尚品汇商城

一、支付成功处理

1.1 更改订单状态

订单支付成功后，我们已经更改了订单支付记录状态，接下来我还有更改订单状态，因为他们是不同的微服务模块，所以我们采用消息队列的方式，保证数据最终一致性；

1.1.1 在 MqConst 常量类添加变量

```
/**
 * 订单支付
 */
public static final String EXCHANGE_DIRECT_PAYMENT_PAY = "exchange.direct.payment.pay";
public static final String ROUTING_PAYMENT_PAY = "payment.pay";
//队列
public static final String QUEUE_PAYMENT_PAY  = "queue.payment.pay";

/**
 * 减库存
 */
public static final String EXCHANGE_DIRECT_WARE_STOCK = "exchange.direct.ware.stock";
public static final String ROUTING_WARE_STOCK = "ware.stock";
//队列
public static final String QUEUE_WARE_STOCK  = "queue.ware.stock";
/**
 * 减库存成功，更新订单状态
 */
public static final String EXCHANGE_DIRECT_WARE_ORDER = "exchange.direct.ware.order";
public static final String ROUTING_WARE_ORDER = "ware.order";
//队列
public static final String QUEUE_WARE_ORDER  = "queue.ware.order";
```

1.1.2 在 service-payment 中添加依赖和配置

```xml
<dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>rabbit-util</artifactId>
    <version>1.0</version>
</dependency>
```

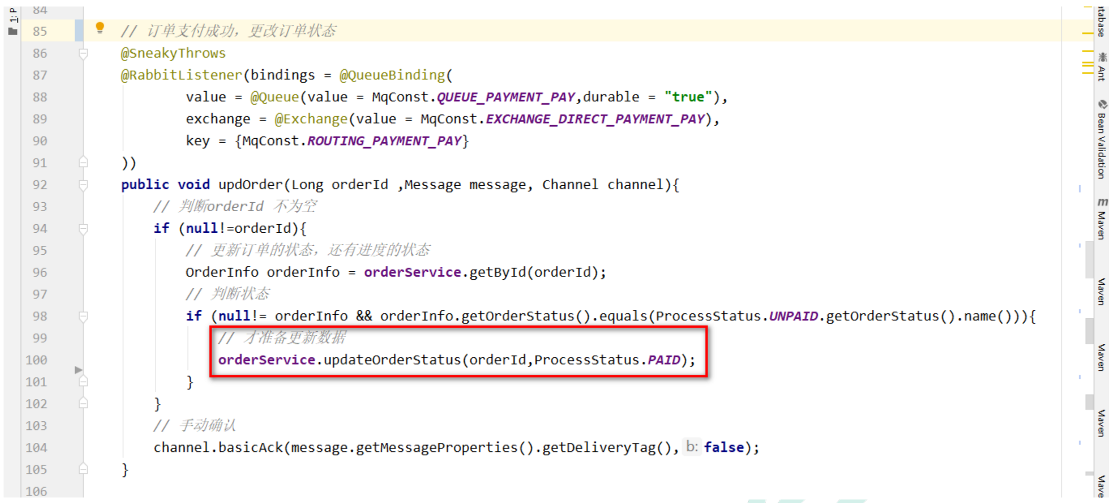
1.1.3 支付成功发送消息

paySuccess

```java
@Override
public void paySuccess(String outTradeNo, String paymentType, Map<String, String> paramsMap) {

    // 根据 outTradeNo，paymentType 查询
  PaymentInfo paymentInfoQuery = this.getPaymentInfo(outTradeNo, paymentType);
    if (paymentInfoQuery==null){
        return;
    }

    try {
        // 改造一下更新的方法！
    PaymentInfo paymentInfo = new PaymentInfo();
        paymentInfo.setCallbackTime(new Date());
        paymentInfo.setPaymentStatus(PaymentStatus.PAID.name());
        paymentInfo.setCallbackContent(paramsMap.toString());
        paymentInfo.setTradeNo(paramsMap.get("trade_no"));
        // 查询条件也可以作为更新条件！
    this.updatePaymentInfo(outTradeNo, paymentType, paymentInfo);
    } catch (Exception e) {
        // 删除 key
        this.redisTemplate.delete(paramsMap.get("notify_id"));
        e.printStackTrace();
    }
    // 发送通知：更新订单的状态！

this.rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_PAYMENT_PAY,MqConst.ROUTING_PAYMENT_PAY,paymentInfoQuery.getOrderId());
}
```

1.1.4 service-order 模块接收消息

OrderReceiver 类添加方法

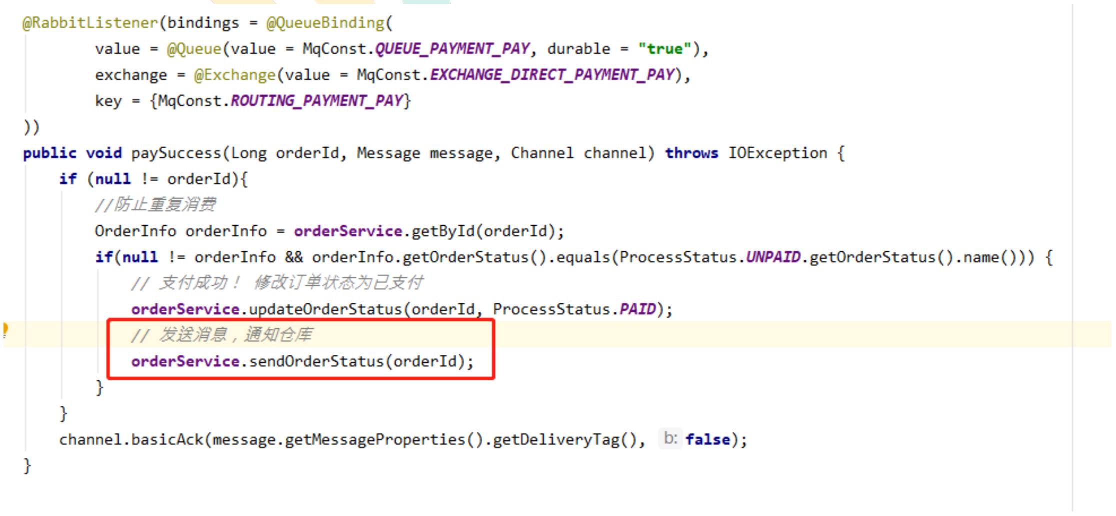


```java
// 订单支付成功，更改订单状态
@SneakyThrows
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(value = MqConst.QUEUE_PAYMENT_PAY,durable = "true"),
        exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_PAYMENT_PAY),
        key = {MqConst.ROUTING_PAYMENT_PAY}
))
public void updOrder(Long orderId ,Message message, Channel channel){
    // 判断orderId 不为空
    if (null!=orderId){
        // 更新订单的状态，还有进度的状态
        OrderInfo orderInfo = orderService.getById(orderId);
        // 判断状态
        if (null!= orderInfo && orderInfo.getOrderStatus().equals(ProcessStatus.UNPAID.getOrderStatus().name())){
            // 才准备更新数据
            orderService.updateOrderStatus(orderId,ProcessStatus.PAID);
        }
    }
    // 手动确认
    channel.basicAck(message.getMessageProperties().getDeliveryTag(), b: false);
}
```

1.2 订单模块发送减库存通知

订单模块除了接收到请求改变单据状态，还要发送库存系统

查看看《库存管理系统接口手册》中【减库存的消息队列消费端接口】中的描述，组织相应的消息数据进行传递。



```java
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(value = MqConst.QUEUE_PAYMENT_PAY, durable = "true"),
        exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_PAYMENT_PAY),
        key = {MqConst.ROUTING_PAYMENT_PAY}
))
public void paySuccess(Long orderId, Message message, Channel channel) throws IOException {
    if (null != orderId){
        //防止重复消费
        OrderInfo orderInfo = orderService.getById(orderId);
        if(null != orderInfo && orderInfo.getOrderStatus().equals(ProcessStatus.UNPAID.getOrderStatus().name())) {
            // 支付成功！ 修改订单状态为已支付
            orderService.updateOrderStatus(orderId, ProcessStatus.PAID);
            // 发送消息，通知仓库
            orderService.sendOrderStatus(orderId);
        }
    }
    channel.basicAck(message.getMessageProperties().getDeliveryTag(),  b: false);
}
```

1.2.1 OrderService 接口

```
/**
 * 发送消息给库存！
 * @param orderId
 */
void sendOrderStatus(Long orderId);

/**
 *将 orderInfo 变为 map 集合
 * @param orderInfo
 */

Map initWareOrder(OrderInfo orderInfo);
```

1.2.2 编写实现类

```
@Override
public void sendOrderStatus(Long orderId) {
    this.updateOrderStatus(orderId, ProcessStatus.NOTIFIED_WARE);

    String wareJson = initWareOrder(orderId);
    rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_WARE_STOCK,
MqConst.ROUTING_WARE_STOCK, wareJson);


}

// 根据 orderId 获取 json 字符串
private String initWareOrder(Long orderId) {
    // 通过 orderId 获取 orderInfo
    OrderInfo orderInfo = getOrderInfo(orderId);

    // 将 orderInfo 中部分数据转换为 Map
    Map map = initWareOrder(orderInfo);

    return JSON.toJSONString(map);
}



//  将 orderInfo 中部分数据转换为 Map
public Map initWareOrder(OrderInfo orderInfo) {
    HashMap<String, Object> map = new HashMap<>();
    map.put("orderId", orderInfo.getId());
    map.put("consignee", orderInfo.getConsignee());
    map.put("consigneeTel", orderInfo.getConsigneeTel());
    map.put("orderComment", orderInfo.getOrderComment());
    map.put("orderBody", orderInfo.getTradeBody());
    map.put("deliveryAddress", orderInfo.getDeliveryAddress());
    map.put("paymentWay", "2");
    map.put("wareId", orderInfo.getWareId());// 仓库 Id ，减库存拆单时需要使
用！
   /*
    details:[{skuId:101,skuNum:1,skuName:
    ’小米手64G’},
    {skuId:201,skuNum:1,skuName:’索尼耳机’}]
     */
    ArrayList<Map> mapArrayList = new ArrayList<>();
    List<OrderDetail> orderDetailList = orderInfo.getOrderDetailList();
    for (OrderDetail orderDetail : orderDetailList) {
        HashMap<String, Object> orderDetailMap = new HashMap<>();
        orderDetailMap.put("skuId", orderDetail.getSkuId());
        orderDetailMap.put("skuNum", orderDetail.getSkuNum());
        orderDetailMap.put("skuName", orderDetail.getSkuName());
        mapArrayList.add(orderDetailMap);
    }
    map.put("details", mapArrayList);
    return map;
}
```

1.3 消费减库存结果

给仓库系统发送减库存消息后，还要接受减库存成功或者失败的消息。

同样根据《库存管理系统接口手册》中**【商品减库结果消息】**的说明完成。消费该消息的消息队列监听程序。

接受到消息后主要做的工作就是更新订单状态。

在订单项目中 OrderReceiver

```
/**
 * 扣减库存成功，更新订单状态
 * @param msgJson
 * @throws IOException
 */
@RabbitListener(bindings = @QueueBinding(
        value = @Queue(value = MqConst.QUEUE_WARE_ORDER, durable = "true"),
        exchange = @Exchange(value = MqConst.EXCHANGE_DIRECT_WARE_ORDER),
        key = {MqConst.ROUTING_WARE_ORDER}
))
public void updateOrderStatus(String msgJson, Message message, Channel channel) throws IOException {
    if (!StringUtils.isEmpty(msgJson)){
        Map<String,Object> map = JSON.parseObject(msgJson,
Map.class);
        String orderId = (String)map.get("orderId");
        String status = (String)map.get("status");
        if ("DEDUCTED".equals(status)){
            // 减库存成功！ 修改订单状态为已支付
        orderService.updateOrderStatus(Long.parseLong(orderId),
ProcessStatus.WAITING_DELEVER);
        }else {
        /*
            减库存失败！远程调用其他仓库查看是否有库存！
            true:            orderService.sendOrderStatus(orderId);
orderService.updateOrderStatus(orderId,
ProcessStatus.NOTIFIED_WARE);
            false:  1.  补货  | 2.   人工客服。
         */
            orderService.updateOrderStatus(Long.parseLong(orderId),
ProcessStatus.STOCK_EXCEPTION);
        }
    }
channel.basicAck(message.getMessageProperties().getDeliveryTag(),
false);
}
```

1.4 拆单接口

1.4.1 库存系统配置拆单回调接口

```
application-dev.yml

order:
  split:
    url: http://localhost:8204/api/order/orderSplit
```

1.4.2 订单实现拆单接口

```
List<OrderInfo> orderSplit(Long orderId, String wareSkuMap);
```

1.4.3 拆单接口实现类

```java
@Override
@Transactional
public List<OrderInfo> orderSplit(Long orderId, String wareSkuMap) {
    ArrayList<OrderInfo> orderInfoArrayList = new ArrayList<>();
    /*
    1.   先获取到原始订单 107
    2.      将  wareSkuMap  转 换 为 我 们 能 操 作 的 对 象
[{"wareId":"1","skuIds":["2","10"]},{"wareId":"2","skuIds":["3"]}]
        方案一：class Param{
                    private String wareId;
                    private List<String> skuIds;
                }
        方 案 二 ： 看 做 一 个  Map  mpa.put("wareId",value);
map.put("skuIds",value)

    3.   创建一个新的子订单 108 109 。。。
    4.   给子订单赋值
    5.   保存子订单到数据库
    6.   修改原始订单的状态
    7.   测试
     */
    OrderInfo orderInfoOrigin = getOrderInfo(orderId);
    List<Map> maps = JSON.parseArray(wareSkuMap, Map.class);
    if (maps != null) {
        for (Map map : maps) {
            String wareId = (String) map.get("wareId");

            List<String> skuIds = (List<String>) map.get("skuIds");

            OrderInfo subOrderInfo = new OrderInfo();
            // 属性拷贝
      BeanUtils.copyProperties(orderInfoOrigin, subOrderInfo);
            // 防止主键冲突
        subOrderInfo.setId(null);
            subOrderInfo.setParentOrderId(orderId);
            // 赋值仓库Id
            subOrderInfo.setWareId(wareId);

            // 计算子订单的金额: 必须有订单明细
      // 获取到子订单明细
      // 声明一个集合来存储子订单明细
      ArrayList<OrderDetail> orderDetails = new ArrayList<>();
```

```
            List<OrderDetail>            orderDetailList           =
orderInfoOrigin.getOrderDetailList();
            // 表示主主订单明细中获取到子订单的明细
      if (orderDetailList != null && orderDetailList.size() > 0) {
                  for (OrderDetail orderDetail : orderDetailList) {
                      // 获取子订单明细的商品 Id
                      for (String skuId : skuIds) {
                          if          (Long.parseLong(skuId)          ==
orderDetail.getSkuId().longValue()) {
                              // 将订单明细添加到集合
                  orderDetails.add(orderDetail);
                          }
                      }
                  }
            }
            subOrderInfo.setOrderDetailList(orderDetails);
            // 计算总金额
      subOrderInfo.sumTotalAmount();
            // 保存子订单
        saveOrderInfo(subOrderInfo);
            // 将子订单添加到集合中！
        orderInfoArrayList.add(subOrderInfo);
        }
    }
    // 修改原始订单的状态
  updateOrderStatus(orderId, ProcessStatus.SPLIT);
    return orderInfoArrayList;
}
```

1.4.4 拆单接口控制器

```
/**
 * 拆单业务
 * @param request
 * @return
 */
@RequestMapping("orderSplit")
public String orderSplit(HttpServletRequest request){
    String orderId = request.getParameter("orderId");
    String wareSkuMap = request.getParameter("wareSkuMap");

    // 拆单：获取到的子订单集合
   List<OrderInfo>                  subOrderInfoList               =
orderService.orderSplit(Long.parseLong(orderId),wareSkuMap);
```

```
    // 声明一个存储map 的集合
   ArrayList<Map> mapArrayList = new ArrayList<>();
    // 生成子订单集合
   for (OrderInfo orderInfo : subOrderInfoList) {
        Map map = orderService.initWareOrder(orderInfo);
        // 添加到集合中！
     mapArrayList.add(map);
    }
    return JSON.toJSONString(mapArrayList);
}
```

二、取消订单业务补充

2.1 在 MqConst 中添加常量

```
/**
 * 关闭交易
 */
public static final String EXCHANGE_DIRECT_PAYMENT_CLOSE = "exchange.direct.payment.close";
public static final String ROUTING_PAYMENT_CLOSE = "payment.close";
//队列
public static final String QUEUE_PAYMENT_CLOSE = "queue.payment.close";
```

2.2 在取消订单实现类中发送消息关闭交易

更改接口

```
@Override
public void execExpiredOrder(Long orderId) {
    // orderInfo
    updateOrderStatus(orderId, ProcessStatus.CLOSED);

    rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_PAYMENT_CLOSE,
MqConst.ROUTING_PAYMENT_CLOSE, orderId);
}
```

2.3 service-payment 模块接收消息

2.3.1 编写消费者

```
package com.atguigu.gmall.payment.receiver;

@Component
public class PaymentReceiver {

    @Autowired
    private PaymentService paymentService;

    @SneakyThrows
    @RabbitListener(bindings = @QueueBinding(
            value  =  @Queue(value  =  MqConst.QUEUE_PAYMENT_CLOSE,durable
= "true"),
            exchange            =            @Exchange(value            =
MqConst.EXCHANGE_DIRECT_PAYMENT_CLOSE),
            key = {MqConst.ROUTING_PAYMENT_CLOSE}
    ))
    public  void  closePayment(Long  orderId  ,  Message  message,  Channel
channel){
        if (null != orderId){
            // 关闭交易
      paymentService.closePayment(orderId);
        }
        // 手动ack

channel.basicAck(message.getMessageProperties().getDeliveryTag(),false)
;
    }
}
```

2.3.2 编写关闭交易记录接口与实现类

```
PaymentService

/**
 * 关闭过期交易记录
* @param orderId
 */
```

```
void closePayment(Long orderId);
```

```
@Override
public void closePayment(Long orderId) {
    // 设置关闭交易记录的条件  118
    QueryWrapper<PaymentInfo>     paymentInfoQueryWrapper     =     new
QueryWrapper<>();
    paymentInfoQueryWrapper.eq("order_id",orderId);
    // 如果当前的交易记录不存在，则不更新交易记录
  Integer                            count                            =
paymentInfoMapper.selectCount(paymentInfoQueryWrapper);
    if (null == count || count.intValue()==0) return;
    // 在关闭支付宝交易之前。还需要关闭paymentInfo
    PaymentInfo paymentInfo = new PaymentInfo();
    paymentInfo.setPaymentStatus(PaymentStatus.CLOSED.name());
    paymentInfoMapper.update(paymentInfo,paymentInfoQueryWrapper);

}
```

2.4 支付宝关闭交易

2.4.1 编写接口

```
AlipayService 接口

/***
 * 关闭交易
 * @param orderId
 * @return
 */
Boolean closePay(Long orderId);
```

2.4.2 编写实现类

```
@SneakyThrows
@Override
public Boolean closePay(Long orderId) {
    OrderInfo orderInfo = orderFeignClient.getOrderInfo(orderId);
    AlipayTradeCloseRequest request = new AlipayTradeCloseRequest();
    HashMap<String, Object> map = new HashMap<>();
    // map.put("trade_no",paymentInfo.getTradeNo()); // 从 paymentInfo
中获取！
    map.put("out_trade_no",orderInfo.getOutTradeNo());
    map.put("operator_id","YX01");
    request.setBizContent(JSON.toJSONString(map));
    AlipayTradeCloseResponse response = alipayClient.execute(request);
    if(response.isSuccess()){
        System.out.println("调用成功");
        return true;
    } else {
        System.out.println("调用失败");
        return false;
    }
}
```

2.4.3 编写控制器

```
AlipayController

http://localhost:8205/api/payment/alipay/closePay/25

// 根据订单 Id 关闭订单
@GetMapping("closePay/{orderId}")
@ResponseBody
public Boolean closePay(@PathVariable Long orderId){
    Boolean aBoolean = alipayService.closePay(orderId);
    return aBoolean;
}
```

2.5 查询支付交易记录

2.5.1 编写接口

```
AlipayService

/**
 * 根据订单查询是否支付成功！
 * @param orderId
 * @return
 */
Boolean checkPayment(Long orderId);
```

2.5.2 编写实现类

```
@SneakyThrows
@Override
public Boolean checkPayment(Long orderId) {
    // 根据订单 Id 查询订单信息
  OrderInfo orderInfo = orderFeignClient.getOrderInfo(orderId);
    AlipayTradeQueryRequest request = new AlipayTradeQueryRequest();
    HashMap<String, Object> map = new HashMap<>();
    map.put("out_trade_no",orderInfo.getOutTradeNo());
    // 根据 out_trade_no 查询交易记录
  request.setBizContent(JSON.toJSONString(map));
    AlipayTradeQueryResponse                response                 =
alipayClient.execute(request);
    if(response.isSuccess()){
        return true;
    } else {
        return false;
    }
}
```

2.5.3 编写控制器

http://localhost:8205/api/payment/alipay/checkPayment/30

```
// 查看是否有交易记录
@RequestMapping("checkPayment/{orderId}")
@ResponseBody
public Boolean checkPayment(@PathVariable Long orderId){
    // 调用退款接口
  boolean flag = alipayService.checkPayment(orderId);
    return flag;
}
```

2.6 整合关闭过期订单

2.6.1 在 AlipayController 添加查询 PaymentInfo 数据接口

```
@GetMapping("getPaymentInfo/{outTradeNo}")
@ResponseBody
public PaymentInfo getPaymentInfo(@PathVariable String outTradeNo){
    PaymentInfo paymentInfo = paymentService.getPaymentInfo(outTradeNo,
PaymentType.ALIPAY.name());
    if (null!=paymentInfo){
        return paymentInfo;
    }
    return null;
}
```

2.6.2 创建 service-payment-client

PaymentFeignClient 接口

```java
package com.atguigu.gmall.payment.client;

@FeignClient(value      =      "service-payment",fallback      =
PaymentDegradeFeignClient.class)
public interface PaymentFeignClient {

    @GetMapping("api/payment/alipay/closePay/{orderId}")
    Boolean closePay(@PathVariable Long orderId);

    @GetMapping("api/payment/alipay/checkPayment/{orderId}")
    Boolean checkPayment(@PathVariable Long orderId);


@GetMapping("api/payment/alipay/getPaymentInfo/{outTradeNo}")
    PaymentInfo       getPaymentInfo(@PathVariable       String
outTradeNo);

}
```

PaymentDegradeFeignClient 实现类

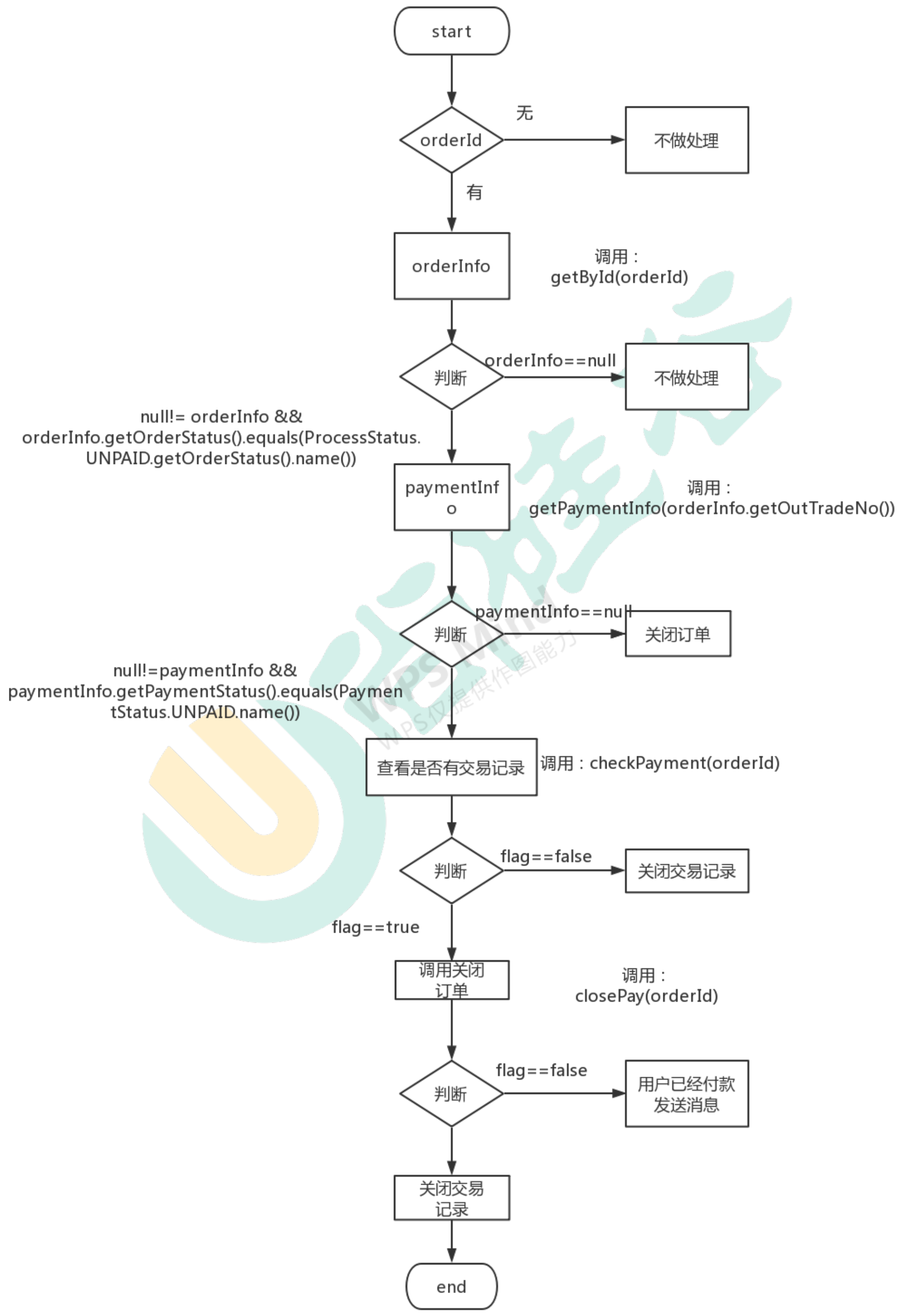
```java
@Component
public      class      PaymentDegradeFeignClient      implements
PaymentFeignClient {
    @Override
    public Boolean closePay(Long orderId) {
        return null;
    }

    @Override
    public Boolean checkPayment(Long orderId) {
        return null;
    }

    @Override
    public PaymentInfo getPaymentInfo(String outTradeNo) {
        return null;
    }

}
```

2.6.3 在订单 service-order 项目中添加依赖

```xml
<dependency>
    <groupId>com.atguigu.gmall</groupId>
    <artifactId>service-payment-client</artifactId>
    <version>1.0</version>
</dependency>
```

2.6.4 OrderReceiver 整合代码

2.6.4.1 关闭订单流程图：



2.6.4.2 代码实现

```
接口: OrderService

/**
 * 更新过期订单
 * @param orderId
 * @param flag
 */
void execExpiredOrder(Long orderId,String flag);
```

```
@Override
public void execExpiredOrder(Long orderId,String flag) {
    // 调用方法 状态
  updateOrderStatus(orderId,ProcessStatus.CLOSED);
    if ("2".equals(flag)){
        // 发送消息队列，关闭支付宝的交易记录。

rabbitService.sendMessage(MqConst.EXCHANGE_DIRECT_PAYMENT_CLOSE,MqConst.ROUTING_PAYMENT_CLOSE,orderId);
    }
}
```

```
@Autowired
private RabbitService rabbitService;

@Autowired
private PaymentFeignClient paymentFeignClient;

//  监听消息
@SneakyThrows
@RabbitListener(queues = MqConst.QUEUE_ORDER_CANCEL)
public void orderCancel(Long orderId, Message message, Channel channel){
    try {
        //  判断订单id 是否存在!
    if (orderId!=null){
            //  根据订单Id 查询订单对象
      OrderInfo orderInfo = orderService.getById(orderId);
            //  判断
      if(orderInfo!=null && "UNPAID".equals(orderInfo.getOrderStatus()) &&
"UNPAID".equals(orderInfo.getProcessStatus())){
                //  关闭过期订单! 还需要关闭对应的 paymentInfo ，还有alipay.
                //  orderService.execExpiredOrder(orderId);
                //  查询paymentInfo 是否存在!
        PaymentInfo paymentInfo =
paymentFeignClient.getPaymentInfo(orderInfo.getOutTradeNo());
                //  判断 用户点击了扫码支付
        if(paymentInfo!=null &&
"UNPAID".equals(paymentInfo.getPaymentStatus())){

                    //  查看是否有交易记录!
          Boolean flag = paymentFeignClient.checkPayment(orderId);
```

```
                    //  判断
          if (flag){
                        //  flag = true , 有交易记录
            //  调用关闭接口！ 扫码未支付这样才能关闭成功！
            Boolean result = paymentFeignClient.closePay(orderId);
                        //  判断
            if (result){
                            //  result = true; 关闭成功！ 未付款！ 需要关闭
orderInfo， paymentInfo，Alipay
                            orderService.execExpiredOrder(orderId,"2");
                        }else {
                            //  result = false; 表示付款！
              //  说明已经付款了！ 正常付款成功都会走异步通知！
            }
                    }else {
                        //  没有交易记录，不需要关闭支付！  需要关闭 orderInfo，
paymentInfo
                        orderService.execExpiredOrder(orderId,"2");
                    }

                }else {
                    //  只关闭订单 orderInfo！
          orderService.execExpiredOrder(orderId,"1");
                }
            }
        }

    } catch (Exception e) {
        //  写入日志...
        e.printStackTrace();
    }
    //  手动确认
  channel.basicAck(message.getMessageProperties().getDeliveryTag(),false);
}
```