第5章 订单支付与出货控制

1. 小程序端功能概述

1.1 产品原型

小程序端是C端用户用来在售货机上购买商品的一端，主要包括商品列表展示、商品详情与支付购买、我的订单列表、 附近售货机搜索等功能。

产品原型：

https://app.mockplus.cn/run/prototype/tPerX4XrY4/BSJNfay9MlZ/oA1CE0pr_?dt=iPhone&ha=1&la=1&ps=1

设计稿：

https://app.mockplus.cn/run/design/WWk4pKFL3gM?dt=iPhone&ha=1&la=1&ps=1

1.2 系统体验

以下是售货机01000028的二维码，我们可以拿起手机扫描二维码体验。

2 小程序-售货机商品查询

2.1 商品列表

2.1.1 需求与实现思路

（1）用户选取售货机之后，小程序端扫码后将售货机编号传入到小程序后台接口。

（2）小程序后台接口通过fegin调用售货机服务里的获取设备商品列表的接口来获取数据。

（3）售货机微服务封装方法，实现根据售货机编号查询商品列表的方法。

2.1.2 立可得**v1.0**代码分析

（1）在`service_common`项目中定义的SkuViewModel是用于需要展示的商品列表中的对象

（2）在小程序后端服务中的`VMController`中添加获取商品列表和商品详情的RESTful接口方法实现

```
@Autowired
private VMService vmService;

/**
     * 获取售货机商品列表
     * @param innerCode
     * @return
     */
@GetMapping("/skuList/{innerCode}")
public List<SkuViewModel> getSkuListByInnercode(@PathVariable String innerCode){
    return vmService.getAllSkuByInnerCode(innerCode);
}
```

（3）服务公共模块service_common的com.lkd.feignService.VMService定义方法

```
@GetMapping("/vm/skuList/{innerCode}")
List<SkuViewModel> getAllSkuByInnerCode(@PathVariable String innerCode);
```

（4）售货机微服务 VendingMachineService接口定义方法

```
/**
 * 获取售货机里所有商品
 * @param innerCode
 * @return
 */
List<SkuViewModel> getSkuList(String innerCode);
```

VendingMachineServiceImpl实现此方法

```
@Override
public List<SkuViewModel> getSkuList(String innerCode) {
    //获取有商品的货道
    List<ChannelEntity> channelList = this.getAllChannel(innerCode)
                                          .stream()
                                          .filter(c->c.getSkuId() > 0 && c.getSku() !=
null)
                                          .collect(Collectors.toList());
    Map<Long,SkuEntity> skuMap = Maps.newHashMap();
    //将商品列表去重之后计算出最终售价返回
    channelList
            .forEach(c->{
                SkuEntity sku = c.getSku();
                sku.setRealPrice(channelService.getRealPrice(innerCode,c.getSkuId()));
                if(!skuMap.containsKey(sku.getSkuId())) {
                    sku.setCapacity(c.getCurrentCapacity());
                    skuMap.put(sku.getSkuId(), sku);
                }else {
                    SkuEntity value = skuMap.get(sku.getSkuId());
                    value.setCapacity(value.getCapacity()+c.getCurrentCapacity());
                    skuMap.put(sku.getSkuId(),value);
                }
            });
    if(skuMap.values().size() <= 0) return Lists.newArrayList();

    return skuMap
                .values()
                .stream()
                .map(s->{
                    SkuViewModel sku = new SkuViewModel();
                    sku.setCapacity(s.getCapacity());
                    sku.setDiscount(s.isDiscount());
                    sku.setImage(s.getSkuImage());
                    sku.setPrice(s.getPrice());
                    sku.setRealPrice(s.getRealPrice());
                    sku.setSkuId(s.getSkuId());
                    sku.setSkuName(s.getSkuName());
                    sku.setUnit(s.getUnit());
                    return sku;
                })
                .sorted(Comparator.comparing(SkuViewModel::getCapacity).reversed())
                .collect(Collectors.toList());
}
```

2.1.3 代码优化

修改VendingMachineServiceImpl的getSkuList方法

```
@Override
public List<SkuViewModel> getSkuList(String innerCode) {
    //获取货道列表
    List<ChannelEntity> channelList = this.getAllChannel(innerCode)
            .stream()
            .filter(c -> c.getSkuId() > 0 && c.getSku() !=
null).collect(Collectors.toList());
    //获取有商品的库存余量
    Map<SkuEntity, Integer> skuMap = channelList
            .stream()
            .collect(Collectors.groupingBy(ChannelEntity::getSku,
Collectors.summingInt(ChannelEntity::getCurrentCapacity)));
    //获取有商品的真实价格
    Map<Long, IntSummaryStatistics> skuPrice =
channelList.stream().collect(Collectors.groupingBy(ChannelEntity::getSkuId,
Collectors.summarizingInt(ChannelEntity::getPrice)));

    return skuMap.entrySet().stream().map( entry->{
        SkuEntity sku = entry.getKey(); //查询商品
        sku.setRealPrice( skuPrice.get(sku.getSkuId()).getMin() );//真实价格
        SkuViewModel skuViewModel = new SkuViewModel();
        BeanUtils.copyProperties( sku,skuViewModel );
        skuViewModel.setImage(sku.getSkuImage());//图片
        return  skuViewModel;
    } ).sorted(Comparator.comparing(SkuViewModel::getCapacity).reversed())  //按库存量降
序排序
            .collect(Collectors.toList());
}
```

优化后的代码有三个优点：

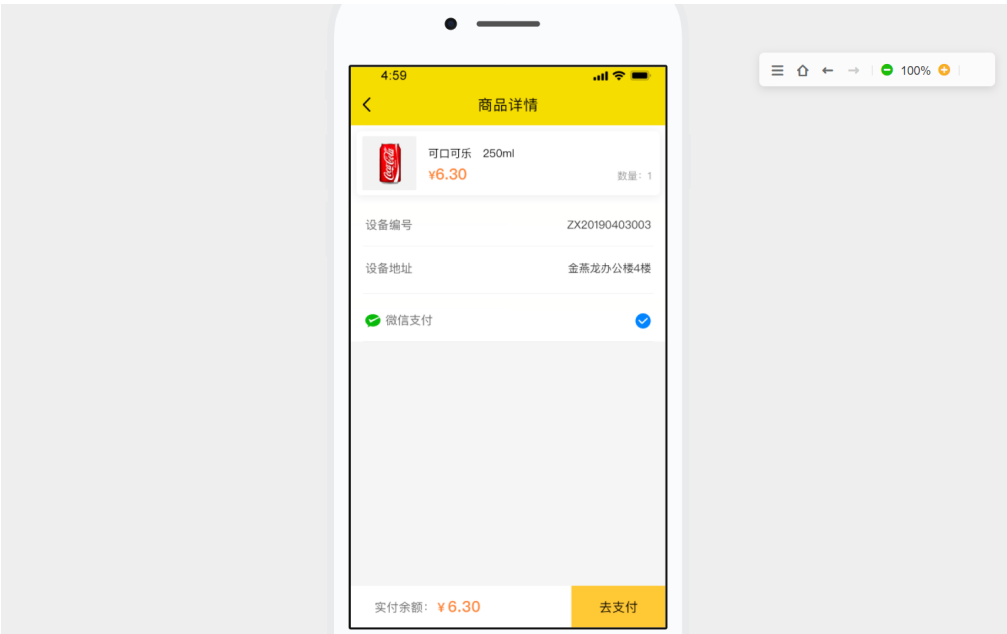
（1）优雅清晰

（2）代码量少

（3）执行高效：只需要查询数据库一次，其余都在内存进行操作。

2.2 商品详情

2.2.1 需求与实现思路



商品详情返回的信息除了包含商品基本信息，还包含设备编号和设备地址

2.2.2 立可得**v1.0**代码分析

微服务公共模块定义了SkuInfoViewModel用于返回商品详情

```
@Data
public class SkuInfoViewModel extends SkuViewModel implements Serializable {

    /**
     * 点位地址
     */
    private String addr;

    /**
     * 设备编号
     */
    private String innerCode;
}
```

小程序微服务（lkd_microapp）的VMController有方法实现对售货机商品详情的查询

2.2.3 代码优化

3.openId

3.1 什么是**openId**

openId是用户在当前公众号下的唯一标识（‘身份证’），就是说通过这个openId，就能区分在这个公众号下具体是哪个用户。

官方提供了http接口地址为：
https://api.weixin.qq.com/sns/jscode2session?appid=APPID&secret=SECRET&js_code=JSCODE&grant_type=authorization_code

这是一个 HTTPS 接口，开发者服务器使用登录凭证 **code**获取 **session_key** 和 **openid**。其中 **session_key** 是对用户数据进行加密签名的密钥。为了自身应用安全，session_key 不应该在网络上传输。

请求参数：

参数	必填	说明
appid	是	小程序唯一标识
secret	是	小程序的 app secret
js_code	是	登录时获取的 code
grant_type	是	填写为 authorization_code

返回参数：

参数	说明
openid	用户唯一标识
session_key	会话密钥
unionid	用户在开放平台的唯一标识符。本字段在满足一定条件的情况下才返回。具体参看UnionID机制说明

返回说明：

```
//正常返回的JSON数据包
{
      "openid": "OPENID",
      "session_key": "SESSIONKEY"
      "unionid":  "UNIONID"
}
//错误时返回JSON数据包(示例为Code无效)
{
    "errcode": 40029,
    "errmsg": "invalid code"
}
```

3.2 代码实现

小程序微服务配置中心相关配置

```
wxpay:
  appId: xxxxxxxxxxxxx #微信支付商户平台里的appId
  appSecret: xxxxxxxxxxxxxxxx  #微信支付商户平台里的appSecret
```

小程序微服务读取配置类

```
package com.lkd.config;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties("wxpay")
@Data
public class WXConfig {
    private String appId;
    private String appSecret;
}
```

小程序微服务接口WXPayService定义getOpenId方法用于openId

```
package com.lkd.service;

/**
 * 微信服务接口
 */
public interface WxService {

    /**
     * 通过jsCode获取openId
     * @param jsCode
     * @return
     */
    String getOpenId(String jsCode);
}
```

WxServiceImpl实现类实现方法

```
package com.lkd.service.impl;
import com.google.common.base.Strings;
import com.lkd.config.WXConfig;
import com.lkd.service.WxService;
import com.lkd.utils.JsonUtil;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
@Service
@Slf4j
public class WxServiceImpl implements WxService {

    @Autowired
    private WXConfig wxConfig;

    @Override
    public String getOpenId(String jsCode) {
        String getOpenIdUrl = "https://api.weixin.qq.com/sns/jscode2session?
appid="+wxConfig.getAppId()+"&secret="+wxConfig.getAppSecret()+"&js_code="+jsCode+"&gran
t_type=authorization_code";
        RestTemplate restTemplate = new RestTemplate();
        String respResult = restTemplate.getForObject(getOpenIdUrl,String.class);

        log.info("weixin pay result:"+respResult);
        if( Strings.isNullOrEmpty(respResult)) return "";
        try{
            String errorCode = JsonUtil.getValueByNodeName("errcode",respResult) ;
            if(!Strings.isNullOrEmpty(errorCode)){
                int errorCodeInt = Integer.valueOf(errorCode).intValue();
                if(errorCodeInt != 0) return "";
            }
            return JsonUtil.getValueByNodeName("openid",respResult);
        }catch (Exception ex){
            log.error("获取openId失败",ex);
            return "";
        }
    }
}
```

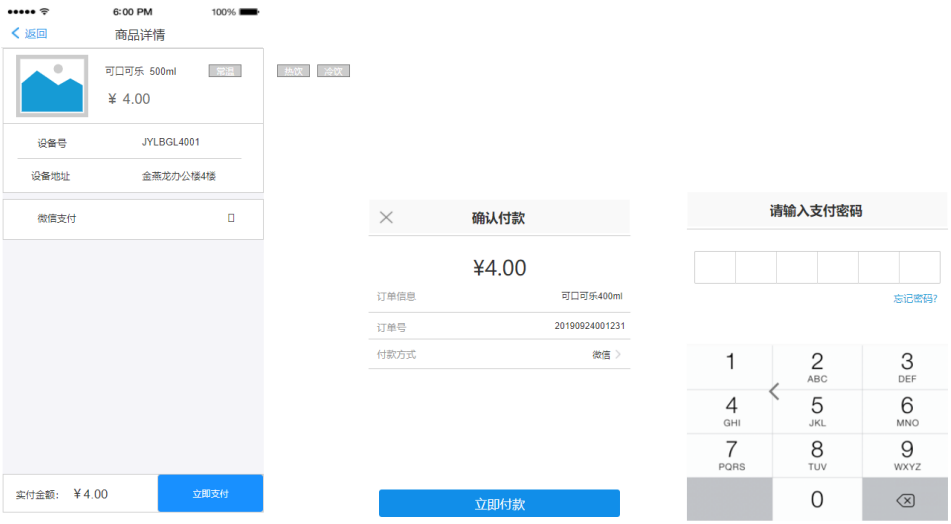
新建OrderController调用方法

```
@RestController
@RequestMapping("/order")
@Slf4j
public class OrderController {

    @Autowired
    private WxService wxService;

    /**
     * 获取openId
     * @param jsCode
     * @return
     */
    @GetMapping("/openid/{jsCode}")
    public String getOpenid(@PathVariable String jsCode){
        return wxService.getOpenId(jsCode);
    }
}
```

4. 小程序支付与回调

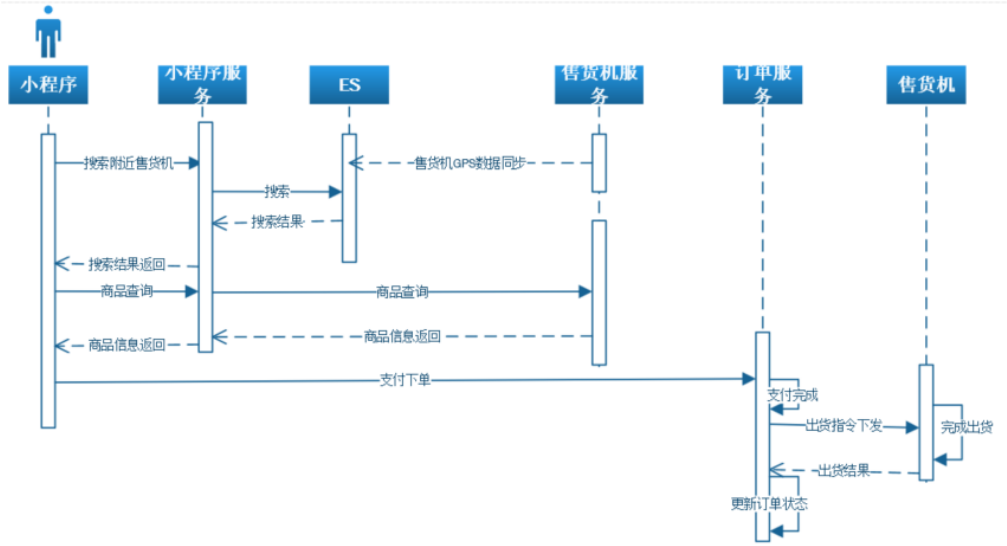
4.1 需求分析

当用户想购买某商品时，点击“立即支付”，点击“立即付款”，输入支付密码成功支付后更改订单状态

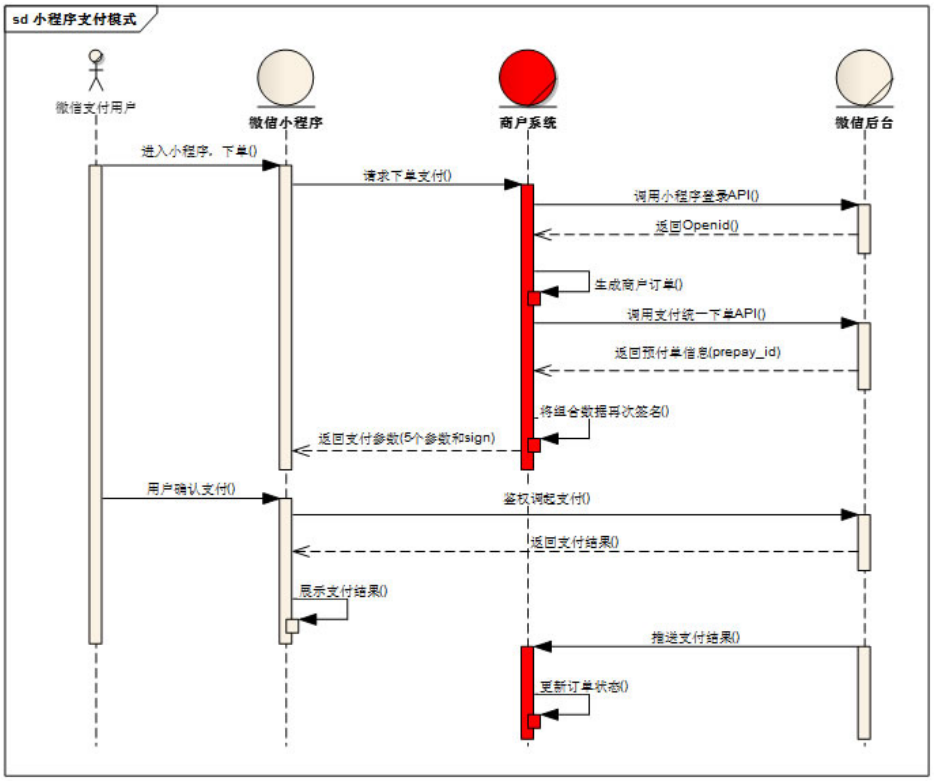


4.2 实现思路

整个业务的时序流程如下：



因为我们立可得2.0里采用了微信小程序支付，所以需要和微信支付平台做对接，微信小程序支付流程如下：



从上面的交互图可以看出微信小程序端需要：

（1）调用后台接口请求下单支付

（2）后台在调用微信统一下单接口之前先要获取openId，

（3）获取openId时需要客户端调用微信登录接口获取微信端返回的jsCode，并将该code传入后台，后台将该code传入微信端接口从而才能得到openId

（4）后台调用微信统一下单接口将相关数据传入到微信接口，此时微信平台会返回相关数据(其中包括prepay_id，也叫预付单信息)

（5）后台将得到的数据再次签名之后返回到前端(5个参数)

（6）前端根据后台返回的数据向微信端调起鉴权支付请求，如果通过则在小程序内部呼起微信支付

小程序调起支付数据签名字段列表：

字段名	变量名	必填	类型	示例值	描述
小程序ID	appId	是	String	wxd678efh567hg6787	微信分配的小程序ID
时间戳	timeStamp	是	String	1490840662	时间戳从1970年1月1日00:00:00至今时间
随机串	nonceStr	是	String	5K8264ILTKCH16CQ2502SI8ZNMTM67VS	随机字符串，不长于32位。推荐随机
数据包	package	是	String	prepay_id=*wx2017033010242291fcfe0db70013231072*	统一下单接口返回的 prepay_id 参数 prepay_id=*wx2017033010242291fc*
签名方式	signType	是	String	MD5	签名类型，默认为MD5，支持HMAC 注意此处需与统一下单的签名类型一

小程序内支付的具体的实现思路如下：

（1）在订单服务里封装和微信支付的相关接口，通过fegin暴露给小程序调用

（2） 在小程序中调用调用这些接口来实现支付的整个流程

4.3 代码实现

4.3.1 创建订单

（1）订单微服务lkd_order_service创建WxPayController（微信支付）

```
@RestController
@RequestMapping("/wxpay")
@Slf4j
public class WxPayController {

    @Autowired
    private OrderService orderService;

    /**
     * 微信小程序支付
     * @param requestPay
     * @return
     */
    @PostMapping("/requestPay")
    public String requestPay(@RequestBody RequestPay requestPay){
        CreateOrder createOrder = new CreateOrder();
        BeanUtils.copyProperties( requestPay,createOrder );
        createOrder.setPayType("2");//支付方式微信支付
        OrderEntity orderEntity = orderService.createOrder(createOrder);//创建订单
        //todo: 发起支付请求
        return "";
    }
}
```

（2）service_common工程com.lkd.feignService包下创建OrderService用于远程调用（FeignClient）

```
@FeignClient(value = "order-service")
public interface OrderService {
    @PostMapping("/wxpay/requestPay")
    String requestPay(@RequestBody RequestPay requestPay);
}
```

（3）服务降级类 OrderServiceFallbackFactory用于处理远程调用失败

```
@Component
@Slf4j
public class OrderServiceFallbackFactory implements FallbackFactory<OrderService> {
    @Override
    public OrderService create(Throwable throwable) {
        log.error("订单服务调用失败",throwable);
        return new OrderService() {
            @Override
            public String requestPay(RequestPay requestPay) {
                return null;
            }
        };
    }
}
```

修改OrderService，配置服务降级类

```
@FeignClient(value = "order-service",fallbackFactory =
OrderServiceFallbackFactory.class)
public interface OrderService {
    @PostMapping("/wxpay/requestPay")
    String requestPay(@RequestBody RequestPay requestPay);
}
```

（4）小程序微服务 lkd_microapp

```
@Autowired
private OrderService orderService;

@Autowired
private VMService vmService;

@Autowired
private WxService wxService;

/**
 * 小程序请求支付
 * @param requestPay
 * @return
 */
@PostMapping("/requestPay")
public String requestPay(@RequestBody RequestPay requestPay){
    if(!vmService.hasCapacity(requestPay.getInnerCode()
                          ,Long.valueOf(requestPay.getSkuId()))){
        throw new LogicException("该商品已售空");
    }
    //如果openId为空，则根据jsCode生成
    if(Strings.isNullOrEmpty(requestPay.getOpenId())){
        requestPay.setOpenId( wxService.getOpenId(requestPay.getJsCode()) );
    }
    String responseData = orderService.requestPay(requestPay);
    if(Strings.isNullOrEmpty(responseData)){
        throw new LogicException("微信支付接口调用失败");
    }
    return responseData;
}
```

4.3.2 对接微信支付

（1）在订单服务`lkd_order_service`中添加微信支付的相关依赖

```
<dependency>
    <groupId>com.github.wxpay</groupId>
    <artifactId>wxpay-sdk</artifactId>
    <version>3.0.9</version>
</dependency>
```

（2）在订单服务中的consul配置中心添加微信支付相关的配置：

```
wxpay:
  appId: xxxxxxxxxxxxxx #微信支付商户平台里的appId
  appSecret: xxxxxxxxxxxxxxxxx  #微信支付商户平台里的appSecret
  mchId: 1234567890 #商户号
  partnerKey: 123456 #商户的key
  notifyUrl: http://******/wxpay/payNotify #微信支付成功之后的回调地址
```

测试账号数据见配套资料

```
wxpay:
  appId: wxb709cf6e6a7d9d2a
  appSecret: d9a9ff00a633cd7353a8925119063b01
  mchId: 1473426802
  partnerKey: T6m9iK73b0kn9g5v426MKfHQH7X8rKwb
  notifyUrl: https://lkd2-java.itheima.net/api/order-service/wxpay/payNotify
```

（3）创建微信支付相关配置映射类

```
package com.lkd.conf;

import lombok.Data;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties("wxpay")
@Data
public class WXConfig {
    private String appId;
    private String appSecret;
    private String mchId;
    private String partnerKey;
    private String notifyUrl = "";
}
```

（4）添加微信支付sdk需要实现的配置类WxPaySdkConfig

```
package com.github.wxpay.sdk;

import com.lkd.conf.WXConfig;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.io.InputStream;

@Component
public class WxPaySdkConfig extends WXPayConfig {
    @Autowired
    private WXConfig wxConfig;

    public String getAppID() {
        return wxConfig.getAppId();
    }

    public String getMchID() {
        return wxConfig.getMchId();
    }

    public String getKey() {
        return wxConfig.getPartnerKey();
    }

     InputStream getCertStream() {
        return null;
    }

     IWXPayDomain getWXPayDomain() {
        return new IWXPayDomain() {
            public void report(String s, long l, Exception e) {

            }

            public DomainInfo getDomain(WXPayConfig wxPayConfig) {
                return new DomainInfo("api.mch.weixin.qq.com",true);
            }
        };
    }
}
```

4.3.3 发起支付请求

（1）订单微服务lkd_order_service新建WXPayService，定义requestPay方法

```java
/**
 * 微信支付服务接口
 */
public interface WXPayService {

    /**
     * 调用统一下单接口发起支付
     * @param openId
     * @param orderNo
     * @return
     */
    String requestPay(String openId,String orderNo);
}
```

创建实现类WXPayServiceImpl实现发起支付方法

```
package com.lkd.service.impl;

import com.github.wxpay.sdk.WXPayRequest;
import com.github.wxpay.sdk.WXPayUtil;
import com.github.wxpay.sdk.WxPaySdkConfig;
import com.google.common.base.Strings;
import com.google.common.collect.Maps;
import com.lkd.common.VMSystem;
import com.lkd.conf.WXConfig;
import com.lkd.entity.OrderEntity;
import com.lkd.exception.LogicException;
import com.lkd.service.OrderService;
import com.lkd.service.WXPayService;
import com.lkd.utils.JsonUtil;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.util.Map;

@Service
@Slf4j
public class WXPayServiceImpl implements WXPayService {
    @Autowired
    private WXConfig wxConfig;

    @Autowired
    private WxPaySdkConfig wxPaySdkConfig;

    @Autowired
    private OrderService orderService;

    @Override
    public String requestPay(String orderNo) {
        OrderEntity orderEntity = orderService.getByOrderNo(orderNo);
        try{
            String nonce_str = WXPayUtil.generateNonceStr();
            //1.封装请求参数
            Map<String,String> map= Maps.newHashMap();
            map.put("appid",wxPaySdkConfig.getAppID());//公众账号ID
            map.put("mch_id",wxPaySdkConfig.getMchID());//商户号
            map.put("nonce_str", nonce_str);//随机字符串
            map.put("body",orderEntity.getSkuName());//商品描述
            map.put("out_trade_no",orderNo);//订单号
            map.put("total_fee",orderEntity.getAmount()+"");//金额
            map.put("spbill_create_ip","127.0.0.1");//终端IP
            map.put("notify_url",wxConfig.getNotifyUrl());//回调地址
            map.put("trade_type","JSAPI");//交易类型
            map.put("openid",orderEntity.getOpenId());//openId
            String xmlParam  = WXPayUtil.generateSignedXml(map,
wxPaySdkConfig.getKey());
            System.out.println("参数："+xmlParam);
            //2.发送请求
            WXPayRequest wxPayRequest=new WXPayRequest(wxPaySdkConfig);
            String xmlResult = wxPayRequest.requestWithCert("/pay/unifiedorder", null,
xmlParam, false);
            //3.解析返回结果
            Map<String, String> mapResult = WXPayUtil.xmlToMap(xmlResult);
            //返回状态码
            String return_code = mapResult.get("return_code");
            //返回给移动端需要的参数
            Map<String, String> response = Maps.newHashMap();

            if(return_code.equals("SUCCESS")){
```

```
                // 业务结果
                String prepay_id = mapResult.get("prepay_id");//返回的预付单信息
                if(Strings.isNullOrEmpty(prepay_id)){
                    log.error("prepay_id is null","当前订单可能已经被支付");
                    throw new LogicException("当前订单可能已经被支付");
                }
                response.put("appId",wxConfig.getAppId());
                response.put("package", "prepay_id=" + prepay_id);
                response.put("signType","MD5");
                response.put("nonceStr", WXPayUtil.generateNonceStr());

                Long timeStamp = System.currentTimeMillis() / 1000;
                response.put("timeStamp", timeStamp + "");//要将返回的时间戳转化成字符串，
不然小程序端调用wx.requestPayment方法会报签名错误

                //再次签名，这个签名用于小程序端调用wx.requesetPayment方法
                String sign =
WXPayUtil.generateSignature(response,wxConfig.getPartnerKey());
                response.put("paySign", sign);
                response.put("appId","");
                response.put("orderNo",orderNo);
                return JsonUtil.serialize(response);
            }else {
                log.error("调用微信统一下单接口失败",response);
                return null;
            }
        }catch (Exception ex){
            log.error("调用微信统一下单接口失败",ex);
            return "";
        }
    }
}
```

（2）修改订单微服务（lkd_order_service）WxPayController 的requestPay方法

```
/**
 * 微信小程序支付
 * @param requestPay
 * @return
 */
@PostMapping("/requestPay")
public String requestPay(@RequestBody RequestPay requestPay){
    CreateOrder createOrder = new CreateOrder();
    BeanUtils.copyProperties( requestPay,createOrder );
    createOrder.setPayType("2");//支付方式微信支付
    OrderEntity orderEntity = orderService.createOrder(createOrder);
    return wxPayService.requestPay(orderEntity.getOrderNo());//调用发起支付请求
}
```

4.3.4 支付回调处理

（1）订单微服务（lkd_order_service）WXPayService新增方法定义

```
/**
 * 微信回调之后的处理
 * @param notifyResult
 * @throws Exception
 */
void notify(String notifyResult) throws Exception;
```

WXPayServiceImpl实现方法

```
@Override
public void notify(String notifyResult) throws Exception {
    //解析
    Map<String, String> map = WXPayUtil.xmlToMap( notifyResult );
    //验签
    boolean signatureValid = WXPayUtil.isSignatureValid(map, wxConfig.getPartnerKey());

    if(signatureValid){
        if("SUCCESS".equals(map.get("result_code"))){
            String orderNo = map.get("out_trade_no");
            OrderEntity orderEntity = orderService.getByOrderNo(orderNo);
            orderEntity.setStatus(VMSystem.ORDER_STATUS_PAYED);
            orderEntity.setPayStatus(VMSystem.PAY_STATUS_PAYED);
            orderService.updateById(orderEntity);
            //todo :支付完成通知出货
        }else {
            log.error("支付回调出错:"+notifyResult);
        }
    }else {
        log.error("支付回调验签失败:"+notifyResult);
    }
}
```

（2）订单微服务（lkd_order_service）WxPayController新增方法

```
/**
 * 微信支付回调接口
 * @param request
 * @return
 */
@RequestMapping("/payNotify")
@ResponseBody
public void payNotify(HttpServletRequest request, HttpServletResponse response){
    try {
        //输入流转换为xml字符串
        String xml = ConvertUtils.convertToString( request.getInputStream() );
        wxPayService.notify(xml);

        //给微信支付一个成功的响应
        response.setContentType("text/xml");
        String data = "<xml><return_code><![CDATA[SUCCESS]]></return_code><return_msg><!
[CDATA[OK]]></return_msg></xml>";
        response.getWriter().write(data);

    }catch (Exception e){
        log.error("支付回调处理失败",e);
    }
}
```

5. 售货机出货并发控制

5.1 需求分析

售货机支付出货和传统电商下单流程最大的区别是，售货机是具备实体的、独占式的。当售货机在处理一个用户的出货过程中，是不能同时处理另一个用户的出货的，并且同一时间只能处理一个商品的出货。所以在下单的接口里用分布式锁进行了排他处理，在用户并发请求下单时会请求分布式锁，只有获得锁的用户才会向微信支付平台发起支付请求，没有获得锁的用户会收到售货机忙碌的提示，这样可以避免引发用户支付完成拿不到商品退款的流程，提高了用户体验。

5.2 实现思路

（1）订单微服务

5.3 代码实现

5.3.1 发送出货通知

（1）OrderServiceImpl的sendVendout方法用于发送出货通知到售货机

```
/**
 * 出货
 * @param orderNo
 */
private void sendVendout(String orderNo){
    OrderEntity orderEntity = this.getByOrderNo(orderNo);

    VendoutReqData reqData = new VendoutReqData();
    reqData.setOrderNo(orderNo);
    reqData.setPayPrice(orderEntity.getAmount());
    reqData.setPayType(Integer.parseInt(orderEntity.getPayType()));
    reqData.setSkuId(orderEntity.getSkuId());
    reqData.setTimeout(60);
    reqData.setRequestTime(LocalDateTime.now().format(DateTimeFormatter.ISO_DATE_TIME));

    //向售货机发送出货请求
    VendoutReq req = new VendoutReq();
    req.setVendoutData(reqData);
    req.setSn(System.nanoTime());
    req.setInnerCode(orderEntity.getInnerCode());
    req.setNeedResp(true);

    try {
        mqttProducer.send(TopicConfig.TO_VM_TOPIC+orderEntity.getInnerCode(),2,req);
    } catch (JsonProcessingException e) {
        log.error("send vendout req error.",e);
    }
}
```

（2）OrderService定义payComplete方法，用于微信支付完成的处理

```
/**
 * 微信支付完成
 * @param orderNo
 * @return
 */
boolean payComplete(String orderNo);
```

OrderServiceImpl实现方法

```
@Override
public boolean payComplete(String orderNo) {
    sendVendout(orderNo);//发送出货通知
    return true;
}
```

（3）修改WXPayServiceImpl的notify方法，调用orderService.payComplete()方法

```
@Override
public void notify(String notifyResult) throws Exception {
    //解析
    Map<String, String> map = WXPayUtil.xmlToMap( notifyResult );
    //验签
    boolean signatureValid = WXPayUtil.isSignatureValid(map, wxConfig.getPartnerKey());

    if(signatureValid){
        if("SUCCESS".equals(map.get("result_code"))){
            String orderNo = map.get("out_trade_no");
            OrderEntity orderEntity = orderService.getByOrderNo(orderNo);
            orderEntity.setStatus(VMSystem.ORDER_STATUS_PAYED);
            orderEntity.setPayStatus(VMSystem.PAY_STATUS_PAYED);

            orderService.updateById(orderEntity);
            //支付完成通知出货
            orderService.payComplete(orderNo);
        }else {
            log.error("支付回调出错:"+notifyResult);
        }
    }else {
        log.error("支付回调验签失败:"+notifyResult);
    }
}
```

5.3.2 出货并发控制

（1）service_common的ConsulConfig用于读取配置文件中Consul的配置，有consulRegisterHost和consulRegisterPort两个属性

（2）DistributedLock 是分布式锁处理类，getLock用于获取锁。

业务实现：

（1）修改小程序微服务（lkd_microapp）OrderController，引入

```
@Autowired
private ConsulConfig consulConfig;

@Autowired
private RedisTemplate<String,String> redisTemplate;
```

（2）修改小程序微服务（lkd_microapp）OrderController类的requestPay方法，在调用orderService.requestPay方法前添加加锁的逻辑

```
//分布式锁，机器同一时间只能处理一次出货
DistributedLock lock = new DistributedLock(
        consulConfig.getConsulRegisterHost(),
        consulConfig.getConsulRegisterPort());
DistributedLock.LockContext lockContext = lock.getLock(requestPay.getInnerCode(),60);
if(!lockContext.isGetLock()){
    throw new LogicException("机器出货中请稍后再试");
}
//存入redis后是为了取消订单时释放锁
redisTemplate.boundValueOps(VMSystem.VM_LOCK_KEY_PREF+requestPay.getInnerCode())
        .set(lockContext.getSession(), Duration.ofSeconds(60));
```

5.3.4 取消订单释放锁

DistributedLock的releaseLock方法用于解锁

业务实现：

OrderController新增方法，用于取消订单

```
/**
 * 取消订单
 * @param innerCode
 */
@GetMapping("/cancelPay/{innerCode}/{orderNo}")
public void cancel(@PathVariable String innerCode,@PathVariable String orderNo){
    DistributedLock lock = new DistributedLock(
            consulConfig.getConsulRegisterHost(),
            consulConfig.getConsulRegisterPort());
    String sessionId = redisTemplate.boundValueOps(VMSystem.VM_LOCK_KEY_PREF +
innerCode).get();
    if(Strings.isNullOrEmpty(sessionId)) return;
    try {
        lock.releaseLock(sessionId);
        orderService.cancel(orderNo);
    }catch (Exception ex){
        log.error("取消订单出错",ex);
    }
}
```

6. 超时订单处理

6.1 需求分析

我们得立可得智能售货机系统中，当用户在小程序中开启支付之后，因为各种不确定的原因未能完成最终的付款，也未点击关闭支付页面的按钮，此时如果程序不处理的话，该订单就永远处于未支付状态，像这样状态的订单显然是一种未结束的订单状态，系统需要将这类的订单在10分钟之后自动处理掉，将订单的状态置于无效状态。

6.2 实现思路

（1）在订单服务中的创建订单方法里，订单创建完成后，将改订单的信息发送到EMQ里的延迟队列中，延迟处理时间为10分钟。

（2）在订单服务里订阅延迟队列主题下的消息，在收到消息后检查该订单的状态，如果订单还处于创建状态，则将订单置于无效状态。

6.3 代码实现

6.3.1 发送消息到延迟队列

（1）在`service_common`项目中定义要放入延迟队列中的消息对象：

```
package com.lkd.contract.server;

import com.lkd.contract.AbstractContract;
import lombok.Data;

import java.io.Serializable;

@Data
public class OrderCheck extends AbstractContract implements Serializable {
    public OrderCheck() {
        this.setMsgType("orderCheck");
    }
    private String orderNo;
}
```

（2）在订单服务项目`lkd_order_service`中的订单服务实现类`OrderServiceImpl`中的创建订单方法中添加向延迟队列发送消息的代码：

```
//将订单放到延迟队列中，10分钟后检查支付状态！！！！！！！！！！！！！！！！！！！
OrderCheck orderCheck = new OrderCheck();
orderCheck.setOrderNo(orderEntity.getOrderNo());
try {
  mqttProducer.send("$delayed/600/"+OrderConfig.ORDER_DELAY_CHECK_TOPIC,2,orderCheck);
} catch (JsonProcessingException e) {
  log.error("send to emq error",e);
}
```

6.3.2 接收消息修改订单状态

（1）在订单服务的配置中添加订阅该主题的配置如下：

```
mqtt:
  client:
    username: admin
    password: public
    serverURI: tcp://192.168.200.128:1883
    clientId: monitor.user${random.int[1000,9999]}
    keepAliveInterval: 10
    connectionTimeout: 30
  producer:
    defaultQos: 2
    defaultRetained: false
    defaultTopic: topic/test1
  consumer:
    consumerTopics: $share/g1/order/delayCheck #延时检查订单状态的topic
```

（2）在订单服务项目中实现接收到该消息的处理代码：

```java
package com.lkd.business;

import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.core.conditions.update.UpdateWrapper;
import com.lkd.annotations.ProcessType;
import com.lkd.common.VMSystem;
import com.lkd.contract.server.OrderCheck;
import com.lkd.entity.OrderEntity;
import com.lkd.service.OrderService;
import com.lkd.utils.JsonUtil;
import org.elasticsearch.common.Strings;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.io.IOException;

@Component
@ProcessType(value = "orderCheck")
public class OrderCheckHandler implements MsgHandler{
    @Autowired
    private OrderService orderService;

    @Override
    public void prosess(String jsonMsg) throws IOException {
        OrderCheck orderCheck = JsonUtil.getByJson(jsonMsg, OrderCheck.class);
        if(orderCheck == null || Strings.isNullOrEmpty(orderCheck.getOrderNo())) return;

        QueryWrapper<OrderEntity> qw = new QueryWrapper<>();
        qw
                .lambda()
                .eq(OrderEntity::getOrderNo,orderCheck.getOrderNo())
                .eq(OrderEntity::getStatus,VMSystem.ORDER_STATUS_CREATE);

        OrderEntity orderEntity = orderService.getOne(qw);
        if(orderEntity == null || orderEntity.getStatus() !=
VMSystem.ORDER_STATUS_CREATE) return;

        UpdateWrapper<OrderEntity> uw = new UpdateWrapper<>();
        uw
                .lambda()
                .eq(OrderEntity::getOrderNo,orderCheck.getOrderNo())
                .set(OrderEntity::getStatus, VMSystem.ORDER_STATUS_INVALID);
        orderService.update(uw);
    }
}
```