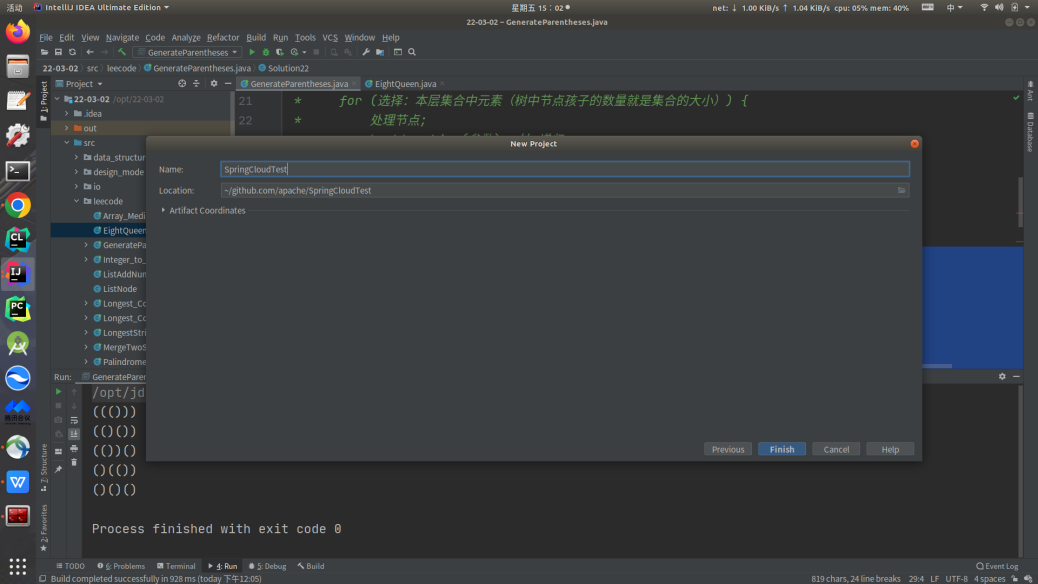
**SpringCloud:学习**

首先学会创建最基本的SpringCloud项目：

手顺如下

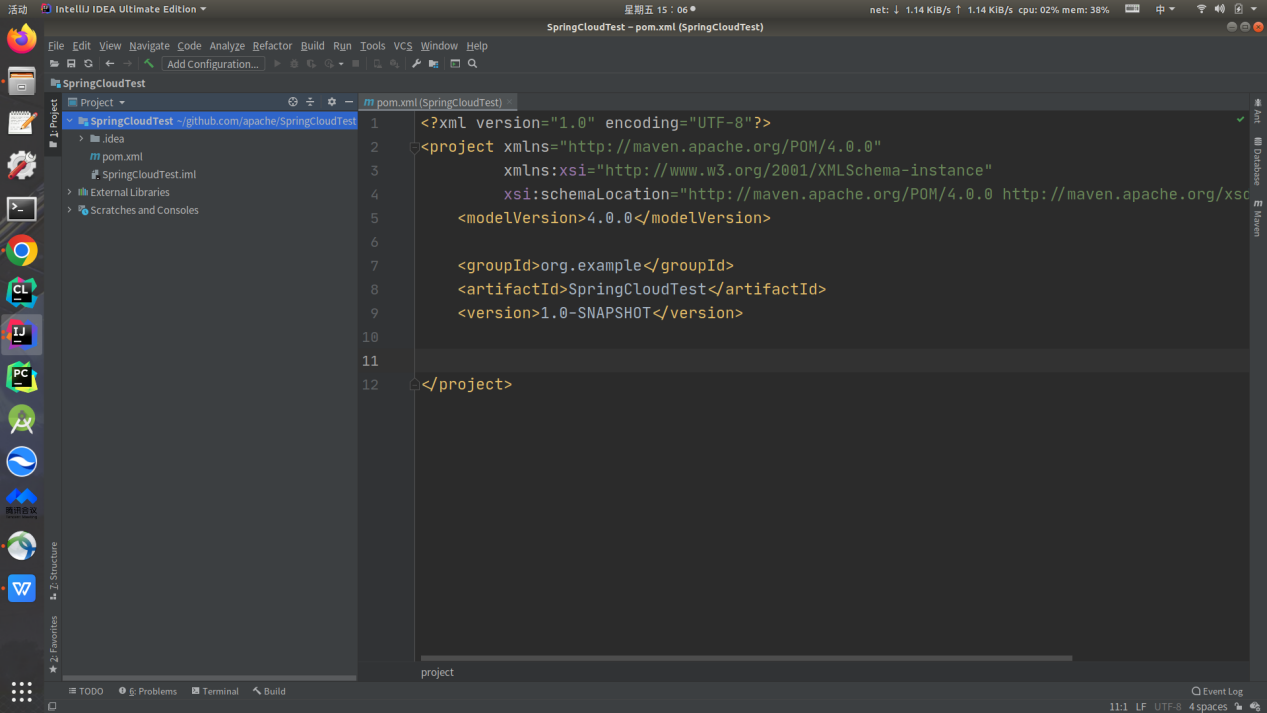
**第一步**

**创建一个MAVEN普通项目**

****

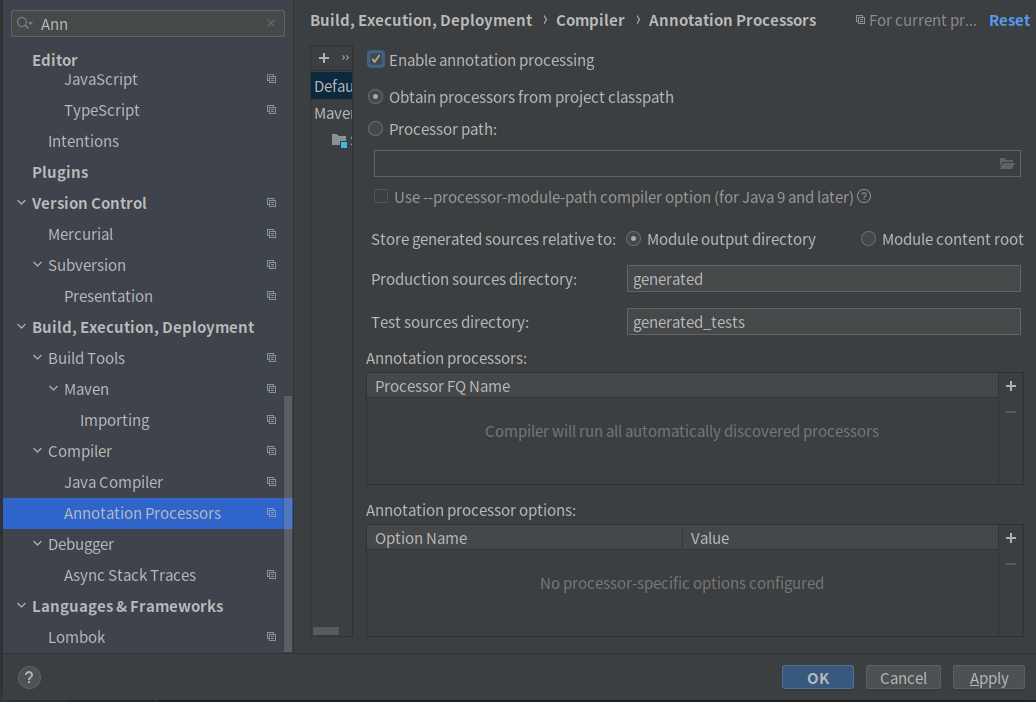
**第二步**

**删除里面的src文件夹**

****

**第三步**

**激活注解->从项目类路径中获取注释处理器并指定输出目录**

****

**第四步**

**加入以下基本依赖**

<!--Maven 项目查看maven依赖的详细信息。 -->

<!--在target的文件夹下，生成一个site文件夹，里边有个dependency.html，进去就能看到详细的依赖的信息-->

<dependency>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.0.0</version>

</dependency>

<!-- 下面三个基本是微服务架构的标配 -->

<!--spring boot 2.2.2-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-dependencies</artifactId>

<version>2.2.2.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<!--spring cloud Hoxton.SR1-->

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>Hoxton.SR1</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<!--spring cloud 阿里巴巴-->

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-alibaba-dependencies</artifactId>

<version>2.1.0.RELEASE</version>

<type>pom</type>

<scope>import</scope>

</dependency>

<!--mysql-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.version}</version>

<scope>runtime</scope>

</dependency>

<!-- druid-->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid</artifactId>

<version>${druid.version}</version>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>${mybatis.spring.boot.version}</version>

</dependency>

<!--junit-->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

</dependency>

<!--log4j-->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

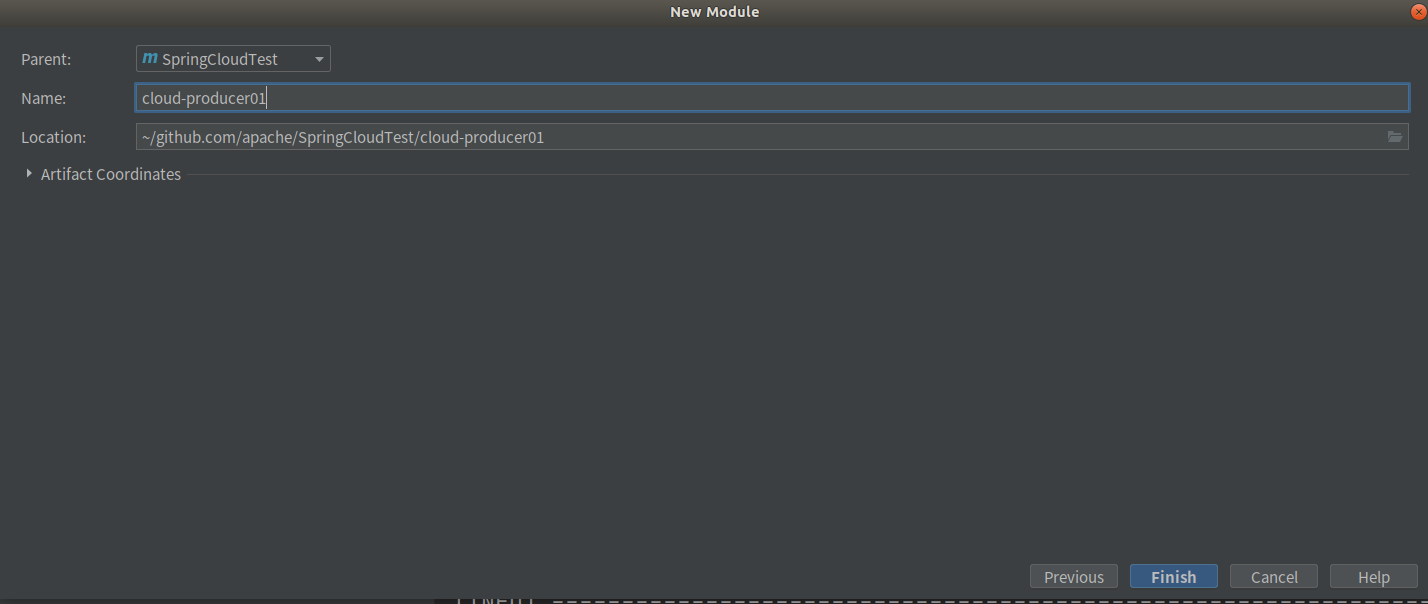
<version>${log4j.version}</version>

</dependency>

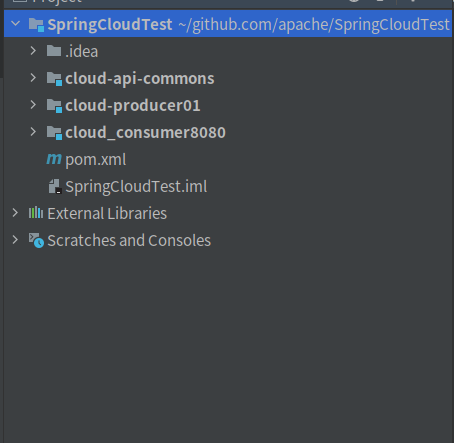
</dependencies>

**第五步**

**创建子项目**

****

**创建成这个样子**

****

结构说明：这是小试牛刀的一个简单小例子，只包含了生产者（cloud-producer01）消费者（cloud\_consumer8080）和项目恢复重构（cloud-api-commons）

**编写**cloud-api-commons**子项目：**

**在pom文件中输入以下依赖**

**<dependency>**

**<groupId>org.projectlombok</groupId>**

**<artifactId>lombok</artifactId>**

**</dependency>**

**<dependency>**

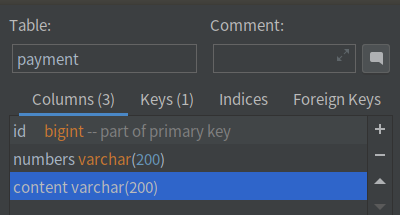
**<groupId>cn.hutool</groupId>**

**<artifactId>hutool-all</artifactId>**

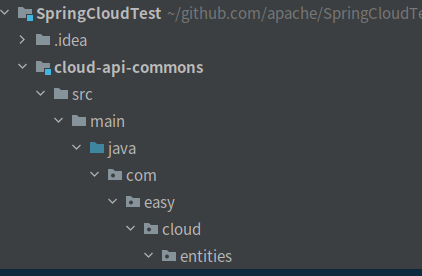
**<version>5.1.0</version>**

**</dependency>**

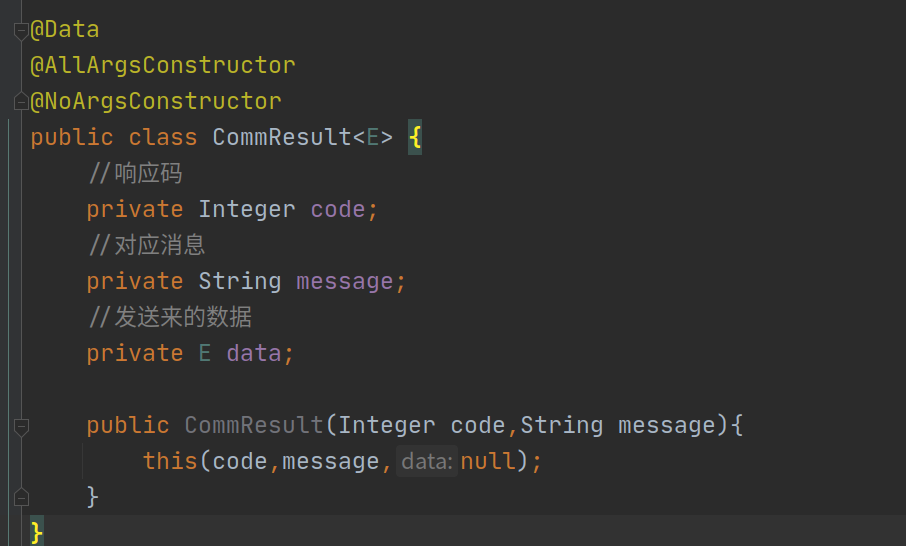
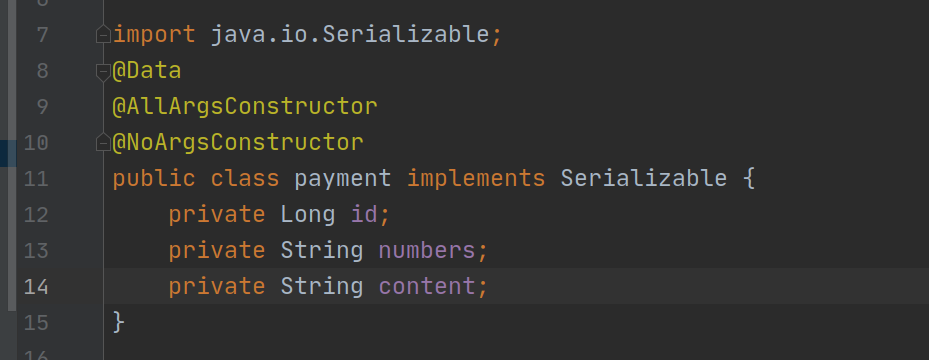
**创建一个订单表**

****

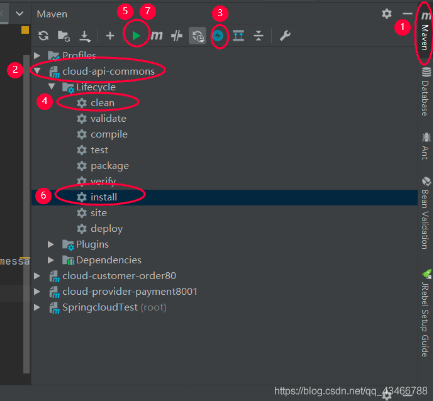
**创建一个基类package**

****

**创造基础类型payment(和数据库对应)CommResult(传输消息体)**

****

**然后按照步骤执行**

****

**第六步**

**编写**cloud-producer01**子项目：**

**在pom文件中输入以下依赖**

<dependencies>

<!--导入公共部分实体类cloud-api-commons-->

<dependency>

<groupId>org.example</groupId>

<artifactId>cloud-api-commons</artifactId>

<version>${project.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>druid-spring-boot-starter</artifactId>

<version>1.1.10</version>

</dependency>

<!--mysql-connector-java-->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<!--jdbc-->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

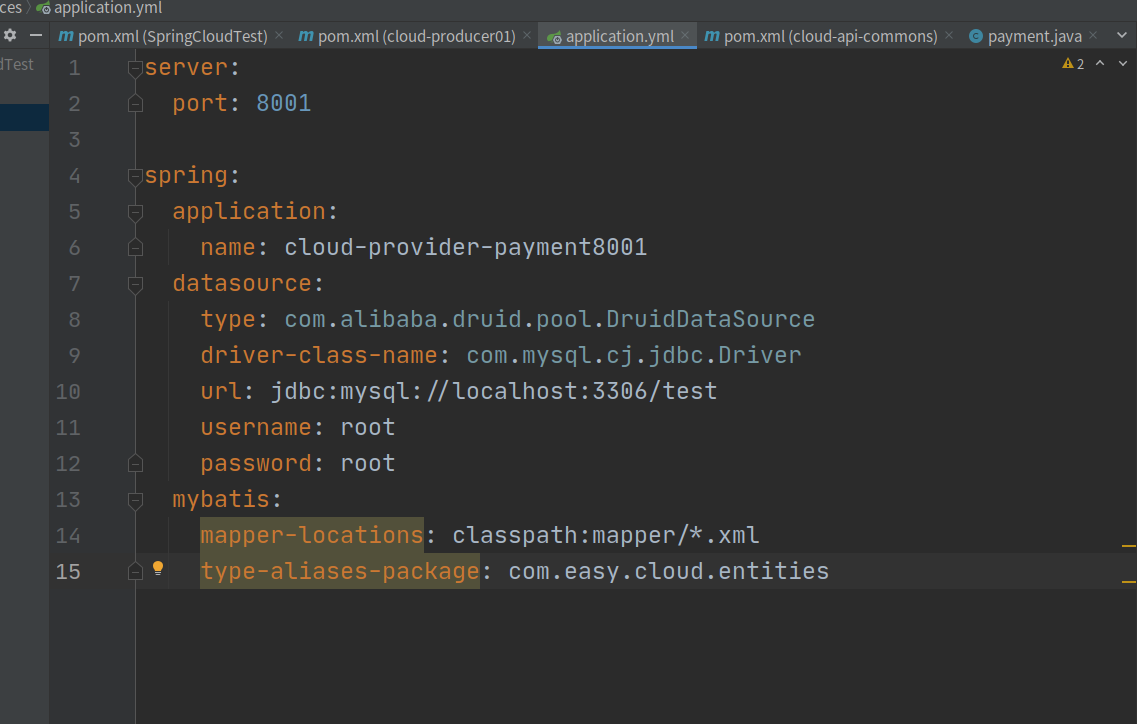
<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

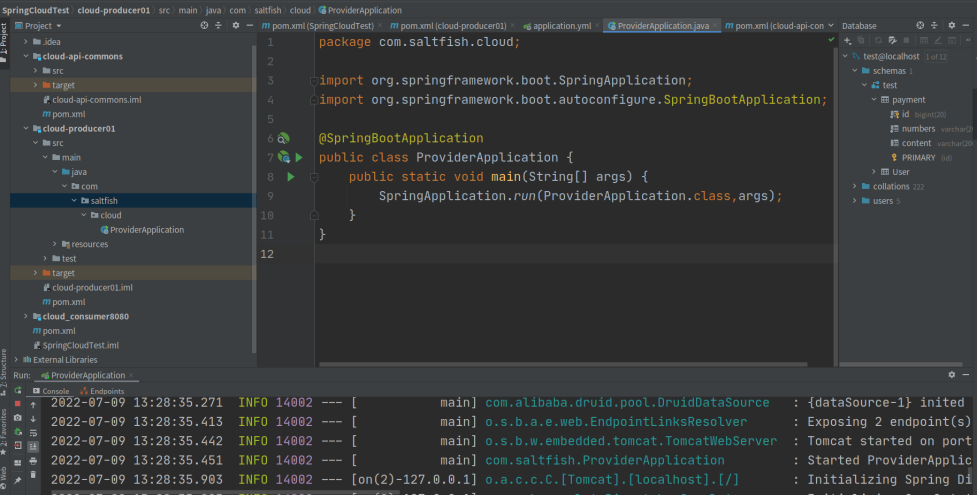
</dependency>

</dependencies>

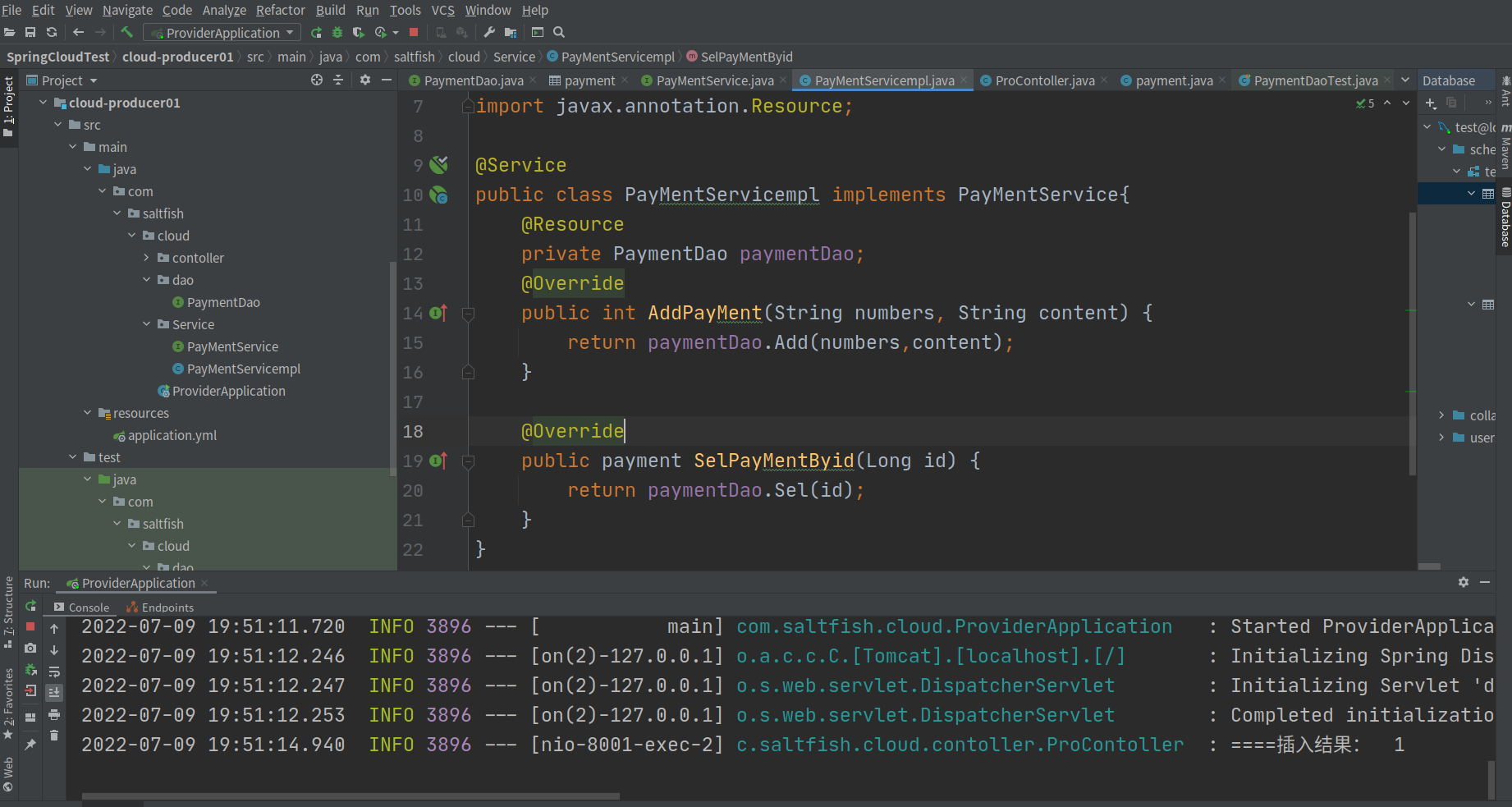
**然后新建配置文件application.yml链接数据库和端口设置**

****

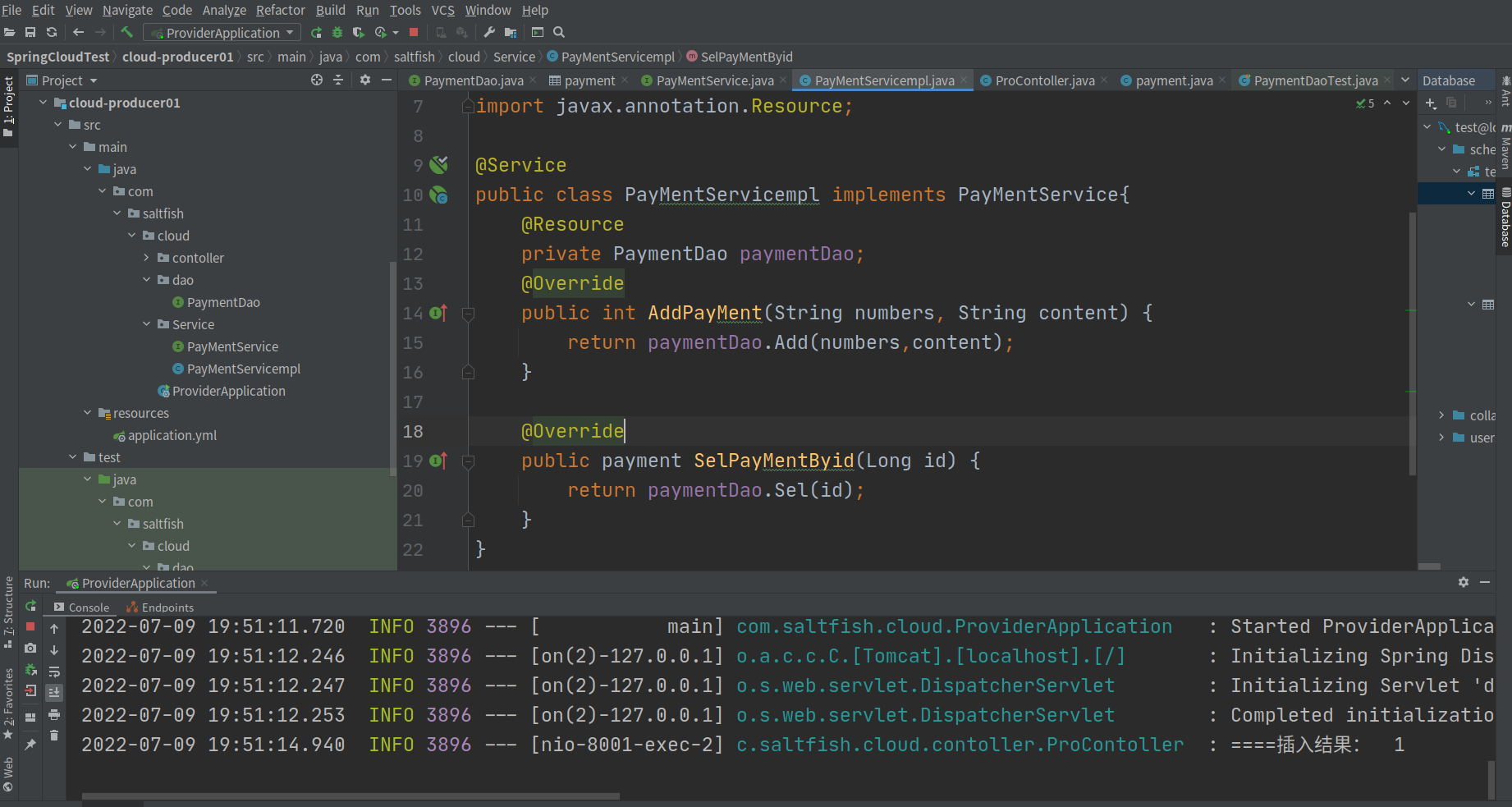
**创建springboot启动类ProvideApplication启动后确定无配置错误**

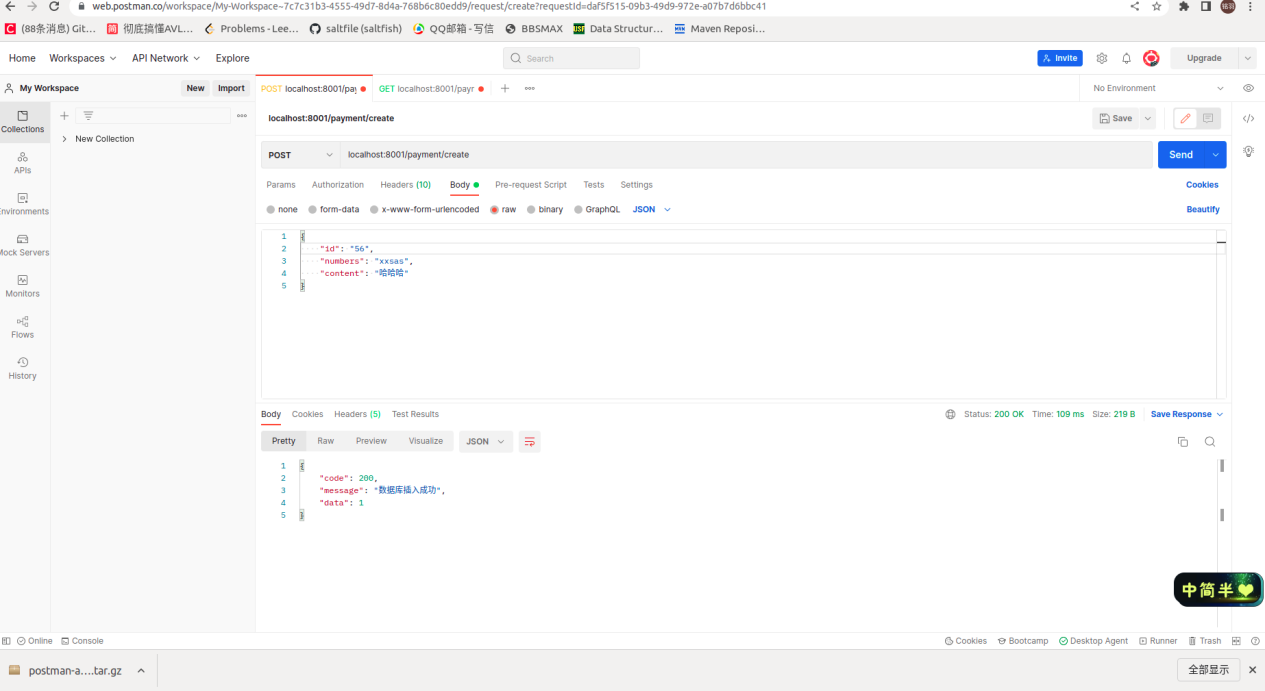
****

**创建dao层service层创建两个功能添加和查询**

****

**将Contoller功能补全并使用postman测试通过**

****

****

**第七步**

**编写**cloud\_consumer8080**子项目：**

**在pom文件中输入以下依赖**

<dependencies>

<dependency>

<groupId>org.example</groupId>

<artifactId>cloud-api-commons</artifactId>

<version>${project.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

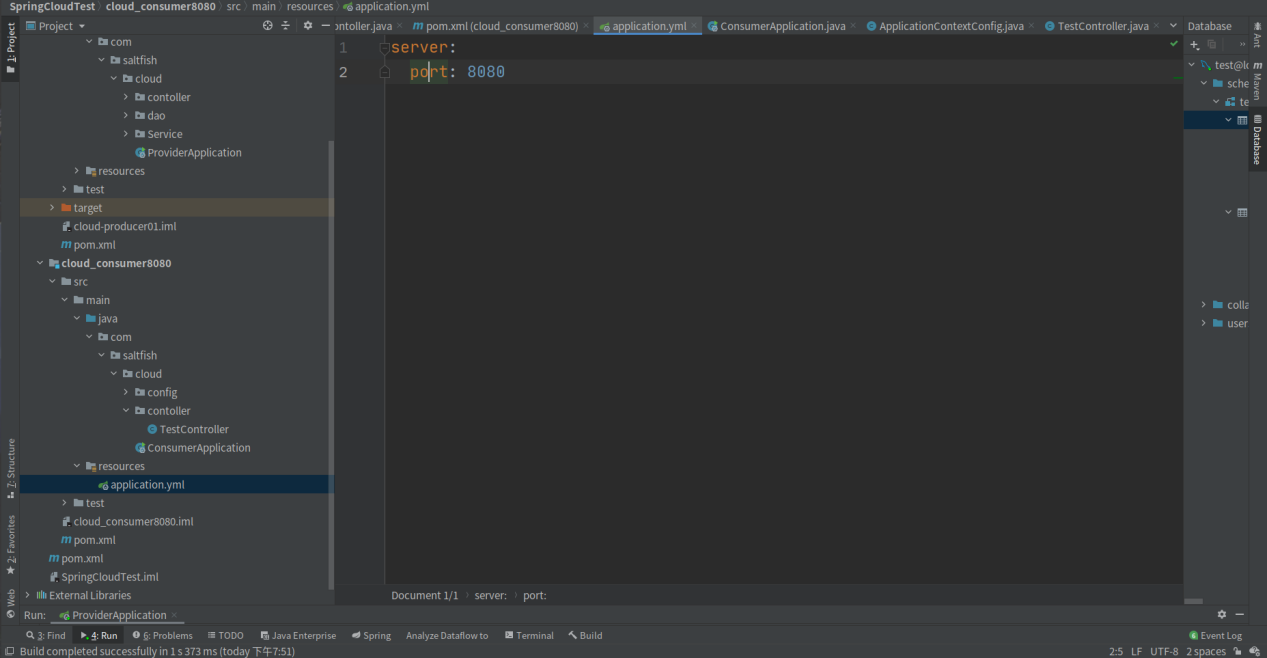
<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

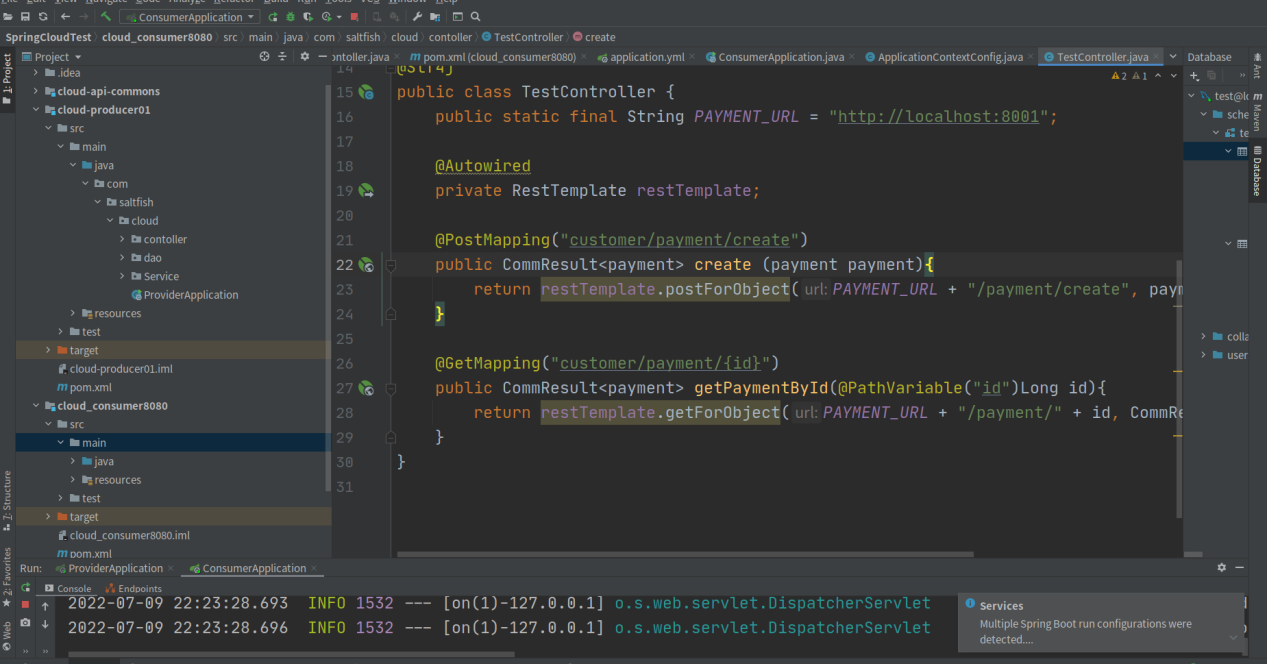
</dependency>

</dependencies>

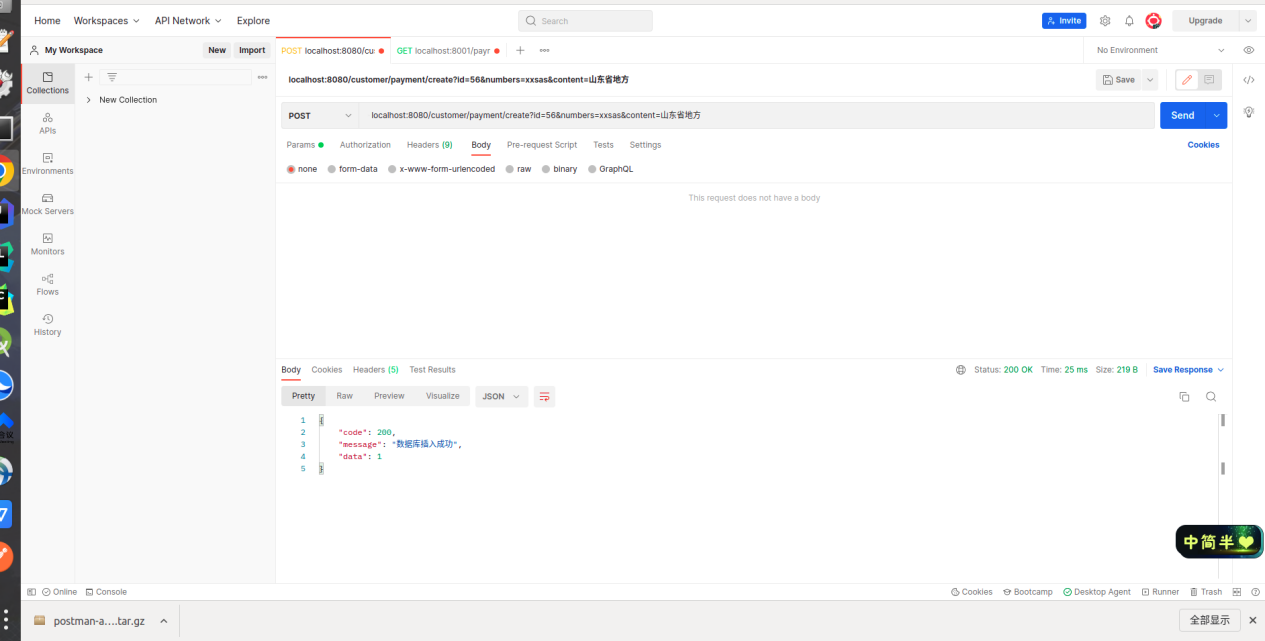
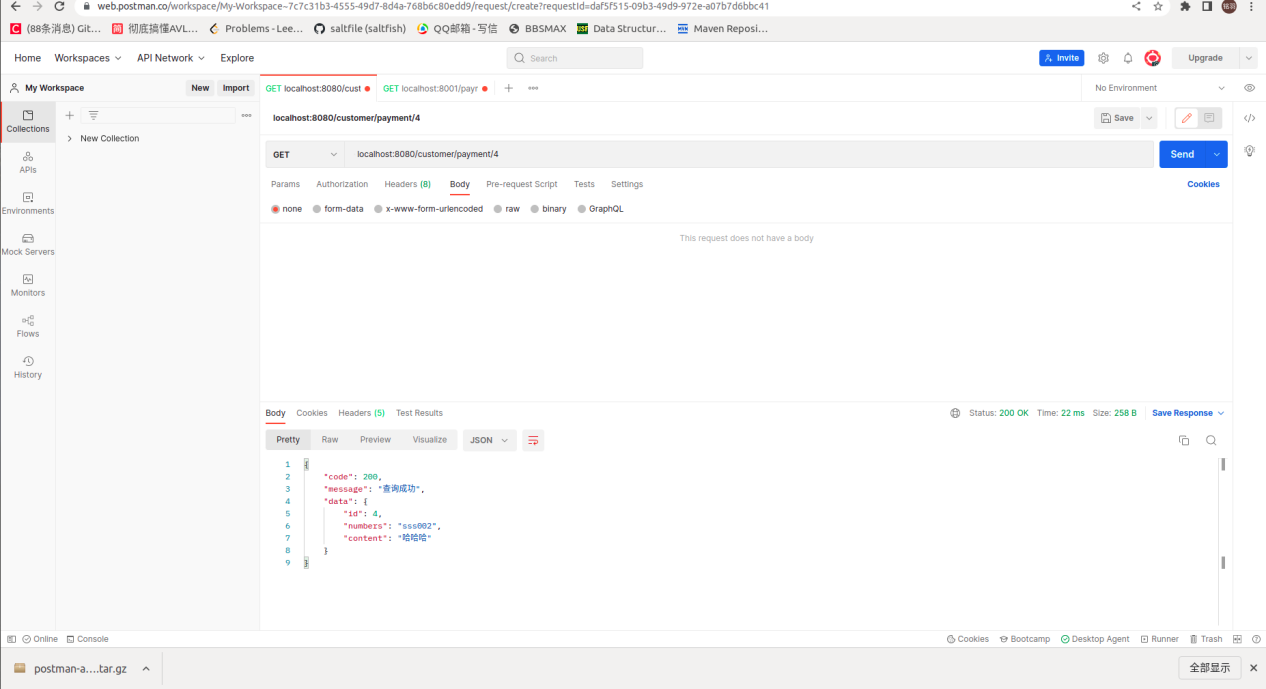
**然后新建配置文件application.yml链接数据库和端口设置**

****

**建立完成contoller**

****

**并用postman测量URL是否打到生产者localhost8001上**

****

**完成测试☺**

**原文网址:https://blog.csdn.net/qq\_43466788/article/details/112968827**

**这只是一个小小的例子，原网站中没有测试，我在步骤中补充了**

**接下来才是阶段性的学习emmm**

**SpringCIoudNetfix：**

官方文档：https://spring.io/projects/spring-cloud-netflix

功能：

服务发现：可以注册 Eureka 实例，客户端可以使用 Spring 管理的 bean 发现实例

服务发现：可以使用声明性 Java 配置创建嵌入式 Eureka 服务器

1. **Archalus**

官方文档:https://www.baeldung.com/netflix-archaius-spring-cloud-integration

首先Archalus是一个强大的配置管理库，可用于从许多不同来源收集配置属性，提供对它们的快速、线程安全的访问。

1. **Archalus提供了一些其他任何配置框架都没有考虑到的方便且有趣的功能.**

动态和类型属性

在属性突变时调用的回调机制

动态配置源（例如 URL、JDBC 和 Amazon DynamoDB）的即用型实施

可由 Spring Boot Actuator 或 JConsole 访问以检查和操作属性的 JMX MBean

动态属性验证

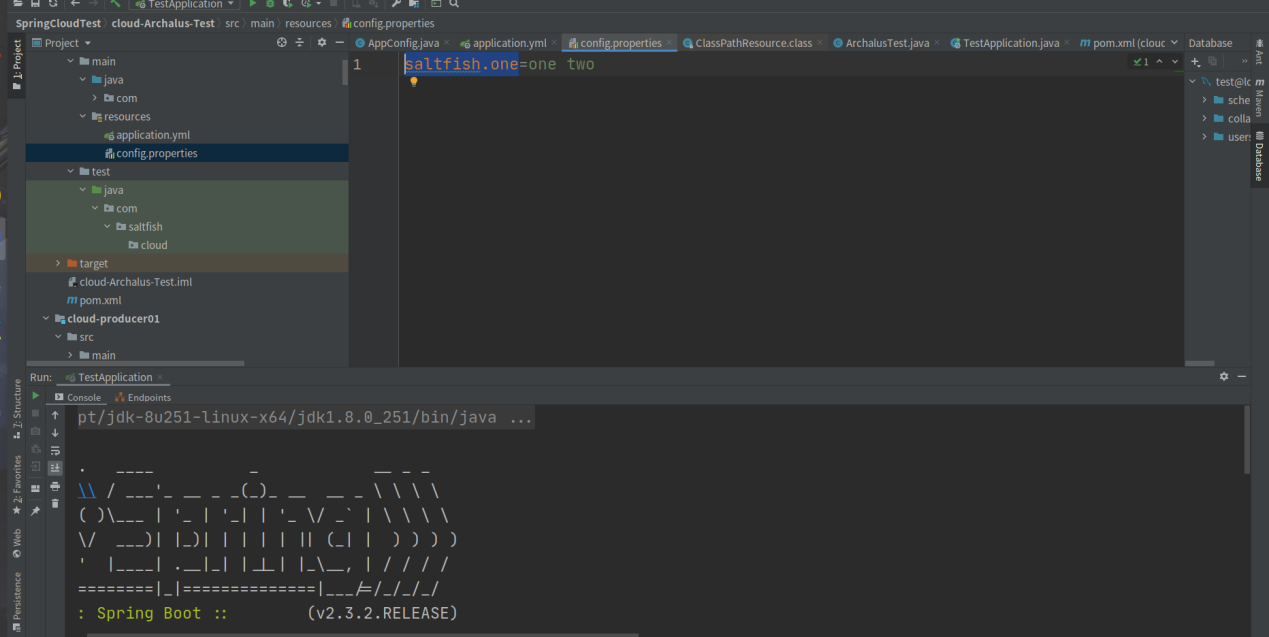
些配置源可以支持在运行时轮询更改

1. **开始做一个例子,浅尝一下**

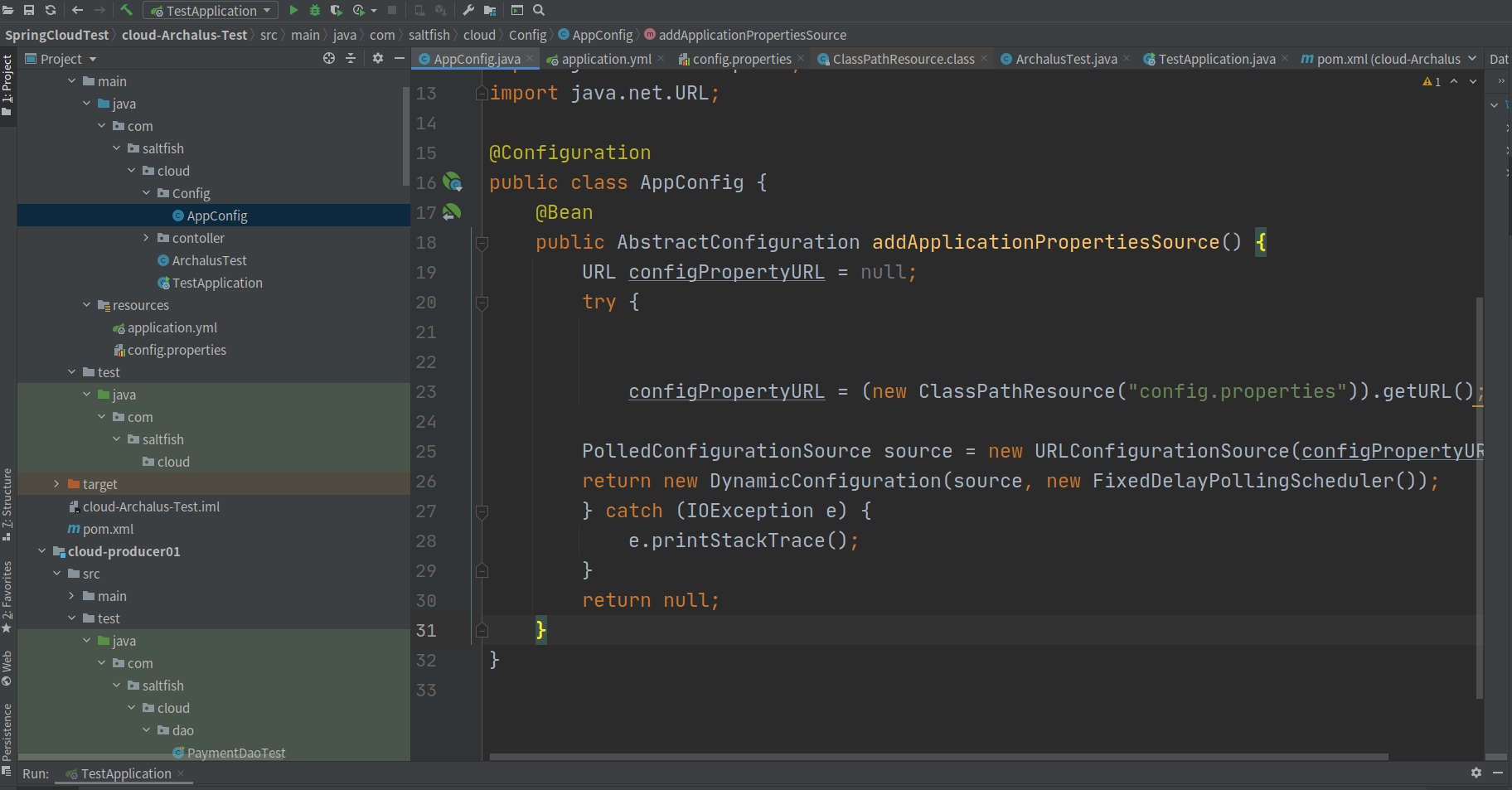
**首先创建一个config.properties文件**

**因为配置文件名必须叫config.properties才能被自动识别读取**

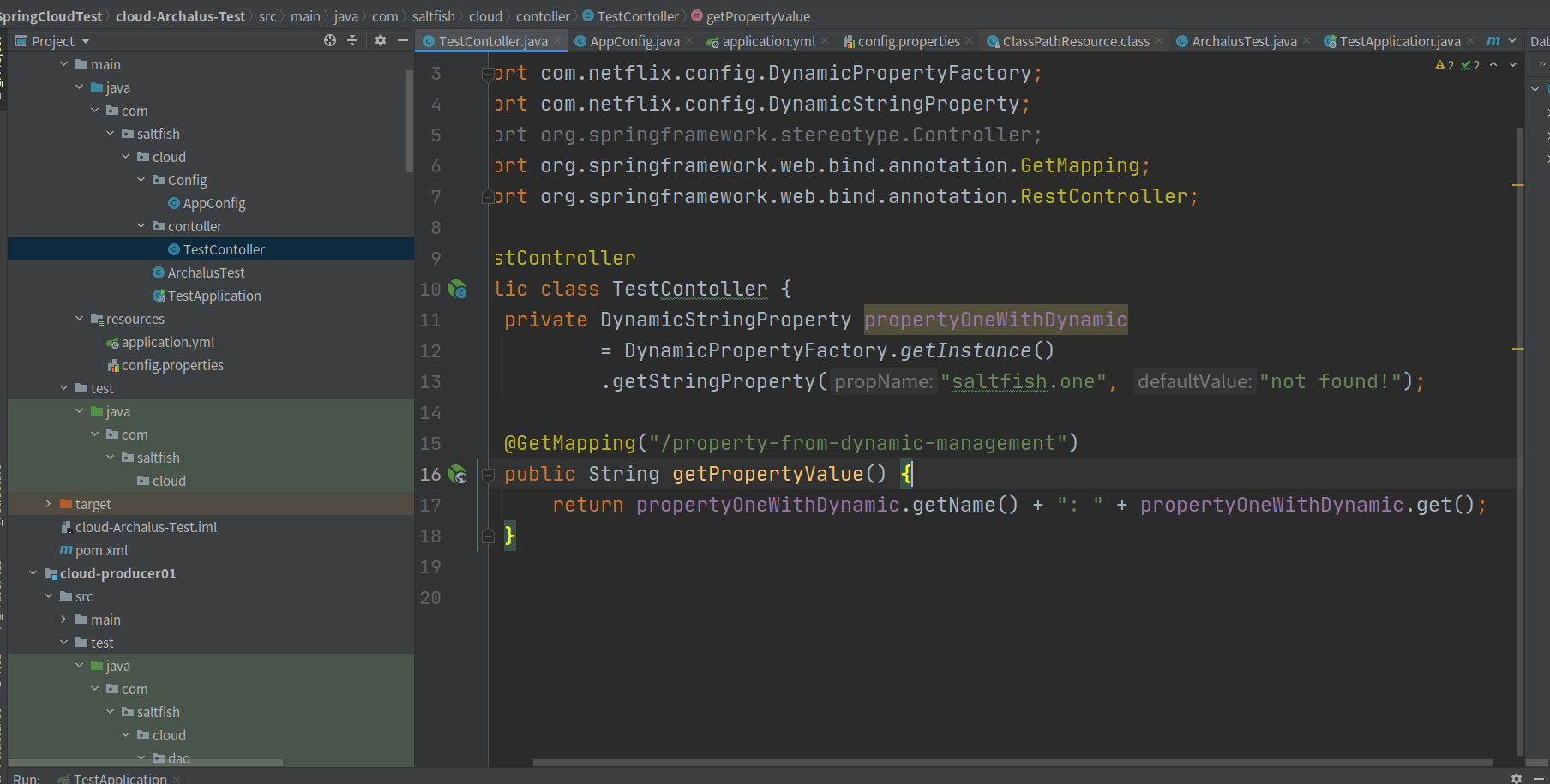
（除了默认值还可以定义系统属性archaius.configurationSource.additionalUrls来指定你的文件所在位置，其中包含本地配置文件的URL路径）



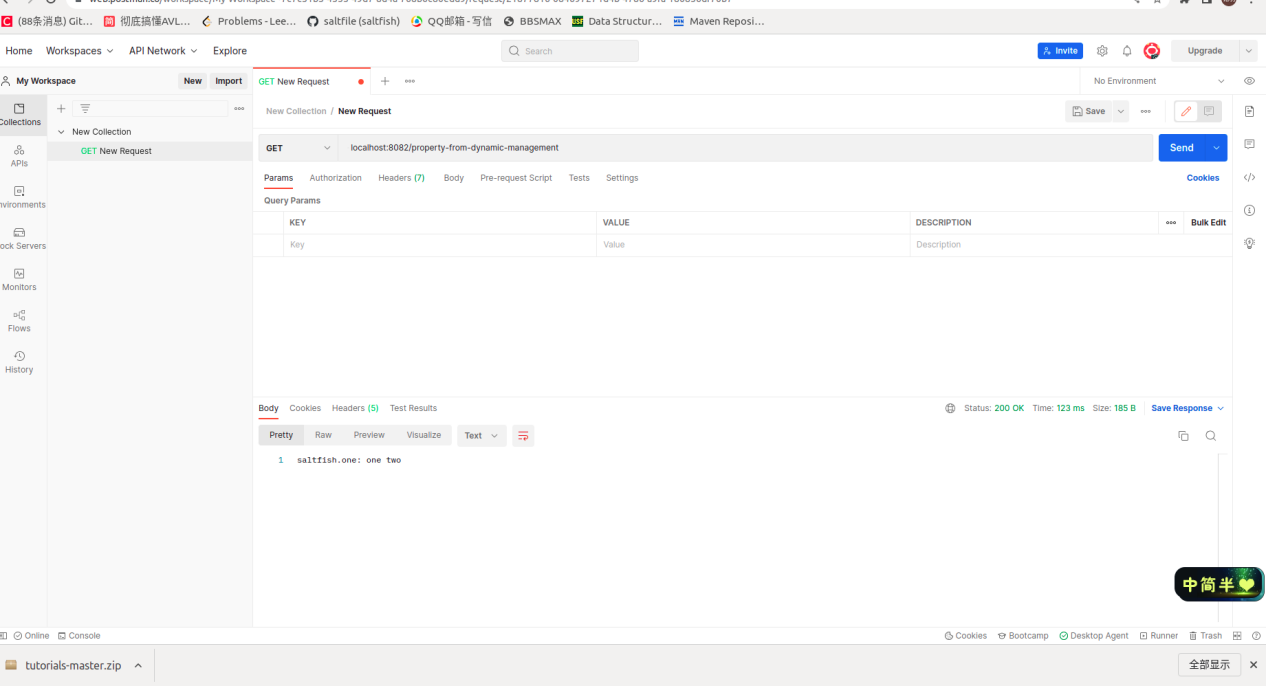
然后添加专有Bean进去



在编写对应的Contoller



在postman上进行测试



1. 最后的总结：这个只是一个粗略的例子，其中**Archalus**的功能还有很多，比如可以动态读取配置还可以合并jdbc,zookeeper等等一些分布式配置或者命令或者sql语句的动态，想读取更多的使用方法就使用官网文档：https://github.com/eugenp/tutorials/tree/master/spring-cloud-modules/spring-cloud-archaius
2. **Eruka**

Eruka是微服务架构中最基础的注册中心，也是一个重要核心管理组件，主要的作用是存储服务信息和提供url路由服务信息去实现其中的远程调用

1. 什么是注册中心

注册中心就是收录了服务和服务地址组成的映射信息，如果需要调用服务的时候就通过注册中心远程拉去调用服务，获取到服务地址进行远程调用。如果把注册中心比作一个电话簿，如果我想打电话给张三，就可以在电话簿上中找到对应的电话号码给他打电话（这就是服务发现）；如果王五有了一个新的电话号码并且告诉我，我把这个号码写进电话簿（这就是服务注册）如果李四有一天号码失效了，我把他及他的号码从本子上划去（这就是服务下线/卸载）。

注：常用的注册中心不止一种，比如Nacos Zookeeper Consul等等并且开发难度不一。

注册中心间特性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 特性 | Eureka | Nacos | Consul | Zookeeper |
| CAP | AP | CP+AP | CP | CP |
| 健康检查 | Client Beat | TCP/HTTP/MYSQL/Client Beat | TCP/HTTP/GRPC/CMD | Keep Alive |
| 雪崩保护 | 有 | 有 | 无 | 无 |
| 自动注销实例 | 支持 | 支持 | 不支持 | 支持 |
| 访问协议 | http | http/dns | http/dns | tcp |
| 监听支持 | 支持 | 支持 | 支持 | 支持 |
| 多数据中心 | 支持 | 支持 | 支持 | 不支持 |
| 跨注册中心同步 | 不支持 | 支持 | 支持 | 不支持 |
| springcloud集成 | 支持 | 支持 | 支持 | 支持 |

1. 为什么需要注册中心

除了上面说到的服务间远程调用之外应该还要考虑注册中心更加重要的功能

1. 服务注册之后，如何被及时发现
2. 服务宕机之后，怎么做才能及时下线（保留服务，做缓存处理）
3. 服务如何有效的水平扩展
4. 服务发现时，如何进行路由发现
5. 服务出错后,怎么做服务降级
6. 注册中心如何实现自身高可用

注册中心处理的事物很复杂，类似于负载均衡器但要比这些均衡器功能复杂得多

1. Eureka注册中心三种角色

Eureka Server：

注册中心负责负载均衡故障转移等等的主要任务。通过Register、Get、Renew提供服务的注册与发现

Eureka Client：

Service Consumer:服务消费者通过读取Eureka Server 中的注册列表，消费获取服务

Service Provider:服务生产者就是注册到Eureka Server之中的服务,将自身的实例注册到Eureka当中。

一个简单案例