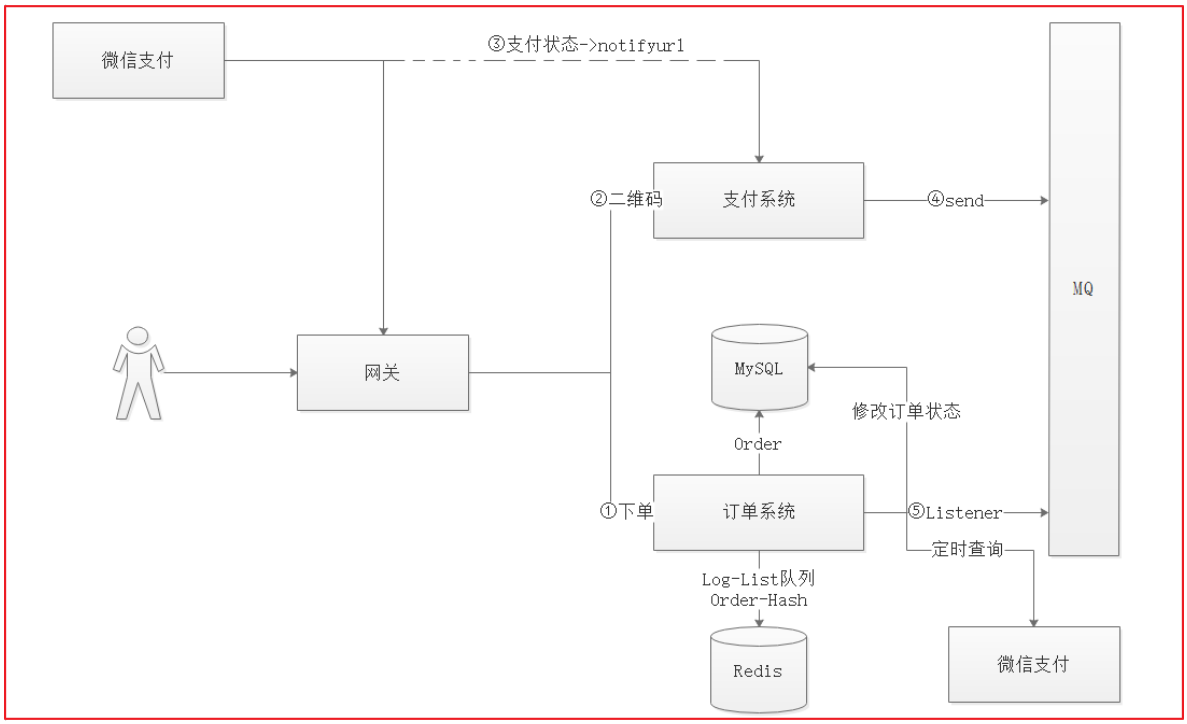
第12章 微信支付(如何调用第三方的api接口)

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(微信的服务器)
- 实现订单状态的修改、删除订单(逻辑删除)
- 支付状态回查->微信服务器将支付状态返回给支付微服务
- MQ处理支付回调状态
- RabbitMQ延时队列实现超时订单取消回滚

1 支付流程分析

1.1 订单支付分析



如上图，步骤分析如下：

```
1.用户下单之后，订单数据会存入到MySQL中，同时会将订单存入到Redis，以队列的方式存储。
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。
```

1.2 二维码创建

今天主要讲微信支付，后面为了看到效果，我们简单说下利用qrious制作二维码插件。

qrious是一款基于HTML5 Canvas的纯JS二维码生成插件。通过qrious.js可以快速生成各种二维码，你可以控制二维码的尺寸颜色，还可以将生成的二维码进行Base64编码。

qrious.js二维码插件的可用配置参数如下：

参数	类型	默认值	描述
background	String	"white"	二维码的背景颜色。
foreground	String	"black"	二维码的前景颜色。
level	String	"L"	二维码的误差校正级别(L, M, Q, H)。
mime	String	"image/png"	二维码输出为图片时的MIME类型。
size	Number	100	二维码的尺寸，单位像素。
value	String	""	需要编码为二维码的值

下面的代码即可生成一张二维码

```
<html>
<head>
<title>二维码入门小demo</title>
</head>
<body>
<img id="qrious">
<script src="qrious.js"></script>
<script>
 var qr = new QRious({
        element:document.getElementById('qrious'),
        size:250,
        level:'H',
        value:'http://www.itheima.com'
    });
</script>
</body>
</html>
```

运行效果：

```
<html>
<head>
<title>二维码入门小demo</title>
</head>
<body>
<img id="qrious">
<script src="qrious.min.js"></script>
<script>
 var qr = new QRious({
        element:document.getElementById('qrious'), 指定要创建二维码的标签
        size:250, 指定二维码大小
        level:'H', 指定二维码校正级别
        value:'http://www.itheima.com' 二维码扫码后跳转的路径
    });
</script>
</body>
</html>
```

大家掏出手机，扫一下看看是否会看到黑马的官网呢?

2 微信扫码支付简介

2.1 微信扫码支付申请

微信扫码支付是商户系统按微信支付协议生成支付二维码，用户再用微信“扫一扫”完成支付的模式。该模式适用于PC网站支付、实体店单品或订单支付、媒体广告支付等场景。

申请步骤：（了解）

第一步：注册公众号（类型须为：服务号）

请根据营业执照类型选择以下主体注册：个体工商户| 企业/公司| 政府| 媒体| 其他类型。

第二步：认证公众号

公众号认证后才可申请微信支付，认证费：300元/次。

第三步：提交资料申请微信支付

登录公众平台，点击左侧菜单【微信支付】，开始填写资料等待审核，审核时间为1-5个工作日内。

第四步：开户成功，登录商户平台进行验证

资料审核通过后，请登录联系人邮箱查收商户号和密码，并登录商户平台填写财付通备付金打的小额资金数额，完成账户验证。

第五步：在线签署协议

本协议为线上电子协议，签署后方可进行交易及资金结算，签署完立即生效。

本课程已经提供好“传智播客”的微信支付账号，学员无需申请。

2.2 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/index.html

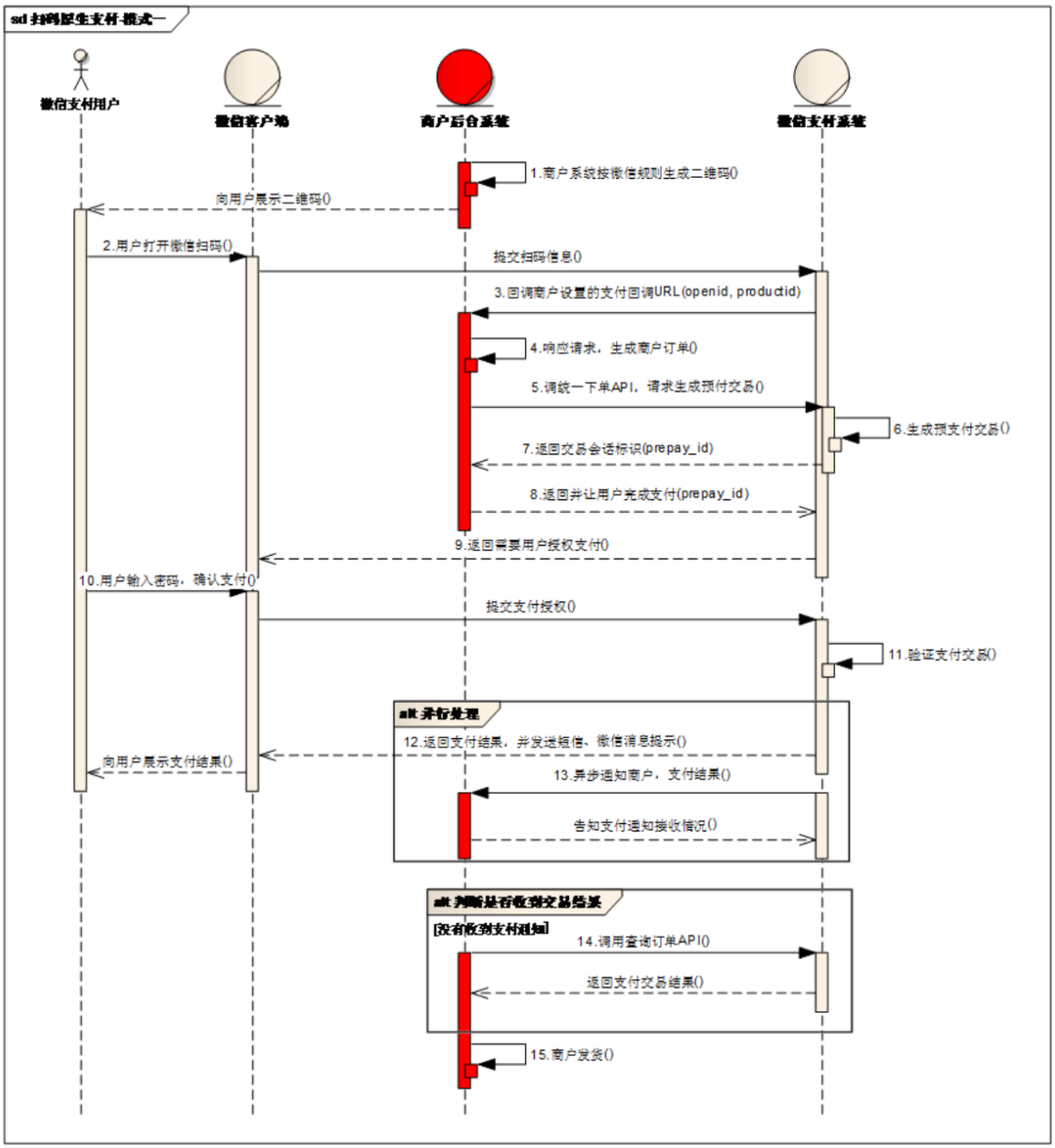
如果你不能联网，请查阅讲义配套资源 （资源\配套软件\微信扫码支付\开发文档）

我们在本章课程中会用到”统一下单”和”查询订单”两组API

```
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
```

2.3 微信支付模式介绍

2.3.1 模式一

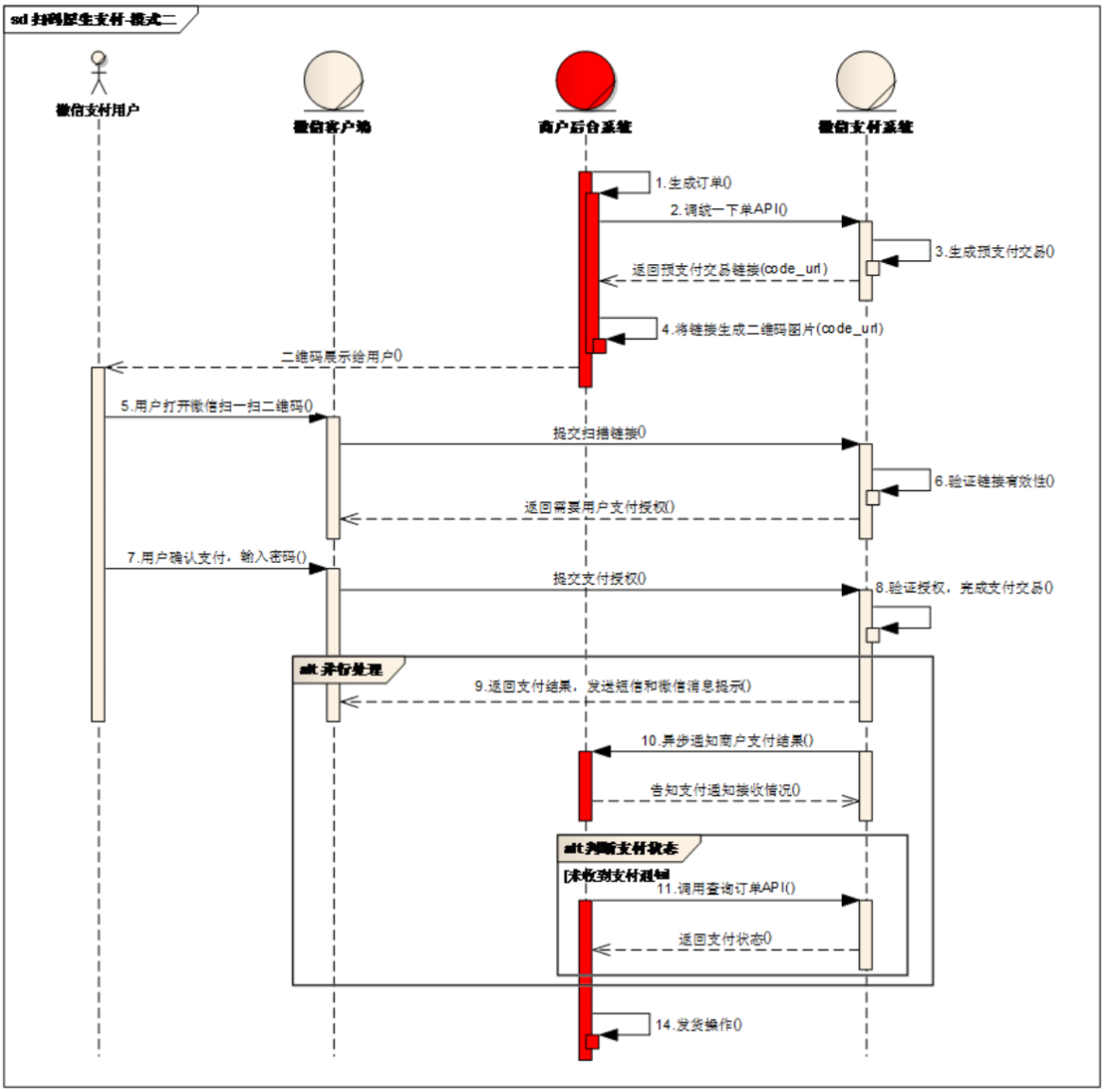


业务流程说明：

```
1.商户后台系统根据微信支付规定格式生成二维码（规则见下文），展示给用户扫码。
2.用户打开微信“扫一扫”扫描二维码，微信客户端将扫码内容发送到微信支付系统。
```

3. 微信支付系统收到客户端请求，发起对商户后台系统支付回调URL的调用。调用请求将带productid和用户的openid等参数，并要求商户系统返回交数据包,详细请见"本节3.1回调数据输入参数"
4. 商户后台系统收到微信支付系统的回调请求，根据productid生成商户系统的订单。
5. 商户系统调用微信支付【统一下单API】请求下单，获取交易会话标识（prepay_id）
6. 微信支付系统根据商户系统的请求生成预支付交易，并返回交易会话标识（prepay_id）。
7. 商户后台系统得到交易会话标识prepay_id（2小时内有效）。
8. 商户后台系统将prepay_id返回给微信支付系统。返回数据见"本节3.2回调数据输出参数"
9. 微信支付系统根据交易会话标识，发起用户端授权支付流程。
10. 用户在微信客户端输入密码，确认支付后，微信客户端提交支付授权。
11. 微信支付系统验证后扣款，完成支付交易。
12. 微信支付系统完成支付交易后给微信客户端返回交易结果，并将交易结果通过短信、微信消息提示用户。微信客户端展示支付交易结果页面。
13. 微信支付系统通过发送异步消息通知商户后台系统支付结果。商户后台系统需回复接收情况，通知微信后台系统不再发送该单的支付通知。
14. 未收到支付通知的情况，商户后台系统调用【查询订单API】。
15. 商户确认订单已支付后给用户发货。

2.3.2 模式二



业务流程说明：

```
1.商户后台系统根据用户选购的商品生成订单。
2.用户确认支付后调用微信支付【统一下单API】生成预支付交易；
3.微信支付系统收到请求后生成预支付交易单，并返回交易会话的二维码链接code_url。
4.商户后台系统根据返回的code_url生成二维码。
5.用户打开微信“扫一扫”扫描二维码，微信客户端将扫码内容发送到微信支付系统。
6.微信支付系统收到客户端请求，验证链接有效性后发起用户支付，要求用户授权。
7.用户在微信客户端输入密码，确认支付后，微信客户端提交授权。
8.微信支付系统根据用户授权完成支付交易。
9.微信支付系统完成支付交易后给微信客户端返回交易结果，并将交易结果通过短信、微信消息提示用户。微信客户端展示支付交易结果页面。
10.微信支付系统通过发送异步消息通知商户后台系统支付结果。商户后台系统需回复接收情况，通知微信后台系统不再发送该单的支付通知。
11.未收到支付通知的情况，商户后台系统调用【查询订单API】。
12.商户确认订单已支付后给用户发货。
```

2.3 微信支付SDK

微信支付提供了SDK(微信支付开发工具包), 大家下载后打开源码，install到本地仓库。

SDK与DEMO下载

平台和语言	说明	支付模式	操作
JAVA	【微信支付】API对应的SDK和调用示例	刷卡支付、公众号支付、扫码支付	下载
.NET C#	【微信支付】API对应的SDK和调用示例	刷卡支付、公众号支付、扫码支付	下载
PHP	【微信支付】API对应的SDK和调用示例	刷卡支付、公众号支付、扫码支付	下载

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

```
<!--微信支付-->
<dependency>
    <groupId>com.github.wxpay</groupId>
    <artifactId>wxpay-sdk</artifactId>
    <version>0.0.3</version>
</dependency>
```

2.4 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工具类代码：

```
public class HttpClient {
    private String url;
    private Map<String, String> param;
    private int statusCode;
    private String content;
    private String xmlParam;
    private boolean isHttps;

    public boolean isHttps() {
        return isHttps;
    }

    public void setHttps(boolean isHttps) {
        this.isHttps = isHttps;
    }

    public String getXmlParam() {
        return xmlParam;
    }

    public void setXmlParam(String xmlParam) {
        this.xmlParam = xmlParam;
    }

    public HttpClient(String url, Map<String, String> param) {
        this.url = url;
        this.param = param;
```

```
28      }
29
30      public HttpClient(String url) {
31          this.url = url;
32      }
33
34      public void setParameter(Map<String, String> map) {
35          param = map;
36      }
37
38      public void addParameter(String key, String value) {
39          if (param == null)
40              param = new HashMap<String, String>();
41          param.put(key, value);
42      }
43
44      public void post() throws ClientProtocolException, IOException {
45          HttpPost http = new HttpPost(url);
46          setEntity(http);
47          execute(http);
48      }
49
50      public void put() throws ClientProtocolException, IOException {
51          HttpPut http = new HttpPut(url);
52          setEntity(http);
53          execute(http);
54      }
55
56      public void get() throws ClientProtocolException, IOException {
57          if (param != null) {
58              StringBuilder url = new StringBuilder(this.url);
59              boolean isFirst = true;
60              for (String key : param.keySet()) {
61                  if (isFirst) {
62                      url.append("?");
63                  }else {
64                      url.append("&");
65                  }
66                  url.append(key).append("=").append(param.get(key));
67              }
68              this.url = url.toString();
69          }
70          HttpGet http = new HttpGet(url);
71          execute(http);
72      }
73
74      /**
75       * set http post,put param
76       */
77      private void setEntity(HttpEntityEnclosingRequestBase http) {
78          if (param != null) {
79              List<NameValuePair> nvps = new LinkedList<NameValuePair>();
80              for (String key : param.keySet()) {
81                  nvps.add(new BasicNameValuePair(key, param.get(key))); //
参数
82              }
83              http.setEntity(new UrlEncodedFormEntity(nvps, Consts.UTF_8));
// 设置参数
```

```
84          }
85          if (xmlParam != null) {
86              http.setEntity(new StringEntity(xmlParam, Consts.UTF_8));
87          }
88      }
89
90      private void execute(HttpUriRequest http) throws
    ClientProtocolException,
91              IOException {
92          CloseableHttpClient httpClient = null;
93          try {
94              if (isHttps) {
95                  SSLContext sslContext = new SSLContextBuilder()
96                          .loadTrustMaterial(null, new TrustStrategy() {
97                              // 信任所有
98                              @Override
99                              public boolean isTrusted(X509Certificate[]
    chain,
100                                                     String authType)
101                                     throws CertificateException {
102                                 return true;
103                             }
104                         }).build();
105                 SSLConnectionSocketFactory sslsf = new
    SSLConnectionSocketFactory(
106                         sslContext);
107                 httpClient =
    HttpClients.custom().setSSLSocketFactory(sslsf)
108                         .build();
109             } else {
110                 httpClient = HttpClients.createDefault();
111             }
112             CloseableHttpResponse response = httpClient.execute(http);
113             try {
114                 if (response != null) {
115                     if (response.getStatusLine() != null) {
116                         statusCode =
    response.getStatusLine().getStatusCode();
117                     }
118                     HttpEntity entity = response.getEntity();
119                     // 响应内容
120                     content = EntityUtils.toString(entity, Consts.UTF_8);
121                 }
122             } finally {
123                 response.close();
124             }
125         } catch (Exception e) {
126             e.printStackTrace();
127         } finally {
128             httpClient.close();
129         }
130     }
131
132     public int getStatusCode() {
133         return statusCode;
134     }
135
136     public String getContent() throws ParseException, IOException {
```

```
            return content;
        }
    }
```

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>
```

2.5 支付微服务搭建

(1)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。

(2)application.yml

创建application.yml，配置文件如下：

```
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
#微信支付信息配置
weixin:
  appid: wx8397f8696b538317
  partner: 1473426802
  partnerkey: T6m9iK73b0kn9g5v426MKfHQH7X8rKwb
  notifyurl: http://www.itcast.cn
```

`appid`： 微信公众账号或开放平台APP的唯一标识

`partner`：财付通平台的商户账号

`partnerkey`：财付通平台的商户密钥

`notifyurl`：回调地址

(3)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);
    }
}
```

支付的域名为`pay-changgou-java.itheima.net`，我们可以在虚拟机中配置下该域名。

3 微信支付二维码生成

3.1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

3.2 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。

3.3 代码实现

(1)业务层

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

```
public interface WeixinPayService {
    /*****
     * 创建二维码
     * @param out_trade_no : 客户端自定义订单编号
     * @param total_fee    : 交易金额,单位：分
     * @return
     */
    Map createNative(String out_trade_no, String total_fee);
}
```

创建 `com.changgou.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：

```
@Service
public class WeixinPayServiceImpl implements WeixinPayService {

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

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

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

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

    /****
     * 创建二维码
     * @param out_trade_no : 客户端自定义订单编号
     * @param total_fee    : 交易金额,单位：分
     * @return
     */
    @Override
    public Map createNative(String out_trade_no, String total_fee){
        try {
            //1、封装参数
```

```
            Map param = new HashMap();
            param.put("appid", appid);                                  //应用ID
            param.put("mch_id", partner);                               //商户ID
号
            param.put("nonce_str", WXPayUtil.generateNonceStr());       //随机数
            param.put("body", "畅购");                                   //订单
描述
            param.put("out_trade_no",out_trade_no);                     //商户订
单号
            param.put("total_fee", total_fee);                          //交易金
额
            param.put("spbill_create_ip", "127.0.0.1");                //终端IP
            param.put("notify_url", notifyurl);                         //回调地址
            param.put("trade_type", "NATIVE");                          //交易类型

            //2、将参数转成xml字符，并携带签名
            String paramXml = WXPayUtil.generateSignedXml(param,
partnerkey);

            ///3、执行请求
            HttpClient httpClient = new
HttpClient("https://api.mch.weixin.qq.com/pay/unifiedorder");
            httpClient.setHttps(true);
            httpClient.setXmlParam(paramXml);
            httpClient.post();

            //4、获取参数
            String content = httpClient.getContent();
            Map<String, String> stringMap = WXPayUtil.xmlToMap(content);
            System.out.println("stringMap:"+stringMap);

            //5、获取部分页面所需参数
            Map<String,String> dataMap = new HashMap<String,String>();
            dataMap.put("code_url",stringMap.get("code_url"));
            dataMap.put("out_trade_no",out_trade_no);
            dataMap.put("total_fee",total_fee);

            return dataMap;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

(2) 控制层

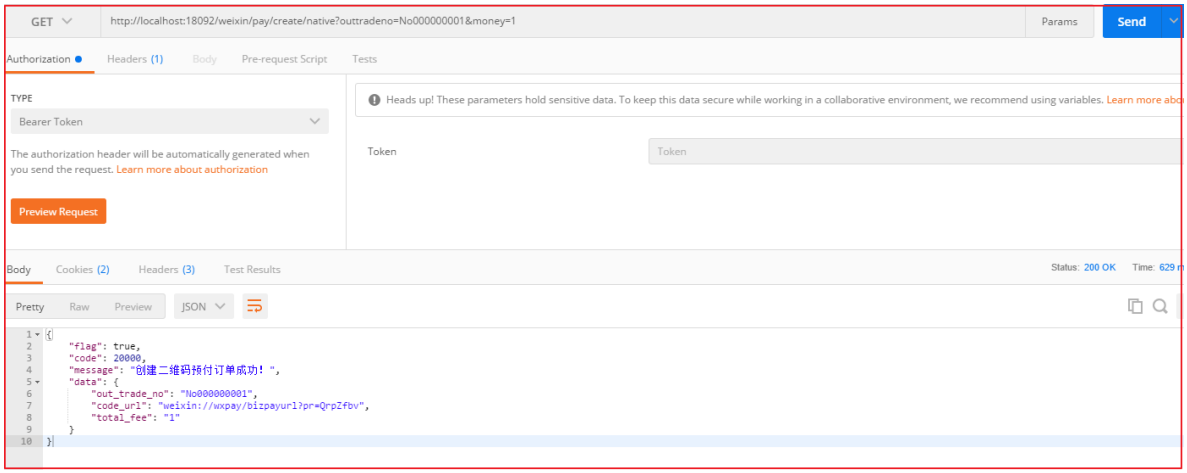
创建`com.changgou.controller.WeixinPayController`,主要调用WeixinPayService的方法获取创建二维码的信息，代码如下：

```
@RestController
@RequestMapping(value = "/weixin/pay")
public class WeixinPayController {

    @Autowired
```

```
    private WeixinPayService weixinPayService;

    /***
     * 创建二维码
     * @return
     */
    @RequestMapping(value = "/create/native")
    public Result createNative(String outtradeno, String totalfee){
        Map<String,String> resultMap = weixinPayService.createNative(outtradeno,totalfee);
        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:18092/weixin/pay/create/native?outtradeno=No000000001&money=1`



打开支付页面/pay.html，修改value路径，然后打开，会出现二维码，可以扫码试试

```
<html>
<head>
<title>二维码入门小demo</title>
</head>
<body>
<img id="qrious">
<script src="qrious.js"></script>
<script>
 var qr = new QRious({
        element:document.getElementById('qrious'),
        size:250,
        level:'H',
        value:'weixin://wxpay/bizpayurl?pr=byibYaQ'
    });
</script>
</body>
</html>
```

测试如下：

4 检测支付状态

4.1 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

4.2 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。

4.3 代码实现

(1)业务层

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

```
/***
 * 查询订单状态
 * @param out_trade_no : 客户端自定义订单编号
 * @return
 */
Map queryPayStatus(String out_trade_no);
```

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

```
/***
 * 查询订单状态
 * @param out_trade_no : 客户端自定义订单编号
 * @return
 */
@Override
public Map queryPayStatus(String out_trade_no) {
    try {
        //1.封装参数
        Map param = new HashMap();
        param.put("appid",appid);                                   //应用ID
        param.put("mch_id",partner);                                //商户号
        param.put("out_trade_no",out_trade_no);                     //商户订单编号
        param.put("nonce_str",WXPayUtil.generateNonceStr()); //随机字符

        //2、将参数转成xml字符，并携带签名
        String paramXml = WXPayUtil.generateSignedXml(param,partnerkey);

        //3、发送请求
        HttpClient httpClient = new
HttpClient("https://api.mch.weixin.qq.com/pay/orderquery");
        httpClient.setHttps(true);
        httpClient.setXmlParam(paramXml);
        httpClient.post();

        //4、获取返回值，并将返回值转成Map
        String content = httpClient.getContent();
        return WXPayUtil.xmlToMap(content);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

(2)控制层

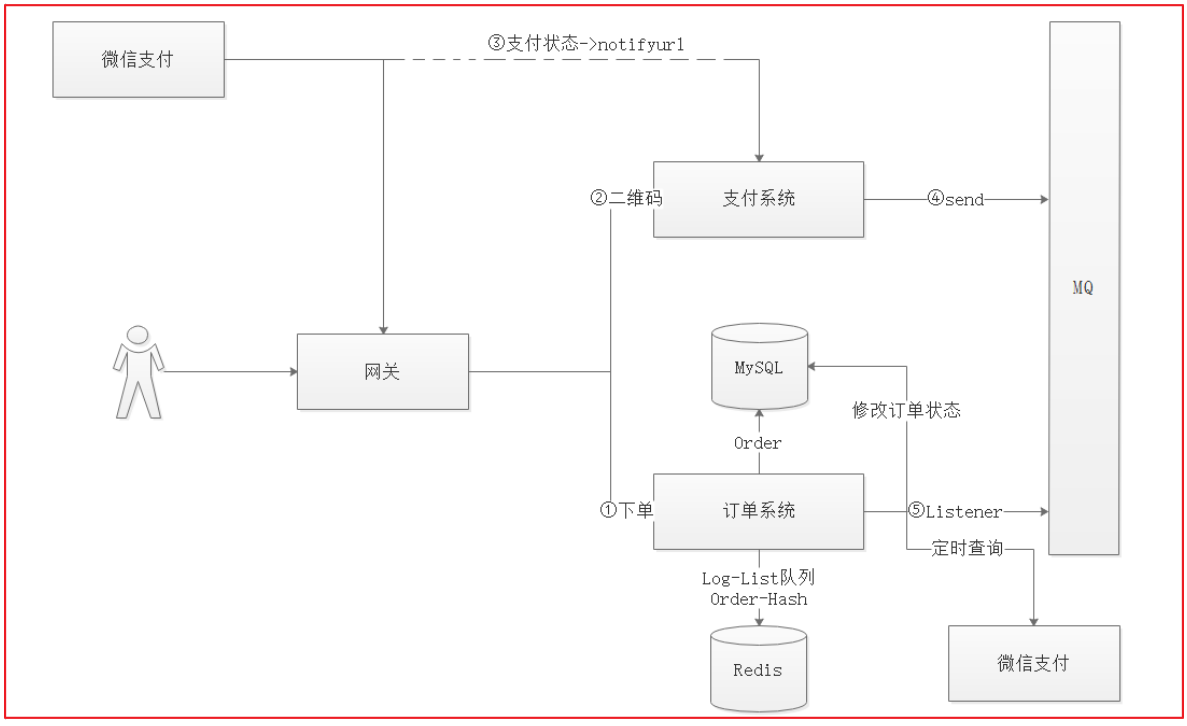
在`com.changgou.controller.WeixinPayController`新增方法，用于查询支付状态，代码如下：

上图代码如下：

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

5 订单状态操作准备工作

5.1 需求分析



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)。
```

5.2 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，所以也需要将添加的订单信息返回。

```
/**
 * 增加Order
 * 金额校验:后台校验
 * @param order
 */
@Override
public Order add(Order order){
    //...略

    //修改库存
    skuFeign.decrCount(order.getUsername());

    //添加用户积分
    userFeign.addPoints(2);

    //线上支付，记录订单
    if(order.getPayType().equalsIgnoreCase( anotherString: "1")){
        //将支付记录存入到Reids namespace  key  value
        redisTemplate.boundHashOps( key: "Order").put(order.getId(),order);
    }

    //删除购物车信息
    //redisTemplate.delete("Cart_" + order.getUsername());

    return order;
}
```

将订单信息存入到Redis中

上图代码如下：

```
/**
 * 增加Order
 * 金额校验:后台校验
```

```
     * @param order
     */
    @Override
    public Order add(Order order){
        //...略

        //修改库存
        skuFeign.decrCount(order.getUsername());

        //添加用户积分
        userFeign.addPoints(2);

        //线上支付，记录订单
        if(order.getPayType().equalsIgnoreCase("1")){
            //将支付记录存入到Reids namespace  key  value
            redisTemplate.boundHashOps("Order").put(order.getId(),order);
        }

        //删除购物车信息
        //redisTemplate.delete("Cart_" + order.getUsername());

        return order;
    }
```

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：

```
/***
 * 新增Order数据
 * @param order
 * @return
 */
@PostMapping
public Result<Order> add(@RequestBody   Order order){
    //获取用户名
    String username = TokenDecode.getUserInfo().get("username");
    order.setUsername(username);
    //调用OrderService实现添加Order
    order = orderService.add(order);
    return new Result<Order>( flag: true, StatusCode.OK, message: "添加成功",order);
}
```

5.3 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：

```
    /***
     * 根据订单ID修改订单状态
     * @param transactionid 交易流水号
     * @param orderId
     */
    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：

```
/***
 * 订单修改
 * @param orderId
 * @param transactionid  微信支付的交易流水号
 */
@Override
public void updateStatus(String orderId,String transactionid) {
    //1.修改订单
    Order order = orderMapper.selectByPrimaryKey(orderId);
    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());    //不允许这么写
    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("Order").delete(orderId);
}
```

5.4 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 id
 */
void deleteOrder(String id);
```

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：

```
/***
 * 订单的删除操作
 */
@Override
public void deleteOrder(String id) {
    //改状态
    Order order = (Order) redisTemplate.boundHashOps("Order").get(id);
    order.setUpdateTime(new Date());
    order.setPayStatus("2");    //支付失败
    orderMapper.updateByPrimaryKeySelective(order);
```

```
11
12      //删除缓存
13      redisTemplate.boundHashOps("Order").delete(id);
14  }
```

6 支付信息回调

6.1 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`

6.1.1 返回参数分析

通知参数如下：

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
返回信息	return_msg	是	String(128)	OK	OK

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	微信分配的公众账号ID（企业号corpid即为此appId）
业务结果	result_code	是	String(16)	SUCCESS	SUCCESS/FAIL
商户订单号	out_trade_no	是	String(32)	1212321211201407033568112322	商户系统内部订单号
微信支付订单号	transaction_id	是	String(32)	1217752501201407033233368018	微信支付订单号

6.1.2 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。

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	请按示例值填写
返回信息	return_msg	是	String(128)	OK	请按示例值填写

举例如下：

```
<xml>
  <return_code><![CDATA[SUCCESS]]></return_code>
  <return_msg><![CDATA[OK]]></return_msg>
</xml>
```

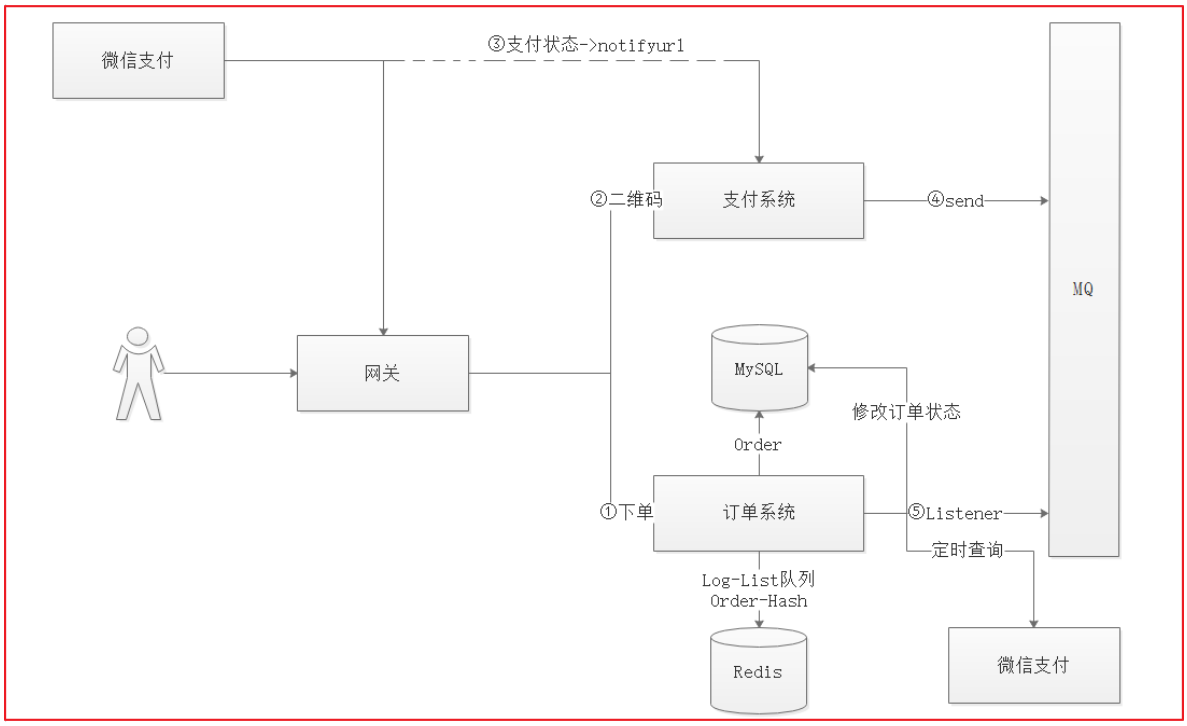
6.2 回调接收数据实现

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
 * @param request
 * @return
 */
@RequestMapping(value = "/notify/url")
public String notifyUrl(HttpServletRequest request){
    InputStream inStream;
    try {
        //读取支付回调数据
        inStream = request.getInputStream();
        ByteArrayOutputStream outSteam = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int len = 0;
        while ((len = inStream.read(buffer)) != -1) {
            outSteam.write(buffer, 0, len);
        }
        outSteam.close();
        inStream.close();
        // 将支付回调数据转换成xml字符串
        String result = new String(outSteam.toByteArray(), "utf-8");
        //将xml字符串转换成Map结构
        Map<String, String> map = WXPayUtil.xmlToMap(result);

        //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;
}
```

7 MQ处理支付回调状态

7.1 业务分析



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。

7.2 发送支付状态

(1)集成RabbitMQ

修改支付微服务，集成RabbitMQ，添加如下依赖：

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

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。

修改application.yml，配置支付队列和交换机信息，代码如下：

```yaml
#位置支付交换机和队列
mq:
  pay:
    exchange:
      order: exchange.order
    queue:
      order: queue.order
    routing:
      key: queue.order
```

记得配置RabbitMQ的地址

```yaml
  rabbitmq:
    host: 192.168.211.132
    port: 5672
```

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：

```
/***
 * 创建DirectExchange交换机
 * @return
 */
@Bean
public DirectExchange basicExchange(){
    return new DirectExchange(env.getProperty("mq.pay.exchange.order"),
true,false);
}

/***
 * 创建队列
 * @return
 */
@Bean(name = "queueOrder")
public Queue queueOrder(){
    return new Queue(env.getProperty("mq.pay.queue.order"), true);
}

/****
 * 队列绑定到交换机上
 * @return
 */
@Bean
public Binding basicBinding(){
    return
BindingBuilder.bind(queueOrder()).to(basicExchange()).with(env.getProperty("
mq.pay.routing.key"));
}
```

7.2.2 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;
```

队列交换机信息注入

```
@Autowired
private WeixinPayService weixinPayService;

@Autowired
private RabbitTemplate rabbitTemplate;

/***
 * 支付回调
 * @param request
 * @return
 */
@RequestMapping(value = "/notify/url")
public String notifyUrl(HttpServletRequest request){
    InputStream inStream;
    try {
        //读取支付回调数据
        inStream = request.getInputStream();
        ByteArrayOutputStream outSteam = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int len = 0;
        while ((len = inStream.read(buffer)) != -1) {
            outSteam.write(buffer, off: 0, len);
        }
        outSteam.close();
        inStream.close();
        // 将支付回调数据转换成xml字符串
        String result = new String(outSteam.toByteArray(), charsetName: "utf-8");
        //将xml字符串转换成Map结构
        Map<String, String> map = WXPayUtil.xmlToMap(result);
        //将消息发送给RabbitMQ
        rabbitTemplate.convertAndSend(exchange,routing, JSON.toJSONString(map));

        //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;
}
```

发送消息到MQ

上图代码如下：

```
@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 WeixinPayService weixinPayService;

```

```java
@Autowired
private RabbitTemplate rabbitTemplate;

/***
 * 支付回调
 * @param request
 * @return
 */
@RequestMapping(value = "/notify/url")
public String notifyUrl(HttpServletRequest request){
    InputStream inStream;
    try {
        //读取支付回调数据
        inStream = request.getInputStream();
        ByteArrayOutputStream outSteam = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int len = 0;
        while ((len = inStream.read(buffer)) != -1) {
            outSteam.write(buffer, 0, len);
        }
        outSteam.close();
        inStream.close();
        // 将支付回调数据转换成xml字符串
        String result = new String(outSteam.toByteArray(), "utf-8");
        //将xml字符串转换成Map结构
        Map<String, String> map = WXPayUtil.xmlToMap(result);
        //将消息发送给RabbitMQ
        rabbitTemplate.convertAndSend(exchange,routing,
JSON.toJSONString(map));

        //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;
}
```

7.3 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。

7.3.1 集成RabbitMQ

在订单微服务中，先集成RabbitMQ，再监听队列消息。

在pom.xml中引入如下依赖：

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

在application.yml中配置rabbitmq配置，代码如下：

```yaml
spring:
  application:
    name: order
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/changgou_order?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
    username: root
    password: itcast
  rabbitmq:
    host: 127.0.0.1 #mq的服务器地址
    username: guest #账号
    password: guest #密码
```

在application.yml中配置队列名字，代码如下：

```yaml
#位置支付交换机和队列
mq:
  pay:
    queue:
      order: queue.order
```

7.3.2 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：

```java
@Component
@RabbitListener(queues = {"${mq.pay.queue.order}"})
public class OrderPayMessageListener {

    @Autowired
    private RedisTemplate redisTemplate;

    @Autowired
    private OrderService orderService;

    /***
     * 接收消息
     */
    @RabbitHandler
    public void consumeMessage(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");
```

```
        //业务结果
        String result_code = result.get("result_code");

        //业务结果 result_code=SUCCESS/FAIL，修改订单状态
        if(return_code.equalsIgnoreCase("success") ){
            //获取订单号
            String outtradeno = result.get("out_trade_no");
            //业务结果
            if(result_code.equalsIgnoreCase("success")){
                if(outtradeno!=null){
                    //修改订单状态  out_trade_no

orderService.updateStatus(outtradeno,result.get("transaction_id"));
                }
            }else{
                //订单删除
                orderService.deleteOrder(outtradeno);
            }
        }

    }
}
```

8 定时处理订单状态(学员完成)

8.1 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10分钟检查一次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
```