第5章 订单管理（阶段实战）

学习目标

- 提升学员分析能力和动手能力，使学员能够独立完成部分业务功能
- 通过对知识点的综合运用巩固某阶段学习成果
- 检验学员代码编写能力与熟练度

1. 订单列表与详情

1.1 需求分析

管理后台展示订单列表，点击“查看”显示订单详情。

1.2 表结构分析

tb_order 表（订单主表）

字段名称	字段含义	字段类型	字段长度	备注
id	订单id	VARCHAR		分布式id
total_num	数量合计	INT		
total_money	金额合计	INT		单位（分）
pre_money	优惠金额	INT		单位（分）
post_fee	邮费	INT		单位（分）
pay_money	实付金额	INT		单位（分）
pay_type	支付类型	VARCHAR		0、货到付款 1、微信支付，2、支付宝 3、银联支付
create_time	订单创建时间	DATETIME		
update_time	订单更新时间	DATETIME		
pay_time	付款时间	DATETIME		
consign_time	发货时间	DATETIME		
	交易			

end_time	完成时间	DATETIME		
close_time	交易关闭时间	DATETIME		
shipping_name	物流名称	VARCHAR		
shipping_code	物流单号	VARCHAR		
username	用户名称	VARCHAR		
buyer_message	买家留言	VARCHAR		
buyer_rate	是否评价	CHAR		
receiver_contact	收货人	VARCHAR		
receiver_mobile	收货人手机	VARCHAR		
receiver_address	收货人地址	VARCHAR		
source_type	订单来源：	CHAR		1:web，2：app，3：微信公众号，4：微信小程序 5 H5手机页面
transaction_id	交易流水号	VARCHAR		
	订单			0待付款、1待发货、2已发货、

order_status	状态	CHAR		3已完成、4已关闭
pay_status	支付状态	CHAR		0未支付、1已支付、2已退款
consign_status	发货状态	CHAR		0未发货、1已发货
is_delete	是否删除	CHAR		0：未删除 1：已删除

tb_order_item 表（订单明细表）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	VARCHAR		分布式id
category_id1	1级分类	INT		
category_id2	2级分类	INT		
category_id3	3级分类	INT		
spu_id	SPU_ID	VARCHAR		
sku_id	SKU_ID	VARCHAR		
order_id	订单ID	VARCHAR		
name	商品名称	VARCHAR		
price	单价	INT		
num	数量	INT		
money	总金额	INT		
pay_money	实付金额	INT		
image	图片地址	VARCHAR		
weight	重量	INT		
post_fee	运费	INT		
is_return	是否退货	CHAR		0：未退款 1：已申请 2：已退

1.3 实现思路

（1）创建订单组合实体类，属性包括订单对象和订单明细列表。

（2）后端实现方法，根据订单id查询订单组合实体类。

（3）前端转递id获取数据后展示。

2. 订单发货

2.1 需求分析

（1）用户在订单列表中，勾选需要发货的订单，然后点击批量发货按钮。

（2）用户在发货列表中选择物流公司和填写运单号，点击确定进行批量发货。

详见静态原型。

（3）发货记录订单日志

2.2 表结构分析

tb_order_log 表（订单日志）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	VARCHAR		
operater	操作员	VARCHAR		
operate_time	操作时间	DATETIME		
order_id	订单ID	VARCHAR		
order_status	订单状态	CHAR		
pay_status	付款状态	CHAR		
consign_status	发货状态	CHAR		
remarks	备注	VARCHAR		

2.3 实现思路

（1）后端编写方法，根据id数组查询未发货订单，前端向后端传递id数据，后端返回给前端订单列表

（2）前端使用elementUI的表格控件展现，物流公司和运单号采用模板列。

（3）批量发货，前端给后端发送订单集合（post），后端代码接受后批量修改订单状态、发货状态、发货时间。

（4）发货操作需要记录订单日志

2.4 关键代码提示

2.4.1 根据选中的ID查询未发货订单

修改OrderServiceImpl，在createExample中添加查询条件

```
// 根据  id 数组查询 查询
if(searchMap.get("ids")!=null   ){
    criteria.andIn("id", Arrays.asList((String[])searchMap.get("ids")));
}
```

前端查询可以发送post请求 order/findList.do ，请求对象为以下格式

```
{
    consignStatus:'0',
    ids:[1,2,3]
}
```

2.4.2 批量发货

（1）修改OrderService接口，新增方法定义

```
    /**
     * 批量发货
     * @param orders
     */
    public void batchSend(List<Order> orders);
```

（2）修改OrderServiceImpl，实现方法

```
    /**
     * 批量发货
     * @param orders
     */
    public void batchSend(List<Order> orders) {
        //判断运单号和物流公司是否为空
        for(Order order :orders){
           if(order.getShippingCode()==null ||
order.getShippingName()==null){
                throw new RuntimeException("请选择快递公司和填写快递单号");
           }
        }
        //循环订单
        for(Order order :orders){
            order.setOrderStatus("3");//订单状态  已发货
            order.setConsignStatus("2"); //发货状态  已发货
            order.setConsignTime(new Date());//发货时间
            orderMapper.updateByPrimaryKeySelective(order);
            //记录订单日志  。。。（代码略）
        }
    }
```

（3）修改OrderController，新增方法

```
    @PostMapping("/batchSend")
    public Result batchSend(@RequestBody List<Order> orders){
        orderService.batchSend(orders);
        return new Result();
    }
```

3. 退货与退款

3.1 需求分析

首先我们要理解退货和退款的区别。

退货：

（1）当用户收到货后感觉不满意，可以在用户中心提出退货申请，需要选择退货原因和上传图片凭证。

（2）平台客服人员看到退货申请后，根据情况选择同意退货或拒绝退货。

（3）平台如果同意退货，用户需要按照平台给出的地址寄回商品。

（4）平台收到商品后，将货款退还给用户。

退款：

（1）退款的情况多半是用户没有收到商品，或是收到的商品已经无法退回，需要走退款流程。

（2）用户在用户中心提交退款申请。

（3）平台客服人员看到退款申请，根据情况选择同意退款或拒绝退款。

（4）平台如果同意退款，直接将货款退还给用户，无需邮寄货物。

退货退款状态：0已申请、1已同意、2已驳回、3已完成

3.2 表结构分析

tb_return_cause 表（退货退款原因表）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	INT		
cause	原因	VARCHAR		
seq	排序	INT		
status	是否启用	CHAR		

tb_return_order 表 （退货退款申请表）

字段名称	字段含义	字段类型	字段长度	备注
id	服务单号	VARCHAR		分布式id
order_id	订单号	VARCHAR		
apply_time	申请时间	DATETIME		
user_id	用户ID	VARCHAR		
user_account	用户账号	VARCHAR		
linkman	联系人	VARCHAR		
linkman_mobile	联系人手机	VARCHAR		
type	类型	CHAR		1.退货 2.退款
return_money	退款金额	INT		
is_return_freight	是否退运费	CHAR		
status	申请状态	CHAR		0：申请 1同意 2驳回
dispose_time	处理时间	DATETIME		
return_cause	退货退款原因	INT		
evidence	凭证图片	VARCHAR		逗号分割
description	问题描述	VARCHAR		
remark	处理备注	VARCHAR		
admin_id	管理员id	INT		

tb_return_order_item （退货退款申请明细表）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	VARCHAR		分布式id
category_id	分类ID	INT		
spu_id	SPU_ID	VARCHAR		
sku_id	SKU_ID	VARCHAR		
order_id	订单ID	VARCHAR		
order_item_id	订单明细ID	VARCHAR		
return_order_id	退货订单ID	VARCHAR		
title	标题	VARCHAR		
price	单价	INT		
num	数量	INT		
money	总金额	INT		
pay_money	支付金额	INT		
image	图片地址	VARCHAR		
weight	重量	INT		

3.3 实现思路

以退款为例：

分别实现同意退款和驳回退款的方法

（1）同意退款

根据id修改退货退款订单的状态为1，记录当前管理员id和当前时间。

需要做一些必要的验证，退款的金额不能大于原订单的金额。

调用支付平台的退款接口（代码不用实现）

（2）驳回退款

根据id修改退货退款订单的状态为2，记录当前管理员id、当前时间和驳回理由。

将原订单明细的退款状态改为未申请。

3.4 关键代码提示

3.4.1 同意退款流程

（1）ReturnOrderService接口新增方法定义

```
    /**
     * 同意退款
     * @param id
     */
    public void agreeRefund(String id,Integer money,Integer adminId );
```

（2）ReturnOrderServiceImpl实现此方法

```
    /**
     * 同意退款
     * @param id
     * @param money
     * @param adminId
     */
    public void agreeRefund(String id, Integer money, Integer adminId) {

        ReturnOrder returnOrder =
returnOrderMapper.selectByPrimaryKey(id);
        if(returnOrder==null){
            throw new RuntimeException("退款订单不存在！");
        }
        if(!returnOrder.getType().equals("2")){
            throw new RuntimeException("不是退款订单！");
        }
        if(money>returnOrder.getReturnMoney() || money<=0){
            throw new RuntimeException("退款金额不合法！");
        }
        returnOrder.setReturnMoney(money);
        returnOrder.setStatus("1");//同意
        returnOrder.setAdminId(adminId);//管理员
        returnOrder.setDisposeTime(new Date());//处理日期
        returnOrderMapper.updateByPrimaryKeySelective(returnOrder);//保存

        //调用支付平台的退款接口
    }
```

（3）ReturnOrderController新增方法定义

```
    /**
     * 同意退款
     * @param id
     * @param money
     */
    public Result agreeRefund(String id, Integer money){
        Integer adminId=0;//获取当前登陆人ID
        returnOrderService.agreeRefund(id,money,adminId);
        return new Result();
    }
```

3.4.2 驳回退款流程

（1）ReturnOrderService接口新增方法定义

```
    /**
     * 驳回退款请求
     * @param id
     * @param remark
     * @param adminId
     */
    public void rejectRefund(String id,String remark,Integer adminId );
```

（2）ReturnOrderServiceImpl实现此方法

```
    @Autowired
    private ReturnOrderItemMapper returnOrderItemMapper;

    @Autowired
    private OrderItemMapper orderItemMapper;

    /**
     * 驳回退款
     * @param id
     * @param remark
     * @param adminId
     */
    public void rejectRefund(String id, String remark, Integer adminId) {
        ReturnOrder returnOrder =
returnOrderMapper.selectByPrimaryKey(id);
        if(returnOrder==null){
            throw new RuntimeException("退款订单不存在！");
        }
        if(!returnOrder.getType().equals("2")){
            throw new RuntimeException("不是退款订单！");
        }
        if(remark.length()<5){
            throw new RuntimeException("请输入驳回理由！");
        }
        //修改属性
        returnOrder.setRemark(remark);//驳回理由
        returnOrder.setStatus("2");//驳回
        returnOrder.setAdminId(adminId);//管理员
        returnOrder.setDisposeTime(new Date());//处理日期
        returnOrderMapper.updateByPrimaryKeySelective(returnOrder);//保存

        //修改对应订单明细的退款状态为未申请
        Example example=new Example(ReturnOrderItem.class);
        Example.Criteria criteria = example.createCriteria();
        criteria.andEqualTo("returnOrderId",id);//退费订单ID
        List<ReturnOrderItem> returnOrderItems =
returnOrderItemMapper.selectByExample(example);
        for(ReturnOrderItem returnOrderItem:returnOrderItems ){
            OrderItem orderitem=new OrderItem();

            orderitem.setId(returnOrderItem.getOrderItemId());//提取订单明
```

```
细ID
            orderitem.setIsReturn("0");
            orderItemMapper.updateByPrimaryKeySelective(orderitem);//更新
状态，让其可以重新发送退款
        }
    }
```

（3）ReturnOrderController新增方法定义

```
    /**
     * 驳回退款
     * @param id
     * @param remark
     */
    public Result rejectRefund(String id, String remark){
        Integer adminId=0;//获取当前登陆人ID
        returnOrderService.rejectRefund(id,remark,adminId);
        return new Result();
    }
```

4. 订单超时自动处理

4.1 需求分析

当订单超过一定时间（后台设置）后用户没有付款，需要将订单关闭。

4.2 表结构分析

tb_order_config （订单设置表）

字段名称	字段含义	字段类型	字段长度	备注
id	ID	INT		
order_timeout	正常订单超时时间（分）	INT		
seckill_timeout	秒杀订单超时时间（分）	INT		
take_timeout	自动收货（天）	INT		
service_timeout	售后期限	INT		
comment_timeout	自动五星好评	INT		

4.3 实现思路

定时执行一段逻辑，查询超时订单，执行订单的关闭处理。

4.4 定时任务解决方案-SpringTask

Spring3.0以后自主开发的定时任务工具，spring-task，可以将它比作一个轻量级的Quartz，而且使用起来很简单，除spring相关的包外不需要额外的包，而且支持注解和配置文件两种形式。

4.4.1 快速入门

测试代码，每间隔一秒自动输出

（1）在qingcheng_common_web工程下的配置文件applicationContext-config.xml中添加配置

```
<!--开启任务调度-->
<task:annotation-driven></task:annotation-driven>
```

需要添加命名空间

```
xmlns:task="http://www.springframework.org/schema/task"
```

还有约束

```
http://www.springframework.org/schema/task
http://www.springframework.org/schema/task/spring-task.xsd
```

（2）qingcheng_web_manager工程新增OrderTask

```
@Component
public class OrderTask {

    @Scheduled(cron = "* * * * * ?")
    public void orderTimeOutLogic(){
        System.out.println(".........");
    }
}
```

运行，就可以看到每秒在控制台输出.....

4.4.2 Cron表达式

Cron表达式是一个字符串，字符串以5或6个空格隔开，分开工6或7个域，每一个域代表一个含义,Cron有如下两种语法
格式：
Seconds Minutes Hours DayofMonth Month DayofWeek Year 或
Seconds Minutes Hours DayofMonth Month DayofWeek

注意：**SpringTask**不支持第一种格式，也就是说只能写**6**个域！

每一个域可出现的字符如下：

代码
Seconds:可出现,- * / 四个字符，有效范围为0-59的整数

Minutes:可出现,- * / 四个字符，有效范围为0-59的整数

Hours:可出现,- * / 四个字符，有效范围为0-23的整数

DayofMonth:可出现,- * / ? L W C八个字符，有效范围为1-31的整数

Month:可出现,- * / 四个字符，有效范围为1-12的整数或JAN-DEc

DayofWeek:可出现,- * / ? L C #四个字符，有效范围为1-7的整数或SUN-SAT两个范围。1表示星期天，2表示星期一， 依次类推

Year:可出现,- * / 四个字符，有效范围为1970-2099年

每一个域都使用数字，但还可以出现如下特殊字符，它们的含义是：

(1) *：表示匹配该域的任意值，假如在Minutes域使用*,即表示每分钟都会触发事件。

(2) ?: 只能用在DayofMonth和DayofWeek两个域。它也匹配域的任意值，但实际不会。因为DayofMonth和DayofWeek会相互影响。
例如想在每月的20日触发调度，不管20日到底是星期几，则只能使用如下写法： 13 13 15 20 * ?,其中最后一位只能用？，而不能使用*，
如果使用*表示不管星期几都会触发，实际上并不是这样。

(3)-:表示范围，例如在Minutes域使用5-20，表示从5分到20分钟每分钟触发一次

(4)/：表示起始时间开始触发，然后每隔固定时间触发一次，例如在Minutes域使用5/20,则意味着5分钟触发一次，而25，45等分别触发一次.

(5),:表示列出枚举值值。例如：在Minutes域使用5,20，则意味着在5和20分每分钟触发一次。

(6)L:表示最后，只能出现在DayofWeek和DayofMonth域，如果在DayofWeek域使用5L,意味着在最后的一个星期四触发。

(7)W:表示有效工作日(周一到周五),只能出现在DayofMonth域，系统将在离指定日期的最近的有效工作日触发事件。
例如：在DayofMonth使用5W，如果5日是星期六，则将在最近的工作日：星期五，即4日触发。如果5日是星期天，则在6日触发；
如果5日在星期一到星期五中的一天，则就在5日触发。另外一点，W的最近寻找不会跨过月份

(8)LW:这两个字符可以连用，表示在某个月最后一个工作日，即最后一个星期五。

(9)#:用于确定每个月第几个星期几，只能出现在DayofMonth域。例如在4#2，表示某月的第二个星期三。

4.5 关键代码提示

（1）OrderService接口新增方法

```
    /**
     * 订单超时处理逻辑
     */
    public void orderTimeOutLogic();
```

（2）OrderServiceImpl实现此方法

```
/**
     * 订单超时处理
     */
    public void orderTimeOutLogic() {

        //订单超时未付款 自动关闭
        //查询超时时间
        OrderConfig orderConfig =
orderConfigMapper.selectByPrimaryKey(1);
        Integer orderTimeout = orderConfig.getOrderTimeout();  //超时时间
（分） 60
        LocalDateTime localDateTime =
LocalDateTime.now().minusMinutes(orderTimeout);  //得到超时的时间点

        //设置查询条件
        Example example=new Example(Order.class);
        Example.Criteria criteria = example.createCriteria();
        criteria.andLessThan("createTime",localDateTime);//创建时间小于超时
时间
        criteria.andEqualTo("orderStatus","0");//未付款的
        criteria.andEqualTo("isDelete","0");//未删除的

        //查询超时订单
        List<Order> orders = orderMapper.selectByExample(example);
        for(Order order :orders){
            //记录订单变动日志
            OrderLog orderLog=new OrderLog();
            orderLog.setOperater("system");// 系统
            orderLog.setOperateTime(new Date());//当前日期
            orderLog.setOrderStatus("4");
            orderLog.setPayStatus(order.getPayStatus());
            orderLog.setConsignStatus(order.getConsignStatus());
            orderLog.setRemarks("超时订单，系统自动关闭");
            orderLog.setOrderId(order.getId());
            orderLogMapper.insert(orderLog);
            //更改订单状态
            order.setOrderStatus("4");
            order.setCloseTime(new Date());//关闭日期
            orderMapper.updateByPrimaryKeySelective(order);

        }
```

```
    }
```

（3）设置规则：每2分钟处理一次请求，将60分钟前未付款订单关闭

```
@Component
public class OrderTask {

    @Reference
    private OrderService orderService;

    @Scheduled(cron = "0 0/2 * * * ?")
    public void orderTimeOutLogic(){
        System.out.println("每两分钟间隔执行一次任务"+ new Date());
        orderService.orderTimeOutLogic();
    }
}
```

5. 合并订单与拆分订单

5.1 合并订单

需求：页面上选择“主订单号”，填写“从订单号”，传递给后端，后端将两个订单合并为一个订单

GET /order/ merge.do

参数 orderId1 主订单号 orderId2 从订单号

实现思路：

后端接收两个订单号，将“从订单”的金额等信息合并到“主订单”，“从订单”的订单明细也归属于主订单。

合并操作完成后，对“从订单”添加删除标记（逻辑删除）

记录订单日志

5.2 拆分订单

需求：页面上选择要拆分的订单明细的商品数量，后端将此订单拆分为两个订单

POST /order/split.do　参数：

```
[{
    id:1,
    num:10
},
{
    id:2,
    num:5
}]
```

实现思路：

后端接收List　循环，判断要拆分的数量是否大于明细数量，如果通过验证，将拆分的数量产生新的订单。