# 设计资料

## 1 图书管理系统数据库及关系

防止在数据库中使用sql的关键字 比如order、class都是关键字

书籍----多个书籍可以添加到一个购物车-----多个购物车记录可以添加到一条订单；



部分内容字段设置不合适单独调整，比如价格应该是双精度的，数据库设置为int整型。

## 2 图书管理系统页面布局及逻辑

分为客户端和管理端两种

客户端用于客户浏览热门书籍，点击购买书籍到购物车，点击结算书籍到订单，点击订单确认以完成派单----服务管理端完成派单派货----点击订单完成以结账----（暂不实现->）完成的订单进入历史订单记录；

管理端用户主要涉及到添加书目及相关标签信息，自动可以更新页面展示效果；订单方面的管理派单功能。

有了逻辑，页面布局自动就出来了！

# 技术要点

## 1.建立数据库

已经完成

## 2.引入若依系统

### 2.1 修改druid.yml配置文件(数据库配置)

url:jdbc:mysql://localhost:3306/{数据库名称}?....

username: root 自己的数据库用户名  
password: root12345 自己的数据库用户密码

**\*补充：**除业务数据表外，还应该引入若依系统自带的基础数据表；

运行根目录下的sql文件夹内的文件 quarz\ry\_....sql

### 2.2 在application.yml中确认端口号80是否被占用，被占用了就换其他端口，一般没有跑服务器的话是没有占用的

port: 80

profile: D:/ruoyi/uploadPath 确认文件路径，自己的操作系统是否匹配/以后资源文件可以统一从这里加载、比如图书对应的图片信息

### 2.3 调试运行程序

在maven组件中先编译compile，有助于解决部分编译器报错问题，同时会生成一个target文件，该文件夹内所有资源都是真正在服务器里运行的文件

#### 2.3.1生成页面

##### 第一步：生成和修改代码。

根据功能我们要生成业务对应的页面，客户端生成一批（客户端包括书籍展示页面，购物车页面，订单管理页面），管理端生成一批（书籍管理页面，订单管理页面）

为使生成的页面，转入到我们需要的目录条目下，我们设置一个“网上书城”的目录，将所有生成的代码挂载在网上书城下面

##### 第二步：数据库支持

生成后的rar解压后对三个sql文件全部在数据库中执行一次，其效果就是在系统菜单下面将相关页面生成在对应的目录子级，但是只有url信息，没有后台服务器的支持。

##### 第三步：服务器支持。

将rar解压后的main文件与源码合并，因为目录结构一致，所以只会在相应位置添加所需的文件结构。由于target没有重新编译，整个项目没有从硬盘中重载，以更新修改的文件结构，所以直接拷贝进去是没有效果的，需要两步：1.停止项目运行，并重新加载，为确保没问题，再加上maven重构（可以发现没有编译，target中仍旧没有三个新增的文件）2.编译完成后即可成功运行在服务器端，该步骤可以与run这个三角标符号（自带编译功能）合并

**综上，目前三个页面的增删改查功能均正常使用**

#### 2.3.2 修改页面布局

目前的页面，是统一的，都是类似管理端的页面，实际上如果不考虑表格实时修改功能，研二的两名同学的任务已经完成。但是布置的任务要求在表格中可以修改信息，即**动态修改功能。**目前是无法动态修改的，这个实例中演示了动态修改，我们需要抄袭他的逻辑和功能。

##### 第一步 使表格可编辑

Demo，其基础资源在resources/demo/table/editable

我们对应修改图书book页面，第一步 table表单的option配置项中增加

onEditableSave: onEditableSave**,//设置为可以编辑**

editable:{

type: "select"**,** //编辑框的类型。

source: [{ value: **1,** text: "画册" }**,** { value: **2,** text: "小说" }**,** {value:**3,**text:"高考"}]**,**

title: "选择图书类别"**,** //编辑框的标题

disabled: false**,** //是否禁用编辑

emptytext: "未定义"**,** //空值的默认文本

mode: "popup"**,** //编辑框的模式：支持popup和inline两种模式，默认是popup

validate: function (value) { //字段验证

if (!$.trim(value)) {

return '不能为空'**;**

}

}

}

function onEditableSave (field**,** row**,** rowIndex**,** oldValue**,** $el) {  
 alert("字段名：" + field + "，当前值：" + row[field] + "，旧值：" + oldValue)**;**}

同时需要引入对应的依赖

<th:block th:include="include :: bootstrap-table-editable-js" />

判断输入的如果是字符串或数值区域判断

if(***Number***(value).toString()=='NaN'){//输入为字符串  
 return '热度必须为数字'**;**}  
if(***Number***(value)>**100**||***Number***(value)<**0**){//热度定义为0-100之间的整数  
 return '热度必须为0-100区间'**;**}

格式化数字为小数点后2位

formatter:function (value**,** row**,** index) {//用于格式化显示内容的  
 //输入为数字,设定格式化为小数点后2位  
 return value.toFixed(**2**)**;**}

##### 第二步 使可编辑内容能够更新到数据库

现在是可以可以修改并在表格中看到展示效果，但一刷新，内容仍旧是原始内容，说明数据库没有实现更新，每次刷新都是从数据库重新读取了一遍数据。

现在要将表单数据取出来，并进行修改。

我们分析这个函数

onEditableSave (field**,** row**,** rowIndex**,** oldValue**,** $el)

field表示当前修改的字段名，row表示当前整行信息，rowindex表示第几行（从0开始算）

我们要修改当前列，应该是执行编辑方法，book的editsave接口带大家看一下

所以请求该接口的地址是ctx(即resources/template/)

@RequestMapping("/system/book")  
public class BookController{

@PostMapping("/edit")

@ResponseBody

public AjaxResult editSave(Book book) {

return toAjax(bookService.updateBook(book))**;**

}

}

Ctx+ ‘system/book’+"/edit"

传入参数Book，根据update sql中的语句

UPDATE `book` SET `name` = '555' WHERE `id` = 2;

Book包括了一堆字段{author: "12412"，classification: "43124"， createBy: null，createTime: null，detail: "34124"，hot: 412，id: 2，imgUrl: "124124"，name: "555"，params: {}，press: "12"，price: 233，remark: ""，searchValue: null，totalPage: null，updateBy: null，updateTime: null}

这些字段只要传入，SET `name` = '555'？？？都会直接用 除了id 这源码如下

update book  
<trim prefix="SET" suffixOverrides=",">  
 <if test="name != null">name = #{name},</if>  
 <if test="price != null">price = #{price},</if>  
 <if test="detail != null">detail = #{detail},</if>  
 <if test="classification != null">classification = #{classification},</if>  
 <if test="hot != null">hot = #{hot},</if>  
 <if test="author != null">author = #{author},</if>  
 <if test="imgUrl != null">img\_url = #{imgUrl},</if>  
 <if test="press != null">press = #{press},</if>  
 <if test="totalPage != null">total\_page = #{totalPage},</if>  
 <if test="remark != null">remark = #{remark},</if>  
</trim>  
where id = #{id}

但是条件查询是通过id查到对应的需要修改的对象

所以假如当前改了一个元素，需要更新该部分信息到数据库中

##### 第三步 完善细节

我们希望图片地址能自动获取配置目录即

ruoyi:  
 profile: D:/ruoyi/uploadPath

配置静态资源地址

spring:  
 *#静态资源路径匹配* resources:  
 static-locations: classpath:/static/,file:/static/,file:D:/ruoyi/uploadPath/

在上述配置好的静态资源地址下会查找相应资源

<img src="/bookstore/bookImg/1.jpg" style="max-height:10vh;">'

相当于D:/ruoyi/uploadPath/bookstore/bookImg/1.jpg

就找到并渲染出该图片了

#### 2.3.3实现特定功能，热搜书籍查询

##### 第一步：创建链接

即目录子级点击后图书推荐 请求地址为/system/book/selectbooks

##### 第二步：创建页面html

##### 第三步：创建controller

其中的getmaping地址

@RequestMapping("/system/book")  
public class BookController{

@GetMapping("/selectbooks")  
 public String selectbooks(ModelMap mmap)  
 {  
 //推荐热度为60以上的书籍  
// List<Book> books= bookService.selectHotBookList();  
 return prefix + "/selectbooks"**;** }}

/system/book+/selectbooks=/system/book/selectbooks

##### 第四步：渲染当前页面

##### 第五步：获取我们需要的热门图书即热度60以上的图书，该方法需要手写

#### 2.3.4 实现特定功能，书籍点击加入

##### 第一步：引用插入购物车接口

/system/cart/add==CTX+ system/cart/add

@RequestMapping("/system/cart")  
public class CartController{

@PostMapping("/add")  
@ResponseBody  
public AjaxResult addSave(Cart cart)  
{  
 return toAjax(cartService.insertCart(cart))**;**}}

##### 第二步：前端页面点击按钮并获取信息

<img src="url" style="max-height:10vh;">

<a onclick="addToCart('+id+')">加入购入车</a>

##### 第三步：前端页面点击函数，用于调用插入函数的接口

//写添加书籍到购物车的逻辑  
function addToCart(bookId){  
 //就是执行一条插入语句  
 //因此我们根据书籍信息，添加数量为1的对象到购物车中  
 var addTime=new ***Date***()  
 var price**,**state=**0,**num=**1  
 *hotbooks***.forEach((value**,** index**,** array) => {  
 if(value.id==bookId){  
 price=value.price  
 }  
 })  
 var data={

bookId:bookId**,**addTime:addTime**,**price:price**,**

state:state**,**num:num}//构建需要写入数据库的购物车信息

//拿到我们要插入到购物车中的数据信息后，调用插入购物车的数据库接口,添加成功  
 $.operate.post(***ctx***+ 'system/cart/add'**,**data)  
}

#### 2.3.5 实现特定功能，订单处理

拿到任务写逻辑，逻辑同书籍管理，主要是实现表格内的可编辑

##### 第一步：订单管理

订单已经成立，主要是对订单状态进行判断，0正在结算，1结算成功待发货，2发货中，3发货成功；

(逻辑复杂，暂时不实现)且当状态为0的订单超过30min没计算自动删除该订单，同步恢复购物车相关商品状态

报错是说字符串传入到后端 但是后端要的是int、Long整型

同前，1.引入依赖

2.修改保存订单的方法/接口 /system/orders /edit

@RequestMapping("/system/orders")

@PostMapping("/edit")  
@ResponseBody  
public AjaxResult editSave(Orders orders)  
{  
 return toAjax(ordersService.updateOrders(orders))**;**}

##### 第二步：订单状态更新

可以更新订单状态等信息，同书籍部分

##### 第三步：实现订单服务人员的信息自动获取

要求存入数据库的是订单服务人员的id，但是显示的却是订单人员的用户名

用到一个formatter函数，可以将数据信息进行处理

首先是页面加载时就应该加载该服务人员的信息进入页面

public String orders(ModelMap mmap)//将所需信息存入到页面信息载体Modelmap中  
{  
 //需要当前登录管理端的用户信息，以下方法即取得当前用户的信息了  
 User user = getSysUser()**;** //将该用户信息直接封装到前端  
 mmap.put("user"**,**user)**;** return prefix + "/orders"**;**}

然后是在前端实现渲染和编辑

var ***user***=[[${user}]]//获取从后端封装传入的用户信息

//option内添加

editable :{  
 type: "select"**,** //直接选用选择框的方式，可以避免格式转换，即存入数据库的是value 显示的是text  
 source: [{ value:***user***.userId**,** text: ***user***.userName }]**,** title: "选择订单状态"**,** //编辑框的标题  
 disabled: false**,** //是否禁用编辑  
 emptytext: ***user***.userName**,** //空值的默认文本  
 mode: "popup"**,** //编辑框的模式：支持popup和inline两种模式，默认是popup  
 validate: function (value) { //字段验证  
 if (!$.trim(value)) {  
 return '不能为空'**;** }  
 }  
}

##### 第四步：更多逻辑及完善操作

订单新创建没结算的状态为0，结算后状态为1，当订单处于状态0超过半小时，则需要将购物车中相关产品的状态恢复为0，并删除该订单。

1. 通过post方法调用查询订单list的接口，获得了所有的list

$.post(***prefix*** + "/list"**,**{}**,**function(res){  
 ***console***.log(res.rows)  
})

2.对所有的list进行判断筛选

//比较订单状态是否超过半小时还处于0  
function checkOrders(){  
 //函数要检查所有当前订单  
 //首先是获取到所有的订单信息  
 $.post(***prefix*** + "/list"**,**{}**,**function(res){  
 //遍历查到的所有的订单  
 res.rows.forEach((value**,** index**,** array) => {  
 var state=value.orderState//订单状态  
 var startTime=new ***Date***(value.startTime).getTime()//开始产生订单的时 并获取时间戳  
 var nowTime=new ***Date***().getTime()  
 var periodTime = (nowTime-startTime)/**1000** //获得该时间与当前时间的秒数  
 ***console***.log("分钟数"**,**periodTime/**60**)  
 if(periodTime>**60**\***30**){//间隔时间超过30min，执行删除 当前订单的操作  
 $.post(***prefix*** + "/remove"**,**{ids:value.id}**,**function(res){  
 $.modal.msgError('订单超时30min，已自动删除，如重新下单请到购物车中重新下单！')//弹出一个自动关闭的提示  
 $.table.refresh()  
 //同时还要对购物车中相关订单内容状态进行恢复  
 //即订单中value.bookIds里面的所有购物车内容都要更新其state为0  
 value.bookIds.split(",").forEach((val**,**i**,**arr)=>{  
 //bookId=val,假设添加到订单的是购物车的记录的id  
 var data={state:**0,**id:val}  
 $.post(***ctx***+'system/cart/edit'**,**data)//更新相应的购物车  
 }) //根据，拆分为数组  
 })  
 }  
 })  
 })  
}

订单进入后，原购物车中的书籍状态变成了1；订单结算后原购物车中的书籍状态变成了2；订单收货后，原购物车相应书籍应当直接删除消失，不涉及恢复到购物车的功能了。

//根据订单状态调整购物车中相关的记录状态  
function changeCartSateByOrder(orderId){  
 //订单进入后，原购物车中的书籍状态变成了1  
 $.post(***prefix***+"/selectById"**,**{id:orderId}**,**function(res){  
 //查询结果res=order，即获得了自己想要的这个order  
 //判断该order的状态  
 switch (res.orderState) {  
 case **0**:  
 //订单状态为0，原购物车中书籍状态更新为1  
 res.bookIds.split(",").forEach((val**,**i**,**arr)=>{  
 //bookId=val,假设添加到订单的是购物车的记录的id  
 var data={state:**1,**id:val}  
 $.post(***ctx***+'system/cart/edit'**,**data)//更新相应的购物车  
 }) //根据，拆分为数组  
 break**;** case **1**:  
 //订1车2  
 res.bookIds.split(",").forEach((val**,**i**,**arr)=>{  
 //bookId=val,假设添加到订单的是购物车的记录的id  
 var data={state:**2,**id:val}  
 $.post(***ctx***+'system/cart/edit'**,**data)//更新相应的购物车  
 }) //根据，拆分为数组  
 break**;** case **3**:  
 //订2车-  
 $.post(***ctx***+'system/cart/remove'**,**{ids:res.bookIds})  
 break**;** default:  
 break**;** }  
 })  
}