

闪聚支付 第1章 讲义-开发环境搭建

1 服务端搭建

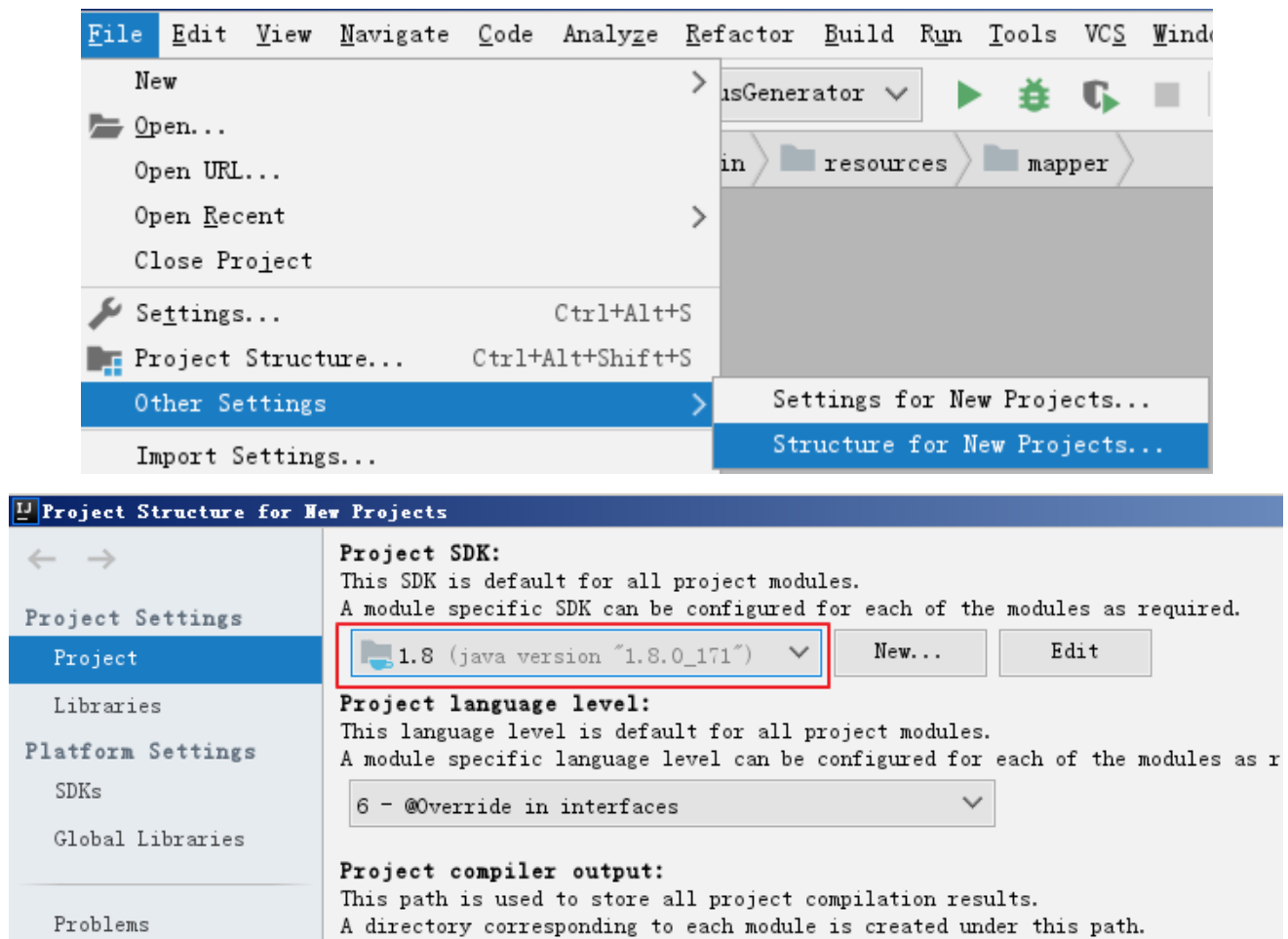
1.1 开发工具配置

服务端工程使用IntelliJ IDEA开发，IDEA开发环境配置如下：

1、首先配置JDK

安装JDK1.8，并设置环境变量。（过程略）

在IDEA配置JDK1.8



2、配置maven环境

安装maven3.3.9，过程略。

关于maven仓库有以下配置方法：

1) 已经有私服地址可在setting.xml中配置私服地址

在setting.xml中<servers>段增加：



```
<server>
  <id>maven-releases</id>
  <username>私服账号</username>
  <password>私服密码</password>
</server>
<server>
  <id>maven-snapshots</id>
  <username>私服账号</username>
  <password>私服密码</password>
</server>
在<mirrors>添加：
  <mirror>
    <id>nexus</id>
    <mirrorOf>*</mirrorOf>
    <url>私服地址</url>
  </mirror>
```

2) 从中央仓库下载

不作任何配置，maven自行从中央仓库下载，也可使用阿里官网Maven库：

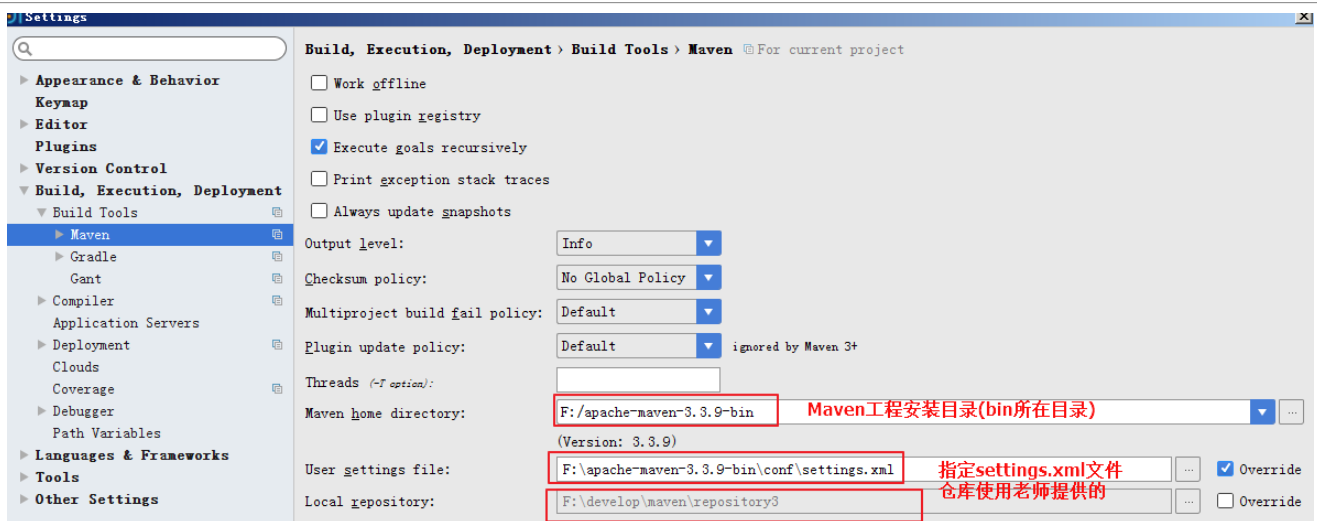
```
<mirror>
  <id>alimaven</id>
  <name>aliyun maven</name>
  <url>http://maven.aliyun.com/nexus/content/groups/public</url>
  <mirrorOf>central</mirrorOf>
</mirror>
```

3) 使用本地仓库

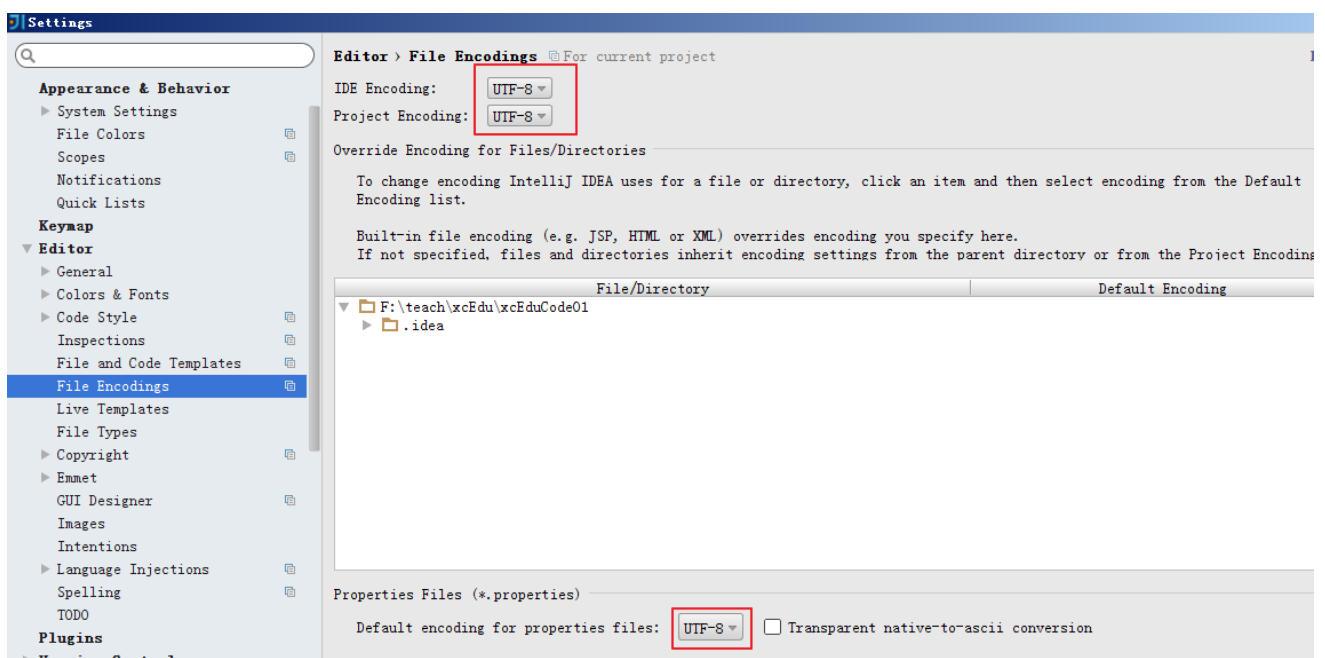
本教程使用本地仓库配置。拷贝老师提供的maven仓库并解压，在maven的setting.xml文件中配置本地仓库的路径，路径位置不要使用中文，配置例子如下：

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
    | The path to the local repository maven will use to store artifacts.
    |
    | Default: ${user.home}/.m2/repository -->
  <localRepository>F:\develop\maven\repository3</localRepository>
```

在IDEA中选择setting.xml文件，自动找到本地仓库目录，如下图：

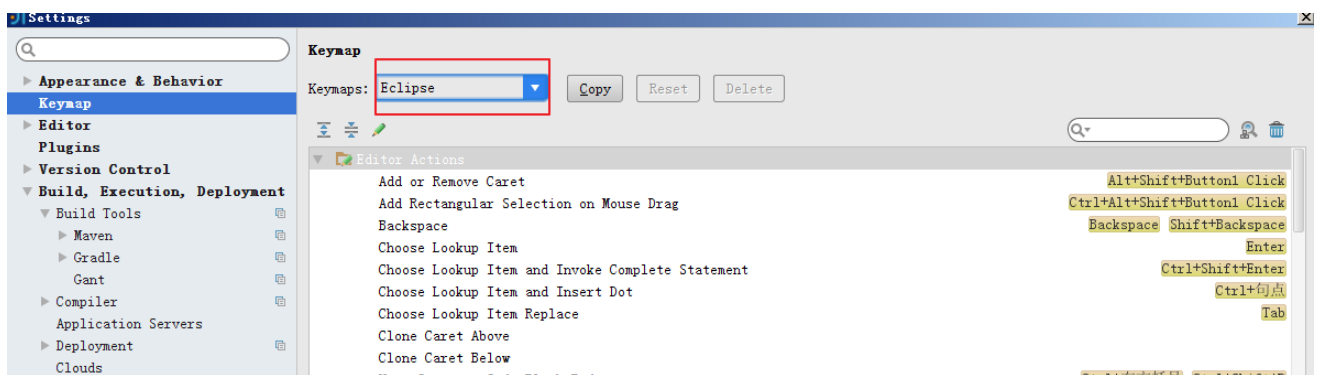


3、配置编码



4、配置快捷键

IDEA可以集成Eclipse的快捷键

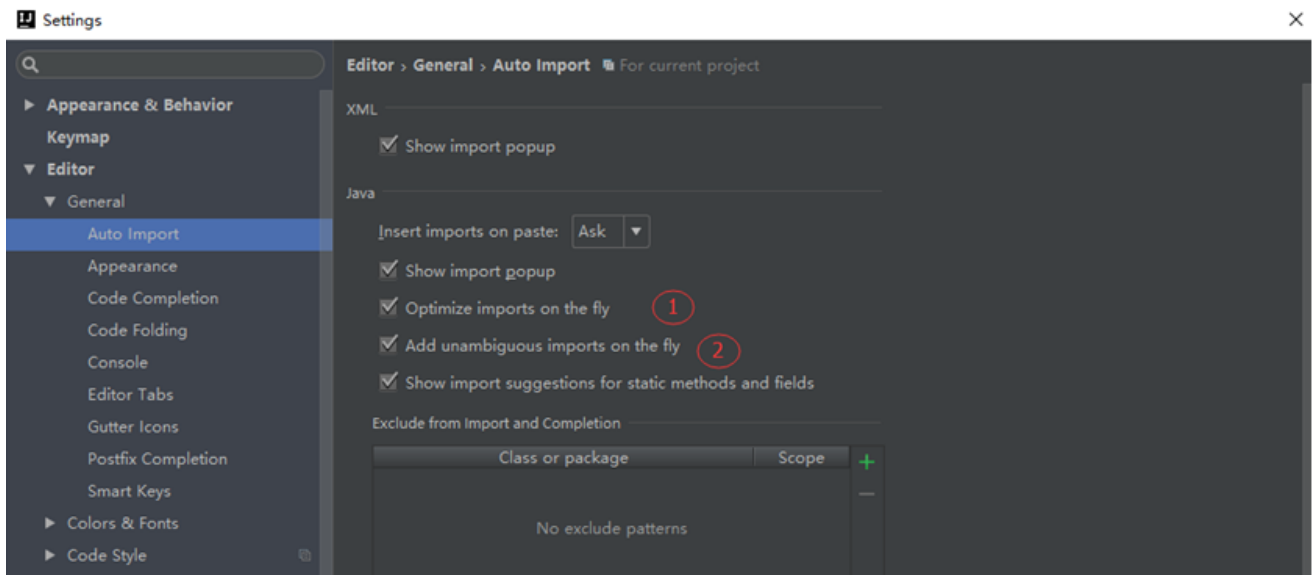


如需自定义则点击“copy”复制一份进行修改

5、自动导入包 快捷方式：

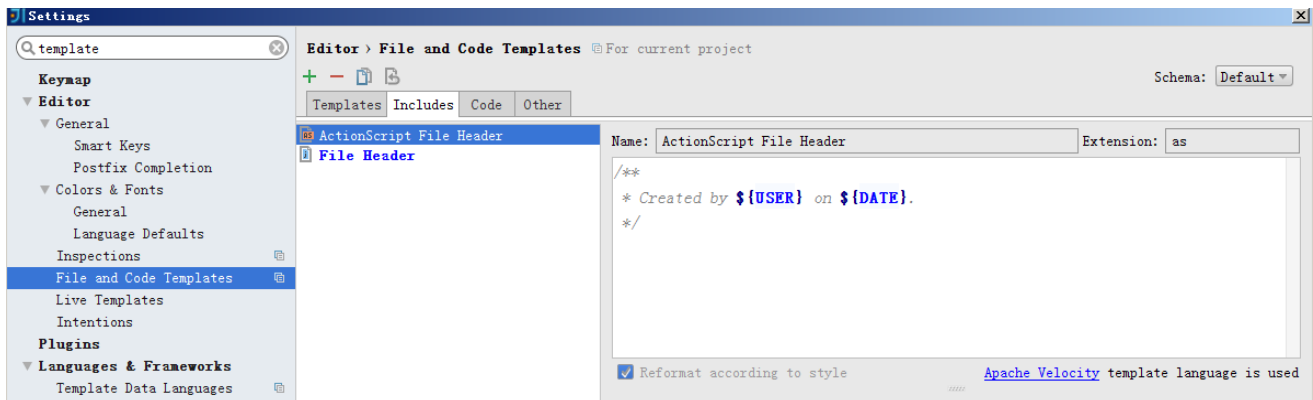
idea可以自动优化导入包，但是有多个同名的类调用不同的包，必须自己手动Alt+Enter设置

设置idea导入包



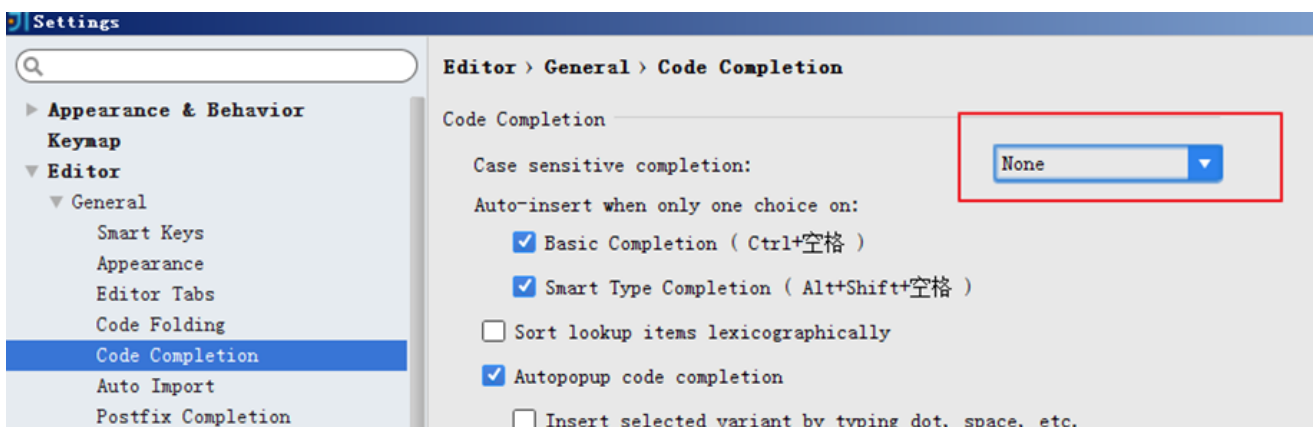
6、代码模板

自定义自己的代码模板



7、提示忽略大小写

默认IDEA的提示是区分大小写的，这里设置为提示忽略大小写



9 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/>

比如上节创建的用户Test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：

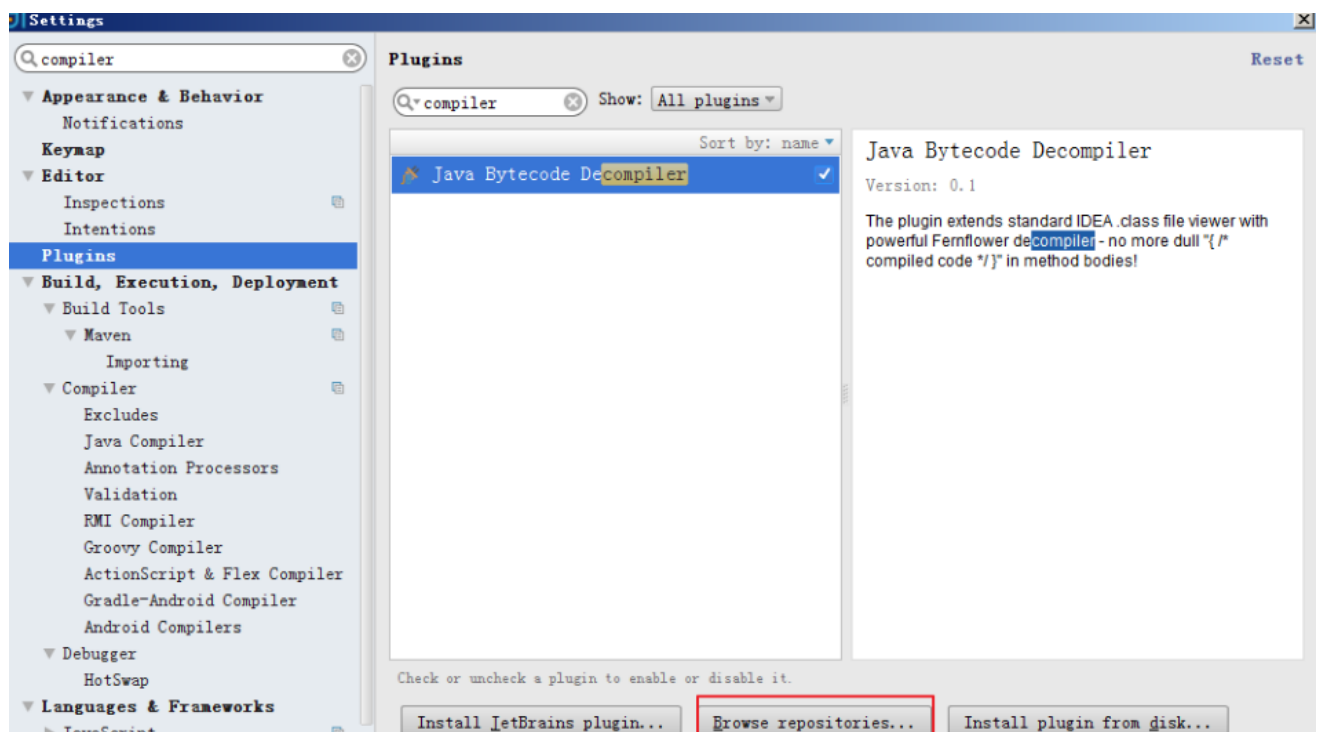
1、在项目中添加Lombok的依赖

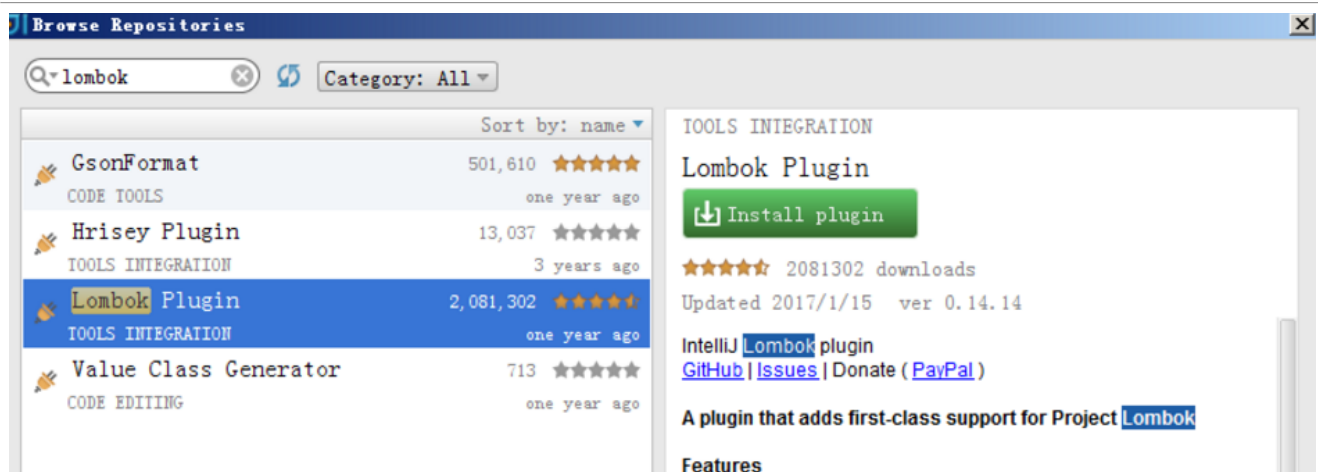
作用：项目在编译时根据Lombok注解生成通用方法，依赖如下：

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
</dependency>
```

2、在IDEA开发工具中添加Lombok插件

作用：使用IDEA开发时根据Lombok注解生成通用方法，不报错。





10、配置虚拟机内存

修改idea64.exe.vmoptions (64位电脑选择此文件)

一个例子，电脑内存8G，设置如下：

```
-Xms1024m -Xmx4096m -XX:MaxPermSize=1024m -XX:ReservedCodeCacheSize=1024m
```

1.2 Nacos服务发现与配置中心

微服务开发需要构建服务发现中心、配置中心，本项目采用Nacos来实现。

详见“Nacos服务发现与配置中心v1.0.pdf”

1.3 Mybatis Plus

本项目数据库使用 mysql-community-5.7，请自行安装MySQL数据库。

本项目持久层采用Mybatis Plus作为技术构架，Mybatis Plus是在Mybatis基础上作了很好的封装，方便系统开发。

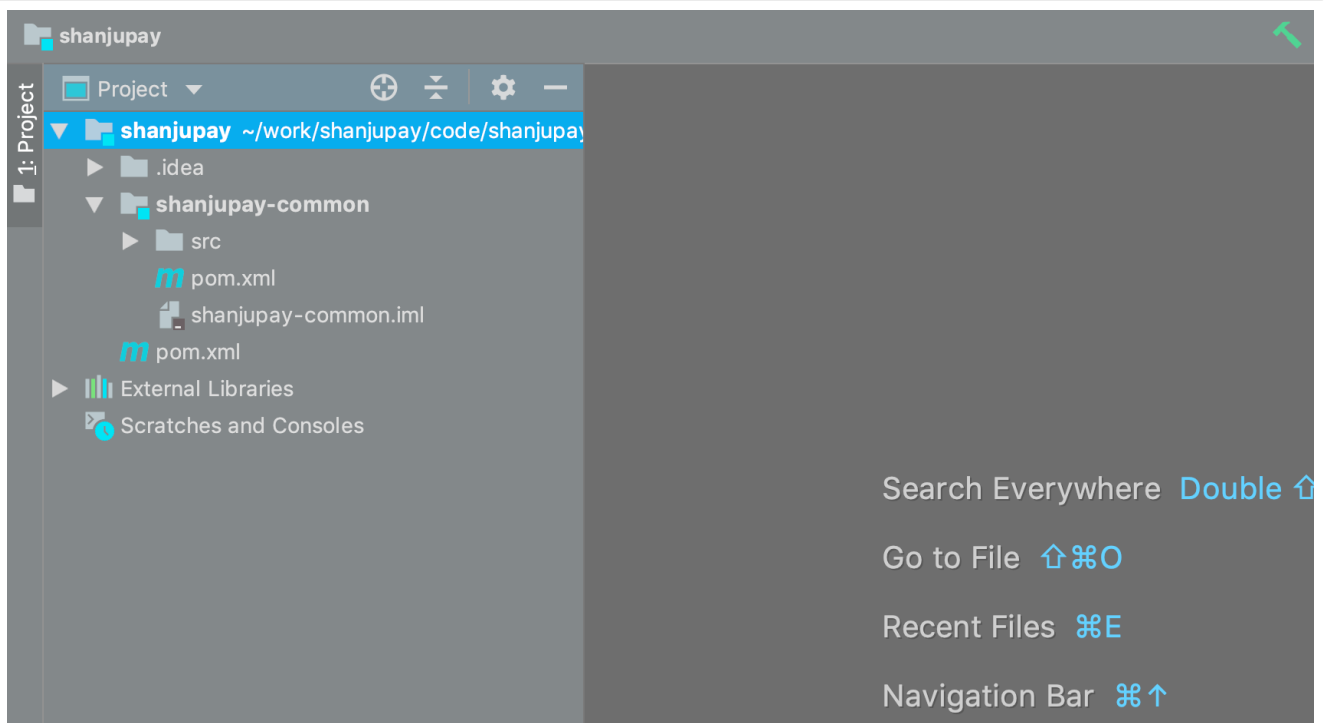
学习Mybatis Plus详见“Mybatis-Plus讲义v1.0.md”。

1.4 导入基础工程

解压“shanjupay基础工程.zip”，将基础工程导入IDEA。

打开IDEA，点击File->Open后选择之前解压的shanjupay目录

如下是基础工程结构图：



| 工程名 | 说明 |
|------------------|--------------------------|
| shanjupay | 闪聚支付父工程 |
| shanjupay-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

使用客户端连接MySQL，执行shanjupay-init.sql，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 | 数据内容 |
|----------------------------|---------|
| shanjupay_merchant_service | 用户中心数据 |
| shanjupay_transaction | 交易服务数据库 |

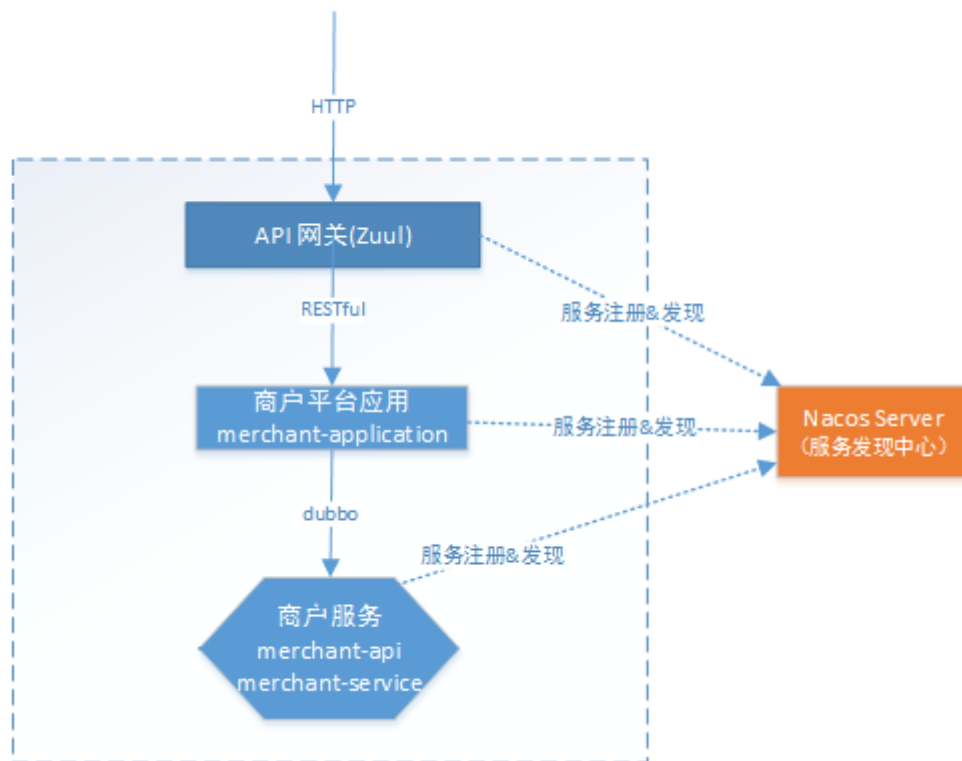
备注：数据库、表中的字段以及表关系会在后续开发过程中随用随讲。

1.6 搭建项目服务

本节搭建如下项目工程：

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

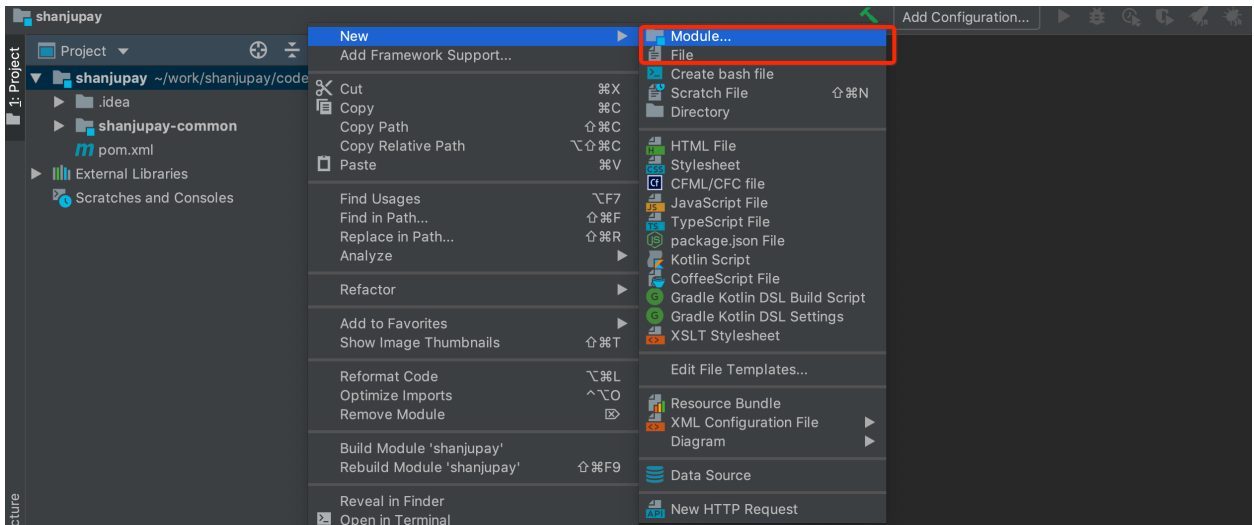
三个工程在架构中的位置如下：



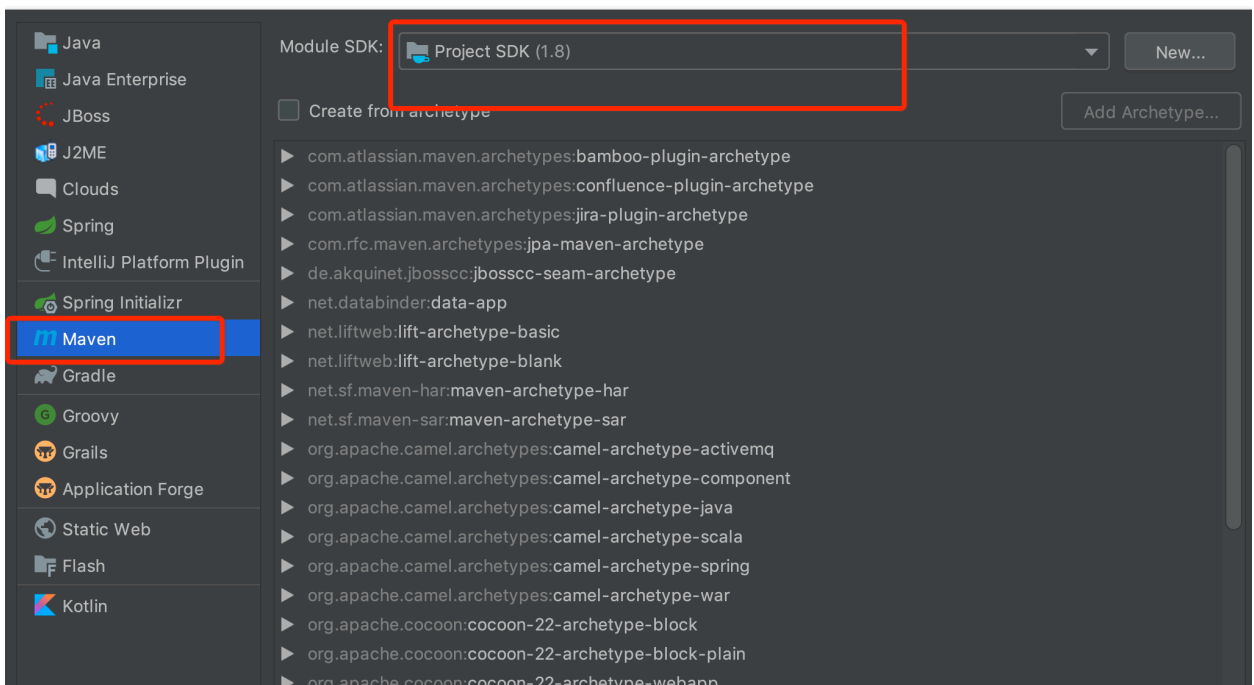
1.6.1 搭建商户平台应用工程

在基础工程的基础上创建商户平台应用工程。

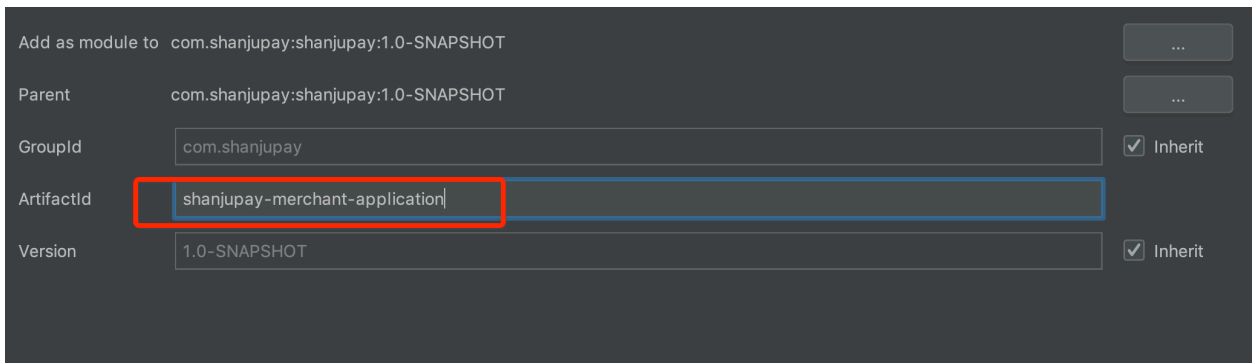
1. 选中shanjupay工程，右键选择新建Module

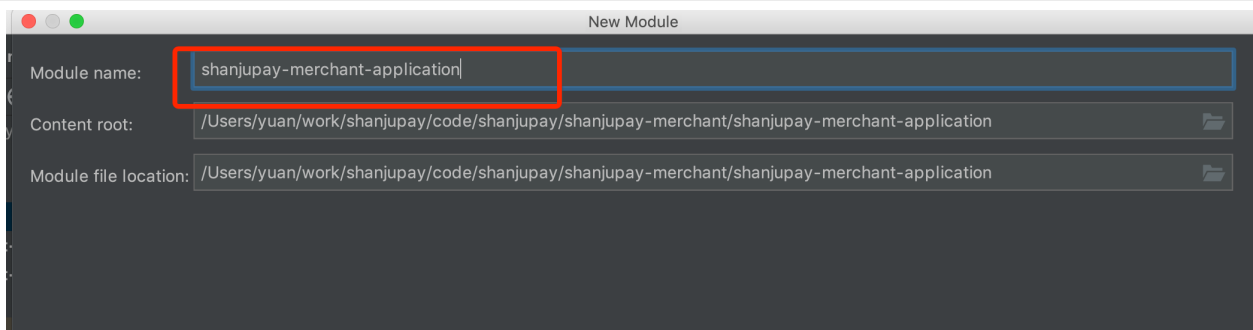


2. 选择Maven和DK1.8



3. 填写ArtifactId: shanjupay-merchant-application, 设置Module Name: shanjupay-merchant-application





4. 点击Finish完成创建

5. 添加依赖，完善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>shanjupay</artifactId>
        <groupId>com.shanjupay</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>shanjupay-merchant-application</artifactId>

    <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>
    </dependencies>
</project>
```



```
<!-- 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>

</dependencies>
</project>
```

6. 配置bootstrap.yml

```
server:
  port: 57010 #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
cloud:
  nacos:
    discovery:
      server-addr: ${nacos.server.addr}
      namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
      cluster-name: DEFAULT
    config:
      server-addr: ${nacos.server.addr} # 配置中心地址
      file-extension: yaml
      namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8 # 命令行注入
      group: SHANJUPAY_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
#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.shanjupay
  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: 3000
    retries: -1

logging:
  config: classpath:log4j2.xml
```

7. 在Nacos中添加spring-boot-http.yaml配置，Group：COMMON_GROUP

这里统一使用dev命名空间，没有此命名空间则在nacos中创建。

```
#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
```

Nacos配置页面如下所示：

* Data ID: spring-boot-http.yaml

* Group: COMMON_GROUP

更多高级选项

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 ? :

```
1 spring:
2   http:
3     encoding:
4       charset: UTF-8
5       force: true
6       enabled: true
7 server:
8   tomcat:
9     remote_ip_header: x-forwarded-for
10    protocol_header: x-forwarded-proto
11  servlet:
12    context-path: /
13    use-forward-headers: true
14
15 management:
16   endpoints:
17     web:
```

8. 在Nacos中添加merchant-application.yaml配置，Group：SHANJUPAY_GROUP

#覆盖访问路径

```
server:
  servlet:
    context-path: /merchant
```

#启用Swagger

```
swagger:
  enable: true
```

* Data ID: merchant-application.yaml

* Group: SHANJUPAY_GROUP

更多高级选项

描述: 商户平台

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 ? :

```
1 server:
2   servlet:
3     context-path: /merchant
4
5 swagger:
6   enable: true
7
```

1. 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>
```

2. 添加启动类

- 1) 添加包：com.shanjupay.merchant
- 2) 新建启动类：MerchantApplicationBootstrap

```
package com.shanjupay.merchant;

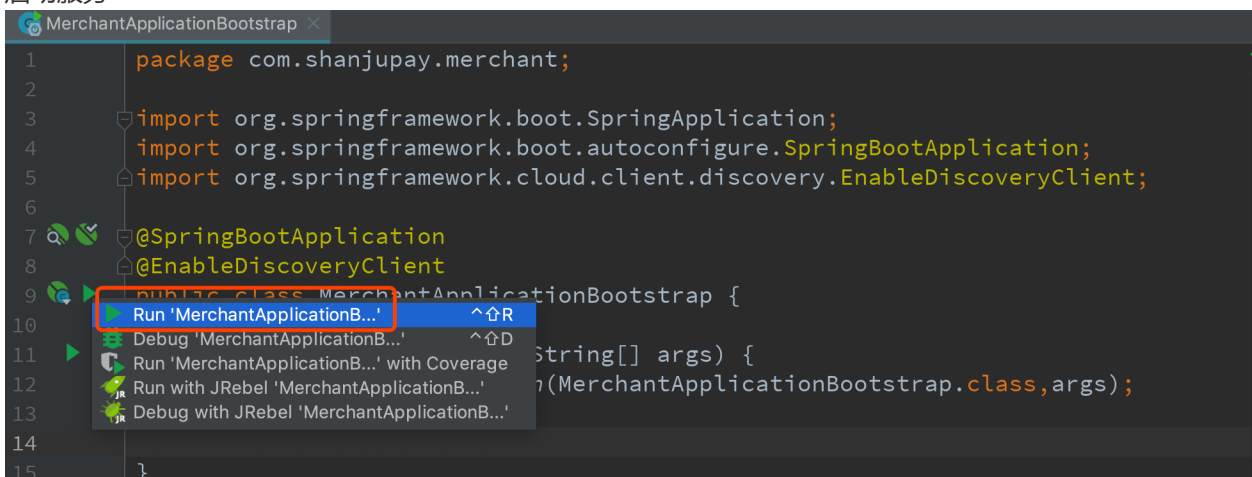
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class MerchantApplicationBootstrap {

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

}
```

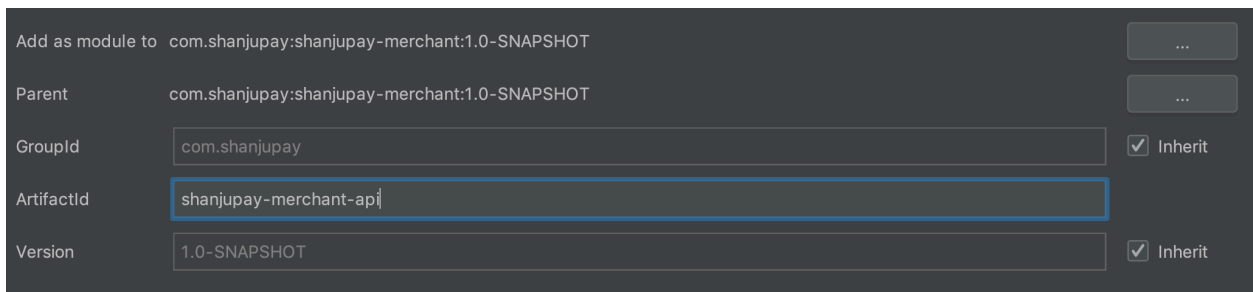
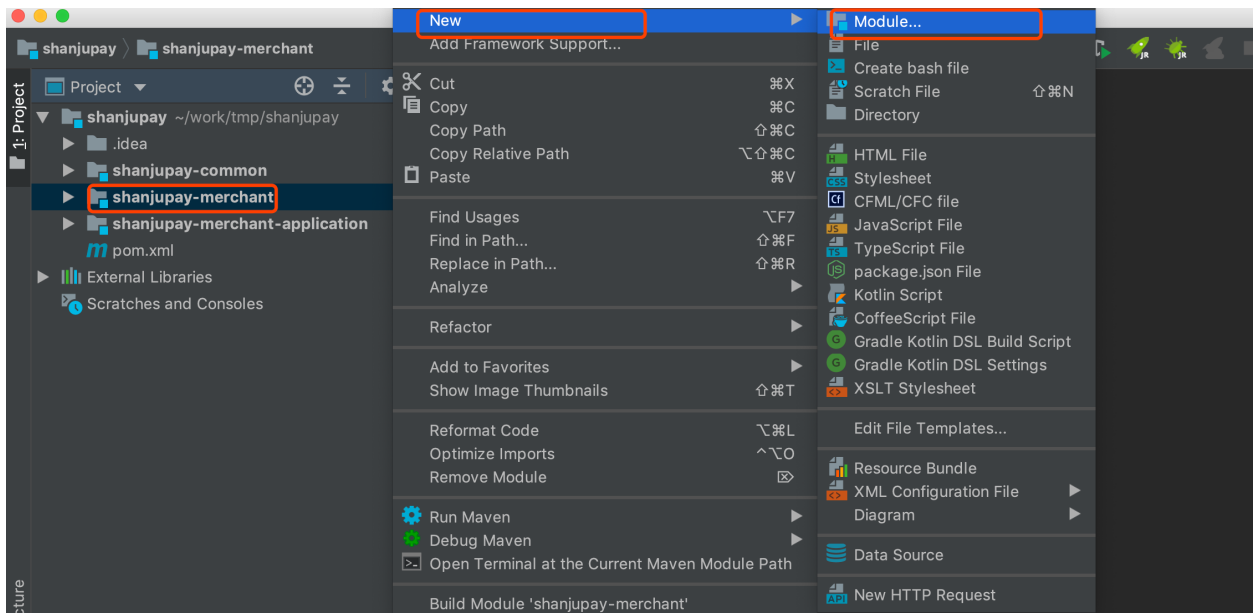
3. 启动服务



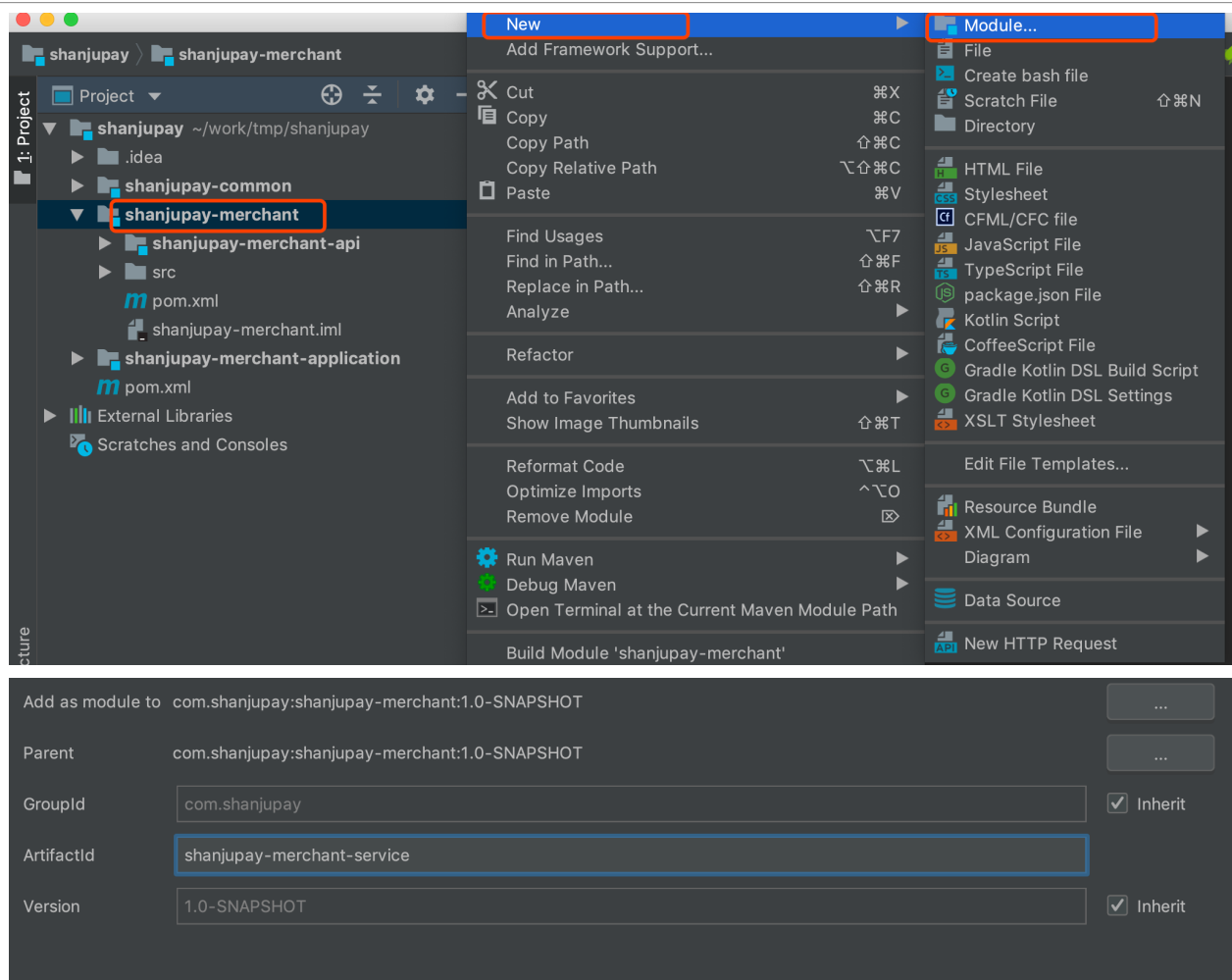
1.6.2 搭建商户服务工程

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。

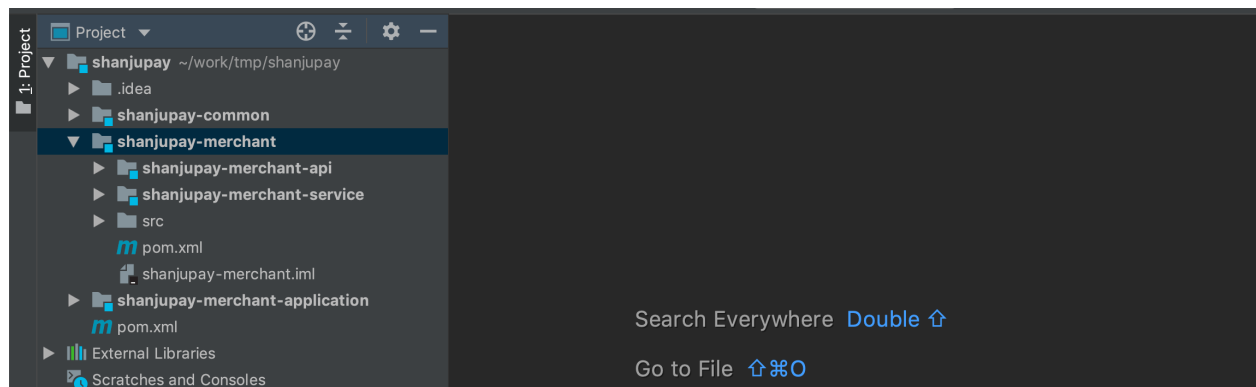
1. 按上述步骤创建商户服务工程 shanjupay-merchant
2. 在shanjupay-merchant下创建shanjupay-merchant-api，选中shanjupay-merchant工程右键-》New-》Module，填写artifactId和Module Name：shanjupay-merchant-api



3. 在shanjupay-merchant下shanjupay-merchant-service，选中shanjupay-merchant工程右键-》New-》Module，填写artifactId和Module Name：shanjupay-merchant-service



4. 两个工程创建成功后，项目整体目录结构如下所示：



5. 完善pom.xml

shanjupay-merchant-api模块依赖如下：

```
<?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>shanjupay-merchant</artifactId>
    <groupId>com.shanjupay</groupId>
    <version>1.0-SNAPSHOT</version>
```



```
</parent>
<modelVersion>4.0.0</modelVersion>

<artifactId>shanjupay-merchant-api</artifactId>

<dependencies>
  <dependency>
    <groupId>com.shanjupay</groupId>
    <artifactId>shanjupay-common</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>

</project>
```

shanjupay-merchant-service模块依赖

```
<?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>shanjupay-merchant</artifactId>
    <groupId>com.shanjupay</groupId>
    <version>1.0-SNAPSHOT</version>
  </parent>
  <modelVersion>4.0.0</modelVersion>

  <artifactId>shanjupay-merchant-service</artifactId>

  <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>
  </dependencies>
```



```
<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>
```

3. 在resources目录下添加日志配置文件：log4j2.xml

4. 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内容如下

```
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: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8
        cluster-name: DEFAULT
```

```
config:
  server-addr: ${nacos.server.addr} # 配置中心地址
  file-extension: yaml
  namespace: a1f8e863-3117-48c4-9dd3-e9ddc2af90a8 # 命令行注入
  group: SHANJUPAY_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

dubbo:
  scan:
    # dubbo 服务扫描基准包
    base-packages: com.shanjupay
  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
```

3. 在Nacos中添加merchant-service.yaml配置，Group：SHANJUPAY_GROUP

```
# 覆盖spring-boot-http.yaml的项目
server:
  servlet:
    context-path: /merchant-service
```

* Data ID:


* Group:

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 :

```
1 # 覆盖spring-boot-http.yaml的项目
2 server:
3   servlet:
4     context-path: /merchant-service
5
```

4. 添加商户中心服务启动类

```
package com.shanjupay.merchant;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
public class MerchantBootstrap {

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

1.7 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。

1.7.1 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shanjupay_merchant_service数据库

```
// 商户服务
dataSourceConfig
    .setUrl("jdbc:mysql://127.0.0.1:3306/shanjupay_merchant_service?
serverTimezone=Asia/Shanghai");
```

2)设置包路径

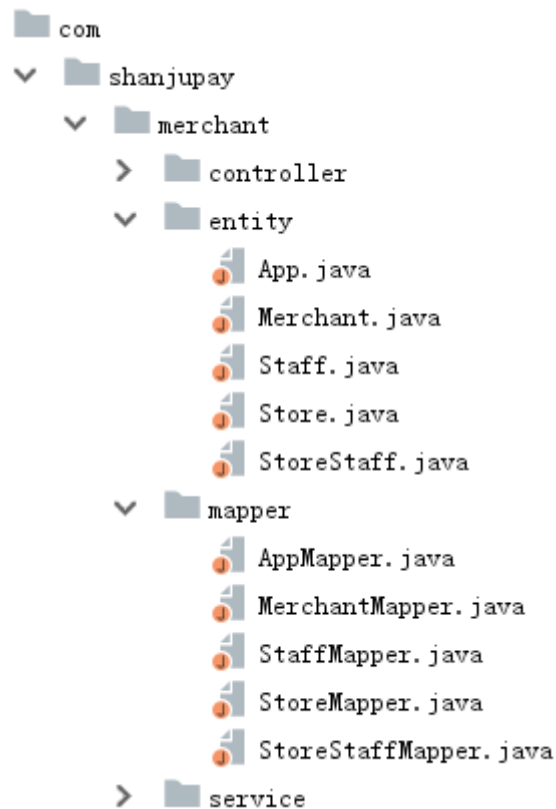
```
// 生成包配置
PackageConfig packageConfig = new PackageConfig();
packageConfig.setParent("com.shanjupay");
```

3) 运行生成类，输入模块名：merchant

请输入模块名：

merchant

生成成功：



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

1.7.2 Mybatis-Plus配置

1.7.2.1 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:
  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.shanjupay.*.service.*
```

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-starter-druid.yaml的项目
spring:
  datasource:
    druid:
      url: jdbc:mysql://127.0.0.1:3306/shanjupay_merchant_service?
      useUnicode=true&characterEncoding=UTF-8&serverTimezone=Asia/Shanghai&useSSL=false
      username: root
      password: yourpassword
```

3. 应用配置到项目中：bootstrap.yml

```
ext-config:
  -
    refresh: true
    data-id: spring-boot-starter-druid.yaml # spring boot starter druid配置
    group: COMMON_GROUP # 通用配置组
```

1.7.2.2 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.shanjupay.user.entity
  mapper-locations: classpath:com/shanjupay/**/*.xml
```

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

```
# 覆盖spring-boot-mybatis-plus.yaml的项目
mybatis-plus:
  typeAliasesPackage: com.shanjupay.merchant.entity
  mapper-locations: classpath:com/shanjupay/**/*.xml
```

3. 应用配置到项目中：bootstrap.yml

```
ext-config:
  -
    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.shanjupay.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;

/**
 * <P>
 * Mybatis-Plus 配置
 * </P>
 */
@Configuration
@MapperScan("com.shanjupay.**.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();  
}  
}
```

1.7.3 创建接口

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

```
package com.shanjupay.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shanjupay-merchant-api工程 定义MerchantDTO

```
package com.shanjupay.merchant.api.dto;  
  
import io.swagger.annotations.ApiModel;  
import io.swagger.annotations.ApiModelProperty;  
import lombok.Data;  
  
import java.io.Serializable;  
  
/**  
 * @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 Long 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;
}
```

@ApiModelProperty和 @ApiModelProperty是Swagger注解后边会学习。

1.7.4 创建接口实现

在shanjupay-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shanjupay_merchant_service数据库的merchant查询数据。

```
package com.shanjupay.merchant.service;

import com.shanjupay.merchant.api.MerchantService;
import org.apache.dubbo.config.annotation.Service;

@Service
public class MerchantServiceImpl implements MerchantService {

    @Autowired
    MerchantMapper merchantMapper;

    /**
     * 根据ID查询详细信息
     *
     * @param merchantId
     * @return
     * @throws BusinessException
     */
    @Override
    public MerchantDTO queryMerchantById(Long merchantId){
        Merchant merchant = merchantMapper.selectById(merchantId);
        MerchantDTO merchantDTO = new MerchantDTO();
        merchantDTO.setId(merchant.getId());
        //设置其它属性...
        return merchantDTO;
    }
}
```

1.7.5 应用层

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

```
<!-- 商户服务API -->
<dependencies>
    <dependency>
        <groupId>com.shanjupay</groupId>
        <artifactId>shanjupay-merchant-api</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
```

在shanjupay-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.shanjupay.merchant.controller;

import com.shanjupay.merchant.api.MerchantService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
```

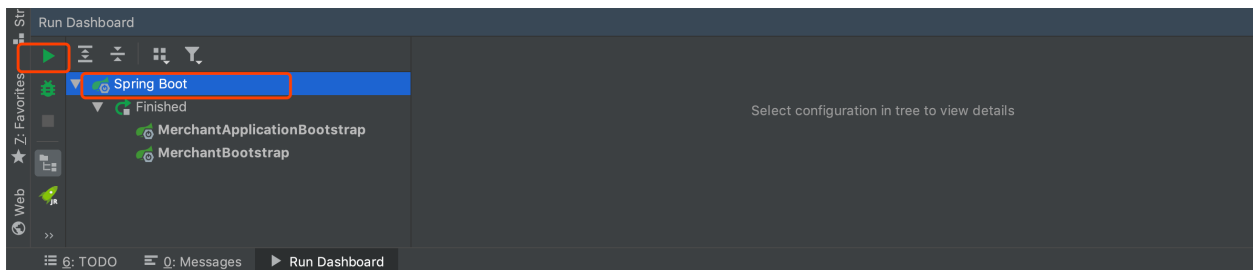
```
import lombok.extern.slf4j.Slf4j;
import org.apache.dubbo.config.annotation.Reference;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class MerchantController {

    @Reference
    private MerchantService merchantService;

    @GetMapping("/merchants/{id}")
    public MerchantDTO queryMerchantById(@PathVariable("id") Long id) {
        MerchantDTO merchantDTO = merchantService.queryMerchantById(id);
        return merchantDTO;
    }
}
```

1. 启动商户平台应用和商户服务



2. 访问<http://localhost:57010/merchant/merchants/>具体的id值

测试查询商户

如果merchant表没有数据可手动添加后再行测试。

2 接口相关工具

2.1 API接口文档利器：Swagger

2.1.1 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。

2.1.2 SpringBoot集成Swagger

1. 在shanjupay-common项目中添加依赖，只需要在shanjupay-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shanjupay-common。

```
<!-- Swagger依赖 -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
</dependency>

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

2. 在shanjupay-merchant-application工程的config包中添加一个Swagger配置类

```
package com.shanjupay.merchant.config;

import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Conditional;
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;

@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.shanjupay.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("闪聚支付-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.shanjupay.merchant.config;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@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/");
    }
}
```

2.1.3 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

@ApiResponse : 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 :

| 属性 | 取值 | 作用 |
|--------------|--------|-------------------------|
| paramType | | 查询参数类型 |
| | path | 以地址的形式提交数据 |
| | query | 直接跟参数完成自动映射赋值 |
| | body | 以流的形式提交 仅支持POST |
| | header | 参数在request headers 里边提交 |
| | form | 以form表单的形式提交 仅支持POST |
| dataType | | 参数的数据类型 只作为标志说明，并没有实际验证 |
| | Long | |
| | String | |
| name | | 接收参数名 |
| value | | 接收参数的意义描述 |
| required | | 参数是否必填 |
| | true | 必填 |
| | false | 非必填 |
| defaultValue | | 默认值 |

上边的属性后边编写程序时用到哪个我再详细讲解，下边写一个swagger的简单例子，我们在MerchantController中添加Swagger注解，代码如下所示：

```
@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
public class MerchantController {

    @Reference
    private MerchantService merchantService;
```



```
@ApiOperation("根据id查询商户")
@GetMapping("/merchants/{id}")
public MerchantDTO queryMerchantById(@PathVariable("id") Long id) {
    MerchantDTO merchantDTO = merchantService.queryMerchantById(id);
    return merchantDTO;
}

@ApiOperation("测试")
@GetMapping(path = "/hello")
public String hello(){
    return "hello";
}

@ApiOperation("测试")
@ApiImplicitParam(name = "name", value = "姓名", required = true, dataType = "string")
@PostMapping(value = "/hi")
public String hi(String name) {
    return "hi,"+name;
}
}
```

2.1.4 Swagger测试

1. 启动商户应用和商户中心服务，访问：<http://localhost:57010/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：

闪聚支付-商户应用API文档 1.0.0

[Base URL: localhost:57010/merchant]
<http://localhost:57010/merchant/v2/api-docs>

商户平台-商户相关 商户平台-商户相关

GET /hello 测试

POST /hi 测试

Parameters

| Name | Description |
|---|-------------|
| name * required string (query) | 姓名 |

Responses

Response content type */*

Try it out

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”

POST /hi 测试

Parameters

Cancel

| Name | Description |
|---|-------------|
| name * required string (query) | 姓名 |

李四

Execute

Execute Clear

Responses

Response content type */*

Curl

```
curl -X POST "http://localhost:57010/merchant/hi?name=%E6%9D%8E%E5%9B%9B" -H "accept: */*"
```

Request URL

```
http://localhost:57010/merchant/hi?name=%E6%9D%8E%E5%9B%9B
```

Server response

| Code | Details |
|------|---|
| 200 | <p>Response body</p> <p>hi,李四</p> <p>Download</p> |

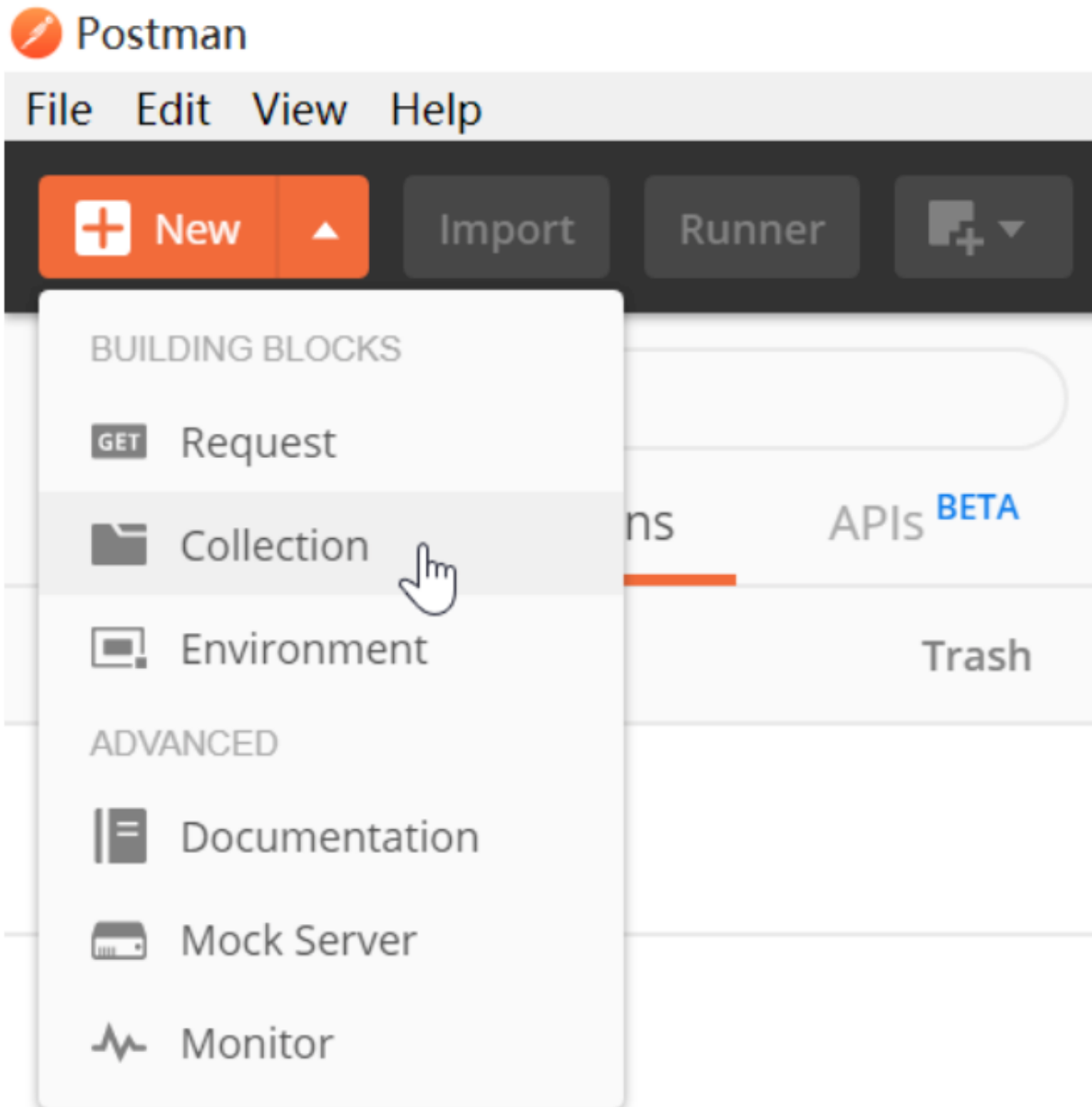
Swagger生成API文档的工作原理：

- 1、shanjupay-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.shanjupay.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

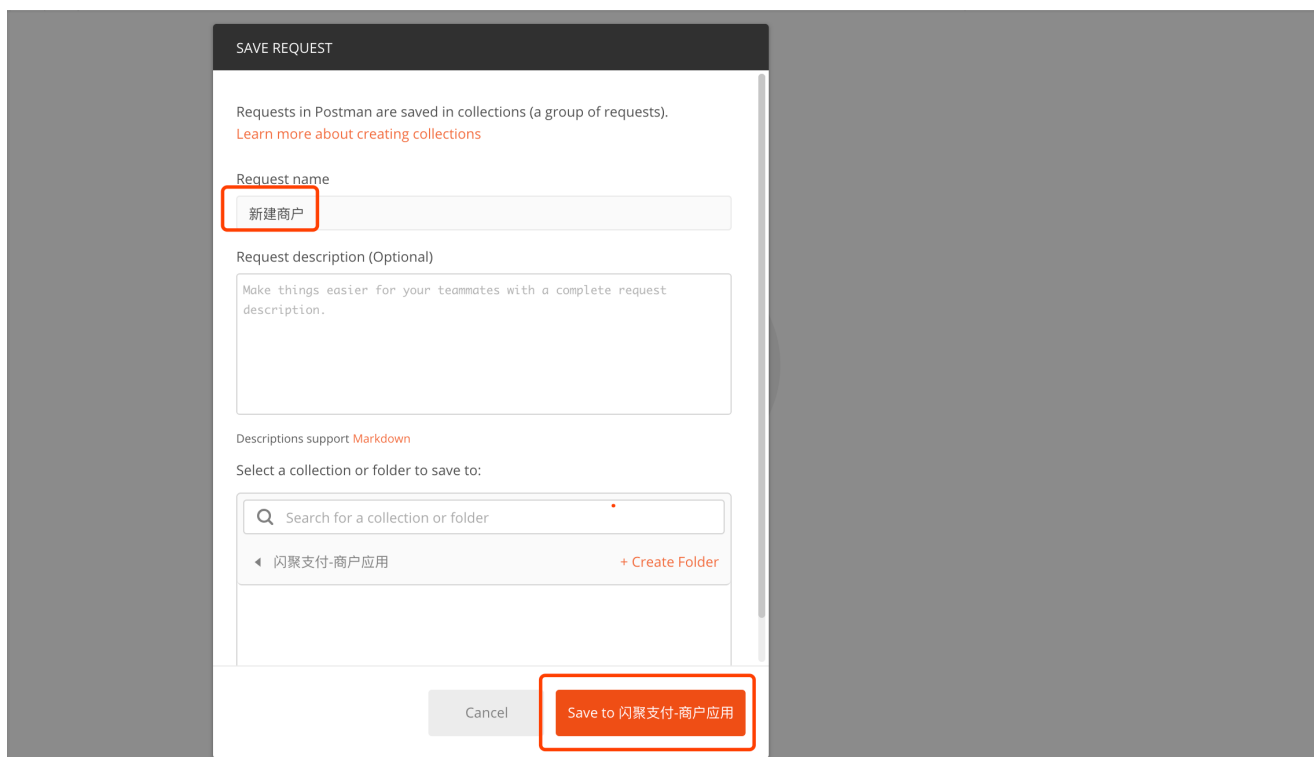
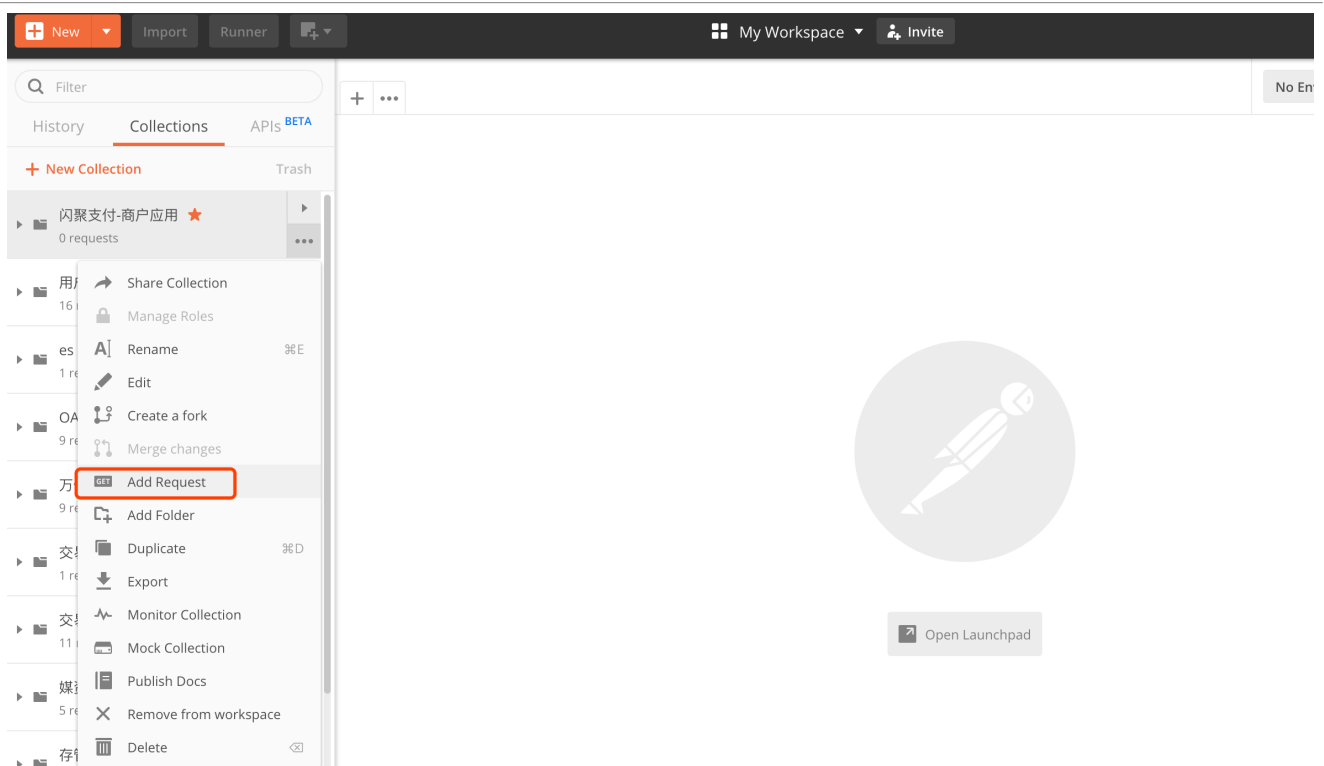
2.2 接口调试利器Postman

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。

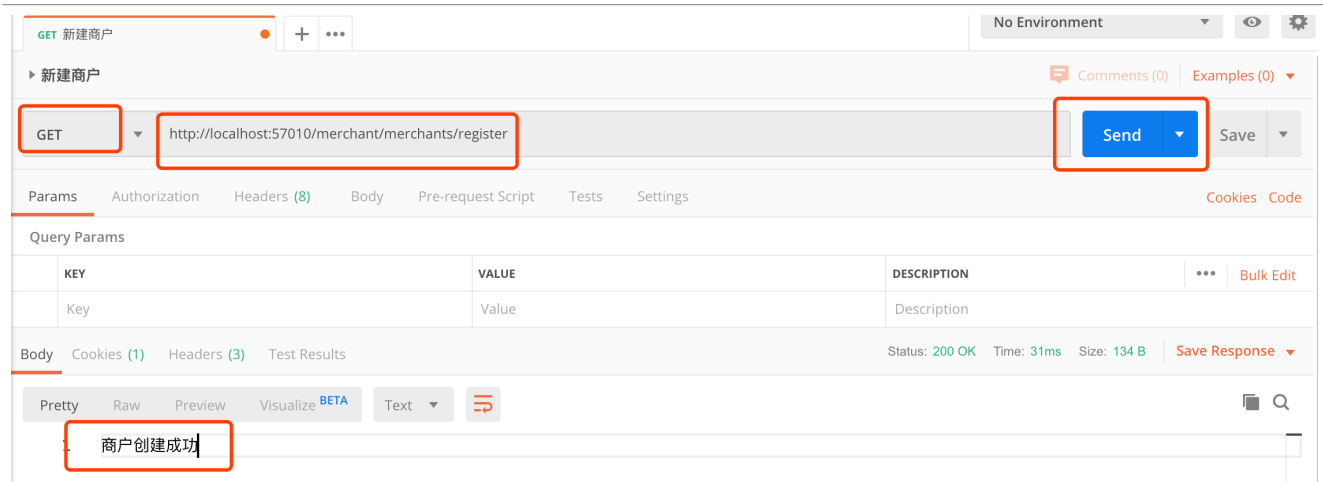
- 1、请自行在本机安装Postman
- 2、新建集合(建议一个微服务新建一个对应的集合)：闪聚支付-商户应用



- 3、在闪聚支付-商户应用集合中新建请求，并录入请求信息



填写新建商户接口地址和请求类型后，点击Send发送请求：



The screenshot shows the Postman interface for a REST client. The top bar indicates the environment is 'No Environment'. The main area shows a GET request to 'http://localhost:57010/merchant/merchants/register'. The 'Send' button is highlighted. Below the request, the 'Body' tab is selected, showing the response status '200 OK', time '31ms', and size '134 B'. The response body is displayed as '商户创建成功' (Merchant creation successful).

小技巧：每个测试都可以进行保存(Ctrl+S)，以便于后续使用。