# EGO商城支付

## 模块定位

就是为用户提供一些支付的手段，提供类似银联支付，微信支付，支付宝支付等等的手段！

## 支付说明

我们在众多的支付系统里面，应该优先选择微信支付（因为我们现在时微信的小程序，但是微信支付必须是企业用户，而且你的小程序已经上线2周左右，你才能使用微信支付）

所以我们这里面只能选择支付宝支付了

因为支付宝宝的支付流程和微信支持的流程非常相似，我们会一个，别的道理差不多

## 申请支付宝账号

### 进入支付宝开放平台，放心登录

<https://open.alipay.com/platform/home.htm>



### 进入研发服务



### 记住APPID



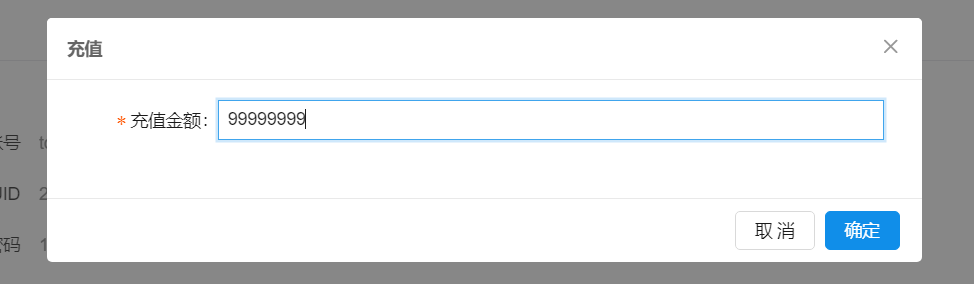
### 下载支付宝测试应用（仅限安卓手机哈）



### 使用测试号登录你下载的支付宝测试应用（你是买家）



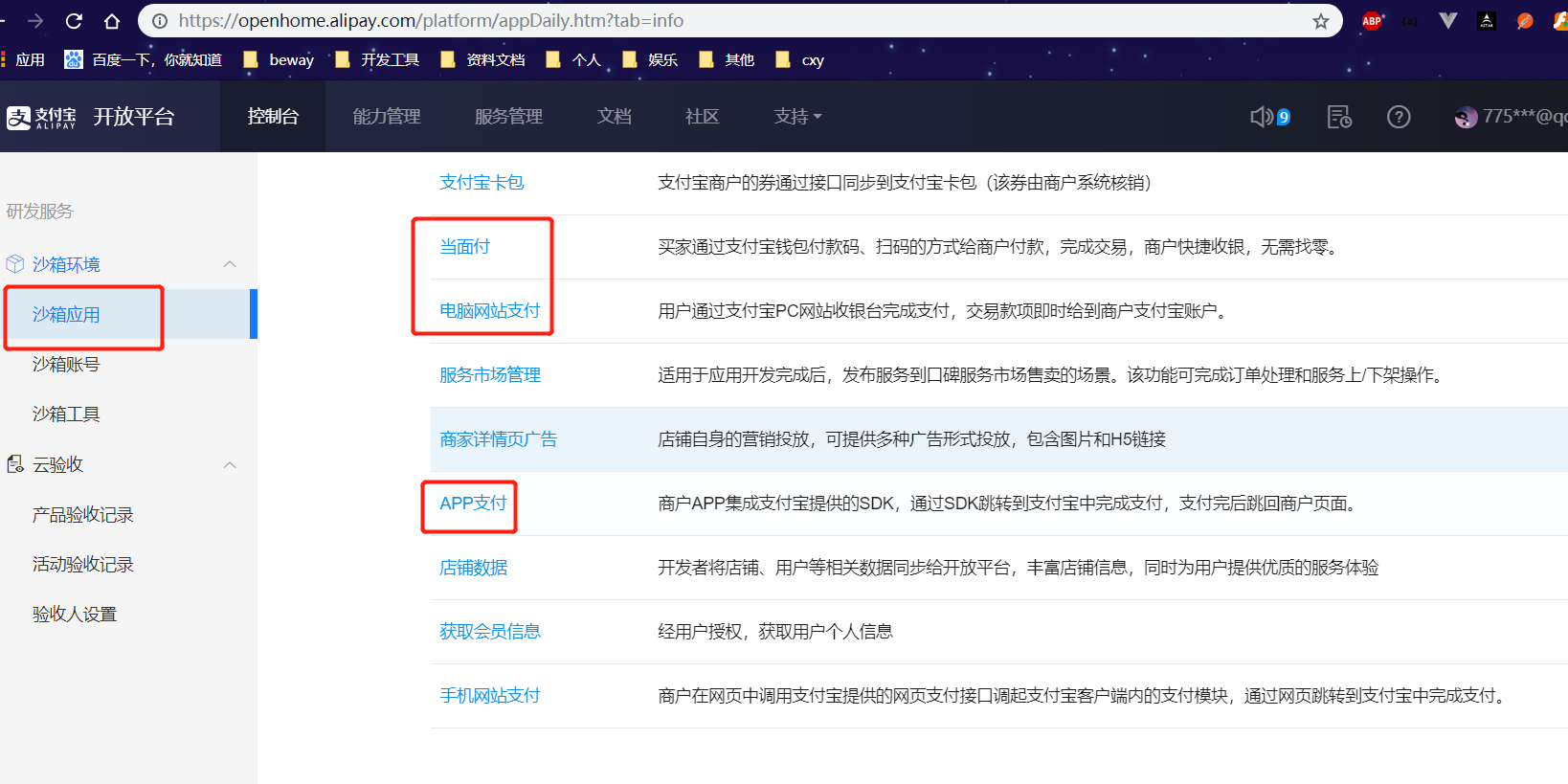
### 记住充值



## 首先了解支付宝支付

### **看支付宝开放平台官方文档**

<https://openhome.alipay.com/platform/appDaily.htm?tab=info>



#### **几种支付方式**

##### 条码支付（商家扫你，使用的多）

商家通过扫你的条形码，直接完成支付（扫描枪）



##### 扫码支付（你扫商家，使用的也多）



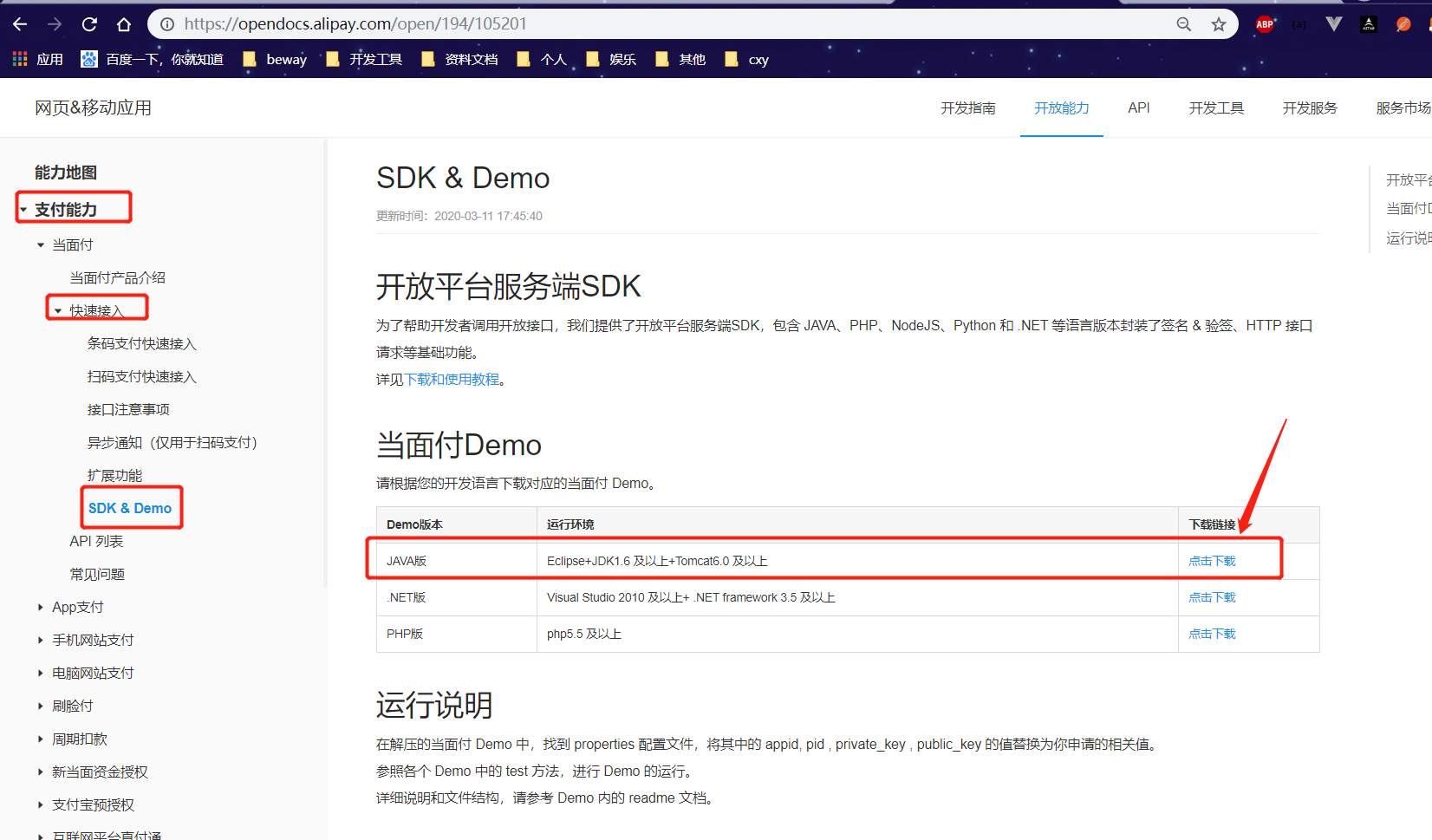
##### 电脑网站支付

在pc端完成的一个支付

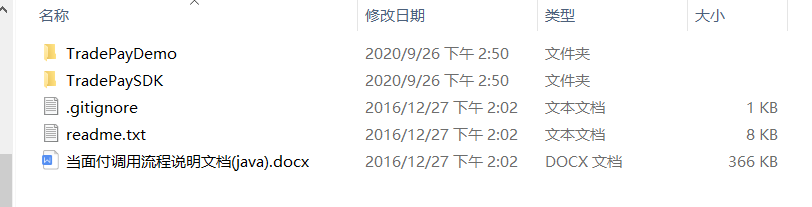
<https://opendocs.alipay.com/open/270>

### 下载扫码支付Demo和SDK学习参考

<https://opendocs.alipay.com/open/194/105201>



#### 解压查看Demo



#### 通过查看Main.java类我们总结得知

**1 读取配置文件zfbinfo.properties**

**2 创建TradeService**

**3 填写一些必须的参数**

**4 发起交易**



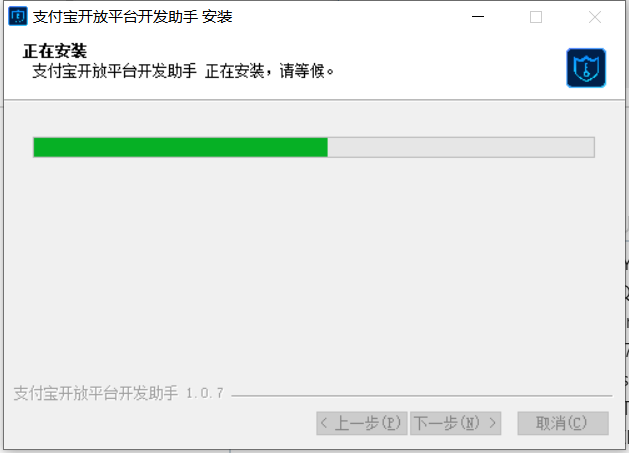
#### 我们还需要RSA2加密工具等

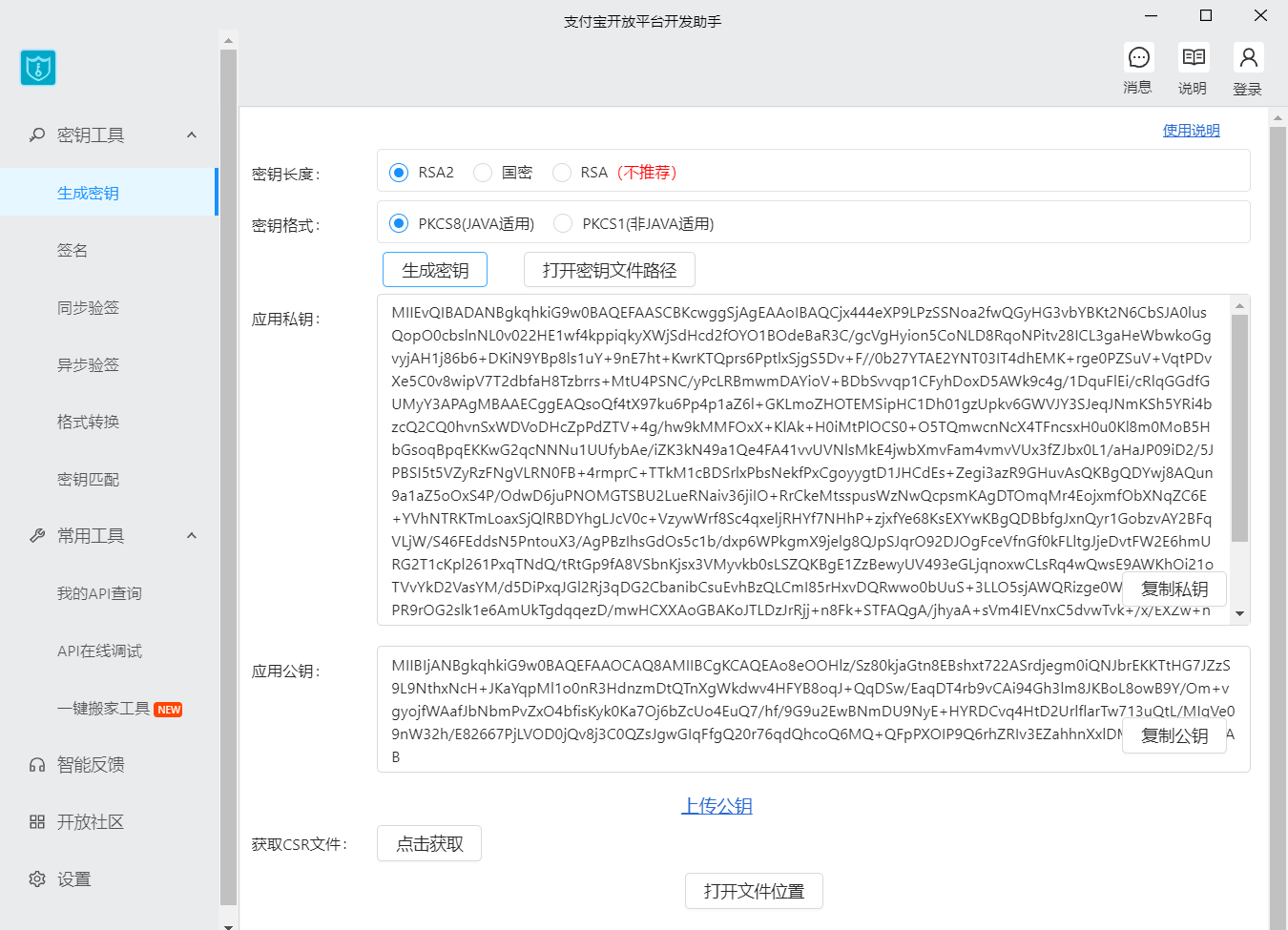




网址是： <https://opendocs.alipay.com/open/00uk9e>

#### 使用加密工具生成私钥和公钥

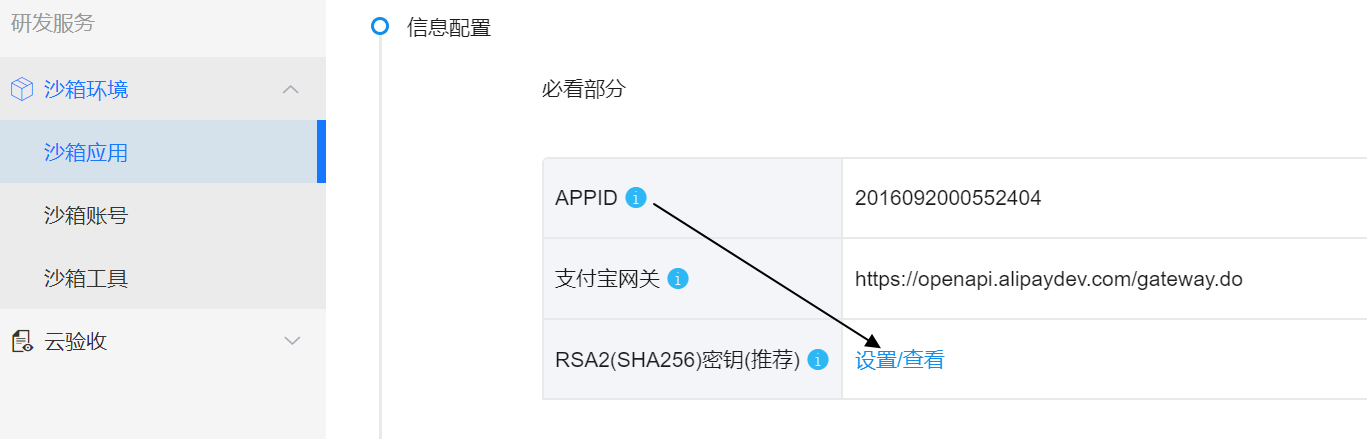




#### 将私钥复制到配置文件中zfbinfo.properties



#### 获得支付宝的公钥



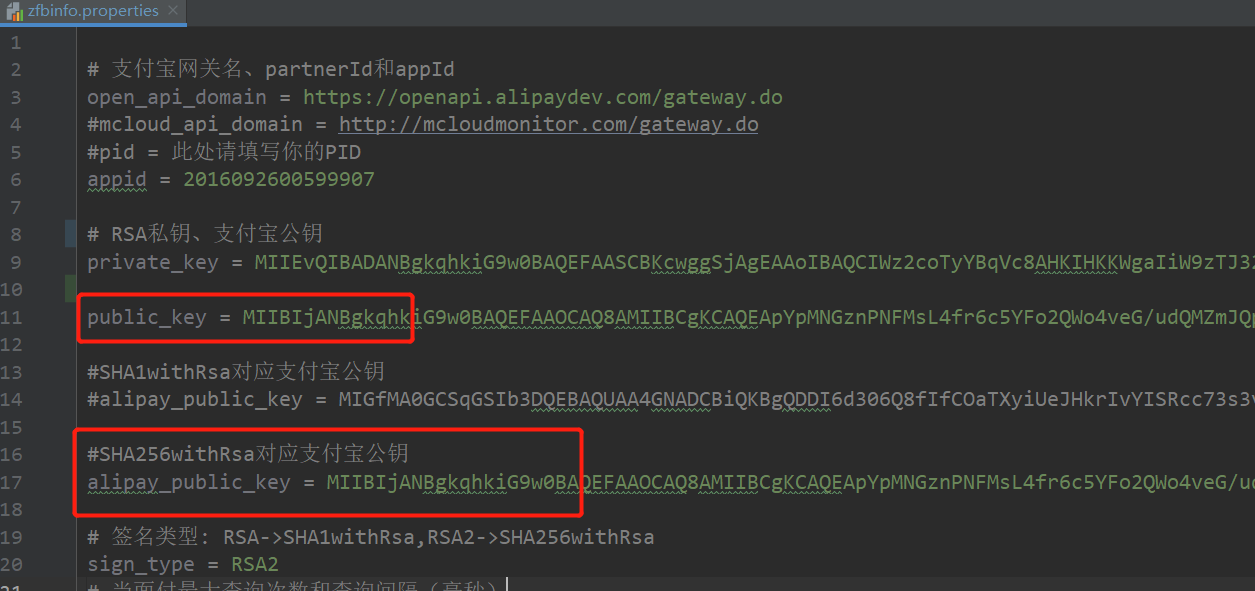
填写你用工具获得的公钥



保存设置以后得到支付宝的公钥



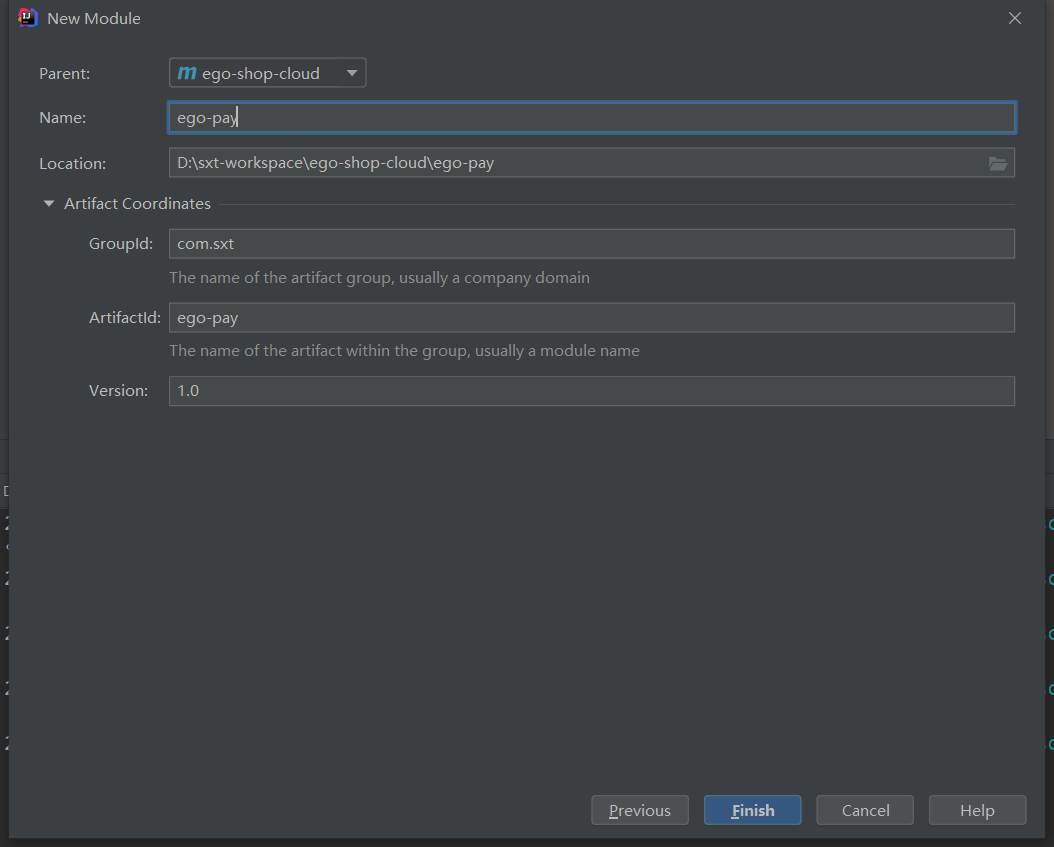
将支付宝的公钥复制到配置文件中zfbinfo.properties



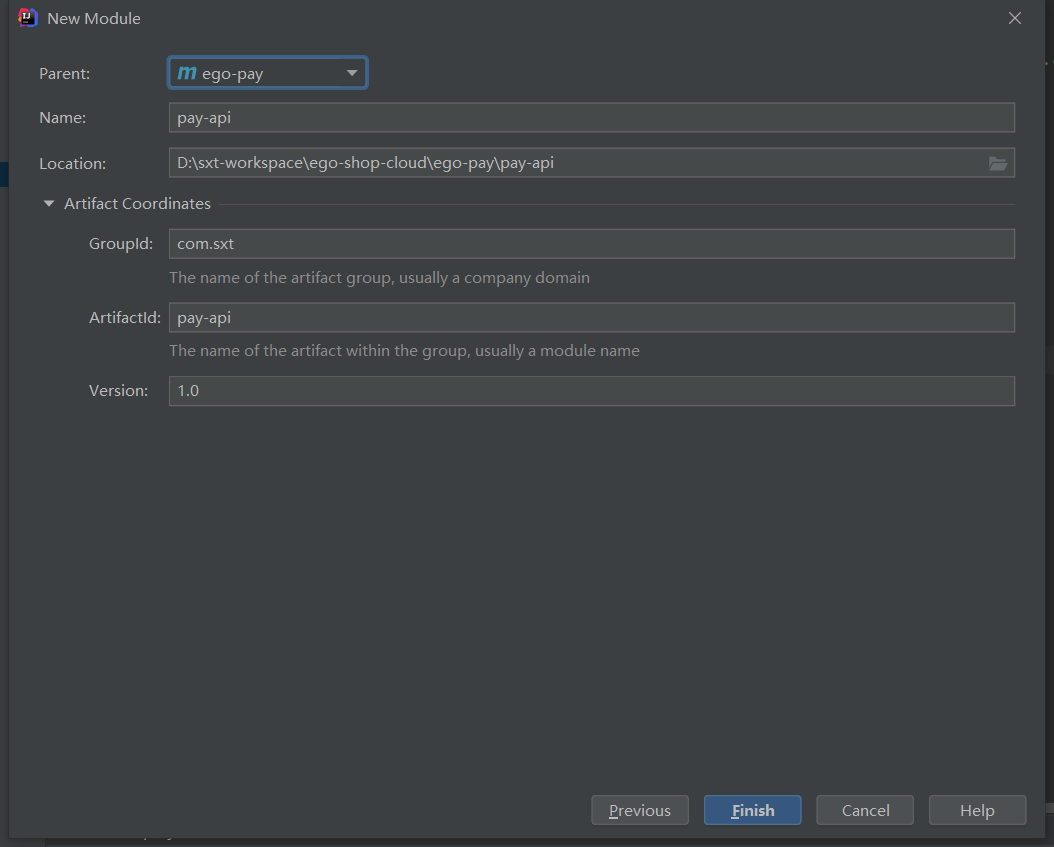
#### 最后我们的配置文件内容如下

|  |
| --- |
| # 支付宝网关名、partnerId和appId  **open\_api\_domain = https://openapi.alipaydev.com/gateway.do**  #mcloud\_api\_domain = http://mcloudmonitor.com/gateway.do  #pid = 此处请填写你的PID  **appid = 2016092600599907**  **# RSA私钥、支付宝公钥**  **private\_key = **  **public\_key = MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApYpMNGznPNFMsL4fr6c5YFo2QWo4veG/udQMZmJQpTtXVlblqXSIgRsTy2EdCDkRyDqMVKj2sMmzv1cIyZqgZOw96g7Z4IKTDuGOkWxz4+dMJX1qnhQbfkvk+tvYTU2sLZ25CDAg4UEHfB5DpGZCru+SeIrCR6lN/ZEfIYKqNlCq4BcHOBD5pXvs1eIOfSmOGfYLgKfWtJP8da0BNYt3EbYgUDRTnagBnqvGUR5eZs226w+OhgS9kWXJ2suuMdkkBfOjNkyOf4n0heNiXYIIDdODUGkuuU+Ph9IkGjRz32u4hZxsZndfJdDpWhxaZYsY9NuLHw5TspdWlgSrXd2ePwIDAQAB**  #SHA1withRsa对应支付宝公钥  #alipay\_public\_key = MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDDI6d306Q8fIfCOaTXyiUeJHkrIvYISRcc73s3vF1ZT7XN8RNPwJxo8pWaJMmvyTn9N4HQ632qJBVHf8sxHi/fEsraprwCtzvzQETrNRwVxLO5jVmRGi60j8Ue1efIlzPXV9je9mkjzOmdssymZkh2QhUrCmZYI/FCEa3/cNMW0QIDAQAB  #SHA256withRsa对应支付宝公钥  **alipay\_public\_key = MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApYpMNGznPNFMsL4fr6c5YFo2QWo4veG/udQMZmJQpTtXVlblqXSIgRsTy2EdCDkRyDqMVKj2sMmzv1cIyZqgZOw96g7Z4IKTDuGOkWxz4+dMJX1qnhQbfkvk+tvYTU2sLZ25CDAg4UEHfB5DpGZCru+SeIrCR6lN/ZEfIYKqNlCq4BcHOBD5pXvs1eIOfSmOGfYLgKfWtJP8da0BNYt3EbYgUDRTnagBnqvGUR5eZs226w+OhgS9kWXJ2suuMdkkBfOjNkyOf4n0heNiXYIIDdODUGkuuU+Ph9IkGjRz32u4hZxsZndfJdDpWhxaZYsY9NuLHw5TspdWlgSrXd2ePwIDAQAB**  # 签名类型: RSA->SHA1withRsa,RSA2->SHA256withRsa  sign\_type = RSA2  # 当面付最大查询次数和查询间隔（毫秒）  max\_query\_retry = 5  query\_duration = 5000  # 当面付最大撤销次数和撤销间隔（毫秒）  max\_cancel\_retry = 3  cancel\_duration = 2000  # 交易保障线程第一次调度延迟和调度间隔（秒）  heartbeat\_delay = 5  heartbeat\_duration = 900 |

## 在ego-shop-cloud下面创建ego-pay模块



### 在ego-pay模块下创建pay-api



#### 修改pay-api的pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>ego-pay</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>pay-api</artifactId>  <description>ego商城支付服务api</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>ego-common</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>order-api</artifactId>  <version>1.0</version>  </dependency>  </dependencies>  </project> |

#### 创建支付的实体类Pay

具体查看：

<https://opendocs.alipay.com/apis/api_1/alipay.trade.precreate?scene=API002020081000013487>

|  |
| --- |
| /\*\*  \* @Author: CXS  \* 这是一个支付的对象类  \*/  @Data  @JSONType(ignores = {"payType"})  public class Pay {  @ApiModelProperty("支付的类型,0:扫描支付")  private Integer payType;  // 需保证商户系统端不能重复，建议通过数据库sequence生成，  @JSONField(name = "out\_trade\_no")  @ApiModelProperty("订单号")  private String outTradeNo;  // (必填) 订单标题，粗略描述用户的支付目的。如“xxx品牌xxx门店当面付扫码消费”  @ApiModelProperty("订单的标题")  private String subject;  // (必填) 订单总金额，单位为元，不能超过1亿元  // 如果同时传入了【打折金额】,【不可打折金额】,【订单总金额】三者,则必须满足如下条件:【订单总金额】=【打折金额】+【不可打折金额】  @JSONField(name = "total\_amount")  @ApiModelProperty("订单的总金额")  private String totalAmount;  // (可选) 订单不可打折金额，可以配合商家平台配置折扣活动，如果酒水不参与打折，则将对应金额填写至此字段  // 如果该值未传入,但传入了【订单总金额】,【打折金额】,则该值默认为【订单总金额】-【打折金额】  @JSONField(name = "undiscountable\_amount")  @ApiModelProperty("订单的折扣金额")  private String undiscountableAmount = "0";  // 卖家支付宝账号ID，用于支持一个签约账号下支持打款到不同的收款账号，(打款到sellerId对应的支付宝账号)  // 如果该字段为空，则默认为与支付宝签约的商户的PID，也就是appid对应的PID  @JSONField(name = "seller\_id")  private String sellerId = "";  // 订单描述，可以对交易或商品进行一个详细地描述，比如填写"购买商品2件共15.00元"  @ApiModelProperty("订单描述")  private String body;  // 商户操作员编号，添加此参数可以为商户操作员做销售统计  @ApiModelProperty("操作员的编号,测试缓存是写死的")  @JSONField(name = "operator\_id")  private String operatorId = "test\_operator\_id";  // (必填) 商户门店编号，通过门店号和商家后台可以配置精准到门店的折扣信息，详询支付宝技术支持  @ApiModelProperty("门店的编号,测试缓存是写死的")  @JSONField(name = "store\_id")  private String storeId = "test\_store\_id";  // 支付超时，定义为120分钟  @ApiModelProperty("支付的超时时间")  @JSONField(name = "timeout\_express")  private String timeoutExpress = "120m";  @ApiModelProperty("产品码")  @JSONField(name = "product\_code")  private String productCode = "FAST\_INSTANT\_TRADE\_PAY";  } |

#### 创建支付接口PayService

|  |
| --- |
| public interface PayService {  /\*\*  \* 支付的接口  \*  \* @param pay  \* @return  \*/  String pay(Pay pay);  /\*\*  \* 签名的检查  \*  \* @param params  \* @return  \*/  boolean rsaCheckV1(Map<String, String> params);  /\*\*  \* 去支付  \*  \* @param orderSettlement  \* @return  \*/  String toPay(OrderSettlement orderSettlement);  /\*\*  \* 写库修改订单状态  \*  \* @param orderSn  \*/  void setPayed(String orderSn);  } |

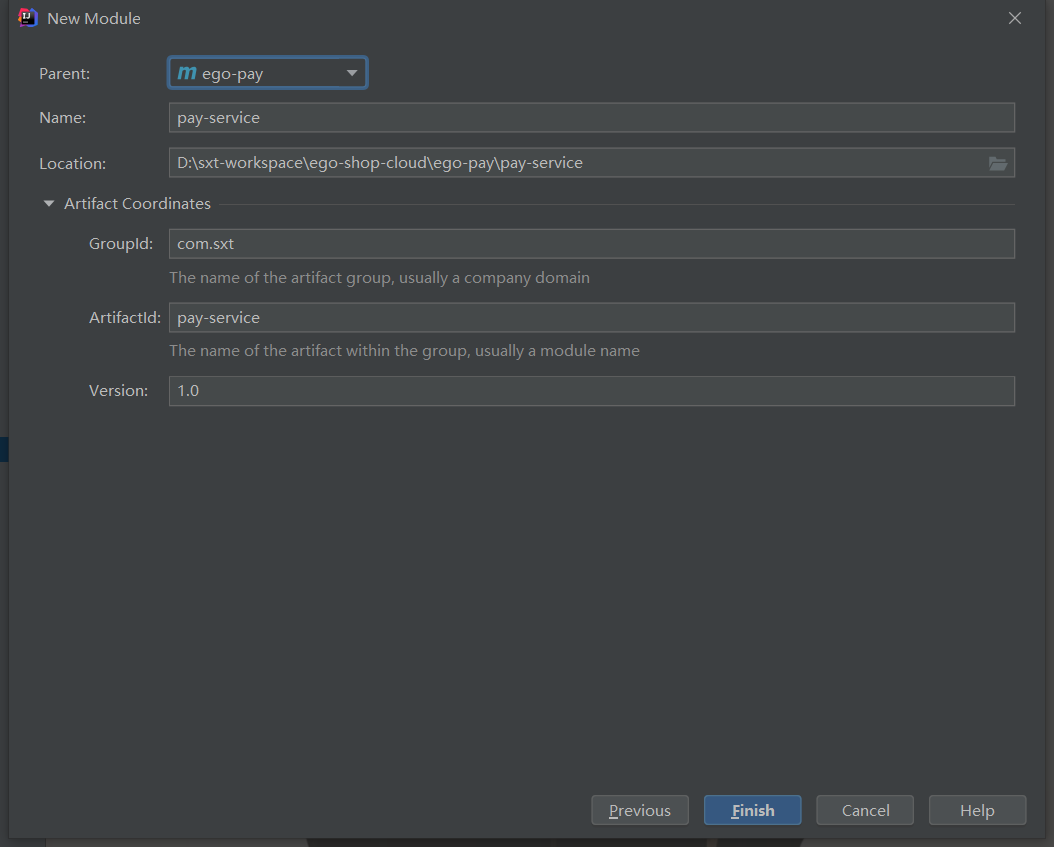
#### 创建远程调用的PayFeign

|  |
| --- |
| @FeignClient(value = "order-service", fallback = PayFeignHystrix.class)  public interface PayFeign {  /\*\*  \* 远程调用查询支付结算  \*  \* @param orderNum  \* @return  \*/  @GetMapping("/order/getOrderSettlement")  OrderSettlement getOrderSettlement(@RequestParam("orderNum") String orderNum);  /\*\*  \* 设置支付状态  \*  \* @param orderNum  \*/  @PostMapping("/order/setPayed")  void setPayed(@RequestBody String orderNum);  } |

#### 创建远程调用熔断的PayFeignHystrix

|  |
| --- |
| @Component  @Slf4j  public class PayFeignHystrix implements PayFeign {  /\*\*  \* 远程调用查询支付结算  \*  \* @param orderNum  \* @return  \*/  @Override  public OrderSettlement getOrderSettlement(String orderNum) {  log.error("远程调用查询订单支付结算失败.订单号{}", orderNum);  return new OrderSettlement();  }  /\*\*  \* 设置支付状态  \*  \* @param orderNum  \*/  @Override  public void setPayed(String orderNum) {  log.error("远程调用修改订单支付状态失败，订单为{}", orderNum);  }  } |

### 在ego-pay模块下创建pay-service



#### 将zfbinfo.properties粘贴到resources下面

#### 修改pom.xml文件

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>ego-pay</artifactId>  <groupId>com.sxt</groupId>  <version>1.0</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>pay-service</artifactId>  <description>ego商城支付服务service</description>  <dependencies>  <dependency>  <groupId>com.sxt</groupId>  <artifactId>pay-api</artifactId>  <version>1.0</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-bus-amqp</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-config-client</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis-reactive</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  </dependency>  **<!-- 支付宝支付的相关jar包-->**  **<dependency>**  **<groupId>com.alipay.sdk</groupId>**  **<artifactId>alipay-sdk-java</artifactId>**  **<version>3.3.0</version>**  **</dependency>**  **<dependency>**  **<groupId>com.alipay</groupId>**  **<artifactId>alipay-trade-sdk</artifactId>**  **<version>20161215</version>**  **</dependency>**  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

#### 添加启动类

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

#### 添加AliPayProperties

|  |
| --- |
| @ConfigurationProperties(prefix = "ailipay")  @Component  @Data  public class AilipayProperties {  // 应用ID,您的APPID，收款账号既是您的APPID对应支付宝账号  @ApiModelProperty("应用的Id")  private String appId;  // 商户私钥，您的PKCS8格式RSA2私钥  @ApiModelProperty("商户私钥")  private String merchantPrivateKey;  // 支付宝公钥,查看地址：https://openhome.alipay.com/platform/keyManage.htm 对应APPID下的支付宝公钥。  @ApiModelProperty("支付宝公钥")  private String alipayPublicKey;  // 服务器异步通知页面路径 需http://格式的完整路径，不能加?id=123这类自定义参数，必须外网可以正常访问  @ApiModelProperty("服务器异步通知页面路径")  private String notifyUrl;  // 页面跳转同步通知页面路径 需http://格式的完整路径，不能加?id=123这类自定义参数，必须外网可以正常访问  @ApiModelProperty("服务器同步通知页面路径")  private String returnUrl;  @ApiModelProperty("签名方式")  private String signType = "RSA2";  @ApiModelProperty("字符编码格式")  // 字符编码格式  private String charset = "utf-8";  // 支付宝网关  private String gatewayUrl = "https://openapi.alipaydev.com/gateway.do";  } |

#### 添加AliPayAutoConfiguration

|  |
| --- |
| @Configuration  @EnableConfigurationProperties({AilipayProperties.class})  public class AilipayAutoConfiguration {  @Autowired  private AilipayProperties ailipayProperties;  public AilipayAutoConfiguration() {  //读取配置文件  Configs.init("zfbinfo.properties");  }  @Bean  public AlipayTradeService alipayTradeService() {  return new AlipayTradeServiceImpl.ClientBuilder().build();  }  @Bean  public AlipayClient alipayClient() {  return new DefaultAlipayClient(  this.ailipayProperties.getGatewayUrl(),  this.ailipayProperties.getAppId(),  this.ailipayProperties.getMerchantPrivateKey(), "json",  this.ailipayProperties.getCharset(),  this.ailipayProperties.getAlipayPublicKey(),  this.ailipayProperties.getSignType());  }  } |

#### 添加bootstrap.yml配置文件

|  |
| --- |
| spring:  application:  name: pay-service  cloud:  config:  discovery:  enabled: true  service-id: config-server  profile: dev  name: pay-service  label: master  main: #允许多个feign调用相同的接口，因为我们依赖了其他的api，有相同的远程调用接口  allow-bean-definition-overriding: true  eureka:  client:  service-url:  defaultZone: ${EUREKA\_SERVER:http://admin:admin@localhost:8761/eureka}  instance:  hostname: ${APP\_HOST:localhost}  prefer-ip-address: true  instance-id: ${spring.application.name}:${server.port}  lease-expiration-duration-in-seconds: 30  lease-renewal-interval-in-seconds: 10 |

#### 添加远端pay-service/pay-service-dev.yml

|  |
| --- |
| server:  port: ${APP\_PORT:8089}  spring:  rabbitmq:  host: 192.168.226.129  password: admin  username: admin  port: 5672  listener:  simple:  acknowledge-mode: manual #手动签收消息 默认是自动的  redis:  host: 192.168.226.129  port: 6380  password: cxs1013??  database: 0  management:  endpoints:  web:  exposure:  include: '\*'  feign:  hystrix:  enabled: true  ribbon:  ReadTimeout: 18000  ConnectTimeout: 18000  hystrix: #hystrix的全局控制  command:  default: #default是全局控制，也可以换成单个方法控制，把default换成方法名即可  fallback:  isolation:  semaphore:  maxConcurrentRequests: 1000  circuitBreaker:  enabled: true #开启断路器  requestVolumeThreshold: 20 #失败次数（阀值）  sleepWindowInMilliseconds: 30000 #窗口时间  errorThresholdPercentage: 60 #失败率  execution:  isolation:  strategy: SEMAPHORE #使用信号量隔离方式  thread:  timeoutInMilliseconds: 5000  semaphore:  maxConcurrentRequests: 1000 #最大并发量  ailipay:  appId: 2016092600599907  merchantPrivateKey:   alipayPublicKey: MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApYpMNGznPNFMsL4fr6c5YFo2QWo4veG/udQMZmJQpTtXVlblqXSIgRsTy2EdCDkRyDqMVKj2sMmzv1cIyZqgZOw96g7Z4IKTDuGOkWxz4+dMJX1qnhQbfkvk+tvYTU2sLZ25CDAg4UEHfB5DpGZCru+SeIrCR6lN/ZEfIYKqNlCq4BcHOBD5pXvs1eIOfSmOGfYLgKfWtJP8da0BNYt3EbYgUDRTnagBnqvGUR5eZs226w+OhgS9kWXJ2suuMdkkBfOjNkyOf4n0heNiXYIIDdODUGkuuU+Ph9IkGjRz32u4hZxsZndfJdDpWhxaZYsY9NuLHw5TspdWlgSrXd2ePwIDAQAB  notifyUrl: http://sxt-pay.utools.club/payNotify  returnUrl: xxx  swagger2:  base-package: com.sxt.controller  name: sxt  url: https://gitee.com/smiledouble  email: 775610843@qq.com  title: 支付管理系统接口  description: 实现支付管理系统的所有接口  license: cxs  license-url: https://gitee.com/smiledouble  terms-of-service-url: https://gitee.com/smiledouble |

#### 添加PayController

|  |
| --- |
| @RestController  @Api(tags = "支付系统接口")  public class PayController {  @Autowired  private PayService payService;  @PostMapping("/p/order/pay")  @ApiOperation("完成订单的支付功能")  public ResponseEntity<String> toPay(@RequestBody @Validated OrderSettlement orderSettlement) {  String payUrl = payService.toPay(orderSettlement);  return ResponseEntity.ok(payUrl);  }  /\*\*  \* 支付回调接口 <https://opendocs.alipay.com/open/194/103296>  \*  \* @return  \*/  @PostMapping({"/payNotify"})  @ApiOperation("支付宝支付后回调接口")  public ResponseEntity<Void> notifyUrl(@RequestParam HashMap<String, String> map) {  System.out.println("有人来访问了");  System.out.println(map);  if (map.get("trade\_status").equals("TRADE\_SUCCESS")) {  System.out.println(JSON.toJSONString(map));  //如果是交易成功则去检查延签和修改数据库  boolean b = this.payService.rsaCheckV1(map);  if (b) {  System.out.println(map.get("out\_trade\_no"));  System.out.println("支付宝他来访问我了,通知我们付款成功了");  payService.setPayed(map.get("out\_trade\_no"));  } else {  System.out.println("我已经记录你的ip，请谨慎操作");  }  }  return ResponseEntity.ok().build();  }  } |

#### 创建PayServiceImpl

|  |
| --- |
| @Service  @Slf4j  public class PayServiceImpl implements PayService {  @Autowired  private AlipayTradeService tradeService;  @Autowired  private AilipayProperties ailipayProperties;  @Autowired  private PayFeign payFeign;  /\*\*  \* 支付的接口  \*  \* @param pay  \* @return  \*/  @Override  public String pay(Pay pay) {  // 创建扫码支付请求builder，设置请求参数  AlipayTradePrecreateRequestBuilder builder = new AlipayTradePrecreateRequestBuilder()  .setSubject(pay.getSubject())  .setTotalAmount(pay.getTotalAmount())  .setOutTradeNo(pay.getOutTradeNo())  .setUndiscountableAmount(pay.getUndiscountableAmount())  .setSellerId(pay.getSellerId())  .setBody(pay.getBody())  .setOperatorId(pay.getOperatorId())  .setStoreId(pay.getStoreId())  .setTimeoutExpress(pay.getTimeoutExpress())  .setNotifyUrl(ailipayProperties.getNotifyUrl());  AlipayF2FPrecreateResult result = tradeService.tradePrecreate(builder);  switch (result.getTradeStatus()) {  case SUCCESS:  log.info("支付宝预下单成功: )");  AlipayTradePrecreateResponse response = result.getResponse();  //返回二维码连接  System.out.println(response.getQrCode());  return response.getQrCode();  case FAILED:  log.error("支付宝预下单失败!!!");  break;  case UNKNOWN:  log.error("系统异常，预下单状态未知!!!");  break;  default:  log.error("不支持的交易状态，交易返回异常!!!");  break;  }  return null;  }  /\*\*  \* 签名的检查  \*  \* @param params  \* @return  \*/  @Override  public boolean rsaCheckV1(Map<String, String> params) {  boolean signVerified = false;  try {  signVerified = AlipaySignature.rsaCheckV1(params, ailipayProperties  .getAlipayPublicKey(), ailipayProperties  .getCharset(), ailipayProperties  .getSignType());  System.out.println(signVerified);  } catch (AlipayApiException e) {  log.error(e.getMessage());  e.printStackTrace();  signVerified = false;  }  return signVerified;  }  /\*\*  \* 去支付  \*  \* @param orderSettlement  \* @return  \*/  @Override  public String toPay(OrderSettlement orderSettlement) {  //远程调用查询订单  @NotNull OrderSettlement settlement = payFeign.getOrderSettlement(orderSettlement.getOrderNumber());  //拿到支付金额  @NotBlank BigDecimal payAmount = settlement.getPayAmount();  @NotBlank String orderNumber = settlement.getOrderNumber();  //创建支付对象  Pay pay = new Pay();  pay.setTotalAmount(payAmount.toString());  pay.setOutTradeNo(orderNumber);  pay.setBody("ego商城购买的商品");  pay.setSubject("【ego】商城" + orderNumber + "订单");  pay.setPayType(0);  //调用支付的方法  String payUrl = pay(pay);  return payUrl;  }  /\*\*  \* 写库修改订单状态  \*  \* @param orderSn  \*/  @Override  public void setPayed(String orderSn) {  payFeign.setPayed(orderSn);  }  } |

### 修改order-service下面的接口

#### 修改OrderController

|  |
| --- |
| /\*\*  \* 用于前台查询订单支付状态的  \*  \* @param orderSn  \* @return  \*/  @GetMapping("/p/order/query")  @ApiOperation("查询订单是否已经支付")  public ResponseEntity<Boolean> queryOrderIsPayed(@RequestParam("orderSn") String orderSn) {  Boolean isPayed = this.orderService.queryOrderIsPayed(orderSn);  return ResponseEntity.ok(isPayed);  }  /\*\*  \* 提供远程调用查询订单结算  \*  \* @param orderNum  \* @return  \*/  @GetMapping("/order/getOrderSettlement")  @ApiOperation("根据订单号查询结算对象")  OrderSettlement getOrderSettlement(@RequestParam("orderNum") String orderNum) {  return orderService.getOrderSettlement(orderNum);  }  /\*\*  \* 提供远程调用修改订单状态  \*  \* @param orderNum  \*/  @PostMapping("/order/setPayed")  @ApiOperation("支付成功，修改订单状态")  void setPayed(@RequestBody String orderNum) {  orderService.setPayed(orderNum);  } |

#### 修改OrderService

|  |
| --- |
| /\*\*  \* 根据订单编号查询订单结算  \*  \* @param orderNum  \* @return  \*/  OrderSettlement getOrderSettlement(String orderNum);  /\*\*  \* 查询订单是否已经支付  \*  \* @param orderSn  \* @return  \*/  Boolean queryOrderIsPayed(String orderSn);  /\*\*  \* 下单成功 修改订单状态  \*  \* @param orderNum  \*/  void setPayed(String orderNum); |

#### 修改OrderServiceImpl

|  |
| --- |
| /\*\*  \* 根据订单编号查询订单结算  \*  \* @param orderNum  \* @return  \*/  @Override  public OrderSettlement getOrderSettlement(String orderNum) {  return orderSettlementMapper.selectOne(new LambdaQueryWrapper<OrderSettlement>()  .eq(OrderSettlement::getOrderNumber, orderNum)  );  }  /\*\*  \* 查询订单是否已经支付  \*  \* @param orderSn  \* @return  \*/  @Override  public Boolean queryOrderIsPayed(String orderSn) {  Order order = orderMapper.selectOne(new LambdaQueryWrapper<Order>()  .eq(Order::getOrderNumber, orderSn)  );  return order.getIsPayed() == null ? Boolean.FALSE : order.getIsPayed();  }  /\*\*  \* 下单成功 修改订单状态  \*  \* @param orderNum  \*/  @Override  public void setPayed(String orderNum) {  Order order = orderMapper.selectOne(new LambdaQueryWrapper<Order>()  .eq(Order::getOrderNumber, orderNum)  );  if (order != null) {  order.setStatus(2);  order.setIsPayed(true);  order.setUpdateTime(new Date());  orderMapper.updateById(order);  }  OrderSettlement settlement = orderSettlementMapper.selectOne(new LambdaQueryWrapper<OrderSettlement>().  eq(OrderSettlement::getOrderNumber, orderNum));  if (settlement != null) {  settlement.setIsClearing(1);  settlement.setPayType(0);  settlement.setPayTypeName("支付宝扫描支付");  orderSettlementMapper.updateById(settlement);  }  } |

### 修改ego-common

#### 修改ResourceConfig放行支付宝访问的请求

|  |
| --- |
| @Override  public void configure(HttpSecurity http) throws Exception {  http.csrf().disable()  .sessionManagement().disable(); // 基于token，所以不需要session  http.authorizeRequests().antMatchers(  "/v2/api-docs",  "/v3/api-docs",  "/swagger-resources/configuration/ui", //用来获取支持的动作  "/swagger-resources", //用来获取api-docs的URI  "/swagger-resources/configuration/security",//安全选项  "/webjars/\*\*",  "/swagger-ui/\*\*",  "/druid/\*\*",  "/actuator/\*\*",  **"/payNotify/\*\*"**  ).permitAll()  .antMatchers("/\*\*").authenticated()  .and().headers().cacheControl();  } |