

# 第13章 微信扫码支付

## 课程内容：

- 能够根据微信支付的开发文档调用微信支付的api
- 完成统一下单生成微信支付二维码功能
- 完成支付回调的逻辑处理，掌握EchoSite的使用
- 完成推送支付通知功能

## 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：

我们解压从官网下载的sdk,安装到本地仓库

com.github.wxpay.sdk.WXPay类下提供了对应的方法：

方法名	说明
microPay	刷卡支付
unifiedOrder	统一下单
orderQuery	查询订单
reverse	撤销订单
closeOrder	关闭订单
refund	申请退款
refundQuery	查询退款
downloadBill	下载对账单
report	交易保障
shortUrl	转换短链接
authCodeToOpenid	授权码查询openid

## 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com.github.wxpay.sdk包，包下创建MyConfig类，继承自抽象类WXPayConfig

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

    String getMchID() {
        return "1473426802";
    }

    String getKey() {
        return "T6m9iK73b0kn9g5v426MKfHQH7X8rKwb";
    }
}
```



```

    }

    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

```

MyConfig config=new MyConfig();
WXPay wxPay=new WXPay( config );

Map<String,String> map=new HashMap();
map.put("body", "畅购");//商品描述
map.put("out_trade_no", "55555211");//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

Map<String, String> result = wxPay.unifiedOrder( map );
System.out.println(result);

```

执行后返回结果

```

{nonce_str=fvMGIlLauUPNctws, code_url=weixin://wmpay/bizpayurl?pr=I5sd2rc,
appid=wx8397f8696b538317,
sign=48B2938F70EDADC9CC235249BC085FD1D83456F67C46601FFD23B5AFBDA502D0,
trade_type=NATIVE, return_msg=OK, result_code=SUCCESS, mch_id=1473426802,
return_code=SUCCESS, prepay_id=wx17193859685440d561c4cef01259098400}

```

其中的code\_url就是我们的支付URI，我们可以根据这个URI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

我们看一下静态原型wmpay.html中的代码,显示二维码的地方放置 `<div id='qrcode'></div>`, 然后编写脚本

```
<script type="text/javascript">
let qrcode = new QRCode(document.getElementById("qrcode"), {
  width : 200,
  height : 200
});
qrcode.makeCode("weixin://wxpay/bizpayurl?pr=Y3hDTzy");
</script>
```

## 2. 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。



收银台

订单提交成功，请您及时付款！订单号：56789065645

应付金额：¥17,654元



### 2.2 实现思路

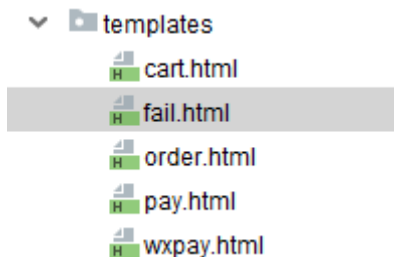
前端页面向后端传递订单号，后端根据订单号查询订单，检查是否为当前用户的未支付订单，如果是则根据订单号和金额生成支付url返给前端，前端得到支付url生成支付二维码。

### 2.3 代码实现

#### 2.3.1 提交订单跳转支付页

1) 更新changgou\_web\_order的application.yml,添加读取超时设置

```
#请求处理的超时时间
ribbon:
  ReadTimeout: 4000
#请求连接的超时时间
ConnectTimeout: 3000
```



3) 更新changgou\_service\_order中add(),设置返回值为订单Id

```
#orderServiceImpl
/**
 * 增加
 * @param order
 */
@Override
public String add(Order order){
    List<OrderItem> cartList =
    cartService.findCartList(order.getUsername());

    int totalMoney=0;
    int totalNum=0;
    int totalPayMoney=0;
    for (OrderItem orderItem : cartList) {
        totalMoney+=orderItem.getMoney();
        totalNum+=orderItem.getNum();
        totalPayMoney+=orderItem.getPayMoney();
    }
    order.setTotalNum(totalNum);
    order.setTotalMoney(totalMoney);
    order.setPayMoney(totalPayMoney);
    order.setPreMoney(totalMoney-totalPayMoney);

    //其他数据完善
    order.setCreateTime(new Date());
    order.setUpdateTime(order.getCreateTime());
    order.setBuyerRate("0"); //0:未评价, 1: 已评价
    order.setSourceType("1"); //来源, 1: WEB
    order.setOrderStatus("0"); //0:未完成, 1:已完成, 2: 已退货
    order.setPayStatus("0"); //0:未支付, 1: 已支付, 2: 支付失败
    order.setConsignStatus("0"); //0:未发货, 1: 已发货, 2: 已收货
    String orderId = idworker.nextId() + "";
    order.setId(orderId);
    int result = orderMapper.insertSelective(order);

    for (OrderItem orderItem : cartList) {
        orderItem.setId(idworker.nextId()+"");
        orderItem.setIsReturn("0");
        orderItem.setOrderId(order.getId());
        orderItemMapper.insertSelective(orderItem);
    }

    //扣减库存
```



```
//清除缓存中的数据
redisTemplate.delete("Cart_"+order.getUsername());

return orderId;
}

#orderController
@PostMapping
public Result<String> add(@RequestBody Order order){
    String username = tokenDecode.getUserInfo().get("username");
    order.setUsername(username);
    String orderId = orderService.add(order);
    return new Result(true, StatusCode.OK, "添加成功", orderId);
}
```

#### 4) 更新order.html中添加订单js

```
add:function () {
    axios.post('/api/worder/add',this.order).then(function
(response) {
        if (response.data.flag){
            alert("添加订单成功");
            var orderId = response.data.data;
            location.href="/api/worder/toPayPage?orderId="+orderId;
        }else{
            alert("添加失败");
        }
    })
}
```

#### 5) 在orderController中新增方法，用于跳转支付页

##### 5.1) changgou\_service\_order\_api的OrderFeign新增接口定义

```
/**
 * 根据ID查询数据
 * @param id
 * @return
 */
@GetMapping("/order/{id}")
public Result findById(@PathVariable("id") String id);
```

##### 5.2) changgou\_order\_web中的orderController新增方法，跳转支付页



```
public String payByFeign(String orderId, Model model) {
    Result<Order> orderResult = orderFeign.findById(orderId);
    Order order = orderResult.getData();
    model.addAttribute("orderId", orderId);
    model.addAttribute("payMoney", order.getPayMoney());
    return "pay";
}
```

### 2.3.2 支付微服务-下单

(1) 创建changgou\_service\_pay（支付微服务），pom.xml添加依赖

```
<dependency>
  <groupId>com.changgou</groupId>
  <artifactId>changgou_common</artifactId>
  <version>1.0-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>com.github.wxpay</groupId>
  <artifactId>wxpay-sdk</artifactId>
  <version>3.0.9</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
  <exclusions>
    <exclusion>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-logging</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

排除log包，否则会因为包冲突无法正常启动

(2) 创建配置文件application.yml

```
server:
  port: 9010
spring:
  application:
    name: pay
  rabbitmq:
    host: 192.168.200.128
  main:
    allow-bean-definition-overriding: true #当遇到同样名字的时候，是否允许覆盖注册
eureka:
  client:
    service-url:
```

```
prefer-ip-address: true
```

(3) 创建com.github.wxpay.sdk包，包下创建Config类

```
public class MyConfig 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);
            }
        };
    }
}
```

(4) 创建com.changgou包，包下创建类PayApplication

```
@SpringBootApplication
@EnableEurekaClient
public class PayApplication {

    public static void main(String[] args) {
        SpringApplication.run(PayApplication.class);
    }

    @Bean
    public WXPay wxPay(){
        try {
            return new WXPay( new MyConfig() );
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}
```





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

    @Autowired
    private WxPayService wxPayService;

    /**
     * 下单
     * @param orderId
     * @param money
     * @return
     */
    @GetMapping("/nativePay")
    public Result nativePay(@RequestParam("orderId")String
orderId,@RequestParam("money") Integer money){
        Map map = wxPayService.nativePay( orderId, money );
        return new Result( true, StatusCode.OK,"",map );
    }
}
```

( 4 ) 创建com.changgou.pay.service包，包下创建接口WxPayService

```
public interface WxPayService {

    /**
     * 生成微信支付二维码
     * @param orderId
     * @param money
     * @return
     */
    Map nativePay(String orderId, Integer money);
}
```

( 5 ) 创建com.changgou.pay.service.impl 包,新增服务类WxPayServiceImpl

```
@Service
public class WxPayServiceImpl implements WxPayService {

    @Autowired
    private WXPay wxPay;

    @Override
    public Map nativePay(String orderId, Integer money) {
        try {
            //1.封装请求参数
            Map<String,String> map=new HashMap();
            map.put("body","畅购商城");//商品描述
            map.put("out_trade_no",orderId);//订单号
            //map.put("total_fee",String.valueOf(money*100)); //金额,以分为单位
            BigDecimal payMoney = new BigDecimal("0.01");
            BigDecimal fen = payMoney.multiply(new BigDecimal("100")); //1.00
            fen = fen.setScale(0,BigDecimal.ROUND_UP); // 1
            map.put("total_fee",String.valueOf(fen));
        }
    }
}
```



```

        map.put("notify_url", "http://www.itcast.cn"); //回调地址,先随便填一个
        map.put("trade_type", "NATIVE"); //交易类型
        Map<String, String> mapResult = wxPay.unifiedOrder( map ); //调用统一
下单
        return mapResult;
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
}

```

测试：地址栏输入<http://localhost:9010/wxpay/nativePay?orderId=990099&money=1>

### 2.3.3 页面对接支付微服务

(1) 新增changgou\_service\_pay\_api模块，并添加common工程依赖，新增com.changgou.pay.feign包，包下创建接口

```

@FeignClient("pay")
public interface WxPayFeign {

    /**
     * 下单
     * @param orderId
     * @param money
     * @return
     */
    @GetMapping("/wxpay/nativePay")
    public Result nativePay(@RequestParam("orderId") String orderId,
                           @RequestParam("money") Integer money );
}

```

(2) changgou\_web\_order新增PayController

```

@Controller
@RequestMapping("/wxpay")
public class PayController {

    @Autowired
    private OrderFeign orderFeign;

    @Autowired
    private WxPayFeign wxPayFeign;

    /**
     * 微信支付二维码
     * @param orderId
     * @return
     */
    @GetMapping

```



```

        Result orderResult = orderFeign.findById( orderId );
        if(orderResult.getData()==null){ //如果订单不存在
            return "fail";//出错页
        }
        Order order = orderResult.getData();
        //判断支付状态
        if( !"0".equals( order.getPayStatus() ) ){// 如果不是未支付订单
            return "fail";//出错页
        }
        Result payResult = wxPayFeign.nativePay( orderId, order.getPayMoney() );
        if(payResult.getData()==null){
            return "fail";//出错页
        }
        Map payMap= (Map)payResult.getData();
        payMap.put( "payMoney", order.getPayMoney() );
        payMap.put( "orderId", orderId );
        model.addAllAttributes( payMap );
        return "wxpay";
    }
}

```

(3) 将静态原型中wxpay.html拷贝到templates文件夹下作为模板，修改模板，部分代码如下：

二维码地址渲染

```

<script type="text/javascript" th:inline="javascript">
let qrcode = new QRCode(document.getElementById("qrcode"), {
    width : 200,
    height : 200
});
qrcode.makeCode([[${code_url}]]);
</script>

```

显示订单号与金额

```

<h4 class="fl tit-txt"><span class="success-icon"></span><span class="success-
info" th:text="| 订单提交成功，请您及时付款！ 订单号: ${orderId}|"></span></h4>
<span class="fr"><em class="sui-lead">应付金额: </em><em class="orange money"
th:text="${payMoney}/100"></em>元</span>

```

设置支付跳转

```

<li><a th:href="/api/wxpay?orderId=${orderId}|"></a>
</li>

```

(4) 更新网关地址过滤器。添加wxpay路径的拦截

```

public class UrlFilter {

    public static String orderFilterPath = "/api/wxpay,/api/wxpay/**,/api/worder

```

同时更新网关服务的application.yml。添加地址路由标识

北京市昌平区建材城西路金燕龙办公楼一层 电话：400-618-9090



```
uri: lb://order-web
predicates:
  - Path=/api/wcart/**,/api/worder/**,/api/wxpay/**
filters:
  - StripPrefix=1
```

## 3. 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) 收到通知后，调用查询接口查询订单。

(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 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3元买一个域名（可以用1个月），点击域名端口---抢注域名

### 选择需要的协议类型

• <http://www.pearsoncmg.com>

https

## 选择服务器

Easy - 共享型

easy.echosite.cn

2核4GB 4M带宽

3元/月

•

Cross - 共享型

[cross.echosite.cn](http://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

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

changgou

cross.echosite.cn

### 选择抢占时间

1个月

○

支付金额: 3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: cross.echosite.cn:4443
trust_host_root_certs: false
```



```
# 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: "changgou"
    proto:
      http: 127.0.0.1:9010
```

然后在echosite目录中输入以下命令

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

运行效果如下：

```
选择C:\Windows\System32\cmd.exe - echosite.exe -config=config.yml start-all
EchoSite      https://www.echosite.cn (Ctrl+C to quit)
Tunnel Status  online
Version        2.0/2.0
Forwarding     http://changgou.cross.echosite.cn -> 127.0.0.1:9011
Web Interface  127.0.0.1:4040
# Conn        0
Avg Conn Time 0.00ms
```

这样你购买的域名就映射到127.0.0.1:9011上了。ctrl+c 结束程序

怎么才能验证域名是否映射到你的计算机上了呢？

WxPayController新增notifyLogic方法

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

测试：

- 1) 通过本地方式访问该接口
- 2) 通过域名形式访问该接口

### 3.3.2 接收回调信息

(1) 修改支付微服务配置文件

```
wxpay:
  notify_url: http://weizhaohui.cross.echosite.cn/wxpay/notify #回调地址
```

(2) 修改WxPayServiceImpl，引入

```
@Value( "${wxpay.notify_url}" )
private String notifyUrl;
```

```
map.put("notify_url", notifyurl); //回调地址
```

测试：重新生成订单并支付。可以发现控制台在不断的触发支付回调通知。这是为什么呢？

**注意：**

- 1、同样的通知可能会多次发送给商户系统。商户系统必须能够正确处理重复的通知。
- 2、后台通知交互时，如果微信收到商户的应答不符合规范或超时，微信会判定本次通知失败，重新发送通知，直到成功为止（在通知一直不成功的情况下，微信总共会发起10次通知，通知频率为15s/15s/30s/3m/10m/20m/30m/30m/30m/60m/3h/3h/3h/6h/6h - 总计 24h4m），但微信不保证通知最终一定能成功。

### 3.3.3 回调消息接收并转换

微信支付平台发送给回调地址的数据是二进制流，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。

（1）在changgou\_common工程添加工具类ConvertUtils。（资源提供：资源\类文件\工具类）

```
/**
 * 转换工具类
 */
public class ConvertUtils {

    /**
     * 输入流转换为xml字符串
     * @param inputStream
     * @return
     */
    public static String convertToString(InputStream inputStream) throws
IOException {
        ByteArrayOutputStream outStream = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int len = 0;
        while ((len = inputStream.read(buffer)) != -1) {
            outStream.write(buffer, 0, len);
        }
        outStream.close();
        inputStream.close();
        String result = new String(outStream.toByteArray(), "utf-8");
        return result;
    }
}
```

（2）修改notifyLogic方法

```
/**
 * 回调
 */
@RequestMapping("/notify")
```



```
System.out.println("支付成功回调。。。");
try {
    //输入流转换为xml字符串
    String xml = ConvertUtils.convertToString( request.getInputStream() );
    System.out.println(xml);

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

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

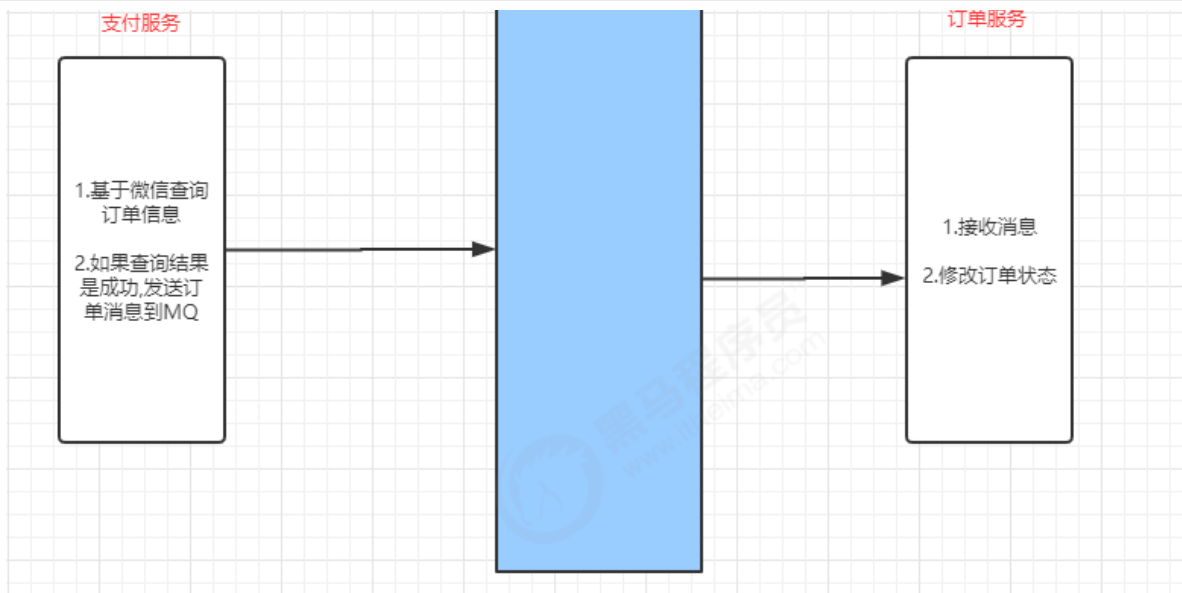
测试后，在控制台看到输出的消息

```
<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。

### 3.3.4 查询订单验证通知





### (1) WxPayService新增方法定义

```
/**
 * 查询订单
 * @param orderId
 * @return
 */
Map queryOrder(String orderId);
```

### (2) WxPayServiceImpl实现方法

```
@Override
public Map queryOrder(String orderId) {
    Map map=new HashMap( );
    map.put( "out_trade_no", orderId );
    try {
        return wxPay.orderQuery( map );
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}
```

### (3) 修改notifyLogic方法

```
@Autowired
private RabbitTemplate rabbitTemplate;

/**
 * 回调
 */
@RequestMapping("/notify")
public void notifyLogic(HttpServletRequest request, HttpServletResponse
response) throws IOException {
    System.out.println("支付成功回调。。。");
    try {
        //输入流转换为xml字符串
```



```
//解析
Map<String, String> map = wxPayUtil.xmlToMap( xml );
//查询订单
if("SUCCESS".equals(map.get( "result_code" ) )){ //如果返回的结果是成功
    Map result = wxPayService.queryOrder( map.get( "out_trade_no" ) );
    System.out.println("查询订单返回结果: "+result);

    //如果查询结果是成功发送到mq
    if("SUCCESS".equals( result.get( "result_code" ) )){
        Map m=new HashMap();
        m.put( "orderId",result.get( "out_trade_no" ) );
        m.put( "transactionId",result.get( "transaction_id" ) );
        rabbitTemplate.convertAndSend( "", "order_pay",
JSON.toJSONString(m) );

        //如果成功，给微信支付一个成功的响应
        response.setContentType("text/xml");
        String data = "<xml><return_code><![CDATA[SUCCESS]]>
</return_code><return_msg><![CDATA[OK]]></return_msg></xml>";
        response.getWriter().write(data);
    }
}else {
    System.out.println(map.get( "err_code_des" ));//错误信息描述
}
} catch (IOException e) {
    e.printStackTrace();
} catch (Exception e) {
    e.printStackTrace();
}
}
```

( 4 ) rabbitmq中添加order\_pay队列

### 3.3.5 修改订单状态

( 1 ) com.changgou.order.listener包下创建OrderPayListener

```
@Component
@RabbitListener(queues = "order_pay")
public class OrderPayListener {

    @Autowired
    private OrderService orderService;

    /**
     * 更新支付状态
     * @param message
     */
    @RabbitHandler
    public void updatePayStatus(String message){
        System.out.println("接收到消息: "+message);
        Map map = JSON.parseObject( message, Map.class );
```



```
}  
}
```

## (2) OrderService接口新增方法定义

```
/**  
 * 修改订单状态为已支付  
 * @param orderId  
 * @param transactionId  
 */  
void updatePayStatus(String orderId,String transactionId);
```

## (3) OrderServiceImpl新增方法实现

```
@Autowired  
private OrderLogMapper orderLogMapper;  
  
@Override  
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.setId( idWorker.nextId()+"");  
        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);  
    }  
}
```

# 4. 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



恭喜您，支付成功啦！

支付方式：微信

支付金额：¥1006.00元

查看订单

继续购物

## 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。**不建议使用**

(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。

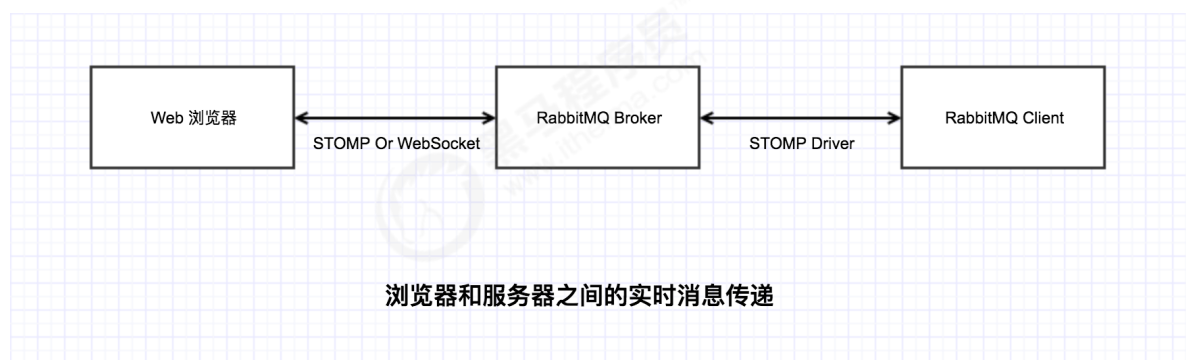
如果使用长轮询，也同样需要后端提供方法，通过调用微信支付接口实现根据订单号查询支付状态的方法，只不过循环是写在后端的。

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。**不建议使用**

(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



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容器，执行下面的命令开启stomp插件

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

将当前的容器提交为新的镜像

```
docker commit 3989ec68bf3c rabbitmq:stomp
```

停止当前的容器

```
docker stop 3989ec68bf3c
```

根据新的镜像创建容器

```
docker run -di --name=changgou_rabbitmq -p 5671:5617 -p 5672:5672 -p 4369:4369 -p 15671:15671 -p 15672:15672 -p 25672:25672 -p 15670:15670 -p 15674:15674 rabbitmq:stomp
```

### 4.3.3 消息推送测试

我们在浏览器访问<http://192.168.200.128:15670> 可以看到stomp的例子代码。

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>
```

- 1./exchange/

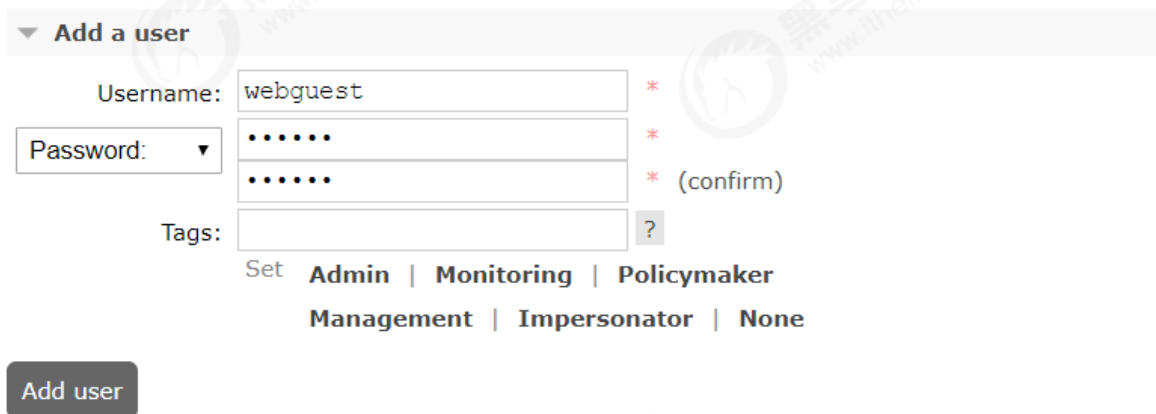
对于 SUBCRIBE frame，destination 一般为/exchange//[/pattern] 的形式。该 destination 会创建一个唯一的、自动删除的、名为的 queue，并根据 pattern 将该 queue 绑定到所给的 exchange，实现对该队列的消息订阅。对于 SEND frame，destination 一般为/exchange//[/routingKey] 的形式。这种情况下消息就会被发送到定义的 exchange 中，并且指定了 routingKey。

- 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 会实现对该队列的消息订阅。对于 SEND frame，消息会通过默认的 exchange 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）



设置虚拟目录权限



## 4.4 代码实现



转。

(1) 修改notifyLogic方法，在 "SUCCESS".equals(map.get("result\_code")) 后添加

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

(2) 修改wxpay.html，渲染js代码订单号和支付金额部分

```
let client = Stomp.client('ws://192.168.200.128:15674/ws');
let on_connect = function(x) {
    id = client.subscribe("/exchange/paynotify", function(d) {
        let orderId=[[${orderId}]];
        if(d.body==orderId){
            location.href='/api/wxpay/topaysuccess?payMoney='+[${payMoney}]];
        }
    });
};
let on_error = function() {
    console.log('error');
};
client.connect('webguest', 'itcast', on_connect, on_error, '/');
```

(3) 将paysuccess.html拷贝到static文件夹。

(4) changgou\_web\_order的PayController中新增跳转支付成功接口

```
@GetMapping("/topaysuccess")
public String topaysuccess(String payMoney,Model model){
    model.addAttribute("payMoney",payMoney);
    return "paysuccess";
}
```