Spring、SpringmMVC、MyBatis整合开发

具体思路：

SpringMVC：视图层，界面层，负责接收请求，显示处理结果
Spring：业务层，管理service，dao以及其他对象的
Mybatis：持久层，访问数据库的

用户发起请求--SpringMVC接收--Spring中的service对象逻辑处理--mybatis处理数据

SSM整合中有两个容器：
1、第一个容器是SpringMVC容器，管理的是Controller类对象的，即处理器
2、第二个是Spring容器，管理service、dao的代理对象

我们需要做的是：
把需要使用的对象交给合适的容器创建，管理。
把Controller和web开发的相关对象交给SpringMVC容器管理，这些web使用的对象是在SpringMVC的配置文件中声明的
把service，dao对象在Spring的配置文件中声明的，让Spring容器管理这些对象
而且SpringMVC容器和Spring容器之间是由关系的，SpringMVC是Spring容器的子容器，类似java中的继承，子类可以访问父类的内容。
就是说SpringMVC容器中的Controller类可以访问Spring容器中的service对象，实现在Controller中使用service对象，调用service中的方法

实现步骤：

1、使用mysqlstudy的MySQL库，数据表使用t_user
2、新建maven项目，使用webapp模板，创建web项目
3、添加依赖
SpringMVC、Spring、mybatis三个框架的依赖，Jackson依赖，MySQL驱动依赖，阿里的druid连接池依赖，jsp依赖，servlet依赖
4、写web.xml文件
1）、注册DispatcherServlet中央调度器，目的是
创建SpringMVC容器对象以及容器中的控制器对象
还有就是其本身作为一个servlet对象，接收请求
2）、注册Spring的监听器：ContextLoaderListener。
目的是创建Spring的容器对象以及该容器内部的service、dao等对象
3）、注册字符集过滤器，解决POST请求乱码的问题
5、创建各层的包，controller包，service包，dao包，实体类包
6、写SpringMVC，Spring，mybatis的配置文件
1）、SpringMVC配置文件
2）、Spring的配置文件
3）、mybatis配置文件
4）、数据库的属性配置文件
7、写代码。dao接口和mapper文件，service和实现类，controller类，实体类
8、写jsp页面

1）、创建maven项目d-SSM-04，使用webapp骨架模板，添加依赖，pom文件如下

```
<?xml version="1.0" encoding="UTF-8"?>
```

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.studymyself</groupId>
  <artifactId>d-SSM-04</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>


  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <!--servlet规范依赖-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>4.0.1</version>
      <scope>provided</scope>
    </dependency>
    <!--jsp依赖-->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>javax.servlet.jsp-api</artifactId>
      <version>2.3.1</version>
    </dependency>
    <!--SpringMVC依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>
    <!--spring的事务依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-tx</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>
    <!--事务相关的依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.2.5.RELEASE</version>
    </dependency>
    <!--Jackson依赖-->
```

```
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-core</artifactId>
      <version>2.10.2</version>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>2.10.2</version>
    </dependency>
    <!--mybatis依赖-->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.5.3</version>
    </dependency>
    <!--mybatis整合spring用的依赖-->
    <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>2.0.3</version>
    </dependency>
    <!--mysql驱动依赖-->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.19</version>
    </dependency>
    <!--阿里连接池依赖-->
    <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>druid</artifactId>
      <version>1.1.20</version>
    </dependency>
  </dependencies>

  <build>
    <!--resources标签表示使设定的目录和文件与resources目录功能一样-->
    <resources>
      <resource>
        <!--表示编译时在src/main/java目录下的-->
        <directory>src/main/java</directory>
        <!--所有.properties、.xml类型的文件可以被扫描到-->
        <includes>
          <include>**/*.properties</include>
          <include>**/*.xml</include>
        </includes>
        <!--filtering的值false表示不启用过滤器，上面include已经是过滤操作了-->
        <filtering>false</filtering>
      </resource>
    </resources>
  </build>
</project>
```

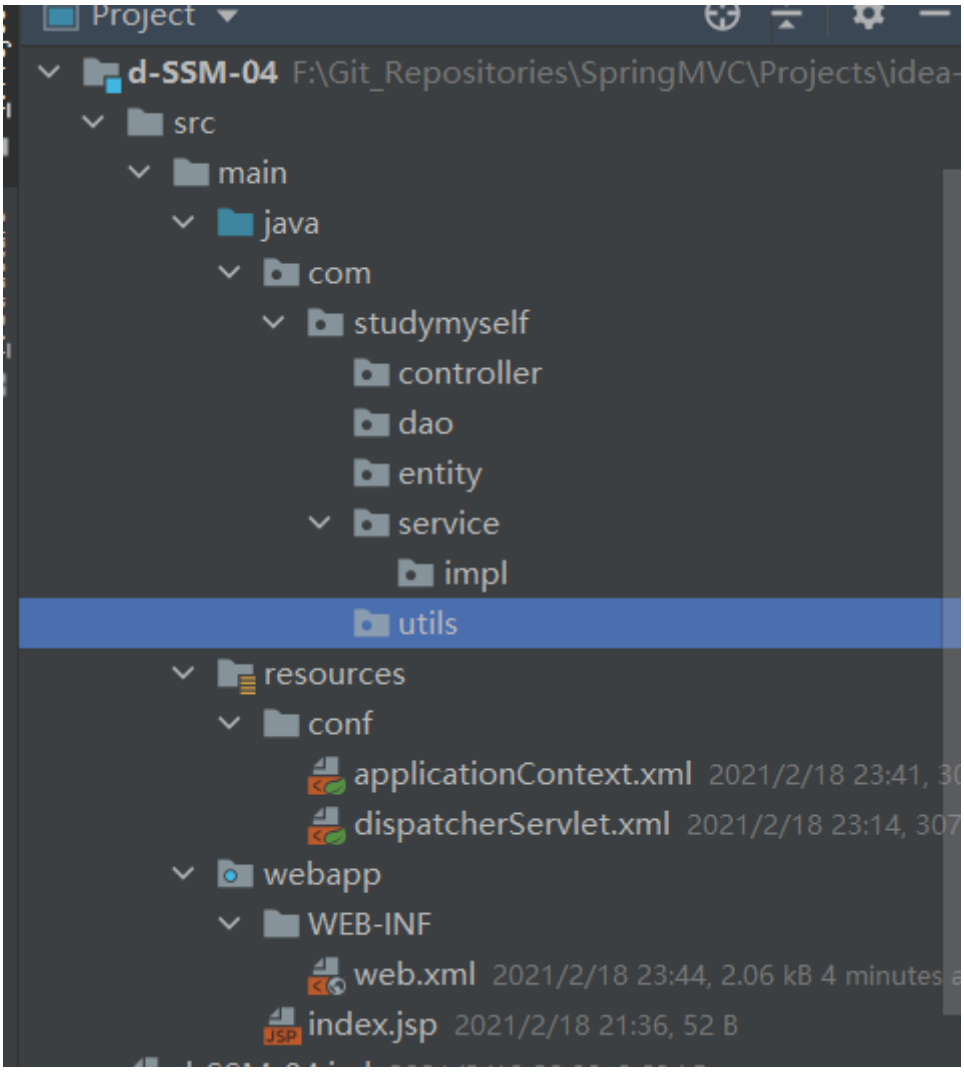
2）、编写项目的web.xml文件，在其中注册中央调度器等对象，需要的配置文件，如SpringMVC配置文件可以先创建，内容先不配置，在resources目录中创建一个子目录conf存放配置文件，具体web.xml内容如下：

web.xml中的标签等过低，到先前的项目中web.xml复制过来，全选ctrl+A

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

  <!--注册SpringMVC的核心对象DispatcherServlet中央调度器-->
  <servlet>
    <servlet-name>springMVC</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <!--注意配置文件放到子目录conf中了,用中央调度器的首字母小写类名-->
      <param-value>classpath:conf/dispatcherServlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springMVC</servlet-name>
    <!--使用斜杠"/"-->
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <!--声明注册过滤器，解决POST请求中文乱码-->
  <filter>
    <filter-name>characterEncodingFilter</filter-name>
    <filter-
class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
      <param-name>forceRequestEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>forceResponseEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

```
  <!--注册Spring的监听器,需要指定自定义的Spring配置文件的位置-->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:conf/applicationContext.xml</param-value>
  </context-param>
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>

</web-app>
```

3）、将程序的各个包结构定义好



然后编写Spring配置文件applicationContext.xml和SpringMVC配置文件dispatcherServlet.xml和属性配置文件jdbc.properties以及mybatis配置文件

dispatcherServlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
```

```
        xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd
http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd">

    <!--springmvc配置文件，声明Controller和web相关的对象-->

    <!--注解开发，声明组件扫描器和controller包的路径-->
    <context:component-scan base-package="com.studymyself.controller"/>

    <!--视图解析器,set注入-->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/static/jsp"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <!--声明注解驱动，作用：
        1、响应Ajax，返回Json
        2、解决静态资源访问问题
    -->
    <mvc:annotation-driven/>

    <!--因为中央调度器的url-pattern是斜杠
    声明使用SpringMVC框架的ResourceHttpRequestHandler对象这个servlet处理静态资源-->
    <mvc:resources mapping="/static/**" location="/static/"/>


</beans>
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">

    <!--spring配置文件，声明service、dao等对象-->

    <!--指定属性资源配置文件的路径，下面数据源需要使用-->
    <context:property-placeholder location="classpath:conf/jdbc.properties"/>

    <!--声明数据源-->
    <bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"
          init-method="init" destroy-method="close">
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

    <!--声明框架中的SqlSessionFactoryBean，间接创建SqlSessionFactory-->
```

```
    <bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
        <property name="dataSource" ref="dataSource"/>
        <property name="configLocation" value="classpath:conf/mybatis.xml"/>
    </bean>

    <!--声明扫描mybatis配置文件的扫描器，属性是dao接口包的路径和sqlSessionFactory对象
        以便创建dao接口的代理对象使用-->
    <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
        <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
        <property name="basePackage" value="com.studymyself.dao"/>
    </bean>

    <!--声明注解扫描器，service类的@Service所在包位置-->
    <context:component-scan base-package="com.studymyself.service"/>

    <!--下面就是需要添加事务时的配置：注解配置，AspectJ配置等，先不加-->

</beans>
```

当我们配置完数据源之后可以把项目放到服务器上部署运行，检查有没有问题，尽早把问题解决了，不要等到最后解决，不然问题太多，很难找出问题所在。完成一部分功能，测试一部分程序，这是一个习惯。

mybatis.xml：这里把日志和分页插件去掉，不使用。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
        PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <!--注意这些属性设置从上往下是有顺序的，写反就会报错-->

    <!--给这个包下的所有类起别名-->
    <typeAliases>
        <package name="com.studymyself.entity"/>
    </typeAliases>

    <mappers>
        <package name="com.studymyself.dao"/>
    </mappers>
</configuration>
```

4）、编写实体类User.java、dao接口和其mapper文件（UserDao.java、UserDao.xml）、service类和其实现类（UserService.java、UserServiceImpl.java）

User.java

```
package com.studymyself.entity;

public class User {
```

```
    private Integer id;
    private String loginName;
    private String loginPwd;
    private String realName;

    public Integer getId() {
        return id;
    }

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

    public String getLoginName() {
        return loginName;
    }

    public void setLoginName(String loginName) {
        this.loginName = loginName;
    }

    public String getLoginPwd() {
        return loginPwd;
    }

    public void setLoginPwd(String loginPwd) {
        this.loginPwd = loginPwd;
    }

    public String getRealName() {
        return realName;
    }

    public void setRealName(String realName) {
        this.realName = realName;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", loginName='" + loginName + '\'' +
                ", loginPwd='" + loginPwd + '\'' +
                ", realName='" + realName + '\'' +
                '}';
    }
}
```

UserDao.java

```
package com.studymyself.dao;

import com.studymyself.entity.User;

import java.util.List;
```

```java
public interface UserDao {

    public int insertUser(User user);
    public List<User> selectAll();
    public User selectByLoginName(String loginName);

}
```

UserDao.xml

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="com.studymyself.dao.UserDao">

    <insert id="insertUser">
        insert into t_user(loginName,loginPwd,realName) values (#{loginName},#
{loginPwd},#{realName})
    </insert>

    <select id="selectAll" resultType="User">
        select id,loginName,realName from t_user
    </select>

    <select id="selectByLoginName" resultType="User">
        select id,loginName,realName from t_user where loginName = #{loginName}
    </select>

</mapper>
```

UserService.java

```java
package com.studymyself.service;

import com.studymyself.entity.User;

import java.util.List;

public interface UserService {

    public int addUser(User user);
    public List<User> queryAll();
    public User queryByLoginName(String loginName);
}
```

UserServiceImpl.java

```java
package com.studymyself.service.impl;

import com.studymyself.dao.UserDao;
import com.studymyself.entity.User;
```

```
import com.studymyself.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import java.util.List;

@Service
public class UserServiceImpl implements UserService {

    @Resource
    private UserDao userDao;

    //这里不作其他业务只做添加和查询的功能

    @Override
    public int addUser(User user) {
        return userDao.insertUser(user);
    }

    @Override
    public List<User> queryAll() {
        return userDao.selectAll();
    }

    @Override
    public User queryByLoginName(String loginName) {
        return userDao.selectByLoginName(loginName);
    }


}
```

5）、编写Controller类和jsp页面

UserController.java

```
package com.studymyself.controller;

import com.studymyself.entity.User;
import com.studymyself.service.UserService;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import javax.annotation.Resource;

@Controller
@RequestMapping("/user")
public class UserController {

    @Resource
    UserService userService;

    /**
```

```
     * 用户注册
     * @param user
     * @return
     */
    @RequestMapping("/addUser")
    public ModelAndView register(User user){

        ModelAndView mav = new ModelAndView();

        //首先判断注册名是否重复，然后进行提示
        Object o = null;
        String tips1 = "注册失败";
        String tips2 = "注册名重复，请重新输入";

        o = userService.queryByLoginName(user.getLoginName());

        if (o!=null){

            mav.addObject("tips2",tips2);
            mav.setViewName("tip");
            return mav;

        }else if (userService.addUser(user)>0){

            tips1 = "用户<br>【"+user+"】<br>注册成功";

        }
        mav.addObject("tips1",tips1);
        mav.setViewName("result");
        return mav;
    }

}
```

result.jsp：注册结果页面

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
    result.jsp 结果页面，注册结果：<br>${tips1}
</body>
</html>
```

index.jsp：欢迎页面

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%
    String basePath = request.getScheme()+"://"+
            request.getServerName()+":"+request.getServerPort()+
            request.getContextPath()+"/";
```

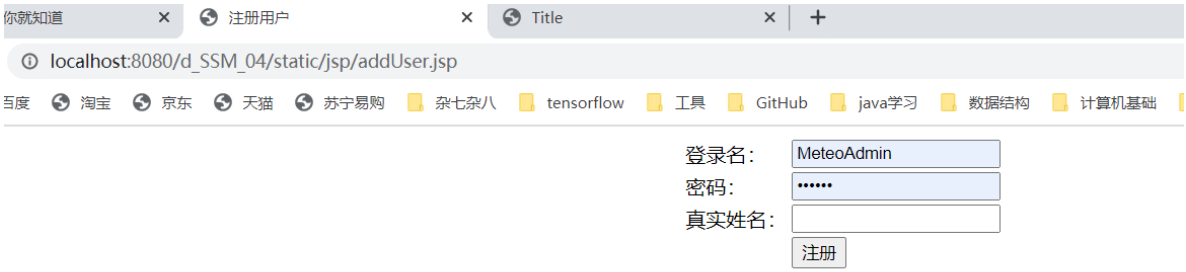
```
%>

<html>
<head>
    <title>功能入口</title>
    <base href="<%=basePath%>">
</head>
<body>
    <div align="center"><!--使其中的内容都居中-->
        <p>SSM整合例子</p>
        <img src="static/images/2.gif"/>
        <table>
            <tr>
                <td><a href="static/jsp/addUser.jsp">注册用户</a></td>
            </tr>
            <tr>
                <td>浏览用户</td>
            </tr>
        </table>
    </div>
</body>
</html>
```

tip.jsp：注册名已存在页面

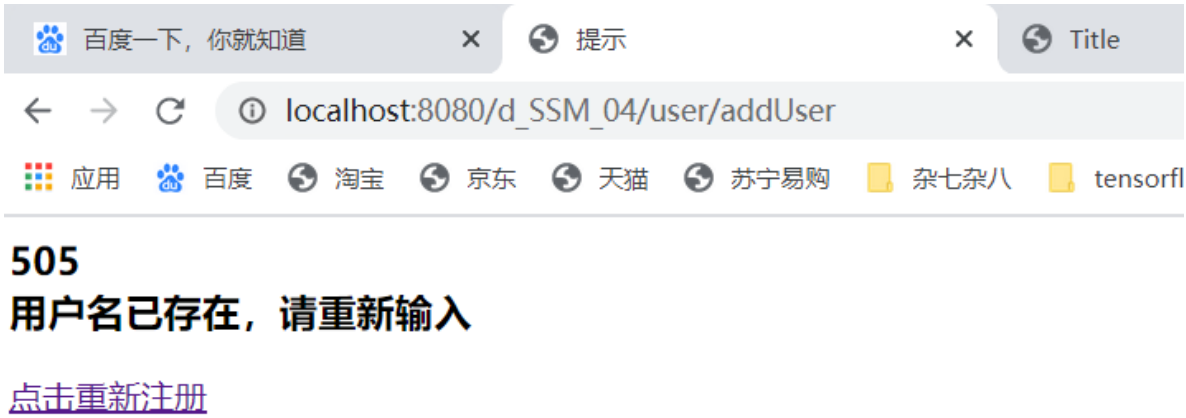
```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    String basePath = request.getScheme()+"://"+
            request.getServerName()+":"+request.getServerPort()+
            request.getContextPath()+"/";
%>
<html>
<head>
    <title>提示</title>
    <base href="<%=basePath%>">
</head>
<body>
    <h3>505<br>${tips2}</h3>
    <a href="static/jsp/addUser.jsp">点击重新注册</a>
</body>
</html>
```

首页

注册页面



注册名已存在页面



注册结果页面

前面测试出现的问题:

1、由于存放静态资源的static目录放在WEB-INF之下，导致无法直接访问以及页面的超链接跳转访问，只能在控制器方法中转发访问，所以把static目录放到webapp目录下，所以修改的位置还有SpringMVC配置文件中声明SpringMVC框架处理静态资源的ResourceHttpRequestHandler对象这个servlet标签中的地址
2、第二就是由于使用的是MySQL驱动是高版本，在属性资源配置文件中指定URL地址的后面要设置时区，以后可能还需要添加其他东西，具体原因以后探究
3、第三就是把模块路径放进@Controller中了，Controller类上面忘记添加@RequestMapping，导致无法访问处理器对象的方法
4、第三就是超链接地址有些写错了，tip.jsp中忘记添加base标签，出错了

6）、完成第二个功能，实现查询用户的功能，使用Ajax请求

使用Ajax请求需要添加jquery的库文件，jquery-3.4.1.js，从网上下载，新建static中新建一个js目录，将库文件粘贴进去。

然后在index页面中引入jquery的库文件

```
jQuery 库是一个 JavaScript 文件，您可以使用 HTML 的 <script> 标签引用它：
<head>
<script src="static/js/jquery-3.4.1.js"></script>
</head>
```

index.jsp如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    String basePath = request.getScheme()+"://"+
            request.getServerName()+":"+request.getServerPort()+
            request.getContextPath()+"/";
%>
<html>
<head>
    <title>浏览用户：查询用户Ajax</title>
    <base href="<%=basePath%>">
    <!--导入jQuery库-->
    <script type="application/javascript" src="static/js/jquery-3.4.1.js">
</script>
    <script type="application/javascript">
        $(function (){
            $("#btnLoader").click(function (){
                $.ajax({
                    url:"user/browserUser",
                    type:"get",
                    dataType:"json",
                    success:function (data){
                        //这里表示每一次进行该方法执行都清除之前html中的数据
                        $("#info").html();
                        //下面就是添加新的数据
                        <!--通过将获取到的json数组进行遍历，将数组中的每一个json对象
                        的数据放到下面table标签的tbody对象中-->
                       $.each(data,function (i,n){
                           $("#info").append("<tr>")
                           .append("<td>"+n.id+"/td")
                           .append("<td>"+n.loginName+"/td")
                           .append("<td>"+n.realName+"/td")
                           .append("</tr>")
                       })
```

```
                    }
                })
            })
        })
    </script>
</head>
<body>
    <div align="center">
        <table>
            <thead>
            <tr>
                <td>主键id</td>
                <td>登录名</td>
                <td>真实姓名</td>
            </tr>
            </thead>
            <!--下面是table的tbody对象-->
            <tbody id="info">

            </tbody>
        </table>
        <input type="button" id="btnLoader" value="查询数据">
    </div>
</body>
</html>
```

UserController类中添加一个如下方法：

```
  /**
     * 处理查询，响应Ajax
     * @return
     */
    @RequestMapping("/browserUser")
    @ResponseBody
    public List<User> browserUser(){

        List<User> userList = userService.queryAll();
        return userList;

    }
```

当我们写完一个控制器方法后，可以直接访问http://localhost:8080/d_SSM_04/user/browserUser进行测试，这就是测试一个对外接口，一个对外的可以被人访问的地址