**上海大学**

**程序设计及应用（Java）大作业成果报告**

题目: **电子图书馆综合管理系统**

**学生姓名: 胡圣翔 学 号: 14124532**

**学生姓名: 柴名扬 学 号: 15123019**

**学生姓名: 段 婕 学 号: 15121073**

**任课教师: 高洪皓**

二零一七年十一月二日

**目 录**

1. **序言**

**1.1项目背景**

**1.2项目要求**

**1.3项目简介**

**二、技术简介**

**2.1 TOMCAT**

**2.2 Web网页**

**2.3 MYSQL数据库**

**2.4 JSP技术**

**2.5 JDBC**

**三、项目实施方案**

**3.1项目整体逻辑**

**3.2系统功能实现**

**3.2.1总体功能**

**3.2.2管理员功能**

**3.2.3读者功能**

**3.2.4其他功能**

**3.3数据库设计**

**3.3.1数据库总体结构设计**

**3.3.2图书信息表结构设计**

**3.3.3 读者信息表结构设计**

**3.3.4 图书借阅信息表结构设计**

**3.3.5 数据库语句算法**

**3.4系统实现（此处可写各部分功能的关键代码）**

1. **项目感想**
2. **成员分工**
3. **源代码**
4. 序言

1.1项目背景

随着计算机技术的发展以及计算机网络的逐渐普及， 英特网逐渐成为成为人们查找、交流信息的重要工具，而使用计算机来管理图书馆已经成为必然趋势。图书馆在正常运营中会面对大量的读者信息、书籍信息以及由两者相互作用产生的借书信息，还书信息。若使用结构化系统分析与设计的方法，建立一套有效的图书信息管理系统，可以大大提高图书馆信息管理工作的效率和质量，因此根据图书馆目前实际的管理情况设计和开发一套图书馆管理系统是十分必要的。

1.2项目功能

模拟现有图书馆的读者登录、注册、借阅、续借、归还，图书管理员图书入库、出库，查询、排序等功能，实现电子图书馆的智慧综合管理。

1.3项目简介

本电子图书馆管理系统使用MVC框架（Jsp+Servlet+JavaBean）进行架构，以Tomcat为Web服务器，以MySQL为后台数据库，使用Jsp进行网页内容开发，实现了面对图书管理员和读者两种对象的各种系统功能。读者可对图书进行查询、借阅、续借和归还等操作，图书管理员可进行图书入库、出库 等管理工作，大大提高了图书馆信息管理的工作效率。

二、相关技术简介

2.1 TOMCAT

该项目以Tomcat为Web服务器。Tomcat服务器是Apache Group Jakarta小组开发的一个免费服务器软件，适合于嵌入Apache中使用。Tomcat服务器的兼容性很好，如WebLogic服务器采用其为Web服务器引擎，Jbuilder将其作为标准的测试服务器，Sun公司也将其作为JSP技术应用的示例服务器。

2.2 Web网页

本系统采用B/S模式来实现客户端对服务器端的调用，所谓B/S模式即浏览器/服务器模式，是一种从传统的二层C/S模式发展起来的新的网络结构模式。在B/S模式中，客户端运行浏览器软件。浏览器以超文本形式向WEB服务器提出访问数据库的要求，WEB服务器接受客户端请求后，将这个请求转化为SQL语法，并交给数据库服务器，数据库服务器得到请求后，验证其合法性，并进行数据处理，然后将处理后的结果返回给WEB服务器，WEB 服务器再一次将得到的所有结果进行转化，变成HTML文档形式，转发给客户端浏览器，以友好的WEB页面形式显示出来。如图2.1所示。

Web Browser

Web Browser

Web Browser

WEB

服务器

数据库服务器

图2.1 B/S模式图

与传统的C/S模式相比，B/S结构把处理功能全部移植到了服务器端，用户的请求通过浏览器发出，无论是使用和数据库维护上都比传统模式更加经济方便。而且使维护任务层次化：管理员负责服务器硬件日常管理和维护，系统维护人员负责后台数据库数据更新维护。

2.3 MYSQL数据库

本项目以MySQL为后台数据库。MySQL是一个多用户、多线程的SQL数据库，是一个客户机/服务器结构的应用，它由一个服务器守护程序mysqld和很多不同的客户程序和库组成。MySQL的快速和灵活性足以满足一个网站的信息管理工作。

2.4 JSP技术

本项目使用JSP技术将数据库与web网页进行连接。JavaServer Pages 技术是一个纯Java平台的技术，它主要用来产生动态网页内容。在传统的网页HTML文件中加入Java程序片段和JSP标记，就构成了JSP网页。Web服务器接收到访问JSP网页的请求时，首先执行其中的程序片段，然后将执行结果以HTML格式返回给客户。JSP是基于Java Servlet以及整个Java体系的Web开发技术，利用这一技术可以建立先进、安全、快速和跨平台的动态网页。

2.5 JDBC

JDBC由一组Java语言编写的类和接口组成，使用内嵌式的SQL，主要实现三方面的功能：建立与数据库的连接，执行SQL声明以及处理SQL执行结果。JDBC支持基本的SQL功能，使用它可方便地与不同的关系型数据库建立连接，进行相关操作，并无需再为不同的DBMS分别编写程序。

1. 项目实施方案

3.1项目整体逻辑

该项目整体分为三部分来实现。首先，利用JSP技术进行网页内容开发，实现电子图书馆综合管理系统的动态网页内容。其次，使用MySQL作为后台数据库，实现电子图书馆综合管理系统中所有数据的管理。最后Java与数据库的连接通过JDBC来完成，实现电子图书馆综合管理系统中jsp网页内容与MySQL之间的连接。整体逻辑框图如图3.1-1所示。

数据库的管理

Java与数据库的连接

界面显示部分

JSP技术

MySQL

JDBC

图3.1-1项目整体逻辑图

3.2功能实现

3.2.1总体功能

该虚拟图书管实现了面对图书管理员和读者两种对象的各种系统功能。进入登录界面以后， 读者和管理员均需通过用户名及密码进行登录，系统将自动识别登录账号的权限，动态的在主页显示相应的模块入口。读者若无登录权限，可进行注册账号。当用户名和密码成功匹配或注册成功之后，自动跳转到图书馆主页。读者可对图书馆的馆藏图书进行查询、借阅、续借和归还等操作，图书管理员可进行图书入库、出库等管理工作。总体功能图如图3.2.1所示。

图书管理系统

返回登录界面

返回登录界面

登录类型选择

否 否

读者

管理员

密码和用户名

密码和用户名

是否匹配

是否匹配

是

是

进入管理员系统

进入读者系统

图书出库

图书归还

图书

入库

图书借阅

图书借阅

图书归还

图书续借

图书

查询

图3.2.1总体功能流程图

3.2.2管理员功能

进入管理员界面以后，管理员可对图书进行图书出库、入库

功能图如图3.2.2所示。

图书出库、入库：可对系统中的图书进行出库、入库等图书管理工作。

管理员

图书管理

图书出库

图书入库

图3.2.2管理员功能图

3.2.3读者功能

进入读者界面以后，读者可根据不同要求对图书进行查询，并进行图书的借阅、归还以及续借。功能图如图3.2.2所示。

1.图书查询：可通过不同关键字对所需借阅的图书进行查询

2.图书借阅：主要功能是当读者进行图书借阅时，在数据库中插入一条借书记录，该记录包括读者ID、图书ID、借出日期、还书期限及其它相关信息。

3.图书归还：主要功能是当读者进行图书归还时，将该借书记录删除。

4.图书续借：进行续借相关功能，在已借书本的表中修改归还时间

读者

图书

查询

图书续借

图书借阅

图书归还

图3.2.2读者功能图

3.3数据库设计

3.3.1数据库总体结构设计

整个系统所包括的信息有图书信息、读者信息、图书借阅信息、图书归还信息、管理员信息、读者类型信息、图书类型信息。可将信息抽象为下列数据项和数据结构：

(1) 图书信息：图书编号，图书名称，图书类型，作者，ISBN号，价格，出版社，所在书架，入库时间等

(2) 成员信息：编号，姓名，密码，是否具有管理员权限；

(4) 图书借阅信息：图书编号，读者ID，借出时间，应还时间，是否续借；

3.3.2图书信息表结构设计

图书信息表主要用于存储图书馆中所藏图书的相关信息，是由操作员在图书入库时进行添加完善的，包括图书编号、书名、作者、出版社、 类别、数目等信息。此图书信息表主要用于读者和图书管理员对馆中图书的查询，系统用户根据图书的某个属性进行查询，便可得知图书的其他相关信息，其中图书所在书架属性是便于读者借阅时对图书的寻找

3.3.3 读者信息表结构设计

读者信息表的设计是为了图书馆管理员对读者进行有效管理，包括读者姓名、读者类型、学号、电话、现借数目、允许借书的最大数目等信息。其中不同学号对应的读者ID都是唯一的，读者在登录时ID与密码需一一对应。读者的电话信息是为了方便与读者进行联系。同时，管理员可查看读者现借的图书数目以及最多可借的图书数目，便于对读者信息进行操作。

3.3.4 图书借阅信息表结构设计

图书借阅信息表的设计是用于对读者借阅图书进行查询和管理，表中图书ID属性是对借阅图书的唯一性识别标识，读者ID号记录借阅该书的相应读者，并包括了借书时间和归还时间。方便管理员查阅和管理借阅图书的信息。

3.4系统实现（此处可写各部分功能的关键代码）

前端主要通过vue+bootstrap+jquery实现，采用MVC模式

3.4.1 数据库连接模块

为了实现模块的重用性，所有的数据库连接都由Connector类的实例实现

public class Connector implements Serializable{  
 private Connection conn;  
 private PreparedStatement ptmt;  
 public Connection getConn(){  
 return conn;  
 }  
 public void connect(){  
 try {  
 Class.*forName*("com.mysql.jdbc.Driver");  
 conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/Library?user=root&password=123&useUnicode=true&characterEncoding=UTF8");  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 public void closeConn(){  
 try {  
 conn.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
}

3.4.2 书本模型模块

将数据库中书本信息表中的一条记录抽象成一个BookModel的实例，用于生成json返回前端

public class BookModel {  
 private String name;  
 private int id, type, total, amount;  
 private String version;  
 private String author;  
 private String cover;  
 private String initTime = "", deadline = "", isExtended = "";  
 File coverFile;  
 public void setName(String name){  
 this.name = name;  
 }  
 public String getName(){  
 return this.name;  
 }  
 public void setType(int type){  
 this.type = type;  
 }  
 public int getType(){  
 return this.type;  
 }  
 public void setTotal(int total){  
 this.total = total;  
 }  
 public int getTotal(){  
 return this.total;  
 }  
 public void setId(int id){  
 this.id = id;  
 }  
 public int getId(){  
 return this.id;  
 }  
 public void setVersion(String version){  
 this.version = version;  
 }  
 public String getVersion(){  
 return this.version;  
 }  
 public void setCover(String cover){  
 this.cover = cover;  
 }  
 public String getCover(){  
 return this.cover;  
 }  
 public void setAuthor(String author){  
 this.author = author;  
 }  
 public String getAuthor(){  
 return this.author;  
 }  
 public void setCoverFile(File coverFile){  
 this.coverFile = coverFile;  
 }  
 public File getCoverFile(){  
 return this.coverFile;  
 }  
 public void setAmount(int amount){  
 this.amount = amount;  
 }  
 public int getAmount(){  
 return this.amount;  
 }  
 public void setInitTime(String t){  
 this.initTime = t;  
 System.*out*.print(t);  
 }  
 public String getInitTime(){  
 return this.initTime;  
 }  
 public void setDeadline(String deadline){  
 this.deadline = deadline;  
 }  
 public String getDeadline(){  
 return this.deadline;  
 }  
 public void setIsExtended(String isExtended){  
 this.isExtended = isExtended;  
 }  
 public String getIsExtended(){  
 return this.isExtended;  
 }  
}

3.4.3 登录模块

通过设置cookie来记录登录的用户及管理员权限

*/\*\*  
 \* 设置cookie  
 \** ***@param*** *cname cookie的键名  
 \** ***@param*** *cvalue cookie的值  
 \* \*/***function** setCookie(cname, cvalue) {  
 document.cookie = cname + "=" + cvalue;  
}  
  
*/\*\*  
 \* 获取cookie函数  
 \** ***@param*** *cname 想要获取值的cookie键名  
 \** ***@return*** *string cookie值，如果cookie不存在返回 ""  
 \*\*/***function** getCookie(cname) {  
 **var** name = cname + "=";  
 **var** ca = document.cookie.split(";");  
 **for**(**var** i = 0; i < ca.length; i++){  
 **var** c = ca[i].trim();  
 **if**(c.indexOf(name) == 0){  
 **return** c.substring(name.length, c.length);  
 }  
 }  
 **return** "";  
}

登录前后端通信函数

本项目所有前后端通信都通过ajax实现

login: **function** () {  
 $.ajax({  
 type: "POST",  
 url: "../jsp/login.jsp",  
 dataType: "html",  
 data: $('#lb\_loginForm').serialize(),  
 success: **function** (data) {  
 eval("var json = " + data);  
// 根据json.isPswCorrect判定登录是否成功  
 **if**(!json.isPswCorrect){  
 login.isPswError = **false**;  
 }**else** {  
// 设置id、user、admin Cookie  
 login.isPswError = **true**;  
 **var** userId = json.id,  
 admin = json.isAdmin;  
 setCookie("user", $(' #userName ').val());  
 setCookie("admin", admin);  
 setCookie("userId", userId);  
 window.location.replace("./index.jsp");  
 }  
 }  
 });  
 },

jsp验证代码

try {  
 connector.connect();  
 String name = request.getParameter("userName");  
 String psw = request.getParameter("psw");  
 String sql = "select \* from customer where name='" + name + "' and password='" + psw + "'";  
 String isAdmin = "";  
 int id = 0;  
 ResultSet rs = connector.query(sql);  
 if(rs != null){  
 if(rs.next()){  
 id = rs.getInt(1);  
 isAdmin = rs.getString(4);  
 String json = "{id:" + id + ", isAdmin:" + isAdmin + ", isPswCorrect: true}";  
 response.getWriter().write(json);  
 connector.closeConn();  
 }else {  
 response.getWriter().write("{isPswCorrect: false}");  
 }  
 }  
 rs.close();  
 connector.closeConn();  
}catch (Exception e){  
 e.printStackTrace();  
}

3.4.4 搜索模块

从前端取到搜索关键字并实例化PreparedStatement，执行sql查询语句，若数据库中有记录则返回json数组

if(method.equals("getSearch")){  
 try {  
 String text = URLDecoder.*decode*(URLDecoder.*decode*(request.getParameter("text"), "utf-8"), "utf-8");  
 sql = "select \* from " + bookTable + " where (name like '%" + text + "%' or writer like '%" + text + "%') and type=" + type + " ;";  
 rs = connector.query(sql);

while(rs.next()){  
 BookModel bookModel = new BookModel();  
 bookModel.setId(rs.getInt(1));  
 bookModel.setName(rs.getString(2));  
 bookModel.setType(rs.getInt(3));  
 bookModel.setVersion(rs.getString(4));  
 bookModel.setAuthor(rs.getString(5));  
 bookModel.setAmount(rs.getInt(6));  
 bookModel.setTotal(rs.getInt(7));  
 bookModel.setCover(rs.getString(8));  
 bookModels.add(bookModel);  
 }  
 if(bookModels.size() != 0){  
 //将ArrayList对象转化成json数组；  
 JSONArray ja = JSONArray.*fromObject*(bookModels);  
// json数组转换成字符串  
 String jas = ja.toString();  
 response.getWriter().write(jas);  
 }else{  
 response.getWriter().write("false");  
 }  
 rs.close();  
 connector.closeConn();

}catch (Exception e){  
 e.printStackTrace();  
 }

3.4.5 排序模块

当用户搜索完成或管理员在管理书籍时，可以依据相应的关键字对书籍进行排序，此处若是升序则进行一次冒泡排序，若是降序则是先进行一次冒泡，再将冒泡后的数据数组进行逆序操作；

*/\*\*  
 \** ***@param*** *e window.event 对象  
 \** ***@param*** *item 排序的关键字  
 \** ***@param*** *toggle 排序标识位  
 \* \*/*sort: **function** (e, item, toggle) {  
 e = window.event;  
 **var** temp;  
 **for**(**var** i = 0; i < **this**.bookTypes.length; i++){  
 // 对象长度  
 **var** length = **this**.bookTypes[i].length;  
 **for**(**var** j = 0; j < length - 1; j++){  
 **for**(**var** k = j + 1; k < length; k++){  
 **if**(**this**.bookTypes[i][j][item] > searchResult[i][k][item]){  
 //改变vue实例数据模型时一定要通过Vue.set()；否则无法触发视图的更新！！！！！！！  
 temp = **this**.bookTypes[i][k];  
 Vue.set(**this**.bookTypes[i], k, **this**.bookTypes[i][j]);  
 Vue.set(**this**.bookTypes[i], j, temp);  
 }  
 }  
 }  
 **if**(**this**[toggle] === 0 && length !== undefined){  
 **this**.bookTypes[i].reverse();  
 }  
 }  
 **if**(**this**[toggle] === 1){  
 $(e.target).find(" .caret ").css({transform: "rotate(180deg)"});  
 **this**[toggle] = 0;  
 }**else** {  
 $(e.target).find(" .caret ").css({transform: "rotate(360deg)"});  
 **this**[toggle] = 1;  
 }  
},

3.4.6 读者借阅模块

读者借阅先需要判断当前书籍是否还有库存，若无则提示无法借阅，若有库存则在前端生成一个时间戳，与书本id、读者id一起传给服务器进行数据库操作，在书本借阅表中插入一条借阅记录

borrow: **function** (e, index, type, id, amount, name) {  
 **var** userId = getCookie("userId");  
 **if**(amount > 0){  
 **var** date = **new** Date(),  
 initDate = date.getTime(),  
 deadline = **new** Date(initDate + 30 \* 24 \* 60 \* 60 \* 1000);  
 **if**(confirm("确认借阅《" + name + "》？请在" + deadline.getFullYear() + "年" + (deadline.getMonth() + 1) + "月" + (deadline.getDate() + 1) + "日之前归还！")){  
 $.ajax({  
 type: "POST",  
 url: "../jsp/borrow.jsp",  
 // dataType: 'json',  
 data: {bookId: id, amount: amount - 1, userId: userId, initDate: initDate, deadline: (initDate + 30 \* 24 \* 60 \* 60 \* 1000), name: encodeURI(encodeURI(name))},  
 success: **function** (data) {  
 console.log("data: " + data + " " + **typeof** data + data.trim());  
 **var** json = JSON.parse(data.trim());  
 console.log( " " + **typeof** data.status);  
 **if**(json.status){  
 Vue.set(vMain.bookTypes[type][index], 'amount', amount - 1);  
 }  
 alert(json.message);  
 }  
 });  
 }  
 }**else**{  
 alert("Oops!无书可借啦，等别的小伙伴归还吧！");  
 }  
}

后台数据库操作代码

// 查询books\_lent表，若该用户已借阅该书则不能再次借阅  
 ResultSet rs1 = connector.query("select \* from books\_lent where id=" + id + " and owner\_id=" + userId + ";");  
 if(rs1.next()){  
 response.getWriter().write("{\"message\":\"已借阅此书，请不要重复借阅！\", \"status\": false}");  
 }else {  
 PreparedStatement ps = conn.prepareStatement("INSERT INTO books\_lent VALUES (?, ?, ?, ?, ?, 'false');");  
 ps.setInt(1, id);  
 ps.setString(2, name);  
 ps.setInt(3, userId);  
 ps.setString(4, initDate);  
 ps.setString(5, deadline);  
 ps.executeUpdate();  
 Statement st = conn.createStatement();  
 st.executeUpdate("UPDATE books SET isAvailable=" + amount + " where id=" + id + ";");  
 response.getWriter().write("{\"message\": \"借阅成功，请前往个人中心查看！\", \"status\": true}");  
 }

3.4.7 读者续借模块

设定读者只能续借一次，一次续借的时间为30天，若读者已经续借过一次之后则不能再次续借。

续借通过读取数据库借阅表中的借书时间戳，在此基础上加上60天的时间，时间由自1970.01.01开始的秒数存储。

续借标识位置true，表示已经续借，无法再次续借。

续借功能逻辑上的判断都在前端进行

extend: **function** (e, id, name, date, isExtended) {  
 **if**(isExtended === 'false'){  
 **var** userId = getCookie('userId');  
 **if**(confirm('确认延长《' + name +' 》的借阅时间？')){  
 $.ajax({  
 type: 'GET',  
 data: {name: encodeURI(encodeURI(name)), id: id, userId: userId, deadline: date.getTime() + (30 \* 24 \* 60 \* 60 \*1000)},  
 url: '../jsp/extendBooks.jsp',  
 success: **function** (data) {  
 **var** json = JSON.parse(data.trim());  
 **var** date = **new** Date(parseInt(json.date));  
 alert("《" + json.name + "》续借成功，请在" + date.getFullYear() + "年" + (date.getMonth() + 1) + "月" + date.getDate() + "日之前归还！");  
 getAllLentBooks();  
 }  
 })  
 }  
 }**else** {  
 alert('请注意，只能延长归还时间一次，请按期归还书本后再借阅！');  
 }  
  
}，

//服务器

**<%** int id = Integer.*parseInt*(request.getParameter("id"));  
 int userId = Integer.*parseInt*(request.getParameter("userId"));  
 String name = URLDecoder.*decode*(URLDecoder.*decode*(request.getParameter("name"), "utf-8"), "utf-8");  
 String deadline = request.getParameter("deadline");  
 Connector connector = new Connector();  
 connector.connect();  
 Connection conn = connector.getConn();  
 try {  
 PreparedStatement ps = conn.prepareStatement("UPDATE books\_lent SET deadline='" + deadline + "',isExtended='true' WHERE id=" + id + " and owner\_id=" + userId);  
 ps.executeUpdate();  
 response.getWriter().write("{\"name\": \"" + name + "\", \"date\": \"" + deadline + "\"}");  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
**%>**

3.4.8 管理员入库模块

管理员入库、出库模块实现较为简单。

入库则读取前端入库表中的书籍信息，通过ajax传给后端后执行sql的插入语句将记录插入到数据库书籍信息表中。

出库则是在管理员确认的情况下数据库书籍信息表中的相关记录删除。

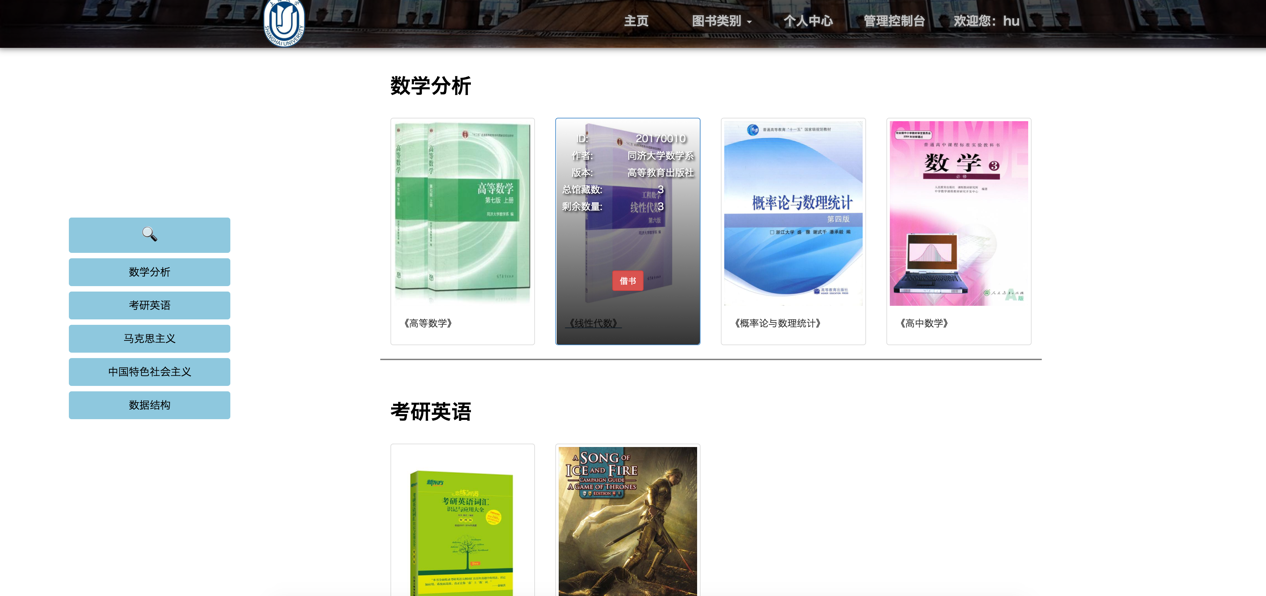




3.5 项目展示



图书馆主页



书籍展示



读者个人中心



图书管理控制台



图书入库

四、项目感想

在刚得知老师对项目具体做什么没有硬性要求的时候是有些庆幸的，这样我们自我发挥的空间更大。既然有这样一个练手的机会，我们决定做一个有前后端交互、数据库操作的项目。在前期选定项目内容的时候我们恰巧看到了上海图书馆的官网易用性并不是非常的高，于是我们决定试着做一个电子图书馆系统，拥有图书浏览、读者借书、还书，管理员管理书籍等基本功能。

敲定项目内容后技术选型阶段，期初我们打算用swing做一款桌面软件，但是发现swing繁琐的组件与布局使得想要做一款界面略优雅一些的图书馆系统过于困难，于是我们决定采用B/S模式来实现。因为之前有web开发经验，前端界面实现起来会轻松很多。

我们设计的图书馆系统主页由相当数量的图片，考虑到当图片多了之后我们在前端对书本进行查找和排序的时候如果之间操作dom将会带来极大的渲染压力，以及考虑到后期系统的可维护性，于是我们决定采用MVC模式进行开发。

综合比较了时下流行的前端框架，我们选用了vue+boostrap+jquery，Vue很好的实现了数据与显示的分离。在开发时只需要专注于功能逻辑与数据部分，而不需要再直接操作dom，在进行查找和排序时vue将自动尽可能的重用dom元素来减少元素的渲染，进而极大的提高了网站运行效率。数据与显示的分离也使得在后期维护或者修改数据的时候无需再修改html，vue实例的更新将自动触发dom树的更新。

在前端的显示方面我们主要采用了bootstrap，boostrap中的网格布局让我们很方便的划分网页布局，从而减少了很多由于dom元素布局不合理带来的对齐、浮动等问题。

在项目开发过程中，遇到的最棘手的问题是中文乱码的问题，无论是在数据库存储数据或是前后端通信的时候，涉及到字符编码的地方很多，只要有一处编码不一致就会出现乱码。我们数据库选用的是mysql，通过gui软件navicat来操作。在最初我们通过navicat在数据库中存入原始书籍数据，然后从前端页面读取所有书籍的时候发现所有的中文都是乱码，于是开始分别排查html页面、jsp页面编码、ajax的请求头编码、mysql字符集，发现都是一致的，但是读取的数据却一直是乱码，网上搜索到的可能错误都排查了一遍依然无济于事。之后排查了很久在偶然在terminal里运行mysql，发现凡是通过navicat插入的中文查询出来都是乱码，这是才发现navicat软件本身也存在中文字符集的选择问题。

在这次项目中因为涉及到的书籍数据是不定的，在项目主页该怎么动态的显示所有书籍的算法上也经过了很久的思考。要实现网站的动态显示，我们选择把每一条书籍记录抽象成一个书本模型，存储在vue实例的一个数组属性中，每次刷新网页时向后台发起一次请求，后台返回完整的数据库数据生成书本模型数组，再通过遍历该数组动态渲染dom元素。

这次项目是个很好的机会去学习jsp和vue，也让我更深入的体会了mvc框架与数据绑定的精髓，加强了团队成员之间的合作，收获满满。

五、成员分工

胡圣翔：功能实现、报告及ppt

段 婕：功能实现、报告及ppt

柴名扬：功能实现、报告及ppt