图书管理系统

1. 项目介绍：

此项目主要用于对图书的管理操作，对图书进行分类和图书管理。其功能通过连接数据库，使用Spring 、springMVC、mybatis整合开发框架搭建开发而成，核心功能主要是对图书进行添加，修改查询操作。再对图书进行分类管理，是让图书资源更好更清晰的管理，让我们在查询图书时，更清晰方便简洁的操作。本项目还整合了jQuery前端框架对前端页面布局进行处理，为实现更好的交互操作，并解决了交互复杂度。

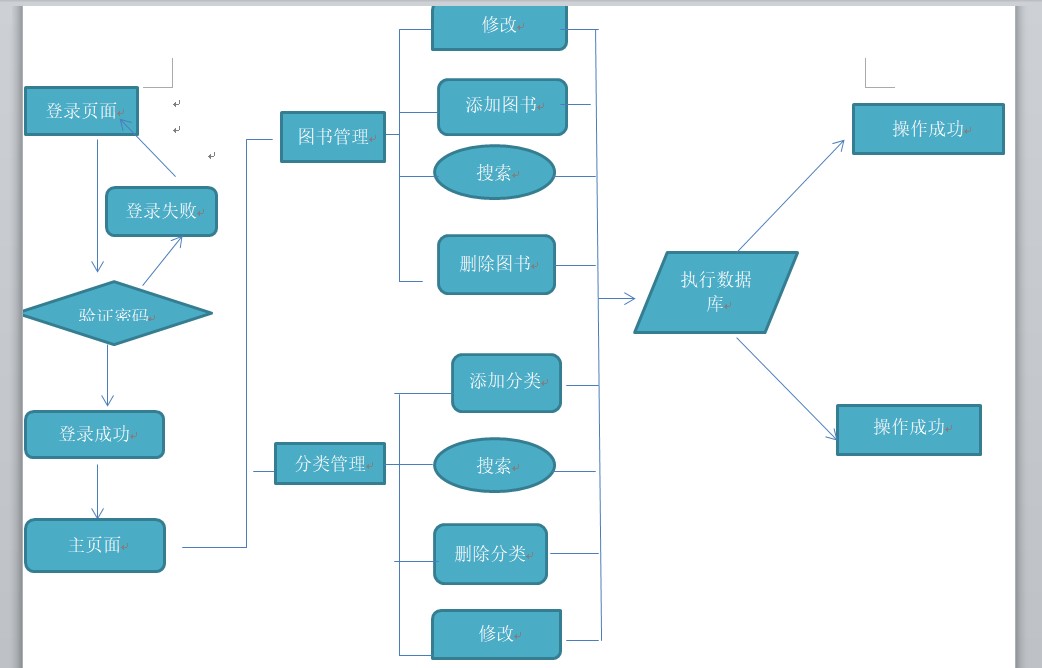
开发环境：jdk1.8,idea

二、需求分析

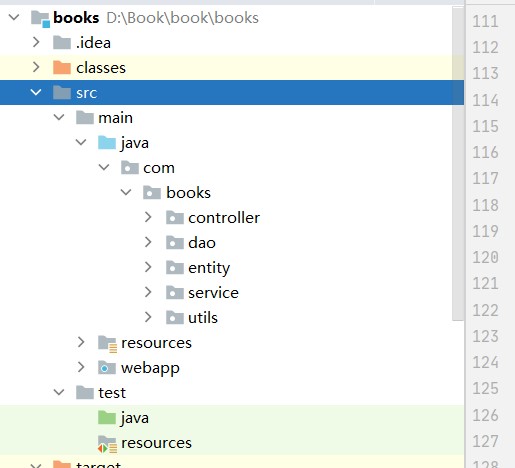
需求主要有登录功能，用户登录后才能进入系统进行使用，然后对图书有查询、添加、删除的核心功能。然后是图书分类功能，可以添加、修改、删除等操作。以便将图书进行分类操作，以便通过分类快速找到图书资源。

三、需求实现

（1）业务流程图：

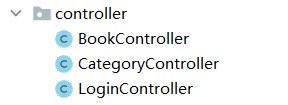


（2）项目结构图：

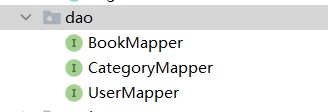


1. 其中controller为控制层主要用于控制转发请求，对浏览器各个请求行为进行处理：

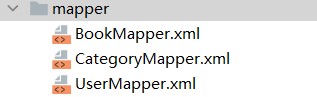
此项目分为三个控制器，分别用于对图书的处理、分类的处理、登录处理。



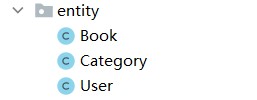
1. dao层用于编写对对数据库的事务操作，实现对数据库的增删改查操作：



其实现类用xml配置文件重写接口方法：



1. entity层用于编写与数据库的映射字段：



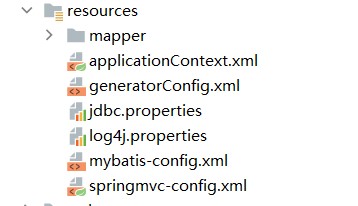
1. service层主要用于编写本项目的业务逻辑代码，进行数据库有关的操作



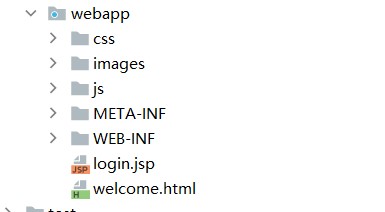
1. util层主要为工具层。

sql

1. resource层主要为配置文件层，主要用于对springmvc.xml、jdbc等的文件进行配置

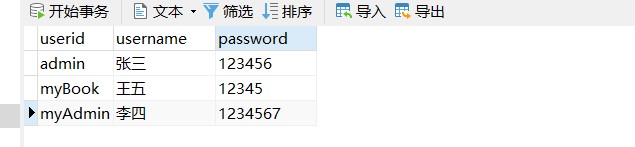


1. webapp层主要用于放置，前端页面、图片等静态页面：



1. 数据表结构：

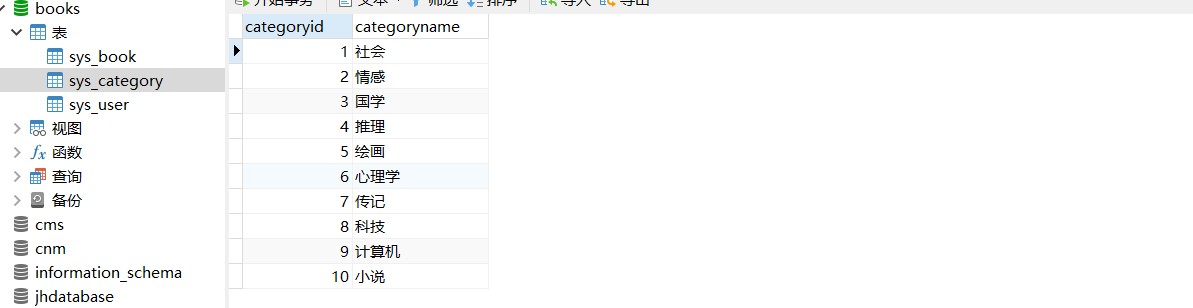
登录账户表：



图书管理表：



图书分类表：

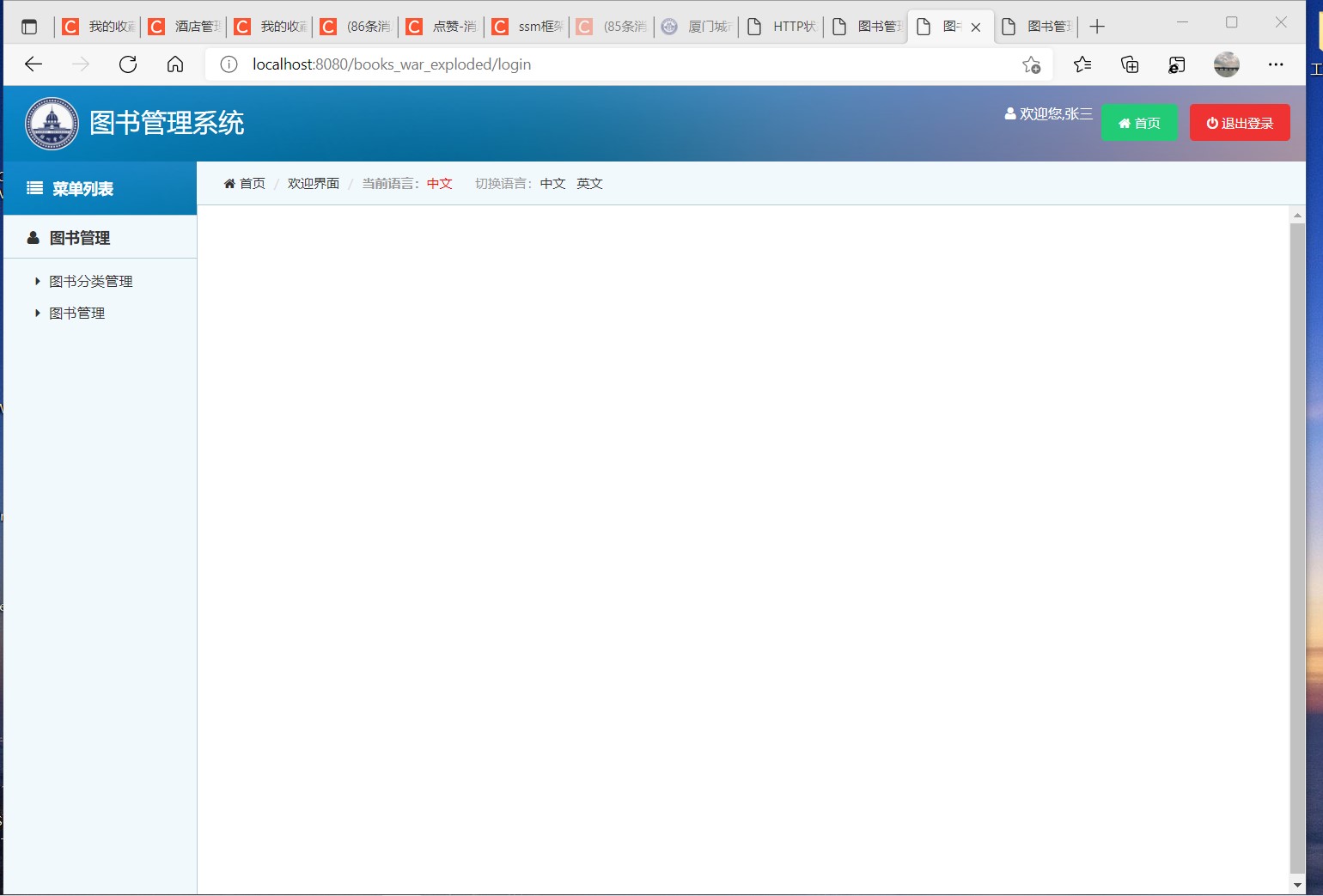


四、结果展示：

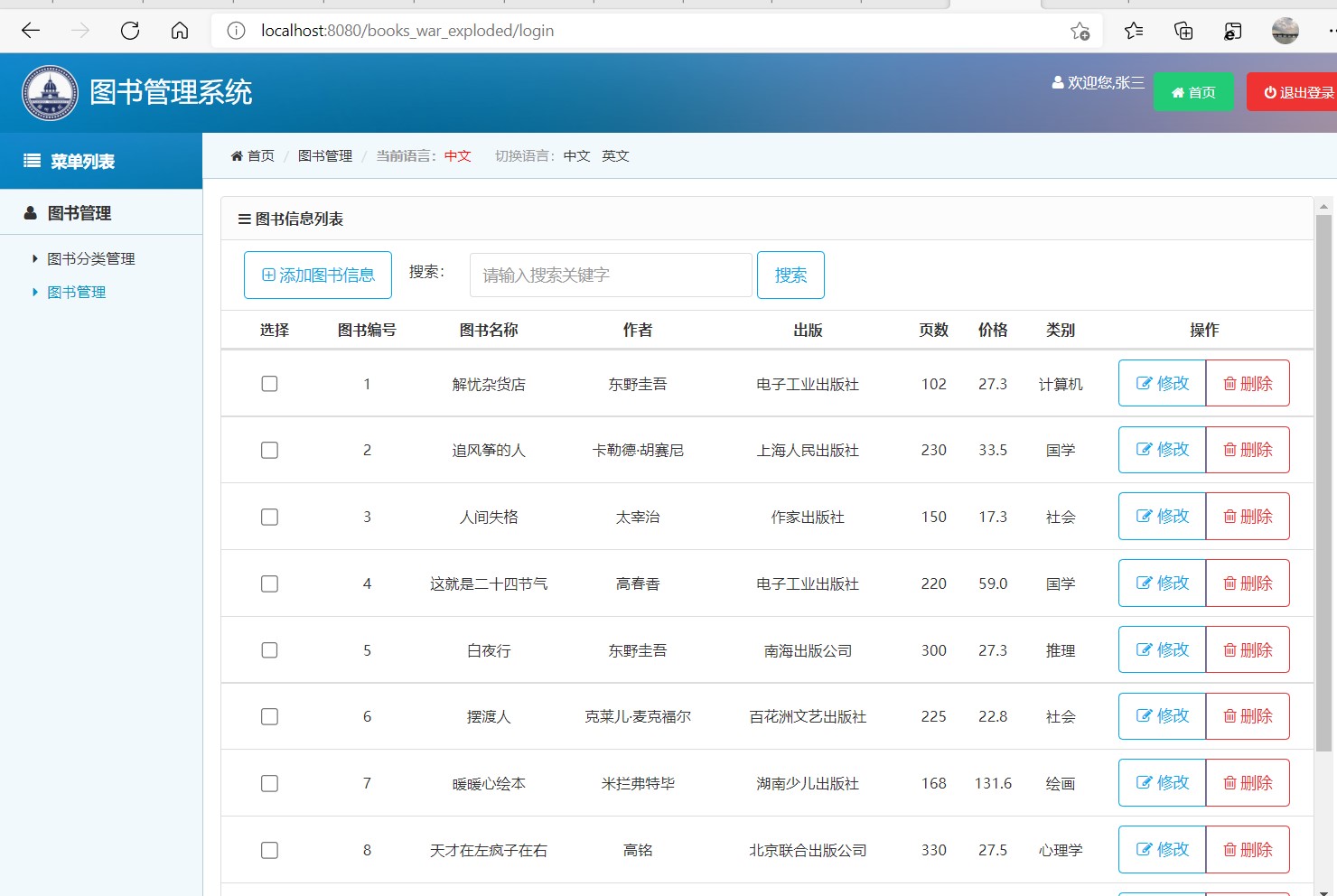
1. 登录页面：



2.登录成功后：



3.图书管理页面：



4.搜索图书：



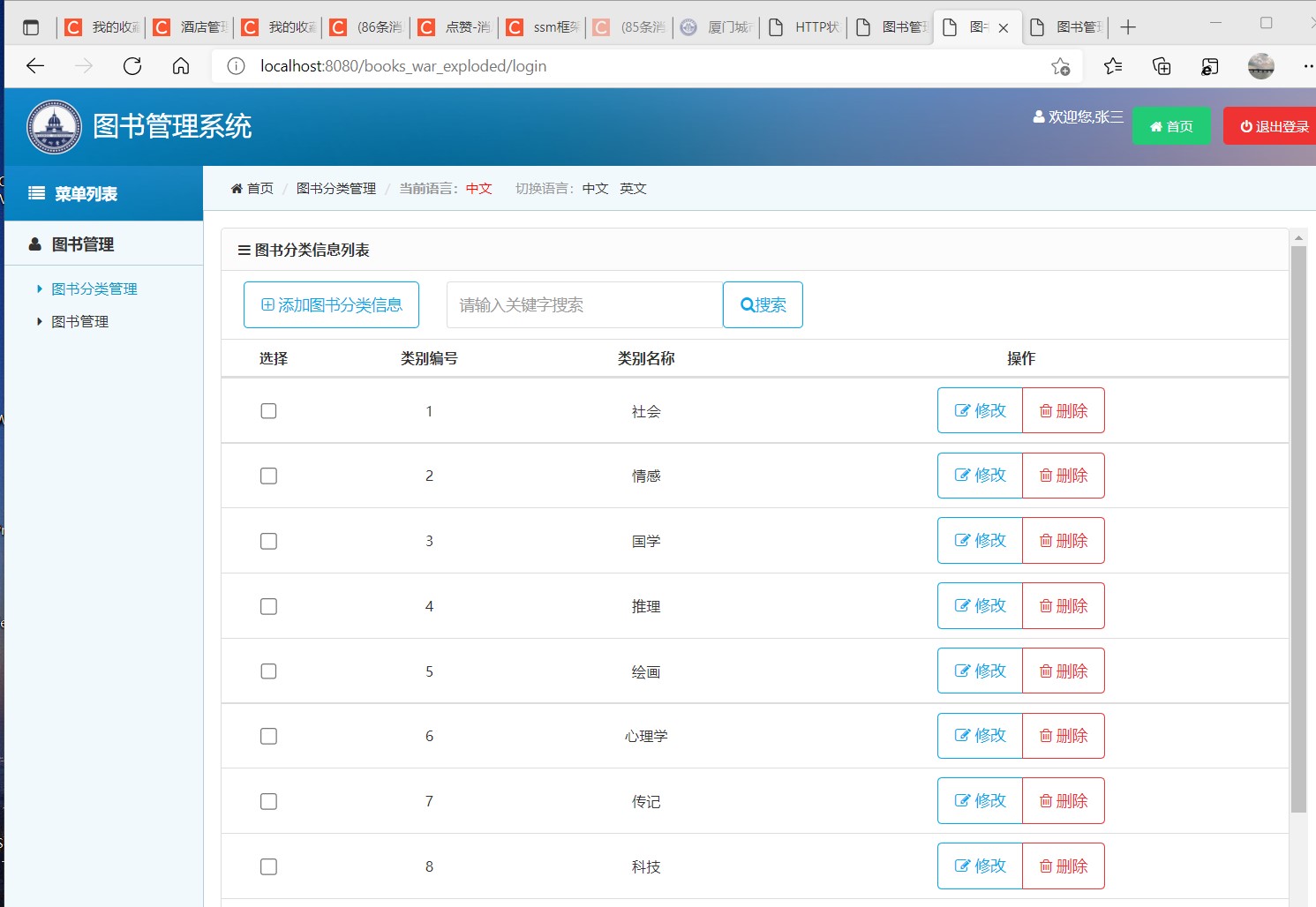
5.添加图书操作：



6.修改图书：



7.分类页面：



8.搜索类别：



9.添加类别：



10.修改类别：



五、具体代码实现（节选登录操作功能的实现）：

1. 首先导入相关的依赖以及配置好springmvcm,连接池，web.xml中配置对页面请求操作的监听器、视图解析器、拦截器，以及前后缀过滤操作：

*<!--spring所需的依赖-->*<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
  
*<!--AOP-->*<dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.2</version>  
</dependency>  
  
*<!--spring的事务管理-->*<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-tx</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
*<!--如果不写测试的话，可以不需要导入-->*<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
*<!--spring-java,如：DataSourceTransactionManager的使用-->*<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
*<!--springmvc-->*<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
</dependency>

2.数据库连接url：

jdbc.driver=com.mysql.jdbc.Driver  
jdbc.url=jdbc:mysql://127.0.0.1:3306/books?serverTimezone=GMT%2B8&useSSL=false&characterEncoding=utf8  
jdbc.username=root  
jdbc.password=000000

1. web.xml:

*<!-- 配置加载Spring文件的监听器-->*<welcome-file-list>  
 <welcome-file>/login.jsp</welcome-file>  
</welcome-file-list>

*<!-- 编码过滤器 -->* <filter>  
 <filter-name>encodingFilter</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>forceEncoding</param-name>  
 <param-value>true</param-value>  
 </init-param>  
 </filter>  
 <filter-mapping>  
 <filter-name>encodingFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
 </filter-mapping>  
*<!-- 配置Spring MVC前端核心控制器 -->*<servlet>  
 <servlet-name>springmvc</servlet-name>  
 <servlet-class>  
 org.springframework.web.servlet.DispatcherServlet  
 </servlet-class>  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:springmvc-config.xml</param-value>  
 </init-param>  
 *<!-- 配置服务器启动后立即加载Spring MVC配置文件 -->* <load-on-startup>1</load-on-startup>

1. 核心代码的实现：

首先在entiy层建立一个类用于映射数据库对应的字段：

public class User {  
 private String userid;  
 private String username;  
 private String password;  
 public String getUserid() {  
 return userid;  
 }  
  
 public void setUserid(String userid) {  
 this.userid = userid == null ? null : userid.trim();  
 }  
 public String getUsername() {  
 return username;  
 }  
 public void setUsername(String username) {  
 this.username = username == null ? null : username.trim();  
 }  
 public String getPassword() {  
 return password;  
 }  
 public void setPassword(String password) {  
 this.password = password == null ? null : password.trim();  
 }  
}

然后在dao层整合mybatis进行编写对数据库操作的接口：

public interface UserMapper<String> {  
 *//删除操作* int deleteByPrimaryKey(String userid);  
*//插入用户* int insert(User record);  
  
 int insertSelective(User record);  
  
 User selectByPrimaryKey(String userid);  
  
 int updateByPrimaryKeySelective(User record);  
  
 int updateByPrimaryKey(User record);  
*//通过用户ID查找用户* User findUserByUserId(String userid);  
}

在mapper层用xml文件形式让dao层字段与数据库字段相对应并且在此文件中编写好对数据库的操作：

*<?*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.books.dao.UserMapper">  
 <resultMap id="BaseResultMap" type="com.books.entity.User">  
 <id column="userid" jdbcType="VARCHAR" property="userid" />  
 <result column="username" jdbcType="VARCHAR" property="username" />  
 <result column="password" jdbcType="VARCHAR" property="password" />  
 </resultMap>  
 <sql id="Base\_Column\_List">  
 userid, username, password  
 </sql>  
  
 <select id="findUserByUserId" resultMap="BaseResultMap">  
 select *\** from sys\_user where userid=#{userid,jdbcType=VARCHAR}  
 </select>  
 <select id="selectByPrimaryKey" parameterType="java.lang.String" resultMap="BaseResultMap">  
 select   
 <include refid="Base\_Column\_List" />  
 from sys\_user  
 where userid = #{userid,jdbcType=VARCHAR}  
 </select>  
 <delete id="deleteByPrimaryKey" parameterType="java.lang.String">  
 delete from sys\_user  
 where userid = #{userid,jdbcType=VARCHAR}  
 </delete>  
 <insert id="insert" parameterType="com.books.entity.User">  
 insert into sys\_user (userid, username, password  
 )  
 values (#{userid,jdbcType=VARCHAR}, #{username,jdbcType=VARCHAR}, #{password,jdbcType=VARCHAR}  
 )  
 </insert>  
 <insert id="insertSelective" parameterType="com.books.entity.User">  
 insert into sys\_user  
 <trim prefix="(" suffix=")" suffixOverrides=",">  
 <if test="userid != null">  
 userid,  
 </if>  
 <if test="username != null">  
 username,  
 </if>  
 <if test="password != null">  
 password,  
 </if>  
 </trim>  
 <trim prefix="values (" suffix=")" suffixOverrides=",">  
 <if test="userid != null">  
 #{userid,jdbcType=VARCHAR},  
 </if>  
 <if test="username != null">  
 #{username,jdbcType=VARCHAR},  
 </if>  
 <if test="password != null">  
 #{password,jdbcType=VARCHAR},  
 </if>  
 </trim>  
 </insert>  
 <update id="updateByPrimaryKeySelective" parameterType="com.books.entity.User">  
 update sys\_user  
 <set>  
 <if test="username != null">  
 username = #{username,jdbcType=VARCHAR},  
 </if>  
 <if test="password != null">  
 password = #{password,jdbcType=VARCHAR},  
 </if>  
 </set>  
 where userid = #{userid,jdbcType=VARCHAR}  
 </update>  
 <update id="updateByPrimaryKey" parameterType="com.books.entity.User">  
 update sys\_user  
 set username = #{username,jdbcType=VARCHAR},  
 password = #{password,jdbcType=VARCHAR}  
 where userid = #{userid,jdbcType=VARCHAR}  
 </update>  
</mapper>

然后在业务层上，通过spring注解形式将dao层数据库接口以注入属性的形式注入进来，编写业务代码：

@Service  
@Transactional  
public class UserService {  
   
 @Autowired  
 private UserMapper userMapper;  
  
 public User findUserByUserId(String userid) {  
 return userMapper.findUserByUserId(userid);  
 }  
}

接下来是控制器操作整合springmvc中关键代码对前端请求处理：主要实现分发响应内部静态资源操作。

@Controller  
public class LoginController {  
 @Autowired  
 private UserService userService;  
  
 @RequestMapping(value = "/login",method = RequestMethod.*POST*)  
 public String checkUser(@RequestParam(name = "userid") String userid,String password,HttpSession session) {  
 User user = userService.findUserByUserId(userid);  
 if (user != null && user.getPassword().equals(password)) {  
 session.setAttribute("userInfo", user);  
 return "index";  
 }  
 return "../../login";  
 }  
   
 @RequestMapping("/welcome")  
 public String openWelcomePage(Model model) {  
 return "welcome";  
 }  
}

其后的对图书的管理操作代码也与上述流程一致，主要区别在于service层和控制层。