小米汽车电池预警大作业开发流程及结果展示

一、创建项目及初始化配置

1、IDEA新建SpringBoot项目

技术栈：JDK1.8、SpringBoot 2.2.1.RELEASE。

1. 项目初始化配置

（1）编码：将控制台编码、文件编码更改为UTF-8；

（2）设置热部署与自动编译。

1. 项目测试结果

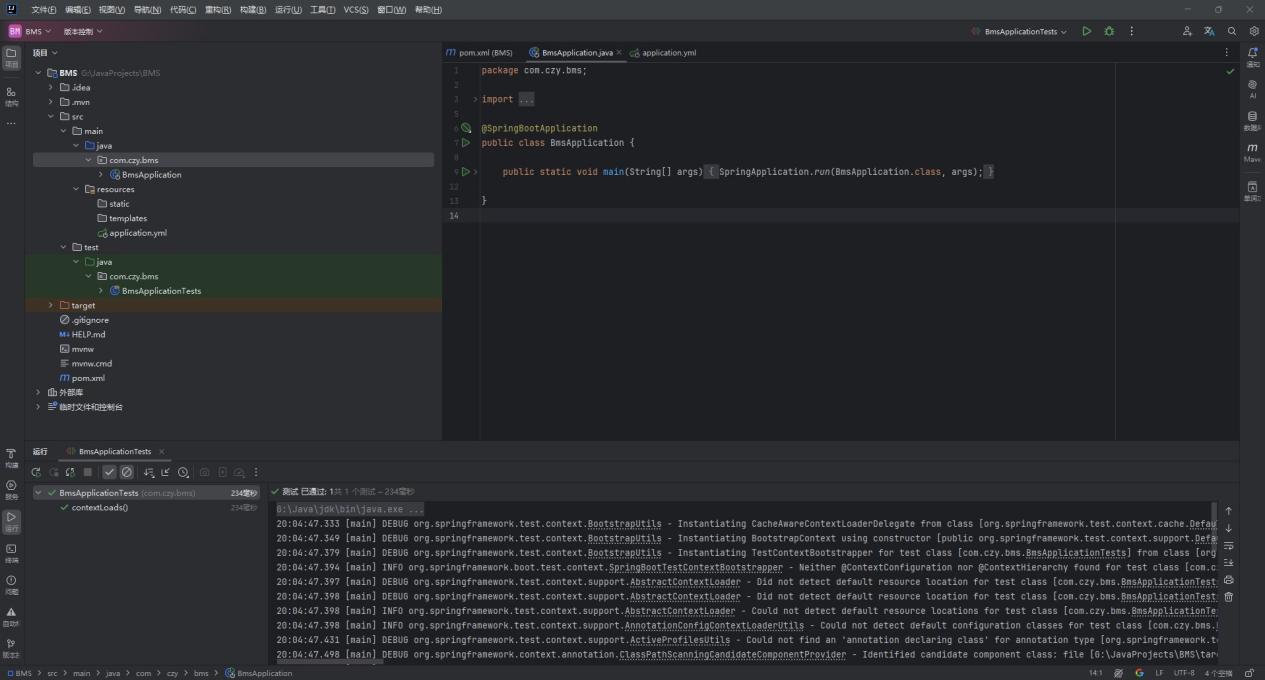


图1.1 项目搭建测试结果图

1. 代码管理GitHub远程仓库
2. 编写TestController测试类
   1. 在application.yml文件中配置端口好和项目名称；
   2. 在TestController类上添加注解@RestController和@RequestMapping("/test")；
   3. 编写测试方法返回字符串结果。
3. TestController类测试结果

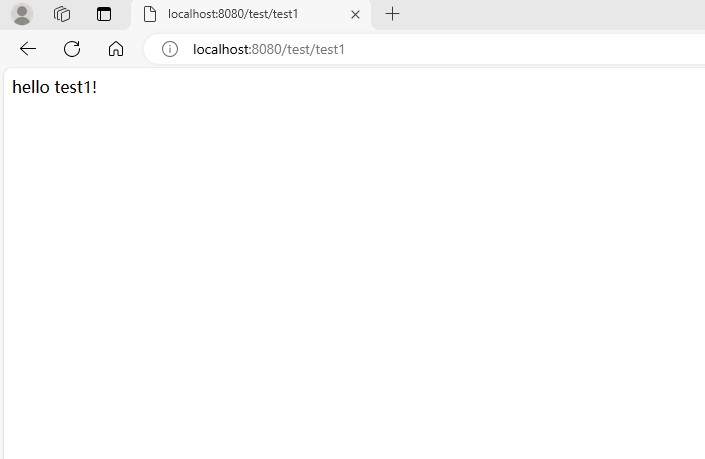


图1.2 TestController类测试结果图

1. 使用HTTP Client完成接口测试

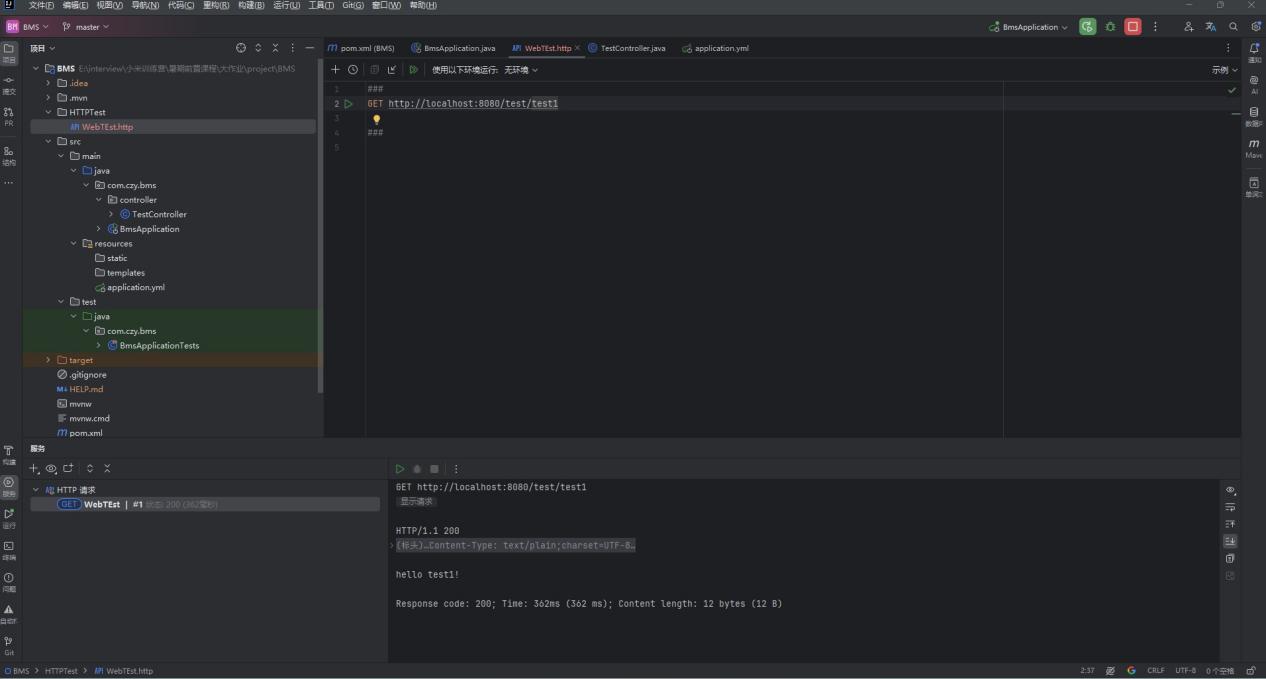


图1.3 HTTP Client接口测试结果图

二、封装返回结果

1、引入Lombok依赖

2、创建response包以及CommonResp类

3、在CommonResp类中定义状态码、信息以及数据

4、在TestController类中编写方法进行测试

5、测试结果

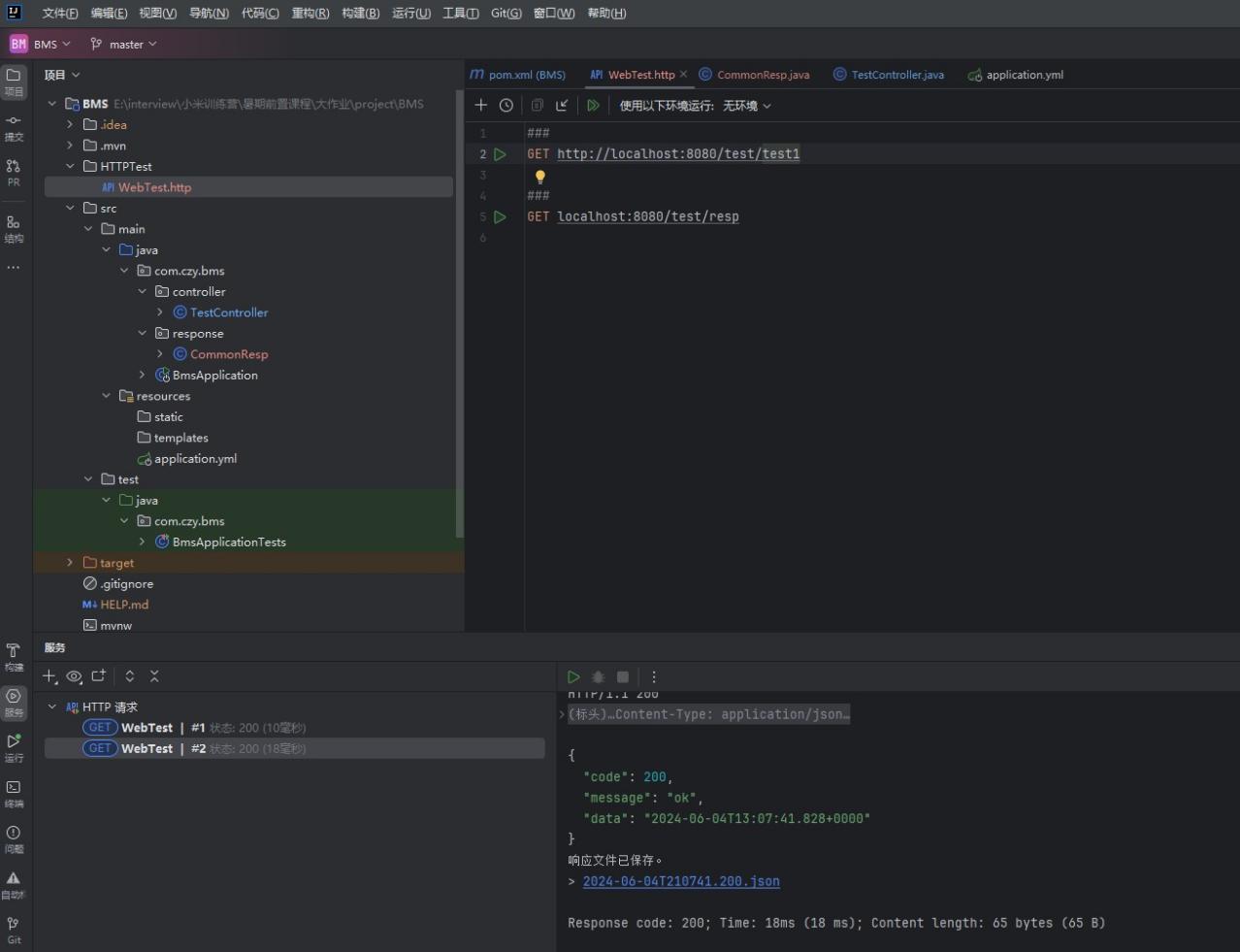


图2.1 封装返回结果测试结果图

1. 配置时间格式及时区
2. 测试结果

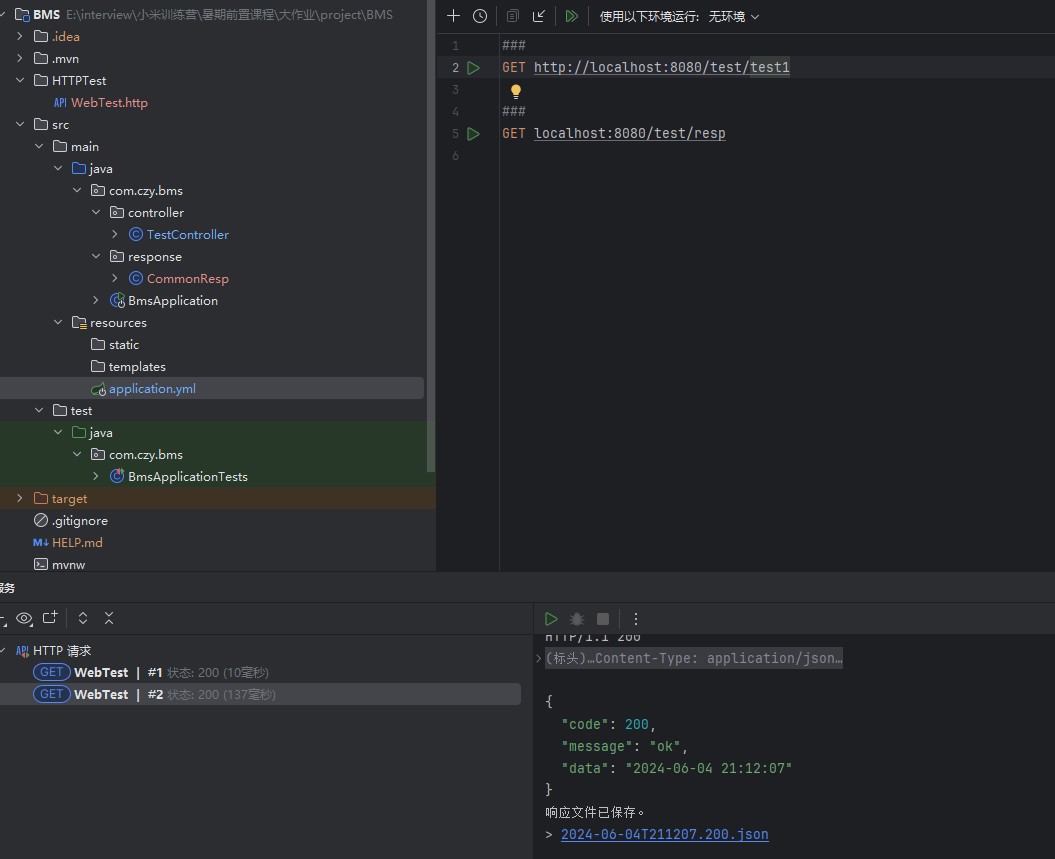


图2.2 时间格式及时区测试结果图

1. 测试以json数组的形式返回数据
   1. 添加json-simple依赖
   2. 测试结果

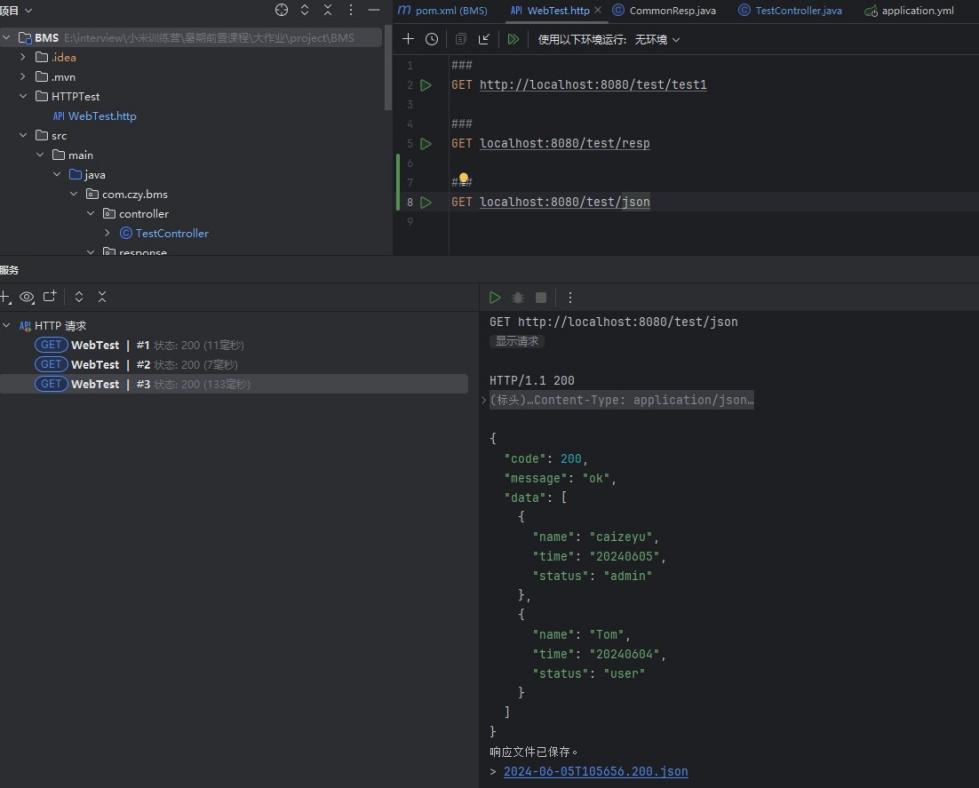


图2.3 以json数组格式返回数据测试结果图

三、连接数据库

1、创建数据库

2、IDEA连接数据源

3、测试数据源连接

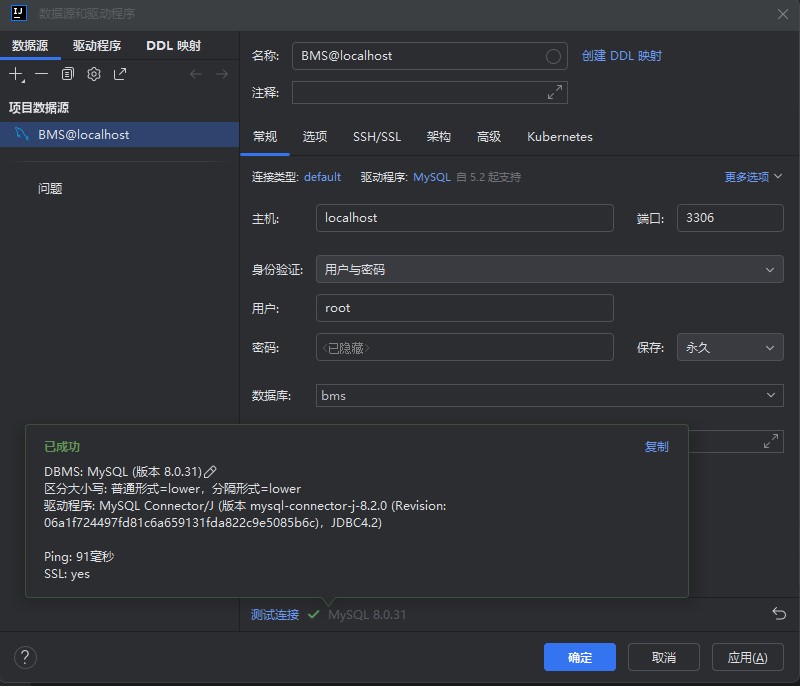


图3.1 IDEA连接数据源测试结果图

4、运行SQL脚本

（1）汽车表vehicle：



图3.2 创建汽车表

（2）规则表battery\_rules

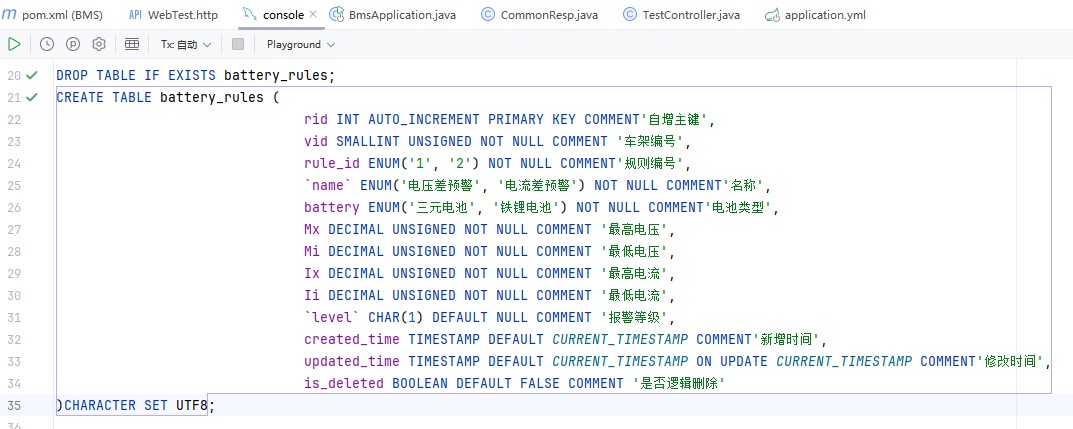


图3.3 创建规则表

四、集成Mybatis

1、引入mysql和mybatis依赖

2、配置数据源

3、定义持久层

（1）mapper包下的VehicleMapper接口

（2）resources/mapper文件夹下的VehicleMapper.xml文件

（3）在application.yml文件中配置classpath

（4）在启动类中添加@MapperScan注解

4、定义应用层

（1）service包下的VehicleService接口

（2）sevice/Impl文件夹下的VehicleServiceImpl类

5、定义控制层、编写测试方法，查询Vehicle表中的记录数量

6、测试结果

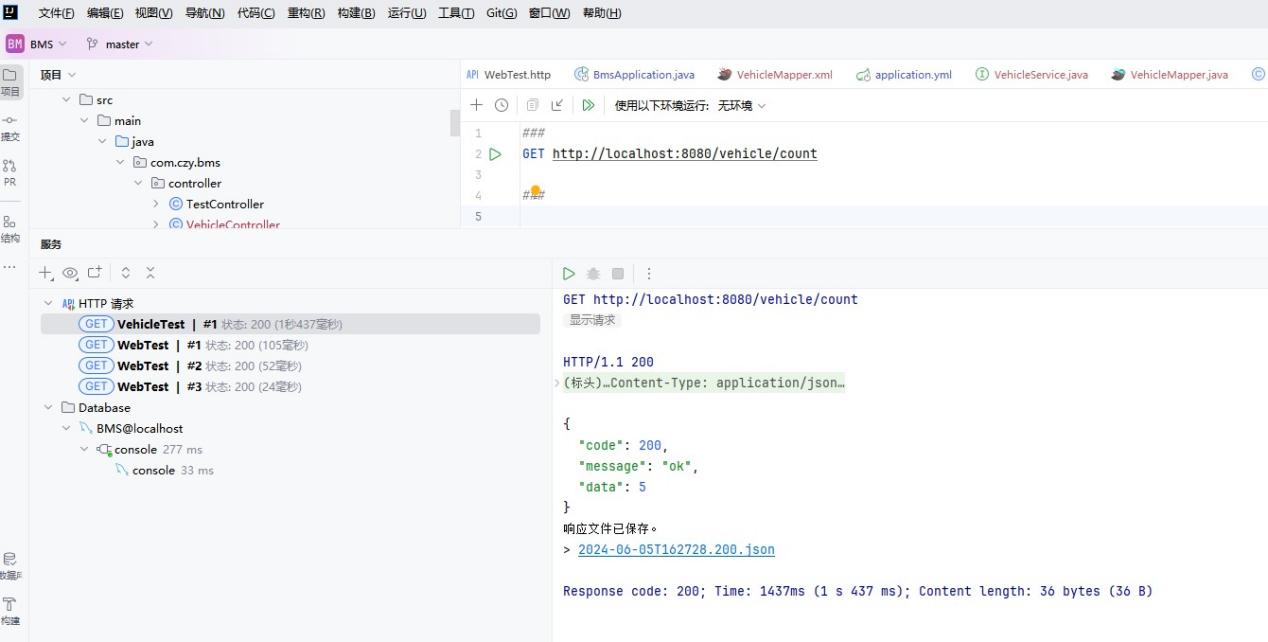


图4.1 Mybatis集成测试结果

五、集成Mybatis官方代码生成器及简单功能测试

1、使用Mybatis官方代码生成器生成文件及代码

（1）在POM文件中添加mybatis generator自动生成代码插件，在<configurationFile>定义配置文件的路径；

（2）在resources文件夹下创建generator-vehicle.xml配置文件；

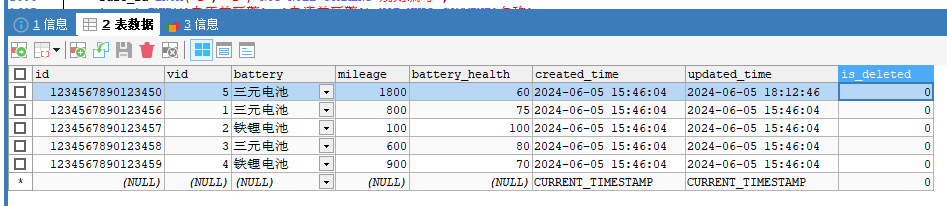
（3）在generator-vehicle.xml文件中配置数据源、实体类的路径、mapper.xml的路径、mapper类的路径、表名以及实体类名；

（4）在maven中点击mybatis-generator插件运行；

（5）代码生成器会生成四个文件，分别是Vehicle.java、VehicleExample.java、VehicleMapper接口和VehicleMapper.xml。

1. 数据库简单查询操作测试

数据库中数据



1. 查询vehicle表中数据的数量

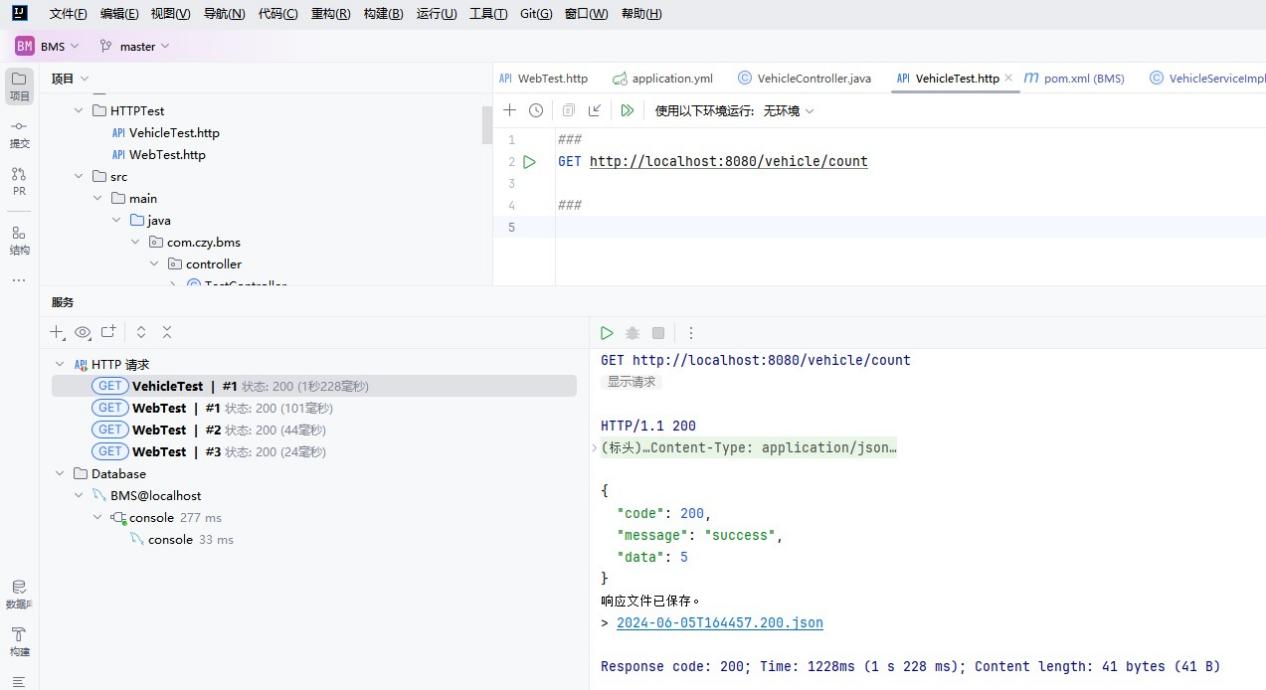


图5.1 查询vehicle表中数据数量

1. 查询vehicle表中的所有数据

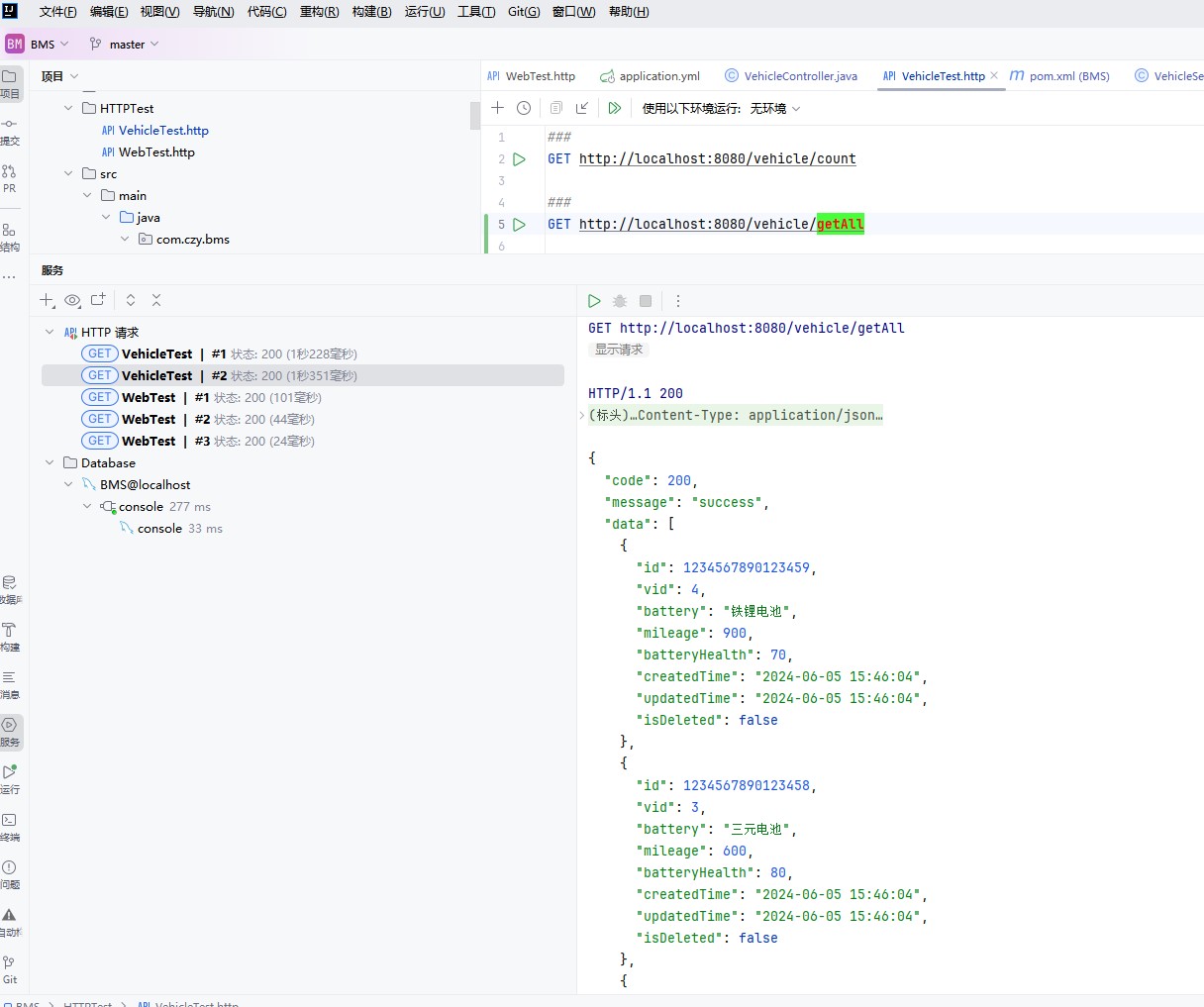


图5.2 查询vehicle表中所有数据

1. 按vid查询车辆

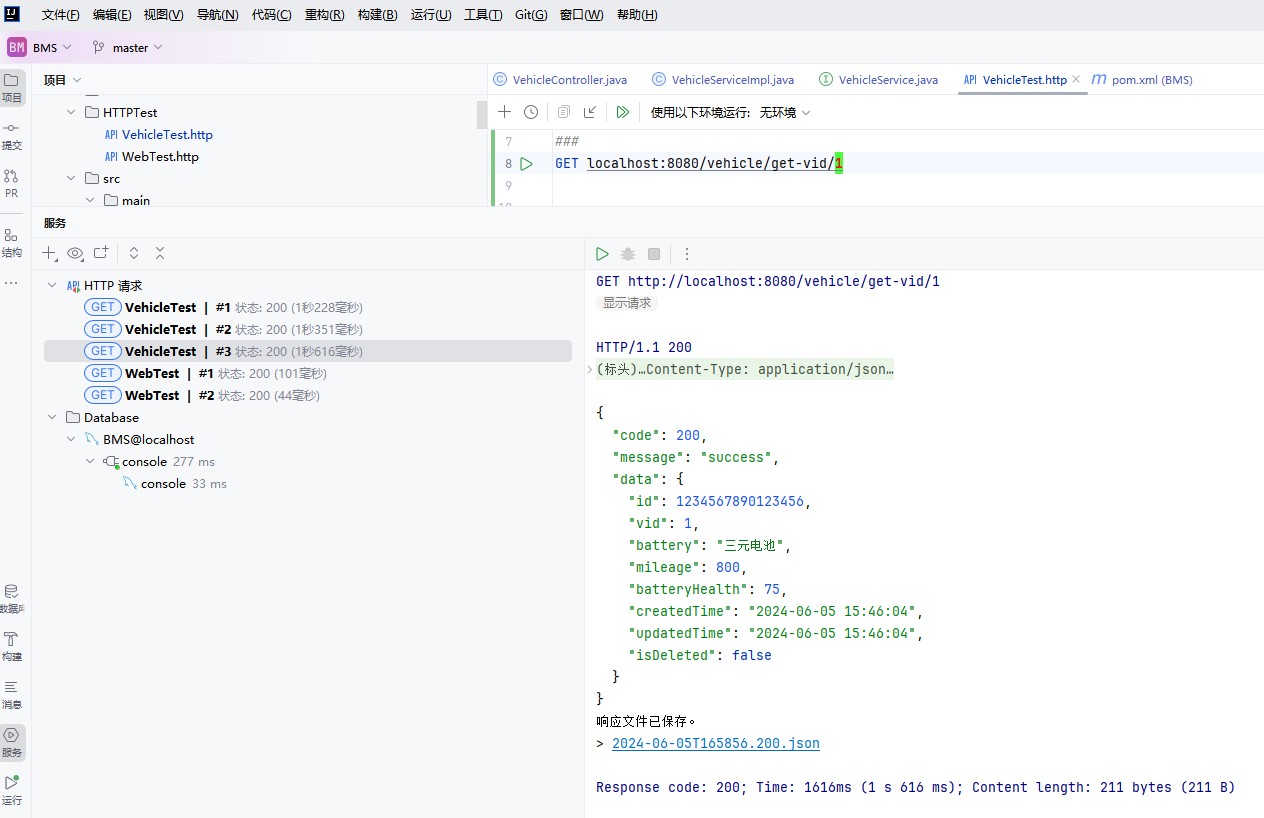
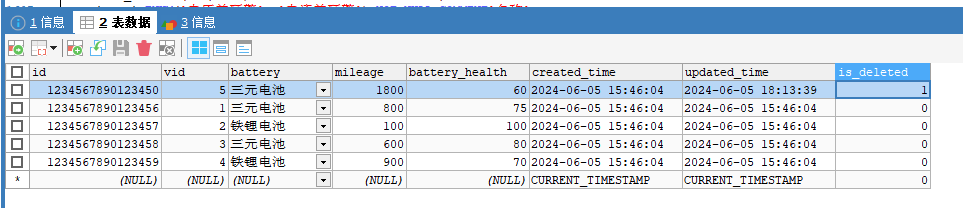


图5.3 按vid查询车辆

1. 查询优化：引入逻辑删除，只能查询出逻辑未删除的数据

a.先将vid为5的数据条目变为逻辑删除



b.查询数据数量

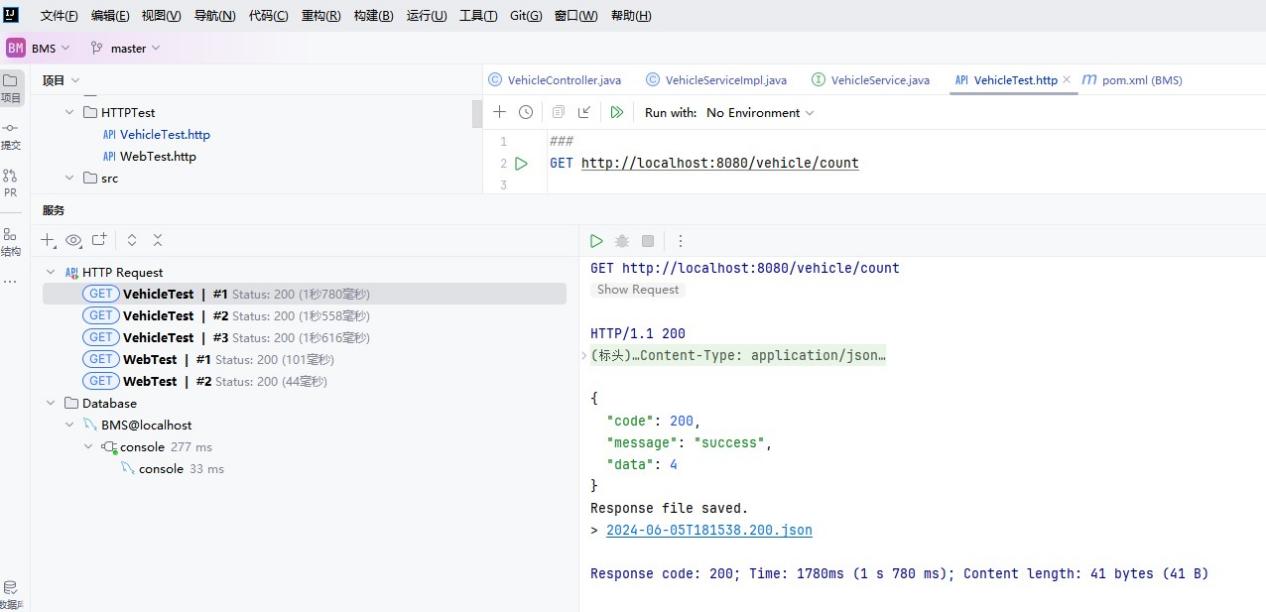


图5.4 优化查询数据表数量

c.查询所有数据

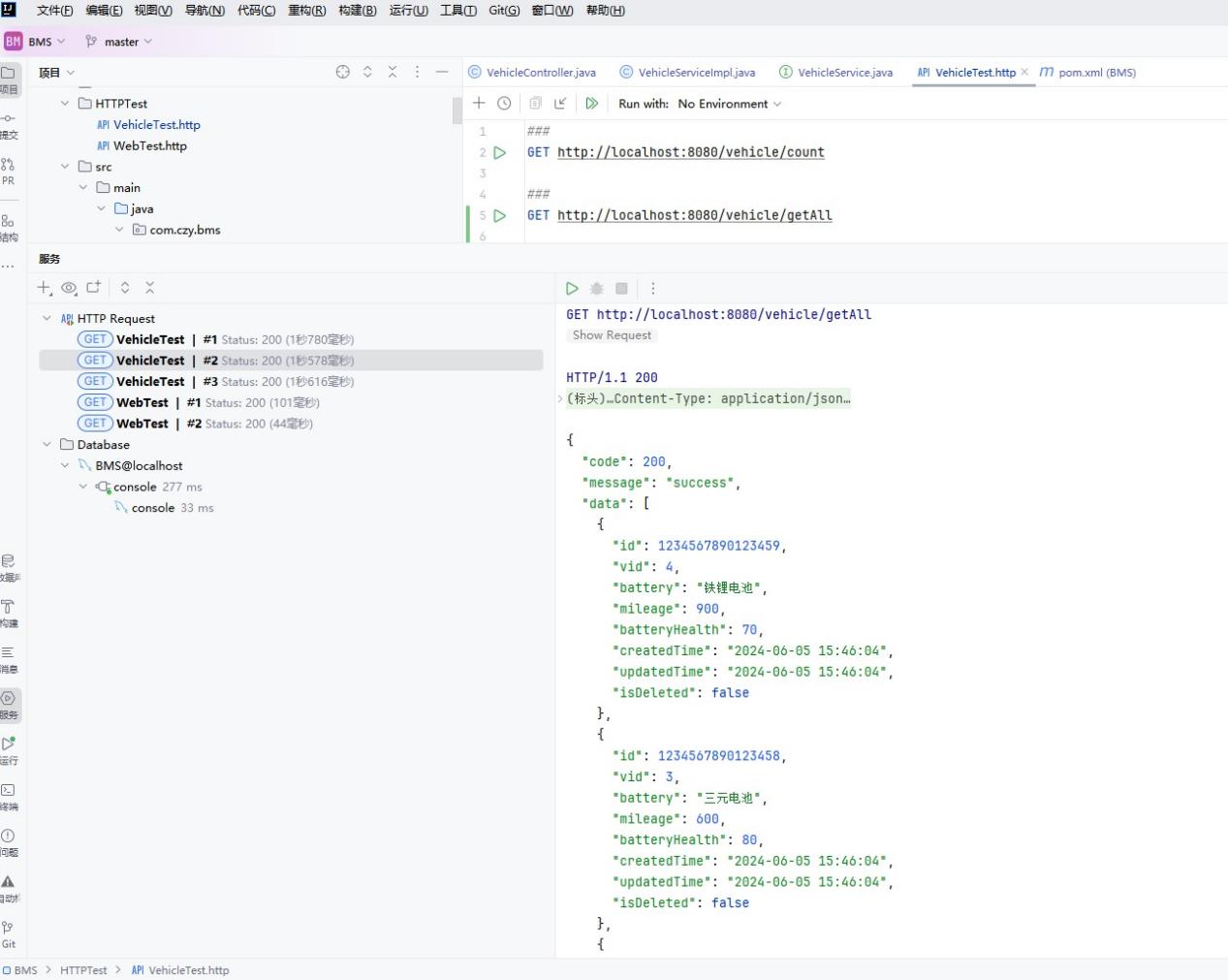


图5.5 优化查询所有数据

d.按vid查询车辆（查询已逻辑删除的5号车辆）

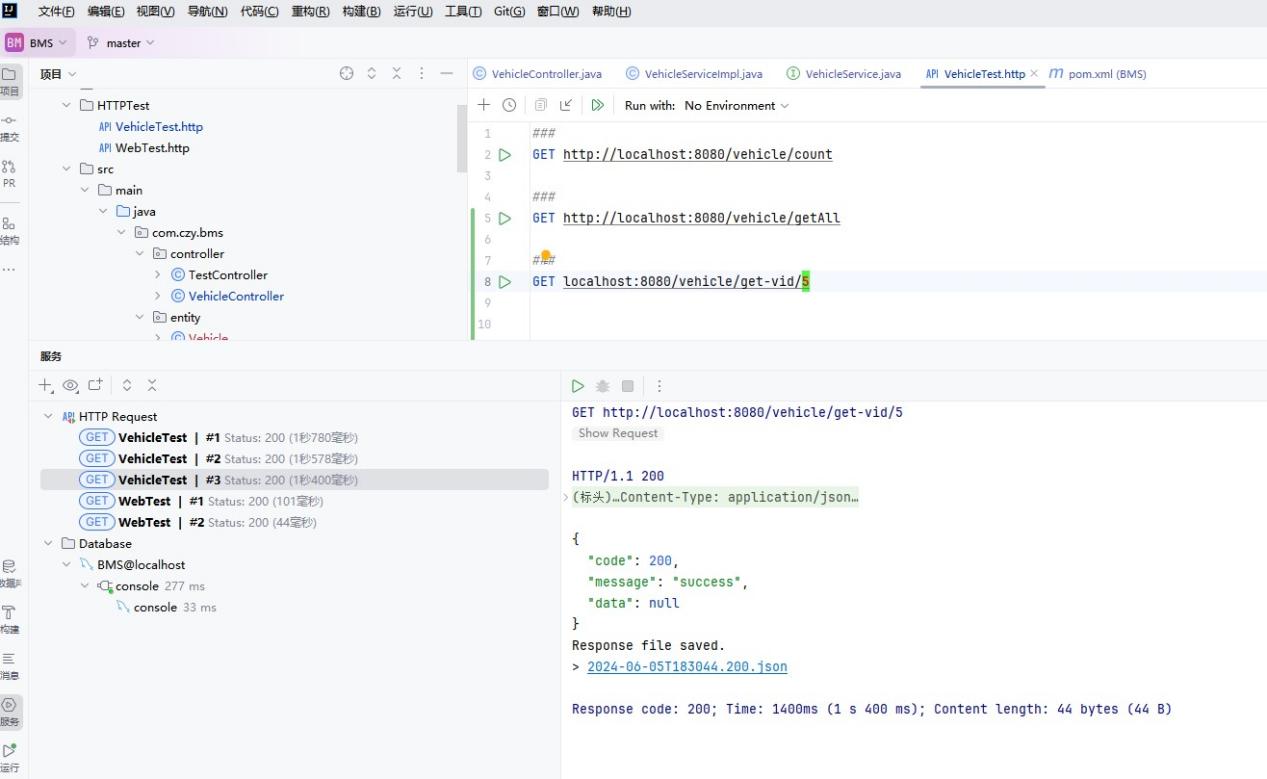


图5.6 优化按vid查询车辆

1. 数据库简单删除操作测试
2. 删除操作（物理删除，直接从数据库中删除vid为5的数据条目）

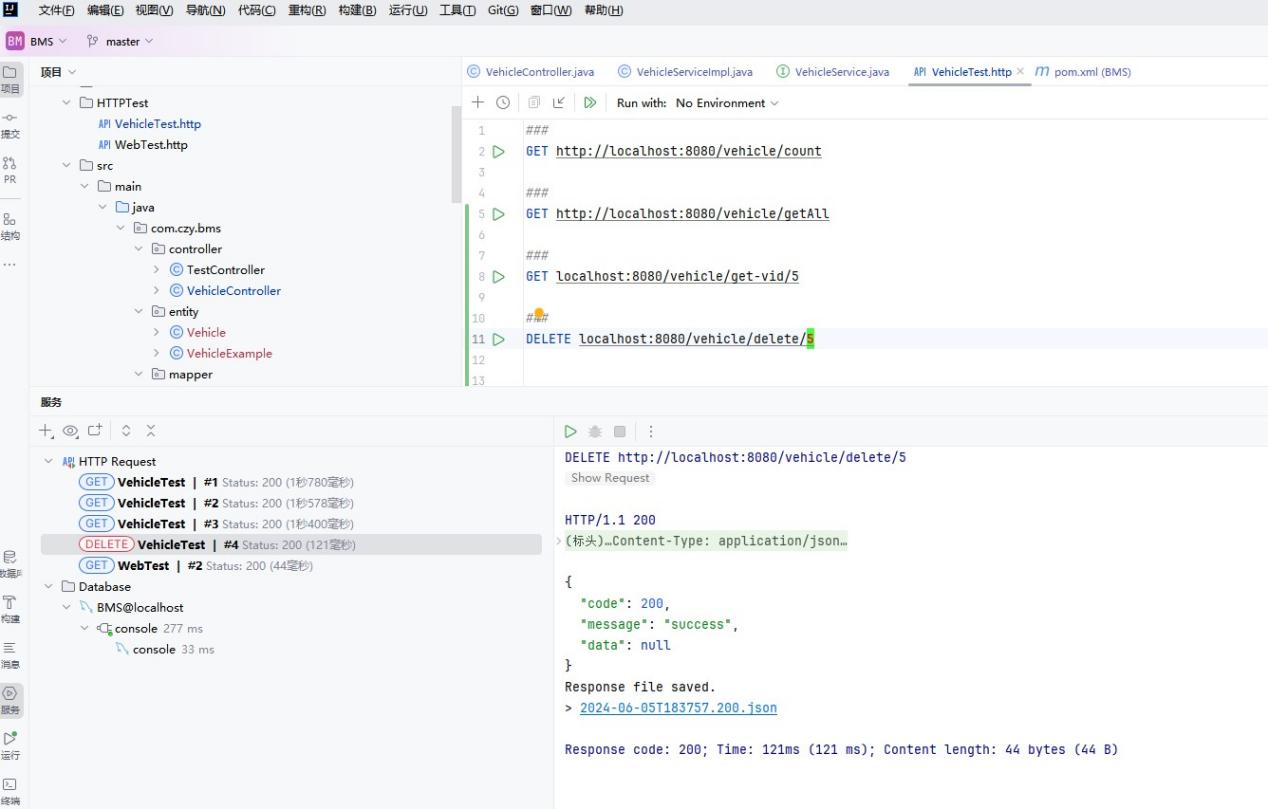


图5.7 按vid物理删除车辆

1. 优化删除操作（先还原数据库数据。逻辑删除，本质上为将vid为5的数据的is\_deleted改为true）

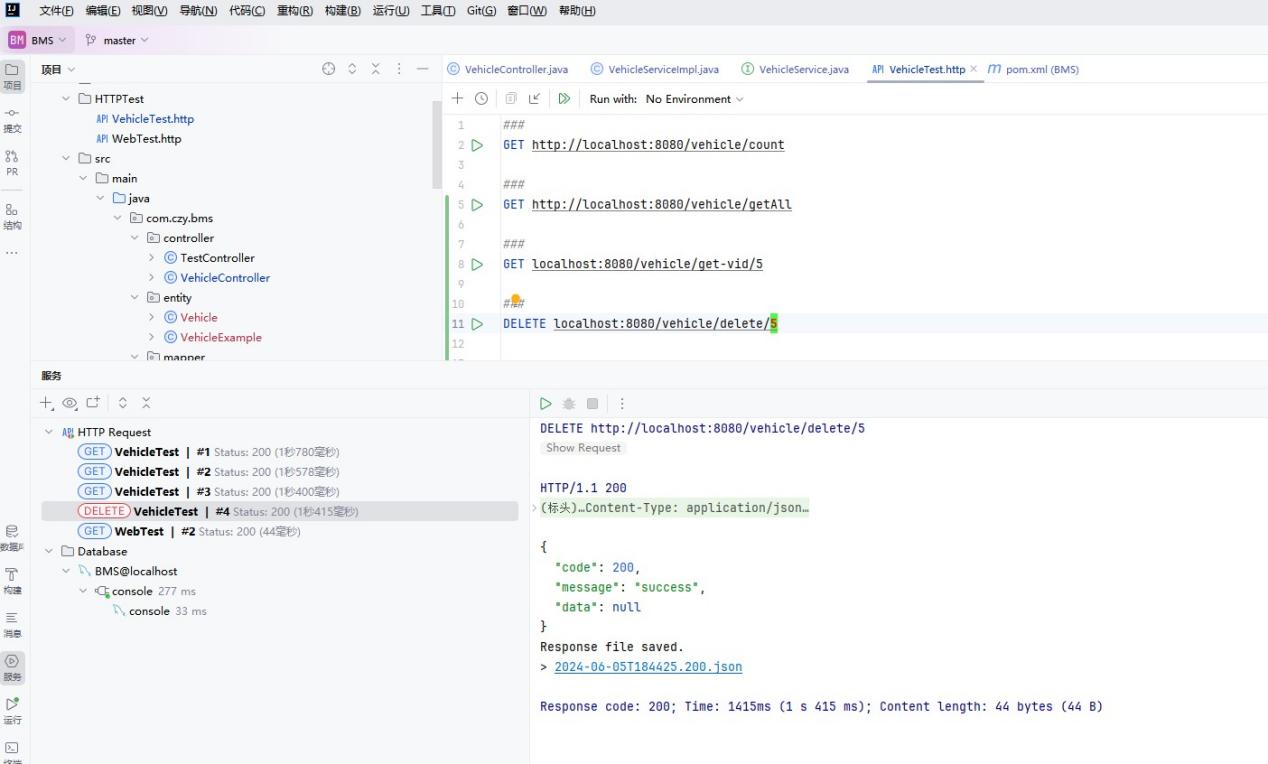


图5.8 按vid逻辑删除车辆

1. 进一步优化（新增判断，如果已经逻辑删除或者没有查询到相应的数据条目，则返回失败信息，下面的结果只展示失败信息）

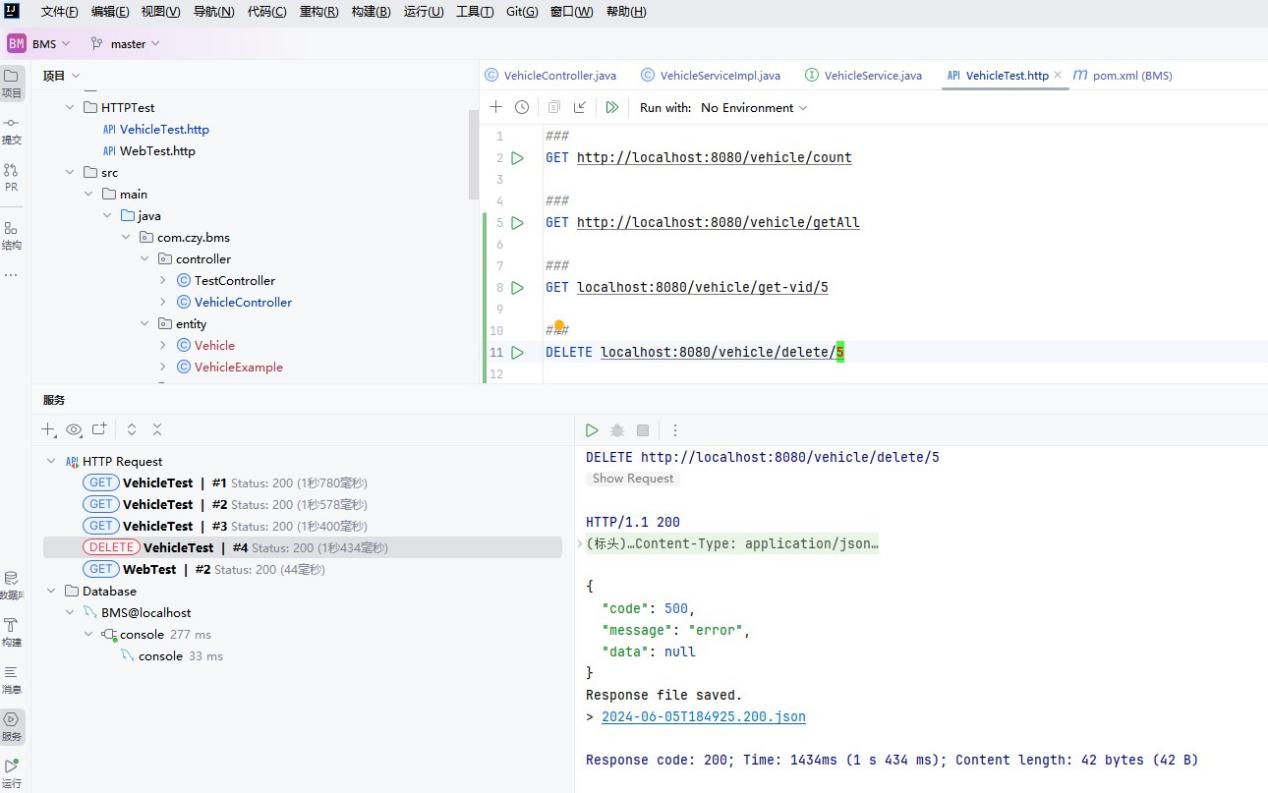


图5.9 按vid逻辑删除车辆失败测试

1. 使用代码生成器生成battery\_rules表相关文件

六、统一封装请求参数（以Vehicle表为例）

1、引入validation依赖；

2、创建request包；

3、创建VehicleSaveReq.java用于传递新增书籍的请求参数，创建VehicleQueryReq.java用于传递查询书籍的请求参数；

4、VehicleSaveReq.java的字段与Vehicle.java相同，并用@NotBlank或者@NotNull注解约束非空字段；

5、VehicleQueryReq.java中只有需要作为查询条件的字段，并用@NotBlank或者@NotNull注解约束非空字段；

6、由于Vehicle表的主键为16位数字组成的随机字符串，因此开发RandomIdUtil.java工具类，使用时间戳和随机生成的三位数拼接而成的字符串作为主键。

七、数据插入、查询、修改操作（以Vehicle表为例）

1、插入新的数据条目

以POST请求的形式插入数据，传入的数据为车架编号、电池类型、总里程和电池健康状况四类数据，其余的数据全部是自动生成。

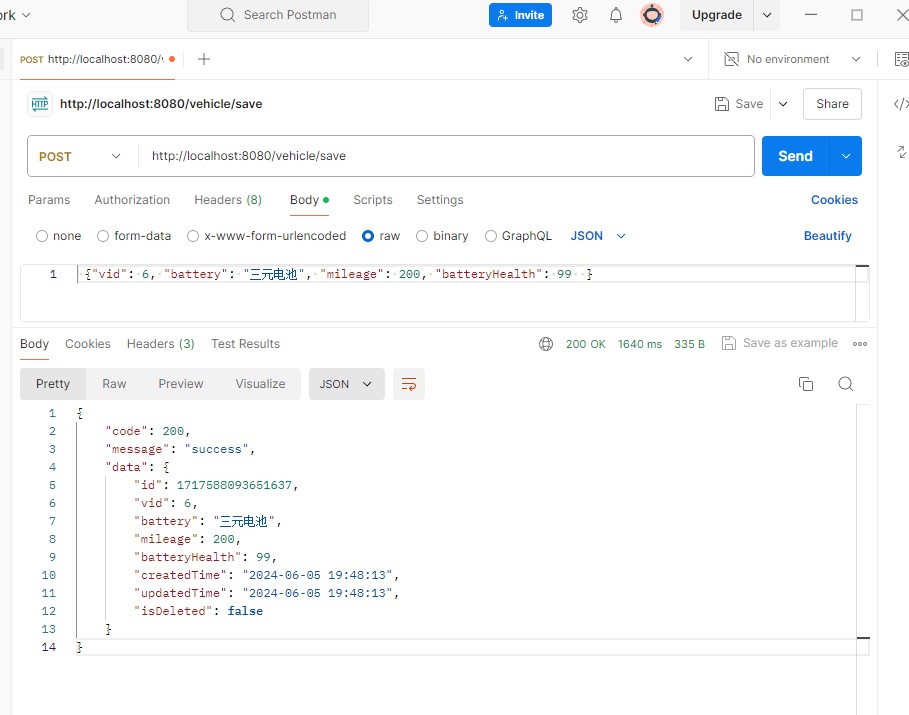


图7.1 插入数据测试

2、查询数据（精准查询，只返回一条数据）

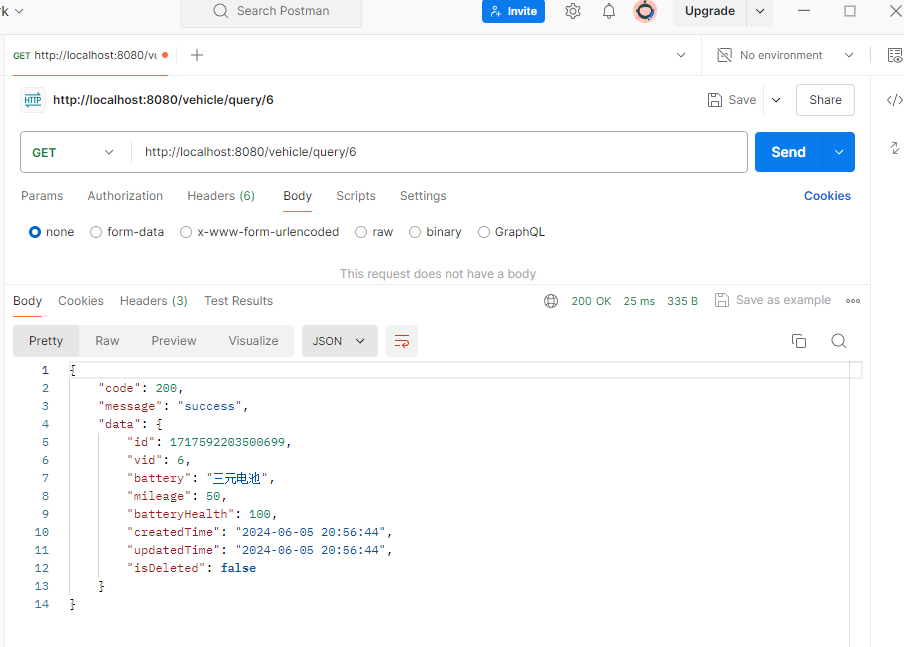


图7.2 查询数据测试

3、优化精准查询，增加逻辑删除操作判断（Vehicle表中，vid为5的数据已经被逻辑删除）

（1）查询未删除的数据

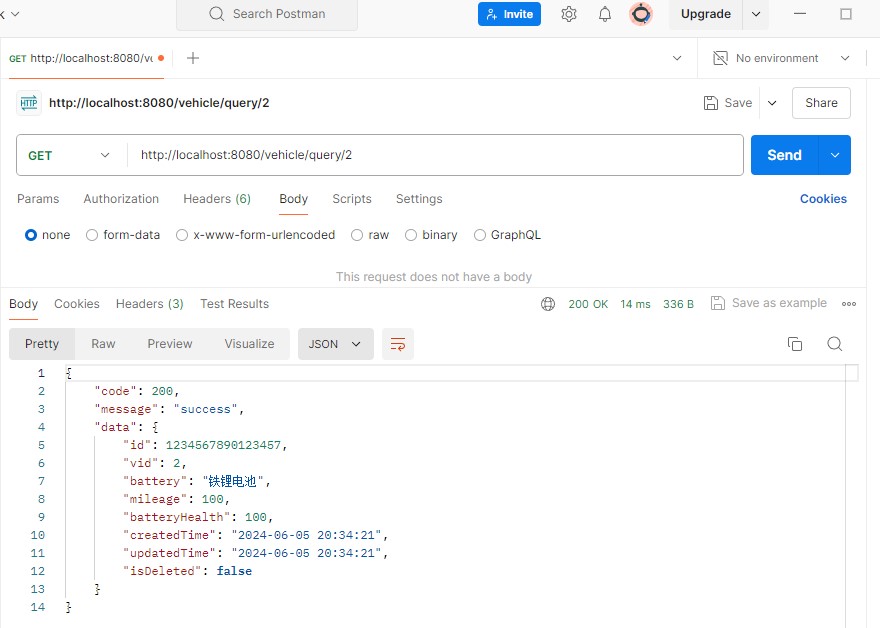


图7.3 查询未删除

（2）查询已删除的数据

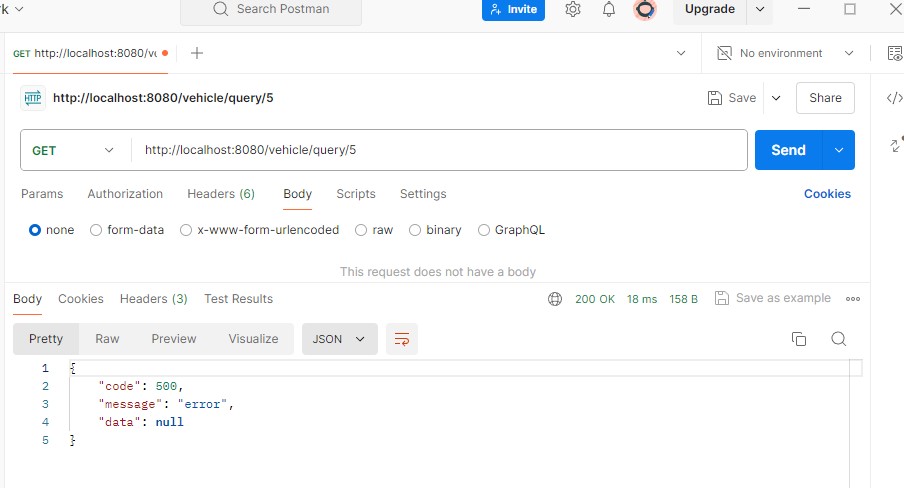


图7.4 查询已删除

4、条件查询，返回所有满足条件的数据（已使用逻辑查询判断进行优化）

查询所有电池种类是三元电池的数据

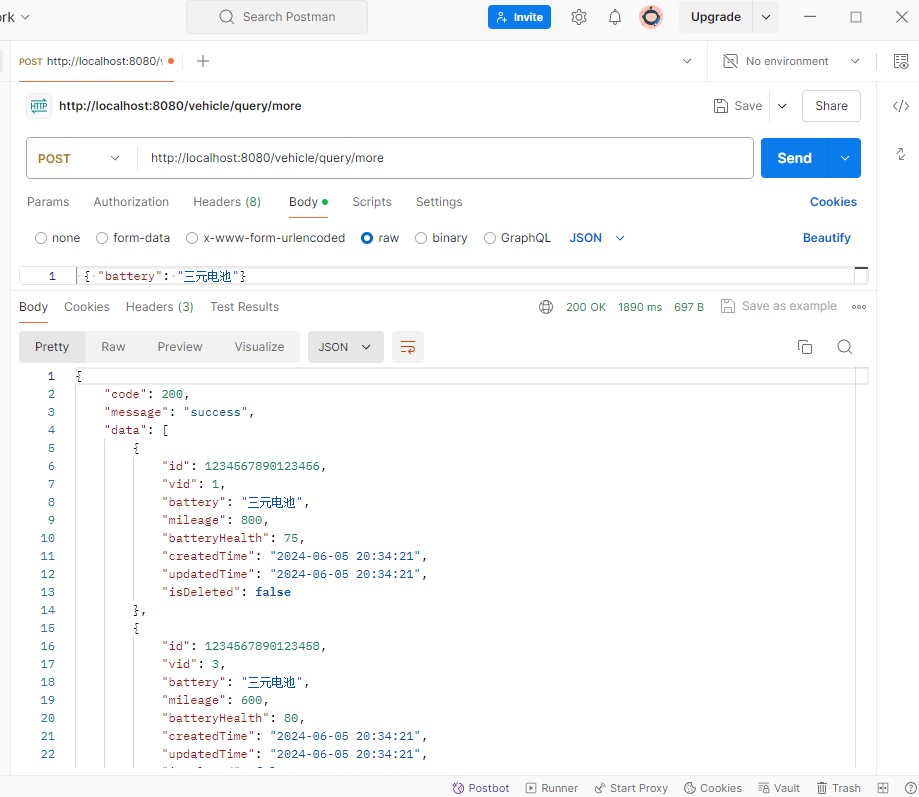


图7.5 优化后条件查询结果

5、修改操作

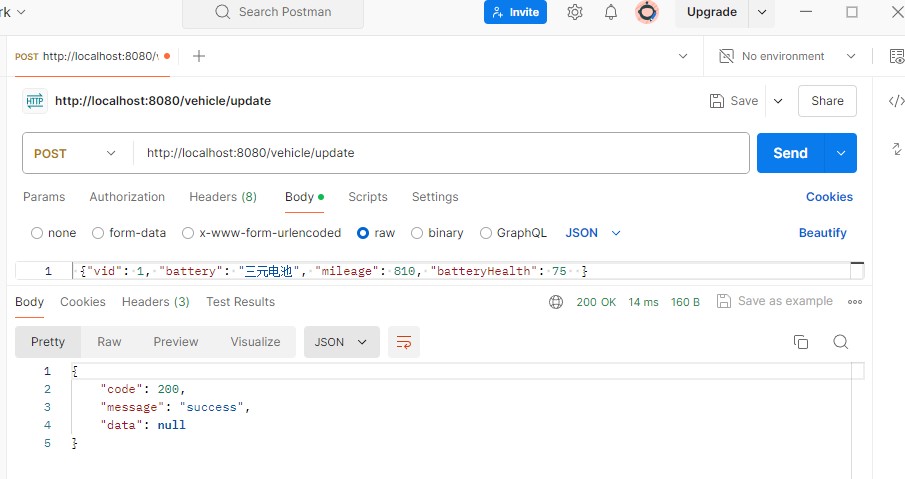


图7.6 修改数据测试结果

6、优化修改操作，加入判断逻辑删除

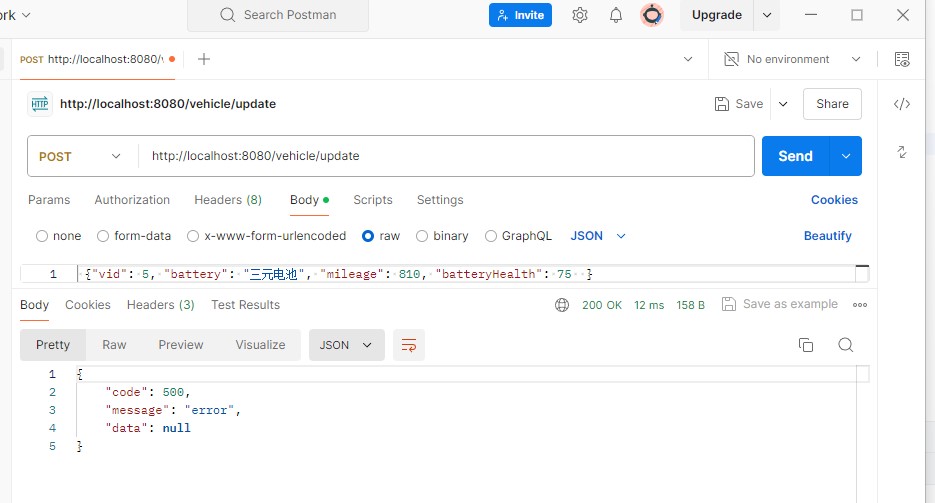


图7.7 优化修改数据测试结果