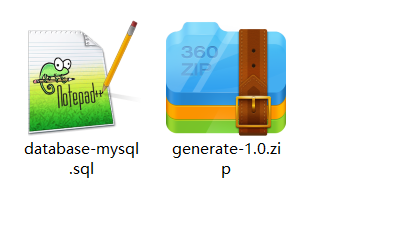
# 关于代码生成器的使用

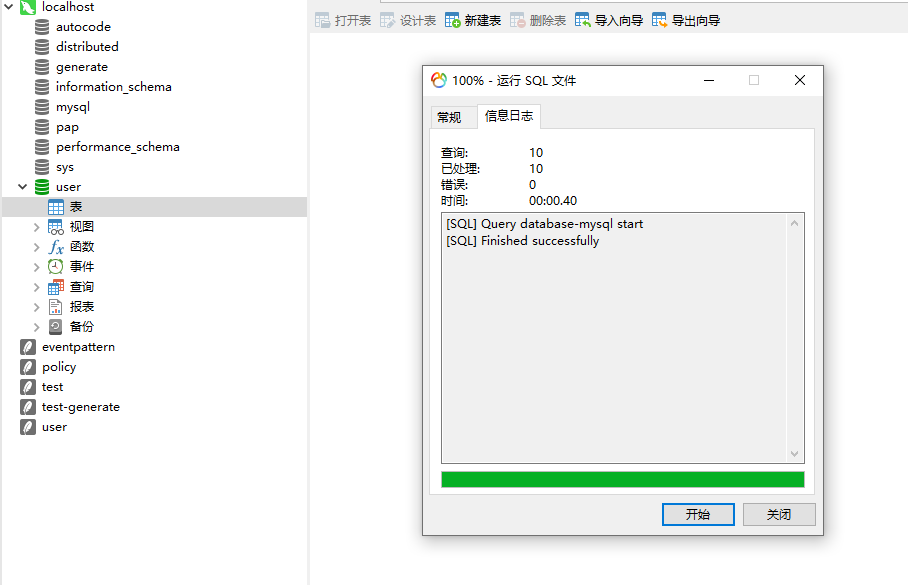
## 准备好代码的生成包及数据库脚本

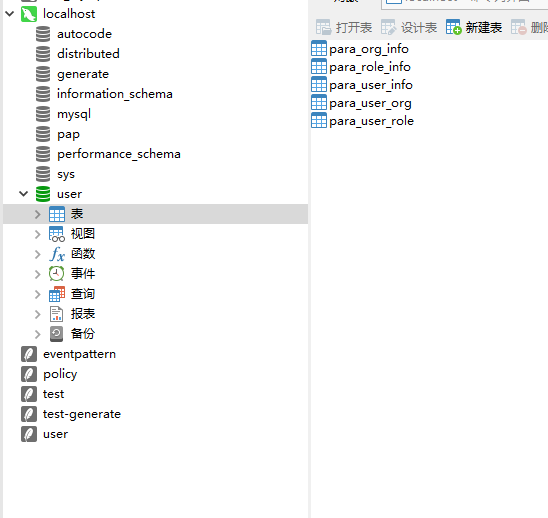


目前代码生成器仅支持mysql

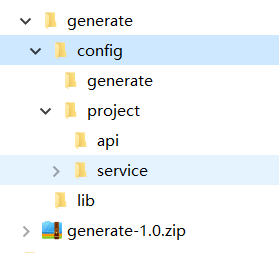
## 数据库中执行数据库脚本

建议：单独创建一个数据库用于生成代码





## 代码生成器进行配制



/config -----------------------------------------配制目录

/config/autocode.properties-----------------------代码生成器的数据库配制

/config/errorPersist.code--------------------------错误码保存的序列化文件

/config/generate---------------------------------代码生成规则配制

/config/generate/generate.yml---------------------代码生成规则配制文件

/config/project----------------------------------代码生成相关的一些模板文件

/config/project/api-------------------------------接口层的模板文件

/config/project/api/pom.xml-----------------------接口层模块的pom.xml

/config/project/service----------------------------业务层的模板文件

/config/project/service/pom.xml--------------------业务层的pom.xml

/config/project/service/src/------------------------业务层的公共配制文件

/config/project/pom.xml---------------------------总工程目录的pom.xml

### 修改数据库配制文件(autocode.properties)

|  |
| --- |
| #驱动名称  driverClassName=com.mysql.jdbc.Driver  #连接数据库的url，不同数据库不一样。例如：  #mysql : jdbc:mysql://10.20.153.104:3306/druid2  #oracle : jdbc:oracle:thin:@10.20.149.85:1521:ocnauto  url=jdbc:mysql://localhost:3306/generate?autoConnect=true&useUnicode=true&characterEncoding=utf-8&zeroDateTimeBehavior=convertToNull  #连接数据库的用户名  username=root  #连接数据库的密码。如果你不希望密码直接写在配置文件中，可以使用ConfigFilter。  # 详细看这里：https://github.com/alibaba/druid/wiki/%E4%BD%BF%E7%94%A8ConfigFilter  password=123456  #初始化时建立物理连接的个数。初始化发生在显示调用init方法，或者第一次getConnection时  initialSize=1  #最大连接池数量  maxActive=8  #最小连接池数量  minIdle=1  #db type 1,oracle,2 mysql  database.dbtype=2 |

红色标注部分即为需要修改的地方

### 修改生成规则文件(config\generate\generate.yml)

|  |
| --- |
| generate:  #当前的模式：1，完整覆盖模式，2，追加模式,, 3，追加方法模式  #1，完整覆盖模式，针对首次生成工程  #在运行之前将所有文件进行删除操作，包括文件夹。  #2，追加覆盖模式,针对同一工程的新加聚合的场景  #当文件已经存在，则会发生文件覆盖，  #特别注意: 当发生表重命名的情况，需要手动清理已经生成的文件。  #根据情况选择是否需要更新开始错误码号。  #3,追加方法模式，针对已经存在的代码，添加新方法的情况  #此模式会在原代码的结束前的位置添加新的方法  model: 1  #生成代码的类型：ddd，使用领域驱动ddd四层架构  type: ddd  #生成的目标数据库类型:mysql目标数据库为mysql;oralce，目标数据库为oracle  databaseType: mysql  #数据库表空间命名  databaseTableSpaceName: generate  #错误码开始编号，依次向后；如果需要重新以此开始，需删除序列文件.否则将以序列化后的文件数字，取整加2000开始  startErrorCode: -10000  #是否修改错误码编号,默认为true，进行修改  updateCode: true  #作者名称，即代码写在@author中的名称  author: liujun  code:  #单个插入方法  - name: insert  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: insert  #方法的描述  comment: 单个添加  #排序  order: 100  #批量插入方法  - name: insertList  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: insert  #方法的描述  comment: 批量添加  #参数列表,table\_name表示以表名生成的对象实体，默认 table\_name  params: java.util.List<table\_name>  #排序  order: 200  #数据修改  - name: update  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: update  #方法的注释  comment: 修改方法  #排序  order: 300  #单个数据删除  - name: delete  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: delete  #方法的注释  comment: 单个删除  #标识当前是否为主键操作  primaryFlag: true  #排序  order: 400  #批量根据主键删除数据  - name: deleteByIds  #当前方法是否禁用 true 表示禁用，false 表示启用，默认值为false，即未配制或者值为false都为启用，true表示禁用  disable: true  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: delete  #方法的注释  comment: 批量删除  #where条件使用数据库字段,多个以逗号分隔,in关键字表示在范围内匹配  where: namespace\_id,oid in  #排序  order: 500  #分页查询  - name: queryPage  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: query  #方法的注释  comment: 分页查询  #分页查询标识  pageQueryFlag: true  #返回值类型信息  returns: java.util.List<table\_name>  #排序  order: 600  #根据主键进行查询  - name: queryById  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: query  #方法的注释  comment: 根据id查询  #返回值类型信息  returns: table\_name  #排序  order: 700  #根据主键集合进行查询操作  - name: queryByIds  disable: true  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: query  #方法的注释  comment: 根据一批id查询  ##where条件使用数据库字段,多个以逗号分隔,in关键字表示在范围内匹配  where: OID in  #返回值类型信息  returns: java.util.List<table\_name>  #排序  order: 800  #根据主键集合进行查询操作  - name: queryByName  #当前方法是否禁用 true 表示禁用，false 表示启用，默认值为false，即未配制或者值为false都为启用，true表示禁用  disable: true  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: query  #方法的注释  comment: 根据名称查询  ##where条件使用数据库字段,多个以逗号分隔,in关键字表示在范围内匹配  where: NAMESPACE\_ID,USER\_NAME in  #返回值类型信息  returns: java.util.List<table\_name>  #排序  order: 900  #根据主键集合进行查询操作  - name: queryByInfo  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: query  #当前方法是否禁用 true 表示禁用，false 表示启用，默认值为false，即未配制或者值为false都为启用，true表示禁用  disable: false  #方法的注释  comment: 根据信息查询  ##where条件使用数据库字段,多个以逗号分隔,in关键字表示在范围内匹配  #where: NAMESPACE\_ID,USER\_NAME  #返回值类型信息  returns: table\_name  #排序  order: 1000  #代码内的目录结构的配制  codeMenuTree:  #最基础的目标配制  baseMenu: com.paraview.example.  #最新输出的模块的名称  domainName: user  #模块名称  moduleName: user  #工程名  projectName: data-user  #maven的配制信息  #maven的artifactId即为各工程或者模块的名称  maven:  groupId: com.paraview.example  version: 1.0-SNAPSHOT  #代码生成器需要包括的范围  #目前存在api,application,domain,repository  #api代码对外暴露的api接口，包含一系列的方法  #application表示聚合层  #domain表示领域服务层  #repository表示存储层的代码生成，目前使用mybatis  scope: api,application,domain,repository  #输出文件的配制  output: D:\java\workspace\selfwork\autocode\testgenerate\example |

红色标注部分即为需要修改的地方

startErrorCode请做好约定，否则会出现不同一模块会有相同的错误码，不能进行唯一的错误码定位，切记。

code 下为需要生成的代码方法的配制。一个配制对应

以插入为例

|  |
| --- |
| #单个插入方法  - name: insert  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: insert  #方法的描述  comment: 单个添加  #排序  order: 100 |

name用于标识插入的方法名

operator:用于标识的操作，insert/update/delete/query

comment：方法注释

order: 表示当前方法在整个生成方法的中序列

再以查询为例

|  |
| --- |
| #根据主键集合进行查询操作  - name: queryByInfo  #此标识用于在方法操作时对返回类型进行改变，修改在数据库层返回int，操作的行数，在业务层返回boolean，标识成功与失败  operator: query  #当前方法是否禁用 true 表示禁用，false 表示启用，默认值为false，即未配制或者值为false都为启用，true表示禁用  disable: false  #方法的注释  comment: 根据信息查询  ##where条件使用数据库字段,多个以逗号分隔,in关键字表示在范围内匹配  #where: NAMESPACE\_ID,USER\_NAME  #返回值类型信息  returns: table\_name  #排序  order: 1000 |

name用于标识插入的方法名

operator:用于标识的操作,当前为查询，insert/update/delete/query

disable:当前方法是否禁用, true 表示 禁用，false表示启用

comment：方法注释

where: where条件使用数据库字段,多个以逗号分隔,in关键字表示在范围内匹配,以NAMESPACE\_ID,USER\_NAME in为例，表示当前使用NAMESPACE\_ID与USER\_NAME为条件，并且USER\_NAME为in匹配,即在请求中的多个值中匹配

returns: 返回类型，table\_name表示以当前表以基础生成的实体。java.util.List<table\_name>表示返回为集合的表实体

order: 表示当前方法在整个生成方法的中序列

|  |
| --- |
| #代码内的目录结构的配制  codeMenuTree:  #最基础的目标配制  baseMenu: com.paraview.example.  #最新输出的模块的名称  domainName: user |

用于生成代码中的基础目录包

domainName 当前领域的包的目标名称

|  |
| --- |
| #模块名称  moduleName: user  #工程名  projectName: data-user  #maven的配制信息  #maven的artifactId即为各工程或者模块的名称  maven:  groupId: com.paraview.example  version: 1.0-SNAPSHOT |

此用于进行maven相关的配制

projectName用于生成总工程名及pom.xml文件中的<artifactId></artifactId>的信息

moduleName用于生成子模块的pom.xml文件及<artifactId></artifactId>的信息

maven: 用于配制maven的信息

|  |
| --- |
| #代码生成器需要包括的范围  #目前存在api,application,domain,repository  #api代码对外暴露的api接口，包含一系列的方法  #application表示聚合层  #domain表示领域服务层  #repository表示存储层的代码生成，目前使用mybatis  scope: api,application,domain,repository  #输出文件的配制  output: D:\java\workspace\selfwork\autocode\testgenerate\example |

scope：代码生成需要包括的范围

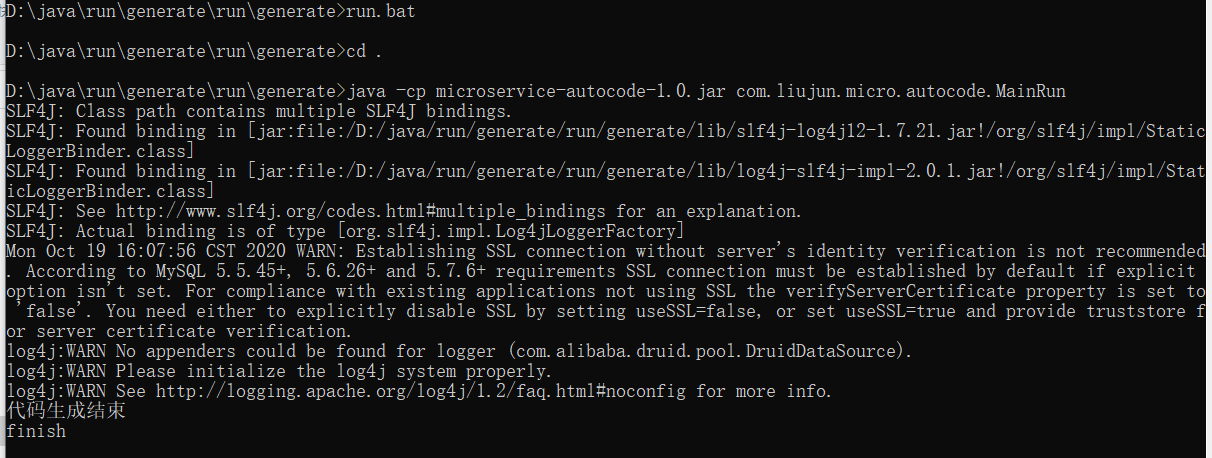
output：代码最终输出的目录

## 代码生成

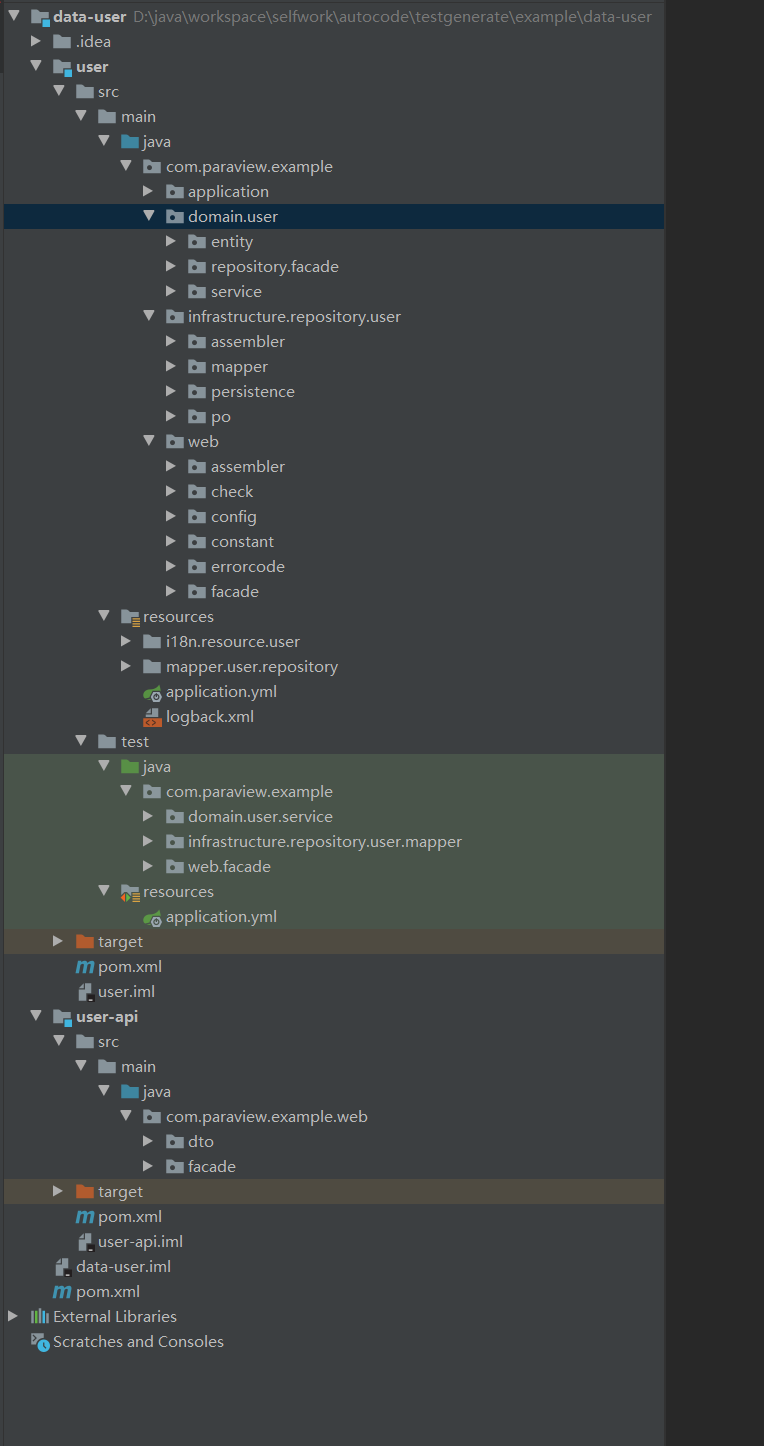
可直接双击运行

run.bat

如果在linux下，可将bat脚本中的命令在linux的命令行中执行



## 使用开发工具打开生成的代码



代码按领讨论确定的领域驱动分层的模型生成

代码目录结构说明

|  |
| --- |
| /data-user  /data-user/user  /data-user/user/src  /data-user/user/src/main  /data-user/user/src/main/java  /data-user/user/src/main/java/com  /data-user/user/src/main/java/com/paraview  /data-user/user/src/main/java/com/paraview/example  /data-user/user/src/main/java/com/paraview/example/application ----------------------------------------聚合层  /data-user/user/src/main/java/com/paraview/example/domain -------------------------------------------领域层  /data-user/user/src/main/java/com/paraview/example/domain/user---------------------------------------用户领域  /data-user/user/src/main/java/com/paraview/example/domain/user/entity--------------------------------领域层实体  /data-user/user/src/main/java/com/paraview/example/domain/user/repository--------------------------领域存储层  /data-user/user/src/main/java/com/paraview/example/domain/user/repository/facade-------------------领域存储接口  /data-user/user/src/main/java/com/paraview/example/domain/user/service----------------------------领域服务  /data-user/user/src/main/java/com/paraview/example/infrastructure-----------------------------------基础层  /data-user/user/src/main/java/com/paraview/example/infrastructure/repository-------------------------基础存储  /data-user/user/src/main/java/com/paraview/example/infrastructure/repository/user---------------------用户存储  /data-user/user/src/main/java/com/paraview/example/infrastructure/repository/user/assembler------------领域与存储实体转换代码  /data-user/user/src/main/java/com/paraview/example/infrastructure/repository/user/mapper---------------数据库操作接口  /data-user/user/src/main/java/com/paraview/example/infrastructure/repository/user/persistence------------存储层实现领域存储层的接口,转换层，用于持久与业务隔离  /data-user/user/src/main/java/com/paraview/example/infrastructure/repository/user/po-------------------持久层的实体  /data-user/user/src/main/java/com/paraview/example/web-------------------------------------------web层  /data-user/user/src/main/java/com/paraview/example/web/assembler---------------------------------web与领域层转换代码  /data-user/user/src/main/java/com/paraview/example/web/check-------------------------------------参数检查  /data-user/user/src/main/java/com/paraview/example/web/config-------------------------------------配制加载错误码  /data-user/user/src/main/java/com/paraview/example/web/constant-----------------------------------错误信息配制  /data-user/user/src/main/java/com/paraview/example/web/errorcode----------------------------------错误码文件  /data-user/user/src/main/java/com/paraview/example/web/facade-------------------------------------web层的实现  /data-user/user/src/main/resources  /data-user/user/src/main/resources/i18n  /data-user/user/src/main/resources/i18n/resource  /data-user/user/src/main/resources/i18n/resource/user----------------------------------------------用户模块的国际化错误码  /data-user/user/src/main/resources/mapper  /data-user/user/src/main/resources/mapper/user  /data-user/user/src/main/resources/mapper/user/repository--------------------------------------用户模块使用mybatis框架的mybatis操作mapper文件  /data-user/user/src/test  /data-user/user/src/test/java  /data-user/user/src/test/java/com  /data-user/user/src/test/java/com/paraview  /data-user/user/src/test/java/com/paraview/example  /data-user/user/src/test/java/com/paraview/example/domain  /data-user/user/src/test/java/com/paraview/example/domain/user  /data-user/user/src/test/java/com/paraview/example/domain/user/service------------------------------领域层的单元测试  /data-user/user/src/test/java/com/paraview/example/infrastructure  /data-user/user/src/test/java/com/paraview/example/infrastructure/repository  /data-user/user/src/test/java/com/paraview/example/infrastructure/repository/user  /data-user/user/src/test/java/com/paraview/example/infrastructure/repository/user/mapper--------------数据库单元测试  /data-user/user/src/test/java/com/paraview/example/infrastructure/repository/user/mapper/config--------数据库单元测试的配制  /data-user/user/src/test/java/com/paraview/example/web  /data-user/user/src/test/java/com/paraview/example/web/facade-------------------------------------web层的单元测试  /data-user/user/src/test/resources  /data-user/user-api  /data-user/user-api/src  /data-user/user-api/src/main  /data-user/user-api/src/main/java  /data-user/user-api/src/main/java/com  /data-user/user-api/src/main/java/com/paraview  /data-user/user-api/src/main/java/com/paraview/example  /data-user/user-api/src/main/java/com/paraview/example/web  /data-user/user-api/src/main/java/com/paraview/example/web/dto-----------------------------------对外暴露的api的接口传输对象  /data-user/user-api/src/main/java/com/paraview/example/web/facade--------------------------------对外暴露的接口 |

## 执行单元测试用例



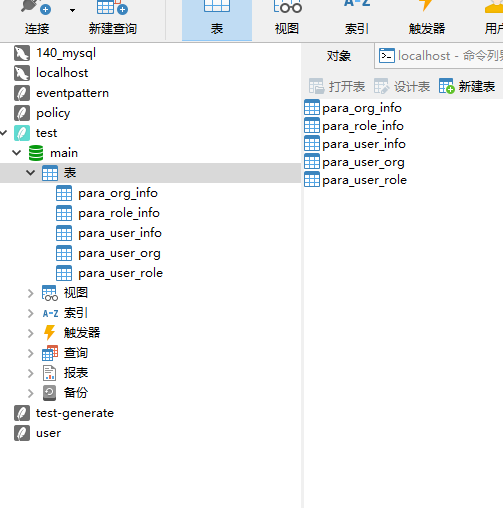
首次单元测试全部不通过

原因，由于单元测试采用sqllite数据库未执行建表操作

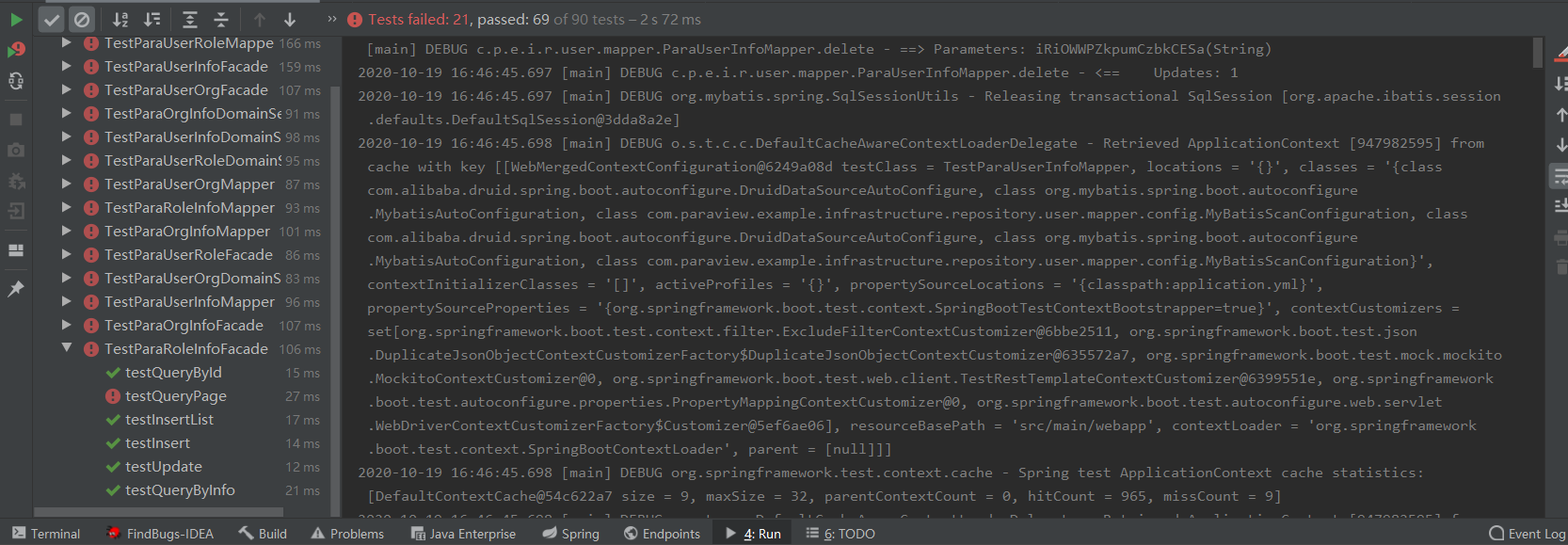
可以使用Navicat Premium 12打开数据库



执行测试数据库脚本



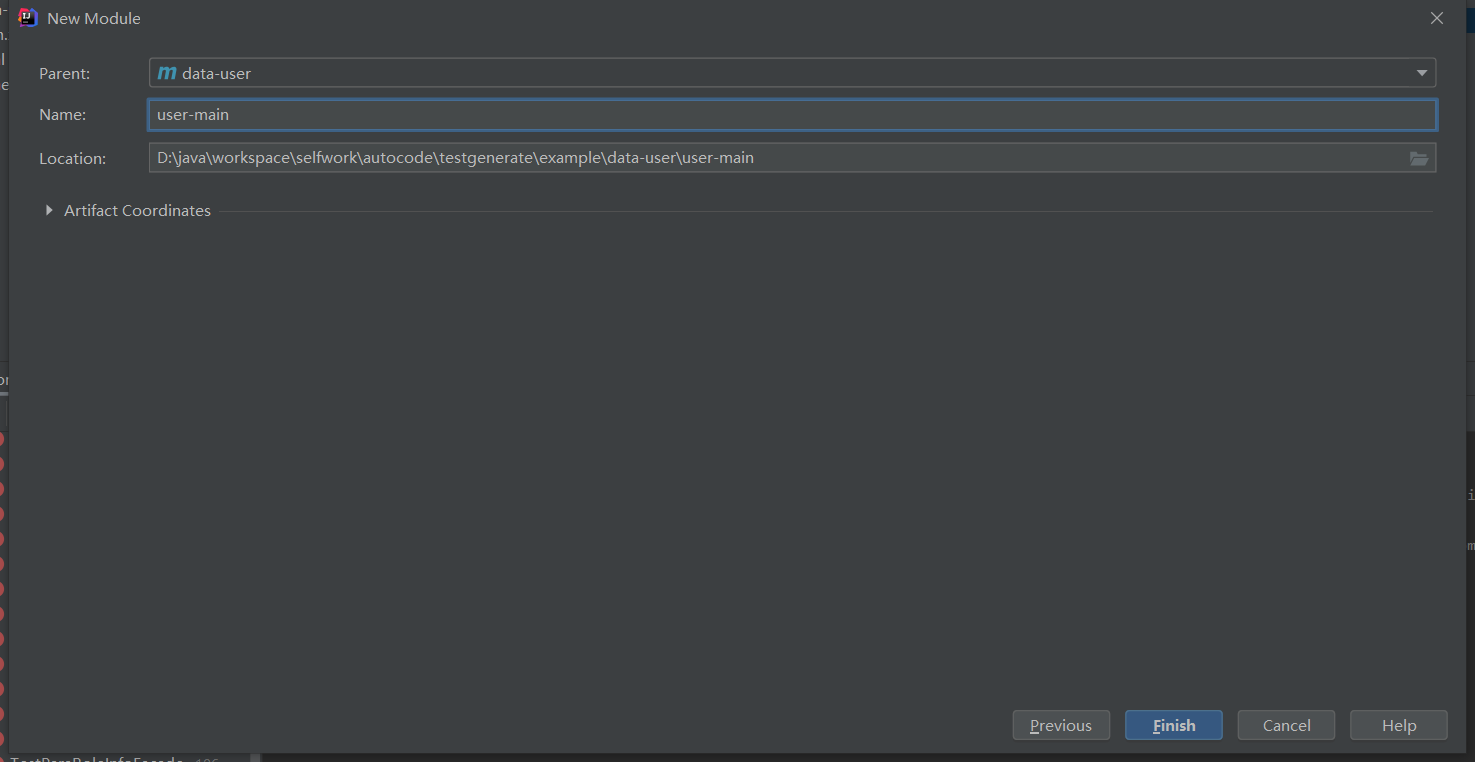
再次执行单元测试



由于在生成的代码中未包括任何的业务代码，像查询就这类带条件的操作，就不能通过，需要对具体的查询进行修改。

## 添加入口工程

生成的代码中仅包括业务与接口中，没有启动入口，添加一个新包，即启动入口，放入配制文件即可。



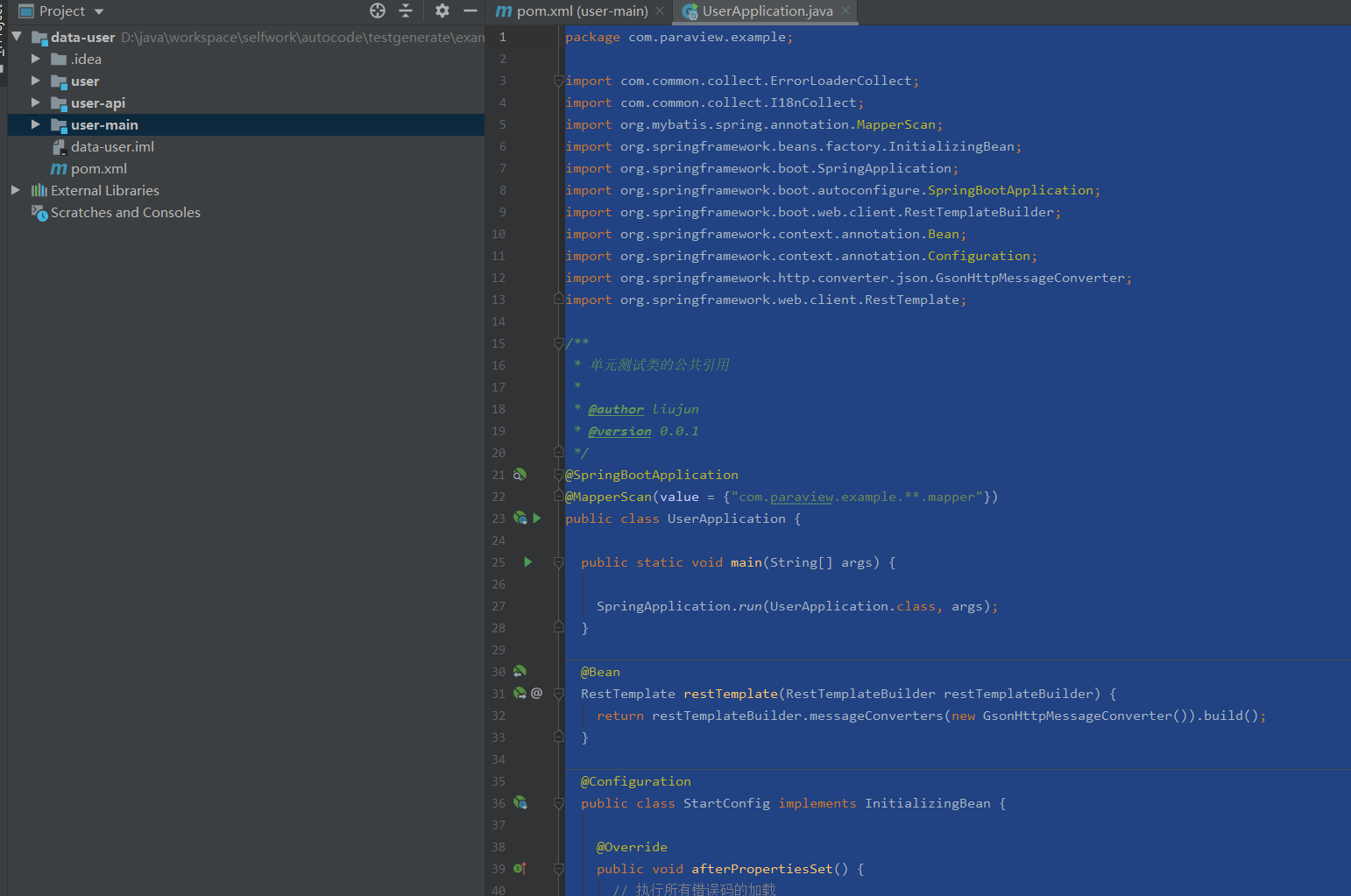
添加对业务包的引用，及主函数入口的设置

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>data-user</artifactId>  <groupId>com.paraview.example</groupId>  <version>1.0-SNAPSHOT</version>  </parent>  <modelVersion>4.0.0</modelVersion>  <artifactId>user-main</artifactId>  <dependencies>  <dependency>  <groupId>com.paraview.example</groupId>  <artifactId>user</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.8.0</version>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <mainClass>com.paraview.example.UserApplication</mainClass>  </configuration>  <executions>  <execution>  <goals>  <!--可以把依赖的包都打包到生成的Jar包中-->  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build>  </project> |

入口main文件

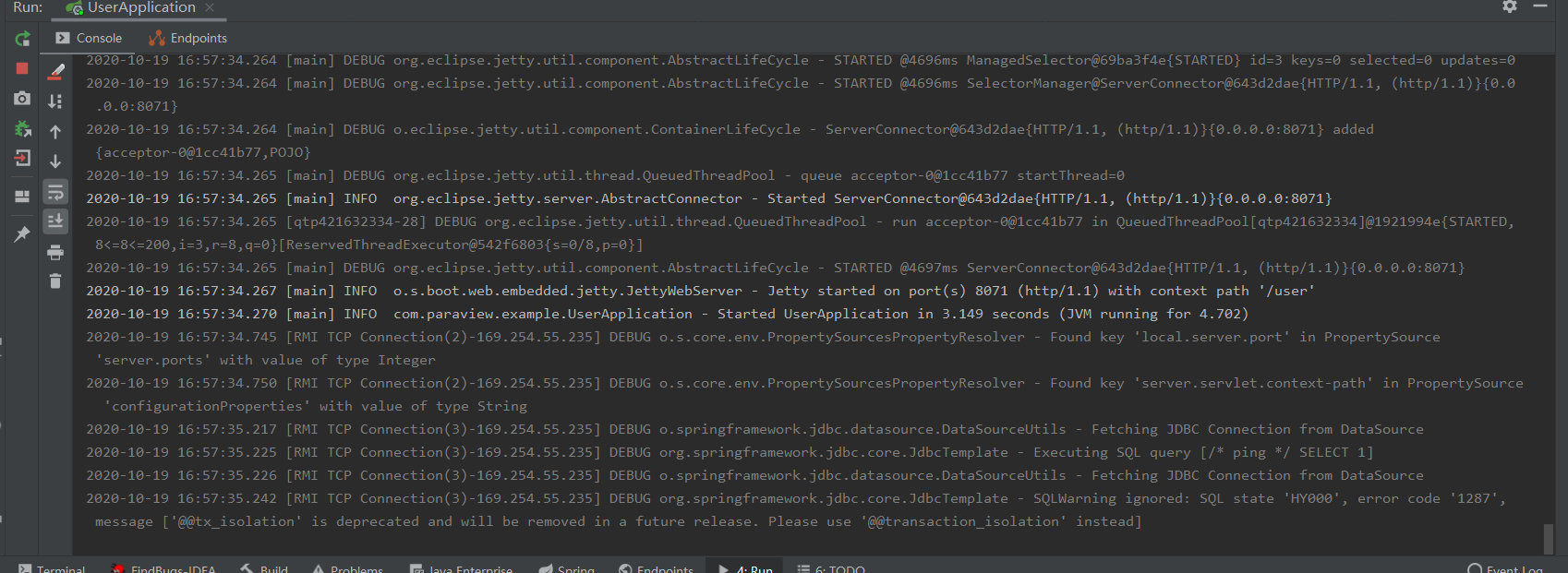
|  |
| --- |
| package com.paraview.example;  import com.common.collect.ErrorLoaderCollect;  import com.common.collect.I18nCollect;  import org.mybatis.spring.annotation.MapperScan;  import org.springframework.beans.factory.InitializingBean;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.boot.web.client.RestTemplateBuilder;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.http.converter.json.GsonHttpMessageConverter;  import org.springframework.web.client.RestTemplate;  /\*\*  \* 单元测试类的公共引用  \*  \* @author liujun  \* @version 0.0.1  \*/  @SpringBootApplication  @MapperScan(value = {"com.paraview.example.\*\*.mapper"})  public class UserApplication {  public static void main(String[] args) {  SpringApplication.run(UserApplication.class, args);  }  @Bean  RestTemplate restTemplate(RestTemplateBuilder restTemplateBuilder) {  return restTemplateBuilder.messageConverters(new GsonHttpMessageConverter()).build();  }  @Configuration  public class StartConfig implements InitializingBean {  @Override  public void afterPropertiesSet() {  // 执行所有错误码的加载  ErrorLoaderCollect.loaderAll();  I18nCollect.INSTANCE.printI18n();  }  }  } |



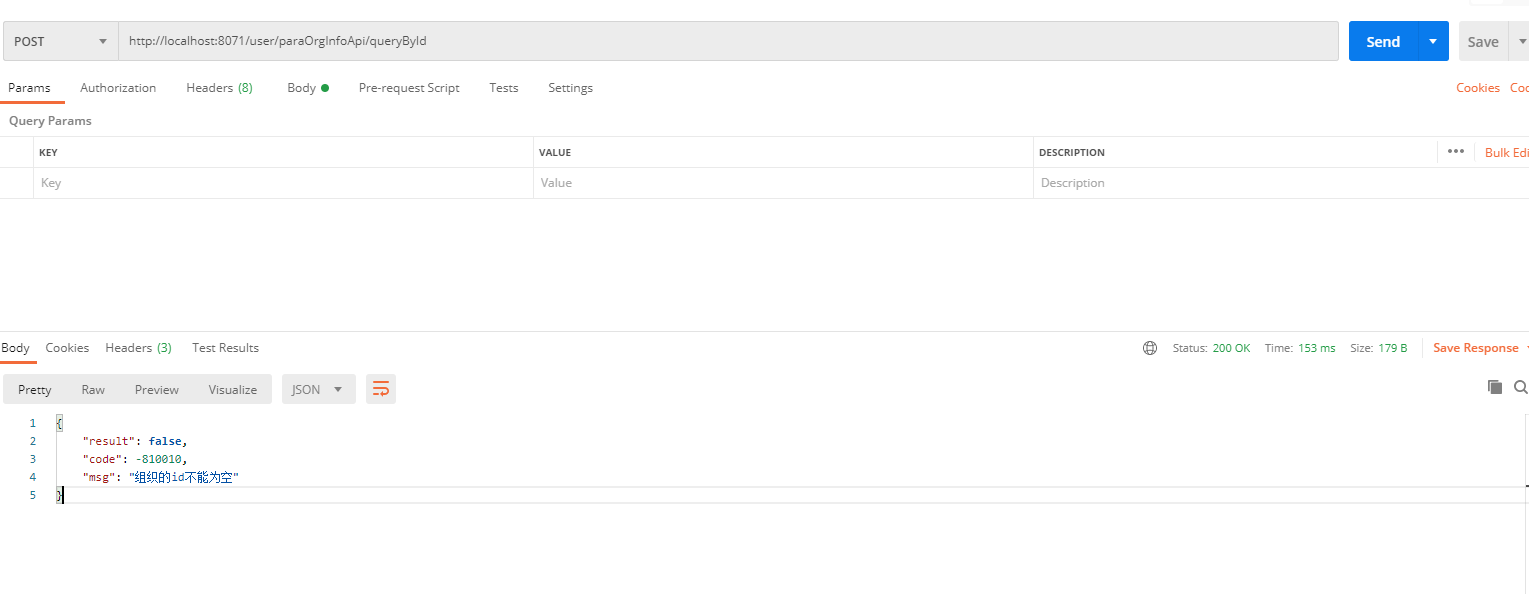
拷贝数据库连接文件及日志文件



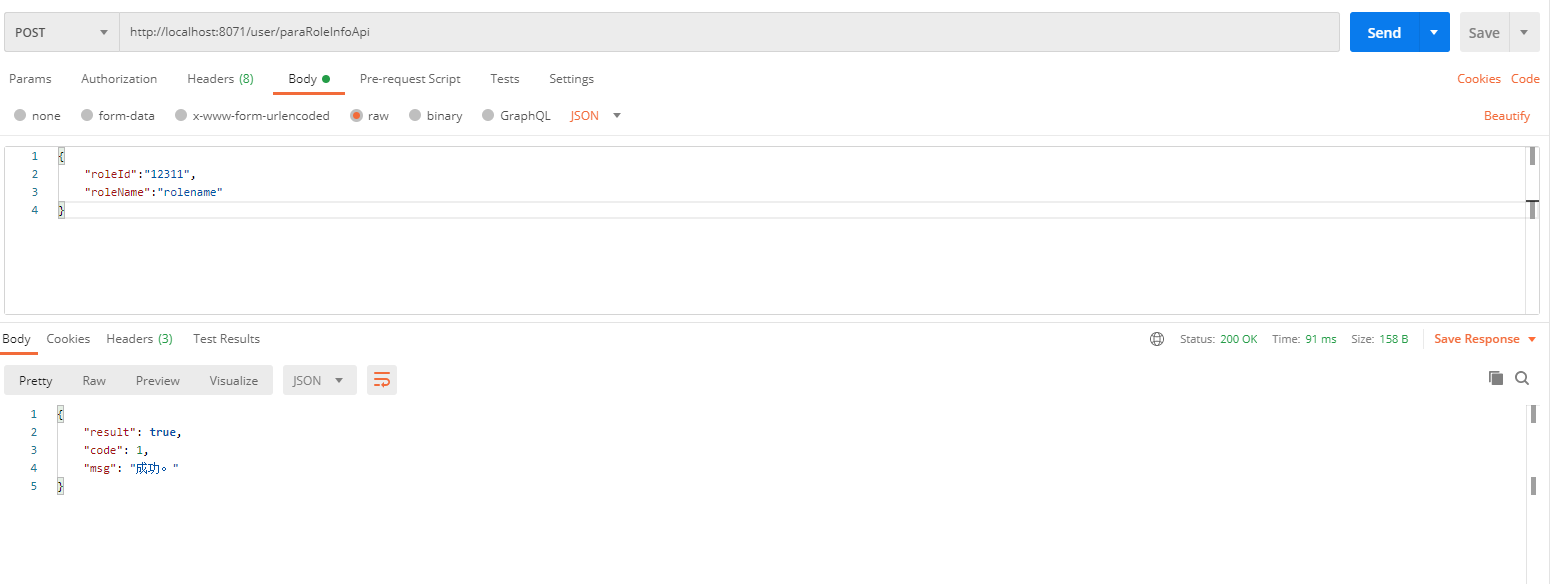
## 启动工程测试



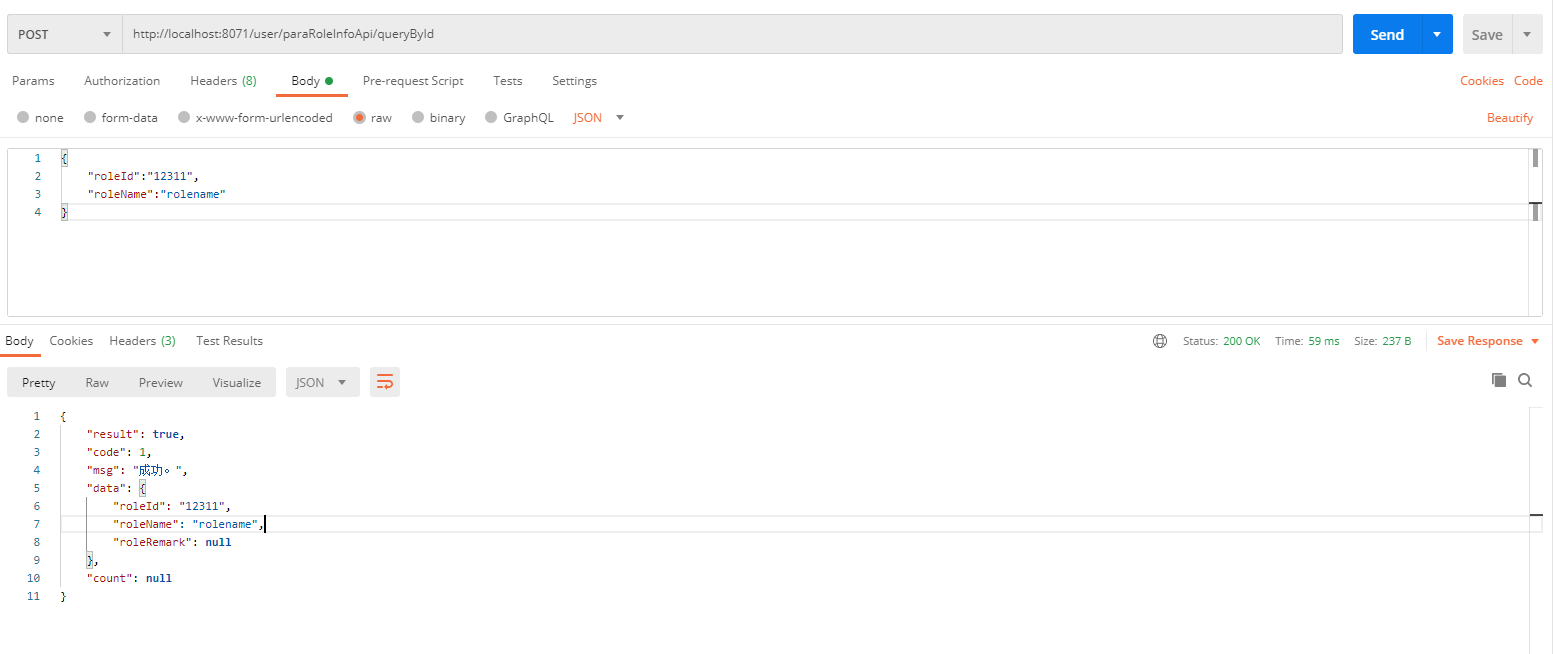
表示当前已经启动成功，可通过postman工具测试



测试保存



测试查询



这是最终的工程

