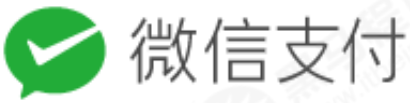
第10章 微信支付解决方案

学习目标：

- 能够根据微信支付的开发文档调用微信支付的api
- 使用QRcode.js在页面生成二维码
- 掌握内网映射工具EchoSite的配置与使用
- 完成统一下单与支付回调的逻辑处理
- 能够运用RabbitMQ Web STOMP 插件实现服务端向浏览器的消息推送
- 掌握rabbitmq的延迟消息的使用方法。

1. 微信支付快速入门



1.1 微信支付申请（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。

完成上述步骤，你可以得到调用API用到的账号和密钥

appid：微信公众账号或开放平台APP的唯一标识 wx8397f8696b538317

mch_id：商户号 1473426802

key：商户密钥 T6m9iK73b0kn9g5v426MKfHQH7X8rKwb

1.2 微信支付开发文档与SDK

在线微信支付开发文档：

[https://pay.weixin.qq.com/wiki/doc/api/index.html

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。

我们解压从官网下载的sdk ,安装到本地仓库

1.3 统一下单API

（1）新建工程，引入微信支付Api

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

（2）创建Config类，继承自抽象类WXPayConfig

```
public class Config extends WXPayConfig {
    String getAppID() {
        return "wx8397f8696b538317";
    }

    String getMchID() {
        return "1473426802";
    }

    String getKey() {
        return "T6m9iK73b0kn9g5v426MKfHQH7X8rKwb";
    }

    InputStream getCertStream() {
        return null;
    }

    IWXPayDomain getWXPayDomain() {
        return new IWXPayDomain() {
            public void report(String domain, long elapsedTimeMillis,
Exception ex) {

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

（3）创建测试类，编写代码

```
Config config=new Config();
//1.封装请求参数
Map<String,String> map=new HashMap();
map.put("appid",config.getAppID());//公众账号ID
map.put("mch_id",config.getMchID());//商户号
map.put("nonce_str",WXPayUtil.generateNonceStr());//随机字符串
map.put("body","青橙");//商品描述
map.put("out_trade_no","555552");//订单号
map.put("total_fee","1");//金额
map.put("spbill_create_ip","127.0.0.1");//终端IP
map.put("notify_url","http://www.baidu.com");//回调地址
map.put("trade_type","NATIVE");//交易类型

String xmlParam = WXPayUtil.generateSignedXml(map, config.getKey());
//xml格式的参数
System.out.println("参数: "+xmlParam);

//2.发送请求
WXPayRequest wxPayRequest=new WXPayRequest(config);
String xmlResult = wxPayRequest.requestWithCert("/pay/unifiedorder",
null, xmlParam, false);
System.out.println("结果: "+xmlResult);

//3.解析返回结果
Map<String, String> mapResult = WXPayUtil.xmlToMap(xmlResult);
String code_url = mapResult.get("code_url");
System.out.println(code_url);
```

在调用API的代码中，我们经常用到WXPayUtil提供的以下功能：

```
WXPayUtil.generateNonceStr();//获取随机字符串
WXPayUtil.generateSignedXml(param, partnerkey);//MAP转换为XML字符串（自动添加签名）
WXPayUtil.xmlToMap(result);//将XML字符串转换为MAP
```

WXPayRequest为API的支付请求类，封装了httpClient。

执行后返回结果

```
<xml><return_code><![CDATA[SUCCESS]]></return_code>
<return_msg><![CDATA[OK]]></return_msg>
<appid><![CDATA[wx8397f8696b538317]]></appid>
<mch_id><![CDATA[1473426802]]></mch_id>
<nonce_str><![CDATA[jcWWtgnQ1aYXmErr]]></nonce_str>
<sign><![CDATA[EBBB64AB3BA017B4C60C09C507E36C1B]]></sign>
<result_code><![CDATA[SUCCESS]]></result_code>
<prepay_id><![CDATA[wx23202155305471f07f6c3d398026955400]]></prepay_id>
<trade_type><![CDATA[NATIVE]]></trade_type>
<code_url><![CDATA[weixin://wxpay/bizpayurl?pr=Y3hDTZy]]></code_url>
</xml>
```

其中的code_url就是我们的支付URl ， 我们可以根据这个URl 生成支付二维码

1.4 二维码JS插件- QRcode.js

QRCode.js 是一个用于生成二维码的 JavaScript 库。主要是通过获取 DOM 的标签,再通过 HTML5 Canvas 绘制而成,不依赖任何库。支持该库的浏览器有：IE6~10, Chrome, Firefox, Safari, Opera, Mobile Safari, Android, Windows Mobile, 等

我们新建页面，编写一下代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
<head>
<title>QRCode</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<script type="text/javascript" src="qrcode.min.js"></script>
</head>
<body>
<div id="qrcode"></div>
<script type="text/javascript">
var qrcode = new QRCode(document.getElementById("qrcode"), {
    width : 200,
    height : 200
});
qrcode.makeCode("weixin://wxpay/bizpayurl?pr=Y3hDTZy");
</script>
</body>
</html>
```

1.5 内网映射工具EchoSite

在请求统一下单接口时，有个参数notify_url，这个是回调地址，也就是在支付成功后微信支付会自动访问这个地址，通知业务系统支付完成。但这个地址必须是互联网可以访问的（也就是有域名的）。

那么如何测试呢？我们可以借助一个工具 EchoSite 内网映射工具

（1）打开网址： https://www.echosite.cn/ 注册用户，登录到控制台下载客户端。

（2）支付2元买一个域名（可以用1个月），点击域名端口---抢注域名

抢注独享域名

选择需要的协议类型

http　　https

选择服务器

Easy - 共享型　　easy.echosite.cn
2核4GB 4M带宽　　2元/月

Cross - 共享型　　cross.echosite.cn
4核8GB 4M带宽　　3元/月

Thanks - 企业型　　thanks.echosite.cn
2 vCPU 4 GiB (I/O优化) 4M带宽（充值达到200元可以开票）　　10元/月

测试服务器不要买　　oauth.echoface.online
可以说配置不堪入目　　50元/月

输入你想要的域名

qingcheng　　easy.echosite.cn

选择抢占时间

1个月

支付金额：2 元

取 消　　确定支付

（3）使用课程提供的软件echosite ， 添加config.yml

```
#   这是你的 EchoSite 购买域名的服务器标志
server_addr: easy.echosite.cn:4443
trust_host_root_certs: false

echosite_id: 17799998888
echosite_token:
$2y$10$bY081mt/2KAFJLJXrsSNHe3f.6SM.SGfXgu1gG7aJvmPMPN9BTqrS

#   以下是你需要开启的通道，只能开启属于你的域名通道
#   以下分别是 http 和 https 以及 tcp 协议的示例
tunnels:
  name1:
    subdomain: "qingcheng"
    proto:
      http: 127.0.0.1:9102
```

然后在echosite目录中输入以下命令

```
echosite -config=config.yml start-all
```

这样你购买的域名就映射到127.0.0.1:9102上了。 ctrl+c 结束程序

2. 青橙-微信支付二维码

2.1 需求分析

用户在提交订单后，如果是选择支付方式为微信支付，那应该跳转到微信支付二维码页面，用户扫描二维码可以进行支付，金额与订单金额相同。

2.2 实现思路

前端页面向后端传递订单号，后端根据订单号查询订单，检查是否为当前用户的未支付订单，如果是则根据订单号和金额生成支付url返给前端，前端得到支付url生成支付二维码。

2.3 代码实现

2.3.1 后端代码

（1）qingcheng_service_order服务 pom.xml添加依赖

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

（2）创建config类

```
public class Config extends WXPayConfig {
    String getAppID() {
        return "wx8397f8696b538317";
    }

    String getMchID() {
        return "1473426802";
    }

    String getKey() {
        return "T6m9iK73b0kn9g5v426MKfHQH7X8rKwb";
    }

    InputStream getCertStream() {
        return null;
    }

    IWXPayDomain getWXPayDomain() {
        return new IWXPayDomain() {
            public void report(String domain, long elapsedTimeMillis,
Exception ex) {
            }
            public DomainInfo getDomain(WXPayConfig config) {
                return new DomainInfo("api.mch.weixin.qq.com",true);
            }
        };
    }
}
```

在spring配置文件中进行配置

```
    <!--微信支付配置-->
    <bean id="config" class="com.github.wxpay.sdk.Config"></bean>
```

（3）qingcheng_interface创建包com.qingcheng.service.pay ， 包下创建接口

```
/**
 * 微信支付接口
 */
public interface WeixinPayService {

    /**
     * 生成微信支付二维码
     * @param orderId 订单号
     * @param money 金额(分)
     * @param notifyUrl 回调地址
     * @return
     */
    public Map createNative(String orderId,Integer money,String
notifyUrl);
}
```

（4）qingcheng_service_pay新增服务类

```
@Service
public class WeixinPayServiceImpl implements WeixinPayService {

    @Autoware
    private Config config;

    public Map createNative(String orderId, Integer money,String
notifyUrl) {

        try {
            //1.创建参数
            Map<String,String> param=new HashMap();//创建参数
            param.put("appid", config.getAppID());//公众号
            param.put("mch_id", config.getMchID());//商户号
            param.put("nonce_str", WXPayUtil.generateNonceStr());//随机字
符串
            param.put("body", "青橙");//商品描述
            param.put("out_trade_no", orderId);//商户订单号
            param.put("total_fee",money+"");//总金额（分）
            param.put("spbill_create_ip", "127.0.0.1");//IP
            param.put("notify_url", notifyUrl );//暂时随便写一个
            param.put("trade_type", "NATIVE");//交易类型
            String xmlParam = WXPayUtil.generateSignedXml(param,
partnerkey);
            //2.发送请求
            String result =
wxPayRequest.requestWithCert("/pay/unifiedorder", null, xmlParam, false);
            //3.封装结果
            System.out.println(result);
            Map<String, String> resultMap = WXPayUtil.xmlToMap(result);
            Map<String, String> map=new HashMap();
            map.put("code_url", resultMap.get("code_url"));//支付地址
            map.put("total_fee", money+"");//总金额
            map.put("out_trade_no",orderId);//订单号
            return map;
        } catch (Exception e) {
            e.printStackTrace();
            return new HashMap();
        }

    }
```

```
}
```

（5）qingcheng_web_portal新增PayController

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

    @Reference
    private OrderService orderService;

    @Reference
    private WxPayService wxPayService;

    @GetMapping("/createNative")
    public Map createNative(String orderId  ){
        String username =
SecurityContextHolder.getContext().getAuthentication().getName();
        Order order = orderService.findById(orderId);
        if(order!=null){
            if("0".equals(order.getPayStatus())  &&
"0".equals(order.getOrderStatus()) &&
username.equals(order.getUsername()) ){
                return
wxPayService.createNative(orderId,order.getPayMoney(),"http://qingcheng.e
asy.echosite.cn/wxpay/notify.do");
            }else{
                return null;
            }
        }else{
            return null;
        }
    }

    /**
     * 回调
     */
    @RequestMapping("/notify")
    public void notifyLogic(){
        System.out.println("支付成功回调。。。。");
    }
}
```

2.3.2 前端代码

（1）qingcheng_web_portal新增weixinpay.html 、paysuccess.html、payfail.html（资源中提供）

（2）将二维码插件 qrcode.min.js 拷贝到qingcheng_web_portal下的js目录

（3）修改weixinpay.html

添加js代码

```
<script src="/js/vue.js"></script>
<script src="/js/axios.js"></script>
<script src="/js/qrcode.min.js"></script>
<script src="/js/util.js"></script>
<script>
    new Vue({
        el:'#app',
        data(){
            return {
                orderId:"",
                money:""
            }
        },
        created(){
            this.createNative();
        },
        methods:{
            createNative(){
                let orderId = getQueryString("orderId");//订单编号
                axios.get(`/wxpay/createNative.do?
orderId=${orderId}`).then( response=>{
                    if( response.data.out_trade_no!=null ){
                        let qrcode = new
QRCode(document.getElementById("qrcode"), {
                            width : 200,
                            height : 200
                        });
                        qrcode.makeCode(response.data.code_url); //生成支
付二维码
                        this.orderId= response.data.out_trade_no;
                        this.money=  response.data.total_fee;
                    }else{
                        location.href="payfail.html";//失败页面
                    }
                } )
            }
        }
    });


</script>
```

显示订单号和金额

```
<div class="checkout-tit" id="app">
    <h4 class="fl tit-txt"><span class="success-icon"></span><span
class="success-info">订单提交成功，请您及时付款！订单号：{{orderId}}</span>
</h4>
    <span class="fr"><em class="sui-lead">应付金额：</em><em
class="orange money">￥{{ (money/100).toFixed(2) }}</em>元</span>
    <div class="clearfix"></div>
</div>
```

修改页面二维码部分

```
<div id="qrcode"></div>
```

（4）修改pay.html，完成到微信支付页面的跳转

```
            wxPay(){
                location.href='weixinpay.html?orderId='+this.ordersn;
            }
```

调用

```
<li @click="wxPay()"><img src="./img/_/pay3.jpg"></li>
```

2.3.3 修改配置文件并测试

修改qingcheng_web_portal的cas.properties

```
cas_url=http://localhost:8080/cas
service_url=http://qingcheng.easy.echosite.cn
```

修改spring-security-portal.xml，添加配置

```
<http pattern="/wxpay/notify.do" security="none"></http>
```

3. 青橙-支付回调逻辑处理

3.1 需求分析

在完成支付后，修改订单状态为已支付，并记录订单日志。

3.2 实现思路

（1）接受微信支付平台的回调信息（xml）

```
<xml><appid><![CDATA[wx8397f8696b538317]]></appid>
<bank_type><![CDATA[CFT]]></bank_type>
<cash_fee><![CDATA[1]]></cash_fee>
<fee_type><![CDATA[CNY]]></fee_type>
<is_subscribe><![CDATA[N]]></is_subscribe>
<mch_id><![CDATA[1473426802]]></mch_id>
<nonce_str><![CDATA[c6bea293399a40e0a873df51e667f45a]]></nonce_str>
<openid><![CDATA[oNpSGwbtNBQROpN_dL8WUZG3wRkM]]></openid>
<out_trade_no><![CDATA[1553063775279]]></out_trade_no>
<result_code><![CDATA[SUCCESS]]></result_code>
<return_code><![CDATA[SUCCESS]]></return_code>
<sign><![CDATA[DD4E5DF5AF8D8D8061B0B8BF210127DE]]></sign>
<time_end><![CDATA[20190320143646]]></time_end>
<total_fee>1</total_fee>
<trade_type><![CDATA[NATIVE]]></trade_type>
<transaction_id><![CDATA[4200000248201903206581106357]]></transaction_id>
</xml>
```

（2）通过签名验证内容的合法性

（3）如果支付结果为成功，则调用修改订单状态和记录订单日志的方法。

3.3 代码实现

3.3.1 接收回调信息

微信支付平台发送给回调地址的是二进制流，我们需要提取二进制流转换为字符串，这个字符串就是xml格式。

修改notifyLogic方法

```java
/**
 * 通知
 * @return
 */
@RequestMapping("/notify")
public Map notifyLogic(HttpServletRequest request){
    System.out.println("回调了.....");
    InputStream inStream;
    try {
        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();
        String result = new String(outSteam.toByteArray(), "utf-8");
        System.out.println(result);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return new HashMap();
}
```

测试后，在控制台看到输出的消息

```
<xml><appid><![CDATA[wx8397f8696b538317]]></appid>
<bank_type><![CDATA[CFT]]></bank_type>
<cash_fee><![CDATA[1]]></cash_fee>
<fee_type><![CDATA[CNY]]></fee_type>
<is_subscribe><![CDATA[N]]></is_subscribe>
<mch_id><![CDATA[1473426802]]></mch_id>
<nonce_str><![CDATA[c6bea293399a40e0a873df51e667f45a]]></nonce_str>
<openid><![CDATA[oNpSGwbtNBQROpN_dL8WUZG3wRkM]]></openid>
<out_trade_no><![CDATA[1553063775279]]></out_trade_no>
<result_code><![CDATA[SUCCESS]]></result_code>
<return_code><![CDATA[SUCCESS]]></return_code>
<sign><![CDATA[DD4E5DF5AF8D8D8061B0B8BF210127DE]]></sign>
<time_end><![CDATA[20190320143646]]></time_end>
<total_fee>1</total_fee>
<trade_type><![CDATA[NATIVE]]></trade_type>
<transaction_id><![CDATA[4200000248201903206581106357]]></transaction_id>
</xml>
```

我们可以将此xml字符串，转换为map，提取其中的out_trade_no（订单号），根据订单号修改订单状态。

3.3.2 内容解析与签名验证

（1）WxPayService新增方法定义

```
    /**
     * 微信支付回调
     * @param xml
     */
    public void notifyLogic(String xml);
```

（2）WxPayServiceImpl实现此方法

```
    @Override
    public void notifyLogic(String xml) {

        try {
            //1.对xml进行解析 map
            Map<String, String> map = WXPayUtil.xmlToMap(xml);
            //2.验证签名
            boolean signatureValid = WXPayUtil.isSignatureValid(map,
config.getKey());

            System.out.println("验证签名是否正确: "+signatureValid);
            System.out.println(map.get("out_trade_no"));
            System.out.println(map.get("result_code"));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

3.3.3 修改订单状态

（1）OrderService接口新增方法定义

```
    /**
     * 修改订单状态
     * @param orderId
     * @param transactionId
     */
    public void updatePayStatus(String orderId,String transactionId);
```

（2）OrderServiceImpl新增方法实现

```
@Override
@Transactional
public void updatePayStatus(String orderId, String transactionId) {
    Order order = orderMapper.selectByPrimaryKey(orderId);
    if(order!=null  && "0".equals(order.getPayStatus())){  //存在订单且状态为0
        order.setPayStatus("1");
        order.setOrderStatus("1");
        order.setUpdateTime(new Date());
        order.setPayTime(new Date());
        order.setTransactionId(transactionId);//微信返回的交易流水号
        orderMapper.updateByPrimaryKeySelective(order);
        //记录订单变动日志
        OrderLog orderLog=new OrderLog();
        orderLog.setOperater("system");// 系统
        orderLog.setOperateTime(new Date());//当前日期
        orderLog.setOrderStatus("1");
        orderLog.setPayStatus("1");
        orderLog.setRemarks("支付流水号"+transactionId);
        orderLog.setOrderId(order.getId());
        orderLogMapper.insert(orderLog);
    }
}
```

（3）在WxPayServiceImpl的notifyLogic方法调用updatePayStatus方法

```
orderService.updatePayStatus(map.get("out_trade_no"),map.get("transaction_id"));//更新订单状态
```

4. 青橙-推送支付通知

4.1 需求分析

当用户完成扫码支付后，跳转到支付成功页面

4.2 服务端推送方案

我们需要将支付的结果通知前端页面，其实就是我们通过所说的服务器端推送，主要有三种实现方案

（1）Ajax 短轮询
Ajax 轮询主要通过页面端的 JS 定时异步刷新任务来实现数据的加载

如果我们使用ajax短轮询方式，需要后端提供方法，通过调用微信支付接口实现根据订单号查询支付状态的方法（参见查询订单API） 。 前端每间隔三秒查询一次，如果后端返回支付成功则执行页面跳转。

缺点：这种方式实时效果较差，而且对服务端的压力也较大。

（2）长轮询

长轮询主要也是通过 Ajax 机制，但区别于传统的 Ajax 应用，长轮询的服务器端会在没有数据时阻塞请求直到有新的数据产生或者请求超时才返回，之后客户端再重新建立连接获取数据。

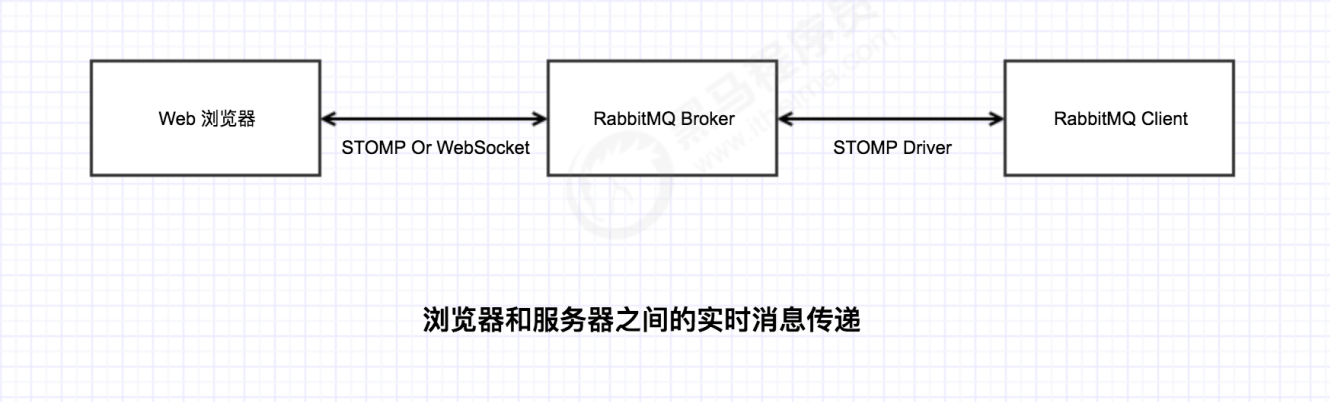
如果使用长轮询，也同样需要后端提供方法，通过调用微信支付接口实现根据订单号查询支付状态的方法，只不过循环是写在后端的。

缺点：长轮询服务端会长时间地占用资源，如果消息频繁发送的话会给服务端带来较大的压力。

（3）WebSocket 双向通信
WebSocket 是 HTML5 中一种新的通信协议，能够实现浏览器与服务器之间全双工通信。如果浏览器和服务端都支持 WebSocket 协议的话，该方式实现的消息推送无疑是最高效、简洁的。并且最新版本的 IE、Firefox、Chrome 等浏览器都已经支持 WebSocket 协议，Apache Tomcat 7.0.27 以后的版本也开始支持 WebSocket。

4.3 RabbitMQ Web STOMP 插件

借助于 RabbitMQ 的 Web STOMP 插件，实现浏览器与服务端的全双工通信。从本质上说，RabbitMQ 的 Web STOMP 插件也是利用 WebSocket 对 STOMP 协议进行了一次桥接，从而实现浏览器与服务端的双向通信。



4.3.1 STOMP协议

STOMP即Simple (or Streaming) Text Orientated Messaging Protocol，简单(流)文本定向消息协议。前身是TTMP协议（一个简单的基于文本的协议），专为消息中间件设计。它提供了一个可互操作的连接格式，允许STOMP客户端与任意STOMP消息代理（Broker）进行交互。STOMP协议由于设计简单，易于开发客户端，因此在多种语言和多种平台上得到广泛地应用。

4.3.2 插件安装

我们进入rabbitmq的sbin目录，执行下面的命令开启stomp插件

```
rabbitmq-plugins enable rabbitmq_web_stomp rabbitmq_web_stomp_examples
```

安装后重新启动rabbitmq

4.3.3 消息推送测试

我们在浏览器访问http://localhost:15670 可以看到stomp的例子代码 。

将stomp.min.js拷贝到web_portal工程（在资源中提供）

我们根据stomp的例子代码创建一个页面，内容如下：

```
<html>
<head>
    <title>RabbitMQ Web STOMP Examples : Echo Server</title>
    <meta charset="UTF-8">
    <script src="js/stomp.min.js"></script>
</head>
<script>
    var client = Stomp.client('ws://localhost:15674/ws');
    var on_connect = function(x) {
        id = client.subscribe("/exchange/paynotify", function(d) {
            alert(d.body);
        });
    };
    var on_error =  function() {
        console.log('error');
    };
    client.connect('guest', 'guest', on_connect, on_error, '/');
</script>
</body>
</html>
```

> destination 在 RabbitMQ Web STOM 中进行了相关的定义，根据使用场景的不同，主要有以下 4 种：
>
> - 1./exchange/
>
> 对于 SUBCRIBE frame，destination 一般为/exchange//[/pattern] 的形式。该 destination 会创建一个唯一的、自动删除的、名为的 queue，并根据 pattern 将该 queue 绑定到所给的 exchange，实现对该队列的消息订阅。
> 对于 SEND frame，destination 一般为/exchange//[/routingKey] 的形式。这种情况下消息就会被发送到定义的 exchange 中，并且指定了 routingKey。
>
> - 2./queue/
>   对于 SUBCRIBE frame，destination 会定义的共享 queue，并且实现对该队列的消息订阅。
>   对于 SEND frame，destination 只会在第一次发送消息的时候会定义的共享 queue。该消息会被发送到默认的 exchange 中，routingKey 即为。
> - 3./amq/queue/
>   这种情况下无论是 SUBCRIBE frame 还是 SEND frame 都不会产生 queue。但如果该 queue 不存在，SUBCRIBE frame 会报错。

> 对于 SUBCRIBE frame，destination 会实现对队列的消息订阅。
> 对于 SEND frame，消息会通过默认的 exhcange 直接被发送到队列中。
- 4./topic/
  对于 SUBCRIBE frame，destination 创建出自动删除的、非持久的 queue 并根据 routingkey 为绑定到 amq.topic exchange 上，同时实现对该 queue 的订阅。
  对于 SEND frame，消息会被发送到 amq.topic exchange 中，routingKey 为。

我们在rabbitmq中创建一个叫paynotify的交换机（fanout类型）

测试，我们在rabbitmq管理界面中向paynotify交换机发送消息，页面就会接收这个消息。

为了安全，我们在页面上不能用我们的rabbitmq的超级管理员用户guest，所以我们需要在rabbitmq中新建一个普通用户webguest（普通用户无法登录管理后台）

4.4 代码实现

实现思路：后端在收到回调通知后发送订单号给mq（paynotify交换器），前端通过stomp连接到mq订阅paynotify交换器的消息，判断接收的订单号是不是当前页面的订单号，如果是则进行页面的跳转。

4.4.1 后端代码

修改WxPayServiceImpl，引入rabbitmq

```
@Autowired
private RabbitTemplate rabbitTemplate;
```

修改notifyLogic方法，在`"SUCCESS".equals(map.get("result_code"))`后添加

```
rabbitTemplate.convertAndSend("paynotify","",map.get("out_trade_no"));
```

4.4.2 前端代码

（1）修改weixinpay.html，引入stomp.min.js (资源提供)

```
<script src="/js/stomp.min.js"></script>
```

添加js代码

```
    let money=0;//金额
    let client = Stomp.client('ws://localhost:15674/ws');
    let on_connect = function(x) {
        id = client.subscribe("/exchange/pay", function(d) {
            if(d.body==getQueryString("orderId")){
                location.href='paysuccess.html?money='+money;
            }
        });
    };
    let on_error =  function() {
        console.log('error');
    };
    client.connect('webguest', 'itcast', on_connect, on_error, '/');
```

在createNative方法的回调处添加代码

```
money=response.data.total_fee;
```

（2）修改paysuccess.html，添加代码

```
<script src="/js/vue.js"></script>
<script src="/js/util.js"></script>
<script>
    new Vue({
        el:'#app',
        data(){
            return {
                money:0
            }
        },
        created(){
            this.money= getQueryString("money");
        }

    });

</script>
```

绑定表达式，略

5. 青橙-超时未支付订单处理

5.1 需求分析

超过60分钟未支付的订单，我们需要进行超时订单的处理：先调用微信支付api，查询该订单的支付状态。如果未支付调用关闭订单的api，并修改订单状态为已关闭，并回滚库存数。如果该订单已经支付，做补偿操作（修改订单状态和记录）。

5.2 实现思路

如何获取超过60分钟的订单？我们目前有两种实现方案

（1）定时任务轮询方案

编写定时任务，查询所有60分钟前创建的订单列表。

循环此订单列表，查询每个订单的支付状态。如果已支付进行状态补偿，如果未支付则关闭订单。

这种实现方案缺点是时间精度不高，对系统压力比较大。

（2）使用延迟消息队列

所谓延迟消息队列，就是消息的生产者发送的消息并不会立刻被消费，而是在设定的时间之后才可以消费。

我们可以在订单创建时发送一个延迟消息，消息为订单号，青橙系统会在60分钟后取出这个消息，然后查询订单的支付状态，根据结果做出相应的处理。

我们在青橙中采用延迟消息队列的实现方案。

另外我们还会用到微信支付api中的查询订单和关闭订单。

5.3 rabbitmq延迟消息

使用RabbitMQ来实现延迟消息必须先了解RabbitMQ的两个概念：消息的TTL和死信Exchange，通过这两者的组合来实现上述需求。

5.3.1 消息的TTL（Time To Live）

消息的TTL就是消息的存活时间。RabbitMQ可以对队列和消息分别设置TTL。对队列设置就是队列没有消费者连着的保留时间，也可以对每一个单独的消息做单独的设置。超过了这个时间，我们认为这个消息就死了，称之为死信。

我们创建一个队列queue.temp，在Arguments 中添加x-message-ttl 为5000 （单位是毫秒），那所在压在这个队列的消息在5秒后会消失。

5.3.2 死信交换器 Dead Letter Exchanges

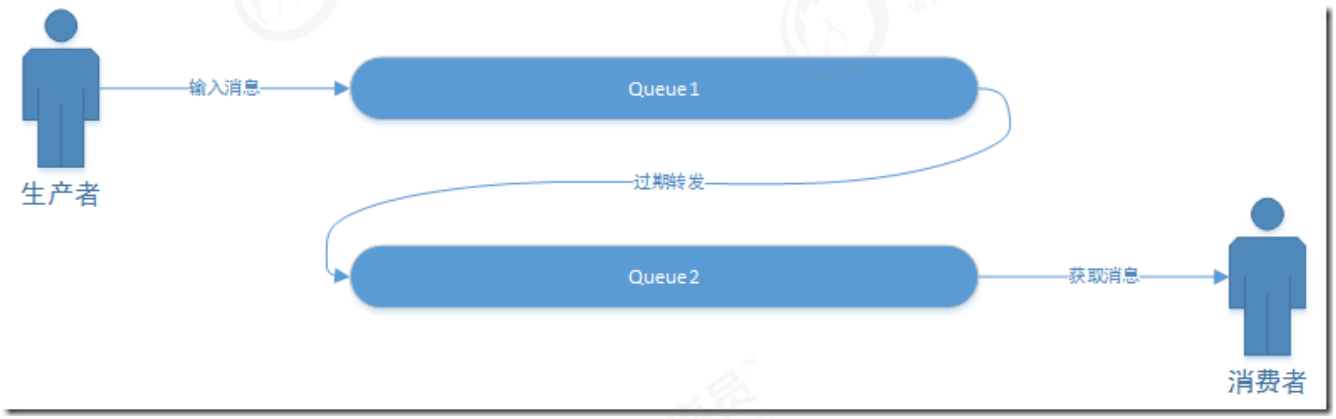
一个消息在满足如下条件下，会进死信路由，记住这里是路由而不是队列，一个路由可以对应很多队列。

（1） 一个消息被Consumer拒收了，并且reject方法的参数里requeue是false。也就是说不会被再次放在队列里，被其他消费者使用。

（2）上面的消息的**TTL**到了，消息过期了。

（3）队列的长度限制满了。排在前面的消息会被丢弃或者扔到死信路由上。

Dead Letter Exchange其实就是一种普通的exchange，和创建其他exchange没有两样。只是在某一个设置Dead Letter Exchange的队列中有消息过期了，会自动触发消息的转发，发送到Dead Letter Exchange中去。



我们现在可以测试一下延迟队列。

（1）创建死信交换器 exchange.ordertimeout （fanout）

（2）创建队列queue.ordertimeout

（3）建立死信交换器 exchange.ordertimeout 与队列exchange.ordertimeout 之间的绑定 routingkey为 exchange.ordertimeout.key

（4）创建队列queue.order，Arguments添加

x-message-ttl=5000

x-dead-letter-exchange： exchange.ordertimeout

x-dead-letter-routing-key： exchange.ordertimeout.key

5.4 代码实现（作业）

（1）创建队列queue.ordercreate，Arguments添加

x-message-ttl=3600000

x-dead-letter-exchange： exchange.ordertimeout

x-dead-letter-routing-key： exchange.ordertimeout.key

（2）在创建订单的方法中添加代码，发送消息到queue.ordercreate ，消息内容为订单号

（3）在WxPayServiceImpl中编写业务逻辑方法，调用微信支付api中的查询订单api，实现根据订单号查询支付结果。

（4）在WxPayServiceImpl中编写业务逻辑方法，调用微信支付api中的关闭订单api，实现根据订单号关闭微信订单。

（5）在OrderServiceImpl中编写关闭订单的业务逻辑方法，逻辑为调用微信支付关闭订单的方法、修改订单表的订单状态、记录订单日志、恢复商品表库存。

（6）在订单服务中提取queue.ordertimeout 队列 中的消息（订单号），先查询业务系统的订单状态，如果订单状态为未支付，调用微信支付查询订单的方法查询。如果返回结果是未支付，调用关闭订单的业务逻辑方法。 如果返回结果是已支付，实现补偿操作。