一、项目介绍

项目名称：springboot+vue实现商城系统。

项目简介：本项目采用前后端分离的结构，实现简单的商城系统网站需求：

网站由五个身份模块构成，分别是游客、顾客、商家、配送员和网站管理员。提供顾客账号、注册商家账号、注册配送员账号的注册与登录、账号管理。

未登录的游客和已登录的顾客可以浏览首页商品、模糊搜索商品、浏览商家列表，模糊搜索商家、查看某个商家的所有商品。顾客可以将商品加入购物车，未登录的游客将商品添加到购物车时会提示登录并打开顾客的登录面板。顾客可以进行账号管理，对账号的昵称和密码进行修改。昵称不可与当前昵称重复。修改密码后会退出登录状态并打开顾客登录面板要求重新登录。顾客可以对购物车和订单进行查询和处理。在购物车中，顾客添加的商品将按商品归属的商家进行分类。顾客可以修改购物车中任一商品的数量，如果数量调整为零，提示并将商品移除购物车。顾客也可以将该商家名下的所有商品一键移除或进行支付，支付时将进行商品库存数量的检测，如果顾客的商品购买数量大于商品在售数量，支付将中止并提示超量购买，购买时顾客可以选择邮递方式：到店购买或快跑配送，购买后如果在售商品数量为零将自动下架。顾客可以对订单状态进行查看，一共有六种状态：待发货、待取货、已发货、配送中、已送达、已完成。如果订单为待取货或已送达状态，确认收货后，顾客可以将订单状态转变为已完成。

商家可以对自家商品管理，包括新增商品、修改商品、上架商品、下架商品、删除商品，支持批量操作。商家可以查看、上传和修改商品的封面，设定定时上架时间，到达时间商品将自动上架。商家可以对顾客的订单进行处理，如果顾客选择到店购买，商家可以将订单转为待收货状态，如果顾客选择快跑配送，商家可以将订单发布到配送员的订单广场。商家可以注册和登录账号，并对账号经进行管理，被管理员封禁的商家无法登录账号。

配送员登录后可以接受所有商家发布的配送订单，订单信息包括商家信息、订单id和订单状态等。配送员可以查看订单广场和自己接受的订单，在完成配送后将订单确认为已送达状态，顾客可以在确认收货后完成订单。配送员可以注册和登录账号，修改账号的昵称和密码，修改密码成功后会退出当前账号并提示重新登录。

管理员通过特定链接进行登录，输入管理员密码后将密码发送到后台，后台将用户输入的密码与配置文件中的密码进行比较，如果密码正确，将用列表展示所有商家信息。管理员可以封禁和解禁商家账号，封禁后商家无法登录账号，只能通过联系管理员解除封禁。

相关技术：后台：springboot、mybatis、mysql、前端：vue2、element、axios、router。

二、项目实现

环境搭建

前端使用vue2框架，配置端口8080，使用router，vuex，axios，element依赖。

后台使用springboot框架，配置端口8181，使用lombok，springmvc，mybaits，mysqldriver起步依赖，使用hutool工具类依赖。插件：mybatis、lombok。

基本思路

项目运行时，进入端口8080的前端网页，在与网页交互的过程中，将数据请求发送到端口8181的后台，后台通过对数据库操作，返回数据给前端，前端获得数据后对数据进行处理，然后重新渲染出网页。

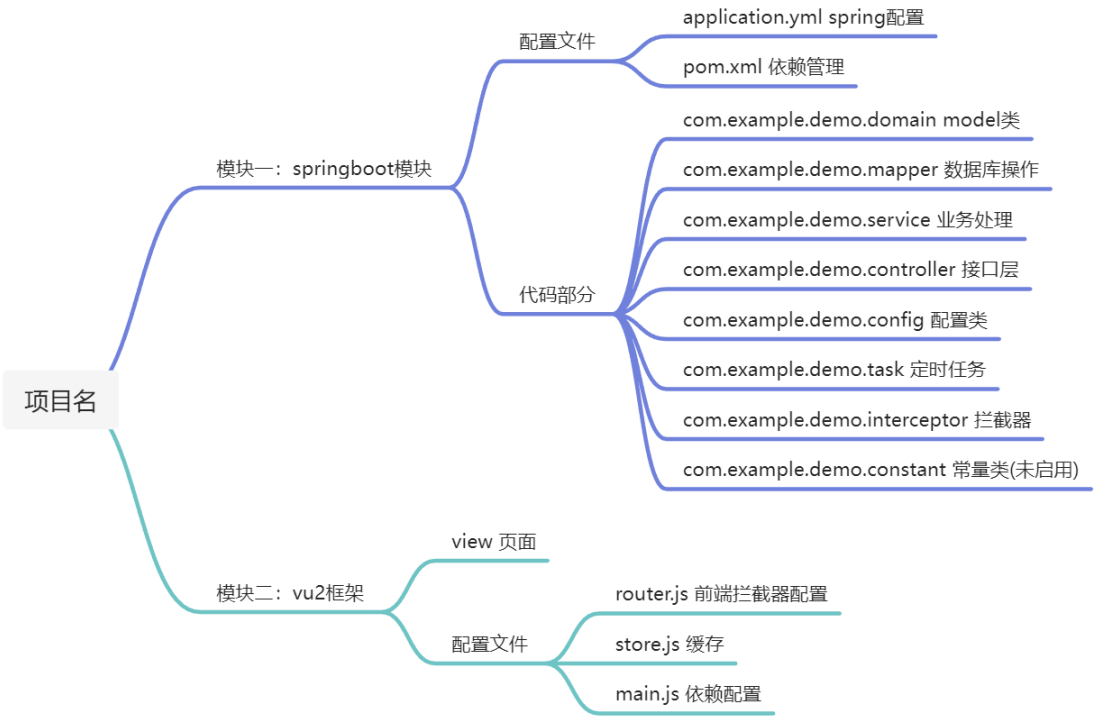
前端采用mvvm思想，将页面数据同脚本数据进行绑定，进而随时刷新页面数据和脚本数据。前端进行简单的数据验证，确认后调用后台的接口。

后台采用mvc思想，使用lombok可以很快完成实体类的编写，实体类分为两类，第一类是直接与表的一行数据对应的实体对象，第二类是根据前端页面展示需求而对第一类实体类进行封装的包含多个表的若干字段的实体。mapper类则是对数据库进行分步的简单操作，将实体类的信息用于数据库中或将数据库中的数据提取到类中。实体类和mapper类均被注解为@Repository对象添加进spring的对象池中。接着是service层对dao层进行调用并包含一些复杂的业务处理代码，可能会调用多步的mapper对象进行数据库的批量数据操作。controller层中提供对外的接口和service的调用。

项目文件结构

项目名称：springbootvue3（临时）

文件树：



表与实体类设计

商品信息相关：

表：goods，src，storekeeper

实体：Goods，Src，StoreKeeper，ShowingGoods

```
-- 存储商品的基本信息，包括商品唯一id，库存，销售量，在售数量，所属商家，商品图资源，商品名，自动上架时间。对应Goods实体类。
CREATE TABLE `goods` (
  `gid` int NOT NULL AUTO_INCREMENT COMMENT '商品的唯一id，goods表的自增主键',
  `gsave` bigint DEFAULT '0' COMMENT '商品库存量',
  `gsales` bigint DEFAULT '0' COMMENT '销售量',
  `sid` int DEFAULT NULL COMMENT '所属商家',
  `state` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '商品状态，包括仓库中、已删除、已上架、封禁中',
  `srcid` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '图片资源id',
  `gname` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT NULL COMMENT '商品名',
```

```
  `gonlinenum` int DEFAULT '0' COMMENT '在售数量',
  `time` datetime DEFAULT NULL COMMENT '自动上架时间，如果没有设定自动上架时间值为null',
  PRIMARY KEY (`gid`)
) ENGINE=InnoDB AUTO_INCREMENT=174 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;

-- 主要用于存储商品的图片资源，包括资源唯一id，资源的url地址和资源的绝对路径。对应Src实体类。
CREATE TABLE `src` (
  `srcid` int NOT NULL AUTO_INCREMENT COMMENT '资源唯一标识',
  `srcurl` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT
NULL COMMENT '资源的url地址',
  `srcname` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL COMMENT '资源在硬盘中的绝对路径',
  PRIMARY KEY (`srcid`)
) ENGINE=InnoDB AUTO_INCREMENT=192 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;

-- 存储商家账号信息。对应StoreKeeper实体类。此外，goods表、src表和storekeeper表共同构成了
一个完整的商品信息实体类ShowingGoods。
CREATE TABLE `storekeeper` (
  `sid` int NOT NULL AUTO_INCREMENT COMMENT '商家id',
  `sname` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT
NULL COMMENT '商家名',
  `spassword` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci
DEFAULT NULL COMMENT '商家密码',
  `isban` tinyint(1) DEFAULT '0' COMMENT '商家是否被封禁',
  PRIMARY KEY (`sid`)
) ENGINE=InnoDB AUTO_INCREMENT=114 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Goods {
    private Integer gid;
    private Integer gsave;
    private Integer gsales;
    private Integer sid;
    private String state;
    private Integer srcid;
    private String gname;
    private Integer gonlinenum;
    private Date time;
}

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Src {
    private Integer srcid;
    private String srcurl;
    private String srcname;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
```

```
public class StoreKeeper {
    private Integer sid;
    private String sname;
    private String spassword;
    private Boolean isban;
}

@Data
public class ShowingGoods {
    private String srcurl;
    private String sname;
    private Goods goods;
}
```

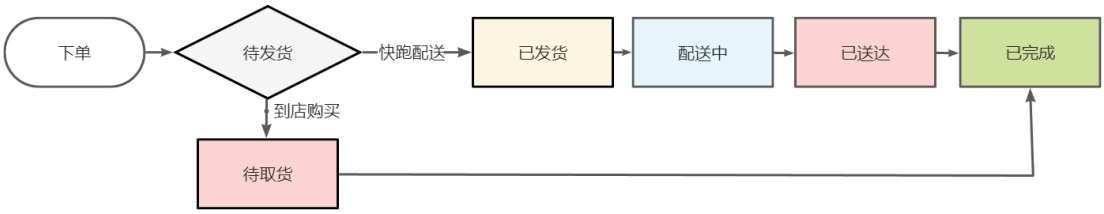
订单信息相关

表：顾客信息表customer，配送员信息表delivery，商家信息表storekeeper（略），订单orders，订单包含的商品orderson

实体：Customer，Delivery，StoreKeeper，Orders

一条完整的订单包括订单号、顾客信息、配送员信息（如果是到店购买则为Null），订单中商品的商家的信息，配送状态、配送方式（是否配送）、包含的商品基本信息以及对应的数量。由多个表的字段构成一条完整的字段信息，这些表包括customer、delivery、storekeepr、orders、orderson、goods。

订单由多个身份进行操作。顾客下单，将购物车项转为订单项，并通过取货方式的选择确定了一条订单完成路径。在商品待取货或已送达时顾客可以确认收货，完成整个顶单。商家对待发货的订单进行处理，根据用户需求将订单状态转变为已发货或者待取货。已发货的订单将发布到订单大厅，配送员接受订单后订单将从已发货的状态转变为配送中，配送员配送完成后订单状态改为已送达，最后等待顾客确认收货完成整个订单。




```
-- 订单
CREATE TABLE `orders` (
  `oid` int NOT NULL AUTO_INCREMENT COMMENT '订单号',
  `ostate` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci DEFAULT
NULL COMMENT '顶单状态，包括待发货，已发货，待取货，配送中，已送达和已完成。',
  `did` int DEFAULT NULL COMMENT '配送员id',
  `cid` int DEFAULT NULL COMMENT '顾客id',
  `sid` int DEFAULT NULL COMMENT '商家id',
  `isdeli` tinyint(1) DEFAULT NULL COMMENT '是否配送',
  PRIMARY KEY (`oid`)
) ENGINE=InnoDB AUTO_INCREMENT=38 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_unicode_ci;
-- 注：该表没有主键，一行写一列商品信息以及其归属的订单号，辅助orders储存完整的订单信息
CREATE TABLE `orderson` (
  `oid` int DEFAULT NULL COMMENT '订单号',
  `gid` int DEFAULT NULL COMMENT '商品id',
```

```
  `oamount` bigint DEFAULT NULL COMMENT '购买数量'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Customer {
    private Integer cid;
    private String cname;
    private String cpassword;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Delivery {
    private Integer did;
    private String dname;
    private String dpassword;
}

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Orders {
    private Integer oid;
    private String ostate;
    private Integer did;
    private Integer cid;
    private Integer sid;
    private Boolean isdeli;
}
```

其他表与实体类

表：cart，shopingcart，类似orders和orderson存储购物车项。涉及表customer、storekeeper等。

实体类：cart，showingcart

具体代码（参考orders和orderson）。

部分代码实现

主页展示

1. 获取所有商品信息并展示

前端调用接口代码

```
axios.get('http://localhost:8181/goods/searchgoods?gname=' +
this.search).then(resp => this.goodsData = resp.data);
```

后台相关代码：

Controller

```
//GoodsController
    /**
     * 搜索商品（模糊搜索）
     * 如果商品名为空则匹配所有商品
     * @param 商品名（模糊搜索）gname
     * @return 商品列表
     */
@GetMapping("searchgoods")
public List<ShowingGoods> searchGoods(String gname) {
    return goodsService.searchGoods(gname);
}
```

Service

```
//GoodsService
@Autowired
private GoodsService goodsService;

public List<ShowingGoods> searchGoods(String gname) {
    return goodsMapper.searchByName(gname, "已上架");
}
```

Dao

```
//Mapper
@Select("select * from storekeeper,goods,src \n" +
        "where goods.srcid=src.srcid \n" +
        "  and state=#{state} \n" +
        "  and storekeeper.sid=goods.sid\n" +
        "  and goods.gname like concat('%',#{gname},'%')")
@Results({
        @Result(column = "srcurl", property = "srcurl"),
        @Result(column = "sname", property = "sname"),
        @Result(column = "gid", property = "goods.gid"),
        @Result(column = "gsave", property = "goods.gsave"),
        @Result(column = "gsales", property = "goods.gsales"),
        @Result(column = "sid", property = "goods.sid"),
        @Result(column = "state", property = "goods.state"),
        @Result(column = "srcid", property = "goods.srcid"),
        @Result(column = "gname", property = "goods.gname"),
        @Result(column = "gonlinenum", property = "goods.gonlinenum"),
        @Result(column = "time", property = "goods.time"),
})
List<ShowingGoods> searchByName(String gname, String state);
```

```
//Goods：面向数据库的实体
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Goods {
    private Integer gid;
    private Integer gsave;
    private Integer gsales;
    private Integer sid;
    private String state;
```

```
    private Integer srcid;
    private String gname;
    private Integer gonlinenum;
    private Date time;
}
```

```
//ShowingGoods：面向前端请求的实体
@Data
public class ShowingGoods {
    private String srcurl;
    private String sname;
    private Goods goods;
}
```

2. 获取店铺信息并展示（类似，略）

模糊搜索

模糊搜索商家商品，模糊搜索首页商品（略），模糊搜索店铺（略）

前端代码

```
if (this.activeIndex === '1') {
  state = '已上架';
} else if (this.activeIndex === '2') {
  state = '仓库中';
}
axios.get('http://localhost:8181/storekeeper/searchgoods?gname=' +
this.searchContext + '&state=' + state + '&sid=' +
this.storeKeeper.sid).then(resp => this.goodsData = resp.data)
```

后台代码

```
@GetMapping("searchgoods")
public List<ShowingGoods> searchGoodsByGname(String gname, String state, Integer
sid) {
    return storeKeeperService.findGoods(gname, state, sid);
}
```

```
@Autowired
private GoodsMapper goodsMapper;

public List<ShowingGoods> findGoods(String gname, String state, Integer sid) {
    return goodsMapper.selectByNamePlus(gname, state,sid);
}
```

```
@Select("select * from storekeeper,goods,src \n" +
        "where goods.srcid=src.srcid \n" +
        "  and state=#{state} \n" +
        "  and storekeeper.sid=goods.sid\n" +
        "  and storekeeper.sid=#{sid}\n" +
        "  and goods.gname like concat('%',#{gname},'%')")
@Results({
        @Result(column = "srcurl", property = "srcurl"),
```

```
        @Result(column = "sname", property = "sname"),
        @Result(column = "gid", property = "goods.gid"),
        @Result(column = "gsave", property = "goods.gsave"),
        @Result(column = "gsales", property = "goods.gsales"),
        @Result(column = "sid", property = "goods.sid"),
        @Result(column = "state", property = "goods.state"),
        @Result(column = "srcid", property = "goods.srcid"),
        @Result(column = "gname", property = "goods.gname"),
        @Result(column = "gonlinenum", property = "goods.gonlinenum"),
        @Result(column = "time", property = "goods.time"),
})
List<ShowingGoods> selectByNamePlus(String gname, String state, Integer sid);
```

实体类代码（参考主页展示）

登录验证

登录前在前端校验表单，符合格式则请求后台接口，后台将获取到的数据与数据库进行比较，返回结果。前端获得登录验证通过后，发送请求到后台获取用户的完整信息，存储到前端缓存中。（由于前后端分离后台读取session为null，所以项目将用户的数据暂时存储于vue框架提供的前端缓存）

前端代码

```
//前端登录验证
customerLogin() {
  // 前端登录验证
  if (this.customer.cpassword === '') {
    this.$message.error("请输入密码");
  } else if (this.customer.cid === '') {
    this.$message.error("请输入用户id");
  } else {
    // 格式转换
    let n = Number(this.customer.cid);
    if (isNaN(n)) {
      this.$message.error("用户id必须为数字");
    } else {
      // 后台登录验证
      this.customer.cid = n;
      axios.post('http://localhost:8181/customer/login',
this.customer).then(resp => {
        if ("登录验证成功" === resp.data) {
          axios.post('http://localhost:8181/customer/getinf',
this.customer).then(resp1 => {
            // 保存本地
            this.customer = resp1.data;
            this.isLogin = true;
            // 保存缓存
            this.$store.commit('saveCustomer', this.customer);
          })
          // 登录成功，关闭弹窗
          this.$message({
            message: resp.data,
            type: 'success'
          });
          this.customerLoginVisible = false;
        } else {
          // 登录失败，错误提示
```

```
            this.$message.error(resp.data);
          }
        })
      }
    }
},
```

```
//前端缓存
import Vue from 'vue'
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    customer: {
      cid:
          window.localStorage
              .getItem('customerinfo' || '[]') == null ? 0 :
JSON.parse(window.localStorage.getItem('customerinfo' || '[]'))
              .cid,
      cname:
          window.localStorage
              .getItem('customerinfo' || '[]') == null ? '' :
JSON.parse(window.localStorage.getItem('customerinfo' || '[]'))
              .cname,
      cpassword:
          window.localStorage
              .getItem('customerinfo' || '[]') == null ? '' :
JSON.parse(window.localStorage.getItem('customerinfo' || '[]'))
              .cpassword

    },
  },
  getters: {
  },
  mutations: {
    saveCustomer(state, customer) {
      state.customer = customer;
      window.localStorage.setItem('customerinfo', JSON.stringify(customer))
    },
  },
  actions: {
  },
  modules: {
  }
})
```

后台代码

```
/**
 * 账号管理：登录验证
 *
 * @param customer
 * @return
 */
```

```
@PostMapping("login")
public String tryLogin(@RequestBody Customer customer) {
    return customerService.tryLogin(customer);
}

/**
 * 账号管理:获取账号信息
 * 登录后获取用户信息.用于存储到前端缓存.
 *
 * @param customer
 * @return
 */
@PostMapping("getinf")
public Customer getInf(@RequestBody Customer customer) {
    return customerService.getInf(customer);
}
```

```
//后台验证
public String tryLogin(Customer customer) {
    Customer c = customerMapper.findById(customer.getCid());
    if (c == null) {
        return "用户不存在";
    }
    if (!c.getCpassword().equals(customer.getCpassword())) {
        return "密码错误";
    }
    return "登录验证成功";
}

public Customer getInf(Customer customer) {
    Customer c = customerMapper.findById(customer.getCid());
    if (c == null || (!c.getCpassword().equals(customer.getCpassword()))) {
        return null;
    }
    return c;
}
```

```
@Select("select * from customer where cid=#{cid}")
public Customer findById(Integer cid);
```

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Customer {
    private Integer cid;
    private String cname;
    private String cpassword;
}
```

支付订单

支付时顾客选择送货方式,支付前进行商品库存检测,支付后对购物车表、订单表、商品表进行操作。

前端代码

```
axios.post('http://localhost:8181/customer/detectamount',
this.shopCartsData[this.shopIndex].carts).then(resp => {
  if (resp.data.length != 0) {
    for (const index in resp.data) {
      this.$message.error('支付失败! ' + resp.data[index]);
    }
  } else {
    //更新远程数据
    axios.get("http://localhost:8181/customer/pay?cid=" + this.customer.cid +
"&sid=" + this.shopCartsData[this.shopIndex].shop.sid + "&way=" + way);
    // 更新本地数据
    this.shopCartsData.splice(this.shopIndex, 1);
  }
  this.payVisible = false;
})
```

后台代码

Controller层

```
/**
 * 支付检测
 * 检测是否符合支付条件-检测在售数量是否充足.
 * 支付过程:支付检测->支付操作
 * @param showingCarts
 * @return 返回字符串列表, 存储库存不足物品对应的文字提示.使用:如果length=0,则库存充足,如果
length!=0则输出字符串.
 */
@PostMapping("detectamount")
private List<String> detectAmount(@RequestBody List<ShowingCart> showingCarts) {
    return customerService.detectAmount(showingCarts);
}

/**
 * 支付操作
 * 和删除商家订单removecarts差不多，多了一步记录在order和orderson表上。
 * @param cid
 * @param sid
 * @param receiveWay
 */
@GetMapping("pay")
public String pay(Integer cid, Integer sid, @RequestParam("way") String
receiveWay) {
    return customerService.pay(cid, sid, receiveWay);
}
```

Service层

```
    public List<String> detectAmount(List<ShowingCart> showingCarts) {
        List<String> result = new LinkedList<>();
```

```
        for (ShowingCart showingCart : showingCarts) {
            Goods byGid = goodsMapper.findByGid(showingCart.getGid());
            if (showingCart.getOamount() > byGid.getGonlinenum()) {
                result.add("商品***" + byGid.getGname() + "***的库存仅为" +
byGid.getGonlinenum() + "件");
            }
        }
        return result;
    }
    public String pay(Integer cid, Integer sid, String receiveWay) {
//        确定购物车中是否有该店铺的商品
        String result = new String();
        List<ShowingCart> carts = cartMapper.findCarts(cid, sid);
        if (carts != null) {
//        添加记录到orders和orderson
            Orders orders = new Orders(null, "待发货", null, cid, sid,
receiveWay.equals("快跑配送"));
            ordersMapper.insert(orders);
            Integer oid = orders.getOid();
            for (ShowingCart s : carts) {
                ordersMapper.insertSon(oid, s.getGid(), s.getOamount());
//                商品在线数量减少，销售量增加。
                Goods byGid = goodsMapper.findByGid(s.getGid());
                byGid.setGonlinenum(byGid.getGonlinenum() - s.getOamount());
                byGid.setGsales(byGid.getGsales() + s.getOamount());
                goodsMapper.update(byGid);
            }
//        删除购物车
            this.removeCarts(cid, sid);
        }
        return result;
    }
```

Dao层

```
//GoodsMapper对商品状态(在售数量,销售量)进行管理
@Select("select * from goods where gid=#{gid}")
Goods findByGid(Integer gid);

@Update("update goods set gname=#{gname},gsave=#{gsave},gsales=#{gsales},state=#
{state},srcid=#{srcid},gonlinenum=#{gonlinenum},time=#{time} where gid=#{gid}")
void update(Goods goods);
```

```
//OrdersMapper新增订单以及项单的子项(订单包含的商品列表)
@Insert("insert into orders values (null,#{ostate},null,#{cid},#{sid},#
{isdeli})")
@Options(useGeneratedKeys = true,keyProperty = "oid",keyColumn = "oid")
void insert(Orders orders);

@Insert("insert into orderson values (#{oid},#{gid},#{oamount})")
void insertSon(Integer oid,Integer gid,Integer oamount);
```

```
//CartMapper对购物车数据进行管理
@Select("select * from shoppingcart s,src,goods g where src.srcid=g.srcid and
cid=#{cid} and g.gid=s.gid and sid=#{sid}")
public List<ShowingCart> findCarts(Integer cid,Integer sid);
```

相关的Model实体类

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Orders {
    private Integer oid;
    private String ostate;
    private Integer did;
    private Integer cid;
    private Integer sid;
    private Boolean isdeli;
}
```

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ShowingCart {
    private Integer cid;
    private Integer gid;
    private String gname;
    private String srcurl;
    private Integer oamount;
}
```

商品管理

在goods表中，一个goods有多种状态，主要有已上架、仓库中和已删除。项目支持批量操作、自动上架和自动下架功能。

Controller

```
/**
 * 商品操作-商家:添加商品
 *
 * @param goods
 * @return
 */
@PostMapping("addgoods/base")
public int addGoodsBase(@RequestBody Goods goods) {
    return storeKeeperService.addGoods(goods);
}

/**
 * 商品操作-商家:修改商品or上下架
 *
 * @param goods
 */
@PostMapping("updategoods")
public void updateGoods(@RequestBody Goods goods) {
```

```
    storeKeeperService.updateGoods(goods);
}

/**
 * 商品操作:
 *
 * @param showingGoods
 */
@PostMapping("updategoodsstate")
public void updateGoodsState(@RequestBody ShowingGoods showingGoods) {
    Goods goods = showingGoods.getGoods();
    goodsService.updateGoods(goods.getGid(), goods.getState());
}
```

Service

```
public int addGoods(Goods goods){
    goodsMapper.insert(goods);
    return goods.getGid();
}

public void updateGoods(Goods goods){
    goodsMapper.update(goods);
}
```

Mapper

```
/**
 * 添加商品
 *
 * @param goods
 */
@Insert("insert into goods values(null,#{gsave},#{gsales},#{sid},#{state},#
{srcid},#{gname},#{gonlinenum},#{time})")
@Options(useGeneratedKeys = true, keyProperty = "gid", keyColumn = "gid")
void insert(Goods goods);

/**
 * 修改商品1
 * 可以和前端直接匹配的数据
 * @param goods
 */
@Update("update goods set gname=#{gname},gsave=#{gsave},gsales=#{gsales},state=#
{state},srcid=#{srcid},gonlinenum=#{gonlinenum},time=#{time} where gid=#{gid}")
void update(Goods goods);
```

实体类参考主页展示。

定时任务

配置类设置定时任务

```
/**
 * 配置定时任务类
 */
@Configuration
```

```
@EnableScheduling
public class TaskConfig {
    @Autowired
    private GoodsTask goodsTask;

    @Scheduled(cron = "0/1 * * * * ?")
    private void updateGoods() {
//        定时上架
        goodsTask.upload();
//        自动下架
        goodsTask.download();
    }
}
```

定时任务类，包括自动上架和自动下架的逻辑操作。读取goods表的time字段判断是否改变商品状态（将商品状态改为'已上架'或'仓库中'）

```
@Repository
public class GoodsTask {
    @Autowired
    private GoodsMapper goodsMapper;

    public void upload() {
        Date date = new Date();
        List<Goods> all = goodsMapper.findAllByTimeNotNull("仓库中");
        for (Goods g : all) {
            if (g.getTime().before(date)) {
                g.setState("已上架");
                Integer temp = g.getGonlinenum();
                g.setGonlinenum(g.getGsave());
                g.setGsave(temp);
                g.setTime(null);
                goodsMapper.update(g);
                System.out.println("定时任务：\n\t事件：商品id" + g.getGid() + "上架
\n\t执行时间：" + LocalDateTime.now());
            }
        }
    }

    public void download() {
        List<ShowingGoods> showingGoods = goodsMapper.searchByName("", "已上架");
        for (ShowingGoods s : showingGoods) {
            if (s.getGoods().getGonlinenum() == 0) {
                s.getGoods().setState("仓库中");
                goodsMapper.update(s.getGoods());
                System.out.println("定时任务：\n\t事件：商品id" +
s.getGoods().getGid() + "下架\n\t执行时间：" + LocalDateTime.now());
            }
        }
    }
}
```

商家部分：设置定时上架的接口实现

Controller

```
/**
 * 商品操作-商家:设置定时上架
 * 和修改商品一起被前端调用.
 *
 * @param gid
 * @param time
 */
@GetMapping("setgoodstime")
public void setGoodsTime(Integer gid, Long time) {
    goodsService.setGoodsTime(gid, time);
}
```

Service

```
public void setGoodsTime(Integer gid, Long time){
    if(time==0||time==null){
        goodsMapper.updateTimeByGid(gid,null);
    }else{
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
        String dateString = formatter.format(time);
        Date newTime = null;
        try {
            newTime = formatter.parse(dateString);
        } catch (ParseException e) {
            System.out.println("GoodsService.setGoodsTime报错啦！！！");
        }
        goodsMapper.updateTimeByGid(gid,newTime);
    }
}
```

Mapper

```
/**
 * 修改商品2
 * 这个和更新商品分开,这里的service处理的是前端传来数据不能自动匹配的.
 * @param gid
 * @param time
 */
@Update("update goods set time=#{time} where gid=#{gid}")
void updateTimeByGid(Integer gid, Date time);
```

文件传输

文件上传的代码写法比较固定，项目通过调用上传接口进行文件上传保存到硬盘中，通过下载文件接口实现前端商品封面图的在线展示。代码中使用了hutool工具类，需要添加hutool依赖。

接口以及io处理。

```
@RestController
@RequestMapping("src")
public class SrcController {

    @Value("${server.port}")
    private String port;
```

```
    private static final String ip = "http://localhost:";

    @Autowired
    private SrcService srcService;

    /**
     * 上传文件
     * @param file
     * @return
     * @throws IOException
     */
    @PostMapping("upload")
    public int upload(MultipartFile file) throws IOException {
        String originalFilename = file.getOriginalFilename();
        String flag = IdUtil.fastSimpleUUID();
        String rootFilePath = System.getProperty("user.dir") +
"\\springbootdemo1\\src\\main\\resources\\files\\" + flag + "_" +
originalFilename;
        FileUtil.writeBytes(file.getBytes(), rootFilePath);
        Src src = new Src(null, ip + port + "/src/" + flag, rootFilePath);
        return srcService.add(src);
    }

    /**
     * 下载文件
     * @param response
     * @param flag
     * @return
     */
    @GetMapping("{flag}")
    public String getFiles(HttpServletResponse response, @PathVariable String
flag) {
        String basePath = System.getProperty("user.dir") +
"\\springbootdemo1\\src\\main\\resources\\files\\";
        List<String> fileNames = FileUtil.listFileNames(basePath);
        String file = fileNames.stream().filter(name ->
name.contains(flag)).findAny().orElse("");
        OutputStream os;
        try {
            if (StrUtil.isNotEmpty(file)) {
                response.addHeader("Content-Disposition", "attachment;filename="
+ URLEncoder.encode(file, "UTF-8"));
                response.setContentType("application/octet-stream");
                byte[] bytes = FileUtil.readBytes(basePath + file);
                os = response.getOutputStream();
                os.write(bytes);
                os.flush();
                os.close();
            }
            return file;
        } catch (Exception e) {
            return "文件下载失败";
        }
    }
}
```

Service和dao层的对src表的操作。存储文件信息。

```
@Service
public class SrcService {

    @Autowired
    private SrcMapper srcMapper;

    public int add(Src src){
        srcMapper.insert(src);
        return src.getSrcid();
    }
    // 未启用的接口
    public Src findBySrcid(String srcid) {
        return srcMapper.findBySrcid(srcid);
    }
}
```

```
@Repository
@Mapper
public interface SrcMapper {

    @Insert("insert into src values(#{srcid},#{srcurl},#{srcname})")
    @Options(useGeneratedKeys = true,keyColumn = "srcid",keyProperty = "srcid")
    public void insert(Src src);

    @Select("select * from src where srcid=#{srcid}")
    Src findBySrcid(String srcid);
}
```

页面编写

在main.js中安装并全局配置element依赖。

```
import ElementUI from 'element-ui';
import 'element-ui/lib/theme-chalk/index.css';

Vue.use(ElementUI);
```

vue页面的编写：根据页面需要选择el-table，el-row、el-dialogue、el-card、el-image、el-button、el-upload等element标签和一些传统html标签、编写css样式、将脚本中axios异步请求得到的数据与页面进行绑定，实现mvvm思想。

一个vue页面由上部分的html代码（包含element标签）和下部分的脚本（包含axios异步请求）构成。下面是简单的代码片段。

template标签中是描述页面的代码.

script标签中包括导入的axios包,和代码主体.代码主体主要编写data中的页面数据、methods中的页面交互和接口请求的业务逻辑以及页面刷新时的初始化方法create。

```
<template>
  <div>
    <!--    导航栏-->
    <div style="background: whitesmoke;height: 60px;margin-left: 100px">
```

```
      <div style="font-size: 40px;margin:30px">{{ sname }} 欢迎您!</div>
    </div>
    <!--    商品信息-->
    <el-row>
      <el-col :span="4" v-for="(o, index) in goodsData" :offset="index%4? 1 : 2"
              style="margin-top: 10px;margin-bottom: 10px">
        <el-card :body-style="{ padding: '10px'}"
style="width:240px;height:350px" shadow="hover">
          <!--          图片-->
          <el-image
              fit="contain"
              style="width: 220px; height: 220px"
              :src="o.srcurl"
              :preview-src-list="[o.srcurl]"></el-image>
          <!--          描述-->
          <div style="padding: 14px;">
            <span>{{ o.goods.gname }}</span>
            <div class="bottom clearfix">
              <time class="time">销售：{{ o.goods.gsales }}  剩余：{{
o.goods.gonlinenum }}</time>
              <el-button type="text" class="button" @click="addToCart(index)">加
入购物车</el-button>
            </div>
          </div>

        </el-card>
      </el-col>
    </el-row>

  </div>
</template>

<script>
import axios from "axios";

export default {
  name: "ShopVies",
  data() {
    return {
      // 商家名，用于导航栏的设计
      sname: '',
      // 界面全局标记登录状态
      isLogin: false,
      // 数据实体
      cart: {
        cid: 0,
        gid: 0,
        oamount: 0
      },
      customer: {
        cid: '',
        cname: '',
        cpassword: ''
      },
      goodsData: []
    }
  },
  methods: {
```

```
    addToCart(index) {
      // 添加商品到购物车
      if (this.isLogin) {
        this.cart.cid = this.customer.cid;
        this.cart.gid = this.goodsData[index].goods.gid;
        this.cart.oamount = 1;
        axios.post('http://localhost:8181/customer/addToCart',
this.cart).then(resp => {
          this.$message({
            message: resp.data,
            type: 'success'
          });
        })
      } else {
        this.$message.error("请先登录账号");
      }
    }
  },
  created() {
    // 从缓存中获取用户信息
    if (this.$store.state.customer.cid) {
      this.isLogin = true;
      this.customer = this.$store.state.customer;
    }
    // 从服务器获取商品信息
    axios.get("http://localhost:8181/goods/shop?sid=" +
this.$route.query.sid).then(resp => {
      this.goodsData = resp.data;
      this.sname = resp.data[0].sname;
    })
  }
}
</script>
```

三、不足与总结

不足：

- 缺乏事务支持和异常处理，无法支持大型项目的需要。
- 没有对service层接口类与实现类进行区分，不方便业务的进一步拓展。
- 代码注释不够规范。
- 没有使用枚举类而在代码文件中直接使用特定的字符串。
- 学习内容不够熟练和深入，在代码过程中需要反复查阅资料。

总结：

- 学习了javaweb、ssm和springboot、vue入门知识，记录了很多的笔记，锻炼了自主学习能力。
- 比较以前的代码有了较大的进步。
- 体会到了独立开发小型项目和自己解决问题的乐趣。
- 更好的认识到自己的不足。