**畅购电商系统开发**

**第12天**

传智播客.黑马程序员.深圳

# 学习目标

* 能够说出微信支付开发的整体思路
* 生成支付二维码
* 查询支付状态
* 实现支付日志的生成与订单状态的修改、删除订单
* 支付状态回查
* MQ处理支付回调状态
* 定时处理订单状态

# 开发准备

## 开发文档

微信支付接口调用的整体思路：

按API要求组装参数，以XML方式发送（POST）给微信支付接口（URL）,微信支付接口也是以XML方式给予响应。程序根据返回的结果（其中包括支付URL）生成二维码或判断订单状态。

在线微信支付开发文档：

[https://pay.weixin.qq.com/wiki/doc/api/in dex.html](https://pay.weixin.qq.com/wiki/doc/api/in%20dex.html)

我们在本章课程中会用到”统一下单”和”查询订单”两组API

properties

1. appid：微信公众账号或开放平台APP的唯一标识

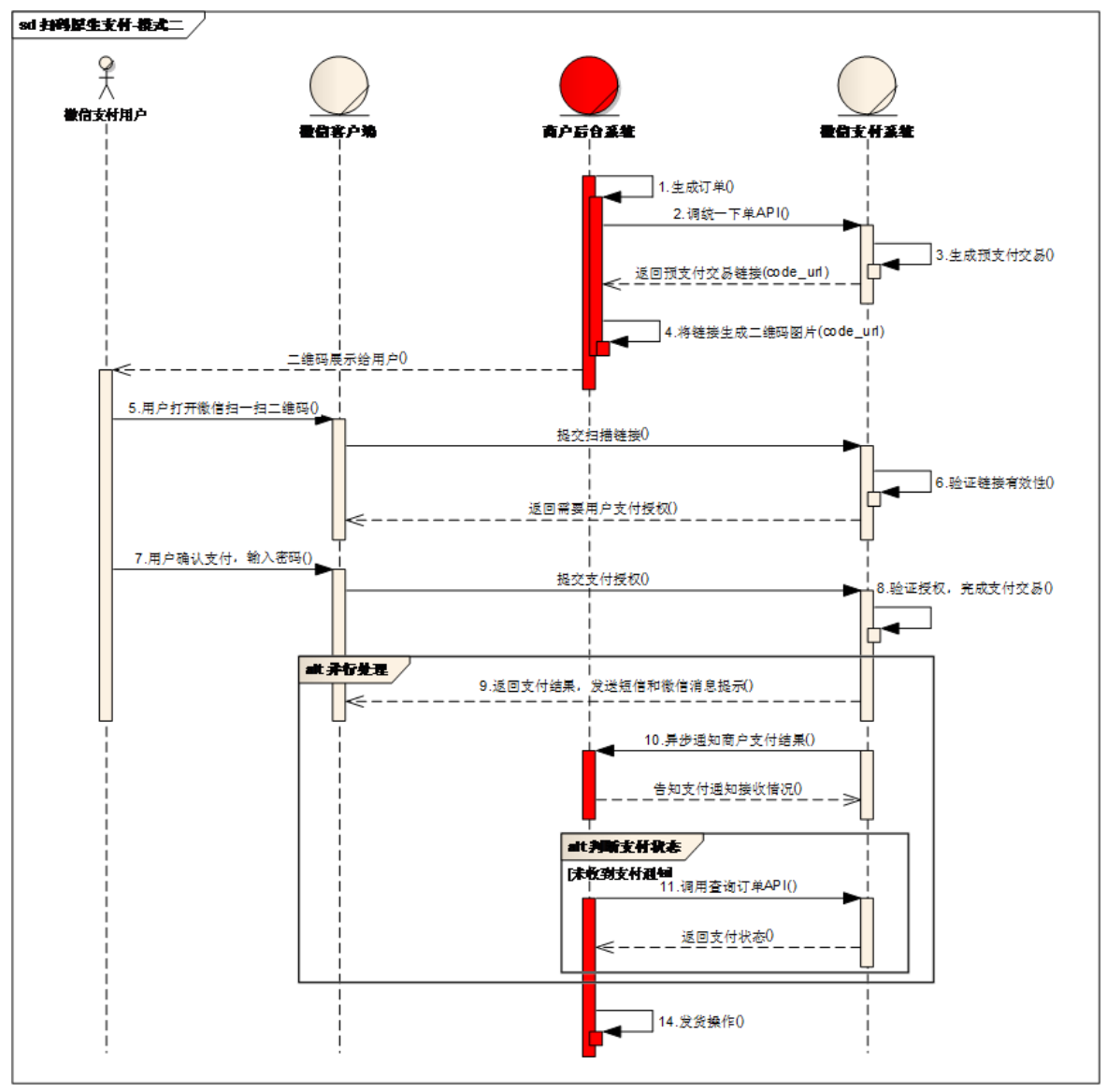
2. mch\_id：商户号 (配置文件中的partner)

3. partnerkey：商户密钥

4. sign:数字签名, 根据微信官方提供的密钥和一套算法生成的一个加密信息, 就是为了保证交易的安全性

## 微信支付模式回顾

模式二



## 微信支付SDK

微信支付提供了SDK, 大家下载后打开源码，install到本地仓库。



课程配套的本地仓库已经提供jar包，所以安装SDK步骤省略。

使用微信支付SDK,在maven工程中引入依赖

<!--微信支付-->

<**dependency**>

<**groupId**>com.github.wxpay</**groupId**>

<**artifactId**>wxpay-sdk</**artifactId**>

<**version**>0.0.3</**version**>

</**dependency**>

我们主要会用到微信支付SDK的以下功能：

获取随机字符串

WXPayUtil.generateNonceStr()

MAP转换为XML字符串（自动添加签名）

WXPayUtil.generateSignedXml(param, partnerkey)

XML字符串转换为MAP

WXPayUtil.xmlToMap(result)

为了方便微信支付开发，我们可以在changgou-common工程下引入依赖（已完成）

## HttpClient工具类

HttpClient是Apache Jakarta Common下的子项目，用来提供高效的、最新的、功能丰富的支持HTTP协议的客户端编程工具包，并且它支持HTTP协议最新的版本和建议。HttpClient已经应用在很多的项目中，比如Apache Jakarta上很著名的另外两个开源项目Cactus和HTMLUnit都使用了HttpClient。

HttpClient通俗的讲就是模拟了浏览器的行为，如果我们需要在后端向某一地址提交数据获取结果，就可以使用HttpClient.

关于HttpClient（原生）具体的使用不属于我们本章的学习内容，我们这里这里为了简化HttpClient的使用，提供了工具类HttpClient（对原生HttpClient进行了封装）

HttpClient工具类在changgou\_comon的entity包中

HttpClient工具类使用的步骤

HttpClient client=new HttpClient(请求的url地址);

client.setHttps(true);//是否是https协议

client.setXmlParam(xmlParam);//发送的xml数据

client.post();//执行post请求

String result = client.getContent(); //获取结果

将HttpClient工具包放到common工程下并引入依赖，引入依赖后就可以直接使用上述的工具包了。

<!--httpclient支持-->

<**dependency**>

<**groupId**>org.apache.httpcomponents</**groupId**>

<**artifactId**>httpclient</**artifactId**>

</**dependency**>

## 支付微服务搭建

### 创建changgou\_service\_pay

创建支付微服务changgou\_service\_pay，只要实现支付相关操作。

### application.yml

**server**:  
 **port**: 18090  
**spring**:  
 **application**:  
 **name**: pay  
 **main**:  
 **allow-bean-definition-overriding**: **true  
eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true***#hystrix 配置***hystrix**:  
 **command**:  
 **default**:  
 **execution**:  
 **timeout**:  
 *#如果enabled设置为false，则请求超时交给ribbon控制* **enabled**: true  
 **isolation**:  
 **strategy**: SEMAPHORE  
  
*#微信支付信息配置***weixin**:  
 **appid**: wx8397f8696b538317 *#微信公众账号或开放平台APP的唯一标识* **partner**: 1473426802 *#财付通平台的商户账号* **partnerkey**: T6m9iK73b0kn9g5v426MKfHQH7X8rKwb *#财付通平台的商户密钥* **notifyurl**: http://www.itcast.cn *#回调地址*

### 启动类创建

在changgou-service-pay中创建com.changgou.WeixinPayApplication，代码如下：

@SpringBootApplication(exclude={DataSourceAutoConfiguration.**class**})  
@EnableEurekaClient  
**public class** WeixinPayApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(WeixinPayApplication.**class**,args);  
 }  
}

# 微信支付二维码生成

## 需求分析与实现思路

在支付页面上生成支付二维码，并显示订单号和金额

用户拿出手机,打开微信扫描页面上的二维码,然后在微信中完成支付



## 实现思路

我们通过HttpClient工具类实现对远程支付接口的调用。

接口链接：<https://api.mch.weixin.qq.com/pay/unifiedorder>

具体参数参见“统一下单”API, 构建参数发送给统一下单的url ，返回的信息中有支付url，根据url生成二维码，显示的订单号和金额也在返回的信息中。

## 业务层

新增com.changgou.pay.service.WeixinPayService接口，代码如下：

*/\*\*  
 \* 微信支付接口  
 \** ***@author*** *Hc  
 \** ***@description*** *com.changgou.pay.service  
 \*/***public interface** WeixinPayService {  
 */\*\*  
 \* 生成微信支付二维码  
 \** ***@param out\_trade\_no*** *订单号  
 \** ***@param total\_fee*** *金额(分)  
 \** ***@return*** *\*/* **public** Map createNative(String out\_trade\_no, String total\_fee);  
}

创建com.changgou.pay.service.impl.WeixinPayServiceImpl类,并发送Post请求获取预支付信息，包含二维码扫码支付地址。代码如下：

*/\*\*  
 \* 微信支付实现  
 \** ***@author*** *Hc  
 \** ***@description*** *com.changgou.pay.service.impl  
 \*/*@Service  
**public class** WeixinPayServiceImpl **implements** WeixinPayService {  
 @Value(**"${weixin.appid}"**)  
 **private** String **appid**;  
 @Value(**"${weixin.partner}"**)  
 **private** String **partner**;  
 @Value(**"${weixin.notifyurl}"**)  
 **private** String **notifyurl**;  
 @Value(**"${weixin.partnerkey}"**)  
 **private** String **partnerkey**;  
 @Override  
 **public** Map createNative(String out\_trade\_no, String total\_fee) {  
 Map map = **new** HashMap();  
 **try** {  
 *//1、包装微信接口需要的参数* Map param = **new** HashMap();  
 param.put(**"appid"**, **appid**); *//公众号ID* param.put(**"mch\_id"**, **partner**); *//商户号* param.put(**"nonce\_str"**, WXPayUtil.*generateNonceStr*()); *//随机字符串* param.put(**"body"**, **"畅购"**); *//商品描述，扫码后用户看到的商品信息* param.put(**"out\_trade\_no"**, out\_trade\_no); *//订单号* param.put(**"total\_fee"**, total\_fee); *//订单总金额，单位为分* param.put(**"spbill\_create\_ip"**, **"127.0.0.1"**); *//终端IP，只要附合ip地址规范，可以随意写* param.put(**"notify\_url"**, **notifyurl**); *//回调地址* param.put(**"trade\_type"**, **"NATIVE"**); *//交易类型，NATIVE 扫码支付  
 //2、生成xml，通过httpClient发送请求得到数据* String xmlParam = WXPayUtil.*generateSignedXml*(param, **partnerkey**);  
 System.***out***.println(**"正在调起统一下单接口，请求参数:"** + xmlParam);  
 HttpClient httpClient = **new** HttpClient(**"https://api.mch.weixin.qq.com/pay/unifiedorder"**);  
 httpClient.setHttps(**true**);  
 httpClient.setXmlParam(xmlParam);  
 httpClient.post();  
 *//3、解析结果* String xmlResult = httpClient.getContent();  
 System.***out***.println(**"调起统一下单接口成功，返回结果："** + xmlResult);  
 Map<String, String> resultMap = WXPayUtil.*xmlToMap*(xmlResult);  
 map.put(**"code\_url"**, resultMap.get(**"code\_url"**));*//支付地址* map.put(**"total\_fee"**, total\_fee);*//总金额* map.put(**"out\_trade\_no"**,out\_trade\_no);*//订单号* } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 **return** map;  
 }  
}

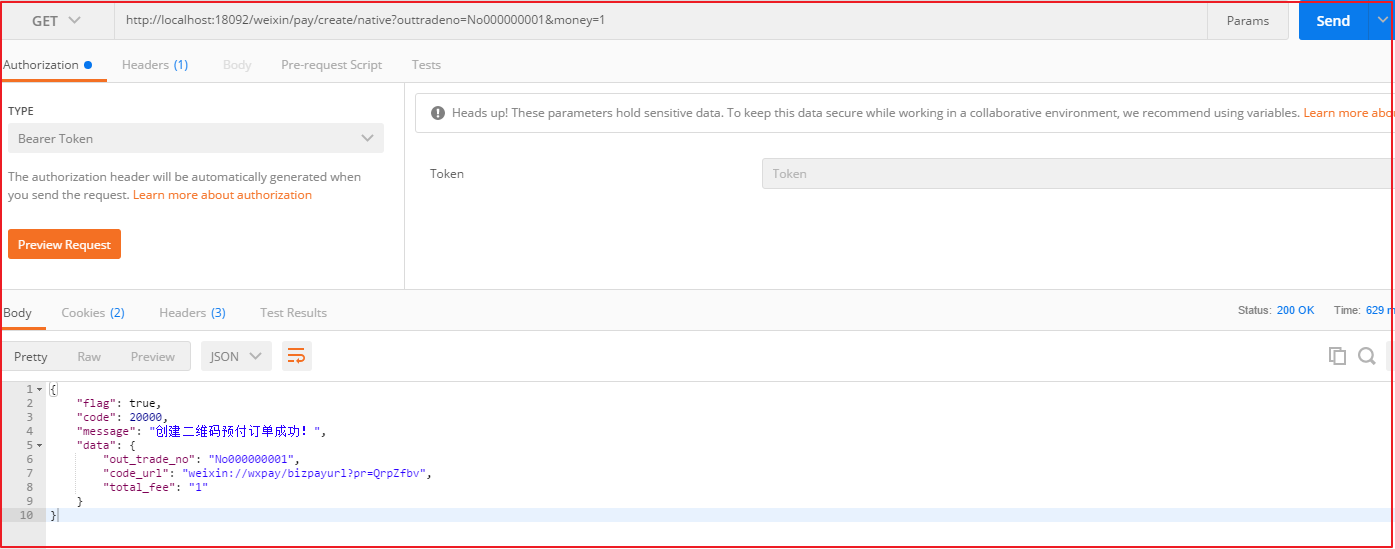
## 控制层

创建com.changgou.pay.controller.WeixinPayController,主要调用WeixinPayService的方法获取创建二维码的信息，代码如下：

@RestController  
@RequestMapping(value = **"/weixin/pay"**)  
@CrossOrigin  
**public class** WeixinPayController {  
 @Autowired  
 **private** WeixinPayService **weixinPayService**;  
 */\*\*\*  
 \* 创建二维码**\*/* @RequestMapping(value = **"/create/native"**)  
 **public** Result createNative(String out\_trade\_no, String total\_fee){  
 Map<String,String> resultMap = **weixinPayService**.createNative(out\_trade\_no,total\_fee);  
 **return new** Result(**true**, StatusCode.***OK***,**"创建二维码预付订单成功！"**,resultMap);  
 }  
}

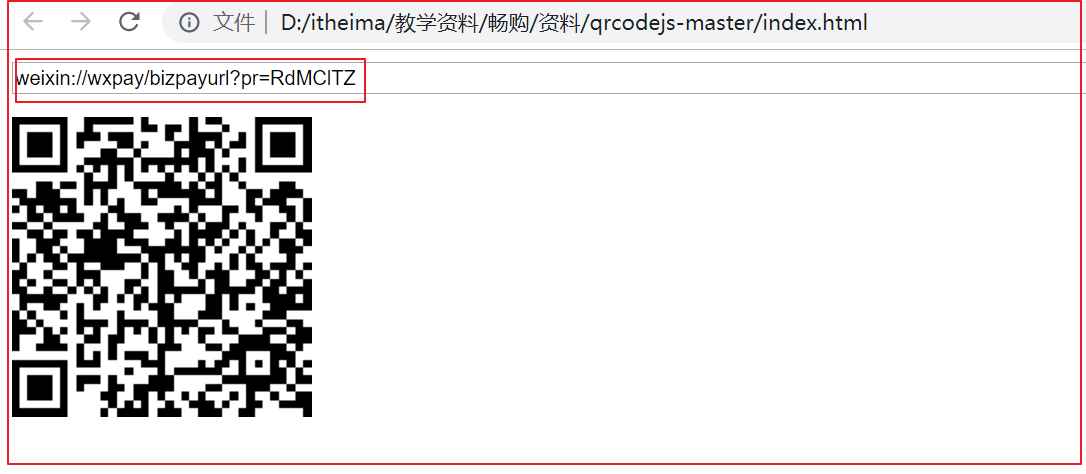
这里我们订单号通过随机数生成，金额暂时写死，后续开发我们再对接业务系统得到订单号和金额

Postman测试<http://localhost:18090/weixin/pay/create/native?out_trade_no=No0000000007&total_fee=1> （别人支付过就返回不了二维码）



## 页面生成二维码

打开我们昨天提供给大家的前端二维码生成插件，大家用微信扫码看看是不是可以支付了。



测试结果



# 支付状态查询

## 需求分析

当用户支付成功后跳转到成功页面



当返回异常时跳转到错误页面



## 实现思路

我们通过HttpClient工具类实现对远程支付接口的调用。

接口链接：<https://api.mch.weixin.qq.com/pay/orderquery>

具体参数参见“查询订单”API, 我们在controller方法中轮询调用查询订单（间隔3秒），当返回状态为success时，我们会在controller方法返回结果。前端代码收到结果后跳转到成功页面。

## 业务层

修改com.changgou.pay.service.WeixinPayService，新增方法定义

*/\*\*  
 \* 查询支付状态  
 \** ***@param out\_trade\_no*** *商户订单号  
 \*/***public** Map queryPayStatus(String out\_trade\_no);

在com.changgou.pay.service.impl.WeixinPayServiceImpl中增加实现方法

@Override  
**public** Map queryPayStatus(String out\_trade\_no) {  
 **try** {  
 *//1、包装微信接口需要的参数* Map param = **new** HashMap();  
 param.put(**"appid"**, **appid**); *//公众号ID* param.put(**"mch\_id"**, **partner**); *//商户号* param.put(**"nonce\_str"**, WXPayUtil.*generateNonceStr*()); *//随机字符串* param.put(**"out\_trade\_no"**, out\_trade\_no); *//订单号  
 //2、生成xml，通过httpClient发送请求得到数据* String xmlParam = WXPayUtil.*generateSignedXml*(param, **partnerkey**);  
 System.***out***.println(**"正在调起查询订单接口，请求参数:"** + xmlParam);  
 HttpClient httpClient = **new** HttpClient(**"https://api.mch.weixin.qq.com/pay/orderquery"**);  
 httpClient.setHttps(**true**);  
 httpClient.setXmlParam(xmlParam);  
 httpClient.post();  
 *//3、解析结果* String xmlResult = httpClient.getContent();  
 System.***out***.println(**"调起查询订单接口成功，返回结果："** + xmlResult);  
 Map<String, String> resultMap = WXPayUtil.*xmlToMap*(xmlResult);  
 **return** resultMap;  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return null**;  
 }  
}

## 控制层

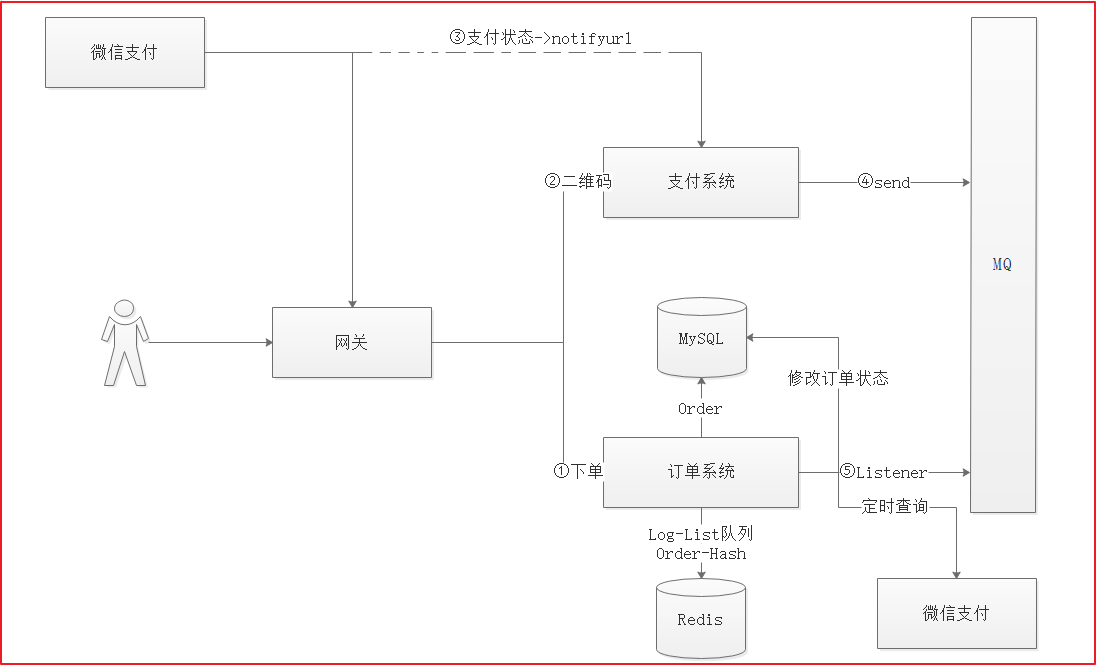
在com.changgou.pay.controller.WeixinPayController新增方法，用于查询支付状态，代码如下：

*/\*\*\*  
 \* 查询支付状态**\*/*@GetMapping(value = **"/status/query"**)  
**public** Result queryStatus(String out\_trade\_no){  
 Map<String,String> resultMap = **weixinPayService**.queryPayStatus(out\_trade\_no);  
 **return new** Result(**true**,StatusCode.***OK***,**"查询状态成功！"**,resultMap);  
}

暂时直接访问测试: [http://localhost:18090/weixin/pay/status/query?out\_trade\_no=NoHc000000007](http://localhost:18090/weixin/pay/status/query?out_trade_no=NoSteven000000007)



# 支付信息回调



我们现在系统还有个问题需要解决：支付后订单状态没有改变

流程回顾：

1.用户下单之后，订单数据会存入到MySQL中，同时会将订单对应的支付日志存入到Redis，以List+Hash的方式存储。

2.用户下单后，进入支付页面，支付页面调用支付系统，从微信支付获取二维码数据，并在页面生成支付二维码。

3.用户扫码支付后，微信支付服务器会通调用前预留的回调地址，并携带支付状态信息。

4.支付系统接到支付状态信息后，将支付状态信息发送给RabbitMQ

5.订单系统监听RabbitMQ中的消息获取支付状态，并根据支付状态修改订单状态

6.为了防止网络问题导致notifyurl没有接到对应数据，定时任务定时获取Redis中队列数据去微信支付接口查询状态，并定时更新对应状态。

需要做的工作：

1.创建订单时，同时将订单信息放到Redis中，以List和Hash各存一份

2.实现回调地址接收支付状态信息

3.将订单支付状态信息发送给RabbitMQ

4.订单系统中监听支付状态信息，如果是支付成功，修改订单状态，如果是支付失败，删除订单(或者改成支付失败)

5.防止网络异常无法接收到回调地址的支付信息，定时任务从Redis List中读取数据判断是否支付，如果支付了，修改订单状态，如果未支付，将支付信息放入队列，下次再检测，如果支付失败删除订单(或者改成支付失败)。

## 扩展-内网穿透解决方案



## 接口分析

每次实现支付之后，微信支付都会将用户支付结果返回到指定路径，而指定路径是指创建二维码的时候填写的notifyurl参数,响应的数据以及相关文档参考一下地址：<https://pay.weixin.qq.com/wiki/doc/api/native.php?chapter=9_7&index=8>

### 返回参数分析

通知参数如下：

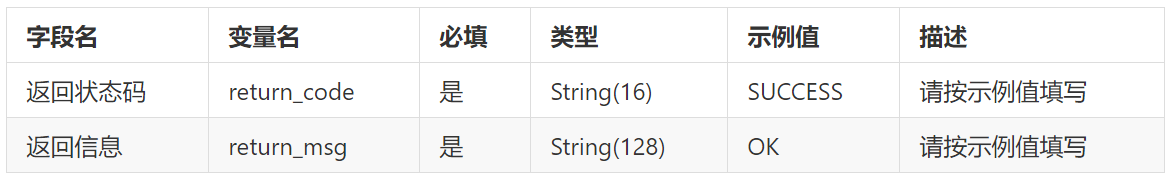


以下字段在return\_code为SUCCESS的时候有返回



### 响应分析

回调地址接收到数据后，需要响应信息给微信服务器，告知已经收到数据，不然微信服务器会再次发送4次请求推送支付信息。



## 回调接收数据实现

修改changgou-service-pay微服务的com.changgou.pay.controller.WeixinPayController,添加回调方法，代码如下：

*/\*\*\*  
 \* 支付回调  
 \* 支付完成后，微信会把相关支付结果及用户信息通过数据流的形式发送给商户，  
 \* 商户需要接收处理，并按文档规范返回应答  
 \** ***@param request*** *\** ***@return*** *\*/*@RequestMapping(value = **"/notify/url"**)  
**public** String notifyUrl(HttpServletRequest request){  
 **try** {  
 *//1.读取支付回调数据* InputStream inStream = request.getInputStream();  
 *//2.使用Apache IOUtils把输入转换成字符* String result = IOUtils.*toString*(inStream,**"UTF-8"**);  
 *//将xml字符串转换成Map结构* Map<String, String> map = WXPayUtil.*xmlToMap*(result);  
 System.***out***.println(map);  
 *//3.包装响应数据设置* Map respMap = **new** HashMap();  
 respMap.put(**"return\_code"**,**"SUCCESS"**);  
 respMap.put(**"return\_msg"**,**"OK"**);  
 **return** WXPayUtil.*mapToXml*(respMap);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 *//记录错误日志* }  
 **return null**;  
}

## 测试

测试前要记得先把回调地址改了，再重启支付服务

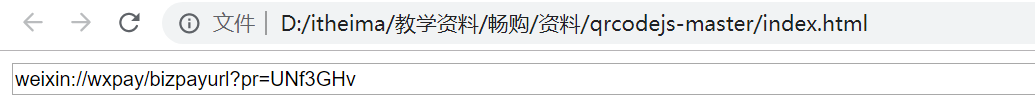
*#微信支付信息配置***weixin**:  
 **appid**: wx8397f8696b538317 *#微信公众账号或开放平台APP的唯一标识* **partner**: 1473426802 *#财付通平台的商户账号* **partnerkey**: T6m9iK73b0kn9g5v426MKfHQH7X8rKwb *#财付通平台的商户密钥* **notifyurl**: http://2g67818f20.wicp.vip:59406/weixin/pay/notify/url *#回调地址*

测试步骤:

1. 重新生成二维码

<http://localhost:18090/weixin/pay/create/native?out_trade_no=Nohc000000009&total_fee=1>

1. 把二维码放入插件得到二维码并扫码支付

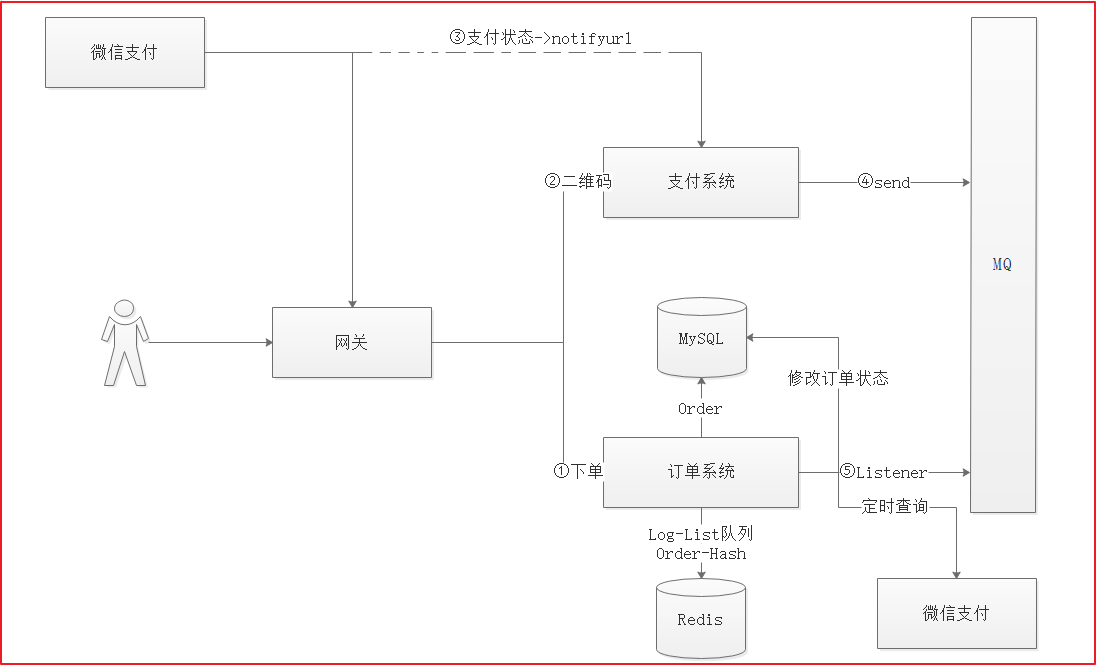


1. 查看结果

[http://localhost:18090/weixin/pay/status/query?out\_trade\_no=](http://localhost:18090/weixin/pay/status/query?out_trade_no=NoSteven000000009) Nohc000000009

# MQ处理支付回调状态

支付系统是独立于其他系统的服务，不做相关业务逻辑操作，只做支付处理，所以回调地址接收微信服务返回的支付状态后，立即将消息发送给RabbitMQ，订单系统再监听支付状态数据，根据状态数据做出修改订单状态或者删除订单操作。



## 发送支付状态

### 集成RabbitMQ

修改支付微服务，集成RabbitMQ，添加如下依赖：

<**dependencies**>  
 *<!--加入ampq-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-amqp</**artifactId**>  
 </**dependency**>  
</**dependencies**>

### 配置application.yml

这里我们建议在后台手动创建队列，并绑定队列。如果使用程序创建队列，可以按照如下方式实现。

<http://192.168.211.132:15672/#/>

默认账号：guest/guest

修改application.yml，配置支付队列和交换机信息，代码如下：

**server**:  
 **port**: 18090  
**spring**:  
 **application**:  
 **name**: pay  
 **main**:  
 **allow-bean-definition-overriding**: **true  
 rabbitmq**:  
 **host**: 192.168.211.132  
 **port**: 5672  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true***#hystrix 配置***hystrix**:  
 **command**:  
 **default**:  
 **execution**:  
 **timeout**:  
 *#如果enabled设置为false，则请求超时交给ribbon控制* **enabled**: true  
 **isolation**:  
 **strategy**: SEMAPHORE  
*#微信支付信息配置***weixin**:  
 **appid**: wx8397f8696b538317 *#微信公众账号或开放平台APP的唯一标识* **partner**: 1473426802 *#财付通平台的商户账号* **partnerkey**: T6m9iK73b0kn9g5v426MKfHQH7X8rKwb *#财付通平台的商户密钥* **notifyurl**: http://2g67818f20.wicp.vip:59406/weixin/pay/notify/url *#回调地址  
#位置支付交换机和队列***mq**:  
 **pay**:  
 **exchange**:  
 **order**: exchange.order  
 **queue**:  
 **order**: queue.order  
 **routing**:  
 **key**: queue.order

### 创建交换机与队列

创建队列以及交换机并让队列和交换机绑定，修改com.changgou.WeixinPayApplication,添加如下代码：

@Autowired  
**private** Environment **env**;  
  
*//创建交换机*@Bean  
**public** DirectExchange orderExchange(){  
 **return new** DirectExchange(**env**.getProperty(**"mq.pay.exchange.order"**), **true**, **false**);  
}  
  
*//创建队列*@Bean  
**public** Queue orderQueue(){  
 **return new** Queue(**env**.getProperty(**"mq.pay.queue.order"**));  
}  
  
*/\*\*  
 \* 队列绑定交换机  
 \** ***@Qualifier*** *可以指定spring bean的id,默认情况下id就是方法名  
 \** ***@return*** *\*/*@Bean  
**public** Binding bindingOrder(@Qualifier(**"orderQueue"**) Queue orderQueue, DirectExchange orderExchange){  
 **return** BindingBuilder.*bind*(orderQueue).to(orderExchange).with(**env**.getProperty(**"mq.pay.routing.key"**));  
}

### 发送MQ消息

修改回调方法，在接到支付信息后，立即将支付信息发送给RabbitMQ，代码如下：

@Value(**"${mq.pay.exchange.order}"**)  
**private** String **exchange**;  
@Value(**"${mq.pay.queue.order}"**)  
**private** String **queue**;  
@Value(**"${mq.pay.routing.key}"**)  
**private** String **routing**;  
@Autowired  
**private** RabbitTemplate **rabbitTemplate**;  
*/\*\*\*  
 \* 支付回调  
 \* 支付完成后，微信会把相关支付结果及用户信息通过数据流的形式发送给商户，  
 \* 商户需要接收处理，并按文档规范返回应答**\** ***@return*** *\*/*@RequestMapping(value = **"/notify/url"**)  
**public** String notifyUrl(HttpServletRequest request){  
 **try** {  
 *//1.读取支付回调数据* InputStream inStream = request.getInputStream();  
 *//2.使用Apache IOUtils把输入转换成字符* String result = IOUtils.*toString*(inStream,**"UTF-8"**);  
 *//将xml字符串转换成Map结构* Map<String, String> map = WXPayUtil.*xmlToMap*(result);  
 System.***out***.println(map);  
  
 *//发送MQ消息* **rabbitTemplate**.convertAndSend(**exchange**,**routing**, JSON.*toJSONString*(map));  
   
 *//3.包装响应数据设置* Map respMap = **new** HashMap();  
 respMap.put(**"return\_code"**,**"SUCCESS"**);  
 respMap.put(**"return\_msg"**,**"OK"**);  
 **return** WXPayUtil.*mapToXml*(respMap);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 *//记录错误日志* }  
 **return null**;  
}

## 监听MQ消息处理订单

在订单微服务中，我们需要监听MQ支付状态消息，并实现订单数据操作。

### 集成RabbitMQ

在订单微服务中，先集成RabbitMQ，再监听队列消息。

### 加入依赖

在pom.xml中引入如下依赖：

*<!--加入ampq-->*<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-amqp</**artifactId**>  
</**dependency**>

### application.yml配置

**server**:  
 **port**: 18089  
**spring**:  
 **application**:  
 **name**: order  
 **datasource**:  
 **driver-class-name**: com.mysql.cj.jdbc.Driver  
 **url**: jdbc:mysql://192.168.211.132:3306/changgou\_order?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC  
 **username**: root  
 **password**: 123456  
 **redis**:  
 **host**: 192.168.211.132  
 **port**: 6379  
 **main**:  
 **allow-bean-definition-overriding**: **true  
 rabbitmq**:  
 **host**: 192.168.211.132  
 **port**: 5672  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true***#hystrix 配置***hystrix**:  
 **command**:  
 **default**:  
 **execution**:  
 **isolation**:  
 **thread**:  
 *# 熔断超时设置，默认为1秒* **timeoutInMilliseconds**: 10000  
 *# 隔离模式使用信号量隔离SEMAPHORE 默认值是线程隔离THREAD* **strategy**: SEMAPHORE  
*#队列名***mq**:  
 **pay**:  
 **queue**:  
 **order**: queue.order

### 监听消息修改订单

在订单微服务于中创建com.changgou.order.consumer.OrderPayMessageListener，并在该类中consumeMessage方法，用于监听消息，并根据支付状态处理订单，代码如下：

*/\*\*  
 \* 订单消息监听器  
 \** ***@author*** *Hc  
 \** ***@description*** *com.changgou.order.consumer  
 \*/*@Component  
**public class** OrderPayMessageListener {  
  
 @RabbitListener(queues = **"${mq.pay.queue.order}"**)  
 **public void** payListener(String msg) {  
 *//将数据转成Map* Map<String, String> result = JSON.*parseObject*(msg, Map.**class**);  
  
 *//return\_code=SUCCESS* String return\_code = result.get(**"return\_code"**);  
  
 *//返回状态码 result\_code=SUCCESS/FAIL，修改订单状态* **if** (return\_code.equalsIgnoreCase(**"success"**)) {  
 *//业务结果* String result\_code = result.get(**"result\_code"**);  
 *//获取订单号* String out\_trade\_no = result.get(**"out\_trade\_no"**);  
 *//业务结果-SUCCESS/FAIL，为success时，支付成功* **if** (result\_code.equalsIgnoreCase(**"success"**)) {  
 *//修改订单状态 out\_trade\_no* } **else** {  
 *//订单删除* }  
 }  
 }  
}

# 订单状态操作

## Redis存储订单信息

每次添加订单后，会根据订单检查用户是否是否支付成功，我们不建议每次都操作数据库，每次操作数据库会增加数据库的负载，我们可以选择将用户的订单信息存入一份到Redis中，提升读取速度。

修改changgou-service-order微服务的com.changgou.order.service.impl.OrderServiceImpl类中的add方法，如果是线上支付，将用户订单数据存入到Redis中,由于每次创建二维码，需要用到订单编号与支付金额，所以也需要将添加的订单信息返回。

@Override  
**public** Order add(Order order){  
 *//省略其它代码…………*  
 *//必须要删除缓存前调用-扣减库存* **skuFeign**.decrCount(order.getUsername());  
 *//添加积分* **userFeign**.addPoints(10);  
 *//清除Redis缓存购物车数据* **redisTemplate**.delete(**"Cart\_"**+order.getUsername());  
  
 *//如果是在线支付* **if**(order.getPayType().equals(**"1"**)){  
 *//把订单放入redis中，便于查询* **redisTemplate**.boundHashOps(**"orders"**).put(order.getId(), order);  
 }  
 **return** order;  
}

## Controller返回订单号与金额

修改com.changgou.order.controller.OrderController的add方法，将订单对象返回，因为页面需要获取订单的金额和订单号用于创建二维码，代码如下：

@PostMapping  
**public** Result<Order> add(@RequestBody Order order){  
 *//获取登录名* String username = TokenDecode.*getUserInfo*().get(**"username"**);  
 order.setUsername(username);  
 *//调用OrderService实现添加Order* order = **orderService**.add(order);  
 **return new** Result(**true**,StatusCode.***OK***,**"下单成功"**,order);  
}

## 修改订单状态

订单支付成功后，需要修改订单状态并持久化到数据库，修改订单的同时，需要将Redis中的订单删除，所以修改订单状态需要将订单日志也传过来，实现代码如下：

修改com.changgou.order.service.OrderService，添加修改订单状态方法，代码如下：

*/\*\*\*  
 \* 支付成功-修改订单状态  
 \** ***@param transactionid*** *交易流水号  
 \** ***@param orderId*** *订单id  
 \*/***void** updateStatus(String orderId,String transactionid);

修改com.changgou.order.service.impl.OrderServiceImpl，添加修改订单状态实现方法，代码如下：

@Override  
**public void** updateStatus(String orderId, String transactionid) {  
 *//1.从redis中查询订单* Order order = (Order) **redisTemplate**.boundHashOps(**"orders"**).get(orderId);  
 *//修改状态* order.setUpdateTime(**new** Date()); *//时间也可以从微信接口返回过来，这里为了方便，我们就直接使用当前时间了* order.setPayTime(order.getUpdateTime());  
 order.setTransactionId(transactionid); *//交易流水号* order.setPayStatus(**"1"**); *//已支付* **orderMapper**.updateByPrimaryKeySelective(order);  
  
 *//2.删除Redis中的订单记录* **redisTemplate**.boundHashOps(**"orders"**).delete(orderId);  
}

## 删除订单

如果用户订单支付失败了，或者支付超时了，我们需要删除用户订单，删除订单的同时需要回滚库存，这里回滚库存我们就不实现了，作为同学们的作业。实现如下：

修改changgou-service-order的com.changgou.order.service.OrderService，添加删除订单方法，我们只需要将订单id传入进来即可实现，代码如下：

*/\*\*\*  
 \* 删除订单操作  
 \** ***@param orderId*** *\*/***void** deleteOrder(String orderId);

修改changgou-service-order的com.changgou.order.service.impl.OrderServiceImpl，添加删除订单实现方法，代码如下：

@Override  
**public void** deleteOrder(String orderId) {  
 *//1.从redis中查询订单* Order order = (Order) **redisTemplate**.boundHashOps(**"orders"**).get(orderId);  
 *//修改状态* order.setUpdateTime(**new** Date());  
 order.setPayStatus(**"2"**); *//支付失败* **orderMapper**.updateByPrimaryKeySelective(order);  
  
 *//注意，此处还要还原库存（作业）  
  
 //2.删除Redis中的订单记录* **redisTemplate**.boundHashOps(**"orders"**).delete(orderId);  
}

## 监听到支付消息时调用相关逻辑

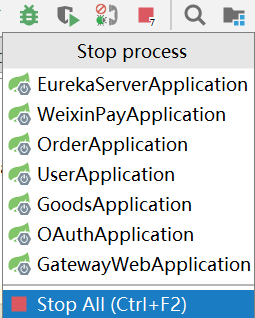
在订单微服务中，修改com.changgou.order.consumer.OrderPayMessageListener

@Component  
**public class** OrderPayMessageListener {  
 @Autowired  
 **private** OrderService **orderService**;  
  
 @RabbitListener(queues = **"${mq.pay.queue.order}"**)  
 **public void** payListener(String msg) {  
 *//将数据转成Map* Map<String, String> result = JSON.*parseObject*(msg, Map.**class**);  
 *//return\_code=SUCCESS* String return\_code = result.get(**"return\_code"**);  
  
 *//返回状态码 result\_code=SUCCESS/FAIL，修改订单状态* **if** (return\_code.equalsIgnoreCase(**"success"**)) {  
 *//业务结果* String result\_code = result.get(**"result\_code"**);  
 *//获取订单号* String out\_trade\_no = result.get(**"out\_trade\_no"**);  
 *//交易流水号* String transaction\_id = result.get(**"transaction\_id"**);  
 *//业务结果-SUCCESS/FAIL，为success时，支付成功* **if** (result\_code.equalsIgnoreCase(**"success"**)) {  
 *//修改订单状态 out\_trade\_no* **orderService**.updateStatus(out\_trade\_no,transaction\_id);  
 } **else** {  
 *//订单删除* **orderService**.deleteOrder(out\_trade\_no);  
 }  
 }  
 }  
}

## 集成整体测试

我们可以从添加购物车开始，一直到支付完成修改订单状态，整体流程测试一次。

**这次测试要启动的工程，如下：**



**浏览器操作：**

1. 登录系统

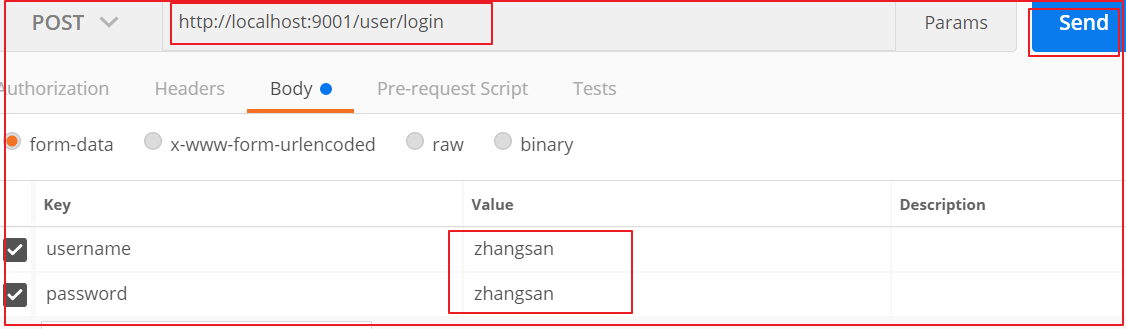
<http://localhost:8001/api/cart/list>

1. 添加购物车

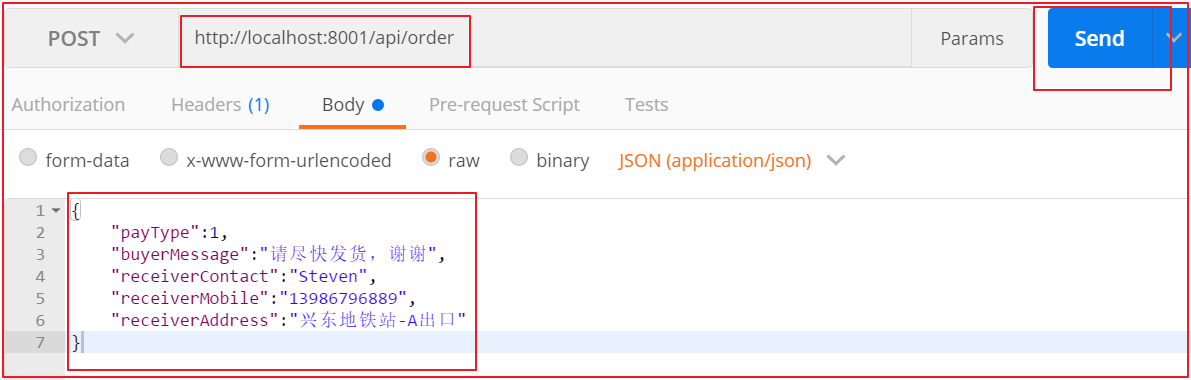
<http://localhost:8001/api/cart/add?num=1&id=1148477873175142400>

**Postman操作：**

1. 在postman中登录



1. 下单

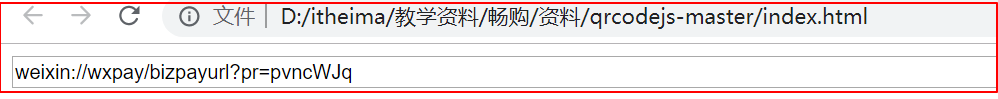


**浏览器操作：**

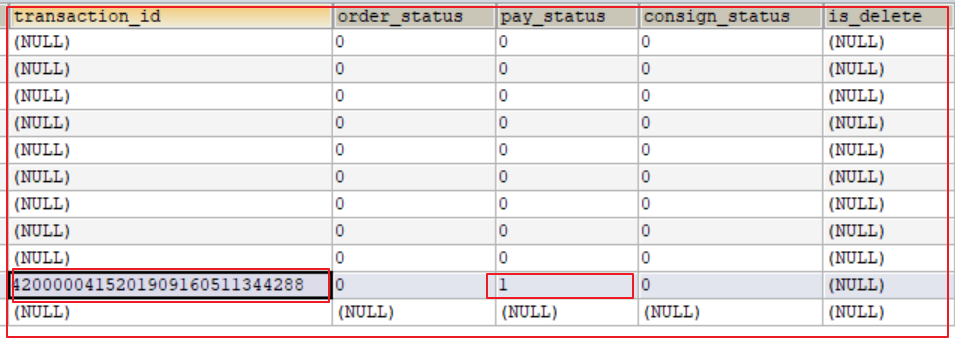
1. 生成二维码

<http://localhost:18090/weixin/pay/create/native?out_trade_no=NO1173449649143545856&total_fee=1>

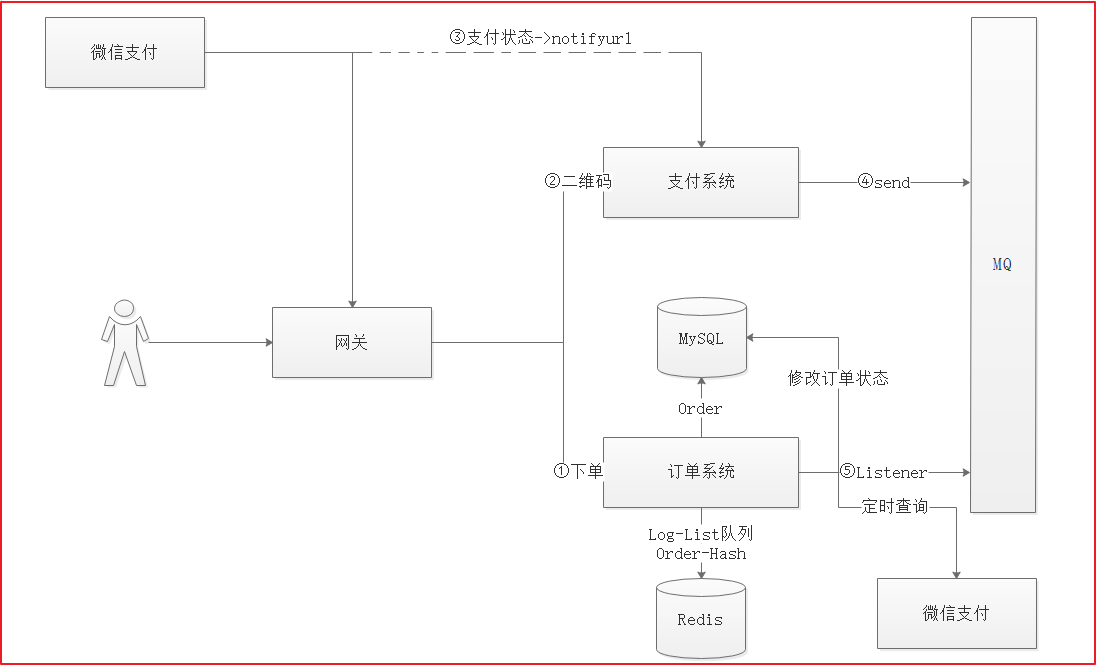
1. 显示二维码



1. 扫码支付，查看订单状态与流水



# 定时处理订单状态(学员完成)



## 业务分析

在现实场景中，可能会出现这么种情况，就是用户支付后，有可能畅购服务网络不通或者服务器挂了，此时会导致回调地址无法接收到用户支付状态，这时候我们需要取微信服务器查询。所以我们之前订单信息的ID存入到了Redis队列，主要用于解决这种网络不可达造成支付状态无法回调获取的问题。

实现思路如下：

1.每次下单，都将订单存入到Reids List队列中

2.定时每5秒检查一次Redis 队列中是否有数据，如果有，则再去查询微信服务器支付状态

3.如果已支付，则修改订单状态

4.如果没有支付，是等待支付，则再将订单存入到Redis队列中，等会再次检查

5.如果是支付失败，直接删除订单信息并修改订单状态