一、微信支付介绍和接入指引

1、微信支付产品介绍

1.1、付款码支付

用户展示微信钱包内的“付款码”给商家，商家扫描后直接完成支付，适用于线下面对面收银的场景。

1.2、JSAPI支付

- 线下场所：商户展示一个支付二维码，用户使用微信扫描二维码后，输入需要支付的金额，完成支付。
- 公众号场景：用户在微信内进入商家公众号，打开某个页面，选择某个产品，完成支付。
- PC网站场景：在网站中展示二维码，用户使用微信扫描二维码，输入需要支付的金额，完成支付。

> 特点：用户在客户端输入支付金额

1.3、小程序支付

在微信小程序平台内实现支付的功能。

1.4、Native支付

Native支付是指商户展示支付二维码，用户再用微信“扫一扫”完成支付的模式。这种方式适用于PC网站。

> 特点：商家预先指定支付金额

1.5、APP支付

商户通过在移动端独立的APP应用程序中集成微信支付模块，完成支付。

1.6、刷脸支付

用户在刷脸设备前通过摄像头刷脸、识别身份后进行的一种支付方式。

2、接入指引

2.1、获取商户号

微信商户平台：https://pay.weixin.qq.com/

场景：Native支付

步骤：提交资料 => 签署协议 => 获取商户号

2.2、获取APPID

微信公众平台：https://mp.weixin.qq.com/

步骤：注册服务号 => 服务号认证 => 获取APPID => 绑定商户号

2.3、获取API秘钥

APIv2版本的接口需要此秘钥

步骤：登录商户平台 => 选择 账户中心 => 安全中心 => API安全 => 设置API密钥

2.4、获取APIv3秘钥

APIv3版本的接口需要此秘钥

步骤：登录商户平台 => 选择 账户中心 => 安全中心 => API安全 => 设置APIv3密钥

随机密码生成工具：https://sujijimimashengcheng.bmcx.com/

2.5、申请商户API证书

APIv3版本的所有接口都需要；APIv2版本的高级接口需要（如：退款、企业红包、企业付款等）

步骤：登录商户平台 => 选择 账户中心 => 安全中心 => API安全 => 申请API证书

2.6、获取微信平台证书

可以预先下载，也可以通过编程的方式获取。后面的课程中，我们会通过编程的方式来获取。

> 注意：以上所有API秘钥和证书需妥善保管防止泄露

二、支付安全（证书/秘钥/签名）

1、信息安全的基础 - 机密性

- **明文**：加密前的消息叫“明文”（plain text）
- **密文**：加密后的文本叫“密文”（cipher text）
- **密钥**：只有掌握特殊“钥匙”的人，才能对加密的文本进行解密，这里的“钥匙”就叫做“密钥”（key）

  > “密钥”就是一个字符串，度量单位是“位”（bit），比如，密钥长度是 128，就是 16 字节的二进制串

- **加密**：实现机密性最常用的手段是“加密”（encrypt）

  > 按照密钥的使用方式，加密可以分为两大类：***对称加密和非对称加密***。

- **解密**：使用密钥还原明文的过程叫“解密”（decrypt）
- **加密算法**：加密解密的操作过程就是“加密算法”

  > 所有的加密算法都是公开的，而算法使用的“密钥”则必须保密

2、对称加密和非对称加密

- **对称加密**
  - 特点：只使用一个密钥，密钥必须保密，常用的有 AES算法
  - 优点：运算速度快
  - 缺点：秘钥需要信息交换的双方共享，一旦被窃取，消息会被破解，无法做到安全的密钥交换
- **非对称加密**
  - 特点：使用两个密钥：公钥和私钥，公钥可以任意分发而私钥保密，常用的有 RSA
  - 优点：黑客获取公钥无法破解密文，解决了密钥交换的问题
  - 缺点：运算速度非常慢
- **混合加密**
  - 实际场景中把对称加密和非对称加密结合起来使用。

3、身份认证

- 公钥加密，私钥解密的作用是加密信息
- 私钥加密，公钥解密的作用是身份认证

4、摘要算法（Digest Algorithm）

摘要算法就是我们常说的散列函数、哈希函数（Hash Function），它能够把任意长度的数据“压缩”成固定长度、而且独一无二的“摘要”字符串，就好像是给这段数据生成了一个数字“指纹”。

作用：

保证信息的完整性

特性：

- 不可逆：只有算法，没有秘钥，只能加密，不能解密
- 难题友好性：想要破解，只能暴力枚举
- 发散性：只要对原文进行一点点改动，摘要就会发生剧烈变化
- 抗碰撞性：原文不同，计算后的摘要也要不同

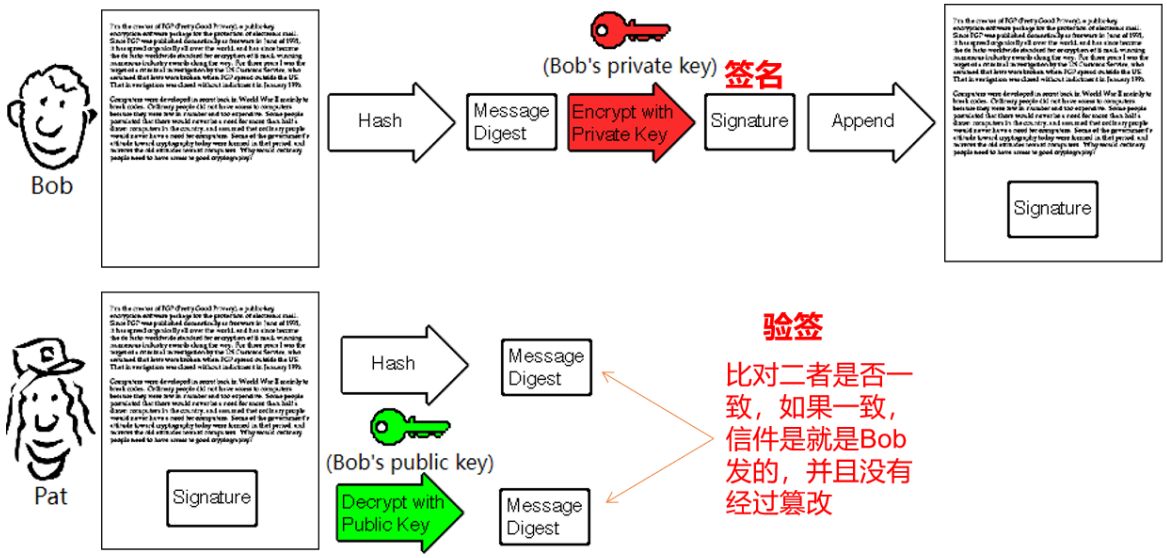
常见摘要算法：

MD5、SHA1、SHA2（SHA224、SHA256、SHA384）

5、数字签名

数字签名是使用私钥对摘要加密生成签名，需要由公钥将签名解密后进行验证，实现身份认证和不可否认

签名和验证签名的流程：

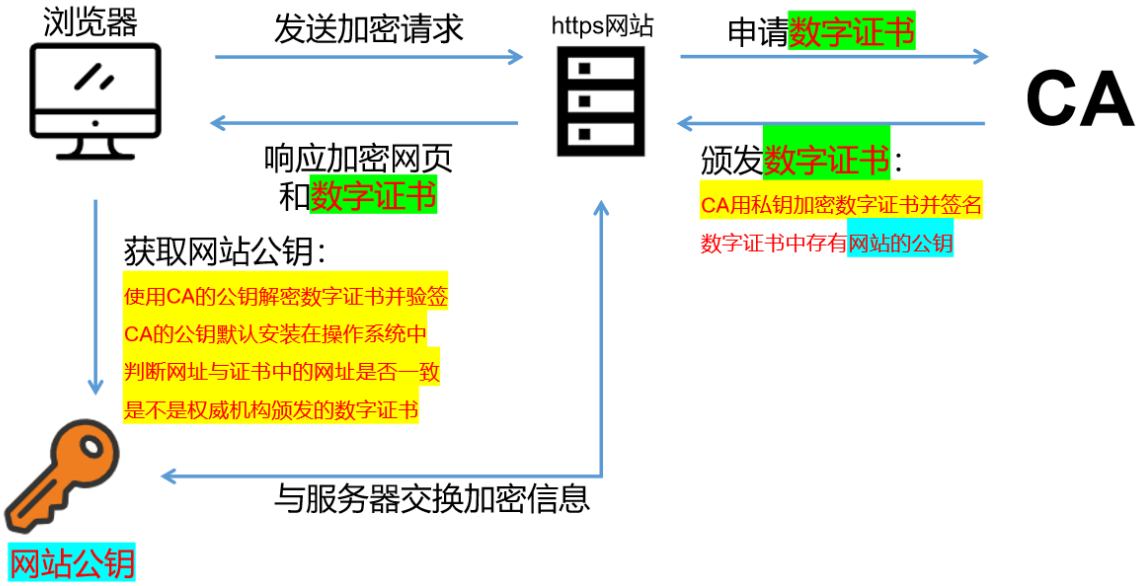


6、数字证书

数字证书解决“公钥的信任”问题，可以防止黑客伪造公钥。

不能直接分发公钥，公钥的分发必须使用数字证书，数字证书由CA颁发

https协议中的数字证书：

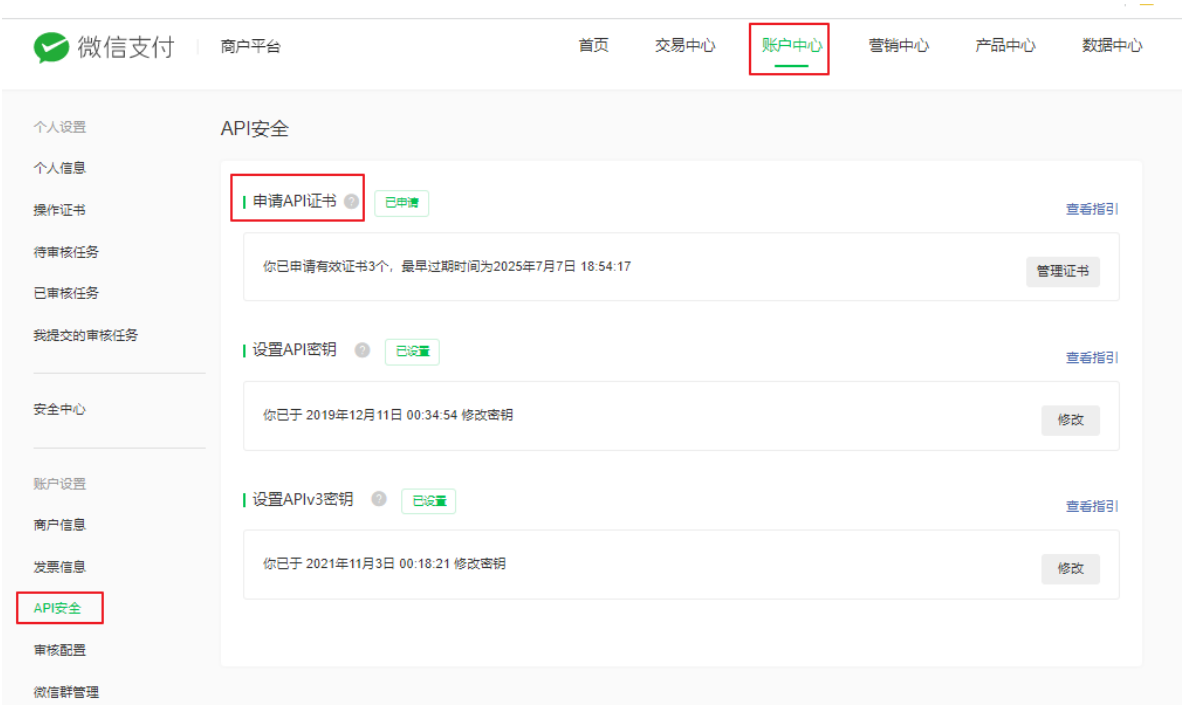


7、微信APIv3证书

商户证书：

商户API证书是指由商户申请的，包含商户的商户号、公司名称、公钥信息的证书。

商户证书在商户后台申请：https://pay.weixin.qq.com/index.php/core/cert/api_cert#/



平台证书（微信支付平台）：

微信支付平台证书是指由*微信支付*负责申请的，包含微信支付平台标识、公钥信息的证书。商户可以使用平台证书中的公钥进行验签。

平台证书的获取：https://pay.weixin.qq.com/wiki/doc/apiv3/wechatpay/wechatpay3_0.shtml

8、API密钥和APIv3密钥

都是对称加密需要使用的加密和解密密钥，一定要保管好，不能泄露。

API密钥对应V2版本的API

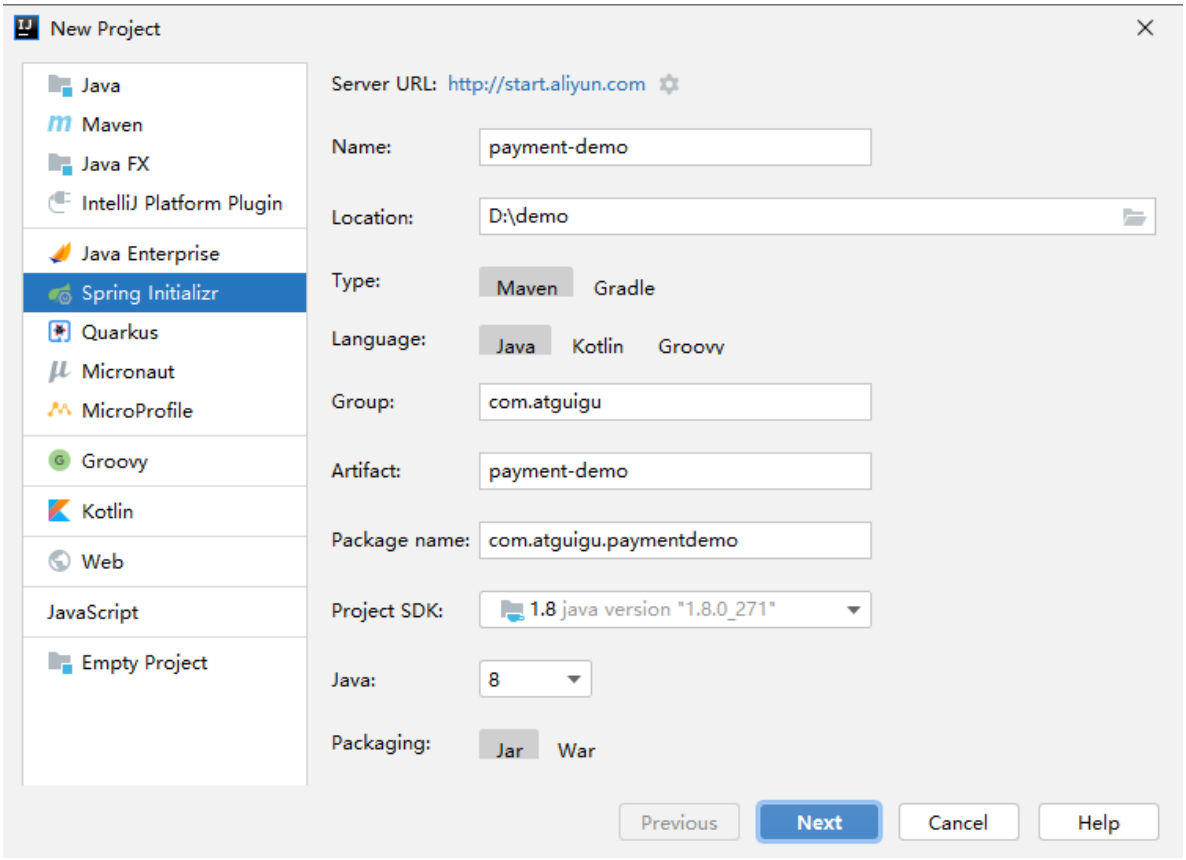
APIv3密钥对应V3版本的API

三、案例项目的创建

1、创建SpringBoot项目

1.1、新建项目

注意：Java版本选择8



1.2、添加依赖

添加SpringBoot web依赖

```
 <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

1.3、配置application.yml文件

```
server:
  port: 8090 #服务端口

spring:
  application:
    name: payment-demo # 应用名称
```

1.4、创建controller

创建controller包，创建ProductController类

```
package com.atguigu.paymentdemo.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/product")
@CrossOrigin //跨域
public class ProductController {

    @GetMapping("/test")
    public String test(){

        return "hello";
    }
}
```

1.5、测试

访问：http://localhost:8090/api/product/test

2、引入Swagger

作用：自动生成接口文档和测试页面。

2.1、引入依赖

```
<!--swagger-->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.7.0</version>
</dependency>

<!--swagger ui-->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.7.0</version>
</dependency>
```

2.2、Swagger配置文件

创建config包，创建Swagger2Config类

```
package com.atguigu.paymentdemo.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class Swagger2Config {

    @Bean
    public Docket docket(){
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(new ApiInfoBuilder().title("微信支付案例接口文
档").build());
    }
}
```

2.3、Swagger注解

controller中可以添加常用注解

```
@Api(tags="商品管理") //用在类上
```

```
@ApiOperation("测试接口") //用在方法上
```

2.4、测试

访问：http://localhost:8090/swagger-ui.html

3、定义统一结果

作用：定义统一响应结果，为前端返回标准格式的数据。

3.1、引入lombok依赖

简化实体类的开发

```
<!--实体对象工具类：低版本idea需要安装lombok插件-->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

3.2、创建R类

创建统一结果类

```
package com.atguigu.paymentdemo.vo;

import lombok.NoArgsConstructor;
import lombok.Setter;

import java.util.HashMap;
import java.util.Map;

@Data //生成set、get等方法
public class R {

    private Integer code;
    private String message;
    private Map<String, Object> data = new HashMap<>();

    public static R ok(){
        R r = new R();
        r.setCode(0);
        r.setMessage("成功");
        return r;
    }

    public static R error(){
        R r = new R();
        r.setCode(-1);
        r.setMessage("失败");
        return r;
    }

    public R data(String key, Object value){
        this.data.put(key, value);
        return this;
    }

}
```

3.3、修改controller

修改test方法，返回统一结果

```
@ApiOperation("测试接口")
@GetMapping("/test")
public R test(){

    return R
        .ok()
        .data("message", "hello")
        .data("now", new Date());
}
```

3.4、配置json时间格式

```
spring:
  jackson: #json时间格式
    date-format: yyyy-MM-dd HH:mm:ss
    time-zone: GMT+8
```

3.5、Swagger测试

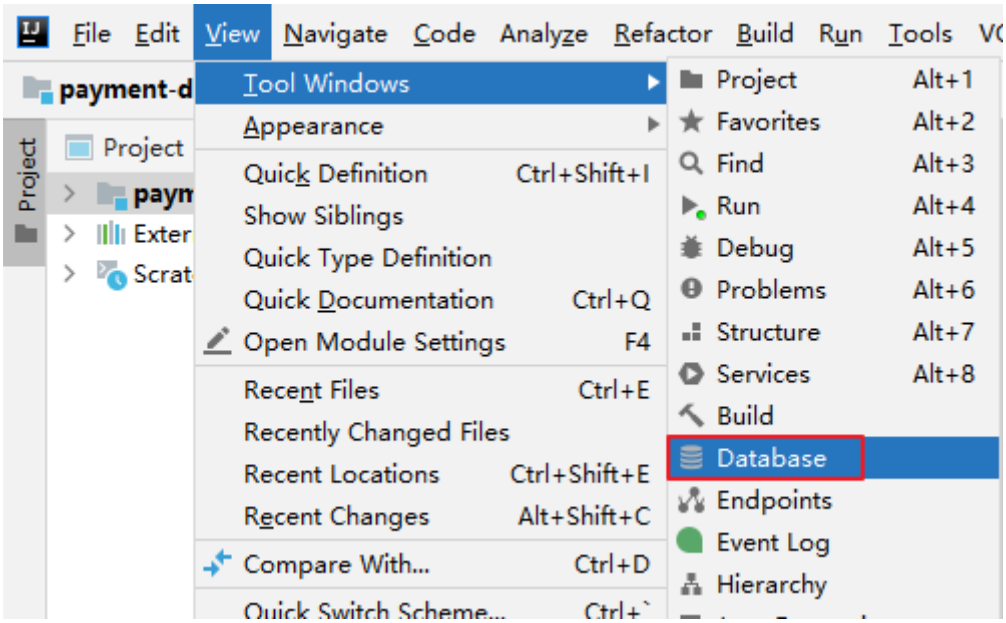
4、创建数据库

4.1、创建数据库

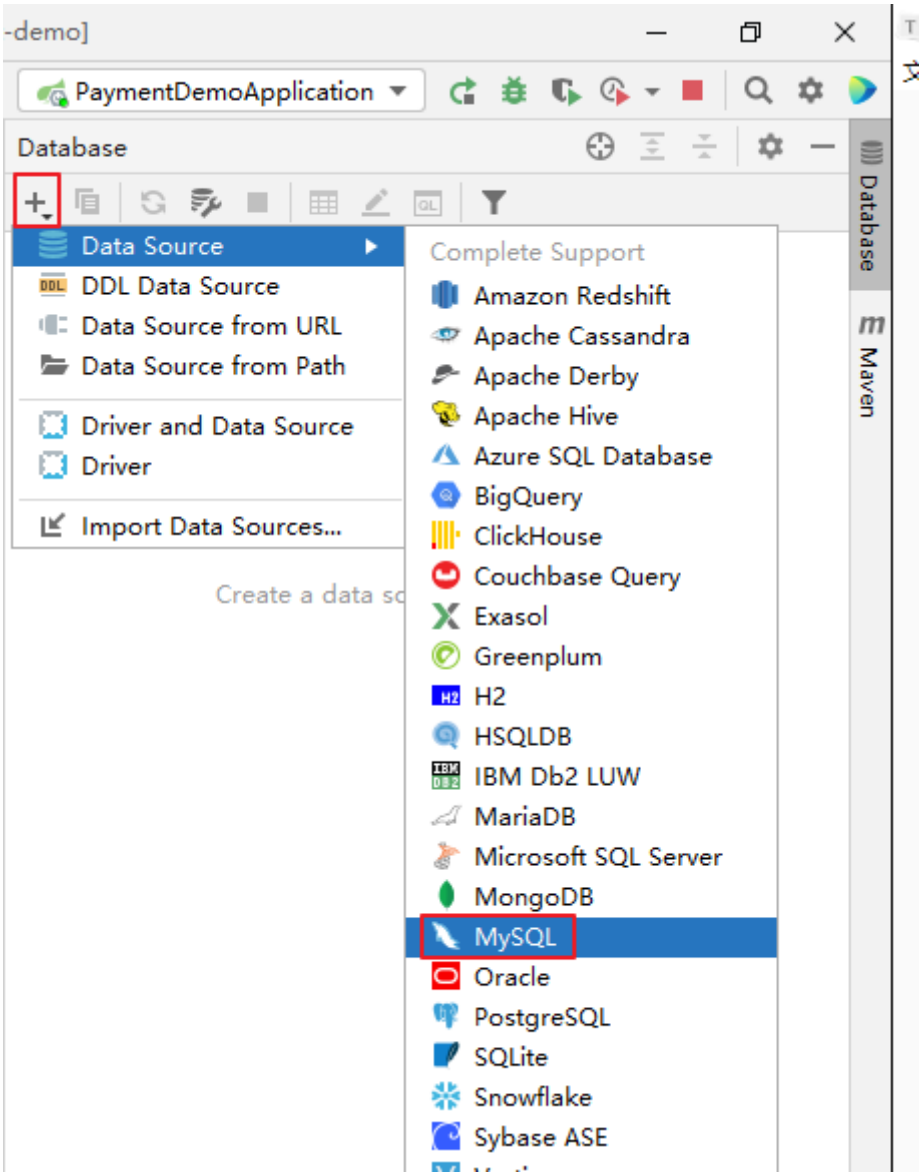
```
mysql -uroot -p
mysql> create database payment_demo;
```

4.2、IDEA配置数据库连接

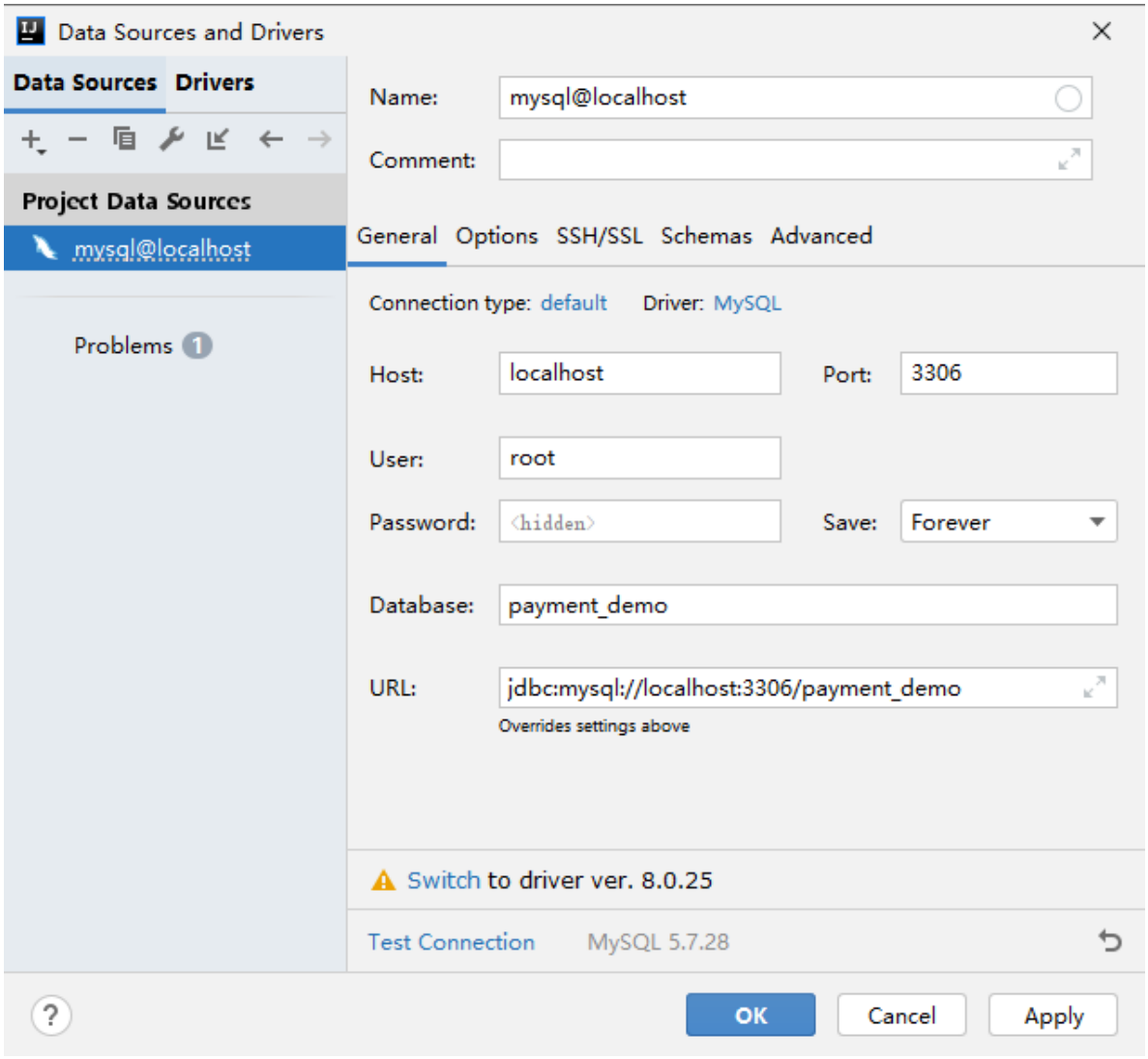
（1）打开数据库面板



(2) 添加数据库

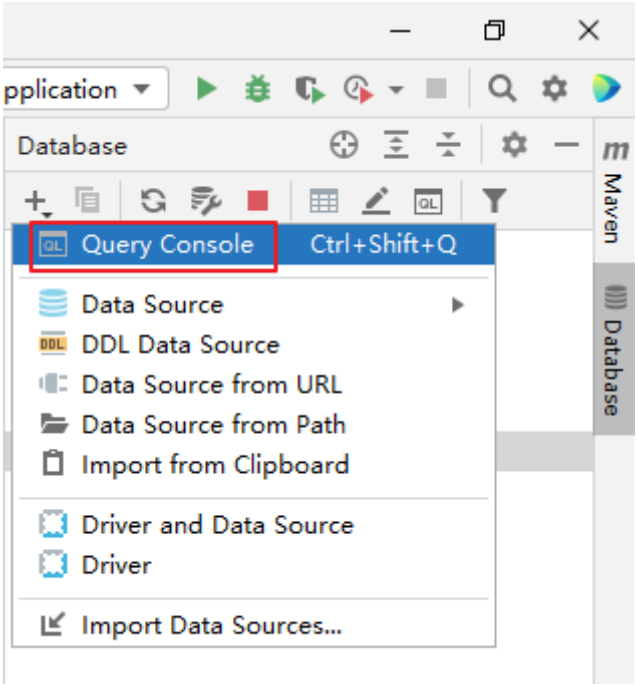


(3) 配置数据库连接参数



4.3、执行SQL脚本

payment_demo.sql



5、集成MyBatis-Plus

5.1、引入依赖

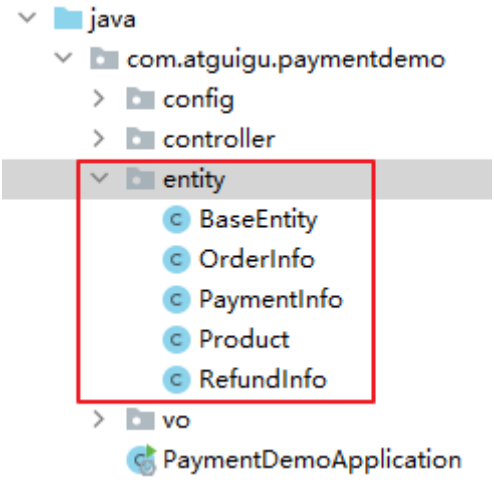
```
<!--mysql驱动-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>

<!--持久层-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.3.1</version>
</dependency>
```

5.2、配置数据库连接

```
spring:
  datasource: #mysql数据库连接
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/payment_demo?
serverTimezone=GMT%2B8&characterEncoding=utf-8
    username: root
    password: 123456
```

5.3、定义实体类

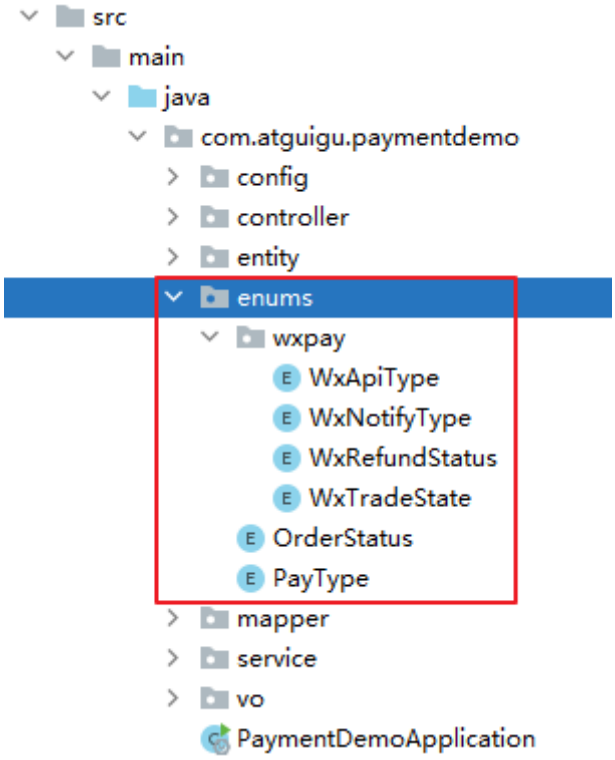
BaseEntity是父类，其他类继承BaseEntity



5.4、定义持久层

定义Mapper接口继承 BaseMapper<>，

定义xml配置文件



5.5、定义MyBatis-Plus的配置文件

在config包中创建配置文件 MybatisPlusConfig

```
package com.atguigu.paymentdemo.config;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@MapperScan("com.atguigu.paymentdemo.mapper") //持久层扫描
@EnableTransactionManagement //启用事务管理
public class MybatisPlusConfig {

}
```

5.6、定义yml配置文件

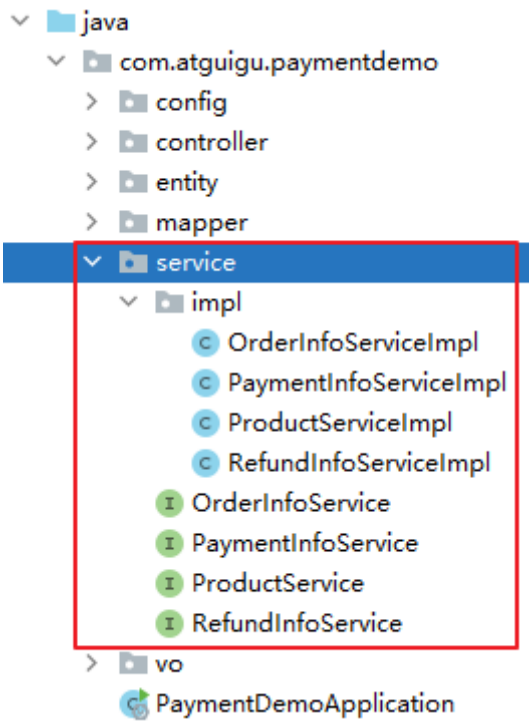
添加持久层日志和xml文件位置的配置

```
mybatis-plus:
  configuration: #sql日志
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
  mapper-locations: classpath:com/atguigu/paymentdemo/mapper/xml/*.xml
```

5.7、定义业务层

定义业务层接口继承 IService<>

定义业务层接口的实现类，并继承 ServiceImpl<,>



5.8、定义接口方法查询所有商品

在 public class ProductController 中添加一个方法

```
@Resource
private ProductService productService;

@ApiOperation("商品列表")
@GetMapping("/list")
public R list(){
    List<Product> list = productService.list();
    return R.ok().data("productList", list);
}
```

5.9、Swagger中测试

5.10、pom中配置build节点

因为maven工程在默认情况下 src/main/java 目录下的所有资源文件是不发布到 target 目录下的，我们在 pom 文件的 节点下配置一个资源发布过滤器

```
<!-- 项目打包时会将java目录中的*.xml文件也进行打包 -->
<resources>
    <resource>
        <directory>src/main/java</directory>
        <includes>
            <include>**/*.xml</include>
        </includes>
        <filtering>false</filtering>
    </resource>
</resources>
```

6、搭建前端环境

6.1、安装Node.js

Node.js是一个基于JavaScript引擎的服务器端环境，前端项目在开发环境下要基于Node.js来运行

安装：node-v14.18.0-x64.msi

6.2、运行前端项目

将项目放在磁盘的一个目录中，例如 D:\demo\payment-demo-front

进入项目目录，运行下面的命令启动项目：

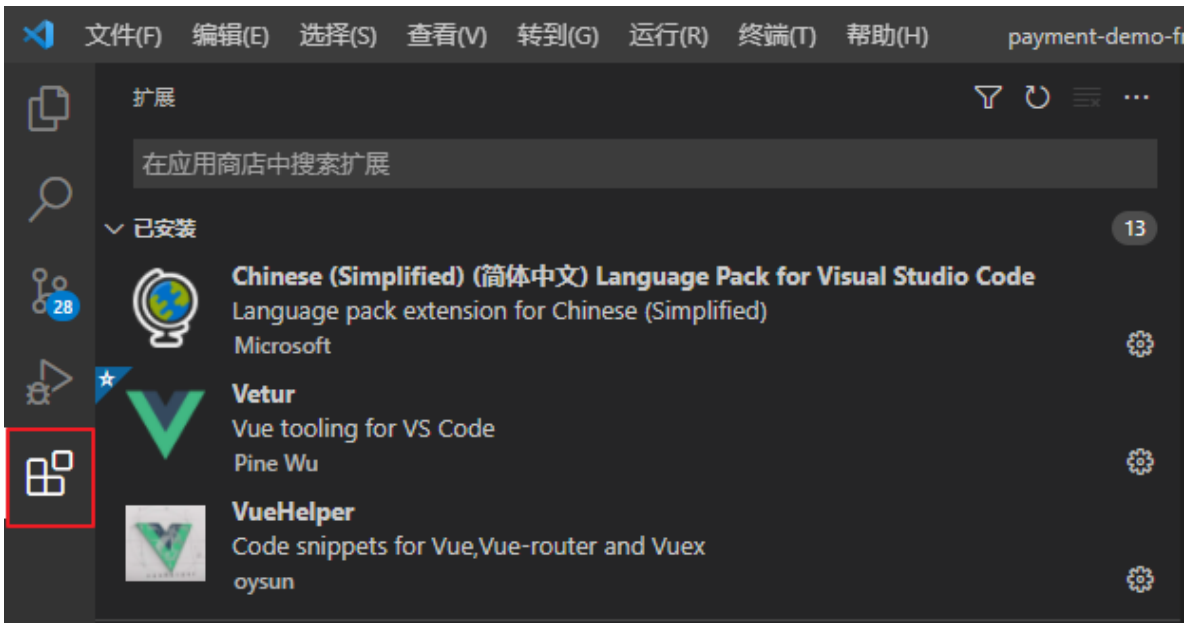
```
npm run serve
```

6.3、安装VSCode

如果你希望方便的查看和修改前端代码，可以安装一个VSCode

安装：VSCodeUserSetup-x64-1.56.2

安装插件：



7、Vue.js入门

官网：https://cn.vuejs.org/

Vue.js是一个前端框架，帮助我们快速构建前端项目。

使用vue有两种方式，一个是传统的在 html 文件中引入 js 脚本文件的方式，另一个是脚手架的方式。我们的项目，使用的是脚手架的方式。

7.1、安装脚手架

配置淘宝镜像

```
#经过下面的配置，所有的 npm install 都会经过淘宝的镜像地址下载
npm config set registry https://registry.npm.taobao.org
```

全局安装脚手架

```
npm install -g @vue/cli
```

7.2、创建一个项目

先进入项目目录（Ctrl + ~），然后创建一个项目

```
vue create vue-demo
```

7.3、运行项目

```
npm run serve
```

指定运行端口

```
npm run serve -- --port 8888
```

7.4、数据绑定

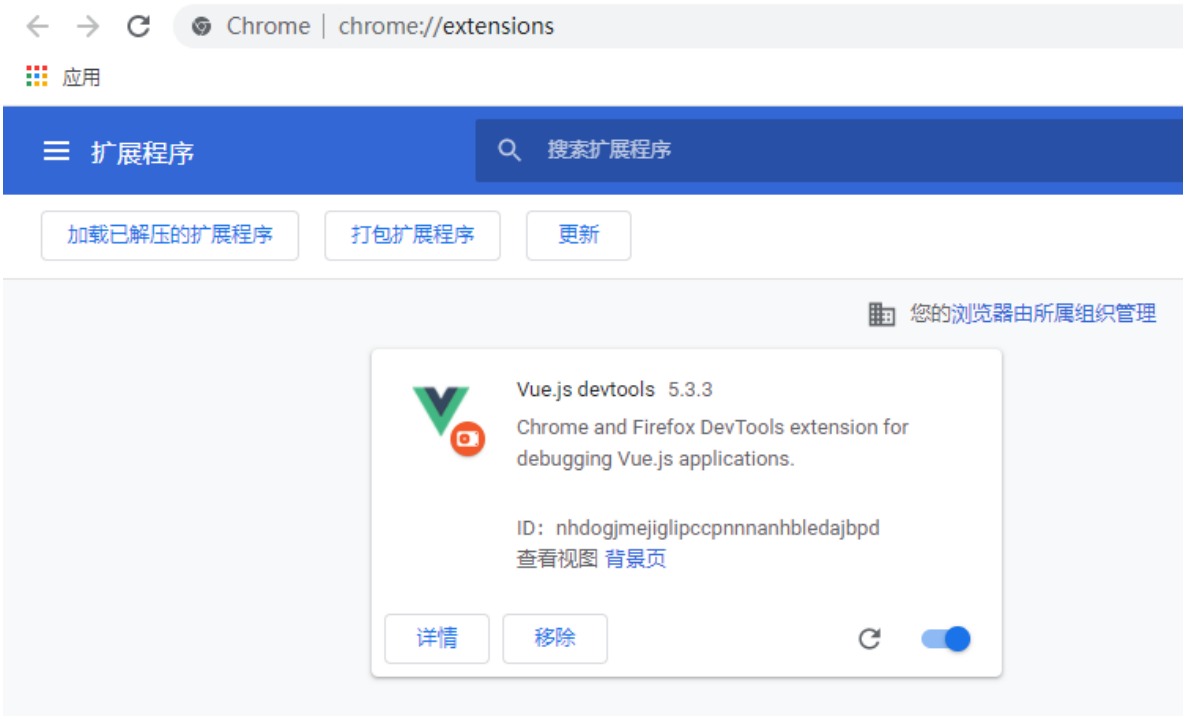
修改 src/App.vue

```
<!--定义页面结构-->
<template>
  <div>
    <h1>Vue案例</h1>
    <!-- 插值 -->
    <p>{{course}}</p>
  </div>
</template>

<!--定义页面脚本-->
<script>
export default {
  // 定义数据
  data () {
    return {
      course: '微信支付'
    }
  }
}
</script>
```

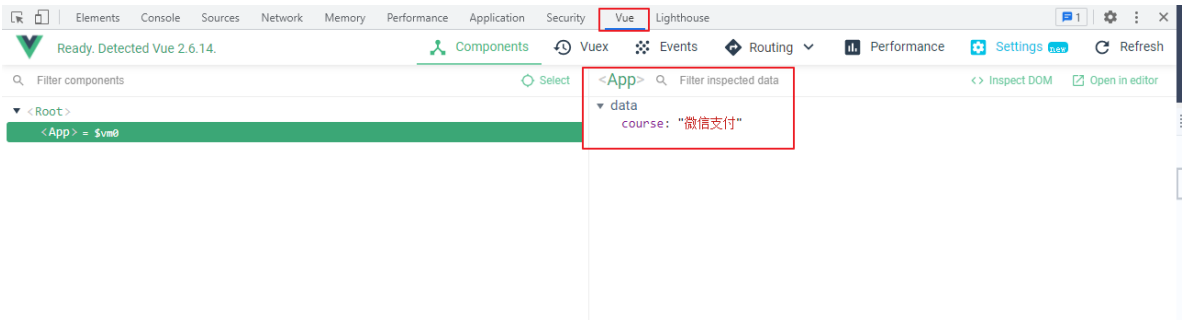
7.5、安装Vue调试工具

在Chrome的扩展程序中安装：Vue.jsDevtools.zip

（1）扩展程序的安装



（2）扩展程序的使用



7.6、双向数据绑定

数据会绑定到组件，组件的改变也会影响数据定义

```
<p>
    <!-- 指令 -->
    <input type="text" v-model="course">
</p>
```

7.7、事件处理

（1）定义事件

```
// 定义方法
methods: {
    toPay(){
        console.log('去支付')
    }
}
```

（2）调用事件

```
<p>
    <!-- 事件 -->
    <button @click="toPay()">去支付</button>
</p>
```

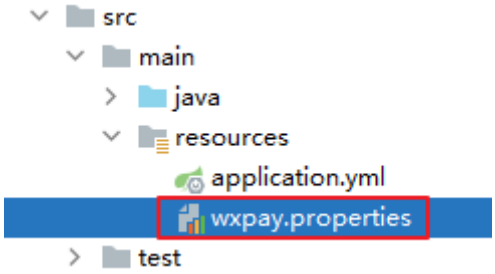
四、基础支付API V3

1、引入支付参数

1.1、定义微信支付相关参数

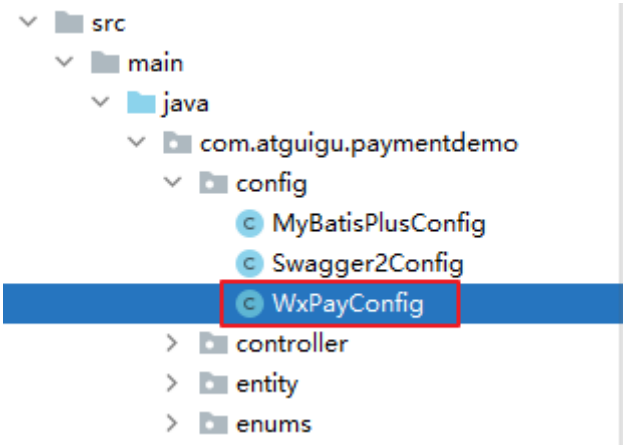
将资料文件夹中的 wxpay.properties 复制到resources目录中

这个文件定义了之前我们准备的微信支付相关的参数，例如商户号、APPID、API秘钥等等



1.2、读取支付参数

将资料文件夹中的 config 目录中的 WxPayConfig.java 复制到源码目录中。



1.3、测试支付参数的获取

在 controller 包中创建 TestController

```
package com.atguigu.paymentdemo.controller;
```

```
import com.atguigu.paymentdemo.config.WxPayConfig;
import com.atguigu.paymentdemo.vo.R;
import io.swagger.annotations.Api;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;

@Api(tags = "测试控制器")
@RestController
@RequestMapping("/api/test")
public class TestController {

    @Resource
    private WxPayConfig wxPayConfig;

    @GetMapping("/get-wx-pay-config")
    public R getWxPayConfig(){

        String mchId = wxPayConfig.getMchId();
        return R.ok().data("mchId", mchId);
    }
}
```

1.4、配置 Annotation Processor

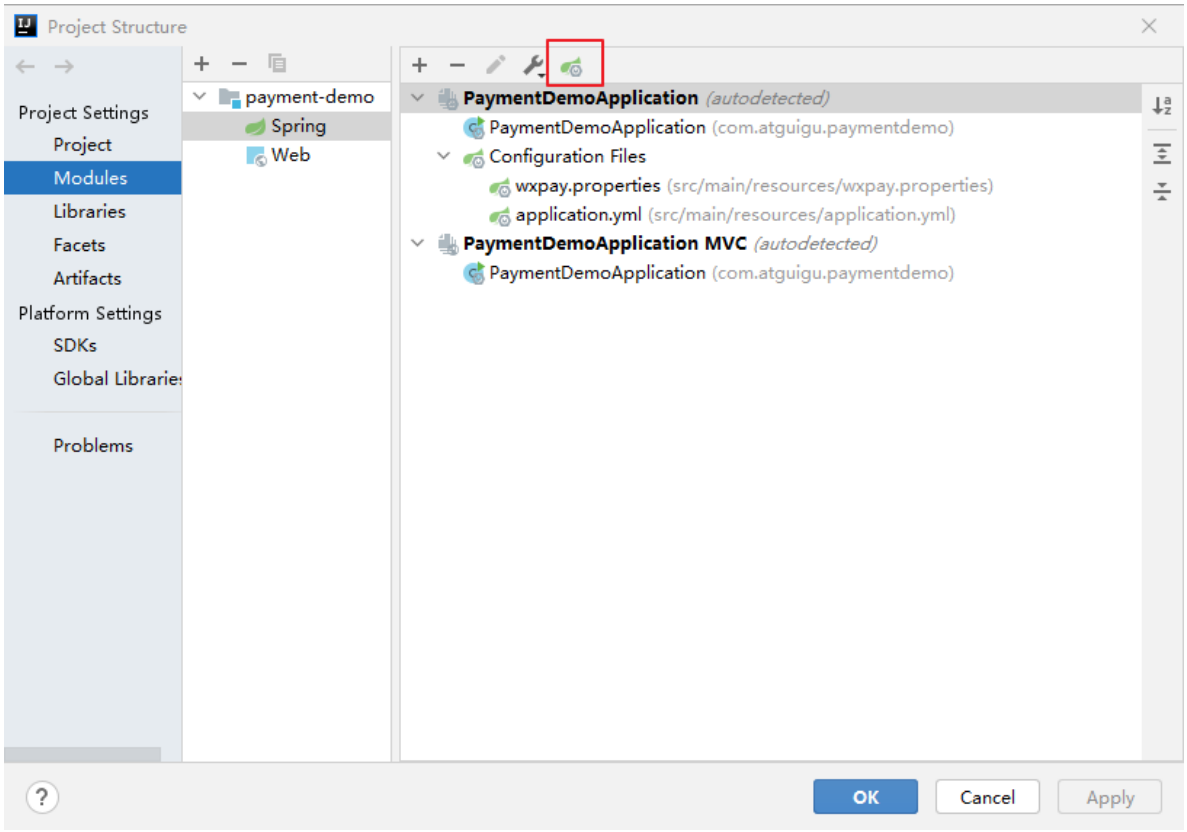
可以帮助我们生成自定义配置的元数据信息，让配置文件和Java代码之间的对应参数可以自动定位，方便开发。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

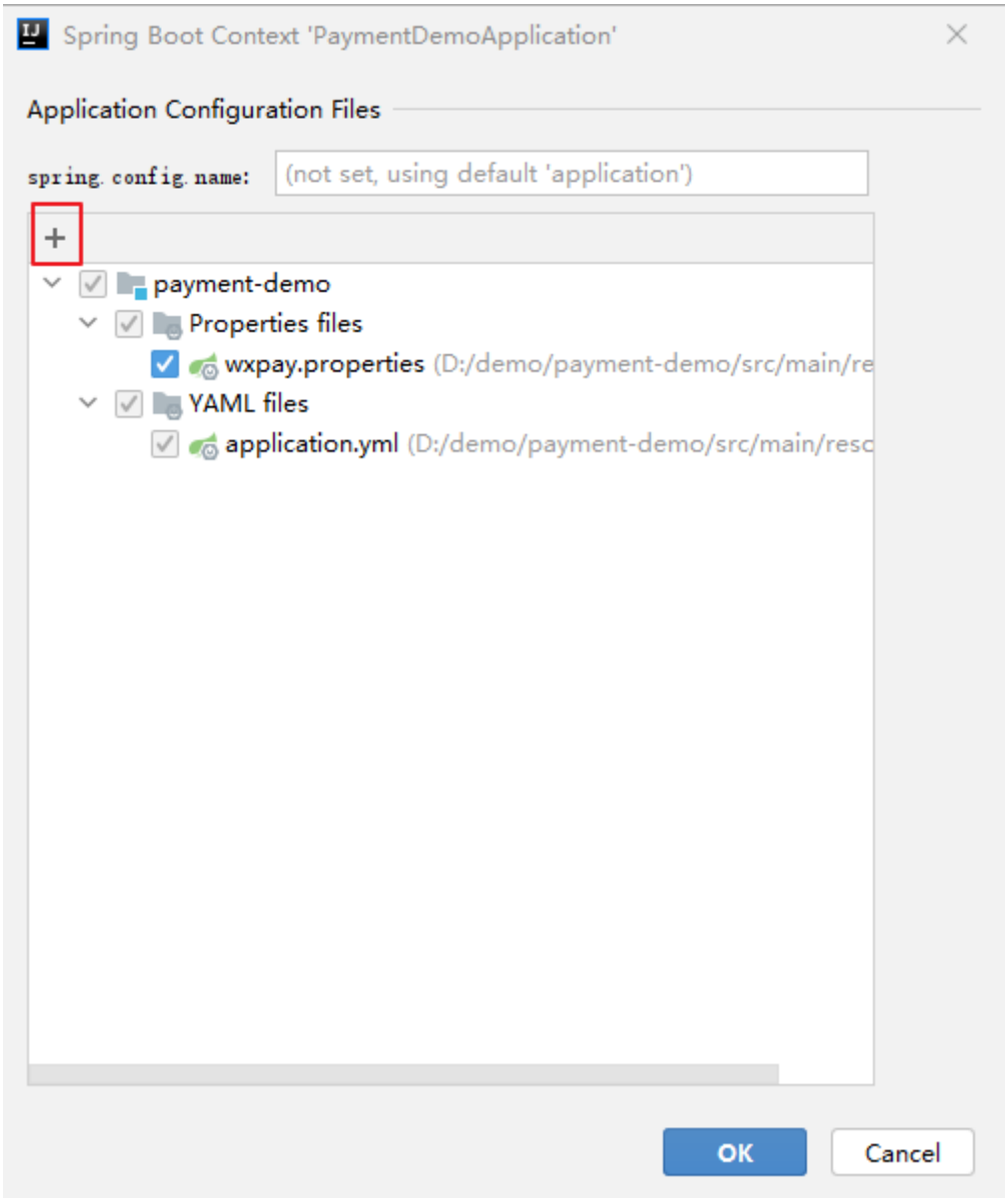
1.5、在IDEA中设置 SpringBoot 配置文件

让IDEA可以识别配置文件，将配置文件的图标展示成SpringBoot的图标，同时配置文件的内容可以高亮显示

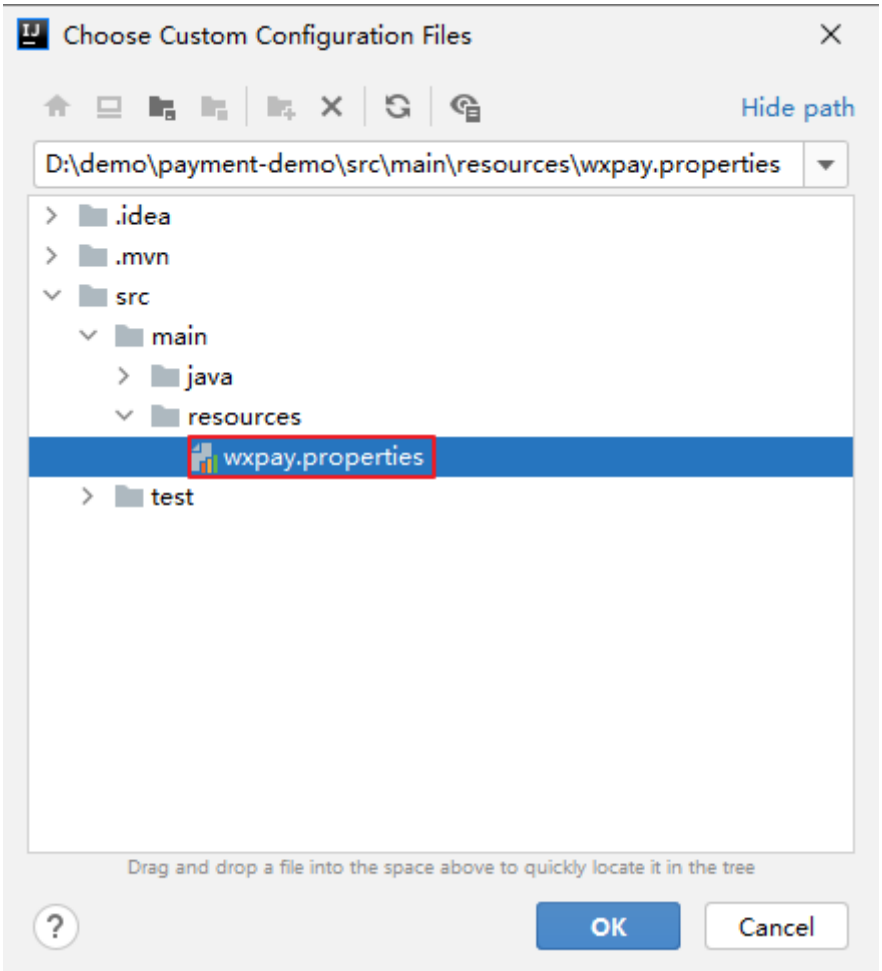
File -> Project Structure -> Modules -> 选择小叶子


Project Structure
Project Settings
Project
Modules
Libraries
Facets
Artifacts
Platform Settings
SDKs
Global Librarie
Problems
payment-demo
Spring
Web
PaymentDemoApplication (autodetected)
PaymentDemoApplication (com.atguigu.paymentdemo)
Configuration Files
wxpay.properties (src/main/resources/wxpay.properties)
application.yml (src/main/resources/application.yml)
PaymentDemoApplication MVC (autodetected)
PaymentDemoApplication (com.atguigu.paymentdemo)
OK
Cancel
Apply

点击（+） 图标


Spring Boot Context 'PaymentDemoApplication'
Application Configuration Files
spring.config.name:
(not set, using default 'application')
payment-demo
Properties files
wxpay.properties (D:/demo/payment-demo/src/main/re
YAML files
application.yml (D:/demo/payment-demo/src/main/resc
OK
Cancel

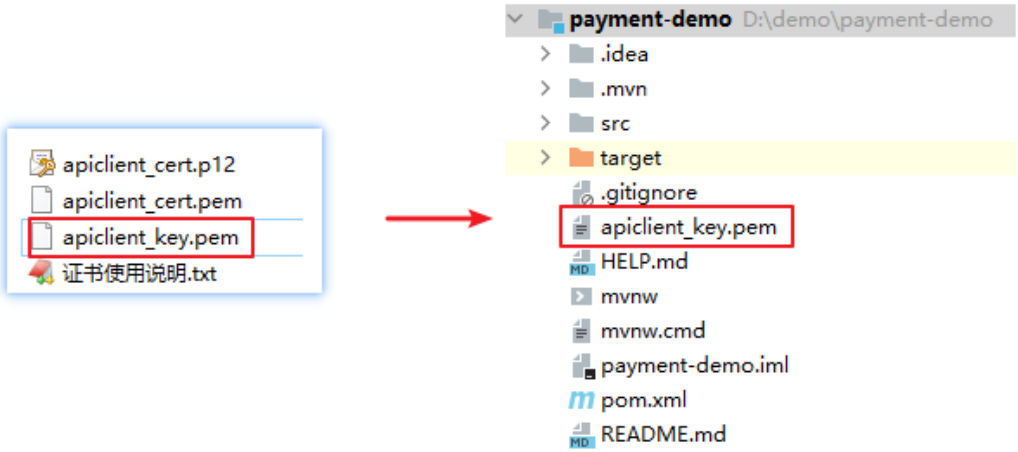
选中配置文件：



2、加载商户私钥

2.1、复制商户私钥

将下载的私钥文件复制到项目根目录下：



2.2、引入SDK

https://pay.weixin.qq.com/wiki/doc/apiv3/wechatpay/wechatpay6_0.shtml

我们可以使用官方提供的 SDK，帮助我们完成开发。实现了请求签名的生成和应答签名的验证。

```
<dependency>
    <groupId>com.github.wechatpay-apiv3</groupId>
    <artifactId>wechatpay-apache-httpclient</artifactId>
    <version>0.3.0</version>
</dependency>
```

2.3、获取商户私钥

https://github.com/wechatpay-apiv3/wechatpay-apache-httpclient （如何加载商户私钥）

```
/**
     * 获取商户私钥
     * @param filename
     * @return
     */
public PrivateKey getPrivateKey(String filename){

    try {
        return PemUtil.loadPrivateKey(new FileInputStream(filename));
    } catch (FileNotFoundException e) {
        throw new RuntimeException("私钥文件不存在", e);
    }
}
```

2.4、测试商户私钥的获取

在 PaymentDemoApplicationTests 测试类中添加如下方法，测试私钥对象是否能够获取出来。

（将前面的方法改成public的再进行测试）

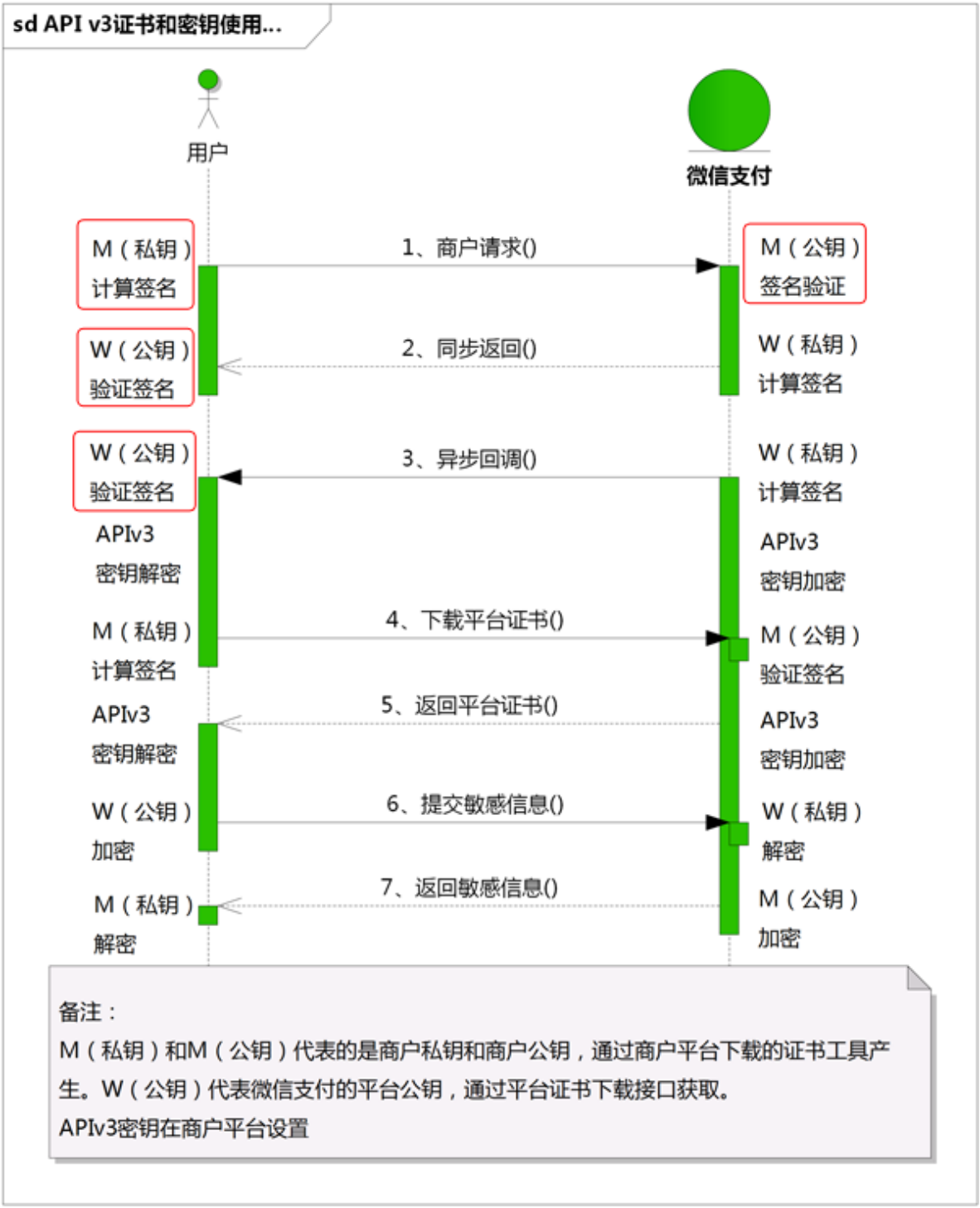
```
package com.atguigu.paymentdemo;

import com.atguigu.paymentdemo.config.WxPayConfig;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import javax.annotation.Resource;
import java.security.PrivateKey;

@SpringBootTest
class PaymentDemoApplicationTests {

    @Resource
    private WxPayConfig wxPayConfig;

    /**
     * 获取商户私钥
     */
    @Test
    public void testGetPrivateKey(){

        //获取私钥路径
```

```
        String privateKeyPath = wxPayConfig.getPrivateKeyPath();

        //获取商户私钥
        PrivateKey privateKey = wxPayConfig.getPrivateKey(privateKeyPath);

        System.out.println(privateKey);
    }
}
```

3、获取签名验证器和HttpClient

3.1、证书密钥使用说明

https://pay.weixin.qq.com/wiki/doc/apiv3_partner/wechatpay/wechatpay3_0.shtml



3.2、获取签名验证器

https://github.com/wechatpay-apiv3/wechatpay-apache-httpclient （定时更新平台证书功能）

平台证书：平台证书封装了微信的公钥，商户可以使用平台证书中的公钥进行验签。

签名验证器：帮助我们进行验签工作，我们单独将它定义出来，方便后面的开发。

```
    /**
     * 获取签名验证器
     * @return
     */
    @Bean
    public ScheduledUpdateCertificatesVerifier getVerifier(){

        //获取商户私钥
        PrivateKey privateKey = getPrivateKey(privateKeyPath);

        //私钥签名对象（签名）
        PrivateKeySigner privateKeySigner = new PrivateKeySigner(mchSerialNo,
privateKey);

        //身份认证对象（验签）
        WechatPay2Credentials wechatPay2Credentials = new
WechatPay2Credentials(mchId, privateKeySigner);

        // 使用定时更新的签名验证器，不需要传入证书
        ScheduledUpdateCertificatesVerifier verifier = new
ScheduledUpdateCertificatesVerifier(
                wechatPay2Credentials,
                apiV3Key.getBytes(StandardCharsets.UTF_8));

        return verifier;
    }
```

3.4、获取 HttpClient 对象

https://github.com/wechatpay-apiv3/wechatpay-apache-httpclient （定时更新平台证书功能）

HttpClient 对象：是建立远程连接的基础，我们通过SDK创建这个对象。

```
/**
     * 获取HttpClient对象
     * @param verifier
     * @return
     */
@Bean
public CloseableHttpClient getWxPayClient(ScheduledUpdateCertificatesVerifier
verifier){

    //获取商户私钥
    PrivateKey privateKey = getPrivateKey(privateKeyPath);
```

```
    //用于构造HttpClient
    WechatPayHttpClientBuilder builder = WechatPayHttpClientBuilder.create()
        .withMerchant(mchId, mchSerialNo, privateKey)
        .withValidator(new WechatPay2Validator(verifier));
    // ... 接下来，你仍然可以通过builder设置各种参数，来配置你的HttpClient

    // 通过WechatPayHttpClientBuilder构造的HttpClient，会自动的处理签名和验签，并进行证书自动更新
    CloseableHttpClient httpClient = builder.build();

    return httpClient;
}
```

4、API字典和相关工具

4.1、API列表

https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2_7_3.shtml

我们的项目中要实现以下所有API的功能。

模块名称	功能列表	描述
Native支付	Native下单	通过本接口提交微信支付Native支付订单
	查询订单	通过此接口查询订单状态
	关闭订单	通过此接口关闭待支付订单
	Native调起支付	商户后台系统先调用微信支付的Native支付接口，微信后台系统返回链接参数code_url，商户后台系统将code_url值生成二维码图片，用户使用微信客户端扫码后发起支付。
	支付结果通知	微信支付通过支付通知接口将用户支付成功消息通知给商户
	申请退款	商户可以通过该接口将支付金额退还给买家
	查询单笔退款	提交退款申请后，通过调用该接口查询退款状态
	退款结果通知	微信支付通过退款通知接口将用户退款成功消息通知给商户
	申请交易账单	商户可以通过该接口获取交易账单文件的下载地址
	申请资金账单	商户可以通过该接口获取资金账单文件的下载地址
	下载账单	通过申请交易/资金账单获取到download_url在该接口获取到对应的账单。

4.2、接口规则

https://pay.weixin.qq.com/wiki/doc/apiv3/wechatpay/wechatpay2_0.shtml

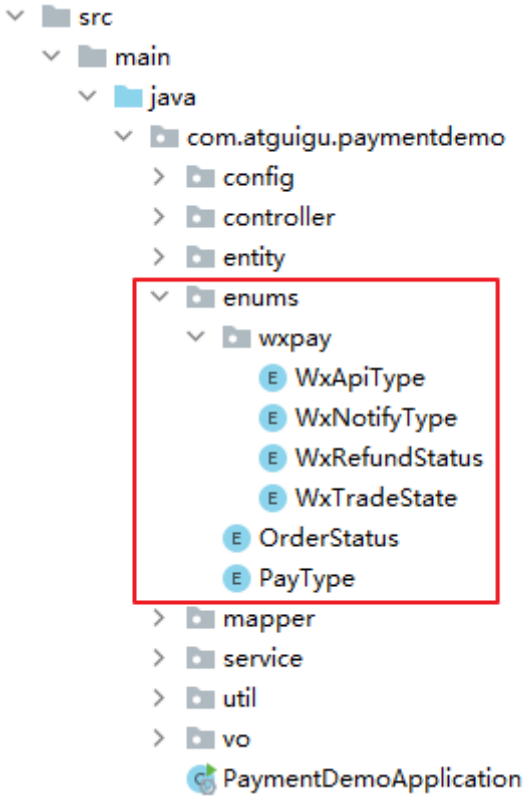
微信支付 APIv3 使用 JSON 作为消息体的数据交换格式。

```
<!--json处理-->
<dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
</dependency>
```

4.3、定义枚举

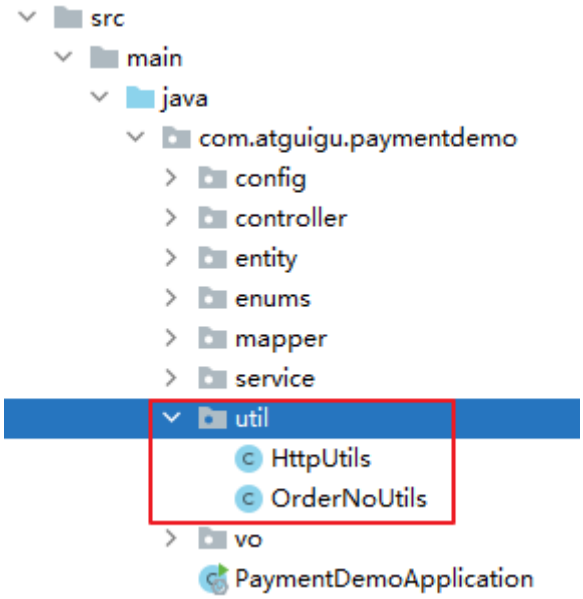
将资料文件夹中的 enums 目录复制到源码目录中。

为了开发方便，我们预先在项目中定义一些枚举。枚举中定义的内容包括接口地址，支付状态等信息。



4.4、添加工具类

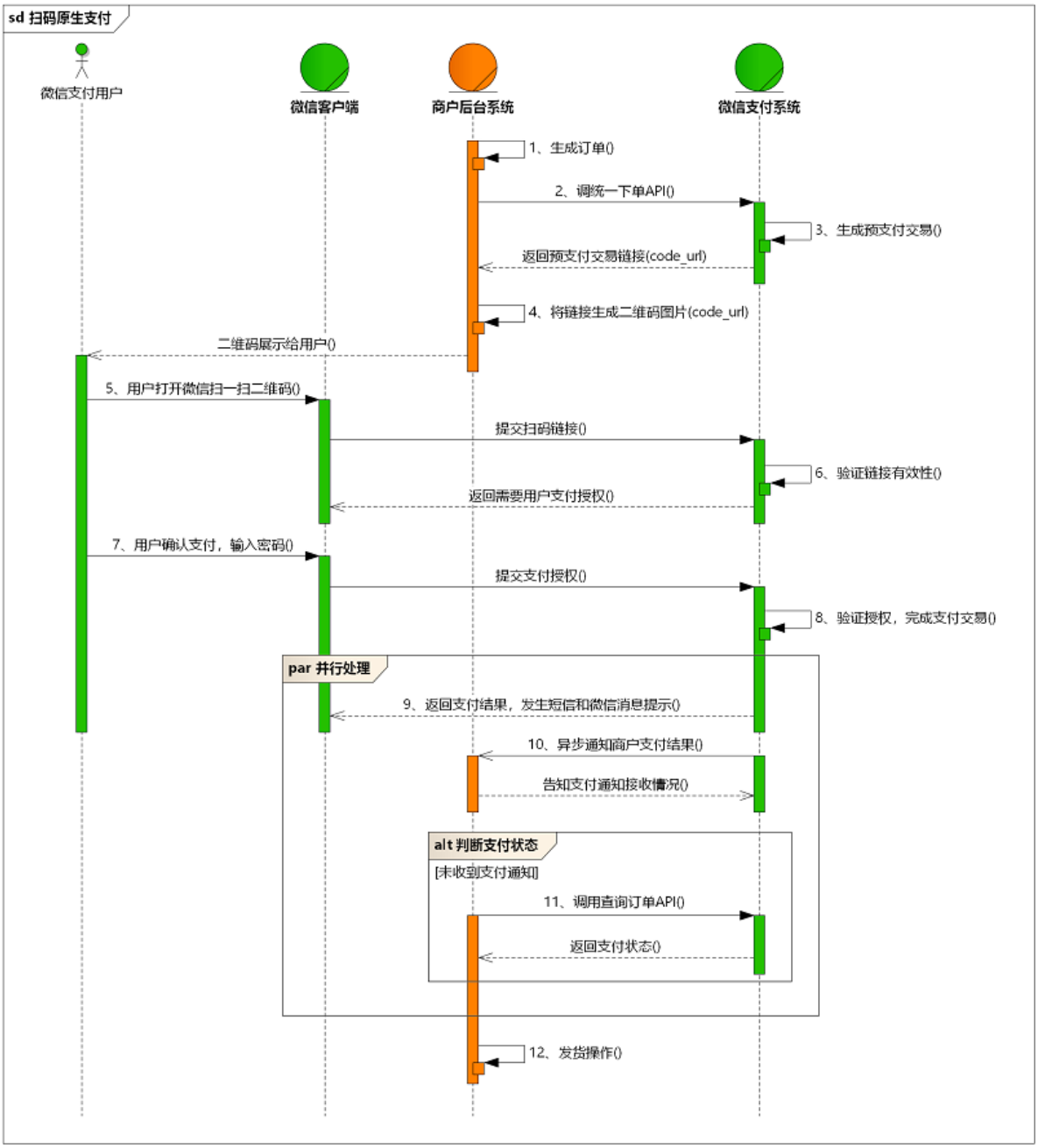
将资料文件夹中的 util 目录复制到源码目录中，我们将会使用这些辅助工具简化项目的开发



5、Native下单API

5.1、Native支付流程

https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_4.shtml



5.2、Native下单API

https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_1.shtml

商户端发起支付请求，微信端创建支付订单并生成支付二维码链接，微信端将支付二维码返回给商户端，商户端显示支付二维码，用户使用微信客户端扫码后发起支付。

（1）创建 WxPayController

```
package com.atguigu.paymentdemo.controller;

import io.swagger.annotations.Api;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RestController;

@CrossOrigin
@RestController
@RequestMapping("/api/wx-pay")
@Api(tags = "网站微信支付")
@Slf4j
public class WxPayController {

}
```

(2) 创建 WxPayService

接口

```
package com.atguigu.paymentdemo.service;

public interface WxPayService {

}
```

实现

```
package com.atguigu.paymentdemo.service.impl;

import com.atguigu.paymentdemo.service.WxPayService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

@Service
@Slf4j
public class WxPayServiceImpl implements WxPayService {

}
```

(3) 定义WxPayController方法

```
@Resource
private WxPayService wxPayService;

/**
     * Native下单
     * @param productId
     * @return
     * @throws Exception
     */
@ApiOperation("调用统一下单API，生成支付二维码")
@PostMapping("/native/{productId}")
public R nativePay(@PathVariable Long productId) throws Exception {

    log.info("发起支付请求");

    //返回支付二维码连接和订单号
    Map<String, Object> map = wxPayService.nativePay(productId);

    return R.ok().setData(map);
```

```
}
```

R对象中添加 @Accessors(chain = true)，使其可以链式操作

```
@Data
@Accessors(chain = true) //链式操作
public class R {
```

（4）定义WxPayService方法

> 参考：
>
> API字典 -> 基础支付 -> Native支付 -> Native下单：
>
> https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_1.shtml
>
> 指引文档 -> 基础支付 -> Native支付 -> 开发指引 ->【服务端】Native下单：
>
> https://pay.weixin.qq.com/wiki/doc/apiv3/open/pay/chapter2_7_2.shtml

接口

```
Map<String, Object> nativePay(Long productId) throws Exception;
```

实现

```
@Resource
private WxPayConfig wxPayConfig;

@Resource
private CloseableHttpClient wxPayClient;

/**
     * 创建订单，调用Native支付接口
     * @param productId
     * @return code_url 和 订单号
     * @throws Exception
     */
@Override
public Map<String, Object> nativePay(Long productId) throws Exception {

    log.info("生成订单");

    //生成订单
    OrderInfo orderInfo = new OrderInfo();
    orderInfo.setTitle("test");
    orderInfo.setOrderNo(OrderNoUtils.getOrderNo()); //订单号
    orderInfo.setProductId(productId);
    orderInfo.setTotalFee(1); //分
    orderInfo.setOrderStatus(OrderStatus.NOTPAY.getType());
    //TODO：存入数据库

    log.info("调用统一下单API");

    //调用统一下单API
    HttpPost httpPost = new
HttpPost(wxPayConfig.getDomain().concat(WxApiType.NATIVE_PAY.getType()));
```

```
    // 请求body参数
    Gson gson = new Gson();
    Map paramsMap = new HashMap();
    paramsMap.put("appid", wxPayConfig.getAppid());
    paramsMap.put("mchid", wxPayConfig.getMchId());
    paramsMap.put("description", orderInfo.getTitle());
    paramsMap.put("out_trade_no", orderInfo.getOrderNo());
    paramsMap.put("notify_url",
wxPayConfig.getNotifyDomain().concat(WxNotifyType.NATIVE_NOTIFY.getType()));

    Map amountMap = new HashMap();
    amountMap.put("total", orderInfo.getTotalFee());
    amountMap.put("currency", "CNY");

    paramsMap.put("amount", amountMap);

    //将参数转换成json字符串
    String jsonParams = gson.toJson(paramsMap);
    log.info("请求参数: " + jsonParams);

    StringEntity entity = new StringEntity(jsonParams,"utf-8");
    entity.setContentType("application/json");
    httpPost.setEntity(entity);
    httpPost.setHeader("Accept", "application/json");

    //完成签名并执行请求
    CloseableHttpResponse response = wxPayClient.execute(httpPost);

    try {
        String bodyAsString = EntityUtils.toString(response.getEntity());//响应体
        int statusCode = response.getStatusLine().getStatusCode();//响应状态码
        if (statusCode == 200) { //处理成功
            log.info("成功, 返回结果 = " + bodyAsString);
        } else if (statusCode == 204) { //处理成功，无返回Body
            log.info("成功");
        } else {
            log.info("Native下单失败,响应码 = " + statusCode+ ",返回结果 = " +
bodyAsString);
            throw new IOException("request failed");
        }

        //响应结果
        Map<String, String> resultMap = gson.fromJson(bodyAsString,
HashMap.class);
        //二维码
        String codeUrl = resultMap.get("code_url");

        Map<String, Object> map = new HashMap<>();
        map.put("codeUrl", codeUrl);
        map.put("orderNo", orderInfo.getOrderNo());

        return map;

    } finally {
        response.close();
    }
}
```

5.3、签名和验签源码解析

（1）签名原理

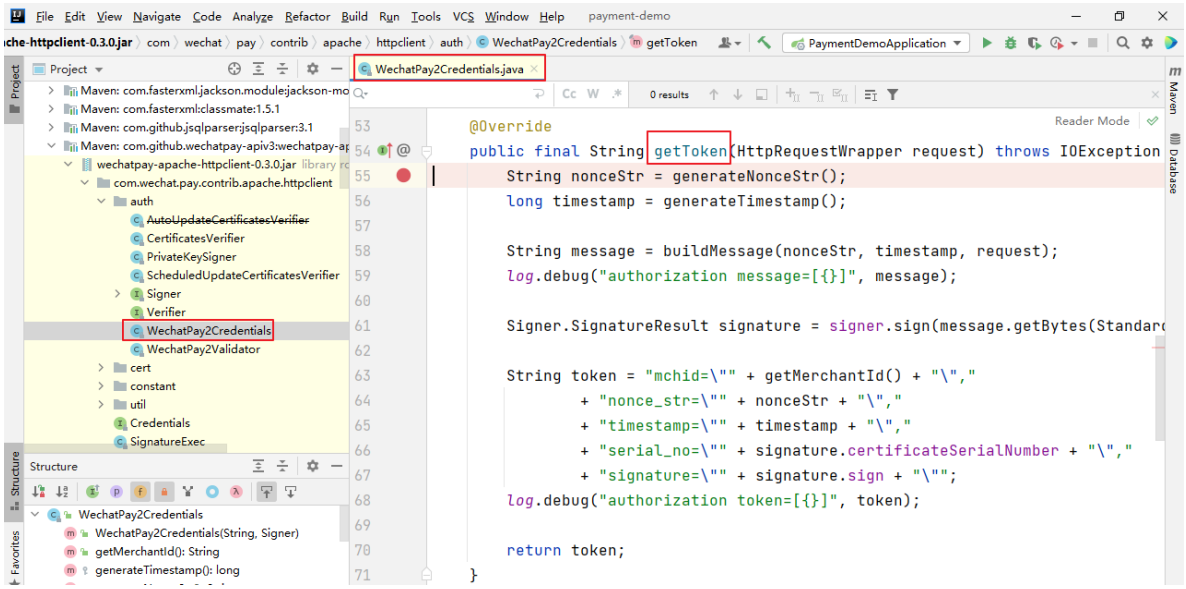
开启debug日志

```
logging:
  level:
    root: info
```

签名生成流程：

https://pay.weixin.qq.com/wiki/doc/apiv3/wechatpay/wechatpay4_0.shtml

签名生成源码：

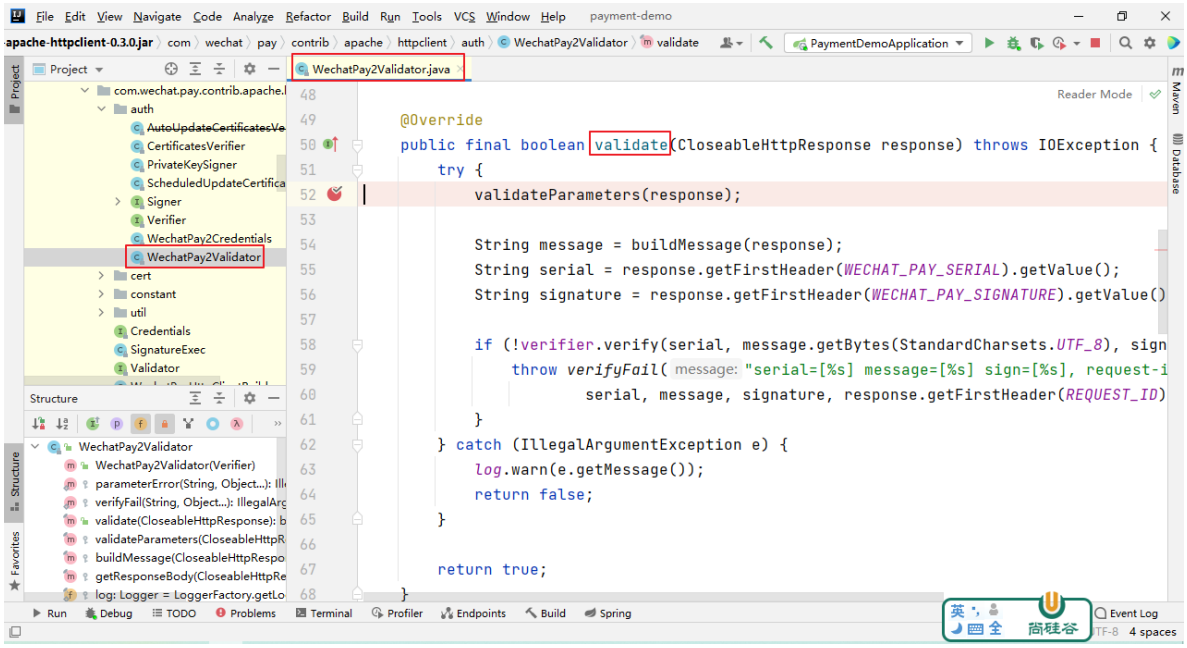


（2）验签原理

签名验证流程：

https://pay.weixin.qq.com/wiki/doc/apiv3/wechatpay/wechatpay4_1.shtml

签名验证源码：



5.4、创建课程订单

（1）保存订单

OrderInfoService

接口：

```
OrderInfo createOrderByProductId(Long productId);
```

实现：

```
@Resource
private ProductMapper productMapper;

@Override
public OrderInfo createOrderByProductId(Long productId) {

    //查找已存在但未支付的订单
    OrderInfo orderInfo = this.getNoPayOrderByProductId(productId);
    if( orderInfo != null){
        return orderInfo;
    }

    //获取商品信息
    Product product = productMapper.selectById(productId);

    //生成订单
    orderInfo = new OrderInfo();
    orderInfo.setTitle(product.getTitle());
    orderInfo.setOrderNo(OrderNoUtils.getOrderNo()); //订单号
    orderInfo.setProductId(productId);
    orderInfo.setTotalFee(product.getPrice()); //分
    orderInfo.setOrderStatus(OrderStatus.NOTPAY.getType());
    baseMapper.insert(orderInfo);
```

```
    return orderInfo;
}
```

查找未支付订单：OrderInfoService中添加辅助方法

```
/**
     * 根据商品id查询未支付订单
     * 防止重复创建订单对象
     * @param productId
     * @return
     */
private OrderInfo getNoPayOrderByProductId(Long productId) {

    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("product_id", productId);
    queryWrapper.eq("order_status", OrderStatus.NOTPAY.getType());
    //        queryWrapper.eq("user_id", userId);
    OrderInfo orderInfo = baseMapper.selectOne(queryWrapper);
    return orderInfo;
}
```

（2）缓存二维码

OrderInfoService

接口：

```
void saveCodeUrl(String orderNo, String codeUrl);
```

实现：

```
/**
     * 存储订单二维码
     * @param orderNo
     * @param codeUrl
     */
@Override
public void saveCodeUrl(String orderNo, String codeUrl) {

    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("order_no", orderNo);

    OrderInfo orderInfo = new OrderInfo();
    orderInfo.setCodeUrl(codeUrl);

    baseMapper.update(orderInfo, queryWrapper);
}
```

（3）修改WxPayServiceImpl 的 nativePay 方法

```
@Resource
private OrderInfoService orderInfoService;

/**
 * 创建订单，调用Native支付接口
```

```
     * @param productId
     * @return code_url 和 订单号
     * @throws Exception
     */
    @Override
    public Map<String, Object> nativePay(Long productId) throws Exception {

        log.info("生成订单");

        //生成订单
        OrderInfo orderInfo = orderInfoService.createOrderByProductId(productId);
        String codeUrl = orderInfo.getCodeUrl();
        if(orderInfo != null && !StringUtils.isEmpty(codeUrl)){
            log.info("订单已存在，二维码已保存");
            //返回二维码
            Map<String, Object> map = new HashMap<>();
            map.put("codeUrl", codeUrl);
            map.put("orderNo", orderInfo.getOrderNo());
            return map;
        }

        log.info("调用统一下单API");

        //其他代码。。。。。。


        try {

            //其他代码。。。。。。


            //保存二维码
            String orderNo = orderInfo.getOrderNo();
            orderInfoService.saveCodeUrl(orderNo, codeUrl);

            //返回二维码
            //其他代码。。。。。。


        } finally {
            response.close();
        }
    }
```

5.5、显示订单列表

在我的订单页面按时间倒序显示订单列表

（1）创建OrderInfoController

```
package com.atguigu.paymentdemo.controller;

import com.atguigu.paymentdemo.entity.OrderInfo;
import com.atguigu.paymentdemo.service.OrderInfoService;
import com.atguigu.paymentdemo.vo.R;
```

```
import io.swagger.annotations.Api;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;
import java.util.List;

@CrossOrigin //开放前端的跨域访问
@Api(tags = "商品订单管理")
@RestController
@RequestMapping("/api/order-info")
public class OrderInfoController {

    @Resource
    private OrderInfoService orderInfoService;

    @ApiOperation("订单列表")
    @GetMapping("/list")
    public R list(){

        List<OrderInfo> list = orderInfoService.listOrderByCreateTimeDesc();
        return R.ok().data("list", list);
    }
}
```

（2）定义 OrderInfoService 方法

接口

```
List<OrderInfo> listOrderByCreateTimeDesc();
```

实现

```
/**
 * 查询订单列表，并倒序查询
 * @return
 */
@Override
public List<OrderInfo> listOrderByCreateTimeDesc() {

    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<OrderInfo>
().orderByDesc("create_time");
    return baseMapper.selectList(queryWrapper);
}
```

6、支付通知API

6.1、内网穿透

（1）访问ngrok官网

https://ngrok.com/

（2）注册账号、登录

（3）下载内网穿透工具

ngrok-stable-windows-amd64.zip

（4）设置你的 authToken

为本地计算机做授权配置

```
ngrok authtoken 6aYc6Kp7kpxVr8pY88LkG_6x9o18yMY8BASrXiDFMeS
```

（5）启动服务

```
ngrok http 8090
```

（6）测试外网访问

```
你获得的外网地址/api/test
```

6.2、接收通知和返回应答

支付通知API：https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_5.shtml

（1）启动ngrok

```
ngrok http 8090
```

（2）设置通知地址

wxpay.properties

> 注意：每次重新启动ngrok，都需要根据实际情况修改这个配置

```
wxpay.notify-domain=https://7d92-115-171-63-135.ngrok.io
```

（3）创建通知接口

> 通知规则：用户支付完成后，微信会把相关支付结果和用户信息发送给商户，商户需要接收处理该消息，并返回应答。对后台通知交互时，如果微信收到商户的应答不符合规范或超时，微信认为通知失败，微信会通过一定的策略定期重新发起通知，尽可能提高通知的成功率，但微信不保证通知最终能成功。（通知频率为 15s/15s/30s/3m/10m/20m/30m/30m/30m/60m/3h/3h/3h/6h/6h - 总计 24h4m）

```
/**
     * 支付通知
     * 微信支付通过支付通知接口将用户支付成功消息通知给商户
     */
@ApiOperation("支付通知")
@PostMapping("/native/notify")
```

```
public String nativeNotify(HttpServletRequest request, HttpServletResponse
response){

    Gson gson = new Gson();
    Map<String, String> map = new HashMap<>();//应答对象

    //处理通知参数
    String body = HttpUtils.readData(request);
    Map<String, Object> bodyMap = gson.fromJson(body, HashMap.class);
    log.info("支付通知的id ===> {}", bodyMap.get("id"));
    log.info("支付通知的完整数据 ===> {}", body);

    //TODO : 签名的验证
    //TODO : 处理订单

    //成功应答：成功应答必须为200或204，否则就是失败应答
    response.setStatus(200);
    map.put("code", "SUCCESS");
    map.put("message", "成功");
    return gson.toJson(map);

}
```

（4）测试失败应答

用失败应答替换成功应答

```
@PostMapping("/native/notify")
public String nativeNotify(HttpServletRequest request, HttpServletResponse
response) throws Exception {

    Gson gson = new Gson();
    Map<String, String> map = new HashMap<>();

    try {

    } catch (Exception e) {

        e.printStackTrace();
        // 测试错误应答
        response.setStatus(500);
        map.put("code", "ERROR");
        map.put("message", "系统错误");
        return gson.toJson(map);
    }
}
```

（5）测试超时应答

回调通知注意事项：https://pay.weixin.qq.com/wiki/doc/apiv3/Practices/chapter1_1_5.shtml

> 商户系统收到支付结果通知，需要在`5秒内`返回应答报文，否则微信支付认为通知失败，后续会重复发送通知。

```
// 测试超时应答：添加睡眠时间使应答超时
TimeUnit.SECONDS.sleep(5);
```

6.3、验签

（1）工具类

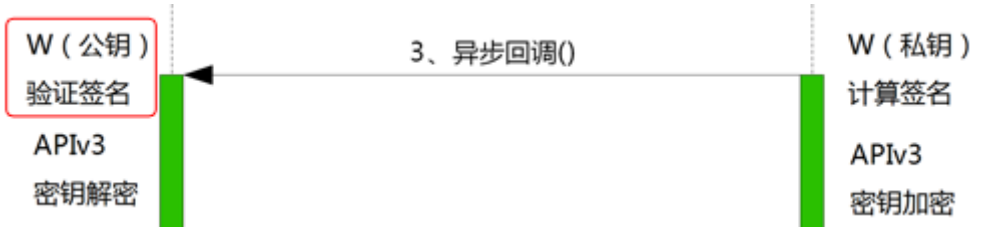
参考SDK源码中的 WechatPay2Validator 创建通知验签工具类 WechatPay2ValidatorForRequest

（2）验签

```
@Resource
private Verifier verifier;
```

```
//签名的验证
WechatPay2ValidatorForRequest validator
    = new WechatPay2ValidatorForRequest(verifier, body, requestId);
if (!validator.validate(request)) {
    log.error("通知验签失败");
    //失败应答
    response.setStatus(500);
    map.put("code", "ERROR");
    map.put("message", "通知验签失败");
    return gson.toJson(map);
}

log.info("通知验签成功");
//TODO : 处理订单
```

6.4、解密



参数解密

下面详细描述对通知数据进行解密的流程：

1、用商户平台上设置的APIv3密钥【微信商户平台—>账户设置—>API安全—>设置APIv3密钥】，记为key；

2、针对resource.algorithm中描述的算法（目前为AEAD_AES_256_GCM），取得对应的参数nonce和associated_data；

3、使用key、nonce和associated_data，对数据密文resource.ciphertext进行解密，得到JSON形式的资源对象；

证书和回调解密需要的AesGcm解密在哪里?

请参考AesUtil.Java。

(1) WxPayController

nativeNotify 方法中添加处理订单的代码

```
//处理订单
wxPayService.processOrder(bodyMap);
```

(1) WxPayService

接口:

```
void processOrder(Map<String, Object> bodyMap) throws GeneralSecurityException;
```

实现:

```
@Override
public void processOrder(Map<String, Object> bodyMap) throws
GeneralSecurityException {
    log.info("处理订单");

    String plainText = decryptFromResource(bodyMap);

    //转换明文

    //更新订单状态

    //记录支付日志
}
```

辅助方法:

```
/**
     * 对称解密
     * @param bodyMap
     * @return
     */
private String decryptFromResource(Map<String, Object> bodyMap) throws
GeneralSecurityException {

    log.info("密文解密");

    //通知数据
    Map<String, String> resourceMap = (Map) bodyMap.get("resource");
    //数据密文
    String ciphertext = resourceMap.get("ciphertext");
    //随机串
    String nonce = resourceMap.get("nonce");
    //附加数据
    String associatedData = resourceMap.get("associated_data");
```

```
    log.info("密文 ===> {}", ciphertext);
    AesUtil aesUtil = new
AesUtil(wxPayConfig.getApiV3Key().getBytes(StandardCharsets.UTF_8));
    String plainText =
aesUtil.decryptToString(associatedData.getBytes(StandardCharsets.UTF_8),

nonce.getBytes(StandardCharsets.UTF_8),
                                                    ciphertext);

    log.info("明文 ===> {}", plainText);

    return plainText;
}
```

6.5、处理订单

（1）完善processOrder方法

```
@Resource
private PaymentInfoService paymentInfoService;

@Override
public void processOrder(Map<String, Object> bodyMap) throws
GeneralSecurityException {
    log.info("处理订单");

    String plainText = decryptFromResource(bodyMap);

    //转换明文
    Gson gson = new Gson();
    Map<String, Object> plainTextMap = gson.fromJson(plainText, HashMap.class);
    String orderNo = (String)plainTextMap.get("out_trade_no");

    //更新订单状态
    orderInfoService.updateStatusByOrderNo(orderNo, OrderStatus.SUCCESS);

    //记录支付日志
    paymentInfoService.createPaymentInfo(plainText);
}
```

（2）更新订单状态

OrderInfoService

接口：

```
void updateStatusByOrderNo(String orderNo, OrderStatus orderStatus);
```

实现：

```
/**
 * 根据订单编号更新订单状态
 * @param orderNo
 * @param orderStatus
```

```
     */
    @Override
    public void updateStatusByOrderNo(String orderNo, OrderStatus orderStatus) {

        log.info("更新订单状态 ===> {}", orderStatus.getType());

        QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
        queryWrapper.eq("order_no", orderNo);

        OrderInfo orderInfo = new OrderInfo();
        orderInfo.setOrderStatus(orderStatus.getType());

        baseMapper.update(orderInfo, queryWrapper);
    }
```

（3）处理支付日志

PaymentInfoService

接口：

```
void createPaymentInfo(String plainText);
```

实现：

```
/**
     * 记录支付日志
     * @param plainText
     */
    @Override
    public void createPaymentInfo(String plainText) {

        log.info("记录支付日志");

        Gson gson = new Gson();
        Map<String, Object> plainTextMap = gson.fromJson(plainText, HashMap.class);

        String orderNo = (String)plainTextMap.get("out_trade_no");
        String transactionId = (String)plainTextMap.get("transaction_id");
        String tradeType = (String)plainTextMap.get("trade_type");
        String tradeState = (String)plainTextMap.get("trade_state");
        Map<String, Object> amount = (Map)plainTextMap.get("amount");
        Integer payerTotal = ((Double) amount.get("payer_total")).intValue();

        PaymentInfo paymentInfo = new PaymentInfo();
        paymentInfo.setOrderNo(orderNo);
        paymentInfo.setPaymentType(PayType.WXPAY.getType());
        paymentInfo.setTransactionId(transactionId);
        paymentInfo.setTradeType(tradeType);
        paymentInfo.setTradeState(tradeState);
        paymentInfo.setPayerTotal(payerTotal);
        paymentInfo.setContent(plainText);

        baseMapper.insert(paymentInfo);
    }
```

6.6、处理重复通知

> **注意：**
> • 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。 推荐的做法是，当商户系统收到通知进行处理时，先检查对应业务数据的状态，并判断该通知是否已经处理。如果未处理，则再进行处理；如果已处理，则直接返回结果成功。在对业务数据进行状态检查和处理之前，要采用数据锁进行并发控制，以避免函数重入造成的数据混乱。
> • 如果在所有通知频率后没有收到微信侧回调，商户应调用查询订单接口确认订单状态。
>
> **特别提醒：**商户系统对于开启结果通知的内容一定要做签名验证，并校验通知的信息是否与商户侧的信息一致，防止数据泄露导致出现“假通知”，造成资金损失。

（1）测试重复的通知

```
//应答超时
//设置响应超时，可以接收到微信支付的重复的支付结果通知。
//通知重复，数据库会记录多余的支付日志
TimeUnit.SECONDS.sleep(5);
```

（2）处理重复通知

在 processOrder 方法中，更新订单状态之前，添加如下代码

```
//处理重复通知
//保证接口调用的幂等性：无论接口被调用多少次，产生的结果是一致的
String orderStatus = orderInfoService.getOrderStatus(orderNo);
if (!OrderStatus.NOTPAY.getType().equals(orderStatus)) {
    return;
}
```

OrderInfoService

接口：

```
String getOrderStatus(String orderNo);
```

实现：

```
/**
     * 根据订单号获取订单状态
     * @param orderNo
     * @return
     */
@Override
public String getOrderStatus(String orderNo) {

    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("order_no", orderNo);
    OrderInfo orderInfo = baseMapper.selectOne(queryWrapper);
```

```
    //防止被删除的订单的回调通知的调用
    if(orderInfo == null){
        return null;
    }
    return orderInfo.getOrderStatus();
}
```

6.7、数据锁

> **注意：**
> • 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。 推荐的做法是，当商户系统收到通知进行处理时，先检查对应业务数据的状态，并判断该通知是否已经处理。如果未处理，则再进行处理；如果已处理，则直接返回结果成功。在对业务数据进行状态检查和处理之前，要采用数据锁进行并发控制，以避免函数重入造成的数据混乱。
> • 如果在所有通知频率后没有收到微信侧回调，商户应调用查询订单接口确认订单状态。
>
> **特别提醒：**商户系统对于开启结果通知的内容一定要做签名验证，并校验通知的信息是否与商户侧的信息一致，防止数据泄露导致出现“假通知”，造成资金损失。

（1）测试通知并发

```
//处理重复的通知

//模拟通知并发
try {
    TimeUnit.SECONDS.sleep(5);
} catch (InterruptedException e) {
    e.printStackTrace();
}

//更新订单状态
//记录支付日志
```

（2）定义ReentrantLock

定义 ReentrantLock 进行并发控制。注意，必须手动释放锁。

```
private final ReentrantLock lock = new ReentrantLock();
```

```
@Override
public void processOrder(Map<String, Object> bodyMap) throws
GeneralSecurityException {
    log.info("处理订单");

    //解密报文
    String plainText = decryptFromResource(bodyMap);

    //将明文转换成map
    Gson gson = new Gson();
```

```
    HashMap plainTextMap = gson.fromJson(plainText, HashMap.class);
    String orderNo = (String)plainTextMap.get("out_trade_no");


    /*在对业务数据进行状态检查和处理之前，
    要采用数据锁进行并发控制，
    以避免函数重入造成的数据混乱*/
    //尝试获取锁：
    // 成功获取则立即返回true，获取失败则立即返回false。不必一直等待锁的释放
    if(lock.tryLock()){
        try {
            //处理重复的通知
            //接口调用的幂等性：无论接口被调用多少次，产生的结果是一致的。
            String orderStatus = orderInfoService.getOrderStatus(orderNo);
            if(!OrderStatus.NOTPAY.getType().equals(orderStatus)){
                return;
            }

            //模拟通知并发
            try {
                TimeUnit.SECONDS.sleep(5);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            //更新订单状态
            orderInfoService.updateStatusByOrderNo(orderNo,
OrderStatus.SUCCESS);

            //记录支付日志
            paymentInfoService.createPaymentInfo(plainText);
        } finally {
            //要主动释放锁
            lock.unlock();
        }
    }
}
```

7、商户定时查询本地订单

7.1、后端定义商户查单接口

支付成功后，商户侧查询本地数据库，订单是否支付成功

```
/**
 * 查询本地订单状态
 */
@ApiOperation("查询本地订单状态")
@GetMapping("/query-order-status/{orderNo}")
public R queryOrderStatus(@PathVariable String orderNo) {

    String orderStatus = orderInfoService.getOrderStatus(orderNo);
    if (OrderStatus.SUCCESS.getType().equals(orderStatus)) {//支付成功
        return R.ok();
    }
    return R.ok().setCode(101).setMessage("支付中...");
}
```

7.2、前端定时轮询查单

在二维码展示页面，前端定时轮询查询订单是否已支付，如果支付成功则跳转到订单页面

（1）定义定时器

```
//启动定时器
this.timer = setInterval(() => {
    //查询订单是否支付成功
    this.queryOrderStatus()
}, 3000)
```

（2）查询订单

```
// 查询订单状态
queryOrderStatus() {

    orderInfoApi.queryOrderStatus(this.orderNo).then(response => {
        console.log('查询订单状态：' + response.code)

        // 支付成功后的页面跳转
        if (response.code === 0) {
            console.log('清除定时器')
            clearInterval(this.timer)
            // 三秒后跳转到订单列表
            setTimeout(() => {
                this.$router.push({ path: '/success' })
            }, 3000)
        }
    })
}
```

8、用户取消订单API

实现用户主动取消订单的功能

8.1、定义取消订单接口

WxPayController中添加接口方法

```
/**
     * 用户取消订单
     * @param orderNo
     * @return
     * @throws Exception
     */
@ApiOperation("用户取消订单")
@PostMapping("/cancel/{orderNo}")
public R cancel(@PathVariable String orderNo) throws Exception {

    log.info("取消订单");

    wxPayService.cancelOrder(orderNo);
    return R.ok().setMessage("订单已取消");
}
```

8.2、WxPayService

接口

```
void cancelOrder(String orderNo) throws Exception;
```

实现

```
/**
     * 用户取消订单
     * @param orderNo
     */
@Override
public void cancelOrder(String orderNo) throws Exception {

    //调用微信支付的关单接口
    this.closeOrder(orderNo);

    //更新商户端的订单状态
    orderInfoService.updateStatusByOrderNo(orderNo, OrderStatus.CANCEL);
}
```

关单方法

```
/**
     * 关单接口的调用
     * @param orderNo
     */
private void closeOrder(String orderNo) throws Exception {

    log.info("关单接口的调用，订单号 ===> {}", orderNo);

    //创建远程请求对象
    String url = String.format(WxApiType.CLOSE_ORDER_BY_NO.getType(), orderNo);
    url = wxPayConfig.getDomain().concat(url);
    HttpPost httpPost = new HttpPost(url);

    //组装json请求体
    Gson gson = new Gson();
```

```
        Map<String, String> paramsMap = new HashMap<>();
        paramsMap.put("mchid", wxPayConfig.getMchId());
        String jsonParams = gson.toJson(paramsMap);
        log.info("请求参数 ===> {}", jsonParams);

        //将请求参数设置到请求对象中
        StringEntity entity = new StringEntity(jsonParams,"utf-8");
        entity.setContentType("application/json");
        httpPost.setEntity(entity);
        httpPost.setHeader("Accept", "application/json");

        //完成签名并执行请求
        CloseableHttpResponse response = wxPayClient.execute(httpPost);

        try {
            int statusCode = response.getStatusLine().getStatusCode();//响应状态码
            if (statusCode == 200) { //处理成功
                log.info("成功200");
            } else if (statusCode == 204) { //处理成功，无返回Body
                log.info("成功204");
            } else {
                log.info("Native下单失败,响应码 = " + statusCode);
                throw new IOException("request failed");
            }

        } finally {
            response.close();
        }
    }
```

9、微信支付查单API

9.1、查单接口的调用

商户后台未收到异步支付结果通知时，商户应该主动调用《微信支付查单接口》，同步订单状态。

（1）WxPayController

```
/**
     * 查询订单
     * @param orderNo
     * @return
     * @throws URISyntaxException
     * @throws IOException
     */
@ApiOperation("查询订单：测试订单状态用")
@GetMapping("query/{orderNo}")
public R queryOrder(@PathVariable String orderNo) throws Exception {

    log.info("查询订单");

    String bodyAsString = wxPayService.queryOrder(orderNo);
    return R.ok().setMessage("查询成功").data("bodyAsString", bodyAsString);
}
```

（2）WxPayService

接口

```
String queryOrder(String orderNo) throws Exception;
```

实现

```
/**
     * 查单接口调用
     */
@Override
public String queryOrder(String orderNo) throws Exception {

    log.info("查单接口调用 ===> {}", orderNo);

    String url = String.format(WxApiType.ORDER_QUERY_BY_NO.getType(), orderNo);
    url = wxPayConfig.getDomain().concat(url).concat("?
mchid=").concat(wxPayConfig.getMchId());

    HttpGet httpGet = new HttpGet(url);
    httpGet.setHeader("Accept", "application/json");

    //完成签名并执行请求
    CloseableHttpResponse response = wxPayClient.execute(httpGet);

    try {
        String bodyAsString = EntityUtils.toString(response.getEntity());//响应体
        int statusCode = response.getStatusLine().getStatusCode();//响应状态码
        if (statusCode == 200) { //处理成功
            log.info("成功, 返回结果 = " + bodyAsString);
        } else if (statusCode == 204) { //处理成功，无返回Body
            log.info("成功");
        } else {
            log.info("Native下单失败,响应码 = " + statusCode+ ",返回结果 = " +
bodyAsString);
            throw new IOException("request failed");
        }

        return bodyAsString;

    } finally {
        response.close();
    }
}
```

9.2、集成Spring Task

Spring 3.0后提供Spring Task实现任务调度

（1）启动类添加注解

statistics启动类添加注解

```
@EnableScheduling
```

（2）测试定时任务

创建 task 包，创建 WxPayTask.java

```
package com.atguigu.paymentdemo.task;

import lombok.extern.slf4j.Slf4j;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Slf4j
@Component
public class WxPayTask {

    /**
     * 测试
     * (cron="秒 分 时 日 月 周")
     * *：每隔一秒执行
     * 0/3：从第0秒开始，每隔3秒执行一次
     * 1-3：从第1秒开始执行，到第3秒结束执行
     * 1,2,3：第1、2、3秒执行
     * ?：不指定，若指定日期，则不指定周，反之同理
     */
    @Scheduled(cron="0/3 * * * * ?")
    public void task1() {
        log.info("task1 执行");
    }
}
```

9.3、定时查找超时订单

（1）WxPayTask

```
@Resource
private OrderInfoService orderInfoService;

@Resource
private WxPayService wxPayService;

/**
     * 从第0秒开始每隔30秒执行1次，查询创建超过5分钟，并且未支付的订单
     */
@Scheduled(cron = "0/30 * * * * ?")
public void orderConfirm() throws Exception {
    log.info("orderConfirm 被执行......");

    List<OrderInfo> orderInfoList = orderInfoService.getNoPayOrderByDuration(5);

    for (OrderInfo orderInfo : orderInfoList) {
        String orderNo = orderInfo.getOrderNo();
        log.warn("超时订单 ===> {}", orderNo);
```

```
        //核实订单状态：调用微信支付查单接口
        wxPayService.checkOrderStatus(orderNo);
    }
}
```

(2) OrderInfoService

接口

```
List<OrderInfo> getNoPayOrderByDuration(int minutes);
```

实现

```
/**
 * 找出创建超过minutes分钟并且未支付的订单
 * @param minutes
 * @return
 */
@Override
public List<OrderInfo> getNoPayOrderByDuration(int minutes) {

    //minutes分钟之前的时间
    Instant instant = Instant.now().minus(Duration.ofMinutes(minutes));

    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("order_status", OrderStatus.NOTPAY.getType());
    queryWrapper.le("create_time", instant);
    List<OrderInfo> orderInfoList = baseMapper.selectList(queryWrapper);
    return orderInfoList;
}
```

9.4、处理超时订单

WxPayService

核实订单状态

接口:

```
void checkOrderStatus(String orderNo) throws Exception;
```

实现:

```
/**
     * 根据订单号查询微信支付查单接口，核实订单状态
     * 如果订单已支付，则更新商户端订单状态，并记录支付日志
     * 如果订单未支付，则调用关单接口关闭订单，并更新商户端订单状态
     * @param orderNo
     */
@Override
public void checkOrderStatus(String orderNo) throws Exception {

    log.warn("根据订单号核实订单状态 ===> {}", orderNo);
```

```
    //调用微信支付查单接口
    String result = this.queryOrder(orderNo);

    Gson gson = new Gson();
    Map resultMap = gson.fromJson(result, HashMap.class);

    //获取微信支付端的订单状态
    Object tradeState = resultMap.get("trade_state");

    //判断订单状态
    if(WxTradeState.SUCCESS.getType().equals(tradeState)){

        log.warn("核实订单已支付 ===> {}", orderNo);

        //如果确认订单已支付则更新本地订单状态
        orderInfoService.updateStatusByOrderNo(orderNo, OrderStatus.SUCCESS);
        //记录支付日志
        paymentInfoService.createPaymentInfo(result);
    }

    if(WxTradeState.NOTPAY.getType().equals(tradeState)){
        log.warn("核实订单未支付 ===> {}", orderNo);

        //如果订单未支付，则调用关单接口
        this.closeOrder(orderNo);

        //更新本地订单状态
        orderInfoService.updateStatusByOrderNo(orderNo, OrderStatus.CLOSED);
    }

}
```

6.8

接口：

```
OrderInfo getOrderByOrderNo(String orderNo);
```

实现：

```
/**
 * 根据订单号获取订单
 * @param orderNo
 * @return
 */
@Override
public OrderInfo getOrderByOrderNo(String orderNo) {
    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("order_no", orderNo);
    OrderInfo orderInfo = baseMapper.selectOne(queryWrapper);

    return orderInfo;
}
```

11、申请退款API

文档：https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_9.shtml

11.1、创建退款单

（1）根据订单号查询订单

OrderInfoService

接口：

```
OrderInfo getOrderByOrderNo(String orderNo);
```

实现：

```
/**
 * 根据订单号获取订单
 * @param orderNo
 * @return
 */
@Override
public OrderInfo getOrderByOrderNo(String orderNo) {

    QueryWrapper<OrderInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("order_no", orderNo);
    OrderInfo orderInfo = baseMapper.selectOne(queryWrapper);

    return orderInfo;
}
```

（2）创建退款单记录

RefundsInfoService

接口：

```
RefundInfo createRefundByOrderNo(String orderNo, String reason);
```

实现：

```
@Resource
private OrderInfoService orderInfoService;

/**
     * 根据订单号创建退款订单
     * @param orderNo
     * @return
     */
@Override
public RefundInfo createRefundByOrderNo(String orderNo, String reason) {

    //根据订单号获取订单信息
    OrderInfo orderInfo = orderInfoService.getOrderByOrderNo(orderNo);

    //根据订单号生成退款订单
    RefundInfo refundInfo = new RefundInfo();
    refundInfo.setOrderNo(orderNo);//订单编号
    refundInfo.setRefundNo(OrderNoUtils.getRefundNo());//退款单编号
    refundInfo.setTotalFee(orderInfo.getTotalFee());//原订单金额(分)
    refundInfo.setRefund(orderInfo.getTotalFee());//退款金额(分)
    refundInfo.setReason(reason);//退款原因

    //保存退款订单
    baseMapper.insert(refundInfo);

    return refundInfo;
}
```

11.2、更新退款单

RefundInfoService

接口：

```
void updateRefund(String content);
```

实现：

```
/**
 * 记录退款记录
 * @param content
 */
@Override
public void updateRefund(String content) {

    //将json字符串转换成Map
    Gson gson = new Gson();
    Map<String, String> resultMap = gson.fromJson(content, HashMap.class);

    //根据退款单编号修改退款单
    QueryWrapper<RefundInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("refund_no", resultMap.get("out_refund_no"));
```

```
    //设置要修改的字段
    RefundInfo refundInfo = new RefundInfo();

    refundInfo.setRefundId(resultMap.get("refund_id"));//微信支付退款单号

    //查询退款和申请退款中的返回参数
    if(resultMap.get("status") != null){
        refundInfo.setRefundStatus(resultMap.get("status"));//退款状态
        refundInfo.setContentReturn(content);//将全部响应结果存入数据库的content字段
    }
    //退款回调中的回调参数
    if(resultMap.get("refund_status") != null){
        refundInfo.setRefundStatus(resultMap.get("refund_status"));//退款状态
        refundInfo.setContentNotify(content);//将全部响应结果存入数据库的content字段
    }

    //更新退款单
    baseMapper.update(refundInfo, queryWrapper);
}
```

11.3、申请退款

(1) WxPayController

```
@ApiOperation("申请退款")
@PostMapping("/refunds/{orderNo}/{reason}")
public R refunds(@PathVariable String orderNo, @PathVariable String reason)
throws Exception {

    log.info("申请退款");
    wxPayService.refund(orderNo, reason);
    return R.ok();
}
```

(2) WxPayService

接口:

```
void refund(String orderNo, String reason) throws Exception;
```

实现:

```
@Resource
private RefundInfoService refundsInfoService;

/**
 * 退款
 * @param orderNo
 * @param reason
 * @throws IOException
 */
@Transactional(rollbackFor = Exception.class)
@Override
public void refund(String orderNo, String reason) throws Exception {
```

```
    log.info("创建退款单记录");
    //根据订单编号创建退款单
    RefundInfo refundsInfo = refundsInfoService.createRefundByOrderNo(orderNo,
reason);

    log.info("调用退款API");

    //调用统一下单API
    String url =
wxPayConfig.getDomain().concat(WxApiType.DOMESTIC_REFUNDS.getType());
    HttpPost httpPost = new HttpPost(url);

    // 请求body参数
    Gson gson = new Gson();
    Map paramsMap = new HashMap();
    paramsMap.put("out_trade_no", orderNo);//订单编号
    paramsMap.put("out_refund_no", refundsInfo.getRefundNo());//退款单编号
    paramsMap.put("reason",reason);//退款原因
    paramsMap.put("notify_url",
wxPayConfig.getNotifyDomain().concat(WxNotifyType.REFUND_NOTIFY.getType()));//退
款通知地址

    Map amountMap = new HashMap();
    amountMap.put("refund", refundsInfo.getRefund());//退款金额
    amountMap.put("total", refundsInfo.getTotalFee());//原订单金额
    amountMap.put("currency", "CNY");//退款币种
    paramsMap.put("amount", amountMap);

    //将参数转换成json字符串
    String jsonParams = gson.toJson(paramsMap);
    log.info("请求参数 ===> {}" + jsonParams);

    StringEntity entity = new StringEntity(jsonParams,"utf-8");
    entity.setContentType("application/json");//设置请求报文格式
    httpPost.setEntity(entity);//将请求报文放入请求对象
    httpPost.setHeader("Accept", "application/json");//设置响应报文格式

    //完成签名并执行请求，并完成验签
    CloseableHttpResponse response = wxPayClient.execute(httpPost);

    try {

        //解析响应结果
        String bodyAsString = EntityUtils.toString(response.getEntity());
        int statusCode = response.getStatusLine().getStatusCode();
        if (statusCode == 200) {
            log.info("成功, 退款返回结果 = " + bodyAsString);
        } else if (statusCode == 204) {
            log.info("成功");
        } else {
            throw new RuntimeException("退款异常, 响应码 = " + statusCode+ ", 退款
返回结果 = " + bodyAsString);
        }

        //更新订单状态
        orderInfoService.updateStatusByOrderNo(orderNo,
OrderStatus.REFUND_PROCESSING);
```

```
            //更新退款单
            refundsInfoService.updateRefund(bodyAsString);

        } finally {
            response.close();
        }
    }
```

12、查询退款API

文档：https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_10.shtml

12.1、查单接口的调用

(1) WxPayController

```
/**
 * 查询退款
 * @param refundNo
 * @return
 * @throws Exception
 */
@ApiOperation("查询退款：测试用")
@GetMapping("/query-refund/{refundNo}")
public R queryRefund(@PathVariable String refundNo) throws Exception {

    log.info("查询退款");

    String result = wxPayService.queryRefund(refundNo);
    return R.ok().setMessage("查询成功").data("result", result);
}
```

(2) WxPayService

接口：

```
String queryRefund(String orderNo) throws Exception;
```

实现：

```
/**
 * 查询退款接口调用
 * @param refundNo
 * @return
 */
@Override
public String queryRefund(String refundNo) throws Exception {

    log.info("查询退款接口调用 ===> {}", refundNo);

    String url =  String.format(WxApiType.DOMESTIC_REFUNDS_QUERY.getType(),
refundNo);
```

```
        url = wxPayConfig.getDomain().concat(url);

        //创建远程Get 请求对象
        HttpGet httpGet = new HttpGet(url);
        httpGet.setHeader("Accept", "application/json");

        //完成签名并执行请求
        CloseableHttpResponse response = wxPayClient.execute(httpGet);

        try {
            String bodyAsString = EntityUtils.toString(response.getEntity());
            int statusCode = response.getStatusLine().getStatusCode();
            if (statusCode == 200) {
                log.info("成功, 查询退款返回结果 = " + bodyAsString);
            } else if (statusCode == 204) {
                log.info("成功");
            } else {
                throw new RuntimeException("查询退款异常, 响应码 = " + statusCode+ ",
查询退款返回结果 = " + bodyAsString);
            }

            return bodyAsString;

        } finally {
            response.close();
        }
    }
```

12.2、定时查找退款中的订单

(1) WxPayTask

```
/**
 * 从第0秒开始每隔30秒执行1次，查询创建超过5分钟，并且未成功的退款单
 */
@Scheduled(cron = "0/30 * * * * ?")
public void refundConfirm() throws Exception {
    log.info("refundConfirm 被执行.....");

    //找出申请退款超过5分钟并且未成功的退款单
    List<RefundInfo> refundInfoList =
refundInfoService.getNoRefundOrderByDuration(5);

    for (RefundInfo refundInfo : refundInfoList) {
        String refundNo = refundInfo.getRefundNo();
        log.warn("超时未退款的退款单号 ===> {}", refundNo);

        //核实订单状态：调用微信支付查询退款接口
        wxPayService.checkRefundStatus(refundNo);
    }
}
```

(2) RefundInfoService

接口

```
List<RefundInfo> getNoRefundOrderByDuration(int minutes);
```

实现

```
/**
     * 找出申请退款超过minutes分钟并且未成功的退款单
     * @param minutes
     * @return
     */
@Override
public List<RefundInfo> getNoRefundOrderByDuration(int minutes) {

    //minutes分钟之前的时间
    Instant instant = Instant.now().minus(Duration.ofMinutes(minutes));

    QueryWrapper<RefundInfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("refund_status", WxRefundStatus.PROCESSING.getType());
    queryWrapper.le("create_time", instant);
    List<RefundInfo> refundInfoList = baseMapper.selectList(queryWrapper);
    return refundInfoList;
}
```

12.3、处理超时未退款订单

WxPayService

核实订单状态

接口：

```
void checkRefundStatus(String refundNo);
```

实现：

```
/**
     * 根据退款单号核实退款单状态
     * @param refundNo
     * @return
     */
@Transactional(rollbackFor = Exception.class)
@Override
public void checkRefundStatus(String refundNo) throws Exception {

    log.warn("根据退款单号核实退款单状态 ===> {}", refundNo);

    //调用查询退款单接口
    String result = this.queryRefund(refundNo);

    //组装json请求体字符串
    Gson gson = new Gson();
    Map<String, String> resultMap = gson.fromJson(result, HashMap.class);

    //获取微信支付端退款状态
```

```
    String status = resultMap.get("status");

    String orderNo = resultMap.get("out_trade_no");

    if (WxRefundStatus.SUCCESS.getType().equals(status)) {

        log.warn("核实订单已退款成功 ===> {}", refundNo);

        //如果确认退款成功，则更新订单状态
        orderInfoService.updateStatusByOrderNo(orderNo,
OrderStatus.REFUND_SUCCESS);

        //更新退款单
        refundsInfoService.updateRefund(result);
    }

    if (WxRefundStatus.ABNORMAL.getType().equals(status)) {

        log.warn("核实订单退款异常  ===> {}", refundNo);

        //如果确认退款成功，则更新订单状态
        orderInfoService.updateStatusByOrderNo(orderNo,
OrderStatus.REFUND_ABNORMAL);

        //更新退款单
        refundsInfoService.updateRefund(result);
    }
}
```

13、退款结果通知API

文档：https://pay.weixin.qq.com/wiki/doc/apiv3/apis/chapter3_4_11.shtml

13.1、接收退款通知

WxPayController

```
/**
     * 退款结果通知
     * 退款状态改变后，微信会把相关退款结果发送给商户。
     */
@PostMapping("/refunds/notify")
public String refundsNotify(HttpServletRequest request, HttpServletResponse
response){

    log.info("退款通知执行");
    Gson gson = new Gson();
    Map<String, String> map = new HashMap<>();//应答对象

    try {
        //处理通知参数
        String body = HttpUtils.readData(request);
        Map<String, Object> bodyMap = gson.fromJson(body, HashMap.class);
```

```
            String requestId = (String)bodyMap.get("id");
            log.info("支付通知的id ===> {}", requestId);

            //签名的验证
            WechatPay2ValidatorForRequest wechatPay2ValidatorForRequest
                = new WechatPay2ValidatorForRequest(verifier, requestId, body);
            if(!wechatPay2ValidatorForRequest.validate(request)){

                log.error("通知验签失败");
                //失败应答
                response.setStatus(500);
                map.put("code", "ERROR");
                map.put("message", "通知验签失败");
                return gson.toJson(map);
            }
            log.info("通知验签成功");

            //处理退款单
            wxPayService.processRefund(bodyMap);

            //成功应答
            response.setStatus(200);
            map.put("code", "SUCCESS");
            map.put("message", "成功");
            return gson.toJson(map);

        } catch (Exception e) {
            e.printStackTrace();
            //失败应答
            response.setStatus(500);
            map.put("code", "ERROR");
            map.put("message", "失败");
            return gson.toJson(map);
        }
    }
```

13.2、处理订单和退款单

WxPayService

接口：

```
void processRefund(Map<String, Object> bodyMap) throws Exception;
```

实现：

```
/**
 * 处理退款单
 */
@Transactional(rollbackFor = Exception.class)
@Override
public void processRefund(Map<String, Object> bodyMap) throws Exception {

    log.info("退款单");

    //解密报文
```

```
    String plainText = decryptFromResource(bodyMap);

    //将明文转换成map
    Gson gson = new Gson();
    HashMap plainTextMap = gson.fromJson(plainText, HashMap.class);
    String orderNo = (String)plainTextMap.get("out_trade_no");

    if(lock.tryLock()){
        try {

            String orderStatus = orderInfoService.getOrderStatus(orderNo);
            if (!OrderStatus.REFUND_PROCESSING.getType().equals(orderStatus)) {
                return;
            }

            //更新订单状态
            orderInfoService.updateStatusByOrderNo(orderNo,
OrderStatus.REFUND_SUCCESS);

            //更新退款单
            refundsInfoService.updateRefund(plainText);

        } finally {
            //要主动释放锁
            lock.unlock();
        }
    }
}
```

14、账单

14.1、申请交易账单和资金账单

(1) WxPayController

```
@ApiOperation("获取账单url：测试用")
@GetMapping("/querybill/{billDate}/{type}")
public R queryTradeBill(
        @PathVariable String billDate,
        @PathVariable String type) throws Exception {

    log.info("获取账单url");

    String downloadUrl = wxPayService.queryBill(billDate, type);
    return R.ok().setMessage("获取账单url成功").data("downloadUrl", downloadUrl);
}
```

(2) WxPayService

接口：

```
String queryBill(String billDate, String type) throws Exception;
```

实现

```
/**
     * 申请账单
     * @param billDate
     * @param type
     * @return
     * @throws Exception
     */
@Override
public String queryBill(String billDate, String type) throws Exception {
    log.warn("申请账单接口调用 {}", billDate);

    String url = "";
    if("tradebill".equals(type)){
        url =  WxApiType.TRADE_BILLS.getType();
    }else if("fundflowbill".equals(type)){
        url =  WxApiType.FUND_FLOW_BILLS.getType();
    }else{
        throw new RuntimeException("不支持的账单类型");
    }

    url = wxPayConfig.getDomain().concat(url).concat("?
bill_date=").concat(billDate);

    //创建远程Get 请求对象
    HttpGet httpGet = new HttpGet(url);
    httpGet.addHeader("Accept", "application/json");

    //使用wxPayClient发送请求得到响应
    CloseableHttpResponse response = wxPayClient.execute(httpGet);

    try {

        String bodyAsString = EntityUtils.toString(response.getEntity());

        int statusCode = response.getStatusLine().getStatusCode();
        if (statusCode == 200) {
            log.info("成功, 申请账单返回结果 = " + bodyAsString);
        } else if (statusCode == 204) {
            log.info("成功");
        } else {
            throw new RuntimeException("申请账单异常, 响应码 = " + statusCode+ ",
申请账单返回结果 = " + bodyAsString);
        }

        //获取账单下载地址
        Gson gson = new Gson();
        Map<String, String> resultMap = gson.fromJson(bodyAsString,
HashMap.class);
        return resultMap.get("download_url");

    } finally {
        response.close();
    }
}
```

14.2、下载账单

(1) WxPayController

```
@ApiOperation("下载账单")
@GetMapping("/downloadbill/{billDate}/{type}")
public R downloadBill(
        @PathVariable String billDate,
        @PathVariable String type) throws Exception {

    log.info("下载账单");
    String result = wxPayService.downloadBill(billDate, type);

    return R.ok().data("result", result);
}
```

(2) WxPayService

接口:

```
String downloadBill(String billDate, String type) throws Exception;
```

实现:

```
/**
 * 下载账单
 * @param billDate
 * @param type
 * @return
 * @throws Exception
 */
@Override
public String downloadBill(String billDate, String type) throws Exception {
    log.warn("下载账单接口调用 {}, {}", billDate, type);

    //获取账单url地址
    String downloadUrl = this.queryBill(billDate, type);
    //创建远程Get 请求对象
    HttpGet httpGet = new HttpGet(downloadUrl);
    httpGet.addHeader("Accept", "application/json");

    //使用wxPayClient发送请求得到响应
    CloseableHttpResponse response = wxPayNoSignClient.execute(httpGet);

    try {

        String bodyAsString = EntityUtils.toString(response.getEntity());

        int statusCode = response.getStatusLine().getStatusCode();
        if (statusCode == 200) {
            log.info("成功, 下载账单返回结果 = " + bodyAsString);
        } else if (statusCode == 204) {
            log.info("成功");
        } else {
```

```
            throw new RuntimeException("下载账单异常, 响应码 = " + statusCode+ ",
下载账单返回结果 = " + bodyAsString);
        }

        return bodyAsString;

    } finally {
        response.close();
    }
}
```

五、基础支付API V2

1、V2和V3的比较

接口版本区别

V2版接口和V3版接口实际上是基于两种接口标准设计的两套接口。 目前大部分接口已升级为V3接口，其余V2接口后续也将逐步升级为V3接口。

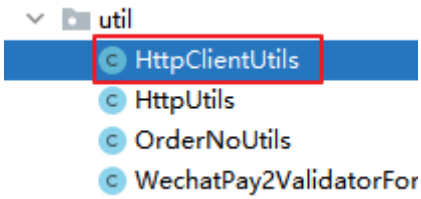
V3	规则差异	V2
JSON	参数格式	XML
POST、GET、PUT、PATCH、DELETE	提交方式	POST
AES-256-GCM加密	回调加密	无需加密
RSA加密	敏感加密	RSA加密
UTF-8	编码方式	UTF-8
非对称密钥SHA256-RSA	签名方式	MD5或HMAC-SHA256

2、引入依赖和工具

2.1、引入依赖

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

2.1、复制工具类



2.3、添加商户APIv2 key

yml文件

```
# APIv2密钥
wxpay.partnerKey: T6m9iK73b0kn9g5v426MKfHQH7X8rKwb
```

WxPayConfig.java

```
private String partnerKey;
```

2.4、添加枚举

enum WxApiType

```
/**
 * Native下单V2
 */
NATIVE_PAY_V2("/pay/unifiedorder"),
```

enum WxNotifyType

```
/**
 * 支付通知V2
 */
NATIVE_NOTIFY_V2("/api/wx-pay-v2/native/notify"),
```

3、统一下单

3.1、创建WxPayV2Controller

```
package com.atguigu.paymentdemo.controller;

import com.atguigu.paymentdemo.service.WxPayService;
import com.atguigu.paymentdemo.vo.R;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import lombok.extern.slf4j.Slf4j;
import org.springframework.web.bind.annotation.*;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
```

```
import java.util.Map;

@CrossOrigin //跨域
@RestController
@RequestMapping("/api/wx-pay-v2")
@Api(tags = "网站微信支付APIv2")
@Slf4j
public class WxPayV2Controller {

    @Resource
    private WxPayService wxPayService;

    /**
     * Native下单
     * @param productId
     * @return
     * @throws Exception
     */
    @ApiOperation("调用统一下单API，生成支付二维码")
    @PostMapping("/native/{productId}")
    public R createNative(@PathVariable Long productId, HttpServletRequest
request) throws Exception {

        log.info("发起支付请求 v2");

        String remoteAddr = request.getRemoteAddr();
        Map<String, Object> map = wxPayService.nativePayV2(productId,
remoteAddr);
        return R.ok().setData(map);
    }

}
```

3.2、WxPayService

接口：

```
Map<String, Object> nativePayV2(Long productId, String remoteAddr) throws
Exception;
```

实现：

```
@Override
public Map<String, Object> nativePayV2(Long productId, String remoteAddr) throws
Exception {

    log.info("生成订单");

    //生成订单
    OrderInfo orderInfo = orderInfoService.createOrderByProductId(productId);
    String codeUrl = orderInfo.getCodeUrl();
    if(orderInfo != null && !StringUtils.isEmpty(codeUrl)){
        log.info("订单已存在，二维码已保存");
        //返回二维码
        Map<String, Object> map = new HashMap<>();
        map.put("codeUrl", codeUrl);
```

```
            map.put("orderNo", orderInfo.getOrderNo());
            return map;
        }

        log.info("调用统一下单API");

        HttpClientUtils client = new
HttpClientUtils("https://api.mch.weixin.qq.com/pay/unifiedorder");

        //组装接口参数
        Map<String, String> params = new HashMap<>();
        params.put("appid", wxPayConfig.getAppid());//关联的公众号的appid
        params.put("mch_id", wxPayConfig.getMchId());//商户号
        params.put("nonce_str", WXPayUtil.generateNonceStr());//生成随机字符串
        params.put("body", orderInfo.getTitle());
        params.put("out_trade_no", orderInfo.getOrderNo());

        //注意，这里必须使用字符串类型的参数（总金额：分）
        String totalFee = orderInfo.getTotalFee() + "";
        params.put("total_fee", totalFee);

        params.put("spbill_create_ip", remoteAddr);
        params.put("notify_url",
wxPayConfig.getNotifyDomain().concat(WxNotifyType.NATIVE_NOTIFY.getType()));
        params.put("trade_type", "NATIVE");

        //将参数转换成xml字符串格式：生成带有签名的xml格式字符串
        String xmlParams = WXPayUtil.generateSignedXml(params,
wxPayConfig.getPartnerKey());
        log.info("\n xmlParams：\n" + xmlParams);

        client.setXmlParam(xmlParams);//将参数放入请求对象的方法体
        client.setHttps(true);//使用https形式发送
        client.post();//发送请求
        String resultXml = client.getContent();//得到响应结果
        log.info("\n resultXml：\n" + resultXml);
        //将xml响应结果转成map对象
        Map<String, String> resultMap = WXPayUtil.xmlToMap(resultXml);

        //错误处理
        if("FAIL".equals(resultMap.get("return_code")) ||
"FAIL".equals(resultMap.get("result_code"))){
            log.error("微信支付统一下单错误 ===> {} ", resultXml);
            throw new RuntimeException("微信支付统一下单错误");
        }

        //二维码
        codeUrl = resultMap.get("code_url");

        //保存二维码
        String orderNo = orderInfo.getOrderNo();
        orderInfoService.saveCodeUrl(orderNo, codeUrl);

        //返回二维码
        Map<String, Object> map = new HashMap<>();
        map.put("codeUrl", codeUrl);
        map.put("orderNo", orderInfo.getOrderNo());
```

```
    return map;
}
```

4、支付回调

```
@Resource
private WxPayService wxPayService;

@Resource
private WxPayConfig wxPayConfig;

@Resource
private OrderInfoService orderInfoService;

@Resource
private PaymentInfoService paymentInfoService;

private final ReentrantLock lock = new ReentrantLock();

/**
     * 支付通知
     * 微信支付通过支付通知接口将用户支付成功消息通知给商户
     */
@PostMapping("/native/notify")
public String wxNotify(HttpServletRequest request) throws Exception {

    System.out.println("微信发送的回调");
    Map<String, String> returnMap = new HashMap<>();//应答对象

    //处理通知参数
    String body = HttpUtils.readData(request);

    //验签
    if(!WXPayUtil.isSignatureValid(body, wxPayConfig.getPartnerKey())) {
        log.error("通知验签失败");
        //失败应答
        returnMap.put("return_code", "FAIL");
        returnMap.put("return_msg", "验签失败");
        String returnXml = WXPayUtil.mapToXml(returnMap);
        return returnXml;
    }

    //解析xml数据
    Map<String, String> notifyMap = WXPayUtil.xmlToMap(body);
    //判断通信和业务是否成功
    if(!"SUCCESS".equals(notifyMap.get("return_code")) ||
!"SUCCESS".equals(notifyMap.get("result_code"))) {
        log.error("失败");
        //失败应答
        returnMap.put("return_code", "FAIL");
        returnMap.put("return_msg", "失败");
        String returnXml = WXPayUtil.mapToXml(returnMap);
        return returnXml;
    }
```

```
    //获取商户订单号
    String orderNo = notifyMap.get("out_trade_no");
    OrderInfo orderInfo = orderInfoService.getOrderByOrderNo(orderNo);
    //并校验返回的订单金额是否与商户侧的订单金额一致
    if (orderInfo != null && orderInfo.getTotalFee() !=
Long.parseLong(notifyMap.get("total_fee"))) {
        log.error("金额校验失败");
        //失败应答
        returnMap.put("return_code", "FAIL");
        returnMap.put("return_msg", "金额校验失败");
        String returnXml = WXPayUtil.mapToXml(returnMap);
        return returnXml;
    }

    //处理订单
    if(lock.tryLock()){
        try {
            //处理重复的通知
            //接口调用的幂等性：无论接口被调用多少次，产生的结果是一致的。
            String orderStatus = orderInfoService.getOrderStatus(orderNo);
            if(OrderStatus.NOTPAY.getType().equals(orderStatus)){
                //更新订单状态
                orderInfoService.updateStatusByOrderNo(orderNo,
OrderStatus.SUCCESS);

                //记录支付日志
                paymentInfoService.createPaymentInfo(body);
            }
        } finally {
            //要主动释放锁
            lock.unlock();
        }
    }

    returnMap.put("return_code", "SUCCESS");
    returnMap.put("return_msg", "OK");
    String returnXml = WXPayUtil.mapToXml(returnMap);
    log.info("支付成功，已应答");
    return returnXml;
}
```