp);
    }

     @Override
    protected void doPost(HttpServletRequest
req, HttpServletResponse resp) throws
ServletException, IOException {
    }
}
```

配置web.xml

```
 <servlet>
     <servlet-name>UserServlet</servlet-name>
     <servlet-class>
com.itbaizhan.ajax.servlet.UserServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>UserServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

查询用户列表

页面相关

```
 $(function () {
     //初始化用户数据
     getData();
 });

// 获取页面初始化数据
function getData(){
    $.getJSON("user.do",{flag:"getData"},function (result) {
        listUser(result);
    });
}
```

```
12
13  // 遍历数据生成数据
14 function listUser(obj){
15     var str ="";
16     $.each(obj,function(){
17         str+= "<tr align='center'>" +
18             "<td id='"+this.id+"'>"+this.id
   +"</td>"+
19             "<td>"+this.username+"</td>" +
20             "<td>"+this.password+"</td>" +
21             "<td>"+this.salary+"</td>" +
22             "<td>"+this.birthday+"</td>" +
23             "<td><a href='#'
   onclick='preUpdateUser("+this.id+")'>更新
   </a>  <a href='#'
   onclick='deleteUser("+this.id+")'>删除 </a>
   </td></tr>"
24     });
25     $("#tBody").prepend(str);
26 }
```

servlet相关

```
1 @Override
2 public void init() throws ServletException {
3     ArrayList<User> list = new ArrayList<>
   ();
4     User user1 = new
   User(1,"zhangsan","123",2000d,"1997-09-01");
5     User user2 = new
   User(2,"lisi","123",3000d,"1998-08-23");
6     User user3 = new
   User(3,"zhaoliu","123",2500d,"1996-05-16");
```

```
7       User user4 = new
User(4,"wangwu","123",2080d,"1995-10-12");
8       User user5 = new
User(5,"zhengsan","123",3200d,"1999-12-20");
9       User user6 = new
User(6,"liugang","123",4200d,"1994-04-10");
10      list.add(user1);
11      list.add(user2);
12      list.add(user3);
13      list.add(user4);
14      list.add(user5);
15      list.add(user6);
16
17      ServletContext servletContext =
this.getServletContext();
18
 servletContext.setAttribute("list",list);
19  }
20
21  // 获取页面初始化数据
22  private void getData(HttpServletRequest req,
HttpServletResponse resp) throws IOException
{
23      List<User> list = (List<User>)
24
 this.getServletContext().getAttribute("list
");
25      String s = JSON.toJSONString(list);
26      resp.setContentType("application/json");
27      PrintWriter pw = resp.getWriter();
28      pw.print(s);
29      pw.flush();
```

```
30      pw.close();
31  }
32
33   @Override
34  protected void doPost(HttpServletRequest
    req, HttpServletResponse resp) throws
    ServletException, IOException {
35      String flag = req.getParameter("flag");
36      if("getData".equals(flag)){
37          this.getData(req,resp);
38      }
39  }
```

添加用户

页面相关

```
1   //添加按钮绑定点击事件
2  $("#add").click(function(){
3      addOrUpdateUser("addUser");
4  });
5
6  // 用户添加或者用户更新
7  function addOrUpdateUser(flag){
8      // 从页面中获取数据
9      var userid = $("#id").val();
10     var username = $("#username").val();
11     var password = $("#password").val();
12     var salary = $("#salary").val();
13     var birthday = $("#birthday").val();
```

```
    var data = {
        userid:userid,
        username:username,
        password:password,
        salary:salary,
        birthday:birthday,
        flag:flag
    }
    $.get("user.do",data,function(result){
        alert(result);
        location.reload();
    });
}
```

servlet相关

```
 /**
     * 处理添加用户请求
     * @param req
     * @param resp
     * @throws ServletException
     * @throws IOException
     */
private void addUser(HttpServletRequest req,
HttpServletResponse resp) throws IOException
{
    User user = this.createUser(req);
    ServletContext servletContext =
this.getServletContext();
    List<User> list = (List<User>)
servletContext.getAttribute("list");
    list.add(user);
```

```
13
 resp.setContentType("text/plain;charset=utf
-8");
14      PrintWriter pw = resp.getWriter();
15      pw.print("添加成功");
16      pw.flush();
17      pw.close();
18  }
19
20  @Override
21  protected void doPost(HttpServletRequest
    req, HttpServletResponse resp) throws
    ServletException, IOException {
22      String flag = req.getParameter("flag");
23      if("getData".equals(flag)){
24          this.getData(req,resp);
25      }else if("addUser".equals(flag)){
26          this.addUser(req,resp);
27      }
28  }
29
30  // 获取请求数据
31  private User createUser(HttpServletRequest
    req){
32      String userid =
    req.getParameter("userid");
33      String username =
    req.getParameter("username");
34      String password =
    req.getParameter("password");
35      String salary =
    req.getParameter("salary");
```

```
36      String birthday =
    req.getParameter("birthday");
37
38      User user = new User();
39      user.setId(Integer.parseInt(userid));
40      user.setUsername(username);
41      user.setPassword(password);
42      user.setSalary(Double.valueOf(salary));
43      user.setBirthday(birthday);
44
45      return user;
46  }
```

更新用户

页面相关

```
1  // 更新之前的数据选择
2  function preUpdateUser(userid){
3      var arr = new Array();
    $("#"+userid).closest("tr").children().each(
    function(index,ele){
4          if(index <=4){
5              arr[index]= ele.innerText
6          }
7      });
8      $("#id").val(arr[0]);
9      $("#id").attr("readonly",true);
10     $("#username").val(arr[1]);
```

```
        $("#password").val(arr[2]);
        $("#salary").val(arr[3]);
        $("#birthday").val(arr[4]);
    }

    //更新按钮绑定点击事件
    $("#update").click(function(){
        addOrUpdateUser("updateUser");
    });

    // 用户添加或者用户更新
    function addOrUpdateUser(flag){
        // 从页面中获取数据
        var userid = $("#id").val();
        var username = $("#username").val();
        var password = $("#password").val();
        var salary = $("#salary").val();
        var birthday = $("#birthday").val();
        var data = {
            userid:userid,
            username:username,
            password:password,
            salary:salary,
            birthday:birthday,
            flag:flag
        }
        $.get("user.do",data,function(result){
           alert(result);
           location.reload();
        });
    }
```

servlet相关

```
/**
     * 处理更新用户请求
     * @param req
     * @param resp
     * @throws IOException
     */
private void updateUser(HttpServletRequest req,
                        HttpServletResponse resp) throws IOException{
    User user = this.createUser(req);
    ServletContext servletContext = this.getServletContext();
    List<User> userList = (List<User>) servletContext.getAttribute("list");
    //list转map
    Map<Integer, User> userMap = userList.stream().collect(Collectors.toMap(User::getId, Function.identity()));
    //根据id获取user
    User user1 = userMap.get(user.getId());
    //删除指定的user
    userList.remove(user1);
    //添加新的user
    userList.add(user);
    //按id排序
    userList.sort(new Comparator<User>() {
        @Override
        public int compare(User o1, User o2) {
```

```
            return o1.getId() - o2.getId();
        }
    });
    //输出至页面

 resp.setContentType("text/plain;charset=utf-8");
    PrintWriter pw = resp.getWriter();
    pw.print("更新成功");
    pw.flush();
    pw.close();
}

@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    String flag = req.getParameter("flag");
    if("getData".equals(flag)){
        this.getData(req,resp);
    }else if("addUser".equals(flag)){
        this.addUser(req,resp);
    }else if("updateUser".equals(flag)){
        this.updateUser(req,resp);
    }
}
```

删除用户

页面相关

```
// 删除用户
function deleteUser(userid){
    $("#"+userid).closest("tr").remove();
    $.post("user.do",
{userid:userid,flag:"delete"},function(result
){
        alert(result)
    })
}
```

servlet相关

```
@Override
protected void doPost(HttpServletRequest
req, HttpServletResponse resp) throws
ServletException, IOException {
    String flag = req.getParameter("flag");
    if("getData".equals(flag)){
        this.getData(req,resp);
    }else if("addUser".equals(flag)){
        this.addUser(req,resp);
    }else if("updateUser".equals(flag)){
        this.updateUser(req,resp);
    }else if("delete".equals(flag)){
        this.deleteUser(req,resp);
    }
}

/**
     * 处理删除用户请求
     * @param req
     * @param resp
```

```
     * @throws ServletException
     * @throws IOException
     */
private void deleteUser(HttpServletRequest req,
                        HttpServletResponse resp) throws ServletException, IOException{
    ServletContext servletContext = this.getServletContext();
    List<User> userList = (List<User>) servletContext.getAttribute("list");
    String userid = req.getParameter("userid");
    //list转map
    Map<Integer, User> userMap = userList.stream().collect(Collectors.toMap(User::getId, Function.identity()));
    //根据id获取user
    if(StringUtils.isNotEmpty(userid)){
        User user1 = userMap.get(Integer.parseInt(userid));
        //删除指定的user
        userList.remove(user1);

 resp.setContentType("text/plain;charset=utf-8");
        PrintWriter pw = resp.getWriter();
        pw.print("删除成功");
        pw.flush();
        pw.close();
    }else{
```

```
40
    resp.setContentType("text/plain;charset=utf-8");
41             PrintWriter pw = resp.getWriter();
42             pw.print("删除失败");
43             pw.flush();
44             pw.close();
45         }
46 }
```

运行效果