

闪聚支付 第3章讲义-C扫B支付

1 需求分析

1.1 C扫B的概念

C扫B，即顾客(Customer)扫描商户(Business)提供的二维码来完成支付。下图是支付宝提供的C扫B业务流程：

- 1、商家出示付款二维码
- 2、客户打开支付宝或微信的扫一扫，扫描二维码
- 3、确认支付，完成支付。



打开支付宝扫一扫



用户扫收款码付款



用户完成支付

C扫B支付分为两种方式：一是固定金额支付，顾客扫描后无需输入金额直接确认支付即可；另外一种输入金额，顾客扫描后需自己输入待支付的金额，然后完成支付。

什么是固定金额支付？

C扫B固定金额比较常见的就是在自动售货机购买饮料时，当你选择一个饮料后屏幕会显示一个二维码，咱们扫描后只需确认支付即可，无需自己输入饮料的价格，这种情况大家可以根据下面的交互流程图来自行实现。

什么是输入金额支付？

C扫B输入金额支付方式可以让买方自由输入金额，商户提前生成一个二维码，将二维码张贴在结账处，买方扫描二维码，输入当前消费的金额，完成支付。

1.2 业务流程

本章节实现C扫B输入金额支付，业务流程如下：



1. 商户点击组织管理-》门店管理-》打开门店列表页面

门店管理

+ 新建

选择应用

账户/用户名

请输入

查询

门店编号	门店名称	地址	二维码	操作
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	展示二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除

2. 选择应用

选择应用

默认支付应用

智慧学成

确定

3. 点击指定门店的生成二维码按钮

门店编号	门店名称	地址	二维码	操作
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	展示二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除

4. 顾客扫描生成的二维码，进入支付页面，输入金额来完成支付





1.3 需求列表

根据业务流程的分析，闪聚支付平台实现C扫B输入金额支付功能，需要完成以下需求：

- 1、为门店生成统一的支付二维码，用户扫一下二维码即可使用微信支付也可使用支付宝完成支付。
- 2、闪聚支付平台与微信、支付宝对接，闪聚支付作为中介，最终的支付动作（银行交易）仍通过微信、支付宝进行。
- 3、闪聚平台作为中介调用微信、支付宝的下单接口，完成支付。

2 支付接口技术预研

根据前边的需求分析，最重要的是闪聚支付平台作为中介，将用户的支付请求通过接口与微信、支付宝等第三方支付渠道进行对接，完成支付通道的聚合，所以首先需要调研微信、支付宝等第三方支付渠道的对接方式。

本项目首期上线要求集成微信和支付宝，下边对微信和支付宝的支付接口进行技术预研，包括：对接的流程，接口协议、接口测试等。

参考：闪聚支付-第3章-支付宝支付接入指南.pdf、闪聚支付-第3章-微信支付接入指南.pdf。

3 生成门店二维码



3.1 业务流程

1. 商户点击组织管理-》门店管理-》打开门店列表页面

一级菜单 / 二级菜单 / 当前页面

帮助 Jay.Liu

门店管理

+ 新建

选择应用

账户/用户名

查询

门店编号	门店名称	地址	二维码	操作
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	展示二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除

2. 选择应用

选择应用

默认支付应用

智慧学成

确定

3. 点击指定门店的生成二维码按钮

门店编号	门店名称	地址	二维码	操作
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	展示二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除
12351312	金燕龙店	北京市昌平区建材城西路	生成二维码	编辑 删除

3.2 生成二维码技术预研

ZXing是一个开源的，用Java编写的多格式的1D / 2D条码图像处理库，使用ZXing可以生成、识别QR Code（二维码）。常用的二维码处理库还有zbar，近几年已经不再更新代码，下边介绍ZXing生成二维码的方法。

1) 引入依赖

在shanjupay-common工程pom.xml中添加依赖：

```
<!-- 二维码生成&识别组件 -->
<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>core</artifactId>
    <version>3.3.3</version>
</dependency>

<dependency>
    <groupId>com.google.zxing</groupId>
    <artifactId>javase</artifactId>
    <version>3.3.3</version>
</dependency>
```

2) 生成二维码方法

复制二维码工具类QRCodeUtil.java到项目中

测试根据内容生成二维码方法，在QRCodeUtil中添加main方法如下：

```
public static void main(String[] args) throws IOException {
    QRCodeUtil qrCodeUtil = new QRCodeUtil();
    System.out.println(qrCodeUtil.createQRCode("http://www.itcast.cn/", 200, 200));
}
```

运行main方法，将输出的内容复制到浏览器地址后回车

```
data:image/png;base64,iVBORw0KGgoAAAANSUHEUgAAAMgAAADIAQAAACFI5MzAAABQE1EQVR42u2YPZKDMayF5aFIuUfIUThafDS0whEoUzC8fZKMySSbrVI8ZuICBX8uIvtZPxjeDfuSf81iPi7LFSgrzRTvV3XCKawXYLptFobviz6ZzB2xEfTjhyS90wXB3A7jbMSngLOQ0I4v2AZf96wqTWJ9+9/dYHSx2RYqfg/oqUgiX3nFBVfcCepcSbiJP67iwZ1G+5+Am7kyTzW90cW/kRAX+QJ953+uC18z05PV5UsaffUp8rqP5+jiySJU8jtNxcNrysetCNK6A/V41EQeU+xa0eZREE1tOTpFYod0VKXsKCqvRqMkW5pkza8Ggy3WgEuTvZcz0dcUBc+9MneL1DqkXjQz0eaZA1LqVtmzmCffTKPiPwz1mh2zkGyNwtT9kguTVI7LWv6u17DCp0jX9iaGV66HDny/ZL1WfILfc/hMHLUpekAAAAASUVORK5CYII=
```



使用手机扫描二维码，即可自动打开传智播客官网

3.3 门店列表

3.3.1 商户服务查询门店列表

3.3.1.1 接口定义

1、接口描述

1) 根据商户id和分页信息查询门店列表

2、接口定义如下：MerchantService


```
/**
 * 分页条件查询商户下门店
 * @param storeDTO
 * @param pageNo
 * @param pageSize
 * @return
 */
PageVO<StoreDTO> queryStoreByPage(StoreDTO storeDTO, Integer pageNo, Integer pageSize);
```

3.3.1.2 接口实现

3、在MerchantServiceImpl中实现queryStoreByPage方法：

```
@Override
public PageVO<StoreDTO> queryStoreByPage(StoreDTO storeDTO, Integer pageNo, Integer pageSize) {
    // 创建分页
    Page<Store> page = new Page<>(pageNo, pageSize);
    // 构造查询条件
    QueryWrapper<Store> qw = new QueryWrapper();
    if (null != storeDTO && null != storeDTO.getMerchantId()) {
        qw.lambda().eq(Store::getMerchantId, storeDTO.getMerchantId());
    }
    // 执行查询
    IPage<Store> storeIPage = storeMapper.selectPage(page, qw);
    // entity List转DTO List
    List<StoreDTO> storeList = StoreConvert.INSTANCE.listentity2dto(storeIPage.getRecords());
    // 封装结果集
    return new PageVO<>(storeList, storeIPage.getTotal(), pageNo, pageSize);
}
```

3.3.2 商户平台应用查询门店列表

3.3.2.1 接口定义

1、接口描述

1) 请求商户服务查询门店列表

2、接口定义如下：StoreController

```
package com.shanjupay.merchant.controller;

@Api(value = "商户平台-门店管理", tags = "商户平台-门店管理", description = "商户平台-门店的增删改查")
@RestController
public class StoreController {

    @ApiOperation("分页条件查询商户下门店")
    @ApiImplicitParams({
```

```
@ApiImplicitParam(name = "pageNo", value = "页码", required = true, dataType = "int",
paramType = "query"),
@ApiImplicitParam(name = "pageSize", value = "每页记录数", required = true, dataType =
"int", paramType = "query"))
@PostMapping("/my/stores/merchants/page")
public PageVO<StoreDTO> queryStoreByPage(@RequestParam Integer pageNo, @RequestParam Integer
pageSize){
}
}
```

3.3.2.2 接口实现

前端JS在Long长度大于17位时会出现精度丢失的问题，由于项目中门店ID的长度会超过17位，所以在此处添加注解将返回给前端的门店ID自动转为string类型

1) 使用jackson来完成自动转换，在shanjupay-merchant-api工程中添加依赖：

```
<dependency>
<groupId>com.fasterxml.jackson.core</groupId>
<artifactId>jackson-databind</artifactId>
<version>2.9.9</version>
<scope>compile</scope>
</dependency>
```

2) 在StoreDTO中添加注解：

```
@ApiModelProperty("门店Id")
@JsonSerialize(using= ToStringSerializer.class)
private Long id;
```

2、在StoreController中实现queryStoreByPage方法：

```
@GetMapping("/my/stores/merchants/page")
public PageVO<StoreDTO> queryStoreByPage(@RequestParam Integer pageNo, @RequestParam Integer
pageSize){
    //获取商户id
    Long merchantId = SecurityUtil.getMerchantId();
    StoreDTO storeDTO = new StoreDTO();
    storeDTO.setMerchantId(merchantId);
    //分页查询
    PageVO<StoreDTO> stores = merchantService.queryStoreByPage(storeDTO, pageNo, pageSize);
    return stores;
}
```

2.3.4.3 接口测试

由于已经接入SaaS，请求统一走网关的端口56010。

1 首先请求认证获取token，及租户的id

1) 启动三个SaaS服务

2) 使用账号申请token

2 使用Postman：POST <http://localhost:56010/merchant/my/stores/merchants/page?pageNo=1&pageSize=20> 查询门店列表

注意在header中添加 Authorization及tenantId。

门店列表查询

POST <http://localhost:56010/merchant/my/stores/merchants/page?pageNo=1&pageSize=10> Params Send

Authorization Headers (2) Body Pre-request Script Tests

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQ..			
<input checked="" type="checkbox"/> tenantId	2			
New key	Value	Description		

Body Cookies (4) Headers (9) Test Results Status: 200 OK Time: 1086 ms

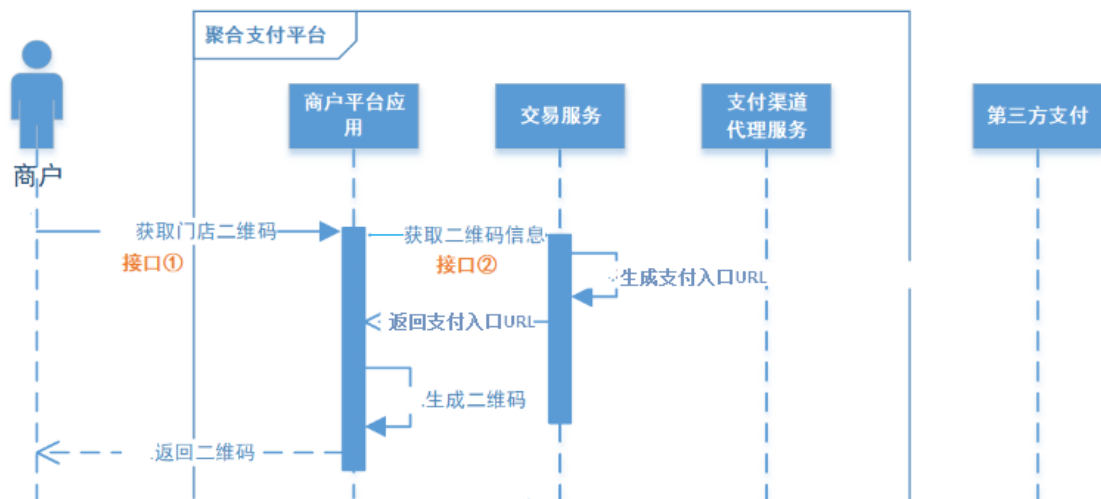
Pretty Raw Preview JSON

```
1 {
2   "items": [
3     {
4       "id": "1207918068689518593",
5       "storeName": "根门店",
6       "storeAddress": null,
7       "storeNumber": null,
8       "storeStatus": null,
9       "merchantId": 1207918068433666049,
10      "parentId": null
11    }
12  ],
13  "counts": 1,
14  "page": 1,
```

3.4 生成二维码

3.4.1 系统交互流程

生成二维码的系统交互 流程如下：



- 1、商户登录商户应用平台，查询门店列表
- 2、商户平台 请求交易 服务获取门店二维码URL
- 3、商户平台 根据 URL生成二维码

3.4.2交易服务生成二维码URL

3.4.2.1 接口定义

接口描述：

生成门店的c扫b二维码

接口参数：

输入：商户id、应用id、门店id、标题，内容

输出：支付入口

- 1、在交易服务的api工程 创建dto

```
@Data
@NoArgsConstructor
public class QRCodeDto implements Serializable {

    private Long merchantId;
    private String appId;
    private Long storeId;
    private String subject;//商品标题
    private String body;//订单描述

}
```

- 2、接口定义如下

```
/**
 * 交易订单相关服务接口
 */
public interface TransactionService {

    /**
     * 生成门店二维码
     * @param qrCodeDto, 传入merchantId, appId、storeId、channel、subject、body
     * @return 支付入口URL, 将二维码的参数组成json并用base64编码
     * @throws BusinessException
     */
    String createStoreQRCode(QRCodeDto qrCodeDto) throws BusinessException;

}
```

3.4.2.2 商户服务应用合法校验接口

商户生成二维码需要根据门店、应用来生成，设计接口需要对应用和门店的合法性来校验。

1、校验该应用是否属于该商户。

2、校验该门店是否属于该商户

1、接口描述

1) 根据商户id和应用id查询应用信息，查询到则说明合法。

2、在AppService下定义接口如下

```
/**
 * 查询应用是否属于某个商户
 * @param appId
 * @param merchantId
 * @return
 */
Boolean queryAppInMerchant(String appId, Long merchantId);
```

3、接口实现如下

在AppServiceImpl中实现queryAppInMerchant接口

```
/**
 * 查询应用是否属于某个商户
 *
 * @param appId
 * @param merchantId
 * @return
 */
@Override
public Boolean queryAppInMerchant(String appId, Long merchantId) {
    Integer count = appMapper.selectCount(new LambdaQueryWrapper<App>().eq(App::getAppId,
appId)
        .eq(App::getMerchantId, merchantId));
    return count>0;
}
```

3.4.2.3 商户服务门店合法校验接口

1、接口描述

1) 根据商户id和门店id查询门店，查询到则说明合法。

2、在MerchantService中定义接口：

```
/**
 * 查询门店是否属于某商户
 * @param storeId
 * @param merchantId
 * @return
 */
Boolean queryStoreInMerchant(Long storeId, Long merchantId);
```

3、接口实现如下

在MerchantServiceImpl中实现queryStoreInMerchant接口

```
/**
 * 查询门店是否属于某商户
 *
 * @param storeId
 * @param merchantId
 * @return
 */
@Override
public Boolean queryStoreInMerchant(Long storeId, Long merchantId) {
    Integer count = storeMapper.selectCount(new LambdaQueryWrapper<Store>().eq(Store::getId,
storeId)
        .eq(Store::getMerchantId, merchantId));
    return count>0;
}
```

3.4.2.4 接口实现

在Nacos中添加支付入口配置：transaction-service.yaml

支付入口是扫码支付的统一入口。

```
#支付入口url
shanjupay:
  payurl: "http://127.0.0.1:56010/transaction/pay-entry/"
```

实现createStoreQRCode接口：

```
@Slf4j
@org.apache.dubbo.config.annotation.Service
public class TransactionServiceImpl implements TransactionService {
    /**
     * 支付入口url
     */
    @Value("${shanjupay.payurl}")
    private String payurl;

    @Reference
    MerchantService merchantService;

    @Reference
    AppService appService;

    /**
     * 生成门店二维码
     * @param qrCodeDto 商户id、应用id、门店id、标题、内容
     * @return
     * @throws BusinessException
     */
    @Override
    public String createStoreQRCode(QRCodeDto qrCodeDto) throws BusinessException {

        //校验应用和门店
        verifyAppAndStore(qrCodeDto.getMerchantId(), qrCodeDto.getAppId(), qrCodeDto.getStoreId());
        //1. 生成支付信息
        PayOrderDTO payOrderDTO = new PayOrderDTO();
        payOrderDTO.setMerchantId(qrCodeDto.getMerchantId());
        payOrderDTO.setAppId(qrCodeDto.getAppId());
        payOrderDTO.setStoreId(qrCodeDto.getStoreId());
        payOrderDTO.setSubject(qrCodeDto.getSubject()); //显示订单标题
        payOrderDTO.setChannel("shanju_c2b"); //服务类型
        payOrderDTO.setBody(qrCodeDto.getBody()); //订单内容
        String jsonString = JSON.toJSONString(payOrderDTO);
        log.info("transaction service createStoreQRCode, jsonString is {}", jsonString);

        //将支付信息保存到票据中
```



```
String ticket = EncryptUtil.encodeUTF8StringBase64(jsonString);
//支付入口
String payEntryUrl = payurl +ticket;
log.info("transaction service createStoreQRCode,pay-entry is {}",payEntryUrl);
return payEntryUrl;
}

/**
 * 校验应用和门店是否属于当前登录商户
 * @param merchantId
 * @param appId
 * @param storeId
 */
private void verifyAppAndStore(Long merchantId,String appId,Long storeId) {
    //判断应用是否属于当前商户
    Boolean contains = appService.queryAppInMerchant(appId, merchantId);
    if (!contains) {
        throw new BusinessException(CommonErrorCode.E_200005);
    }
    //判断门店是否属于当前商户
    Boolean containsStore = merchantService.queryStoreInMerchant(storeId, merchantId);
    if (!containsStore) {
        throw new BusinessException(CommonErrorCode.E_200006);
    }
}
}
```

3.4.3 商户平台应用生成二维码

3.4.3.1 接口实现

1、配置参数

定义c扫b二维码的标题和内容



* Data ID: merchant-application.yaml

* Group: SHANJUPAY_GROUP

[更多高级选项](#)

描述: null

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 ? :

```
19 |  
20 | shanjupay:  
21 |   c2b:  
22 |     subject: "%s商品"  
23 |     body: "向%s付款"
```

内容如下：

```
shanjupay:  
  c2b:  
    subject: "%s商品"  
    body: "向%s付款"
```

2、定义接口

```
/**  
 * 门店二维码订单标题  
 */  
@Value("${shanjupay.c2b.subject}")  
private String subject;  
  
/**  
 * 门店二维码订单内容  
 */  
@Value("${shanjupay.c2b.body}")  
private String body;  
  
@ApiOperation("生成商户应用门店二维码")  
@ApiImplicitParams({  
    @ApiImplicitParam(name = "appId", value = "商户应用id", required = true, dataType =  
"String", paramType = "path"),  
    @ApiImplicitParam(name = "storeId", value = "商户门店id", required = true, dataType =  
"String", paramType = "path"),  
})  
@GetMapping(value = "/my/apps/{appId}/stores/{storeId}/app-store-qr-code")  
  
public String createCScanBStoreQRCode(@PathVariable String appId, @PathVariable Long
```

```

storeId) throws BusinessException {
    //商户id
    Long merchantId = SecurityUtil.getMerchantId();
    //生成二维码链接
    QRCodeDto qrCodeDto = new QRCodeDto();
    qrCodeDto.setMerchantId(merchantId);
    qrCodeDto.setAppId(appId);
    qrCodeDto.setStoreId(storeId);
    //标题
    MerchantDTO merchantDTO = merchantService.queryMerchantById(merchantId);
    //"%s 商品"
    qrCodeDto.setSubject(String.format(subject,merchantDTO.getMerchantName()));
    //内容,格式: "向%s 付款"
    qrCodeDto.setBody(String.format(body,merchantDTO.getMerchantName()));

    String storeQRCodeUrl = transactionService.createStoreQRCode(qrCodeDto);
    log.info("[merchantId:{},appId:{},storeId:{}]createCScanBStoreQRCode is
    {}",merchantId,appId,storeId,storeQRCodeUrl);
    try {
        //根据返回url,调用生成二维码工具类,生成二维码base64返回
        return QRCodeUtil.createQRCode(storeQRCodeUrl, 200, 200);
    } catch (IOException e) {
        throw new BusinessException(CommonErrorCode.E_200007);
    }
}
    
```

3.4.3.2 测试

1、请求认证，获取token及租户id

2、请求获取二维码

注意：应用及门店的合法性

► 生成门店二维码

GET

http://localhost:56010/merchant/my/apps/e7c58be3f47740898d725bfd21f8529d/stores/1210775286885425153/app-store-qrcode

Params

Send

Authorization

Headers (2)

Body

Pre-request Script

Tests

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdW...			
<input checked="" type="checkbox"/> tenantId	3			
New key	Value	Description		

Body

Cookies (4)

Headers (9)

Test Results

Status: 200 OK Time: 9026 ms

Pretty

Raw

Preview

Text

Copy

Find

Search

```

1 ·PKHxcBk7AYa24mGM+I3h/xICk/SByStOW2QEiAceoZr7cN2FnWhoy0wWvPhEjtPFR/Tux8PK6M4f1UcMrESTh1j9f8e8EgVWe8LhbrjuRtQiYM6ftkPz/f6T/nvw8peEp454
    
```

4 支付入口

4.1 需求分析

买方扫描门店二维码，进入支付入口 即进入订单确认页面，流程如下：

- 1) 顾客扫描二维码
- 2) 进入订单确认页面



The image shows a mobile app payment confirmation screen. At the top, there is a navigation bar with a blue arrow and the text '< 返回' (Return), the title '向商家付款' (Pay to merchant), and three dots. Below the navigation bar is a large blue rectangular area with the text '常西湖分店' (Changxi Lake Branch) in white. Underneath this is a form with three input fields. The first field is labeled '商品名称:' (Goods name) and contains the text '常西湖分店'. The second field is labeled '订单内容:' (Order content) and contains the text '特购商品'. The third field is labeled '付款金额(元):' (Payment amount in Yuan) and is empty. At the bottom of the form is a large blue button with the text '立即支付' (Pay immediately).

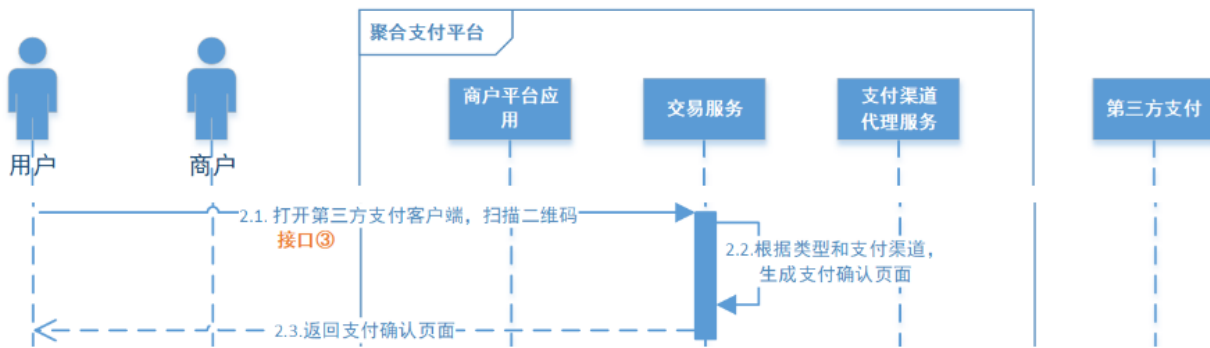
4.2 Freemarker技术预研

支付确认页面由服务端渲染生成，常用的技术有jsp、freemarker velocity Thymeleaf等。

本项目采用freemarker模板引擎，参考freemarker基础v1.1.pdf。

4.3 交易服务支付入口

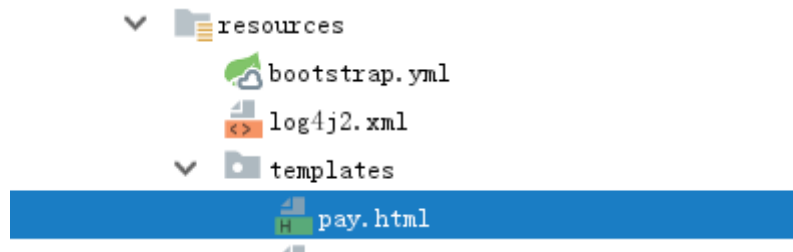
交互流程如下：



- 1、顾客扫描二维码，请求交易服务支付入口
- 2、交易服务解析请求，生成支付确认页面
- 3、交易服务向服务响应支付确认页面

4.3.1 支付确认页面

- 1、从资料->代码拷贝pay.html、pay_error.html到交易服务工程下。



- 2、在交易服务接口实现工程的pom.xml中引入依赖

```
<!-- freemarker依赖 -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-freemarker</artifactId>
</dependency>
```

- 3、在Nacos中配置spring-boot-freemarker.yaml，Group: COMMON_GROUP

```
#freemarker基本配置
spring:
  freemarker:
    charset: UTF-8
    request-context-attribute: rc
    content-type: text/html
    suffix: .html
    enabled: true
  resources:
    add-mappings: false #关闭工程中默认的资源处理
  mvc:
    throw-exception-if-no-handler-found: true #出现错误时直接抛出异常
```

在shanjupay-transaction-service工程的bootstrap.yml引入spring-boot-freemarker.yaml

```
config:
  server-addr: ${nacos.server.addr} # 配置中心地址
  file-extension: yaml
  namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8 #2ed00aaa-b760-4171-baa9-83d92e509a41 #
  group: SHANJUPAY_GROUP # 聚合支付业务组
  ext-config:
    -
      refresh: true
      data-id: spring-boot-http.yaml # spring boot http配置
      group: COMMON_GROUP # 通用配置组
    -
      refresh: true
      data-id: spring-boot-starter-druid.yaml # spring boot starter druid配置
      group: COMMON_GROUP # 通用配置组
    -
      refresh: true
      data-id: spring-boot-mybatis-plus.yaml # spring boot mybatisplus配置
      group: COMMON_GROUP # 通用配置组
    -
      refresh: true
      data-id: spring-boot-redis.yaml # redis配置
      group: COMMON_GROUP # 通用配置组
    -
      refresh: true
      data-id: spring-boot-freemarker.yaml # spring boot freemarker配置
      group: COMMON_GROUP # 通用配置组
```

4、新建WebMvcConfig配置确认支付页面视图名：

```
package com.shanjupay.transaction.config;

@Component
public class WebMvcConfig implements WebMvcConfigurer {
    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/pay-page").setViewName("pay");
    }
}
```

5、定义支付入口接口

注意：PayController要向前端响应页面，使用@Controller注解。

```
@Slf4j
@Controller
public class PayController {

    /**
     * 支付入口
     */
}
```

```
* @param ticket
* @param type
* @param request
* @return
*/
@RequestMapping(value = "/pay-entry/{ticket}")
public String payEntry(@PathVariable("ticket") String ticket,HttpServletRequest request) {

    return "forward:/pay-page";
}
}
```

6、测试页面渲染

1) 在nacos配置网关转发到交易服务的路由

* Data ID:

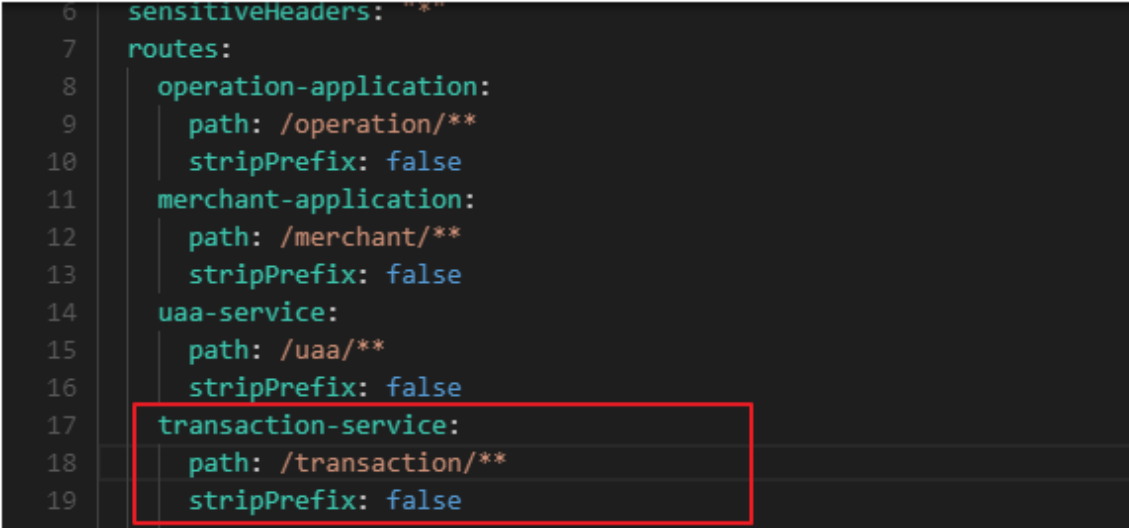
* Group:

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 ? : 

```
6 sensitiveHeaders: **
7 routes:
8   operation-application:
9     path: /operation/**
10    stripPrefix: false
11   merchant-application:
12     path: /merchant/**
13    stripPrefix: false
14   uaa-service:
15     path: /uaa/**
16    stripPrefix: false
17   transaction-service:
18     path: /transaction/**
19    stripPrefix: false
```

2) 用chrome浏览器访问 (用手机视图模式) 二维码的统一入口地址。

<http://127.0.0.1:56010/transaction/pay-entry/...>

注意：由于没有数据，订单信息均为空。



The image shows a mockup of a payment confirmation page. It features a large blue rectangular area at the top, likely for a logo or header. Below this, there are three input fields: '商品名称:' (Product Name), '订单描述:' (Order Description), and '付款金额(元):' (Payment Amount in Yuan). Each field has a light gray border and a small blue icon on the right. At the bottom of the form is a prominent blue button with the white text '立即支付' (Pay Now).

4.3.2 页面完善

前边测试的支付确认页面由于没有数据所以显示为空，下边对页面进行完善。

1、页面完善需求如下：

- 1) 进入支付入口，传入ticket，需要解析ticket得到订单信息，在支付确认页面显示。
- 2) 解析出客户端类型，目前支付微信、支付宝两种。

4.3.2.1 解析ticket

拷贝资料-->代码下的PayOrderDTO.java、PayOrderConvert.java到交易服务工程。

解析ticket代码如下：

```
@RequestMapping(value = "/pay-entry/{ticket}")
public String payEntry(@PathVariable("ticket") String ticket, HttpServletRequest request) {
    //将ticket的base64还原
    String ticketStr = EncryptUtil.decodeUTF8StringBase64(ticket);
    //将ticket ( json ) 转成对象
    PayOrderDTO order = JSON.parseObject(ticketStr, PayOrderDTO.class);
    //将对象转成url格式
    String params = ParseURLPairUtil.parseURLPair(order);
    return "forward:/pay-page?" + params;
}
```

4.3.2.2 解析客户端类型

- 1、拷贝到资料-->代码下的BrowserType.java到交易服务controller包下，此类是系统定义的客户端配置类型。
- 2、修改支付入口代码：

```
@RequestMapping(value = "/pay-entry/{ticket}")
public String payEntry(@PathVariable("ticket") String ticket,HttpServletRequest request) {
    try {
        //将ticket的base64还原
        String ticketStr = EncryptUtil.decodeUTF8StringBase64(ticket);
        //将ticket ( json ) 转成对象
        PayOrderDTO order = JSON.parseObject(ticketStr, PayOrderDTO.class);
        //将对象转成url格式
        // String url = toParamsString(order);

        BrowserType browserType = BrowserType.valueOfUserAgent(request.getHeader("user-agent"));
        switch (browserType) {
            case ALIPAY: //直接跳转收银台pay.html
                return "forward:/pay-page?" + ParseURLPairUtil.parseURLPair(order);
            case WECHAT: //获取授权码(待实现)
                return "forward:/pay-page?" + ParseURLPairUtil.parseURLPair(order);
            default:
        }
    } catch (Exception e){
        e.printStackTrace();
        log.error(e.getMessage(),e);
    }

    return "forward:/pay-page-error";
}
```

4.3.3 接口测试

由于添加上了客户端类型的解析，使用微信或支付宝扫码方可进入支付入口，需要使用模拟器进行测试。

1、修改支付入口地址

修改为模拟器可以访问到的地址，模拟器安装在开发电脑上，支付入口地址修改为开发电脑局域网的地址。

```
C:\Users\Administrator>ipconfig

Windows IP 配置

以太网适配器 Bluetooth 网络连接:

    媒体状态 . . . . . : 媒体已断开
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 无线网络连接:

    连接特定的 DNS 后缀 . . . . . : DHCP HOST
    本地链接 IPv6 地址 . . . . . : fe80::7cd7:870c:f7b2:f7d4%12
    IPv4 地址 . . . . . : 192.168.1.100
    子网掩码 . . . . . : 255.255.255.0
    默认网关 . . . . . : 192.168.1.1
```


* Data ID: transaction-service.yaml

* Group: SHANJUPAY_GROUP

[更多高级选项](#)

描述: null

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 ? :

```
server:
  time-zone: Asia/Shanghai
  use-ssl: false
11   username: root
12   password: mysql
13
14 # 覆盖spring-boot-mybatis-plus.yaml的项目
15 mybatis-plus:
16   typeAliasesPackage: com.shanjupay.transaction.entity
17   mapper-locations: classpath:com/shanjupay/*/mapper/*.xml
18
19 #支付入口url
20 shanjupay:
21   payurl: "http://192.168.1.100/transaction"
22
```

2、生成门店二维码

3、使用模拟器运行支付宝沙箱APP，扫描二维码查看支付确认页面



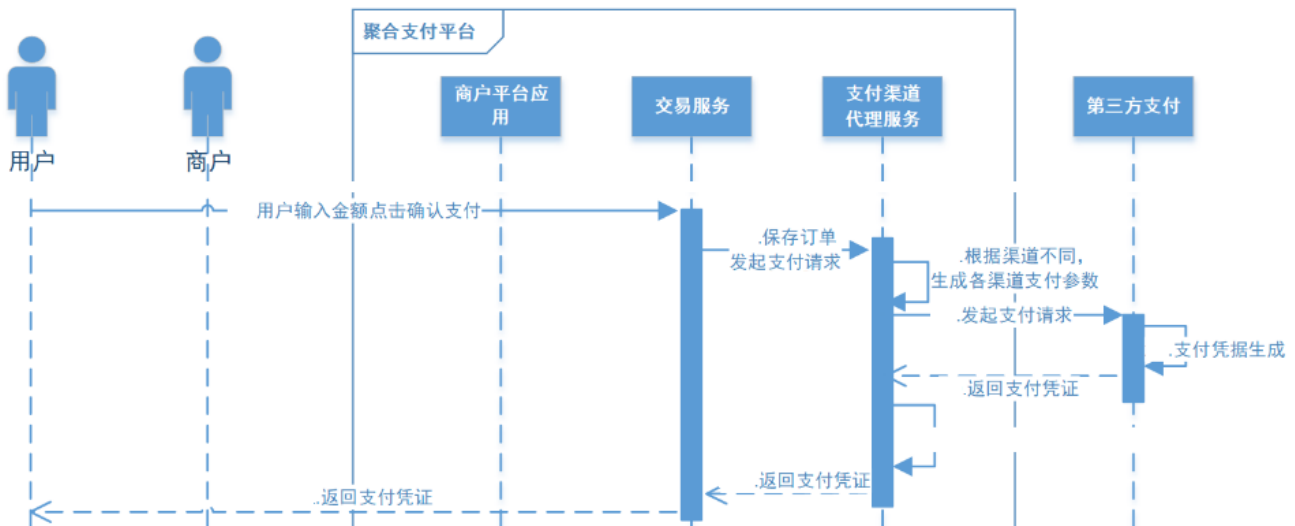
5 立即支付

5.1 需求分析

顾客扫码进入支付确认页面，输入金额，点击立即支付，打开支付客户端（微信或支付宝），输入支付密码完成支付。

立即支付需要调用第三方支付渠道的统一下单接口，本章节完成支付宝统一下单接口对接。

交互流程如下：



支付渠道代理服务介绍：

有支付需求的微服务统一通过支付渠道代理服务调用“第三方支付服务”提供的接口，这样做的好处由支付渠道代理服务将第三方支付系统和闪聚支付内部服务进行解耦合。

整体执行流程如下：

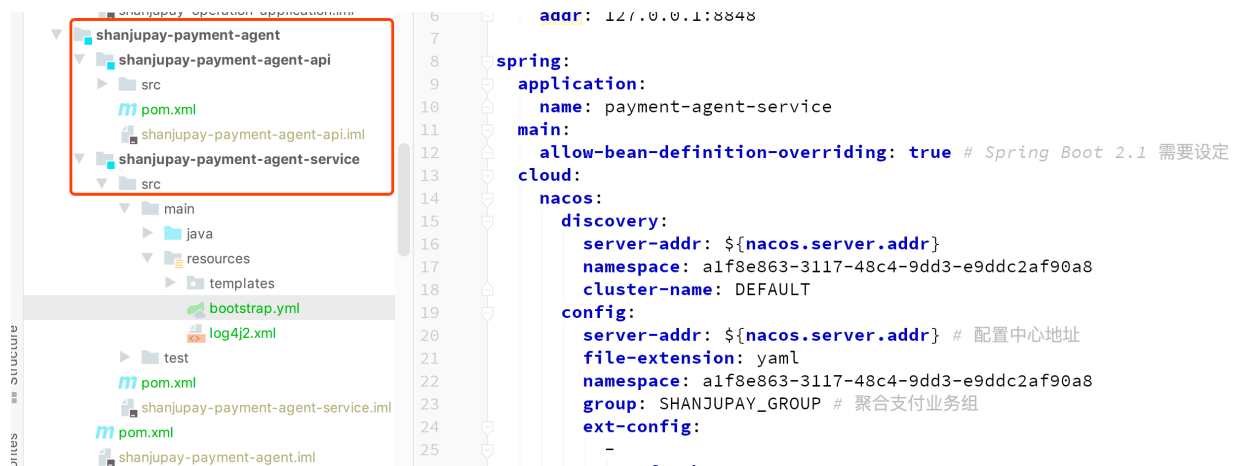
- 1、顾客输入金额，点击立即支付
- 2、请求交易服务，交易服务保存订单
- 3、交易服务调用支付渠道代理服务的支付宝下单接口
- 4、支付渠道代理服务调用支付宝的统一下单接口。
- 5、支付凭证返回

5.2 搭建支付渠道代理工程

支付渠道代理服务包括如下工程：

服务名	职责
支付渠道代理服务API(shanjupay-payment-agent-api)	定义支付渠道代理服务提供的接口
支付渠道代理服务(shanjupay-payment-agent-service)	实现支付渠道代理服务的所有接口

1. 复制提供的shanjupay-payment-agent目录到shanjupay根目录
2. 添加Module到IDEA中



3. 在Nacos中添加payment-agent-service.yaml配置，Group: SHANJUPAY_GROUP

```

server:
  servlet:
    context-path: /payment-receiver
  
```

4. 打开shanjupay-payment-agent-service工程的bootstrap.yml，将其中的namespace替换为自己创建的dev命名空间ID

```

spring:
  application:
    name: payment-agent-service
  main:
    allow-bean-definition-overriding: true # Spring Boot 2.1 需要设定
  cloud:
    nacos:
      discovery:
        server-addr: ${nacos.server.addr}
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        cluster-name: DEFAULT
      config:
        server-addr: ${nacos.server.addr} # 配置中心地址
        file-extension: yaml
        namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        group: SHANJUPAY_GROUP # 聚合支付业务组
  
```

5. 启动PaymentAgentBootstrap测试：

```
4      import ...
7
8      @SpringBootApplication
9      @EnableDiscoveryClient
10     public class PaymentAgentBootstrap {
11
12         public static void main(String[] args) {
13             SpringApplication.run(PaymentAgentBootstrap.class, args);
14         }
15
16     }
17
```

5.3 支付渠道代理服务支付宝下单

5.3.1 接口定义

5.3.1.1 支付宝接口参数

支付渠道代理服务调用支付宝手机网站下单接口，下边梳理接口的参数：

公共参数如下：

标记蓝色的由sdk设置、标记红色的已在支付渠道参数配置中（需要支付渠道代理服务接口处理），标记绿色的需支付渠道代理服务接口处理。

参数	类型	是否必填	最大长度	描述	示例值
app_id	String	是	32	支付宝分配给开发者的应用ID	2014072300007148
method	String	是	128	接口名称	alipay.trade.wap.pay
format	String	否	40	仅支持JSON	JSON
return_url	String	否	256	HTTP/HTTPS开头字符串	https://m.alipay.com/Gk8NF23
charset	String	是	10	请求使用的编码格式，如utf-8,gbk,gb2312等	utf-8
sign_type	String	是	10	商户生成签名字符串所使用的签名算法类型，目前支持RSA2和RSA，推荐使用RSA2	RSA2

sign	String	是	256	签名串，详见 签名	详见示例
timestamp	String	是	19	发送请求的时间，格式"yyyy-MM-dd HH:mm:ss"	2014-07-24 03:07:50
version	String	是	3	调用的接口版本，固定为：1.0	1.0
notify_url	String	否	256	支付宝服务器主动通知商户服务器里指定的页面http/https路径。	<code>https://api.x.com/receive_notify.htm</code>
biz_content	String	是	-	业务请求参数的集合，最大长度不限，除公共参数外所有请求参数都必须放在这个参数中传递，具体参照各产品快速接入文档	

业务参数如下：

参数	类型	是否必填	最大长度	描述	示例值
body	String	否	128	对一笔交易的具体描述信息。如果是多种商品，请将商品描述字符串累加传给body。	Iphone6 16G
subject	String	是	256	商品的标题/交易标题/订单标题/订单关键字等。	大乐透
out_trade_no	String	是	64	商户网站唯一订单号	70501111111S001111119

timeout_express	String	否	6	该笔订单允许的最晚付款时间，逾期将关闭交易。取值范围：1m~15d。m-分钟，h-小时，d-天，1c-当天（1c-当天的情况下，无论交易何时创建，都在0点关闭）。该参数数值不接受小数点，如1.5h，可转换为90m。注：若为空，则默认为15d。	90m
total_amount	Price	是	9	订单总金额，单位为元，精确到小数点后两位，取值范围[0.01,100000000]	9.00
auth_token	String	否	40	针对用户授权接口，获取用户相关数据时，用于标识用户授权关系注：若不属于支付宝业务经理提供签约服务的商户，暂不对外提供该功能，该参数使用无效。	appopenBb64d181d0146481ab6a762c00714cC27
product_code	String	是	64	销售产品码，商家和支付宝签约的产品码。该产品请填写固定值：QUICK_WAP_WAY	QUICK_WAP_WAY

passback_params	String	否	512	公用回传参数，如果请求时传递了该参数，则返回给商户时会回传该参数。支付宝会在异步通知时将该参数原样返回。本参数必须进行UrlEncode之后才可以发送给支付宝	merchantBizType%3d3C%26 merchantBizNo%3d201601010 1111
-----------------	--------	---	-----	---	--

5.3.1.2 接口定义

1、接口描述：

调用支付宝手机wap下单接口

2、接口定义

在PayChannelAgentService中定义接口：

```

package com.shanjupay.paymentagent.api;

public interface PayChannelAgentService {

    /**
     * 调用支付宝手机WAP下单接口
     * @param aliConfigParam 支付渠道参数
     * @param alipayBean 请求支付参数
     * @return
     * @throws BusinessException
     */
    PaymentResponseDTO createPayOrderByAliWAP(AliConfigParam aliConfigParam, AlipayBean alipayBean)
        throws BusinessException;
}
    
```

5.3.2 接口实现

```

@org.apache.dubbo.config.annotation.Service
@Slf4j
public class PayChannelAgentServiceImpl implements PayChannelAgentService {

    /**
     * 调用支付宝手机WAP下单接口
     *
     * @param aliConfigParam 支付渠道参数
     * @param alipayBean 请求支付参数
    
```

```
* @return
* @throws BusinessException
*/
@Override
public PaymentResponseDTO createPayOrderByAliWAP(AliConfigParam aliConfigParam, AlipayBean
alipayBean) throws BusinessException {
    log.info("支付宝请求参数", alipayBean.toString());

    //支付宝渠道参数
    String gateway = aliConfigParam.getUrl();//支付宝下单接口地址
    String appId = aliConfigParam.getAppId();//appid
    String rsaPrivateKey = aliConfigParam.getRsaPrivateKey();//私钥
    String format = aliConfigParam.getFormat();//数据格式json
    String charest = aliConfigParam.getCharest();//字符编码
    String alipayPublicKey = aliConfigParam.getAlipayPublicKey();//公钥
    String signtype = aliConfigParam.getSigntype();//签名算法类型
    String notifyUrl = aliConfigParam.getNotifyUrl();//支付结果通知地址
    String returnUrl = aliConfigParam.getReturnUrl();//支付完成返回商户地址

    //支付宝sdk客户端
    AlipayClient client = new DefaultAlipayClient(gateway, appId, rsaPrivateKey, format,
charest,
        alipayPublicKey, signtype);

    // 封装请求支付信息
    AlipayTradeWapPayRequest alipayRequest = new AlipayTradeWapPayRequest();
    AlipayTradeWapPayModel model = new AlipayTradeWapPayModel();
    model.setOutTradeNo(alipayBean.getOutTradeNo());//闪聚平台订单
    model.setSubject(alipayBean.getSubject());//订单标题
    model.setTotalAmount(alipayBean.getTotalAmount());//订单金额
    model.setBody(alipayBean.getBody());//订单内容
    model.setTimeoutExpress(alipayBean.getExpireTime());//订单过期时间
    model.setProductCode(alipayBean.getProductCode());//商户与支付宝签定的产品码，固定为
QUICK_WAP_WAY
    alipayRequest.setBizModel(model);//请求参数集合
    String jsonString = JSON.toJSONString(alipayBean);
    log.info("createPayOrderByAliWAP..alipayRequest:{})", jsonString);
    // 设置异步通知地址
    alipayRequest.setNotifyUrl(notifyUrl);
    // 设置同步地址
    alipayRequest.setReturnUrl(returnUrl);

    try {
        // 调用SDK提交表单
        AlipayTradeWapPayResponse response = client.pageExecute(alipayRequest);
        log.info("支付宝手机网站支付预支付订单信息" + response);
        PaymentResponseDTO res = new PaymentResponseDTO();
        res.setContent(response.getBody());
        return res;
    } catch (Exception e) {
        e.printStackTrace();
        throw new BusinessException(CommonErrorCode.E_400002);//支付宝确认支付失败
    }
}
```



```
}  
}
```

5.4 交易服务支付宝下单

5.4.1 接口定义

交易服务支付宝下单是提供给支付入口请求的支付宝付款接口，当用户用支付宝客户端扫描二维码进入确认支付页面，点击确认支付即将请求此接口。

1、接口描述

- 1) 接收前端支付请求
- 2) 保存订单信息到闪聚支付平台
- 3) 调用支付渠道代理服务请求支付宝下单接口
- 4) 将支付宝下单接口响应结果返回到前端，前端开始进行支付

2、接口定义

在PayController中定义接口如下：

定义OrderConfirmVO接收前端请求的支付参数：

```
package com.shanjupay.transaction.vo;  
  
import io.swagger.annotations.ApiModel;  
import lombok.Data;  
  
@ApiModel(value = "OrderConfirmVO", description = "订单确认信息")  
@Data  
public class OrderConfirmVO {  
  
    private String appId; //应用id  
    private String tradeNo; //交易单号  
    private String openId; //微信openid  
    private String storeId; //门店id  
    private String channel; //服务类型  
    private String body; //订单描述  
    private String subject; //订单标题  
    private String totalAmount; //金额  
  
}
```

接口定义如下：

```
@ApiOperation("支付宝门店下单付款")
@PostMapping("/createAliPayOrder")
public void createAlipayOrderForStore(OrderConfirmVO orderConfirmVO, HttpServletRequest request,
    HttpServletResponse response)
    throws Exception {

}
```

5.4.2 接口实现

5.4.2.1 保存订单

- 1、拷贝资料--》代码中的IdWorkerUtils.java、PaymentUtil.java工具类到common 工程。
- 2、在TransactionServiceImpl中编写submitOrderByAli方法。

```
/**
 * 支付宝订单保存
 *
 * @param payOrderDTO
 * @return
 */
@Override
public PaymentResponseDTO submitOrderByAli(PayOrderDTO payOrderDTO) throws BusinessException {
    //保存订单
    payOrderDTO.setPayChannel("ALIPAY_WAP");
    //保存订单
    payOrderDTO = save(payOrderDTO);
    //调用支付代理服务请求第三方支付系统
    //...
    return null;
}
```

- 3、编写保存订单的方法，此方法由

```
/**
 * 保存订单到闪聚平台
 *
 * @param payOrderDTO
 * @return
 */
private PayOrderDTO save(PayOrderDTO payOrderDTO) throws BusinessException {
    PayOrder entity = PayOrderConvert.INSTANCE.dto2entity(payOrderDTO);
    entity.setTradeNo(PaymentUtil.genUniquePayOrderNo());
    //订单创建时间
    entity.setCreateTime(LocalDateTime.now());
    //设置过期时间，30分钟
    entity.setExpireTime(LocalDateTime.now().plus(30, ChronoUnit.MINUTES));
    entity.setCurrency("CNY");//设置支付币种
    entity.setTradeState("0");//订单状态
}
```

```
int insert = payOrderMapper.insert(entity);

return PayOrderConvert.INSTANCE.entity2dto(entity);
}
```

5.4.2.2 请求代理服务调用支付宝下单

在TransactionServiceImpl中编写私有方法，实现请求代理服务调用支付宝下单接口。

此私有方法被submitOrderByAli方法调用。

```
//调用支付宝下单接口
private PaymentResponseDTO alipayH5(String tradeNo) {
    //构建支付实体
    AlipayBean alipayBean = new AlipayBean();

    //根据订单号查询订单详情
    PayOrderDTO payOrderDTO = queryPayOrder(tradeNo);
    alipayBean.setOutTradeNo(tradeNo);
    alipayBean.setSubject(payOrderDTO.getSubject());
    String totalAmount = null; //支付宝那边入参是元
    try {
        //将分转成元
        totalAmount = AmountUtil.changeF2Y(payOrderDTO.getTotalAmount().toString());
    } catch (Exception e) {
        e.printStackTrace();
        throw new BusinessException(CommonErrorCode.E_300006);
    }
    alipayBean.setTotalAmount(totalAmount);
    alipayBean.setBody(payOrderDTO.getBody());
    alipayBean.setStoreId(payOrderDTO.getStoreId());
    alipayBean.setExpireTime("30m");

    //根据应用、服务类型、支付渠道查询支付渠道参数
    PayChannelParamDTO payChannelParamDTO =
    payChannelService.queryParamByAppPlatformAndPayChannel(payOrderDTO.getAppId(),
        payOrderDTO.getChannel(), "ALIPAY_WAP");
    if(payChannelParamDTO == null){
        throw new BusinessException(CommonErrorCode.E_300007);
    }
    //支付宝渠道参数
    AliConfigParam aliConfigParam = JSON.parseObject(payChannelParamDTO.getParam(),
    AliConfigParam.class);
    //字符编码
    aliConfigParam.setCharest("utf-8");
    PaymentResponseDTO payOrderResponse = payChannelAgentService
        .createPayOrderByAliWAP(aliConfigParam, alipayBean);
    log.info("支付宝H5支付响应Content:" + payOrderResponse.getContent());
}
```

```
        return payOrderResponse;  
    }  
}
```

定义根据订单号查询订单信息。

```
/**  
 * 根据订单号查询订单信息  
 * @param tradeNo  
 * @return  
 */  
@Override  
public PayOrderDTO queryPayOrder(String tradeNo) {  
    PayOrder payOrder = payOrderMapper  
        .selectOne(new QueryWrapper<PayOrder>().lambda().eq(PayOrder::getTradeNo, tradeNo));  
    return PayOrderConvert.INSTANCE.entity2dto(payOrder);  
}
```

完善submitOrderByAli方法，调用alipayH5方法：

```
public PaymentResponseDTO submitOrderByAli(PayOrderDTO payOrderDTO) throws BusinessException {  
    //保存订单  
    payOrderDTO.setPayChannel("ALIPAY_WAP");  
    payOrderDTO = save(payOrderDTO);  
    //调用支付代理服务请求第三方支付系统  
    return alipayH5(payOrderDTO.getTradeNo());  
}
```

5.4.2.3 完善交易服务下单接口

完善createAliPayOrder接口，调用submitOrderByAli提交支付宝订单。

```
@ApiOperation("支付宝门店下单付款")  
@PostMapping("/createAliPayOrder")  
public void createAlipayOrderForStore(OrderConfirmVO orderConfirmVO, HttpServletRequest request,  
    HttpServletResponse response)  
    throws Exception {  
    if (StringUtils.isBlank(orderConfirmVO.getAppId())) {  
        throw new BusinessException(CommonErrorCode.E_300003);  
    }  
  
    PayOrderDTO payOrderDTO = PayOrderConvert.INSTANCE.vo2dto(orderConfirmVO);  
  
    payOrderDTO.setTotalAmount(Integer.valueOf(AmountUtil.changeY2F(orderConfirmVO.getTotalAmount()))  
);  
    payOrderDTO.setClientIp(IPUtil.getIpAddr(request));  
    //获取下单应用信息  
    AppDTO app = appService.getAppById(payOrderDTO.getAppId());  
    //设置所属商户  
    payOrderDTO.setMerchantId(app.getMerchantId());  
    PaymentResponseDTO payOrderResult = transactionService.submitOrderByAli(payOrderDTO);  
  
    String content = String.valueOf(payOrderResult.getContent());  
}
```

```
log.info("支付宝H5支付响应的结果：" + content);
response.setContentType("text/html;charset=UTF-8");
response.getWriter().write(content); //直接将完整的表单html输出到页面
response.getWriter().flush();
response.getWriter().close();
}
```

5.4.3 测试

1、生成二维码

注意：二维码的URL可以被模拟器访问。

2、扫码进入支付入口，进入支付确认页面

3、输入金额，点击确认支付。

订单写入闪聚平台数据库。

调用支付宝下单接口是否成功。

6 获取支付结果

6.1 需求分析

获取支付结果的需求包括如下几个方面：

1、服务间异步通信

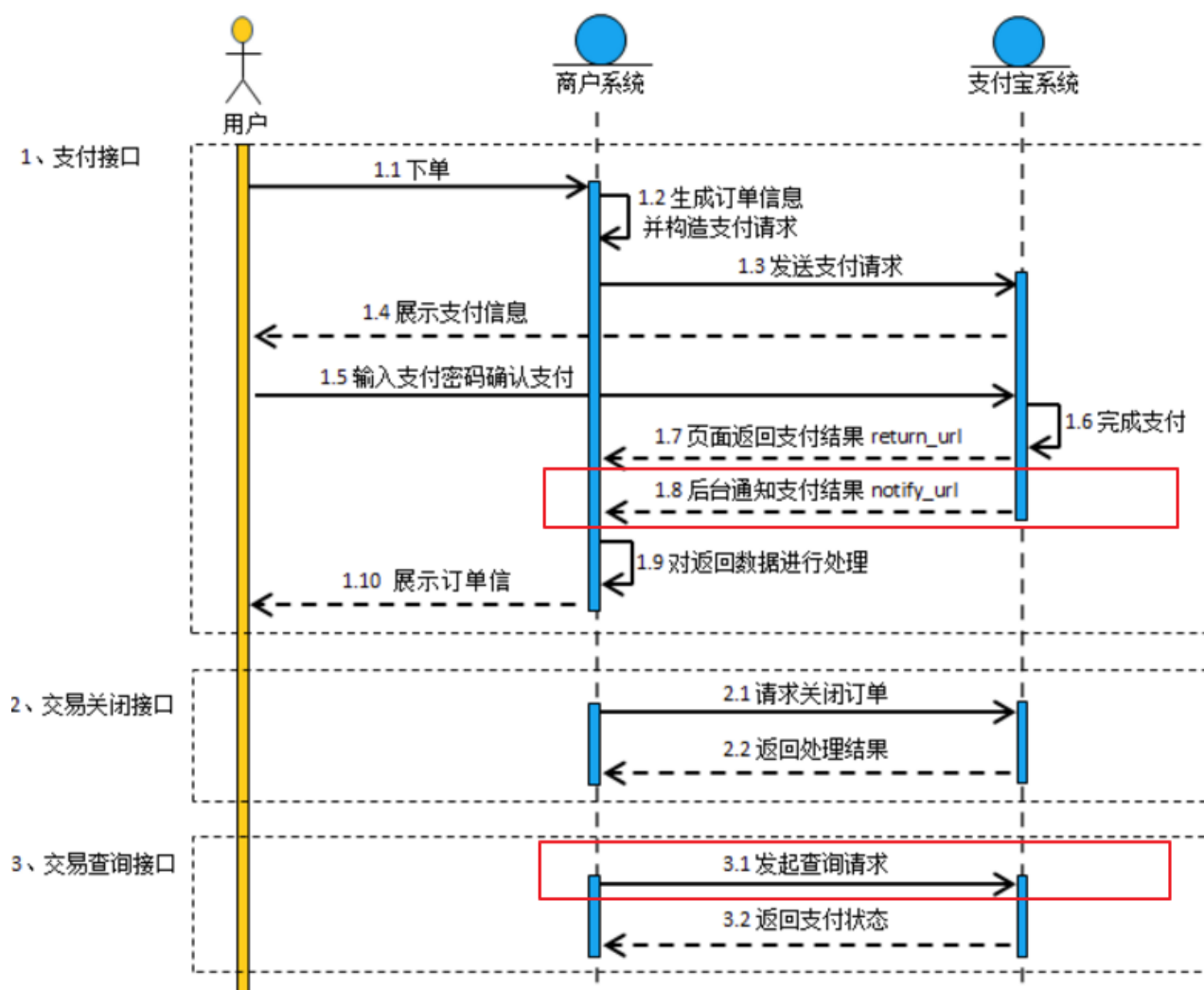
顾客支付完成后，平台需要及时得到支付结果并更新数据库中的订单状态。根据微服务职责划分，支付渠道代理服务负责与支付宝、微信接口对接，交易服务负责维护订单的数据，支付渠道代理服务如何把查询到的订单结果通知给交易服务呢？项目中会采用消息队列来完成。

2、实现第三方支付系统支付结果查询接口

完成支付后第三方支付系统提供两种方式获取支付结果，如下图：

1) 第三方支付系统主动通知闪聚支付平台支付结果。

2) 闪聚支付平台主动从第三方支付系统查询支付结果。



以上两种方法在实际项目中可以都用，其中第二种是必须用的，因为第一种是由第三方支付系统主动通知闪聚支付平台，当调用闪聚平台接口无法通信达到一定的次数后第三方支付系统将不再通知。

本项目支付渠道代理服务集成第二种方法完成支付结果的查询。

3、下单成功延迟向第三方支付系统查询支付结果

在调用第三方支付下单接口之后此时用户正在支付中，所以需要延迟一定的时间再去查询支付结果。

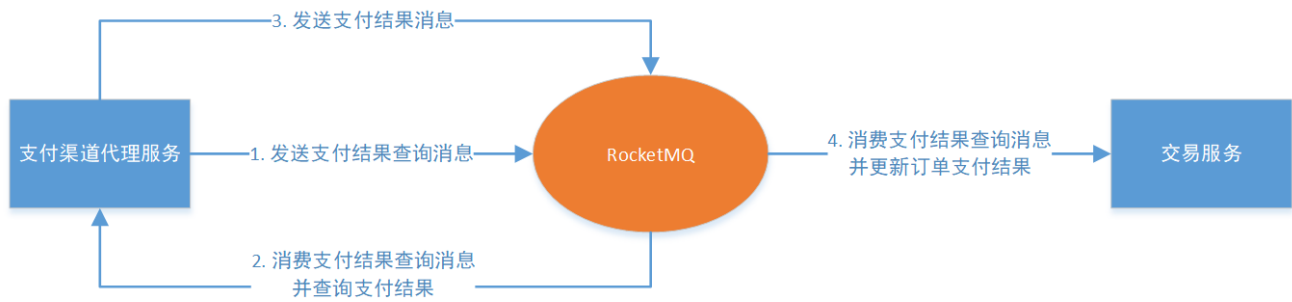
如果查询支付结果还没有支付再继续等待一定的时间再去查询，当达到订单的有效期还没有支付则不再查询。

6.2 RocketMQ技术预研

参考：RocketMQ研究v1.0.pdf。

6.3 技术方案

项目使用消息队列RocketMQ完成支付渠道代理服务与交易服务之间的通信，如下图：



- 0、支付渠道代理服务调用第三方支付下单接口。（此时顾客开始输入密码进行支付）
- 1、支付渠道代理向消息队列发送一条延迟消息（查询支付结果），消费方仍是支付渠道代理服务。
- 2、支付渠道代理接收消息，调用支付宝接口查询支付结果
- 3、支付渠道代理查询到支付结果，将支付结果发送至MQ，消费方是交易服务。
- 4、交易服务接收到支付结果消息，更新订单状态。

6.4 支付渠道代理查询支付宝交易状态

6.4.1 支付宝交易状态查询接口

参考手机网支付产品介绍文档（<https://docs.open.alipay.com/203>），查看[alipay.trade.query](#)交易状态查询接口文档。

接口名称	描述
alipay.trade.wap.pay 手机网页支付接口	通过此接口传入订单参数，同时唤起支付宝手机网页支付页面
alipay.trade.close 交易关闭接口	通过此接口关闭此前已创建的交易，关闭后，用户将无法继续付款。仅能关闭创建后未支付的交易。
alipay.trade.query 交易状态查询接口	通过此接口查询某笔交易的状态，交易状态：交易创建，等待买家付款；未付款交易超时关闭，或支付完成后全额退款；交易支付成功；交易结束，不可退款。
alipay.trade.refund 交易退款接口	通过此接口对单笔交易完成退款操作
alipay.trade.fastpay.refund.query 退款查询	查询退款订单的状态
alipay.data.dataservice.bill.downloadurl.query 账单查询接口	调用此接口获取账单的下载链接

6.4.1.1 请求参数

公共请求参数基本都是支付渠道设置的参数，实现方法参考支付宝下单接口即可。

公共请求参数

参数	类型	是否必填	最大长度	描述	示例值
app_id	String	是	32	支付宝分配给开发者的应用ID	2014072300007148
method	String	是	128	接口名称	alipay.trade.query
format	String	否	40	仅支持JSON	JSON
charset	String	是	10	请求使用的编码格式，如utf-8,gbk,gb2312等	utf-8
sign_type	String	是	10	商户生成签名字符串所使用的签名算法类型，目前支持RSA2和RSA，推荐使用RSA2	RSA2
sign	String	是	344	商户请求参数的签名串，详见 签名	详见示例
timestamp	String	是	19	发送请求的时间，格式"yyyy-MM-dd HH:mm:ss"	2014-07-24 03:07:50
version	String	是	3	调用的接口版本，固定为：1.0	1.0
app_auth_token	String	否	40	详见 应用授权概述	
biz_content	String	是		请求参数的集合，最大长度不限，除公共参数外所有请求参数都必须放在这个参数中传递，具体参照各产品快速接入文档	

业务请求参数中out_trade_no即闪聚支付平台订单号，根据此订单号查询支付状态。

请求参数

参数	类型	是否必填	最大长度	描述	示例值
out_trade_no	String	特殊可选	64	订单支付时传入的商户订单号,和支付宝交易号不能同时为空。 trade_no,out_trade_no如果同时存在优先取trade_no	20150320010101001
trade_no	String	特殊可选	64	支付宝交易号，和商户订单号不能同时为空	2014112611001004680073956707
org_pid	String	可选	16	银行间联模式下有用，其它场景请不要使用； 双联通过该参数指定需要查询的交易所属收单机构的pid;	2088101117952222
query_options	String[]	可选	10	查询选项，商户通过上送该字段来定制查询返回信息	TRADE_SETTLE_INFO

6.4.1.2 响应参数

支付宝交易状态查询接口的响应参数如下：

公共响应参数

参数	类型	是否必填	最大长度	描述	示例值
code	String	是	-	网关返回码,详见 文档	40004
msg	String	是	-	网关返回码描述,详见 文档	Business Failed
sub_code	String	否	-	业务返回码，参见具体的API接口文档	ACQ.TRADE_HAS_SUCCESS
sub_msg	String	否	-	业务返回码描述，参见具体的API接口文档	交易已被支付
sign	String	是	-	签名,详见 文档	DZXh8eeTuAHOYE3w1J+POiPhfDxOYBfUNn1lkeTV7P4zJdyojWEa6lZs6Hz0yDW5CpviufUb5l0V5WENS3OYR8zRedqo6D+fUTdLHdc+EFyCkiQhBxIzgngPdPdfp1PIS7BdhhsZhbRqb7o4k3Dxc+AAAnFauu4V6Zdwczo=

响应参数

参数	类型	是否必填	最大长度	描述	示例值
trade_no	String	必填	64	支付宝交易号	2013112011001004330000121536
out_trade_no	String	必填	64	商家订单号	6823789339978248
buyer_logon_id	String	必填	100	买家支付宝账号	159****5620
trade_status	String	必填	32	交易状态：WAIT_BUYER_PAY（交易创建，等待买家付款）、TRADE_CLOSED（未付款交易超时关闭，或支付完成后全额退款）、TRADE_SUCCESS（交易支付成功）、TRADE_FINISHED（交易结束，不可退款）	TRADE_CLOSED
total_amount	Price	必填	11	交易的订单金额，单位为元，两位小数。该参数的值为支付时传入的total_amount	88.88

以上参数主要解析code和trade_status：

1、根据code判断接口请求是否成功

参考：<https://docs.open.alipay.com/common/105806>

2、根据trade_status判断具体的支付状态

交易状态如下：

WAIT_BUYER_PAY（交易创建，等待买家付款）

TRADE_CLOSED（未付款交易超时关闭，或支付完成后全额退款）

TRADE_SUCCESS（交易支付成功）

TRADE_FINISHED（交易结束，不可退款）

6.4.2 支付渠道代理接口定义

6.4.2.1 接口定义

接口描述：

1) 使用支付宝SDK发起支付结果查询请求 2) 返回查询结果

在PayChannelAgentService定义如下接口：

```
/**
 * 支付宝交易状态查询
 * @param aliConfigParam 支付渠道参数
 * @param outTradeNo 闪聚平台订单号
 * @return
 */
PaymentResponseDTO queryPayOrderByAli(AliConfigParam aliConfigParam, String outTradeNo);
```

6.4.2.2 接口实现

参考支付宝官方提供的样例代码与支付宝通信。https://docs.open.alipay.com/api_1/alipay.trade.query

1) 定义支付宝查询返回状态码常量类：AliCodeConstants

```
package com.shanjupay.paymentagent.common.constant;

/**
 * 支付宝查询返回状态码常量类
 */
public class AliCodeConstants {

    public static final String SUCCESSCODE = "10000"; // 支付成功或接口调用成功
    public static final String PAYINGCODE = "10003"; // 用户支付中
    public static final String FAILEDCODE = "40004"; // 失败
    public static final String ERRORCODE = "20000"; // 系统异常

    /**
     * 支付宝交易状态
     *    WAIT_BUYER_PAY ( 交易创建，等待买家付款 )
     *    TRADE_CLOSED ( 未付款交易超时关闭，或支付完成后全额退款 )
     *    TRADE_SUCCESS ( 交易支付成功 )
     *    TRADE_FINISHED ( 交易结束，不可退款 )
     */

    public static final String WAIT_BUYER_PAY = "WAIT_BUYER_PAY";
    public static final String TRADE_CLOSED = "TRADE_CLOSED";
    public static final String TRADE_SUCCESS = "TRADE_SUCCESS";
    public static final String TRADE_FINISHED = "TRADE_FINISHED";

}
```

2) sdk示例如下

参考sdk示例：<https://docs.open.alipay.com/api/1/alipay.trade.query/>

```
AlipayClient alipayClient = new
DefaultAlipayClient("https://openapi.alipay.com/gateway.do", "app_id", "your
private_key", "json", "GBK", "alipay_public_key", "RSA2");
AlipayTradeQueryRequest request = new AlipayTradeQueryRequest();
request.setBizContent("{\"" +
    "\"out_trade_no\": \"20150320010101001\", " +
    "\"trade_no\": \"2014112611001004680 073956707\", " +
    "\"org_pid\": \"2088101117952222\", " +
    "    \"query_options\": [" +
    "        \"TRADE_SETTLE_INFO\"" +
    "    ]" +
    "}");
AlipayTradeQueryResponse response = alipayClient.execute(request);
if(response.isSuccess()){
    System.out.println("调用成功");
} else {
    System.out.println("调用失败");
}
```

3) 接口实现如下

支付宝交易状态查询实现方法如下：

```
/**
 * 支付宝交易状态查询
 *
 * @param aliConfigParam
 * @param outTradeNo
 * @return
 */
@Override
public PaymentResponseDTO queryPayOrderByAli(AliConfigParam aliConfigParam, String outTradeNo) {
    String gateway = aliConfigParam.getUrl();//支付接口网关地址
    String appId = aliConfigParam.getAppId();//appid
    String rsaPrivateKey = aliConfigParam.getRsaPrivateKey(); //私钥
    String format = aliConfigParam.getFormat();//json格式
    String charest = aliConfigParam.getCharest();//编码 utf-8
    String alipayPublicKey = aliConfigParam.getAlipayPublicKey(); //公钥
    String signtype = aliConfigParam.getSigntype();//签名算法类型
    log.info("C扫B请求支付宝查询订单，参数：{}", JSON.toJSONString(aliConfigParam));

    //构建sdk客户端
    AlipayClient client = new DefaultAlipayClient(gateway, appId, rsaPrivateKey, format, charest,
        alipayPublicKey, signtype);
    AlipayTradeQueryRequest queryRequest = new AlipayTradeQueryRequest();
    AlipayTradePayModel model = new AlipayTradePayModel();
    //闪聚平台订单号
    model.setOutTradeNo(outTradeNo);
    //封装请求参数
    queryRequest.setBizModel(model);

    PaymentResponseDTO dto;
    try {
        //请求支付宝接口
        AlipayTradeQueryResponse qr = client.execute(queryRequest);
        //接口调用成功
        if (AliCodeConstants.SUCCESSCODE.equals(qr.getCode())) {
            //将支付宝响应的状态转换为闪聚平台的状态
            TradeStatus tradeStatus = covertAliTradeStatusToShanjuCode(qr.getTradeStatus());
            dto = PaymentResponseDTO.success(qr.getTradeNo(), qr.getOutTradeNo(), tradeStatus,
                qr.getMsg() + " " + qr.getSubMsg());
            log.info("----查询支付宝H5支付结果" + JSON.toJSONString(dto));
            return dto;
        }
    } catch (AlipayApiException e) {
        log.warn(e.getMessage(), e);
    }
    dto = PaymentResponseDTO.fail("查询支付宝支付结果异常", outTradeNo, TradeStatus.UNKNOWN);
    return dto;
}
```

定义支付宝响应状态与闪聚平台的转换方法：

```
/**
 * 将支付宝查询时订单状态trade_status 转换为 闪聚订单状态
 * @param aliTradeStatus 支付宝交易状态
 *      WAIT_BUYER_PAY ( 交易创建，等待买家付款 )
 *      TRADE_CLOSED ( 未付款交易超时关闭，或支付完成后全额退款 )
 *      TRADE_SUCCESS ( 交易支付成功 )
 *      TRADE_FINISHED ( 交易结束，不可退款 )
 * @return
 */
private TradeStatus covertAliTradeStatusToShanjuCode(String aliTradeStatus) {
    switch (aliTradeStatus) {
        case AliCodeConstants.WAIT_BUYER_PAY:
            return TradeStatus.USERPAYING;
        case AliCodeConstants.TRADE_SUCCESS:
        case AliCodeConstants.TRADE_FINISHED:
            return TradeStatus.SUCCESS;
        default:
            return TradeStatus.FAILED;
    }
}
```

6.4.2.3 接口测试

对queryPayOrderByAli方法进行单元测试：

- 1、通过C扫B进行支付定下单，从数据库找到订单号
- 2、编写测试类

APP_ID、APP_PRIVATE_KEY、ALIPAY_PUBLIC_KEY使用自己申请的支付宝沙箱参数。

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class PayChannelAgentServiceTest {

    @Autowired
    PayChannelAgentService payChannelAgentService;

    @Test
    public void testQueryPayOrderByAli(){
        String APP_ID = "";
        String APP_PRIVATE_KEY = "";
        String ALIPAY_PUBLIC_KEY = "";
        String CHARSET = "UTF-8";
        String serverUrl = "https://openapi.alipaydev.com/gateway.do";//正式"https://openapi.alipay.com/gateway.do"

        //支付渠道参数
        AliConfigParam aliConfigParam = new AliConfigParam();
        aliConfigParam.setUrl(serverUrl);
        aliConfigParam.setCharest(CHARSET);
```

```
        aliConfigParam.setAlipayPublicKey(ALIPAY_PUBLIC_KEY);
        aliConfigParam.setRsaPrivateKey(APP_PRIVATE_KEY);
        aliConfigParam.setAppId(APP_ID);
        aliConfigParam.setFormat("json");
        aliConfigParam.setSigntype("RSA2");
        PaymentResponseDTO paymentResponseDTO =
payChannelAgentService.queryPayOrderByAli(aliConfigParam, "SJ1216325162370383873");
        System.out.println(paymentResponseDTO);
    }
}
```

6.5 支付结果查询

6.5.1 交互流程

根据技术方案的分析，交互流程如下：

- 1、支付渠道代理服务调用支付宝下单接口完成后向MQ发送“支付结果查询”消息（延迟消息），消费方为支付渠道代理服务。
- 2、支付渠道代理服务监听消息队列，接收“支付结果查询”消息。
- 3、支付渠道代理服务调用第三方支付系统的支付结果查询接口。

6.5.2 发送消息

6.5.2.1 配置RocketMQ

1) 在支付渠道代理工程中添加RocketMQ依赖：

```
<dependency>
    <groupId>org.apache.rocketmq</groupId>
    <artifactId>rocketmq-spring-boot-starter</artifactId>
    <version>2.0.2</version>
</dependency>
```

2) 在Nacos中添加spring-boot-starter-rocketmq.yaml配置，Data ID: spring-boot-starter-rocketmq.yaml, Group: COMMON_GROUP

* Data ID:

* Group:

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 ? :

```
1 rocketmq:
2   nameServer: 127.0.0.1:9876
3   producer:
4     group: PID_PAY_PRODUCER
```

```
rocketmq:
  nameServer: 127.0.0.1:9876
  producer:
    group: PID_PAY_PRODUCER
```

3) 在shanjupay-payment-agent-service工程bootstrap.yml中引入此配置：

```
-
  refresh: true
  data-id: spring-boot-starter-rocketmq.yaml # rocketmq配置
  group: COMMON_GROUP # 通用配置组
```

6.5.2.2 生产消息类

1、修改支付宝下单调用方法createPayOrderByAliWAP，调用PayProducer发送消息。

发送支付结果查询延迟消息代码如下：

```
try {
    //发送支付结果查询延迟消息
    PaymentResponseDTO<AliConfigParam> notice = new PaymentResponseDTO<AliConfigParam>();
    notice.setOutTradeNo(alipayBean.getOutTradeNo());
    notice.setContent(alipayBean.getOutTradeNo());
    notice.setMsg("ALIPAY_WAP");
    payProducer.payOrderNotice(notice);

    // 调用SDK提交表单
    AlipayTradeWapPayResponse response = client.pageExecute(alipayRequest);
    log.info("支付宝手机网站支付预支付订单信息" + response);
    PaymentResponseDTO res = new PaymentResponseDTO();
    res.setContent(response.getBody());

    return res;
} catch (Exception e) {
    e.printStackTrace();
    throw new BusinessException(CommonErrorCode.E_400002); //支付宝确认支付失败
}
```

2、在支付渠道代理服务中编写生产消息类PayProducer

```
package com.shanjupay.paymentagent.message;

import com.shanjupay.common.api.dto.PaymentResponseDTO;
import lombok.extern.slf4j.Slf4j;
import org.apache.rocketmq.spring.core.RocketMQTemplate;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

@Slf4j
```



```
@Component
public class PayProducer {
    //消息Topic
    private static final String TOPIC_ORDER = "TP_PAYMENT_ORDER";

    @Resource
    private RocketMQTemplate rocketMQTemplate;

    public void payOrderNotice(PaymentResponseDTO result) {
        log.info("支付通知发送延迟消息:{}", result);
        try {
            //处理消息存储格式
            Message<PaymentResponseDTO> message = MessageBuilder.withPayload(result).build();
            SendResult sendResult = rocketMQTemplate.syncSend(TOPIC_ORDER, message, 1000, 3);
        } catch (Exception e) {
            log.warn(e.getMessage(), e);
        }
    }
}
```

6.5.3 接收消息

定义PayConsumer类，监听“支付结果查询”消息队列。

```
package com.shanjupay.paymentagent.message;

@Slf4j
@Service
@RocketMQMessageListener(topic = "TP_PAYMENT_ORDER", consumerGroup = "CID_PAYMENT_CONSUMER")
public class PayConsumer implements RocketMQListener<MessageExt> {

    @Resource
    private PayChannelAgentService payAgentService;

    @Override
    public void onMessage(MessageExt messageExt) {
        log.info("开始消费支付结果查询消息:{}", messageExt);
        //取出消息内容
        String body = new String(messageExt.getBody(), StandardCharsets.UTF_8);
        PaymentResponseDTO response = JSON.parseObject(body, PaymentResponseDTO.class);
        String outTradeNo = response.getOutTradeNo();
        String msg = response.getMsg();
        String param = String.valueOf(response.getContent());
        AliConfigParam aliConfigParam = JSON.parseObject(param, AliConfigParam.class);
        //判断是支付宝还是微信
        PaymentResponseDTO result = new PaymentResponseDTO();
        if ("ALIPAY_WAP".equals(msg)) {
            //查询支付宝支付结果
            result = payAgentService.queryPayOrderByAli(aliConfigParam, outTradeNo);
        } else if ("WX_JSAPI".equals(msg)) {
            //查询微信支付结果
        }
    }
}
```

```
    }

    //返回查询获得的支付状态
    if (TradeStatus.UNKNOWN.equals(result.getTradeState()) || TradeStatus.USERPAYING
        .equals(result.getTradeState())) {
        //在支付状态未知或支付中，抛出异常会重新消息此消息
        log.info("支付代理---支付状态未知，等待重试");
        throw new RuntimeException("支付状态未知，等待重试");
    }

}

}
```

6.6 支付结果更新

6.6.1 交互流程

支付渠道代理服务查询到支付结果，将支付结果更新消息发送给交易服务，实现订单状态更新，流程如下：

- 1、支付渠道代理服务查询到支付结果
- 2、向MQ发送“支付结果更新”消息
- 3、交易服务监听“支付结果更新”消息队列
- 4、交易服务接收到“支付结果更新”消息，更新订单状态

6.6.2 发送消息

在支付渠道代理服务的PayProducer中定义发送“支付结果更新”消息方法

```
private static final String TOPIC_RESULT = "TP_PAYMENT_RESULT";

/**
 * 发送支付结果消息
 * @param result
 */
public void payResultNotice(PaymentResponseDTO result){
    rocketMQTemplate.convertAndSend(TOPIC_RESULT,result);
}
```

修改支付渠道代理服务的PayConsumer，在查询到支付结果后调用payResultNotice


```
....  
//不管支付成功还是失败都需要发送支付结果消息  
log.info("交易中心处理支付结果通知, 支付代理发送消息:{})", result);  
payProducer.payResultNotice(result);  
...
```

6.6.3 接收消息

6.6.3.1 交易服务更新订单接口

在TransactionServiceImpl中实现updateOrderTradeNoAndTradeState方法，根据闪聚支付订单号和支付宝订单号更新订单状态：

```
/**  
 * 更新订单支付状态  
 *  
 * @param tradeNo 闪聚平台订单号  
 * @param payChannelTradeNo 支付宝或微信的交易流水号  
 * @param state 订单状态 交易状态支付状态,0-订单生成,1-支付中(目前未使用),2-支付成功,4-关闭 5--失败  
 */  
@Override  
public void updateOrderTradeNoAndTradeState(String tradeNo, String payChannelTradeNo, String state) {  
    final LambdaUpdateWrapper<PayOrder> lambda = new UpdateWrapper<PayOrder>().lambda();  
    lambda.eq(PayOrder::getTradeNo, tradeNo).set(PayOrder::getPayChannelTradeNo, payChannelTradeNo)  
        .set(PayOrder::getTradeState, state);  
    if (state != null && "2".equals(state)) {  
        lambda.set(PayOrder::getPaySuccessTime, LocalDateTime.now());  
    }  
    payOrderMapper.update(null, lambda);  
}
```

6.6.3.2 交易服务接收消息

1) 在交易服务工程中添加RocketMQ依赖：

```
<dependency>  
    <groupId>org.apache.rocketmq</groupId>  
    <artifactId>rocketmq-spring-boot-starter</artifactId>  
    <version>2.0.2</version>  
</dependency>
```

2) 在shanjupay-transaction-service的bootstrap.yml中引入此配置：

```
-  
refresh: true  
data-id: spring-boot-starter-rocketmq.yaml # rocketmq配置  
group: COMMON_GROUP # 通用配置组
```

3) 在交易服务定义“支付结果消息”消费类。

```
@Slf4j  
@Component  
@RocketMQMessageListener(topic = "TP_PAYMENT_RESULT", consumerGroup = "CID_ORDER_CONSUMER")  
public class TransactionPayConsumer implements RocketMQListener<MessageExt> {  
    @Resource  
    private TransactionService transactionService;  
  
    @Override  
    public void onMessage(MessageExt messageExt) {  
        String body = new String(messageExt.getBody(), StandardCharsets.UTF_8);  
        PaymentResponseDTO res = JSON.parseObject(body, PaymentResponseDTO.class);  
        log.info("交易中心消费方接收支付结果消息: {}", body);  
        final TradeStatus tradeState = res.getTradeState();  
        String payChannelTradeNo = res.getTradeNo();  
        String tradeNo = res.getOutTradeNo();  
        switch (tradeState){  
            case SUCCESS:  
                //支付成功时, 修改订单状态为支付成功  
                transactionService.updateOrderTradeNoAndTradeState(tradeNo,  
payChannelTradeNo, "2");  
                return ;  
            case REVOKED:  
                //支付关闭时, 修改订单状态为关闭  
                transactionService.updateOrderTradeNoAndTradeState(tradeNo,  
payChannelTradeNo, "4");  
                return ;  
            case FAILED:  
                //支付失败时, 修改订单状态为失败  
                transactionService.updateOrderTradeNoAndTradeState(tradeNo,  
payChannelTradeNo, "5");  
                return ;  
            default:  
                throw new RuntimeException(String.format("无法解析支付结  
果:%s", body));  
        }  
    }  
}
```

7 接入微信

7.1 接入分析

闪聚支付平台是将各各常用的第三方支付渠道统一为一个支付通道，前边实现了C扫B支付宝支付的流程，下边接入微信支付，根据接入支付宝的流程分析接入微信需要实现的如下：

1、支付入口

顾客扫码进入支付入口，根据客户端类型判断是微信还是支付宝，是支付宝则直接进入收银台，如果是微信则需要首先获取openid，再进入收银台。

2、立即支付

点击立即支付调用微信的统一下单接口，若下单成功则唤起微信客户端开始支付。

3、获取支付结果

调用微信的“支付结果查询”接口获取支付结果。

7.2 支付入口

7.2.1 获取openid接口

参考：闪聚支付-第3章-微信支付接入指南.pdf

7.2.2 获取微信授权码

用户进入支付入口，判断客户端类型如果是微信则获取微信授权码。

根据获取openid的流程得知，第一步获取微信授权码，这里需要生成获取微信授权码的URL，由页面重定向即可。

1、在nacos中交易服务的主配置文件中添加如下参数：

```
weixin:
  oauth2RequestUrl: "https://open.weixin.qq.com/connect/oauth2/authorize"
  oauth2CodeReturnUrl: "http://xfc.nat300.top/transaction/wx-oauth-code-return"
  oauth2Token: "https://api.weixin.qq.com/sns/oauth2/access_token"
```

2、在交易服务PayController中添加获取微信授权码方法。

```
@Value("${weixin.oauth2RequestUrl}")
private String wxOAuth2RequestUrl;

@Value("${weixin.oauth2CodeReturnUrl}")
private String wxOAuth2CodeReturnUrl;

/**
 * 获取微信授权码
 *
 * @param order 订单对象
 * @return
 * @throws Exception
 */
```

```
@Override
public String getWXOAuth2Code(PayOrderDTO order) throws BusinessException {
    //将订单信息封装到state参数中
    String state = EncryptUtil.encodeUTF8StringBase64(JSON.toJSONString(order));
    //应用id
    String appId = order.getAppId();
    //服务类型
    String channel = order.getChannel();
    //获取微信支付渠道参数，根据应用、服务类型、支付渠道查询支付渠道参数
    PayChannelParamDTO payChannelParamDTO =
payChannelService.queryParamByAppPlatformAndPayChannel(appId,
        channel, "WX_JSAPI");
    if(payChannelParamDTO == null){
        throw new BusinessException(CommonErrorCode.E_300007);
    }
    //支付渠道参数
    String payParam = payChannelParamDTO.getParam();
    WXConfigParam wxConfigParam = JSON.parseObject(payParam, WXConfigParam.class);

    try {
        String url = String
            .format("%s?appid=%s&scope=snsapi_base&state=%s&redirect_uri=%s",
wxOAuth2RequestUrl, wxConfigParam.getAppId(),
                state, URLEncoder.encode(wxOAuth2CodeReturnUrl, "utf-8"));
        log.info("微信生成授权码url:{},url);
        return "redirect:" + url;
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
        return "forward:/pay-page-error";//生成获取授权码链接失败
    }
}
```

2、支付入口调用微信授权码获取方法

```
@RequestMapping(value = "/pay-entry/{ticket}")
public String payEntry(@PathVariable("ticket") String ticket,HttpServletRequest request) {
    ...
    BrowserType browserType = BrowserType.valueOfUserAgent(request.getHeader("user-agent"));
    switch (browserType) {
        case ALIPAY: //直接跳转收银台pay.html
            return "forward:/pay-page?" + toParamsString(order);
        case WECHAT: //获取授权码(待实现)
            return transactionService.getWXOAuth2Code(order);
        default:
    }
    ...
}
```

7.2.3 微信授权码回调接口

授权码获取成功后微信会将授权码传入授权码回调URL，在授权码回调接口中实现获取openid。

7.2.3.1 接口定义

1、在PayController中定义微信授权码回调接口

```
@ApiOperation("微信授权码回调")
@GetMapping("/wx-oauth-code-return")
public String wxOAuth2CodeReturn(@RequestParam String code, @RequestParam String state) {
    //获取openid
    //重定向到支付确认页面
}
```

2、在TransactionService中定义获取openid方法

```
/**
 * 获取微信openid
 *
 * @param code 授权id
 * @param appId 应用id, 用于获取微信支付的参数
 * @return openid
 */
public String getWXOAuthOpenId(String code, String appId);
```

7.2.3.2 接口实现

1、添加RestTemplate配置

使用RestTemplate发起http请求，在shanjupay-transaction-service工程的pom.xml中添加依赖：

```
<!-- okhttp3 -->
<dependency>
    <groupId>com.squareup.okhttp3</groupId>
    <artifactId>okhttp</artifactId>
</dependency>
```

在TransactionBootstrap中注入RestTemplate

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate(new OkHttp3ClientHttpRequestFactory());
}
```

2、添加获取openid地址配置

在nacos中向交易服务添加获取openid地址配置，如下：

```
weixin:
oauth2Token: "https://api.weixin.qq.com/sns/oauth2/access_token"
```

3、getWXOAuthOpenId接口实现

```
/**
 * 获取微信openid
 *
 * @param code 授权id
 * @param appId 应用id, 用于获取微信支付的参数
 * @param channel 服务类型, 用于获取微信支付的参数
 * @return openid
 */
@Override
public String getWXOAuthOpenId(String code, String appId) {
    //获取微信支付渠道参数, 根据应用、服务类型、支付渠道查询支付渠道参数
    PayChannelParamDTO payChannelParamDTO =
        payChannelService.queryParamByAppPlatformAndPayChannel(appId,
            "shanju_c2b", "WX_JSAPI");
    if(payChannelParamDTO == null){
        throw new BusinessException(CommonErrorCode.E_300007);
    }
    //支付渠道参数
    String payParam = payChannelParamDTO.getParam();
    WXConfigParam wxConfigParam = JSON.parseObject(payParam, WXConfigParam.class);
    //密钥
    String appSecret = wxConfigParam.getAppSecret();
    //获取openid地址
    String url = String.format("%s?appid=%s&secret=%s&code=%s&grant_type=authorization_code",
        oauth2Token, wxConfigParam.getAppId(), wxConfigParam.getAppSecret(), code);
    ResponseEntity<String> exchange = restTemplate.exchange(tokenUrl, HttpMethod.GET, null,
        String.class);
    String response = exchange.getBody();
    return JSONObject.parseObject(response).getString("openid");
}
```

4、微信授权码回调接口实现

```
@ApiOperation("微信授权码回调")
@GetMapping("/wx-oauth-code-return")
public String wxOAuth2CodeReturn(@RequestParam String code, @RequestParam String state) {
    //将之前state中保存的订单信息读取出来
    PayOrderDTO payOrderDTO = JSON
        .parseObject(EncryptUtil.decodeUTF8StringBase64(state), PayOrderDTO.class);
    //应用id
    String appId = payOrderDTO.getAppId();
    //获取openid
    String openId = transactionService.getWXOAuthOpenId(code, appId);
    try {
        //将订单信息转成query参数的形式拼接起来
    }
```

```
String orderInfo = ParseURLPairUtil.parseURLPair(payOrderDTO);  
//返回所有的query参数到支付确认页面  
return String.format("forward:/pay-page?openId=%s&%s", openId, orderInfo);  
} catch (Exception e) {  
    e.printStackTrace();  
    return "forward:/pay-page-error";  
}  
  
}
```

7.2.4 测试

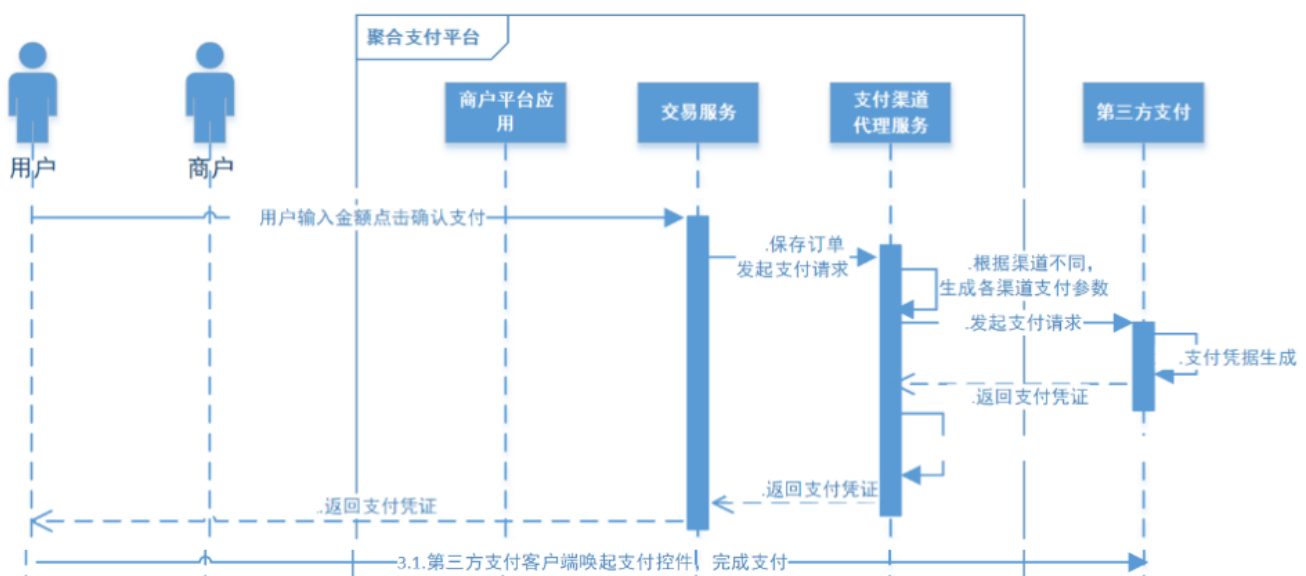
- 1、生成门店c扫b的二维码
- 2、打开内网穿透工具
- 3、打开模拟器，使用微信扫码
- 4、观察程序输出日志，确认openId是否生成成功，支付确认页面是否正常打开

7.3 立即支付

7.3.1 交互流程

点击立即支付调用第三方支付系统的下单接口，微信客户端扫码进入确认页面，点击立即支付则由渠道代理服务调用微信支付的下单接口，具体的流程如下：

- 1、微信客户端扫码进入确认页面，点击立即支付请求交易服务微信下单接口
- 2、交易服务通过支付渠道代理服务调用微信下单接口
- 3、调用微信下单接口成功，返回H5网页
- 4、在H5网页调起微信客户端支付。



7.3.1 支付渠道代理服务微信下单

7.3.1.1 接口定义

1、微信支付下单接口参数

请求参数如下，主要关注必填项目：

红色：支付渠道参数配置的内容

蓝色：微信sdk自动配置

绿色：程序设置

字段名	变量名	必填	类型	示例值	描述
公众账号ID	appid	是	String(32)	wxd678efh567hg6787	微信支付分配的公众账号ID（企业号corpid即为此appId）
商户号	mch_id	是	String(32)	1230000109	微信支付分配的商户号
设备号	device_info	否	String(32)	013467007045764	自定义参数，可以为终端设备号（门店号或收银设备ID），PC网页或公众号内支付可以传"WEB"
随机字符串	nonce_str	是	String(32)	5K8264ILTKCH16CQ2502SI8ZNMTM67VS	随机字符串，长度要求在32位以内。 推荐随机数生成算法
签名	sign	是	String(32)	C380BEC2BFD727A4B6845133519F3AD6	通过签名算法计算得出的签名值，详见 签名生成算法
签名类型	sign_type	否	String(32)	MD5	签名类型，默认为MD5，支持HMAC-SHA256和MD5。
商品描述	body	是	String(128)	腾讯充值中心-QQ会员充值	商品简单描述，该字段请按照规范传递，具体请见 参数规定
附加数据	attach	否	String(127)	深圳分店	附加数据，在查询API和支付通知中原样返回，可作为自定义参数使用。
商户订单号	out_trade_no	是	String(32)	20150806125346	商户系统内部订单号，要求32个字符内，只能是数字、大小写字母_- * 且在同一个商户号下唯一。详见 商户订单号
标价币种	fee_type	否	String(16)	CNY	符合ISO 4217标准的三位字母代码，默认人民币：CNY，详细列表请参见 货币类型
标价金额	total_fee	是	Int	88	订单总金额，单位为分，详见 支付金额
终端IP	spbill_create_ip	是	String(64)	123.12.12.123	支持IPv4和IPv6两种格式的IP地址。用户的客户端IP

通知地址	notify_url	是	String(256)	http://www.weixin.qq.com/wxpay/pay.php	异步接收微信支付结果通知的回调地址，通知url必须为外网可访问的url，不能携带参数。
交易类型	trade_type	是	String(16)	JSAPI	JSAPI -JSAPI支付 NATIVE -Native支付 APP -APP支付 说明详见 参数规定

响应参数：

字段名	变量名	必填	类型	示例值	描述
返回状态码	return_code	是	String(16)	SUCCESS	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看result_code来判断
返回信息	return_msg	是	String(128)	OK	当return_code为FAIL时返回信息为错误原因，例如 签名失败 参数格式校验错误

以下字段在return_code为SUCCESS的时候有返回

字段名	变量名	必填	类型	示例值	描述
公众账号ID	appid	是	String(32)	wx8888888888888888	调用接口提交的公众账号ID
商户号	mch_id	是	String(32)	1900000109	调用接口提交的商户号
设备号	device_info	否	String(32)	013467007045764	自定义参数，可以为请求支付的终端设备号等
随机字符串	nonce_str	是	String(32)	5K8264ILTKCH16CQ2502SI8ZNMTM67VS	微信返回的随机字符串
签名	sign	是	String(32)	C380BEC2BFD727A4B6845133519F3AD6	微信返回的签名值，详见 签名算法
业务结果	result_code	是	String(16)	SUCCESS	SUCCESS/FAIL
错误代码	err_code	否	String(32)		当result_code为FAIL时返回错误代码，详细参见下文错误列表
错误代码描述	err_code_des	否	String(128)		当result_code为FAIL时返回错误描述，详细参见下文错误列表

以下字段在return_code和result_code都为SUCCESS的时候有返回

字段名	变量名	必填	类型	示例值	描述
交易类型	trade_type	是	String(16)	JSAPI	JSAPI -JSAPI支付 NATIVE -Native支付 APP -APP支付 说明详见 参数规定
预支付交易会话标识	prepay_id	是	String(64)	wx20141027200939552 2657a690389285100	微信生成的预支付会话标识，用于后续接口调用中使用，该值有效期为2小时

2、支付渠道代理服务下单接口定义

1) 接口描述：

调用微信jsapi下单接口

2) 接口定义

在PayChannelAgentService中定义接口：

```
/**
 * 微信jsapi下单接口请求
 * @param wxConfigParam
 * @param weChatBean
 * @return h5页面所需要的数据
 */
public Map<String, String> createPayOrderByWeChatJSAPI(WXConfigParam wxConfigParam, WeChatBean weChatBean);
```

7.3.1.2 接口实现

将资料-->代码下的WXSDKConfig.java工具类拷贝至支付渠道代理服务。

1、在shanjupay-payment-agent-service工程的pom.xml引入依赖：

```
<!-- 微信支付SDK -->
<dependency>
    <groupId>com.github.tedzhdz</groupId>
    <artifactId>wxpay-sdk</artifactId>
    <version>3.0.10</version>
</dependency>

<dependency>
    <groupId>com.github.binarywang</groupId>
    <artifactId>weixin-java-pay</artifactId>
    <version>3.4.0</version>
</dependency>
```

2、在PayChannelAgentService中实现createPayOrderByWeChatJSAPI方法



```
/**
 * 微信jsapi下单接口请求
 *
 * @param wxConfigParam
 * @param weChatBean
 * @return
 */
@Override
public Map<String, String> createPayOrderByWeChatJSAPI(WXConfigParam wxConfigParam, WeChatBean weChatBean) {
    WXSDKConfig config = new WXSDKConfig(wxConfigParam); //通过实际支付参数匹配

    try {
        WXPAY wxpay = new WXPAY(config);

        //按照微信统一下单接口要求构造请求参数
        //https://pay.weixin.qq.com/wiki/doc/api/jsapi.php?chapter=9_1
        Map<String, String> requestParam = new HashMap<String, String>();
        requestParam.put("body", weChatBean.getBody());
        requestParam.put("out_trade_no", weChatBean.getOutTradeNo());
        requestParam.put("fee_type", "CNY");
        requestParam.put("total_fee", String.valueOf(weChatBean.getTotalFee()));
        requestParam.put("spbill_create_ip", weChatBean.getSpbillCreateIp());
        requestParam.put("notify_url", weChatBean.getNotifyUrl());
        requestParam.put("trade_type", "JSAPI");
        requestParam.put("openid", weChatBean.getOpenId());

        //调用微信统一下单API
        Map<String, String> resp = wxpay.unifiedOrder(requestParam);

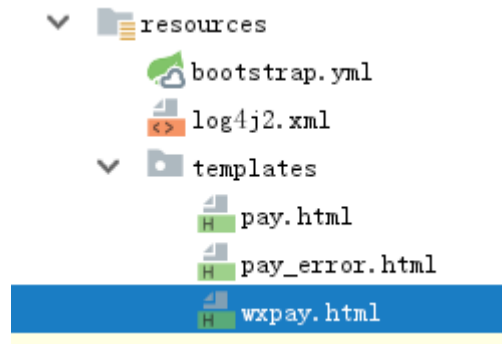
        //返回h5网页需要的数据
        String timestamp = String.valueOf(System.currentTimeMillis() / 1000);
        String key = wxConfigParam.getKey();
        Map<String, String> jsapiPayParam = new HashMap<>();
        jsapiPayParam.put("appId", resp.get("appid"));
        jsapiPayParam.put("package", "prepay_id=" + resp.get("prepay_id"));
        jsapiPayParam.put("timeStamp", timestamp);
        jsapiPayParam.put("nonceStr", UUID.randomUUID().toString());
        jsapiPayParam.put("signType", "HMAC-SHA256");
        jsapiPayParam.put("paySign",
            WXPAYUtil.generateSignature(jsapiPayParam, key, WXPAYConstants.SignType.HMACSHA256));
        log.info("微信JSAPI支付响应内容：" + jsapiPayParam);
        return jsapiPayParam;
    } catch (Exception e) {
        log.error(e.getMessage(), e);
        throw new BusinessException(CommonErrorCode.E_400001);
    }
}
```

7.3.2 交易服务微信下单

交易服务微信下单是提供给支付入口请求的微信付款的接口，当用户用微信客户端扫描二维码进入确认支付页面，点击确认支付即将请求此接口。

7.3.2.1 H5页面

按照微信官方例子编写调起微信客户端支付的H5页面，从资料文件夹拷贝“wxpay.html”到交易服务下



7.3.2.2 接口定义

1、接口描述

- 1) 接收前端支付请求
- 2) 保存订单信息到闪聚支付平台
- 3) 调用支付渠道代理服务请求微信下单接口

2、接口定义

- 1、在TransactionServiceImpl中编写submitOrderByWechat接口。

```
/**
 * 微信确认支付
 * @param payOrderDTO
 * @return 微信下单接口的响应数据
 * @throws BusinessException
 */
Map<String, String> submitOrderByWechat(PayOrderDTO payOrderDTO) throws BusinessException;
```

- 2 在PayController中定义接口如下：

```
@ApiOperation("微信门店下单付款")
@PostMapping("/wxjspay")
public ModelAndView createWXOrderForStore(OrderConfirmVO orderConfirmVO, HttpServletRequest request) {
    if (StringUtils.isBlank(orderConfirmVO.getOpenId())) {
        throw new BusinessException(CommonErrorCode.E_300002);
    }
    //应用id
    String appId = payOrderDTO.getAppId();
```



```
AppDTO app = appService.getAppById(appId);
//商户id
payOrderDTO.setMerchantId(app.getMerchantId());
//客户端ip
payOrderDTO.setClientIp(IPUtil.getIpAddr(request));
//将前端输入的元转成分
payOrderDTO.setTotalAmount(Integer.parseInt(AmountUtil.changeY2F(orderConfirmVO.getTotalAmount().
toString())));

//调用微信下单接口
Map<String, String> jsapiResponse = transactionService.submitOrderByWechat(payOrderDTO);
log.info("/wxjspay 微信门店下单接口响应内容: {}", jsapiResponse);
return new ModelAndView("wxpay", jsapiResponse);
}
```

7.3.2.3 接口实现

本接口实现两部分内容：

- 1) 保存订单到闪聚支付数据库
- 2) 调用支付渠道代理服务请求微信下单接口

1、保存订单

实现方法同支付宝下单，需要注意订单信息的支付渠道标识为WX_JSAPI：

```
/**
 * 微信确认支付
 *
 * @param payOrderDTO
 * @return 微信下单接口的响应数据
 * @throws BusinessException
 */
@Override
public Map<String, String> submitOrderByWechat(PayOrderDTO payOrderDTO) {
    //微信openid
    String openId = payOrderDTO.getOpenId();
    payOrderDTO.setPayChannel("WX_JSAPI");
    //保存订单
    PayOrderDTO save = save(payOrderDTO);
    String tradeNo = save.getTradeNo();
    //微信统一下单
    return weChatJsapi(openId, tradeNo);
}
```

2、请求支付渠道代理服务进行微信下单

```
/**
 * 微信jsapi 调用支付渠道代理
 * @param orderResult 下单请求
 * @return
```

```
*/
private Map<String, String> weChatJsapi(String openId,String tradeNo) {

    //根据订单号查询订单详情
    PayOrderDTO payOrderDTO = queryPayOrder(tradeNo);
    if(payOrderDTO == null){
        throw new BusinessException(CommonErrorCode.E_400002);
    }
    //构造微信订单参数实体
    WeChatBean weChatBean = new WeChatBean();
    weChatBean.setOpenId(openId);//openid
    weChatBean.setSpbillCreateIp(payOrderDTO.getClientIp());//客户ip
    weChatBean.setTotalFee(payOrderDTO.getTotalAmount());//金额
    weChatBean.setBody(payOrderDTO.getBody());//订单描述
    weChatBean.setOutTradeNo(payOrderDTO.getTradeNo());//使用聚合平台的订单号 tradeNo
    weChatBean.setNotifyUrl("none");//异步接收微信通知支付结果的地址(暂时不用)

    //根据应用、服务类型、支付渠道查询支付渠道参数
    PayChannelParamDTO payChannelParamDTO =
    payChannelService.queryParamByAppPlatformAndPayChannel(payOrderDTO.getAppId(),
        "shanju_c2b", "WX_JSAPI");
    if(payChannelParamDTO == null){
        throw new BusinessException(CommonErrorCode.E_300007);
    }
    //支付宝渠道参数
    WXConfigParam wxConfigParam = JSON.parseObject(payChannelParamDTO.getParam(),
    WXConfigParam.class);
    return payChannelAgentService.createPayOrderByWeChatJSAPI(wxConfigParam, weChatBean);
}
```

7.3.2.4 接口测试

- 1、生成门店c扫b的二维码
- 2、打开模拟器，使用微信扫码，进入支付确认页面
- 3、输入金额，点击立即支付
- 4、观察控制台日志，最终订单写入闪聚平台数据库
- 5、调起微信支付客户端，输入密码支付成功

7.4 获取支付结果

7.4.1 微信支付结果查询接口

根据获取支付结果的技术方案，接入微信需要请求微信查询支付结果，接口参数如下：

请求参数：

请求参数

字段名	变量名	必填	类型	示例值	描述
公众账号ID	appid	是	String(32)	wxd678efh567hg6787	微信支付分配的公众账号ID（企业号corpid即为此appId）
商户号	mch_id	是	String(32)	1230000109	微信支付分配的商户号
微信订单号	transaction_id	二选一	String(32)	1009660380201506130728806387	微信的订单号，建议优先使用
商户订单号	out_trade_no		String(32)	20150806125346	商户系统内部订单号，要求32个字符内，只能是数字、大小写字母_-!*@，且在同一个商户号下唯一。详见 商户订单号
随机字符串	nonce_str	是	String(32)	C380BEC2BFD727A4B6845133519F3AD6	随机字符串，不长于32位。推荐 随机数生成算法
签名	sign	是	String(32)	5K8264ILTKCH16CQ2502SI8ZNMTM67VS	通过签名算法计算得出的签名值，详见 签名生成算法
签名类型	sign_type	否	String(32)	HMAC-SHA256	签名类型，目前支持HMAC-SHA256和MD5，默认为MD5

响应参数：

返回结果

字段名	变量名	必填	类型	示例值	描述
返回状态码	return_code	是	String(16)	SUCCESS	SUCCESS/FAIL 此字段是通信标识，非交易标识，交易是否成功需要查看trade_state来判断
返回信息	return_msg	是	String(128)	OK	当return_code为FAIL时返回信息为错误原因，例如 签名失败 参数格式校验错误

以下字段在return_code为SUCCESS的时候有返回

字段名	变量名	必填	类型	示例值	描述
公众账号ID	appid	是	String(32)	wxd678efh567hg6787	微信分配的公众账号ID
商户号	mch_id	是	String(32)	1230000109	微信支付分配的商户号
随机字符串	nonce_str	是	String(32)	5K8264ILTKCH16CQ2502SI8ZNMTM67VS	随机字符串，不长于32位。推荐 随机数生成算法
签名	sign	是	String(32)	C380BEC2BFD727A4B6845133519F3AD6	签名，详见 签名生成算法
业务结果	result_code	是	String(16)	SUCCESS	SUCCESS/FAIL
错误代码	err_code	否	String(32)		当result_code为FAIL时返回错误代码，详细参见下文错误列表
错误代码描述	err_code_des	否	String(128)		当result_code为FAIL时返回错误描述，详细参见下文错误列表

以下字段在return_code、result_code、trade_state都为SUCCESS时有返回，如trade_state不为SUCCESS，则只返回out_trade_no（必传）和attach（选传）。

字段名	变量名	必填	类型	示例值	描述
设备号	device_info	否	String(32)	013467007045764	微信支付分配的终端设备号
用户标识	openid	是	String(128)	oUpF8uMuAJO_M2pxb1Q9zNjWeS6o	用户在商户appid下的唯一标识
是否关注公众账号	is_subscribe	是	String(1)	Y	用户是否关注公众账号，Y-关注，N-未关注
交易类型	trade_type	是	String(16)	JSAPI	调用接口提交的交易类型，取值如下：JSAPI，NATIVE，APP，MICRO PAY，详细说明见 参数规定
交易状态	trade_state	是	String(32)	SUCCESS	SUCCESS—支付成功 REFUND—转入退款 NOTPAY—未支付 CLOSED—已关闭 REVOKED—已撤销（付款码支付） USERPAYING--用户支付中（付款码支付） PAYERROR--支付失败(其他原因，如银行返回失败) 支付状态机请见下单API页面
付款银行	bank_type	是	String(16)	CMC	银行类型，采用字符串类型的银行标识
标价金额	total_fee	是	Int	100	订单总金额，单位为分
现金支付金额	cash_fee	是	Int	100	现金支付金额订单现金支付金额，详见 支付金额
微信支付订单号	transaction_id	是	String(32)	1009660380201506130728806387	微信支付订单号
商户订单号	out_trade_no	是	String(32)	20150806125346	商户系统内部订单号，要求32个字符内，只能是数字、大小写字母_- *@，且在同一个商户号下唯一。
附加数据	attach	否	String(128)	深圳分店	附加数据，原样返回
支付完成时间	time_end	是	String(14)	20141030133525	订单支付时间，格式为yyyyMMddHHmmss，如2009年12月25日9点10分10秒表示为20091225091010。其他详见 时间规则
交易状态描述	trade_state_desc	是	String(256)	支付失败，请重新下单支付	对当前查询订单状态的描述和下一步操作的指引

支付结果：

SUCCESS—支付成功

REFUND—转入退款

NOTPAY—未支付

CLOSED—已关闭

REVOKED—已撤销（付款码支付）

USERPAYING--用户支付中（付款码支付）

PAYERROR--支付失败(其他原因，如银行返回失败)

7.4.2 支付渠道代理接口定义

7.4.2.1 接口定义

接口描述：

1) 使用微信SDK发起支付结果查询请求 2) 返回查询结果

在PayChannelAgentService定义如下接口：

```
/**
 * 查询微信支付结果
 *
 * @param wxConfigParam
 * @param outTradeNo
 * @return
 */
public PaymentResponseDTO queryPayOrderByWeChat(WXConfigParam wxConfigParam, String
outTradeNo);
```

7.4.2.2 接口实现

```
/**
 * 查询微信支付结果
 *
 * @param wxConfigParam
 * @param outTradeNo
 * @return
 */
@Override
public PaymentResponseDTO queryPayOrderByWeChat(WXConfigParam wxConfigParam, String outTradeNo) {
    WXSDKConfig config = new WXSDKConfig(wxConfigParam); //通过实际支付参数匹配
    Map<String, String> resp = null;
    try {
        WXPAY wxpay = new WXPAY(config);
        Map<String, String> data = new HashMap<String, String>();
        data.put("out_trade_no", outTradeNo);
        resp = wxpay.orderQuery(data);
    } catch (Exception e) {
        log.warn(e.getMessage(), e);
        return PaymentResponseDTO.fail("调用微信查询订单异常", outTradeNo, TradeStatus.UNKNOWN);
    }
```

```
}
String returnCode = resp.get("return_code");
String resultCode = resp.get("result_code");
String tradeState = resp.get("trade_state");
String transactionId = resp.get("transaction_id");
String tradeType = resp.get("trade_type");
String returnMsg = resp.get("return_msg");

    if ("SUCCESS".equals(returnCode) && "SUCCESS".equals(resultCode)) { // 接口调用成功
        if ("SUCCESS".equals(tradeState)) { //交易成功
            return PaymentResponseDTO.success(transactionId, outTradeNo, TradeStatus.SUCCESS,
""");
        } else if ("USERPAYING".equals(tradeState)) { //等待用户支付
            return PaymentResponseDTO.success(transactionId, outTradeNo,
TradeStatus.USERPAYING, "");
        } else if ("PAYERROR".equals(tradeState)) { //交易失败
            return PaymentResponseDTO.success(transactionId, outTradeNo, TradeStatus.FAILED,
returnMsg);
        } else if ("CLOSED".equals(tradeState)) { //交易关闭
            return PaymentResponseDTO.success(transactionId, outTradeNo, TradeStatus.REVOKED,
returnMsg);
        }
    }

    return PaymentResponseDTO.success("暂不支持其他状态", transactionId, outTradeNo,
TradeStatus.UNKNOWN);
}
```

7.4.2.3 单元测试

```
@Test
public void testQueryPayOrderByWeChat(){
    String appID = "wxd2bf2dba2e86a8c7";
    String mchID = "1502570431";
    String appSecret = "cec1a9185ad435abe1bcd4b93f7ef2e";
    String key = "95fe355daca50f1ae82f0865c2ce87c8";
    //支付渠道参数
    WXConfigParam wxConfigParam = new WXConfigParam();
    wxConfigParam.setAppId(appID);
    wxConfigParam.setMchId(mchID);
    wxConfigParam.setAppSecret(appSecret);
    wxConfigParam.setKey(key);
    PaymentResponseDTO paymentResponseDTO =
payChannelAgentService.queryPayOrderByWeChat(wxConfigParam, "SJ1216580930633506817");
    System.out.println(paymentResponseDTO);
}
```

7.4.3 支付查询

7.4.3.1 发送支付结果查询消息

支付渠道代理服务完成微信下单接口的调用即向MQ发送支付结果查询消息

修改支付渠道代理服务的createPayOrderByWeChatJSAPI方法：

```
@Override
public Map<String, String> createPayOrderByWeChatJSAPI(WXConfigParam wxConfigParam, WeChatBean
weChatBean) {
    WXSDKConfig config = new WXSDKConfig(wxConfigParam); //通过实际支付参数匹配

    try {
        //发送支付结果查询延迟消息
        PaymentResponseDTO<WXConfigParam> notice = new PaymentResponseDTO<WXConfigParam>();
        notice.setOutTradeNo(weChatBean.getOutTradeNo());
        notice.setContent(wxConfigParam);
        notice.setMsg("ALIPAY_WAP");
        payProducer.payOrderNotice(notice);
        ...
    }
}
```

7.4.3.2 消费支付结果查询消息

修改支付渠道代理服务的PayConsumer

```
...
@Override
public void onMessage(MessageExt messageExt) {
    log.info("开始消费支付结果查询消息:{}", messageExt);
    //取出消息内容
    String body = new String(messageExt.getBody(), StandardCharsets.UTF_8);
    PaymentResponseDTO response = JSON.parseObject(body, PaymentResponseDTO.class);
    String outTradeNo = response.getOutTradeNo();
    String msg = response.getMsg();
    String param = String.valueOf(response.getContent());

    //判断是支付宝还是微信
    PaymentResponseDTO result = new PaymentResponseDTO();
    if ("ALIPAY_WAP".equals(msg)) {
        //查询支付宝支付结果
        AliConfigParam aliConfigParam = JSON.parseObject(param, AliConfigParam.class);
        result = payAgentService.queryPayOrderByAli(aliConfigParam, outTradeNo);
    } else if ("WX_JSAPI".equals(msg)) {
        //查询微信支付结果
        WXConfigParam wxConfigParam = JSON.parseObject(param, WXConfigParam.class);
        result = payAgentService.queryPayOrderByWeChat(wxConfigParam, outTradeNo);
    }
    ...
}
```

7.4.3.3 测试

- 1、生成门店c扫b的二维码
- 2、打开模拟器，使用微信扫码，进入支付确认页面

- 3、输入金额，点击立即支付
- 4、观察控制台日志，最终订单写入闪聚平台数据库
- 5、调起微信支付客户端，输入密码支付成功

观察控制台日志，支付结果是否发送至交易服务。

数据库订单状态是否正常更新。