## 服务端搭建

### 1.1.开发工具配置

#### 1.1.1.配置JDK1.8

参照：[5.JDK1.8的安装和配置.docx](../00234002%20宜亩网（支付项目）2-各种手册/5.JDK1.8的安装和配置.docx)

#### 配置Maven 3.6.3环境

参照：[6.Maven 3.6.3的安装和配置.docx](../00234002%20宜亩网（支付项目）2-各种手册/6.Maven%203.6.3的安装和配置.docx)

#### lombok

Lombok是一个实用的java工具，使用它可以消除java代码的臃肿，Lombok提供一系列的注解，使用这些注解可以不用定义getter/setter、equals、构造方法等，它会在编译时在字节码文件自动生成这些通用的方法，简化开发人员的工作。

项目官方地址：<https://www.projectlombok.org/>

比如上节创建的UserTest模型，@Data注解可以自动生成getter/setter方法，@ToString生成tostring方法。

使用方法： 在项目中添加Lombok的依赖作用：项目在编译时根据Lombok注解生成通用方法，依赖如下：

|  |
| --- |
| <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.0</version>  </dependency> |

### 1.2.Nacos服务发现与配置中心

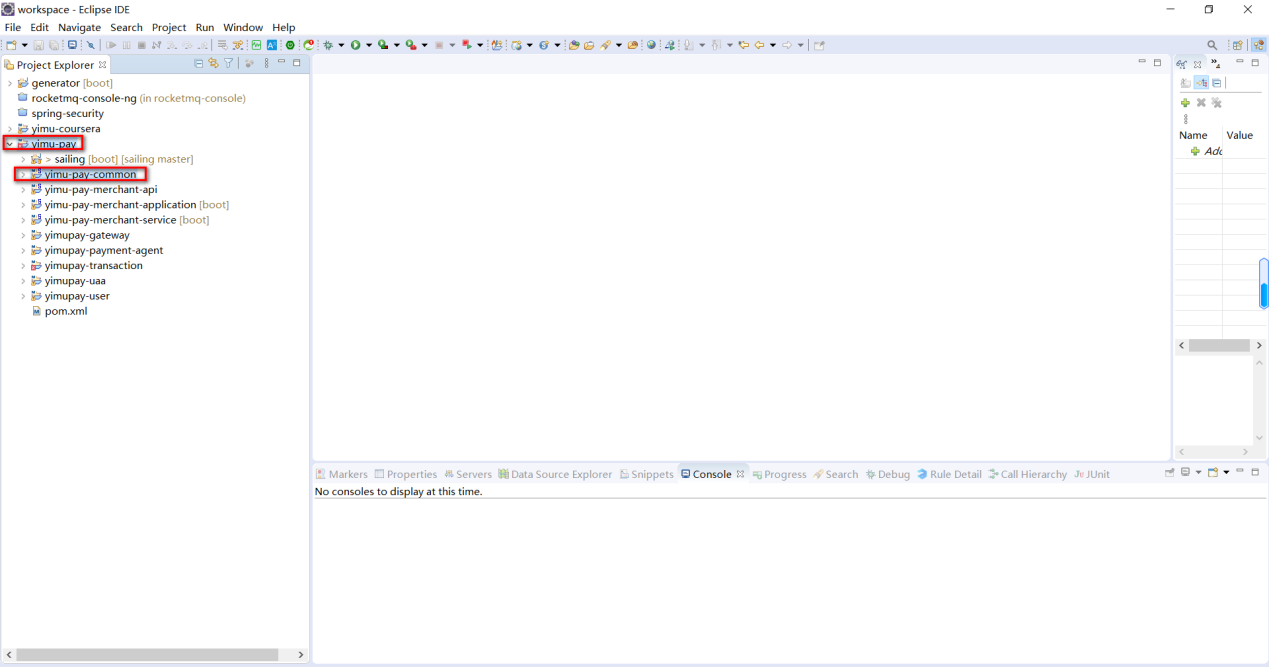
参照：<3.项目环境搭建-Nacos（服务发现和管理配置）.docx>

### 1.3.Mybatis Plus

参照：<4.项目环境搭建-Mybatis-Plus.docx>

### 1.4.创建基础工程

|  |  |
| --- | --- |
| 工程名 | 说明 |
| yimu-pay | 宜亩支付父工程 |
| yimu-pay-common | 项目通用工程包括：常用工具类和分页信息封装VO等 |



### 1.5.导入项目初始SQL

执行【yimupay-init.sql】，创建数据库和基础表，执行脚本自动创建数据库并导入初始数据。

数据库清单如下：

|  |  |
| --- | --- |
| 数据库名称 | 说明 |
| yimupay\_merchant\_service | 用户中心数据 |
| yimupay\_transaction | 交易服务数据库 |

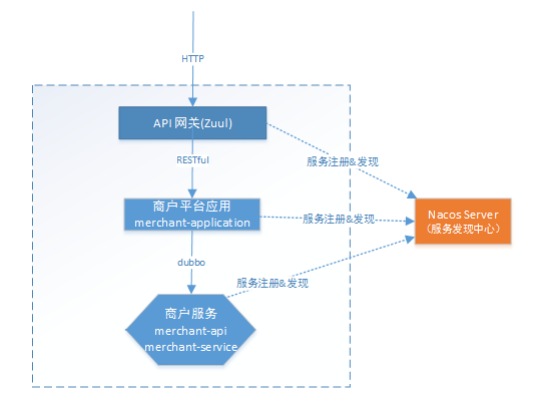
**备注：数据库、表中的字段以及表关系会在后续开发过程中随用随讲。**

### 1.6.搭建微服务

本节搭建如下项目工程：

|  |  |
| --- | --- |
| 服务名称 | 说明 |
| 商户平台应用  (yimu-pay-merchant-application) | 为前端提供商户管理功能 |
| 商户服务API(yimu-pay-merchant-api) | 定义商户服务提供的接口 |
| 商户服务(yimu-pay-merchant-service) | 实现商户服务的所有接口 |

三个工程在架构中的位置如下：



#### 1.6.1.搭建商户平台应用

在基础工程的基础上创建商户平台应用工程。

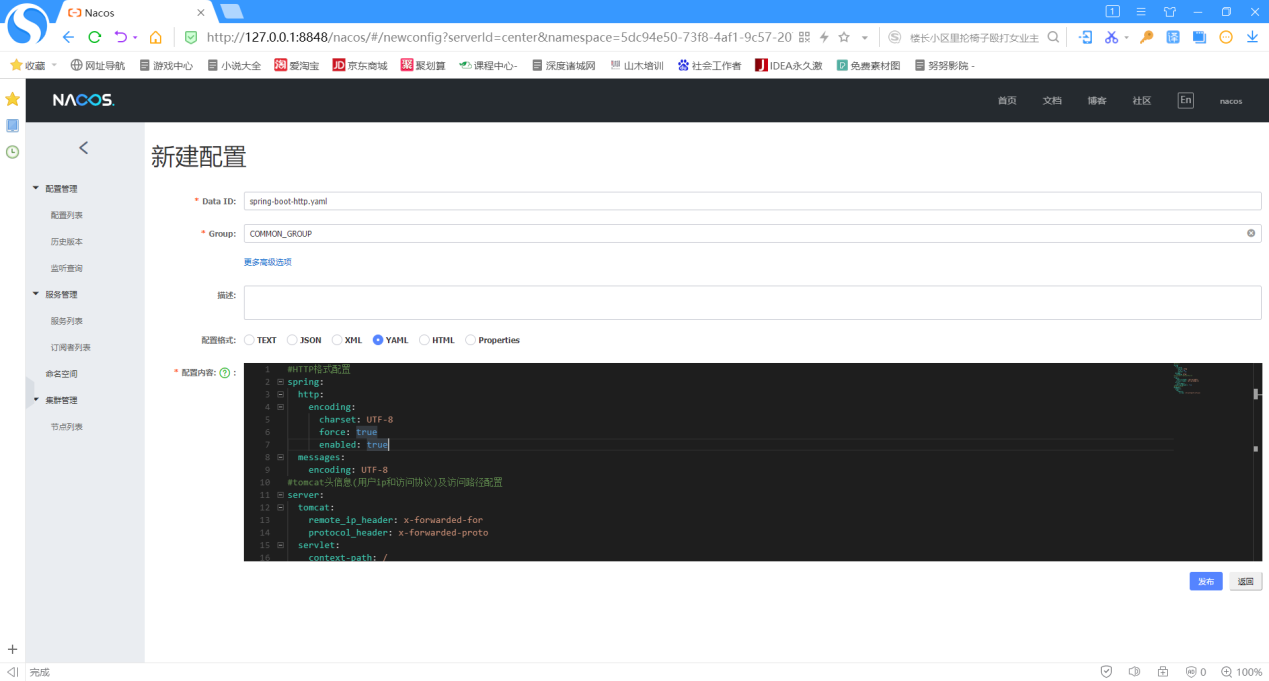
##### 1.6.1.1.pom.xml

|  |
| --- |
| <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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.me.yimu.pay.parent</groupId>  <artifactId>yimu-pay</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.me.yimu.pay.parent.merchant.application</groupId>  <artifactId>yimu-pay-merchant-application</artifactId>  <name>商户平台应用</name>    <dependencies>  <!-- Nacos配置中心 -->  <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>  </dependency>    <!-- Nacos注册中心 -->  <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>  </dependency>    <!-- Dubbo启动器 -->  <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-dubbo</artifactId>  </dependency>    <!-- Web启动器 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>    <!-- Spring Boot启动器 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-logging</artifactId>  </exclusion>  </exclusions>  </dependency>    <!-- log4j4启动器 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-log4j2</artifactId>  </dependency>    <!-- 注释处理器 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-configuration-processor</artifactId>  <optional>true</optional>  </dependency>    <!-- 健康检查，运维相关 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>    <!-- 测试启动器 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>    <!-- 商户服务API -->  <dependency>  <groupId>com.me.yimu.pay.parent.merchant.api</groupId>  <artifactId>yimu-pay-merchant-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>    <!-- 交易服务API -->  <dependency>  <groupId>com.me.yimu.pay.transaction.api</groupId>  <artifactId>yimupay-transaction-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>    <!-- okhttp3依赖:HTTP 网络请求的依赖库 -->  <dependency>  <groupId>com.squareup.okhttp3</groupId>  <artifactId>okhttp</artifactId>  </dependency>  </dependencies>  </project> |

##### 1.6.1.2.配置bootstrap.yml

|  |
| --- |
| server:  port: 57011 #启动端口 命令行注入  max-http-header-size: 100KB  nacos:  server:  addr: 127.0.0.1:8848  spring:  application:  name: merchant-application  main:  allow-bean-definition-overriding: **true** # Spring Boot 2.1 需要设定， 覆盖重复的Bean  cloud:  nacos:  discovery:  server-addr: ${nacos.server.addr}  namespace: 5dc94e50-73f8-4af1-9c57-207a43c6d457  cluster-name: DEFAULT  config:  server-addr: ${nacos.server.addr} # 配置中心地址  file-extension: yaml  namespace: 5dc94e50-73f8-4af1-9c57-207a43c6d457 # 命令行注入  group: YIMUPAY\_GROUP # 聚合支付业务组  ext-config:  -  refresh: **true** #配置信息是否动态刷新  data-id: spring-boot-http.yaml # spring boot http配置  group: COMMON\_GROUP # 通用配置组  # ext-config[0]:  # refresh: true  # data-id: spring-boot-http.yaml # spring boot http配置  # group: COMMON\_GROUP # 通用配置组  # ext-config[1]:  # refresh: true  # data-id: spring-boot-http.yaml # spring boot http配置  # group: COMMON\_GROUP # 通用配置组  #SpringMVC上传文件配置  servlet:  multipart:  #默认支持文件上传.  enabled: **true**  #支持文件写入磁盘.  file-size-threshold: 0  # 上传文件的临时目录  location:  # 最大支持文件大小  max-file-size: 1MB  # 最大支持请求大小  max-request-size: 30MB  dubbo:  scan:  # dubbo 服务扫描基准包  base-packages: com.me.yimu.pay  protocol:  # dubbo 协议  name: dubbo  port: 20891  registry:  address: nacos://127.0.0.1:8848  application:  qos:  port: 22310 # dubbo qos端口配置 命令行注入  consumer:  check: **false**  timeout: 90000  retries: -1  logging:  config: classpath:log4j2.xml  # Nacos默认负载均衡策略  #nacos-restful-provider:  # ribbon:  # NFLoadBalancerRuleClassName: com.netflix.loadbalancer.RandomRule  ## 项目的主配置文件 |

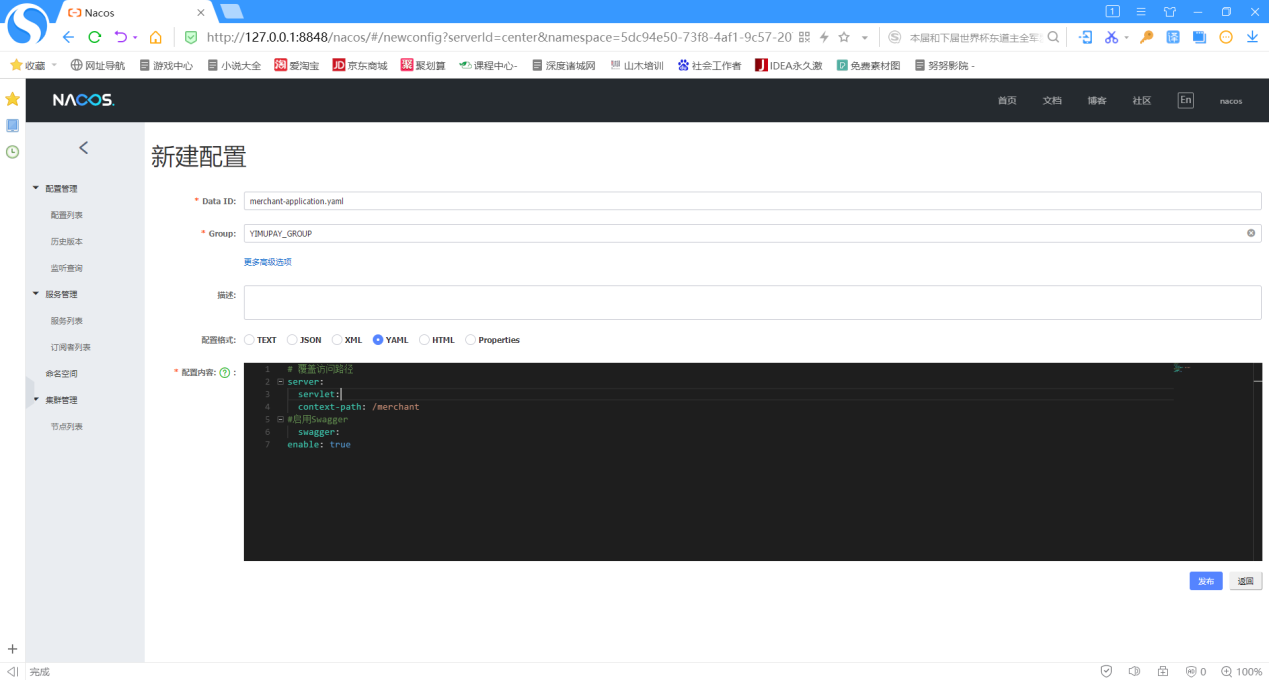
##### 1.6.1.3.在Nacos中添加spring-boot-http.yaml配置，Group：COMMON\_GROUP



配置信息：

|  |
| --- |
| #HTTP格式配置  spring:  http:  encoding:  charset: UTF-8  force: true  enabled: true  messages:  encoding: UTF-8  #tomcat头信息(用户ip和访问协议)及访问路径配置  server:  tomcat:  remote\_ip\_header: x-forwarded-for  protocol\_header: x-forwarded-proto  servlet:  context-path: /  use-forward-headers: true  #服务监控与管理配置，运维相关  management:  endpoints:  web:  exposure:  include: refresh,health,info,env |

##### 1.6.1.4.在Nacos中添加merchant-application.yaml配置，Group：YIMUPAY\_GROUP



配置信息：

|  |
| --- |
| **# 覆盖访问路径**  **server:**  **servlet:**  **context-path: /merchant**  **#启用Swagger**  **swagger:**  **enable: true**  spring:  profiles:  active: dev  sms:  url: "http://localhost:56085/sailing"  effectiveTime: 600  ## 七牛云配置  oss:  qiniu:  url: http://up-z0.qiniup.com  accessKey: lDZ4GHa3wN73xBboENGdCOkVLtWnVy2c7Qx7O7gY  secretKey: Bs7XOewJFNc923bdSZ4EY4WkS8j073X3dZKwHAHx  bucket: yimu-pay |

##### 1.6.1.5.在resources目录下添加log4j2配置文件：log4j2.xml

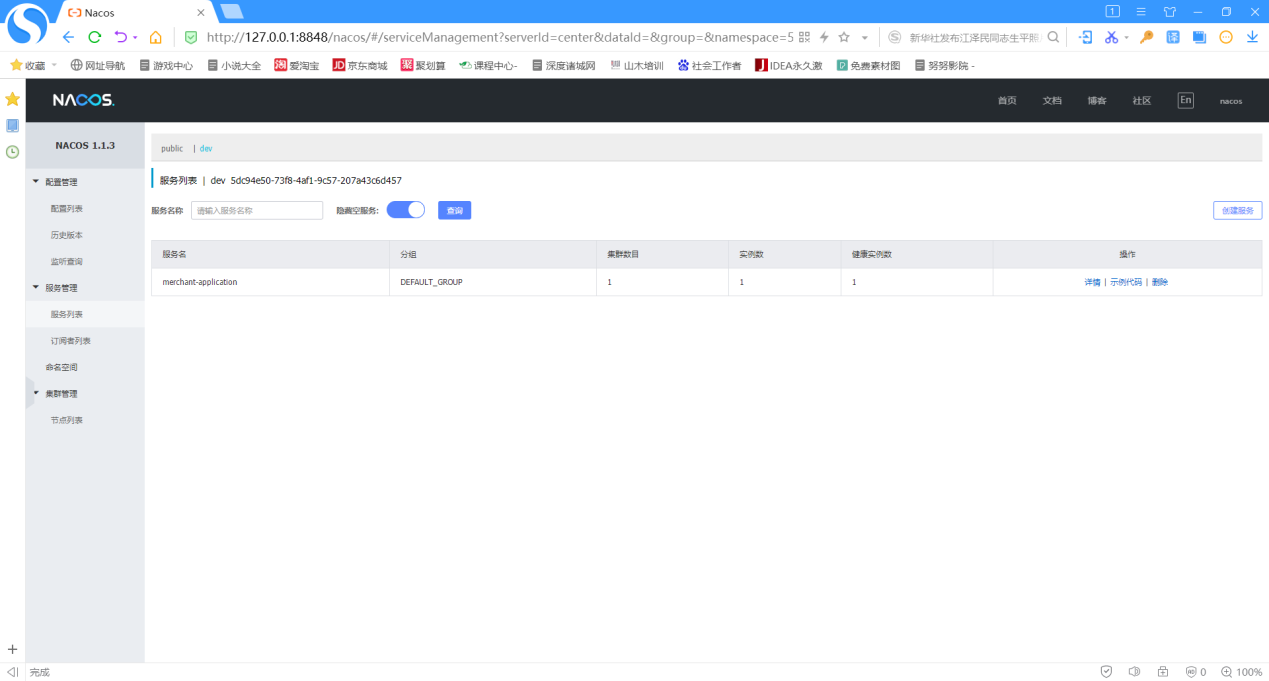
log4j2是log4j的改进版本，性能比log4j要高，通常日志配置文件在开发可以调整日志级别，输出详细的日志 来跟踪程序的执行。

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <Configuration monitorInterval=*"180"* packages=*""*>  <properties>  <property name=*"prjname"*>${project.name}</property>  <property name=*"logdir"*>logs</property>  <property name=*"PATTERN"*>[${project.name}][${env:SERVER\_PORT}] %date{YYYY-MM-dd HH:mm:ss,SSS} %highlight{%level} [%thread][%file:%line] - %msg%n%throwable</property>  </properties>  <Appenders>  <Console name=*"Console"* target=*"SYSTEM\_OUT"*>  <PatternLayout pattern=*"${PATTERN}"*/>  </Console>  <RollingFile name=*"ErrorAppender"* fileName=*"${logdir}/${prjname}\_error.log"*  filePattern=*"${logdir}/$${date:yyyy-MM-dd}/${prjname}\_error.%d{yyyy-MM-dd-HH}.log"* append=*"true"*>  <PatternLayout pattern=*"${PATTERN}"*/>  <ThresholdFilter level=*"ERROR"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>  <Policies>  <TimeBasedTriggeringPolicy interval=*"1"* modulate=*"true"* />  </Policies>  </RollingFile>  <RollingFile name=*"DebugAppender"* fileName=*"${logdir}/${prjname}\_info.log"*  filePattern=*"${logdir}/$${date:yyyy-MM-dd}/${prjname}\_info.%d{yyyy-MM-dd-HH}.log"* append=*"true"*>  <PatternLayout pattern=*"${PATTERN}"*/>  <ThresholdFilter level=*"DEBUG"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>  <Policies>  <TimeBasedTriggeringPolicy interval=*"1"* modulate=*"true"* />  </Policies>  </RollingFile>  <!--异步appender-->  <Async name=*"AsyncAppender"* includeLocation=*"true"*>  <AppenderRef ref=*"ErrorAppender"*/>  <AppenderRef ref=*"DebugAppender"*/>  </Async>  </Appenders>  <Loggers>  <!--过滤掉spring和mybatis的一些无用的debug信息-->  <logger name=*"org.springframework"* level=*"INFO"*>  </logger>  <logger name=*"org.mybatis"* level=*"INFO"*>  </logger>  <logger name=*"springfox"* level=*"INFO"*>  </logger>  <logger name=*"org.apache.http"* level=*"INFO"*>  </logger>  <logger name=*"com.alibaba.nacos"* level=*"WARN"*>  </logger>  <!--OFF 0-->  <!--FATAL 100-->  <!--ERROR 200-->  <!--WARN 300-->  <!--INFO 400-->  <!--DEBUG 500-->  <!--TRACE 600-->  <!--ALL Integer.MAX\_VALUE-->  <Root level=*"INFO"* includeLocation=*"true"*>  <AppenderRef ref=*"AsyncAppender"*/>  <AppenderRef ref=*"Console"*/>  </Root>  </Loggers>  </Configuration> |

##### 1.6.1.6.添加启动类

|  |
| --- |
| **package** com.me.yimu.pay.merchant;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.cloud.client.discovery.EnableDiscoveryClient;  @SpringBootApplication  @EnableDiscoveryClient  **public** **class** MerchantApplicationBootstrap {    **public** **static** **void** main(String[] args) {  SpringApplication.*run*(MerchantApplicationBootstrap.**class**,args);  System.***out***.println("Nacos请求路径：http://127.0.0.1:8848/nacos/");  }    } |

##### 1.6.1.7.启动项目，查看服务注册



#### 1.6.2.搭建商品服务工程

在基础工程的基础上创建商户服务工程，商户服务工程包括接口和接口实现两个子工程。

##### 1.6.2.1.pom.xml

|  |
| --- |
| <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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.me.yimu.pay.parent</groupId>  <artifactId>yimu-pay</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.me.yimu.pay.parent.merchant.service</groupId>  <artifactId>yimu-pay-merchant-service</artifactId>  <name>商品服务工程</name>    <dependencies>  <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>  </dependency>    <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>  </dependency>    <dependency>  <groupId>com.alibaba.cloud</groupId>  <artifactId>spring-cloud-starter-dubbo</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-logging</artifactId>  </exclusion>  </exclusions>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jdbc</artifactId>  </dependency>    <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  </dependency>    <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-generator</artifactId>  </dependency>    <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-typehandlers-jsr310</artifactId>  </dependency>    <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid-spring-boot-starter</artifactId>  </dependency>    <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-log4j2</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-configuration-processor</artifactId>  <optional>true</optional>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>    <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>    <!-- 对象池 -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-pool2</artifactId>  </dependency>    <dependency>  <groupId>org.mapstruct</groupId>  <artifactId>mapstruct-jdk8</artifactId>  </dependency>    <dependency>  <groupId>org.mapstruct</groupId>  <artifactId>mapstruct-processor</artifactId>  <version>${org.mapstruct.version}</version>  </dependency>    <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>${org.projectlombok.version}</version>  </dependency>  </dependencies>  </project> |

##### 1.6.2.2.在resources目录下添加日志配置文件：log4j2.xml

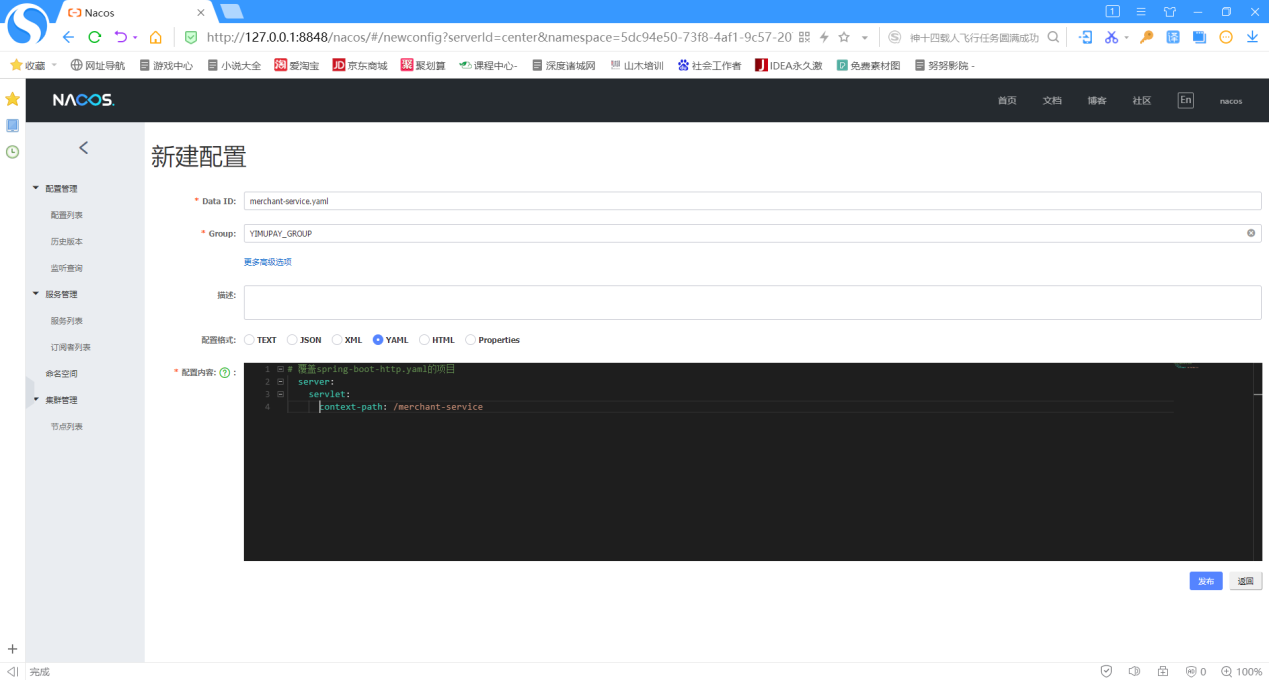
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <Configuration monitorInterval=*"180"* packages=*""*>  <properties>  <property name=*"prjname"*>${project.name}</property>  <property name=*"logdir"*>logs</property>  <property name=*"PATTERN"*>[${project.name}][${env:SERVER\_PORT}] %date{YYYY-MM-dd HH:mm:ss,SSS} %highlight{%level} [%thread][%file:%line] - %msg%n%throwable</property>  </properties>  <Appenders>  <Console name=*"Console"* target=*"SYSTEM\_OUT"*>  <PatternLayout pattern=*"${PATTERN}"*/>  </Console>  <RollingFile name=*"ErrorAppender"* fileName=*"${logdir}/${prjname}\_error.log"*  filePattern=*"${logdir}/$${date:yyyy-MM-dd}/${prjname}\_error.%d{yyyy-MM-dd-HH}.log"* append=*"true"*>  <PatternLayout pattern=*"${PATTERN}"*/>  <ThresholdFilter level=*"ERROR"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>  <Policies>  <TimeBasedTriggeringPolicy interval=*"1"* modulate=*"true"* />  </Policies>  </RollingFile>  <RollingFile name=*"DebugAppender"* fileName=*"${logdir}/${prjname}\_info.log"*  filePattern=*"${logdir}/$${date:yyyy-MM-dd}/${prjname}\_info.%d{yyyy-MM-dd-HH}.log"* append=*"true"*>  <PatternLayout pattern=*"${PATTERN}"*/>  <ThresholdFilter level=*"DEBUG"* onMatch=*"ACCEPT"* onMismatch=*"DENY"*/>  <Policies>  <TimeBasedTriggeringPolicy interval=*"1"* modulate=*"true"* />  </Policies>  </RollingFile>  <!--异步appender-->  <Async name=*"AsyncAppender"* includeLocation=*"true"*>  <AppenderRef ref=*"ErrorAppender"*/>  <AppenderRef ref=*"DebugAppender"*/>  </Async>  </Appenders>  <Loggers>  <!--过滤掉spring和mybatis的一些无用的debug信息-->  <logger name=*"org.springframework"* level=*"INFO"*>  </logger>  <logger name=*"org.mybatis"* level=*"INFO"*>  </logger>  <logger name=*"springfox"* level=*"INFO"*>  </logger>  <logger name=*"org.apache.http"* level=*"INFO"*>  </logger>  <logger name=*"com.alibaba.nacos"* level=*"WARN"*>  </logger>  <!--OFF 0-->  <!--FATAL 100-->  <!--ERROR 200-->  <!--WARN 300-->  <!--INFO 400-->  <!--DEBUG 500-->  <!--TRACE 600-->  <!--ALL Integer.MAX\_VALUE-->  <Root level=*"INFO"* includeLocation=*"true"*>  <AppenderRef ref=*"AsyncAppender"*/>  <AppenderRef ref=*"Console"*/>  </Root>  </Loggers>  </Configuration> |

##### 1.6.2.3.在resources目录下添加配置文件：bootstrap.yml，将下边配置中namespace的ID替换为之前创建的dev命名空间的ID

**bootstrap.yml:**

|  |
| --- |
| server:  port: 56040 #启动端口 命令行注入  nacos:  server:  addr: 127.0.0.1:8848  spring:  application:  name: merchant-service  main:  allow-bean-definition-overriding: **true** # Spring Boot 2.1 需要设定  cloud:  nacos:  discovery:  server-addr: ${nacos.server.addr}  namespace: 5dc94e50-73f8-4af1-9c57-207a43c6d457  cluster-name: DEFAULT  config:  server-addr: ${nacos.server.addr} # 配置中心地址  file-extension: yaml  namespace: 5dc94e50-73f8-4af1-9c57-207a43c6d457 # 命令行注入  group: YIMUPAY\_GROUP # 聚合支付业务组  ext-config:  - refresh: **true**  data-id: spring-boot-http.yaml # spring boot http配置  group: COMMON\_GROUP # 通用配置组  - refresh: **true**  data-id: spring-boot-starter-druid.yaml # spring boot starter druid配置  group: COMMON\_GROUP # 通用配置组  - refresh: **true**  data-id: spring-boot-mybatis-plus.yaml # spring boot mybatisplus配置  group: COMMON\_GROUP # 通用配置组  dubbo:  scan:  # dubbo 服务扫描基准包  base-packages: com.me.yimu.pay  protocol:  # dubbo 协议  name: dubbo  port: 20890  registry:  address: nacos://127.0.0.1:8848  application:  qos:  port: 22240 # dubbo qos端口配置 命令行注入  consumer:  check: **false**  timeout: 90000  retries: -1  logging:  config: classpath:log4j2.xml |

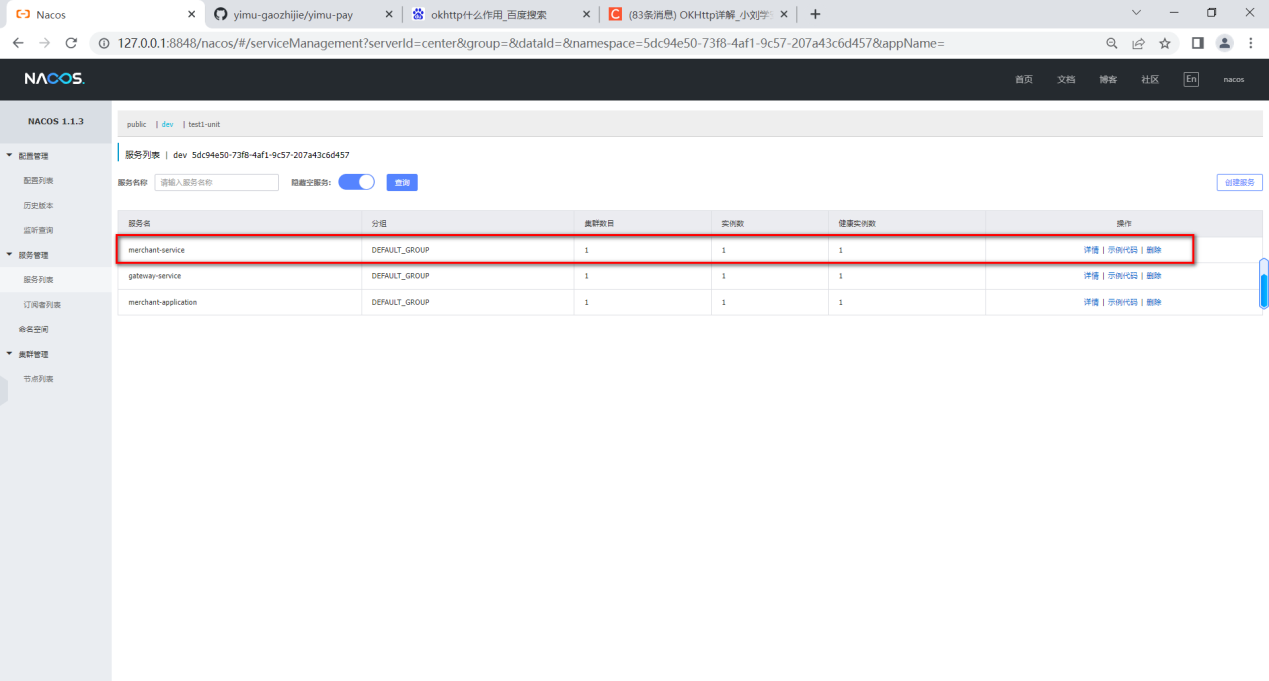
##### 1.6.2.4.在Nacos中添加merchant-service.yaml配置，Group：YIMUPAY\_GROUP



|  |
| --- |
| **# 覆盖spring‐boot‐http.yaml的项目**  **server:**  **servlet:**  **context-path: /merchant‐service**  # 覆盖spring‐boot‐starter‐druid.yaml的项目  spring:  datasource:  druid:  url: jdbc:mysql://127.0.0.1:3306/yimupay\_merchant\_service?useUnicode=true&serverTimezone=Asia/Shanghai&useSSL=false  username: root  password: mnj852123  # 覆盖spring‐boot‐mybatis‐plus.yaml的项目  mybatis‐plus:  typeAliasesPackage: com.me.yimu.pay.merchant.entity  mapper‐locations: classpath:com/yimupay/\*/mapper/\*.xml |

##### 1.6.2.5.创建商品中心服务启动类，查看服务注册

|  |
| --- |
| package com.me.yimu.pay.merchant;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @SpringBootApplication(scanBasePackages = "com.me.yimu.pay.merchant")  @MapperScan("com.me.yimu.pay.merchant.mapper")  public class MerchantBootstrap {  public static void main(String[] args) {  SpringApplication.run(MerchantBootstrap.class,args);  System.out.println("Nacos请求路径：http://127.0.0.1:8848/nacos/");  }  } |



#### 1.6.3.搭建商品服务API

##### 1.6.3.1.pom.xml

|  |
| --- |
| <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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>com.me.yimu.pay.parent</groupId>  <artifactId>yimu-pay</artifactId>  <version>0.0.1-SNAPSHOT</version>  </parent>  <groupId>com.me.yimu.pay.parent.merchant.api</groupId>  <artifactId>yimu-pay-merchant-api</artifactId>  <name>商品服务工程</name>    <dependencies>  <dependency>  <groupId>com.me.yimu.pay.common</groupId>  <artifactId>yimu-pay-common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  </dependencies>  </project> |

#### 1.6.4.工程测试

通过一个案例”根据Id查询商户“的开发去熟悉项目架构的基本开发方法。

##### 1.6.4.1.生成代码

使用mp的自动生成工程生成entity、mapper等文件。

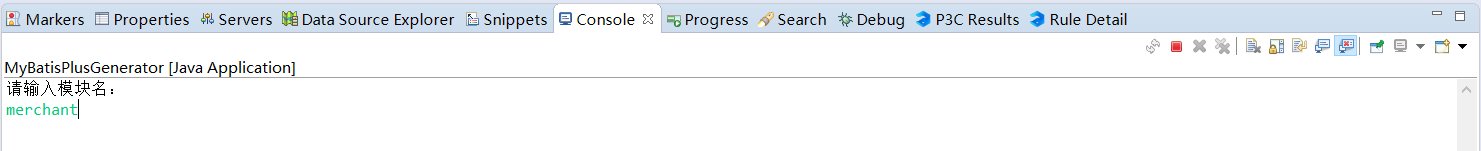
1）修改生成类中的数据库链接，连接yimupay\_merchant\_service数据库

|  |
| --- |
| dataSourceConfig  .setUrl("jdbc:mysql://localhost:3306/yimupay\_merchant\_service?useSSL=false&serverTimezone=Asia/Shanghai"); |

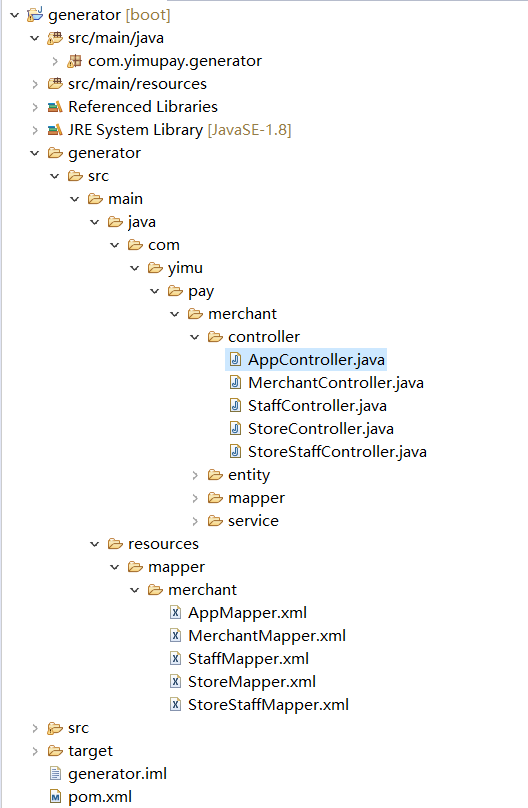
2）设置包路径

|  |
| --- |
| // 生成包配置  PackageConfig packageConfig = **new** PackageConfig();  packageConfig.setParent("com.yimu.pay"); |

3）运行生成类，输入模块名：merchant



生成成功：



将生成的entity、mapper拷贝到yimupay-merchant-service工程。

##### 1.6.4.2.Mybatis-Plus配置

###### 1.6.4.2.1.配置连接池Druid

1）在nacos中新建连接池Druid配置：spring-boot-starter-druid.yaml，Group为：COMMON\_GROUP 内容如下：

|  |
| --- |
| spring:  ## 通用配置，可以被各微服务的主配置覆盖  datasource:  type: com.alibaba.druid.pool.DruidDataSource  driver-class-name: com.mysql.cj.jdbc.Driver  url: jdbc:mysql://localhost:3306/oauth?useUnicode=true  username: root  password: yourpassword  druid:  initial-size: 5  min-idle: 5  max-active: 20  max-wait: 60000  time-between-eviction-runs-millis: 60000  min-evictable-idle-time-millis: 300000  validation-query: SELECT 1 FROM DUAL  test-while-idle: true  test-on-borrow: true  test-on-return: false  pool-prepared-statements: true  max-pool-prepared-statement-per-connection-size: 20  filter:  stat:  slow-sql-millis: 1  log-slow-sql: true  filters: config,stat,wall,log4j2  web-stat-filter:  enabled: true  url-pattern: /\*  exclusions: "\*.js,\*.gif,\*.jpg,\*.png,\*.css,\*.ico,/druid/\*"  session-stat-enable: false  session-stat-max-count: 1000  principal-cookie-name: admin  principal-session-name: admin  profile-enable: true  stat-view-servlet:  enabled: true  url-pattern: /druid/\*  allow: 127.0.0.1,192.168.163.1  deny: 192.168.1.73  reset-enable: false  login-password: admin  login-username: admin  aop-patterns: com.me.yimu.pay.\*.service.\*  #aop‐patterns: com.me.yimu.pay.\*.api.\* |

2）商户服务覆盖部分配置（数据库名和用户名密码）：merchant-service.yaml

|  |
| --- |
| # 覆盖spring‐boot‐http.yaml的项目  server:  servlet:  context‐path: /merchant‐service  **# 覆盖spring‐boot‐starter‐druid.yaml的项目**  **spring:**  **datasource:**  **druid:**  **url: jdbc:mysql://127.0.0.1:3306/yimupay\_merchant\_service?**  **useUnicode=true&characterEncoding=UTF‐8&serverTimezone=Asia/Shanghai&useSSL=false**  **username: root**  **password: mnj852123** |

3）应用配置到项目中：bootstrap.yml

|  |
| --- |
| ext-config:  -  refresh: **true**  data-id: spring-boot-http.yaml # spring boot http配置  group: COMMON\_GROUP # 通用配置组  **- refresh: true**  **data-id: spring-boot-starter-druid.yaml # spring boot starter druid配置**  **group: COMMON\_GROUP # 通用配置组**  - refresh: **true**  data-id: spring-boot-mybatis-plus.yaml # spring boot mybatisplus配置  group: COMMON\_GROUP # 通用配置组 |

###### 1.6.4.2.2.配置Mybatis-Plus

1）在nacos中添加配置：spring-boot-mybatis-plus.yaml，Group为：COMMON\_GROUP

|  |
| --- |
| mybatis-plus:  configuration:  cache-enabled: **false**  map-underscore-to-camel-case: **true**  global-config:  id-type: 0  field-strategy: 0  db-column-underline: **true**  refresh-mapper: **true**  typeAliasesPackage: com.me.yimu.pay.merchant.entity  mapper‐locations: classpath:com/me/yimupay/\*/mapper/\*.xml |

2）商户服务覆盖部分Mybatis-Plus配置：merchant-service.yaml

|  |
| --- |
| # 覆盖spring‐boot‐http.yaml的项目  server:  servlet:  context-path: /merchant‐service  # 覆盖spring‐boot‐starter‐druid.yaml的项目  spring:  datasource:  druid:  url: jdbc:mysql://127.0.0.1:3306/yimupay\_merchant\_service?useUnicode=true&serverTimezone=Asia/Shanghai&useSSL=false  username: root  password: mnj852123  **# 覆盖spring‐boot‐mybatis‐plus.yaml的项目**  **mybatis-plus:**  **typeAliasesPackage: com.me.yimu.pay.merchant.entity**  **mapper-locations: classpath:com/yimupay/\*/mapper/\*.xml** |

3）应用配置到项目中：bootstrap.yml

|  |
| --- |
| - refresh: **true**  data-id: spring-boot-mybatis-plus.yaml # spring boot mybatisplus配置  group: COMMON\_GROUP # 通用配置组 |

4）添加分页和性能分析插件：MybatisPlusConfig

|  |
| --- |
| package com.me.yimu.pay.merchant.config;  import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;  import com.baomidou.mybatisplus.extension.plugins.PerformanceInterceptor;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @Configuration  @MapperScan("com.yimupay.\*\*.mapper")  public class MybatisPlusConfig {  /\*\*  \* 分页插件，自动识别数据库类型  \*/  @Bean  public PaginationInterceptor paginationInterceptor() {  return new PaginationInterceptor();  }  /\*\*  \* 启用性能分析插件  \*/  @Bean  public PerformanceInterceptor performanceInterceptor(){  return new PerformanceInterceptor();  }  } |

##### 1.6.4.3.创建接口

在yimupay-merchant-api下新建商户接口：MerchantService

|  |
| --- |
| **package** com.me.yimu.pay.merchant.api;  **public** **interface** MerchantService {  /\*\*  \* 根据ID查询详细信息  \* **@param** merchantId  \* **@return**  \* **@throws** BusinessException  \*/  MerchantDTO queryMerchantById(Long merchantId);    } |

注意：DTO类型的对象作为service层传输的对象。

在yimupay-merchant-api工程 定义MerchantDTO

|  |
| --- |
| **package** com.me.yimu.pay.merchant.api.dto;  **import** java.io.Serializable;  **import** io.swagger.annotations.ApiModel;  **import** io.swagger.annotations.ApiModelProperty;  **import** lombok.Data;  /\*\*  \* **@author** Administrator  \* **@version** 1.0  \*\*/  @ApiModel(value = "MerchantDTO", description = "商户信息")  @Data  **public** **class** MerchantDTO **implements** Serializable {  @ApiModelProperty("商户id")  **private** Long id;  @ApiModelProperty("企业名称")  **private** String merchantName;  @ApiModelProperty("企业编号")  **private** String merchantNo;  @ApiModelProperty("企业地址")  **private** String merchantAddress;  @ApiModelProperty("行业类型")  **private** String merchantType;  @ApiModelProperty("营业执照")  **private** String businessLicensesImg;  @ApiModelProperty("法人身份证正面")  **private** String idCardFrontImg;  @ApiModelProperty("法人身份证反面")  **private** String idCardAfterImg;  @ApiModelProperty("联系人")  **private** String username;  @ApiModelProperty("密码")  **private** String password;  @ApiModelProperty("手机号,关联统一账号")  **private** String mobile;  @ApiModelProperty("联系人地址")  **private** String contactsAddress;  @ApiModelProperty("审核状态,0-未申请,1-已申请待审核,2-审核通过,3-审核拒绝")  **private** String auditStatus;  @ApiModelProperty("租户ID")  **private** Long tenantId;  } |

@ApiModel和 @ApiModelProperty是Swagger注解。

##### 1.6.4.4.创建接口的实现

在yimupay-merchant-service下新建商户接口实现类：MerchantServiceImpl，并添加新建商户测试方法

本方法从yimupay\_merchant\_service数据库的merchant查询数据。

|  |
| --- |
| **package** com.me.yimu.pay.merchant.service.impl;  **import** com.me.yimu.pay.merchant.api.MerchantService;  **import** com.me.yimu.pay.merchant.api.dto.MerchantDTO;  **import** com.me.yimu.pay.merchant.entity.Merchant;  **import** com.me.yimu.pay.merchant.mapper.MerchantMapper;  **import** com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;  **import** com.me.yimu.pay.merchant.service.IMerchantService;  **import** lombok.extern.slf4j.Slf4j;  **import** org.springframework.stereotype.Service;  **import** org.springframework.transaction.annotation.Transactional;  **import** org.springframework.beans.factory.annotation.Autowired;  /\*\*  \* <p>  \* 服务实现类  \* </p>  \*  \* **@author** author  \* **@since** 2022-12-05  \*/  @Slf4j  @Service  @Transactional  //public class MerchantServiceImpl extends ServiceImpl<MerchantMapper, Merchant> implements IMerchantService {  //  //}  **public** **class** MerchantServiceImpl **implements** MerchantService {  @Autowired  MerchantMapper merchantMapper;  /\*\*  \* 根据Id查询详细信息  \*/  @Override  **public** MerchantDTO queryMerchantById(Long id) {  Merchant merchant = merchantMapper.selectById(id);  MerchantDTO merchantDTO = **new** MerchantDTO();  merchantDTO.setId(merchant.getId());  merchantDTO.setMerchantName(merchant.getMerchantName());  //....  **return** merchantDTO;  }  } |

##### 1.6.4.5.创建应用层

在yimupay-merchant-application下添加如下pom依赖：

|  |
| --- |
| <dependency>  <groupId>com.me.yimu.pay.parent.merchant.api</groupId>  <artifactId>yimu-pay-merchant-api</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency> |

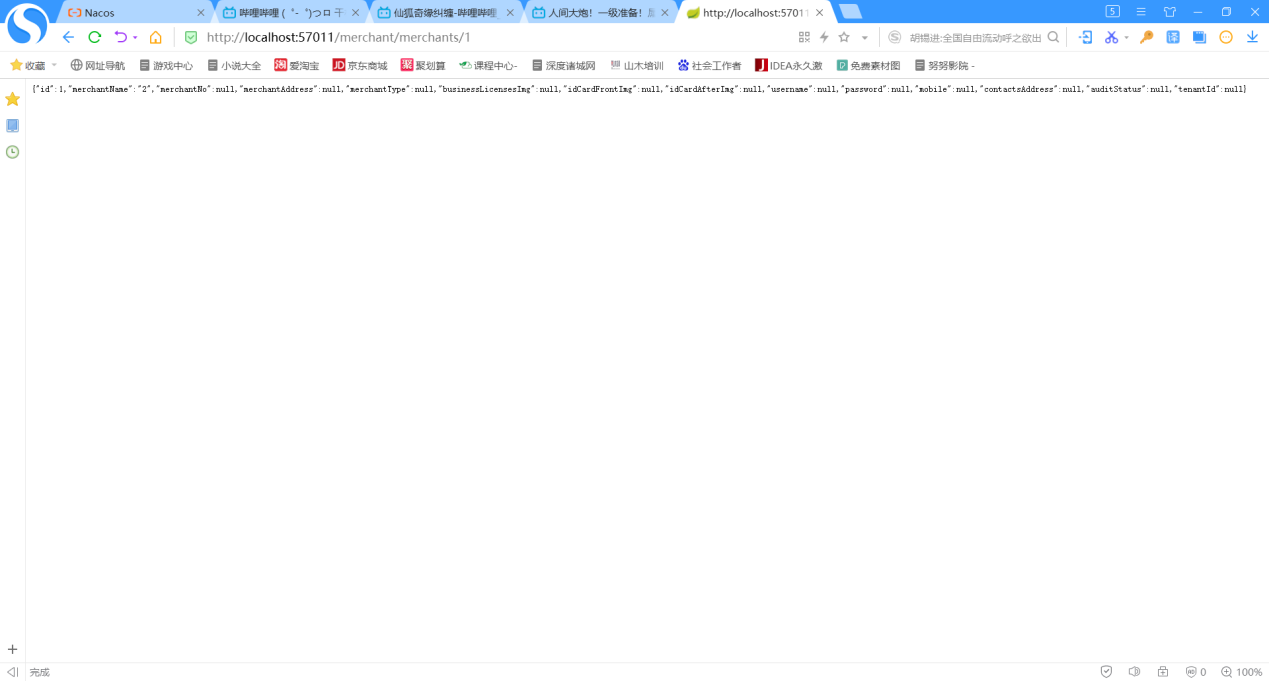
在yimupay-merchant-application下新建商户Controller：MerchantController，并调用商户中心服务提供的新 建商户接口。

|  |
| --- |
| **package** com.me.yimu.pay.merchant.controller;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PathVariable;  **import** com.me.yimu.pay.merchant.api.MerchantService;  **import** com.me.yimu.pay.merchant.api.dto.MerchantDTO;  **import** io.swagger.annotations.ApiOperation;  **public** **class** MerchantController {  @org.apache.dubbo.config.annotation.Reference  MerchantService merchantService;    @ApiOperation(value="根据id查询商户信息")  @GetMapping("/merchants/{id}")  **public** MerchantDTO queryMerchantById(@PathVariable("id") Long id){  MerchantDTO merchantDTO = merchantService.queryMerchantById(id);  **return** merchantDTO;  }    } |

##### 1.6.4.6.测试

1）启动商户平台应用和商户服务

2）访问http://localhost:57011/merchant/merchants/1



如果merchant表没有数据可手动添加后再行测试。

## 2.API接口文档利器：Swagger接口工具

### 2.1.Swagger介绍

Swagger 是一个规范和完整的框架，用于生成、描述、调用和可视化**RESTful风格**的Web 服务 (https://swagger.io/)。 它的主要作用是：

1. 使得前后端分离开发更加方便，有利于团队协作

2. 接口的文档在线自动生成，降低后端开发人员编写接口文档的负担

3. 功能测试Spring已经将Swagger纳入自身的标准，建立了Spring-swagger项目，现在叫Springfox。通过在项目中引入Springfox ，即可非常简单快捷的使用Swagger。

### 2.2.SpringBoot集成Swagger

1）在yimupay-common项目中添加依赖，只需要在yimupay-common中进行配置即可，因为其他微服务工 程都直接或间接依赖yimupay-common。

|  |
| --- |
| <!-- Swagger利用 -->  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger2</artifactId>  </dependency>  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger-ui</artifactId>  </dependency> |

2）在yimupay-merchant-application工程的config包中添加一个Swagger配置类:

|  |
| --- |
| package com.me.yimu.pay.merchant.config;  import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import springfox.documentation.builders.ApiInfoBuilder;  import springfox.documentation.builders.PathSelectors;  import springfox.documentation.builders.RequestHandlerSelectors;  import springfox.documentation.service.ApiInfo;  import springfox.documentation.service.Contact;  import springfox.documentation.spi.DocumentationType;  import springfox.documentation.spring.web.plugins.Docket;  import springfox.documentation.swagger2.annotations.EnableSwagger2;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @Configuration  @ConditionalOnProperty(prefix = "swagger",value = {"enable"},havingValue = "true")  @EnableSwagger2  public class SwaggerConfiguration {  @Bean  public Docket buildDocket() {  return new Docket(DocumentationType.SWAGGER\_2)  .apiInfo(buildApiInfo())  .select()  // 要扫描的API(Controller)基础包  .apis(RequestHandlerSelectors.basePackage("com.me.yimu.pay.merchant.controller"))  .paths(PathSelectors.any())  .build();  }  /\*\*  \* @param  \* @return springfox.documentation.service.ApiInfo  \* @Title: 构建API基本信息  \* @methodName: buildApiInfo  \*/  private ApiInfo buildApiInfo() {  Contact contact = new Contact("开发者","","");  return new ApiInfoBuilder()  .title("宜亩支付-商户应用API文档")  .description("")  .contact(contact)  .version("1.0.0").build();  }  } |

3) 添加SpringMVC配置类：WebMvcConfig，让外部可直接访问Swagger文档

|  |
| --- |
| package com.me.yimu.pay.merchant.config;  import org.springframework.stereotype.Component;  import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;  import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;  /\*\*  \* @author Administrator  \* @version 1.0  \*\*/  @Component  public class WebMvcConfig implements WebMvcConfigurer {  /\*\*  \* 添加静态资源文件，外部可以直接访问地址  \*  \* @param registry  \*/  @Override  public void addResourceHandlers(ResourceHandlerRegistry registry) {  registry.addResourceHandler("/\*\*").addResourceLocations("classpath:/static/");  registry.addResourceHandler("swagger-ui.html")  .addResourceLocations("classpath:/META-INF/resources/");  registry.addResourceHandler("/webjars/\*\*")  .addResourceLocations("classpath:/META-INF/resources/webjars/");  }  } |

### 2.3.Swagger常用注解

在Java类中添加Swagger的注解即可生成Swagger接口文档，常用Swagger注解如下： @Api：修饰整个类，描述Controller的作用

@ApiOperation：描述一个类的一个方法，或者说一个接口

@ApiParam：单个参数的描述信息

@ApiModel：用对象来接收参数

@ApiModelProperty：用对象接收参数时，描述对象的一个字段

@ApiResponse：HTTP响应其中1个描述

@ApiResponses：HTTP响应整体描述

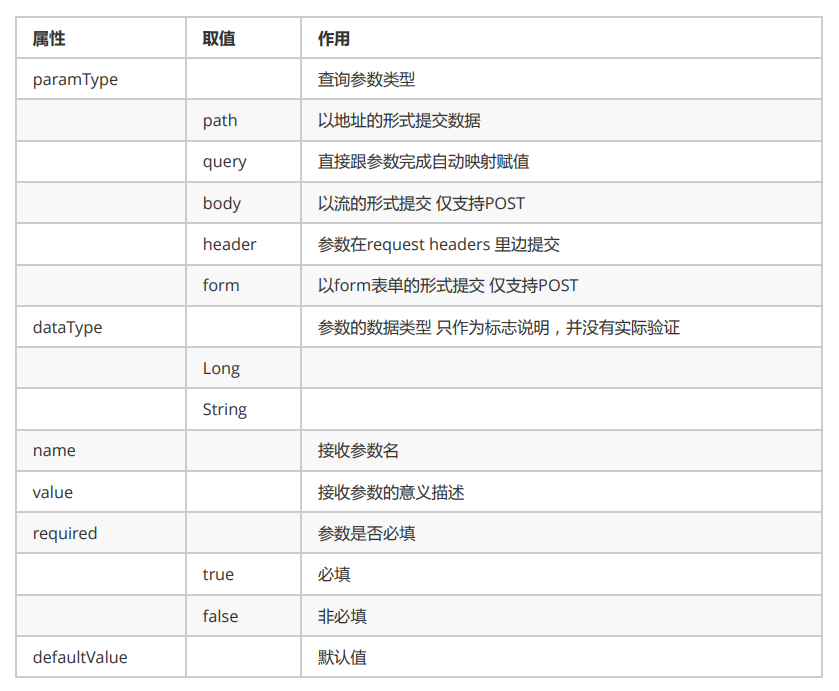
@ApiIgnore：使用该注解忽略这个API

@ApiError ：发生错误返回的信息

@ApiImplicitParam：一个请求参数

@ApiImplicitParams：多个请求参数的描述信息

@ApiImplicitParam属性：



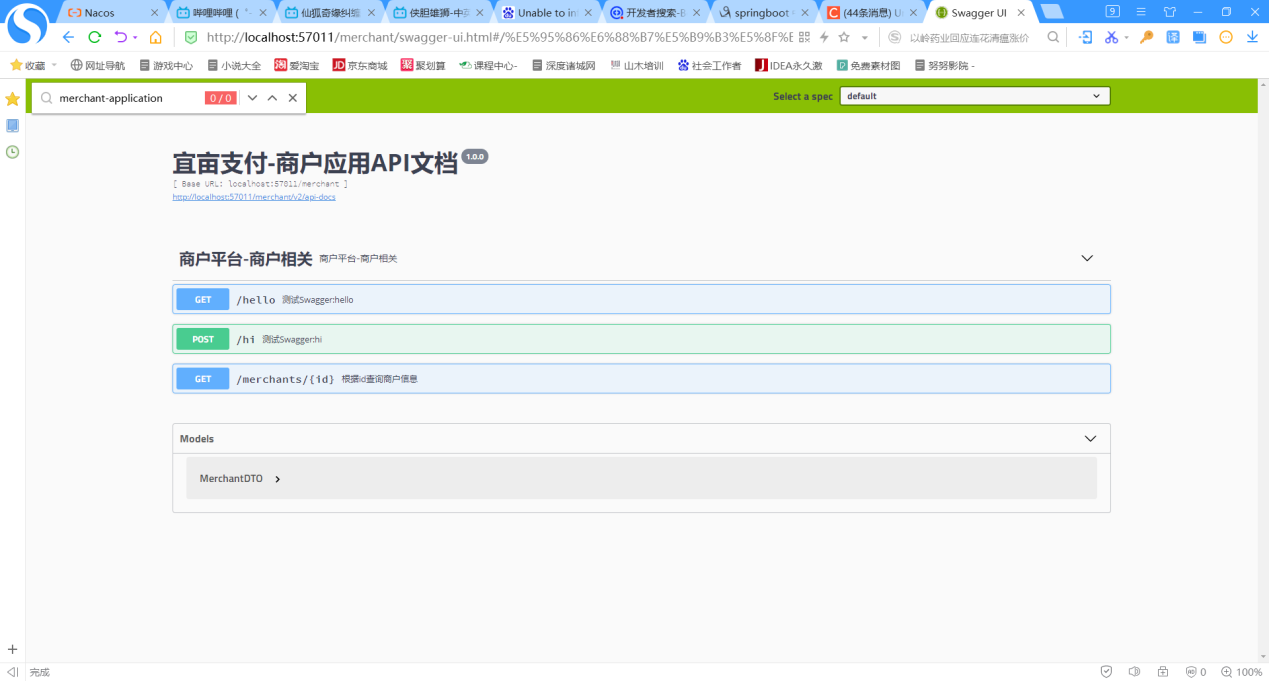
在MerchantController 中添加Swagger注解，代码如下所示：

|  |
| --- |
| package com.me.yimu.pay.merchant.controller;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import com.me.yimu.pay.merchant.api.MerchantService;  import com.me.yimu.pay.merchant.api.dto.MerchantDTO;  import io.swagger.annotations.Api;  import io.swagger.annotations.ApiImplicitParam;  import io.swagger.annotations.ApiOperation;  @Api(value = "商户平台‐商户相关", tags = "商户平台‐商户相关", description = "商户平台‐商户相关")  @RestController  @RequestMapping("/merchant")  public class MerchantController {  @org.apache.dubbo.config.annotation.Reference  MerchantService merchantService;    **@ApiOperation(value="根据id查询商户信息")**  **@GetMapping("/merchants/{id}")**  **public MerchantDTO queryMerchantById(@PathVariable("id") Long id){**  **MerchantDTO merchantDTO = merchantService.queryMerchantById(id);**  **return merchantDTO;**  **}**    **@ApiOperation("测试Swagger:hello")**  **@GetMapping(path = "/hello")**  **public String hello() {**  **return "hello";**  **}**    **@ApiOperation("测试Swagger:hi")**  **@ApiImplicitParam(name = "name", value = "姓名", required = true, dataType = "string")**  **@PostMapping(value = "/hi")**  **public String hi(String name) {**  **return "hi,"+name;**  **}**    } |

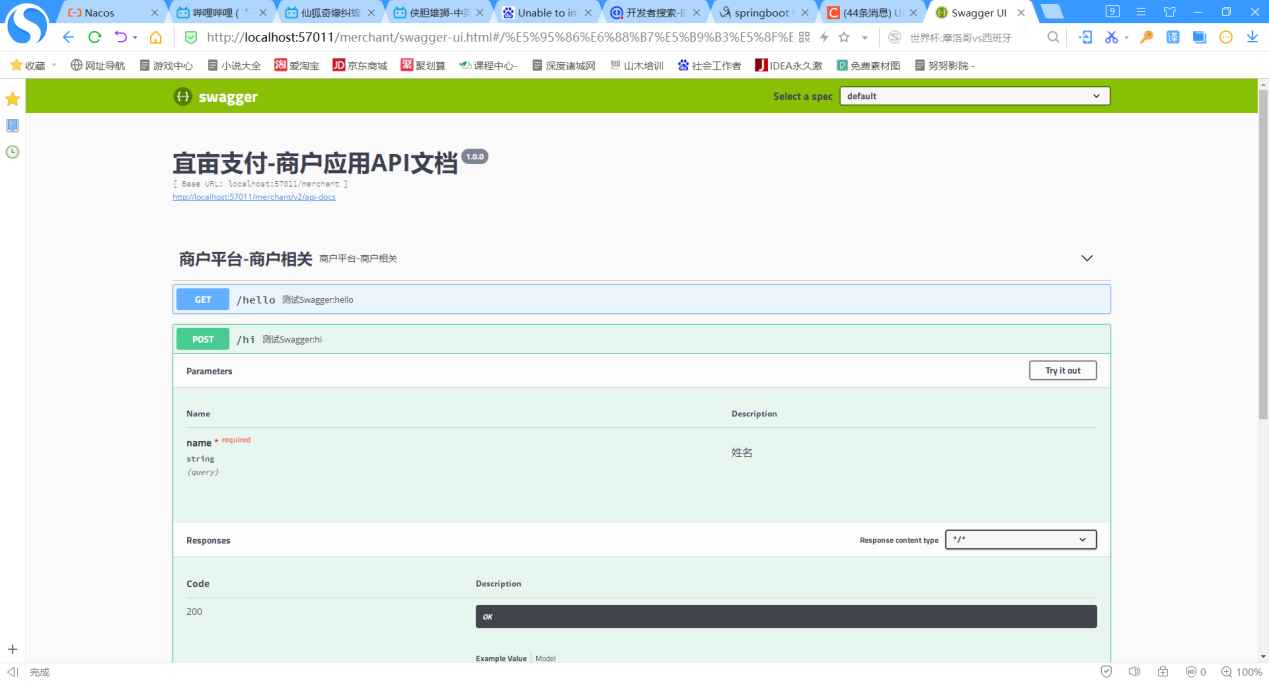
### 2.4.Swagger测试

1）启动商户应用和商户服务，

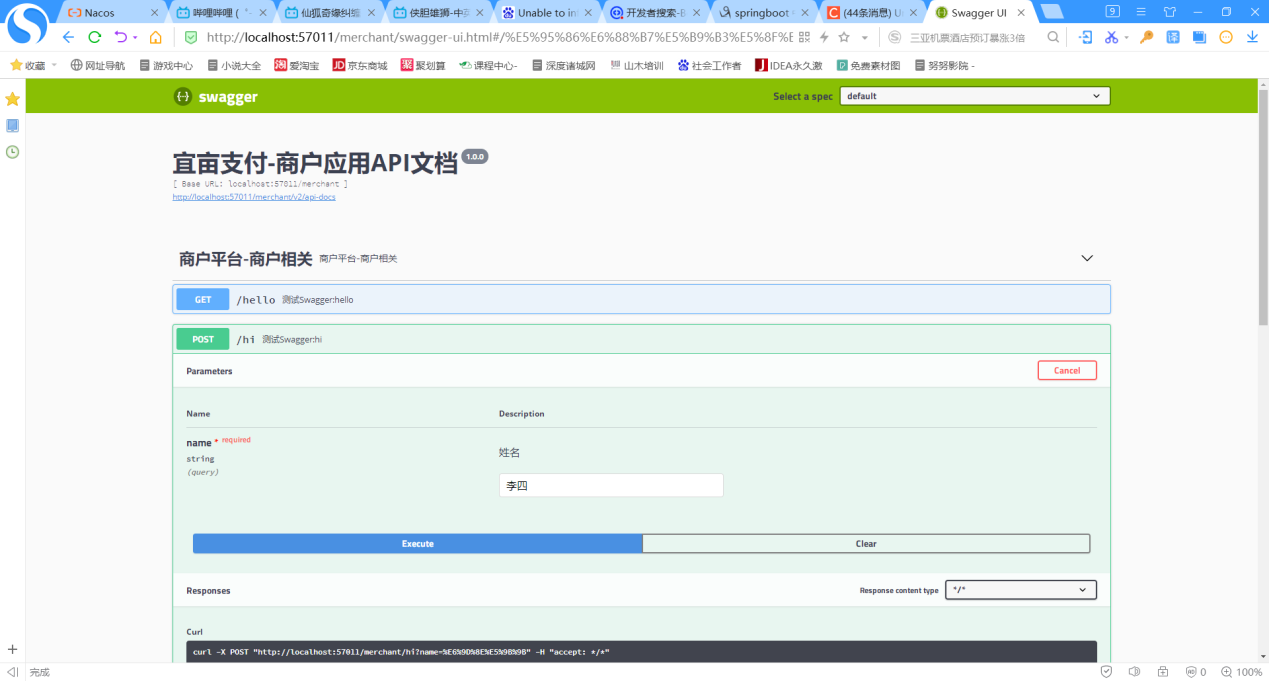
访问： <http://localhost:57011/merchant/swagger-ui.html>



2）点击其中任意一项即可打开接口详情，如下图所示：



3）点击“Try it out”开始测试，并录入参数信息，然后点击“Execute"发送请求，执行测试返回结果：“hi,李四”



Swagger生成API文档的工作原理：

1、yimupay-merchant-application启动时会扫描到SwaggerConfiguration类

2、在此类中指定了扫描包路径com.me.yimu.pay.merchant.controller，会找到在此包下及子包下标记有 @RestController注解的controller类

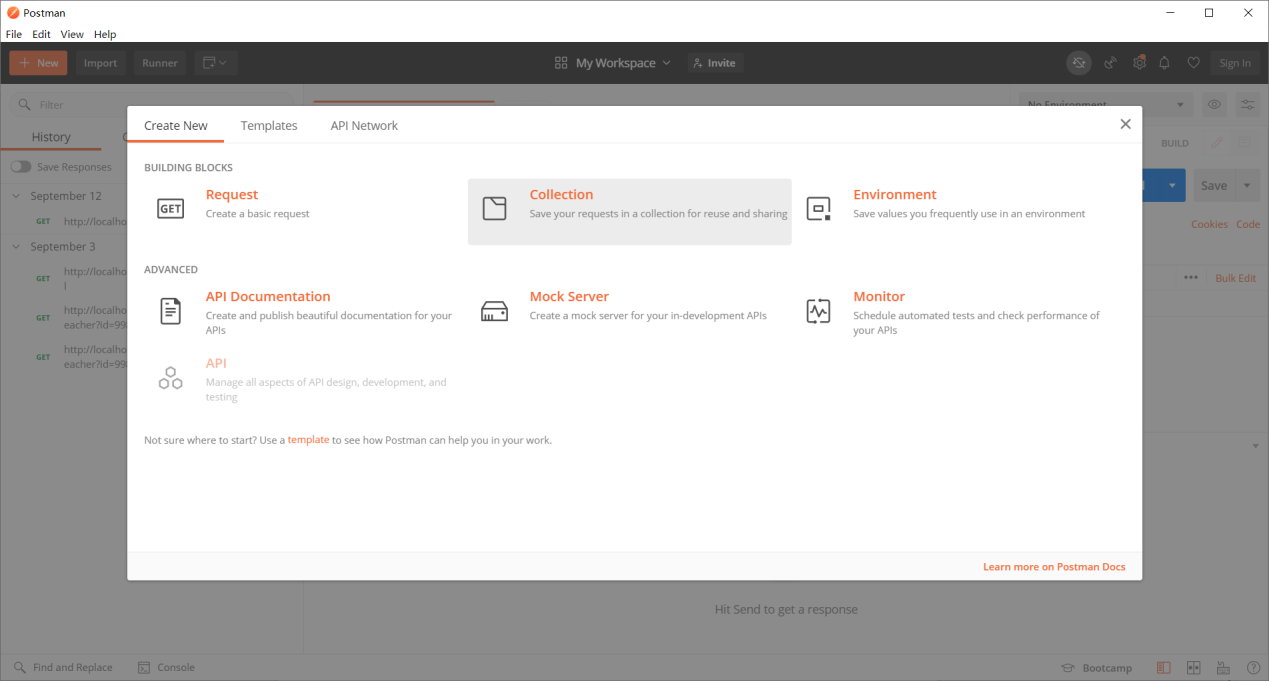
3、根据controller类中的Swagger注解生成API文档

## 3.Postmain接口工具

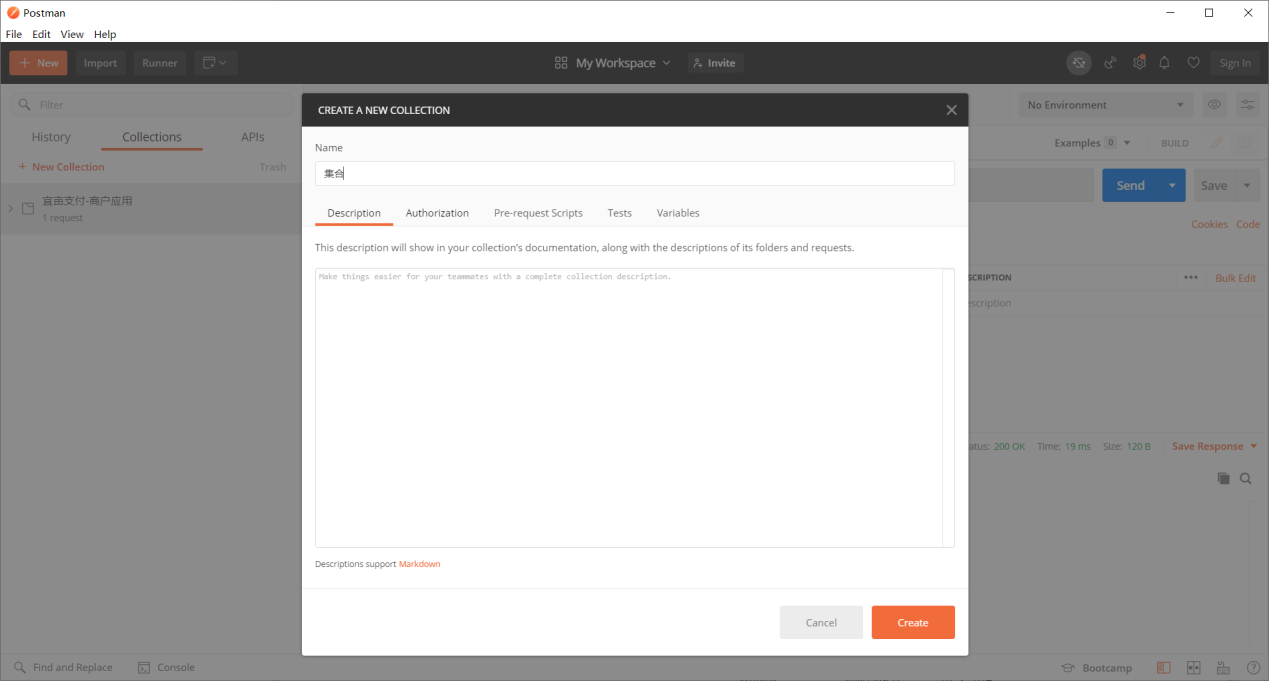
Postman是一款功能强大的http接口测试工具，使用Postman可以完成http各种请求的功能测试。作为服务器端 开发人员，当一个业务功能开发完毕后，应该用Postman进行功能测试。

1、请自行在本机安装Postman

2、新建集合(建议一个微服务新建一个对应的集合)：宜亩支付-商户应用



3. 在宜亩支付-商户应用集合中新建请求，并录入请求信息



小技巧：每个测试都可以进行保存(Ctrl+S)，以便于后续使用。

## 4.参照

<https://blog.csdn.net/m0_54355172/article/details/123543152>

<https://blog.csdn.net/weixin_42949480/article/details/126261525>

<https://www.cnblogs.com/aolia/p/12263158.html>

<https://blog.csdn.net/black_to_duck/article/details/40779525>

<https://qa.1r1g.com/sf/ask/671684051/>

https://blog.csdn.net/qq\_35006507/article/details/129040536?spm=1001.2101.3001.6650.1&utm\_medium=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EAD\_ESQUERY%7Eyljh-1-129040536-blog-115146824.pc\_relevant\_3mothn\_strategy\_and\_data\_recovery&depth\_1-utm\_source=distribute.pc\_relevant.none-task-blog-2%7Edefault%7EAD\_ESQUERY%7Eyljh-1-129040536-blog-115146824.pc\_relevant\_3mothn\_strategy\_and\_data\_recovery&utm\_relevant\_index=2